Jin Gu
Department of Automation, Tsinghua University
Email: jgu@tsinghua.edu.cn
Phone: (010) 62794294-866

Chapter 11 Learning with Incomplete Data

2021 Fall

Jin Gu (古槿)

Outlines

- Some Examples
  - Variables are *Not Detectable*
    - Hidden Markov Revisited
    - Mixture Models
    - Latent Linear Models
  - *Missing Values* and *Data Outliers*
- General Principles and Methods
  - General Principles
  - Expectation Maximization (EM)
  - MCMC Sampling

Chapter 11 Learning with Incomplete Data

Textbook1

- **Chapter 19.2.1** Gradient Ascent
- **Chapter 19.2.2.1-19.2.2.4, 19.2.2.6** Expectation Maximization (EM)
- **Chapter 20.1-20.3** Maximum Likelihood Estimation in Markov Networks

Textbook2

- **Chapter 11** Mixture models and the EM algorithm
- **Chapter 12** Latent Linear Models
- **Chapter 27*** Latent variable models for discrete data

* Advanced Readings

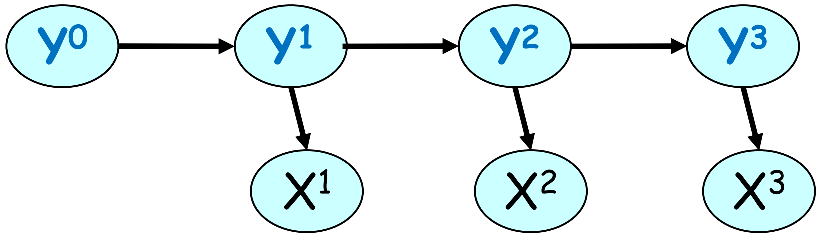
The Reasons of Incomplete Data

- Variables are not detectable (hidden variables)
  - Variables cannot be observed
  - Variables only in concept

- Missing values and data outliers
  - The systems miss some observations
  - A few exceptional data points

Hidden Markov Models Revisited

- The state variables are not observable



$$\theta = \left\{ \begin{array}{l} T = \{ t_{i,j}, \quad i, j = 1 \ldots N \}, \\ E = \{ e_{i,j}, \quad i = 1 \ldots N, j = 1 \ldots K \}, \\ \pi = \{ \pi_i, \quad i = 1 \ldots N \} \end{array} \right\}$$

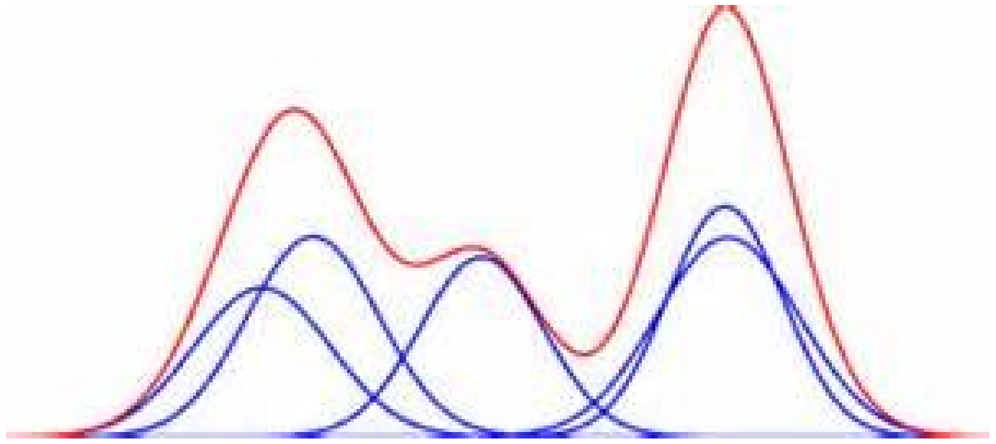
$$X = \{ x_t, \quad t = 1, \ldots, T \}$$

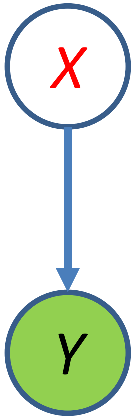
Mixture Models

- The data are randomly drawn from two or more different distributions (or classes), but we do not know the origin of each data point
- Gaussian Mixture Models (GMMs)

$$P(X = k) = \alpha_k, \;\; \sum_k \alpha_k = 1, \alpha_k > 0$$

$$p(Y|X = k) = N\left(\mu_k, \sigma_k^2\right)$$





Latent Linear Models

- Traditional linear regression (all observed)

  $\boldsymbol{Y} = \boldsymbol{b} + \boldsymbol{B}X$

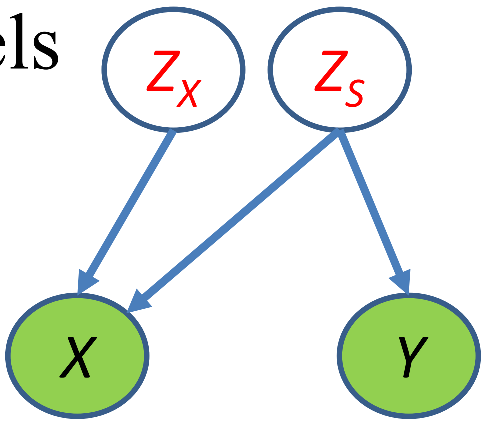
- The observation variables $\boldsymbol{X}$ can be explained by linear combinations of a set of shared latent variables $\boldsymbol{Z}$

  $X_i = \beta_{i,0} + \sum_{j=1}^{k} \beta_{i,j} z_j + \varepsilon \Leftrightarrow \boldsymbol{X} = \boldsymbol{b} + \boldsymbol{BZ}$

- Extended to partial least square models

  $\boldsymbol{Y} = \boldsymbol{b_Y} + \boldsymbol{W_Y Z_S}$

  $\boldsymbol{X} = \boldsymbol{b_X} + \boldsymbol{W_X Z_S} + \boldsymbol{B_X Z_X}$



Learning in Gaussian Mixture Models

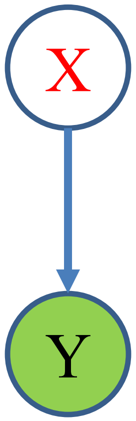
- Under the framework of MLE：

$$\arg\max_{\theta} p(\mathcal{D}|\theta) \qquad \mathcal{D} = \{y[1], \ldots, y[M]\}$$

- For complete data, the learning is simple (MLE)
  - Complete data: $\mathcal{D}_C = \{(x[i], y[i])\}_{i=1\ldots M}$

$$\pi_k^* = \frac{M[x=k]}{M}$$

  - For *k*-th mixture Gaussian component
    - $\mu_k^* = \frac{1}{M[x=k]} \sum_m y[m] \,|_{x[m]=k}$
    - $\Sigma_k^* = \frac{1}{M[x=k]} \sum_m (y[m] - \mu_k^*)(y[m] - \mu_k^*)^T \,|_{x[m]=k}$



Learning in Gaussian Mixture Models

- *Key problem: component labels are* ***UNKNOWN****!!*
- For partially observed data, we can calculate the posterior (inference) if the parameters are given

$$Q(x=k) = P(x=k|y,\theta) = \frac{p(y|x=k,\theta)P(x=k|\theta)}{\sum_{k=1}^{K} p(y|x=k,\theta)P(x=k|\theta)}$$

$$Q(x[m]=k) = \frac{\pi_k^{(t)} N_k^{(t)}(y[m])}{\sum_{k=1}^{K} \pi_k^{(t)} N_k^{(t)}(y[m])}$$

$$N_k(y) = \frac{1}{\sqrt{|2\pi\Sigma|}} e^{-\frac{1}{2}(y-\mu_k)^T \Sigma^{-1}(y-\mu_k)}$$

