Introduction to Restricted Boltzmann Machines

Mark Johnson

Department of Computing
Macquarie University

7th October, 2012
(updated September 2015)

MACQUARIE University

Outline

MACQUARIE University

Boltzmann machines

- Boltzmann machines are *Markov Random Fields* with *pairwise interaction potentials*
- Developed by Smolensky as a probabilistic version of neural nets
- Boltzmann machines are basically MaxEnt models with *hidden nodes*
- Boltzmann machines often have a similar structure to multi-layer neural networks
- Nodes in a Boltzmann machine are (usually) binary valued
- A Boltzmann machine only allows *pairwise interactions* (cliques)
- Hinton developed *sampling-based methods* for training and using Boltzmann machines
- Restricted Boltzmann Machines (RBMs) are Boltzmann machines with a network architecture that enables efficient sampling

Applications of Boltzmann machines

- RBMs are used in computer vision for object recognition and scene denoising
- RBMs can be stacked to produce *deep RBMs*
- RBMs are *generative models*

$\Rightarrow$ don't need labelled training data

- *Generative pre-training:* a semi-supervised learning approach
  - train a (deep) RBM from large amounts of unlabelled data
  - use Backprop on a small amount of labelled data to tune the network weights

Boltzmann distributions

- Boltzmann distributions were first introduced by Gibbs (!) in *statistical mechanics* to describe the distribution of configurations of particles x as a function of their energy

$$\mathrm{P}(x) \;=\; \frac{1}{Z}\exp(\,-E(x)\,)$$

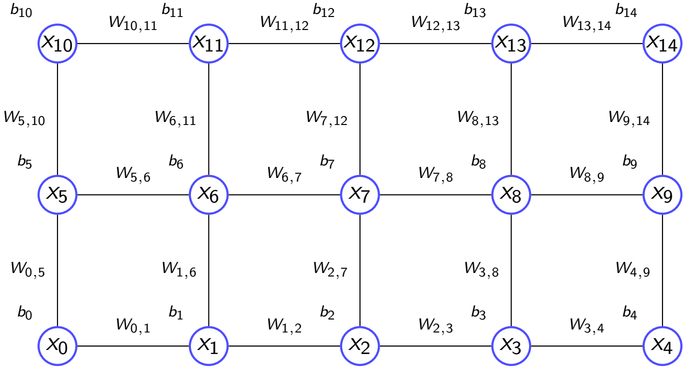
  where $E(x)$ is the *energy of configuration x*
  - every *MaxEnt model* follows a Boltzmann distribution with $E(x) \,=\, -\boldsymbol{\theta}^\top \mathbf{f}(x) = \sum_{j=1}^{m} \theta_j\, f_j(x)$
- In statistical mechanics, x is often a configuration of n particles, and $E(x)$ is approximated by a function of their *pairwise interactions*
- If the interaction graph is regular and the interactions are homogeneous, these are often called *Ising models*

Outline

MACQUARIE University

Boltzmann machines

- A Boltzmann machine is a Markov Random Field over (usually) binary variables and *only unary and binary factors*
  - ⇒ a Boltzmann machine can be represented by *a weighted undirected graph*



MACQUARIE University

Boltzmann machines

- A Boltzmann machine is a Markov Random Field with (usually) only binary nodes and only unary and binary factors
- If there are n nodes, then $\mathbf{x} \in 2^n$ is a binary vector of length n
  - $x_i = 1$ means that node i is "on", $x_i = 0$ means that node i is "off"

$$\begin{aligned} E(\mathbf{x}) &= -\mathbf{b}^\top \mathbf{x} - \mathbf{x}^\top \mathbf{W} \mathbf{x} = -\sum_{j=1}^{m} b_j x_j - \sum_{i,j} x_i W_{i,j} x_j \\ \mathrm{P}(\mathbf{x}) &= \frac{1}{Z} \exp(\,-E(\mathbf{x})\,) \\ Z &= \sum_{\mathbf{x}'} \exp(\,-E(\mathbf{x}')\,) \end{aligned}$$

