Instituto Tecnológico de Aeronáutica

CT-213: Inteligência Artificial para Robótica Móvel

Lab 6: Redes Neurais

Bruno Benjamim Bertucci - Turma 23.2

1 Introdução

A área de aprendizado de máquina consiste na busca de algorítmos que se ajustam automaticamente para executar alguma tarefa através de treinamento. Existem vários paradigmas de aprendizado de máquina, sendo um deles o aprendizado supervisionado, no qual o treinamento se dá pelo fornecimento de dados ao algorítmo, sendo seu desempenho avaliado pela comparação entre os resultados gerados por ele a partir desses dados e os resultados ideais esperados. A partir dos erros cometidos pelo algorítmo nessa comparação, ele se ajusta para tentar diminuir esse erro na próxima iteração de treinamento.

2 Redes Neurais

Um dos algorítmos de aprendizado de máquina mais populares atualmente é a chamada rede neural. O elemento fundamental de uma rede neural é um neurônio artificial, cujo princípio de funcionamento é: receber uma dada quantidade k de valores de entrada, chamadas de *inputs*, realizar uma operação linear usando esses valores (Equação 1), e finalmente aplicar uma função, denominada função de ativação, em z (Equação 2).

O valor de a é o resulado final, ou *output*, do neurônio. Observa-se, então, que para um neurônio, existem $k+1$ parâmetros, levando em conta os pesos $w_1, ..., w_k$ e o *bias* b do neurônio. Uma função de ativação comum é o sigmoide (Equação 3).

$$z = \sum_{i=1}^{k} (w_i x_i) + b \tag{1}$$

$$a = g(z) \tag{2}$$

$$\sigma(z) = \frac{1}{1 + e^{-x}} \tag{3}$$

2.1 Implementação da Rede Neural

Uma rede neural consiste de múltiplas camadas, cada qual com um determinado número desses neurônios artificiais. As camadas são ligadas entre si, de tal forma que os *outputs* de uma camada são os *inputs* da próxima. A configuração das camadas é chamada de arquitetura da rede neural. Existem diferentes formas de ligar as camadas, sendo uma delas ligar cada neurônio de uma camada com cada neurônio da camada seguinte. Sendo assim, é criado um parâmetro de peso para cada ligação entre dois neurônios, e um parâmetro *bias* para cada neurônio.

Sendo assim, cada peso é denotado por w^l_{jk}, onde j é o j-ésimo neurônio da camada l e k é o k-ésimo neurônio da camada $l-1$, e cada *bias* é denotado por b^l_j. A numeração comum para l parte do zero nos *inputs*, fazendo com que a camada 1 represente a primeira camada de neurônios, e denota a última camada como L. O *output* da última camada é o resultado final da rede neural.

O conjunto de operações realizadas desde a alimentação dos valores de entrada até o fornecimento do resultado final da rede neural é denominado *forward propagation*. Essa operação é

dada pelas Equações 4 e 5, onde a^l e b^l são os vetores de elementos a^l_j e b^l_j, respectivamente, para a camada l, e W^l é a matriz dos pesos para a camada l, com dimensões $n_l \times n_{l-1}$, onde n_l denota o número de neurônios na camada l, sendo o peso w^l_{jk} posicionado na linha j e na coluna k dessa matriz.

$$z^l = W^l \cdot a^{l-1} + b^l \tag{4}$$

$$a^l = g(z^l) \tag{5}$$

Após a operação de *forward propagation*, é necessário calcular o erro cometido pela rede neural e atualizar todos os parâmetros da rede de acordo com esse erro, processo denominado de *back propagation*. Observa-se que o número de parâmetros é dado por $\sum_{l=1}^{L}(n_l \cdot n_{l-1} + n_l)$. Ou seja, há uma grande quantidade de parâmetros para serem otimizados. Para realizar essa tarefa, utiliza-se o algorítmo de Descida de Gradiente, descrito na Equação 6, onde θ é um parâmetro qualquer, e J é a função de custo.

$$\theta_f = \theta_i - \alpha \cdot \frac{\partial J}{\partial \theta} \tag{6}$$

Esse algorítmo é aplicado a cada um dos parâmetros da rede neural, restando somente a tarefa de encontrar as derivadas parciais da função de custo com relação a cada parâmetro. Nessa implementação, a função de custo utilizada é dada pela função J na Equação 7, onde a função L é denominada *loss function* e m é o número de *inputs* destinados ao treinamento da rede neural.

