INF8225 – Leçon 4
Partie 1

Christopher Pal
École Polytechnique de Montréal

Au menu

- Le problème avec les cycles
- Les arbres d'intersections « junction trees» et d'autres algorithmes d'inférence
- D-séparation « dependancy separation »
- Apprentissage dans un réseau bayésien quand toutes les variables sont observées
- L'algorithme EM dans un réseau bayésien - quand il y des variables cachées

Quel est le problème avec les cycles?

- Exercice: Créer un réseau bayésien pour les relations en logique suivantes :

  Si $A=a_1$, alors $B = b_1$ et $C = c_1$

  Si $A=a_2$, alors $B = b_2$ et $C = c_1$

  Si $A=a_3$, alors $B = b_1$ et $C = c_2$

  Si $B=b_1$ et $C = c_1$, alors $D = d_1$

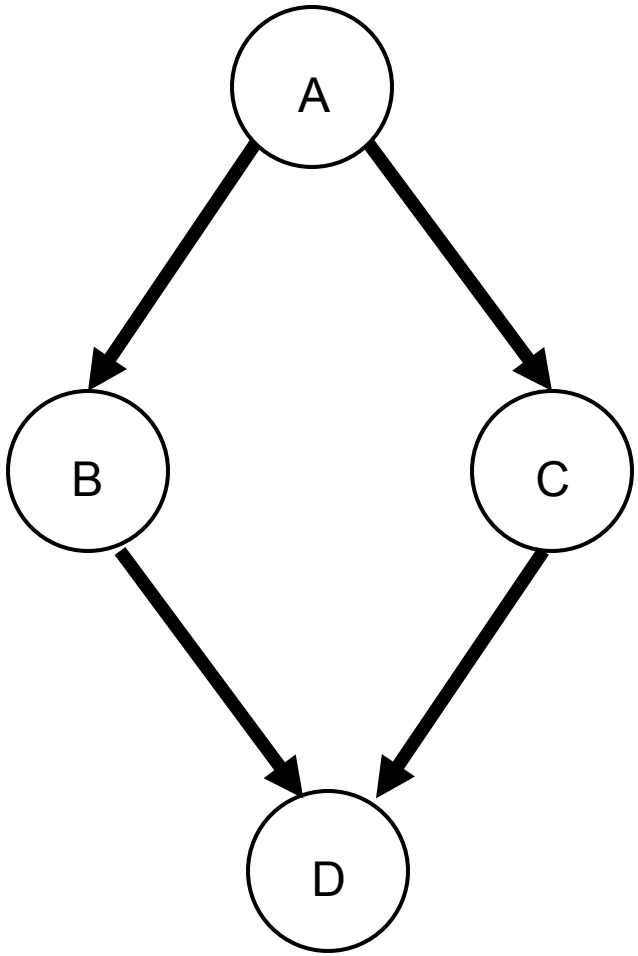
  Si $B=b_1$ et $C = c_2$, alors $D = d_2$

  Si $B=b_2$ et $C = c_1$, alors $D = d_2$

  Si $B=b_2$ et $C = c_2$, alors $D = d_1$

- Avec l'observation : $D = d_1$

Quel est le problème avec les cycles?



- En utilisant une stratégie avec des messages seulement sur P(B) et P(C), mais pas sur P(B,C) nous allons avoir un problème si notre but est d'inférer (correctement) que $A=a_1$

Quel est le problème avec les cycles?

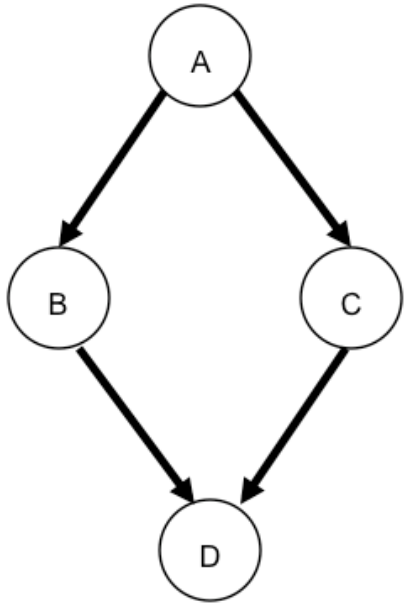
Si $A=a_1$, alors $B = b_1$ et $C = c_1$

Si $A=a_2$, alors $B = b_2$ et $C = c_1$

Si $A=a_3$, alors $B = b_1$ et $C = c_2$

Si $B=b_1$ et $C = c_1$, alors $D = d_1$

Si $B=b_2$ et $C = c_2$, alors $D = d_1$



- Avec l'observation : $D = d_1$
  En utilisant une stratégie où l'information sur b et c peuvent devenir découplés, nous allons avoir un problème.

En pratique il y a une autre représentation utilisé pour obtenir des calculs exacte

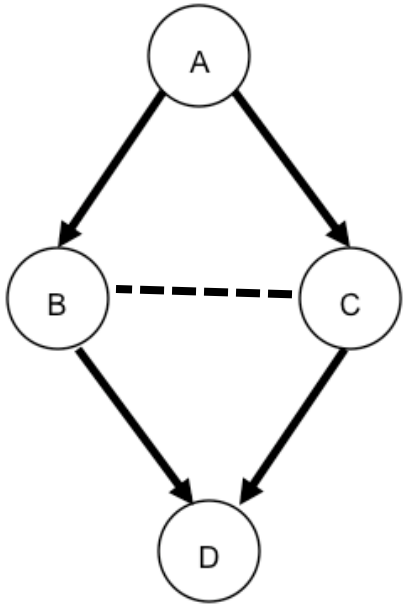
- On peut transformer notre réseau de Bayes à la représentation l'ensemble - chaîne « set-chain representation »
- Il consiste de toutes les variables X_G dans un graphe et il utilise le concept de cliques X_c et séparateurs X_s

$$P(X_G) = \frac{\prod_{c \in C} P(X_c)}{\prod_{s \in S} P(X_s)} \qquad P(X_G = x_g) = \frac{\prod_{c \in C} \phi_c(X_c = x_c)}{\prod_{s \in S} \varphi_s(X_s = x_s)}$$

- La deuxième équation est appelée une représentation potentielle « potential representation »

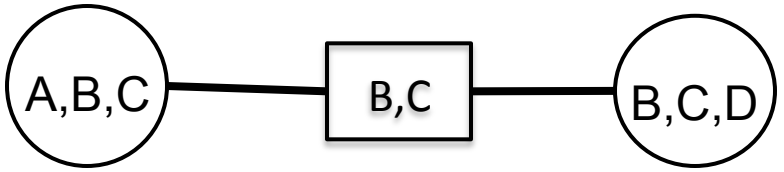
Exemple Simple

- Etant donné une observation D=d
- Calculer P(A|D=d) exactement!



$$P(X_G) = \frac{\prod_{c \in C} P(X_c)}{\prod_{s \in S} P(X_s)}$$

$$= \frac{P(A,B,C)P(B,C,D)}{P(B,C)}$$



- Étape 1: Transformer le réseau à un arbre consistant de clusters et les intersections, « a junction tree »
- Étape 2: Appliquer un autre algorithme utilisant le passage de messages (voir Huang et Darwiche, 1995)

Exact inference in general Bayes Nets

(See Huang and Darwiche, 1995 for more details)

1. Construct an undirected graph, called a moral graph, from the DAG (the Bayes Net).
2. Selectively add arcs to the moral graph to form a triangulated graph.
3. From the triangulated graph, identify select subsets of nodes, called cliques.
4. Build a join tree (junction tree), starting with the cliques as clusters: connect the clusters to form an undirected tree satisfying the join tree property, inserting the appropriate sepsets.
5. *Apply another alternative message passing algorithm in this graph (see the paper for details)*

Pour construire un arbre de jonction

1. Construire un graphe non orienté, appelé un graphe moral de la DAG -- effacer les orientations et ajouter des arêtes entre des parents
2. Ajouter sélectivement des arcs au graphe moral pour former un graphe triangulé
   – c.-à-d. chaque cycle de la longueur de quatre ou plus contiens une arête entre deux noeuds non adjacents dans le cycle

Exemple : la construction d'un graphe moral

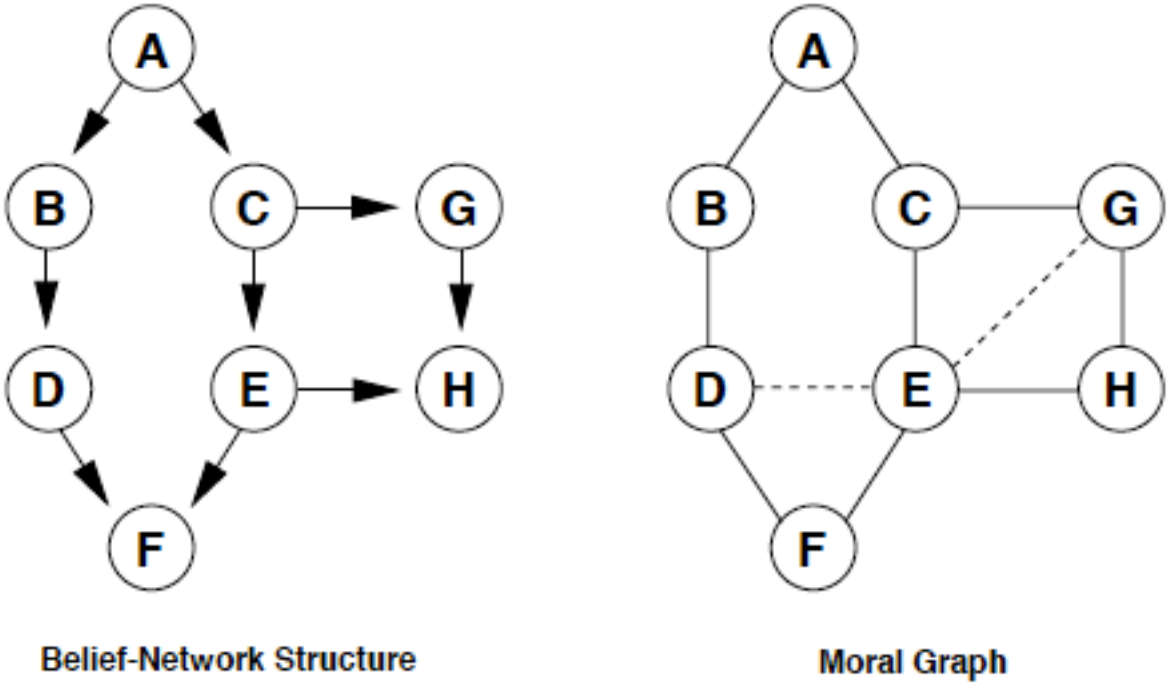


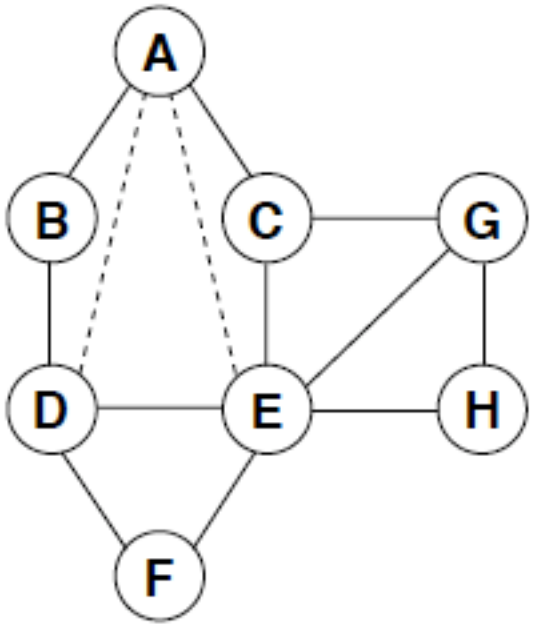
Figure 1. Constructing the moral graph.

Huang et Darwiche, 1995

La création d'un arbre de jonction

3. Dans le graphe triangulé, identifier des sous-ensembles des nœuds appelés des cliques -- c.-à-d. des sous-ensembles où il existe une arête entre chaque nœud dans l'ensemble.
4. Construire un arbre d'intersections « junction tree», en commençant par les cliques comme clusters, puis joindre les clusters pour former un arbre non orienté satisfaisant la propriété des arbres d'intersections en ajoutant des séparateurs «sepsets» appropriés (variables en commun)

La triangulation d'un graphe moral



Triangulated Graph

Eliminated Vertex	Induced Cluster	Edges Added
H	EGH	none
G	CEG	none
F	DEF	none
C	ACE	(A,E)
B	ABD	(A,D)
D	ADE	none
E	AE	none
A	A	none

Elimination Ordering

Figure 2. Triangulating the moral graph.

Article de Huang et Darwiche, 1994 p. 15

Building an Optimal Join Tree

1. Begin with a set of n trees, each consisting of a single clique, and an empty set $\mathcal{S}$.

2. For each distinct pair of cliques $\mathbf{X}$ and $\mathbf{Y}$:[9]

   (a) Create a candidate sepset, labeled $\mathbf{X} \cap \mathbf{Y}$, with backpointers to the cliques $\mathbf{X}$ and $\mathbf{Y}$. Refer to this sepset as $\mathbf{S_{XY}}$.
   (b) Insert $\mathbf{S_{XY}}$ into $\mathcal{S}$.

3. Repeat until $n - 1$ sepsets have been inserted into the forest.

   (a) Select a sepset $\mathbf{S_{XY}}$ from $\mathcal{S}$, according to the criterion specified in Section 4.4.2. Delete $\mathbf{S_{XY}}$ from $\mathcal{S}$.
   (b) Insert the sepset $\mathbf{S_{XY}}$ between the cliques $\mathbf{X}$ and $\mathbf{Y}$ *only if* $\mathbf{X}$ and $\mathbf{Y}$ are on different trees in the forest.[10] (Note that the insertion of such a sepset would merge two trees into a larger tree.)

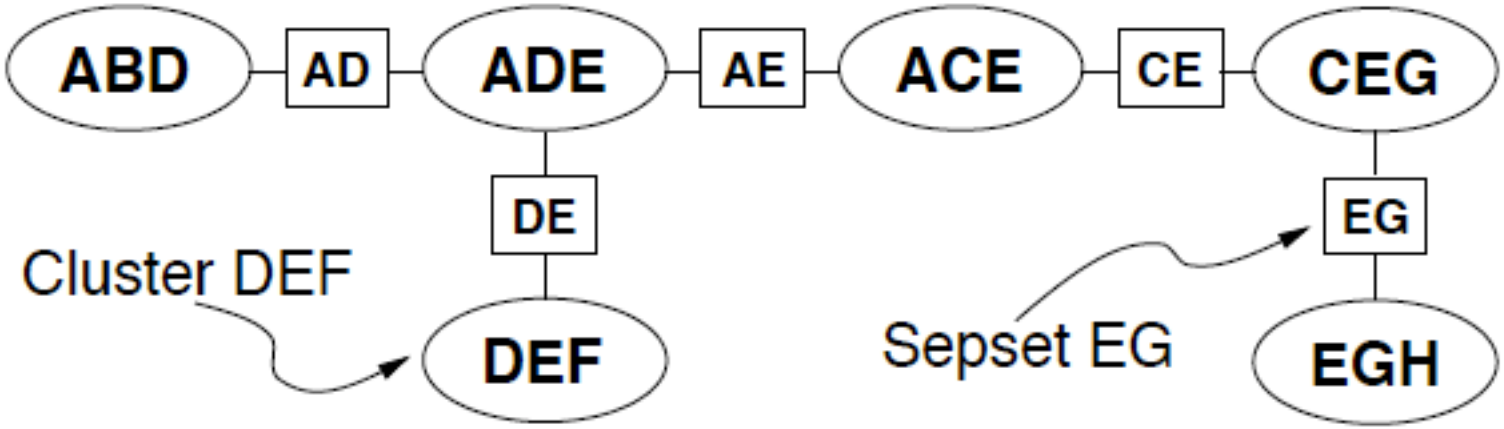
Choisir les sepsets appropriés (i.e. section 4.4.2)

- Rappel : “sepset” = le sous-ensemble de variables commun dans deux cliques ou plus
  - Afin de décrire comment choisir le prochain candidat sepset, nous définissons les notions de masse, et de coût, comme suit :
  - La masse d'un sepset, S_{XY} est le nombre de variables qu'il contient, ou le nombre de variables dans X \ Y.
  - Le coût d'un sepset S_{XY} est le poids de X, plus le poids de Y, où le poids est défini comme suit:
    * Le poids d'une variable V est le nombre de valeurs de V.
    * Le poids d'un ensemble de variables X est le produit des poids des variables dans X.