  - $\mathbf{b}$ is a vector of length n of *bias weights*, and
  - $\mathbf{W}$ is an $n \times n$ matrix of *connection weights*
    - $W_{i,j}$ is the "interaction" when nodes i and j are both "on"

Visible nodes and hidden nodes

- BMs typically have *hidden nodes* as well as *visible nodes*
  - BMs are undirected $\Rightarrow$ input and output nodes are visible nodes

$$\mathbf{x} \;=\; \mathbf{v} \cdot \mathbf{h}$$

  - "$\cdot$" means concatenation, not dot product!
- Training data $D = (\mathbf{v}_1, \ldots, \mathbf{v}_n)$ only gives values for *visible nodes*
- Maximum likelihood estimation: *find parameters* $\mathbf{W}$ *and* $\mathbf{b}$ *that maximise likelihood of training data* D

$$\begin{aligned}
\widehat{\mathbf{W}}, \widehat{\mathbf{b}} &= \underset{\mathbf{W},\mathbf{b}}{\operatorname{argmax}}\, \ell_D(\mathbf{W}, \mathbf{b}) \\
\ell_D(\mathbf{W}, \mathbf{b}) &= \prod_{i=1}^{n} \mathrm{P}(\mathbf{v}_i) \\
\mathrm{P}(\mathbf{v}) &= \sum_{\mathbf{h}'} \mathrm{P}(\mathbf{v} \cdot \mathbf{h}') = \frac{\sum_{\mathbf{h}'} \exp(-E(\mathbf{v} \cdot \mathbf{h}'))}{\sum_{\mathbf{v}',\mathbf{h}'} \exp(-E(\mathbf{v}' \cdot \mathbf{h}'))} \\
E(\mathbf{x}) &= -\mathbf{b}^\top \mathbf{x} - \mathbf{x}^\top \mathbf{W} \mathbf{x}
\end{aligned}$$

MACQUARIE University

Learning BM parameters requires expectations

- Boltzmann machines are typically learnt by *minimising negative log likelihood* using *stochastic gradient descent*
  - select a *mini-batch* from training data (possibly 1 item)
  - calculate derivative $\frac{\partial - \log L}{\partial \boldsymbol{\theta}}$ of negative log likelihood $-\log L$ wrt model parameters $\boldsymbol{\theta}$ (where $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$)
  - SGD/minibatch update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \varepsilon \frac{\partial - \log L}{\partial \boldsymbol{\theta}}$
- *Derivative is a difference of expectations*

$$\begin{aligned}\frac{\partial - \log \mathrm{P}(\mathbf{v})}{\partial \theta} &= \sum_{\mathbf{h}'} \mathrm{P}(\mathbf{h}' \mid \mathbf{v}) \frac{\partial E(\mathbf{v} \cdot \mathbf{h}')}{\partial \theta} - \sum_{\mathbf{h}', \mathbf{v}'} \mathrm{P}(\mathbf{v}' \cdot \mathbf{h}') \frac{\partial E(\mathbf{v}' \cdot \mathbf{h}')}{\partial \theta} \\ &= \mathrm{E}\left[\frac{\partial E}{\partial \theta} \middle| \mathbf{v} \right] - \mathrm{E}\left[\frac{\partial E}{\partial \theta} \right]\end{aligned}$$

  where θ is a parameter such as $W_{i,j}$ or b_j
- These expectations can't be calculated analytically, so we *estimate them using sampling*

What derivatives do we need?

$$\begin{aligned}
E(\mathbf{x}) &= -\mathbf{b}^\top \mathbf{x} - \mathbf{x}^\top \mathbf{W}\mathbf{x}, \text{ so:} \\
\frac{\partial E}{\partial b_j} &= -x_j \\
\frac{\partial E}{\partial w_{i,j}} &= -x_i\, x_j, \text{ so:} \\
\frac{\partial \log \mathrm{P}(\mathbf{v})}{\partial b_j} &= \mathrm{E}[x_j \mid \mathbf{v}] - \mathrm{E}[x_j] \\
\frac{\partial \log \mathrm{P}(\mathbf{v})}{\partial w_{i,j}} &= \mathrm{E}[x_i\, x_j \mid \mathbf{v}] - \mathrm{E}[x_i\, x_j]
\end{aligned}$$

$\Rightarrow$ At MLE, *feature expectations without data equal feature expectations with data*

MACQUARIE
University

Outline

MACQUARIE
University

Why sample?

