Métodos Computacionais em Física - 4300331 - 1s/ 2020

Prof. Dr. Luis Gregório Dias da Silva - FMT - IFUSP

Projeto 5 - Redes Neurais SUB (entrega em 02/08/2020)

Prazo de entrega: 23h55 de **02/08/2020** (Domingo) através do site da disciplina.
NÃO SERÃO ACEITOS projetos enviados após esta data ou por e-mail!!

1 Orientação geral:

O projeto é individual e tem como objetivo o estudo computacional de o seguinte tema:

Simulação de uma porta XOR com uma rede neural de camada oculta.

1.1 Arquivos:

Você deverá fazer o upload online dos arquivos de MatLab/Octave ("NomeDoAluno_Projeto5.m" (caso haja vários arquivos, enviar um arquivo zip "NomeDoAluno_Projeto5.zip") prontos para serem executados (caso seja feita a opção por outra linguagem, fazer o upload do código-fonte e instruções de compilação) e de um Relatório escrito ("NomeDoAluno_Projeto5_Relatorio.pdf") contendo:

- Introdução

  Introduza o problema a ser estudado. Escreva como se você estivesse explicando a um colega ou um outro professor. Lembre-se: o relatório do Projeto é um documento pensado para uma plateia mais ampla!

- Descrição da simulação numérica e dos resultados obtidos. Descreva em detalhe o método numérico que você utilizou e sua escolha de parâmetros. Exemplo: O passo está adequado? Que testes você fez?

  Use quantos gráficos você quiser para ilustrar suas conclusões. Não se limite aos tipos de gráficos usados nas tarefas. Quanto mais, melhor! Nos gráficos, serão avaliados aspectos como legenda, labels nos eixos, clareza na apresentação dos dados (símbolos, linhas, etc).

- "Manual do usuário" do seu script: Se um colega for rodar seu código, como ele deve proceder? Quais as variáveis importantes? Quais os parâmetros podem ser modificados?

- Conclusão Que tipo de informação sobre o sistema físico a simulação numérica trouxe? Lembre-se: muita gente pode fazer programas mas poucos sabem interpretar o resultado!

- As legendas e labels tem que ser FACILMENTE legíveis. Use a opção 'FontSize' para aumentar o tamanho da fonte.

2 Descrição do tema proposto

Durante o curso, vimos como usar redes neurais em problemas de *classificação binária*. Estudamos o exemplo de simulação de portas lógicas AND e OR com redes de apenas um neurônio (reveja essa aula!). Na aula, foram apresentados os conceitos básicos de uma rede neural de classificação binária como a função custo do tipo log-loss e também a função de ativação do tipo sigmóide, que serão utilizadas neste projeto.

Uma outra porta lógica que pode ser usada para classificação binária é a *porta XOR*, descrita pela tabela a seguir:

s	$X_1(s)$	$X_2(s)$	$y(s)$
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

Tabela 1: Tabela da porta XOR

Foi mencionado na aula que, ao contrário das portas OR e AND, a porta lógica XOR não poderia ser simulada de forma eficiente em uma rede com apenas um neurônio. O objetivo deste projeto *entender o porquê disso* e simular a porta XOR utilizar uma rede neural com uma camada oculta (como a mostrada na figura abaixo) para fazer sua simulação.

