Redes Neuronais e Backpropagation

Victor B. R. Jorge

17 de Maio de 2017

1 Introdução

Redes neuronais artificiais são algoritmos que apresentam um modelo matemático inspirado na estrutura dos neurônios do cérebro humano e que adquirem conhecimento através da experiência. Uma grande rede neuronal pode ter centenas ou milhares de unidades de processamento; já o cérebro de um mamífero pode ter muitos bilhões de neurônios. Eles têm um papel essencial na construção da inteligência humana.

Uma rede neuronal artificial é composta por várias unidades de processamento, cujo funcionamento é bastante simples. Essas unidades, geralmente são conectadas por canais de comunicação que estão associados a determinado peso. As unidades fazem operações apenas sobre seus dados locais, que são entradas recebidas pelas suas conexões. A inteligência de uma rede vem das ligações entre seus neurônios.

O uso de redes neuronais artificiais para resolver problemas complexos ou considerados sem solução vem ganhando força nos últimos anos. Hoje, as redes são consideradas estado da arte para a resolução de problemas nas áreas de visão computacional, reconhecimento de linguagem natural, entre tantas outras.

Neste trabalho será implementada uma rede neuronal artificial sem realimentação (*feed-forward*) com três camadas: uma camada de entrada (teórica), uma camada oculta e uma camada de saída. A rede possui 784 entradas e 10 saídas e tem o intuito de classificar a base de dados MNIST (reconhecimento de dígitos escritos à mão). Será implementando o algoritmo de retropropagação (*backpropagation*) para atualizar os pesos da camada oculta. Além disso, implementaremos três algoritmos para o calculo do gradiente: a descida de gradiente estocática, a descida de gradiente e a descida de gradiente em lotes. Por fim, o trabalho tem como objetivo analisar a convergência do erro empírico para diferentes parâmetros na rede (tamanho da camada oculta, taxa de aprendizado e algoritmo de calculo do gradiente).

2 Descrição e implementação

A rede neuronal foi implementada na linguagem Python utilizando-se de orientação a objetos. Basicamente, foram implementadas duas classes: a classe Neuron e a classe NeuralNetwork. Cada Neuron possui sua função de ativação, seus pesos e bias, seu delta e sua saída. Os pesos e o bias foram inicializados com valores entre $-0,001$ e $0,001$. A NeuralNetwork é responsável por criar as camadas de acordo com os parâmetros e adicionar neurônios a elas. Além disso, ela quem cuida de todos os outros procedimentos que serão descritos em maiores detalhes.

2.1 Conjunto de treinamento

O conjunto de treinamento consiste em 5000 exemplos da base de dados MNIST (reconhecimento de dígitos escritos à mão). Cada linha da base de dados possui 784 valores que

correspondem a um digito escrito à mão. Portanto, temos 10 possíveis classes (dígitos de 0 a 9). A base de dados foi normalizada através da classe *Normalizer* da biblioteca *scikit learn*. A normalização das entradas de uma rede é um fator de extrema importância para o bom funcionamento dos algoritmos de redes neuronais, em geral.

2.2 Funções de custo, ativação e delta

Foi utilizada a função de custo de entropia cruzada neste trabalho. Essa função é bastante utilizada em problemas de classificação multiclasse, quando a saída de de cada neurônio corresponde a uma probabilidade de um certo exemplo pertencer a certa classe. Além disso, quando combinada com a função sigmóide como função de ativação, ela possui uma característica muito interessante: a regra delta passa a ser o valor absoluto do erro. Ou seja: $\Delta = \sigma(x) - y$

2.3 Propagação

A propagação das entradas é realizada de maneira muito simples. Cada exemplo do treinamento é colocado como entrada para a primeira camada da rede e seus neurônios são ativados. A saída dos neurônios ativados é armazenada e serve como entrada para os neurônios da camada seguinte.

