Trabalho prático 1
Aprendizado de Máquina

Gustavo Penha
DCC, UFMG
guzpenha@dcc.ufmg.br

1. IMPLEMENTAÇÃO DA REDE NEURAL

Nesta seção serão descritos a arquitetura da rede implementada e o cálculo do gradiente utilizado nos algoritmos iterativos.

1.1 Arquitetura da rede

A arquitetura da rede neural deste trabalho prático é composta por três camadas (contando com a camada de entrada). A função de ativação é a Sigmóide e como os dados são do *dataset* de reconhecimento de digitos do MNIST temos 784 (imagens 28 x 28) dimensões de entrada e a saída é composta por 10 unidades, representando a chance da entrada de ser de cada digito. A função de perda a ser minimizada é a Entropia Cruzada, Equação 1.

$$L(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) - (1-y_k^i)\log(1-(h_\theta(x^{(i)}))_k)] \tag{1}$$

Um desenho dessa arquitetura pode ser observado na Figura 1, sendo que $t = 10$. Podemos classificar essa rede como uma *Feed Foward Network* ou *Multi-Layer Perceptron*.

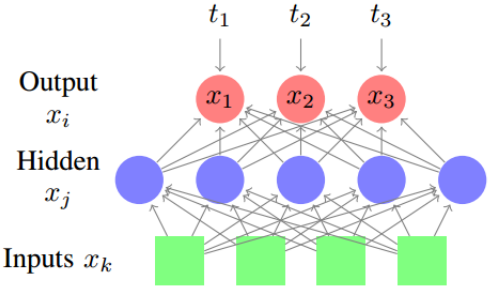

Figure 1: Ilustração da arquitetura da rede implementada.

1.2 Cálculo do gradiente

Para realizar o algoritmo de *backpropagation* é necessário calcular as derivadas parciais da Equação 1 (*Loss function*) em relação aos pesos das duas camadas.

Utilizando a regra da cadeia, chegamos que a derivada da função de custo em relação aos pesos da última camada é a Equação 2.

$$\frac{\partial E}{\partial w_{ji}} = (x_i - t_i)x_j \tag{2}$$

Já para a camada mais próxima da entrada, a derivada da função de perda em relação aos pesos é a Equação 3. A prova das duas derivadas podem ser encontrada em [1].

$$\frac{\partial E}{\partial w_{kj}} = (x_i - t_i)w_j(x_j(1-x_j)(x_k) \tag{3}$$

Estas foram as equações utilizadas para realizar o *back-propagation* implementado no trabalho prático.

2. AVALIAÇÃO EXPERIMENTAL

Nesta seção os resultados da avaliação experimental do trabalho prático são descritos.

2.1 Sensitividade em relação aos hiper-parâmetros

Um resumo dos resultados pode ser observado na Fig. 2.

2.1.1 Método do cálculo do gradiente

Os métodos de cálculo do gradiente avaliados foram:

- **GD**: A estimativa do gradiente é calculada utilizando todos os dados.
- **MB-GD**: A estimativa do gradiente é calculada utilizando *batches* de tamanho *batch_size*.
- **SGD**: A estimativa do gradiente é calculada utilizando apenas uma entrada.

Vemos que após um número de iterações igual para os três algoritmos na Figura 3, o que apresenta o menor erro é o GD, seguido do MB-GD e por fim o SGD. Esse resultado é o esperado, já que a melhor estimativa do gradiente é o GD, seguido do MB-GD e por fim o SGD.

2.1.2 Número de unidades ocultas

O modelo apresenta resultados melhores para um número de camadas maior apenas para o método GD. Já para os outros métodos de cálculo do gradiente isso nem sempre acontece, como observamos na Figura 2. Se olharmos os resultados por época, em outro resumo

2.1.3 Taxa de aprendizado

Avaliando os resultados que foram resumidos na Figura 2 vemos que uma taxa de aprendizado pequena melhora os resultados dos métodos, enquanto valores grandes como 10 fazem com que o erro varie muito, não chegando a convergir para um valor pequeno.