- Setup: Model has visible variables $\mathbf{v}$, whose value we observe, and hidden variables $\mathbf{h}$, whose value we don't know
  - in *Bayesian estimation*, the hidden variables include any *parameters* we want to estimate, such as $\mathbf{W}$ and $\mathbf{b}$
- Goal: compute the *expected value* of a function f
  - for estimating Boltzmann machines, $f = \frac{\partial E}{\partial \theta}$

$$\begin{aligned} \mathrm{E}[f] &= \sum_{\mathbf{h}', \mathbf{v}'} f(\mathbf{h}' \cdot \mathbf{v}') \, \mathrm{P}(\mathbf{h}' \cdot \mathbf{v}') \\ \mathrm{E}[f \mid \mathbf{v}] &= \sum_{\mathbf{h}'} f(\mathbf{h}' \cdot \mathbf{v}) \, \mathrm{P}(\mathbf{h}' \mid \mathbf{v}) \end{aligned}$$

- In the rest of this section, let y be the variables we want to sample over (e.g., $\mathbf{h}'$ or $\mathbf{h}' \cdot \mathbf{v}'$)

MACQUARIE University

Using sampling to compute expectations

- Suppose we can produce n samples $y^{(1)}, \ldots, y^{(n)}$, where $Y^{(t)} \sim \mathrm{P}(Y)$
- Then we can estimate:

$$\begin{aligned} \mathrm{E}[f] &= \sum_{y} f(y)\,\mathrm{P}(y) \\ &\approx \frac{1}{n} \sum_{t=1}^{n} f(y^{(t)}) \end{aligned}$$

- This converges under very general conditions
  - the error decreases as the square root of the number of samples n if the samples are independent

MACQUARIE University

Markov chains

- A (first-order) *Markov chain* is a distribution over random variables $S^{(0)}, \ldots, S^{(n)}$ all ranging over the same *state space* $\mathcal{S}$, where:

$$\mathrm{P}(S^{(0)}, \ldots, S^{(n)}) \;=\; \mathrm{P}(S^{(0)}) \prod_{t=0}^{n-1} \mathrm{P}(S^{(t+1)} \mid S^{(t)})$$

  $S^{(t+1)}$ is *conditionally independent* of $S^{(0)}, \ldots, S^{(t-1)}$ given $S^{(t)}$
- A Markov chain in *homogeneous* or *time-invariant* iff:

$$\mathrm{P}(S^{(t+1)} = s' \mid S^{(t)} = s) \;=\; P_{s',s} \quad \text{for all } t, s, s'$$

  The matrix P is called the *transition probability matrix* (tpm) of the Markov chain
- If $\mathrm{P}(S^{(t)} = s) = \pi_s^{(t)}$ (i.e., $\pi^{(t)}$ is a vector of state probabilities at time t) then:
  - $\pi^{(t+1)} \;=\; P\, \pi^{(t)}$
  - $\pi^{(t)} \;=\; P^t\, \pi^{(0)}$

MACQUARIE University

Ergodicity

- A Markov chain with tpm P is *ergodic* iff there is a positive integer m s.t. all elements of P^m are positive (i.e., there is an m-step path between any two states)
- Informally, an ergodic Markov chain "forgets" its past states
- Theorem: For each homogeneous ergodic Markov chain with tpm P there is a *unique limiting distribution* D_P, i.e., as n approaches infinity, the distribution of S_n converges on D_P
- D_P is called the *stationary distribution* of the Markov chain
- Let π be the vector representation of D_P, i.e., $D_P(y) = \pi_y$. Then:

$$\begin{aligned} \pi &= P\,\pi, \qquad \text{and} \\ \pi &= \lim_{n\to\infty} P^n \pi^{(0)} \qquad \text{for every initial distribution } \pi^{(0)} \end{aligned}$$

