Universidade Federal de Santa Catarina
Programa de Pós-Graduação em Ciência da Computação
Inteligência Artificial Conexionista

Augusto André Souza Berndt 202000735

Exercício 3.

Neste exercício é feito a classificação de um problema multi classe com o dataset MNIST. O dataset se trata de imagens em preto e branco de dígitos escritos à mão, este dataset foi desenvolvido utilizando-se da escrita de crianças de uma certa e escola e funcionários de uma empresa e é composto de um total de 60 mil imagens de treinamento e 10 mil imagens de teste, mas o exercício utiliza uma versão em matlab em que o conjunto de treinamento possui 50 mil imagens.

As imagens são organizadas na matriz X de maneira que se tem 50 mil vetores, sendo que cada vetor representa uma imagem, ou seja, ela é um vetor contínuo que conecta cada linha na matriz da imagem. Os valores de y armazenam os labels das imagens, sendo composto por um valor inteiro de 0 a 9.

É implementado uma regressão logística estilo *um contra todos* na intenção de proporcionar um modelo capaz de classificar corretamente os dígitos que são escritos na correspondente imagem.

A biblioteca numpy foi utilizada para realizar os problemas do exercício. Não se deve utilizar laços *for* para equações de base, como de custo ou gradiente. Os códigos devem funcionar de maneira vetorizada e parametrizada. A biblioteca numpy ajuda bastante neste quesito, com uma fácil codificação que trata as multiplicações vetorizadas de matrizes com alta abstração.

No primeiro desafio é requisitado que se implemente uma função custo e o gradiente para uma regressão logística. A função *sigmoid* implementada no exercício anterior é disponibilizado também na pasta *Utils*. A Figura 1 mostra que a função custo *J* e o gradiente foram implementados da maneira como esperado.

Once you finished your implementation, you can call the function `lrCostFunction` to test your solution using the following cell:

```
J, grad = lrCostFunction(theta_t, X_t, y_t, lambda_t)

print('Cost         : {:.6f}'.format(J))
print('Expected cost: 2.534819')
print('-----------------------')
print('Gradients:')
print(' [{:.6f}, {:.6f}, {:.6f}, {:.6f}]'.format(*grad))
print('Expected gradients:')
print(' [0.146561, -0.548558, 0.724722, 1.398003]');
```

```
Cost         : 2.534819
Expected cost: 2.534819
-----------------------
Gradients:
 [0.146561, -0.548558, 0.724722, 1.398003]
Expected gradients:
 [0.146561, -0.548558, 0.724722, 1.398003]
```

Figura 1.

O segundo desafio se dá em implementar a organização dos valores de *um contra todos* a partir das funções anteriormente implementadas. O código para este desafio já se encontra nos comentários, só precisou ser organizado e adicionado um laço *for* e definir a escrita em *all_theta,* sendo este a variável de saída da função. A função *optimize.minimize* da biblioteca *scipy* é utilizada, no qual efetua a minimização de uma função escalar para otimizar o valor de *theta*. Esta variável vetorizada de saída tem shape de (10, 401), ou seja, temos 10 perceptrons e cada um tem 400 valores de pesos mais o valor de bias.

A predição é feita e alguns valores de acurácia são obtidos através da modificação de parâmetros de entrada na minimização dos thetas. A Figura 2 mostra o valor de acurácia para valores de parâmetro padrões do exercício.

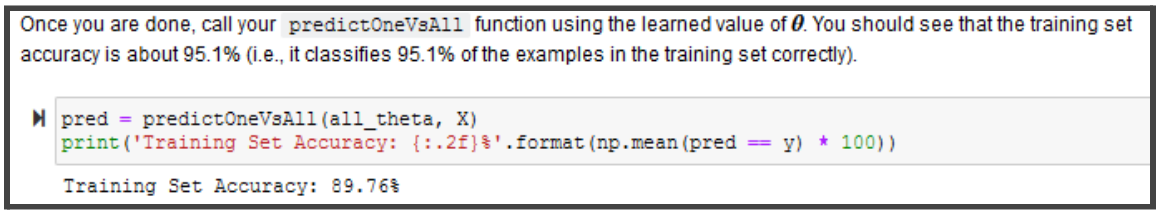


Figura 2.

A Figura 3 demonstra a utilização da função *np.random.randn* para experimentar novos valores iniciais de theta, porém não se obteve melhores resultados do que se inicializado com zeros, como por padrão na Figura 2.

