Probabilistic Models with Latent Variables

Goodfellow 13, 15, 20 Summary

박찬연

Probabilistic models with latent variables

- Generative model: $p_{\text{model}}(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{h}} p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{h})$ (Ch 20)
- Representation: $p_{\text{model}}(\boldsymbol{h} \mid \boldsymbol{x})$ (Ch 15)
- A linear factor model is the simplest probabilistic models with latent variables. (Ch 13)

13 Linear Factor Model

- Defined by the use of a stochastic, linear decoder function that generates $\boldsymbol{x}$ by adding noise to a linear transformation of $\boldsymbol{h}$.
- Data generation process
  - Sample $\boldsymbol{h}$ from $p(\boldsymbol{h})$,

$$p(\boldsymbol{h}) = \prod_i p(h_i)$$

  - Sample the real-valued observables given the factors.

$$\boldsymbol{x} = \boldsymbol{W}\boldsymbol{h} + \boldsymbol{b} + \text{noise}$$

13.1 Factor Analysis

- The latent variable prior is the unit variance Gaussian.

$$\mathbf{h} \sim \mathcal{N}(\boldsymbol{h}; \mathbf{0}, \boldsymbol{I})$$

- Noise is drawn from a diagonal covariance Gaussian.

$$\boldsymbol{\psi} = \text{diag}(\boldsymbol{\sigma}^2), \text{ with } \boldsymbol{\sigma}^2 = [\sigma_1^2, \sigma_2^2, \ldots, \sigma_n^2]^\top$$

- Latent variables capture the dependencies between x_i.
- $\boldsymbol{x}$ is a multivariate normal random variable.

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{x}; \boldsymbol{b}, \boldsymbol{W}\boldsymbol{W}^\top + \boldsymbol{\psi})$$

13.1 Probabilistic PCA

- Make the conditional variances σ_i^2 equal to each other.
- Conditional distribution: $\mathbf{x} \sim \mathcal{N}(\boldsymbol{x}; \boldsymbol{b}, \boldsymbol{W}\boldsymbol{W}^\top + \sigma^2 \boldsymbol{I})$
- Equivalently: $\mathbf{x} = \boldsymbol{W}\mathbf{h} + \boldsymbol{b} + \sigma\mathbf{z}\,, \;\; \mathbf{z} \sim \mathcal{N}(\boldsymbol{z}; \mathbf{0}, \boldsymbol{I})$
- Becomes PCA as $\sigma \to 0$.

13.2 Independent Component Analysis (ICA)

- Require $p(\boldsymbol{h})$ to be non-Gaussian.
- From a user-fixed $p(\boldsymbol{h})$, generates $\boldsymbol{x} = \boldsymbol{W}\boldsymbol{h}$ by nonlinear change of variables to determine $p(\boldsymbol{x})$.
- Example
  - Separating an observed signal into many underlying signals that are scaled and added together to form the observed data.
  - The signals are intended to be fully independent.

13.3 Slow Feature Analysis (SFA)

- Idea: important features change very slowly.
- Slowness principle

$$\lambda \sum_t L(f(\boldsymbol{x}^{(t+1)}), f(\boldsymbol{x}^{(t)}))$$

  - f is the feature extractor.
  - L is a loss function measuring the distance, usually the mean squared difference.

13.3 Slow Feature Analysis (SFA)

- SFA algorithm
  - Define $f(\boldsymbol{x}; \boldsymbol{\theta})$ to be linear and solve

$$\min_{\boldsymbol{\theta}} \mathbb{E}_t (f(\boldsymbol{x}^{(t+1)})_i - f(\boldsymbol{x}^{(t)})_i)^2 \quad \mathbb{E}_t f(\boldsymbol{x}^{(t)})_i = 0 \quad \mathbb{E}_t [f(\boldsymbol{x}^{(t)})_i^2] = 1$$

  - May be solved in closed form by a linear algebra package.
- So far the slowness principle has not become the basis of any state of the art applications.
  - Perhaps the slowness prior is too strong.
  - Slowness principle encourages the model to ignore the position of objects that have high velocity.

13.4 Sparse Coding

- Assume that the linear factors have Gaussian noise with isotropic precision β.

$$p(\boldsymbol{x} \mid \boldsymbol{h}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{W}\boldsymbol{h} + \boldsymbol{b}, \frac{1}{\beta}\boldsymbol{I})$$

- $p(\boldsymbol{h})$ is chosen to be one with sharp peaks near 0.
- The encoder is an optimization algorithm.

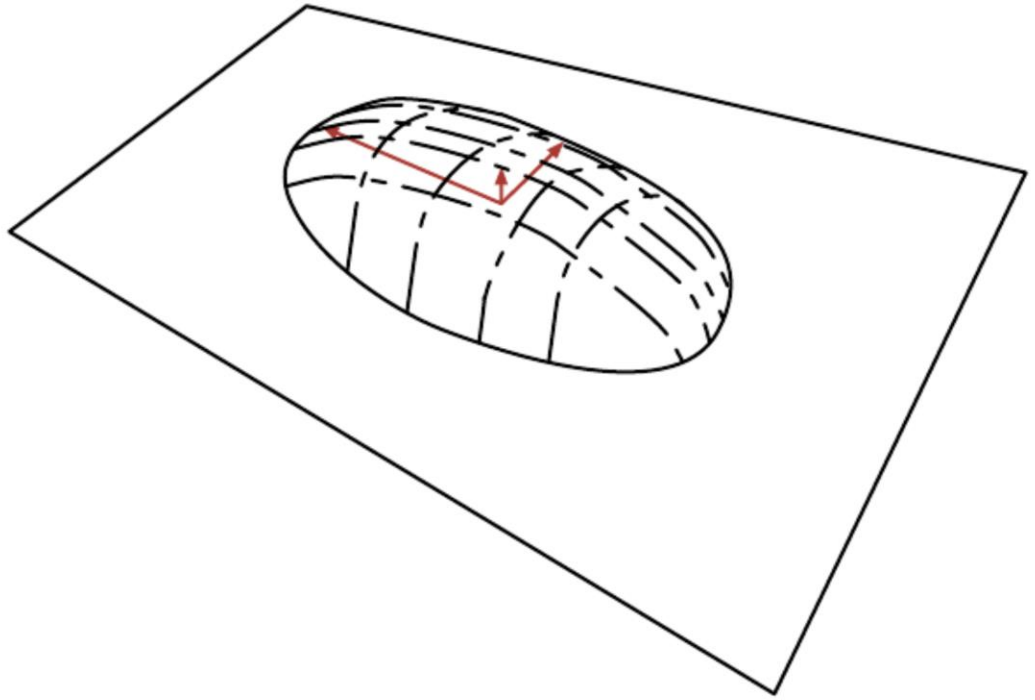
$$\begin{aligned}
\boldsymbol{h}^* = f(\boldsymbol{x}) &= \arg\max_{\boldsymbol{h}} p(\boldsymbol{h} \mid \boldsymbol{x}) \\
&= \arg\max_{\boldsymbol{h}} \log p(\boldsymbol{h} \mid \boldsymbol{x}) \quad p(h_i) = \text{Laplace}(h_i; 0, \frac{2}{\lambda}) = \frac{\lambda}{4} e^{-\frac{1}{2}\lambda|h_i|} \\
&= \arg\min_{\boldsymbol{h}} \lambda||\boldsymbol{h}||_1 + \beta||\boldsymbol{x} - \boldsymbol{W}\boldsymbol{h}||_2^2
\end{aligned}$$

13.4 Sparse Coding

- Due to the L^1 norm on $\boldsymbol{h}$, this procedure yield a sparse $\boldsymbol{h}^*$.
- To train the model, alternate between
  - minimization w.r.t $\boldsymbol{h}$
  - and minimization w.r.t. $\boldsymbol{W}$,
  - where we treat β as a hyperparameter.

