TP1 - Machine Learning
Redes Neurais e Backpropagation

Felipe Monfardini Giori França

1 Introdução

Neste trabalho foi aplicado redes neurais para reconhecimento de dígitos escritos à mão. Os dados consistem de 5000 dígitos, onde cada dígito é uma imagem de 28 pixels por 28 pixels.
Uma rede neural foi treinada usando feedforward, backpropagation e três diferentes tipos de algoritmo de descida de gradiente. Também foram adotados diferentes valores para o número de unidades na camada oculta e diferentes valores para a taxa de aprendizado. Ao longo do trabalho serão discutidos os resultados dos experimentos.

2 Estrutura dos dados

Os dados com os exemplos foram armazenados em uma matriz X de dimensões 5000x784, onde cada linha corresponde a um exemplo do treino (imagem) e cada coluna corresponde a um pixel da imagem.

$$X = \begin{bmatrix} -(x^{(1)})- \\ -(x^{(2)})- \\ \cdots \\ -(x^{(m)})- \end{bmatrix}$$

Os *labels* dos exemplos foram armazenados em um vetor y de tamanho 5000, onde cada linha corresponde ao *label* do exemplo, ou seja, o número correspondente àquela imagem.

$$y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ ... \\ y^{(m)} \end{bmatrix}$$

As matrizes de pesos *theta1* e *theta2* têm dimensões mxn, onde m corresponde ao número de unidades na próxima camada e n ao número de unidades na camada atual contando com o termo *bias*.

3 Feedforward e Backpropagation

O custo da rede neural foi calculado através da função

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

No backpropagation os pesos são iniciados de forma aleatória e é calculado o erro de cada unidade e o gradiente dos pesos. Os gradientes são armazenados em matrizes *theta1grad* e *theta2grad*, com dimensões iguais a *theta1* e *theta2*. O passo de atualizar os pesos ficou a cargo dos algoritmos de descida de gradiente.

4 Experimentos

Os algoritmos de descida de gradiente utilizaram as matrizes de gradientes calculadas pelo *backpropagation* para fazer as atualizações dos pesos. Nos testes realizados, foram utilizados três valores para a taxa de aprendizado (0.5, 1, 10) e para o número de unidades na camada oculta (25, 50, 100).

4.1 Gradient descent (GD)

No *Gradient descent*, a atualização dos pesos foi feita depois que o erro de todos os 5000 exemplos foi calculado e acumulado. Os gráficos a seguir mostram o erro de treino do GD para os diferentes tamanhos de camada oculta e taxa de aprendizado. Foram feitas 1500 iterações, onde para cada iteração o peso foi alterado uma vez.

