UNIVERSIDADE FEDERAL DE MINAS GERAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Exercício Computacional 5: Backpropagation

Aluno: *Pedro Vinícius A. B. de Venâncio*

Disciplina: *Redes Neurais Artificiais (EEE950)* – Professor: *Antônio de Pádua Braga*
Data: *Setembro de 2019*

Introdução

Este relatório tem como objetivo desenvolver[1] a rede neural artificial do tipo *perceptron* de múltiplas camadas, ou *Multilayer Perceptron (MLP)*, cujo algoritmo de treinamento mais usual é conhecido como *backpropagation*. Tais redes apresentam, no mínimo, uma camada intermediária, o que permite a aproximação de qualquer função contínua, bem como a solução de problemas não linearmente separáveis.

Multilayer Perceptrons (MLPs)

Os *Multilayer Perceptrons* são redes neurais similares ao *perceptron* simples, no entanto, como o próprio nome já diz, sua arquitetura contém uma ou mais camadas intermediárias. Em geral, o aprendizado dos pesos durante o treinamento é feito por um método baseado em otimização por gradiente descendente. Tal algoritmo, denominado *backpropagation*, utiliza um mecanismo de correção de erros composto por duas fases: propagação direta (*forward pass*) e propagação reversa (*backward pass*) (Braga, 2000). A primeira fase, presente também nas demais arquiteturas discutidas anteriormente, consiste basicamente em predizer a saída $\hat{\boldsymbol{Y}}$ dado um vetor de entradas $\boldsymbol{X}$. Já a segunda fase, oriunda especificadamente do *backpropagation*, tem como objetivo utilizar o erro entre a saída obtida $\hat{\boldsymbol{Y}}$ e a saída desejada $\boldsymbol{Y}$ para ajustar os pesos dos neurônios.

Considere agora uma rede neural com uma única camada intermediária, onde as funções de ativação na camada intermediária e de saída são, respectivamente, $f(.)$ e $h(.)$. Dado um conjunto de pesos $\boldsymbol{W_h}$ para a primeira camada e um conjunto de pesos $\boldsymbol{W_o}$ para a segunda camada, a saída da rede pode ser obtida por um processamento *forward pass* (1):

$$\hat{\boldsymbol{Y}} = h(f(\boldsymbol{X}\boldsymbol{W_h})\boldsymbol{W_o}) \tag{1}$$

Para que o treinamento com retropropagação de erros seja possível, as funções de ativação devem ser contínuas, diferenciáveis e, preferencialmente, não decrescentes. Isso torna possível o cálculo da derivada parcial do erro $\mathcal{L}(.)$ em relação a um determinado peso $\boldsymbol{W}$, ou seja, o quão o peso $\boldsymbol{W}$ influencia no erro. A relação do peso da camada de saída com o erro pode ser obtida pela regra da cadeia (2):

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W_o}} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{Y}}} \frac{\partial \hat{\boldsymbol{Y}}}{\partial f(\boldsymbol{X}\boldsymbol{W_h})\boldsymbol{W_o}} \frac{\partial f(\boldsymbol{X}\boldsymbol{W_h})\boldsymbol{W_o}}{\partial \boldsymbol{W_o}} \tag{2}$$

[1] Todas as implementações foram feitas em linguagem R.

Tendo em vista que a retropropagação do erro é acumulativa, a influencia dos pesos das camadas iniciais no erro dependem de uma quantidade maior de derivadas parciais, o que implica em uma regra da cadeia mais complexa para obter a relação entre os pesos da camada intermediária e o erro (3):

