Redes Neuronais e Backpropagation
Utilizando o dataset MNIST

André Correia Lacerda Mafra

Maio 2017

1 Resumo

Seres humanos são muito bons em reconhecer imagens, padrões, dentre outras coisas relativas ao campo visual. Sem esforço, somos capazes de reconhecer dígitos escritos em uma folha de papel.

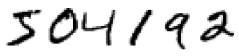

Figure 1: Digitos escritos manualmente

Isso é possível devido à forma com a qual nosso cerébro funciona, dividido em várias camadas, o cerébro atua sobre a informação em partes e com grande processamento, através de bilhares de conexões de neurônios e sinapses.

Dada a complexidade do cerébro, é extremamente complexo montar uma modelagem computacional que solucione problemas, como o de reconhecimento de dígitos.Ainda que em constante evolução, algoritmos de Aprendizado de Máquina foram desenvolvidos e, hoje, possuem um excelente desempenho.

Redes neuronais são um paradigma de programação inspirado na biologia dos neurônios, tal paradigma permite computadores aprenderem através de um treino com dados informacionais, os quais atravessam uma rede de camadas repletas de "neurônios", responsáveis pela computação dos dados.

No trabalho prático proposto, foi pedido que um programa de computador fosse elaborado, utilizando Redes Neuronais e o algoritmo de Backpropagation, a fim de que fosse possível reconhecer dígitos escritos manualmente, através do dataset MNIST, disponibilizado conjuntamente com a especificação do problema.

2 Especificação do problema

O cenário do problema consiste em reconhecer dígitos em forma de imagem, que são representados na escala de cinza, 0 à 255. O dataset utilizado foi retirado do MNIST Database (http://yann.lecun.com/exdb/mnist/), o qual possui imagens já formatadas na escala de cinza. A especificação pede para que seja criada uma rede neuronal de três camadas, que recebem 5 mil amostras distintas do dataset, na qual cada uma contém 784 features, representando cada pixel da imagem (28x28)

3 Solução

3.1 Modelagem da solução

A modelagem especificada na solução constitui-se de uma rede neuronal de três camadas, sendo elas, respectivamente,camada de entrada,*hidden layer* e uma camada de saída.

3.1.1 Camada de entrada

A camada de entrada possui 785 neurônios, pois cada uma das 5000 entradas são representadas por um vetor de 784 posições + o bias.

3.1.2 Hidden Layer

A Hidden Layer varia entre, 25, 50, 100) neurônios,Sendo acrescidos de 1 por conta do Bias, assim como foi pedido na especificação do problema.

3.1.3 Camada de saída

A camada de saída conta com 10 neurônios, um para cada resultado possível, dígito entre 0 e 9. Cada neurônio irá conter uma probabilidade entre 0 e 1, de que aquele neurônio correspondente seja o valor real, o maior valor corresponte a predição do algoritmo.

3.1.4 Rede Neuronal

No final da modelagem, a rede teria uma representação gráfica como a da Figura 2. No desenho, as setas representam os pesos, que são as ligações entre os neurônios, existem dois feixes de setas no desenho, que representam duas matrizes de peso, em termos de modelagem computacional, as dimensões das matrizes refletem na quantidade de ligações de cada neurônio, como na camada de entrada, existem 785 neurônios e cada um se liga à todos os neurônios da *hidden layer*, uma boa forma de representá-los é através de uma matriz 785xn, onde n é o número de neurônios na hidden layer. Dessa mesma forma, a segunda matriz de pesos é constituída de nx10, em que 10 representa o número de neurônios da terceira camada.