13.5 Manifold Interpretation of PCA

- Linear factor models can be interpreted as learning a manifold.
- Think of PCA as learning a very crude approximation of the data manifold.



15 Representation Learning

- Shared representations are useful
  - to handle multiple modalities or domains,
  - or to transfer learned knowledge.
- A feed-forward network trained by supervised learning can be thought of as performing representation learning.
  - Layers except the last one, which is a linear classifier, learns to provide a representation to the classifier.
- We can design an objective function that encourages the elements of the representation vector $\boldsymbol{h}$ to be independent.
  - There can be a tradeoff between preserving as much information as possible and attaining nice properties such as independence.
- Representation learning via semi- or unsupervised learning can utilize unlabeled training data and resolve overfitting.

15.1 Greedy Layer-Wise Unsupervised Pretraining

- A canonical example of how a representation learned for one task can sometimes be useful for another task.
  - For example from unsupervised learning, trying to capture the shape of the input distribution, to supervised learning with the same input domain.
- Relies on a single-layer representation learning algorithm.
  - Such as RBM, a single-layer autoencoder, a sparse coding model.
- Now we know that greedy layer-wise pretraining is not required to train fully connected architectures.
  - But the unsupervised pretraining approach was the first method to succeed in training deep network architectures.

15.1 Greedy Layer-Wise Unsupervised Pretraining

- Greedy because it optimizes each piece independently, one piece at a time, rather than jointly optimizing all pieces.
- Pretraining because It is only a first step before a joint training algorithm is applied to fine-tune all the layers together.
- Can be used as initialization for
  - other unsupervised learning algorithms such as deep autoencoders,
  - and probabilistic models such as deep belief networks and deep Boltzmann machines.

15.1.1 Why Does Unsupervised Pretraining Work?

- There are two ideas underlying unsupervised pretraining.
- Regularizing effect on initial parameters
  - It is least well understood.
  - It is possible that pretraining initializes the model in a location that would otherwise be inaccessible.
  - Can expect the unsupervised training to be more effective when the initial representation is poor.
    - Especially useful when processing words, less useful when processing images.
  - Likely to be most useful when the function to be learned is extremely complicated.

15.1.1 Why Does Unsupervised Pretraining Work?

- Learning about input distribution can help learning the mapping from inputs to outputs.
  - This is better understood.
  - But not yet at a mathematical, theoretical level.
- Pretraining reduces the variance of the estimation process.
- Less is known about the effect of unsupervised pretraining in conjunction with contemporary approaches.

15.1.1 When Does Unsupervised Pretraining Work?

- For Boltzmann machines, greedy layer-wise unsupervised pretraining can yield substantial improvements in test error for classification tasks.
- On many other tasks, however, greedy layer-wise unsupervised pretraining either does not confer a benefit or even causes noticeable harm.
- Unsupervised learning has the disadvantage that it operates with two separate training phases, pretraining and fine-tuning, which makes choosing hyperparameters difficult.

15.1.1 When Does Unsupervised Pretraining Work?

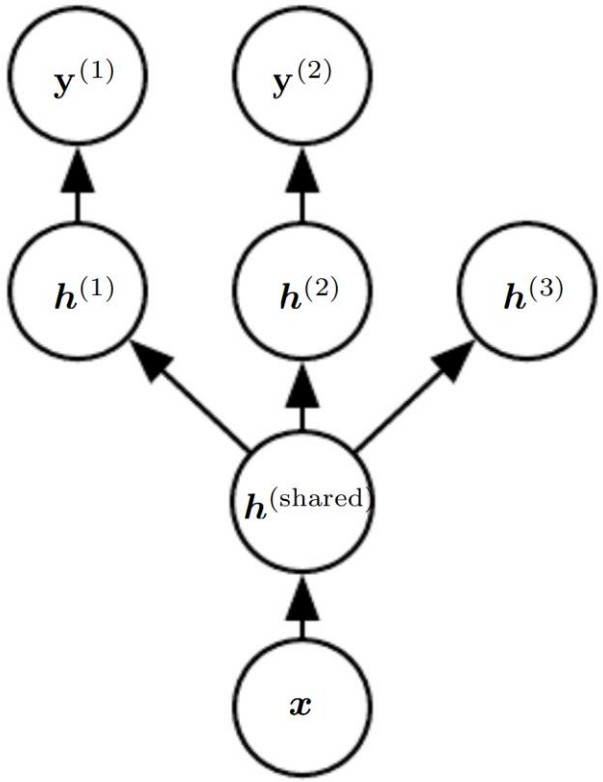
- Today, unsupervised pretraining has been largely abandoned, except in the field of natural language processing.
- Supervised learning, regularized with dropout or batchnorm, outperform unsupervised pretraining on medium-sized datasets.
  - CIFAR-10, MNIST: have ~5k labeled examples per class
- On extremely small datasets Bayesian methods outperform unsupervised pretraining.

15.2 Transfer Learning

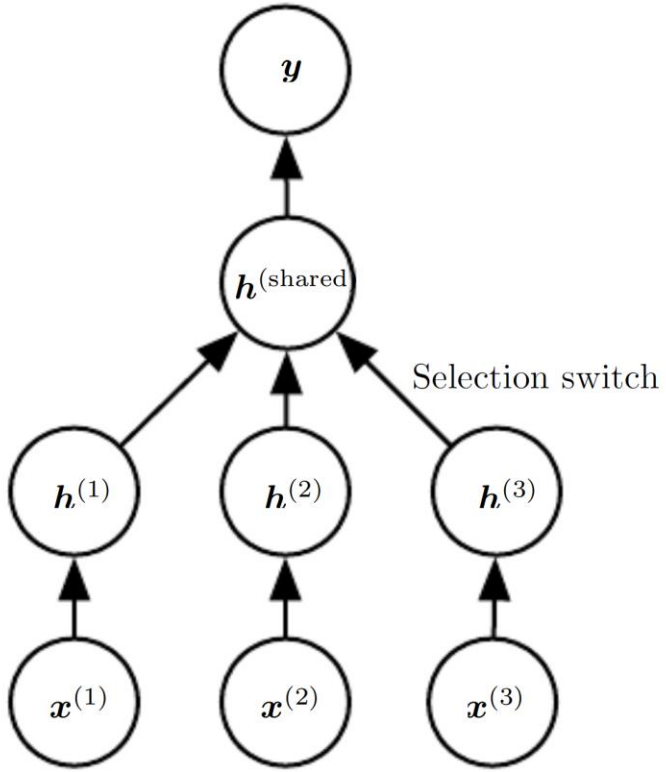
- The learner must perform two or more different tasks P_i.
- Assume that many of the factors that explain the variations in P_1 are relevant to the variations that need to be captured for learning P_2.
  - Many visual categories share **low-level** notions such as
    - edges and visual shapes,
    - the effects of geometric changes,
    - changes in lighting, etc.
  - Sometimes what is shared among the different tasks is not the semantics of the input but the semantics of the output, when it makes more sense to share the **upper layers** (near the output) of the neural network, and have a task-specific preprocessing.

