Autoencoder

Lecture 04

Outline

Autoencoder

An autoencoder takes an input $\mathbf{x} \in [0, 1]^d$ and first maps it (with an encoder) to a hidden representation $\mathbf{y} \in [0, 1]^{d'}$ through a deterministic mapping

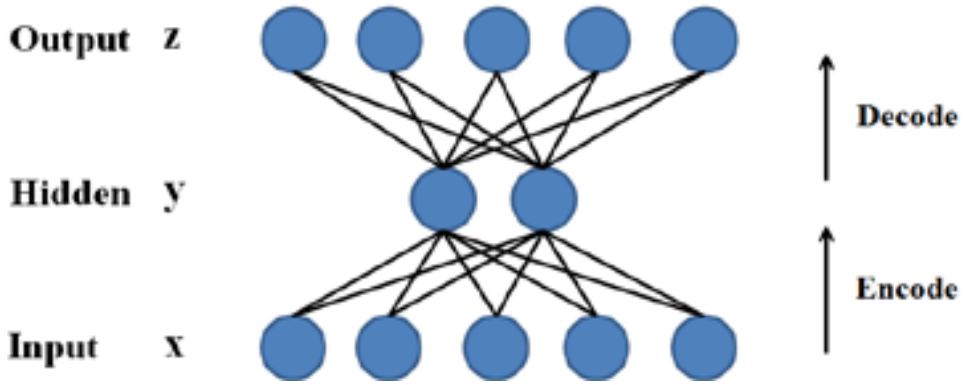
$$\mathbf{y} = s(\mathbf{W}\mathbf{x} + \mathbf{b})$$

where s is a non-linear activation function (such as sigmoid).

$\mathbf{y}$ is mapped back (with a decoder) into a reconstruction $\mathbf{z}$ of the same shape as $\mathbf{x}$,

$$\mathbf{z} = s(\mathbf{W}'y + \mathbf{b}')$$

$\mathbf{z}$ is seen as a prediction of $\mathbf{x}$.



Autoencoder

Encoder

$$\mathbf{y} = f_\theta(\mathbf{x})$$

Decoder

$$\mathbf{z} = g_\theta(\mathbf{y})$$

$$\theta = \{\mathbf{W}, \mathbf{W}^I, \mathbf{b}, \mathbf{b}^I\}$$

It is important to add regularization in the training criterion or the parametrization to prevent the auto-encoder from learning the identity function, which would lead zero reconstruction error everywhere

A particular form of regularization consists in constraining the code to have a low dimension, and this is what the classical auto-encoder or PCA do.

Autoencoder

Optionally, the weight matrix $\mathbf{W}^I$ of the reverse mapping may be constrained to be the transpose of the forward mapping: $\mathbf{W}^I = \mathbf{W}^T$, referred to as **tied weights**

The objective function measures the reconstruction error

Squared error: $J(\mathbf{x}, \mathbf{z}) = ||\mathbf{x} - \mathbf{z}||^2$

Cross-entropy: $J(x, z) = -\sum_{i=1}^{d}[x_i log z_i + (1 - x_i)\log(1 - z_i)]$

y is expected a distributed representation that captures the main factors of variation in data.

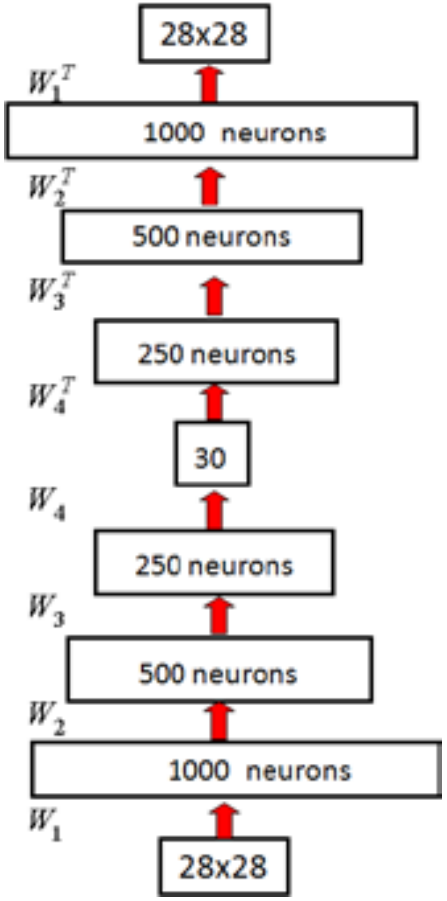
If there is one linear hidden layer and the mean squired error criterion is used to train the network, the *k* hidden unites learn to project the input in the span of the first *k* principal components of data.