- So the parameter π can be updated as (MLE)

$$\pi_k^{(t+1)} = \frac{1}{M}\sum_{m=1}^{M} Q^{(t)}(x[m]=k)$$

Learning in Gaussian Mixture Models

- Accordingly, the parameters of $p(y|x = k)$ can also be updated based on $Q^{(t)}(x[m] = k)$

$$\mu_k^{(t+1)} = f\left(y, Q^{(t)}\right) = \frac{\sum_{m=1}^{M} Q^{(t)}(x[m]=k)y[m]}{\sum_{m=1}^{M} Q^{(t)}(x[m]=k)}$$

$$\Sigma_k^{(t+1)} = f\left(y, Q^{(t)}, \mu_k^{(t+1)}\right) =$$

$$\frac{\sum_{m=1}^{M} Q^{(t)}(x[m]=k)\left(y[m]-\mu_k^{(t+1)}\right)\left(y[m]-\mu_k^{(t+1)}\right)^T}{\sum_{m=1}^{M} Q^{(t)}(x[m]=k)}$$

- Again, based on the updated parameters, we can easily calculate $Q^{(t+1)}(x[m] = k)$

Treat the expectation ***as a weight*** for the sample: a sample is divided into small pieces

Comments

Aim: $\arg\max_{\theta} p(\mathcal{D}|\theta)$

- The key problem for GMM learning is that both *the component labels* of the samples and *the model parameters* are unknown
- We can *iteratively* update the component labels based on *posteriors* (inference) and the model parameters based on *MLE* (learning)

Q1: will this iterative process converge?
Q2: If converged, can it get the unbiased estimation?

Expectation Maximization

- If a variable is unobserved or partially observed, *you can use all the possible values based on it posterior probability*
- The likelihood function can sum out these variables and then update the parameters which maximize the likelihood function
- Intuitive explanation
  - Given parameters do *INFERECE* for unobserved data
  - Maximize the marginal likelihood according to inference results (*LEARNING*)
  - Iteratively do inference and learning

General Principles

- The likelihood
  - Maximize the marginal over observed variables
    - $\max_{\theta} \mathrm{P}(D_{obs}|\theta) = \max_{\theta} \int \mathrm{P}(D_{obs}, X_{miss}|\theta) dX_{miss}$
  - Maximize the MAP over observed variables
    - $\max_{\theta} \mathrm{P}(D_{obs}|\theta) \approx \max_{\theta} \mathrm{P}\left(D_{obs}, \hat{X}_{miss}|\theta\right)$

- Iterative approaches
  - Define a scoring function (for example, likelihood function plus regularization terms)
  - Set parameters => do inference => Re-estimate parameters => re-do inference => …

GMM Inference Alternative: MAP

- The posterior as above

$$Q(x=k) = P(x=k|y,\theta) = \frac{\pi_k^{(t)} N_k^{(t)}(y[m])}{\sum_{k=1}^{K} \pi_k^{(t)} N_k^{(t)}(y[m])}$$

$$\tilde{Q}(x[m]=k) = \pi_k^{(t)} N_k^{(t)}(y[m])$$

- Set $x[m]$ as the MAP

$$x[m] = \arg\max_k \tilde{Q}(x[m]=k)$$

- Update the parameters based on the full samples (MLE)

Principle: assign the m-th sample to the most likely component

GMM Inference Alternative: Sampling

- The posterior as above

$$Q(x = k) = P(x = k|y, \theta) = \frac{\pi_k^{(t)} N_k^{(t)}(y[m])}{\sum_{k=1}^{K} \pi_k^{(t)} N_k^{(t)}(y[m])}$$

$$\tilde{Q}(x[m] = k) = \pi_k^{(t)} N_k^{(t)}(y[m])$$

- Directly sampling from $x \sim \tilde{Q}(x = k)$ to get full samples $(x[m], y[m])$
- Update the parameters based on the full samples (MLE)

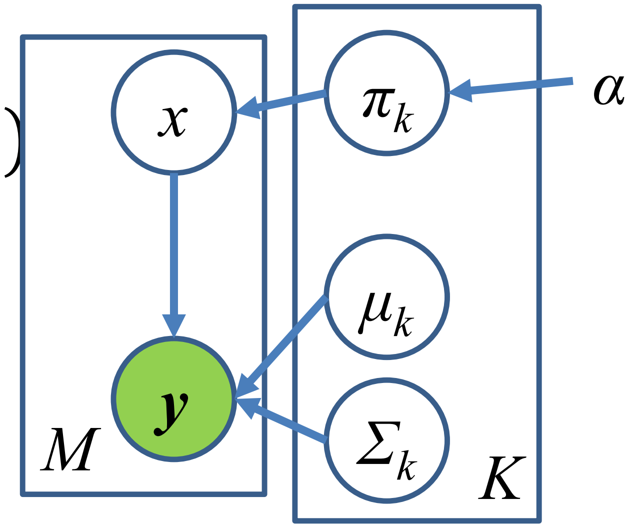
Principle: use samples from the posterior to approximate the distribution

GMM Learning Alternative: Hierarchical Bayesian Model

- The same for inference

  $\tilde{Q}(x[m] = k) = \pi_k^{(t)} N_k^{(t)}(y[m])$

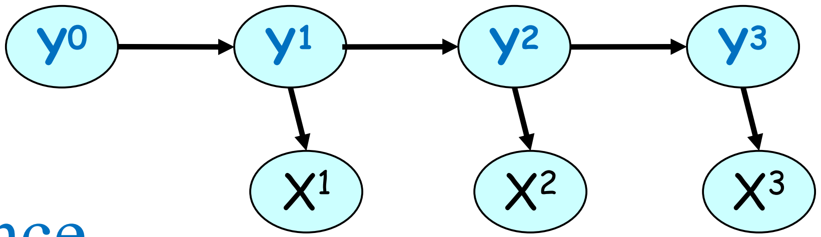
- Learning: re-calculate the posterior distributions of parameters based on the complete data
  - Priors
    - $p(\pi_k) = Dir(\alpha);\ p(\mu_k) = N\left(0, \frac{1}{L}\right);\ p(\Sigma_k) = N(0, I)$
  - Posteriors
    - $p(\pi_k|\mathcal{D}) = Dirichlet\big(\alpha_k + M^*(k)\big)$
    - $p(\mu_k|\mathcal{D}) \propto p(\mathcal{D}|\mu_k)p(\mu_k)$
    - $p(\Sigma_k|\mathcal{D}) \propto p(\mathcal{D}|\Sigma_k)p(\Sigma_k)$



Compute $\underset{\theta}{\operatorname{argmax}} \boldsymbol{P(X|\theta)}$

Baum–Welch Algorithm: Revisited

- General strategy
  1) set an initial value of the parameters
  2) then do ***inferences* of hidden values**
  3) then **re-*estimate* or re-*learn* the parameters**
  4) Repeat the processes till **convergence**



- Possible problems
  - No guarantee of convergence
  - No guarantee of global optimization

Baum–Welch Algorithm

- Define an intermediate variables for inference
  - The probability of ***y=i*** for time point ***t*** and ***y=j*** for the following time point ***t+1***

$$\xi_t(i,j) = P(y_t = i, y_{t+1} = j \mid X, \theta)$$

Principle: due to the *Markov* property, all the local transitions are “equal”.