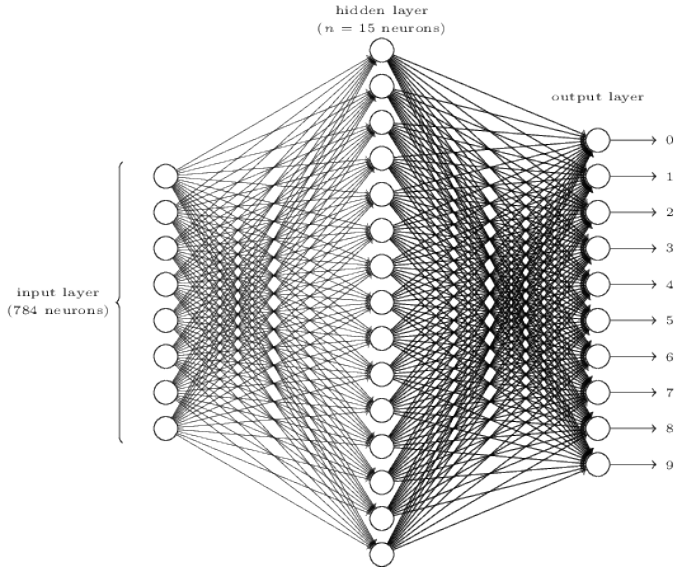

Figure 2: Representação da rede neuronal com 15 neurônios na hidden layer

3.2 Algoritmos

Para o treinamento da rede, foram implementados diversos algoritmos, a fim de obter uma minimização do erro de treino.

3.2.1 Algoritmos de Propagação do erro

- *Forward Propagation:* Propagação dos erros, desde à primeira camada até a camada de saída. O erro é dado pela sigmoide do produto vetorial de X e w, onde X são as entradas e w os pesos. A sigmoide tem como papel a não linearização do erro, transformando-o em uma probabilidade entre 0 e 1. O erro é propagado para a camada seguinte, uma vez que a camada anterior se transforma na matriz X da conseguinte.

- *Backpropagation:* De forma similar ao Forward Propagation, o Backpropagation propaga os erros entre camadas, a fim de melhorar separar a responsabilidade de cada camada. Porém este algoritmo propaga o erro na ordem inversa, de trás para frente,desde a camada de saída até a de entrada. A tendência é que o erro empírico diminua à cada passada pela rede.

3.2.2 Algoritmos de Minimização

- *Gradient Descent (GD):* Algoritmo utilizado para minimizar a função de custo.A cada época (5000 entradas) é atualizado os novos pesos, até que, como o esperado, chegue à um valor mínimo, podendo este muitas vezes

ser minimo local ou global.Um contraponto do algoritmo é que ele é mais custoso, por ler todo o *training set* de uma só vez.

- *Stochastic Gradient Descent (SGD):* Neste método, o valor dos parâmetros são atualizados á cada interação, possibilitando uma convergência mais rápida, por outro lado, a convergência tende à ser não tão minimizada quanto á do Gradient Descent.Os custos minimizados no SGD também são mais ruidosos e oscilam bastante.

- *Mini-Batch Gradient Descent:* o Mini-batch Gradient Descent é uma alternativa para tentar balancear as qualidades e dificuldades do SGD e GD, de tal forma que o algoritmo continue sendo rápido e com melhor minimização. Por isso o Mini-batch é considerado um bom trade-off.

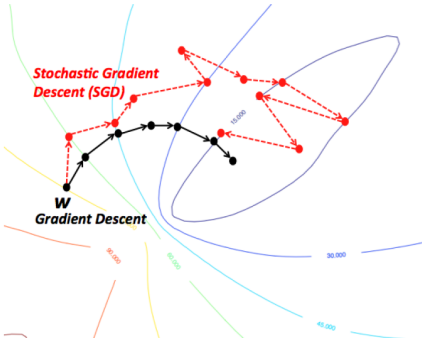

Figure 3: Diferença entre Gradient Descent e Stochastich Gradient Descent

3.3 Passo a Passo

O programa final é um compêndio dos algortimos citados na última subseção, os quais se relacionam à fim de melhorar o desempenho do programa final em reconhecer os digitos das entradas.

Basicamente, o algoritmo, após a inicialização dos pesos e inputs, começa com o *forward propagation*, que realiza os cálculos de propagação do erro, ao final calculando o erro da última camada, subtraindo o erro da camada de saída do erro esperado, extraído do dataset.

O passo seguinte é o backpropagation, o qual propaga o erro até a camada de entrada, de forma reversa.

Por fim, algum algortimo de minimização deve entrar em ação a fim de atualizar os pesos, seja ele o GD, SGD ou Mini-Batch.