Autoencoder gives low reconstruction error on test examples from the same distribution as the training examples, but generally high reconstruction error on samples randomly chosen from the input space

Autoencoder is a multi-layer neural network. The only difference is that the size of its output layer is the same as the input layer and the objective function.

Deep autoencoder

Stack multiple encoders (and their corresponding decoders)



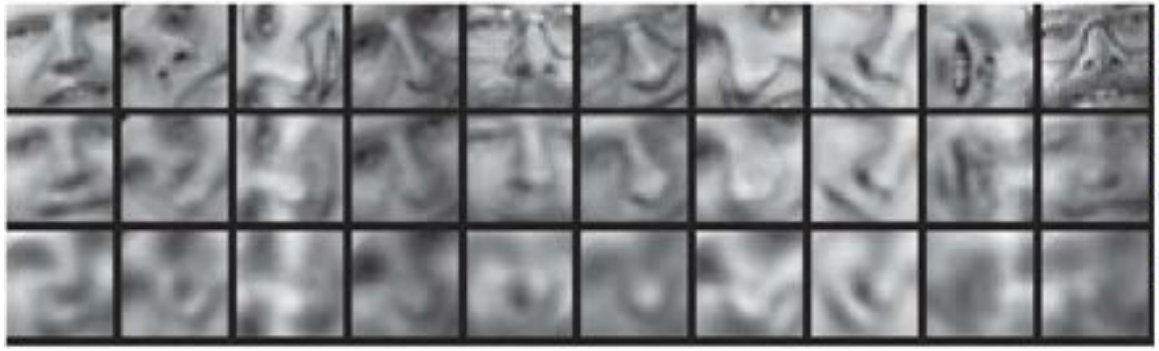
Deep autoencoder

Very difficult to optimize deep autoencoders using backpropagation

Pre-training + fine-tuning

- First train a stack of RBMs
- Then “unroll” them
- Then fine-tune with backpropagation

Comparison of methods of compressing images



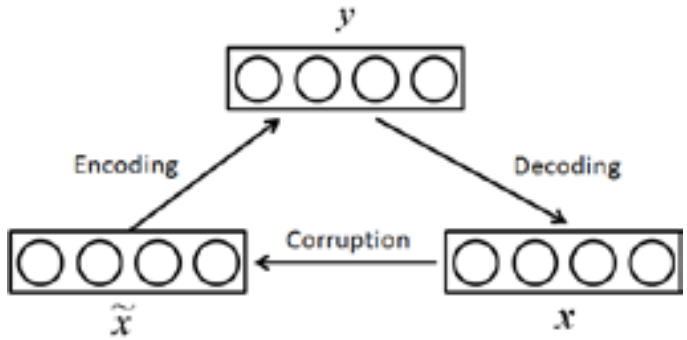
Denoising autoencoder

In order to force the hidden layer to discover more robust features and prevent it from simply learning the identity function, train the autoencoder to reconstruct the input from a corrupted version of it

- Encode the input (preserve the information about the input)
- Undo the effect of a corruption process stochastically applied to the input of the auto-encoder

To convert the autoencoder to a denoising autoencoder, all we need to do is to add a stochastic corruption step operating on the input

- Randomly sets some of the inputs (as many as half of them) to zero.
- Hence the denoising auto-encoder is trying to predict the corrupted (i.e. missing) values from the uncorrupted (i.e., non-missing) values, for randomly selected subsets of missing patterns.
- The input can be corrupted in other ways