Compute $\underset{\theta}{\operatorname{argmax}} \boldsymbol{P(X|\theta)}$

Baum–Welch Algorithm

$$\theta^0 = \left\{T^0, E^0, \pi^0\right\}$$

- Use forward and backward algorithm to calculate probability

$$\alpha_t(i) = P\left(x_1, \cdots \qquad = i \mid \theta^0\right)$$

$$\beta_t(i) = P\left(x_{t+1}, \cdots \qquad = i, \theta^0\right)$$

$$\xi_t(i,j) = P\left(y_t = i, y_{t+1} = j \mid X, \theta\right)$$

$$\Longrightarrow \quad \xi_t(i,j) = \frac{\alpha_t(i)\, t_{i,j}\, e_{j,x_{t+1}}\, \beta_{t+1}(j)}{\sum_{i=1}^{Y}\sum_{j=1}^{Y} \alpha_t(i)\, t_{i,j}\, \beta_{t+1}(j)\, e_{j,x_{t+1}}}$$

Inference for the local transitions

Compute $\underset{\theta}{\operatorname{argmax}}\, \boldsymbol{P(X|\theta)}$

Baum–Welch Algorithm

- The probability of ***Y=i*** for time point ***t***:

$$\gamma_t(i) = \sum_{j=1}^{Y} \xi_t(i,j)$$

- So expected times for state stayed in ***Y=i*** and the expected times for state transition ***i-j***:

$$\sum_{t=0}^{T-1} \gamma_t(i) \qquad\qquad \sum_{t=0}^{T-1} \xi_t(i,j)$$

Compute $\underset{\theta}{\operatorname{argmax}} \boldsymbol{P(X|\theta)}$

Baum–Welch Algorithm

- Re-estimate all parameters (MLE)

$$t_{i,j} = \frac{\sum_{t=0}^{T-1} \xi_t(i,j)}{\sum_{t=0}^{T-1} \gamma_t(i)} \qquad e_{i,x} = \frac{\sum_{t=0}^{T} \mathrm{I}(x_t = x)\gamma_t(i)}{\sum_{t=0}^{T} \gamma_t(i)}$$

- Repeat above steps until convergence

$$\left|\log\left(P(X \mid \theta)\right) - \log\left(P\left(X \mid \theta^0\right)\right)\right| < \varepsilon$$

Generate the hidden variables using current parameter $\boldsymbol{Y}^t \sim \mathrm{P}(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{\theta}^t)$

Replace the Inference by Sampling

- Initialization: generate the labeling sequencing according to the prior probability

$$\pi \Rightarrow Y_t = y_i, \quad 1 \le t \le T$$

- Re-generate Y_t according to the initial setting

$$P(y_t \mid Y, X, \theta) = P(y_t \mid y_{t-1}, y_{t+1}, x_t, \theta)$$

$$= \frac{P(y_{t+1}, x_t \mid y_t, y_{t-1}, \theta) P(y_t \mid y_{t-1}, \theta)}{P(y_{t+1}, x_t \mid y_{t-1}, \theta)}$$

$$\propto P(y_{t+1} \mid y_t, \theta) P(x_t \mid y_t, \theta) P(y_t \mid y_{t-1}, \theta)$$

$$= t_{y_t, y_{t+1}} e_{y_t, e_t} t_{y_{t-1}, y_t}$$

We get a "score" proportional to the probability. So we can randomly generate Y_t based on these scores.

MLE for Updating Parameters

- Simply update the parameter in transition and emission probability matrix

$$t_{i \to j} = \frac{\sum_{t=1}^{T} \mathrm{I}\left(Y^{t+1}=j, Y^{t}=i\right)}{\sum_{t=1}^{T} \mathrm{I}(Y^{t}=i)}$$

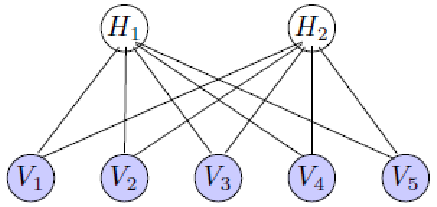
Note: $\mathrm{I}(\cdot)$ is an indicator function

$$e_{i \to k} = \frac{\sum_{t=1}^{T} \mathrm{I}\left(Y^{t}=i, X^{t}=k\right)}{\sum_{t=1}^{T} \mathrm{I}(Y^{t}=i)}$$

- *Q: Why does Gibbs Sampling also work?*

Learning in RBMs

3. An RBM(Resctricted Boltzmann Machine) is a bipartite Markov network consisting of a visible (observed) layer and a hidden layer, where each node is a 0-1 binary random variable. Consider the following RBM:



Hidden variables

Observed variables

The joint distribution of a configuation is given by:

$$P(\boldsymbol{H}=\boldsymbol{h},\boldsymbol{V}=\boldsymbol{v})=\frac{1}{Z}e^{-E(\boldsymbol{h},\boldsymbol{v})} \qquad \boldsymbol{h}=(h_1,h_2)^T,\boldsymbol{v}=(v_1,\cdots,v_5)^T$$

And:

$$E(\boldsymbol{h},\boldsymbol{v})=-\sum_{i=1}^{2}a_ih_i-\sum_{j=1}^{5}b_jv_j-\sum_{ij}w_{ij}h_iv_j$$

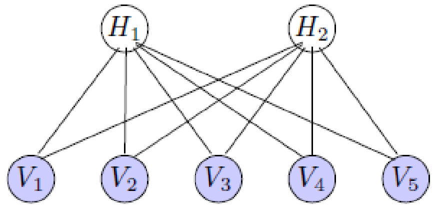
Inference step: all parameters are given

$$\tilde{E}(h_1)=-a_1h_1-\sum_{j=1}^{5}w_{1,j}h_1v_j$$

$$\tilde{E}(h_2)=-a_2h_2-\sum_{j=1}^{5}w_{2,j}h_2v_j$$

Proportional sampling according to above likelihood

3. An RBM(Resctricted Boltzmann Machine) is a bipartite Markov network consisting of a visible (observed) layer and a hidden layer, where each node is a 0-1 binary random variable. Consider the following RBM:



Hidden variables

Observed variables

The joint distribution of a configuation is given by:

$$P(\boldsymbol{H}=\boldsymbol{h},\boldsymbol{V}=\boldsymbol{v})=\frac{1}{Z}e^{-E(\boldsymbol{h},\boldsymbol{v})} \qquad \boldsymbol{h}=(h_1,h_2)^T, \boldsymbol{v}=(v_1,\cdots,v_5)^T$$

And:

$$E(\boldsymbol{h},\boldsymbol{v})=-\sum_{i=1}^{2}a_ih_i-\sum_{j=1}^{5}b_jv_j-\sum_{ij}w_{ij}h_iv_j$$

Learning step: all variables are observed

Recall: parameter learning in Markov Networks (gradient ascent)

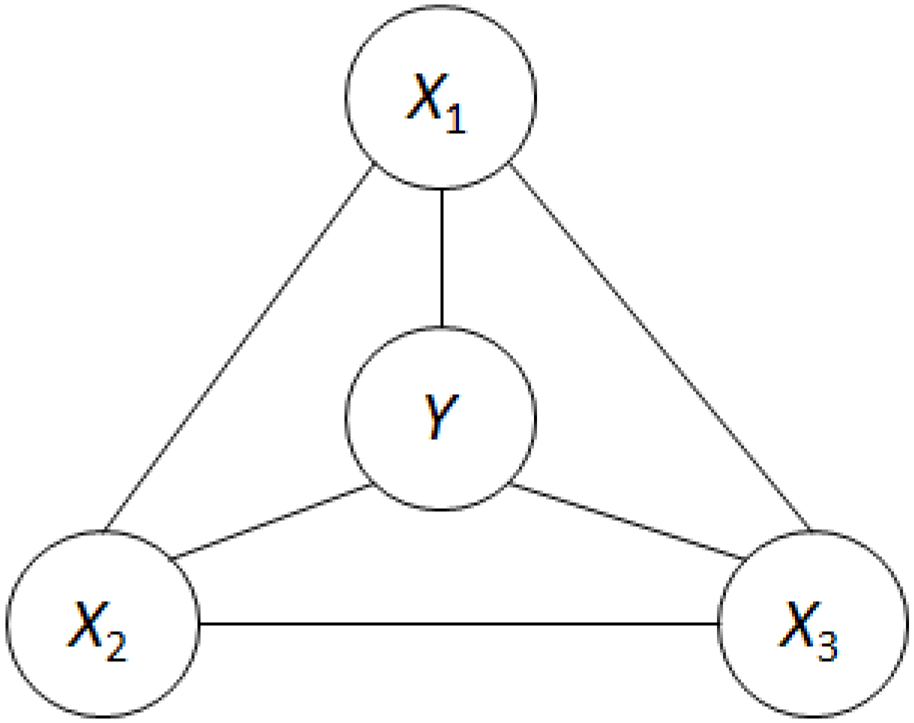
$$\frac{1}{M}l(\theta:x)=a_1E(h_1)+a_2E(h_2)+\sum_{j=1}^{5}w_{1,j}E(h_1v_j)+\sum_{j=1}^{5}w_{2,j}E(h_2v_j)+\sum_{j=1}^{5}b_jE(v_j)-\ln Z$$

$$\frac{1}{M}\frac{\partial l(\theta:x)}{\partial\theta}\Rightarrow\theta^{t+1}=\theta^t+\lambda^t\frac{1}{M}\frac{\partial l(\theta:x)}{\partial\theta}$$

OR ***stochastic gradient ascent*** sample by sample!