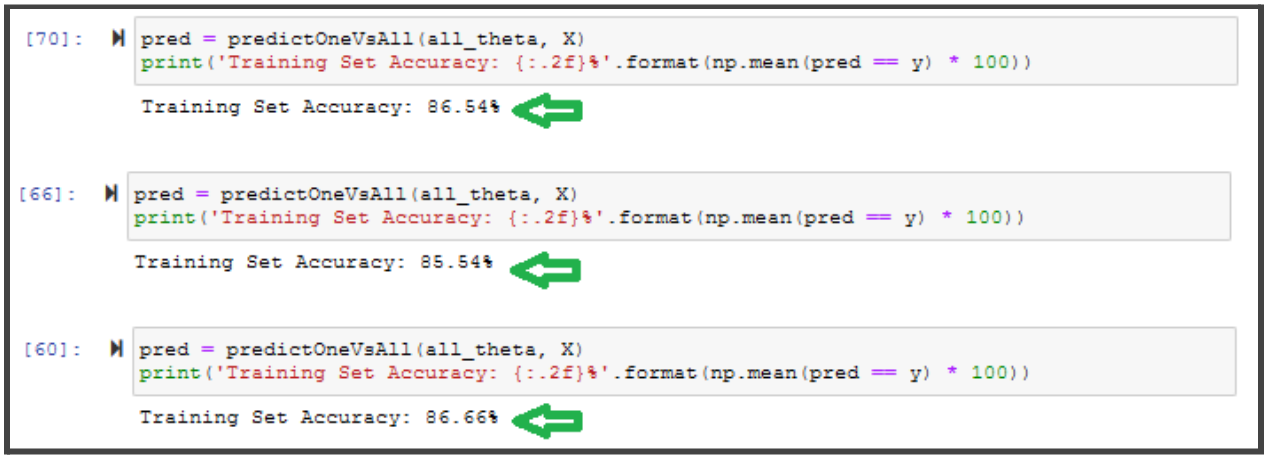


Figura 3.

O método de minimização por padrão é o TNC *truncated Newton,* alterando-se para o método CG se obteve um acurácia de treino bem superior, com mais de 5% de melhora, como demonstra a Figura 4.

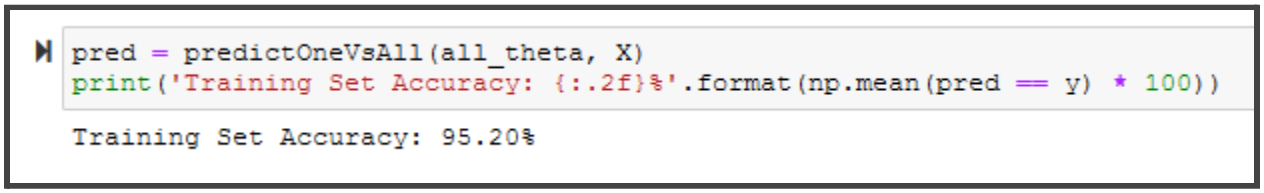


Figura 4.

Além da utilização do método CG, também se obteve um aumento de quase 1%, como demonstra a Figura 5, fazendo-se 100 iterações em vez de 50.

```
pred = predictOneVsAll(all_theta, X)
print('Training Set Accuracy: {:.2f}%'.format(np.mean(pred == y) * 100))
```

```
Training Set Accuracy: 96.12%
```

