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

Augusto André Souza Berndt 202000735

Exercício 4.

No exercício anterior foi feita a classificação de um problema multi classe com o dataset MNIST. Somente o processo de *feedforward* foi implementado anteriormente, onde se utiliza a rede para avaliar um conjunto de entradas, inclusive os valores de pesos já foram previamente fornecidos no exercício anterior. Nesta próxima parte, no exercício 4, será feita a implementação do treinamento da rede neural, ou seja, o processo de *backpropagation* através do cálculo de gradiente descendente, no qual se aplica derivadas parciais aos pesos e ativações da rede na intenção de direcionar a alteração dos valores de pesos do modelo a uma região mais próxima a uma solução com erro reduzido ou mínimo.

Antes do primeiro desafio, o exercício 4 faz um apanhado do necessário a ser entendido antes de se implementar a função custo e o gradiente descendente. Diferentemente de como nos exercícios anteriores, a hipótese agora é dada pela propagação de toda entrada na rede neural. Onde antes era dada somente com a função sigmoid, uma vez que só tínhamos perceptrons em uma única camada. Agora a rede precisa ser "compactada" na variável *a3* que dá a saída da rede, isso é feito com o processo de *feedfoward* da variável X nas primeiras linhas do primeiro desafio. Logo em seguida é implementado a função custo como indicado no enunciado primeiramente sem a regularização, como consta na Figura 1. Boa parte do código do exercício anterior é utilizado neste desafio.

```
lambda_ = 0
J, _ = nnCostFunction(nn_params, input_layer_size, hidden_layer_size,
                      num_labels, X, y, lambda_)
print('Cost at parameters (loaded from ex4weights): %.6f ' % J)
print('The cost should be about                   : 0.287629.')
```

```
(5000,)
[0 0 0 ... 9 9 9]
[[1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]]
(5000, 10)
(25, 400)
m 5000
reg_term 0.0
Cost at parameters (loaded from ex4weights): 0.287629
The cost should be about                   : 0.287629.
```

Figura 1.

Logo em seguida é implementada a mesma função custo, mas com a variável de regularização. Nesta situação se deve ignorar o valor de bias, por isso se faz *theta1[:,1:]* e *theta2[:,1:]* de

maneira a se ignorar os bias, selecionando todas linhas e todas colunas a partir da segunda coluna (na primeira coluna se armazena os valores de bias), os valores obtidos são os mesmos esperados como enunciado, como expõe a Figura 2, espera-se um theta1 com shape 25x400 e o theta2 com tamanho 10x25. A seta azul na Figura 3 mostra como os *shapes* de ambos thetas batem com o valor esperado, logo a parametrização para qualquer tamanho de theta deve funcionar.

1.4 Regularized cost function ¶

The cost function for neural networks with regularization is given by:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log\left(\left(h_\theta\left(x^{(i)}\right)\right)_k\right) - \left(1-y_k^{(i)}\right)\log\left(1-\left(h_\theta\left(x^{(i)}\right)\right)_k\right)\right] + \frac{\lambda}{2m}\left[\sum_{j=1}^{25}\sum_{k=1}^{400}\left(\Theta_{j,k}^{(1)}\right)^2 + \sum_{j=1}^{10}\sum_{k=1}^{25}\left(\Theta_{j,k}^{(2)}\right)^2\right]$$

Figura 2.

A Figura 3 também apresenta o valor de erro esperado, indicado por uma seta vermelha, para o erro com variável de regularização.

```
# Weight regularization parameter (we set this to 1 here).
lambda_ = 1
J, _ = nnCostFunction(nn_params, input_layer_size, hidden_layer_size,
                      num_labels, X, y, lambda_)

print('Cost at parameters (loaded from ex4weights): %.6f' % J)
print('This value should be about                 : 0.383770.')
```

```
(5000,)
[0 0 0 ... 9 9 9]
[[1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]]
(5000, 10)
theta1 no bias (25, 400)
theta2 no bias (10, 25)
m 5000
regularization 0.09614069392960475
Cost at parameters (loaded from ex4weights): 0.383770
This value should be about                 : 0.383770.
```

Figura 3..