Using a Markov chain to sample from $\mathrm{P}(Y)$

- Set the state space $\mathcal{S}$ of the Markov chain to the range of $\mathbf{Y}$ ($\mathcal{S}$ may be *astronomically large*)
- Find a tpm P such that $\mathrm{P}(\mathbf{Y}) \sim D_P$
- "Run" the Markov chain, i.e.,
  - Pick $\mathbf{y}^{(0)}$ somehow
  - For $t = 0, \ldots, n-1$:
    - sample $\mathbf{y}^{(t+1)}$ from $\mathrm{P}(\mathbf{Y}^{(t+1)} \mid \mathbf{Y}^{(t)} = \mathbf{y}^{(t)})$, i.e., from $P_{\cdot,\mathbf{y}^{(t)}}$
  - After discarding the first *burn-in* samples, use remaining samples to calculate statistics
- *WARNING:* in general the samples $\mathbf{y}^{(t)}$ are *not independent*

MACQUARIE University

The Gibbs sampler

- The Gibbs sampler is useful when:
  - $\mathbf{Y}$ is multivariate, i.e., $\mathbf{Y} = (Y_1, \ldots, Y_m)$, and
  - easy to sample from $\mathrm{P}(Y_j \mid \mathbf{Y}_{-j})$ (where $\mathbf{Y}_{-j}$ is $\mathbf{Y}$ *except* Y_j)
- The *Gibbs sampler* for $\mathrm{P}(Y)$ is the tpm $P = \prod_{j=1}^{m} P^{(j)}$, where:

$$P^{(j)}_{\mathbf{y}',\mathbf{y}} = \begin{cases} 0 & \text{if } \mathbf{y}'_{-j} \neq \mathbf{y}_{-j} \\ \mathrm{P}(Y_j = y'_j \mid \mathbf{Y}_{-j} = \mathbf{y}_{-j}) & \text{if } \mathbf{y}'_{-j} = \mathbf{y}_{-j} \end{cases}$$

- Informally, the Gibbs sampler cycles through each of the variables Y_j, replacing the current value y_j with a sample from $\mathrm{P}(Y_j \mid \mathbf{Y}_{-j} = \mathbf{y}_{-j})$
- There are *sequential scan* and *random scan* variants of Gibbs sampling

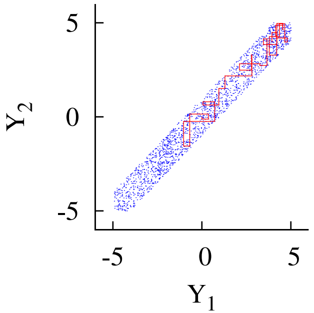
MACQUARIE University

A simple example of Gibbs sampling

$$\mathrm{P}(Y_1, Y_2) = \begin{cases} c & \text{if } |Y_1| < 5, |Y_2| < 5 \text{ and } |Y_1 - Y_2| < 1 \\ 0 & \text{otherwise} \end{cases}$$

- The Gibbs sampler for $\mathrm{P}(Y_1, Y_2)$ samples repeatedly from:

$$\begin{aligned} \mathrm{P}(Y_2 \mid Y_1) &= \text{Uniform}(\max(-5, Y_1 - 1), \min(5, Y_1 + 1)) \\ \mathrm{P}(Y_1 \mid Y_2) &= \text{Uniform}(\max(-5, Y_2 - 1), \min(5, Y_2 + 1)) \end{aligned}$$



