Instituto Superior de Engenharia de Lisboa

ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Inteligência Artificial e Sistemas Cognitivos

Inteligência Artificial e Sistemas Cognitivos

Mestrado em Engenharia Informática e Multimédia

Rúben Santos, nº 49063

Semestre de Inverno, 2021/2022

Índice

Índice de Figuras

Índice de Equacões

Índice de Tabelas

1. Introdução

Este projeto tem como objetivo a aprendizagem e concretização de modelos e arquiteturas de inteligência artificial e sistemas cognitivos.

Neste relatório são apresentados os diversos problemas propostos na cadeira de inteligência artificial e sistemas cognitivos e as respetivas resoluções onde foram aplicadas as técnicas estudadas que incidiram principalmente nos seguintes temas:

1. Redes neuronais artificiais
2. Aprendizagem por reforço, com recurso aos métodos de aprendizagem:
    1. Q-Learning
    2. Q-Learning com memoria episódica
    3. Dyna-Q
3. Raciocínio automático para planeamento, com recurso a:
    1. Métodos de procura em espaços de estados A* ponderada (Weighted A*)
    2. Metodos de planeamento automático com base no método Frente-Onda (Wavefront)
4. Raciocínio automático para otimização, com recurso a:
    1. Método Simulated Anealing
    2. Hill-Climbing estocástico
    3. Hill-Climbing com reinício aleatório

Conforme as diferentes técnicas de aprendizagem automática vão sendo abordadas, determinados problemas vão sendo introduzidos e implementados neste projeto.

O código fonte deste projeto encontra-se disponível neste repositório Github.

2. Inteligência Artificial e Sistemas Cognitivos

O impacto da inteligência artificial (IA) na sociedade atualmente é um tema amplamente debatido. Há décadas que já se explora esta área, no entanto até a relativamente pouco tempo não existia muito desenvolvimento devido a capacidade computacional para realizar os problemas em tempo útil e estruturas pouco conseguidas do pouto de vista da eficiência.

Muitos pensam e argumentam que a IA tem vindo a melhorar a qualidade de vida do nosso dia a dia ao realizar que estes modelos são capazes de desenvolver tarefas simples de forma segura e eficiente. Contudo outros argumentam que a IA apresenta riscos perigosos como o risco de perda de privacidade e risco de existirem substituições de trabalhadores por máquinas e algoritmos.

Atualmente, o conceito de inteligência artificial consiste em sistemas com a capacidade de um computador realizar tarefas que geralmente são feitas por humanos. Estes sistemas dotados dos processos intelectuais característicos dos humanos formam-se através de um processo iterativo onde são realizados ajustes de novas entradas de dados para ajustar um modelo e por sua vez ter melhores resultados com este modelo.

Mais concretamente, os sistemas cognitivos fazem parte da capacidade de compreender e conhecer o processo metal através dos processos de interpretação, ou seja, são sistema capazes de utilizar as informações contidas no ambiente envolvente para tomar decisões de forma autónoma. Para que um sistema se considere racional é preciso este consigo realizar uma "ação certa" dado o conhecimento que possui, ou seja apresenta a capacidade de agir, no sentido de conseguir o melhor resultado possível perante os objetivos que se pretende atingir.

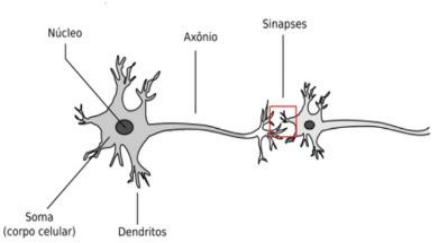
Existem imensas técnicas para proporcionar essa capacidade a uma máquina, das quais algumas serão abordadas neste projeto da seguinte forma organizada:

- Introdução teórica do tema, técnica ou algoritmo;
- Explicação do(s) problema(s) a resolver;
- Implementação;
- Testes e resultados.

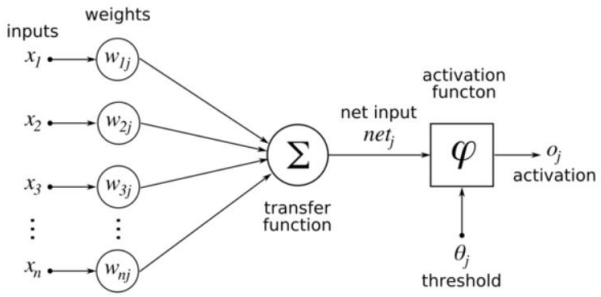
3. Parte 1 – Redes Neurais artificiais

3.1. Introdução Teórica

Esta primeira parte foca-se em estudar o comportamento de redes neuronais artificiais (RNA). As redes neuronais artificias são um modelo computacional que foi inspirado na forma como as redes neuronais biológicas no cerebro humano processam informações, ou seja, é uma tentativa de replicar o modelo biológico composto por neurónios interligados entre si que comunicam através das sinapses, passando impulsos elétricos (figura 1).



Equação 1 – Modelo de um neurónio biológico
(Exemplo gráfico)

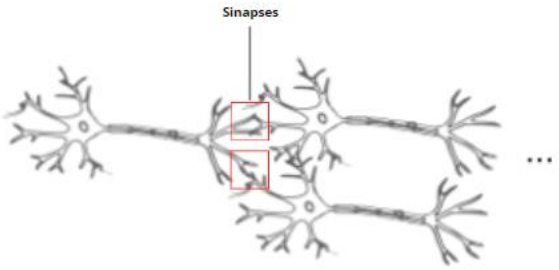


Equação 2 – Modelo de um neurónio artificial
(Exemplo gráfico)

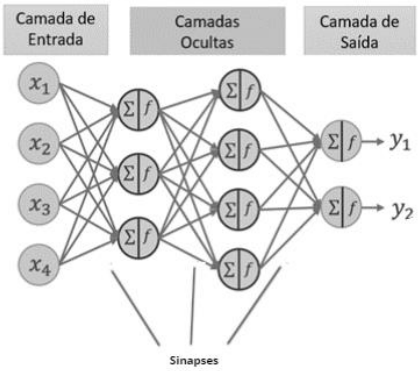
Como podemos verificar na figura 1 um neurónio biológico transmite através das sinapses impulsos para outras neurónios, já num neurónio artificial (figura 2) podemos verificar que este pode ser representado por um conjunto de entradas $\mathbf{x_i}$, em que cada entrada possui um peso associado $\mathbf{w_i}$, que simboliza a permeabilidade da sinapse, sendo a soma ponderada das entradas pelos pesos aplicada a uma função de ativação $\boldsymbol{\varphi}$, a qual determina se é produzida resposta ou não.

Um modelo de um neurónio artificial (Figura 2) permite resolver problemas linearmente separáveis como por exemplo problemas que definam uma fronteira linear (geometricamente) entre a região da resposta excitadora e a região da resposta inibitória.

Uma vez que um neurónio artificial é capaz de resolver apenas problemas linearmente separáveis, para problemas mais complexos, como por exemplo o problema dos XOR, é necessário construir um modelo de multicamada constituída por muitos neurónios como podemos ver nas figuras seguintes (figuras 3 e 4).



Equação 3 - Modelo de uma rede neuronal biológica (Exemplo gráfico)



Equação 4 – Modelo de uma neuronal artificial (Exemplo gráfico)

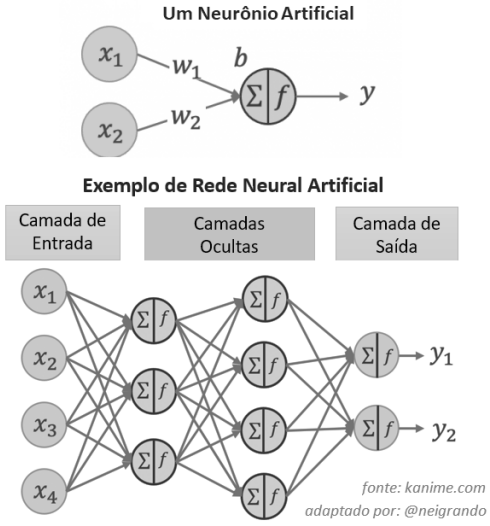
As redes neurais multicamada (figura 4) apresentam sempre uma camada de input e uma de output. Entre elas, poderão existir um número de camadas escondidas arbitrário, consoante a dimensão do problema que se pretende resolver.

As redes neurais, enquanto algoritmo de computação automatizado, apresentam grandes vantagens, porque se baseiam na estrutura do sistema nervoso do próprio se humano. No caso de algoritmos de retro propagação, o processo de treino pode ser muito demorado.

Além disso, uma grande desvantagem é o facto de, na maior parte das vezes, não se saber como uma rede chega a um determinado resultado, no sentido em que os modelos não apresentam um algoritmo pré-estabelecido que justifique as suas decisões (não é possível ver como as camadas ocultas se desenvolvem).

Esta primeira parte foca-se em estudar o comportamento de redes neuronais com o objetivo de conseguir detetar padrões e semelhanças num conjunto de amostras para ser capaz de prever qual o resultado em dados nunca introduzidos. Esta abordagem baseia-se no comportamento de neurónios biológicos e a ideia é simular o comportamento deles, já que é destes que o cérebro do ser humano é composto e permitem a que sejamos capazes de raciocinar e detetar e reconhecer padrões.

Para este objetivo 1, são usadas técnicas de aprendizagem artificial supervisionada. Isto que dizer que, ao treinar a rede, indica-se qual o resultado esperado de forma a informar a rede se a previsão foi correta ou incorreta, ajustando assim os seus parâmetros até convergir para uma solução aceitável. Numa abordagem não supervisionada já não seria preciso indicar qual o resultado suposto da previsão.

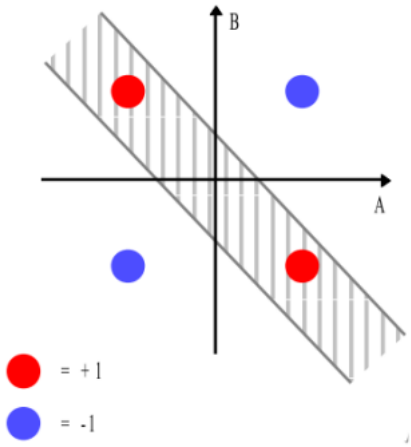


3.2. Problema XOR

3.2.1. Objetivo e Implementação

Nesta primeira parte das redes neuronais a implementação dos problemas foi realizada recorrendo á biblioteca SciKit-Learn e a linguagem de programação Python.

O primeiro exercício da primeira parte do projeto é onde explorado o problema clássico de redes neuronais, o XOR, que tem como objetivo prever um output da função logística XOR quando lhe é dado dois binários como input. Uma funcao logística XOR tem as seguintes entradas e saídas previstas, como são apresentadas na figura 5.



Equação 5 - Todas as entradas e saídas prevista da funcao logística XOR.

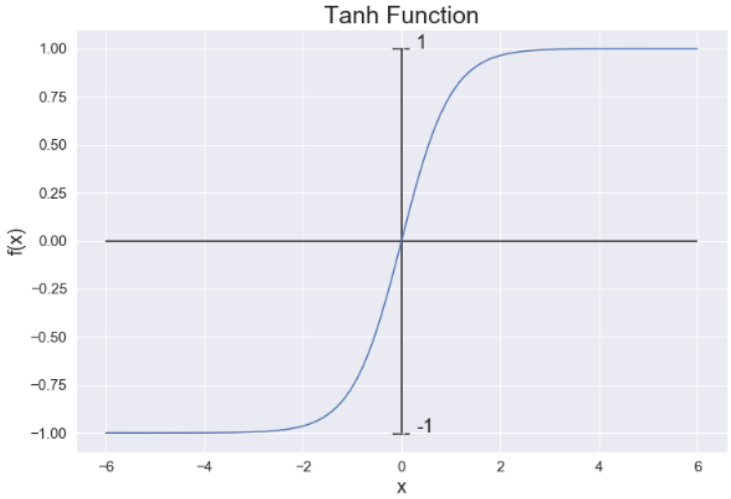
Para a realização da implementação deste problema, utilizaram-se as coordenadas dos pontos mencionados na figura 5 (**[-1, -1], [-1, 1], [1, -1], [1, 1]**) como inputs numa rede neuronal, com o objetivo de treinar a rede a resolver o problema de classificação e conseguir prever da melhor forma possível as entradas da função logística, retornando corretamente as respetivas classes (**[-1, 1, 1, -1]**) dos inputs.

