Fine-tuning Stacked AEs

From Ufldl

Introduction

Fine tuning is a strategy that is commonly found in deep learning. As such, it can also be used to greatly improve the performance of a stacked autoencoder. From a high level perspective, fine tuning treats all layers of a stacked autoencoder as a single model, so that in one iteration, we are improving upon all the weights in the stacked autoencoder.

General Strategy

Fortunately, we already have all the tools necessary to implement fine tuning for stacked autoencoders! In order to compute the gradients for all the layers of the stacked autoencoder in each iteration, we use the Backpropagation Algorithm, as discussed in the sparse autoencoder section. As the backpropagation algorithm can be extended to apply for an arbitrary number of layers, we can actually use this algorithm on a stacked autoencoder of arbitrary depth.

Finetuning with Backpropagation

For your convenience, the summary of the backpropagation algorithm using element wise notation is below:

1. Perform a feedforward pass, computing the activations for layers L_2, L_3, up to the output layer L_{n_l}, using the equations defining the forward propagation steps.
2. For the output layer (layer n_l), set

   $$\delta^{(n_l)} = -(\nabla_{a^{n_l}} J) \bullet f'(z^{(n_l)})$$

   (When using softmax regression, the softmax layer has $\nabla J = \theta^T(I - P)$ where *I* is the input labels and *P* is the vector of conditional probabilities.)

3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$

   Set

   $$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \bullet f'(z^{(l)})$$

4. Compute the desired partial derivatives:

$$\nabla_{W^{(l)}} J(W, b; x, y) = \delta^{(l+1)} (a^{(l)})^T,$$
$$\nabla_{b^{(l)}} J(W, b; x, y) = \delta^{(l+1)}.$$

$$J(W, b) = \left[\frac{1}{m} \sum_{i=1}^{m} J(W, b; x^{(i)}, y^{(i)}) \right]$$

Note: While one could consider the softmax classifier as an additional layer, the derivation above does not. Specifically, we consider the "last layer" of the network to be the features that goes into the softmax classifier. Therefore, the derivatives (in Step 2) are computed using $\delta^{(n_l)} = -(\nabla_{a^{n_l}} J) \bullet f'(z^{(n_l)})$, where $\nabla J = \theta^T(I - P)$.

From Self-Taught Learning to Deep Networks | Deep Networks: Overview | Stacked Autoencoders | **Fine-tuning Stacked AEs** | Exercise: Implement deep networks for digit classification

Language : 中文