Atividade Prática 5: Redes Neurais

Inteligência Artificial

Raoni F. S. Teixeira

Introdução

Este documento é parte da quinta avaliação prática da disciplina de inteligência artificial ministrada na UFMT no segundo semestre de 2016. Nesta atividade, você irá implementar um algoritmo para reconhecimento de dígitos em imagens. Lembre-se de que esta é uma atividade individual e é muito importante que você escreva a sua própria solução e o seu próprio relatório.

Antes de iniciar o exercício, recomendamos fortemente a leitura atenta do Capítulo 18 do livro texto [1] e a consulta ao material sobre Octave/MATLAB disponível na página da disciplina [2].

Arquivos incluídos na atividade

- `ex5.m` Script Octave/Matlab de teste que orientará você no exercício.
- `ex5data.mat` Dados utilizados na atividade.
- `ex5weights.mat` Parâmetros $\Theta^{(1)}$ e $\Theta^{(2)}$ pré-calculados para teste.
- `displayData.m` Função que ajuda a visualizar a base de dados.
- `fmincg.m` Função que implementa uma rotina de otimização.
- `sigmoid.m` Função sigmoid.

[1] Stuart Russell e Peter Norvig. Inteligência Artificial. Tradução da 3a edição. Editora Campus/Elsevier.

[2] `http://www.students.ic.unicamp.br/~ra089067/ensino/2016_2/ia.html`

- `computeNumericalGradient.m` Função que calcula numericamente o gradiente.
- `checkNNGradients.m` Função que ajuda a verificar a implementação do algoritmo brackpropagation.
- `randInitializeWeights.m` Função que atribui valores aleatórios aos parâmetros da rede.
- `sigmoidGradient.m` Função que implementa a derivada da função sigmoid.
- `cost_function.m`$^+$ Função que deve implementar função de custo e seu vetor gradiente (via backpropagation).
- `predict.m`$^+$ Função que deve implementar a propagação feedfoward e classificar amostras.

Todos os arquivos marcados com $^+$ devem ser implementados (alterados).

---

1 Redes Neurais

Nesta atividade, você irá escrever um agente baseado em rede neural para o reconhecimento de dígitos manuscritos em imagens. Como vimos em aula, uma rede neural artificial é um classificador não-linear muito útil quando temos de lidar com dados naturais (e.g., fala, visão). Uma breve discussão dos passos que você deve executar são apresentados a seguir (consulte também o arquivo `ex5.m`).

1.1 Base de dados

O conjunto de dados utilizado nesta atividade é uma pequena parte da base MNIST de dígitos manuscritos [3]. A amostra considerada contém 5000 imagens de tamanho 20×20. Para facilitar um pouco a implementação, estas imagens foram pré-processados e uma matriz `X` com tamanho 5000×400 foi criada (n.b.: cada linha de `X` é uma imagem). Também foi criado um vetor

[3] `http://yann.lecun.com/exdb/mnist/`

`y` com a classe de cada imagem (i.e., `y` tem tamanho 5000×1). Os valores de `y` variam de 1 à 10. O dígito 0 (zero) é representado pelo número 10 e os outros dígitos são representados de maneira natural (i.e., os dígitos de 1 à 9 são representados utilizando os números de 1 à 9). O resultado deste processamento foi salvo no arquivo `ex5data.mat`.

Assim, por exemplo, para visualizar a primeira imagem pode-se:

- carregar o arquivo fazendo `load('ex5data.mat')` e
- mostrar a imagem fazendo `displayData(X(1,:))`.

O código no início do arquivo `ex5.m` faz algo parecido para exibir um conjunto aleatório de imagens.

1.2 Representação do modelo

A Figura 1 apresenta a arquitetura da rede neural considerada neste trabalho. Esta rede tem 3 (três) camadas: entrada, intermedária e saída. Como estamos trabalhando com imagens de tamanho 20×20, precisamos de 400 unidades na camada de entrada (uma unidade para cada pixel da imagem). Há ainda 25 unidades na camada intermediária e 10 unidades de saída (uma para cada dígito possível). Nesta representação não incluímos a unidade +1 que corresponde ao viés (*bias*) do modelo.

