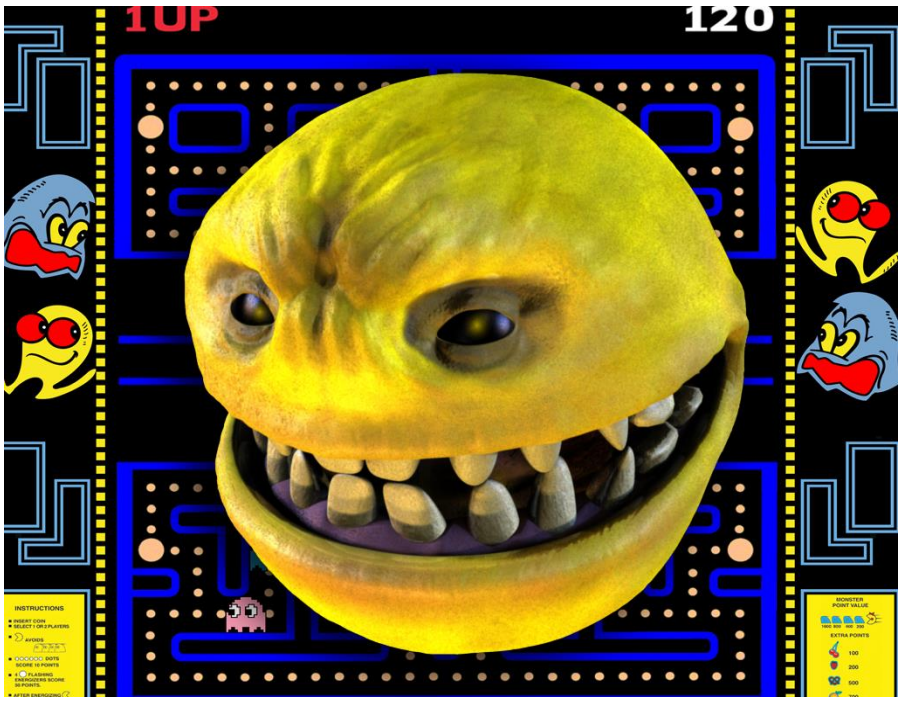
INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL



2011/2012

Pacman – Back to School

Gonçalo Silva Pereira - 2009111643

Igor Nelson Garrido da Cruz - 2009111924

Maria Rita Nogueira - 2009115231

Índice

Divisão de Trabalho

Gonçalo Silva Pereira - gsp@student.dei.uc.pt

- **Tempo de estudo**: 6 horas
- **Tempo de implementação**: 20 horas
- **Contributo para o trabalho**:
  - Recolha de casos de treino;
  - Implementação das funções que guardam os pesos da rede neuronal num ficheiro;
  -

Igor Nelson Garrido da Cruz - igorcruz@student.dei.uc.pt

- **Tempo de estudo**: 4 horas
- **Tempo de implementação**: 30 horas
- **Contributo para o trabalho**:
  - Implementação e adaptação do código relativo à rede neuronal;
  - Avaliação experimental da performance da rede;
  -

Maria Rita Nogueira - mariarita@student.dei.uc.pt

- **Tempo de estudo**: 7 horas
- **Tempo de implementação**: 18 horas
- **Contributo para o trabalho**:
  - Participou na recolha de casos de treino;
  - Análise das funções relativas a rede neuronal

Introdução

Com a realização deste trabalho, esperamos compreender o funcionamento das redes neuronais como um método de resolução de problemas possivelmente complexos na área da inteligência artificial.

Em termos genéricos, pretende-se que o nosso trabalho em grupo e capacidade experimental, nos permita alcançar os seguintes objetivos:

1. Compreender o funcionamento de uma rede neuronal feedforward;
2. Compreender o funcionamento do algoritmo de retro-propagação;
3. Aprender a desenhar uma rede neuronal;
4. Apreender a treinar uma rede neuronal;
5. Compreender as forças, fraquezas e limitações desta abordagem.

Em termos práticos, este relatório pretende evidenciar os testes que nós realizámos bem como as conclusões a que chegamos em termos de parâmetros da rede neuronal, como por exemplo taxa de aprendizagem, número de casos de treino, número de neurónios na camada escondida. Serão, assim apresentadas comparações das diferentes parametrizações testadas, bem como a nossa análise.

