REDES NEURONAIS

FCUP

Trabalho 4

Inteligência Artificial

09-06-2022

Ana Catarina da Silva Costa Gomes (up201804545)
Cláudia da Costa Maia (up201905492)
Maria Santos Sobral (up201906946)

ÍNDICE

Ana Catarina da Silva Costa Gomes (up201804545)
Cláudia da Costa Maia (up201905492)
Maria Santos Sobral (up201906946)

INTRODUÇÃO

As redes neuronais pretendem imitar o comportamento do cérebro humano, através do reconhecimento de padrões entre grandes conjuntos de dados, permitindo que certos programas resolvam problemas nas áreas de IA, machine learning e deep learning.

Estas são compostas por várias camadas, contendo uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. Cada nó corresponde a um neurónio e conecta-se a todos os nós da camada seguinte. Existe um peso e um limite associados a cada uma destas ligações.

As redes neuronais têm enumeras aplicações, nomeadamente no reconhecimento de voz e de imagens, por exemplo.

Ana Catarina da Silva Costa Gomes (up201804545)
Cláudia da Costa Maia (up201905492)
Maria Santos Sobral (up201906946)

ALGORITMOS PARA REDES NEURONAIS

1. Gradiente Descendente

Este algoritmo é usado para encontrar valores para os coeficientes (pesos) que minimizam a função de custo.
Começamos por definir um valor para cada peso e, de seguida, ajustamos iterativamente os valores para cada um dos pesos, de forma que a função de custo seja reduzida a cada iteração. A estratégia passa por executar quantos passos forem necessários até chegar ao mínimo global.

A fórmula seguinte diz-nos como encontrar a próxima posição x_{k+1}, onde x_k é a posição corrente e α é a taxa de aprendizagem (previamente definida):

$$x_{k+1} = x_k + \alpha \nabla f(x_k)$$

Apesar de ser um bom algoritmo, existem situações onde o gradiente descendente não funciona corretamente, nomeadamente quando a taxa de aprendizagem é muito alta ou muito baixa. Além disso, existe também a possibilidade de o algoritmo ficar preso num mínimo local.

2. Método de Newton

É um algoritmo de otimização de segunda ordem, uma vez que utiliza a matriz Hessiana. O objetivo é encontrar as raízes/pontos estacionários da função. A fórmula seguinte mostra como se chega à próxima posição:

$$x_{k+1} = x_k - H(x_k)^{-1} \nabla f(x_k)$$

Ao contrário do método do gradiente descendente, no método de Newton não há necessidade de se especificar uma taxa de aprendizagem, pois a própria Hessiana explicita o comportamento de segunda ordem da função em torno do ponto. Este método leva muito menos iterações para convergir do que o método do gradiente descendente. No entanto, ele ainda não é amplamente usado, porque o cálculo exato da matriz hessiana e do seu inverso são computacionalmente muito caros. Além disso, se a função analisada não for convexa o método pode não funcionar.

3. Gradiente conjugado

É um algoritmo que está entre o gradiente descendente e o método de Newton. A principal diferença é que acelera a convergência lenta que geralmente associamos ao gradiente descendente. Além disso, pode ser usado tanto para sistemas lineares como para sistemas não lineares e é um algoritmo iterativo.

Este algoritmo converge mais rapidamente que o gradiente descendente e é também mais eficaz que o método de Newton no treino da rede neural, uma vez que não requer a matriz Hessiana.

Ana Catarina da Silva Costa Gomes (up201804545)
Cláudia da Costa Maia (up201905492)
Maria Santos Sobral (up201906946)

4. Método Quasi-Newton

É uma alternativa ao método de Newton menos custosa a nível computacional. Em vez de calcular a matriz Hessiana e depois calcular a sua inversa, este método cria uma aproximação da inversa da Hessiana a cada iteração do algoritmo.

Assim, este é provavelmente o método mais adequado para lidar com grandes redes neuronais, pois economiza tempo de computação e é muito mais rápido que o método do gradiente descendente ou gradiente conjugado.

5. Método Levenberg-Marquardt

Este algoritmo está desenhado para trabalhar especificamente com a função do custo total.

Ele não precisa de calcular a matriz Hessiana, em vez disso, trabalha com o vetor gradiente e com a matriz Jacobina, o que faz com que seja muito mais rápido.

No entanto, este método apresenta alguns contras: não consegue minimizar funções como o erro quadrático médio ou o erro de entropia e, além disso, a matriz Jacobiana é enorme para grandes conjuntos de dados e redes neuronais muito grandes, exigindo muita memória.

Comparando os cinco algoritmos descritos, conclui-se que aquele que requer menos memória é o gradiente descendente, mas este é também o mais lento. O método Levenberg-Marquardt é o mais rápido, mas também o que gasta mais memória.