How About Other Models?

- Most popular probabilistic graphic models contain hidden variables:
  - For latent factor analysis?
  - For restricted Boltzmann machine?
  - For latent *Dirichlet* allocation?
  - For conditional random fields?
  - …

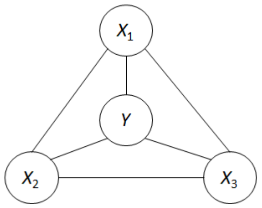


x: $\alpha_1=\alpha_2=\alpha_3=\alpha$

y: $\beta=2\alpha$

xx: $w_{12}=w_{13}=w_{23}=w$

xy: $h_{1y}=h_{2y}=h_{3y}=-w$



x: $\alpha_1 = \alpha_2 = \alpha_3 = \alpha$

y: $\beta = 2\alpha$

xx: $w_{12} = w_{13} = w_{23} = w$

xy: $h_{1y} = h_{2y} = h_{3y} = -w$

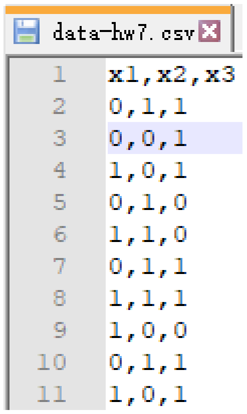
Inference Step

$$\boxed{P(x, y \mid \alpha, w) = \frac{1}{Z}\exp\left\{\alpha\left(x_1 + x_2 + x_3 + 2y\right) + w\left(x_1x_2 + x_2x_3 + x_1x_3 - y\left(x_1 + x_2 + x_3\right)\right)\right\}}$$

$$P(y \mid x, \alpha, w) \propto P(y, x \mid \alpha, w)$$

$$\Rightarrow \frac{P(y=1, x \mid \alpha, w)}{P(y=0, x \mid \alpha, w)} = \exp\left\{2\alpha + w\left(-\left(x_1 + x_2 + x_3\right)\right)\right\} = s$$

$$\Rightarrow P(y=1, x \mid \alpha, w) = \frac{s}{1+s}, \quad P(y=0, x \mid \alpha, w) = \frac{1}{1+s}$$


data-hw7.csv

```
x1,x2,x3
0,1,1
0,0,1
1,0,1
0,1,0
1,1,0
0,1,1
1,1,1
1,0,0
0,1,1
1,0,1
```

You can either use the expectation (probability) or *sampling* to fill the single hidden variable!!

Learning Step (SGD)

$$P(x,y \mid \alpha,w)=\frac{1}{Z}\exp\left\{\alpha\left(x_1+x_2+x_3+2y\right)+w\left(x_1x_2+x_2x_3+x_1x_3-y\left(x_1+x_2+x_3\right)\right)\right\}$$

- SGD: $\boldsymbol{\theta}^{(k+1)}=\boldsymbol{\theta}^{(k)}+\lambda^{(k)}\nabla l_{m_k}(\boldsymbol{\theta};\boldsymbol{x}[m_k])$
- Calculate the partial derivatives
  - For α
  - For w

$$\begin{aligned}\frac{\partial}{\partial\theta_i}\ln Z(\boldsymbol{\theta}) &= \frac{1}{Z(\boldsymbol{\theta})}\sum_{\xi}\frac{\partial}{\partial\theta_i}\exp\left\{\sum_j\theta_j f_j(\xi)\right\}\\ &= \frac{1}{Z(\boldsymbol{\theta})}\sum_{\xi}f_i(\xi)\exp\left\{\sum_j\theta_j f_j(\xi)\right\}\\ &= \boldsymbol{E}_{\boldsymbol{\theta}}[f_i].\end{aligned}$$

How to Deal With Missing Values?

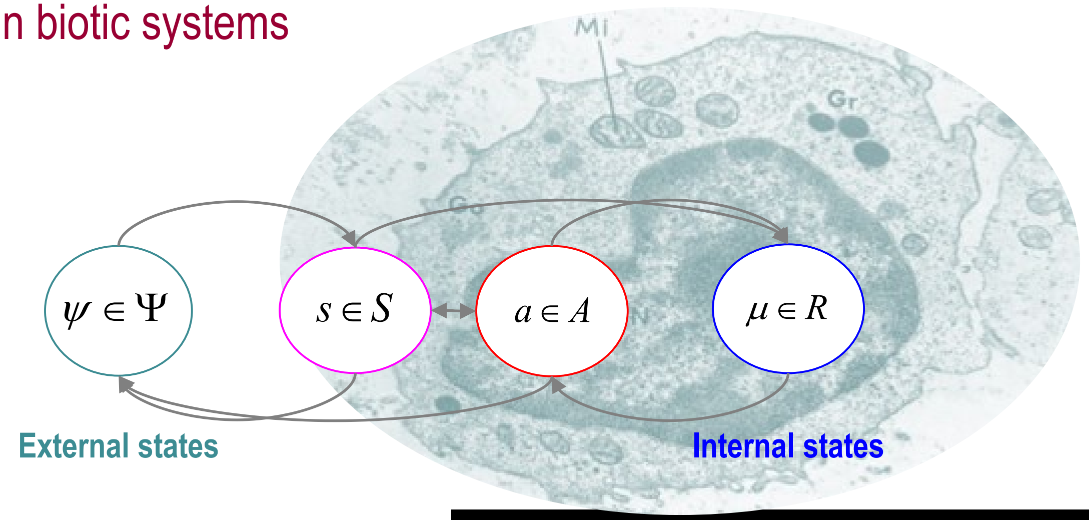
- Under the framework of probabilistic inferences, just use the same strategy as filling the hidden variables!

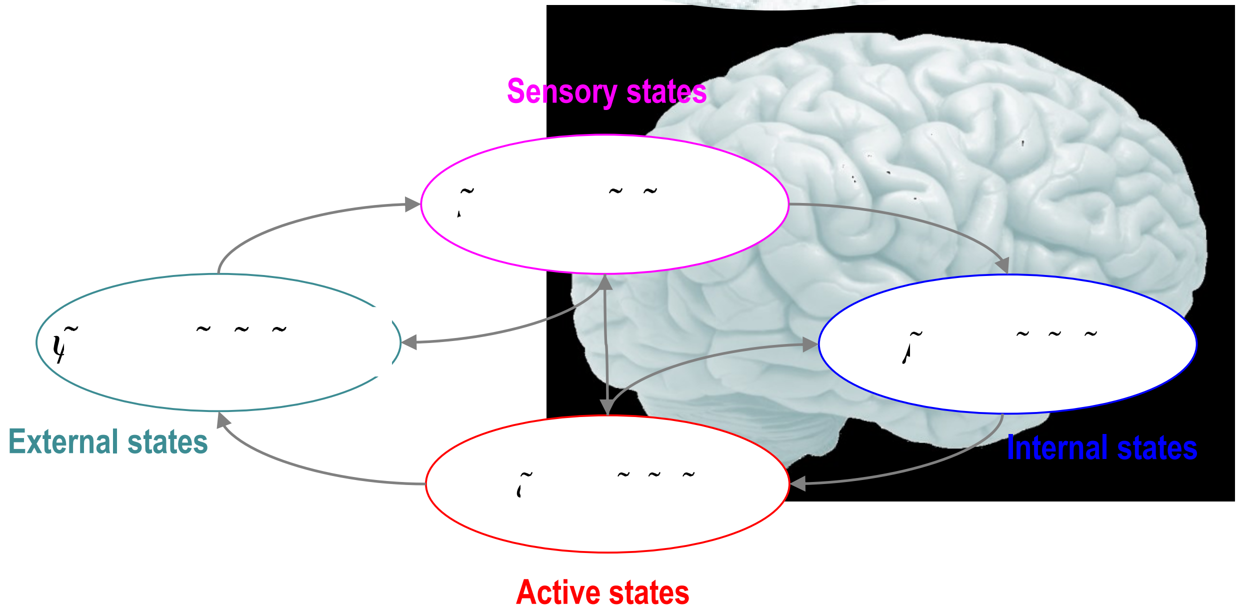
Generalization

Inference
Learning

- Inference
  - MCMC
  - Variational inference
  - Belief propagation
- Learning
  - MLE, MAP
  - Gradient ascent/descent
  - Hierarchical Bayesian