Este processo foi desenvolvido de forma que se treinassem o mesmo modelo várias vezes para observar a consistência do modelo mesmo este começando com pesos diferentes (pois começava com pesos random sempre que se começava a treinar). Com isto, foram anotados os numeros das iterações onde os modelos conseguiram chegar a um critério de convergência de 0,3, ou seja, ao observarmos todos os modelos sempre que este não conseguia atingir o valor de loss de 0,3 era adicionada uma nota de NaN.

Com os dados definidos, passamos para a criação de uma rede. Contando que o problema do XOR é um problema de classificação de classes binarias (-1 e 1), onde o objetivo é fazer o reconhecimento dos padrões das classes -1 e 1 utilizou-se a função **MPLClassifier()** com os seguintes parâmetros de treino:

1. **Função de ativação (tanh)**

A função de ativação é a transformação não linear que fazemos ao longo do sinal de entrada, onde a saída é então enviada para a próxima camada de neurónios como entrada. A função escolhida foi a funcao Tanh, pois as classes de output para quais o modelo está a aprender são -1 e 1, ou seja, limita os valores entre -1 e 1. Esta limitação ocorre com a seguinte evolução, representada na figura x.



Equação 6 - Representação da função de ativação Tanh

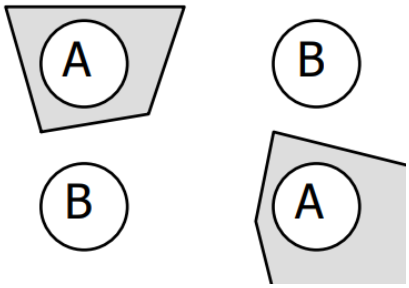
2. **Solver (sgd)**

O stochastic gradient descent ('sgd'), é um optimizador que atualiza os pesos com o recurso ao algoritmo de back-propagation, fazendo assim com que a rede aprenda ao longo das iterações. Isto é, para cada iteração, coloca-se um dos conjuntos de valores de entrada na entrada, observa-se o que está à saída (que inicialmente há de ser aleatório pois os pesos são iniciados aleatoriamente) e define-se um valor de erro que é calculado e utilizado para fazer a backpropagação de forma a ajustar os pesos da rede.

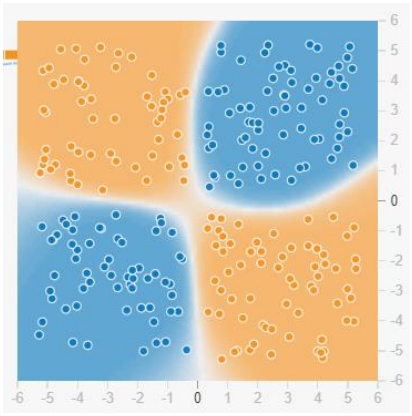
3. **hidden_layer_sizes** (2,) e (4,2)

Representa o número de neurónios nas camadas ocultas, onde o número é o número de neurónios e o número de número separados por virgulas á o número de camadas. No problema do XOR foi necessário utilizar uma camada escondida com dois neurónios (2,), pois a operação do XOR é uma combinação de dois operações, AND e OR e apenas é necessário dividir as classes com duas retas de forma geométrica como podemos observar na figura 5 anteriormente representada.

No entanto, nas diversas execuções com diversos parâmetros foi possível observar uma outra estrutura que conseguiu chegar a mesma convergência (0,3 de loss) de forma mais rápida e eficiente nas diversas execuções. Esta estrutura era constituída por um número de camadas escondidas igual a (4, 2, 1). Isto acontece porque á medida que aumentamos o número de neurónios, o modelo consegue classificar os pontos com maior precisão e quando aumentamos o número de camadas ocultas, os tipos das regiões de decisão, deixam de ser lineares e passam a ser uma combinação não linear de limites de decisão individuais. Uma outra forma de perceber é imaginar, em nível abstrato, que existem vários classificadores combinados de maneira não linear com o objetivo de encontrar um plano de decisão não linear. Nas seguintes figuras 7 e 8 podemos verificar exemplos destas separações não lineares.



Equação 8 - Representação gráfica de uma separação não linear.



Equação 7 – Exemplo de uma separação não linear.

Com o objetivo de dizer ao modelo para deixar de treinar por algumas iterações, utilizaram se alguns parâmetros de paragem como o Max_iter (=30000), que se refere ao máximo de iterações que o modelo deve executar, o Tol (=0.0000001) e o n_iter_no_change (=500), que se referem ao máximo de numeros de episódios em que não houve tol de melhoria, ou seja, sempre que os 500 episódios consecutivos falharem em diminuir a perda do treino em pelo menos 0.0000001 o modelo para.

Como um segundo objetivo deste exercício era ficar a conhecer como alguns parâmetros de treino evoluíam com diversos parâmetros, foi atribuída a seguinte uma lista de valores como parâmetros:

1. Learning_rate_init ([0.05, 0.25, 0.5, 1, 2]):
2. Shuffle (False ou True):
3. Momentum ([0, 0.5, 1]):

3.2.2. Testes e Resultados

Como um segundo objetivo do problema do XOR era ficar a compreender como alguns parâmetros de treino influenciam a forma como a rede preformava, por esse motivo, foram atribuídas as seguintes listas de valores como parâmetros:

1. Learning_rate_init ([0.05, 0.25, 0.5, 1, 2]):
2. Shuffle (False ou True):
3. Momentum ([0, 0.5, 1]):

Efeito da taxa de aprendizagem

A tabela da seguinte (Tabela 1) enumera algumas execuções para as diferentes taxas de aprendizagem (r), indicando o número de iterações até à convergência. No caso do modelo na atinguir uma convergência de taxa de erro de **0.3**, a execução é marcada como **nan** (Not a Number).

α = 0 e shuffle= False					
Execução	**r = 0.05**	**r = 0.25**	**r = 0.5**	**r = 1**	**r = 2**
1	1410	nan	132	153	nan
2	602	817	119	82	nan
3	927	143	nan	nan	nan
4	nan	404	444	61	71
5	nan	nan	75	152	34
6	nan	nan	144	nan	nan
7	813	356	128	50	33
8	nan	nan	nan	72	nan
9	nan	nan	110	nan	30
10	nan	458	112	105	nan
Média:	938	435,6	158	96,4286	42
Contagem	4	5	8	7	4

Tabela 1 - Efeito a taxa da aprendizagem (Momento - 0 e Shuffle - False)

Ao observarmos as execuções de quando se varia a taxa de aprendizagem consegue-se concluir que a taxa de aprendizagem faz com que o número de execuções diminua de forma drástica. Esta variação não é uma surpresa, pois a taxa de aprendizagem é o parâmetro que efetua atualizações diretamente nos pesos, de forma que o modelo consegue "saltar".

Uma vez que a taxa de aprendizagem é pequena o modelo efetua saltos mais pequenos em cada iteração pelo que precisa de executar mais iterações para chagar ao mínimo global, sendo que o contrário acontece a uma taxa de aprendizagem menor pois salta a passos largos.

Tanto a taxa de aprendizagem muito pequena e muito grande tem vantagens e desvantagens. Se o valor da taxa for muito pequeno, o modelo efetua saltos mais pequenos em cada iteração pelo que precisa de executar mais iterações para chagar ao mínimo. Uma desvantagem da taxa pequena é que pode entrar num mínimo local e não conseguir sair para encontrar o mínimo global.

Já no caso de a taxa ser muito grande, são efetuadas menos iterações e a aprendizagem por ser grande pode passar pelos mínimos locais facilmente, no entanto pode também passar pelos mínimos globais, não conseguindo assim entrar, ou mesmo que entre pode vir a afastar-se facilmente. Por isso, o ideal é procura-se um valor intermédio que seja adequado.

É ainda possível observar pelas colunas da taxa de aprendizagem (ou pela linha da contagem), que a taxa de aprendizagem que mais não conseguiu atingir o valor do critério da convergência foi a r=0.05, o que indica que a taxa é tao lenta a atualizar os pesos que não conseguiu chegar ao erro mínimo. Já nos outros casos de nan da tabela, o modelo pode ter ficado preso em mínimos locais de onde não foi capaz de sair.

- ## Efeito da introdução de um termo de momento

Para perceber o efeito do momento, utilizou-se o mesmo processo anterior mas com um termo de momento igual a 0,5 e outro momento 1. Nas tabelas seguintes (tabela 2 e 3) são desmotrados os resultados ao se adicionar o termo de momento de 0.5 e 1.

α = 0.5 e shuffle= False					
Execução	**r = 0.05**	**r = 0.25**	**r = 0.5**	**r = 1**	**r = 2**
1	nan	86	nan	131	48
2	914	106	134	nan	17
3	610	255	nan	21	31
4	1213	136	106	56	nan
5	641	155	134	81	nan
6	728	99	nan	43	15
7	2473	123	41	nan	24
8	380	nan	95	98	13
9	1822	nan	93	32	18
10	822	150	118	nan	15
Média:	1067	138,75	103	66	22,625
Contagem	9	8	7	7	8

Tabela 2 - Efeito da instrução de um termo de momento, com momento de 0,5 e shuffle=false

α = 1 e shuffle= False					
Execução	**r = 0.05**	**r = 0.25**	**r = 0.5**	**r = 1**	**r = 2**
1	69	nan	nan	16	14
2	79	44	14	nan	25
3	119	nan	41	12	nan
4	nan	33	58	15	8
5	nan	48	65	21	nan
6	76	31	nan	12	nan
7	nan	33	22	17	15
8	165	26	20	nan	nan
9	117	52	29	14	nan
10	84	44	42	15	16
Média:	101,286	38,875	36,375	15,25	15,6
Contagem	7	8	8	8	5

Tabela 3 - Efeito da instrução de um termo de momento, com momento de 1 e shuffle=false

O termo de momento foi utilizado para realizar o treino do classificador MLPClassifier, pois este parametro ajuda a evitar que a rede neuronal fique presa em minimos locais (local onde a funcao de custo e baixa). O momento serve como "impulso" para que exista uma hipotese do modelo sair de um minimo local. Este parametro pode variar entre 0 e 1 , onde quanto maior o seu valor maior será o "impulso" dado ao modelo.

Ao adicionarmos o momento á rede neunal foi possivel ver que o efeito foi positivo na velocidade de convergencia do modelo. Em comparação com um modelo sem momento, o modelo ao qual se introduziu o momento de 0.5 mostrou-se mais eficaz, pois consegui-se ver (ao comparar as tabelas 1 e 2) que o erro do modelo com 0.5 de momento efetua menos iteracoes.

Por outro lado, ao utilizar o momento igual a 1 (tabela 3) podemos verificar que ocurreram muit mais execucoes onde o modelo não conseguiu convergir (valores "nan" da tabela), o que nnos indica que o valor

de momento de 1 é tao elevado que está a dar demasiado "impulso", fazendo com que o modelo não consiga convergir para o erro desejado.

Com estes resultado, podemos concluir que o momento de 0.5 é o mais indicado entre as opcoes de 0, 0.5 e 1.

