

UNIVERSIDADE FEDERAL DE MATO GROSSO
CURSO DE ENGENHARIA DE COMPUTAÇÃO

Bruno Fernandes de Lima Santos

Relatório de atividade prática 5:
Classificação de imagens com Redes Neurais Artificiais

Cuiabá, 2017

Bruno Fernandes de Lima Santos

Relatório de atividade prática 5:
Classificação de imagens com Redes Neurais Artificiais

Relatório de projeto referente à atividade prática 5, da disciplina de Inteligência Artificial, curso de Engenharia de Computação, na Universidade Federal do Mato Grosso.

Prof. Dr. Raoni F. S. Teixeira

Cuiabá, 2017

RESUMO

O documento apresenta uma descrição detalhada sobre projeto, desenvolvimento, treinamento e teste de uma Rede Neural Artificial para a classificação de imagens contendo dígitos manuscritos de 0 a 9. Treinamento e teste foram executados com uma amostra contendo 5000 destas imagens e utilizando-se da técnica de validação cruzada. Experimentos utilizando a rede também foram executados, de forma a testar sua funcionalidade.

Palavras-chave: RNA. Redes. Neurais. Classificação.

SUMÁRIO

1	INTRODUÇÃO.....	1
2	MÉTODOS EMPREGADOS.....	3
2.1	Implementação e arquitetura da RNA.....	3
2.1.1	Função de limiar (threshold).....	4
2.1.2	Predição e hipótese (feed forwarding).....	5
2.1.3	Função de custo J e regularização.....	6
2.1.4	Backpropagation: obtendo o vetor gradiente de J.....	7
2.2	Experimentos com a RNA.....	10
2.2.1	Teste de implementação.....	10
2.2.2	Treinamento e avaliação de acurácia.....	10
2.2.3	Obtendo o melhor parâmetro lambda.....	11
3	RESULTADOS E DISCUSSÃO.....	12
3.1	Experimentos com a RNA.....	12
3.1.1	Teste de implementação.....	12
3.1.2	Treinamento e avaliação de acurácia.....	13
3.1.3	Obtendo o melhor parâmetro lambda.....	15
4	CONCLUSÃO.....	17

1 INTRODUÇÃO

Redes neurais artificiais (RNAs) são um tipo de classificador paramétrico não-linear, isto é, tratam de problemas de classificação onde os dados não necessariamente são linearmente separáveis. Atualmente são o tipo de classificador mais utilizados no mundo, sendo amplamente utilizadas na constituição de modelos de classificação multiclasse (não-binário).

A problemática deste trabalho é o desenvolvimento, treinamento e teste de uma RNA capaz de analisar dígitos manuscritos, armazenados em pequenas imagens de dimensão 20×20 pixels, e mapeá-los corretamente ao seu respectivo valor numérico. Por exemplo, na Figura 1.1 a RNA elaborada deve classificar tal imagem como o dígito “6”.

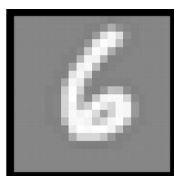


Figura 1.1: Exemplo de dígito manuscrito classificado pela RNA.

A base de dados contendo tais imagens, como as exibidas nas Figura 1.1 e Figura 1.2, é uma pequena amostra proveniente da base de dados do MNIST ¹ de dígitos manuscrito. Ao todo 5000 imagens são utilizadas para treinamento e teste da rede neural desenvolvida neste experimento.

Naturalmente, como em qualquer modelo de classificação, erros ocorrerão. Sendo assim, o objetivo deste projeto é treinar a rede tal que a mesma possua uma acurácia de classificação relativamente alta (considere como alta uma acurácia superior a 90%). Mais detalhes sobre o treinamento da rede e os métodos de otimização utilizados estão contidos na seção 2.

¹ MNIST. Disponível em: <http://yann.lecun.com/exdb/mnist/>. Acesso em: 11 abr 2017.



Figura 1.2: Cem imagens de exemplo de dígitos manuscritos.

Em seguida alguns experimentos são realizados com a RNA implementada, onde alguns parâmetros são alterados e testados. Os experimentos são descritos na seção 2 deste documento e os resultados são discutidos na seção 3.