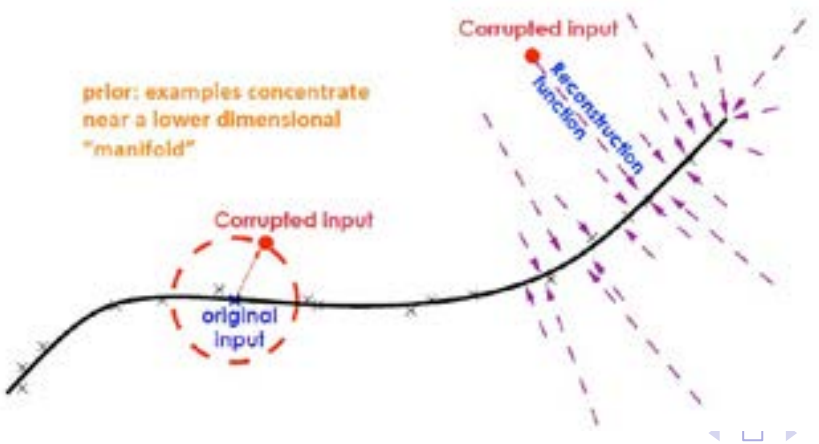


Denoising autoencoder

The learner must capture the structure of the input distribution in order to optimally undo the effect of the corruption process, with the reconstruction essentially being a nearby but higher density point than the corrupted input

The denoising autoencoder is learning a reconstruction function that corresponds to a vector field pointing towards high-density regions (the manifold where examples concentrate)

Denosing autoencoder basically learns in $r(\tilde{\mathbf{x}}) - \tilde{\mathbf{x}}$ a vector pointing in the direction $\frac{\partial \log P(\tilde{\mathbf{x}})}{\partial \tilde{\mathbf{x}}}$



Predictive Sparse Decomposition

Sparse coding

Solving the encoder f_θ is non-trivial because of L_1 minimization and entails an iterative optimization

$$\mathbf{y}^* = f_\theta(\mathbf{x}) = \arg\min_{\mathbf{y}} ||\mathbf{x} - \mathbf{W}\mathbf{y}||_2^2 + \lambda||\mathbf{y}||_1$$

$$J_{SC} = \sum_n ||\mathbf{x}^{(n)} - \mathbf{W}\mathbf{y}^{*(n)}||_2^2$$

Predictive sparse decomposition

Approximation to sparse coding
Add sparse penalty to auto-encoder
Replace the costly and highly non-linear encoding step by a fast non-iterative approximation
The training criterion is simultaneously optimized with respect to the hidden codes (representation) $\mathbf{y}^{(n)}$ and with respect to the parameters θ

$$J_{PSD} = \sum_n \lambda||\mathbf{y}^{(n)}||_1 + ||\mathbf{x}^{(n)} - \mathbf{W}\mathbf{y}^{(n)}||_2^2 + ||\mathbf{y}^{(n)} - f_\theta(\mathbf{x}^{(n)})||_2^2$$

$$f_\theta(\mathbf{x}^{(n)}) = \sigma(\mathbf{b} + \mathbf{W}^T\mathbf{x}^{(n)})$$

Multimodal deep learning

Ngiam et al. ICML'11 (audio-visual speech recognition)

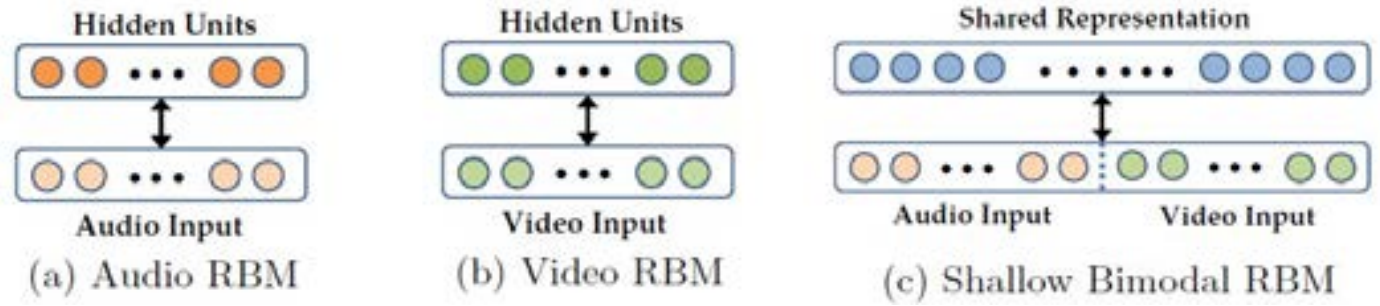
Multimodal fusion: data from all modalities is available at all phases

Cross modality learning: data from multiple modalities is available only during feature learning; during supervised training and testing, only data from a single modality is provided. The aim is to learn better single modality representations given unlabeled data from multiple modalities.

Matching across different modalities

A direct approach is to train a RBM over the concatenated audio and video data. Limited as a shallow model, it is hard for a RBM to learn the highly nonlinear correlations and form multimodal representations

It was found that learning a shallow bimodal RBM results in hidden units that have strong connections to variables from individual modality but few units that connect across the modalities.