Sample run

Y_1	Y_2
0	0
0	-0.119
0.363	-0.119
0.363	0.146
-0.681	0.146
-0.681	-1.551

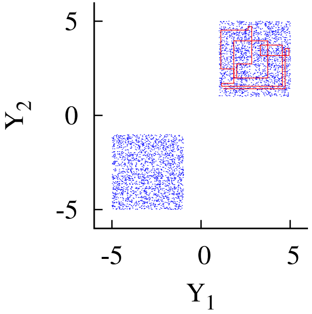
A non-ergodic Gibbs sampler

$$\mathrm{P}(Y_1, Y_2) = \begin{cases} c & \text{if } 1 < Y_1, Y_2 < 5 \text{ or } -5 < Y_1, Y_2 < -1 \\ 0 & \text{otherwise} \end{cases}$$

- The Gibbs sampler for $\mathrm{P}(Y_1, Y_2)$, initialized at (2,2), samples repeatedly from:

$$\begin{aligned} \mathrm{P}(Y_2 \mid Y_1) &= \text{Uniform}(1,5) \\ \mathrm{P}(Y_1 \mid Y_2) &= \text{Uniform}(1,5) \end{aligned}$$

I.e., *never visits the negative values of Y_1, Y_2*



Sample run

Y_1	Y_2
2	2
2	2.72
2.84	2.72
2.84	4.71
2.63	4.71
2.63	4.52
1.11	4.52

Why does the Gibbs sampler work?

- The Gibbs sampler tpm is $P = \prod_{j=1}^{m} P^{(j)}$, where $P^{(j)}$ replaces y_j with a sample from $\mathrm{P}(Y_j \mid \mathbf{Y}_{-j} = \mathbf{y}_{-j})$ to produce y'
- But if $\mathbf{y}$ is a sample from $\mathrm{P}(\mathbf{Y})$, then so is $\mathbf{y}'$, since $\mathbf{y}'$ differs from $\mathbf{y}$ only by replacing y_j with a sample from $\mathrm{P}(Y_j \mid \mathbf{Y}_{-j} = \mathbf{y}_{-j})$
- Since $P^{(j)}$ maps samples from $\mathrm{P}(\mathbf{Y})$ to samples from $\mathrm{P}(\mathbf{Y})$, so does P

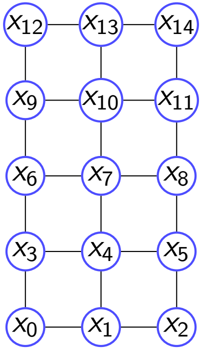
$\Rightarrow$ $\mathrm{P}(\mathbf{Y})$ is a stationary distribution for P

- If P is ergodic, then $\mathrm{P}(\mathbf{Y})$ is the unique stationary distribution for P, i.e., the sampler converges to $\mathrm{P}(\mathbf{Y})$

MACQUARIE
University

Gibbs sampling with Boltzmann machines

- Recall: need samples of hidden (and visible) node values
- Gibbs sampler: update x_j with sample from $\mathrm{P}(X_j \mid \mathbf{X}_{-j}) \propto \mathrm{P}(X_j, \mathbf{X}_{-j})$
- Only need to evaluate terms in $\mathrm{P}(X_j, \mathbf{X}_{-j})$ that involve X_j
  - these are the *neighbours* of X_j in the MRF graph



The wake-sleep algorithm for Boltzmann machines

- Boltzmann machines are typically learnt by *minimising negative log likelihood* using *stochastic gradient descent*
- Use sampling to compute gradient of log likelihood

$$\begin{aligned}\frac{\partial \log \mathrm{P}(\mathbf{v})}{\partial b_j} &= \mathrm{E}[x_j \mid \mathbf{v}] - \mathrm{E}[x_j] \\ \frac{\partial \log \mathrm{P}(\mathbf{v})}{\partial w_{i,j}} &= \mathrm{E}[x_i\, x_j \mid \mathbf{v}] - \mathrm{E}[x_i\, x_j]\end{aligned}$$

