Machine Learning - Documentação do TP1

Lucas C. Lima[1]

[1]Universidade Federal de Minas Gerais (UFMG)

1cs.chv@gmail.com[1]

1. Introdução

O aprendizado de máquina é uma das áreas da ciência da computação que mais evoluiu nos últimos anos. Hoje, diversas técnicas de aprendizado de máquina vem sendo utilizados em diversas tarefas diárias como detecção de fraudes, resultados de pesquisa na web, reconhecimento de imagens e reconhecimento de voz, entre outros. Um novo campo de pesquisa de rede neural, conhecido como aprendizagem profunda vem se tornando uma técnica bastante popular. A aprendizagem profunda combina avanços no poder de processamento e tipos especiais de redes neurais para aprender padrões em grandes quantidades de dados. Neste trabalho, temos como objetivo implementar uma rede neural capaz de reconhecer dígitos manu-escritos. O restante deste documento apresentará as principais decisões, detalhes sobre a implementação desta rede neural e uma análise experimental.

2. Principais decisões

- A rede foi implementada utilizando a linguagem Python na versão 2.7.12.
- Foi criada uma classe NeuralNet que possui os métodos e necessários para a execução do algoritmo backpropagation
- Foi implementado uma rede neural de 3 camadas sendo elas: camada de entrada, camada oculta e camada de saída.
- A função de ativação utilizada foi a Sigmóide.
- A função de perda utilizada como função de perda foi a *Cross Entropy*.
- Foram implementadas 3 algoritmos para realização do cálculo do gradiente: Gradient Descent, Stochastic Gradient Descent e Mini-batch.
- Foram utilizados matrizes para representação dos dados de entrada, pesos e erros.
- Foram utilizadas as bibliotecas Pandas, scikit-learn e numpy para auxiliar na implementação e avaliação.
- Para a avaliação, foram executados 1000 épocas (onde uma época representa 5000 entradas processadas) para cada variação de parâmetros e algoritmos de cálculo de gradiente.

3. Descrição da Rede Neural implementada

Nesta seção são apresentados detalhes sobre a arquitetura da rede, implementação do algoritmo backpropagation e como foram realizados os cálculos de gradientes.

3.1. Arquitetura da Rede Neural

A rede neural a ser implementada deverá ser capaz de realizar o reconhecimento de dígitos manu-escritos utilizando como treinamento o MNIST_dataset. MNIST trata-se de 5000 entradas, onde a primeira coluna representa o dígito correto e o restante, é representado por uma matriz de 28x28, ou seja, um vetor de 784 dimensões representando um dígito manu-escrito. A rede será composta de três camadas:

1. Camada de entrada: cada unidade representa uma dimensão do dado de entrada.
2. Camada oculta: cada unidade representa uma transformação a partir das unidades de entrada.
3. Camada de saída: cada unidade representa a chance da saída correspondente ser a correta.

Abaixo é apresentado na 1 figura é apresentando o esquema da arquitetura da rede neural implementada.