- ## Efeito da apresentação das amostras de treino com ordem fixa ou aleatória

Com o intuito de estudar e perceber o efeito que o parâmetro de treino com ordem fixe ou ordem aleatória), realizaram-se testes com o modelo usando o parâmetro de shuffle igual a true.

O objetivo aqui seria analisar se o uso de amostras de treino em ordem aleatória melhora o desempenho do modelo quando comparado com o treino do modelo que utiliza a ordem fixe. Para isso foi utilizado o momento igual a 0,5, pois foi o melhor parâmetro nos testes anteriores. Os resultados obtidos nesta análise foram apresentados numa tabela, como demostra tabela 4:

α = melhor resultado(momento = 0.5) e shuffle= True					
Execução	**r = 0.05**	**r = 0.25**	**r = 0.5**	**r = 1**	**r = 2**
1	755	303	89	65	36
2	895	nan	130	47	36
3	1588	nan	69	63	44
4	725	192	91	nan	69
5	541	136	178	102	nan
6	629	222	nan	37	34
7	1099	206	135	nan	43
8	653	210	111	60	nan
9	1010	178	nan	27	63
10	916	256	nan	44	34
Média:	881,1	212,875	114,714	55,625	44,875
Contagem	10	8	7	8	8

Tabela 4 – Efeito da apresentação das amostras de treino com aleatória, com momento de 0,5.

Ao utilizarmos o parâmetro de treino de ordem aleatória, podemos verificar que o modelo tem benefícios, principalmente porque agora o modelo esta exposto aos dados de treino de forma na sequencial o que evita que o modelo aprenda de uma so forma, mas tambem generalizando.

Este parâmetro é importante pois ajuda a evita o overfiting, uma vez que a ordem é variada e o modelo tem de ter a capacidade de se ajustar melhor aos dados de forma a não se ajustar demasiado e possibilitando também a melhor previsão de novos dados.

Ao comparar o numero de iterações e o numero de contagem (numero de vezes que o modelo conseguiu convergir em 10 execuções) da tabela 4, podemos verificar que o as media dos numero de iterações não alteram muito, no entanto o modelo consegue convergir mais vezes, provavelmente porque este modelo consegue generalizar mais .

Em conclusão podemos dizer que utilizar a ordem aleatória é uma boa pratica para trabalhar com modelos de redes neuronais pois ajuda o modelo a generalizar, não ocorrendo tantas vezes o overfiting ao realizar o treino.

- ## Efeito de uma codificação binária ou bipolar

Por fim, para avaliar o efeito de uma codificação binaria ou bipolar nos modelos de redes neuronais, efetuou-se um teste onde se utilizou utilizar-se-á uma codificação binária **[0, 1]** ao invés de uma codificação bipolar **[-1, 1]**. Uma vez que se altera os dados é necessário efetuar uma alteração na funcao de ativação, pois a tangente hiperbólica produz valores entre -1 e 1, enquanto a codificação binaria produz apenas valores entre 0 e 1. Ao testar o modelo com a tanh, verifica-se que a taxa de erro é elevada, logo foi utilizada a funcao logistic que lida com esta nova codificação de dados. Na tabela 5 podemos observar os resultados dos testes:

α = melhor resultado(momento = 0.5) e shuffle= False					
Execução	**r = 0.05**	**r = 0.25**	**r = 0.5**	**r = 1**	**r = 2**
1	1	4	3	3	2
2	9	5	3	3	2
3	9	6	3	2	3
4	5	5	3	2	2
5	9	4	3	3	2
6	10	4	3	3	2
7	9	5	3	3	2
8	6	4	4	3	2
9	8	5	4	2	3
10	10	4	3	3	2
Média:	7,6	4,6	3,2	2,7	2,2
Contagem	10	10	10	10	10

Tabela 5 - Efeito da codificação binaria em vez de codificação bipolar.

Ao observarmos a tabela 5 podemos verificar que o modelo conseguiu convergir em todas as iterações o que demostra uma melhoria do desempenho. Verifica-se que ambas as codificações conseguem obter bons resultados no problema do xor, no entanto aprendeu-se dependendo esta métrica depende de problema para problema, e que a codificação binária foi mais adequada para este caso específico. Além disso, é importante notar que a escolha da função de ativação também foi crucial para o desempenho do modelo, pois a utilização da função logística permitiu uma melhoria significativa na precisão do modelo.

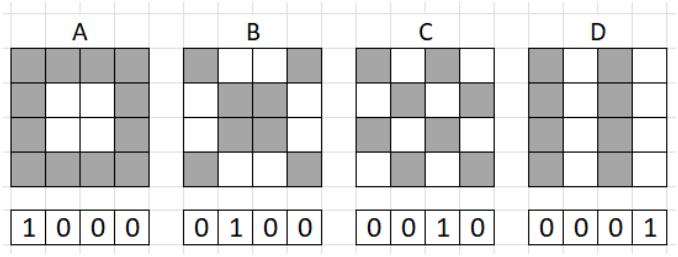
3.3. Problema dos Padrões de imagens

3.3.1. Implementação

Este problema consiste em criar um protótipo de rede neural usando uma biblioteca (neste caso o scikit-learn) para aprender e reconhecer padrões de imagem específicos. O objetivo deste problema é treinar uma rede para reconhecer dois padrões diferentes, designados como padrão A e padrão B, e classificá-los corretamente como pertencentes às classes 1 e 0, respetivamente.

Os padrões A e B são fornecidos como matrizes de 4x4, onde cada elemento da matriz representa um pixel de uma "imagem", com valores 0 ou 1. A solução para este problema pode ser encontrada através do uso de redes neurais com uma ou mais camadas ocultas, treinadas com algoritmos de otimização para aprender a reconhecer e classificar esses padrões.

Foram fornecidos a rede mais padrões para "confundir" a rede e para que esta consiga não so diferenciar os dois padrões entre um e outro, como detetar entre múltiplos outros padrões quais é que são so padrões A e b. Na figura **x** seguinte são demostrados os padrões utilizados.



Equação 9 - Padrões utilizados na rede neuronal para treinar a identificação dos padrões A e B.

Neste problema também se recorreu a utilização de padrões aleatórios para ajudar no treino desta rede, pelo que foi necessário criar uma funcao que se cria padrões aleatórios. Apos isto juntou-se todos os dados e utilizou-se os conhecimentos do exercício anterior para construir uma rede neuronal.

3.3.2. Implementação

Recorreu-se mais uma vez a biblioteca sklearn para classificar este problema, mais especificamente há classe multi-layer pecetron classifier (MLPClassifier). Nesta classe prosseguiu-se em fazer uma configuração da rede com os seguintes parâmetros:

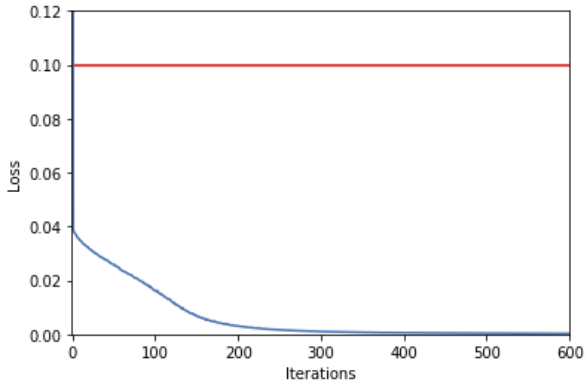
1. hidden_layer_sizes: (32,8), que especifica que a rede terá uma camada escondida com 16 neurônios. Neste caso utilizou-se na primeira camada 32 neuronios porque se imaginou que como existem (4x4) 16 elementos em cada matriz onde cada um pode ser 0 ou1, logo (16x2) 32 neuronios parecem ser o número indicado. Já para uma segunda camada foi necessário efetuar alguns testes onde o resultado foi 8 neurónios, o qual melhor preformou.
2. activation: relu, que especifica que a função de ativação será a relu, que é geralmente usada em problemas de machine learning entre os valores 0 e 1. Alem de ser o intervalo indicado para este problema, utilizou-se esta funcao devido a suas propriedades de não saturação (não ficar "estagnado" em valores próximos de zero) e a sua capacidade de aprender relações lineares e não-lineares.

3. solver: 'sgd', que especifica que o algoritmo de otimização será o gradiente descendente estocástico, que atualiza os pesos da rede em cada amostra de treinamento.
4. learning_rate_init: 0.1, que especifica a taxa de aprendizado inicial.
5. max_iter: 30000, que especifica o número máximo de iterações para o algoritmo de otimização.
6. Shuffle, tol, n_iter_no_change: Estes paramentros utilizados com no exercício anterior, pois baralhar os dados é sempre bom não a rede na se habituar aos dados, o tol para que exista um limite de tolerância e o n_iter_no_change para indicar o numero de iterações consecutivas.

3.3.3. Testes e Resultados

Com base nos resultados obtidos, é possível concluir que a rede neural treinada apresentou um desempenho altamente satisfatório na tarefa de reconhecimento de padrões de imagem. A taxa de erro obtida foi de cerca de 9.96*10^-5, o que indica que a rede neural conseguiu classificar corretamente a maioria dos padrões de teste. Isso demonstra a capacidade da rede neural em aprender e reconhecer padrões complexos e variados.

Durante o treinamento, foi observado que a taxa de erro diminuiu rapidamente nas primeiras 200 iterações, o que sugere que a rede neural aprendeu rapidamente a reconhecer os padrões A e B, como podemos ver na seguinte figura. Isso pode ser atribuído ao uso de um algoritmo de otimização eficaz, o gradiente descendente estocástico (SGD), e às escolhas de parâmetros de treinamento, como a taxa de aprendizado e a função de ativação.



Equação 10 - Desenvolvimento da perda ao longo das iterações de aprendizagem da rede.

Além disso, a escolha da arquitetura da rede foi fundamental para o sucesso do treinamento. A utilização de duas camadas escondidas, uma com 32 neurônios e outra com 8 neurônios, permitiu evitar tanto o overfitting quanto o underfitting. Isso mostra que, para problemas de reconhecimento de padrões de imagem, uma arquitetura de rede com duas camadas escondidas pode ser uma boa opção. A escolha das camadas e dos neurônios foi importante para garantir uma boa capacidade de generalização da rede neural.

Após avaliar o modelo da rede neural criado utilizando os padrões A, B, C, D e a matriz random, foram feitas previsões dos mesmos padrões e o desempenho do modelo é apresentado na seguinte figura.

```
Score: 100.0
Prever o padrao A: [[1 0 0 0]] True
Prever o padrao B: [[0 1 0 0]] True
Prever o padrao C: [[0 0 1 0]] True
Prever o padrao D: [[0 0 0 1]] True
```

Equação 11 - Previsões ao utilizar o modelo

Ao analisarmos os resultados obtidos, notamos que o nosso modelo de rede neural teve um desempenho excecional, com um score de 100% na previsão de todos os padrões testados (A, B, C, D e Matriz aleatoria). Isso indica que a rede neural foi capaz de aprender e reconhecer todos os padrões de forma precisa e eficiente. Este resultado é especialmente impressionante, considerando a complexidade dos padrões testados e a quantidade de dados aleatórios utilizados no conjunto de treinamento.

Concluindo, este problema de reconhecimento de padrões de imagem é uma boa forma de se familiarizar com a implementação de redes neurais e como elas podem ser usadas para resolver problemas de classificação. No entanto, é importante relembrar que quase sempre os 100% de score pode ser uma consequência de overfiting e ou de dados extremamente bem separados o que não acontece muito frequentemente no mundo real.

