Autoencoders

David Dohan
Unsupervised Deep Learning

• So far: supervised models
  • Multilayer perceptrons (MLP)
  • Convolutional NN (CNN)
• Up next: unsupervised models
  • Autoencoders (AE)
  • Deep Boltzmann Machines (DBM)
The Goal

• Build high-level representations from large unlabeled datasets

• Feature learning

• Dimensionality reduction

• A good representation may be:
  • Compressed
  • Sparse
  • Robust
Hierarchical Representation

Layer 3
Parts combine to form objects

Layer 2

Layer 1

High-level linguistic representations

Prior: underlying factors & concepts compactly expressed w/ multiple levels of abstraction
The Goal

- Uncover implicit structure in unlabeled data
- Use labelled data to finetune the learned representation
  - Better initialization for traditional backpropagation
  - Semi-supervised learning
Manifold Hypothesis

- Realistic data clusters along a manifold
- Natural images v. static
- Discovering a manifold, assigning coordinate system to it
Manifold Hypothesis

- Realistic data clusters along a manifold
- Natural images v. static
- Discovering a manifold, assigning coordinate system to it
Principal Component Analysis

Reduce dimensions by keeping directions of most variance

Direction of first principal component i.e. direction of greatest variance
Principal Component Analysis

Given $N \times d$ data matrix $X$, want to project using largest $m$ components

1. Zero mean columns of $X$
2. Calculate SVD of $X = U\Sigma V$
3. Take $W$ to be first $m$ columns of $V$
4. Project data by $Y = XW$

Output $Y$ is $N \times m$ matrix
Autoencoder Structure

- Input, hidden, output layers
- Learning *encoder* to and *decoder* from feature space
- Information bottleneck
Shallow Autoencoder

- AE with 2 hidden layers
- Try to make the output be the same as the input in a network with a central bottleneck.

- The activities of the hidden units in the bottleneck form an efficient code.
- Similar to PCA if layers are linear.
Encoder: \[ h_j = \frac{1}{1 + \exp(-\sum_i v_i W_{ij})}, \quad j = 1, \ldots, K. \]

Decoder: \[ \hat{v}_i = \frac{1}{1 + \exp(-\sum_j h_j W_{ij})}, \quad i = 1, \ldots, D. \]
Shallow Autoencoder

Minimize reconstruction error:

\[
\min_{W} \text{Loss}(v, \hat{v}, W) + \text{Penalty}(h, W)
\]

Loss functions: cross-entropy or squared loss.

Typically, one imposes \( l_1 \) regularization on hidden units \( h \) and \( l_2 \) regularization on parameters \( W \) (related to sparse coding).
Deep Autoencoder

- Non-linear layers allow an AE to represent data on a non-linear manifold

- Can initialize MLP by replacing decoding layers with a softmax classifier
Training Autoencoders

- Backpropagation
- Trained to approximate the identity function
- Minimize reconstruction error

Objectives:
- Mean Squared Error: $\|v - \hat{v}\|^2$
- Cross Entropy:

$$\sum_{k=1}^{d} [v_k \log \hat{v}_k + (1 - v_k) \log (1 - \hat{v}_k)]$$
Reconstruction Example

Data
30-D AE
30-D PCA
Learned Filters

- Each image represents a neuron
- Color represents connection strength to that pixel
- Trained on MNIST dataset
Learned Filters

- Trained on natural image patches
- Get Gabor-filter like receptive fields
Deep Autoencoder

- Face “vanishing gradient” problem
- Solution: Greedy layer-wise pretraining
  - First approach used RBMs (Up next!)
  - Can initialize with several shallow AE
Denoising Autoencoder

- Want to prevent AE from learning identity function
- Corrupt input during training
  - Still train to reconstruct input
  - Forces learning correlations in data
- Leads to higher quality features
- Capable of learning overcomplete codes
Denoising Autoencoder

Web Demo
Whitening

- AE work best for data with all features equal variance
- PCA whitening
  - Rotate data to principal axes
  - Take top $K$ eigenvectors
  - Rescale each feature to have unit variance

