Short Writeup of Deep Learning via Pretraining

Kevin Lin

September 17, 2014

We describe the R package SMLdbn we designed to understand deep learning algorithms. It uses the R package neuralnet in one of its subroutines. It uses resilient backpropagation [Rie]. We use [lab14][Chapter 5,7-11] and [Ben08][Chapter 4-6] to build the code for pretraining.

1 Overview and Notation

Let \((x_1, y_1), \ldots, (x_n, y_n) \in [0, 1]^m \times [0, 1]^p\) (binary visible units) be \(n\) data points used to train the neural network where each data point \(x_b\) is \(m\)-dimensional and the label \(y_b\) is a vector of length \(p\) with exactly one 1 to denote the class of \(x_b\). We write the class \(y_b\) as an indicator vector instead of a natural number since this will be more natural when computing the loss function. When we index a vector by \(i\) (ie: \(x_i\) or \(h^k_i\)), we refer to the \(i\)th dimension of the vector. When we index a matrix by \(i\) (ie: \(W^k_i\)), we refer to the \(i\)th column of the matrix. If we double-index a matrix by \(ij\), we refer to the \((i,j)\)th element of the matrix. If we index a vector by \(b\) however (ie: \(x_b\) or \(h_b\)), we are referring to distinct vectors.

Assume we are dealing with a neural network with \(\ell\) hidden layers. Let \(d_k\) denote the number of units in the \(k\)th layer. (Note that \(\ell\) and \(d_1, \ldots, d_\ell\) are parameters that the user gets to set.) Let \(h^0\) be the numeric vector of the units in the visible layer (the bottom layer) where \(d_0 = m\) (ie: if we are focusing on \(x_i\), then \(h^0 = x_i\)). Let \(h^k \in [0, 1]^{d_k}\) (binary hidden units) denote the numeric vector of the units at the \(k\)th layer where \(k \in \{1, 2, \ldots, \ell + 1\}\). We call the \((\ell + 1)\)th-layer the classification layer (top layer) since the values of the units in this layer, \(h^\ell\), will tell us how to classify a data point \(x\). In the top layer, \(d_{\ell+1} = p\). When fitting neural networks, our goal is to fit \(\{b^k, c^k, W^k\}_{k=1}^{\ell+1}\) where \(b^k \in \mathbb{R}^{d_k}\), \(c^k \in \mathbb{R}^{d_{k-1}}\), \(W^k \in \mathbb{R}^{d_{k-1} \times d_k}\) to minimize an objective function. We will be minimizing the cross-entropy loss function \(^1\)

\[
\sum_{b=1}^{n} H(y_b, f(x_b)) = -\sum_{b=1}^{n} \sum_{i=1}^{p} \log(f(x_b)_i) y_{bi}. \tag{1.1}
\]

\(^1\)In words, the cross entropy is the negative sum over all \(n\) data points where, for each data point, we sum over all dimensions (classes) \(i\) by computing \(\log(f(x_b)_i)\), the log predicted “probability” for \(x_b\) to be assigned to class \(i\) multiplied by \(y_{bi}\), a 0 or 1 depending on whether (in truth) \(x_b\) belongs to class \(i\). We put quotations around “probability” since in our particular application, \(f(x_b)\) is not enforced to have its elements sum to 1.
To do prediction with a neural network, we compute \( f(x_b) \) which is best explained as a recursive equation\(^2\)

\[
\begin{align*}
  x_b &\Rightarrow h^0 \\
  \sigma(b^k + W^k h^{k-1}) &\Rightarrow h^k \quad \forall k \in \{1, \ldots, \ell\} \\
  g(b^{\ell+1} + W^{\ell+1} h^\ell) &\Rightarrow h^{\ell+1} = f(x_b) \in \mathbb{R}^p
\end{align*}
\]

where \( \sigma \) is called the activation function which we will model as the sigmoid function \( \text{Sig}(x) = 1/(1+e^{-v}) \). If the input to \( \sigma \) is actually a vector, we perform \( \sigma(x) \) for each element of the vector. \( g \) is the final transformation function. Typically, \( g \) is the softmax function, \( \text{Softmax}(x)_i = e^{x_i}/\sum_{j=1}^{p} e^{x_j} \), where the index \( i \) (or \( j \)) refers the element in \( i^{th} \) dimension, however in our particular setting, we will set \( g(x) = \sigma(x) \). (We do this because the \texttt{neuralnet} package only allows this.) We then assign \( x \) to class \( i' \) if \( i' = \text{argmax}f(x)_i \). Typically, when people fit a neural network, they perform some version of backpropagation which amounts to taking the derivative of Equation (1.1) with respect to each parameter with heavy utilization of the chain rule due to Equation (1.2). Typically, all the parameters \( \{b^k, c^k, W^k\}_{k=1}^{\ell+1} \) are randomly initialized or set to 0.