3.4. Problema 2 - DIGITS RECOGNITION

3.4.1. Objetivo

Neste problema o objetivo é desenvolver um modelo de classificação de dígitos utilizando o dataset "digits recognition", também conhecido como MNIST. Este dataset contém 70.000 imagens de dígitos manuscritos (de 0 a 9) com resolução de 28x28 pixels. Cada imagem foi pré-processada e normalizada para facilitar o treinamento do modelo. O objetivo final é treinar um modelo capaz de classificar corretamente os dígitos nas imagens, dado os valores dos pixels como entrada. Com este problema, espera-se que se possa fornecer uma base sólida para o desenvolvimento de modelos de reconhecimento de imagens mais complexos e para a comparação do desempenho de diferentes algoritmos de aprendizado de máquina.

3.4.2. Implementação e resultados

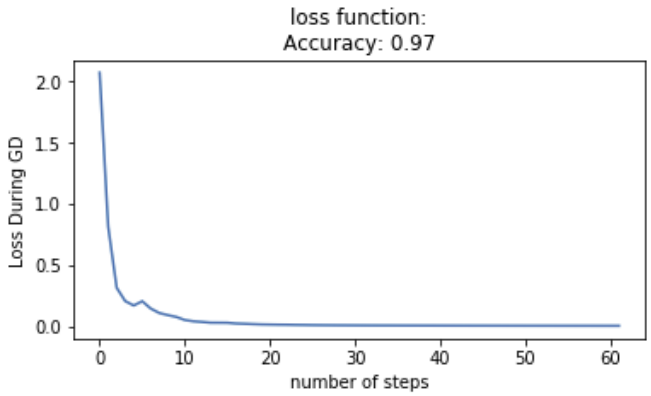
Para chegar a este objetivo desenvolveu-se uma implementação de um classificador Multi-Layer Perceptron (MLP) para realizar uma tarefa de classificação de multiclasse, mais especificamente para o conjunto de dados de reconhecimento de dígitos.

O primeiro passo passou por dividir os dados em conjuntos de treino e teste utilizando a função train_test_split da biblioteca scikit-learn. Esta função inclui os dados de características (X), ou seja, as matrizes de números inteiros dos dígitos e os dados target (y) que são as avaliações verdadeiras dos dados e divide-os num conjunto de formação (X_train e y_train) e num conjunto de testes (X_test e y_test) com um tamanho de teste especificado (neste caso 0,2, o que significa que 20% dos dados serão utilizados para testes).

De seguida, utiliza-se a classe MLPClassifier com os respetivos parâmetros de congfiguracao do modelo, incluindo o número de camadas ocultas (neste caso uma com 100 neurónios), a função de ativação (logística) e o algoritmo de optimização (SGD). Com isto, o modelo é treinado com os dados de treino usando o método “fit” e os dados para treinar. Depois disto, o modelo é utilizado para prever com os de teste e podemos fazer a avaliacao da previsão do modelo.

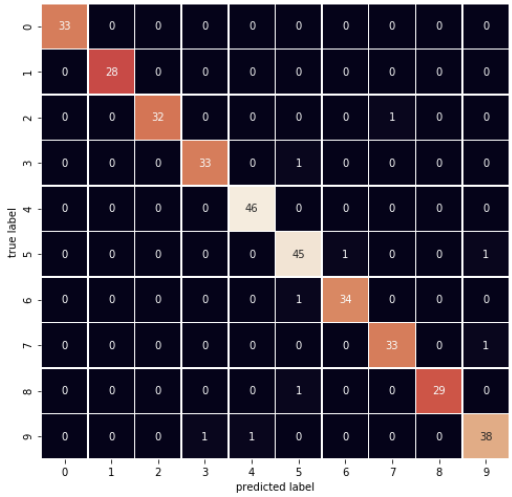
O modelo de classificação de dígitos obteve 97% de accuracy, o que é um resultado bastante bom, pois demostra que o modelo é capaz de classificar corretamente 97% dos dígitos nas imagens dos conjunto de teste.

Um dos gráficos utilizados para avaliar o desempenho do modelo foi a curva de perda (loss curve). Esta curva apresentada na figura seguinte, mostra como a perda (ou erro) do modelo diminuiu ao longo das iterações do treino. Pode-se observar que a curva decresceu rapidamente até cerca da 20 iteração e depois estabilizou, demostrando que o modelo convergiu rapidamente para uma solução ótima.



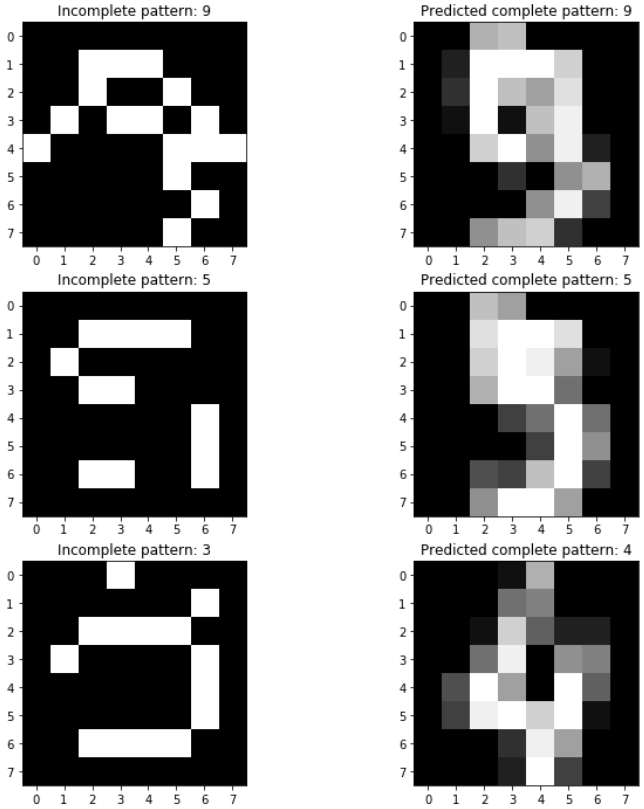
Equação 12 - Gráfico da curva de perda (loss curve) do modelo

Outra ferramenta que se utilizou para avaliar o desempenho do modelo foi a matriz de confusão, que demonstra como é que cada digito foi classificado pelo modelo em relação aos dígitos corretos. Observando a diagonal da matriz, pode-se ver que esta bastante preenchida, o que indica que o classificador conseguiu classificar corretamente a maioria dos dígitos, como podemos ver na figura seguinte:



Equação 13- Matriz de confusão do classificador

Por fim, foram realizadas algumas previsões com imagens incompletas dos números 9, 5 e3 de propósitos, para avaliar a capacidade do classificador "adivinhar" qual era o número na imagem, como podemos ver na imagem seguinte:



Equação 14 -Padrões incorretos e as respetivas classificações do modelo.

Com isto, é possível verificar que o modelo tem a capacidade de generalizar e de lidar com imagens incorretas, o que na realidade pode ser importante o modelo ser capaz de lidar com esta situações. O fato de o modelo ter obtido um bom desempenho mesmo com imagens incorretas ou incompletas indica que este é robusto.

3.4.3. Problemas e técnicas para aperfeiçoar

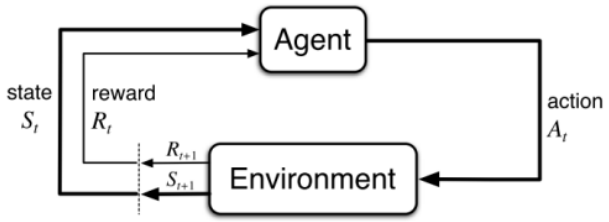
Como podemos observar na figura anterior 14, o classificador não conseguiu classificar corretamente o último numero 3, o que pode ter ocorrido por várias razoes como tamanho, orientação ou ate mesmo ter tido pouco exemplos de aprendizagem.

4. Parte 2 - Aprendizagem por Reforço

4.1. Introdução Teórica

Aprendizagem por reforço é uma área da aprendizagem automática que se concentra em como um ou mais agentes devem tomar decisões para maximizar a recompensa ao longo do tempo. O agente sendo uma entidade que aprende através de uma série de ações e observações do ambiente, tenta maximizar a sua recompensa ao longo do tempo. O ambiente é o local que o agente irá interagir.

Na aprendizagem por reforço, o agente ao realizar uma ação no seu ambiente, recebe como resultado recompensas ou punições do seu comportamento. Na seguinte figura 11 podemos ver este esquema de interações entre o agente e o ambiente.



Equação 15 - Interações entre o agente e o ambiente

Nesta interação o agente tem como objetivo maximizar a soma das recompensas ao longo do tempo, uma vez que para isso acontecer precisa tomar decisões que maximizam a sua recompensa. A agente assim sendo, precisa planear as suas ações com base no seu estado atual e na sua recompensa futura.

Como podemos verificar na figura 15, para cada interação do agente com o ambiente, este recebe um estado e uma recompensa. No entanto, no início do agente começa sem qualquer aproveitamento das ações que toma, sendo que recebe sempre uma reward negativa por ser o início. Na no lado oposto, podemos ver que um agente já descobrir um caminho para chegar ao alvo, mas que não é o caminho mais eficiente por isso e por outras razoes é que existe o dilema de um agente fazer a exploração e aproveitamento (explore-exploit).

Ou seja, a exploração refere-se a tentar novas ações para descobrir quais são as melhores, enquanto aproveitamento refere-se a continuar tomando as ações que já foram aprendidas como as melhores. É aqui que entram técnicas como o Greedy e EGreedy.

A técnica Greedy é uma abordagem determinística, onde o agente escolhe a ação com maior recompensa esperada e a técnica EGreedy é uma abordagem estocástica onde o agente escolhe a ação com maior recompensa espera com uma determinada probabilidade (fator de exploração) e escolhe a ação aleatória com uma probabilidades.

Existem dois tipos de aprendizagem por reforço, on-policy e off-policy. On-policy utiliza a mesma política de seleção de ação para comportamento e exploração de todas as ações (por exemplo, estratégia **EGreedy**), enquanto off-policy utiliza políticas diferentes de seleção de ação para comportamento e para propagação de valor (algoritmo **QLearning**).

Algoritmo Q-Learning

Introdução Teórica

Um dos algoritmos que se vai implementar é o algoritmo Q-learning. O algoritmo Q-learning é um algoritmo de aprendizagem por reforço off-policy utiliza uma técnica interativa para estimar a função Q, onde cada iteração o agente escolhe uma ação num estado atual s, executando uma ação e recebe uma recompensar e uma transição para um estado s'.

A função Q é atualizada usando a equação de Bellman:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

Equação 16 16 - Equação de Bellman

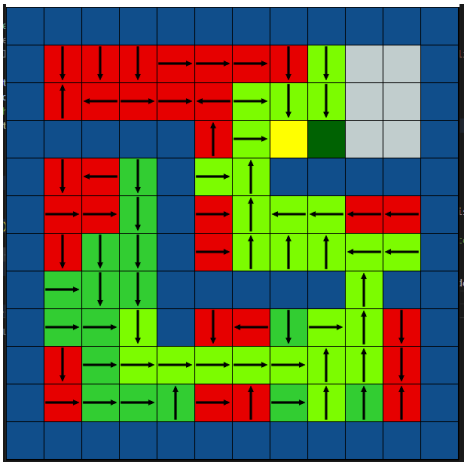
Onde s é o estado, a é a ação, α é a taxa de aprendizado, γ é o fator de desconto de recompensa futura e max(Q(s', a')) é a ação com a maior estimativa de recompensa futura no novo estado s'.

Implementação e resultados

A implementação do algoritmo Q-learning envolveu os seguintes passos:

1. Inicialização dos valores iniciais.
2. Seleção da Acão: É escolhida a Acão que o agente irá tomar no estado atual s de acordo com a política.
3. Execução da Acão: A Acão é executada e o agente recebe uma recompensar e a transição do estado novo s'.
4. Atualização da memoria/tabela dos dados Q (s, a)
5. Iteração dos passos anteriores até que o agente consiga tomar decisões no ambiente em cada estado.