O próximo desafio do exercício se dá por realizar o *backpropagation* da rede neural. Antes de se realizar o backpropagation, primeiro se implementa a função de gradiente descendente sigmoid, no qual se dá pela derivada da função sigmoid, simplesmente. Além disso, antes de se implementar o backpropagation, é realizada a inicialização aleatória dos pesos. O enunciado já dá um código inicial e funcional para inicializar pesos aleatórios, um *print* nos valores mostra que os valores iniciais de pesos são pequenos, como é o esperado e como mostra a Figura 4. Após a implementação do *backpropagation,* irei demonstrar alguns valores com pesos diferentes.

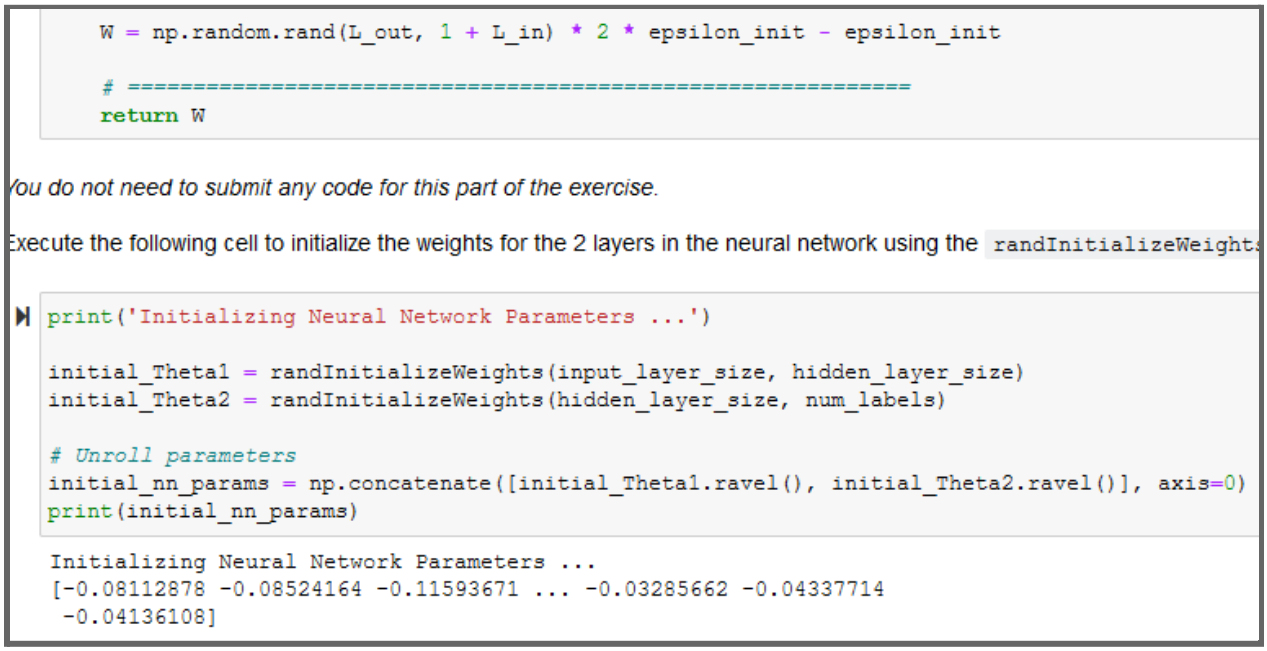

Figura 4.

Agora é implementado o próprio *backpropagation*, no qual é altamente dependente da função erro. A ideia é definir o quanto cada neurônio da rede influência no erro da própria rede ao classificar uma entrada. Primeiramente se implementa a versão sem regularização do *backpropagation*.

O exercício providencia um passo a passo para implementação, composto de 5 passos totais. O processo é feito camada por camada, como o *feedfoward* já foi feito neste caso, o processo se inicia pela camada de saída, comparando os valores de *feedforward* obtidos com os valores esperados em *y.* O delta da última camada é feita de maneira direta se calculando a diferença entre a última camada *a3* com os labels esperados em *one-hot.* A já a diferença da camada interna é feita já se englobando os sinais da camada de entrada e com a utilização da função *sigmoidGradient* anteriormente implementada. Logo em seguida se define os novos valores de pesos para *Theta1_grad* e *Theta2_grad,* estes são concatenados e retornados para saída da função. A Figura 5 demonstra que os valores obtidos batem com os esperados para os thetas após o calculo do gradiente. Já para implementação da regularização do gradiente descendente se remove os valores de bias dos arrays dos thetas e a variável *lambda* é adicionada a equação como indicada pelo enunciado.