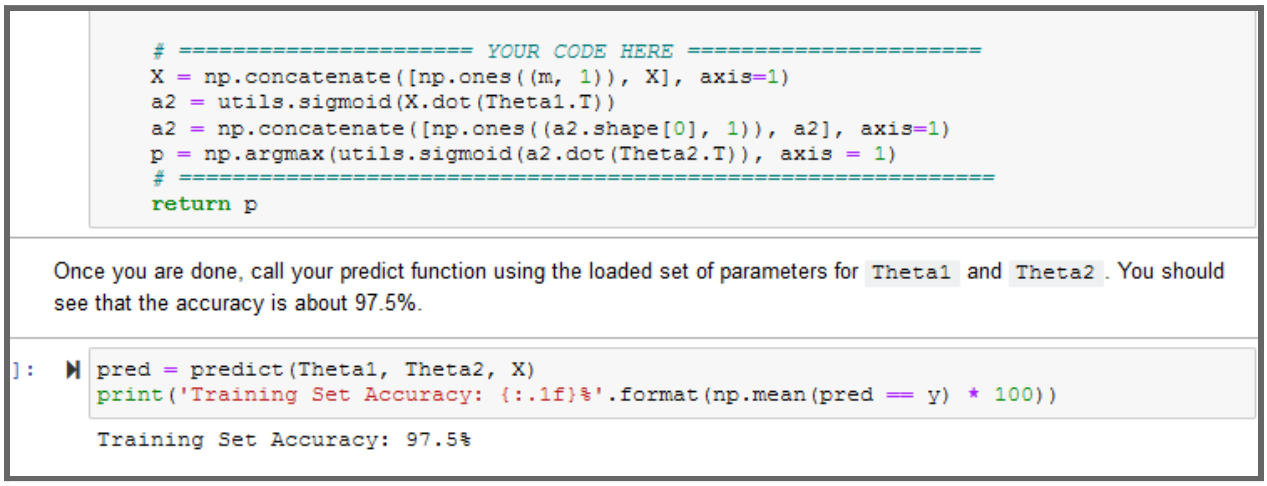
Figura 5.

A Figura 6 demonstra os parametros utilizados para experimento e obtenção de uma acurácia maior.

```
    Example Code
    ------------

        # Set Initial theta
        initial_theta = np.zeros(n + 1)

        # Set options for minimize
        options = {'maxiter': 50}

        # Run minimize to obtain the optimal theta. This function will
        # return a class object where theta is in `res.x` and cost in `res.fun`
        res = optimize.minimize(lrCostFunction,
                                initial_theta,
                                (X, (y == c), lambda_),
                                jac=True,
                                method='TNC',
                                options=options)
    """
    # Some useful variables
    m, n = X.shape

    # You need to return the following variables correctly
    all_theta = np.zeros((num_labels, n + 1))

    # Add ones to the X data matrix
    X = np.concatenate([np.ones((m, 1)), X], axis=1)

    # ====================== YOUR CODE HERE ======================
    for c in np.arange(num_labels):
        initial_theta = np.zeros(n + 1)
#         initial_theta = np.ones(n + 1)
#         initial_theta = np.random.randn(n+1)
        options = {'maxiter': 100}
#         options = {'maxiter': 50}
        res = optimize.minimize(lrCostFunction,
                                initial_theta,
                                (X, (y == c), lambda_),
                                jac=True,
                                method='CG',
#                                 method='TNC',
                                options=options)
        all_theta[c]=res.x

    # ============================================================
    return all_theta
```

Figura 6.

A segunda parte do exercício se da por realizar a predição de uma rede neural com pesos já trreinados. Os pesos são disponibilizados nas variáveis *Theta1* e *Theta2.* Cada camada da rede neural é implementada como indicado na imagem do exercício. A camada interna e a de saída são realizadas com a função sigmoid como indicado por $g(z^2)$ e $g(z^3)$ para as variáveis a^2 e a^3 respectivamente. Os valores de z são obtidos através da multiplicação verotiral dos pesos com os sinais de ativação, dados por *X.dot(Theta1.T)* e *a2.dot(Theta2.T)*. A figura 7 mostra o valor de acurácia obtido com valores padrões dos parâmetros.



```
    # ===================== YOUR CODE HERE =====================
    X = np.concatenate([np.ones((m, 1)), X], axis=1)
    a2 = utils.sigmoid(X.dot(Theta1.T))
    a2 = np.concatenate([np.ones((a2.shape[0], 1)), a2], axis=1)
    p = np.argmax(utils.sigmoid(a2.dot(Theta2.T)), axis = 1)
    # ============================================================
    return p
```

Once you are done, call your predict function using the loaded set of parameters for `Theta1` and `Theta2`. You should see that the accuracy is about 97.5%.

```
]:  pred = predict(Theta1, Theta2, X)
    print('Training Set Accuracy: {:.1f}%'.format(np.mean(pred == y) * 100))
```

```
Training Set Accuracy: 97.5%
```

Figura 7.

O fato de os pesos já serem disponibilizados dificulta a exploração de novas opções para realizarmos experimentos novos. Os pesos estão em formato .mat ainda por cima, o que impossibilita até a edição direta no arquivo (tinha pensado em replicar alguns pesos e aumentar o tamanho da rede no código).