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W_h}} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{Y}}} \frac{\partial \hat{\boldsymbol{Y}}}{\partial f(\boldsymbol{X}\boldsymbol{W_h})\boldsymbol{W_o}} \frac{\partial f(\boldsymbol{X}\boldsymbol{W_h})\boldsymbol{W_o}}{\partial f(\boldsymbol{X}\boldsymbol{W_h})} \frac{\partial f(\boldsymbol{X}\boldsymbol{W_h})}{\partial \boldsymbol{X}\boldsymbol{W_h}} \frac{\partial \boldsymbol{X}\boldsymbol{W_h}}{\partial \boldsymbol{W_h}} \tag{3}$$

Assim, tem-se que os ajustes dos pesos $\boldsymbol{W_o}$ e $\boldsymbol{W_h}$ são (4) e (5), respectivamente:

$$\boldsymbol{W_o} = \boldsymbol{W_o} + \alpha \frac{\partial \mathcal{L}}{\partial \boldsymbol{W_o}} \tag{4}$$

$$\boldsymbol{W_h} = \boldsymbol{W_h} + \alpha \frac{\partial \mathcal{L}}{\partial \boldsymbol{W_h}} \tag{5}$$

onde α é uma taxa de aprendizado suficientemente pequena para minimizar o erro $\mathcal{L}(.)$.

Problema - Senoide A fim de verificar que um *Multilayer Perceptron* com uma única camada intermediária é capaz de aproximar qualquer função contínua, deseja-se realizar a regressão de um ciclo de uma senoide. Os dados de treinamento e teste foram obtidos de uma senoide com ruído uniforme, conforme apresentado na Figura 1.

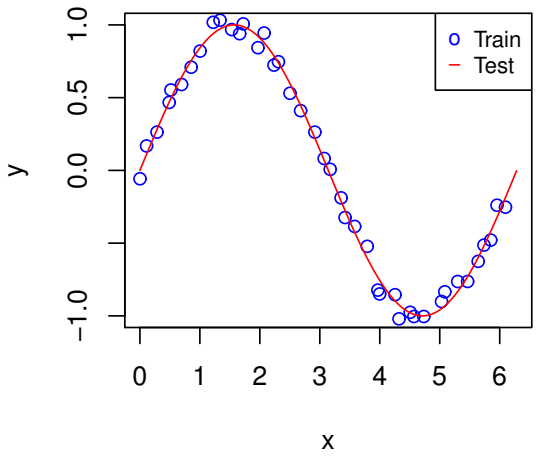

Figura 1: Dados de treinamento e teste.

A arquitetura considerada apresenta 3 neurônios na camada escondida e apenas um neurônio na camada de saída. A função de erro escolhida foi o *Mean Squared Error (MSE)* e as funções de ativação para a camada intermediária e de saída foram, respectivamente, sigmóide e linear. Tais escolhas das funções de ativação permitiram simplificar as Equações (2) e (3) em (6) e (7):

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W_o}} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{Y}}} f(\boldsymbol{X}\boldsymbol{W_h}) \tag{6}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_h} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{Y}}} \boldsymbol{W}_o \frac{\partial f(\boldsymbol{X}\boldsymbol{W}_h)}{\partial \boldsymbol{X}\boldsymbol{W}_h} \boldsymbol{X} \tag{7}$$

As etapas *forward pass*, *backward pass* e atualização de pesos são realizadas iterativamente até que um número de épocas seja atingido ou o erro obtido fique inferior a uma tolerância.

```
# Forward pass
A <- sigmoid(X %*% W_hidden)
Y_hat <- A %*% W_output

# Backward pass
dW_output <- t(A) %*% (loss_derivative(Y, Y_hat) * 1)
dW_hidden <- t(X) %*% (((loss_derivative(Y, Y_hat) * 1)
             %*% t(W_output))*sigmoid_derivative(A))

# Update weights
W_output <- W_output + alpha*dW_output
W_hidden <- W_hidden + alpha*dW_hidden
```

O aprendizado dos pesos ocorreu durante 45000 épocas e com uma taxa de aprendizado de $\alpha = 0.01$. Conforme a Figura 2, percebe-se que o erro no treinamento durante uma execução decresceu de forma consistente e rápida até atingir um erro de $MSE_{treino} = 0.1431$, constatando uma boa escolha dos hiperparâmetros.

