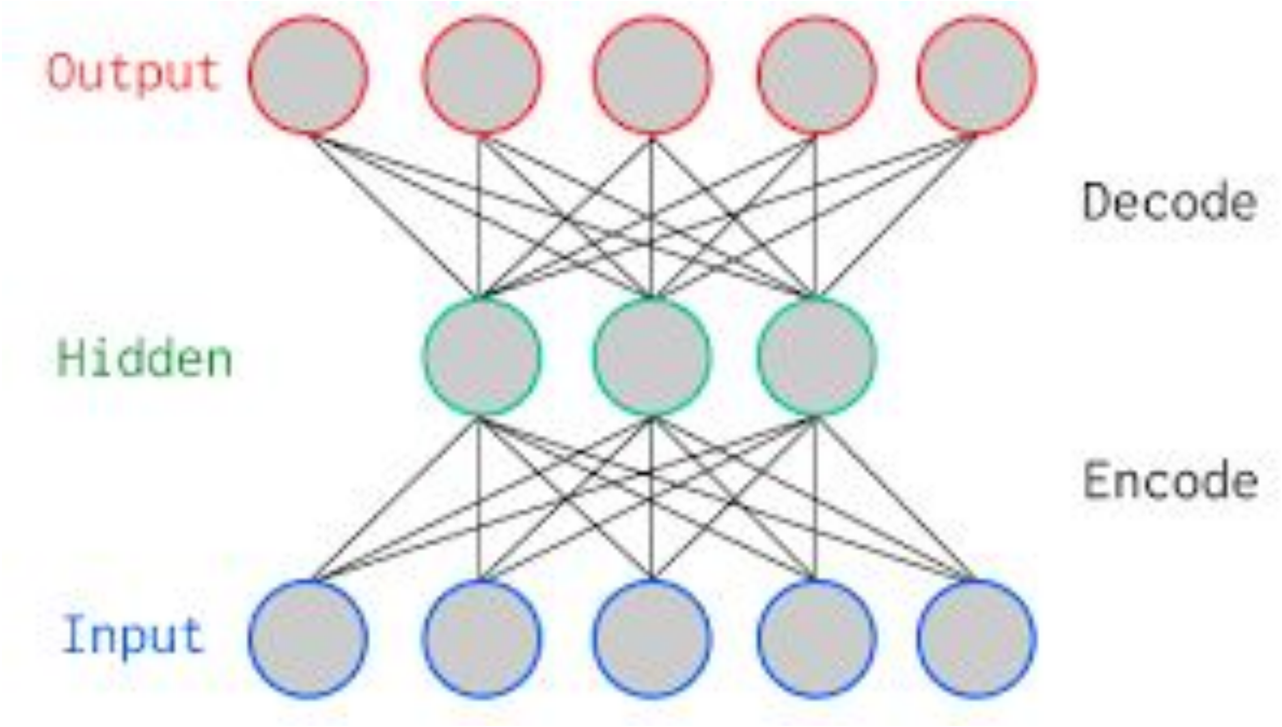
Sparse Feature Learning for Deep Belief Networks

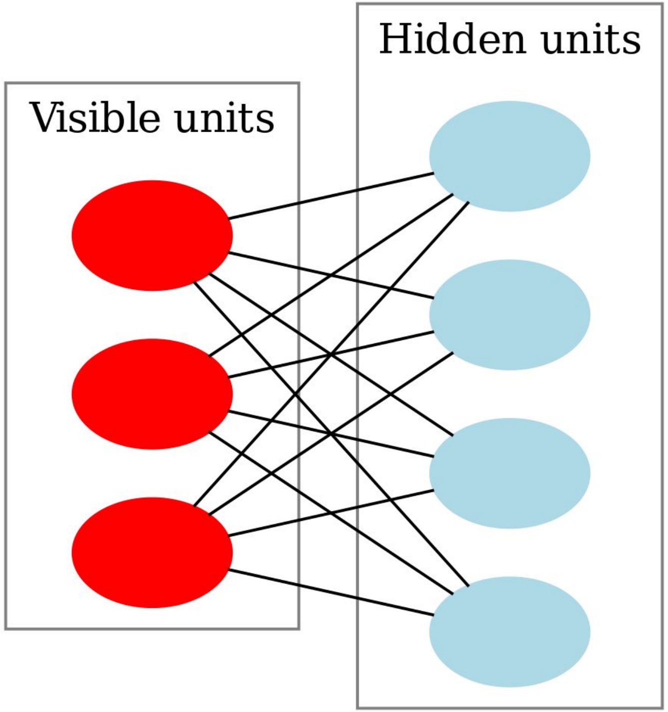
Marc' Aurelio Ranzato, Y-Lan Boureau, Yann LeCun