Choisir les sepsets appropriés (i.e. section 4.4.2)

- Avec ces notions établies, nous pouvons maintenant choisir le prochain candidat sepset de S :
- Pour l'arbre de cliques résultant de satisfaire la propriété d'un arbre de jonction, nous devons choisir le candidat avec le sepset de la plus grande masse.
- Lorsqu'il existe deux ou plusieurs sepsets de masse égale, nous pouvons optimiser le temps de l'inférence sur l'arbre résultant comme suit : choisir le candidat sepset au moindre coût.
- La base de cette méthode de construction d'un arbre de jonction optimal peut être trouvée dans un article de recherche de Jensen, F. V., and Jensen, F., "Optimal junction trees", UAI 1994.

Avec notre exemple nous avons



ϕ_{ABD} =

a	b	d	ϕ_{ABD}(abd)
on	on	on	.225
on	on	off	.025
on	off	on	.125
on	off	off	.125
off	on	on	.180
off	on	off	.020
off	off	on	.150
off	off	off	.150

ϕ_{AD} =

a	d	ϕ_{AD}(ad)
on	on	.35
on	off	.15
off	on	.33
off	off	.17

etc.

Data Mining

Practical Machine Learning Tools and Techniques

Slides for Chapter 9, Probabilistic methods

of *Data Mining* by I. H. Witten, E. Frank,
M. A. Hall, and C. J. Pal

Reminder -- Estimating Bayesian network parameters

- The log-likelihood of a Bayesian network with *V* variables and *N* examples of complete variable assignments to the network is

$$\sum_{i=1}^{N} \log P(\{\tilde{A}_1, \tilde{A}_2, ..., \tilde{A}_V\}_i) = \sum_{i-1}^{N} \sum_{v=1}^{V} \log P(\tilde{A}_{v,i} \left| \text{Parents}(\tilde{A}_{v,i}); \Theta_v \right.)$$

  where the parameters of each conditional or unconditional distribution are given by Θ_v

- We use the $\tilde{A}_{v,i}$ notation to indicate the *i*th observation of variable *v*

Estimating probabilities in Bayesian networks

- The estimation problem *decouples* into separate estimation problems for each conditional or unconditional probability
- Unconditional probabilities can be written as

$$P(A = a) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}(\tilde{A}_i = a)$$

where $\mathbf{1}(\tilde{A}_i = a)$ is an indicator function returning 1 when the i^{th} observed value for A_i=a and 0 otherwise

Estimating conditional distributions

- Estimating conditional distributions in Bayesian networks is equally easy and amounts to simply counting configurations and dividing, ex.

$$P(B = b \mid A = a) = \frac{P(B = b, A = a)}{P(A = a)} = \frac{\sum_{i=1}^{N} \mathbf{1}(\tilde{A}_i = a, \tilde{B}_i = b)}{\sum_{i=1}^{N} \mathbf{1}(\tilde{A}_i = a)}.$$

- Zero counts cause problems and this motivates the use of Bayesian priors

Estimating network structure

- One possibility is to use cross-validation to estimate the goodness of fit on held out data (so as to avoid over fitting) another is to penalize model complexity
- Let *K* be the number of parameters, *LL* the log-likelihood, and *N* the number of instances in the data.
- Two popular measures for evaluating the quality of a network are the *Akaike Information Criterion* (AIC):

$$\text{AIC score} = -LL + K$$

and the following *MDL metric* based on the MDL principle:

$$\text{MDL score} = -LL + \frac{K}{2}\log N$$

- In both cases the log-likelihood is negated, so the aim is to minimize these scores.
- More Bayesian approach: use prior over model structures

Apprentissage (simple)

dans un réseau bayésien

(Quand chaque variable est observée)

Comment construire un réseau bayésien

Situations typiques :

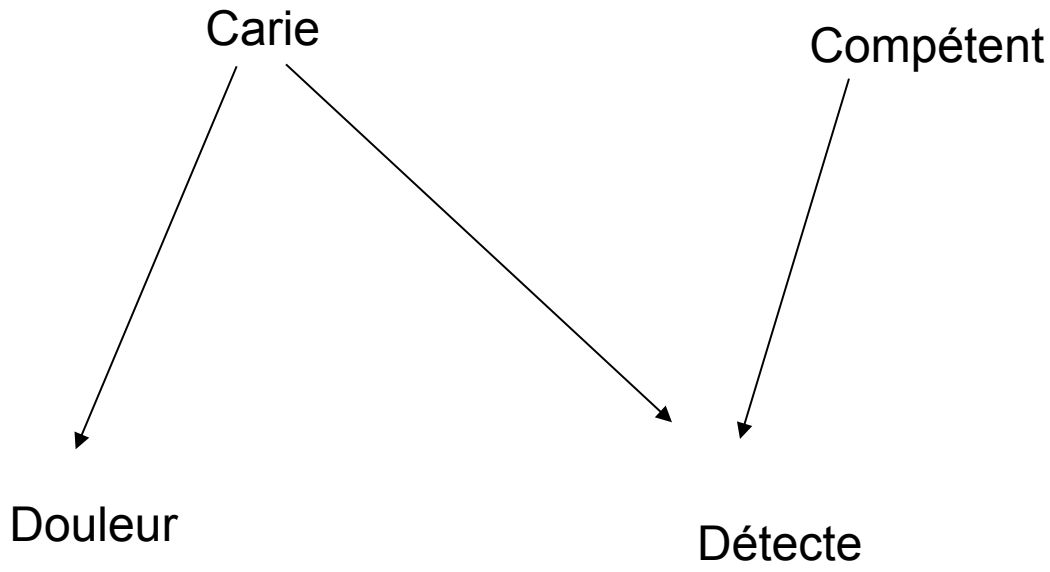
- Par les relations de causes à effet, nous savons construire le réseau: il ne manque alors que les tables de probabilité conditionnelle
  - Cas 1: Nous avons des tables de donnée complètes
  - Cas 2: Certaines valeurs sont manquantes ou cachées
- Nous ne savons pas quelle topologie de réseau est la meilleure, selon les données dont nous disposons

Premier cas: le réseau est connu

- Si nous sommes chanceux, nous avons un très grand jeu de données
- Ce jeu de données consiste en une table qui fournit les fréquences calculées pour chaque combinaison de valeurs
- Dans ce cas, nous avons tout ce qu'il nous faut pour établir les tables de probabilité conditionnelle

Premier cas: le réseau est connu (suite)

- Soit par exemple le réseau suivant:



Premier cas: le réseau est connu (suite)

Carie	Compétent	Douleur	Détecte	# occurrences
0	0	0	0	3
0	0	1	0	45
0	1	0	0	1
0	1	1	0	16
1	0	1	0	3
1	1	0	1	1
1	1	1	0	1
1	0	1	1	4
1	1	1	1	26
			TOTAL	100

Premier cas: le réseau est connu (suite)

Carie	Compétent	Douleur	Détecte	# occurrences
0	0	0	0	3
0	0	1	0	45
0	1	0	0	1
0	1	1	0	16
1	0	1	0	3
1	1	0	1	1
1	1	1	0	1
1	0	1	1	4
1	1	1	1	26
			TOTAL	100

P(Carie) = (3 + 1 + 1 + 4 + 26) / 100 = 35/100 = 0,35

Premier cas: le réseau est connu (suite)

Carie	Compétent	Douleur	Détecte	# occurrences
0	0	0	0	3
0	0	1	0	45
0	1	0	0	1
0	1	1	0	16
1	0	1	0	3
1	1	0	1	1
1	1	1	0	1
1	0	1	1	4
1	1	1	1	26
			TOTAL	100

P(Douleur|Carie) = (3 + 1 + 4 + 26) / (3 + 1 + 1 + 4 + 26) = 34/35= 0,97

Premier cas: le réseau est connu (suite)

Carie	Compétent	Douleur	Détecte	# occurrences
0	0	0	0	3
0	0	1	0	45
0	1	0	0	1
0	1	1	0	16
1	0	1	0	3
1	1	0	1	1
1	1	1	0	1
1	0	1	1	4
1	1	1	1	26
			TOTAL	100

P(Detecte|Carie, Pas Compétent) = 4 / (3 + 4) = 4/7 = 0,57

Apprentissage dans un réseau bayésien

(Quand il y a des variables inconnue)

L'algorithme EM

Espérance-maximisation

Quand le réseau est connu, mais il y a des variables inconnue

- On utilise les tables de probabilité et le réseau bayesien pour calculer les valeurs de probabilité de la variable inconnue
- Dans le cas de la valeur inconnue pour douleur, supposons que la probabilité est de 0,6
- On sépare alors l'entrée en deux entrées dont les poids sont modulés par la probabilité que nous venons de calculer

Espérance-maximisation (EM)

- Le problème est que pour faire cela, il faut utiliser les tables de probabilité conditionnelle que nous sommes justement en train de calculer!
- Solution: une processus itératif
- L'algorithme le plus utilisé dans ce cas est l'algorithme EM

Espérance-maximisation (EM)

- D'abord on fixe des valeurs arbitraires pour les tables de probabilité conditionnelles
- **Phase E :**
  - On calcule les probabilités des variables inconnues
  - On ajoute à la table les lignes supplémentaires requise en fonction des probabilités obtenues pour le valeurs inconnues
- **Phase M:**
  - On calcule les tables de probabilités du réseau, et on répète le processus jusqu'à convergence

EM – un peu plus formellement

- But: ajuster θ pour maximiser la probabilité des données $\tilde{x}_i$

$$P(\tilde{X};\theta) = \prod_{i=1}^{N} P(\tilde{x}_i;\theta) = \prod_{i=1}^{N} \int_{z_i} \sum_{h_i} p(\tilde{x}_i, z_i, h_i;\theta) dz_i$$

avec une marginalisation sur les variable cachées, h_i – discrète, et z_i – continue

- Cependant, il est possible de montrer que

$$\frac{\partial}{\partial\theta} \log p(\tilde{x}_i;\theta) = \sum_{i=1}^{n} \mathrm{E}\left[\frac{\partial}{\partial\theta} \log p(\tilde{x}_i, z_i, h_i;\theta)\right]_{p(z_i, h_i \mid \tilde{x}_i;\theta)},$$

ou, la dérivée du log de la probabilité marginale est égale à la dérivée du log de la probabilité jointe espérée en fonction de la distribution a posteriori de h_i et z_i

Apprentissage dans un réseau bayésien

(Quand il y a des variables inconnue)

L'algorithme EM

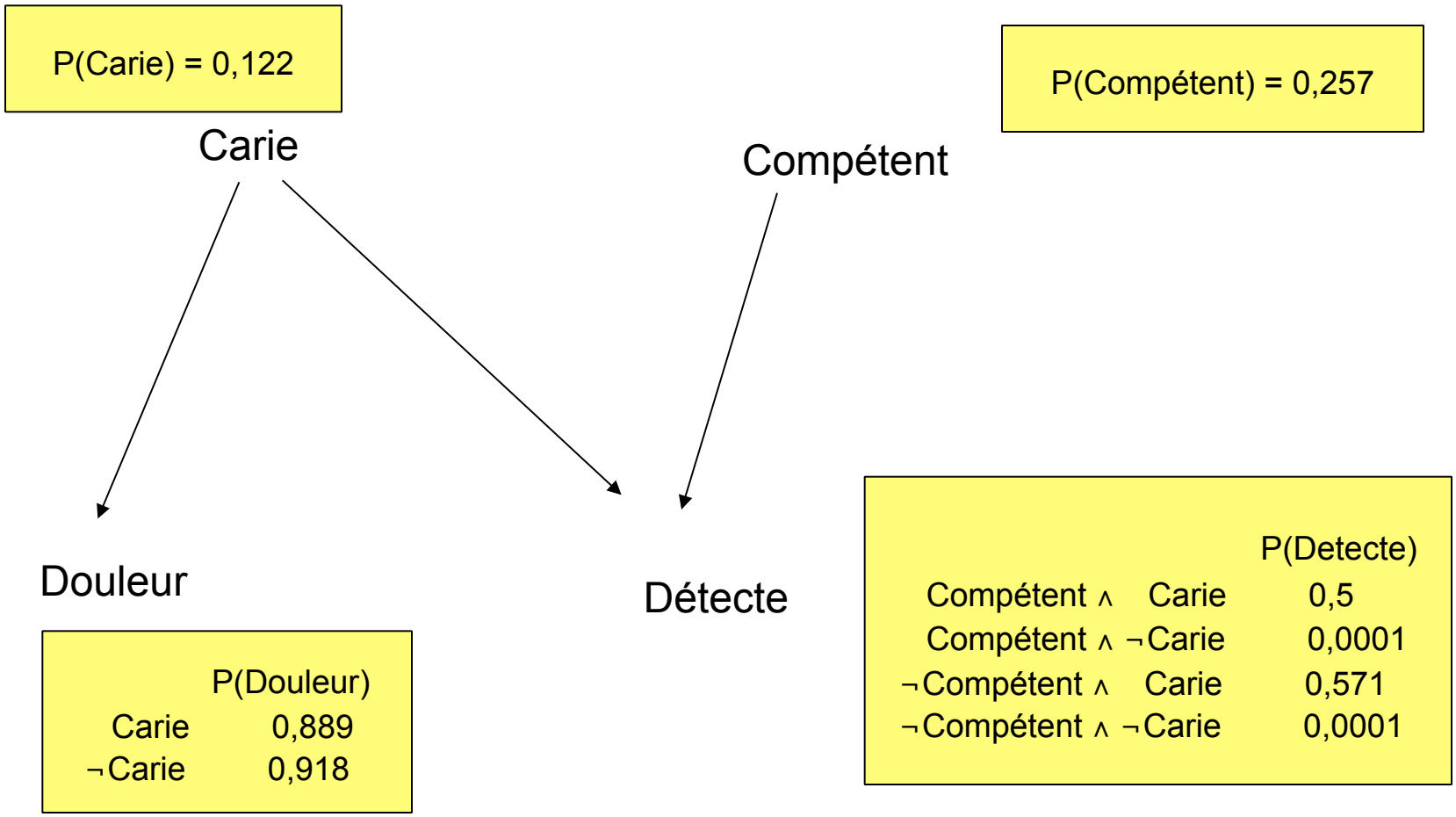
Espérance-maximisation

Algorithme EM - exemple

- Que fait-on maintenant si certaines valeurs sont inconnues:

Carie	Compétent	Douleur	Détecte	#
0	0	0	0	3
0	0	1	0	45
0	1	*	0	**17**
1	0	1	0	3
1	1	0	1	1
1	1	1	0	1
*	*	1	1	**30**
			Total	100

Valeurs initiales (fixées arbitrairement)



Algorithme EM - exemple

- Première itération:

P(Douleur = 1 | Carie = 0, Compétent = 1, Détecte = 0) = 0,918

P(Carie = 0, Compétent = 0 | Douleur = 1, Détecte = 1) = 0,001017
P(Carie = 0, Compétent = 1 || Douleur = 1, Détecte = 1) = 0,000351
P(Carie = 1, Compétent = 0 | Douleur = 1, Détecte = 1) = 0,7665
P(Carie = 1, Compétent = 1 | Douleur = 1, Détecte = 1) = 0,2322

Carie	Compétent	Douleur	Détecte	#	
0	0	0	0	3	
0	0	1	0	45	
0	1	0	0	1,394	(= 0,072*__17__)
0	1	1	0	15,606	(= 0,918*__17__)
1	0	1	0	3	
1	1	0	1	1	
1	1	1	0	1	
0	0	1	1	0,02993	(= 0,001017*__30__)
0	1	1	1	0,0104	(= 0,000351*__30__)
1	0	1	1	22,9949	(= 0,7665*__30__)
1	1	1	1	6,9648	(= 0,2322*__30__)
			Total	100	

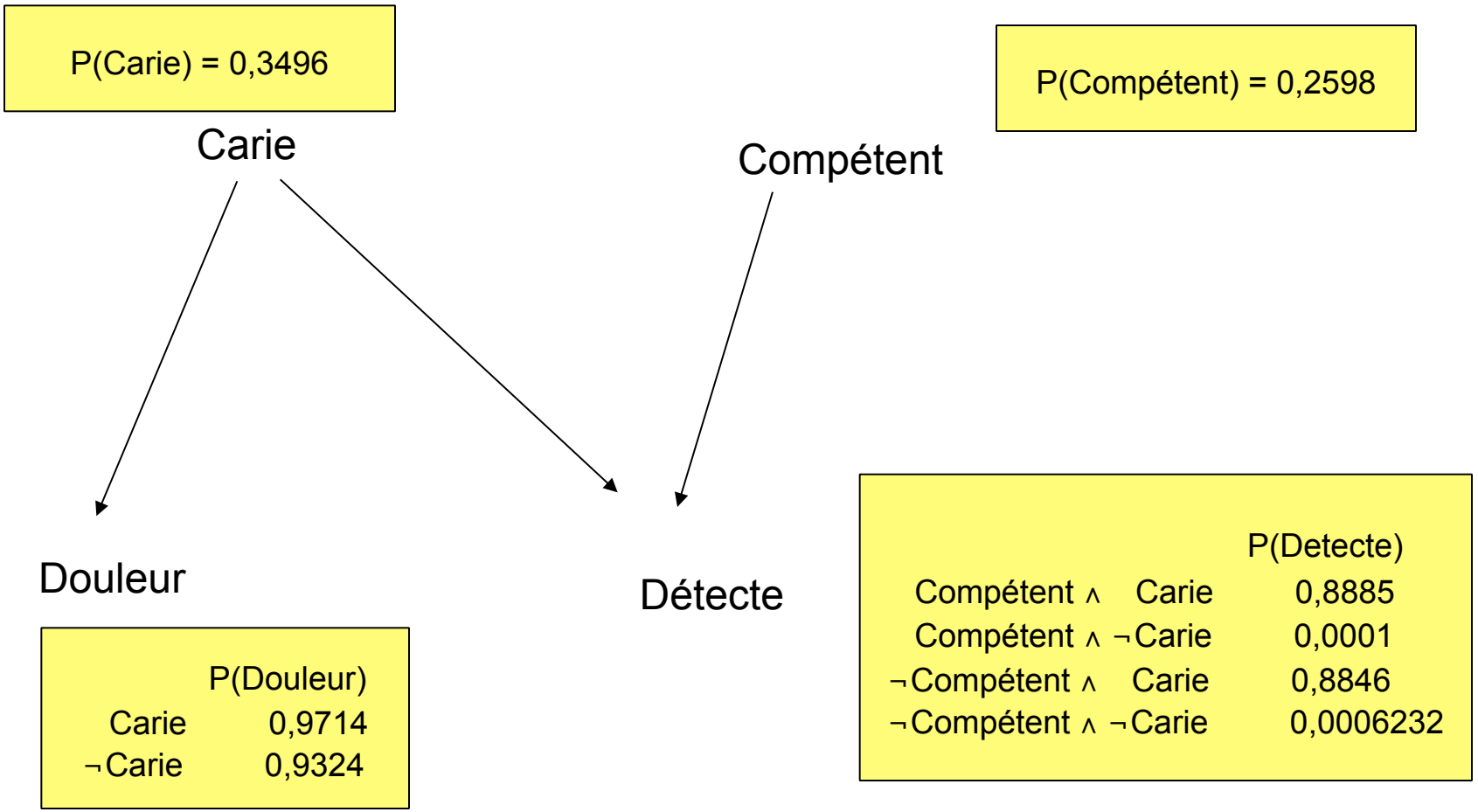
Algorithme EM - exemple

Carie	Compétent	Douleur	Détecte	N	P	N*ln(P)
0	0	0	0	3	0,0535	-8,78
0	0	1	0	45	0,5988	-23,08
0	1	*	0	17	0,2256	-25,31
1	0	1	0	3	0,0346	-10,09
1	1	0	1	1	0,0017	-6,35
1	1	1	0	1	0,0139	-4,27
*	*	1	1	30	0,0600	-84,39

Log vraisemblance = -162,28

-162,28

Algorithme EM - exemple



Algorithme EM - exemple

P(Douleur = 1 | Carie = 0, Compétent = 1, Détecte = 0) = 0,9324

P(Carie = 0, Compétent = 0 | Douleur = 1, Détecte = 1) = 0,0009293
P(Carie = 0, Compétent = 1 || Douleur = 1, Détecte = 1) = 0,0000523
P(Carie = 1, Compétent = 0 | Douleur = 1, Détecte = 1) = 0,7387
P(Carie = 1, Compétent = 1 | Douleur = 1, Détecte = 1) = 0,2603

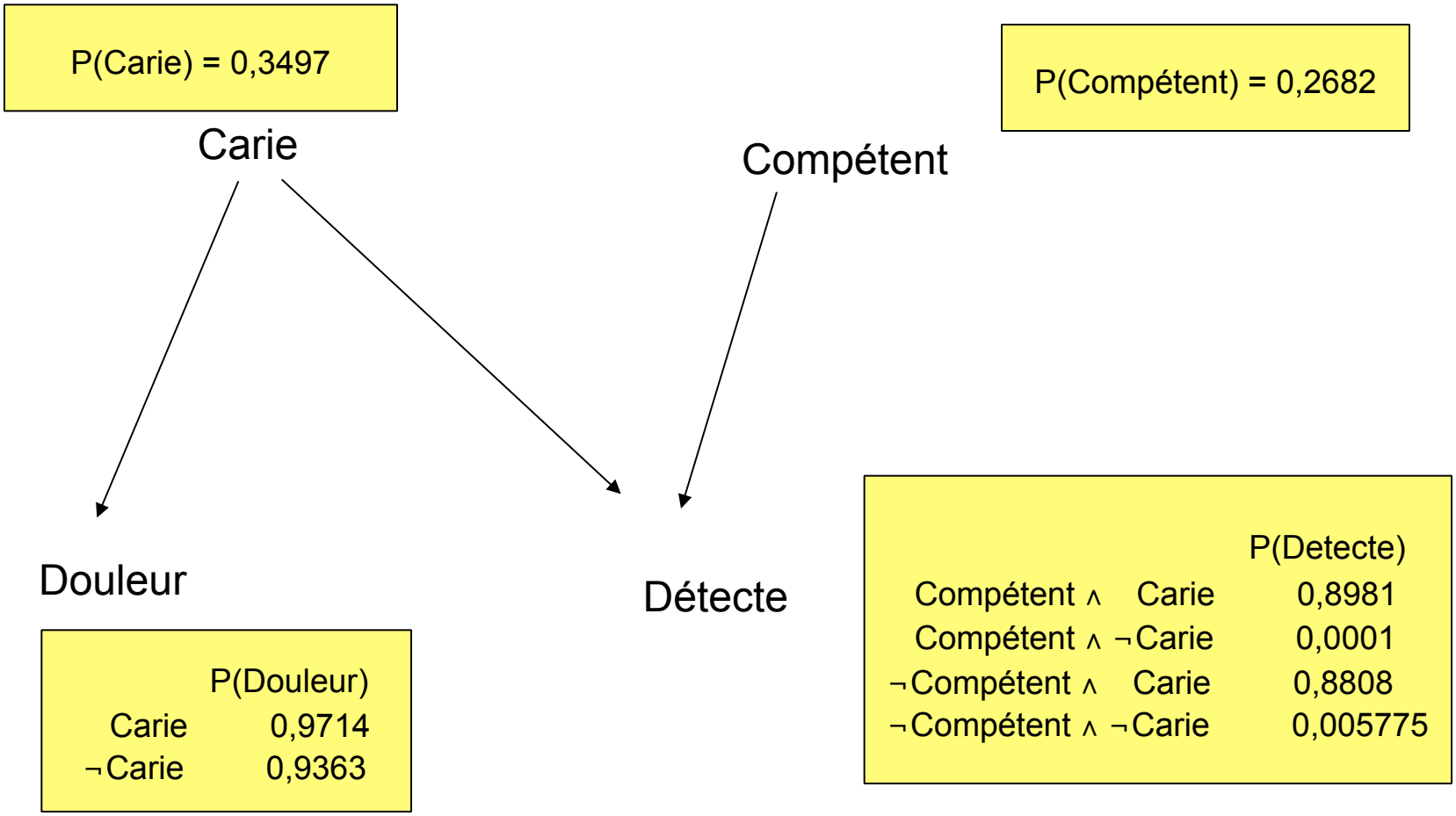
Carie	Compétent	Douleur	Détecte	#
0	0	0	0	3
0	0	1	0	45
0	1	0	0	1,158
0	1	1	0	15,852
1	0	1	0	3
1	1	0	1	1
1	1	1	0	1
0	0	1	1	0,2788
0	1	1	1	0,00157
1	0	1	1	22,16
1	1	1	1	7,8103
			Total	100

Algorithme EM - exemple

Carie	Compétent	Douleur	Détecte	N	P	N*ln(P)
0	0	0	0	3	0,0325	-10,28
0	0	1	0	45	0,4486	-36,07
0	1	*	0	17	0,1690	-30,23
1	0	1	0	3	0,0290	-10,62
1	1	0	1	1	0,0023	-6,07
1	1	1	0	1	0,0098	-4,62
*	*	1	1	30	0,3011	-36,01
						-133,91

Log vraisemblance = -133,91

Algorithme EM - exemple



Algorithme EM - exemple

P(Douleur = 1 | Carie = 0, Compétent = 1, Détecte = 0) = 0,9324
P(Carie = 0, Compétent = 0 | Douleur = 1, Détecte = 1) = 0,0008265
P(Carie = 0, Compétent = 1 || Douleur = 1, Détecte = 1) = 0,0000525
P(Carie = 1, Compétent = 0 | Douleur = 1, Détecte = 1) = 0,7062
P(Carie = 1, Compétent = 1 | Douleur = 1, Détecte = 1) = 0,2855

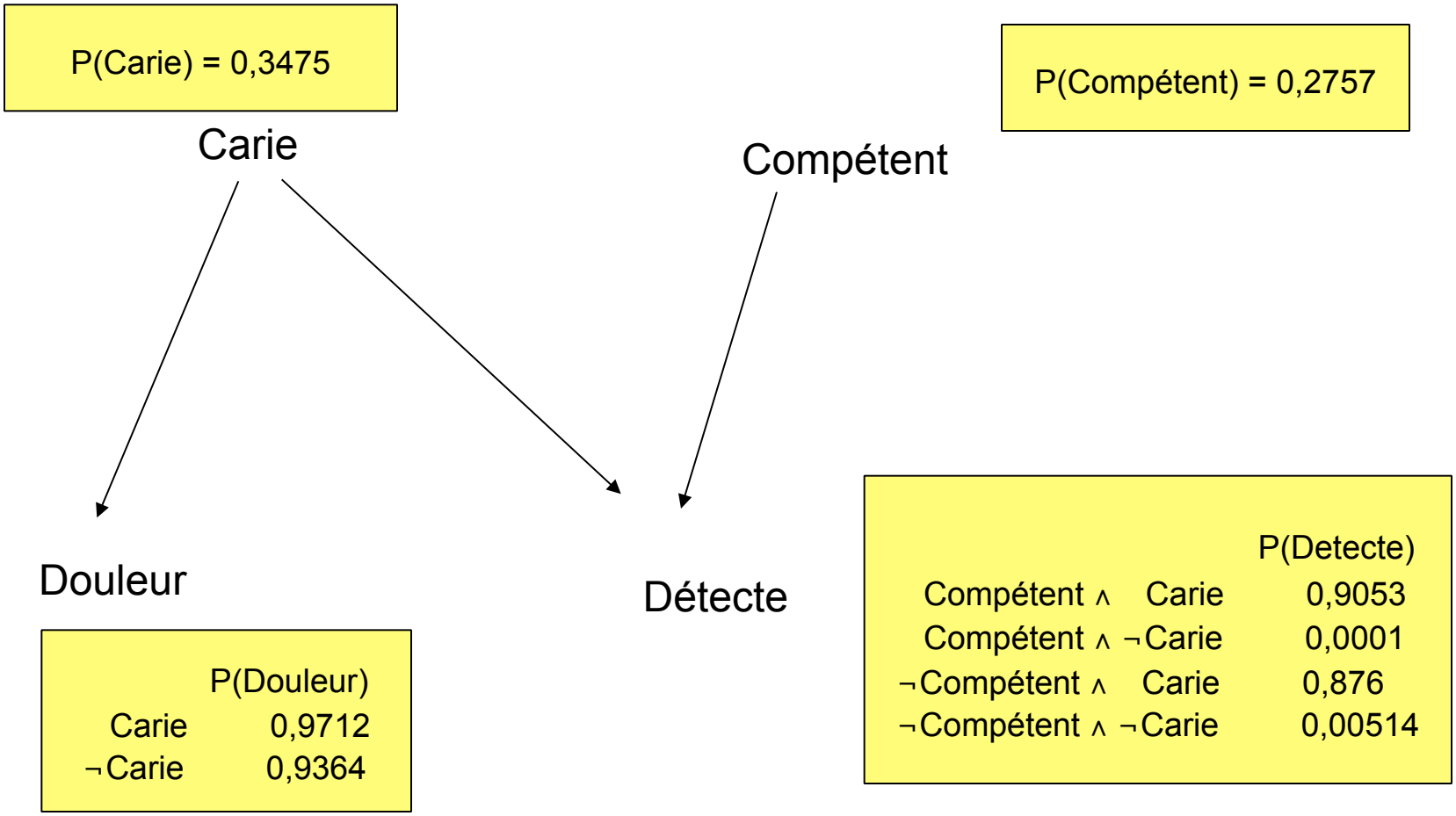
Carie	Compétent	Douleur	Détecte	#
0	0	0	0	3
0	0	1	0	45
0	1	0	0	1,15
0	1	1	0	15,85
1	0	1	0	3
1	1	0	1	1
1	1	1	0	1
0	0	1	1	0,0269
0	1	1	1	0,00168
1	0	1	1	21,83
1	1	1	1	8,15
			Total	100