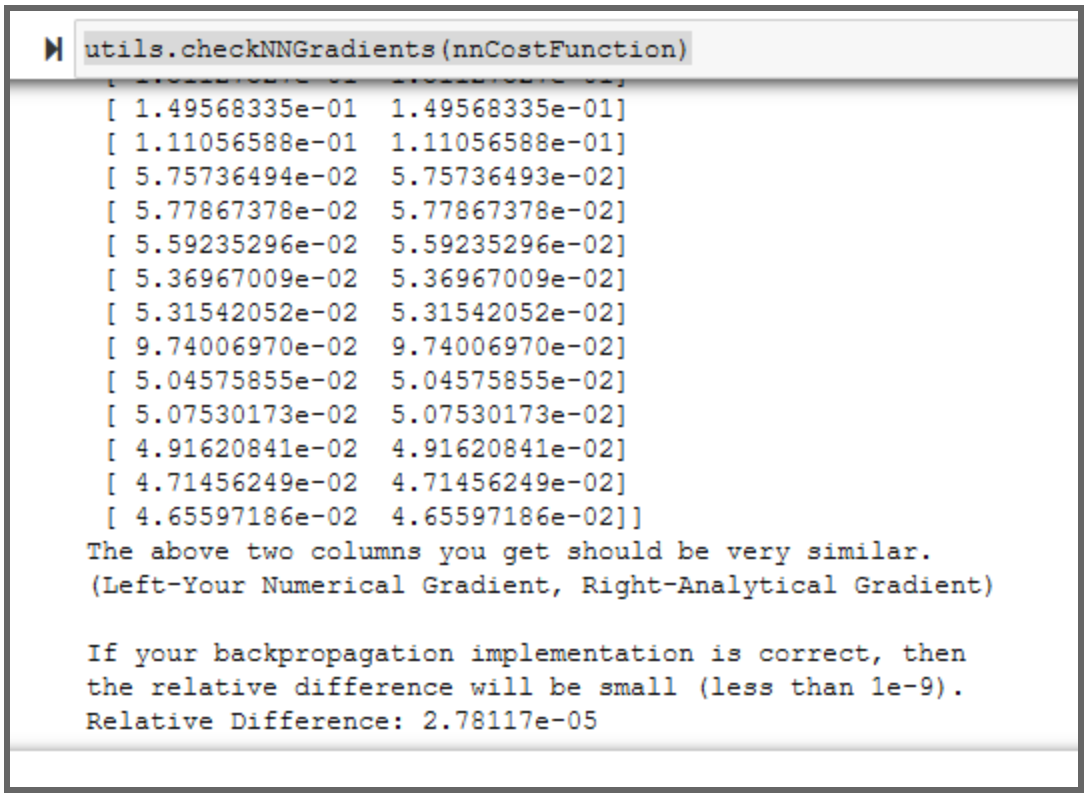

Figura 5.

O próximo passo se dá por minimizar os valores de theta com a função minimize da biblioteca scipy, a fim de realizar o aprendizado da rede. A Figura 6 mostra a acurácia obtida com inicialização aleatória e valores padrões do exercício de treinamento.

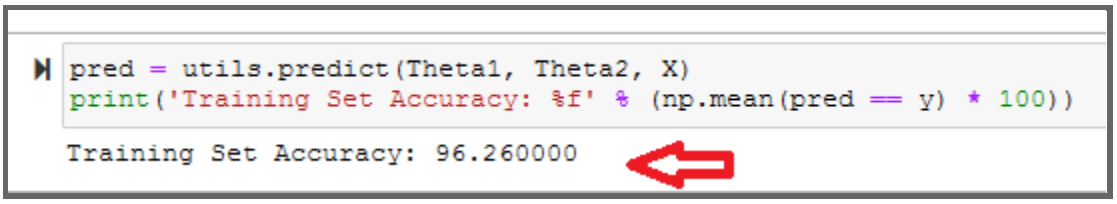

Figura 6.

A Figura 7 mostra um teste feito com a inicialização dos **pesos em 0**. Diferente do que esperava, até se obteve um valor de acurácia, imaginei que algo daria errado.

