Deep Learning

Developing an R package and addressing open questions in the MNIST application

Adam Maidman, Aaron Molstad, Yang Yang, Lin Zhang

April 30, 2015
Overview

1. Introduction to “representation learning”
   - Deep learning
   - Restricted Boltzman machines

2. Deep learning in R
   - Our R package (in progress)
   - Computational difficulties

3. Results
   - Empirical results
   - Future work, availability
Introduction to “representation learning”

Main idea: transform raw inputs into (abstract) representations that can be exploited for a learning task.

- Familiar example: Principal components analysis
- Applications:
  - Image recognition
  - Speech recognition
  - Recommender systems
MNIST example

An abstract representation of the above may be

- The number of vertical or horizontal “changes”
- The presence of “enclosed” white space
- Others?
Deep learning

“Deep learning” is a class of representation learning techniques that uses many layers of nonlinear abstractions of the raw inputs.

- Can be supervised or unsupervised
- Each additional layer uses output from the previous layer as input
Building deep learning architecture

The most popular approach for fitting a deep network is the Restricted Boltzmann Machine.

- Input set of “visible” units and fits set of “hidden” units
- Restricted $\implies$ acyclic
- For many layers, use “hidden” units as new “visible” units for next layer
Energy-Based Models

Incorporating hidden units, we can write the “energy function” as

\[ P(v) = \sum_{i=1}^{H} P(v, h_i) = \sum_{i=1}^{H} \frac{e^{-E(v, h_i)}}{Z}. \]

Introducing a “free energy” function, defined as

\[ \mathcal{F}(v) = -\log \sum_{i=1}^{H} e^{-E(v, h_i)} \]

so that \( P(v) = \frac{e^{-\mathcal{F}(v)}}{Z} \). In an RBM,

\[ E(v, h) = -b^T v - c^T h - h^T W v \]
Objective Function

Using the above formulation, fitting one layer of a RBM requires solving for \( \hat{\theta} = (\hat{b}, \hat{c}, \hat{W}) \),

\[
\hat{\theta} = \arg \max_{b,c,W} \sum_{t=1}^{n} P(v^{(t)}) = \arg \min_{b,c,W} \sum_{t=1}^{n} - \log P(v^{(t)}) \\
= \arg \min_{b,c,W} \sum_{t} \mathcal{F}(v^{(t)}) \\
= \arg \min_{b,c,W} \sum_{t} - \log \sum_{i=1}^{H} e^{-E(v^{(t)}, h_i)} \\
= \arg \min_{b,c,W} \sum_{t=1}^{n} \left( -b^T v^{(t)} - \sum_{i=1}^{H} \log(1 + e^{c_i + w_i^T v^{(t)}}) \right)
\]

where \( h_i \) is the \( i^{th} \) hidden unit for \( i = 1, \ldots, H \)
and \( t = 1, \ldots, n \) are the observations with Bernoulli distribution.
Solving

Taking the derivative of $P(v)$ with respect to the elements of $W$, 

$$
- \frac{d \log P(v)}{dW_{ij}} = \sum_{t=1}^{n} (E(P(h_i|v^{(t)}))v_j^{(t)}) - v_j^{(t)} \text{sigmoid}(c_i + w_i^T v^{(t)}) \\
= \sum_{t} E(v_j^{(t)} h_i)_{\text{model}} - E(v_j h_i)_{\text{data}}
$$

using gradient descent, updates follow as

$$
w_{ij}^{k} = w_{ij}^{k-1} - \epsilon \left( \sum_{t=1}^{n} E(v_j^{(t)} h_i)_{\text{model}} - E(v_j h_i)_{\text{data}} \right)
$$

and update $b_j, c_i$ sequentially.
Computational Difficulties

- We can directly compute an unbiased estimate of $E(v_j h_i)_{data}$ by drawing $h_j$ from Bernoulli($\text{sigmoid}(b_j + \sum_i v_i w_{ij})$)

- Computing $E(v_j^{(t)} h_i)_{model}$ requires MCMC and is computationally expensive if one desired an unbiased estimate.

  - Initialize at random $v$, run Gibbs sampler until convergence
  - For the $k$–th iteration,

$$h_i^k \overset{ind}{\sim} \text{Bernoulli} \left( \text{sigmoid} \left[ \sum_{j=1}^{p} w_{ij} v_j^{k-1} + c_i \right] \right)$$

$$v_j^k \overset{ind}{\sim} \text{Bernoulli} \left( \text{sigmoid} \left[ \sum_{i=1}^{H} w_{ij} h_i^k + b_j \right] \right)$$
Hinton (2002) proposed contrastive divergence ($CD_k$)

To estimate $E(v^{(t)}h)_{model}$

- Initialize Gibbs sampler at observed “visible” input, draw $k$ Gibbs samples
- In practice, $k = 1$ is often used $\Rightarrow E(v^{(t)}h)_{model}$ is biased AND highly dependent on initializing value
Open question

Some major open questions

- How does the length of the Markov chain affect classification results?
- How does the convergence rate of back propagation depend on the $k$ using in training?
Our work

1. Develop an R package capable of fitting neural nets with deep architecture

2. How does the number of Gibbs steps \((k)\) effect classification on a benchmark classification problem (MNIST)?

3. Does the value of \(k\) effect any other computational aspects? Could larger \(k\) decrease computational time in some settings?
The \textit{R} package

Initially based on the binary input version of Hinton’s MATLAB code and \textit{A practical guide to training restricted Boltzmann machines} (2010).

Our package differs from existing \textit{R} packages:

- Use \texttt{Rcpp Armadillo}
- Allows for more customization:
  - choose number of layers,
  - choose number of hidden units,
  - choose $k$ Gibbs steps for computation of contrastive divergence
- We plan to expand capabilities to real input/outputs
Classification Performance; $k$ varying

- We used 20,000 observations randomly selected observations from the MNIST training database as training data.
- We test on 3,000 observations randomly selected from the MNIST test database as test data.
- Fit a deep neural network with 784-1000-500-250 nodes and used backpropogation (formulation Hinton’s MATLAB code) to fit a neural network after pretraining with our package.
- Note that results here are from only one replication. More replications are in progress right now. So far, additional replications show similar results.
- Tried first with $k = 1, 2, 5, 10$ (consider; $k = 1000$ would take 400 hours with this subsetted data)
Classification Performance; $k$ varying

![Graph showing test error vs Gibbs steps $K$.]

**Figure:** Error rates for MNIST test data using deep neural network with 784-1000-500-250 nodes.
Classification Performance; $k$ varying

Figure: Error rates for MNIST test data using deep neural network with 784-1000-500-250 nodes measured over epochs required for convergence of backpropogation algorithm.
Conclusions and future work

- Deep learning has many interesting unanswered questions

- Running the Gibbs sampler longer for Constrastive Divergence appears to lead to smaller prediction error; more replications needed to verify

- It is possible that larger $k$ reduces the computational time required for backpropogation; it may be computationally advantageous to use larger $k$ in classification problems

- Look on CRAN for our R package over the summer, or request a GitHub pull from yang1375@umn.edu


Thank You

Special thanks to Si Peng and Abhirup Mallik for useful discussions

Questions?