Apos a execução deste algoritmo em dois ambientes que podemos verificar nas seguintes imagens 11 e 11, podemos verificar o mecanismo de aprendizagem do algoritmo.



Equação 17 - Agente a utilizar o método Q-learning para aprender a chegar ao algo do ambiente 1

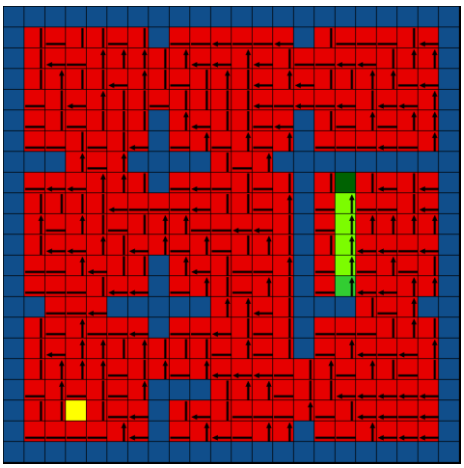


Figura 1 - Agente a utilizar o método Q-learning para aprender a chegar ao algo do ambiente 2

Algoritmo Q-Learning com memoria de experiência (QME)

O Q-Learning com memória de experiência (QME) é uma variante do algoritmo Q-Learning que usa uma memória de experiência para armazenar as interações passadas do agente com o ambiente. Isso permite que o agente aprenda com as experiências passadas e acelera o processo de aprendizagem.

Implementação e resultados

Apos a execução deste algoritmo nos dois ambiente fornecidos que podemos verificar nas seguintes imagens 12 e 22, conseguimos observar que com a memoria das experiências passadas o agente é capaz de aprender de forma mais rápida.

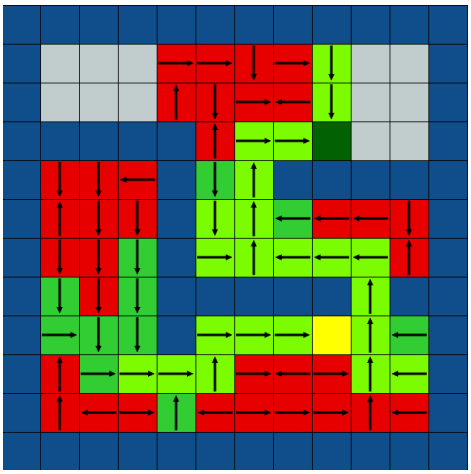


Figura 2 - Agente a utilizar o método QME para aprender a chegar ao algo do ambiente 1

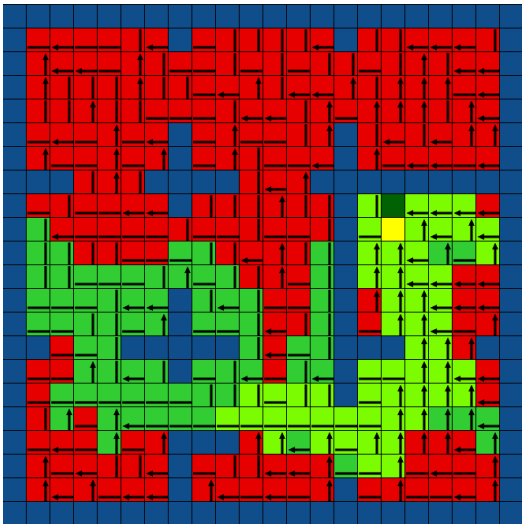


Figura 3 - Agente a utilizar o método QME para aprender a chegar ao algo do ambiente 2

Algoritmo Dyna-Q

Dyna-Q é uma variante do algoritmo Q-Learning que combina a aprendizagem por reforço com planejamento com modelos. Sendo assim o algoritmo Dyna-Q, implementa um passo a mais do que o Q-learning que é o ponto a seguir ao ponto 4, que utiliza o modelo contruído para simular ações e transições de estado e atualiza a memoria/tabela com base nessas simulações.

Enquanto o Q-Learning apenas usa a informação da interação real do agente com o ambiente para atualizar a memoria/tabela Q. Isso permite que o Dyna-Q explore mais eficientemente o ambiente.

Implementação e resultados

Apos implementar o algoritmo Dyna-Q, foi possível ter um agente que utiliza-se este algoritmo nos seguintes ambientes das figuras 33 e 33.

Na implementação utilizaram-se os parâmetros 1) alfa, na aprendizagem por reforço classe **ReinforcedLearning**) que simboliza o fator de aprendizagem e indica ao agente o quão ele deverá ter em conta esse tipo de situações e o parâmetro 2) gama, também na aprendizagem por reforço, que simboliza o fator de desconto.

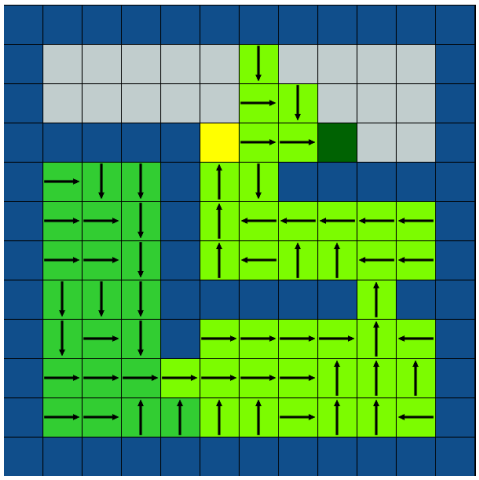


Figura 5 - Agente a utilizar o método Dyna-Q para aprender a chegar ao algo do ambiente 1

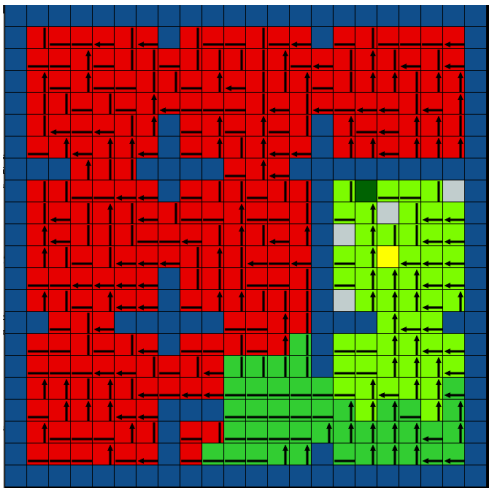


Figura 4 - Agente a utilizar o método Dyna-Q para aprender a chegar ao algo do ambiente 2

4.2. Análise dos resultados

Todos os três algoritmos de aprendizagem por reforço utilizados neste trabalho compartilham semelhanças entre eles, sendo as semelhanças as seguintes:

1. Como são algoritmos que se baseiam no mesmo algoritmo Q-learning tem utilizam todos a técnica de valorização de estado-ação para aprender a sua política.
2. Compartilham todos a equação de Bellman para atualizar os seus valores Q para cada estado ação com base na sua recompensa esperada futura.
3. Uma vez que são variações do mesmo algoritmo, tem de ser baseado na interação do agente com o ambiente, sendo que que o agente tem o mesmo objetivo de encontrar a política ótima para cada ambiente.

Numa visão geral, o algoritmo Q-learning foi o algoritmo que demorou relativamente mais tempo, principalmente no ambiente 2. Se olharmos para os outros algoritmos QME e Dyna-Q, observou-se que ambos tiveram performa-se semelhantes, pelo que para se fazer a distinção dos mesmos ter-se-ia de recorrer a outras métricas.

Algumas das razoes para que o Qlearning tenha demorados mais do que os algoritmos QME e Dyna-Q foram as seguintes:

1. O Dyna-Q ter a vantagem de utilizar um planeamento, ou seja, utilizou simulações de interações para planear as ações.
2. Os algoritmos QME ter a vantagem de utilizar uma memória de experiencia para armazenar e aprender as interações passada do agente.
3. O Algoritmo Q-learning ter dificuldades em situações em que o ambiente é muito grande pois se existem muitos estados e ações na sua memória/tabela este pode ter dificuldades em atualizar o que o torna mais lento.

Para concluir, os algoritmos de aprendizagem por reforço Dyna-Q, Q-learning e Q-Learning com memória de experiência (QME) são ferramentas poderosas para resolver problemas de tomada de decisão em ambientes incertos ou dinâmicos. Cada algoritmo tem as suas vantagens e desvantagens, no entanto para cada problema deve-se escolher aquele que é o ideal para as necessidades do problema.

Para a escolhe devemos ter em conta que o algoritmo Q-learning é o mais fácil de implementar , o tem a habilidades de acelerar o processo de aprendizagem, por saber as experiencias passadas o Dyna-Q consegue combinar aprendizagem por reforço e planeamento para acelerar o processo.

5. Parte 3 - Raciocínio Automático para Planeamento

Nesta parte do trabalho, são abordadas as técnicas e métodos de raciocínio automático para planeamento, que é uma área da inteligência artificial que se concentra em encontrar ações que levam a um objetivo, ou seja, verificar qual é a sequência de ações que o agente tem que percorrer para chegar a um estado final de maneira eficiente.

Este processo depende sempre da existência de um modelo de um mundo e um agente com ações disponíveis a serem executadas neste mundo, pois é necessário este modelo saber disso para conseguir construir um plano no mundo.

5.1. Procura em espaços de estados A* ponderada

Introdução Teórica

Os Problemas em espaços de estados são problemas onde alem de querer encontrar um caminho até um alvo, tem se como objetivo encontrar um caminho mais curto entre um ponto de partida e um ponto de chegada.

Um dos algoritmos mais populares utilizados para resolver este problema é a A* ponderada, pois este algoritmo faz parte dos métodos de procurar de informada, o que significa que este algoritmo utiliza informações adicionais para a avaliação de estados e uma funcao heurística que ajuda o agente a procurar.

O algoritmo A* ponderada é uma variante do algoritmo de procura Melhor-Primeiro(best-fisrt), que prioriza a expansão dos nós com a menor avaliação do estado, garantindo assim que se escolhe o melhor caminho ou seja, o mais curto.

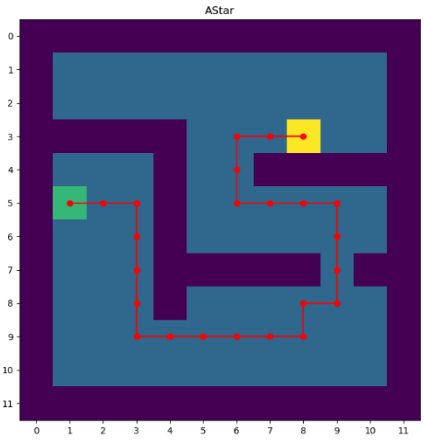
A principal diferença entre o A* ponderado e o best-Fist é a utilização da funcao heurística h(n) que estima o custo para o qual o agente precisa para chegar ao objetivo a partir de um determinado nó. Esta função em conjunto com o custo real, g(n), f(n)=g(n) + h(n), combina a eficiência do algoritmo Best-First com a precisão da procura informada, garantindo um procura eficaz para encontrar uma solução ótima.

Implementação e resultados

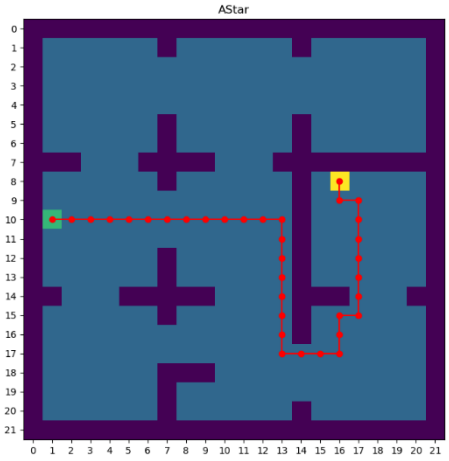
O algoritmo de procura A* é um dos métodos mais populares para resolver problemas de procura em espaço de estados. Para avaliar a eficácia deste algoritmo, utilizaram-se dois ficheiros fornecidos para o trabalho com diferentes mundos, para executar o método e comparar os resultados obtidos.