The Markov blanket in biotic systems



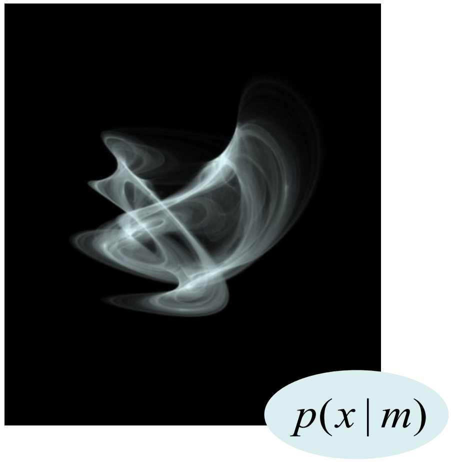


Free energy and active inference

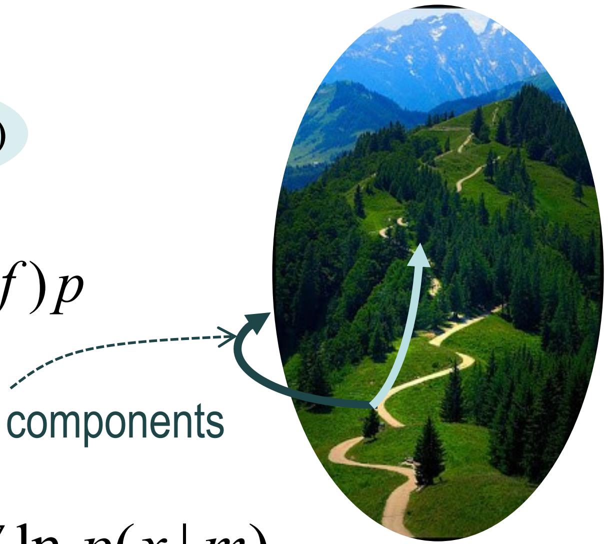
Karl Friston, University College London

lemma: *any (ergodic random) dynamical system (m) that possesses a Markov blanket will appear to actively maintain its structural and dynamical integrity*

$:$ ω



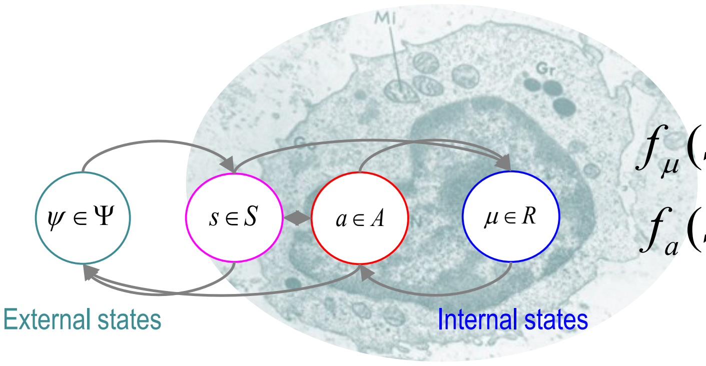
The Fokker-Planck equation $\quad \nabla \cdot (\Gamma \nabla - f) p$



And its solution in terms of curl-free and divergence-free components

$$0 \Leftrightarrow f(x) = (\Gamma - Q) \nabla \ln p(x \mid m)$$

But what about the Markov blanket?



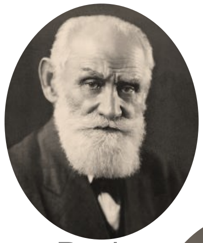
$$f_\mu(s,a,\mu) = (\Gamma - Q)\nabla_\mu \ln p(s,a,\mu \mid m)$$
$$f_a(s,a,\mu) = (\Gamma - Q)\nabla_a \ln p(s,a,\mu \mid m)$$

$\ln p(s,a,\mu \mid m) =$ **Value** → Reinforcement learning, optimal control and expected utility theory

$-\ln p(s,a,\mu \mid m) =$ **Surprise (free energy)** → Information theory and minimum redundancy

$E_t[-\ln p(s,a,\mu \mid m)] =$ **Entropy** → Self-organisation, cybernetics and homoeostasis

$p(\tilde{\ } \ \tilde{\ }$ = **Model evidence** → Bayesian brain, active inference and predictive coding