15.2 Transfer Learning

- Many visual categories share low-level notions such as edges and visual shapes, the effects of geometric changes, changes in lighting, etc.



- Sometimes what is shared among the different tasks is not the semantics of the input but the semantics of the output, when it makes more sense to share the upper layers (near the output) of the neural network, and have a task-specific preprocessing.



15.2 Domain Adaptation

- The task remains the same, but the input distribution is slightly different.
- For example, a sentiment predictor trained on **customer reviews of media content** such as books, videos, and music is later used to analyze **comments about consumer electronics** such as televisions or smartphones.

15.2 Other Forms of Transfer Learning

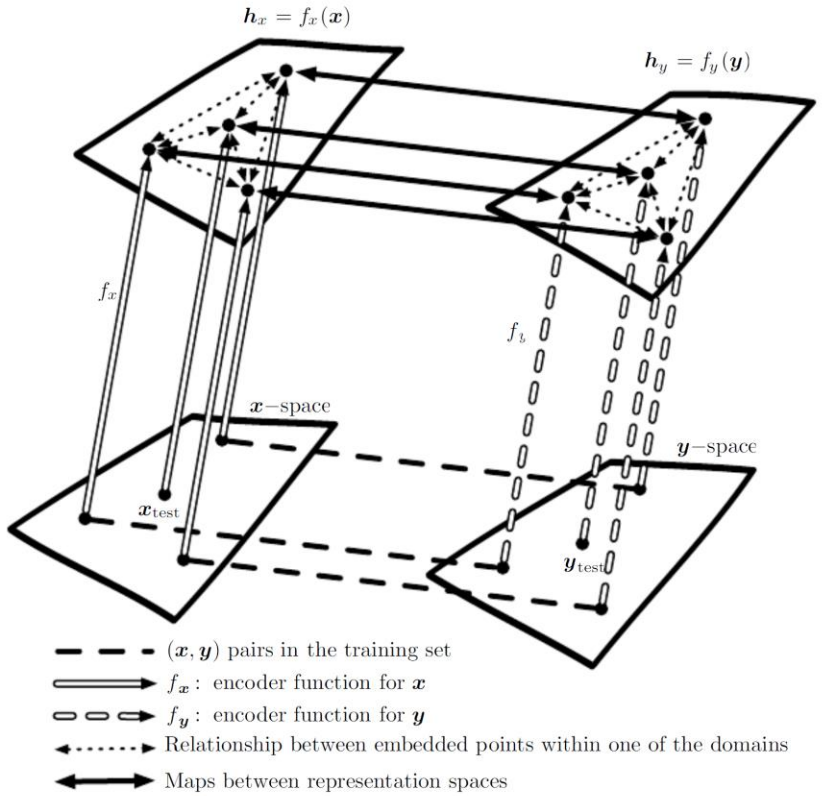
- Concept Drift
  - A form of transfer learning due to gradual changes in the data distribution over time.
- One-shot learning
  - Only one labeled example of the transfer task is given.
  - Possible because the representation is learned to cleanly **separate** the underlying classes **during the first stage**.
  - Then during **the transfer learning stage** only one labeled example is needed to **infer the label** of many possible test examples that all cluster around the same point in representation space.

15.2 Other Forms of Transfer Learning

- Zero-shot learning, zero-data learning
  - No labeled examples are given at all.
  - Only possible because additional information has been exploited during training.
  - Includes three random variables
    - The traditional inputs, $\boldsymbol{x}$,
    - The traditional outputs or targets, $\boldsymbol{y}$,
    - An additional random variable describing the **task**, T.
      - For example, a representation of a question such as “Is there a cat in this image?”.
  - The model is trained to estimate $\mathrm{p}(\boldsymbol{y}|\boldsymbol{x}, T)$.

15.2 Zero-shot learning & Zero-data Learning

- Zero-shot learning between similar input domains.
  - A word $\boldsymbol{x}_{\text{test}}$ and another word $\boldsymbol{y}_{\text{test}}$ from two different languages.
- Zero-data learning between different domains.
  - An image $\boldsymbol{x}_{\text{test}}$ and a word $\boldsymbol{y}_{\text{test}}$.



15.3 Semi-Supervised Disentangling of Causal Factors

- Ideal representation is the one whose features or **directions** in the feature space correspond to different **causes**, thereby disentangling the causes.
- Two main strategies for dealing with a large number of causes
  - Do supervised learning and unsupervised learning at the same time.
  - Use much larger representations if using purely unsupervised learning.
- An emerging strategy is designing a model that **learns** how to determine what is salient, such as generative adversarial networks (GANs).
- The benefit of learning causal factors is that the causal mechanisms remain **invariant** when considering changes in distribution of the cause due to different domains, temporal non-stationarity, or changes in the nature of the task.

15.4 Distributed Representation

- Representations composed of many elements that can be set separately from each other.
  - n features with k values describe k^n different concepts,
  - whereas symbolic representations such as one-hot representation can only divide n different regions in input space.
- Examples of learning algorithms based on non-distributed representations
  - Clustering methods including k-means algorithm
  - k-nearest neighbors algorithms
  - Decision trees
  - Gaussian mixtures
  - Kernel machines
  - Language models based on n-grams

15.4 Distributed Representation

- Examples of distributed representation
  - Hidden units in a deep convolutional network trained on the ImageNet data.
  - Latent space of a deep convolutional generative adversarial network (DCGAN) trained on faces.
  - Word2Vec

15.5 Exponential Gains from Depth

- Deep distributed representations
  - The higher level features (functions of the input) or factors (generative causes) are obtained through the composition of many **nonlinearities**.
  - This can give an exponential boost to statistical efficiency **on top of** the exponential boost given by using a distributed representation.

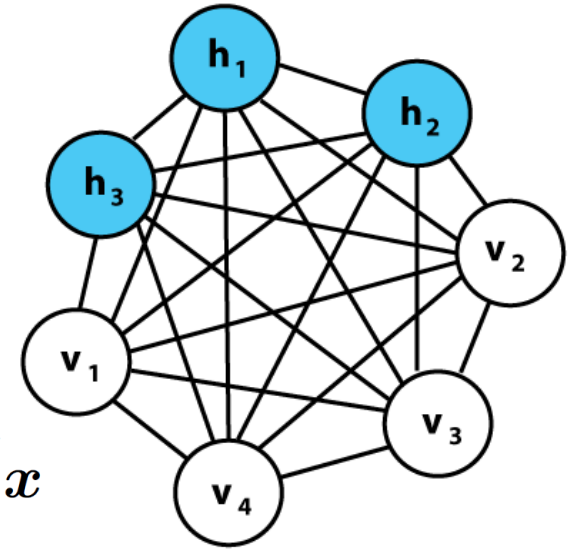
15.6 Providing Clues to Discover Underlying Causes

- To use unlabeled data, representation learning makes use of implicit prior beliefs about the underlying factors that act as generic regularization strategies.
  - Smoothness and linearity
  - Multiple explanatory factors, causal factors, and hierarchical organization of explanatory factors
  - Shared factors across tasks
  - Manifold of probability mass and natural clustering
  - Temporal and spatial coherence from slowness principle
  - Sparsity in the sense that most features should not be relevant to describing most inputs
  - Simplicity of factor dependencies

20. Deep Generative Models

- All of the models represent probability distributions over multiple variables in some way.
- Some allow the probability distribution function to be evaluated explicitly.
  - Boltzmann machines
- Others do not allow the evaluation of the probability distribution function, but support drawing samples from the distribution.
  - GAN
- Some of these models are structured probabilistic models (graphical models) described in terms of graphs and factors.