Algorithme EM - exemple

Carie	Compétent	Douleur	Détecte	N	P	N*ln(P)
0	0	0	0	3	0,0303	-10,49
0	0	1	0	45	0,4453	-36,40
0	1	*	0	17	0,1744	-29,69
1	0	1	0	3	0,0296	-10,56
1	1	0	1	1	0,0024	-6,03
1	1	1	0	1	0,0093	-4,68
*	*	1	1	30	0,3011	-36,01
						-133,86

Log vraisemblance = -133,86

Algorithme EM - exemple



Algorithme EM - exemple

P(Douleur = 1 | Carie = 0, Compétent = 1, Détecte = 0) = 0,9364

P(Carie = 0, Compétent = 0 | Douleur = 1, Détecte = 1) = 0,007566
P(Carie = 0, Compétent = 1 || Douleur = 1, Détecte = 1) = 0,00005602
P(Carie = 1, Compétent = 0 | Douleur = 1, Détecte = 1) = 0,7122
P(Carie = 1, Compétent = 1 | Douleur = 1, Détecte = 1) = 0,2801

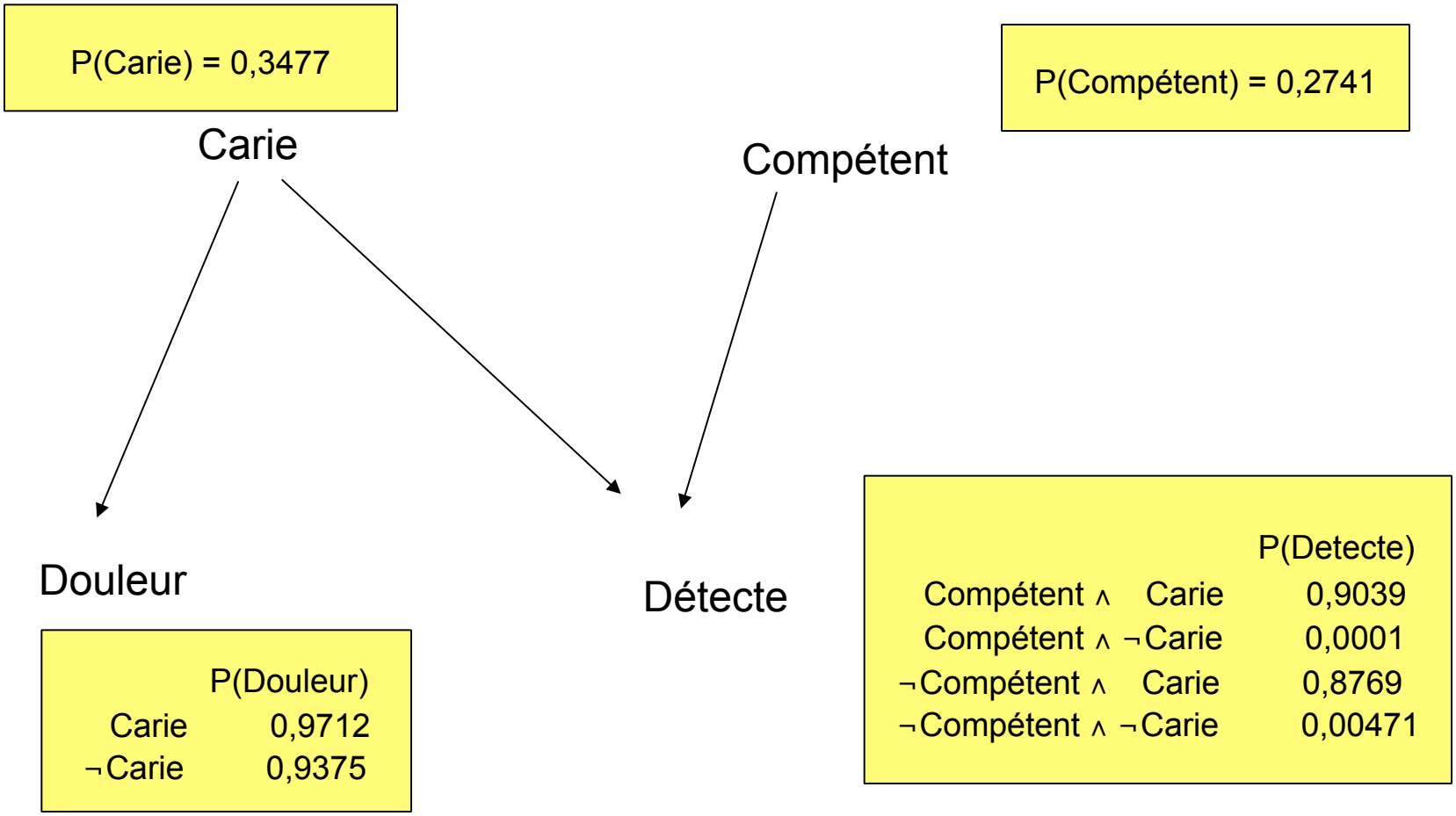
Carie	Compétent	Douleur	Détecte	#
0	0	0	0	3
0	0	1	0	45
0	1	0	0	1,08
0	1	1	0	15,92
1	0	1	0	3
1	1	0	1	1
1	1	1	0	1
0	0	1	1	0,227
0	1	1	1	0,00168
1	0	1	1	21,37
1	1	1	1	8,404
			Total	100

Algorithme EM - exemple

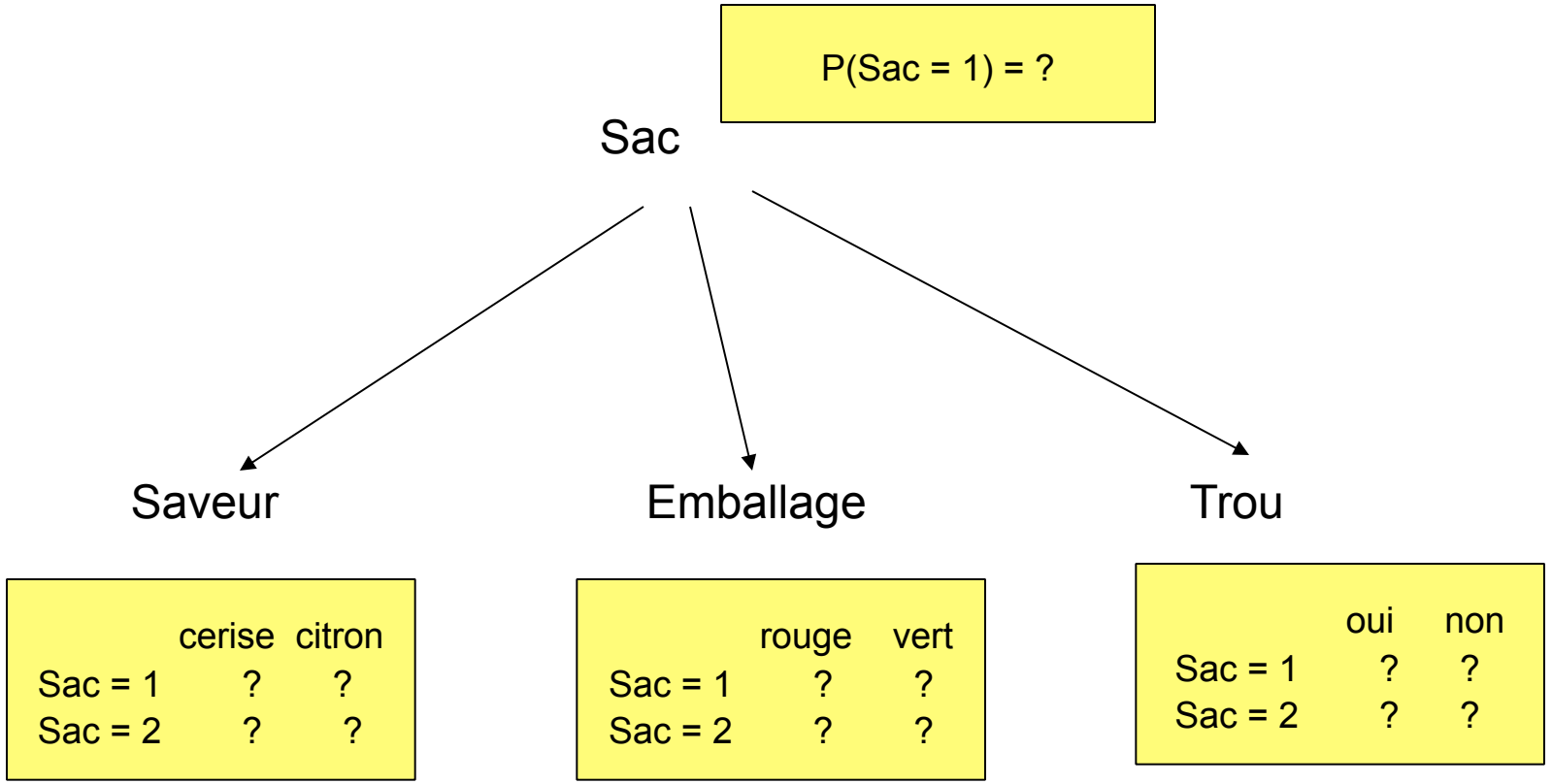
Carie	Compétent	Douleur	Détecte	N	P	N*ln(P)
0	0	0	0	3	0,0299	-10,53
0	0	1	0	45	0,4403	-36,92
0	1	*	0	17	0,1799	-29,16
1	0	1	0	3	0,0303	-10,49
1	1	0	1	1	0,0025	-5,99
1	1	1	0	1	0,0088	-4,73
*	*	1	1	30	0,3007	-36,05
						-133,87

Log vraisemblance = -133,87

Algorithme EM - exemple



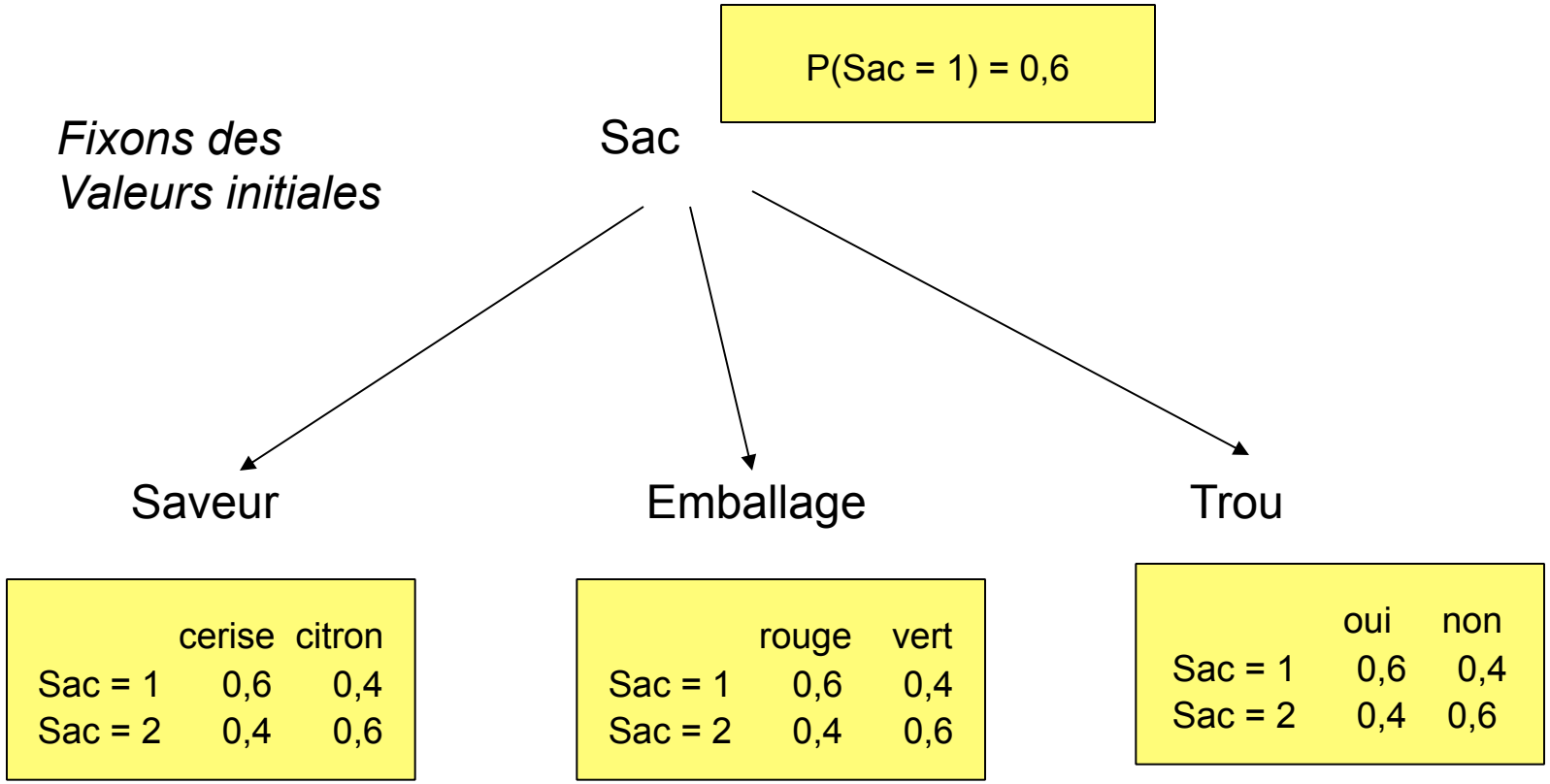
Algorithme EM – exemple avec "Naïve Bayes" et la "class" cachée (sac)



Algorithme EM – exemple des données

Sac	Saveur	Emballage	Trou	N
*	cerise	rouge	oui	273
*	cerise	rouge	non	93
*	cerise	vert	oui	104
*	cerise	vert	non	90
*	citron	rouge	oui	79
*	citron	rouge	non	100
*	citron	vert	oui	94
*	citron	vert	non	167

Algorithme EM – les paramètres préliminaire



Algorithme EM – Phase E, calculer les postérieurs

Probabilités selon le réseau bayesien avec les probabilités initiales

P(sac=1 | cerise, rouge, oui)	0,8351
P(sac=1 | cerise, rouge, non)	0,6923
P(sac=1 | cerise, vert, oui)	0,6923
P(sac=1 | cerise, vert, non)	0,5000
P(sac=1 | citron, rouge, oui)	0,6923
P(sac=1 | citron, rouge, non)	0,5000
P(sac=1 | citron, vert, oui)	0,5000
P(sac=1 | citron, vert, non)	0,3077