2.4 Retropropagação

A retropropagação também é realizada da maneira clássica. O delta de cada neurônio da camada de saída é calculado como o valor absoluto do erro. Para a camada oculta o valor esperado é igual a soma do valor do delta de cada neurônio da camada de saída multiplicado pelo peso respectivo a conexão entre os dois neurônios em questão.

2.5 Atualização dos pesos e bias

2.5.1 Descida de Gradiente Estocática

Na descida de gradiente estocática os pesos são atualizados a cada exemplo do conjunto de treinamento. Portanto, para cada exemplo no conjunto de treinamento após as etapas de propagação e retropropação, os pesos são atualizados para cada neurônio pela seguinte regra: $W = W - \alpha * \Delta * I$ e $b = b - \alpha * \Delta$. Onde I corresponde ao exemplo dado para a camada oculta e à saída da camada oculta para a camada de saída.

2.5.2 Descida de Gradiente e descida de gradiente em lotes

A descida de gradiente e a descida de gradiente são praticamente o mesmo algoritmo. O valor de $\Delta * I$ é acumulado para um número m de exemplos e os pesos atualizados após todos esses exemplos passarem pela rede. Na descida de gradiente m = quantidade de exemplos do conjunto de treinamento. E na descida de gradiente em lotes m é menor que esse valor. Ao atualizar os pesos é tirada uma média do valor acumulado. Portanto, temos: $W = W - \alpha * \frac{1}{m} \sum(\Delta * I)$ e $b = b - \alpha * \frac{1}{m} \sum(\Delta)$

2.6 Cálculo do Erro e Épocas

A Rede foi treinada em épocas e o erro médio de cada época foi calculado através da função de custo. Para todos os experimentos realizados neste trabalho o número de épocas foi igual a 100.

3 Resultados e Análise

Os resultados da convergência do erro empírico para cada variação de parâmetro podem ser visualizados nos gráficos apresentados nas seções abaixo. Todos os experimentos foram realizados considerando-se a Descida de Gradiente com 50 neurônios na camada oculta e taxa de aprendizado $\alpha = 1$ como parâmetros de referência. Isso quer dizer que se estamos comparando a variação de algum dos parâmetros, por padrão, os outros pertencerão a essa configuração. Isso se deve ao fato de esses parâmetros terem se mostrado os mais estáveis durantes os experimentos. A Descida de gradiente possui uma curva mais suave, enquanto 50 neurônios na camada oculta se mostrou bem ágil na execução. Além disso, a taxa de aprendizado $\alpha = 1$ se mostrou bem mais conveniente que $\alpha = 0.5$, enquanto $\alpha = 10$ não converge na maioria dos experimentos.