Rede Neuronal - Implementação

Em termos de implementação da Rede Neuronal, optámos por se basear na implementação BPNN (Back-Propagation Neural Networks) de Neil Schemenauer adaptando-a às estruturas utilizadas para o ficheiro learningAgents.py.

Funcionamento do BPNN:

A classe NeuralNetworkAgent é responsável por manter a rede neuronal e fazer todas as operações necessárias à implementação do trabalho prático.

A matriz pesosInput mapeia os neurónios de input nos neurónios da camada escondida atribuindo-lhe os pesos pretendidos, de forma semelhante a matriz pesosOutput mapeia os neurónios da camada escondida nos neurónios de output definindo os seus pesos.

A função de treino trainNeuralNetwork() recebe um caso de treino e durante 1000 iterações ou até atingir uma variação de erro na ordem das 0.05 unidades de distância quadrática prossegue ao treino da rede neuronal servindo-se para isso do algoritmo de retro-propagação.

O algoritmo de retro-propagação funciona de forma bastante simples. Para atualizar os pesos da camada de saída, calcula a saída da rede neuronal para um dado input, compara com a saída esperada e guarda a diferença. Para cada neurónio serve-se da derivada da sigmoide e multiplica por esta diferença, encontrando assim o valor de variação delta.

De seguida, calcula o erro para cada neurónio na camada escondida. Este erro é dado pela multiplicação dos valores de variação da camada de saída, calculados anteriormente, com os pesos da camada de saída. Posteriormente, este erro é multiplicado pelo valor no neurónio da camada escondida respondente, encontrando-se assim a variação na camada escondida.

Após ter as variações, o algoritmo prossegue para as atualizações dos pesos das ligações entre neurónios. Para isso percorre as ligações entre os neurónios da camada atual e da camada seguinte e encontra um valor "*change*" que é a multiplicação da variação da camada seguinte com os outputs da camada atual. Este valor "*change*" é de seguida multiplicado pela taxa de aprendizagem e incrementado aos pesos.

Existe ainda a função update que dados determinados inputs, calcula o respetivo output da rede neuronal.

Alterações Efetuadas:

Quanto a alterações funcionais à implementação explicitada acima, desenvolvemos algumas funções que nos permitem guardar os pesos da rede neuronal num ficheiro chamado "MatrizPesos.txt", após o seu treino.

Quando o ficheiro não existir na diretoria do projeto, o agente aprendiz procede ao treino da rede neuronal com os casos de treino que encontrar na diretoria. No caso de existir, o agente carrega os pesos que encontra no ficheiro para a matriz de pesos que se encontra em memória e utiliza esses pesos para jogar.

Para além disso limitámos o treino da rede quando atingimos variações no valor do erro na ordem dos 0.05. Erro esse que é calculado através do somatório 0.5*(valor _esperado – valor_de_saída) ^ 2.

Criámos ainda a função translateActionList() que percorre o vetor de saída da rede neuronal procura o neurónio com valor de saída máximo e converte num vetor de 0-1, sendo que o 1 se encontra na posição máxima e os zeros nas restantes. Posteriormente pegamos neste vetor e com a função getDirection() convertemos numa direção válida para retornar ao motor de jogo.

Também o agente reativo desenvolvido no trabalho 1 foi alterado de forma a guardar o estado dos sensores à volta do Pacman e ações que o Pacman escolhe.

Para além destas alterações, fizemos ainda alterações que nos permitiram selecionar 70% dos casos de treino para treino da rede neuronal, 20% para teste e 10% para validação, de forma a medir a eficácia da rede.

Escolha da função de Ativação e Avaliação do erro:

A função de ativação utilizada foi a Sigmoide, pois pode assumir todos os valores entre 0 e 1 e tem uma derivada bastante fácil de calcular.

Em relação ao cálculo do erro, como já enunciado acima, utilizamos o somatório das distâncias quadráticas entre o valor esperado e o valor obtido nos neurónios de saída da rede neuronal.