Tendo isto em consideração, se a nossa rede neural tem muitos parâmetros podemos usar o gradiente descendente ou o gradiente conjugado, para poupar memória.

Se tivermos muitas redes neuronais para testar com poucas amostras e poucos parâmetros, então a melhor escolha é provavelmente o algoritmo Levenberg-Marquardt.

Nas restantes situações, o método Quasi-Newton é o mais apropriado.

Ana Catarina da Silva Costa Gomes (up201804545)
Cláudia da Costa Maia (up201905492)
Maria Santos Sobral (up201906946)

DESCRIÇÃO DA IMPLEMENTAÇÃO

A linguagem de programação que escolhemos mais uma vez foi o Java, visto que como referido anteriormente é a linguagem que nos consideramos mais confortáveis a programar.

Quanto à implementação, criamos uma classe NN que representa a Neural Network, sendo que os neurónios da rede são representados pela classe Neuronio, como o nome indica.

Os neurónios podem representar um neurónio de input, output ou da camada oculta, assim sendo criamos uma String layer para os diferenciar, no entanto apenas criamos 2 construtores, um para os da camada de input e o outro para os outros dois tipos de camadas.

A classe Neuronio tem como atributos a camada a que o neurónio pertence (input, oculta ou output), o input para o caso de o neurónio ser da camada de input, assim como o output, no caso de ser da camada de output, bias e ainda a soma, utilizadas na camada oculta e na de output.

Em adição, a classe NN tem atributos como: 3 arrays de neurónios, um para cada camada, uma vez que decidimos colocar apenas uma camada oculta; 3 arrays de doubles com os deltas das 3 camadas e ainda 2 matrizes para guardar os pesos de todas as ligações entre os neurónios da camada de input com a camada oculta e desta camada com a de output. Inicialmente inicializamos as duas matrizes com erros aleatórios de modo a depois os poder corrigir recorrendo à back propagation que utiliza as funções refreshHiddenLayer e refreshOutpLayer da classe NN.

Por fim, no ficheiro IA04, começamos por fazer a leitura dos dados e tratamento dos mesmos, de modo a treinar a nossa rede neuronal com cada um dos exemplos. De seguida, inicializamos a rede e prosseguimos para a função BackPropLearning que implementa o algoritmo da back propagation de modo a corrigir os pesos de cada ligação entre neurónios. Este algoritmo consiste em submeter os exemplos à rede e comparar as respostas obtidas com o verdadeiro output, com base nessa comparação corrigimos os pesos dos neurónios de modo a obter uma melhor resposta ao exemplo de treino que utilizamos. Esta atualização de pesos é feita recorrendo primeiro à alteração dos deltas de cada neurónio, da camada de output até à de input, que consiste em derivar a função sigmoide, aplicá-la no delta do neurónio e multiplicar pelo somatório de cada peso que sai desse neurónio multiplicado por cada delta do neurónio seguinte (no caso da camada de output, como não existe ligações a sair para outra camada, logo o somatório é substituído pela diferença entre o output obtido e o esperado. De seguida a atualização dos pesos de cada ligação, mais uma vez do fim (output) para o início (input), é feita adicionando ao peso anterior a multiplicação do alfa (taxa de aprendizagem) escolhido, de valor 0.00025, pelo delta e o somatório dos pesos que entram no neurónio vezes o valor do neurónio que criou a ligação mais o bias. Concluindo, é feita também a atualização do bias de cada neurónio, somando ao que se tem, o alpha vezes o delta do neurónio em questão. Isto é corrido para cada um dos exemplos, e sendo epoch uma iteração para todos os exemplos de treino, escolhemos epoch=7, ou seja, cada exemplo é corrido 7 vezes.

RESULTADOS

```
Doing the epoch number 1...
RMSE = 1.660446747073299
Doing the epoch number 2...
RMSE = 0.8315970614214329
Doing the epoch number 3...
RMSE = 0.5978089478866413
Doing the epoch number 4...
RMSE = 0.5296685098168573
Learning time = 195373630900 ms

Test file (path)
C:\\Users\\anaca\\Downloads\\mnist_test.csv
Hit rate = 0.7898000000000001
```

Figura 1 Epoch=4, 15000 exemplos, camada oculta com 75% dos neurónios da camada de input e taxa de aprendizagem=0.0025

```
Doing the epoch number 1...
RMSE = 1.6055124410321648
Doing the epoch number 2...
RMSE = 0.8188986350811152
Doing the epoch number 3...
RMSE = 0.601633077463536
Doing the epoch number 4...
RMSE = 0.5350983763724043
Doing the epoch number 5...
RMSE = 0.5095885997837986
Doing the epoch number 6...
RMSE = 0.4991375824991034
Doing the epoch number 7...
RMSE = 0.5031893873624019
Learning time = 200824093100 ms

Test file (path)
//C:\\Users\\anaca\\Downloads\\mnist_test.csv
Hit rate = 0.7661
```