- The *wake-sleep algorithm* calculates these using two samplers
  - *wake step:* generate samples $\mathbf{h}'$ from $\mathrm{P}(\mathbf{h}' \mid \mathbf{v})$, i.e., "clamp" $\mathbf{v}$ to visible data
  - *sleep step:* generate samples $\mathbf{v}' \cdot \mathbf{h}'$ from $\mathrm{P}(\mathbf{v}' \cdot \mathbf{h}')$, i.e., let the network "dream"
- At MLE, network's data-driven expectations $=$ network's "dreams"

MACQUARIE University

Advantages and disadvantages of the wake-sleep algorithm

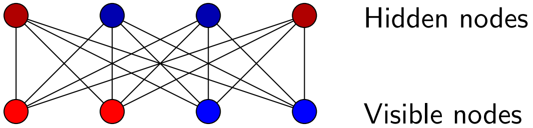
- $+$ The wake-sleep algorithm doesn't require us to calculate the partition function Z
- $+$ It's easy to sample using Gibbs sampling (sample each node conditional on its neighbours)
  - only requires normalising over the possible values of each node
- $-$ It may take *many samples* to accurately estimate these expectations, to perform just one SGD update!
- $\Rightarrow$ *Restricted Boltzmann Machines:* constrain network structure so wake steps (on visible data) don't require sampling
- $\Rightarrow$ *Contrastive Divergence:* reinitialise sampler after each update so updates can be computed from just a few samples
- $\Rightarrow$ *Persistent Contrastive Divergence:* don't reinitialise sampler after each update

MACQUARIE University

Outline

Restricted Boltzmann Machines

- A *Restricted Boltzmann Machine* is a Boltzmann Machine where all connections are between hidden and visible units
  - ⇒ no hidden-to-hidden or visible-to-visible connections
  - ⇒ an RBM is a Markov Random Field over a *bipartite graph*

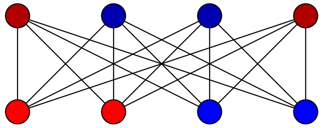


$$E(\mathbf{v},\mathbf{h}) \;=\; -\mathbf{b}^\top\mathbf{v} - \mathbf{c}^\top\mathbf{h} - \mathbf{h}^\top\mathbf{W}\mathbf{v}$$

MACQUARIE University

Wake-sleep in Restricted Boltzmann Machines

- A *Restricted Boltzmann Machine* is a Boltzmann Machine where all connections are between hidden and visible units



- During the wake step, the values of all visible nodes are fixed:
  - ⇒ the hidden nodes are *independent* given the visible nodes
  - ⇒ wake-step expectations can be calculated exactly (without sampling)
- Sleep step still requires sampling, but it is more structured
  - ▸ Blocked Gibbs sampler for sleep step: repeat
    - – sample hidden nodes conditional on visible nodes
    - – sample visible nodes conditional on hidden nodes
  - ▸ each block sampling step can be done in parallel because
    - – hidden nodes are independent conditional on visible nodes, and
    - – visible nodes are independent conditional on hidden nodes

Outline

MACQUARIE University

Contrastive Divergence

- Blocked Gibbs sampler for RBM sleep step: repeat
  - sample hidden nodes conditional on visible nodes
  - sample visible nodes conditional on hidden nodes
- How do we initialise the sampler?
- *Contrastive divergence:*
  - initialise "dream" *with visible data*
  - *don't run many block sampler iterations* (maybe just 1?)
- Intuition: in a "good" model, sampler should stay "close" to visible data

$\Rightarrow$ Update model parameters to "stop dreams moving away from reality"

  - if dreams are same as visible data $\Rightarrow$ identical wake and sleep expectations $\Rightarrow$ no weight change
  - a 1-step contrastive divergence sampler is a bit like a neural net auto-encoder!

