Report on UFLDL-Part III

Yunfei WANG

[1]School of Computer Science & Technology
Huazhong University of Science & Technology

June 25th, 2013

Table of contents

Advantages of Deep Networks

Deep Networks have significantly greater representational power than a shallow one.

Learn part-whole decompositions:

- *First Layer*: Group together pixels to detect edges
- *Second Layer*: Group together edges to detect contours
- *Third Layer*: Group together contours to detect more objects.

Note: It's important to use on-linear activation function in hidden layers.(Because multiple layers of linear functions would compute only a linear function of the input)

Difficulty of Training Deep Architectures

Traditional strategy of training deep architectures

- Randomly initialize weights of a deep network
- Train the network on a labeled set with a supervised learning objective

1. Availability of data
   The method above only relies on labeled data for training.
   Difficult to obtain enough labeled examples to fit model parameters.
   Training on insufficient data would result in overfitting.
2. Local Optima
   Minimizing $\sum_i \|h_W(x^i) - y^i\|^2$ in a deep network turn out to be rife with bad local optima.Training with gradient descent,conjugate gradient or L-BFGS no longer work well.
3. Diffusion of Gradients
   When using backpropagation to compute derivatives,gradients propagated backwards rapidly diminish in magnitude as the depth of network increases.Weights of earlier layers change slowly.

Greedy Layer-wise Training

Main idea of greedy layer-wise training:

- Step 1:Train the layers of network one after another(supervised or unsupervised);
- Step 2:Weights from training layers individually are used to initialize deep network
- Step 3:Fine-tuning entire architecture

What does greedy layer-wise training bring to us?

- Availability of data
  Massive unlabeled data is used to learn and discover patterns by self-taught learning.Compared with purely supervised approaches,this brings us much better classifiers.
- Better local optima
  Unlabeled data provides a significant amount of "prior" information about patterns of input data.Weights start at a better location in parameter space after pre-training than randomly initialization.

Stacked Autoencoders I

First, train a sparse autoencoder on raw inputs x^k to learn primary features $h^{1,k}$

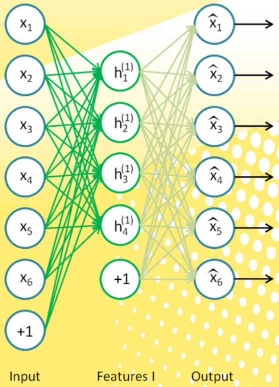


Figure: 1st-layer Autoencoder

Stacked Autoencoders II

Second, train a sparse autoencoder on primary features $h^{1,k}$ to learn second features $h^{2,k}$

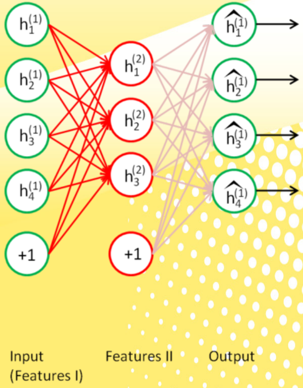


Figure: 2nd-layer Autoencoder

Stacked Autoencoders III

Next, train a softmax classifier on the last features $h^{2,k}$ to map them to digit labels.

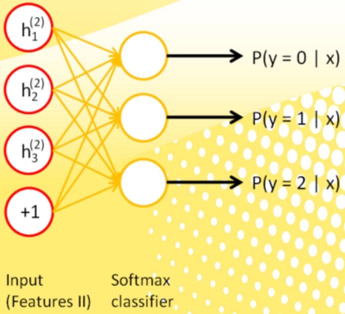
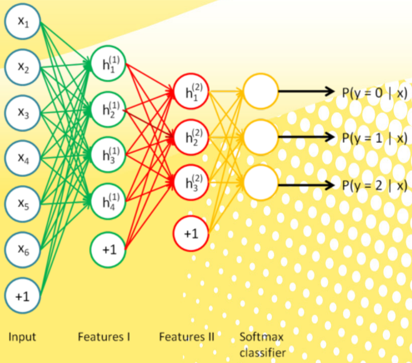


Figure: Softmax Classifier in last layer

Stacked Autoencoders IV

Finally, combine all layers to form a stacked autoencoder with 2 hidden layers and a final softmax classifier layer capable of classifying.



Fine-tuning Stacked AEs

Fine tuning treats all layers as a single model and tunes all weights in each iteration,greatly improving the performance of a stacked autoencoder.

```
1 Perform a feedforward pass to compute activations for all layers;
2 For output layer,set
      $$\delta^L = -(\nabla_{a^L} J) \bullet f'(z^L) \quad (1)$$
  for $l = L-1, L-2, \cdots, 2$ do
3     $$\delta^l = ((W^l)^T \delta^{l+1}) \bullet f'(z^l) \quad (2)$$
4 end
5 Computing partial derivations:
      $$\nabla_{W^l} J(W,b;x,y) = \delta^{l+1}(a^l)^T \quad (3)$$
      $$\nabla_{b^l} J(W,b;x,y) = \delta^{l+1} \quad (4)$$
```

Experiments on Stacked AEs

Dataset	Dimensionality	Number of Samples
Training Data	784	60000
Testing Data	784	10000

Table: MNIST

NUM of Autoencoders	Before Fine-tuning	After Fine-tuning
2(Time=81min)	ACC=87.83%	ACC=97.59%
3(Time=106min)	ACC=65.86%	ACC=97.35%
4(Time=118min)	ACC=11.35%	ACC=97.05%
5(Time=83min)	ACC=11.35%	ACC=11.35%
6(Time=86min)	ACC=11.35%	ACC=11.35%

Table: Experiment Results