R250 Presentation By: Vikash Singh March 8, 2019

Brief Autoencoder Review



Deep Belief Networks



- Repeating units of visible/hidden
- Attempt to reconstruct visible unit from hidden representation, trained in greedy fashion
- Can learn features for supervised learning

Main Idea: Enforce Sparsity in Hidden Representation

- Already a natural bottleneck by causing dimensionality of latent state to be less than initial input
- Introduction of new method to enforce sparsity in latent representation
- Sparsity constraint ideally generates more compact latent representations

Viewing Unsupervised Learning Through Energy

$$P(Y|W) = \int_z P(Y, z|W) = \frac{\int_z e^{-\beta E(Y,z,W)}}{\int_{y,z} e^{-\beta E(y,z,W)}}$$

Probability constructing an input Y given W

$$L(W,Y) = -\frac{1}{\beta}\log\int_z e^{-\beta E(Y,z)} + \frac{1}{\beta}\log\int_{y,z} e^{-\beta E(y,z)}$$

Loss function: Free energy and **log partition function** (penalty term for low energy, wants high energy everywhere else)

Simplification Assuming Peaked Distribution over Z

$$F_{\infty}(Y) = E(Y, Z^*(Y)) = \lim_{\beta \to \infty} -\frac{1}{\beta} \log \int_z e^{-\beta E(Y,z)}$$

In the case that perfect reconstruction is possible, value goes close to 0

$$L^*(W,Y) = E(Y, Z^*(Y)) + \frac{1}{\beta} \log \int_y e^{-\beta E(y, Z^*(y))}$$

Rewrite simplified loss function as a result of this simplification

What is the Problem with Just Using Log Partition?

- Even with the simplification assumption, integrating over all possible latent representations is difficult and so is approximating the gradient
- Resort to expensive computation to approximate this gradient
- Approach in this paper inspired by desire to free from minimizing log partition

Sparse-Encoding Symmetric Machine (SESM)

- Symmetric encoder-decoder paradigm designed to produce sparse latent representations
- **Key Idea: Add sparsity penalty to loss function**

SESM Architecture

$$l(x) = 1/(1 + exp(-gx))$$

Sigmoid function (introduce non-linearity)

$$f_{enc}(Y) = W^T Y + b_{enc}, \qquad f_{dec}(Z) = Wl(Z) + b_{dec}$$

Encoder and Decoder share same weights

$$E(Y,Z) = \alpha_e \|Z - f_{enc}(Y)\|_2^2 + \|Y - f_{dec}(Z)\|_2^2$$

Energy function dictated by reconstruction of both Z and Y

The Main Story: Adding Sparsity Penalty to Loss

$$\begin{aligned} L(W,Y) &= E(Y,Z) + \alpha_s h(Z) + \alpha_r \|W\|_1 \\ &= \alpha_e \|Z - f_{enc}(Y)\|_2^2 + \|Y - f_{dec}(Z)\|_2^2 + \boxed{\alpha_s h(Z)} + \alpha_r \|W\|_1 \end{aligned}$$

$$h(Z) = \sum_{i=1}^{M} \log(1 + l^2(z_i)),$$

Apply penalty on l(z) for being further away from zero

Optimization is Slightly More Complex..

- We want to learn W, $b_{enc,}$ and b_{dec} , so why not use straight-forward gradient descent?
- Because the loss function couples Z with these parameters, and we **DO NOT** want to marginalize over the code distribution (Z)

Iterative Online Coordinate Descent Algorithm

- For a given Y and parameter setting, find Z* (minimize the loss function w.r.t Z)
- Fix Y and Z*, and optimize w.r.t to parameters of interest, do **one step** gradient descent to update parameters

What is the Big Deal with Symmetry?!

