CPSC 540: Machine Learning

Boltzmann Machines

Mark Schmidt

University of British Columbia

Winter 2020
We discussed log-linear parameterization of UGMs,

\[ \phi_j(s) = \exp(w_{j,s}), \quad \phi_{jk}(s, s') = \exp(w_{j,k,s,s'}), \quad \phi_{jkl}(s, s', s'') = \exp(w_{j,k,l,s,s',s''}). \]

The likelihood of an example \( x \) given parameter \( w \) is given by

\[ p(x \mid w) = \frac{\exp(w^T F(x))}{Z}, \]

and the feature functions \( F(x) \) count the number of times we use each \( w_j \).

This leads to a convex NLL of the form

\[ -\log p(x \mid w) = -w^T F(x) + \log(Z), \]

and gradient of the form

\[ \nabla_w -\log p(x \mid w) = -F(x) + \mathbb{E}[F(x)], \]

which (if you can do inference) can be optimized with gradient descent methods.
Log-Linear UGM Gradient

- For 1 example, gradient in log-linear UGM with respect to parameter $w_j$ is
  \[ \nabla w_j f(w) = -F_j(x) + \mathbb{E}[F_j(x)]. \]

- Example of $\phi_{10}(3) = \exp(w_{10,3})$ (potential that feature 10 is in state 3).
  - Averaging over $n$ examples, the gradient with no parameter tieing is given by
    \[ \nabla w_{10,3} f(w) = -\frac{1}{n} \left[ \sum_{i=1}^{n} I[x_{10}^i = 3] \right] + p(x_{10} = 3). \]
    - frequency in data
    - model “frequency”

- So if $\nabla w_{10,3} f(w) = 0$, probabilities match frequencies in training data.
- At MLE, you match the frequencies of all the potentials in the training data.
- Typical training method: deterministic gradient descent methods (if have $Z$).
- But computing gradient requires inference (computing marginals like $p(x_{10} = 3)$).
Approximate Learning: Alternate Objectives

- One way to avoid cost of inference is to change the objective:
  - **Pseudo-likelihood** (fast, convex, and crude):

\[
p(x_1, x_2, \ldots, x_d) \approx \prod_{j=1}^{d} p(x_j \mid x_{-j}) = \prod_{j=1}^{d} p(x_j \mid x_{\text{nei}(j)}),
\]

which turns learning into \(d\) single-variable problems (similar to DAGs).
Approximate Learning: Approximate Marginals

Alternately, we can use approximate inference to use NLL:

- **Monte Carlo** approximation of $\mathbb{E}[F_j(x)]$ given current parameters $w$:

\[
\nabla f(w) = -F(x) + \mathbb{E}[F(x)] \\
\approx -F(x) + \frac{1}{t} \sum_{i=1}^{t} F(x^i)
\]

- Simple method: generate lots of samples to approximate gradient given $w$, then update $w$ (many samples per iteration, can grow batch to converge fast).

- **Younes algorithm**: alternate between steps of Gibbs sampling and stochastic gradient, using 1 sample per iteration ("persistent contrastive divergence" in deep learning).

(SG updates $w$, Gibbs updates $x$)

- Deterministic **variational approximations** of $\mathbb{E}[F(x)]$ can alternately be used (later).
Pairwise UGM on MNIST Digits

- Samples from a lattice-structured pairwise UGM:

- Training: 100k stochastic gradient w/ Gibbs sampling steps with $\alpha_t = 0.01$.
- Samples are iteration 100k of Gibbs sampling with fixed $w$. 
Recall that in Ising UGMs, our edge potentials have the form

\[ \phi_{ij}(x_i, x_j) = \exp(w_{ij}x_ix_j). \]

If we set \( w_{ij} = 0 \), it sets \( \phi_{ij}(x_i, x_j) = 1 \) for all \( x_i \) and \( x_j \).

Potential just “multiplies by 1”, which is equivalent to removing the edge.

L1-regularization of \( w_{ij} \) values performs structure learning in UGM.

For general log-linear, each edge has multiple parameters \( w_{i,j,s,s'} \).

In this case we can use “group L1-regularization” for structure learning.