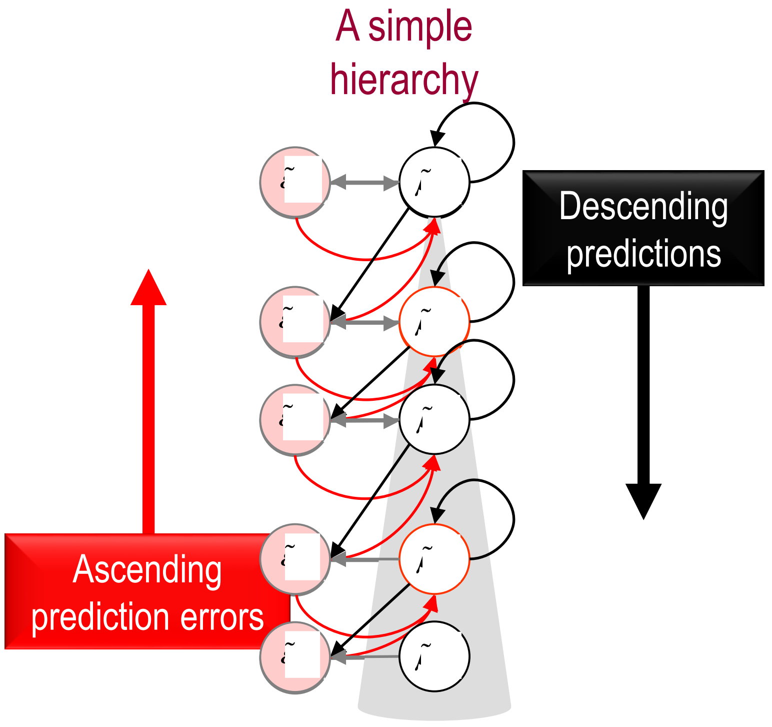

Pavlov

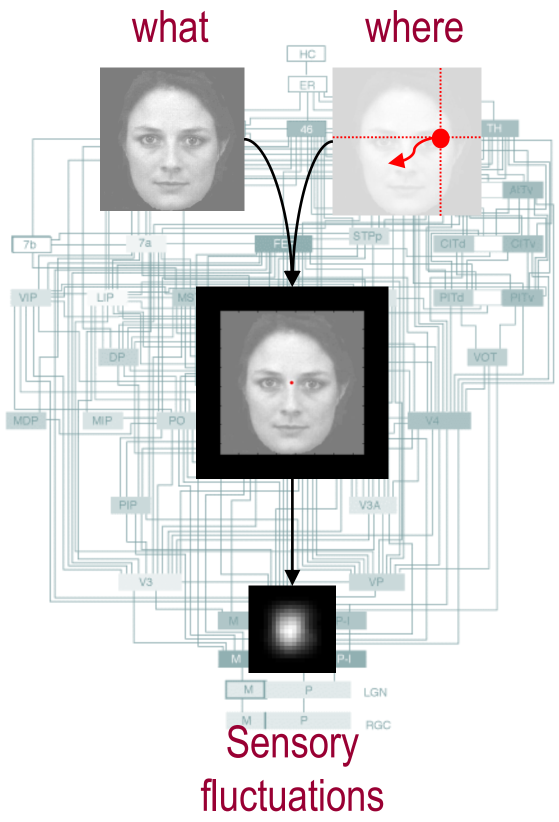
Barlow

Ashby

Helmholtz

Hierarchical generative models

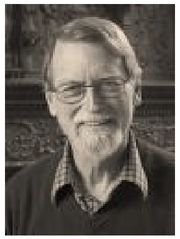


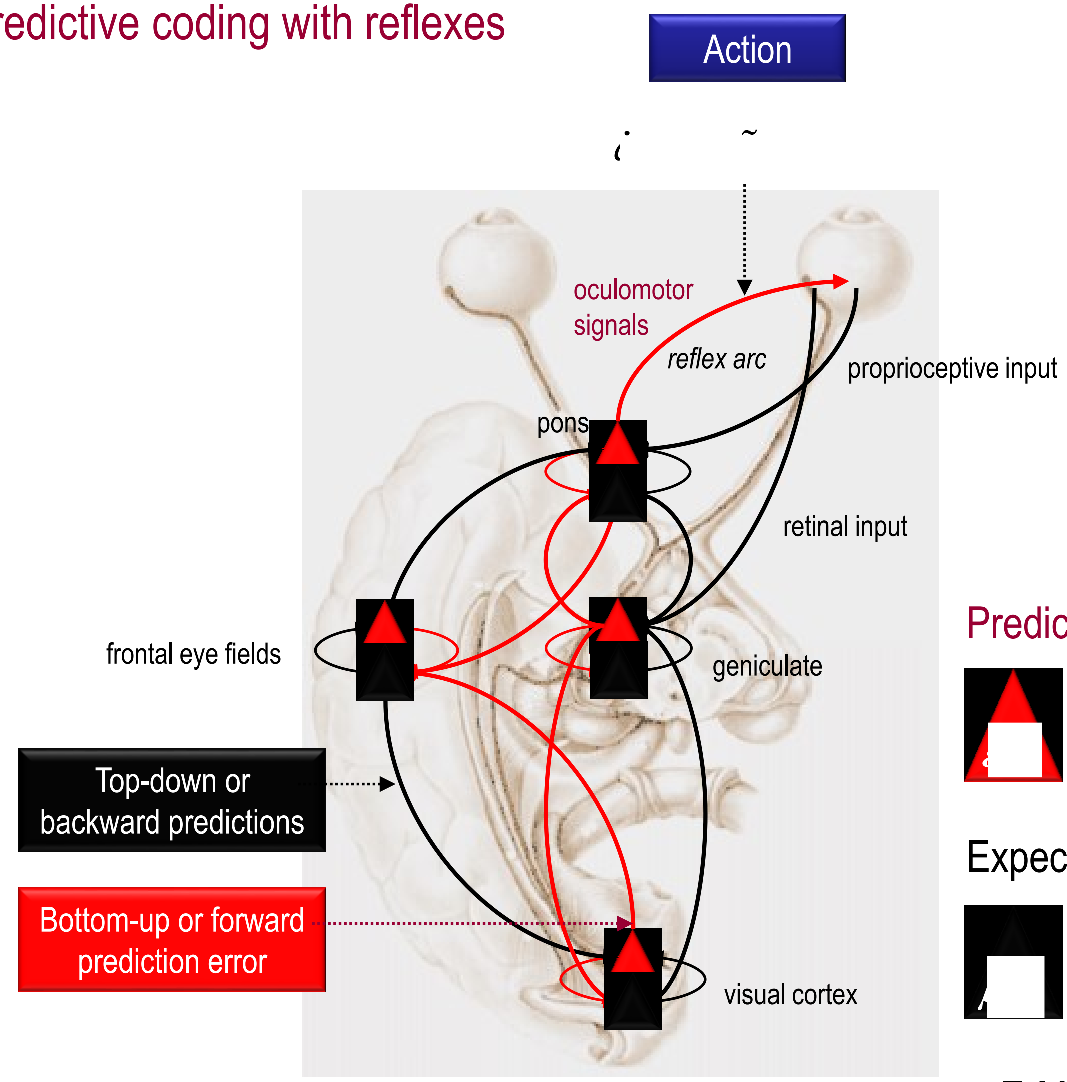


Kalman filter

Predictive coding with reflexes

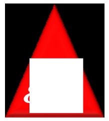
David Mumford



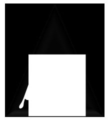


Perception

Prediction error (superficial pyramidal cells)



Expectations (deep pyramidal cells)



E-M algorithm!
Message passing / Belief propagation

Why Need Deep Latent Models?

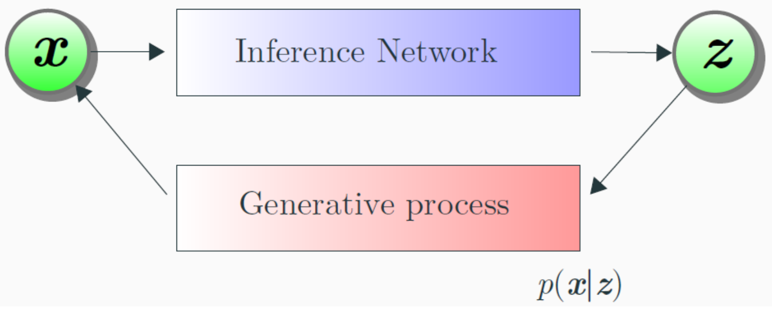
- Low-order models cannot fully understand the data generated from high-order models

Deep VAE & GAN

- VAE
  - Target distribution in *low-dimension subspace*
  - Minimize the likelihood
- GAN
  - Learn a *latent generator* by mapping noise to data
  - Minimize the discrimination

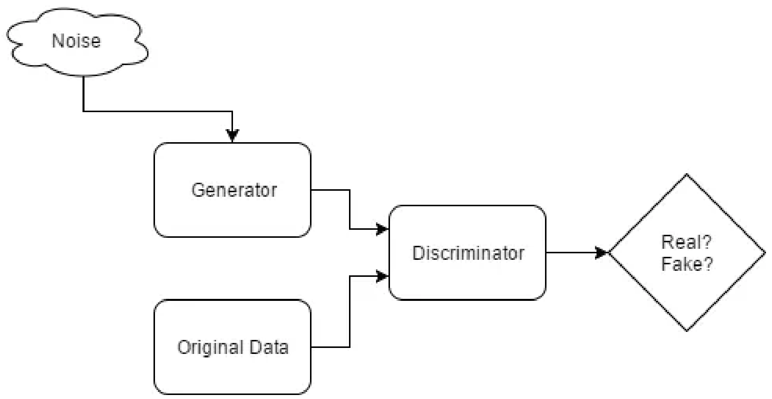
$$\begin{aligned}\mathcal{L}(\theta;\boldsymbol{x}) &= \mathbb{E}_{z\sim q(z|x;\theta)} \log p(\boldsymbol{x},\boldsymbol{z}) + \mathbb{H}(q(z|x;\theta)) \\ &= \mathbb{E}_{z\sim q(z|x;\theta)} \log p(\boldsymbol{x}|\boldsymbol{z}) - \mathbb{KL}(q(\boldsymbol{z}|\boldsymbol{x};\theta)\ ||\ p(\boldsymbol{z}))\end{aligned}$$

$$q(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(f_\mu(\boldsymbol{x}), f_\sigma(\boldsymbol{x}))$$