4 Resultados

4.1 Critérios

Foram realizados 36 testes, alterando variáveis com os seguintes critérios:

- Taxa de aprendizado, variando com valores 0.5, 1.0 e 10.0.
- Alternando com 25, 50 e 100 neurônios da hidden layer.
- Mini-Batch: Foram realizados testes alterando o tamanho do Batch, entre 10 e 50.

4.2 Testes

Os testes foram divididos entre aqueles utilizado GD, SGD e Mini-Batch. As matrizes de peso utilizadas são sempre as mesmas, em todos os testes, um arquivo .txt foi criado contendo uma matriz randômica de pesos que é lida toda vez que o programa é executado.

4.3 Ambiente de teste

Todos os testes foram executados em uma máquina Dell Inspiron 15R, 8GB de RAM, Processador i7 com Sistema Operacional Ubuntu 16.04.

4.4 Gradient Descent

Os algoritmos foram executados 100 vezes em um loop,com excessão do caso de 100 hidden layers que foram 300 vezes chamadas, ambos utilizando uma média de pesos para melhor aferir a atualização dos mesmos.

A variação de 100 chamadas para 300, foi proposital. Pois após esta mudança o algoritmo demorou muito mais tempo para rodar, uma diferença de vários minutos, em compensação a rede convergiu com uma melhor minimização e o gráfico ficou com um aspecto de menos oscilação.

Os gráficos foram divididos pela quantidade de neurônios na hidden-layer e cada um deles, levava em consideração o *learning rate*, ou taxa de aprendizado. Os resultados mostraram que para altas taxas de learning rate, como 10, o resultado não converge e acaba oscilando entre uma mesma faixa de valores, isso acontece por que o gradient descent tende à dar "passos maiores", atualizando os pesos em um grande faixa, o que gera um baixo desempenho e uma divergência do resultado.

Também pode ser observado a discrepância de convergência entre os exemplos com taxa de aprendizado 0.5 e 1, o primeiro caso converge melhor do que o primeiro, isso acontece, porque dessa vez, o algortimo dá "pequenos passos" o que permite uma maior aproximação à tangente de descida do gradiente.Por outro lado, uma taxa de aprendizado menor, significa maior custo de processamento, demorando um pouco mais para retornar um valor.

Por fim, outra comparação possível, é com a quantidade de neurônios na *hidden layer*, comparando os três diferentes gráficos do Gradient Descent, observa-se que, quanto mais camadas escondidas, maior o desempenho, isso se dá, pelo fato de haverem mais neurônios processando os pesos e realizando mais "sinapses".

