4.1 Data Augmentation

Training images. Specifically, we perform PCA on the set of RGB pixel values throughout the forced us to use much smaller networks. At test time, the network makes a prediction by extracting

tions. We do this by extracting random

GPU is training on the previous batch of images. So these data augmentation schemes are, in effect,

of data augmentation, both of which allow transformed images to be produced from the original

4 Reducing Overfitting

make each training example impose 10 bits of constraint on the mapping from image to label, this

pooling or normalization layers. The third convolutional layer has 384 kernels of size

neurons in a kernel map). The second convolutional layer takes as input the (response-normalized

the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–
The two papers in 2006


Three ways to avoid outfitting

- Penalize the weights by adding the norm of the weights in the network to the objective function
- Use dropout
- Use unlabeled data to train a different network and then use the weight to initialize our network
  - Deep belief networks (based on restricted Boltzmann machine)
  - Deep autoencoders (based on autoencoder)
Outline

- Points crossed
  - Generative model
  - Unsupervised
  - The research trend
  - The representation power
- Basic structure - linear autoencoder
  - Optimal solution vs. PCA vs. undercomplete autoencoder
- Regularized autoencoder
  - Sparse autoencoder vs. overcompleteness
  - Denoising autoencoder (DAE)

AE as initialization methods

- Pretraining step
  - Train a sequence of shallow autoencoders, greedily one layer at a time, using unsupervised data
- Fine-tuning step 1
  - Train the last layer using supervised data
- Fine-tuning step 2
  - Use backpropagation to fine-tune the entire network using supervised data
**General structure**

\[ y = f_{\theta_1}(W_1 x + b_1) \]
\[ z = g_{\theta_2}(W_2 y + b_2) \]
\[ \theta_1 = \{W_1, b_1\}, \theta_2 = \{W_2, b_2\} \]

\[ \theta_1^e, \theta_2^e = \arg \min_{\theta_1, \theta_2} \frac{1}{n} \sum_{i=1}^{n} L(x^{(i)}, z^{(i)}) = \arg \min_{\theta_1, \theta_2} \frac{1}{n} \sum_{i=1}^{n} L(x^{(i)}, g_{\theta_2}(f_{\theta_1}(x^{(i)}))) \]

\[ L(x, z) = \frac{1}{2} \|x - z\|_2^2 \]

\[ L_H(x, z) = H(B_x||B_z) = -\sum_{k=1}^{d} [x_k \log z_k + (1 - x_k) \log(1 - z_k)] \]

\[ \theta_1^*, \theta_2^* = \arg \min_{\theta_1, \theta_2} E_{q(X)}[L_H(X, g_{\theta_2}(f_{\theta_1}(X)))] \]

**Sparse autoencoder**

\[ L(x, z) = \|x - z\|_2^2 + \alpha_s h(y) + \alpha_r \|W_2\|_1 \]

http://www.ericwilkinson.com/blog/2014/11/19/deep-learning-sparse-autoencoders
Denoising autoencoder (DAE)

\[ \theta_1^*, \theta_2^* = \arg \min_{\theta_1, \theta_2} E_q(x, \tilde{x}) [L_H(X, g_{\theta_2}(f_{\theta_1}(\tilde{x})))] \]