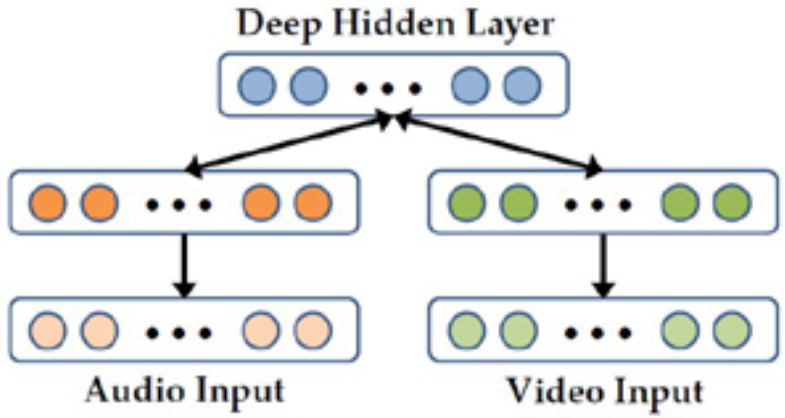
(a) Audio RBM (b) Video RBM (c) Shallow Bimodal RBM

Multimodal deep learning

Bimodal DBN: greedily layerwise training a RBM over the pre-trained layers for each modality; by representing the data through learned multilayer representations, it can be easier for the model to learn higher-order correlations across modalities. In (d), the first layer representations correspond to phonemes and visemes and the second layer models the relationships between them.

Problems

- It is possible for the model to find representations such that some hidden units are tuned only for audio while others are tuned only for video
- It is not applicable in a cross modality learning setting where only one modality is present during supervised training and testing.

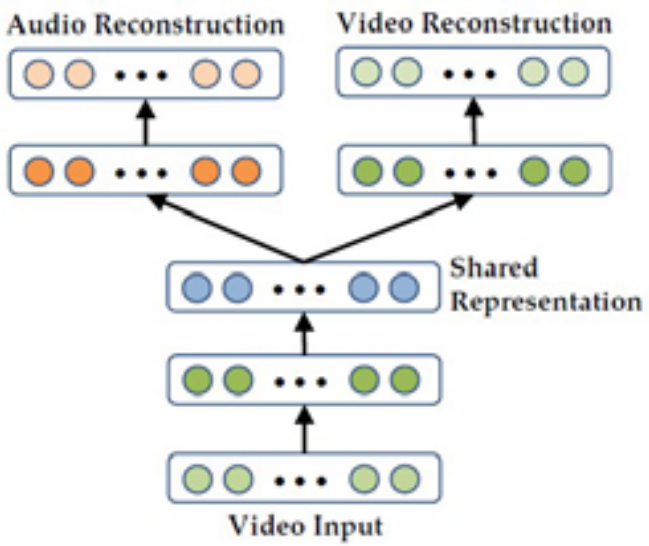


(d) Bimodal DBN

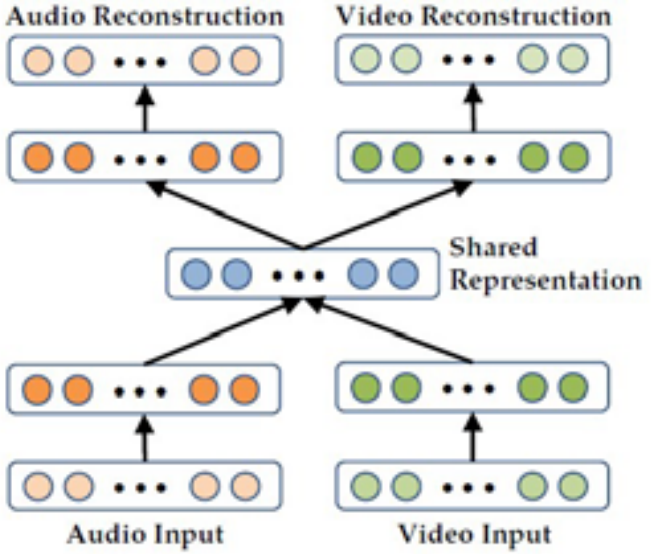
Multimodal deep auto-encoder

In (a), the deep auto-encoder is trained to reconstruct both modalities when given only video data and thus discovers correlations across the modalities. Initialize the deep autoencoder with the bimodal DBN weights and discard any weights that are no longer present. The middle layer can be used as the new feature representation.