Algorithme EM – Phase E, calculer les statistiques équivalentes

Sac	Saveur	Emballage	Trou	N	
1	cerise	rouge	oui	227,9823	(= 0,8351 * 273)
1	cerise	rouge	non	64,3839	
1	cerise	vert	oui	71,9992	
1	cerise	vert	non	45	
1	citron	rouge	oui	54,6917	
1	citron	rouge	non	50	
1	citron	vert	oui	47	
1	citron	vert	non	51,3859	
2	cerise	rouge	oui	45,0177	(= 0,1649 * 273)
2	cerise	rouge	non	28,6161	
2	cerise	vert	oui	32,0008	
2	cerise	vert	non	45	
2	citron	rouge	oui	24,3083	
2	citron	rouge	non	50	
2	citron	vert	oui	47	
2	citron	vert	non	115,6141	
				1000	

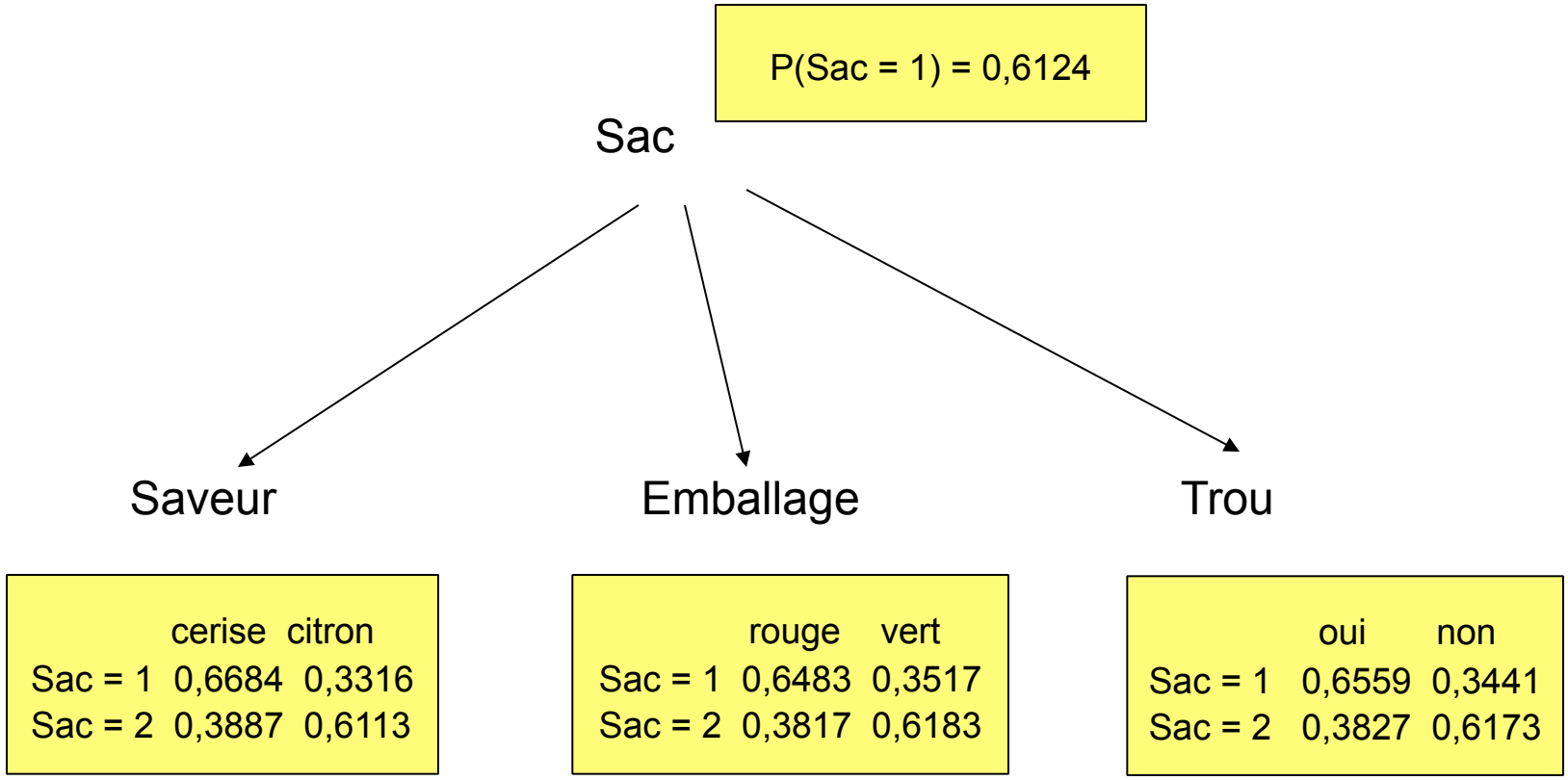
Algorithme EM – vérification, calculer la log vraisemblance

Saveur	Emballage	Trou	P(S,E,T)	N*Ln(P)	
cerise	rouge	oui	0,1552	-508,61	=273 * ln(0,1552)
cerise	rouge	non	0,1248	-193,54	
cerise	vert	oui	0,1248	-216,43	
cerise	vert	non	0,1152	-194,50	
citron	rouge	oui	0,1248	-164,40	
citron	rouge	non	0,1152	-216,11	
citron	vert	oui	0,1152	-203,14	
citron	vert	non	0,1248	-347,53	
			1	-2044,26	

Log vraisemblance = -2044,26

Algorithme EM – Phase M

Mettre à jour les probabilités inconditionnelles et conditionnelles en utilisant les statistiques équivalentes



Algorithme EM – Phase E

P(sac=1 | cerise, rouge, oui)	0,8878
P(sac=1 | cerise, rouge, non)	0,7201
P(sac=1 | cerise, vert, oui)	0,7259
P(sac=1 | cerise, vert, non)	0,4628
P(sac=1 | citron, rouge, oui)	0,7139
P(sac=1 | citron, rouge, non)	0,448
P(sac=1 | citron, vert, oui)	0,4552
P(sac=1 | citron, vert, non)	0,2137

Algorithme EM – Phase E

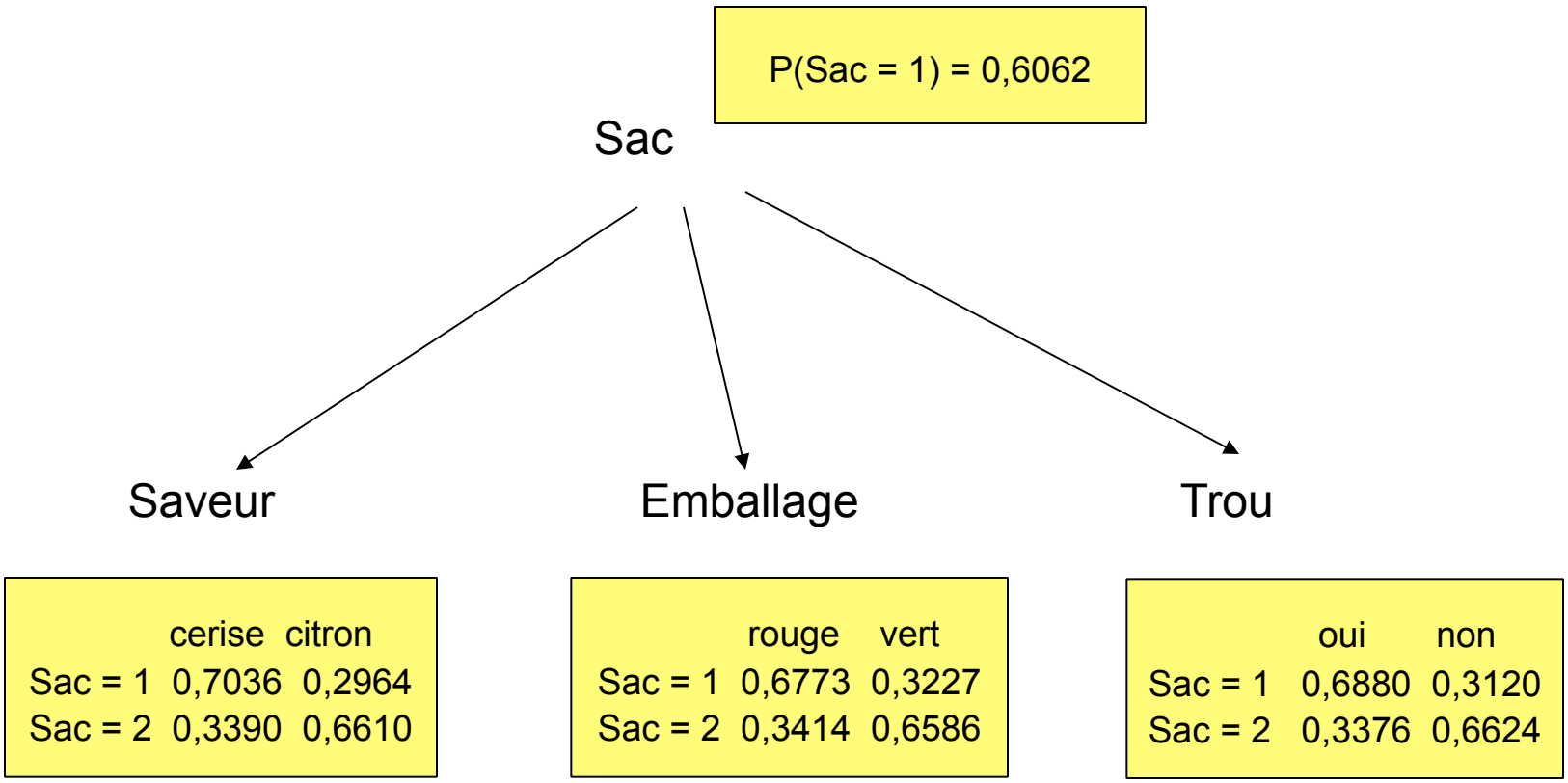
Sac	Saveur	Emballage	Trou	N	
1	cerise	rouge	oui	242,37	(= 0,8878 * 273)
1	cerise	rouge	non	66,97	
1	cerise	vert	oui	75,49	
1	cerise	vert	non	41,65	
1	citron	rouge	oui	56,40	
1	citron	rouge	non	44,80	
1	citron	vert	oui	42,79	
1	citron	vert	non	35,69	
2	cerise	rouge	oui	30,63	(= 0,1122 * 273)
2	cerise	rouge	non	26,03	
2	cerise	vert	oui	28,51	
2	cerise	vert	non	48,35	
2	citron	rouge	oui	22,60	
2	citron	rouge	non	55,20	
2	citron	vert	oui	51,21	
2	citron	vert	non	131,31	
				1000	

Algorithme EM – exemple avec variable cachée

Saveur	Emballage	Trou	P		N*Ln(p)
cerise	rouge	oui	0,1961		-444,75
cerise	rouge	non	0,1268		-192,06
cerise	vert	oui	0,1301		-212,10
cerise	vert	non	0,107		-201,14
citron	rouge	oui	0,121		-166,85
citron	rouge	non	0,1011		-229,16
citron	vert	oui	0,1029		-213,76
citron	vert	non	0,115		-361,19
				1	-2021,01

Log vraisemblance = -2021

Algorithme EM – Phase M



INF8225

Modèles probabilistes séquentiels et apprentissage automatique

Leçon 4, partie 2

Christopher Pal

École Polytechnique de Montréal

Plan du cours

- Modèles temporels et séquentiels.
- Modèles de Markov cachés et réseaux bayésiens dynamiques.
- Champs aléatoires de Markov et
- Généralisations de ces modèles avec les modèles probabilistes graphiques
- Modèles génératifs vs modèles discriminatifs
- Réseaux récurrents
- MRFs et machines de Boltzmann

Markov models

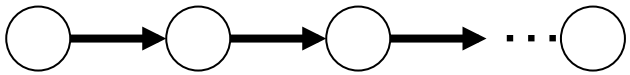
- One simple and effective probabilistic model for discrete sequential data is known as a *Markov model*
- A first-order Markov model assumes that each symbol in a sequence can be predicted using its conditional probability given the preceding symbol.
- An unconditional probability is used for the first symbol
- Given observation variables O={O_1,…,O_T}, we have

$$P(\mathrm{O}) = P(O_1)\prod_{t=1}^{T} P(O_{t+1} \mid O_t).$$

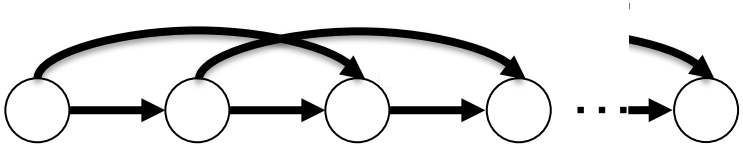
- Usually, every conditional probability used in such models is the same
- The Bayesian network is a linear chain of variables with directed edges between each successive pair

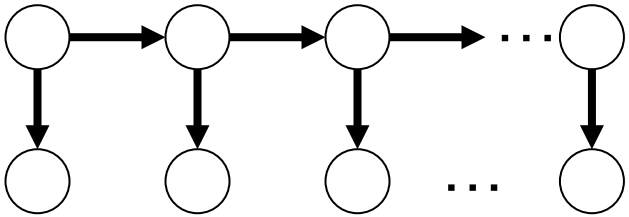
Extending Markov models

First order Markov model

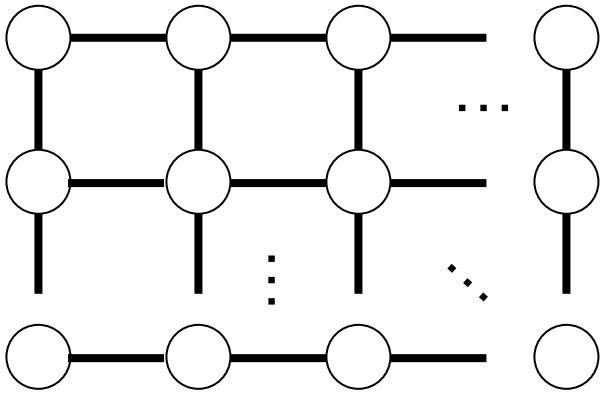


Second order Markov model





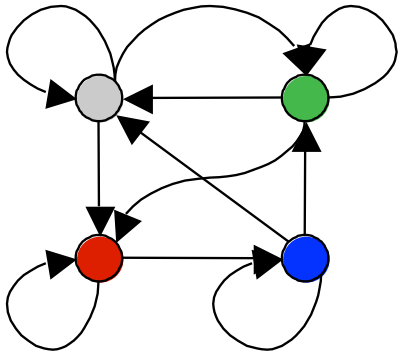
Hidden Markov model

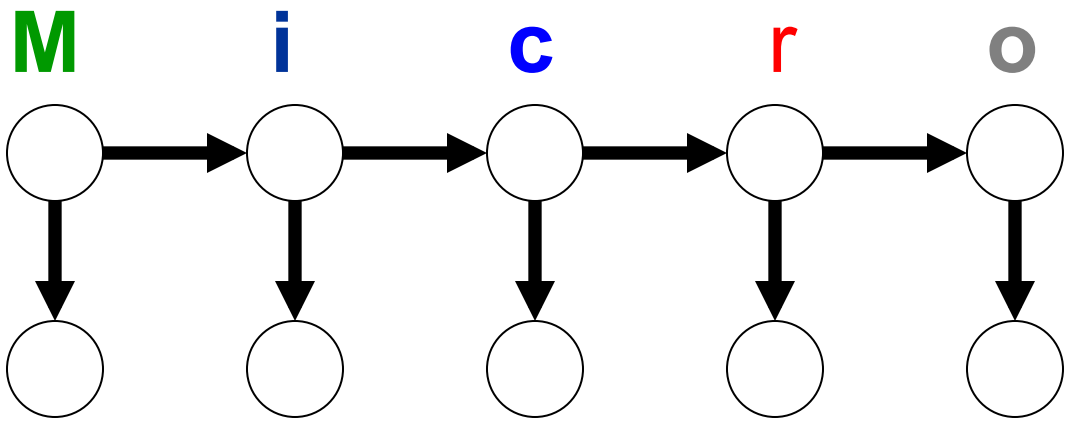


Markov random field

Modèles probabilistes séquentiels

- Les modèles de Markov cachés (Hidden Markov Models HMMs)