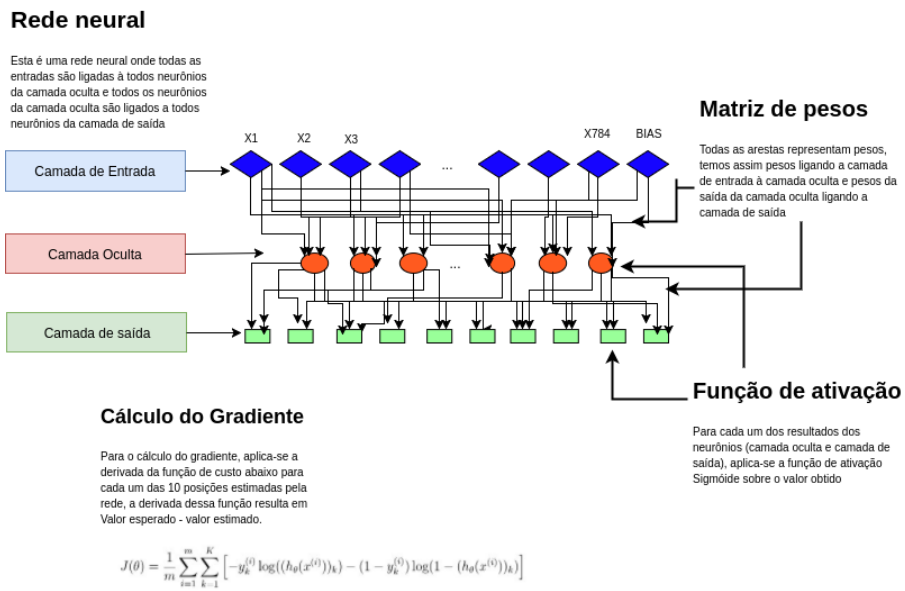


Figura 1. Arquitetura da rede neural

3.2. Algoritmo backpropagation

A realização do algoritmo backpropagation [Rumelhart et al. 1995, Ber] pode ser dividida em três etapas:

1. ForwardPass
2. Cálculo do gradiente
3. Atualização dos pesos

Primeiramente, cria-se uma matriz de pesos aleatórios, $weight$, ligando cada entrada a todos neurônios da camada oculta, em seguida cria-se a matriz de pesos aleatórios que ligam a saída dos neurônios da camada oculta com a camada de saída $weight\_out$. O próximo passo é realizar o *forwardpass*, para isso realiza-se o produto interno entre o vetor de entrada $input$ e a matriz $weight$, após esse cálculo, aplica-se a função de ativação em todos os resultados obtidos desse produto interno, temos então que $Y = sigmoide(input \cdot weight)$. Mais adiante, podemos obter o resultado da nossa predição calculando $Z_{pred} = sigmoide(Y \cdot weight\_out)$.

Após o *forwardpass* é necessário realizamos o cálculo do gradiente, primeiramente devemos mapear o valor esperado(label) para um vetor de 10 posições, onde esse vetor apenas terá 1 na posição do label de entrada. Feito isso, o gradiente pode ser obtido através do resultado das derivadas parciais da função de perda [Sadowski]. Aplicando a derivada na função de perda, verifica-se que o gradiente pode ser obtido da subtração do

vetor esperado pelo vetor de predição (resultados estimados por cada um dos neurônios da camada de saída), temos então que $derror = Z_{exp} - Z_{pred}$, onde $derror$ é o erro obtido na camada de saída. Em seguida, deve-se calcular o erro dos neurônios da camada oculta, valor este que pode ser obtido através da propagação do erro para cada neurônio, para isso multiplica-se o vetor $derror$ pela transposta da matriz de pesos da saída $weight\_out$ e pela derivada da sigmóide $h(x)$, pode-se então obter o erro de cada neurônio da seguinte maneira: $neurons\_err = (derror \cdot weight\_out^T) * sig(y)'$.

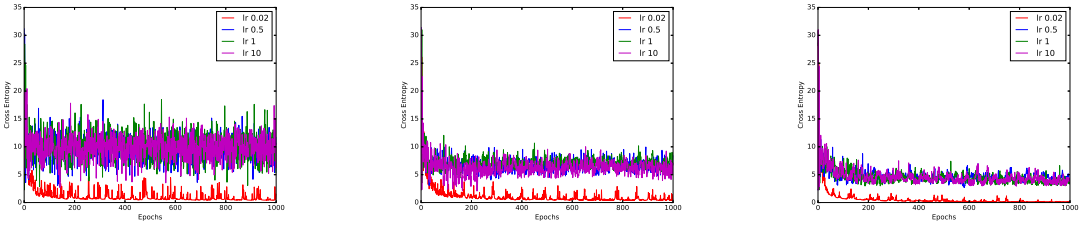
Por fim devemos realizar a atualização dos pesos. Para isso, devemos atualizarmos a matriz $weight$, fazemos isso da seguinte maneira $weight+ = learning\_rate * (input^T \cdot neurons\_err)$ e $weight\_out = learning\_rate * (Y^T \cdot derror)$.

4. Análise experimental

Com intuito de avaliar os diferentes algoritmos de cálculo de gradiente, realizamos as variações dos parâmetros para cada um dos algoritmos tais como: learning rate, tamanho do batch, e número de neurônios na camada oculta. Para cada uma dessas variações executamos 1000 épocas.