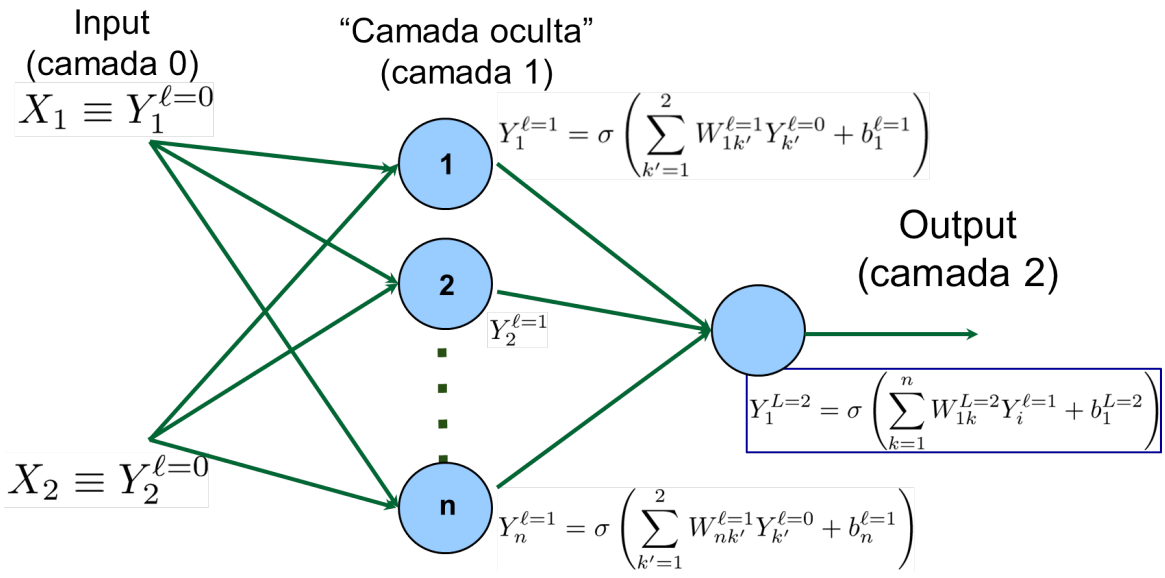

Figura 1: Rede neural com uma camada de entrada, uma camada oculta e a camada de saída.

2.1 Notação a ser usada

Com a camada extra, temos mais pesos W (um para cada linha) e bias b (um para cada nerônio) que definem a rede neural e que devem ser convergidos no processo de aprendizagem. Vamos padronizar a notação da seguinte forma:

- $W^{\ell}_{kk'}$: peso da linha que sai do neurônio k' da camada $\ell - 1$ para o neurônio k da camada ℓ.
- b^{ℓ}_{k}: bias do neurônio k na camada ℓ.
- $Y^{\ell}_{k} = \sigma(Z^{\ell}_{k})$: output do neurônio k na camada ℓ com função de ativação do tipo sigmoide: $\sigma(Z) = (1 + e^{-Z})^{-1}$.
- $Z^{\ell}_{k} \equiv \left(\sum_{k' \in \ell - 1} W^{\ell}_{kk'} Y^{\ell-1}_{k'}\right) + b^{\ell}_{k}$: argumento da sigmoide que define o valor de Y^{ℓ}_{k}. Note que envolve o output da camada anterior.
- $(X_1, X_2) \equiv (Y^0_1, Y^0_2)$: os dados de entrada são considerados como a "saida da camada zero".
- $Y_{\text{out}} \equiv Y^L_1$: o dado de saída é considerado o output do neurônio da última camada, denominada de L. No caso em que temos apenas uma camada oculta (como nesse projeto), $L = 2$.

2.2 Função-custo

Usando o conjunto de aprendizado $[(X_1(s), X_2(s)) \to y(s)]$ dado na Tabela 1, podemos definir uma função custo do tipo log-loss:

$$\mathcal{C}(\mathbf{W}, \mathbf{b}) = \frac{1}{N_s} \sum_{s=1}^{N_s} \left[-y(s) \ln\left(Y_{\text{out}}(s)\right) - (1 - y(s)) \ln\left(1 - Y_{\text{out}}(s)\right)\right] \equiv \frac{1}{N_s} \sum_{s=1}^{N_s} C_s(\mathbf{W}, \mathbf{b}) \tag{1}$$

onde toda a depência nos pesos $W^{\ell}_{kk'}$ e bias b^{ℓ}_{k} está contida nos termos $C_s(\mathbf{W}, \mathbf{b})$ que são calculados para cada um dos $s = 1, 2, 3, 4$ inputs da tabela.

O primeiro objetivo do projeto é construir um algoritmo para encontrar o conjunto de $W^{\ell}_{kk'}$ e b^{ℓ}_{k} que *minimizem* os $C_s(\mathbf{W}, \mathbf{b})$ e, consequentemente, a função-custo $\mathcal{C}(\mathbf{W}, \mathbf{b})$.