4.4.1 25 Hidden Layers

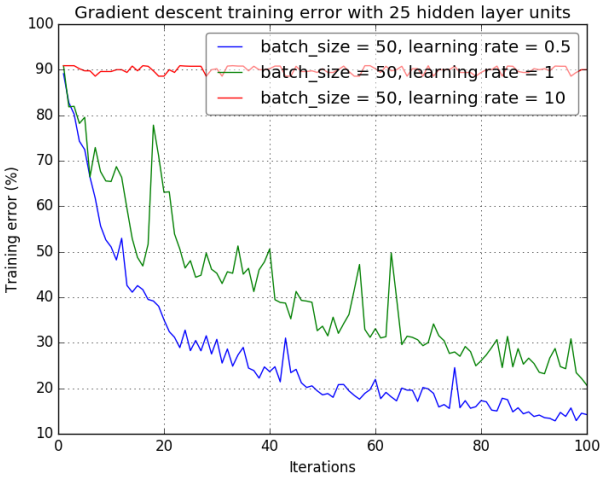

Figure 4: Gradient Descent com 25 hidden Layers

4.4.2 50 Hidden Layers

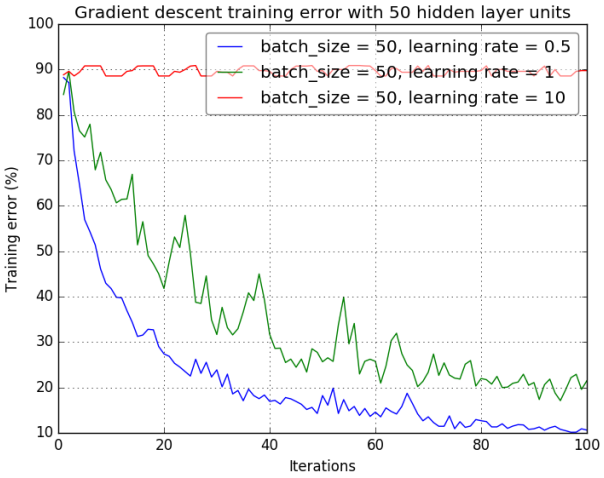

Figure 5: Gradient Descent com 50 hidden Layers

4.4.3 100 Hidden Layers

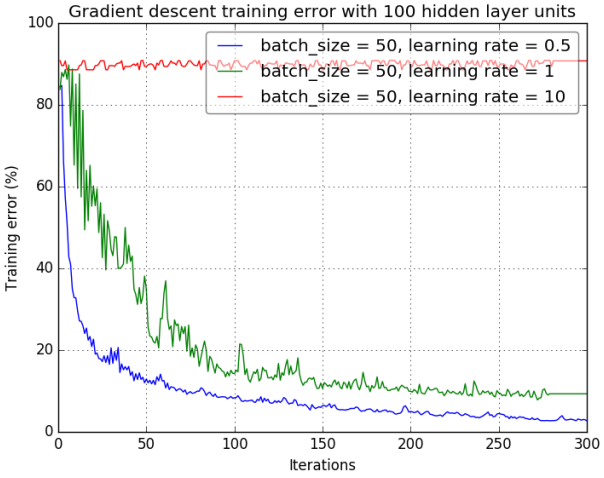

Figure 6: Gradient Descent com 100 hidden Layers

4.5 Stochastic Gradient Descent

Os algoritmos foram executados 70 vezes em um loop que chama a função de backpropagation.

O SGD em relação ao Gradient Descent teve uma conversão mais baixa, na maioria dos casos, isso é normal, dado que o SGD realiza um *tradeoff* de minimização da conversão por desempenho de tempo. Isso acontece pois o SGD atualiza os pesos à cada passada.

Assim como o GD, o SGD demonstrou divergência com uma taxa de aprendizado alta de 10, porém os resultados foram melhores nesse caso , comparado ao GD.