Os mundos fornecidos foram previamente definidos com obstáculos e os pontos de partida e de chegada do agente, para que o algoritmo tivesse que procurar o caminho mis curto entre estes dois pontos.

Além disso utilizou-se uma função heurística admissível para garantir a escolha do caminho mais curto, sendo que esta foi a função de distância de manhattan, que é uma medida de distância entre dois pontos na qual é realizada a soma da diferença entre x e y em cada dimensão.



*Equação 18 - Percurso sao solução do problema no ambiente 1 ao utilizar a procura A**



*Equação 19 - Percurso sao solução do problema no ambiente 2 ao utilizar a procura A**

Como podemos ver nas figuras 18 e 9 em ambas as execuções foi possível o algoritmo de procura A*chegar a uma solução ótima, demostrando assim a eficiência do algoritmo à procura o caminha mais curto, com uma complexidades temporal e espacial aceitável.

No entanto neste problema foram utilizados ambiente pequenos que não demostraram que este algoritmo tem como problema a complexidade temporal quando deparado comum problema de muitos estados, sendo um algoritmo apropriada para problemas de estado finito e determinístico.

5.2. Planeamento automático com base no método Frente-Onda (Wavefront)

Introdução Teórica

O algoritmo Wavefront é um algoritmo de propagação de onda que é usado para encontrar o caminho mais curto entre um ponto inicial e um ponto final em um mundo bidimensional representado por uma matriz.

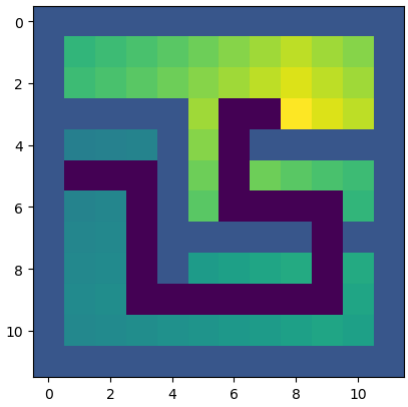
Este algoritmo funciona com um tipo de procura em largura atribuindo a todos os estados valores de acordo com a distância até ao objetivo e de segunda, expandido esses valores para os estados adjacentes, o que demostra que o algoritmo se baseia na expansão de nos com o tipo de procura e, largura.

O processo para este algoritmo começar com a atribuição de um valor inicial a todos os estados do mundo, exceto ao estado do objetivo que é 0. Apos isto é realizada a propagação dos valores através do mundo, expandindo-os dos estados objetivos para os estados adjacentes, onde os estados adjacentes ao objetivo são atualizados com o valor de decremento de numa determinada taxa.

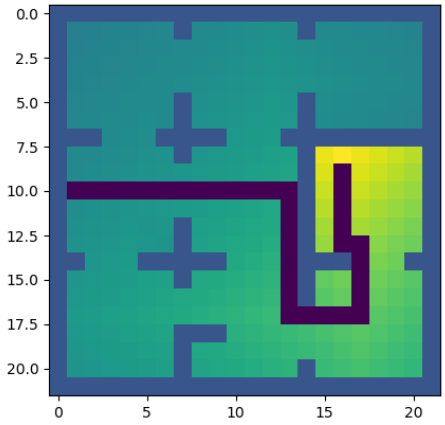
Apos atualizar os valores adjacente continua a expandir até que chega um ponto que o algoritmo usa estes valores para encontrar o caminho mais curtos entre o estado inicial e o estado objetivo, através da escolha do estado com menor valor para servir de próximo estado até chegar ao objetivo.

Implementação e resultados

Para avaliar o desempenho do algoritmo warefront, utilizara-me os ficheiros fornecidos amb.txt e amb2.txt para verificar que o algoritmo nos indica-se qual era o caminho menor. Após isto, obteve-se com recurso ao matplotlib as seguintes ilustrações do "world1" e "world2" que estão representadas nas seguintes figuras 20 21. É de nota que as cores da imagem indicam que os estados com maiores valores são os estados mais próximos do alvo, com cor mais clara, fazendo-se notar um gradiente desde os estados mais longes e os estados mais pertos do alvo.



Equação 21 - Percurso resultante do algoritmo Warefront no ambiente 1



Equação 20 - Percurso resultante do algoritmo Warefront no ambiente 2

Como podemos ver nas figuras 20 e 21, estre processo resulta numa representação do mundo em que cada estado tem um valor atribuído, pelo que se trata de uma vantagem quando se trata de pequenos mundos garantindo sempre que haja um caminho possível (caso exista) para que o agente chega ao alvo.

No entanto, como o algoritmo recorre á memoria para representar o mundo, no caso de o mundo ser grande e tenha muitos obstáculos pode custar muito computacionalmente o que é uma desvantagem.

5.3. Planeamento automático Wavefront

Os algoritmos de procura A* e Wavefront são ambos algoritmos de procura em espaços de estados que são utilizados de forma a encontrarem o cominho mais curtos entre dois pontos num ambiente com obstáculos.

Uma das principais diferenças é a forma como cada algoritmo calcula o seu curto em cada caminho, sendo que a procura A* utiliza a funcao heurística (qua é uma estimativa do custo para chegar ao destino) e a procura wavefront utilizar uma propagação de ondas a partir do destino para determinar o custo de cada caminho.

Em resumo, ambos os algoritmos A* e Wavefront são uteis para encontrar caminhos mais curtos e ambientes com obstáculos, uma vez que a procura A* é uma procura mais precisa, mas que pode levar mais tempo devido a necessidade de utilizar a funcao heurística. E por outro lado, a procura Wavefront é geralmente mais rápida que a A*, mas menos precisa devido á falta de uma função heurística.

6. Raciocínio Automático para otimização

Neste último problema, procurou-se explorar os métodos de raciocínio automático para otimização que são uma categoria específica de métodos de raciocínio automático que se concentram em encontrar a melhor solução possível dentro de certas restrições, ou seja, na categoria da otimização com métodos concentram-se em encontrar a melhor solução possível de acordo com um critério de avaliação dentro de um conjunto de opções possíveis.

Dependendo do algoritmo, o critério de avaliação poderá ser um dos seguintes:

1. Medida de ganho (performance);
2. Medida de perda (loss);
3. Medida de adequação (fitness);

Dentro dos vários métodos computacionais de otimização, os de meta-heurísticas que serão utilizados neste ponto do projeto são as formas de procura em estados de estados, como:

1. Procura local sôfrega (Hill-climbing);
2. Têmpera simulada (Simulated anealing);

Para resolver problemas de otimização é necessário configurar os parâmetros da melhor forma a maximizar ou minimizar uma funcao valor. Esse processo pode ser representado por meio de estados, transições, valor e solução, sendo que:

1. **Estado**: Representa uma configuração (Espaço de estados);
2. **Transição**: Representa uma transformação de estado (mudança de configuração, podendo ser um operador ou um vetor);
3. **Valor**: função de valor avalia cada configuração;
4. **Solução:** é a configuração final que obtém o melhor resultado de acordo com o critério de avaliação (estado final);

6.1. Introdução dos problemas

Introdução do problema do caixeiro viajante

O problema do caixeiro viajante é um problema clássico de otimização que consiste em encontrar a menor rota que percorre todas as cidades de uma lista, passando por cada uma apenas uma vez, retornando finalmente à cidade de origem.

Este problema é representado por um espaço bidimensional, no qual são indicadas as coordenadas x e y de cada cidade e onde o caixeiro viajante precisa percorrer uma ordem de visitas de forma a encontrar a menor distância possível entre todos os caminhos. Contudo, a posição das cidades(valores das coordenadas) não deve ser alterada durante a resolução do problema, pois é apenas necessário alterar a ordem e não as coordenadas.

Aqui o começa-se por identificar que existe uma lista das coordenadas das cidades onde cada possibilidade de caminho é um estado ou uma configuração deste problema. Sendo que o critério de avaliação neste problema é a distância total percorria, ou seja, para avaliar o valor, sendo que quanto maior pior, pois a solução pretende-se encontrar a menor distância possível.

Introdução do problema N Rainhas

O problema das N-Rainhas é um problema clássico de otimização que consiste em colocar N rainhas num tabuleiro de xadrez de NxN quadrados, de forma a que nenhuma rainha possa atacar as outras, por outras palavras é preciso configurar o tabuleiro de maneira a que nenhuma das N rainha estrega na mesma coluna, linha ou diagonal. Cada rainha pode se mover em qualquer direção (horizontal, vertical e diagonal) e, portanto, tem a capacidade de atacar qualquer outra rainha no tabuleiro.

Neste problema recorre-se a soluções de otimização para resolver um problema cujo o objetivo é encontrar uma disposição (solução) valida, sem ataques entre as rainhas.

Para isso utilizou-se um critério de avaliação ao contar o número de colisões entre as rainhas, sendo que quanto maior pior tentando encontrar o estado com 0 posições.

6.2. Introdução dos métodos de raciocínio automático para otimização

Hill-Climbing estocástico

Hill-Climbing estocástico é uma técnica de procura heurística que é utilizada para resolver problemas de otimização por meio de passos iterativos, onde o algoritmo inicia com um estado inicial e de seguida, procura estados vizinho que tenham um estado melhor do que o inicial.

Para fazer isso, este algoritmo procura um vizinho aleatório a partir do estado corrente e depois assume esse valor pra a próxima iteração, onde sempre que o novo estado foi melhor do que o estado atual, o algoritmo pode ou não mudar para o novo estado com uma determinada probabilidade.

No Hill-Climing, uma vez que encontra o melhor vizinho o algoritmo termina por ter encontrado um local ótimo, no entanto o hill-Climing estocástico por ser uma variação deste, tem por detrás a ideia de incluir elementos aleatórios para tentar "escapar" de ótimos locais permitindo procurar novos estados e portanto ter melhor oportunidades de encontrar uma solução globalmente ótima.

Por outro lado, este algoritmo pode ser mais lento do que outro algoritmo de otimização devido ao fato de ele precisar de gerar muitos vizinhos.

Hill-Climbing com reinício aleatório

Semelhante ao hill-Climing estocástico, o hill-Climing estocástico com reinício aleatório é uma variação do algoritmo Hill-Climbing, que adiciona a capacidade de reiniciar o processo de procura a partir de um estado inicial aleatório quando o algoritmo fica preso nos locais otimos.

Também como o algoritmo hill-Climing este tentar aumentar as suas oportunidades de "escapar" dos ótimos locais, mas neste caso com o recurso de reiniciar o processo de procura.

Simulated Annealing

O algoritmo de simulated Annealing é um algoritmo de procura heurística que se baseia num processo de resfriamento, para encontrar a solução ótima de problemas de otimização. Este método utiliza tecnicas de adaptação dinâmica de parâmetros de procura, amostragem e restrição temporal para enconrar soluções aproximadas.