2 MÉTODOS EMPREGADOS

Foi utilizado o software MatlabTM (versão R2017a) como plataforma de desenvolvimento da RNA, junto ao pacote de ferramentas Statistics and Machine Learning Toolbox, que pode ser baixado da central de downloads do MatlabTM. Foi também utilizada a função de otimização `fmincg`², que utiliza a função de custo e seu vetor gradiente para obter o mínimo global do custo, similar ao Algoritmo do *Gradient Descent*.

2.1 Implementação e arquitetura da RNA

A arquitetura da RNA aqui implementada constitui-se de três camadas e, conseqüentemente, tem-se duas matrizes de parâmetros entre as camadas.

Sejam $A^{(l)}$ a matriz que denota os valores de saída da l -ésima camada e $\Theta^{(l)}$ a matriz dos parâmetros da rede, e que faz a conversão de valores entre as camadas l e $l + 1$. A camada de entrada $A^{(1)} = X$ recebe cada um dos pixels de uma determinada imagem mais o valor de viés 1, totalizando 401 valores de entrada, isto é, $X = (1, x_1, x_2, \dots, x_{400})$ é o vetor contendo os valores da entrada. A segunda camada $A^{(2)}$ possui 25 unidades mais o viés 1, e recebe os dados “filtrados” da camada de entrada através da multiplicação com a matriz de parâmetros $\Theta^{(1)} \in \mathbb{R}^{401 \times 26}$. A terceira e camada final da rede, $A^{(3)}$, possui 10 unidades (cada uma representando um dígito na faixa de 0 a 9) e recebe os dados da camada anterior pelo parâmetro $\Theta^{(2)} \in \mathbb{R}^{26 \times 10}$. A tabela a seguir resume a arquitetura utilizada na implementação desta RNA.

² `fmincg`. Disponível em: <https://www.mathworks.com/matlabcentral/fileexchange/42770-logistic-regression-with-regularization-used-to-classify-hand-written-digits?focused=3791937&tab=function>. Acesso em: 15 abr 2017.

Componente	Notação(ões)	Dimensão
Camada de entrada	$A^{(1)}, X$	401
Camada intermediária	$A^{(2)}$	26
Camada final	$A^{(3)}, h_{\Theta}$	10
Parâmetros de pesos 1	$\Theta^{(1)}$	25×401
Parâmetros de pesos 2	$\Theta^{(2)}$	10×26

Tabela 2.1: Arquitetura da RNA.

Note que a entrada é um vetor de 401 unidades e a saída da rede é um outro vetor de 10 unidades. Dado que as operações da rede são apenas multiplicações de matrizes, é possível receber como entrada uma amostra de imagens em formato matricial $X \in \mathbb{R}^{N \times 401}$, onde N é o número de imagens da amostra (uma imagem em cada vetor linha da matriz). Assim, a rede pode processar de uma vez uma amostra contendo N imagens e devolver uma matriz $h_{\Theta} \in \mathbb{R}^{N \times 10}$ contendo as respectivas N respostas. Consequentemente, a dimensão da camada intermediária também muda, sendo agora $N \times 26$.

2.1.1 Função de limiar (threshold)

Uma função de *soft threshold* foi utilizada na saída de cada neurônio da rede. Trata-se da função sigmoide σ , definida como:

$$\sigma(Z) = \left\{ \frac{1}{1 + e^{-z_{i,j}}} \right\}_{i,j}, \forall z_{i,j} \in Z \quad (2.1)$$

Observe pela notação acima que, caso Z seja uma matriz (ou mesmo um vetor), a função sigmoide é aplicada em cada um de seus elementos, logo, $\dim(\sigma(Z)) = \dim(Z)$.

```
function g = sigmoid(z)
g = 1.0 ./ (1.0 + exp(-z));
end
```

Código 2.1: Função sigmoid.