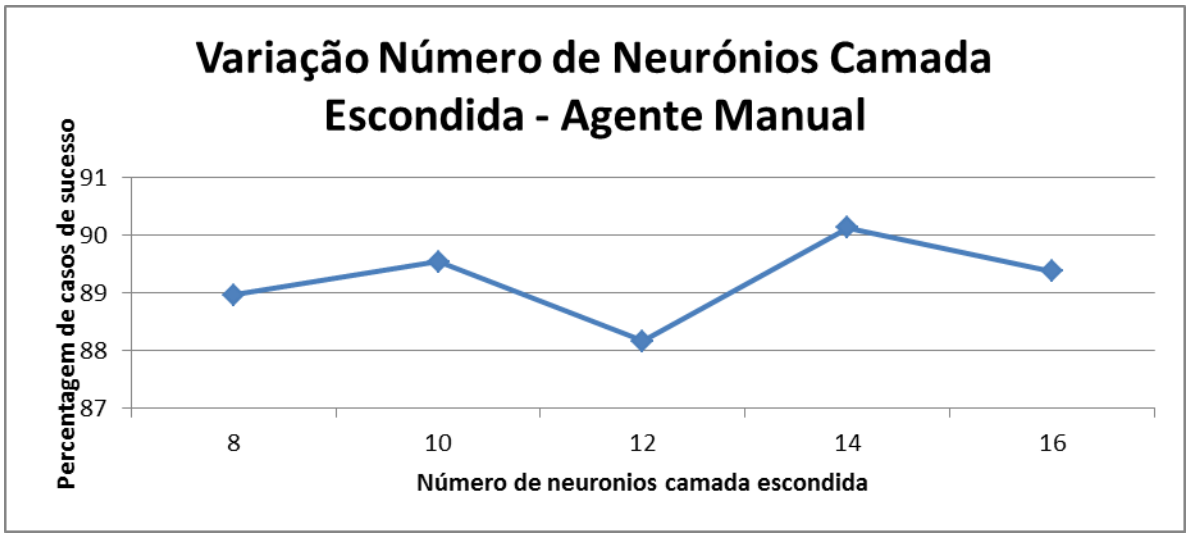
Número de casos de treino:

Relativamente ao número de casos de treino, inicialmente criámos 20 e detectamos que o Pacman não tinha aprendido o suficiente e que morria e/ou executava acções erradas em muitos casos. Logo fomos criando mais casos de treino até chegar a um número de casos em que se pudesse verificar alguma estabilidade no seu comportamento, atingindo os 100 casos de teste.

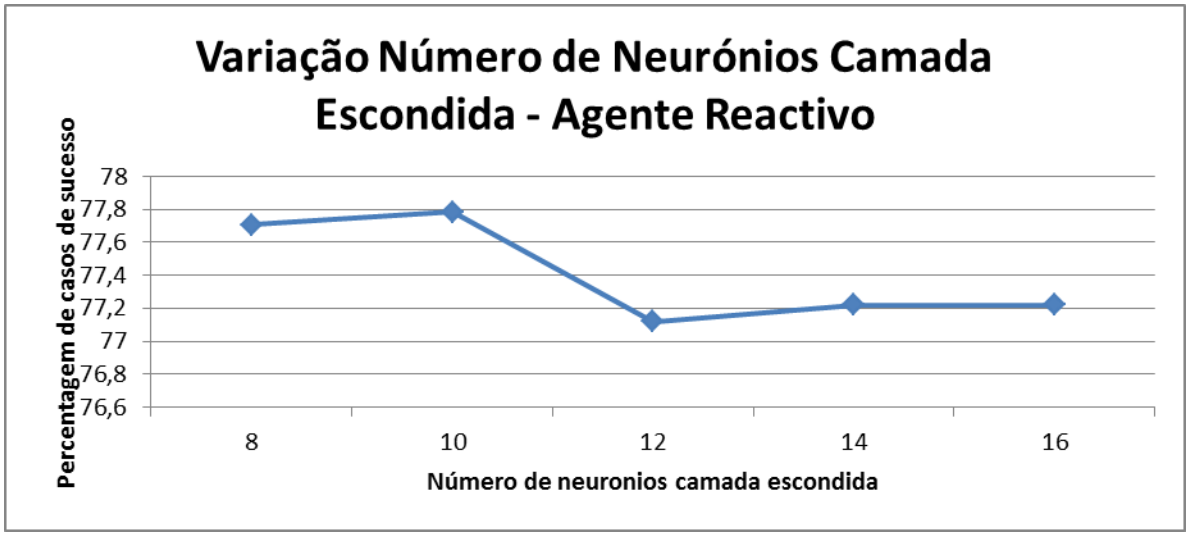
Número de neurónios na camada escondida:

O número de neurónios na camada escondida foi definido empiricamente. Tivemos o cuidado de não utilizar um número muito elevado, que poderia levar a rede a memorizar os dados de treino, facto denominado overfitting nem um número muito reduzido, que poderia não ser suficiente para a rede efetuar as acções pretendidas. Centrámo-nos então num valor entre o número de neurónios na camada de entrada e na camada de saída.