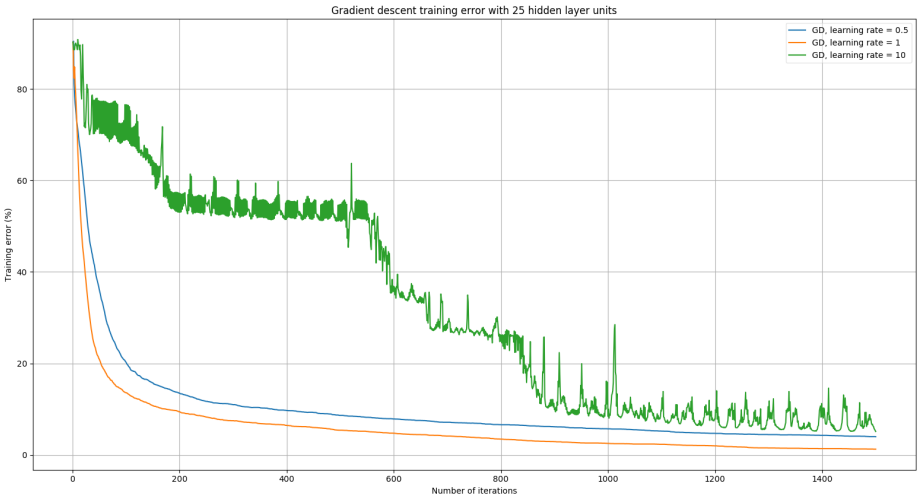

Figure 1: Gradient descent training error with 25 hidden layer units

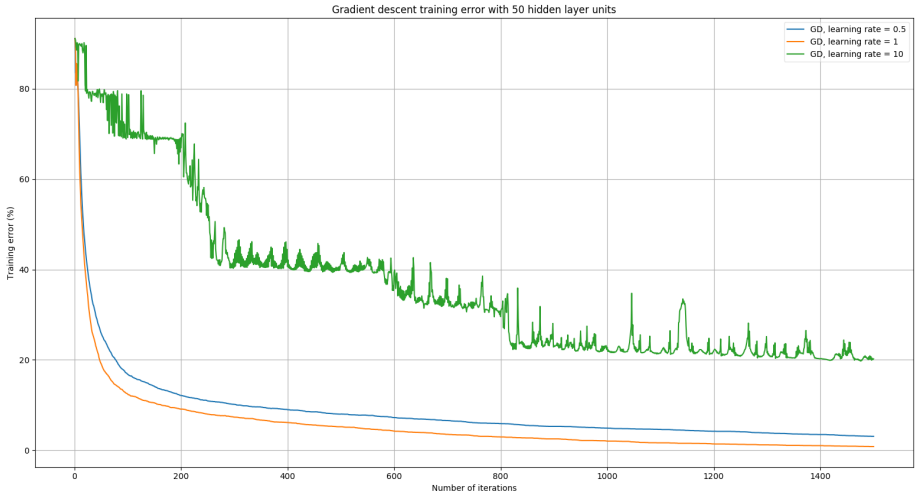

Figure 2: Gradient descent training error with 50 hidden layer units

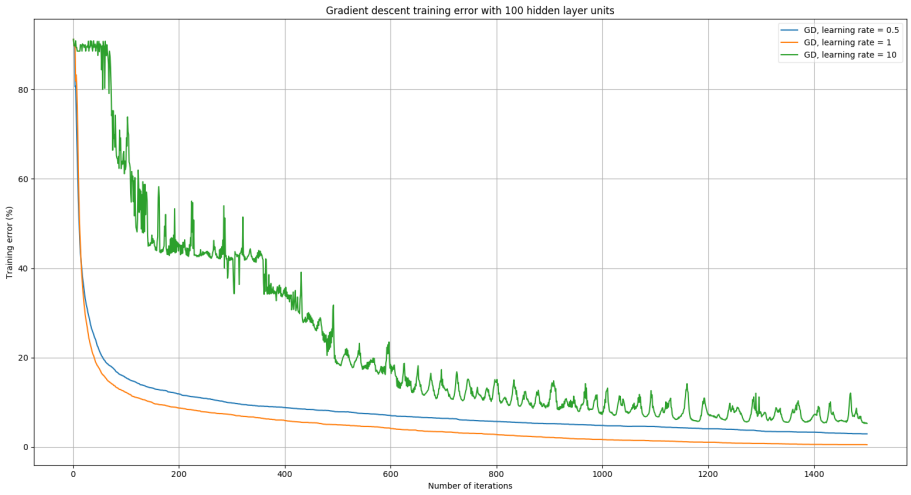

Figure 3: Gradient descent training error with 100 hidden layer units

4.1.1 Análise do Gradient Descent

Training set error			
Hidden layer units	learning rate = 0.5	learning rate = 1	learning rate = 10
25	4.02%	1.28%	5.14%
50	3.06%	0.8%	20.22%
100	2.94%	0.52%	5.28%

Para as taxas de aprendizado iguais a 0.5 e 1, é possível perceber que o aumento do número de unidades da camada oculta ajuda a diminuir o erro, mas o mesmo não pode ser dito para a taxa de aprendizado = 10. Isso ocorre porque taxas de aprendizado mais altas fazem mudanças mais agressivas nos pesos, sendo possível até mesmo a divergência. Apesar disso não ter ocorrido neste experimento, é possível perceber como o erro é instável ao longo do treino.

O caso em que a taxa de aprendizado = 10 e a rede neural possui 50 unidades na camada oculta apresenta um erro muito maior do que nos outros casos. Isso se deve à possibilidade do *gradient descent* ter encontrado um mínimo local que não é muito bom.

Para esse experimento, a taxa de aprendizado = 1 apresentou os melhores resultados quando comparado a 0.5, pois taxas de aprendizado menores tendem

a convergir mais lentamente.