2.1.2 Predição e hipótese (feed forwarding)

A função de predição é trivial em qualquer RNA. É o resultado de fato da classificação de um determinado exemplo X , com base no cálculo realizado pela função de hipótese.

$$\text{pred}(X) = k \mid X_k = \max(h_{\Theta}(X)) \quad (2.2)$$

Temos que $h_{\Theta}(X)$ (a.k.a. hipótese) é o resultado da última camada da rede. Assim, a função devolve um escalar que corresponde ao índice do vetor $h_{\Theta}(X)$ com a maior probabilidade de ser igual a 1. Esta probabilidade é o resultado da aplicação do *soft thresholding*, que é o último passo realizado pela rede (ver Código 2.3).

```
function p = predict(Theta1, Theta2, X)
[~, p] = max(hyp(Theta1, Theta2, X), [], 1);
p = p';
end
```

Código 2.2: Função predict.

A transposição do vetor p ao final da função serve para corrigir o resultado da operação anterior, onde o vetor resultante acaba sendo um vetor linha (lembre-se que cada linha de X contém um exemplo, logo, $\text{pred}(X)$ deve ser um vetor coluna onde cada uma de suas linhas classifica uma linha de exemplo de X).

O cálculo da hipótese $h_{\Theta}(X)$ é obtido através das seguintes operações, levando em conta a arquitetura desta RNA: $h_{\Theta}(X) = A^{(3)}$, onde cada camada $A^{(l)}$ tem seus valores calculados através da seguinte recorrência.

$$A^{(l)} = \begin{cases} \sigma(\Theta^{(l-1)} A^{(l-1)}) & \text{se } l \neq 1 \\ X & \text{se } l = 1 \end{cases} \quad (2.3)$$

Vale ressaltar que a equação acima presume que os respectivos valores de viés já estão inclusos nos $\Theta^{(l)}$ (sendo assim, a primeira coluna de $A^{(l)}$ é constituída de

valores 1, exceto quando esta é a camada de saída $A^{(3)}$ da rede). Com base nesta definição temos o algoritmo para obtenção da hipótese.

```
function [ A3, A2, A1, Z3, Z2 ] = hyp( Theta1, Theta2, X )
A1 = [ones(1, size(X, 1)) ; X'];
Z2 = Theta1*A1;
A2 = [ones(1, size(Z2, 2)) ; sigmoid(Z2)];
Z3 = Theta2*A2;
A3 = sigmoid(Z3);
end
```

Código 2.3: Função de hipótese.

Note que valores de viés (ou *bias*) já estão incluídos nas matrizes de $\Theta^{(1)}$ (Theta1) e $\Theta^{(2)}$ (Theta2) e, por isso, há a inclusão das colunas de 1 nas camadas inicial e intermediária da rede. As múltiplas saídas da função de hipótese acima são utilizadas posteriormente durante a otimização do custo (erro) da RNA com o Algoritmo Backpropagation.

2.1.3 Função de custo J e regularização

A função de custo calcula o erro total da rede dado o parâmetro Θ . A seguir tem-se a definição de uma função de custo J com regularização.

$$J(\Theta) = (C + R)(\Theta) \quad (2.4)$$

Onde C é o custo bruto da rede e R é a regularização da rede (prevenindo sobreajuste, ou *overfitting*, dos dados). Ambos são definidos como:

$$C(\Theta) = \frac{1}{m} \sum_i \sum_k (-y_i^k \log(h_{\Theta}(X_i)_k) - (1 - y_i^k) \log(1 - h_{\Theta}(X_i)_k)) \quad (2.5)$$

$$R(\Theta) = \frac{\lambda}{2m} \left(\sum_i \sum_j (\theta_{i,j}^{(1)})^2 + \sum_i \sum_j (\theta_{i,j}^{(2)})^2 \right), \forall \theta_{i,j}^{(l)} \in \Theta^{(l)} \quad (2.6)$$

Onde λ (lambda) é o peso atribuído à regularização da RNA, m é a quantidade de exemplos da amostra e X_i é o i -ésimo exemplo. A seguir temos um pedaço de código da função de custo, que calcula o custo J definido na equação 2.4.

```
m = size(X, 1);
I = eye(num_labels);
Y = zeros(m, num_labels);
for i=1:m
    Y(i, :) = I(y(i), :);
end
H = hyp(Theta1, Theta2, X)';
J = sum(sum(-(Y.*log(H)) - (1 - Y).*log(1 - H)))/m;
if lambda ~= 0,
    J = J + (lambda/(2*m))*(sum(sum(Theta1(:, 2:end).^2)) + ...
        sum(sum(Theta2(:, 2:end).^2)));
end
```

Código 2.4: Parte da função de custo (com a regularização).