Variação Número de Neurónios da Camada Escondida - Agente Manual				
Número de neurónios	Casos Certos	Casos Errados	Total Casos	Rácio = certos/total
8	3674	456	4130	88,95883777
10	3698	432	4130	89,53995157
12	3641	489	4130	88,1598063
14	3722	408	4130	90,12106538
16	3691	439	4130	89,37046005



Variação Número de Neurónios da Camada Escondida - Agente Reactivo				
Número de neurónios	Casos Certos	Casos Errados	Total Casos	Rácio = certos/total
8	3043	873	3916	77,70684372
10	3046	870	3916	77,7834525
12	3020	896	3916	77,1195097
14	3024	892	3916	77,22165475
16	3024	892	3916	77,22165475



Para os casos de treino recolhidos com recurso ao teclado escolhemos valores de 8, 10, 12, 14 e 16 neurónios e a quantidade que se revelou mais favorável foi 14. Com esta quantidade de neurónios verificámos que tínhamos cerca de 90.1 % de casos de teste certos, em detrimento de valores na ordem dos 89% verificados com as outras quantidades.

Já para o treino da rede neuronal com auxílio do agente reativo verificámos que o número de neurónios ideal seria 10, em detrimento dos 14 verificados anteriormente, o que faz bastante sentido, visto que a nossa implementação do agente reativo não faz uso dos sensores de inimigos na diagonal, acabando por morrer muitas vezes em cruzamentos.

Rede Neuronal - Treino

Controlando o Pacman através do teclado

Inicialmente foi criado um caso de treino em cada cenário, de seguida fomos alterando alguns cenários de forma a criar situações que não teriam sido comtempladas anteriormente, reduzindo o número de comida no cenário e o número de capsulas, uma vez que em alguns cenários não é possível criar casos em que o Pacman tenha de fugir de fantasmas em situações onde não há comida. Para além disso também aumentamos o número de fantasmas em alguns cenários para mais facilmente simular casos em que o Pacman tem de fugir deles.

A maioria dos casos de treino criados concentram-se nos cenários: contestClassic, originalClassic e mediumClassic. O treino que foi efectuado teve por base o Pacman sair vitorioso dos vários cenários ao mesmo tempo que se tentava criar um comportamento o mais coerente possível tendo como base algumas prioridades, preferindo comida a fantasmas vulneráveis, comida a capsulas.

Ao todo foram criados 100 casos de treino, uma vez que foram efectuados diversos testes à medida que iam sendo feitos mais casos de treino e o comportamento do Pacman passou a ser mais coerente quando atingimos os 100 casos de treino. Dos 100 casos de treino foram usados 70 para o treino da rede, 20 para o teste da rede e os restantes 10 para a validação da rede.

Usando um agente reativo para jogar Pacman, gerando assim os padrões de treino de forma automática

Usamos o agente reactivo criado no primeiro projecto, apesar do mesmo não ter visibilidade nas diagonais, uma vez que o que se pretende é que a rede neuronal replique o mais fielmente possível o comportamento do mesmo, logo a visibilidade nas diagonais não