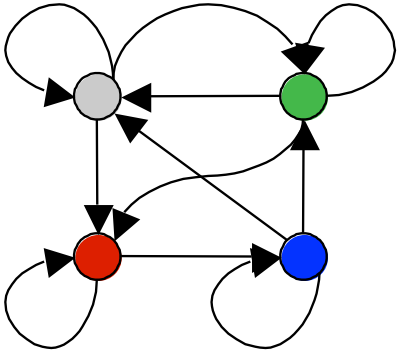


Today in Redmond, **Microsoft** Chair Bill Gates…

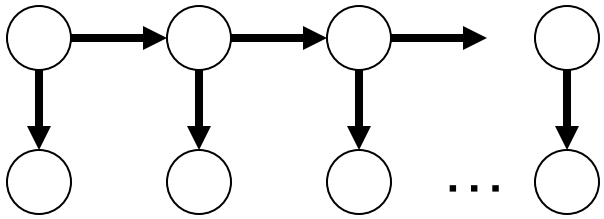
- La modélisation des séquences et des données temporels : ex. texte, parole, la reconnaissance des mot et l'extraction d'information

Automates stochastiques, réseaux bayésiens dynamiques, champs aléatoires de Markov et champs aléatoires conditionnels

(1) Automates finis stochastiques

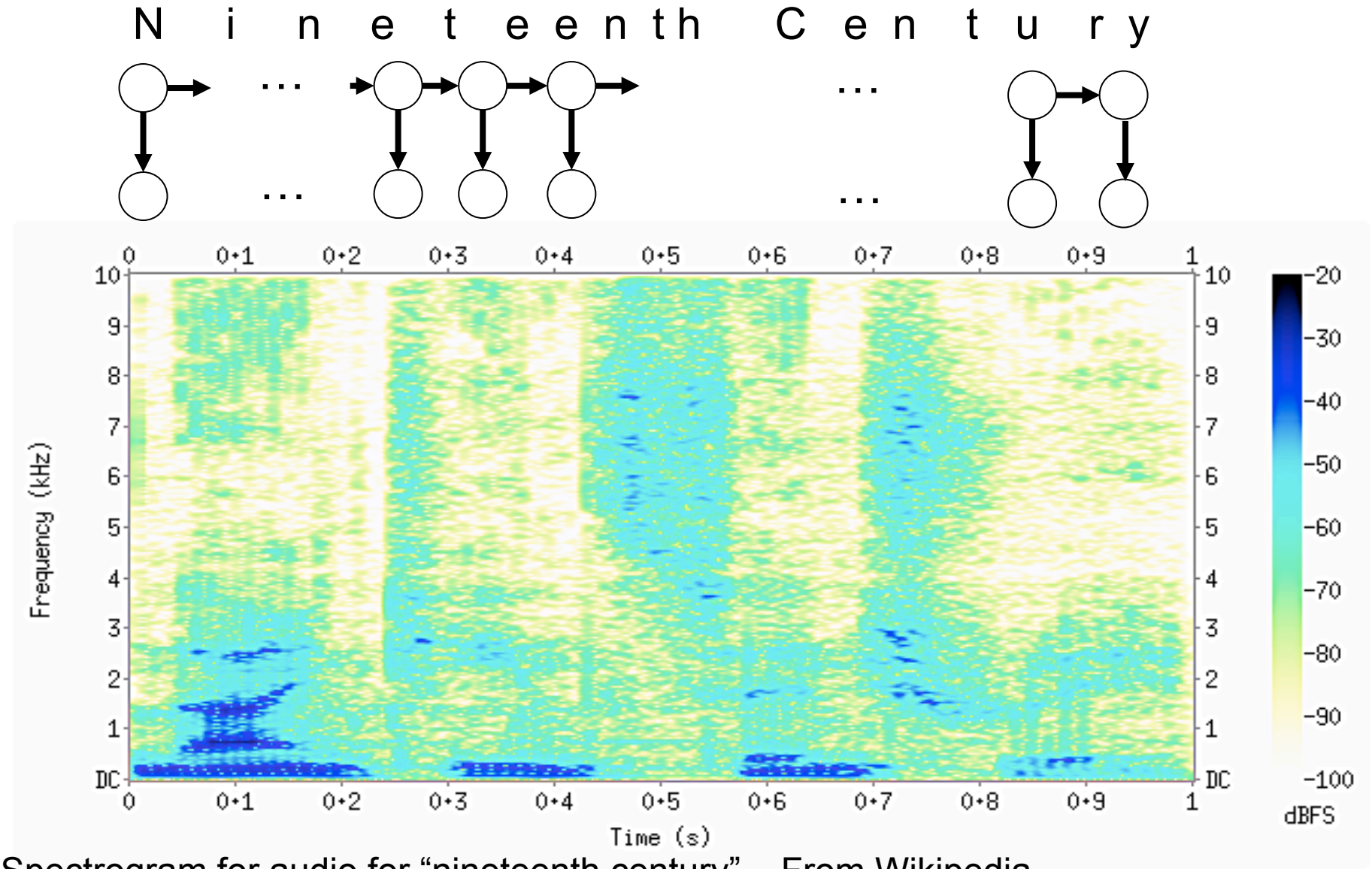


(2) Réseaux bayésiens dynamiques (DBN)



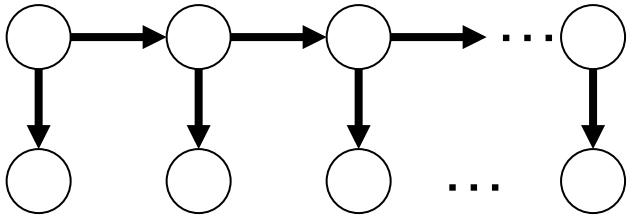
- Confrontons alors ces *modèles génératifs* aux *modèles conjoints* (a) et *discriminatifs* (b)

Example: HMMs for speech recognition



Spectrogram for audio for “nineteenth century” – From Wikipedia

Hidden Markov models



- A hidden Markov model is a joint probability model of a set of discrete observed variables O={O_1,…,O_T} and discrete hidden variables H={H_1,…,H_T} for T observations that factors the joint distribution as follows

$$P(\mathrm{O},\mathrm{H}) = P(H_1)\prod_{t=1}^{T} P(H_{t+1} \mid H_t)\prod_{t=1}^{T} P(O_t \mid H_t),$$

- Each O_t is a discrete random variable with N possible values, and each H_t is a discrete random variable with M possible values.
- The previous figure illustrates a hidden Markov model as a type of Bayesian network that is known as a "dynamic" Bayesian network where variables are replicated dynamically over the appropriate number of time steps.

Hidden Markov models (HMMs)

- Common to use "time-homogeneous" models where the transition matrix $P(H_{t+1}|H_t)$ is the same at each time step
- Define **A** to be a transition matrix whose elements encode $P(H_{t+1}= j|H_t= i)$, and **B** to be an emission matrix **B** whose elements b_{ij} correspond to $P(O_t= j|H_t= i)$
- For t=1, the initial state probability distribution is encoded in a vector π with elements $\pi_i=P(H_t=i)$
- The complete set of parameters is $\theta=\{\mathbf{A},\mathbf{B},\boldsymbol{\pi}\}$, i.e. a set containing two matrices and one vector
- We write a particular observation sequence as a set of observations

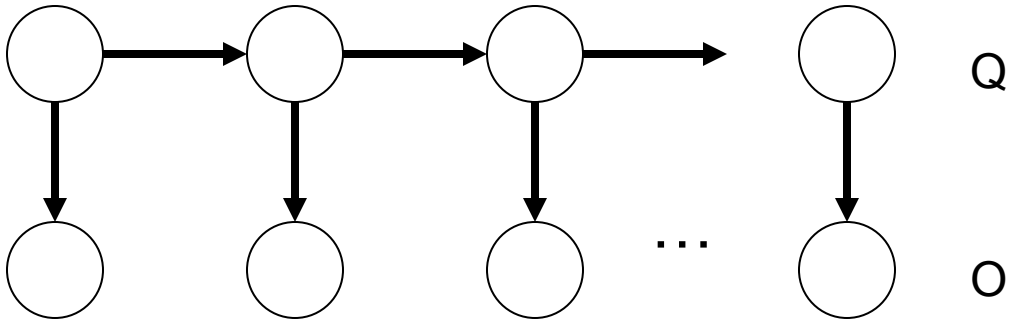
$$\tilde{\mathrm{O}}=\{O_1 = o_1,\ldots,O_T = o_T\}$$

HMMs: key problems

1. Compute $P(\tilde{O};\theta)$, the probability of a sequence under the model with parameters θ
2. Find the most probable explanation—the best sequence of states $H^* = \{H_1 = h_1, \ldots, H_T = h_T\}$ that explains an observation
3. Find the best parameters θ for the model given a data set of observed sequences

- The first problem can be solved using the sum-product algorithm, the second using the max-product algorithm, and the third using the EM algorithm in which the required expectations are computed using the sum-product algorithm

Les modèles de Markov caché (MMC) « Hidden Markov Models - HMMs »



- Un MMC est un modèle de probabilité joint des variables observées $\{O_t\}$ (discret or continu) et des variables cachée $\{Q_t\}$ (discret)

$$P(\{O\},\{Q\}) = \prod_{t=1}^{T} P(O_t \mid Q_t) \prod_{t=1}^{T} P(Q_{t+1} \mid Q_t) P(Q_t)$$

MMCs - Définitions en détail

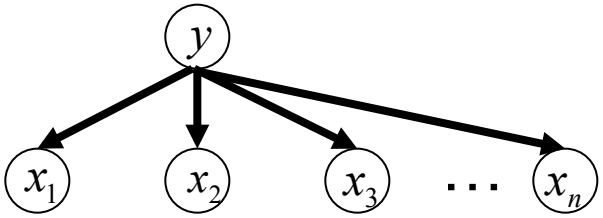
- Un séquence d'observation est donné par $O = (O_1 = o_1, \ldots, O_T = o_T)$
- Q_t est une variable aléatoire discrète avec des valeurs possibles parmi $\{1, \ldots, D\}$.
- Donc, nous pouvons définir une matrice de transition stochastique et indépendante du temps $A = \{a_{ij}\} = P(Q_{t+1} = j \mid Q_t = i)$.
- Normalement on utilise la même probabilité conditionnelle $P(Q_{t+1} \mid Q_t)$ à chaque étape t
- *Nous appelons* $P(O_t \mid Q_t)$ la matrice d'émission
- Le cas spécial du temps t=1 est décrit par la distribution d'état initial $\pi_i = P(Q_t = i)$

Les solutions de trois problèmes

- Nous décrivons l'ensemble complet de paramètres d'un MMC par λ = (A, B, π)
- Il y a trois problèmes de base liés aux MMCs

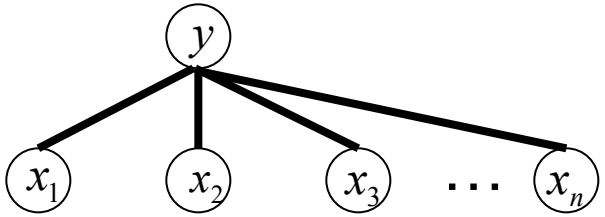
1. Trouvez p(O|λ) pour certains O = (O_1= o_1,…,O_T= o_T). Nous employons l'algorithme « **forward-backward** » ou l'algorithme somme-produit
2. Donné quelques observations et quelques λs, trouvez le meilleur ordre d'état q = (q_1,…, q_T) qui explique O L'algorithme de **Viterbi** qui résout ce problème (C'est le même chose que l'algorithme max-sum)
3. Trouvez arg max λ P(O|λ) . L'algorithme de « **Baum-Welch** » résout ce problème (C'est simplement l'approche d'**EM**)

Rappel - les modèles (simple) analogues : génératif, conjoint et conditionnel



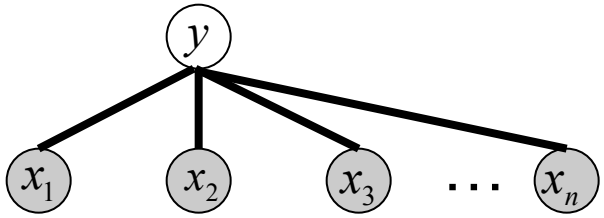
- Modèle génératif

$$P(y,x) = P(y)\prod_{i=1}^{N} P(x_i \mid y)$$



- Modèle conjoint

$$P(y,x) = \frac{1}{Z}\phi(y)\prod_{i=1}^{N} \phi(x_i, y)$$

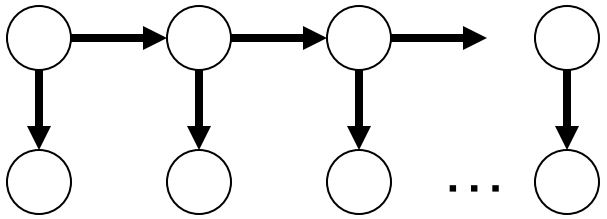


- Modèle conditionnel

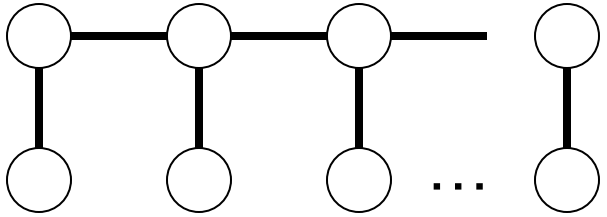
$$P(y \mid x) = \frac{1}{Z(\mathbf{x})}\phi(y)\prod_{i=1}^{N} \phi(x_i, y)$$

Modèles génératifs, conjoints et conditionnels

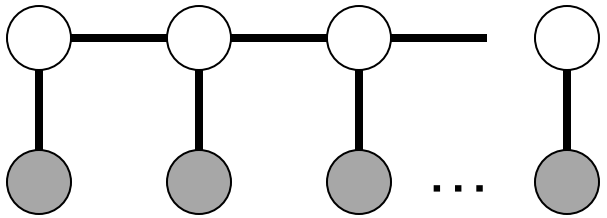
- Modèle de Markov caché (MMC)
  Hidden Markov model (HMM)



- Champ aléatoire
  « Markov Random Field » (MRF)



- Champ aléatoire conditionnel
  « conditional random field »
  (CRF)



Conditional random fields (CRFs)

- Recall, a Markov random field factorizes the *joint distribution* for X using an exponentiated energy function $F(X)$:

$$P(X) = \frac{1}{Z}\exp(-F(X)), \qquad Z = \sum_X \exp(-F(X)),$$

- Conditional random fields condition on some observations X, yielding a *conditional distribution*

$$P(Y \mid X) = \frac{1}{Z(X)}\exp(-F(Y,X)), \qquad Z(X) = \sum_Y \exp(-F(Y,X)),$$