4.1. Gradient Descent

O algoritmo Gradient Descent realizará o cálculo do gradiente apenas após a execução de todas as entradas da base, isto é realizará atualização de seus pesos apenas a cada época. Nos gráficos abaixo é possível verificar as curvas de convergência do algoritmo gradiente. Observa-se que a medida em que diminuímos o learning rate obtemos melhores resultados. É notável também que com o aumento do número de neurônios na camada oculta o algoritmo obtêm melhores resultados, chegando a valores próximos de zero quando utilizado 100 neurônios e learning rate 0,02. Um fator a se observar é que para esse algoritmo, ele só aproxima-se de convergir para valores pequenos de learning rate.



(a) 25 neurônios na camada oculta

(b) 50 neurônios na camada oculta

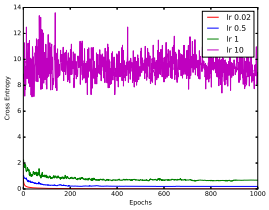
(c) 100 neurônios na camada oculta

Figura 2. Resultados do algoritmo Gradient Descent variando os parâmetros learning rate e número de neurônios na camada oculta

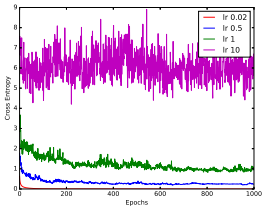
4.2. Stochastic Gradient

O algoritmo Stochastic gradient, diferentemente do gradient descent, realiza o cálculo de seu gradiente a cada entrada processada. Após o cálculo de seus gradientes atualiza seus pesos, desta forma ele realizará 5000 atualizações de pesos a cada época. É notável que o SGD obtêm resultados melhores do que o GD, no entanto, diferentemente do GD, a medida que aumentamos o número de neurônios na camada oculta as melhorias observadas não são tão significativas. Em relação ao learning rate, novamente podemos observar

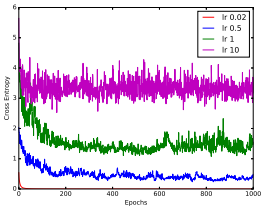
que quando utilizado learning rate baixo o algoritmo tem melhores resultados e quando utilizado learning rate 1 e 10 o algoritmo não converge. Abaixo encontra-se os resultados obtidos para esse algoritmo.



(a) 25 neurônios na camada oculta



(b) 50 neurônios na camada oculta



(c) 100 neurônios na camada oculta

Figura 3. Resultados do algoritmo Stochastic Gradient variando os parâmetros learning rate e número de neurônios na camada oculta

4.3. Mini-batch

O algoritmo Mini-batch consiste em calcular o gradiente após um certo número de entradas processadas. Para essas análises, utilizaremos batchs de tamanhos 10 e 50. Isto é, para cada época, o cálculo do gradiente será realizado 500 e 100 vezes respectivamente. Nesta análise dispomos os gráficos em duas colunas, à esquerda os gráficos referentes ao algoritmo com tamanho de batch igual a 10 e a direita igual a 50. Em ambos o comportamento dos resultados variando o learning rate e número de neurônios é similar, obtendo melhorias a medida que diminuimos o learning rate e aumentamos o número de neuronios, no entanto o algoritmo obtêm resultados melhores para batchs de tamanho igual a 10. Novamente o algoritmo não converge para learning rates igual a 1 e 10. Os resultados são mostrados na figura 4

5. Considerações finais

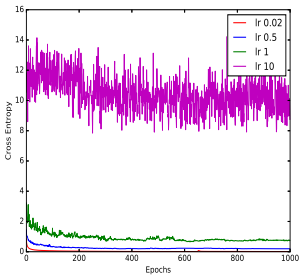
Visto esta analise, podemos concluir que o algoritmo GD é mais constante pois estamos dando apenas um passo a cada passada na base de treino, no entanto, para bases maiores, o algoritmo demorará mais para atualizar os pesos e levará mais tempo para convergir para o mínimo global. Por outro lado, o algoritmo SGD não é tão constante em direção do minimo global pois realiza atualizações a cada entrada da base de treino, formando zig-zags em direção ao mínimo global, no entanto mostramos que quase sempre converge para o mínimo global. Para esta base, quando utilizado 100 neurônios na camada oculta e learning rate de 0,02 os algoritmos SGD, MB10 e MB50 tem praticamente a mesma performance convergindo para erros empíricos praticamente zero. Na 5 encontra-se uma comparação dos 4 algoritmos utlizando 100 neurônios na camada oculta e learning rate de 0,02.