20.1 Boltzmann Machines



- An energy-based model

$$P(\boldsymbol{x}) = \frac{\exp\left(-E(\boldsymbol{x})\right)}{Z} \qquad E(\boldsymbol{x}) = -\boldsymbol{x}^\top \boldsymbol{U}\boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{x}$$

- When all variables are observed, the probability of one unit being on is given by a linear model.
- Becomes more powerful when having non-observed variables, or latent variables.
  - In this case the Boltzmann machine becomes a universal approximator over discrete variables.

20.1 Boltzmann machine

- Learning algorithms for Boltzmann machines are usually based on maximum likelihood.
- All Boltzmann machines have an intractable partition function.
  - So the maximum likelihood gradient must be approximated using MCMC algorithms for maximizing the log-likelihood with an intractable partition function using gradient ascent, for example contrastive divergence (CD, or CD-k to indicate CD with k Gibbs steps)

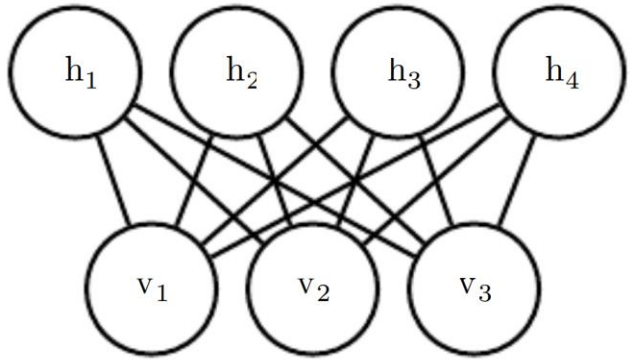
20.2 Restricted Boltzmann Machine

- Has a bipartite graph model.

$$P(\mathbf{v} = \boldsymbol{v}, \mathbf{h} = \boldsymbol{h}) = \frac{1}{Z} \exp\left(-E(\boldsymbol{v}, \boldsymbol{h})\right)$$

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\boldsymbol{b}^\top \boldsymbol{v} - \boldsymbol{c}^\top \boldsymbol{h} - \boldsymbol{v}^\top \boldsymbol{W} \boldsymbol{h}$$

$$Z = \sum_{\boldsymbol{v}} \sum_{\boldsymbol{h}} \exp\left\{-E(\boldsymbol{v}, \boldsymbol{h})\right\}$$



- Naïve method of computing Z is intractable, implying $P(\boldsymbol{v})$ is also intractable to evaluate.

20.2.1 Conditional Distributions

- The bipartite graph structure of the RBM has the very special property that its conditional distributions $P(\boldsymbol{h} \mid \boldsymbol{v})$ and $P(\boldsymbol{v} \mid \boldsymbol{h})$ are factorial and relatively simple to compute and to sample from.

$$
\begin{aligned}
P(\boldsymbol{h} \mid \boldsymbol{v}) &= \frac{P(\boldsymbol{h}, \boldsymbol{v})}{P(\boldsymbol{v})} \\
&= \frac{1}{P(\boldsymbol{v})} \frac{1}{Z} \exp\left\{\boldsymbol{b}^\top \boldsymbol{v} + \boldsymbol{c}^\top \boldsymbol{h} + \boldsymbol{v}^\top \boldsymbol{W} \boldsymbol{h}\right\} \\
&= \frac{1}{Z'} \exp\left\{\boldsymbol{c}^\top \boldsymbol{h} + \boldsymbol{v}^\top \boldsymbol{W} \boldsymbol{h}\right\} \\
&= \frac{1}{Z'} \exp\left\{\sum_{j=1}^{n_h} c_j h_j + \sum_{j=1}^{n_h} \boldsymbol{v}^\top \boldsymbol{W}_{:,j} \boldsymbol{h}_j\right\} \\
&= \frac{1}{Z'} \prod_{j=1}^{n_h} \exp\left\{c_j h_j + \boldsymbol{v}^\top \boldsymbol{W}_{:,j} \boldsymbol{h}_j\right\}
\end{aligned}
$$

$$
\begin{aligned}
P(h_j = 1 \mid \boldsymbol{v}) &= \frac{\tilde{P}(h_j = 1 \mid \boldsymbol{v})}{\tilde{P}(h_j = 0 \mid \boldsymbol{v}) + \tilde{P}(h_j = 1 \mid \boldsymbol{v})} \\
&= \frac{\exp\left\{c_j + \boldsymbol{v}^\top \boldsymbol{W}_{:,j}\right\}}{\exp\{0\} + \exp\left\{c_j + \boldsymbol{v}^\top \boldsymbol{W}_{:,j}\right\}} \\
&= \sigma\left(c_j + \boldsymbol{v}^\top \boldsymbol{W}_{:,j}\right).
\end{aligned}
$$

$$
P(\boldsymbol{h} \mid \boldsymbol{v}) = \prod_{j=1}^{n_h} \sigma\left((2\boldsymbol{h} - 1) \odot (\boldsymbol{c} + \boldsymbol{W}^\top \boldsymbol{v})\right)_j
$$

$$
P(\boldsymbol{v} \mid \boldsymbol{h}) = \prod_{i=1}^{n_v} \sigma\left((2\boldsymbol{v} - 1) \odot (\boldsymbol{b} + \boldsymbol{W}\boldsymbol{h})\right)_i
$$

20.2.2 Training RBM

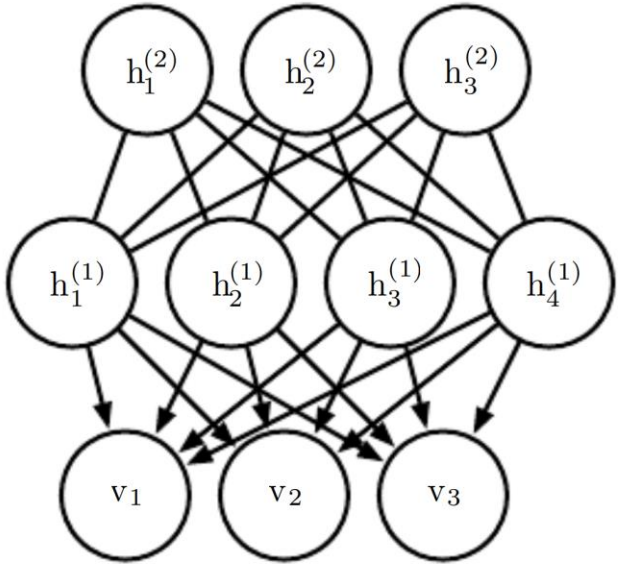
- RBM admits efficient evaluation and differentiation of $\tilde{P}(\boldsymbol{v})$ and efficient MCMC sampling in the form of block Gibbs sampling.
- Can be trained with any technique for training models that have intractable partition functions.
  - This includes contrastive divergence (CD), stochastic maximum likelihood (SML, or persistent contrastive divergence (PCD)), ratio matching and so on.
- Compared to other undirected models used in deep learning, the RBM is relatively straightforward to train because we can compute $P(\boldsymbol{h} \mid \boldsymbol{v})$ exactly in closed form.