Figura 2 Epochs=7, 15000 exemplos, camada oculta com 75% dos neurónios da camada de input e alfa=0.0025

```
Doing the epoch number 1...
RMSE = 1.616627741305791
Doing the epoch number 2...
RMSE = 0.8264172701600438
Doing the epoch number 3...
RMSE = 0.5963528652717781
Doing the epoch number 4...
RMSE = 0.5142500203187552
Doing the epoch number 5...
RMSE = 0.49843472837041936
Doing the epoch number 6...
RMSE = 0.4936303840337198
Doing the epoch number 7...
RMSE = 0.4911488554062319
Doing the epoch number 8...
RMSE = 0.4754726739515431
Doing the epoch number 9...
RMSE = 0.4771998376486742
Doing the epoch number 10...
RMSE = 0.5002718762385119
Learning time = 291357890200 ms

Test file (path)
C:\\Users\\anaca\\Downloads\\mnist_test.csv
Hit rate = 0.7745
```

Figura 3 Epochs=10, 15000 exemplos, camada oculta com 75% dos neurónios da camada de input e alfa=0.0025

```
Doing the epoch number 1...
RMSE = 1.1642370385600336
Doing the epoch number 2...
RMSE = 1.0423364595582165
Doing the epoch number 3...
RMSE = 1.0361325065505091
Doing the epoch number 4...
RMSE = 1.0534441031072193
Doing the epoch number 5...
RMSE = 1.0434340863222369
Doing the epoch number 6...
RMSE = 1.0381544035103836
Doing the epoch number 7...
RMSE = 1.037722813079612
Learning time = 214264878200 ms

Test file (path)
C:\\Users\\anaca\\Downloads\\mnist_test.csv
Hit rate = 0.27659999999999996
```

Figura 4 Epochs=7, 15000 exemplos, camada oculta com 75% dos neurónios da camada de input e alfa=0.00025

```
Doing the epoch number 1...
RMSE = 1.287148313040332
Doing the epoch number 2...
RMSE = 1.29527854304326
Doing the epoch number 3...
RMSE = 1.3004729916978417
Doing the epoch number 4...
RMSE = 1.739754567608913
Doing the epoch number 5...
RMSE = 2.0385219586648127
Doing the epoch number 6...
RMSE = 2.1062666666666665
Doing the epoch number 7...
RMSE = 1.9601404001857718
Learning time = 206318448000 ms

Test file (path)
C:\\Users\\anaca\\Downloads\\mnist_test.csv
Hit rate = 0.10470000000000002
```

Figura 5 Epochs=7, 15000 exemplos, camada oculta com 75% dos neurónios da camada de input e alfa=0.025

```
Doing the epoch number 1...
RMSE = 1.7703624786112149
Doing the epoch number 2...
RMSE = 1.1283435325832152
Doing the epoch number 3...
RMSE = 0.7179866844275946
Doing the epoch number 4...
RMSE = 0.5877194801560124
Doing the epoch number 5...
RMSE = 0.5288981771277103
Doing the epoch number 6...
RMSE = 0.4987857822011102
Doing the epoch number 7...
RMSE = 0.4817123886216271
Learning time = 135185982200 ms

Test file (path)
C:\\Users\\anaca\\Downloads\\mnist_test.csv
Hit rate = 0.7577
```

Figura 6 Epochs=7, 10000 exemplos, camada oculta com 75% dos neurónios da camada de input e alfa=0.0025

```
Doing the epoch number 1...
RMSE = 1.5674807232631904
Doing the epoch number 2...
RMSE = 0.7922310567898274
Doing the epoch number 3...
RMSE = 0.585619884112994
Doing the epoch number 4...
RMSE = 0.5235131202037921
Doing the epoch number 5...
RMSE = 0.4986755007550738
Doing the epoch number 6...
RMSE = 0.4957129520981052
Doing the epoch number 7...
RMSE = 0.5133048769611379
Learning time = 129501647100 ms

Test file (path)
C:\\Users\\anaca\\Downloads\\mnist_test.csv
Hit rate = 0.7389
```