Implementation Details
Additional Resources

- Unsupervised Feature Learning and Deep Learning Tutorial
  - http://ufldl.stanford.edu/wiki/
  - deeplearning.net
  - deeplearning.net/tutorial/
- Thorough introduction to main topics in deep learning
Deep Boltzmann Machines

David Dohan
Generative Models

- Discriminative models learn $p(y \mid x)$
  - Probability of a label given some input
- Generative models instead model $p(x)$
- Sample model to generate new values
Boltzmann Machines (BM)

- Visible and hidden layers
- Stochastic binary units
- Fully connected
- Undirected
- Difficult to train
Energy of a Joint Configuration

\[-E(v, h, \theta) = \sum_{i,j} W_{ij} v_i h_j + \sum_{i<j} W_{ij} v_i v_j + \sum_{k<l} W_{kl} h_k h_l + \sum_{i} b_i v_i + \sum_{j} c_j h_j\]

- $v_i, h_i$ are binary states
- Notice that the energy of any connection is local
  - Only depends on connection strength and state of endpoints
Energy Based Models

- Assign an energy to possible configurations
- For no connections, map to probability with:

\[ P(v) = \frac{e^{-E(v)}}{\sum_v e^{-E(v)}} \]

- \( v \) is a vector representing a configuration
- Denominator is normalizing constant \( Z \)
  - Intractable in real systems
  - Requires summing over \( 2^n \) states
- Low energy \( \rightarrow \) high probability
Energy Based Models

- Use hidden units to model more abstract relationships between visible units
- With hidden units and connections:

\[ P(v) = \frac{\sum_h e^{-E(v,h,\theta)}}{\sum_{v,h} e^{-E(v,h,\theta)}} \]

- \( \theta \) is model parameters (e.g. connection weight)
- \( v, h \) vectors representing a layer configuration
- Similar form to Boltzmann distribution, therefore Boltzmann machines
Energy Based Models

\[ P(v) \propto \sum_h e^{-E(v,h,\theta)} \]

- This is equivalent to defining the probability of a configuration to be the probability of finding the network in that configuration after many stochastic updates.
Hidden Variables

- Latent factors/explanations for data
- Example: movie prediction
Restricted BM (RBM)

- Remove visible-visible and hidden-hidden connections
- Hidden units conditionally independent given visible units (and vice-versa)
  - Makes training tractable
Energy of an RBM

- For $n$ visible and $m$ hidden units
- $W$ is $n \times m$ weight matrix
- $\theta$ denotes parameters $W$, $b$, $c$

\[-E(v, h, \theta) = \sum_{i,j} W_{ij} v_i h_i + \sum_i b_i v_i + \sum_j c_j h_j\]

\[= v W h^\top + b v^\top + c h^\top\]

- $b$, $v$ length $n$ row vectors
- $c$, $h$ length $m$ row vectors
- Equation represents:
  
  \((\text{vis} \leftrightarrow \text{hid}) + \text{visible bias} + \text{hidden bias}\)
Inference in RBMs

- Conditional distribution of visible and hidden units given by

\[ p(v_i = 1 | h) = \sigma(b_i + \sum_j h_j W_{ij}) \]

\[ p(h_j = 1 | v) = \sigma(c_j + \sum_i v_i W_{ij}) \]

- Each layer distribution completely determined given other layer
  - Given \( v \), \( p(h_j = 1 | v) \) is exact
RBM Training

\[
P_{\text{model}}(v) = \sum_h P(v, h) = \frac{1}{\mathcal{Z}} \sum_h \exp \left( \sum_{ij} v_i h_j W_{ij} \right)
\]

- Maximizing likelihood of training examples \( v \) using SGD

\[
\frac{\partial \log P(v)}{\partial W_{ij}} = \mathbb{E}_{P_{\text{data}}} [v_i h_j] - \mathbb{E}_{P_{\text{model}}} [v_i h_j] = \langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{model}}
\]