Persistent Contrastive Divergence

- Contrastive Divergence algorithm reinitialises "sleep" sampler to data item at each SGD step
- Persistent Contrastive Divergence algorithm initialises "sleep" sampler with "sleep" samples from last SGD step
  - intuition: a single SGD iteration won't have changed $\mathbf{W}$ and $\mathbf{b}$ much, so "sleep" samples probably still have high probability
- Typically maintain multiple "sleep particles" to improve expectations
  - usually number of sleep particles = mini-batch size
- This is regular Gibbs sampling for sleep state, except that the model parameters $\mathbf{W}$ and $\mathbf{b}$ are changing

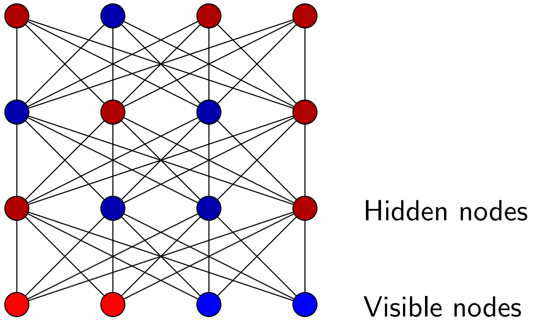
MACQUARIE University

Outline

MACQUARIE University

Stacking Restricted Boltzmann Machines

- RBMs can be “stacked” to form more complex machines
- Stacked RBMs are trained in layers
  - Hidden layer i serves as the visible data for training hidden layer $i+1$



Using deep RBMs for semi-supervised learning

- Semi-supervised learning:
  - large amount of unlabelled data (e.g., images from web)
  - much smaller amount of labelled data
- *Unsupervised pre-training* for semi-supervised learning:
  - train a deep RBM from unlabelled data
  - add a final output node connected to top hidden layer (and input as well?)
  - use Backprop on labelled data to fine-tune connection weights

Outline

MACQUARIE University

Representing words in RBMs

- Visible layer uses "one-hot" representation $\mathbf{e}_k$ of word k
  - organise visible nodes into groups binary nodes for each word
  - require that *exactly one node* in each word group is "on"
- Equivalent to MaxEnt *multinomial model* of words given hidden variables

$$\mathrm{P}(\mathbf{v}^{(i)}{=}\mathbf{e}_k \mid \mathbf{h}) \;=\; \frac{\exp(\mathbf{b}^{(i)\top}\mathbf{e}_k + \mathbf{h}^\top \mathbf{W}^{(i)}\mathbf{e}_k)}{\sum_{k'} \exp(\mathbf{b}^{(i)\top}\mathbf{e}_{k'} + \mathbf{h}^\top \mathbf{W}^{(i)}\mathbf{e}_{k'})}$$

+ This model can give good performance

– But it requires us to *sum over the entire vocabulary* to compute a partition function

– This partition function is required to Gibbs sample each visible node at each iteration of the sleep step in the training algorithm

MACQUARIE University

Metropolis-Hastings word sampler

- A Metropolis-Hastings (MH) sampler requires a *proposal distribution* that can be efficiently sampled
- The MH algorithm: repeat
    - sample a proposal from the proposal distribution
    - accept the proposal with an *acceptance probability* that depends on the ratio of the probability of proposal and probability of previous value under target distribution
- Because the acceptance probability depends a *ratio* of target probabilities, the partition functions cancel
- The efficiency of the MH algorithm depends on how close the proposal distribution is to the target distribution
    - apparently a *unigram proposal distribution* is good enough

Outline

MACQUARIE University

Conclusion

- Boltzmann Machines are Markov Random Fields with binary potentials
- The *wake-sleep algorithm* can compute SGD gradients
- Restricted Boltzmann Machines only allow connections between visible and hidden nodes
  - $\Rightarrow$ expectations required in wake step can be efficiently computed
- *Contrastive Divergence* initialises sleep step with visible data, and only runs a few iterations
- RBMs can be *stacked* just like neural nets, where hidden units of level i are used as visible units of level $i + 1$
- Metropolis-Hastings with a unigram proposal can be used to avoid calculating partition function over words in sleep step of RBM language models

MACQUARIE University