One of the ideas in \textit{deep learning} is to perform a “smarter” initialization before we do backpropagation. The particular method presented here does this by building a stack of \textit{Restricted Boltzmann Machines} (RBMs), training the RBMs without knowledge of the labels \( y_b \) and then using the finalized parameters for the RBM to initialize our neural network. We carry over our notation. Our stack of RBMs has \( \ell \) hidden layers and the visible layer \( h^0 \), and the architecture of these layers match that of the neural network exactly (ie: \( d^k \) match exactly in both architectures for \( k = \{0, \ldots, \ell\} \)). When fitting RBM’s, our goal is to fit \( \{b^k, c^k, W^k\}_{k=1}^{\ell} \) with the objective of maximizing \( \prod_{b=1}^{n} P(x_b) \) where the probability measure \( P \) has a specific form for RBMs (more below).

[Ben08][Chapter 5],[lab14][Chapter 9],[LCH+06]

## 2 Quick Results of RBM

For RBMs (energy-based models with hidden units where no hidden units are connected to one another and no visible unit are connected to one another), the probability of observing a sample \( x \) is

\[
P(x) = \frac{e^{-F(x)}}{Z(x)} \quad \text{where} \quad F(x) = -\log \sum_{h} e^{-E(x, h)} \quad \text{and} \quad E(x, h) = -b^T x - c^T h - h W^T x.
\]

Here, \( Z \) is called the partition number (to ensure \( P(x) \) is a valid probability, ie: \( Z(x) = \int e^{-F(x)} \)), \( F(x) \) is the free energy function and \( E(x, h) \) is the energy function. Our objective during pretraining is to find a set of parameters \( \{\tilde{W}', \tilde{b}', \tilde{c}'\} \) to maximize the following criterion

\[
\{\tilde{W}', \tilde{b}', \tilde{c}'\} = \arg \max_{\tilde{W}, \tilde{b}, \tilde{c}} \sum_{b=1}^{n} \log P(x_b).
\]

\(^2\)Note that \( c^k \) appears nowhere in the prediction algorithm. \( c^k \) is needed only as an artifact of pretraining.
Let us derive the gradient of this objective function. Since \(x\) and \(h\) are binary, we can derive [Ben08][Equation 5.22]\(^3\)

\[
\mathbb{P}(h_i = 1|x) = \text{Sig}(\tilde{c}_i + \tilde{W}_T^i \mathbf{x}) \quad (2.1)
\]

\[
\mathbb{P}(x_i = 1|h) = \text{Sig}(\tilde{b}_i + \tilde{W}_i \mathbf{h}). \quad (2.2)
\]

Let \(Q(h|x)\) denote the distribution of \(h\) given \(x\). Our goal is to minimize the negative log-likelihood with respect to \(\log \mathbb{P}(x)\). It can be derived that this amounts to [lab14][Equation 9.5]

\[
\frac{\partial \log \mathbb{P}(x)}{\partial \theta} = -\sum_h \mathbb{P}(h|x) \frac{\partial E(x, h)}{\partial \theta} + \sum_{\tilde{x}, h} \mathbb{P}(h|\tilde{x}) \frac{\partial E(\tilde{x}, h)}{\partial \theta} = -\mathbb{E}_{\mathbf{h}}\left[\frac{\partial E(\mathbf{x}, \mathbf{h})}{\partial \theta}\right]|_{\text{data}} + \mathbb{E}_{\tilde{x}, h}\left[\frac{\partial E(\tilde{x}, \mathbf{h})}{\partial \theta}\right]|_{\text{model}}
\]

where \(\theta\) is a scalar representing any parameter (ie: \(\tilde{b}, \tilde{c}, \tilde{W}\)), and \(\tilde{x}\) denotes any other possible \(x\). Hinton calls the first term “data” since the partial is taken with a “clamped” \(x\) and the second term “model” since we take the expectation jointly between all possible \(\tilde{x}\) and \(h\) [Hin10][Section 2]. Evaluating these partials, we can show that

\[-\mathbb{E}_{\mathbf{h}}\left[\frac{\partial E(\mathbf{x}, \mathbf{h})}{\partial \theta}\right]|_{\text{data}} = \{xh^T\}_{ij} \quad \text{where} \quad \mathbf{h} = \text{Sig}(\tilde{c} + \tilde{W}^T \mathbf{x})\]

The partial taken with respect to other parameters can be derived easily. We take our equations for [Ben08][Chapter 5, Algorithm 1],[Hin10][Section 2-3].