20.3 Deep Belief Networks

- The introduction of deep belief networks (DBNs) in 2006 began the current deep learning renaissance.
- Prior to the introduction of DBNs, deep models were considered too difficult to optimize, kernel machines with convex objective functions dominated the research landscape.
- DBNs demonstrated that deep architectures can be successful, by outperforming kernelized SVMs on the MNIST dataset.
- Today, DBNs are rarely used, even compared to other unsupervised or generative learning algorithms.
- But they are still deservedly recognized for their important role in deep learning history.

20.3 DBNs

- Generative models with several layers of latent variables.
- The latent variables are typically binary, while the visible units may be binary or real.
- There are no intra-layer connections.
- The connections between the top two layers are undirected.
- The connections between all other layers are directed.



20.3 DBNs

- The probability distribution represented by the DBN

$$P(\boldsymbol{h}^{(l)}, \boldsymbol{h}^{(l-1)}) \propto \exp\left(\boldsymbol{b}^{(l)\top}\boldsymbol{h}^{(l)} + \boldsymbol{b}^{(l-1)\top}\boldsymbol{h}^{(l-1)} + \boldsymbol{h}^{(l-1)\top}\boldsymbol{W}^{(l)}\boldsymbol{h}^{(l)}\right),$$

$$P(h_i^{(k)} = 1 \mid \boldsymbol{h}^{(k+1)}) = \sigma\left(b_i^{(k)} + \boldsymbol{W}_{:,i}^{(k+1)\top}\boldsymbol{h}^{(k+1)}\right) \forall i, \forall k \in 1, \ldots, l-2,$$

$$P(v_i = 1 \mid \boldsymbol{h}^{(1)}) = \sigma\left(b_i^{(0)} + \boldsymbol{W}_{:,i}^{(1)\top}\boldsymbol{h}^{(1)}\right) \forall i.$$

- In the case of real-valued visible units,

$$\mathbf{v} \sim \mathcal{N}\left(\boldsymbol{v}; \boldsymbol{b}^{(0)} + \boldsymbol{W}^{(1)\top}\boldsymbol{h}^{(1)}, \boldsymbol{\beta}^{-1}\right)$$

20.3 DBNs

- To generate a sample,
  - first run several steps of Gibbs sampling on the top two hidden layers,
  - then use a single pass of ancestral sampling through the rest of the model to draw a sample from the visible units.

20.3 DBNs

- Have many of the problems associated with both directed and undirected models.
- Inference is intractable
  - due to the explaining away effect within each directed layer,
  - and due to the interaction between the two hidden layers with undirected connections.
- Evaluating or maximizing the standard ELBO on the log-likelihood is also intractable
  - because the ELBO takes the expectation of cliques of size equal to the network width.
- Evaluating or maximizing the log-likelihood requires
  - not just confronting the problem of intractable inference to marginalize out the latent variables,
  - but also the problem of an intractable partition function within the undirected model of the top two layers.

20.3 DBNs

- To train DBNs,
  - begin by training an RBM to maximize $\mathbb{E}_{\mathbf{v} \sim p_{\text{data}}} \log p(\boldsymbol{v})$ using CD or SML,
  - then the RBM defines the first layer of the DBN.
  - Next a second RBM is trained to approximately maximize

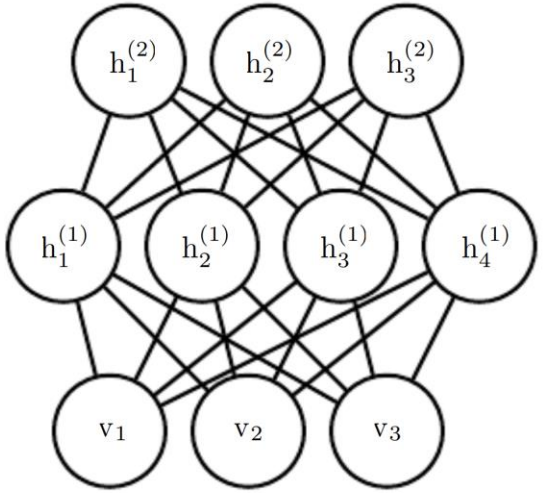
$$\mathbb{E}_{\mathbf{v} \sim p_{\text{data}}} \mathbb{E}_{\mathbf{h}^{(1)} \sim p^{(1)}(\boldsymbol{h}^{(1)} | \boldsymbol{v})} \log p^{(2)}(\boldsymbol{h}^{(1)})$$

  - And repeat adding as many as layers to the DBN as desired.
  - This can be justified as increasing a variational lower bound on the log-likelihood of the data under the DBN.
- In most applications, no effort is made to jointly train the DBN after the greedy layer-wise procedure is complete.

20.3 DBNs

- The trained DBN may be used directly as a generative model.
- But most of the interest in DBNs arose from their ability to improve classification models by taking the weights from the DBN and using them to define an MLP.
- The term “deep belief network” may cause some confusion because the term “belief network” is sometimes used to refer to purely directed models, while DBNs contain an undirected layer.

20.4 Deep Boltzmann Machines



- Entirely undirected model unlike DBN.
- Has several layers of latent variables unlike RBM.
- Within each layer, like RBM,
  - each variable is mutually independent,
  - and is conditioned on the variables in the neighboring layers.
- Typically contain only binary units like RBMs and DBNs, but it is straightforward to include real-valued visible units.

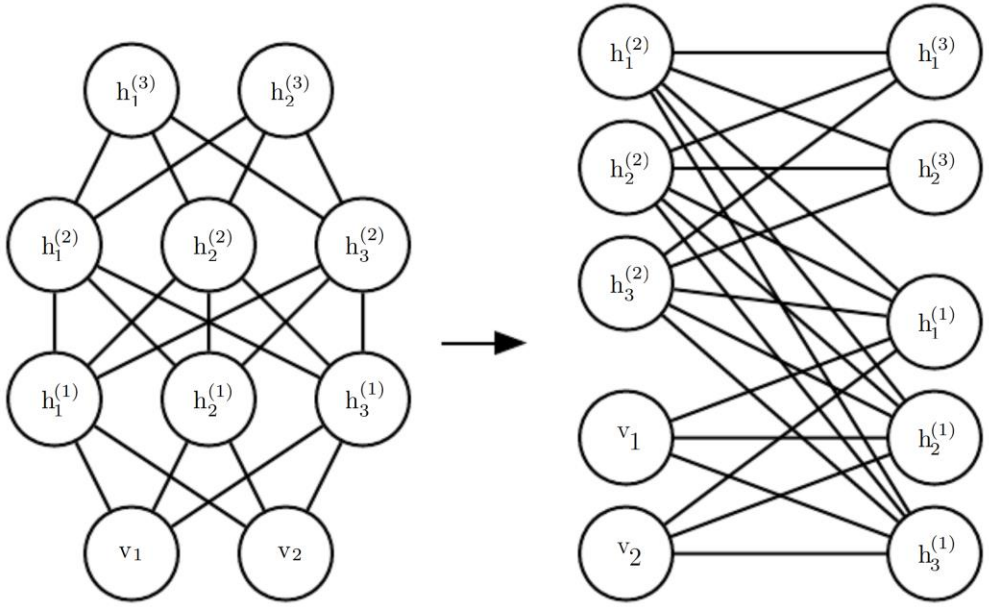
20.4 DBMs

- An energy-based model

$$P\left(\boldsymbol{v}, \boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \boldsymbol{h}^{(3)}\right) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left(-E(\boldsymbol{v}, \boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \boldsymbol{h}^{(3)}; \boldsymbol{\theta})\right)$$