Figura 7 Epochs=7, 15000 exemplos, camada oculta com 50% dos neurónios da camada de input e alfa=0.0025

```
Doing the epoch number 1...
RMSE = 1.650713041431951
Doing the epoch number 2...
RMSE = 0.8304185540243102
Doing the epoch number 3...
RMSE = 0.606557255245106
Doing the epoch number 4...
RMSE = 0.5464234983020996
Doing the epoch number 5...
RMSE = 0.5187896689529654
Doing the epoch number 6...
RMSE = 0.5171941092586472
Doing the epoch number 7...
RMSE = 0.5160712373299124
Learning time = 257736871600 ms

Test file (path)
C:\\Users\\anaca\\Downloads\\mnist_test.csv
Hit rate = 0.7666
```

Figura 8 Figura 2 Epochs=7, 15000 exemplos, camada oculta com 100% dos neurónios da camada de input e alfa=0.0025

Ana Catarina da Silva Costa Gomes (up201804545)
Cláudia da Costa Maia (up201905492)
Maria Santos Sobral (up201906946)

Como podemos verificar pelas figuras 2 e 3, com o aumento do número de epochs a precisão aumenta, isto porque ao fazer mais vezes a back propagation, mais a rede aprende. No entanto, verificamos que a figura 1 contraria a afirmação anterior, mas sendo apenas uma tentativa, isto pode dever-se ao facto de os pesos serem aleatórios e neste caso, eles terem sido melhor escolhidos.

Para alem disto, é de notar pela figura 2, 4 e 5 que com a diminuição da taxa de aprendizagem, a taxa de erro aumenta, no entanto, com o aumento desta taxa de aprendizagem também a taxa de erro aumenta, isto acontece, pois, se o alpha for demasiado elevado, ocorre um overfitting dos dados de treino.

Em adição, verificamos também que a diminuição do número de exemplos piora a taxa de acerto da rede, pois, a rede fica consequentemente com menos exemplos por onde aprender, mas isso faz com que o algoritmo fique mais rápido.

Por fim, verificamos ainda que o aumento do número de nos da rede oculta provoca uma melhor taxa de acerto, e isto acontece pois como existe mais neurónios cada erro nos pesos conta menos para a quantidade de erros no geral, no entanto, ao fazer a back propagation mais pesos são modificados, logo a rede fica mais assertiva.

CONCLUSÕES

Como era de esperar, são notórias as diferenças de tempo de execução e a taxa de erros entre datasets de tamanhos diferentes. Sendo o que tem mais exemplos é mais lento, no entanto mais preciso. A lentidão deve-se ao facto de ser necessário um número exponencial de iterações a mais, quando o número de exemplos é maior. No entanto, a taxa de erros melhora precisamente porque, como há mais exemplos, a rede fica mais bem treinada, melhorando os valores dos pesos.

Neste sentido, é necessário encontra o valor mais acertado para a taxa de aprendizagem, pois se for muito pequeno a aprendizagem é mais lenta logo a rede não fica suficientemente treinada, porém se a esta taxa for muito elevada, ocorre uma overfitting dos dados de treino e a rede fica também mal treinada.

Por fim, é de notar que a estrutura da rede neuronal implementada não é a melhor, por não ser genérica, e por isso acaba por se perder o intuito de uma boa implementação: a generalidade e abstração, permitindo alterações de alguns parâmetros para avaliar o seu efeito. Assim, não foi possível testar os resultados de uma rede com outra estrutura, por exemplo com mais camadas ocultas, que poderia fazer a diferença pelo facto de cada uma das camadas fazer uma espécie de filtro distinto. No entanto a variação de parâmetros, como: o número de exemplos de treino, o número de epochs e a taxa de aprendizagem, já originaram uma certa variedade de resultados.

Na tentativa de contornar esta adversidade, tentou-se implementar uma estrutura mais genérica: em notação matricial, de modo a facilitar o processo de feed forward e de back propagation. No entanto, a alteração da implementação da função que faria a aprendizagem para esta estrutura não era óbvia, nem imediata, dada a implementação para a estrutura anterior, pelo que segue em anexo essa tentativa com algum erro que não foi possível detetar a tempo. O problema era visível na má qualidade do classificador na fase de testes, revelando-se uma taxa de acerto na ordem dos 10%. Ainda assim, é notória uma melhoria: a redução do tempo de aprendizagem, dada a facilidade com que se passa a informação do input pela rede.

Ana Catarina da Silva Costa Gomes (up201804545)
Cláudia da Costa Maia (up201905492)
Maria Santos Sobral (up201906946)

REFERÊNCIAS BIBLIOGRÁFICAS

• Slides da cadeira Inteligência Artificial (CC2006) do ano letivo 2021/2022

• Artificial Intelligence: a Modern Approach, by Stuart Russell and Peter Norvig, 3rd edition, Prentice Hall

Ana Catarina da Silva Costa Gomes (up201804545)
Cláudia da Costa Maia (up201905492)
Maria Santos Sobral (up201906946)