# [DeepLearnToolbox使用总结](http://blog.csdn.net/lifeitengup/article/details/10219075)

DeepLearnToolbox

A Matlab toolbox for Deep Learning.

Deep Learning is a new subfield of machine learning that focuses on learning deep hierarchical models of data. It is inspired by the human brain's apparent deep (layered, hierarchical) architecture. A good overview of the theory of Deep Learning theory is [Learning Deep Architectures for AI](http://www.iro.umontreal.ca/~bengioy/papers/ftml_book.pdf)

Directories included in the toolbox

NN/ - A library for Feedforward Backpropagation Neural Networks

CNN/ - A library for Convolutional Neural Networks

DBN/ - A library for Deep Belief Networks

SAE/ - A library for Stacked Auto-Encoders

CAE/ - A library for Convolutional Auto-Encoders

util/ - Utility functions used by the libraries

data/ - Data used by the examples

tests/ - unit tests to verify toolbox is working

For references on each library check REFS.md

Setup

1. Download.
2. addpath(genpath('DeepLearnToolbox'));

Windows下把文件夹加入 path 即可

**[plain]** [view plaincopy](http://blog.csdn.net/lifeitengup/article/details/10219075)

1. %LiFeiteng
3. path = pwd;
4. files = dir(path);
6. for i = 1:length(files)
8. if files(i).isdir
9. file = files(i).name;
10. addpath([path '/' file])
11. disp(['add ' file ' to path!'])
12. end
14. end

我不打算解析代码，想从代码里面学算法是stupid的；有相应的论文，readlist，talk等可以去学习。

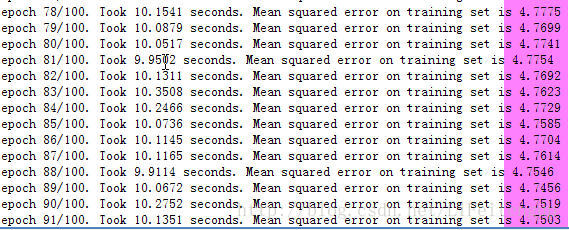
DeepLearnToolbox单隐藏层NN的优化策略：mini-Batch SGD

**[plain]** [view plaincopy](http://blog.csdn.net/lifeitengup/article/details/10219075)

1. function [nn, L]  = nntrain(nn, train\_x, train\_y, opts, val\_x, val\_y)
2. %NNTRAIN trains a neural net
3. % [nn, L] = nnff(nn, x, y, opts) trains the neural network nn with input x and
4. % output y for opts.numepochs epochs, with minibatches of size
5. % opts.batchsize. Returns a neural network nn with updated activations,
6. % errors, weights and biases, (nn.a, nn.e, nn.W, nn.b) and L, the sum
7. % squared error for each training minibatch.
9. assert(isfloat(train\_x), 'train\_x must be a float');
10. assert(nargin == 4 || nargin == 6,'number ofinput arguments must be 4 or 6')
12. loss.train.e               = [];
13. loss.train.e\_frac          = [];
14. loss.val.e                 = [];
15. loss.val.e\_frac            = [];
16. opts.validation = 0;
17. if nargin == 6
18. opts.validation = 1;
19. end
21. fhandle = [];
22. if isfield(opts,'plot') && opts.plot == 1
23. fhandle = figure();
24. end
26. m = size(train\_x, 1);
28. batchsize = opts.batchsize;
29. numepochs = opts.numepochs;
31. numbatches = m / batchsize;
33. assert(rem(numbatches, 1) == 0, 'numbatches must be a integer');
35. L = zeros(numepochs\*numbatches,1);
36. n = 1;
37. for i = 1 : numepochs
38. tic;
40. kk = randperm(m);
41. for l = 1 : numbatches
42. batch\_x = train\_x(kk((l - 1) \* batchsize + 1 : l \* batchsize), :);
44. %Add noise to input (for use in denoising autoencoder)
45. if(nn.inputZeroMaskedFraction ~= 0)
46. batch\_x = batch\_x.\*(rand(size(batch\_x))>nn.inputZeroMaskedFraction);
47. end
49. batch\_y = train\_y(kk((l - 1) \* batchsize + 1 : l \* batchsize), :);
51. nn = nnff(nn, batch\_x, batch\_y);
52. nn = nnbp(nn);
53. nn = nnapplygrads(nn);
55. L(n) = nn.L;
57. n = n + 1;
58. end
60. t = toc;
62. if ishandle(fhandle)
63. if opts.validation == 1
64. loss = nneval(nn, loss, train\_x, train\_y, val\_x, val\_y);
65. else
66. loss = nneval(nn, loss, train\_x, train\_y);
67. end
68. nnupdatefigures(nn, fhandle, loss, opts, i);
69. end
71. disp(['epoch ' num2str(i) '/' num2str(opts.numepochs) '. Took ' num2str(t) ' seconds' '. Mean squared error on training set is ' num2str(mean(L((n-numbatches):(n-1))))]);
72. nn.learningRate = nn.learningRate \* nn.scaling\_learningRate;
73. end
74. end