$$E(\boldsymbol{v}, \boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \boldsymbol{h}^{(3)}; \boldsymbol{\theta}) = -\boldsymbol{v}^{\top} \boldsymbol{W}^{(1)} \boldsymbol{h}^{(1)} - \boldsymbol{h}^{(1)\top} \boldsymbol{W}^{(2)} \boldsymbol{h}^{(2)} - \boldsymbol{h}^{(2)\top} \boldsymbol{W}^{(3)} \boldsymbol{h}^{(3)}$$

- Can be organized into a bipartite graph.
  - Conditional independence between variables of odd layers and variables of even layers.



20.4 DBMs

- The bipartite structure means that we can apply the same equations used for an RBM to determine the conditional distributions in a DBM.
- The units within a layer are conditionally independent from each other given the values of the neighboring layers.
  - For example, when a DBM has two hidden layers,

$$P(v_i = 1 \mid \boldsymbol{h}^{(1)}) = \sigma\left(\boldsymbol{W}^{(1)}_{i,:}\boldsymbol{h}^{(1)}\right)$$

$$P(h^{(1)}_i = 1 \mid \boldsymbol{v}, \boldsymbol{h}^{(2)}) = \sigma\left(\boldsymbol{v}^\top \boldsymbol{W}^{(1)}_{:,i} + \boldsymbol{W}^{(2)}_{i,:}\boldsymbol{h}^{(2)}\right)$$

$$P(h^{(2)}_k = 1 \mid \boldsymbol{h}^{(1)}) = \sigma\left(\boldsymbol{h}^{(1)\top}\boldsymbol{W}^{(2)}_{:,k}\right)$$

- The bipartite structure makes Gibbs sampling in a DBM efficient.
  - Gibbs sampling can be divided into two blocks of updates.

20.4.1 Interesting Properties

- The posterior distribution $P(\boldsymbol{h}|\boldsymbol{v})$ is simpler compared to DBNs.
  - The simplicity allows richer approximation.
  - DBNs have interactions between hidden units within the same layer because of the explaining-away effect, but a heuristic choice of Q(h) to obtain a variational lower bound of the posterior ignores those interactions, for example.
  - In DBMs, all of the hidden units within a layer are conditionally independent given the other layers. This lack of intra-layer interaction makes it possible to use fixed point equations to actually optimize the variational lower bound and find the true optimal mean field expectations.
- Sampling from DMBs is relatively difficult.
  - DBNs only need to use MCMC sampling in the top pair of layers.
  - For DBMs it is necessary to use MCMC across all layers.

20.4.2 DBM Mean Field Inference

- The conditional distribution over one DBM layer given the neighboring layers is factorial.
- The DBM posterior distribution over their hidden units is easy to approximate with a variational approximation, specifically a mean field approximation.
  - The mean field approximation is a simple form of variational inference, where we restrict the approximating distribution to fully factorial distributions.
  - In the case of the mean field approximation, the approximating family is the set of distributions where the hidden units are conditionally independent.

20.4.2 DBM Mean Field Inference

- The mean field approach for the example with two hidden layers
- Let $Q(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}|\boldsymbol{v})$ be the approximation of $P(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}|\boldsymbol{v})$.
  - The mean field assumption

$$Q(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)} \mid \boldsymbol{v}) = \prod_j Q(h_j^{(1)} \mid \boldsymbol{v}) \prod_k Q(h_k^{(2)} \mid \boldsymbol{v})$$

  - Minimize

$$\mathrm{KL}(Q\|P) = \sum_{\boldsymbol{h}} Q(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)} \mid \boldsymbol{v}) \log \left(\frac{Q(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)} \mid \boldsymbol{v})}{P(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)} \mid \boldsymbol{v})} \right)$$

  - Parametrize Q as a product of Bernoulli distributions.

$$Q(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)} \mid \boldsymbol{v}) = \prod_j (\hat{h}_j^{(1)})^{h_j^{(1)}} (1 - \hat{h}_j^{(1)})^{(1-h_j^{(1)})} \times \prod_k (\hat{h}_k^{(2)})^{h_k^{(2)}} (1 - \hat{h}_k^{(2)})^{(1-h_k^{(2)})}$$

20.4.2 DBM Mean Field Inference

- Choosing a member of this $Q(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}|\boldsymbol{v})$ family that best fits P.

$$\tilde{q}(h_i \mid \boldsymbol{v}) = \exp\left(\mathbb{E}_{\mathbf{h}_{-i} \sim q(\mathbf{h}_{-i}|\boldsymbol{v})} \log \tilde{p}(\boldsymbol{v}, \boldsymbol{h})\right)$$

- Applying the above, we obtain the update rule.

$$\hat{h}_j^{(1)} = \sigma\left(\sum_i v_i W_{i,j}^{(1)} + \sum_{k'} W_{j,k'}^{(2)} \hat{h}_{k'}^{(2)}\right), \quad \forall j$$

$$\hat{h}_k^{(2)} = \sigma\left(\sum_{j'} W_{j',k}^{(2)} \hat{h}_{j'}^{(1)}\right), \quad \forall k.$$

- At a fixed point, we have a local maximum of the variational lower bound $\mathcal{L}(Q)$.

20.4.2 DBM Mean Field Inference

- These fixed point update equations define an iterative algorithm where we alternate updates of $\hat{h}_j^{(1)}$ and updates of $\hat{h}_k^{(2)}$.
- On small problems such as MNIST, ~10 iterations can be sufficient.

20.4.3 DBM Parameter Learning

- Learning in the DBM must confront both
  - the challenge of an intractable posterior distribution,
  - and the challenge of an intractable partition function.
- Variational inference allows the construction of $Q(\boldsymbol{h}|\boldsymbol{v})$ that approximate the intractable posterior distribution $P(\boldsymbol{h}|\boldsymbol{v})$.
- Learning then proceeds by maximizing $\mathcal{L}(\boldsymbol{v}, Q, \boldsymbol{\theta})$, the variational lower bound on the intractable log-likelihood, $\log P(\boldsymbol{v}; \boldsymbol{\theta})$, w.r.t $\boldsymbol{\theta}$.

20.4.3 DBM Parameter Learning

- For a DBM with two hidden layers,

$$\mathcal{L}(Q, \boldsymbol{\theta}) = \sum_i \sum_{j'} v_i W^{(1)}_{i,j'} \hat{h}^{(1)}_{j'} + \sum_{j'} \sum_{k'} \hat{h}^{(1)}_{j'} W^{(2)}_{j',k'} \hat{h}^{(2)}_{k'} - \log Z(\boldsymbol{\theta}) + \mathcal{H}(Q)$$

- This contains the log partition function $\log Z(\boldsymbol{\theta})$.
- Evaluating the probability of a DBM requires approximate methods such as annealed importance sampling (AIS).
- Training the model requires approximations to the gradient of the log partition function, typically stochastic maximum likelihood/persistet contrastive divergence (SML/PCD).
  - Use variational stochastic maximum likelihood as applied to the DBM, Algorithm 20.1.

