Código de Rede Neural com uma camada escondida com backpropagation

Thiago Figueiró Ribeiro e Gustavo Ramos Abreu

***Abstract*— Códigos de redes neurais podem ser implementados em qualquer linguagem de programação. Nesse trabalho, mostramos uma implementação em Python de uma rede neural de uma camada com backpropagation. A arquitetura com a maior acurácia para classificação usando dados de mamografia foi uma de 5 neurônios na camada escondida com acurácia de 81.293%**

Keywords*— *Redes Neurais Artificiais, Classificação, Mamografia

I. Introdução

Apesar de clássica, a implementação de uma rede neural ainda tem certas armadilhas.

Na seção II iremos descrever a principal sequencia de eventos que acontecem no código proposto de rede neural com uma única camada escondida de quantidade variável de neurônios. Na seção III mostraremos alguns resultados de treinamento usando esse código. Na seção IV são mostradas as conclusões desse trabalho.

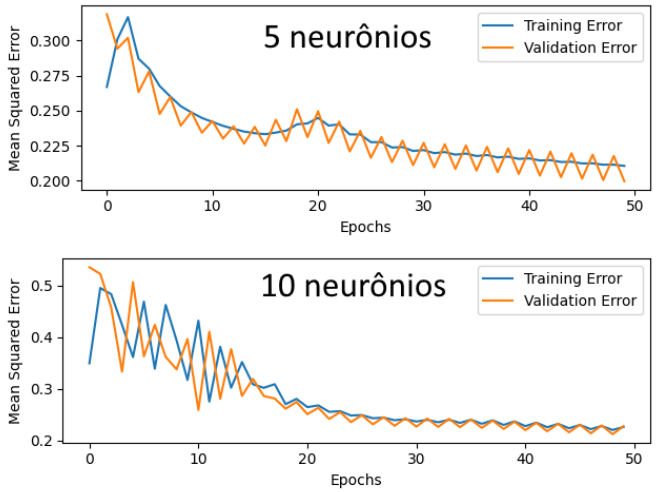

Fig. 1. Erro quadrático médio com o passar de épocas para as redes usando 5 e 10 neurônios.

II. Etapas do treino de uma Rede Neural Artificial usando a ferramenta train do Matlab

Nesse trabalho será mostrado o treino da nossa rede neural da sua camada mais exterior, tal como a função de treino da rede, para a mais interior, tal como as funções de criação da rede, de feed forward e o backpropagation, além das funções de ativação.

Nesse primeiro momento, iremos mostrar a função treinar(). Em primeiro lugar, os dados são armazenados pelo Python (1), então normalizados com base nos dados de maior e menores valores (2), separados em dados de treino e validação de forma explícita e os de teste de forma implícita

```
    # Carregando os dados do Excel
    data = pd.read_excel("dadosmamografia.xlsx")

    # Normalizando os dados de input
    X = normalize_data(data.iloc[:, :-1].values)

    # Dividindo os dados entre validação, treino e
teste
    train_size = int(0.6 * len(X))
    val_size = int(0.2 * len(X))
```

Criamos então a nossa rede com a quantidade de inputs (nesse caso 5), quantidade de neurônios na camada escondida (que pode ser um valor arbitrário), o output (1), a taxa de aprendizado, a quantidade de épocas que o treino deve ter caso ele não atenda ao critério de parada e finalmente o número de épocas que o erro deve crescer para o treino parar.

```
        # Criando os parâmetros da rede
        input_size = 5
        hidden_size = 5
        output_size = 1
        learning_rate = 0.1
        epochs = 50
        early_stop_threshold = 5

        # Criando uma instância da rede com base nos
dados acima
        nn = NeuralNetwork(input_size, hidden_size,
output_size, learning_rate)

        # Treinando a rede
        nn.train(X_train, y_train, X_val, y_val,
epochs, early_stop_threshold)
```

TABELA I. *Acurácia da rede neural sobre os dados de teste em função da quantidade de neurônios na camada escondida e tentativa.*

Acurácia		Quantidade de Neurônios			
		5	10	15	30
Tentativa	1	**79,348%**	79,477%	79,927%	79,885%
	2	75,623%	79,935%	80,132%	79,872%
	3	78,733%	79,871%	79,669%	79,675%
	4	81,293%	79,733%	80,041%	79,826%
	5	79,513%	79,584%	79,829%	79,776%

Após isso são mostrados os pesos antes do treino e os pesos após o treino, além de calculado o erro entre os valores previstos para os parâmetros de teste e os valores corretos dos testes. Esses valores são então mostrados na tela.

```
        # Comparando os resultados da rede treinada
com os dados de teste
        test_output = nn.predict(X_test)
```

Thiago Figueiró Ribeiro, PPGEE, Universidade Federal do Pará, Belém-PA, e-mail: thiago.ribeiro@ig.ufpa.br;

Gustavo Ramos Abreu, e-mail: gustavoramos2596@gmail.com

```
        test_error = np.mean(np.square(y_test - test_output))

        print("Test Error:", test_error)
```

O código acima mostra de forma superficial como o framework deve gerar resultados. Iremos mostrar agora de forma mais profunda os elementos da rede. A classe **NeuralNetwork**, que representa uma rede neural de uma camada escondida, tem seus parâmetros definidos (tal como tamanho do input, tamanho da hidden layer, tamanho do output e taxa de aprendizado) e seus pesos inicializados. Nesse momento são gravados os valores dos pesos para serem posteriormente comparados.

```
    # Inicializando os pesos
    self.weights_input_hidden = np.random.uniform(-1, 1, size=(self.input_size, self.hidden_size))

    # Salvando os pesos iniciais
    self.weights_input_hidden_before = copy.copy(self.weights_input_hidden)
```