1.不管是在 nntrain、nnbp还是nnapplygrads中我都没看到 对算法收敛性的判断，

而且在实测的过程中 有观察到 epoch过程中 mean-squared-error有 下降-上升-下降 的走势——微小抖动在SGD中 算是正常



多数还都是在下降（epoch我一般设为 10-40，这个值可能偏小；Hinton 06 science的文章代码记得epoch了200次，我跑了3天也没跑完）

在SAE/CNN等中 也没看到收敛性的判断。

2.CAE  没有完成

3.dropout的优化策略也可以选择

我测试了 SAE CNN等，多几次epoch(20-30)，在MNIST上正确率在 97%+的样子。

其实cost-function 可以有不同的选择，如果使用 UFLDL的优化方式(固定的优化方法，传入cost-function的函数句柄)，在更改cost-function上会更自由。

可以改进的地方：

1. mini-Bathch SGD算法 增加收敛性判断

2.增加 L-BFGS/CG等优化算法

3.完善CAE等

4.增加min KL-熵的 Sparse Autoencoder等

5.优化算法增加对 不同cost-function的支持

# [Deep Learning 优化小结](http://blog.csdn.net/lifeitengup/article/details/10212867)

胡乱写一点，做个记录；不保证可读性。

Hinton 06 年的 A fast learning algorithm for deep belief nets首先提出了 pre-training + fine-tuning的优化 deep network的方法策略；

mutil-layer network多年前就提出，但优化一直是个问题；非凸且参数巨多， 单隐藏层NN的优化算法Backpropagation Algorith往往得不到较好的局部最优解；

pre-training 往往能够得到比较接近局部最优解的初始迭代点，然后使用fine-tuning来微调pre-training得到的参数。

pre-training是一种策略，不是具体的优化方法。pre-training使用无监督学习来训练一个单隐藏层NN，输入输出相同。从而学习到参数，然后保留第一层；递归进行多次。最后使用 有监督学习来trian一下 feature-label的参数，可以使用sigmoid/softmax regression等。

单隐藏层NN Autoencoder/Sparse Autoencoder等；另 RBM等。

RBM 用Hinton 02年的一篇文章提出的方法来trian 基本思想是 极大似然的时候不断抬高下界 做个两三层即可

单隐藏层NN 的训练方法有 梯度下降(GD)/随机梯度下降(SGD)/mini-Batch GD/L-BFGS/共轭梯度(CG)

其实单隐藏层 一般不需要这么多方法， 但是Deep Learning往往需要 大量的训练数据 才能刻画数据的分布，所以需要 优化技巧来提高训练速度。

这里有许多参数要调  比如Sparse Autoencoder如果使用KL-熵来做优化的目标函数 就需要选择 优化比例lambda等

GD/SGD中需要调节 learning rate 尤其是SGD；mini-Batch除步长外 还需调节 batchsize epoch等参数/L-BFGS 如果内存占用太多 还需调节选用个数的参数。

此外优化 还有添加 动量 or 平均随机梯度的方法等。

fine-tuning也是用 SGD/LBFGS/CG来微调各层参数，但是这种微调 其实对底层的参数影响比较小，多半调在了最上层的参数上；不同的task 会带来不同的微调效果 尽管他们使用相同的pre-training

近来，Hinton 还提出 使用 dropout的技术来trian，有点 Denoise Autoencoder的意思；Bengio顺带搞出来 maxout等技术。

Ng组 11年 有篇文章——On Optimization Methods for Deep Learning分析了SGD/L-BFGS/CG

华盛顿大学的 Pedro-Domingos等 提出的 Sum-Product Networks的优化 完全不是上面套路，reuse很多中间结果，据说训练很快 没细看

关于优化等问题 可以参考    部分章节

# [Neural Networks: Tricks of the Trade](http://link.springer.com/book/10.1007/978-3-642-35289-8/page/1)

Second Edition

Deep Learning在图像上 为了克服图像尺寸、平移、旋转等的问题会使用 patch/convolution-pooling等技术

早期Deep learning 基本用在了图像/语音的分类问题和提取特征上，近年有拓展应用领域的趋势：图像分割/图像检索等。

我也用 Stacked （Sparse） Autoencoder/Convolution NN等 在MNIST上做了一些实验，基本在 97.5%+的正确率上，更高的准确率，可以通过增加训练参数(旋转图像/平移图像等)、数据的预处理(白化等)、更细致的调参数、更换模型等来提高。