Model (a) is used when only a single modality is present at supervised training and testing



(a) Video-Only Deep Autoencoder

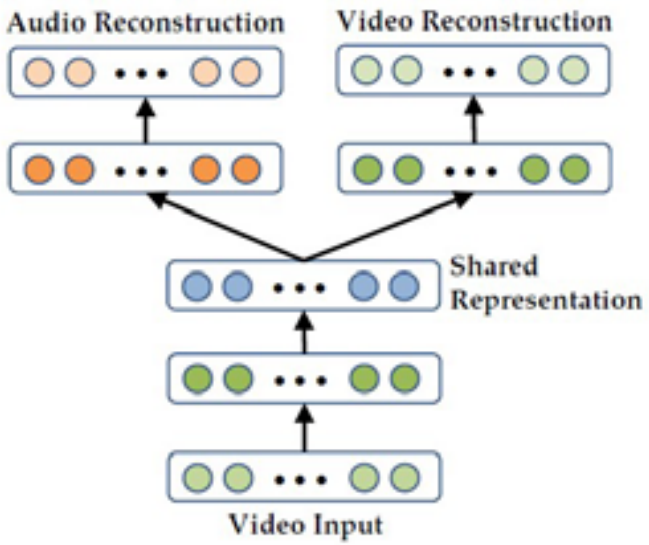


(b) Bimodal Deep Autoencoder

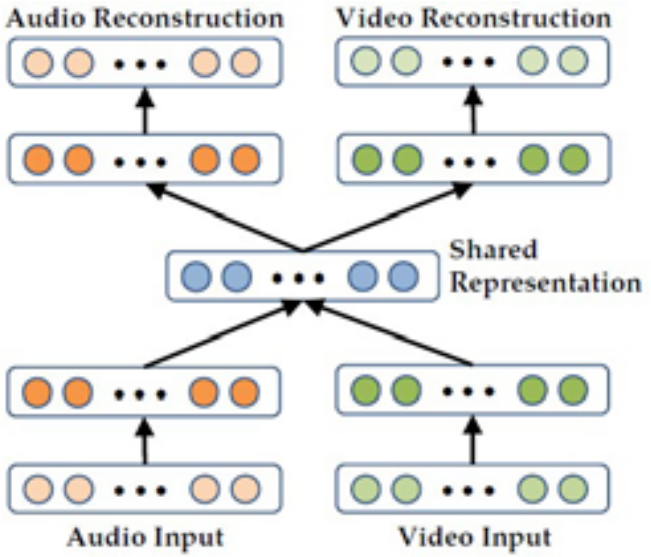
Multimodal deep auto-encoder

Inspired by denoising auto-encoder, train model (b) with an augmented dataset

- One-third of the training data has only video for input (setting zero values for the audio data)
- Another one-third of the data has only audio
- The last one-third of the data has both audio and video