A variável `num_label` acima é igual à quantidade de rótulos (ou classes) possíveis que, neste caso, vale 10. Note que, na regularização, a primeira coluna de $\Theta^{(1)}$ e $\Theta^{(2)}$ são descartadas. Isto porque tais colunas correspondem aos valores de viés de cada vetor linha de $\Theta^{(l)}$ e não devem ser incluídos nos cálculos da regularização.

2.1.4 Backpropagation: obtendo o vetor gradiente de J

Para a otimização da acurácia da rede, é necessário obter o vetor gradiente do custo, ∇J , de forma a minimizar o erro. O algoritmo responsável pela obtenção de ∇J é o Algoritmo *Backpropagation*, que foi empregado aqui com base na arquitetura descrita pela Tabela 2.1.

Para o Algoritmo Backpropagation, os exemplos são preditos pela rede (método de *feed forwarding*). A partir daí o erro da predição é calculado na última camada e propagado para trás nos cálculos dos erros das camadas anteriores.

Sejam $\delta^{(l)}$ o erro acumulado na l -ésima camada. Temos então que:

$$\delta^{(3)} = A^{(3)} - Y \quad (2.7)$$

Onde Y é o resultado de fato do(s) exemplo(s) e $A^{(3)}$ contém os respectivos valores preditos. Então, os erros das camadas anteriores são calculados da seguinte forma:

$$\delta^{(2)} = \left(\left(\Theta^{(2)} \right)^\top \delta^{(3)} \right) .* \left(\sigma'(Z^{(2)}) \right) \quad (2.8)$$

Onde σ' é a derivada da função sigmoide e $Z^{(2)}$ é o resultado de entrada da camada $A^{(2)}$ sem o viés e sem aplicação da sigmoide, isto é, $Z^{(2)} = \Theta^{(1)} A^{(1)}$. O operador $.*$ indica multiplicação elemento a elemento das matrizes, onde, para isto, suas dimensões devem ser iguais. A derivada da sigmoide foi implementada utilizando o seguinte pedaço de código.

```
function g = sigmoidGradient(z)
s = sigmoid(z);
g = s.*(1 - s);
end
```

Código 2.5: Função de derivada (gradiente) da sigmoide.

Os erros para $\Theta^{(1)}$ e $\Theta^{(2)}$ são então acumulados e as derivadas parciais constituintes de ∇J são obtidas:

$$\Delta^{(l)} = \delta^{(l+1)} \left(A^{(l)} \right)^\top \quad (2.9)$$

$$\frac{\partial J}{\partial \Theta^{(l)}} = \frac{1}{m} \Delta^{(l)} \quad (2.10)$$

O algoritmo descrito pelas equações acima são implementados pelo seguinte pedaço de código.

```

[A3, A2, A1, ~, Z2] = hyp(Theta1, Theta2, X);
G3 = A3 - Y';
T = (Theta2')*G3; % Matriz temporária T
T = T(2:end, :); % Remove a coluna de viés de G3
G2 = T.*sigmoidGradient(Z2);
Theta1_grad = (1/m)*(G2*A1');
Theta2_grad = (1/m)*(G3*A2');

```

Código 2.6: Algoritmo Backpropagation (sem regularização).

A seguir temos o código que implementa a regularização de $\frac{\partial J}{\partial \Theta^{(1)}}$ e $\frac{\partial J}{\partial \Theta^{(2)}}$.

```

if lambda ~= 0,
    R1 = (lambda/m)*Theta1;
    R2 = (lambda/m)*Theta2;
    R1(:, 1) = zeros(size(R1, 1), 1); % Descarta o viés de Theta1
    R2(:, 1) = zeros(size(R2, 1), 1); % Descarta o viés de Theta2
    Theta1_grad = Theta1_grad + R1;
    Theta2_grad = Theta2_grad + R2;
end

```

Código 2.7: Regularização do grad J.

Note que as colunas de viés foram substituídas por zeros. Assim, as somas que ocorrem em seguida não levam em conta a regularização do viés.