3.1 Variação da taxa de aprendizado

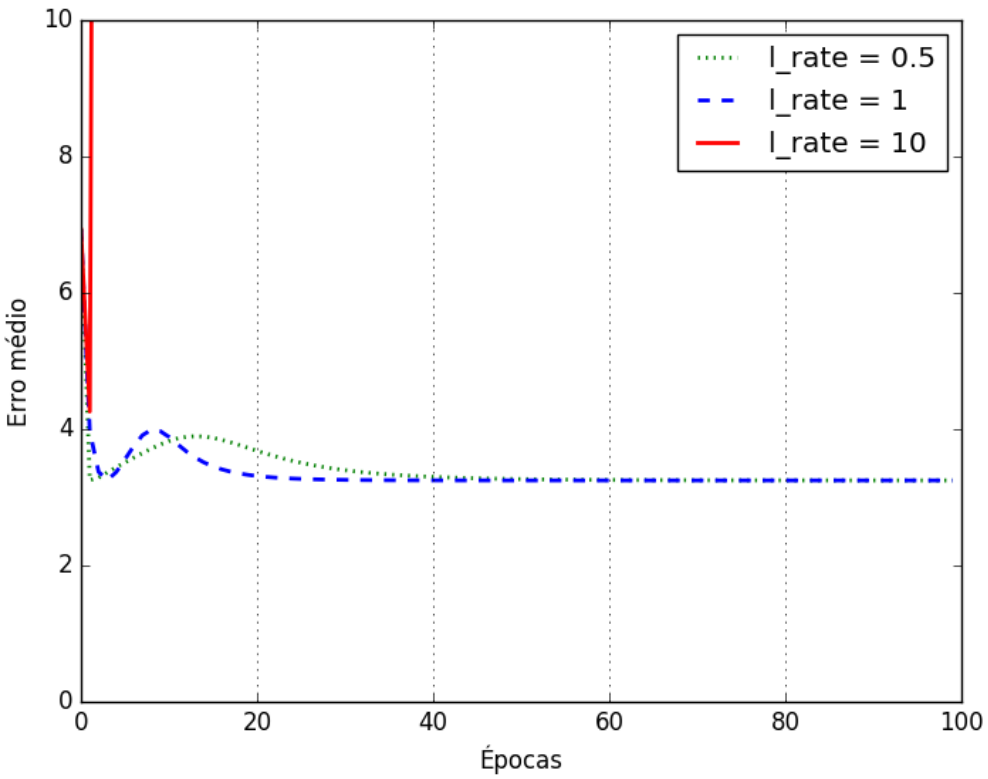

Figure 1: Erro médio em função das épocas para a Descida de Gradiente com 50 neurônios na camada oculta

Como podemos observar, para $\alpha = 10$ o erro empírico diverge e portanto esse parâmetro se mostrou incompatível com nossa rede. $\alpha = 1$ e $\alpha = 0.5$ convergem, porém $\alpha = 1$ converge entre a 15^a e a 20^a época, enquanto $\alpha = 0.5$ converge em torno da 40^a época. Isso mostra que a taxa de aprendizado é um parâmetro sensível e deve ser bem testado para se alcançar um valor com um bom *trade-off* entre convergência e rapidez de convergência.