4.2 Stochastic Gradient Descent

No *Stochastic Gradient Descent* (SGD), os pesos da rede foram atualizados a cada calculo de erro de um exemplo, isto é, para cada exemplo era feito e *feed forward* e o *backpropagation*, e os gradientes calculados eram imediatamente utilizados para atualizar os pesos. O próximo exemplo $x^{(i+1)}$ era executado com a nova matriz de pesos.

Na análise do SGD, o erro da rede foi calculado a cada 1000 atualizações de peso, ou seja, uma "época" corresponde a 1000 exemplos (imagens) lidos e calculados e, consequentemente, 1000 atualizações nos pesos. Em termos de tempo, cinco épocas do SGD correspondem a uma iteração do GD.

Os gráficos a seguir mostram o erro de treino do SGD para os diferentes tamanhos de camada oculta e taxa de aprendizado.

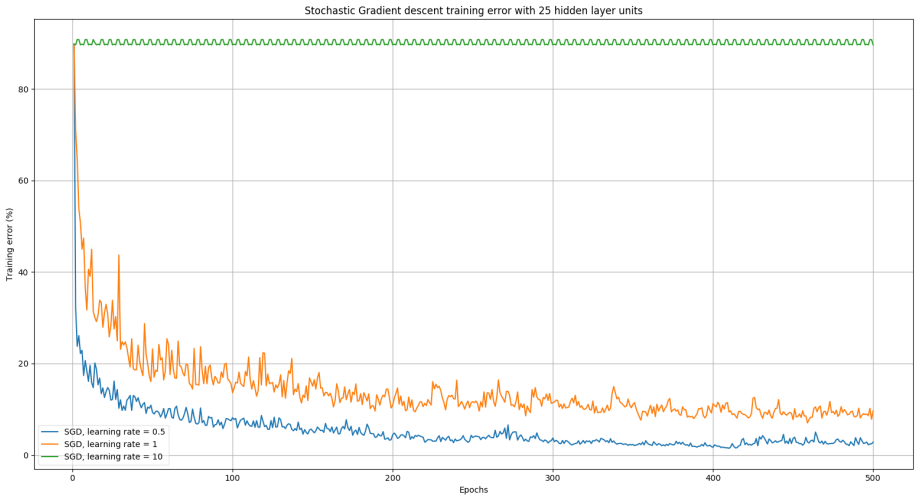

Figure 4: Stochastic gradient descent training error with 25 hidden layer units

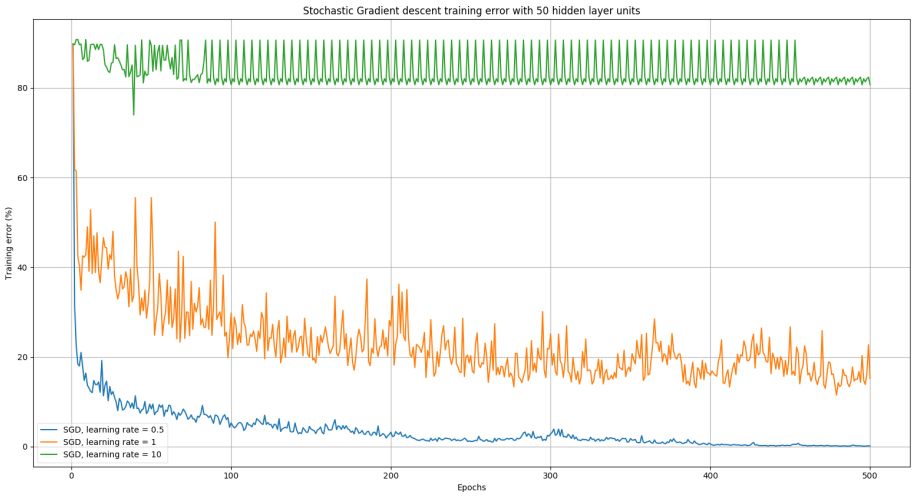

Figure 5: Stochastic gradient descent training error with 50 hidden layer units

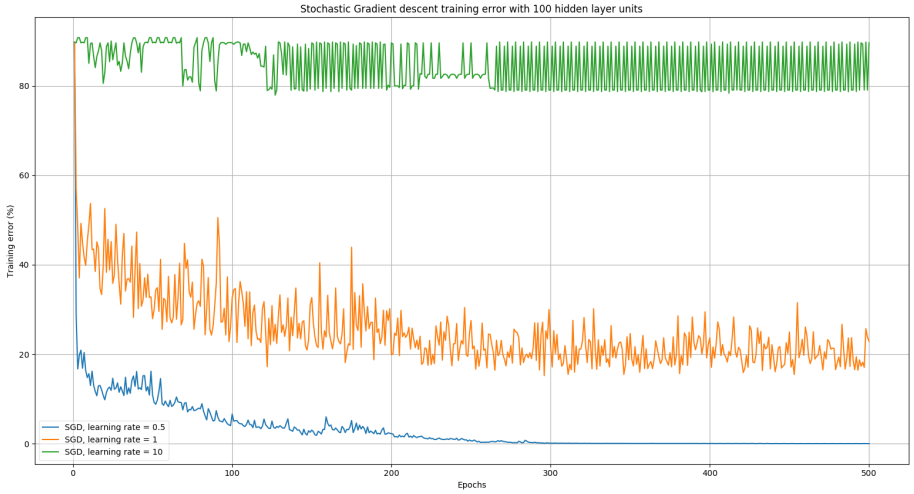

Figure 6: Stochastic gradient descent training error with 100 hidden layer units

4.2.1 Análise do Stochastic Gradient Descent

Training set error			
Hidden layer units	learning rate = 0.5	learning rate = 1	learning rate = 10
25	2.86%	9.68%	89.6%
50	0.14%	15.22%	80.68%
100	0.02%	22.88%	89.78%