- First term is exact
  - Calculate \( p(h_j = 1|v) \) for every example
- Second term must be approximated
• Consider the gradient of a single example $\mathbf{v}$

\[
\frac{\partial \log P(\mathbf{v})}{\partial W_{ij}} = \mathbb{E}_{P_{data}}[v_i h_j] - \mathbb{E}_{P_{model}}[v_i h_j]
\]

• First term is exactly $\sigma(c_j + \sum_i v_i W_{ij}) v_i$

• Approximate second term by taking many samples from model and averaging across them
RBM Training

• Bias terms are even simpler
  • Treat as a unit that is always on

\[
\frac{\partial \log P(v)}{\partial b_i} = \langle v_i \rangle_{data} - \langle v_i \rangle_{model}
\]

\[
\frac{\partial \log P(v)}{\partial c_j} = \langle h_j \rangle_{data} - \langle h_j \rangle_{model}
\]
Sampling in an RBM

- Approximate model expectation by drawing many samples and averaging

\[ p(v_i = 1|h) = \sigma(b_i + \sum_j h_j W_{ij}) \]

\[ p(h_j = 1|v) = \sigma(c_j + \sum_i v_i W_{ij}) \]

- Stochastically update each unit based on input

- Initialize randomly
Sampling in an RBM

- Update each layer in parallel
  \[ p(v_i = 1|\mathbf{h}) = \sigma(b_i + \sum_j h_j W_{ij}) \]
- Alternate layers
  \[ p(h_j = 1|\mathbf{v}) = \sigma(c_j + \sum_i v_i W_{ij}) \]
- Known as a markov chain or fantasy particle
Contrastive Divergence (CD)

- Reaching convergence while sampling may take hundreds of steps
- K step contrastive divergence (CD-k)
  - Use only k sampling steps to approximate the expectations
  - Initialize chains to training example
- Much less computationally expensive
- Found to work well in practice
Algorithm 1 Calculating gradients given data and RBM parameters using CD-1

procedure RBMGRADIENT\((v_{pos}, W, b, c)\)

\[ N \leftarrow \# \text{ examples in } v_{pos} \]

\[ h_{pos} \leftarrow S(v_{pos} W + c) \]

\[ h_{state} \leftarrow \text{sample from } h_{pos} \]

\[ v_{neg} \leftarrow \text{sample from } S(h_{state} W^\top + b) \]

\[ h_{neg} \leftarrow S(v_{neg} W + c) \]

\[ \Delta W \leftarrow \frac{1}{N}(v_{pos}^\top h_{pos} - v_{neg}^\top h_{neg}) \]

\[ \Delta b \leftarrow \text{column\_means}(v_{pos} - v_{neg}) \]

\[ \Delta c \leftarrow \text{column\_means}(h_{pos} - h_{neg}) \]

return \(\Delta W, \Delta b, \Delta c\)

end procedure

Notice that \(h_{pos}\) is real valued while \(v_{neg}\) is binary
Algorithm 2 Basic RBM training using stochastic gradient descent

procedure TRAINRBM(data, $\epsilon$, $n_v$, $n_h$, $n_{epoch}$, batchsize)

    $W \leftarrow$ matrix of size $n_v \times n_h$
    $b \leftarrow$ vector of size $n_v$
    $c \leftarrow$ vector of size $n_h$
    $N \leftarrow \#$ examples in data
    for all $1, \ldots, n_{epoch}$ do
        for all $mb \in 1, \ldots, \left\lfloor \frac{N}{\text{batchsize}} \right\rfloor$ do
            batch $\leftarrow$ data$[mb \cdot \text{batchsize} : (mb + 1) \cdot \text{batchsize}]$
            $\Delta W, \Delta b, \Delta c \leftarrow$ RBMGRADIENT(batch, $W$, $b$, $c$)
            $W \leftarrow W + \epsilon \Delta W$
            $b \leftarrow b + \epsilon \Delta b$
            $c \leftarrow c + \epsilon \Delta c$
        end for
        d $\leftarrow$ shuffle order of d
    end for
    return ($W$, $b$, $c$)
end procedure
Persistent CD (PCD)

- Markov chains persist between updates
- Allows chains to explore energy landscape
- Much better generative models in practice