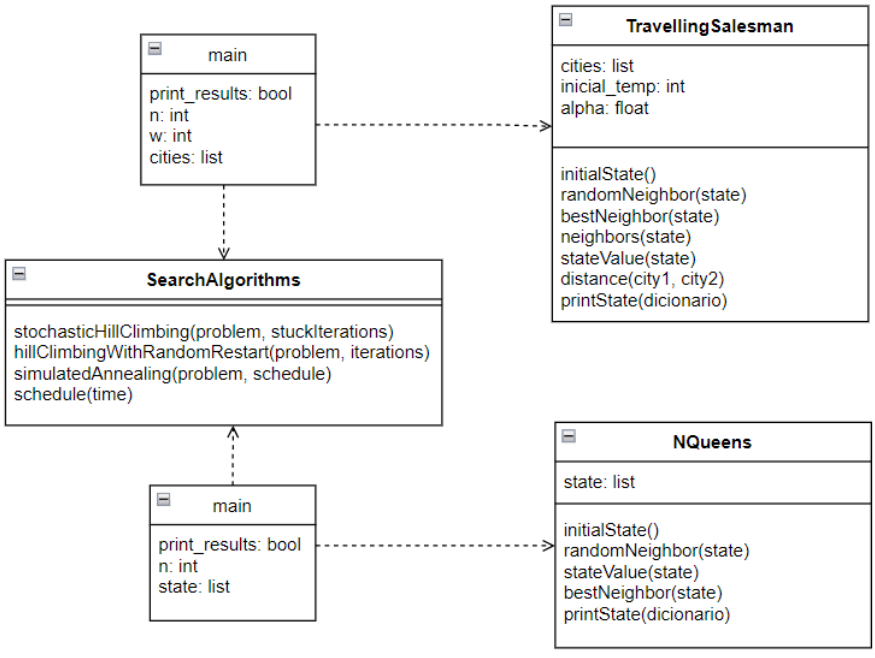
A analogia com a têmpera do metal é usada para modelar o processo de aquecimento e arrefecimento, o que permite a flexibilidade necessária para explorar diferentes soluções e evitar ficar preso em ótimos locais sub ótimos.

Uma vez que este algoritmo é capaz de lidar com problemas de otimização com grandes espaços de procura, tem a desvantagem de poder ficar lento para encontrar uma solução.

6.3. Implementação dos métodos raciocínio automático para otimização

A implementação de algoritmos de raciocínio automático para otimização nos problemas específicos como N-Rainhas e o Caixeiro viajante, tem como objetivo encontrar a solução ótima através de uma funcao de desempenho.

Ao executar a implementação do codigo tentou-se estruturar cada classe para cada problema da forma como é apresentada a seguinte figura 15:

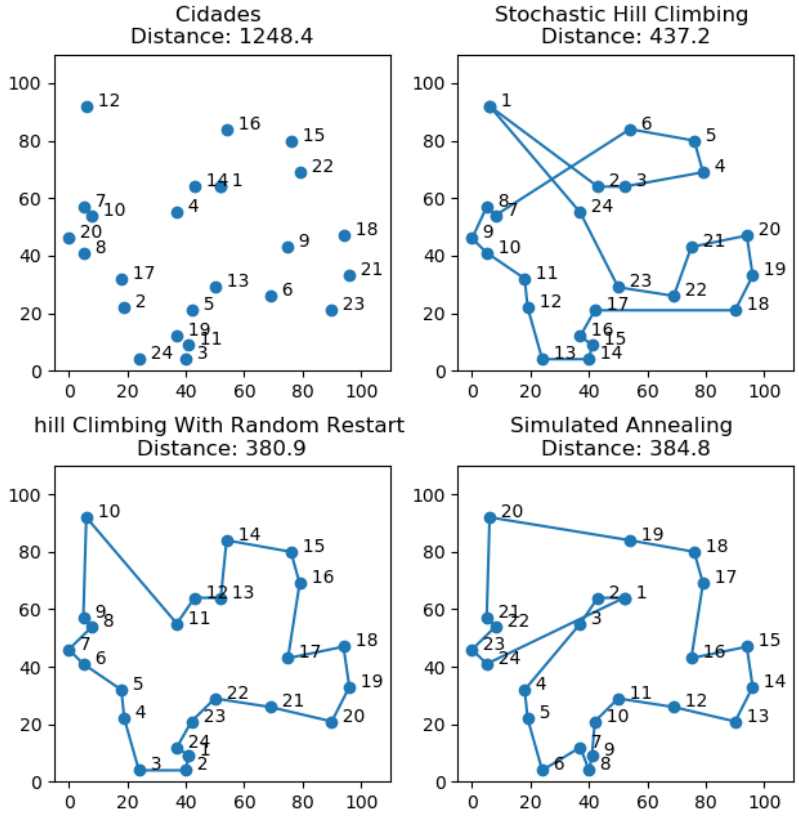


Equação 22 - Arquitetura das classes para resolver os problemas N rainhas e o Caixeiro viajante

Caixeiro viajante

A classe TravellingSalesman é a implementação do problema do caixeiro viajante que estamos a tentar resolver que recebe as cidades (coordenadas) que o caixeiro viajante deseja visitar. Foram criados métodos para o auxílio do problema como:

1. O método initialState que é utilizado para retornar uma solução inicial aleatória, onde é criado uma caminhode forma aleatória que visita as cidades.
2. O método randomNeighbor retorna uma solução vizinha modificando a solução atual de forma aleatória, trocando a ordem de duas cidades na rota.
3. O método bestNeighbor é utilizado para encontrar a melhor solução vizinha comparando a função de custo das soluções vizinhas.
4. O método stateValue é utilizado como funcao de custo, onde é calculada a distancia total do caminho calculando a soma das distancias entre todas as cidades do caminho.
5. O método distance é utilizado para calcular a distancia euclidiana entre duas cidades.
6. O método printState é utilizado para mostrar a evolução das soluções encontradas por todos os algoritmos, ou seja, apresenta com recurso ao matplotlib o caminho inicial e as soluções finais de cada algoritmo.



Equação 23 – Exemplo de soluções finais de cada algoritmo com os respetivos caminhos do caixeiro viajante.

Após algumas execuções, podemos verificar pela tabela seguinte 6 que o algoritmo Hill-Climbing com reinício aleatório foi o que apresentou melhor desempenho, enquanto o algoritmo Simulated Annealing foi o que apresentou pior desempenho.

Métodos raciocínio automático para otimização				
	Distancias			
Execução	Inicial	Hill-Climbing estocástico	Hill-Climbing com reinício aleatório	Simulated Annealing
1	1089	444	438	459
2	1328	483	424	470
3	1318	539	401	588
4	1373	470	367	491
5	1223	491	410	505
Média:	**1266,2**	**485,4**	**408**	**502,6**

Tabela 6 - Tabela com o desempenho dos algoritmos após algumas execuções no problema do caixeiro-viajante.

As principais razoes que levam a concluir estes resultados resumem-se as diferenças entre os algoritmos, ente elas podem ter sido 1) a estrutura do problema, uma vez que cada problema pode favorecer alguns desempenhos melhores pela forma como o algoritmo é estruturado.

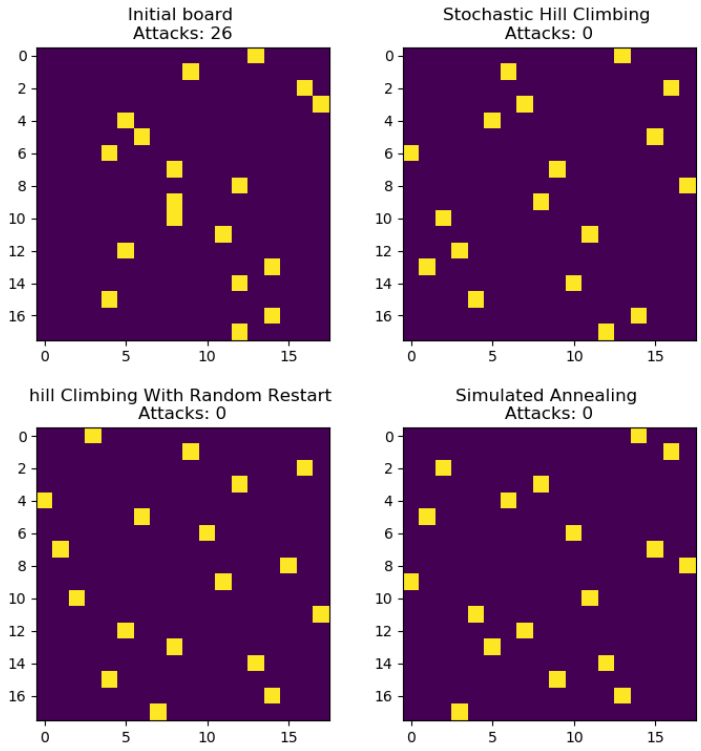
1.1) Uma possível má parametrização da função de temperatura, ou seja, o Simulated Annealing pode ter saído pior por ter sido dado uma temperatura inadequada.

1.2) O algoritmo Hill-Climbing com reinício aleatório pode ter sido melhor devido ao seu algoritmo de procurar conseguir explorar diferentes áreas de procura, que aumenta a suas probabilidades de encontrar uma solução ótima.

N-Rainhas

A classe N Queens é a implementação do problema N rainhas que estamos a tentar resolver que receber inicialmente um estado das rainhas dispostas num tabuleiro de xadrez, ou seja, as suas posições. Para auxílio deste problema foram criados os seguintes métodos:

1. O Método initialState devolve esse estado inicial.
2. O método randomNeighbor retorna um vizinho aleatório do estado atual, movendo uma das rainhas para uma posição aleatória no tabuleiro.
3. O método stateValue devolve o valor do estado atual, que é o número de rainhas que se atacam.
4. O método bestNeighbor devolve o melhor vizinho do estado atual, ou seja, o vizinho que tem o menor número de rainhas que se atacam.
5. Por fim, o método printState que mostra o estado atual no formato de um tabuleiro de xadrez dos estados inicial, e da solução final dos algoritmos.



Equação 24 - Exemplo de solucoes finais de cada algoritmo com a respetiva disposicao das rainhas.

Após algumas execuções, podemos verificar pela tabela seguinte 9 que todos os algoritmos de procura heurística utilizados, conseguiram encontrar a as soluções que são classificadas com globalmente ótimas para o problema das N-rainhas.

N rainhas

Métodos raciocínio automático para otimização

		Ataques			
Execução	**N Rainhas**	**Inicial**	**Hill-Climbing estocástico**	**Climbing com reinício aleat**	**Simulated Annealing**
1	8	7	0	0	0
2	12	12	0	0	0
3	18	18	0	0	0
4	17	17	0	0	0
5	26	26	0	0	0
Média:	**16,2**	**16**	**0**	**0**	**0**

Tabela 7 - Tabela com o desempenho dos algoritmos após algumas execuções no problema das N Rainhas

As principais razoes que levam a concluir que todos os algoritmos conseguiram chegar a resolução do problema das N-rainhas podem ter sito que a resolução do problema fosse simples.

Uma outra maneira de avaliar a eficiência os algoritmos sendo que todos conseguiram chegar a melhor solução será através dos cálculos do tempo e de velocidade dos mesmos.

7. Tema de escolha livre - ChatGPT

Introdução

A inteligência artificial tem sido um campo em rápida evolução desde sua criação, e um dos seus subcampos mais fascinantes é o desenvolvimento de modelos de linguagem, onde esses modelos têm como objetivo imitar e melhorar a capacidade humana de compreender e gerar linguagem natural. A evolução dos modelos de linguagem tem sido marcada por muitos avanços notáveis, desde os primeiros modelos baseados em regras, até os modelos baseados em aprendizado de máquina, como o chatGPT.

O chatGPT é um modelo de linguagem desenvolvido pela OpenAI, que foi treinado em milhões de documentos de texto para gerar linguagem natural. Ele é baseado em uma arquitetura de transformers, que é uma forma de processamento paralelo que permite que o modelo processe uma grande quantidade de informação de uma só vez. Isso permite que o chatGPT seja capaz de gerar texto com alta qualidade e coerência, o que o torna uma ferramenta poderosa para uma variedade de tarefas, incluindo geração de texto, tradução automática e resposta a perguntas.

Arquitetura do ChatGPT

A arquitetura do chatGPT é baseada no modelo de transformers, um tipo de rede neural onde a principal vantagem dos transformers é a sua capacidade de lidar com séries temporais de dados de comprimento variável, o que é essencial para o processamento de linguagem natural.