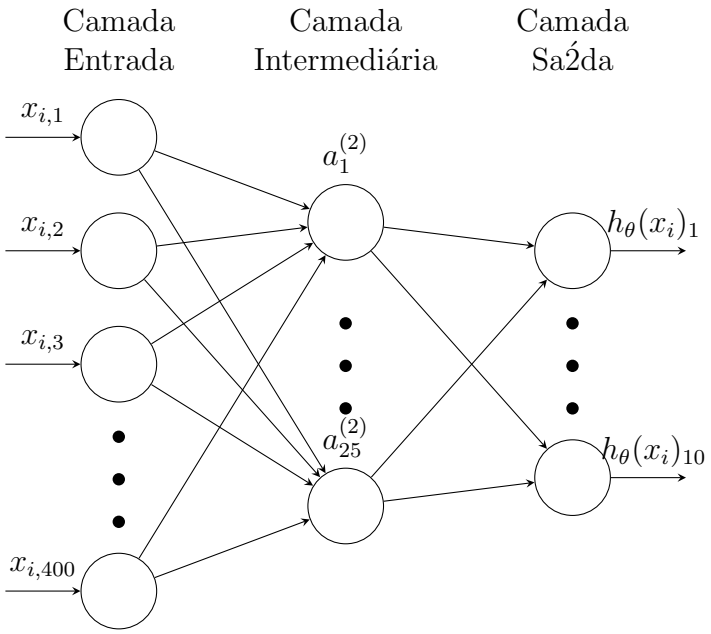

Figura 1: Arquitetura da rede.

Nesta representação, os parâmetros $\Theta^{(1)}$ e $\Theta^{(2)}$ tem tamanho 25×401 e 10×26, respectivamente.

1.3 Propagação feedfoward e classificação

Na primeira parte da atividade, você deve implementar a propagação *feedfoward* para classificar cada imagem i. Para tanto você deve preencher o arquivo `predict.m` que deve devolver um número inteiro entre 1 e 10 indicando a qual classe a imagem pertence.

Tal como vimos em sala, este processo envolve os seguintes cálculos:

1. $a^{(1)} = x_i$ (não esqueça do viés i.e. faça: $a_0^{(1)} = 1$);
2. $z^{(2)} = \Theta^{(1)} a^{(1)}$;
3. $a^{(2)} = g(z^{(2)})$ (não esqueça do viés i.e. faça: $a_0^{(2)} = 1$);
4. $z^{(3)} = \Theta^{(2)} a^{(2)}$ e
5. $a^{(3)} = h_\Theta(x_i) = g(z^{(3)})$.

em que $g(z)$ é a função sigmoid que está implementada no arquivo `sigmoid.m` e $h_\Theta(x_i)$ é um vetor com 10 elementos. Para identificar a classe prevista pelo modelo, você pode utilizar a função `max` que devolve também o índice do elemento com maior valor em um vetor.

Depois de terminar, você pode testar sua implementação utilizando os parâmetros `Theta1` e `Theta2` gravados no arquivo `ex5weights.mat` (carregue-o com o comando `load`). A acurácia (precisão) do reconhecimento com estes parâmetros deve ser igual a 97.5%.

1.4 Função de custo

Agora você deve preencher o arquivo `cost_function.m`. A função de custo para uma rede neural com regularização é dada por:

$$J(\Theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}[\ -y_i^k\ \log(h_\Theta(x_i)_k) -\ \ (1-y_i^k)\ \log(1-h_\Theta(x_i)_k)] + \frac{\lambda}{2m}\sum_{i=1}^{25}\sum_{j=1}^{400}(\Theta_{i,j}^{(1)})^2 + \sum_{i=1}^{10}\sum_{j=1}^{25}(\Theta_{i,j}^{(2)})^2$$

em que K obviamente vale 10 (10 dígitos) e m é a quantidade de amostras (imagens) a serem consideradas. Os índices 25 e 10 (nas dimensões de $\Theta^{(1)}$ e $\Theta^{(2)}$) estão relacionados com a arquitetura da rede.