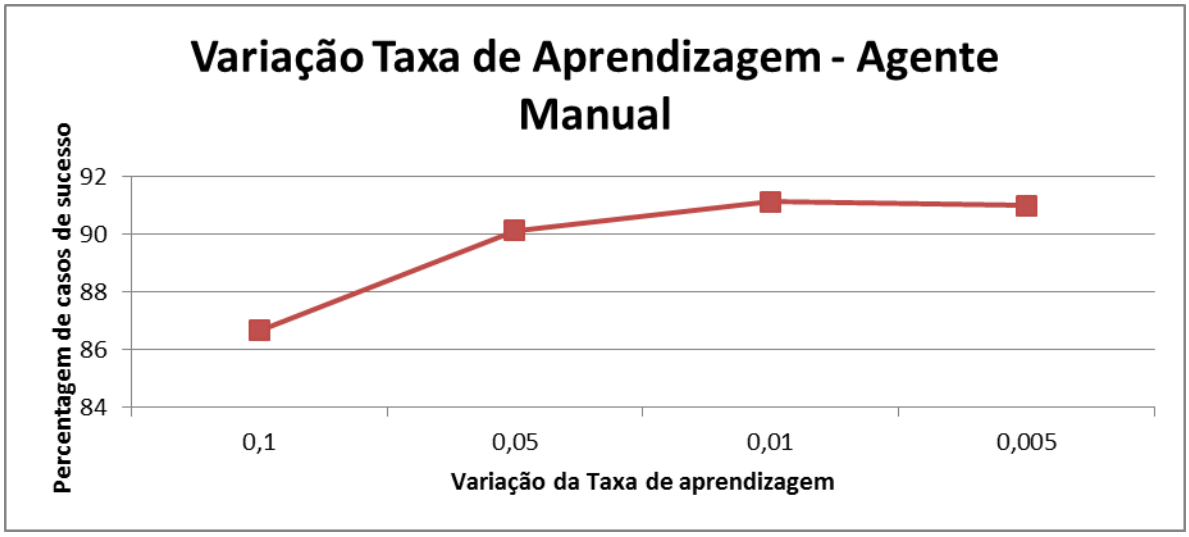
compromete de todo a conclusão da coerência ou não do comportamento replicado pela rede neuronal.

Efectuamos algumas modificações no agente, de modo a que o mesmo grave os estados dos sensores e as acções a efectuar em ficheiros de texto, neste caso 100 casos de treino, uma vez que para o agente criado através do teclado também criámos 100 casos e como queremos comparar os resultados obtidos relativamente à coerência do comportamento obtido com o comportamento esperado.

Ritmo de aprendizagem

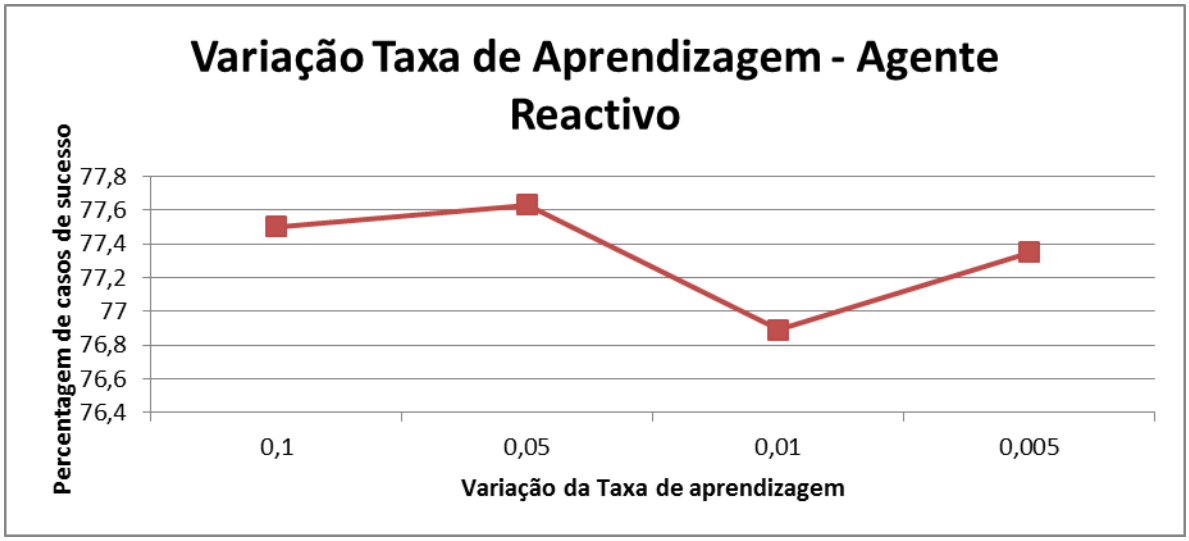
Efectuamos vários testes, variando a taxa de aprendizagem, tanto no agente manual, como com o agente reactivo, obtendo os seguintes resultados:

Variação Taxa de Aprendizagem - Agente Manual - 14 neurónios				
Taxa de Aprendizagem	Casos Certos	Casos Errados	Total Casos	Rácio = certos/total
0,1	3579	551	4130	86,65859564
0,05	3722	408	4130	90,12106538
0,01	3763	367	4130	91,11380145
0,005	3758	372	4130	90,99273608



Conforme podemos verificar no gráfico anterior, a melhor taxa de aprendizagem para o agente manual foi de 0.01, podemos ainda verificar que a taxa de aprendizagem 0.05 e 0.005 são taxas de aprendizagem intermédias e a taxa de aprendizagem 0.1 foi a pior taxa testada para aprendizagem da rede.

