Project Proposal

An analysis of performance improvement in deep architectures via different pre-training schemes

Michael Downs

Motivation

- We have seen several pre-training approaches using stacked autoencoders
- Training different varieties of autoencoders, stacking them, and then fine-tuning
- Holding everything else fixed, which of these methods yields superior weight initialization?

Methods

- Use same network architecture throughout experiments
- For each type of pretraining technique
- Initialize weights using that technique
- Fine-tune resulting network keeping training methods (some combination of appropriate cost function, weight decay, momentum, etc.) consistent

Data

- MNIST
- Easy to work with
- Minimal preprocessing
- 60,000 training samples
- 10,000 test samples
- Split training set into 50,000 training data and 10,000 validation data

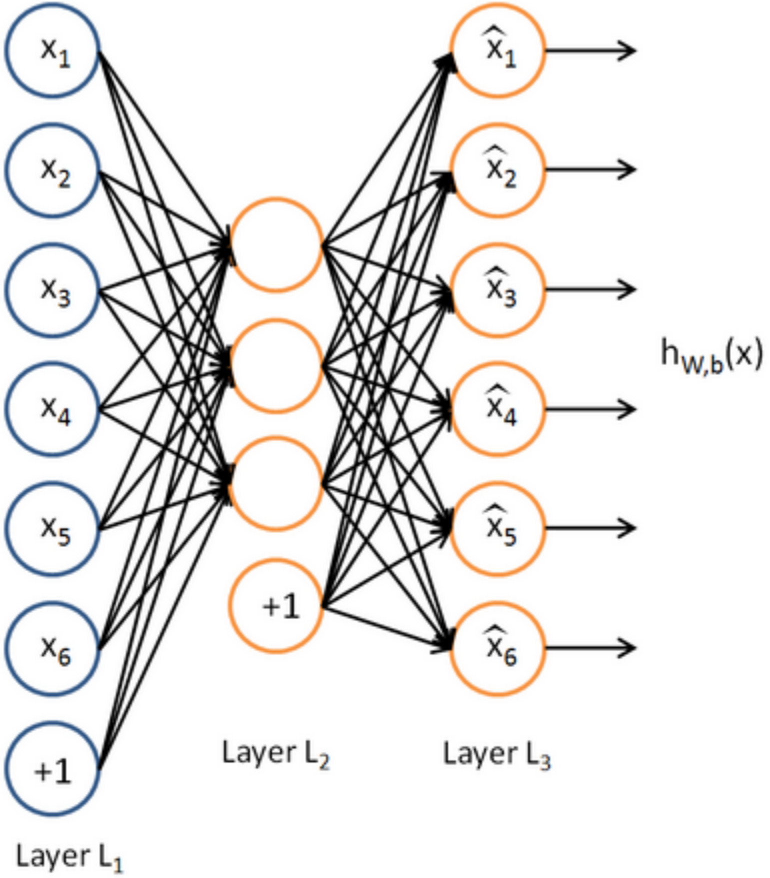
Tools

- Theano
- Symbolic construction of equations
- Expression optimization
- Automatic differentiation
- Use GPU to do calculations

(Plain) Autoencoder

$$\mathcal{D} = \{x^{(1)}, ..., x^{(m)}\}$$

$$y^{(i)} = x^{(i)}$$



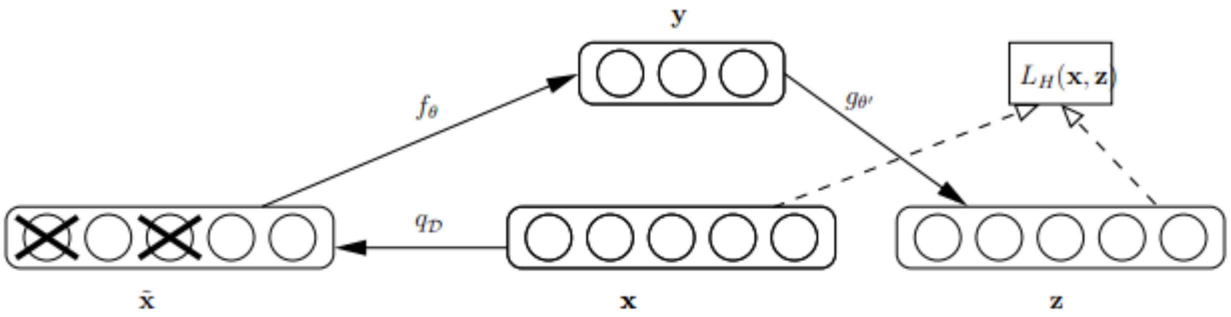
Sparse Autoencoder

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} \left(a_j^{(2)}(x^{(i)}) \right)$$