Não é possível obter as derivadas parciais do J em relação aos parâmetros diretamente, porém, é possível obter primeiramente as derivadas parciais de L para cada *input* (Equação 8) e, em seguida, obter a derivada parcial de J utilizando a Equação 9, onde θ é um parâmetro qualquer.

$$\begin{aligned} J &= \tfrac{1}{m}\textstyle\sum_{i=1}^{m} L(y^i, \hat{y}^i) \\ L(y^i, \hat{y}^i) &= -\textstyle\sum_{c=1}^{n_L}((1-y^i_c)log(1-\hat{y}^i_c) + y^i_c log(\hat{y}^i_c)) \end{aligned} \tag{7}$$

$$\begin{aligned} \tfrac{\partial L}{\partial w^l_{jk}} &= \delta^l_j \cdot a^{l-1}_k \\ \tfrac{\partial L}{\partial b^l_j} &= \delta^l_j \\ \delta^l_j &= \begin{cases} \sum_{p=1}^{n_{l+1}}(w^{l+1}_{pj} \cdot \delta^{l+1}_p) g^{l\prime}(z^l_j) & , l \neq L \\ (\hat{y}^i_j - y^l_j) g^{l\prime}(z^l_j) & , l = L \end{cases} \end{aligned} \tag{8}$$

$$\frac{\partial J}{\partial \theta} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial L}{\partial \theta} \tag{9}$$

Após os gradientes da função de custo serem calculados, o algorítmo de descida de gradiente é utilizado para atualizar os parâmetros para todas as camadas da rede neural (Equação 10).

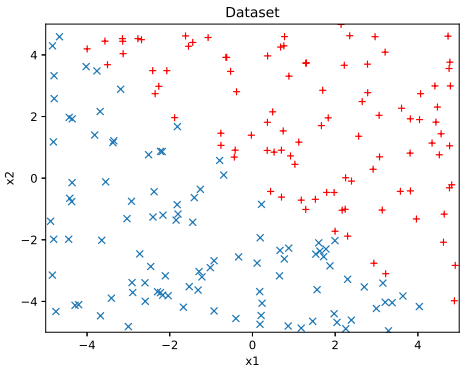
$$\begin{aligned} W^l_f &= W^l_i - \alpha \cdot \nabla_{W^l_i}(J) \\ b^l_f &= b^l_i - \alpha \cdot \nabla_{b^l_i}(J) \end{aligned} \tag{10}$$

Na implementação da rede neural, foram utilizadas somente duas camadas de neurônios.

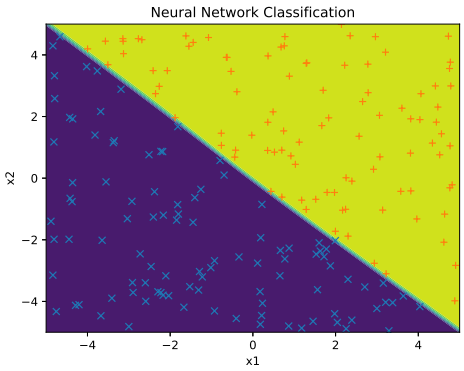
3 Teste da Rede Neural

Após a implementação da rede neural descrita acima, foram realizados, inicialmente, testes com duas funções de classificação diferentes, ambas de duas variáveis. Ou seja, essas funções assumem apenas valores de 0 ou 1. Para cada função, foram amostrados 200 pontos no plano e um vetor contendo a classificação correta de cada ponto para a função usada. Em seguida, foi instanciado uma rede neural com 10 neurônios na primeira camada, denominada camada escondida, e um neurônio na segunda camada. Ou seja, essa rede neural é destinada a classificação para uma única classe.