2.3 Backpropagation

Para isso, usaremos o conceito de *backpropagation*. Como os $C_s(\mathbf{W}, \mathbf{b})$ são funções dos pesos e bias da rede inteira, para cada s, definimos a derivada parcial de C_s em relação ao output Y^{ℓ}_{k} do neurônio k da camada ℓ. Melhor ainda, definimos:

$$\delta^{\ell}_{k} \equiv \frac{\partial C_s}{\partial Z^{\ell}_{k}} \tag{2}$$

Em particular, pode-se mostrar (faça isso no seu projeto) que

$$\delta^{\ell}_{k} = \frac{\partial C_s}{\partial Y^{\ell}_{k}} \sigma'(Z^{\ell}_{k}) \tag{3}$$

onde $\sigma'(Z) = (1 - \sigma(Z))\sigma(Z)$ é a derivada da sigmoide. Em particular, é possível mostrar que (faça!), no caso do output da camada de saída $\ell = L$, o resultado é bem simples:

$$\delta^{L}_{1} = Y_{\text{out}}(s) - y(s) \tag{4}$$

A ideia de bactkpropagation é, partindo de δ^{L}_{1}, pode-se calcular todos os δ^{ℓ}_{k} e, a partir desses, utilizar o método do gradiente descendente para minimizar $C_s(\mathbf{W}, \mathbf{b})$ de forma iterativa.

3 Pontos a abordar no Relatório

Este projeto será avaliado como um **projeto de pesquisa no tema do redes neurais e back-propagation**. Os pontos abaixo são uma sugestão de roteiro daquilo minimamente esperado do projeto mas o trabalho não necessariamente se limita a eles. Perguntas e explorações propostas são encorajadas.

3.1 Derivação das equações fundamentais de backpropagation

Apresente a notação e **derive** as equações (3) e (4) acima. Além dessas, **derive** as seguintes equações:

$$\delta_k^\ell = \left(\sum_{k' \in \ell+1} W_{k'k}^{\ell+1} \delta_{k'}^{\ell+1} \right) \sigma'(Z_k^\ell) \tag{5}$$

$$\frac{\partial C_s}{\partial b_k^\ell} = \delta_k^\ell \tag{6}$$

$$\frac{\partial C_s}{\partial W_{kk'}^\ell} = \delta_k^\ell Y_{k'}^{\ell-1} \tag{7}$$

3.2 Construção do algoritmo

A partir das Eqs. (3), (4) e (5-7), estabeleça um algoritmo para calcular as variações $\Delta W_{kk'}^\ell$ e Δb_k^ℓ dados por

$$\Delta W_{kk'}^\ell = \frac{-\eta}{N_s} \sum_{s=1}^{N_s} \frac{\partial C_s}{\partial W_{kk'}^\ell} \tag{8}$$

$$\Delta b_k^\ell = \frac{-\eta}{N_s} \sum_{s=1}^{N_s} \frac{\partial C_s}{\partial b_k^\ell} \tag{9}$$

A cada passo, o algoritmo deverá ter uma fase de "forward propagation" (para o cálculo de δ_1^L) e outra de "backpropagation" (para calcular os demais δ_k^ℓ). Ao final do passo, os pesos e bias serão atualizados na forma:

$$W_{kk'}^\ell \to W_{kk'}^\ell + \Delta W_{kk'}^\ell \tag{10}$$

$$b_k^\ell \to b_k^\ell + \Delta b_k^\ell \tag{11}$$

3.3 Implementação, resultados e discussão

Implemente o algoritmo em um programa e elabore um critério de convergência para a minimização da função-custo. O número n de neurônios da camada escondida deve ser um parâmetro que pode ser variado pelo usuário. Não esqueça de incluir um "Manual do Usuário" com instruções de como rodar o código!

Faça testes com: o "chute inicial" das matrizes $W_{kk'}^\ell$ e b_k^ℓ, o valor do hiperparâmetro η e o número máximo de iterações. Por exemplo, para $n = 4$ e $\eta = 2$, e com o "chute" inicial a seguir

$$W^{\ell=1} = \begin{pmatrix} 0.4 & 0.3 \\ 0.6 & 0.8 \\ 0.2 & 0.4 \\ 0.1 & 0.5 \end{pmatrix}, \; b^{\ell=1} = \begin{pmatrix} 0.3 \\ 0.5 \\ 0.2 \\ 0.1 \end{pmatrix},$$

$$W^{\ell=2} = (0.4 \;\; 0.3 \;\; 0.1 \;\; 0.2)\,, \; b^{\ell=2} = 0.3\,,$$

o algoritmo deveria estar razoavelmente convergido em cerca de 5000 iterações. Discuta outros casos.

Uma vez "treinada" a rede neural (após a convergência), faça um contour plot de pontos $(X_1, X_2) \to Y$ com X_i variando entre 0 e 1. Discuta o resultado e compare com os casos da portas AND e OR discutidos em sala.

Com base no gráfico, responda: por quê a porta lógica XOR não pode ser simulada de forma eficiente em uma rede com apenas um neurônio?