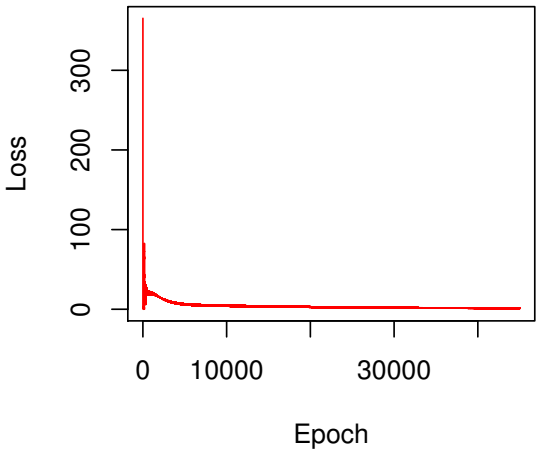

Figura 2: Erro durante o treinamento para uma única execução.

Entretanto, como o treinamento de uma rede neural trata-se de um processo não-determinístico, conclusões obtidas com apenas uma execução podem não ser muito confiáveis. Além do mais, deve-se analisar também o erro no conjunto de teste, uma vez que o modelo pode apresentar baixo erro no treinamento e ainda sim não generalizar para novos dados (*overfitting*). Diante disso, foram realizadas 5 execuções independentes para estimar a média do *Mean Squared Percentage Error (MSPE)* e o seu respectivo desvio padrão no conjunto de teste (Tabela 1).

Execução	1	2	3	4	5	Média	Desvio
MSPE	5.07	6.32	3.66	6.71	4.97	5.35	1.21

Tabela 1: Estatísticas de teste em cinco execuções.

Através dos resultados é possível afirmar que o *perceptron* de múltiplas camadas proposto obteve erros médios percentuais baixos em todas as execuções, apresentando, inclusive, certa robustez ao manter o desvio padrão absoluto substancialmente baixo. Por fim, pode-se observar pela sobreposição das curvas que a aproximação ficou bem fidedigna ao ciclo da senoide no intervalo $(0, 2\pi)$ (Figura 3).

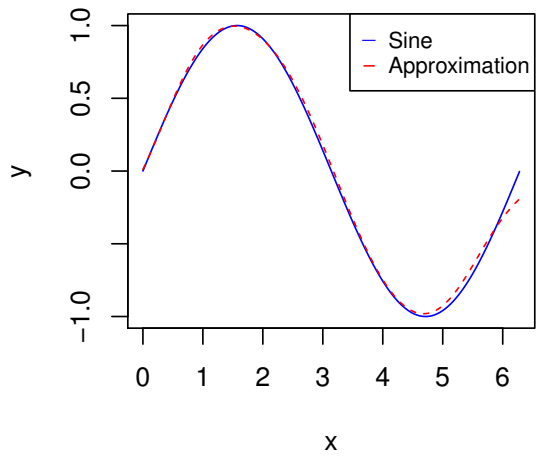

Figura 3: Aproximação obtida na terceira execução.

Conclusão

As *Multilayer Perceptrons* apresentam um poder computacional muito superior ao das redes sem camadas intermediárias. Devido a simples adição de uma camada intermediária com função de ativação não linear em sua topologia, tais redes se tornam capazes de classificar dados não linearmente separáveis. No entanto, o que mais revolucionou as pesquisas em redes neurais artificiais não foi a arquitetura de múltiplas camadas, mas sim o algoritmo iterativo responsável pelo aprendizado dos pesos: o *backpropagation*. Sem tal abordagem, possivelmente não se existiria o que é conhecido hoje como *deep learning*.

Referências

Braga, A. Redes neurais artificiais: teoria e aplicações. Livros Técnicos e Científicos, 2000.