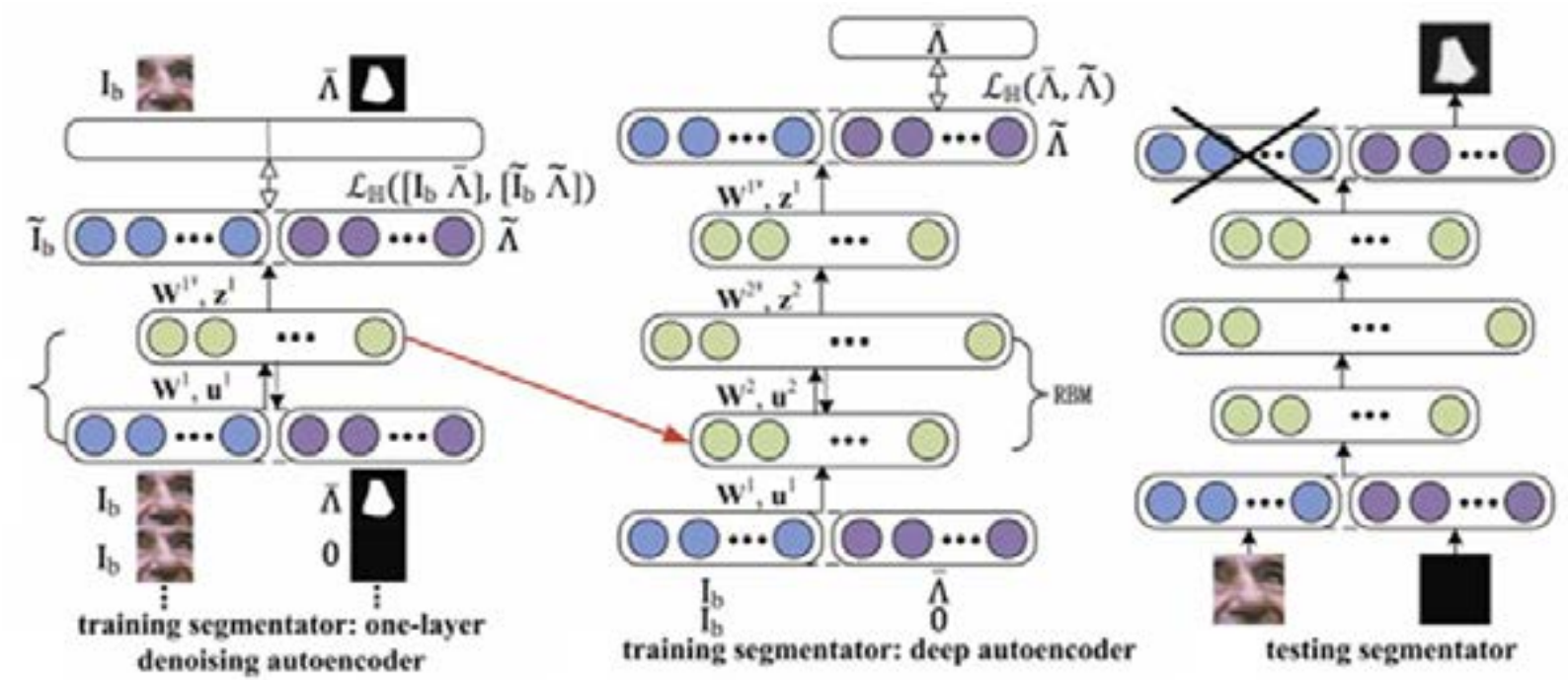
(a) Video-Only Deep Autoencoder



(b) Bimodal Deep Autoencoder

Multimodal deep auto-encoder

For image segmentation: Luo et al. CVPR'12



Energy-Based Models (EBM)

Energy-based models associate a scalar energy to each configuration of the variables of interest. Learning corresponds to modifying that energy function so that its shape has desirable properties. For example, we would like plausible or desirable configurations to have low energy. Energy-based probabilistic models define a probability distribution through an energy function, as follows:

$$p(x) = \frac{e^{-E(x)}}{Z}. \tag{1}$$

The normalizing factor Z is called the **partition function** by analogy with physical systems.

$$Z = \sum_x e^{-E(x)}$$

An energy-based model can be learnt by performing (stochastic) gradient descent on the empirical negative log-likelihood of the training data. As for the logistic regression we will first define the log-likelihood and then the loss function as being the negative log-likelihood.

$$\mathcal{L}(\theta, \mathcal{D}) = \frac{1}{N} \sum_{x^{(i)} \in \mathcal{D}} \log\ p(x^{(i)})$$

$$\ell(\theta, \mathcal{D}) = -\mathcal{L}(\theta, \mathcal{D})$$

using the stochastic gradient $-\frac{\partial \log p(x^{(i)})}{\partial \theta}$, where θ are the parameters of the model.

EBMs with Hidden Units

In many cases of interest, we do not observe the example x fully, or we want to introduce some non-observed variables to increase the expressive power of the model. So we consider an observed part (still denoted x here) and a **hidden** part h. We can then write:

$$P(x) = \sum_h P(x, h) = \sum_h \frac{e^{-E(x,h)}}{Z}. \tag{2}$$

In such cases, to map this formulation to one similar to Eq. (1), we introduce the notation (inspired from physics) of **free energy**, defined as follows:

$$\mathcal{F}(x) = -\log \sum_h e^{-E(x,h)} \tag{3}$$

which allows us to write,

$$P(x) = \frac{e^{-\mathcal{F}(x)}}{Z} \text{ with } Z = \sum_x e^{-\mathcal{F}(x)}.$$

The data negative log-likelihood gradient then has a particularly interesting form.

$$-\frac{\partial \log p(x)}{\partial \theta} = \frac{\partial \mathcal{F}(x)}{\partial \theta} - \sum_{\tilde{x}} p(\tilde{x}) \frac{\partial \mathcal{F}(\tilde{x})}{\partial \theta}. \tag{4}$$

Notice that the above gradient contains two terms, which are referred to as the **positive** and **negative phase**. The terms positive and negative do not refer to the sign of each term in the equation, but rather reflect their effect on the probability density defined by the model. The first term increases the probability of training data (by reducing the corresponding free energy), while the second term decreases the probability of samples generated by the model.