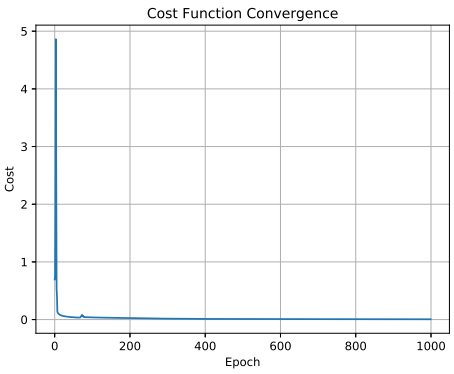
Esses 200 pontos foram então utilizados para treinar a rede neural. Ao todo, foram feitas 1000 épocas de treinamento. Em seguida, a rede neural foi aplicada para classificar pontos discretos e uniformemente distribuídos no plano cartesiano, indicando em quais regiões ela prevê que o resultado da função sendo testada seria 0, e em quais regiões seria 1.



(a) Dados de entrada.



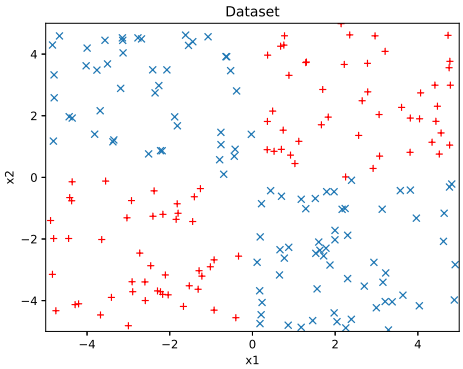
(b) Resultado gerado pela rede neural.



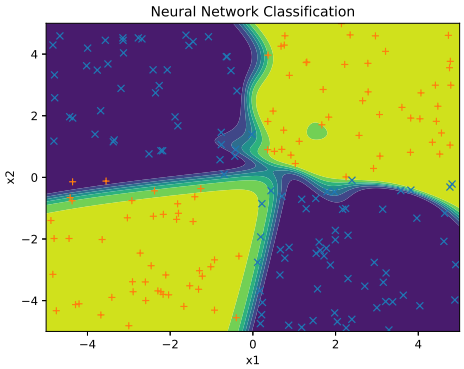
(c) Evolução da função de custo.

Figura 1: Treinamento da rede neural para a função sum_gt_zero.

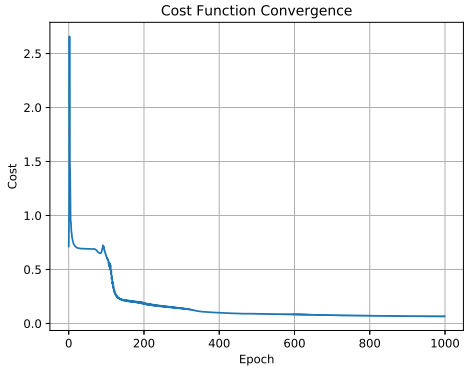
A primeira função testada classifica como 1 todos os pontos acima da reta $y = -x$, e como 0, os pontos abaixo dessa reta. Os resultados do treinamento da rede neural utilizando essa função encontram-se na Figura 1. Observa-se que, nesse caso, a rede neural foi capaz de indicar com exatidão as regiões de cada classificação, separando-as de forma semelhante a uma reta $y = -x$.



(a) Dados de entrada.