Variação Taxa de Aprendizagem - Agente Reactivo - 14 neurónios				
Taxa de Aprendizagem	Casos Certos	Casos Errados	Total Casos	Rácio = certos/total
0,1	3035	881	3916	77,50255363
0,05	3011	905	3916	77,63023493
0,01	3040	876	3916	76,88968335
0,005	3029	887	3916	77,34933606

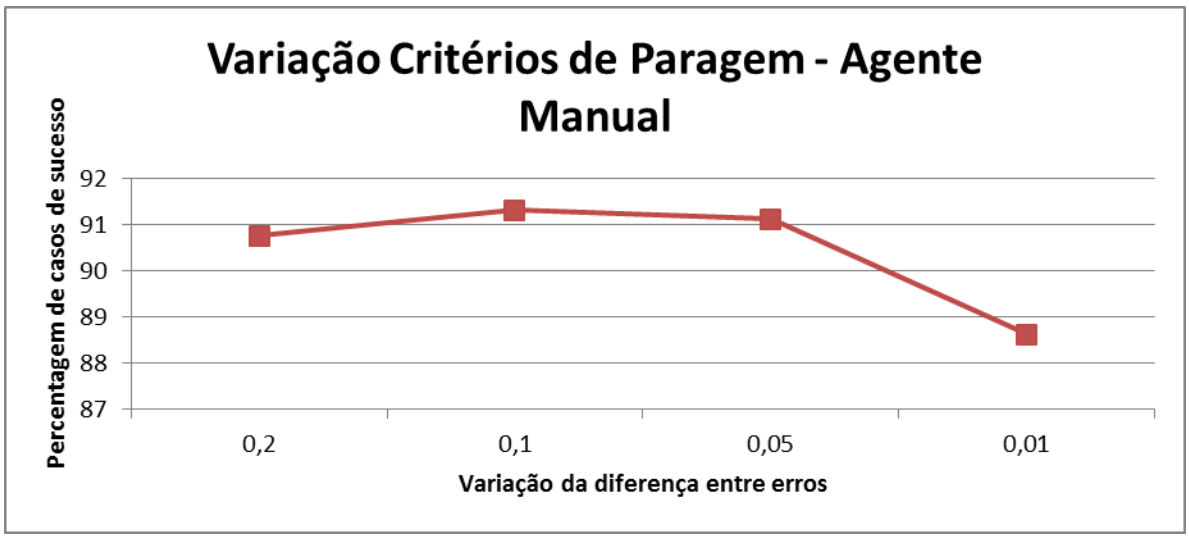


Conforme podemos verificar no gráfico anterior, a melhor taxa de aprendizagem para o agente reactivo foi de 0.05, podemos ainda verificar que a taxa de aprendizagem 0.1 e 0.005 são taxas de aprendizagem intermédias e a taxa de aprendizagem 0.01 foi a pior taxa testada para aprendizagem da rede.

Critério de paragem

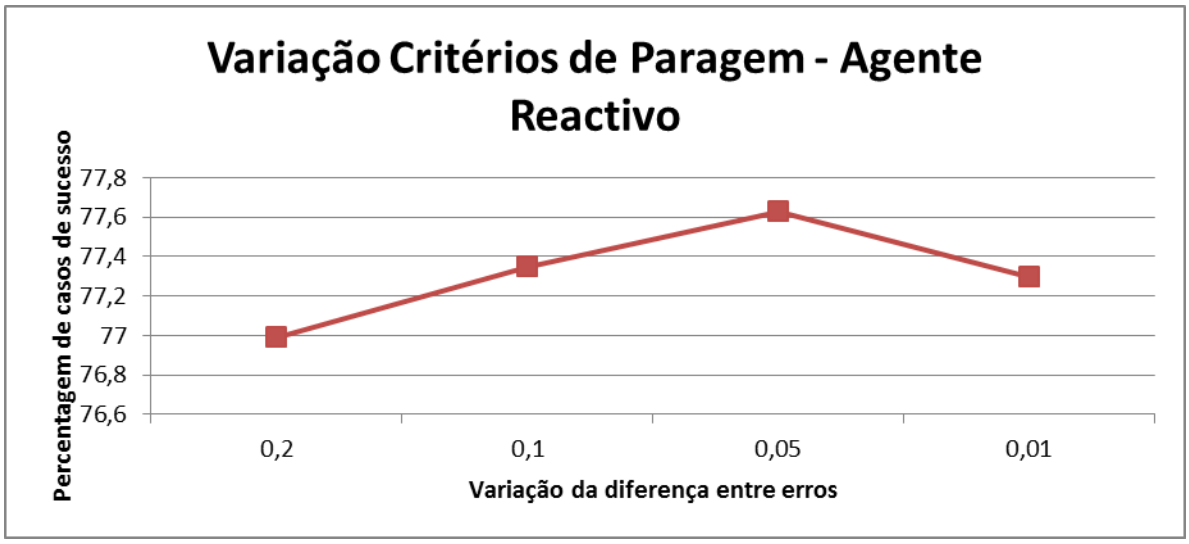
Variamos os critérios de paragem, definindo-os com os valores de 0.2, 0.1, 0.05 e 0.01. Obtendo os seguintes resultados:

Variação Critérios de Paragem - Agente Manual				
Critério de paragem	Casos Certos	Casos Errados	Total Casos	Rácio = certos/total
0,2	3771	359	4130	90,75060533
0,1	3748	382	4130	91,30750605
0,05	3763	367	4130	91,11380145
0,01	3660	470	4130	88,61985472



Podemos concluir que para o Agente Manual o melhor valor para o critério de paragem seria de 0.1, tendo uma taxa de 91.30% de casos certos.

Variação Critérios de Paragem - Agente Reactivo				
Critério de paragem	Casos Certos	Casos Errados	Total Casos	Rácio = certos/total
0,2	3015	901	3916	76,9918284
0,1	3029	887	3916	77,34933606
0,05	3040	876	3916	77,63023493
0,01	3027	889	3916	77,29826353

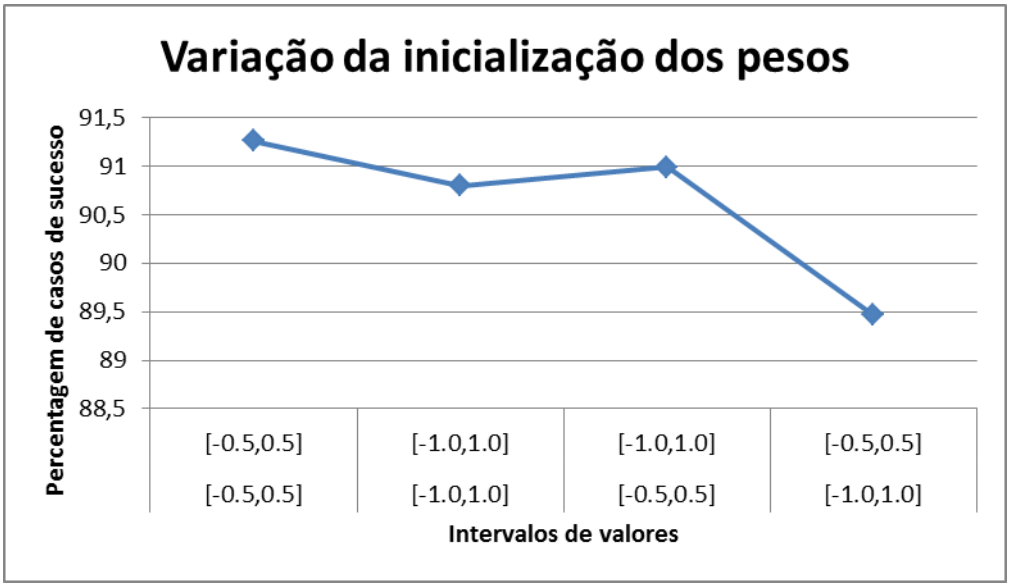


Podemos concluir que para o Agente Reactivo o melhor valor para o critério de paragem seria de 0.05, tendo uma taxa de 77.63% de casos certos.