A arquitetura do chatGPT consiste numa camada de entrada, seguida por um conjunto de camadas de encoder e decoder. As camadas de encoder são responsáveis por codificar a entrada de texto em um espaço de representação de alta dimensionalidade, enquanto as camadas de decoder são responsáveis por decodificar essa representação em texto de saída.

A camada de entrada é composta por um embedding de palavras, que é responsável por mapear cada palavra de entrada em um vetor de embedding. Esses vetores são então passados para as camadas de encoder, que são compostas por várias camadas de Multi-Head Attention e Feed-Forward.

As camadas de Multi-Head Attention permitem que o modelo preste atenção a diferentes partes do input ao mesmo tempo, enquanto as camadas Feed-Forward permitem que o modelo aprenda relações não-lineares entre as palavras de entrada.

As camadas de decoder, por sua vez, são compostas por várias camadas de Multi-Head Attention e Feed-Forward, além de uma camada de saída que é responsável por gerar a saída de texto.

Em resumo, a arquitetura do chatGPT é baseada no modelo de transformers e é composta por camadas de encoder e decoder que permitem ao modelo compreender e gerar texto natural. É um dos modelos de linguagem maiores já treinados e é capaz de lidar com uma variedade de idiomas e estilos de escrita.

Treino do ChatGPT

O processo de treino do chatGPT é um dos aspectos mais importantes para garantir que o modelo produza resultados precisos e relevantes uma vez que o modelo é treinado usando grandes quantidades de dados de texto, geralmente chamados de corpus de dados. Esse corpus de dados é usado para ensinar o modelo a reconhecer padrões e relações entre palavras e frases, permitindo que ele gere respostas coerentes e precisas.

Uma das técnicas de otimização mais comuns utilizadas no treino do chatGPT é o algoritmo de otimização Adam. Esse algoritmo é especialmente eficaz para modelos de linguagem, pois é capaz de ajustar os parâmetros do modelo de forma eficiente, minimizando a perda de precisão.

Outra técnica importante utilizada no treino do chatGPT é o uso de múltiplos GPUs para acelerar o processo de treino o que permite que o modelo seja treinado em paralelo, o que é especialmente útil quando se trabalha com grandes conjuntos de dados.

O processo de treino do chatGPT é essencial para garantir que o modelo produza resultados precisos e relevantes envolvendo o uso de grandes quantidades de dados de texto, técnicas de otimização sofisticadas e o uso de múltiplos GPUs para acelerar o processo de treino.

Avaliação do Desempenho do chatGPT

O desempenho do chatGPT é avaliado em diversas tarefas específicas, como ao gerar de texto, tradução automática e resposta a perguntas como se fosse uma conversa. Uma das principais vantagens do chatGPT é sua capacidade de gerar texto de forma fluente e coerente, o que o torna ideal para aplicações como escrita assistida, geração de resumos e criação de diálogos. Além disso, o chatGPT também tem demonstrado desempenho significativo em tarefas de tradução automática, como a tradução de idiomas menos comuns ou tradução de textos técnicos.

Aplicações do chatGPT

Assistentes virtuais são uma das aplicações mais comuns do chatGPT visto que o modelo é treinado para compreender o contexto de uma conversa e responder de forma adequada, permitindo que os utilizadores interajam com dispositivos de forma natural. Além disso, o chatGPT também é utilizado em sistemas de reconhecimento de voz para melhorar a precisão e a compreensão do usuário.

Outra área em que o chatGPT é amplamente utilizado é na geração de conteúdo, pois é capaz de criar textos, descrições de imagens e até mesmo código, tornando-o útil para tarefas como a criação de resumos de artigos e geração de histórias.

Por fim, o chatGPT também é utilizado em análise de sentimentos, onde é treinado para classificar textos como positivos, negativos ou neutros. Isso é útil em aplicações como o monitoramento de redes sociais e a análise de opiniões de clientes.

Em resumo, o chatGPT é uma ferramenta poderosa e versátil que é utilizada em uma variedade de aplicações, desde assistentes virtuais até análise de sentimentos, com o objetivo de melhorar a eficiência e a precisão das tarefas automatizadas.

Limitações e desafios do chatGPT

A arquitetura do chatGPT é baseada em transformers, um tipo de rede neural que foi projetada para lidar com dados sequenciais, como texto, onde a principal característica dos transformers é o uso de atenção, que permite que o modelo considere todas as palavras de uma frase ao mesmo tempo, em vez de processá-las uma a uma. Isso permite que o chatGPT compreenda o contexto de uma frase e gere respostas mais precisas.

O chatGPT é um modelo de linguagem de grande porte, com milhões de parâmetros. Isso permite que ele processa uma grande quantidade de dados e aprenda a gerar texto com fluência e naturalidade. No entanto, essa grande capacidade também requer muitos recursos computacionais para treinamento e uso, e pode resultar em overfitting se não for devidamente gerenciada.

O treinamento do chatGPT é feito usando um processo chamado reinamento, onde o modelo é alimentado com grandes volumes de texto e ajustado para prever as palavras seguintes em uma frase. Isso é feito usando técnicas de otimização, como o algoritmo Adam, que ajusta os parâmetros do modelo para minimizar a perda. O corpus de dados utilizado para o treinamento do chatGPT é composto por milhões de documentos de texto, incluindo livros, artigos e conversas.

O desempenho do chatGPT é impressionante em tarefas de geração de texto, tradução automática e resposta a perguntas. Ele é capaz de gerar texto fluente e natural, traduzir entre idiomas com precisão e responder perguntas de forma precisa e rápida. No entanto, é importante notar que o desempenho do chatGPT pode variar dependendo do corpus de dados utilizado e da tarefa específica.

O chatGPT tem muitas aplicações em diversas áreas, incluindo assistentes virtuais, geradores de conteúdo e análise de sentimentos. Ele é usado para criar chatbots que conversam com os usuários de forma natural, gerar texto automaticamente, traduzir entre idiomas e analisar o sentimento dos usuários em comentários de mídia social.

Embora o chatGPT seja um modelo de linguagem avançado, ele ainda tem limitações e desafios a serem superados. Uma das principais limitações é a falta de compreensão do mundo real, o que pode resultar em respostas imprecisas ou irrelevantes.

Comparação com outros modelos (BERT e GPT-2)

A comparação entre o chatGPT e outros modelos de linguagem é uma área de interesse crescente na pesquisa em inteligência artificial. Um dos modelos mais comparados com o chatGPT é o BERT (Bidirectional Encoder Representations from Transformers). Enquanto o chatGPT é um modelo autoregressivo, o BERT é um modelo pré-treinado de encoder-decoder. Isso significa que o BERT é treinado para prever uma palavra a partir do contexto, enquanto o chatGPT é treinado para prever uma palavra a partir do seu histórico de palavras. Isso leva a diferenças de desempenho nas tarefas de compreensão de linguagem, com o BERT tendo um desempenho superior em tarefas como a resposta à pergunta e a classificação de entidade.

Outro modelo de linguagem comparado com o chatGPT é o GPT-2 (Generative Pre-trained Transformer 2). O GPT-2 é semelhante ao chatGPT em sua arquitetura de transformer autoregressivo, mas é treinado em um corpus de dados significativamente maior e possui um número maior de parâmetros. Isso leva a um desempenho superior em tarefas de geração de texto, mas pode levar a problemas de bias e não-veracidade na geração de texto.

Em resumo, o chatGPT é um modelo de linguagem avançado que apresenta desempenho excelente em tarefas de geração de texto, mas tem desempenho inferior em tarefas de compreensão de linguagem em comparação com modelos como o BERT. Ele também tem desempenho superior ao GPT-2 quando se trata de problemas de bias e não-veracidade.

Futuro do chatGPT

O futuro do chatGPT é muito promissor, com tendências a serem seguidas no desenvolvimento do modelo. Uma das tendências é o aumento do tamanho do modelo, o que permitirá que ele processe mais informações e ofereça respostas mais precisas e detalhadas. Além disso, a utilização de dados não estruturados, como imagens e vídeos, pode ampliar as aplicações do chatGPT.

Outra tendência é o uso de técnicas de aprendizado profundo mais avançadas, como o uso de redes neuronais recorrentes, permitindo que o modelo compreenda contextos mais complexos e responda com mais precisão e naturalidade.

Ao mesmo tempo, a comunidade de ciência de dados e inteligência artificial está trabalhando para desenvolver novas técnicas para lidar com problemas éticos e sociais levantados pelo uso de modelos de linguagem avançados, incluindo questões de privacidade e desinformação.

Enfim, a evolução do chatGPT é uma área fascinante de estudo e suas implicações no futuro são vastas e importantes.

Conclusão

Na conclusão do meu trabalho, revisarei os principais tópicos discutidos ao longo do texto, incluindo a história de desenvolvimento de modelos de linguagem e a evolução do chatGPT. Também explicarei a arquitetura do modelo, incluindo o uso de transformers e o tamanho do modelo, bem como o processo de treinamento do modelo, incluindo o corpus de dados utilizado e as técnicas de otimização utilizadas. Além disso, avaliarei o desempenho do modelo em tarefas específicas, como geração de texto, tradução automática e resposta a perguntas, e discutirei exemplos de como o chatGPT é utilizado em diversas áreas. Também discutirei as limitações atuais do modelo e os desafios futuros para o seu desenvolvimento, bem como compararei o chatGPT com outros modelos de linguagem, incluindo BERT e GPT-2. Além disso, discutirei os impactos éticos e sociais do uso do chatGPT, como preocupações com a privacidade e a desinformação. Finalmente, farei previsões sobre as tendências futuras no desenvolvimento do chatGPT, incluindo aumento do tamanho do modelo e uso de dados não estruturados. Após revisar esses tópicos, darei minhas perspectivas futuras para o chatGPT e concluirei o trabalho.

8. Conclusão

Em conclusão, este trabalho teve como objetivo estudar e concretizar modelos e arquiteturas de inteligência artificial e sistemas cognitivos.

Na primeira parte, foi realizado um protótipo baseado numa plataforma de desenvolvimento de redes neuronais artificiais para resolver problemas de aprendizagem da função lógica XOR, assim como a aprendizagem de padrões de imagem.

Na segunda parte, foi desenvolvida uma biblioteca de aprendizagem por reforço, incluindo os métodos Q-Learning, Q-Learning com memória episódica e Dyna-Q, que foi aplicada num agente capaz de navegar num espaço discreto com obstáculos e um alvo.

Na terceira parte, foi desenvolvida uma biblioteca de raciocínio automático para planeamento, incluindo os métodos A* ponderada e Frente-Onda, assim como uma biblioteca de métodos de raciocínio automático para optimização, incluindo Simulated Annealing e Hill-Climbing estocástico, que foram aplicadas na resolução de problemas como o Caixeiro viajante e N-Rainhas.

Em geral, foi possível concluir que os modelos e arquiteturas de inteligência artificial e sistemas cognitivos desenvolvidos foram capazes de resolver os problemas propostos com sucesso e poderiam ser aplicados em outros contextos similares.

9. Referencias

https://www.geeksforgeeks.org/travelling-salesman-problem-using-dynamic-programming/

https://dev.to/jbahire/demystifying-the-xor-problem-1blk

https://www.isep.ipp.pt/files/Redes%20Neuronais%20-%20Conceitos%20-%20JMS%20(vers%C3%A3o_final).pdf

https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

https://towardsdatascience.com/a-visual-introduction-to-neural-networks-68586b0b733b

https://medium.com/ensina-ai/redes-neurais-roots-1-introdu%C3%A7%C3%A3o-ffdd6f8b9f01

https://www.britannica.com/technology/artificial-intelligence/Methods-and-goals-in-AI

introd-ia.pdf

sist-cog.pdf

https://www.britannica.com/technology/artificial-intelligence