3.1.1 Comparação entre os algoritmos de cálculo de gradiente

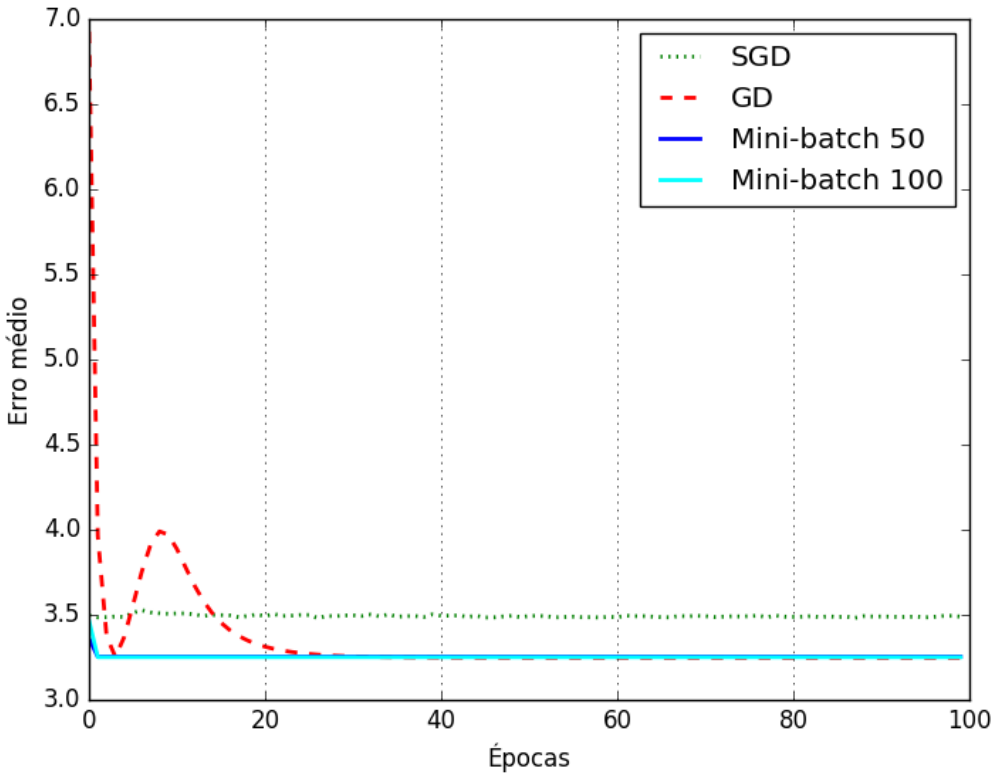

Figure 2: Erro médio em função das épocas com 50 neurônios na camada oculta e $\alpha = 1$

Como podemos observar, o algoritmo de descida em lotes se mostrou muito mais veloz ao convergir que os outros algoritmos. A descida estocástica não chega a convergir para o erro mínimo. Isso indica que a execução possa ter tido o problema de ficar presa em um mínimo local. Existem algumas técnicas para evitar esse problema que não serão discutidas aqui. Enquanto isso, a descida de gradiente chega a um mínimo com poucas épocas, porém o erro volta a aumentar e e se estabiliza no mínimo por volta de 25 épocas.

3.1.2 Variação do número de neurônios na camada oculta

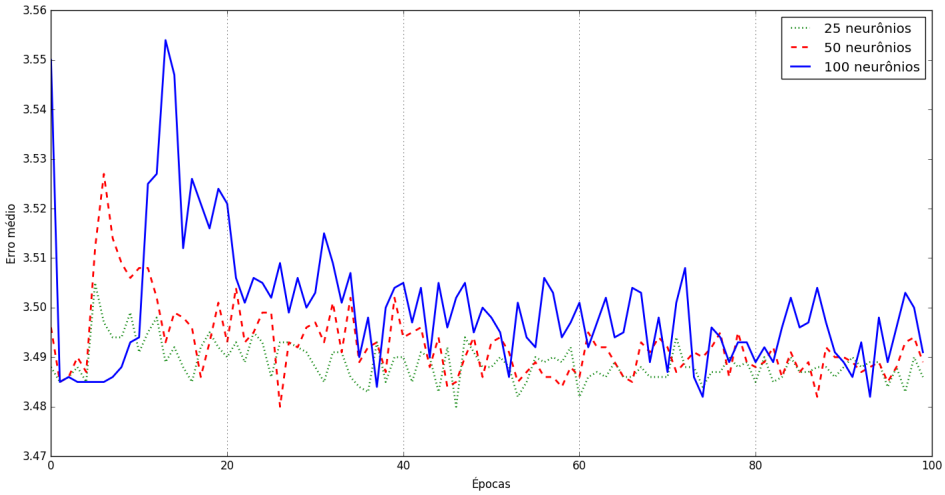

Figure 3: Erro médio em função das épocas com descida de gradiente estocástica e $\alpha = 1$