Por fim, ∇J é acumulado pela seguinte linha de código, resultante em um enorme vetor coluna. Este gradiente é utilizado pela função `fmincg` para obter o mínimo da função de custo.

```

grad = [Theta1_grad(:) ; Theta2_grad(:)];

```

Código 2.8: Obtenção de grad J após Backpropagation.

2.2 Experimentos com a RNA

Alguns experimentos com a RNA foram realizados a fim de testar sua funcionalidade e sua acurácia. Os resultados de todos os experimentos são listados e discutidos em subseções da seção 3 de respectivo nome.

2.2.1 Teste de implementação

Um pequeno teste foi executado para testar a implementação dos algoritmos de predição, custo e gradiente do custo. Para isto, foi fornecido pelo professor da disciplina um script que realiza esse teste. Devido ao tamanho do código de tal script, a exibição deste será omitida aqui. Uma amostra de valores de $\Theta^{(1)}$ e $\Theta^{(2)}$ também foi fornecida para testar a implementação do custo J .

Um algoritmo para obtenção de ∇J de forma analítica também foi fornecido e executado, e os resultados de ambos algoritmos (analítico e o Backpropagation) foram comparados. Para que a implementação do Backpropagation esteja correta, foi definido que a diferença relativa entre os dois algoritmos (isto é, a diferença entre os resultados) deveria ser inferior a 10^{-9} .

2.2.2 Treinamento e avaliação de acurácia

O treinamento da rede ocorre da seguinte forma: Inicia-se com valores aleatórios para $\Theta^{(1)}$ e $\Theta^{(2)}$. Em seguida, um ponteiro da função de custo é passado como parâmetro para a função `fmincg`, chamada para a otimização do custo (note que, para que isto ocorra, a função de custo deve devolver tanto o valor do custo como seu vetor gradiente). Um dos parâmetros de `fmincg` é o número de iterações que a função fará para aproximar J do mínimo global. A fim de se evitar o *overfitting*, o método de validação cruzada (k-fold) foi empregado no conjunto de dados X , cujo parâmetro $k = 3$ foi escolhido (k é o número de separações entre treinamento e teste que a validação cruzada faz). Além disso, foi escolhido neste experimento $\lambda = 2$ para a regularização.

2.2.3 Obtendo o melhor parâmetro lambda

Durante o treinamento da rede deve-se levar em conta qual o valor do parâmetro λ na regularização do modelo (equação 2.6). Um λ muito pequeno significa pequeno peso para a regularização, o que pode acarretar em *overfitting*, enquanto um λ muito grande pode implicar em *underfitting*. Para isto, a seguinte amostra Λ , contendo diversos valores de λ escolhidos arbitrariamente, foi testada e os resultados foram analisados, também utilizando validação cruzada.

$$\Lambda = \{0, 10^{-6}, 1, 2, 3, 4, 10, 10^3, 10^6\} \quad (2.11)$$

O que ocorre quando o modelo não é regularizado ($\lambda = 0$)? E quando o peso atribuído à regularização é exageradamente alto? Os resultados serão discutidos na próxima seção.

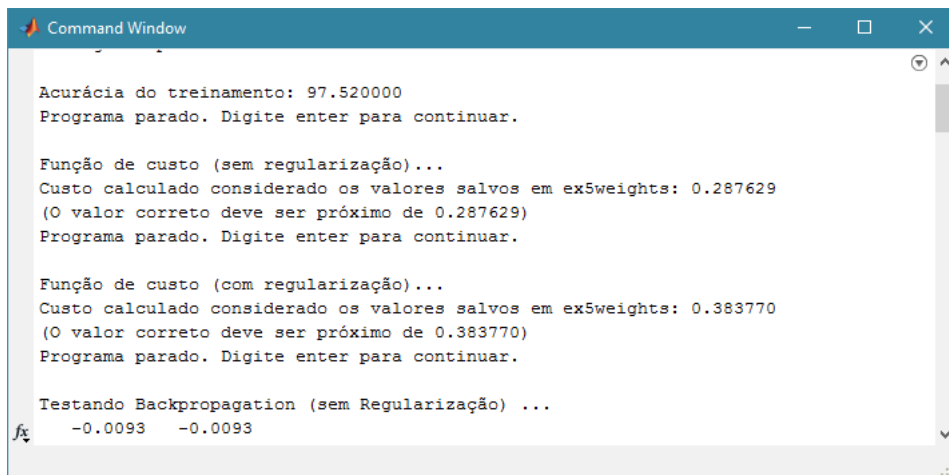
3 RESULTADOS E DISCUSSÃO

3.1 Experimentos com a RNA

Os resultados de cada experimento serão exibidos aqui da melhor forma encontrada.