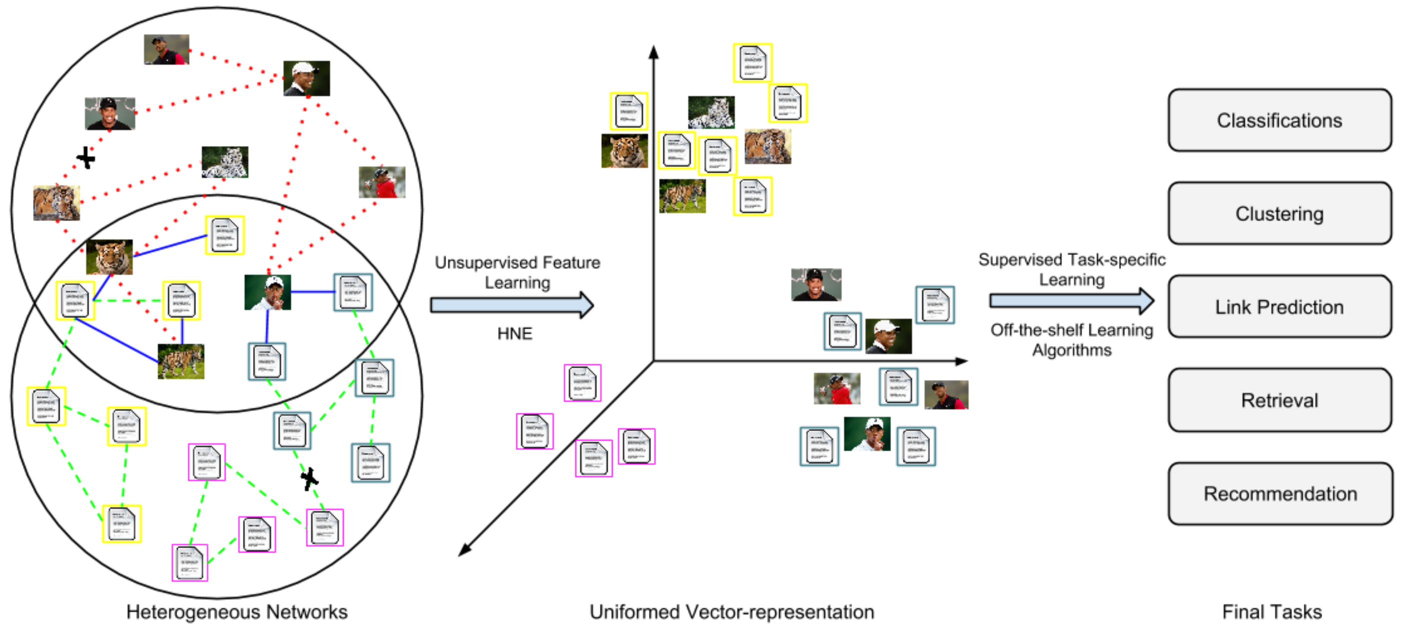
$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{D},\mathcal{G}) =$$

$$\mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log \mathcal{D}(x)] + \mathbb{E}_{(\mathbf{z})\sim p_z(((\mathbf{z})))}[\log(1-\mathcal{D}(\mathcal{G}(\mathbf{z})))]$$

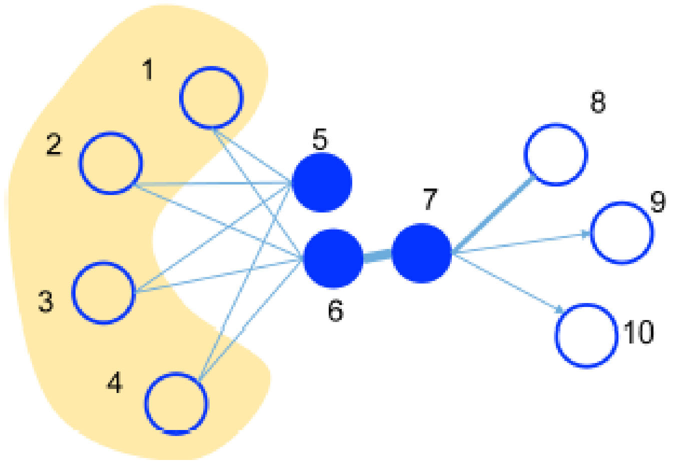


Embedding for Relevant Networks

- Task: mapping nodes in networks to a latent Euclidean space (low-dimensional subspace) by preserving their "similarities"


Unsupervised Feature Learning
HNE
Supervised Task-specific Learning
Off-the-shelf Learning Algorithms
Classifications
Clustering
Link Prediction
Retrieval
Recommendation
Heterogeneous Networks
Uniformed Vector-representation
Final Tasks

LINE: Second-Order Similarity



- First-order similarity

$$p_1(v_i, v_j) = \frac{1}{1 + \exp(-\vec{u}_i^T \cdot \vec{u}_j)}$$

$$O_1 = - \sum_{(i,j)\in E} w_{ij} \log p_1(v_i, v_j)$$

- Second-order similarity

$$p_2(v_j|v_i) = \frac{\exp(\vec{u}_j'^T \cdot \vec{u}_i)}{\sum_{k=1}^{|V|} \exp(\vec{u}_k'^T \cdot \vec{u}_i)}$$

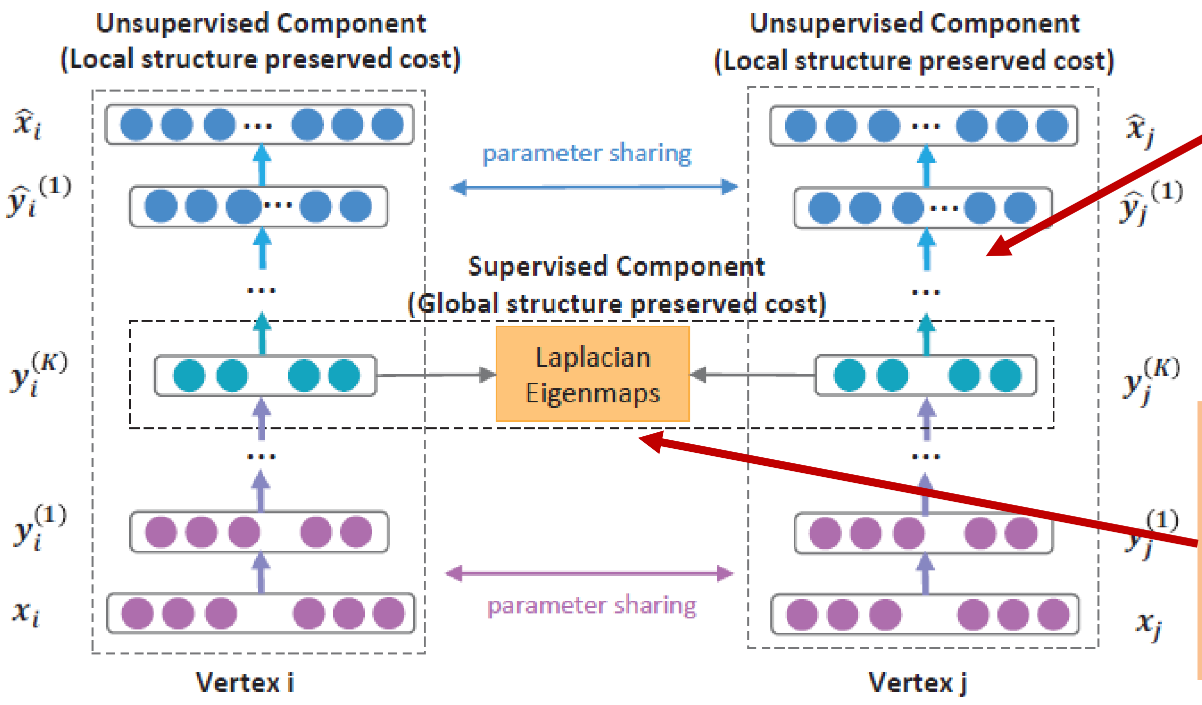
$$O_2 = - \sum_{(i,j)\in E} w_{ij} \log p_2(v_j|v_i)$$

Second-order relations are more stable than the *first-order*

Tang *et al*. LINE: large-scale information network embedding. *IW3C2* 2015.

Deep Structural Network Embedding

- Aim: find a low-dimensional representation of the adjacent matrix *S*



AutoEncoder for dimension reduction (unsupervised)

The representations in the reduced space should mimic the Laplacian eigenmap (supervised)

Want *et al*. Structural deep network embedding. *KDD* 2016.