Each group will be all parameters \( w_{i,j,\cdot,\cdot} \) associated with an edge \( (i, j) \).
Structure Learning on Rain Data

Large $\lambda$ (and optimal tree):

Small $\lambda$: 
Structure Learning on USPS Digits

Structure learning of pairwise UGM with group-L1 on USPS digits:
Structure Learning on News Words

Group-L1 on newsgroups data:
Structure Learning on News Words

Group-L1 on newsgroups data:
Learning UGMs

2. Boltzmann Machines
“THE REVOLUTION WILL NOT BE SUPERVISED” PROMISES FACEBOOK’S YANN LECUN IN KICKOFF AI SEMINAR

POSTED: MARCH 6TH, 2018

http://engineering.nyu.edu/news/2018/03/06/revolution-will-not-be-supervised-promises-facebook-s-yann-lecun-kickoff-ai-seminar
Deep Density Estimation

- In 340 we discussed deep learning methods for supervised learning.

- Does it make sense to talk about deep unsupervised learning?

- Standard argument:
  - Human learning seems to be mostly unsupervised.
  - Supervision gives limited feedback: bits in a class label vs. an image.
  - Could we learn unsupervised models with much less data?

Cool Pictures Motivation for Deep Learning

- First layer of $z_i$ trained on 10 by 10 image patches:

- Visualization of second and third layers trained on specific objects:

- Many classes use these particular images to motivate deep neural networks.
  - But they’re not from a neural network: they’re from a DAG model.
Mixture of Independent Models

- Recall the mixture of independent models:

\[
p(x) = \sum_{c=1}^{k} p(z = c) \prod_{j=1}^{d} p(x_j \mid z = c).
\]

- Given \(z\), each variable \(x_j\) comes from some “nice” distribution.

- This is enough to model any distribution.
  - Just need to know cluster of example \(x\) and distribution of \(x_j\) given \(z\).
  - But not an efficient representation: number of cluster might need to be huge.
    - Need to learn each cluster independently (no “shared” information across clusters).
Latent DAG Model

- Consider the following model with binary $z_1$ and $z_2$:

- Have we gained anything?
  - We have 4 clusters based on two hidden variables.
  - Each cluster shares a parent/part with 2 of the other clusters.

- Hope is to achieve some degree of composition
  - Don’t need to re-learn basic things about the $x_j$ in each cluster.
  - Maybe one hidden $z_c$ models clusters, and another models correlations.
Consider the following model:

Now we have 16 clusters, in general we’ll have $2^k$ with $k$ hidden binary nodes.

- This discrete latent-factors give combinatorial number of mixtures.
  - You can think of each $z_c$ as a “part” that can be included or not (“binary PCA”).

- Usually assume $p(x_j \mid z_1, z_2, z_3, z_4)$ is a linear model (Gaussian, logistic, etc.).
  - Distributed representation where $x$ is made of parts $z$.
  - With $d$ visible $x_j$ and $k$ hidden $z_j$, we only have $dk$ parameters.

- Unfortunately, somewhat hard to use:
  - Combinatorial “explaining away” between $z_c$ value when conditioning on $x$.
  - Restricted Boltzmann Machines (RBMs) are a similar undirected model...
Boltzmann Machines

- **Boltzmann machines** are UGMs with binary latent variables:

  - Yet another **latent-variable model** for density estimation.
  - Hidden variables again give a combinatorial latent representation.
  - **Hard** to do anything in this model, even if you know all the $z$ (or $x$).
By restricting graph structure, some things get easier:

- **Restricted Boltzmann machines (RBMs):** edges only between the $x_j$ and $z_c$.

Bipartite structure allows **block Gibbs sampling** given one type of variable:

- **Conditional UGM** is disconnected.

Given visible $x$, we can sample each $z_c$ independently.

Given hidden $z$, we can sample each $x_j$ independently.
Restricted Boltzmann Machines

- The **RBM** graph structure leads to a joint distribution of the form

\[
p(x, z) = \frac{1}{Z} \left( \prod_{j=1}^{d} \phi_j(x_j) \right) \left( \prod_{c=1}^{k} \phi_c(z_c) \right) \left( \prod_{j=1}^{d} \prod_{c=1}^{k} \phi_{jc}(x_j, z_c) \right).
\]

- RBMs usually use a **log-linear** parameterization like

\[
p(x, z) \propto \exp \left( \sum_{j=1}^{d} x_j w_j + \sum_{c=1}^{k} z_c v_c + \sum_{j=1}^{d} \sum_{c=1}^{k} x_j w_{jc} z_c \right),
\]

for parameters \(w_j, v_c,\) and \(w_{jc}\) (first term would be different for continuous \(x_j\)).
Generating Digits with RBMs