Inicialização dos pesos

Variámos também o intervalo de valores em que são gerados os pesos entre a camada de entrada e a camada escondida e entre a camada escondida e a camada de saída, obtendo os seguintes valores:

Variação da inicialização dos pesos					
input - escondida	escondida - output	Casos Certos	Casos Errados	Total Casos	Rácio = certos/total
[-0.5,0.5]	[-0.5,0.5]	3769	361	4130	91,2590799
[-1.0,1.0]	[-1.0,1.0]	3750	380	4130	90,79903148
[-0.5,0.5]	[-1.0,1.0]	3758	372	4130	90,99273608
[-1.0,1.0]	[-0.5,0.5]	3695	435	4130	89,46731235



Podemos verificar que o melhor intervalo para a geração de valores entre a camada de input e a camada escondida e entre a camada escondida e a camada de output é o intervalo: [-0.5,0.5].

Para o agente controlado através do teclado podemos concluir que as definições de parâmetros empiricamente que melhor taxa de sucesso apresenta são:

Número de neurónios:	14
Taxa de aprendizagem:	0,01
Critério de paragem:	0,1
Inicialização de pesos:	[-0.5,0.5] [-0,5,0,5]

E para o agente reactivo são:

Número de neurónios:	10
Taxa de aprendizagem:	0,05
Critério de paragem:	0,05
Inicialização de pesos:	[-0.5,0.5] [-0,5,0,5]

Análise e Comparação

Após implementação e decisão dos melhores parâmetros a utilizar na rede neuronal, é importante analisar o comportamento desta e verificar se corresponde ao pretendido.

Segundo o teste e a validação que efectuamos, chegamos à conclusão que a rede neuronal desempenha na maioria das vezes (91% no caso do agente manual e 77% no caso do agente reactivo) o objectivo pretendido, apesar de existirem alguns casos em que o comportamento não corresponde ao esperado.

Quando posta **lado a lado com o agente reactivo**, a rede neuronal, apesar de tomar decisões correctas, muitas vezes leva a situações em que ocorrem ciclos ou em que o Pacman morre. A principal causa das mortes do Pacman está directamente relacionada com o número de sensores usados e com a falta de visibilidade nas diagonais. Ainda assim, podemos afirmar que de uma forma local a rede neuronal desempenha um comportamento similar ao do agente reactivo.

No entanto, em termos globais a rede neuronal é incapaz de simular o comportamento do agente reactivo levando à existência de ciclos em diversos cantos do mapa, uma vez que o agente reactivo em diversas situações tem um comportamento aleatório e a estrutura da rede neuronal permite trabalhar com sistemas determinísticos.

Após treinar a rede com os **casos provenientes do agente manual** facilmente verificamos que à semelhança do agente reactivo as acções – reacções são bem correspondidas localmente no entanto, isto leva a uma discrepância em termos de funcionalidade em termos globais, sendo o factor limitante, a existência de ciclos.

Assim, podemos dizer que podemos simular os padrões gerados por um humano, porém o número de neurónios de entrada não satisfaz o necessário para esta simulação, pois o Pacman não tem a mesma quantidade de informação que um jogador humano tem. Note-se

que o jogador humano consegue ver o mapa todo, ao passo que a definição de agente Pacman fornecida apenas vê algumas casas à volta.

O desempenho do agente aprendiz parece bastante satisfatório, uma vez que consegue aprender algumas coisas, como por exemplo dar preferência às casas com comida e cápsulas, desviar-se dos fantasmas quando estes estão invulneráveis e come-los quando estão vulneráveis.

As limitações encontradas poderiam ser ultrapassadas recorrendo a um número de neurónios que permitisse ao Pacman ter conhecimento do estado de todas as casas presentes no mapa, mas nesse caso a rede teria de ser adaptada ao mapa onde se iria jogar.

Conclusões

Em termos de objetivos genéricos, eles foram cumpridos.

Através da realização deste trabalho conseguimos perceber como é construída uma rede neuronal e qual a sua importância no mundo atual, percebemos também que uma estrutura tão simples pode resolver problemas bastante avançados que parecem não ter solução objetiva. Em termos mais específicos, foi-nos possível compreender melhor o funcionamento do algoritmo de retro-propagação e a importância que a escolha dos parâmetros pode ter para cada problema especificamente.

Por outro lado também conseguimos compreender as fraquezas e limitações deste tipo de abordagem. Que muitas vezes consistem em conseguir definir os parâmetros empiricamente, ou em conseguir definir a elevada quantidade de inputs. Neste caso específico, uma das limitações era a existência de ciclos quando não existiam estímulos na rede neuronal, pois a rede tem de obedecer a regras determinísticas.