Deep Structural Network Embedding

- First-order penalty

$$\begin{aligned}\mathcal{L}_{1st} &= \sum_{i,j=1}^{n} s_{i,j}\|\mathbf{y}_i^{(K)} - \mathbf{y}_j^{(K)}\|_2^2 \\ &= \sum_{i,j=1}^{n} s_{i,j}\|\mathbf{y}_i - \mathbf{y}_j\|_2^2\end{aligned}$$

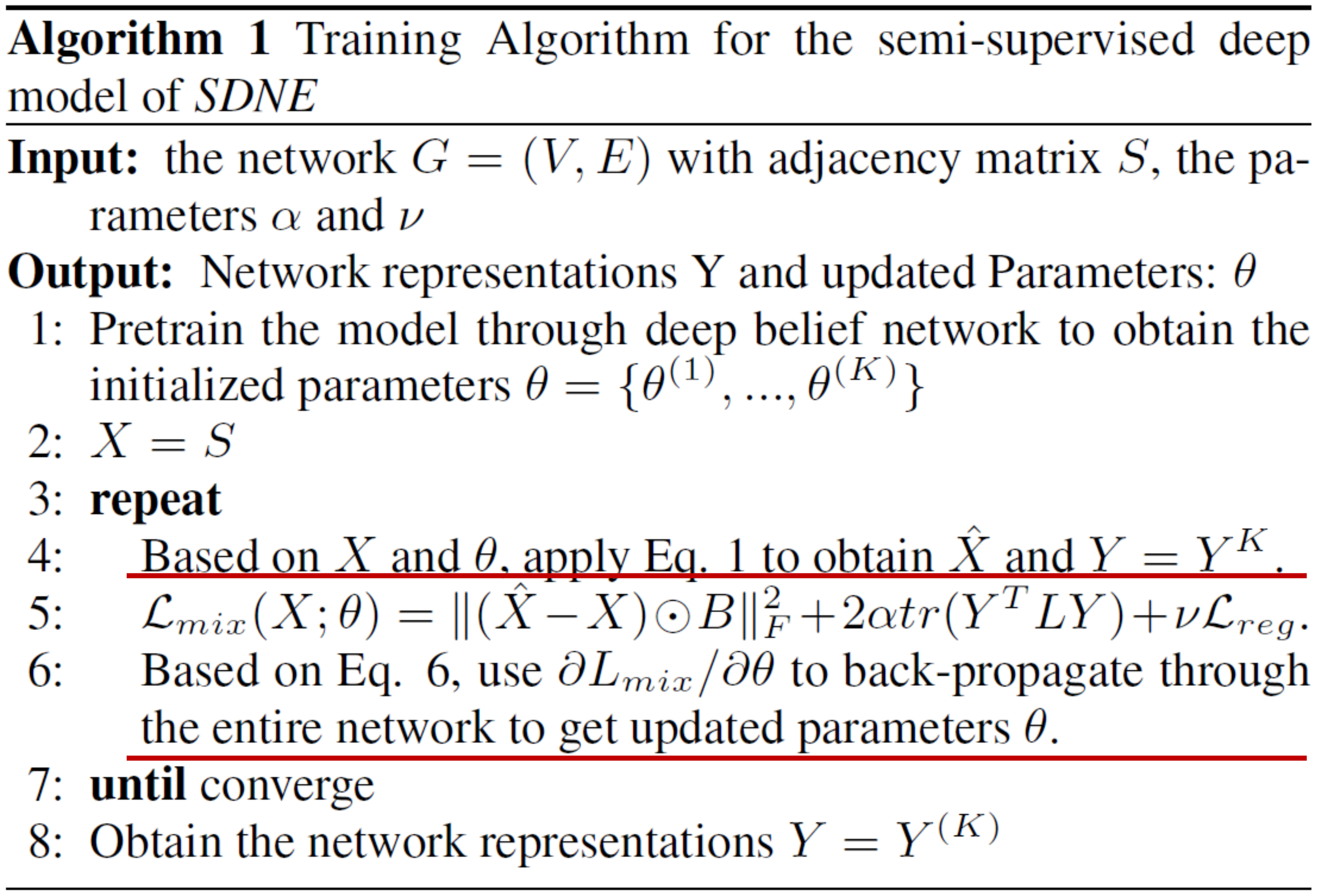
- Second-order penalty

$$\begin{aligned}\mathcal{L}_{2nd} &= \sum_{i=1}^{n} \|(\hat{\mathbf{x}}_i - \mathbf{x}_i) \odot \mathbf{b_i}\|_2^2 \\ &= \|(\hat{X} - X) \odot B\|_F^2\end{aligned}$$

$$\boxed{\begin{aligned}\mathcal{L}_{mix} &= \mathcal{L}_{2nd} + \alpha\mathcal{L}_{1st} + \nu\mathcal{L}_{reg} \qquad \boxed{\mathcal{L}_{reg} = \frac{1}{2}\sum_{k=1}^{K}(\|W^{(k)}\|_F^2 + \|\hat{W}^{(k)}\|_F^2)} \\ &= \|(\hat{X} - X) \odot B\|_F^2 + \alpha \sum_{i,j=1}^{n} s_{i,j}\|\mathbf{y}_i - \mathbf{y}_j\|_2^2 + \nu\mathcal{L}_{reg}\end{aligned}}$$

$$\boxed{\begin{aligned}\frac{\partial\mathcal{L}_{mix}}{\partial\hat{W}^{(k)}} &= \frac{\partial\mathcal{L}_{2nd}}{\partial\hat{W}^{(k)}} + \nu\frac{\partial\mathcal{L}_{reg}}{\partial\hat{W}^{(k)}} \\ \frac{\partial\mathcal{L}_{mix}}{\partial W^{(k)}} &= \frac{\partial\mathcal{L}_{2nd}}{\partial W^{(k)}} + \alpha\frac{\partial\mathcal{L}_{1st}}{\partial W^{(k)}} + \nu\frac{\partial\mathcal{L}_{reg}}{\partial W^{(k)}}, k = 1, ..., K\end{aligned}}$$

Deep Structural Network Embedding



Algorithm 1 Training Algorithm for the semi-supervised deep model of *SDNE*

```
Input: the network $G = (V, E)$ with adjacency matrix $S$, the parameters $\alpha$ and $\nu$
Output: Network representations Y and updated Parameters: $\theta$
1: Pretrain the model through deep belief network to obtain the initialized parameters $\theta = \{\theta^{(1)}, ..., \theta^{(K)}\}$
2: $X = S$
3: repeat
4:     Based on $X$ and $\theta$, apply Eq. 1 to obtain $\hat{X}$ and $Y = Y^K$.
5:     $\mathcal{L}_{mix}(X;\theta) = \|(\hat{X} - X) \odot B\|_F^2 + 2\alpha tr(Y^T L Y) + \nu \mathcal{L}_{reg}$.
6:     Based on Eq. 6, use $\partial L_{mix}/\partial\theta$ to back-propagate through the entire network to get updated parameters $\theta$.
7: until converge
8: Obtain the network representations $Y = Y^{(K)}$
```

Inference

Learning

Summary: General Principles

- The likelihood
  - Maximize the marginal over observed variables
    - $\max_{\theta} \mathrm{P}(D_{obs}|\theta) = \max_{\theta} \int \mathrm{P}(D_{obs}, X_{miss}|\theta) dX_{miss}$
  - Maximize the MAP over observed variables
    - $\max_{\theta} \mathrm{P}(D_{obs}|\theta) \approx \max_{\theta} \mathrm{P}\left(D_{obs}, \hat{X}_{miss}|\theta\right)$
- Iterative approaches
  - Define a scoring function (for example, likelihood function plus regularization terms)
  - Set parameters => do inference => Re-estimate parameters => re-do inference => …

General Principles Extended

- Define an objective/scoring function
  - With both parameters and hidden variables (or missing values)
  - Set parameters => do inference => re-estimate parameters => re-do inference => …
  - Stop until the objective/scoring function converges (or the posterior probability converges for Bayesian models)

The End

Do Inference & Learning Alternatively