$$KL(\rho || \hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_j}$$

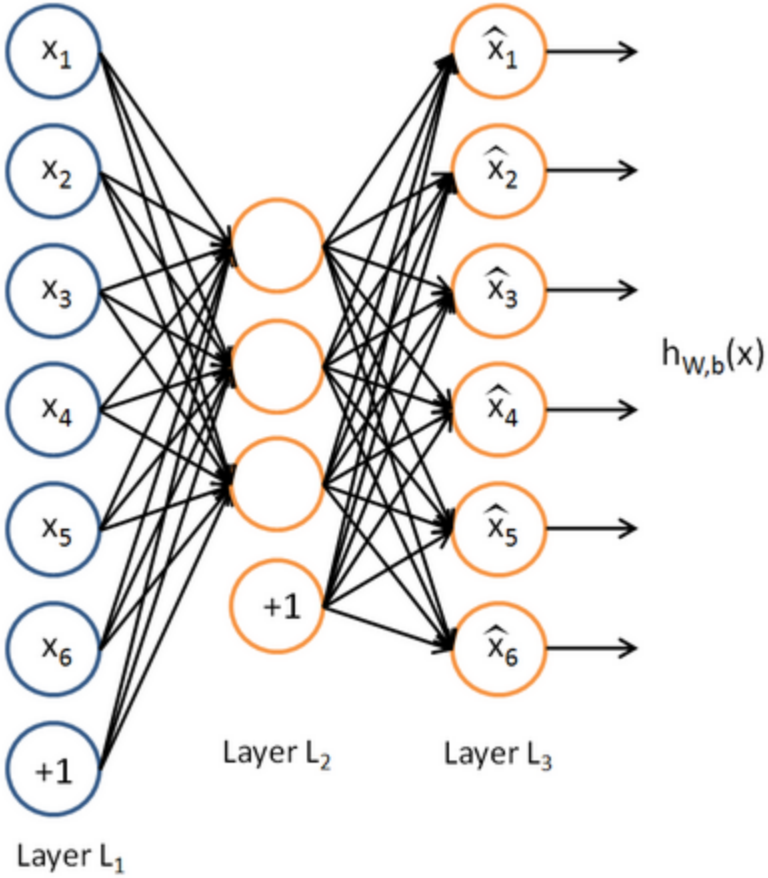
$$C_{Sparse}(W, b) = C(W, b) + \lambda \sum_{i=1}^{n_2} KL(\rho || \hat{\rho}_i)$$

Denoising Autoencoder



Contractive Autoencoder

$$C_{Contractive}(W,b) = C(W,b) + \lambda ||\nabla_{x^{(i)}} h(x^{(i)})||_F^2$$



Restrictive Autoencoder

- Idea motivated by a paper from Nando de Freitas
- Restricts the autoencoder to learning only a small amount of weights and constructing the rest from the learned weights

Restrictive Autoencoder

$$a^{(2)} = f\left(W^{(1)}a^{(1)} + b^{(1)}\right)$$

$$W^{(1)} \in \mathbb{R}^{n_2 \times n_1}$$

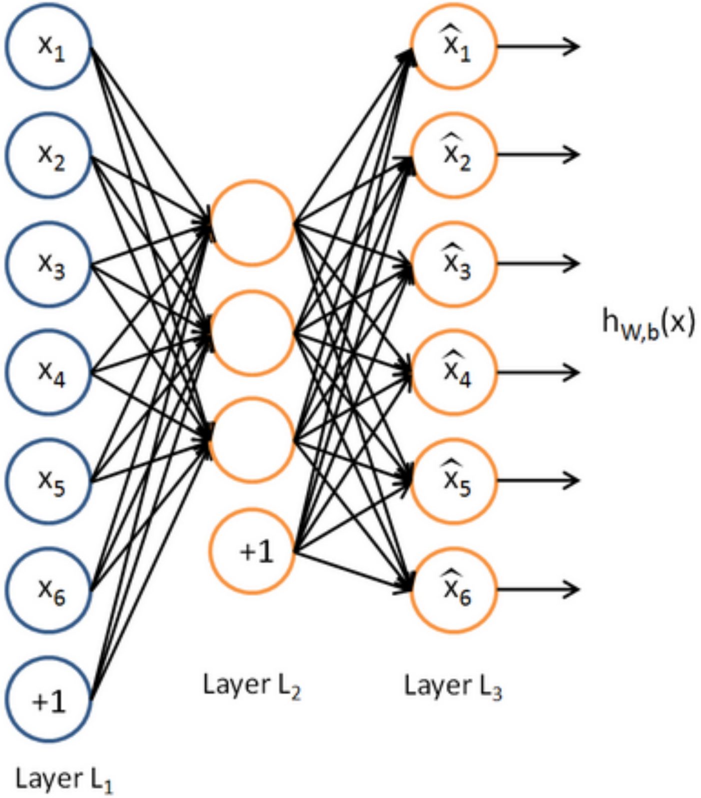
$$W^{(1)} = U^{(1)}V^{(1)}$$

$$U^{(1)} \in \mathbb{R}^{n_2 \times \alpha}$$

$$V^{(1)} \in \mathbb{R}^{\alpha \times n_1}$$

$$\alpha << n_1$$

$$\alpha << n_2$$



Restrictive Autoencoder

- Only want to learn one of the factors since for any invertible $Q \in \mathbb{R}^{\alpha \times \alpha}$ we have
  $UV = (UQ)(Q^{-1}V) = \tilde{U}\tilde{V}$
- Fix V and only learn U. Initialize V once using some heuristic depending on the data, perhaps PCA

Milestone Goal

- Hope to have Theano code working for a regular deep network as well as at least regular and sparse autoencoders