Diferentemente do GD, nestes experimentos, a taxa de aprendizado = 0.5 teve resultados muito melhores que as outras e foi a única que teve o erro reduzido com o aumento do número de unidades na camada oculta, porque com mais neurônios na camada oculta, é possível expressar funções mais complexas que conseguem acomodar melhor as entradas.

Entretanto, para os outros casos, não houve redução do erro com o aumento do número de unidades na camada oculta. Isso se deve por causa da taxa de aprendizado alta. Isto é, como com o aumento de unidades da camada oculta gera-se funções mais complexas, essas são mais sensíveis à mundaças e, como o peso é alterado a cada exemplo $x^{(i)}$ em valores relativamente altos, ela acaba ficando instável. Nos gráficos é possível visualizar como os valores do erro são mais instáveis com o aumento do número de unidades na camada oculta.

4.3 Mini-batch Gradient Descent

No *Mini-batch Gradient Descent*, os erros e os gradientes são calculados e acumulados em intervalos de tamanho 10 e 50. Chamando de n o tamanho do intervalo, a cada n exemplos $(x^{(i)}, x^{(i+1)}, x^{(i+2)}, ..., x^{(i+n)})$ é calculado a média dos gradientes e os pesos são atualizados com essa média. Os próximos n exemplos têm seus erros e gradientes calculados com os novos pesos.

Na análise do *Mini-batch Gradient Descent*, o erro de treino da rede é calculado a cada 1000 exemplos lidos, que corresponde a uma época no gráfico. Similarmente ao SGD, cada iteração do GD corresponde a cinco épocas do *Mini-batch GD*.

Os gráficos a seguir mostram o erro de treino do *Mini-batch GD* para os diferentes tamanhos de camada oculta, taxa de aprendizado e tamanho de intervalo.