Nesta fórmula, assumimos a classe está codificado em um vetor e por isto há dois índices em y_i^k. O código a seguir faz esta mudança de representação.

```
I = eye(num_labels);
Y = zeros(m, num_labels);
for i=1:m
  Y(i, :)= I(y(i), :);
end
```

`Y` é uma matriz cujas linhas correspondem a representação vetorial da classe.

1.5 Backpropagation

A principal parte desta atividade é a implementação do algoritmo backpropagation. Para cada amostra do treinamento (x_t, y_t), este algoritmo calcula

a ativação da última camada da rede (via propagação feedfoward) e, em seguida, calcula (*de trás pra frente*) uma estimativa do erro em cada unidade da rede. Tal como vimos, isto é feito da seguinte maneira:

1. Calcule os valores da última camada da rede $a^{(3)}$ utilizando o processo feedfoward (veja Seção 1.3).

2. Calcule o erro na camada de saída da rede:

$$\Gamma^{(3)} = (a^{(3)} - y). \tag{1}$$

3. Calcule o erro para segunda camada ($l = 2$) da seguinte maneira:

$$\Gamma^{(2)} = (\Theta^{(2)})^T \Gamma^{(3)} .* g'(z^{(2)}). \tag{2}$$

   $g'(z)$ corresponde a derivada da função sigmoid e está implementada no arquivo `sigmoidGradient.m`.

4. Acumule o gradiente deste exemplo para cada camada l da rede:

$$\Delta^{(l)} = \Delta^{(l)} + \Gamma^{(l+1)}(a^{(l)})^T. \tag{3}$$

5. Calcule a estimativa do gradiente utilizando a seguinte fórmula:

$$\frac{\partial}{\partial \Theta_{i,j}^{(l)}} J(\Theta) = D_{i,j}^{(l)} = \frac{1}{m} \Delta_{i,j}^{(l)}. \tag{4}$$

   novamente m é quantidade de imagens de treinamento. i e j correspondem aos índices de cada dimensão da matriz $\Theta^{(l)}$. O gradiente é um vetor coluna com as derivadas de cada parâmetro $\Theta^{(1)}$ e $\Theta^{(2)}$. Você deve remover $\Gamma_0^{(2)}$ (em Octave/Matlab, isto pode ser feito fazendo `gamma2 = gamma2(:, 2:end)`).

Depois de terminar a implementação execute o arquivo `ex5.m` e teste sua implementação.

Em seguida, implemente a regularização do gradiente da seguinte maneira:

$$\frac{\partial}{\partial \Theta_{i,j}^{(l)}} J(\Theta) = D_{i,j}^{(l)} = \frac{1}{m} \Delta_{i,j}^{(l)}, \quad \text{para } j = 0$$

$$\frac{\partial}{\partial \Theta_{i,j}^{(l)}} J(\Theta) = D_{i,j}^{(l)} = \frac{1}{m} \Delta_{i,j}^{(l)} + \frac{\lambda}{m} \Theta_{i,j}^{(l)}, \quad \text{para } j \geq 1$$

ou seja, o viés (+1) não deve ser regularizado.

2 Exercício opcional (extra)

Você pode optar por resolver uma tarefa extra que consiste em comparar os resultados da rede neural com o algoritmo de regressão logística. O classificador baseado em regressão logítica para o problema de reconhecimento de dígitos está implementado no arquivo `reglog.zip`.

3 Relatório, entrega e notas

Depois de terminar a implementação, você deve escrever um pequeno relatório contendo obrigatoriamente:

- uma seção "Resumo" que deve claramente contextualizar e apresentar os principais resultados do trabalho e
- uma seção "Resultados e discussões" em que os resultados (i.e., tabelas, gráficos) devem ser apresentados e interpretados.

Os resultados dos experimentos devem ser analisados utilizando validação cruzada ou amostragem *bootstrap*.

O relatório e os códigos devem ser entregues até o dia **13 de abril de 2017**. A pontuação de cada tarefa deste exercício é apresentada a seguir.

Tarefa	**Arquivo**	**Pontuação (nota)**
FeedFoward	`predict.m`	2.0
Função custo e gradiente	`cost_function.m`	5.0
Relatório	`ap5.pdf`	3.0
Comparação de métodos	`ap5.pdf`	3.0 (extra)