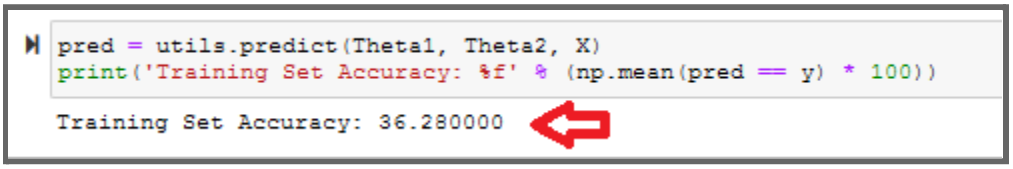

Figura 7.

Também se experimentou para inicialização de **pesos bem maiores** (10 vezes) do que por padrão do exercício. A Figura 8 mostra a saída para esta situação, foi encontrada um divisão por zero ao longo do processo, isso provavelmente se deve ao efeito de *vanishing gradient*. Faz sentido a acurácia estar em 10%, uma vez que se tem 10 possíveis classes, 1 em 10 a chance de acerto, ou seja, total acaso.

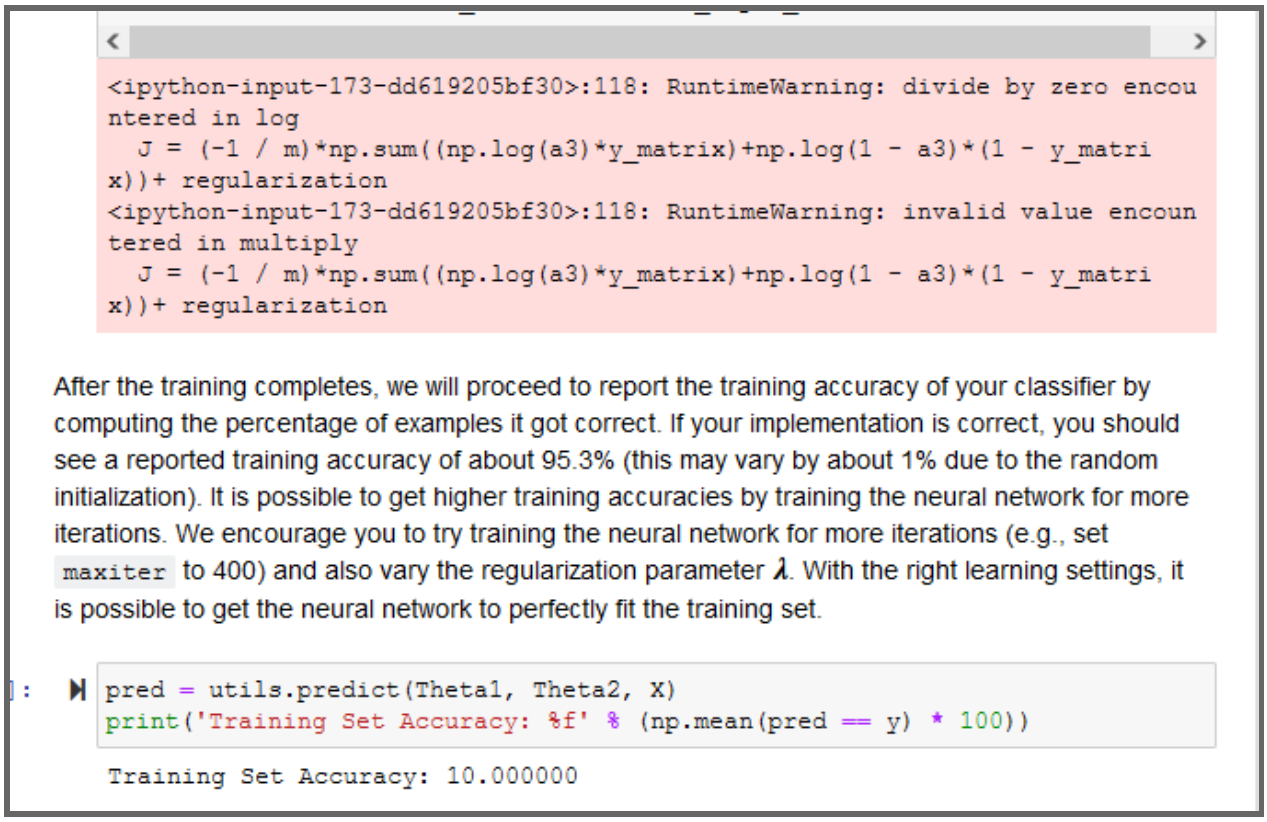

Figura 8.

O exercício ainda mostra o que é “visto” pela camada interna para algumas imagens de entrada desconsiderando o bias. Bem interessante.