We would traditionally use a Gibbs sampler to compute the second expectation, but the theory of contrastive divergence (CD) combined with empirical evidence states that we can reasonably approximate the expectation with not only just one sample of the Gibbs sampler, but one step of the Gibbs sampler as well [BD09][Theorem 4.1],[MH]. CD starts the Markov chain with \(x\) itself and takes only one step (ie: does not wait for the chain to converge). The easiest way to see how we evaluate the second expectation is to look at the algorithm Algorithm 1.

3 Pretraining + Finetuning Algorithm

Equipped with a method to computing the gradient, we now can implement a stochastic gradient approach to train our stack of RBMs. Hinton justified empirically that training a stack of RBM layer-by-layer and showed that this procedure is reasonable since \(\prod_{b=1}^n \mathbb{P}(x_b)\) can only increase as you fit successive layers [HO06][Section 4],[BLPL][Section 2.3].

Let us focus on the algorithm for a particular RBM in our stack first shown in Algorithm 1. Assume that we have \(x_1, \ldots, x_n\), the inputs to this particular RBM. Let \(\mathbf{h}\) denote the vector of hidden units.

\(^3\)That is, the transitions between two layers are random in an RBM. This is unlike the perceptron model described in the first section (Equation (1.2)). Also, we should emphasize that it is only through our clever choice of the model where we got a sigmoid function which coincidentally matches the activation function \(\sigma\) we typically use in a neural network (ie: if \(x \notin [0, 1]^m\), then Equation (2.2) would not hold).
In our simulation, our learning rate $\epsilon$ is 0.1, and we perform the update over 10 epochs each with 20 data points. These are taken from [lab14]. [Fin10][Section 4] warns not to use too many data points in one batch.

\textbf{Algorithm 1} Updating the $k^{th}$ RBM (Outline for sml\_DBN\_costupdate)

\begin{algorithm}
\caption{Updating the $k^{th}$ RBM (Outline for sml\_DBN\_costupdate)}
\begin{algorithmic}[1]
\Require Inputs $x_1, \ldots, x_n$, parameters $\widehat{W}, \widehat{b}, \widehat{c}$, learning rate $\epsilon$
\For {A set number of epochs, considering a batch $\{x_b\}$ of size $B$ at a time} 
\State Compute $\tilde{h}_b = \text{Sig}(\tilde{c}_i + \widehat{W}_b x_b)$ from each $x_b$ in the batch. \{One-Step Gibbs Sample\} 
\State Sample $h_b \in [0, 1]^{d_k}$ for each $h_b$. \{See Equation (2.1)\} 
\State Compute $\tilde{x}_b' = \text{Sig}(\tilde{b}_i + \widehat{W}_b^T h_b)$ for each $h$. 
\State Sample $x_b' \in [0, 1]^{d_k-1}$ for each $x_b'$. \{See Equation (2.2)\}
\State Compute $\tilde{h}_b' = \text{Sig}(\tilde{c}_i + \widehat{W}_b x_b')$ from each $x_b'$.
\State $\widehat{W} \leftarrow \widehat{W} + \frac{\epsilon}{B} (\sum_b x_b h_b^T - \tilde{x}_b' h_b^T)$ \{Update parameters\}
\State $\tilde{b} \leftarrow \tilde{b} + \frac{\epsilon}{B} (\sum_b x_b - x_b')$
\State $\tilde{c} \leftarrow \tilde{c} + \frac{\epsilon}{B} (\sum_b h_b - h_b')$
\EndFor 
\State \Return $\widehat{W}, \tilde{b}, \tilde{c}$
\end{algorithmic}
\end{algorithm}

We describe our pretraining algorithm in Algorithm 2. Notice Step #4 where we are maximizing the objective function in a greedy layer-by-layer fashion.