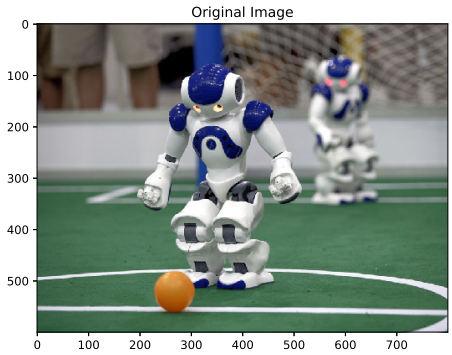
(b) Resultado gerado pela rede neural.



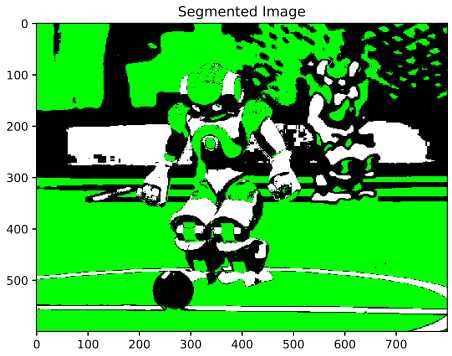
(c) Evolução da função de custo.

Figura 2: Treinamento da rede neural para a função xor.

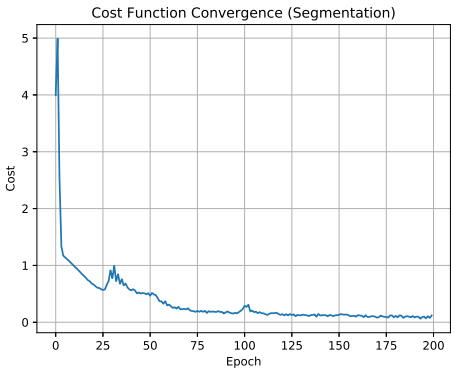
A segunda função testada classifica como 1 os pontos nos quais ambas as coordenadas possuem o mesmo sinal, incluindo o zero nesse caso, e como 0 os pontos cujas coordenadas têm sinais distintos. Os resultados da aplicação dessa função na rede neural encontram-se na Figura 2. Nesse caso, a rede neural foi capaz de separar quase todos os pontos nas suas regiões corretas, sobrando apenas alguns pontos classificados errôneamente, porém, não demarcou as transições entre as regiões de classificação 0 e de classificação 1 exatamente nos eixos x e y, seja por falta de pontos suficientes amostrados nos *inputs*, por falta de mais épocas de treinamento, ou por limitações dessa rede neural específica.



(a) Imagem de entrada.



(b) Resultado da imagem gerada usando segmentação de cores.



(c) Evolução da função de custo.

Figura 3: Treinamento da rede neural para segmentação em branco, verde e preto.

Após o teste da rede neural com as duas funções se mostrar adequado, uma nova instância da implementação, dessa vez com 20 neurônios na camada escondida e 2 de *output*, sendo ela, portanto, destinada a classificação em 2 classes. Essa instância foi, então, usada para realizar a segmentação de cores de uma imagem, podendo classificar pontos da imagem como verdes ou brancos. No caso das duas classes assumirem valor zero, o ponto é classificado como preto.

Para esse treinamento foi usado o conceito de *mini-batch* de 100 elementos de cada classe, ou seja, para cada época de treinamento são escolhidos aleatóriamente 100 pontos classificados como pretos, brancos, e verdes para serem os *inputs* do treinamento. Ao todo, foram realizadas 200 iterações de treinamento.

Em seguida, essa rede neural treinada foi utilizada para realizar a segmentação da imagem inteira. O resultado obtido (Figura 3) foi satisfatório, uma vez que as linhas brancas e as regiões verdes do campo de futebol foram segmentadas corretamente.

Em suma, a rede neural implementada mostrou-se capaz de fornecer bons resultados, mesmo com poucas camadas e uma arquitetura mais símples. Ademais, o algorítmo de Descida do Gradiente se mostrou útil para otimizar grandes quantidades de parâmetros, uma vez que o treinamento da rede neural foi razoavelmente rápido.