4.5.1 25 Hidden Layers

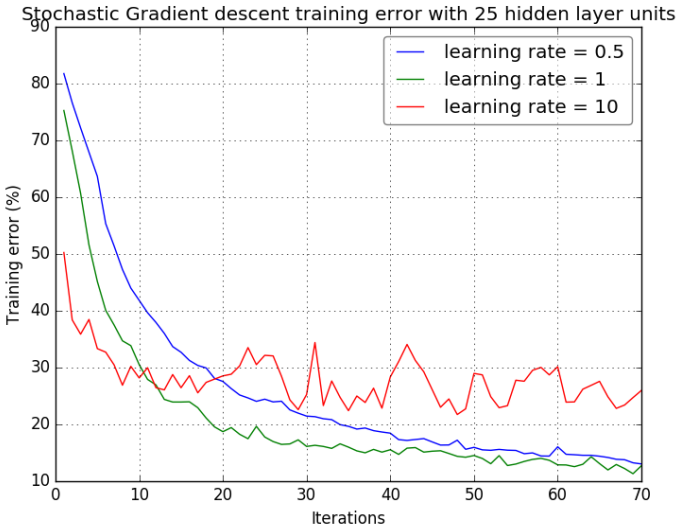

Figure 7: Stochastic Gradient Descent com 25 hidden Layers

4.5.2 50 Hidden Layers

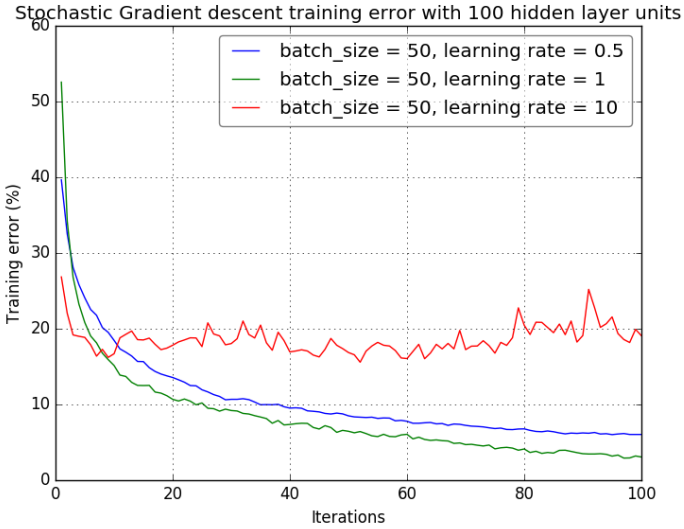

Figure 8: Stochastic Gradient Descent com 100 hidden Layers

4.5.3 100 Hidden Layers

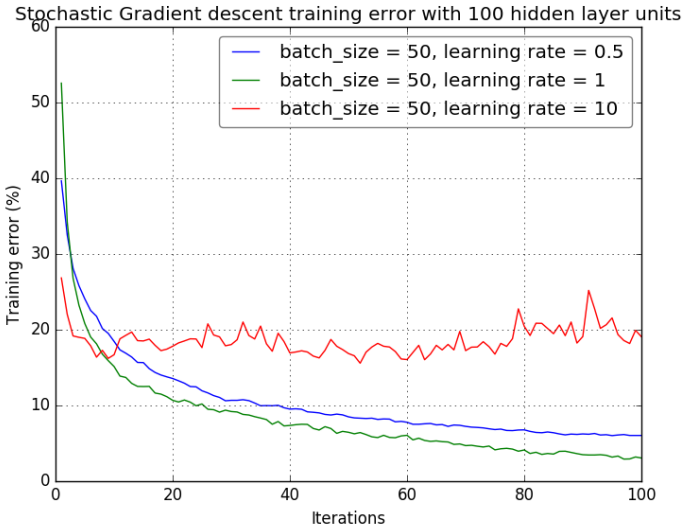

Figure 9: Stochastic Gradient Descent com 100 hidden Layers

4.6 Mini-Batch Gradient Descent

Os algoritmos foram executados 100 vezes em um loop, utilizando uma média de pesos para melhor aferir a atualização dos mesmos.

Comparando a variação do tamanho do Batch, é percebido que o desempenho não melhora muito, o conjunto com maior batch minimiza melhor o erro, porém não tem muito a diferença, por outro lado, é um pouco mais lento, uma vez que lê mais exemplos do treino de uma vez.