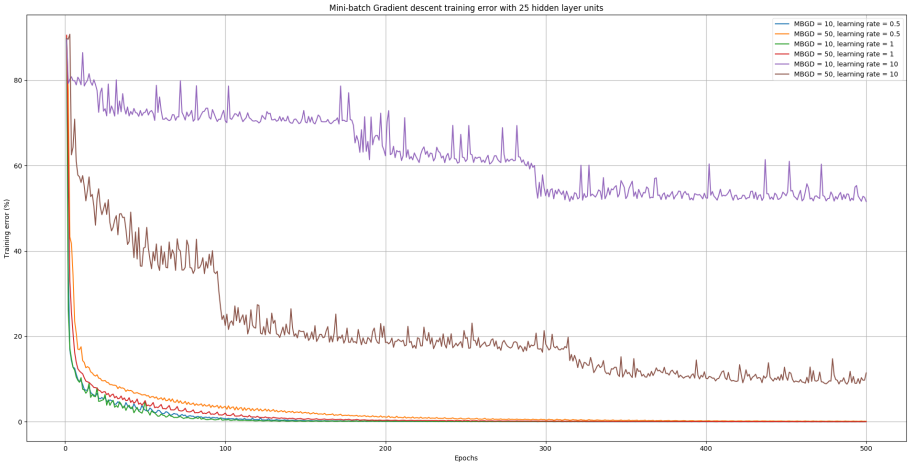

Figure 7: Mini-batch gradient descent training error with 25 hidden layer units

Figure 8: Mini-batch gradient descent training error with 50 hidden layer units

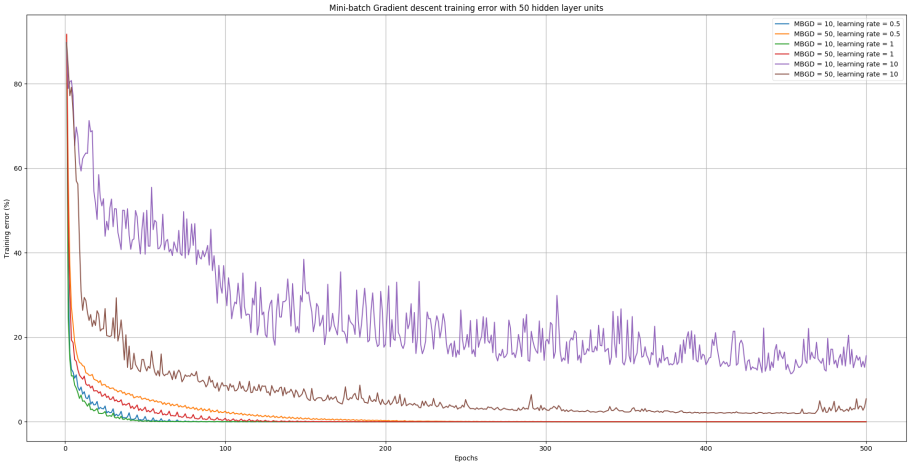

Figure 9: Mini-batch gradient descent training error with 100 hidden layer units

4.3.1 Análise do Mini-batch Gradient Descent

Training set error			
Hidden layer units / batch size	learning rate = 0.5	learning rate = 1	learning rate = 10
25 / 10	0.0%	0.02%	51.58%
25 / 50	0.1%	0.02%	11.42%
50 / 10	0.0%	0.0%	29.46%
50 / 50	0.02%	0.0%	5.54%
100 / 10	0.0%	0.0%	15.6%
100 / 50	0.0%	0.0%	3.5%

Para os casos em que a taxa de aprendizado é igual a 0.5 ou 1, o erro do treino foi muito baixo, sendo igual a zero na maioria dos casos. Isso acontece porque o *Mini-batch* GD funciona como um meio termo entre o SGD e o GD, onde não é necessário atualizar o peso a cada exemplo, mas também não é necessário calcular o erro médio de todos os casos. Por causa disso, as atualizações que acontecem nos pesos, apesar de serem menos frequentes do que no SGD, são mais certeiras, pois tira-se a média do erro de alguns casos, o que é suficiente para melhorar bastante a precisão do treino. Entretanto, é importante lembrar

que, mesmo que o erro do treino esteja baixo, a função não necessariamente vai generalizar bem para exemplos que não foram vistos (overfitting).

Um outro resultado obtido que mostra a melhora significativa com o *Mini-batch* é quando comparamos os resultados para taxa de aprendizado igual a 1 do *Mini-batch* e do SGD. No SGD foram obtidos erros maiores dos que os obtidos no *Mini-batch*, porque, como a mudança do *Mini-batch* tende a ser melhor direcionada, então mesmo uma taxa de aprendizado um pouco mais alta acaba não causando tanta instabilidade.

Para a taxa de aprendizado igual a 10, é possível perceber como o aumento do *batch* diminuiu consideravelmente o erro do treino pelos mesmos motivos citados anteriormente, a mudança de pesos melhor direcionada.

5 Conclusão

No geral, foi possível perceber que taxas de aprendizado menores tendem a treinar redes com erros menores, porém demoram a convergir, e com o aumento do número de unidades na camada oculta é possível criar funções mais complexas que acomodam melhor os exemplos.