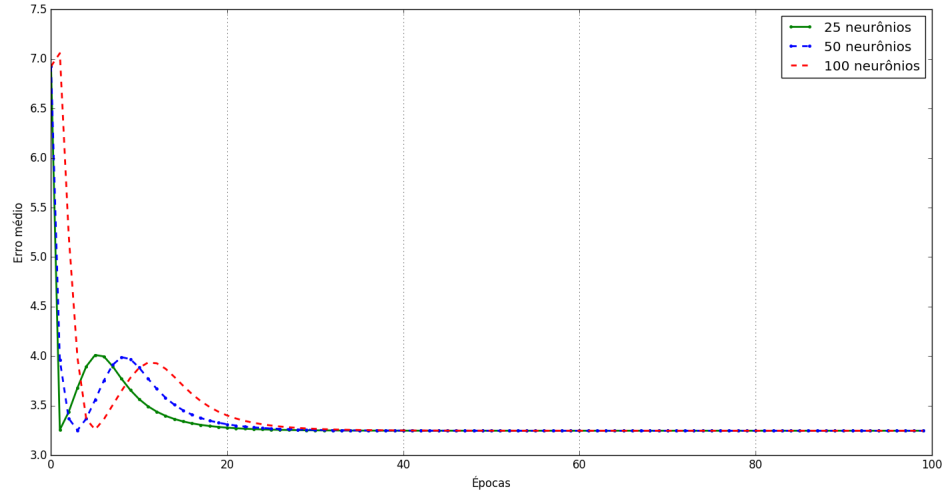

Figure 4: Erro médio em função das épocas com descida de gradiente e $\alpha = 1$

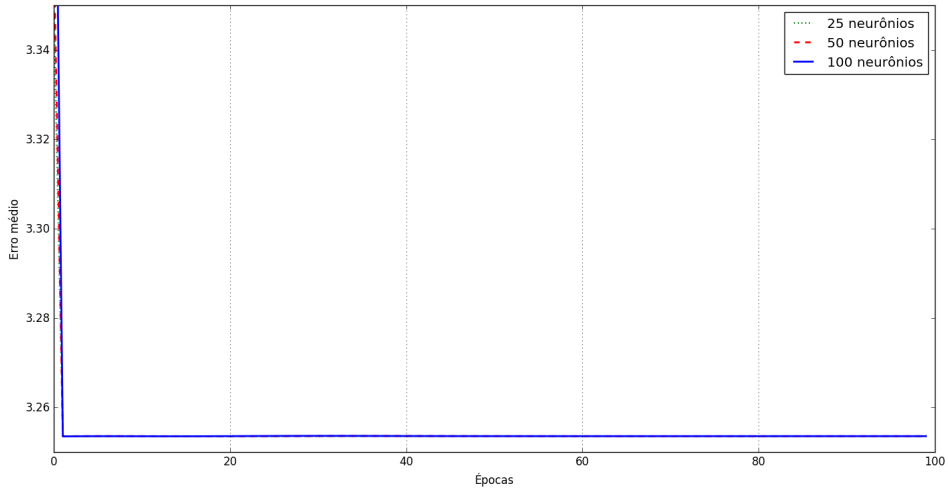

Figure 5: Erro médio em função das épocas com descida de gradiente em lotes de tamanho 50 e $\alpha = 1$

Como podemos observar nos gráficos acima, a decida de gradiente estocástica é bastante instável e apresenta um valor de erro menor para menos neurônios na camada oculta. Apesar de ser um algoritmo mais estável, a descida de gradiente também apresenta esse comportamento. O algoritmo em lotes é tão rápido na convergência que fica impossível realizar qualquer comparação de causa e efeito relativo à alteração no número de neurônios da camada oculta. Devemos lembrar que o erro empírico não consiste em uma boa métrica para avaliar a qualidade de uma rede neuronal artificial. Na maioria das vezes, a minimização do erro empírico leva à uma divergência do erro esperado.

Além disso, é possível relacionar uma piora na convergência do erro empírico a um aumento na complexidade da rede. Complexidade que está relacionada ao aumento da camada oculta que é responsável pela transformação do espaço de entrada do problema.

4 Conclusão

Concluí-se que as redes neuronais são de extrema importância na resolução de problemas complexos atuais. Além disso, a implementação de uma rede neuronal é trabalhosa e sua execução e sucesso é bastante sensível aos parâmetros de entrada. Ademais, há de se destacar a importância do algoritmo de retropropagação de erros no sucesso do aprendizado da rede.

A implementação deste trabalho foi bastante trabalhosa porém os resultados são satisfatórios. Fazer a máquina aprender é uma experiência fantástica. Também foi importante aprender, na prática, a diferença entre os algoritmos de cálculo de gradiente e o quão sensível é uma rede aos parâmetros de entrada.