3.1.1 Teste de implementação

O script fornecido, bem como a amostra contendo os parâmetros $\Theta^{(l)}$, foi executado e parte de seu resultado é exibido na imagem a seguir, em impressão da tela de terminal de comandos do Matlab.



```
Command Window

Acurácia do treinamento: 97.520000
Programa parado. Digite enter para continuar.

Função de custo (sem regularização)...
Custo calculado considerado os valores salvos em ex5weights: 0.287629
(O valor correto deve ser próximo de 0.287629)
Programa parado. Digite enter para continuar.

Função de custo (com regularização)...
Custo calculado considerado os valores salvos em ex5weights: 0.383770
(O valor correto deve ser próximo de 0.383770)
Programa parado. Digite enter para continuar.

Testando Backpropagation (sem Regularização) ...
fx -0.0093 -0.0093
```

Figura 3.1: Parte da saída do script de testes, exibida no terminal do Matlab.

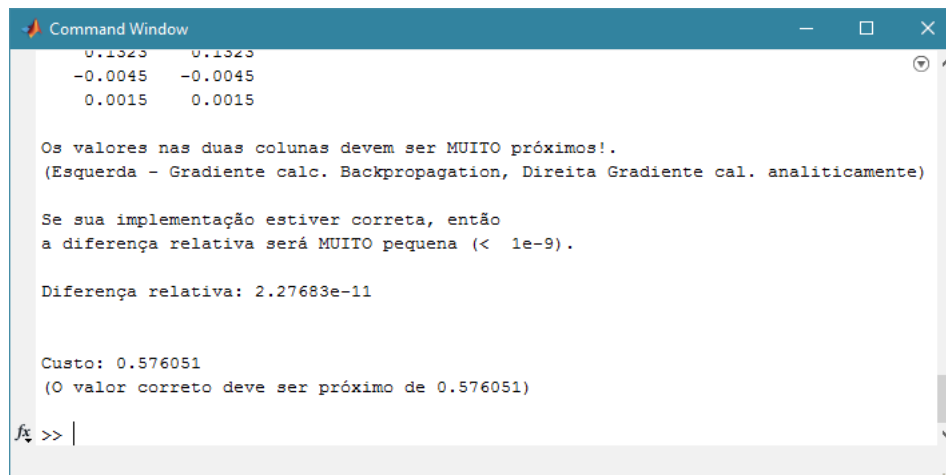
Observa-se que os valores de custo (com ou sem regularização) são idênticos aos seus valores calculados de forma teórica, o que mostra que a implementação do custo J está correta.

Em seguida é testado a implementação do Algoritmo Backpropagation. Dois testes foram executados: um desconsiderando a regularização ($\lambda = 0$) e outro considerando a regularização $\lambda = 3$. Os resultados são resumidos na tabela a seguir.

Algoritmo testado	Diferença relativa
Backpropagation sem reg.	2.3671e-11
Backpropagation com reg.	2.2768e-11

Tabela 3.1: Resultados do teste de validação do Backpropagation.

Como o objetivo era obter uma diferença relativa inferior a 10^{-9} , vemos que, de acordo com os resultados deste experimento, a implementação do Backpropagation também está correta.



```

Command Window
0.1323    0.1323
-0.0045   -0.0045
0.0015    0.0015

Os valores nas duas colunas devem ser MUITO próximos!.
(Esquerda - Gradiente calc. Backpropagation, Direita Gradiente cal. analiticamente)

Se sua implementação estiver correta, então
a diferença relativa será MUITO pequena (< 1e-9).

Diferença relativa: 2.27683e-11

Custo: 0.576051
(O valor correto deve ser próximo de 0.576051)

fx >> |

```

Figura 3.2: Parte da saída do script de testes, com o comparativo entre os algoritmos.

Com isto, concluímos que a implementação desta RNA está correta e podemos então prosseguir com os outros experimentos.