(a) Example MNIST training data

(b) Samples drawn from a model trained with CD
(c) Samples drawn from model trained with PCD
Persistent CD (PCD)

- In CD, # chains = batch size
  - Initialized to data in the batch
- Any # of chains in PCD
  - Initialized once, allowed to run
  - More chains lead to more accurate expectation

(a) Example MNIST training data

(b) Samples drawn from a model trained with CD

(c) Samples drawn from model trained with PCD
KL Divergence

- Measure of difference between probability distributions
- CD learning minimizes KL divergence between data and model distributions
- NOT the log likelihood
Deep Belief Networks (DBNs)

- Limitations on what a single layer model can efficiently represent
- Want to learn multi-layer models
- Create a stack of easy to train RBMs
Greedy layer-wise Pretraining

Greedy layer-by-layer learning:

- Learn and freeze $W^1$

- Sample $h^1 \sim P(h \mid v, W^1)$
  treat $h^1$ as if it were data

- Learn and freeze $W^2$

- …

- Repeat
Greedy Layer-wise Pretraining

- Each extra layer improves lower bound on log probability of data

- Additional layers capture higher-order correlations between unit activities in the layer below
Deep Belief Networks (DBNs)

- Top two layers from an RBM
- Other connections directed
- Can generate a sample by sampling back and forth in top two layers before propagating down to visible layers
Greedy Layer-wise Pretraining

Web Demo
Deep Boltzmann Machines

- All connections undirected
- Bottom-up and top-down input to each layer
- Use layer-wise pretraining followed by joint training of all layers
Pretraining DBMs

- Layerwise pretraining
- Must account for input doubling for each layer
Joint Training of DBMs

- Pretraining initializes parameters to favorable settings for joint training
- Update equations take same basic form:

\[ \langle \cdot \rangle_{\text{data}} - \langle \cdot \rangle_{\text{model}} \]

- Model statistic remains intractable
  - Approximate with PCD
- Data statistic, which was exact in the RBM, must also be approximated

\[ \frac{\partial \log P(v)}{\partial W_{ij}} = \langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{model}} \]
Data Dependent Statistic

- No longer exact in DBM
- Approximate with mean-field variational inference
- Clamp data, sample back and forth in hidden layers
- Use expectation instead of binary state
Model Dependent Statistic

- Approximate with gibbs sampling as in an RBM
- Always use PCD
- Alternate sampling even/odd layers
Finetuning for Classification

- Can use to initialize MLP for classification
- Ideal with lots of unsupervised and little supervised data
Semi-supervised Learning

- Makes use of unlabelled data together with some labelled data
- Initialize by training a generative model of the data
- Slightly adjust for discriminative tasks using the labelled data
- Most of the parameters come from generative model
Sparsity

- Hidden units that are rarely active may be easier to interpret or better for discriminative tasks
- Add a “sparsity penalty” to the objective
  - Target sparsity: want each unit on in a fraction $p$ of the training data
  - Actual sparsity $q_{new} = \lambda q_{old} + (1 - \lambda) q_{current}$

Sparsity penalty $\propto -p \log q - (1 - p) \log(1 - q)$

- Used to adjust bias and weights for each hidden unit
Initializing Autoencoders
Example: ShapeBM

- Weight sharing and sparse connections
- Each layer models a different part of the data
Example: ShapeBM

- Used DBM-like structure to learn a shape prior for segmentation
- Good showcase of generative completion abilities
Completing unknown data

- Repeatedly sample from model
- For any visible layer sample, clamp known locations to the known values
Extensions

- Gaussian RBM for image data
- Convolutional RBM
Accelerating Training

- Most operations are implemented as large matrix multiplications
- Use optimized BLAS libraries
  - Matlab, OpenBLAS, IntelMKL
- GPUs can accelerate most operations
  - CUDA
  - Matlab Parallel Computing Toolbox
  - Theano
Example Matlab code
Additional Resources

- [http://goo.gl/UWtRWT](http://goo.gl/UWtRWT)
- Neural Networks, deep learning, and sparse coding course videos from Hugo Larochelle
- A Practical Guide to Training RBMs