Here are the samples generated by the RBM after training. Each row represents a mini-batch of negative particles (samples from independent Gibbs chains). 1000 steps of Gibbs sampling were taken between each of those rows.

http://deeplearning.net/tutorial/rbm.html
Generating Digits with RBMs

Visualizing each $z_c$'s interaction parameters ($w.c$ values) as images:

http://deeplearning.net/tutorial/rbm.html
Learning UGMs with Hidden Variables

- For **RBMs** we have hidden variables:

- With hidden ("nuissance") variables $z$ the observed likelihood has the form

\[
p(x) = \sum_z p(x, z) = \sum_z \tilde{p}(x, z) / Z
\]

\[
= \frac{1}{Z} \sum_z \tilde{p}(x, z) = \frac{Z(x)}{Z},
\]

where $Z(x)$ is the partition function of the conditional UGM given $x$.  
- $Z(x)$ is cheap in RBMs because the $z$ are independent given $x$.  

Learning UGMs with Hidden Variables

- This gives an observed NLL of the form
  \[- \log p(x) = - \log(Z(x)) + \log Z,\]
  where \(Z(x)\) sums over hidden \(z\) values, and \(Z\) sums over \(z\) and \(x\).

- The second term is convex but the first term is non-convex.
  - This is expected when we have hidden variables.

- With a log-linear parameterization, the gradient has the form
  \[- \nabla \log p(x) = - \mathbb{E}_{z \mid x}[F(X, Z)] + \mathbb{E}_{z, x}[F(X, Z)].\]

- For RBMs, first term is cheap due to independence of \(z\) given \(x\).
- We can approximate second term using block Gibbs sampling.
  - For other problems, you would also need to approximate first term.
Deep Belief Networks

- Deep belief networks are latent DAGs with more binary hidden layers:

- Data is at the bottom.
- First hidden layer could be “basic ingredients”.
- Second hidden layer could be general “parts”.
- Third hidden layer could be “abstract concept”.

![Deep Belief Network Diagram](image)
If we were conditioning on *top* layer:
- Sampling would be easy.

But we’re conditioning on the *bottom* layer:
- Everything is hard.
- There is combinatorial “explaining away”.

Common training method:
- Greedy “layerwise” training as a restricted Boltzmann machine.
Greedy Layerwise Training of Stacked RBMs

- Step 1: Train an RBM (alternating between block Gibbs and stochastic gradient)
Greedy Layerwise Training of Stacked RBMs

- Step 1: Train an RBM (alternating between block Gibbs and stochastic gradient)
- Step 2:
  - Fix first hidden layer values.
  - Train an RBM.
Greedy Layerwise Training of Stacked RBMs

- **Step 1:** Train an RBM (alternating between block Gibbs and stochastic gradient)
- **Step 2:**
  - Fix first hidden layer values.
  - Train an RBM.
- **Step 3:**
  - Fix second hidden layer values.
  - Train an RBM.
Deep Belief Networks

- Keep top as an RBM.
- For the other layers, use DAG parameters that implement block sampling.
  - Can sample by running block Gibbs on top layer for a while, then ancestral sampling.

Convolutional:

http://www.cs.toronto.edu/~rgrosse/icml09-cdbn.pdf
Deep Belief Networks

- Can add a class label to last layer.

- Can use “fine-tuning” as a feedforward neural network to refine weights.
  
  - [https://www.youtube.com/watch?v=KuPai0ogiHk](https://www.youtube.com/watch?v=KuPai0ogiHk)
Deep Boltzmann Machines

- **Deep Boltzmann machines** just keep as an undirected model.
  - Sampling is nicer: no explaining away within layers.
  - Variables in layer are independent given variables in layer above and below.
Deep Boltzmann Machines

Performance of **deep Boltzmann machine** on NORB data:

Figure 5: **Left:** The architecture of deep Boltzmann machine used for NORB. **Right:** Random samples from the training set, and samples generated from the deep Boltzmann machines by running the Gibbs sampler for 10,000 steps.

Summary

- **Approximate UGM learning:**
  1. Change objective function: pseudolikelihood.
  2. Approximate marginals: Monte Carlo or variational methods.

- **Structure learning in UGMs** with [group] L1-regularization.

- **Boltzmann machines** are UGMs with binary hidden variables.
  - Restricted Boltzmann machines only allow connections between hidden/visible.

- **Deep belief networks and Boltzmann machines** have layers of hidden variables.

- Next time: we’ll use these tools for supervised learning.