3.1.2 Treinamento e avaliação de acurácia

Aqui foi utilizada a função `fmincg` para o treinamento da rede (otimização do custo). Um dos parâmetros desta função é o número máximo de iterações, isto é, o número máximo de aproximações sucessivas do valor mínimo da função na qual se quer otimizar. Neste experimento, foi definido o valor de 500 iterações para a `fmincg`. Em seguida, executou-se a validação cruzada com 3 subdivisões, e obtidos os respectivos custos. Alguns valores de custo e iteração estão contidos na tabela abaixo.

Iteração	Custo 1	Custo 2	Custo 3
1	3.323328e+00	3.317392e+00	3.314868e+00
10	1.580734e+00	1.869649e+00	1.916741e+00
50	7.292524e-01	7.354726e-01	7.737697e-01
100	6.925289e-01	6.876965e-01	7.161133e-01
500	6.594349e-01	6.505961e-01	6.731439e-01

Tabela 3.2: Alguns valores de custo \times iteração na otimização do custo.

Com isto, os parâmetros $\Theta^{(1)}$ e $\Theta^{(2)}$ foram otimizados e testados em cada iteração da validação cruzada, resultando nos respectivos valores de acurácia. Em seguida a acurácia média foi calculada, resultando em 93.159879 %.

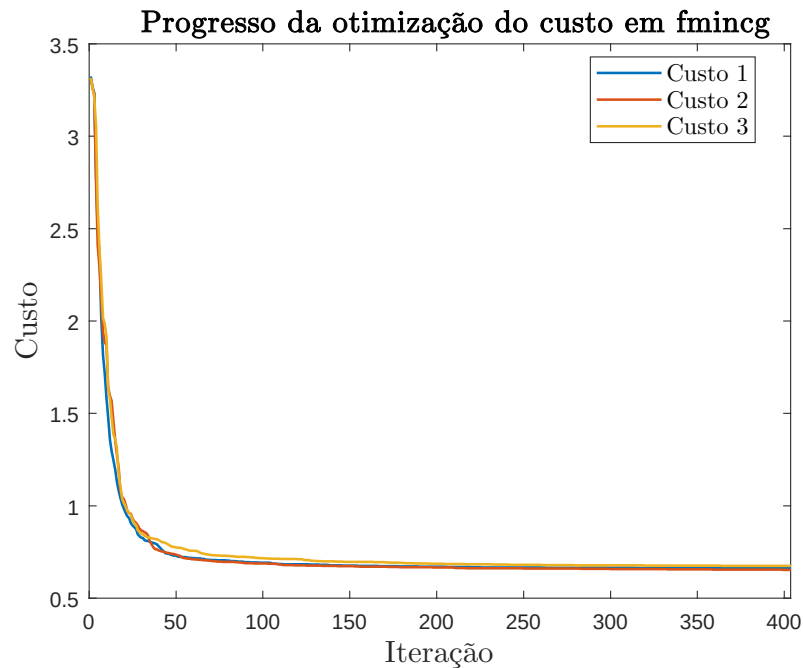


Figura 3.3: Otimização do custo.

Pelo gráfico acima vemos que não são necessárias muitas iterações para alcançar um custo relativamente baixo. Note também que as curvas convergem a um determinado valor de custo, sendo este o mínimo da função. Valores ainda menores de custo podem ser obtidos alterando-se a arquitetura da RNA, inserindo mais camadas intermediárias, por exemplo.

3.1.3 Obtendo o melhor parâmetro lambda

Neste experimento utilizamos 50 iterações da função `fmincg` para cada $\lambda \in \Lambda$ (equação 2.11). A acurácia média para cada valor de λ foi então analisada.

Valor de lambda	Acurácia média (%)
0	92.0599
1e-6	91.9599
1	92.3000
2	91.9599
3	91.5198
4	91.5599
10	90.7999
1e+3	9.9600
1e+6	9.9600

Tabela 3.3: Acurácia do modelo em função do lambda.