Referências

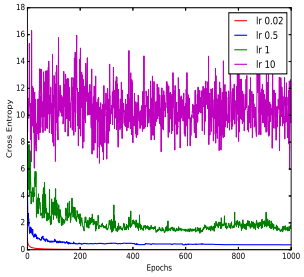
Principles of training multi-layer neural network using backpropagation. http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html. Accessed: 2017-05-10.

Rumelhart, D. E., Durbin, R., Golden, R., and Chauvin, Y. (1995). Backpropagation. chapter Backpropagation: The Basic Theory, pages 1–34. L. Erlbaum Associates Inc., Hillsdale, NJ, USA.

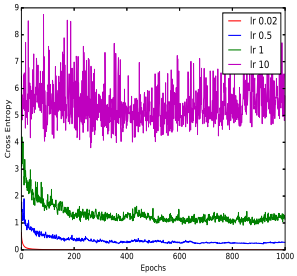
Sadowski, P. Notes on backpropagation.



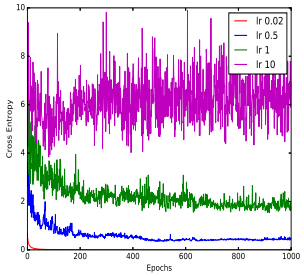
(a) MB10 - 25 neurônios na camada oculta



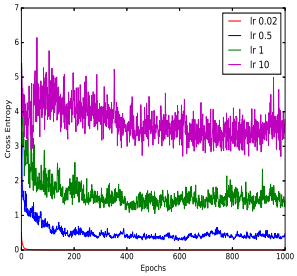
(b) MB50 - 25 neurônios na camada oculta



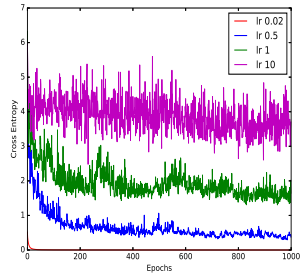
(c) MB10 - 50 neurônios na camada oculta



(d) MB50 - 50 neurônios na camada oculta



(e) MB10 - 100 neurônios na camada oculta



(f) MB50 - 100 neurônios na camada oculta

Figura 4. Mini batch - comparações entre batchs size de tamanho 10 e 50, variando learning rate

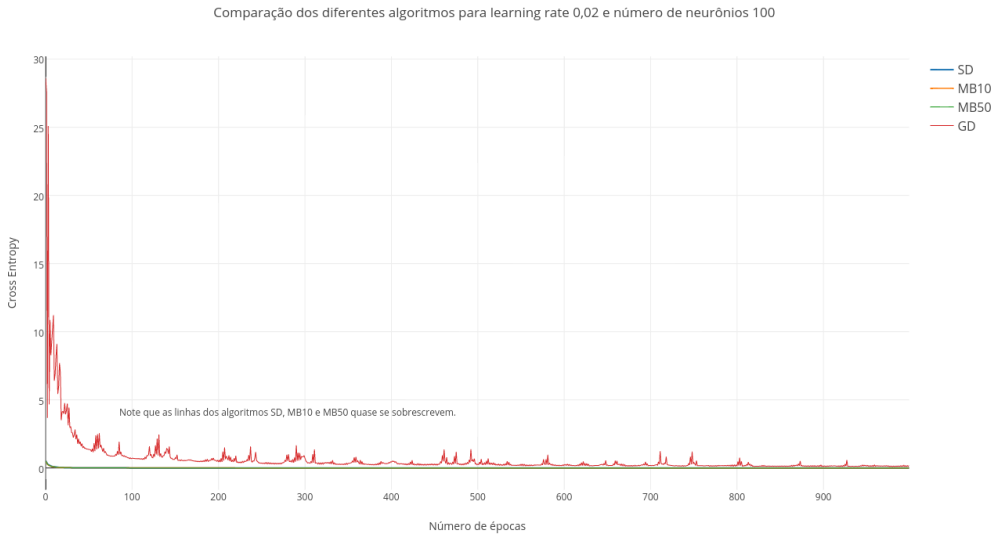


Figura 5. Comparação dos algoritmos variando o tamanho dos batchs