Appendix of the paper "Greedy Layer-Wise Training of Deep Networks"

Yoshua Bengio
Université de Montréal
Montréal, Québec
bengioy@umontreal.ca

Pascal Lamblin
Université de Montréal
Montréal, Québec
lamblinp@iro.umontreal.ca

Dan Popovici
Université de Montréal
Montréal, Québec
popovicd@iro.umontreal.ca

Hugo Larochelle
Université de Montréal
Montréal, Québec
larocheh@iro.umontreal.ca

Algorithm 1 `RBMupdate`$(\mathbf{v}_0, \epsilon, W, b, c)$
This is the RBM update procedure for binomial units. It also works for exponential and truncated exponential units, and for the linear parameters of a Gaussian unit (using the appropriate sampling procedure for Q and P). It can be readily adapted for the variance parameter of Gaussian units, as discussed in the text.
v_0 is a sample from the training distribution for the RBM
ϵ is a learning rate for the stochastic gradient descent in Contrastive Divergence
W is the RBM weight matrix, of dimension (number of hidden units, number of inputs)
b is the RBM biases vector for hidden units
c is the RBM biases vector for input units

---

for all hidden units i **do**
  • compute $Q(\mathbf{h}_{0i} = 1|\mathbf{v}_0)$ (for binomial units, $\mathrm{sigm}(b_i + \sum_j W_{ij}\mathbf{v}_{0j})$)
  • sample $\mathbf{h}_{0i}$ from $Q(\mathbf{h}_{0i} = 1|\mathbf{v}_0)$
end for
for all visible units j **do**
  • compute $P(\mathbf{v}_{1j} = 1|\mathbf{h}_0)$ (for binomial units, $\mathrm{sigm}(c_j + \sum_i W_{ij}\mathbf{h}_{0i})$)
  • sample $\mathbf{v}_{1j}$ from $P(\mathbf{v}_{1j} = 1|\mathbf{h}_0)$
end for
for all hidden units i **do**
  • compute $Q(\mathbf{h}_{1i} = 1|\mathbf{v}_1)$ (for binomial units, $\mathrm{sigm}(b_i + \sum_j W_{ij}\mathbf{v}_{1j})$)
end for
• $W \leftarrow W + \epsilon(\mathbf{h}_0\mathbf{v}_0' - Q(\mathbf{h}_{1\cdot} = 1|\mathbf{v}_1)\mathbf{v}_1')$
• $b \leftarrow b + \epsilon(\mathbf{h}_0 - Q(\mathbf{h}_{1\cdot} = 1|\mathbf{v}_1))$
• $c \leftarrow c + \epsilon(\mathbf{v}_0 - \mathbf{v}_1)$

Algorithm 2 `TrainUnsupervisedDBN`($\widehat{p}, \epsilon, L, n, W, b$)
Train a DBN in a purely unsupervised way, with the greedy layer-wise procedure in which each added layer is trained as an RBM by contrastive divergence.
$\widehat{p}$ is the input training distribution for the network
ϵ is a learning rate for the stochastic gradient descent in Contrastive Divergence
L is the number of layers to train
$n = (n^1, \ldots, n^L)$ is the number of hidden units in each layer
W^i is the weight matrix for level i, for i from 1 to L
b^i is the bias vector for level i, for i from 0 to L

- initialize $b^0 = 0$

for $\ell = 1$ to L **do**
  - initialize $W^i = 0$, $b^i = 0$

  while not stopping criterion **do**
    - sample $\mathbf{g}^0 = x$ from $\widehat{p}$

    for $i = 1$ to $\ell - 1$ **do**
      - sample $\mathbf{g}^i$ from $Q(\mathbf{g}^i|\mathbf{g}^{i-1})$

    end for
    - `RBMupdate`($\mathbf{g}^{\ell-1}, \epsilon, W^\ell, b^\ell, b^{\ell-1}$)

  end while

end for

Algorithm 3 `PreTrainGreedyAutoEncodingDeepNet`($\widehat{p}, C, \epsilon, L, n, W, b$)
Initialize all layers except the last in a multi-layer neural network, in a purely unsupervised way, with the greedy layer-wise procedure in which each added layer is trained as an auto-associator that tries to reconstruct its input.
$\widehat{p}$ is the training distribution for the network
$C = -\log P_\theta(u)$ is a reconstruction error criterion that takes θ and u as input, with θ the parameters of a predicted probability distribution and u an observed value.
ϵ is a learning rate for the stochastic gradient descent in reconstruction error
L is the number of layers to train
$n = (n^0, \ldots, n^L)$, with n^0 the inputs size and n^i the number of hidden units in each layer $i \geq 1$.
W^i is the weight matrix for level i, for i from 1 to L
b^i is the bias vector for level i, for i from 0 to L

- initialize $b^0 = 0$.
- define $\mu^0(x) = x$.