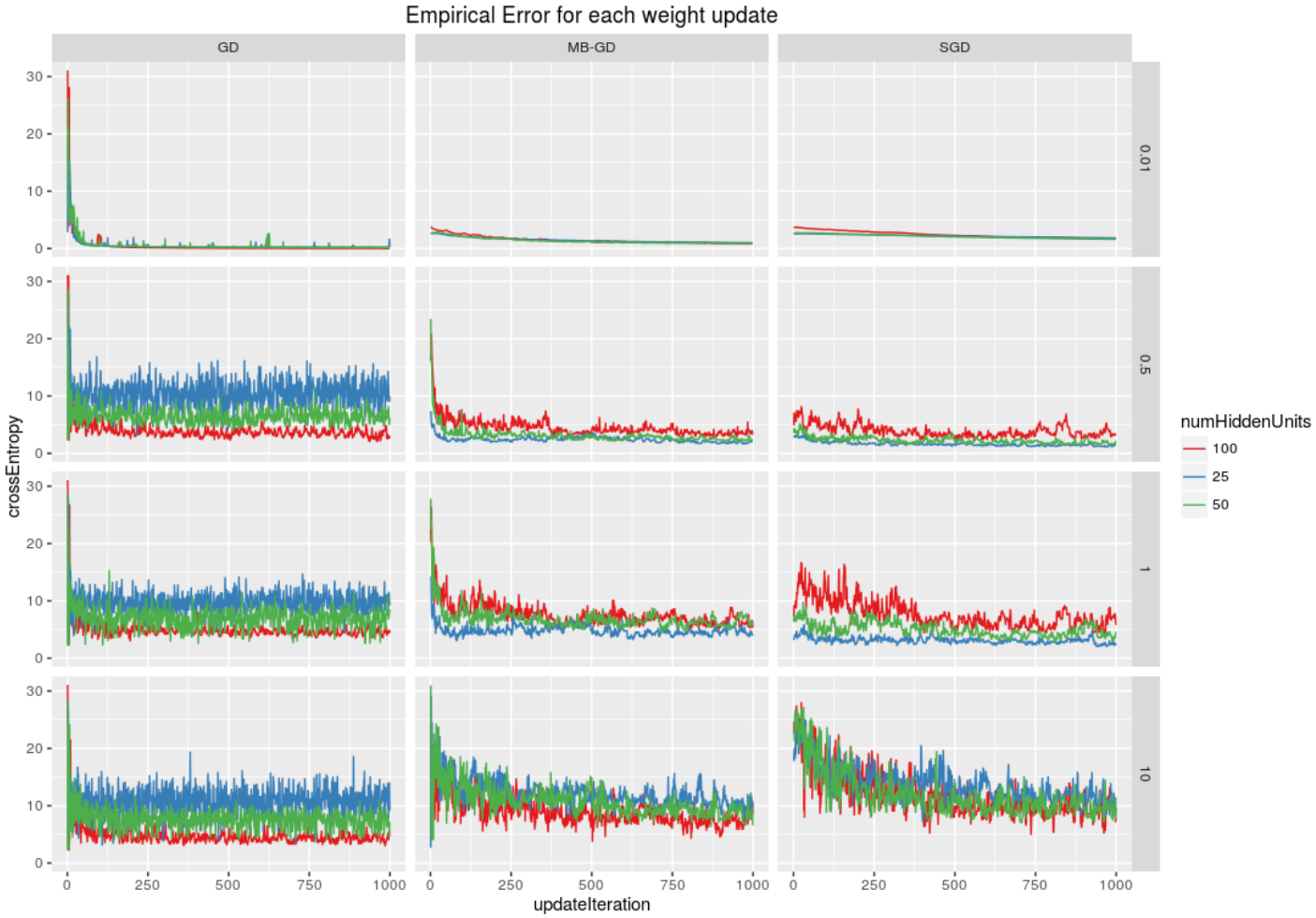

Figure 2: Resumo dos resultados experimentais, observando a convergência do erro empírico ao variar a taxa de aprendizado (linhas horizontais), o método de cálculo do gradiente (colunas) e o número de *hidden units* (cores).

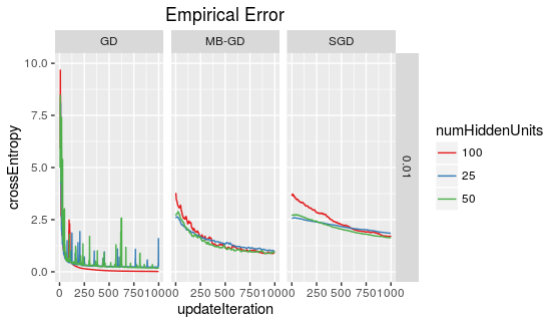

Figure 3: Comparação da convergência dos métodos de cálculo de gradiente.

2.1.4 Tamanho do batch

Observando a Figura 4 vemos que utilizando um tamanho de *batch* menor (10) converge mais rapidamente que um de tamanho 50. Isso pode ser explicado pela quantidade de atualizações nos pesos que são realizadas por tamanho de *batch*. Para o tamanho 50, realizamos $5000/50 = 100$ atualizações por época, já para o tamanho 10 realizamos 1000 atualizações de pesos por época.

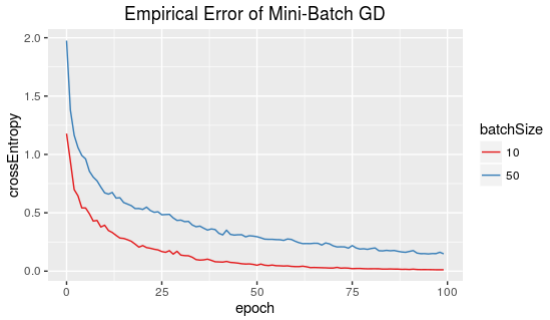

Figure 4: Comparação da convergência variando o tamanho do *batch*.

3. REFERENCES

[1] P. Sadowski. Notes on backpropagation, 2016.