A função **Forward**, que representa a etapa de Feed Forward da rede é descrita a seguir. Nela os dados de entrada são multiplicados pelos pesos naquele momento. Então, o resultado disso (equivalente à camada escondida) é multiplicada pela sua função de ativação naquele ponto. Por fim, o output é retornado como sendo a função de ativação multiplicada pelo resultado de e os pesos da camada escondida.

```
    self.hidden_input=np.dot(X_train,self.weights_input_hidden)

    self.hidden_output = sigmoid(self.hidden_input)

    self.output = sigmoid(np.dot(self.hidden_output, self.weights_hidden_output))
```

Finalmente, a etapa de **backpropagation**. Nela, o erro é definido como o resultado esperado menos o resultado do treino naquele momento. O gradiente de erro é a derivada da função de ativação multiplicada pelo erro.

```
    error = y_train - output
    output_delta = error * sigmoid_derivative(output)
```

Da mesma forma, o erro na camada escondida é esse gradiente multiplicado por cada peso dessa camada. E assim o gradiente de erro da camada escondida vai ser o erro de cada neurônio multiplicado pela derivada da função de ativação daquela camada.

```
hidden_error=
output_delta.dot(self.weights_hidden_output.T)

hidden_delta=hidden_error*sigmoid_derivative(self.hidden_output)
```

Por fim, os pesos da camada oculta e camada de saída são atualizados com base no sinal de erro na camada de saída e no erro da camada oculta.

```
self.weights_hidden_output+=self.hidden_output.T.dot(output_delta) * self.learning_rate
self.weights_input_hidden += X_train.T.dot(hidden_delta)* self.learning_rate
```

A função **Train** descreve a etapa de treino da rede. Após a inicialização dos vetores que irão receber os resultados de erro de treino, erro de validação, a rede é treinada por um número definido de épocas. É chamado o método de feed forward, então o de backpropagation.

```
    # Feed forward
     output = self.forward(X_train)

    # Backward propagation
    self.backward(X_train, y_train, output)
```

Então, é calculado o valor do erro médio quadrático para fins de teste de parada do algoritmo. Esse cálculo é feito tanto nos valores de treino quanto nos valores de validação. Esses valores são gravados nos vetores que foram anteriormente inicializados.

```
    # Cálculo do erro médio quadrático do treino
train_error = np.mean(np.square(y_train - output))

train_errors.append(train_error)

    # Cálculo do erro médio quadrático da validação
val_output = self.forward(X_val)

val_error = np.mean(np.square(y_val - val_output))

val_errors.append(val_error)
```

Caso o erro médio quadrático na validação aumente por 5 épocas, o treino deve ser interrompido.

```
  if epoch > 0 and val_errors[-1] > val_errors[-2]:
     consecutive_increases += 1
  else:
     consecutive_increases = 0

  if consecutive_increases >= early_stop_threshold:
     break
```

Após o treino ser interrompido, é criado um gráfico mostrando os erros de treino e de validação.

```
Pesos antes:
[[-0.60709095  0.41773991  0.94181655 -0.30544841  0.48280062]
 [-0.19009926  0.29667248  0.66150385  0.59749819  0.19791162]
 [-0.21721272 -0.85615728 -0.33444624 -0.73611222  0.65319398]
 [ 0.26681389 -0.20119676 -0.68792548  0.25161102 -0.79712667]
 [-0.09306751  0.16323631  0.1269861  -0.39170511 -0.47664175]]
[[-0.47126846]
 [-0.7597565 ]
 [-0.09030283]
 [ 0.57817694]
 [-0.3778149 ]]
Pesos depois:
[[-3.4505866  -2.75869841 -1.33600559  3.8390255  -2.20347637]
 [-3.01458239 -2.81586675 -3.38482737 -1.41310775 -3.16616292]
 [-4.98106319 -6.22927767 -3.80529582  7.05507166 -3.74384296]
 [-6.08377775 -7.37304367 -5.24782505 11.04642301 -6.62680625]
 [-0.07715622  0.19049613 -0.05561642 -0.98204282 -0.52911668]]
[[-6.97382004]
 [-7.61368681]
 [-5.3208157 ]
 [ 4.06717489]
 [-6.43712747]]
```

Fig. 2. Pesos antes e depois do treinamento

III. RESULTADOS

Ao treinar a rede com 5 e 10 neurônios na camada escondida, chegou-se nos gráficos da Fig. 1. Como o critério de parada foi que o valor do erro de validação deve aumentar por 5 épocas consecutivas, isso aconteceu muito raramente. Isso acontece em função desse erro aumentar em uma época e diminuir na próxima, o que anulava o critério de parada do algoritmo.

Os pesos são atualizados como mostrado na Fig. 2. Após cada término de treino, os dados de teste são comparados com os resultados previstos pelo algoritmo já treinado. Nesse caso, a arquitetura que melhor gerou resultado foi uma de 5 neurônios na camada escondida com um erro de ~0.18707 ou ~81.293% de acurácia, tal como mostrado na Tabela I.

O código usado para a aquisição dos dados acima está no link a seguir:

https://github.com/ThiagoFigueiroRibeiro/Trabalho-Backpropagation