4.6.1 25 Hidden Layers

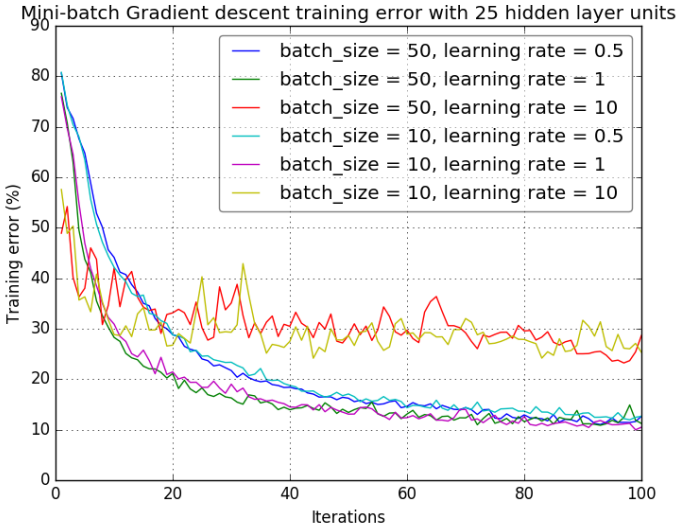

Figure 10: Mini-Batch com 25 hidden Layers

4.6.2 50 Hidden Layers

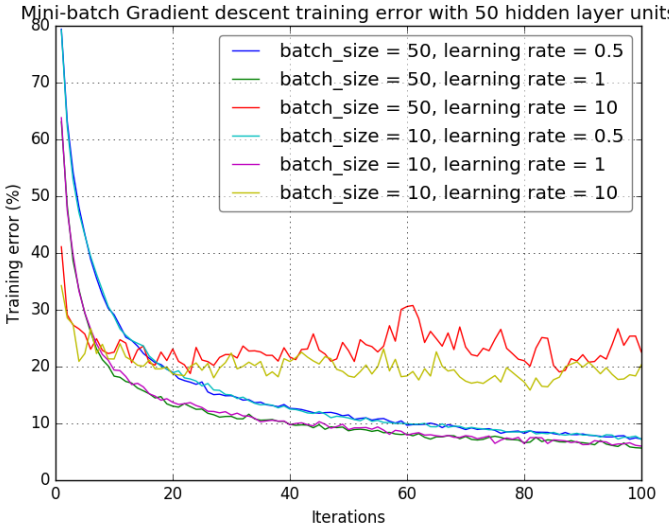

Figure 11: Mini-Batch com 50 hidden Layers

4.6.3 100 Hidden Layers

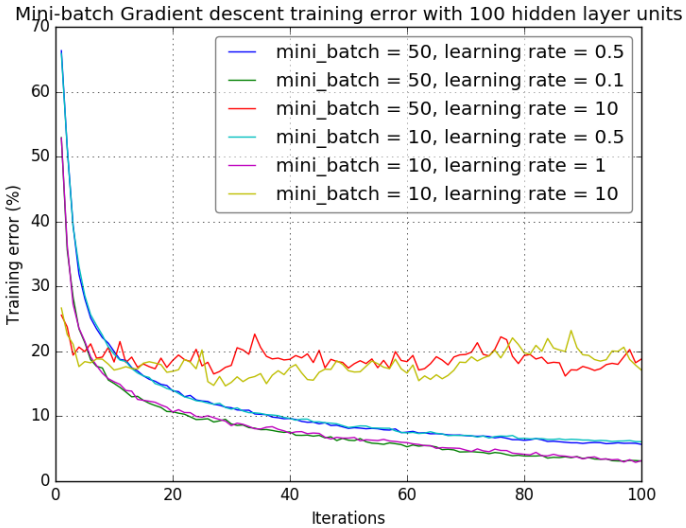

Figure 12: Mini-Batch com 100 hidden Layers

4.7 Observações

O Gradient Descent é uma abordagem que computa todos o treino de uma só vez, isso faz com que ele tenha um bom desempenho ao minimizar seu erro, porém converge mais lentamente e tem mais tempo de processamento.

O Stochastic Gradient Descent traz outra abordagem, em que computa um treino por vez, convergindo mais rapidamente, porém não minimiza tão bem o erro quanto o Gradient Descent.

Uma boa escolha entre ambos é o mini-Batch Gradient Descent, que computa pacotes menores, dessa forma, o resultado converge melhor e mais rapidamente.

Nos resultados obtidos no experimento deste trabalho, o mini batch demonstrou maior rapidez que o Gradient Descent, e melhor minimização que o SGD, porém não obteve melhor minimização que o Gradient Descent, o que reinforça a idéia do *tradeoff*, sendo o mini-batch a melhor escolha em muitos casos, por trazer um melhor custo-benefício.

4.8 Técnicas Utilizadas

Foi utiizado para a implementação, a linguagem Python e uma biblioteca para desenvolvimento, Numpy, essa é uma biblioteca para cálculos matemáticos e representações geométricas, como matrizes e vetores.

Também foi utilizado a bilbioteca matplotlib para plotar os gráficos.

5 Conclusão

Foram implementados nesse trabalho os algoritmos de Forward Propagation, Backpropagation, GD, SGD e mini-batch para resolver o problema de reconhecimento de digitos provindos do dataset MNIST. Com este trabalho foi possível praticar os conceitos de Aprendizagem de Máquina, implementando algoritmos vistos em sala de aula, além de reforçar conceitos teóricos, como Capacidade, Regularização, Variância, dentre outros. O trabalho também foi de grande importância para rever conceitos de álgebra e aprender a linguagem Python.