\textbf{Algorithm 2} Pretraining a Neural Network (Outline for sml\_DBN\_pretrain)

\begin{algorithm}
\caption{Pretraining a Neural Network (Outline for sml\_DBN\_pretrain)}
\begin{algorithmic}[1]
\Require A neural network $\mathcal{N} = \{(x_1, y_1), \ldots, (x_n, y_n), \{W^k, b^k, c^k\}_{k=1}^\ell\}$. 
\State Based on the architecture of $\mathcal{N}$ (ie: $\{d_k\}_{k=1}^\ell$), construct an analogous stack of Restricted Boltzmann Machines (RBMs) $\mathcal{R} = \{x_1, \ldots, x_n, \{\widehat{W}^k, \widehat{b}^k, \widehat{c}^k\}_{k=1}^\ell\}$. That is, the stack of RBMs has the same number of layers and the same number of units-per-layer as $\mathcal{N}$ and inherits the parameters $\{\widehat{W}^k, \widehat{b}^k, \widehat{c}^k\}_{k=1}^\ell$.
\State Set $h_b^0 \leftarrow x_b$ for all $b \in \{1, \ldots, n\}$.
\For {$k = 1$ to $k = \ell$}
\State Compute and set 
$$\{\widehat{W}^k, \widehat{b}^k, \widehat{c}^k\} \leftarrow \text{arg max}_{\widehat{W}, \widehat{b}, \widehat{c}} \sum_{b=1}^n \log P(h_b^k | h_b^{k-1})$$
\State using sml\_RB\_costupdate using learning rate $\epsilon$.
\State Sample $h_b^k$ from $Q(h_b^k | h_b^{k-1})$ for all $b \in \{1, \ldots, n\}$ using sml\_RB\_sampl\_h\_giv\_env.
\EndFor 
\State \Return $\{\widehat{W}^k, \widehat{b}^k, \widehat{c}^k\}_{k=1}^\ell$
\end{algorithmic}
\end{algorithm}

We describe the finetuning procedure in Algorithm 3. We set a parameter $\tau$ with is a tunable parameter for our function \texttt{neuralnet} to determine when to stop training. When the partial derivatives of the loss function Equation (1.1) fall below this threshold, \texttt{neuralnet} exits. We use the default setting $\tau = 0.01$. 

4
Algorithm 3 Finetuning a Neural Network (Outline for \texttt{sml\_DBN.finetune})

\textbf{Require:} A neural network $\mathcal{N}$ and a fitted RBM $\mathcal{R} = \{x_1, \ldots, x_n, \{\tilde{W}_k, \tilde{b}_k, \tilde{c}_k\}_{k=1}^\ell\}$, threshold $\tau$.

1: Reinitialize $\{W^k, b^k, c^k\} \leftarrow \{\tilde{W}_k, \tilde{b}_k, \tilde{c}_k\}$ \quad $\forall k \in \{1, \ldots, \ell\}$.
2: Fit the neural network using \texttt{neuralnet} for loss function Equation (1.1) \[\{W^k, b^k, c^k\}_{k=1}^{\ell+1} \leftarrow \text{neuralnet}\{(x_1, y_1), \ldots, (x_n, y_n), \{W^k, b^k, c^k\}_{k=1}^\ell, \tau\}\]
3: \textbf{return} $\{W^k, b^k, c^k\}_{k=1}^{\ell+1}$

Algorithm 4 Deep Learning (Outline for \texttt{main})

\textbf{Require:} Labeled training dataset $(x_1, y_1), \ldots, (x_n, y_n)$, number of hidden layers $\ell$, number of hidden units per layer $\{d_k\}_{k=1}^\ell$, learning rate $\epsilon$, threshold $\tau$.

1: Initialize a neural network $\mathcal{N} = \{(x_1, y_1), \ldots, (x_n, y_n), \{W^k, b^k, c^k\}_{k=1}^\ell\}$ where for each $k \in \{1, \ldots, \ell\}$, $W^k$ is initialized by sampling uniformly from $(\pm 4\sqrt{\frac{6}{d_k-1+d_k}})$. All other parameters are initialized to be 0.
2: Construct and set a stack of RBMs $\mathcal{R} \leftarrow \texttt{sml\_DBN.pretrain}(\mathcal{N}, \epsilon)$.
3: \textbf{Fit} $\mathcal{N} \leftarrow \texttt{sml\_DBN.finetune}(\mathcal{N}, \mathcal{R}, \tau)$.
4: \textbf{return} $\{W^k, b^k, c^k\}_{k=1}^{\ell+1}$

References