Pelos resultados da tabela acima podemos inferir que não houve *overfitting* quando desconsideramos a regularização do modelo. Contudo, para valores de λ muito grandes, ocorreu *underfitting*, comprometendo sua acurácia. Note também que a acurácia para $\lambda = 3$ foi ligeiramente inferior à acurácia medida no experimento anterior (seção 3.1.2) devido ao número reduzido de iterações neste experimento. Conclui-se então que o valor ótimo, para este experimento e para esta faixa de valores para λ (amostra Λ), é $\lambda = 1$.

Algo interessante ocorre quando visualizamos o gráfico do custo médio do modelo a cada iteração e a cada valor de λ . O *underfitting* do modelo é visível quando aumentamos demasiadamente o valor de λ (ver Figura 3.4).

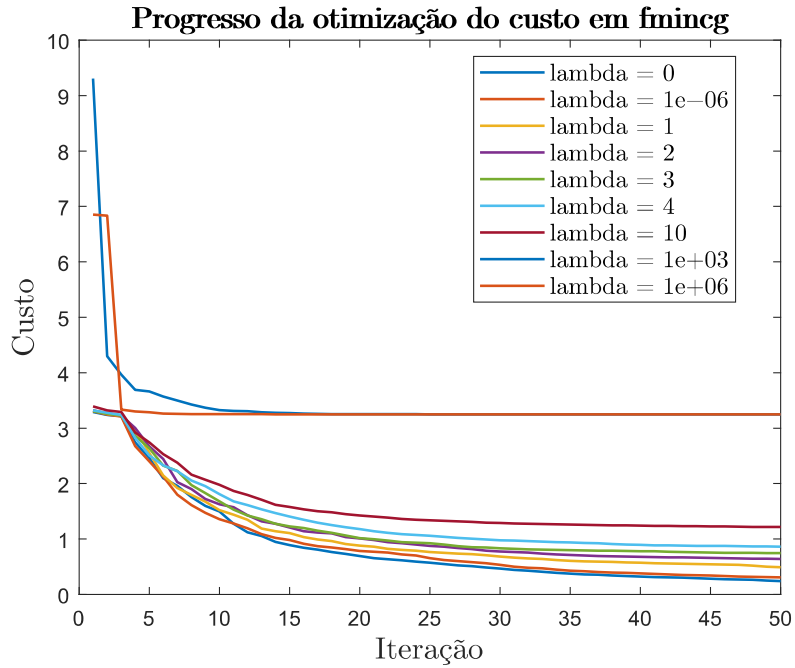


Figura 3.4: Otimização do custo para diversos valores de lambda.

Analizando os dados contidos na Tabela 3.3 e na Figura 3.4, vemos que a função de custo possui melhores resultados quando $\lambda = 0$ ou $\lambda = 10^{-6}$ mas a melhor acurácia ocorre quando $\lambda = 1$, possivelmente sendo indício de um leve *overfitting* (lembre-se que o cálculo do custo ocorre durante a etapa de treinamento da rede). Note também que o *overfitting* do modelo não é tão grande possivelmente devido ao pequeno conjunto de dados disponível (5000 imagens). Na validação cruzada, a cada k -ésima iteração temos $\frac{5000}{k}$ imagens para testarmos o modelo e $\frac{5000(k-1)}{k}$ imagens para seu treinamento. Com $k = 3$ temos um equilíbrio entre um conjunto de testes não muito reduzido e uma quantidade suficiente de validações. Contudo, a quantidade de dados brutos de 5000 imagens continua sendo um conjunto pequeno para visualização de sobreajustes.

4 CONCLUSÃO

A construção de uma RNA com uma acurácia suficiente para um dado problema depende de alguns fatores: sua arquitetura, contendo um número suficiente de camadas intermediária e parâmetros $\Theta^{(l)}$ para “filtrarmos” mais os dados, o número de passos de otimização (neste trabalho, o número de iterações da função de otimização *fmincg*) e a quantidade de dados disponíveis para treinamento e teste, onde uma maior quantidade de dados privilegia a acurácia final do modelo.

Com todo o material disponível, foi possível construir uma RNA com acurácia superior a 90 % conforme proposto, e com os experimentos realizados foi possível obter parâmetros otimizados para a regularização do modelo. Foi escolhido a técnica de validação cruzada para avaliação do modelo, embora a técnica de *bootstrap* também poderia ser utilizada sem prejuízos de avaliação.