CRF Energy Function

- Both Markov and conditional random fields can be defined for general model structures, but the energy functions usually include just one or two variables—unary and pairwise potentials.
- Conceptually, to create a conditional random field for $P(Y|X)$ based on U unary and V pairwise functions of variables in Y, the energy function takes the form

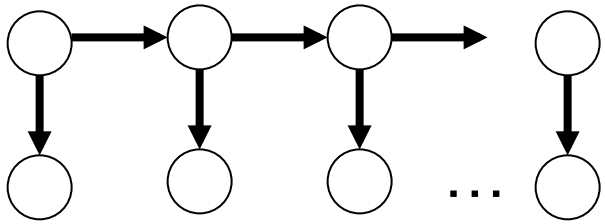
$$F(Y,\tilde{X}) = \sum_{u=1}^{U} U(Y_u,\tilde{X}) + \sum_{v=1}^{V} V(Y_v,\tilde{X})$$

$$P(\mathrm{Y}|\tilde{\mathrm{X}}) = \frac{1}{Z(\tilde{\mathrm{X}})}\exp\left[\sum_{u=1}^{U} U(\mathrm{Y}_u,\tilde{X}) + \sum_{v=1}^{V} V(\mathrm{Y}_v,\tilde{X})\right]$$

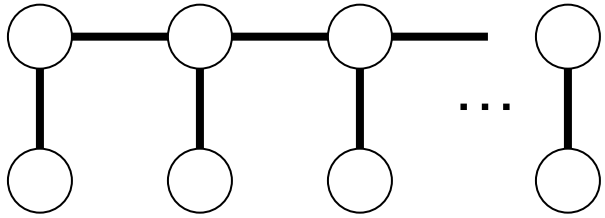
$$= \frac{1}{Z(\tilde{\mathrm{X}})}\prod_{u=1}^{U}\phi_u(\mathrm{Y}_u,\tilde{\mathrm{X}})\prod_{v=1}^{V}\Psi_v(\mathrm{Y}_v,\tilde{\mathrm{X}}).$$

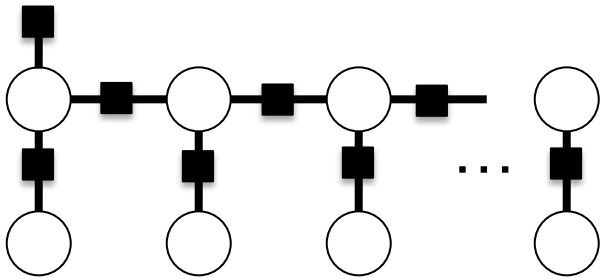
HMMs, MRFs, CRFs and Factor Graphs

Hidden Markov model

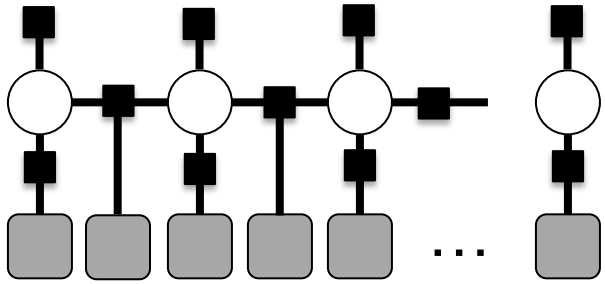


Markov random field



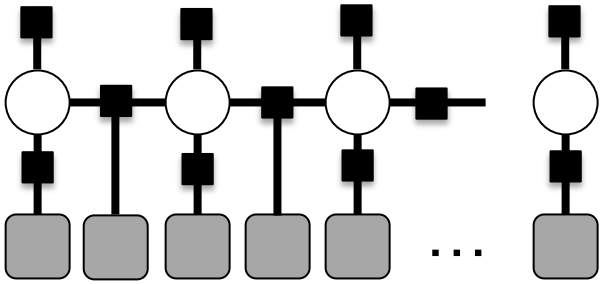


Factor graph for an HMM



Conditional random field

Linear chain CRFs



- For a given sequence of length *N* define

$$P(\mathrm{Y}|\tilde{\mathrm{X}}) = \frac{1}{Z(\tilde{\mathrm{X}})}\prod_{u=1}^{N}\phi_u(\mathrm{y}_u,\tilde{\mathrm{X}})\prod_{v=1}^{N}\Psi_v(\mathrm{y}_v,y_{v+1},\tilde{\mathrm{X}})$$

$$\phi_i[\mathrm{y}_\mathrm{i}] = \exp\left[\sum_{\mathrm{j}=1}^{\mathrm{J}}\theta_j^u u_\mathrm{j}(\mathrm{y}_\mathrm{i},\tilde{\mathrm{X}},i)\right] \quad \Psi_i[\mathrm{y}_\mathrm{i},\mathrm{y}_{\mathrm{i}+1}] = \exp\left[\sum_{\mathrm{k}=1}^{\mathrm{K}}\theta_k^v v_\mathrm{k}(\mathrm{y}_\mathrm{i},\mathrm{y}_{\mathrm{i}+1},\tilde{\mathrm{X}},i)\right]$$

- There are two types of features: a set of *J single variable* (state) features $u_j(y_\mathrm{i},\mathrm{X},i)$ that are a function of a single y_i, and which are computed for each y_i in the sequence i=1…N; and a set of *K pairwise* (transition) features $v_k(y_{\mathrm{i}-1},y_i,\mathrm{X},i)$ for i>1.
- Each type has its own associated unary weights $\theta_\mathrm{j}^\mathrm{u}$ and pairwise weights $\theta_\mathrm{k}^\mathrm{v}$

The global feature vector view for CRFs

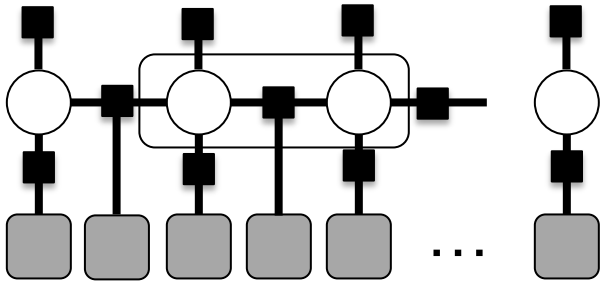
- Features can be a function of the entire observed sequence
- The global dependence on the input is a major advantage of conditional random fields over HMMs
- Can be useful to combine unary and pairwise features and their parameters to work with all features for a given position *i*
- Define $\mathbf{f}(y_i, y_{i+1}, \mathrm{X}, i)$ as a length-*L* vector containing all single-variable and pairwise features & define the global feature vector to be the sum over position dependent feature vectors so that

$$\mathbf{g}(\mathrm{Y},\mathrm{X}) = \sum_{i=1}^{N} \mathbf{f}(\mathrm{y}_i, \mathrm{y}_{i+1}, \mathrm{X}, i) \qquad P(\mathrm{Y}|\mathrm{X}) = \frac{\exp\left(\boldsymbol{\theta}^{\mathrm{T}} \mathbf{g}(\mathrm{Y},\mathrm{X})\right)}{\sum_{Y} \exp\left(\boldsymbol{\theta}^{\mathrm{T}} \mathbf{g}(\mathrm{Y},\mathrm{X})\right)}$$

- The gradient has similar form to logistic regression

$$\frac{\partial}{\partial \boldsymbol{\theta}} \log P(\{\tilde{\mathrm{Y}}_1,\ldots,\tilde{\mathrm{Y}}_{\mathrm{M}}\} \mid \{\tilde{\mathrm{X}}_1,\ldots,\tilde{\mathrm{X}}_M\}) = \sum_{m=1}^{M} \left[\mathbf{g}(\tilde{\mathrm{Y}}_m, \tilde{\mathrm{X}}_{\mathrm{m}}) - \mathrm{E}_m \left[\mathbf{g}(\mathrm{Y}_m, \tilde{\mathrm{X}}_{\mathrm{m}}) \mid P(\mathrm{Y}_m \mid \tilde{\mathrm{X}}_{\mathrm{m}}) \right] \right]$$

Computing gradients



- A linear chain CRF with L_2 regularization

$$\log P(\mathrm{B}\mid \mathrm{A}) = \sum_{m=1}^{M}\sum_{i=1}^{N_m}\sum_{l=1}^{L}\theta_l g_l(\tilde{y}_{m,i},\tilde{y}_{m,i+1},\tilde{\mathrm{X}}_{\mathrm{m}}) - \sum_{\mathrm{m}=1}^{\mathrm{M}}\log Z(\tilde{\mathrm{X}}_{\mathrm{m}}) - \sum_{l=1}^{L}\frac{\theta_l^2}{2\sigma^2}.$$

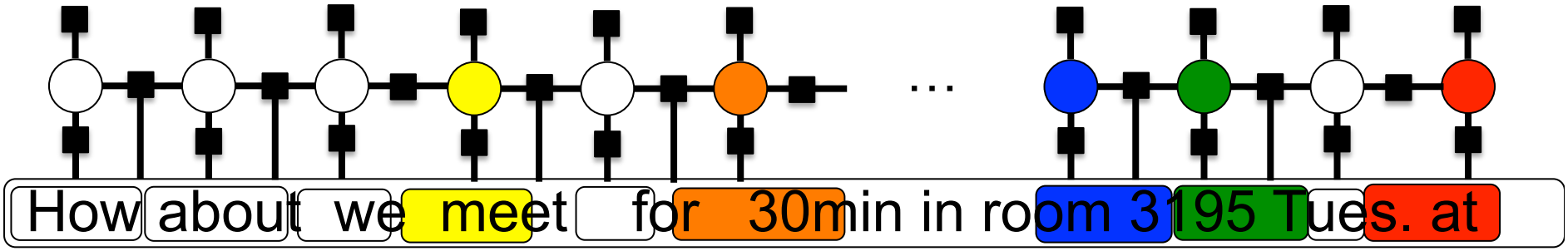
- The contribution to the gradient of each example is

$$\frac{\partial}{\partial\theta_l}\log P(\tilde{\mathrm{Y}}_{\mathrm{m}}\mid\tilde{\mathrm{X}}_{\mathrm{m}}) = \sum_{i=1}^{N_m} g_l(\tilde{y}_i,\tilde{y}_{i+1},\tilde{\mathrm{X}}) - \sum_{i=1}^{N_m}\sum_{y_i}\sum_{y_{i+1}} g_l(y_i,y_{i+1},\tilde{\mathrm{X}})P(y_i,y_{i+1}\mid\tilde{\mathrm{X}}) - \frac{\theta_l}{\sigma^2}.$$

- This is the difference between the observed occurrence of the feature and its expectation under the current prediction of the model, taken with respect to $P(y_i, y_{i+1}\mid\tilde{\mathrm{X}})$, (marginal conditionals) - minus regularization terms
- For unary functions the expectation is w.r.t. $P(y_i\mid\tilde{\mathrm{X}})$
- These are the single- and pairwise-variable marginal conditional distributions, efficiently obtained with the sum-product algorithm

Example: CRFs for information extraction

- Idea: label each word with a *named entity* so as to populate a database record with extracted information
- Consider the following CRF for the line of text



Meeting Record	
Date	Tuesday, March 1, 2016
Time	12:00pm
Location	3195
Duration	unknown

- Each word is tagged with one of the named entities using features computed locally and globally from the sequence

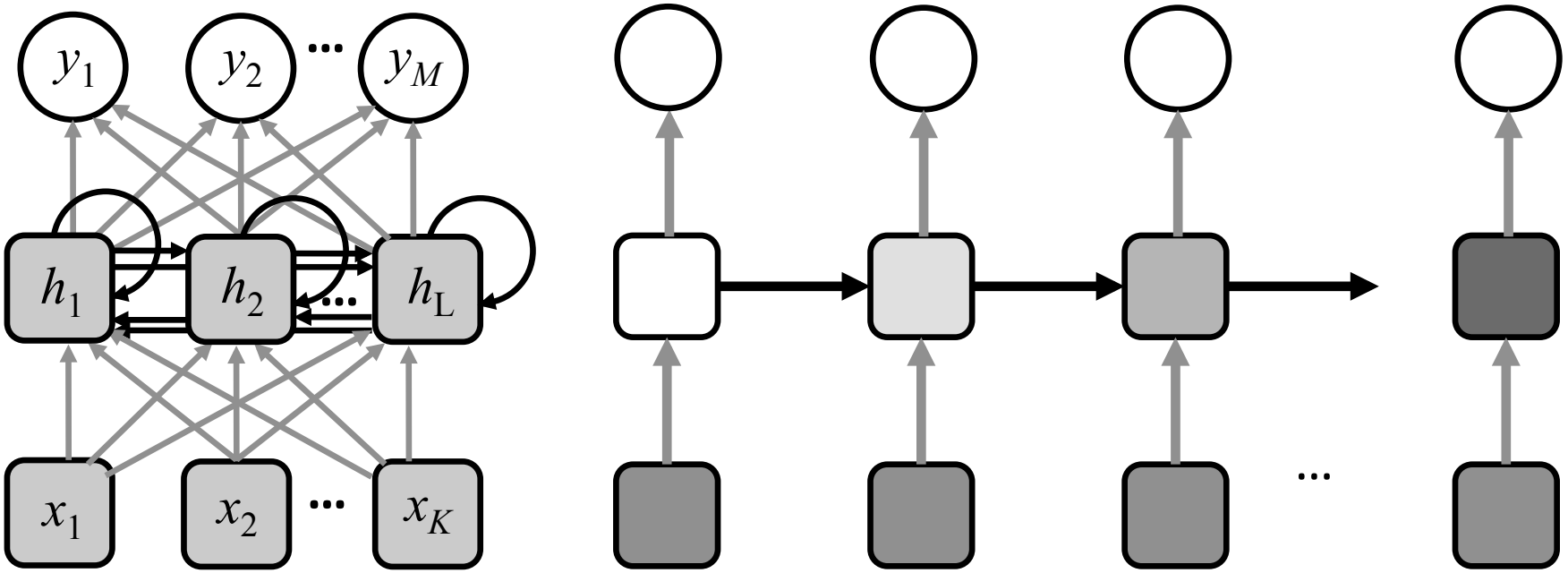
Recurrent neural networks

Recurrent neural networks

- Recurrent neural networks are networks with connections that form directed cycles.
- As a result, they have an internal state, which makes them prime candidates for tackling learning problems involving sequences of data—such as handwriting recognition, speech recognition, and machine translation.
- A feedforward network can be transformed into a recurrent network by adding connections from all hidden units h_i to h_j.
- Each hidden unit has connections to both itself and other hidden units.
- Imagine unfolding a recurrent network over time by following the sequence of steps that perform the underlying computation.
- Like a hidden Markov model, a recurrent network can be unwrapped and implemented using the same weights and biases at each step to link units over time.

Recurrent neural networks (RNNs)

- An RNN can be unwrapped and implemented using the same weights and biases at each step to link units over time as shown below
- The resulting unwrapped RNN is similar to a hidden Markov model, but keep in mind that the hidden units in RNNs are not stochastic



Recurrent neural networks (RNNs)

- Recurrent neural networks apply linear matrix operations to the current observation and the hidden units from the previous time step, and the resulting linear terms serve as arguments of activation functions act():

$$\mathbf{h}_t = \text{act}(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h \mathbf{h}_{t-1} + \mathbf{b}_h)$$

$$\mathbf{o}_t = \text{act}(\mathbf{W}_o \mathbf{h}_t + \mathbf{b}_o)$$

- The same matrix $\mathbf{U}_h$ is used at each time step
- The hidden units in the previous step $\mathbf{h}_{t-1}$ influence the computation of $\mathbf{h}_t$ where the current observation contributes to a $\mathbf{W}_h\mathbf{x}$ term that is combined with $\mathbf{U}_h\mathbf{h}_{t-1}$ and bias $\mathbf{b}_h$ terms
- Both $\mathbf{W}_h$ and $\mathbf{b}_h$ are typically replicated over time
- The output layer is modeled by a classical neural network activation function applied to a linear transformation of the hidden units, the operation is replicated at each step.