- Weight sharing between the encoder and decoder helps in cases when decoder weights collapse to zero or blow up
- Since same weights are used in the encoder, we stop the decoder from going wild (leads to poor reconstruction and also larger code units which are penalized)

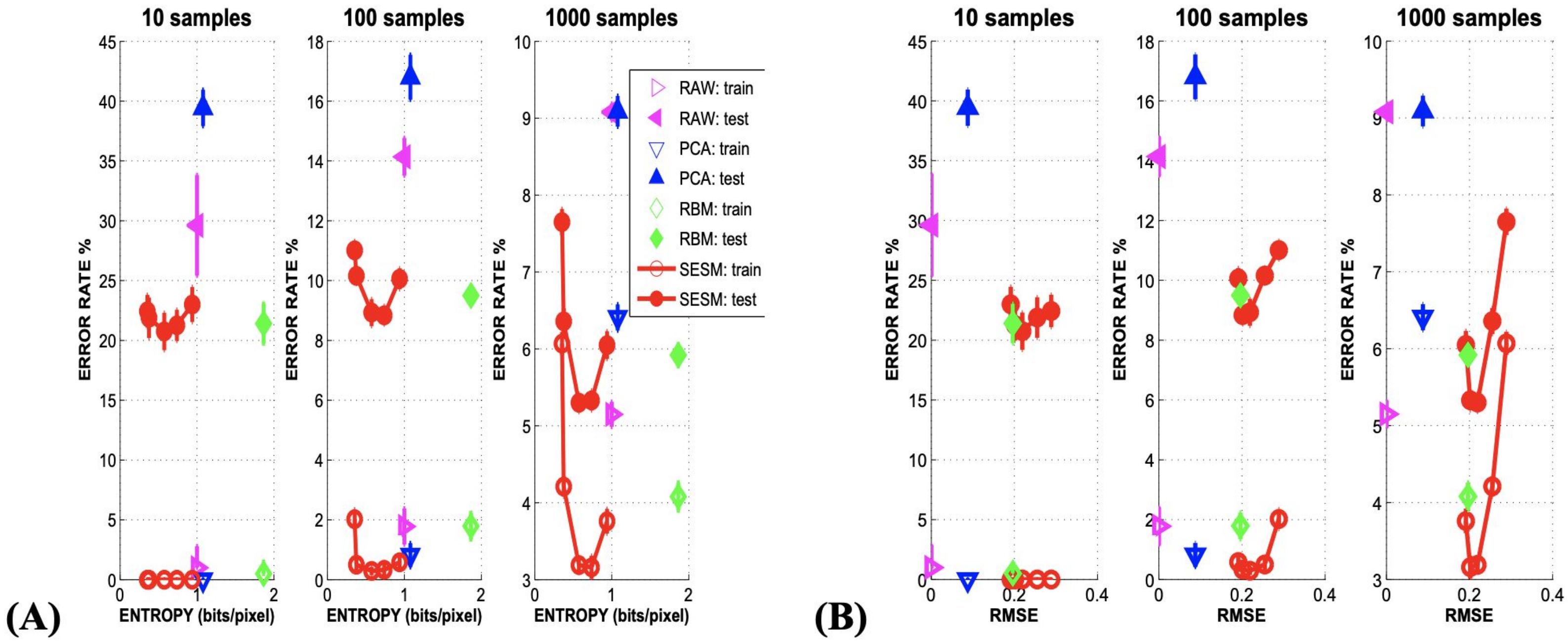
RBM vs SESM

- Both have symmetric encoder and decoder
- SESM has sparsity instead of loss instead of log partition function
- RBM uses Contrastive Divergence (approximate log partition) to prevent flat energy surfaces
- SESM is faster!

Experimental Comparison

- Train SESM, RBM, and PCA on first 20000 digits on MNIST to produce codes of 200 components
- Use test data to produce codes, and measure reconstruction via RMSE (varying degrees of precision)
- Assess discriminative nature of each representation by feeding into linear classifier

Experimental Comparison: Results



Evaluating the Discriminative Ability of a Code

- Is using a linear classifier on the code vector a viable way of assessing the discriminative ability of the code?
- Ties back into validation of autoencoders discussed in the first lecture

What are the benefits of sparsity?

- Sparse PCA developed with the authors stressing that it assisted in interpretability of the loadings?
- How does this compare to the coefficients derived from standard linear regression?

Questions?