It is usually difficult to determine this gradient analytically, as it involves the computation of $E_P[\frac{\partial \mathcal{F}(x)}{\partial \theta}]$. This is nothing less than an expectation over all possible configurations of the input x (under the distribution P formed by the model) !

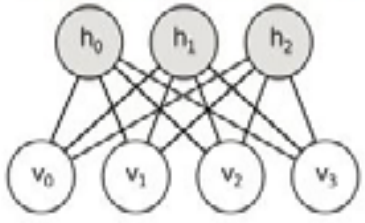
The first step in making this computation tractable is to estimate the expectation using a fixed number of model samples. Samples used to estimate the negative phase gradient are referred to as **negative particles**, which are denoted as $\mathcal{N}$. The gradient can then be written as:

$$-\frac{\partial \log p(x)}{\partial \theta} \approx \frac{\partial \mathcal{F}(x)}{\partial \theta} - \frac{1}{|\mathcal{N}|} \sum_{\tilde{x} \in \mathcal{N}} \frac{\partial \mathcal{F}(\tilde{x})}{\partial \theta}. \tag{5}$$

where we would ideally like elements $\tilde{x}$ of $\mathcal{N}$ to be sampled according to P (i.e. we are doing Monte-Carlo). With the above formula, we almost have a pratical, stochastic algorithm for learning an EBM. The only missing ingredient is how to extract these negative particles $\mathcal{N}$. While the statistical literature abounds with sampling methods, Markov Chain Monte Carlo methods are especially well suited for models such as the Restricted Boltzmann Machines (RBM), a specific type of EBM.

Restricted Boltzmann Machines (RBM)

Boltzmann Machines (BMs) are a particular form of log-linear Markov Random Field (MRF), i.e., for which the energy function is linear in its free parameters. To make them powerful enough to represent complicated distributions (i.e., go from the limited parametric setting to a non-parametric one), we consider that some of the variables are never observed (they are called hidden). By having more hidden variables (also called hidden units), we can increase the modeling capacity of the Boltzmann Machine (BM). Restricted Boltzmann Machines further restrict BMs to those without visible-visible and hidden-hidden connections. A graphical depiction of an RBM is shown below.



The energy function $E(v, h)$ of an RBM is defined as:

$$E(v, h) = -b'v - c'h - h'Wv \tag{6}$$

where W represents the weights connecting hidden and visible units and b, c are the offsets of the visible and hidden layers respectively.

This translates directly to the following free energy formula:

$$\mathcal{F}(v) = -b'v - \sum_i \log \sum_{h_i} e^{h_i(c_i + W_i v)}.$$

Because of the specific structure of RBMs, visible and hidden units are conditionally independent given one-another. Using this property, we can write:

$$p(h|v) = \prod_i p(h_i|v)$$

$$p(v|h) = \prod_j p(v_j|h).$$

RBMs with binary units

In the commonly studied case of using binary units (where v_j and $h_i \in \{0,1\}$), we obtain from Eq. (6) and (2), a probabilistic version of the usual neuron activation function:

$$P(h_i = 1|v) = sigm(c_i + W_i v) \tag{7}$$

$$P(v_j = 1|h) = sigm(b_j + W'_j h) \tag{8}$$

The free energy of an RBM with binary units further simplifies to:

$$\mathcal{F}(v) = -b'v - \sum_i \log(1 + e^{(c_i + W_i v)}). \tag{9}$$

Update Equations with Binary Units

Combining Eqs. (5) with (9), we obtain the following log-likelihood gradients for an RBM with binary units:

$$-\frac{\partial \log p(v)}{\partial W_{ij}} = E_v[p(h_i|v) \cdot v_j] - v_j^{(i)} \cdot sigm(W_i \cdot v^{(i)} + c_i) \tag{10}$$

$$-\frac{\partial \log p(v)}{\partial c_i} = E_v[p(h_i|v)] - sigm(W_i \cdot v^{(i)})$$

$$-\frac{\partial \log p(v)}{\partial b_j} = E_v[p(v_j|h)] - v_j^{(i)}$$