for $\ell = 1$ to L **do**
  - initialize $b^\ell = 0$.
  - initialize temporary parameter vector $c^\ell = 0$.
  - initialize W^ℓ by sampling from uniform$(-a, a)$, with $a = 1/n^{\ell-1}$.
  - define the ℓ-th hidden layer output $\mu^\ell(x) = \text{sigm}(b^\ell + W^\ell \mu^{\ell-1}(x))$.
  - define the ℓ-th hidden layer reconstruction parameter function, e.g. in the binomial case $\theta^\ell = \text{sigm}(c^\ell + W^{\ell'} \mu^\ell(x))$ is the vector of probabilities for the each bit to take value 1.

  while not stopping criterion **do**

    for $i = 1$ to $\ell - 1$ **do**
      - compute $\mu^i(x)$ from $\mu^{i-1}(x)$.

    end for
    - compute $\mu^\ell(x)$ from $\mu^{\ell-1}(x)$.
    - compute reconstruction probability parameters θ^ℓ from $\mu^\ell(x)$.
    - compute the error C in reconstructing $\mu^{\ell-1}$ from probability with parameters θ^ℓ.
    - compute $\frac{\partial C}{\partial \omega}$, for $\omega = (W^\ell, b^\ell, c^\ell)$
    - update layer parameters: $\omega \leftarrow \omega - \epsilon \frac{\partial C}{\partial \omega}$

  end while

end for

Algorithm 4 `TrainSupervisedDBN`($\widehat{p}, C, \epsilon_{CD}, \epsilon_C, L$,$n$,$W$,$b$,$V$)
Train a DBN for a supervised learning task, by first performing pre-training of all layers (except the output weights V), followed by supervised fine-tuning to minimize a criterion C.
$\widehat{p}$ is the supervised training distribution for the DBN, with (input,target) samples (x, y)
C is a training criterion, a function that takes a network output $f(x)$ and a target y and returns a scalar differentiable in $f(x)$
ϵ_{CD} is a learning rate for the stochastic gradient descent with Contrastive Divergence
ϵ_C is a learning rate for the stochastic gradient descent on supervised cost C
L is the number of layers
$n = (n^1, \ldots, n^L)$ is the number of hidden units in each layer
W^i is the weight matrix for level i, for i from 1 to L
b^i is the bias vector for level i, for i from 0 to L V is a weight matrix for the supervised output layer of the network

- Let $\widehat{p}_x$ the marginal over the input part of $\widehat{p}$
- TrainUnsupervisedDBN($\widehat{p}_x, \epsilon_{CD}, L, n, W, b$)
- DBNSupervisedFineTuning($\widehat{p}, C, \epsilon_C, L$,$n$,$W$,$b$,$V$)

Algorithm 5 `DBNSupervisedFineTuning`($\widehat{p}, C, \epsilon_C, L$,$n$,$W$,$b$,$V$)
After a DBN has been initialized by pre-training, this procedure will optimize all the parameters with respect to the supervised criterion C, using stochastic gradient descent.
$\widehat{p}$ is the supervised training distribution for the DBN, with (input,target) samples (x, y)
C is a training criterion, a function that takes a network output $f(x)$ and a target y and returns a scalar differentiable in $f(x)$
ϵ_{CD} is a learning rate for the stochastic gradient descent with Contrastive Divergence
ϵ_C is a learning rate for the stochastic gradient descent on supervised cost C
L is the number of layers
$n = (n^1, \ldots, n^L)$ is the number of hidden units in each layer
W^i is the weight matrix for level i, for i from 1 to L
b^i is the bias vector for level i, for i from 0 to L V is a weight matrix for the supervised output layer of the network

- Recursively define mean-field propagation $\mu^i(x) = E[\mathbf{g}^i|\mathbf{g}^{i-1} = \mu^{i-1}(x)]$
  where $\mu^0(x) = x$, and $E[\mathbf{g}^i|\mathbf{g}^{i-1} = \mu^{i-1}]$ is the expected value of $\mathbf{g}^i$ under the RBM conditional distribution $Q(\mathbf{g}^i|\mathbf{g}^{i-1})$, when the values of $\mathbf{g}^{i-1}$ are replaced by the mean-field values $\mu^{i-1}(x)$.
  In the case where $\mathbf{g}^i$ has binomial units, $E[\mathbf{g}^i_j|\mathbf{g}^{i-1} = \mu^{i-1}] = \mathrm{sigm}(b^i_j + \sum_k W^i_{jk}\mu^{i-1}_k(x))$.
- Define the network output function $f(x) = V(\mu^L(x)', 1)'$
- Iteratively minimize the expected value of $C(f(x), y)$ for pairs (x, y) sampled from $\widehat{p}$ by tuning parameters W, b, V. This can be done by stochastic gradient descent with learning rate ϵ_C, using an appropriate stopping criterion such as early stopping on a validation set.