20.4.3 DBM Parameter Learning

- Methods not applicable to DBM
  - Techniques such as pseudolikelihood require the ability to evaluate the unnormalized probabilities, rather than merely obtain a variational lower bound on them.
  - Contrastive divergence is slow for deep Boltzmann machines because they do not allow efficient sampling of the hidden units given the visible units — instead, contrastive divergence would require burning in a Markov chain every time a new negative phase sample is needed.

20.4.4 Layer-Wise Pretraining

- Unfortunately, training a DBM using SML/PCD from a random initialization usually results in failure.
  - In some cases, the model fails to learn to represent the distribution adequately.
  - In other cases, the DBM may represent the distribution well, but with no higher likelihood than could be obtained with just an RBM.
- The original and most popular method for overcoming the joint training problem of DBMs is greedy layer-wise pretraining.
  - In this method, each layer of the DBM is trained in isolation as an RBM.
- The DBM may then be trained with PCD.
  - Typically PCD training will make only a small change in the model's parameters
  - and its performance as measured by the log-likelihood it assigns to the data, or its ability to classify inputs.

20.4.5 Jointly Training DBMs

- Classic DBMs require greedy unsupervised pretraining and a separate MLP-based classifier on top of the hidden features they extract.
- This has some undesirable properties.
  - It is hard to track performance during training because we cannot evaluate properties of the full DBM while training the first RBM.
  - Implementations of DBMs need to have many different components
    - for CD training of individual RBMs,
    - PCD training of the full DBM,
    - and training based on back-propagation through the MLP.
  - Thus, it is hard to tell how well our hyperparameters are working until quite late in the training process.

20.4.5 Jointly Training DBMs

- Two main ways to resolve the joint training problem
  - Centered DBM
    - Unable to compete with appropriately regularized MLPs as a classifier.
  - Multi-prediction DBM
    - Does not lead to good likelihood or samples.
    - Does lead to superior classification and ability to reason well about missing inputs.

20.5 Boltzmann Machines for Real-Valued Data

- Gaussian-Bernoulli RBMs
  - RBM with binary hidden units and real-valued visible units
- Undirected Models of Conditional Covariance
  - Gaussian RBM inductive bias is not well suited to some types of data.
    - Information in natural images is embedded in the covariance between pixels rather than in the raw pixel values, the Gaussian RBM only models the conditional mean.
  - These models include the mean and covariance RBM (mcRBM), the mean-product of t-distribution (mPoT) model and the spike and slab RBM (ssRBM).

20.5 Boltzmann Machines for Real-Valued Data

- mcRBM
  - Uses its hidden units to independently encode the conditional mean and covariance of all observed units.

$$E_{\mathrm{mc}}(\boldsymbol{x}, \boldsymbol{h}^{(m)}, \boldsymbol{h}^{(c)}) = E_{\mathrm{m}}(\boldsymbol{x}, \boldsymbol{h}^{(m)}) + E_{\mathrm{c}}(\boldsymbol{x}, \boldsymbol{h}^{(c)})$$

$$E_{\mathrm{m}}(\boldsymbol{x}, \boldsymbol{h}^{(m)}) = \frac{1}{2}\boldsymbol{x}^{\top}\boldsymbol{x} - \sum_j \boldsymbol{x}^{\top}\boldsymbol{W}_{:,j} h_j^{(m)} - \sum_j b_j^{(m)} h_j^{(m)} \qquad E_{\mathrm{c}}(\boldsymbol{x}, \boldsymbol{h}^{(c)}) = \frac{1}{2}\sum_j h_j^{(c)} \left(\boldsymbol{x}^{\top}\boldsymbol{r}^{(j)}\right)^2 - \sum_j b_j^{(c)} h_j^{(c)}$$

$$p_{\mathrm{mc}}(\boldsymbol{x} \mid \boldsymbol{h}^{(m)}, h^{(c)}) = \mathcal{N}\left(\boldsymbol{x}; \boldsymbol{C}_{\boldsymbol{x}|\boldsymbol{h}}^{\mathrm{mc}} \left(\sum_j \boldsymbol{W}_{:,j} h_j^{(m)}\right), \boldsymbol{C}_{\boldsymbol{x}|\boldsymbol{h}}^{\mathrm{mc}}\right) \qquad \boldsymbol{C}_{\boldsymbol{x}|\boldsymbol{h}}^{\mathrm{mc}} = \left(\textstyle\sum_j h_j^{(c)} \boldsymbol{r}^{(j)} \boldsymbol{r}^{(j)\top} + \boldsymbol{I}\right)^{-1}$$

  - Sampling from $p_{\mathrm{mc}}(\boldsymbol{x}|\boldsymbol{h}^{(m)}, \boldsymbol{h}^{(c)})$ requires computing $(\boldsymbol{C}^{\mathrm{mc}})^{-1}$ at every iteration of learning, which can be impractical.
  - Avoid it by sampling directly from the marginal $p_{\mathrm{mc}}(\boldsymbol{x})$ using Hamiltonian (hybrid) Monte Carlo.

20.6 Convolutional Boltzmann Machines

- Unclear how to generalize a pooling operation of a CNN to the setting of energy-based models.
- CNN work well with inputs of many different sizes; for Boltzmann machines, it is difficult to change the input size.
- Pixels at the boundary of the image pose some difficulty.
  - Both doing zero-padding and not doing zero-padding have issues.

20.7 Boltzmann Machines for Structured or Sequential Outputs

- A natural way to represent the relationships between the entries in a structured output $\boldsymbol{y}$ is to use a probability distribution $p(\boldsymbol{y}|\boldsymbol{x})$.
- Boltzmann machines extended to model conditional distributions, can supply this probabilistic model.
- For sequence modeling, conditional Boltzmann machines can represent factors of the form $p(\boldsymbol{x}^{(t)}|\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t-1)})$.
  - Conditional RBM modeling $p(\boldsymbol{x}^{(t)}|\boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(t-m)})$ for small m.
  - RNN-RBM sequence model that consists of an RNN that emits the RBM parameters for each time step.

20.8 Other Boltzmann Machines

- Discriminative RBM to maximize $p(y|\boldsymbol{v})$
  - Not as powerful as MLP.
- Higher-order Boltzmann machines in energy functions

20.9 Backprop through Random Operations

- Reparametrization trick
  - $y \sim \mathcal{N}(\mu, \sigma^2) \rightarrow y = \mu + \sigma \cdot z,\ z \sim \mathcal{N}(0,1)$
- More generally, $\boldsymbol{y} \sim p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) \rightarrow \boldsymbol{y} = f(\boldsymbol{z}; \boldsymbol{x}, \boldsymbol{\theta})$
  - f is continuous and differentiable everywhere.
  - $\boldsymbol{\theta}$ must not be a function of $\boldsymbol{z}$ and vice versa.
  - $\boldsymbol{z}$ from unit uniform or unit Gaussian.

20.9.1 Backprop through Discrete Stochastic Operations

- $\boldsymbol{y} = f(\boldsymbol{z}; \boldsymbol{x}, \boldsymbol{\theta})$
  - When $\boldsymbol{y}$ is discrete, f must be a step function.
  - Then the derivative of any cost function $J(\boldsymbol{y})$ do not give any information.
- REINFORCE algorithm
  - The expected cost $\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})} J(f(\boldsymbol{z}; \boldsymbol{x}, \boldsymbol{\theta}))$ is often a smooth function.
  - It can be estimated without bias using a Monte Carlo average.

$$\mathbb{E}_z[J(\boldsymbol{y})] = \sum_{\boldsymbol{y}} J(\boldsymbol{y}) p(\boldsymbol{y})$$

$$\begin{aligned}
\frac{\partial \mathbb{E}[J(\boldsymbol{y})]}{\partial \boldsymbol{\omega}} &= \sum_{y} J(\boldsymbol{y}) \frac{\partial p(\boldsymbol{y})}{\partial \boldsymbol{\omega}} \\
&= \sum_{y} J(\boldsymbol{y}) p(\boldsymbol{y}) \frac{\partial \log p(\boldsymbol{y})}{\partial \boldsymbol{\omega}} \\
&\approx \frac{1}{m} \sum_{\mathbf{y}^{(i)} \sim p(\boldsymbol{y}),\, i=1}^{m} J(\boldsymbol{y}^{(i)}) \frac{\partial \log p(\boldsymbol{y}^{(i)})}{\partial \boldsymbol{\omega}}
\end{aligned}$$

20.9.1 Backprop through Discrete Stochastic Operations

- One issue with the simple REINFORCE estimator is that it has a high variance, therefore requires many samples.
- Use variance reduction by using a baseline $b(\boldsymbol{\omega})$ to offset $J(\boldsymbol{y})$.
- Optimal baseline $b^*(\boldsymbol{\omega})$ and the changed gradient estimator are

$$b^*(\boldsymbol{\omega})_i = \frac{E_{p(\boldsymbol{y})}\left[J(\boldsymbol{y})\frac{\partial \log p(\boldsymbol{y})}{\partial \omega_i}^2\right]}{E_{p(\boldsymbol{y})}\left[\frac{\partial \log p(\boldsymbol{y})}{\partial \omega_i}^2\right]} \qquad (J(\boldsymbol{y}) - b(\boldsymbol{\omega})_i)\frac{\partial \log p(\boldsymbol{y})}{\partial \omega_i}$$

- $b(\boldsymbol{\omega})$ is obtained by training a neural network to estimate the numerator and the denominator of $b^*(\boldsymbol{\omega})$ for each element of $\boldsymbol{\omega}$.

20.10 Directed Generative Nets

- DBNs are a partially directed model.
- Spare coding models can be thought of as shallow directed generative models.
- Sigmoid belief networks
  - Very similar to DBNs, but the beginning layer units are independent, rather than being a RBM.
- Differentiable generator networks
  - Transforms samples of latent variables $\boldsymbol{z}$ to samples $\boldsymbol{x}$ or to distributions over samples using a differentiable function $g(\boldsymbol{z}; \boldsymbol{\theta})$, which is a neural network.
  - Includes VAEs (generator network + inference network) and GANs (generator network + discriminator network)

20.10.3 Variational Autoencoders

- Sampling
  - Draws a sample $\boldsymbol{z}$ from $p_{\text{model}}(\boldsymbol{z})$.
  - Run $\boldsymbol{z}$ through a differentiable generator network $g(\boldsymbol{z})$.
  - Sample $\boldsymbol{x}$ from $p_{\text{model}}(\boldsymbol{x}; g(\boldsymbol{z})) = p_{\text{model}}(\boldsymbol{x}|\boldsymbol{z})$.
- Training
  - Approximate inference network (i.e. encoder) $q(\boldsymbol{z}|\boldsymbol{x})$ is used to obtain $\boldsymbol{z}$.
  - $p_{\text{model}}(\boldsymbol{x}|\boldsymbol{z})$ works as a decoder network

20.10.3 Variational Autoencoders

- VAE are trained by maximizing the variational lower bound (ELBO).

$$\mathcal{L}(q) = \mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{x})} \log p_{\text{model}}(\boldsymbol{z}, \boldsymbol{x}) + \mathcal{H}(q(\mathbf{z} \mid \boldsymbol{x}))$$
$$= \mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{x})} \log p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{z}) - D_{\text{KL}}(q(\mathbf{z} \mid \boldsymbol{x})||p_{\text{model}}(\mathbf{z}))$$

- The first equality
  - The first term is the joint log-likelihood of the visible and hidden variables under the approximate posterior over the latent variables.
    - Just like with EM, except that we use an approximate rather than the exact posterior.
  - The second term is the entropy of the approximate posterior.
    - This term encourages $q(\boldsymbol{z}|\boldsymbol{x})$ to place high probability mass on many $\boldsymbol{z}$ that could have generated $\boldsymbol{x}$, rather than collapsing to a single point estimate of the most likely value.
- The second equality
  - The first term is the reconstruction log-likelihood found in other autoencoders.
  - The second term tries to make the approximate posterior distribution $q(\boldsymbol{z}|\boldsymbol{x})$ and the model prior $p_{\text{model}}(\boldsymbol{z})$ approach each other.

20.10.3 Variational Autoencoders

- Train a parametric encoder (inference network) $f(\boldsymbol{x}; \theta)$ that produces the parameters of $q(\boldsymbol{z}|\boldsymbol{x})$.
- Backprop through samples of $\boldsymbol{z}$ drawn from $q(\boldsymbol{z}|\boldsymbol{x}) = q(\boldsymbol{z}; f(\boldsymbol{x}; \theta))$
- Maximize $\mathcal{L}$ w.r.t θ and the parameters of $p_{\text{model}}(\boldsymbol{x}|\boldsymbol{z})$ by approximating the expectations in $\mathcal{L}$ by sampling.
- Main drawback is that samples are blurry.
  - Intrinsic effect of maximum likelihood that minimizes $D_{\text{KL}}(p_{\text{data}} \parallel p_{\text{model}})$.
  - Practice of using a Gaussian distribution for $p_{\text{model}}(\boldsymbol{x}; g(\boldsymbol{z}))$.

20.14 Evaluating Generative Models

- Evaluation metrics for generative models are often hard research problems.
- Often evaluate generative models by visually inspecting the samples.
  - A generative model trained on data may ignore some modes (for example dropping cats and only generating dogs),
  - and a human observer would not easily be able to inspect or remember enough images to detect the missing variation.
- Inception score
  - Shane Barratt, Rishi Sharma, *A Note on the Inception Score*, https://arxiv.org/abs/1801.01973

$$\text{IS}(G) = \exp\left(\mathbb{E}_{\mathbf{x} \sim p_g} \, D_{KL}(\, p(y|\mathbf{x}) \parallel p(y) \,) \right)$$

  - IS aimed to codify two desirable qualities of a generative model into a metric
    - The images generated should contain clear objects (i.e. the images are sharp rather than blurry), or $p(y|x)$ should be low entropy. In other words, the Inception Network should be highly confident there is a single object in the image.
    - The generative algorithm should output a high diversity of images from all the different classes in ImageNet, or $p(y)$ should be high entropy.