Algorithm 6 `TrainPartiallySupervisedLayer`$(\widehat{p}, C, \epsilon_C, \epsilon_{CD}, W, b, V)$

This procedure should be called as an alternative to the loop that calls `RBMupdate` *in* `TrainUnsupervisedDBN`*, in order to train with partial supervision: perform unsupervised parameters updates with contrastive divergence, followed by greedy supervised gradient stochastic updates with respect to C, using temporary output weights V to map the hidden layer outputs to predictions.*

$\widehat{p}$ is the supervised training distribution, with samples (x, y), x being the input of the layer, and y the target for the network

C is a training criterion, a function that takes a prediction $f(x)$ and a target y and returns a scalar differentiable in $f(x)$

ϵ_{CD} is a learning rate for the stochastic gradient descent with Contrastive Divergence

ϵ_C is a learning rate for the stochastic gradient descent on supervised cost C

W is the weight matrix for the layer to train

b is the bias vector for that layer

V is a weight matrix that transforms hidden activations into predictions $f(x)$

- Define the mean-field output of the hidden layer, $\mu(x) = E[\mathbf{h}|x]$, for example $\mu(x) = \text{sigm}(b_j + \sum_k W_{jk} x_k)$ for binomial hidden units.
- Define the layer predictive output function $f(x) = V(\mu(x)', 1)'$
- Initialize all parameters $\theta = (W, b, V)$ to 0

while not stopping criterion **do**

- sample (x, y) from $\widehat{p}$
- compute units activation (e.g. $b + Wx$)
- using these activations, compute hidden units mean-field output $\mu(x)$
- using these activations, sample $\mathbf{h}_0$ from $Q(\mathbf{h}|x)$
- compute predictive output $f(x)$ from $\mu(x)$
- compute predictive cost C from $f(x)$ and y
- compute $\frac{\partial C}{\partial \theta}$ by standard back-propagation
- sample $\mathbf{v}_1$ from $P(\mathbf{v}|\mathbf{h}_0)$
- compute $Q(\mathbf{h}_1|\mathbf{v}_1)$
- perform supervised stochastic gradient update $\theta \leftarrow \theta - \epsilon_C \frac{\partial C}{\partial \theta}$
- $W \leftarrow W + \epsilon_{CD}(\mathbf{h}_0 x' - Q(\mathbf{h}_{1\cdot} = 1|\mathbf{v}_1)\mathbf{v}_1')$
- $b \leftarrow b + \epsilon_{CD}(\mathbf{h}_0 - Q(\mathbf{h}_{1\cdot} = 1|\mathbf{v}_1))$
- $c \leftarrow c + \epsilon_{CD}(\mathbf{v}_0 - \mathbf{v}_1)$

end while

Algorithm 7 `TrainGreedySupervisedDeepNet`$(\widehat{p}, C, \epsilon, L, n, W, b, V)$

Greedily train a deep network layer-wise, using a supervised criterion to optimize each layer, as if it were the hidden layer of a one-hidden-layer neural network.

$\widehat{p}$ is the supervised training distribution, with samples (x, y), x being the input of the layer, and y the target for the network with (input,target) samples (x, y)

C is a training criterion, a function that takes a network output $f(x)$ and a target y and returns a scalar differentiable in $f(x)$

ϵ is a learning rate for the stochastic gradient descent on supervised cost C

W is the weight matrix for the layer to train

b is the bias vector for that layer

V is a weight matrix that transforms top-layer hidden activations into predictions $f(x)$

- initialize $b^0 = 0$.
- define $\mu^0(x) = x$.

for $\ell = 1$ to L **do**
- initialize $b^\ell = 0$.
- initialize temporary parameter vector $c^\ell = 0$ and temporary matrix $V^\ell = 0$.
- initialize W^ℓ by sampling from uniform($-a, a$), with $a = 1/n^{\ell-1}$.
- define the ℓ-th hidden layer output $\mu^\ell(x) = \mathrm{sigm}(b^\ell + W^\ell \mu^{\ell-1}(x))$.
- define the ℓ-th temporary output layer prediction $f^\ell(x) = c^\ell + V^\ell \mu^\ell(x)$

    while not stopping criterion **do**

        for $i = 1$ to $\ell - 1$ **do**
- compute $\mu^i(x)$ from $\mu^{i-1}(x)$.

        end for
- compute $\mu^\ell(x)$ from $\mu^{\ell-1}(x)$.
- compute temporary output $f^\ell(x)$ from $\mu^\ell(x)$.
- compute the prediction error C from $f^\ell(x)$ and y.
- compute $\frac{\partial C}{\partial \omega}$, for $\omega = (W^\ell, b^\ell, c^\ell, V^\ell)$
- update layer parameters: $\omega \leftarrow \omega - \epsilon \frac{\partial C}{\partial \omega}$

    end while

end for