The loss, exploding and vanishing gradients

- The loss for a particular sequence in the training data can be computed either at each time step or just once, at the end of the sequence.
- In either case, predictions will be made after many processing steps and this brings us to an important problem.
- The gradient for feedforward networks decomposes the gradient of parameters at layer *l* into a term that involves the product of matrix multiplications of the form $\mathbf{D}^{(l)}\mathbf{W}^{\mathsf{T}(l+1)}$ (see the analysis for feedforward networks above)
- A recurrent network uses the same matrix at each time step, and over many steps the gradient can very easily either diminish to zero or explode to infinity—just as the magnitude of any number other than one taken to a large power either approaches zero or increases indefinitely

Dealing with exploding gradients

- The use of L_1 or L_2 regularization can mitigate the problem of exploding gradients by encouraging weights to be small.
- Another strategy is to simply detect if the norm of the gradient exceeds some threshold, and if so, scale it down.
- This is sometimes called gradient (norm) clipping where for a gradient vector $\mathbf{g}$ and threshold T,

$$\text{if } \|\mathbf{g}\| \geq T \text{ then } \mathbf{g} \leftarrow \frac{T}{\|\mathbf{g}\|}\mathbf{g}$$

where T is a hyperparameter, which can be set to the average norm over several previous updates where clipping was not used.

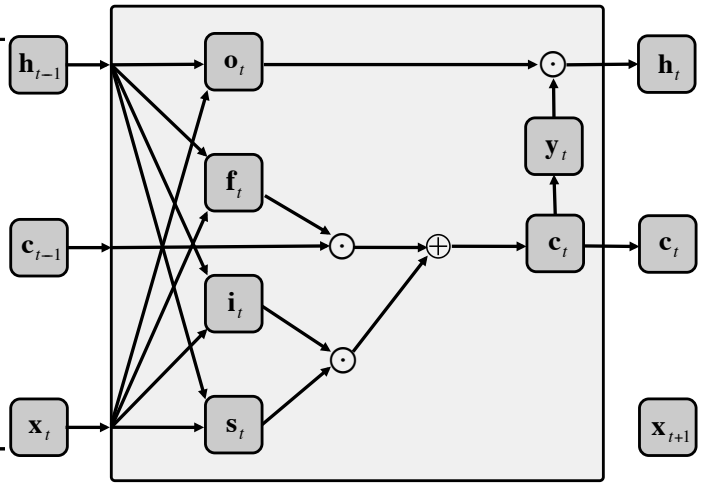
LSTMs and vanishing gradients

- The so-called “long short term memory” (LSTM) RNN architecture was specifically created to address the vanishing gradient problem.
- Uses a combination of hidden units, elementwise products and sums between units to implement gates that control “memory cells”.
- Memory cells are designed to retain information without modification for long periods of time.
- They have their own input and output gates, which are controlled by learnable weights that are a function of the current observation and the hidden units at the previous time step.
- As a result, *backpropagated error terms from gradient computations can be stored and propagated backwards without degradation*.
- The original LSTM formulation consisted of *input gates* and *output gates*, but *forget gates* and “peephole weights” were added later.
- Below we present the most popular variant of LSTM RNNs which does not include peephole weights, but which does use forget gates.
- The architecture is complex, but has produced state-of-the-art results on a wide variety of problems.

LSTM achitecture

- At each time step there are three types of gates: input $\mathbf{i}_t$, forget $\mathbf{f}_t$, and output $\mathbf{o}_t$.
- Each are a function of both the underlying input $\mathbf{x}_t$ at time *t* as well as the hidden units at time *t*-1, $\mathbf{h}_{t-1}$
- Each gate multiplies $\mathbf{x}_t$ by its own gate specific **W** matrix, by its own **U** matrix, and adds its own bias vector **b**.
- This is usually followed by the application of a sigmoidal elementwise non-linearity.

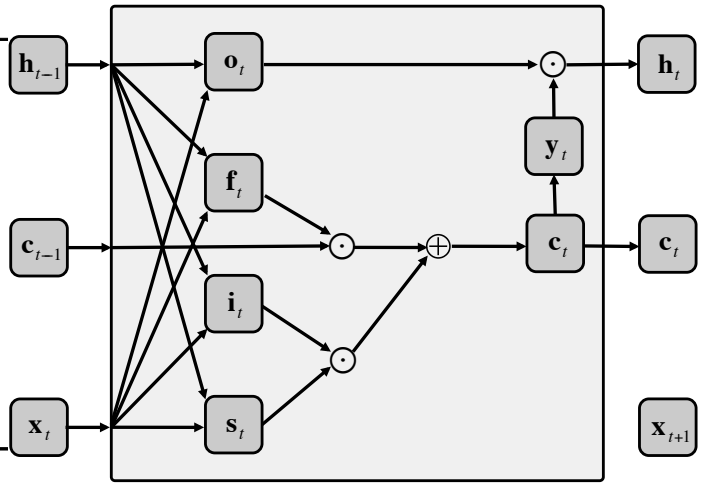
LSTM unit output	$\mathbf{h}_t = \mathbf{o}_t \circ \mathbf{y}_t$
Output gate units	$\mathbf{o}_t = \text{sigmoid}(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o)$
Transformed memory cell contents	$\mathbf{y}_t = \tanh(\mathbf{c}_t)$
Gated update to memory cell units	$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \mathbf{s}_t$
Forget gate units	$\mathbf{f}_t = \text{sigmoid}(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f)$
Input gate units	$\mathbf{i}_t = \text{sigmoid}(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i)$
Potential input to memory cell	$\mathbf{s}_t = \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c)$



LSTM achitecture

- At each time step *t*, input gates $\mathbf{i}_t$ are used to determine when a potential input given by $\mathbf{s}_t$ is important enough to be placed into the memory unit or cell, $\mathbf{c}_t$
- Forget gates $\mathbf{f}_t$ allow memory unit content to be erased
- Output gates $\mathbf{o}_t$ determine whether $\mathbf{y}_t$, the content of the memory units transformed by activation functions, should be placed in the hidden units $\mathbf{h}_t$
- Typical gate activation functions and their dependencies are shown below

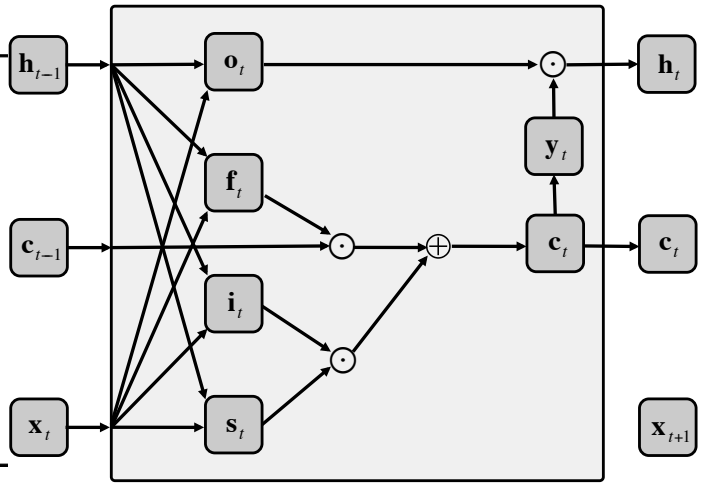
LSTM unit output	$\mathbf{h}_t = \mathbf{o}_t \circ \mathbf{y}_t$
Output gate units	$\mathbf{o}_t = \text{sigmoid}(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o)$
Transformed memory cell contents	$\mathbf{y}_t = \tanh(\mathbf{c}_t)$
Gated update to memory cell units	$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \mathbf{s}_t$
Forget gate units	$\mathbf{f}_t = \text{sigmoid}(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f)$
Input gate units	$\mathbf{i}_t = \text{sigmoid}(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i)$
Potential input to memory cell	$\mathbf{s}_t = \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c)$



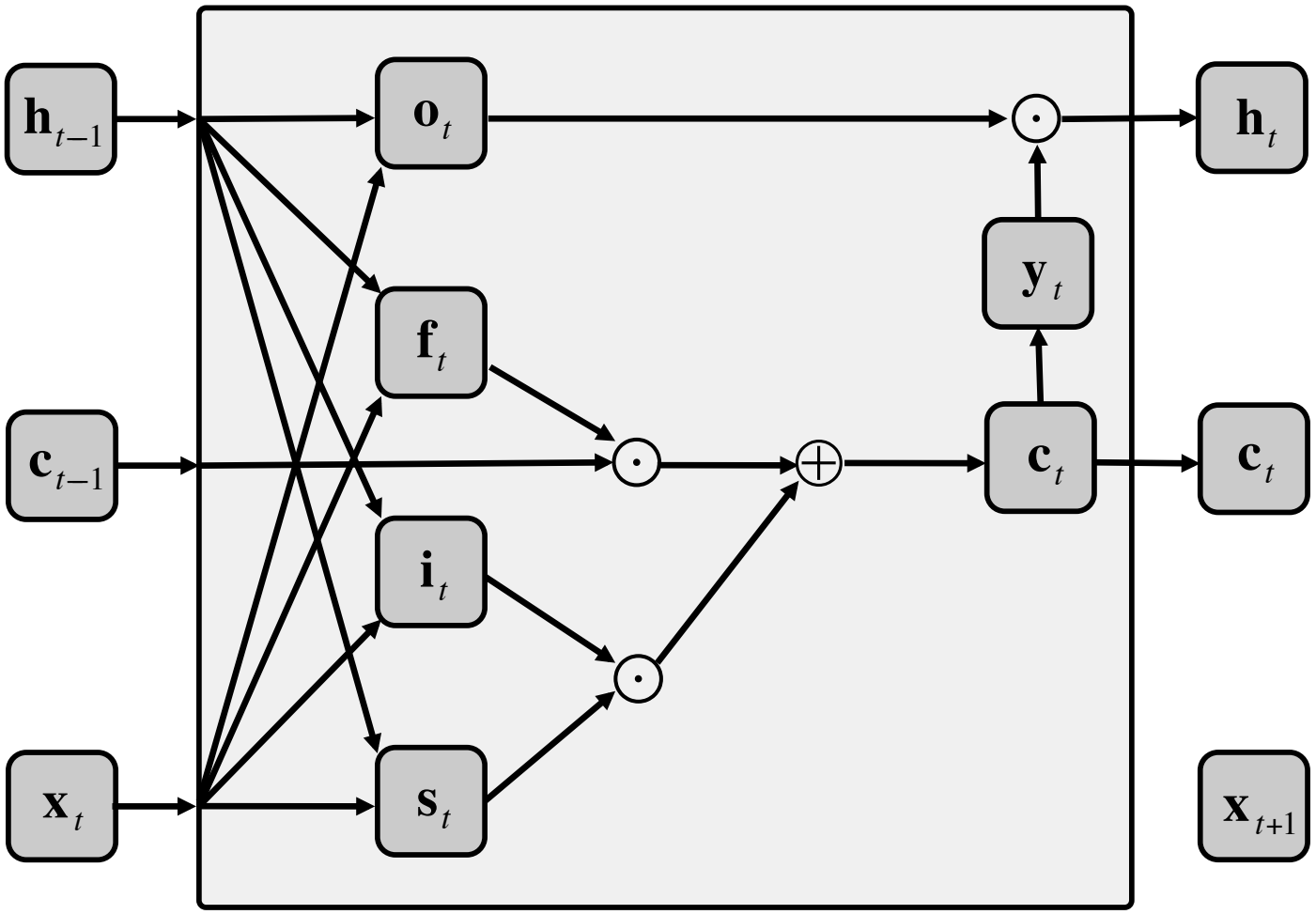
LSTM achitecture

- The final gating is implemented as an elementwise product between the output gate and the transformed memory contents, $\mathbf{h}_t = \mathbf{o}_t \circ \mathbf{y}_t$
- Memory units are typically transformed by the tanh function prior to the gated output, such that $\mathbf{y}_t = \tanh(\mathbf{c}_t)$
- Memory units or cells are updated by $\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \mathbf{s}_t$ an elementwise product between the forget gates and the previous contents of the memory units, plus the elementwise product of the input gates and the new potential inputs .

LSTM unit output	$\mathbf{h}_t = \mathbf{o}_t \circ \mathbf{y}_t$
Output gate units	$\mathbf{o}_t = \text{sigmoid}(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o)$
Transformed memory cell contents	$\mathbf{y}_t = \tanh(\mathbf{c}_t)$
Gated update to memory cell units	$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \mathbf{s}_t$
Forget gate units	$\mathbf{f}_t = \text{sigmoid}(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f)$
Input gate units	$\mathbf{i}_t = \text{sigmoid}(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i)$
Potential input to memory cell	$\mathbf{s}_t = \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c)$



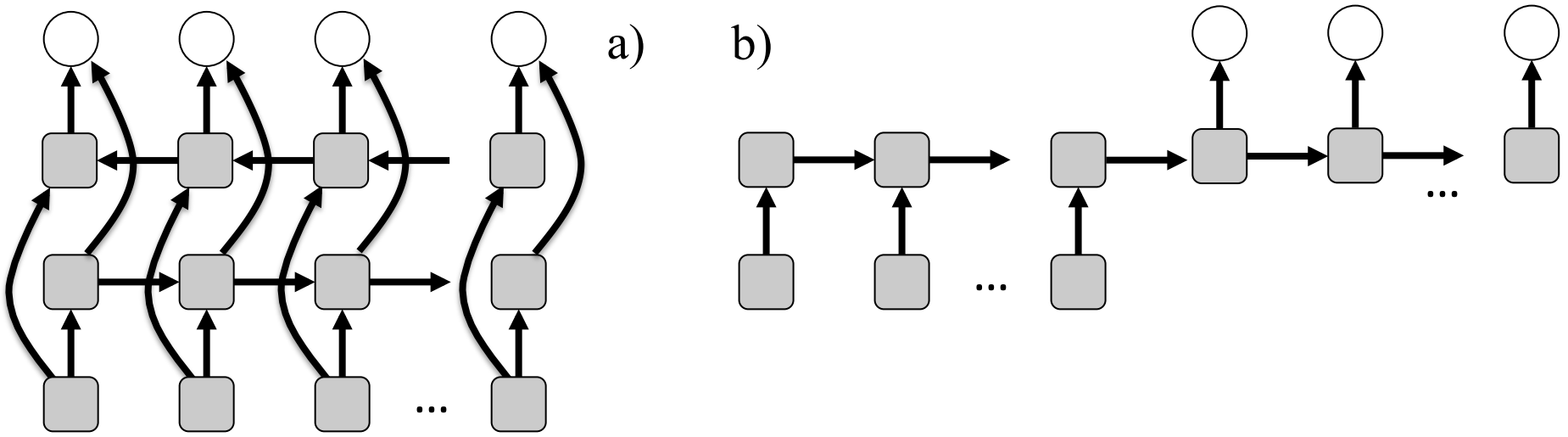
LSTM achitecture



Other RNN architectures

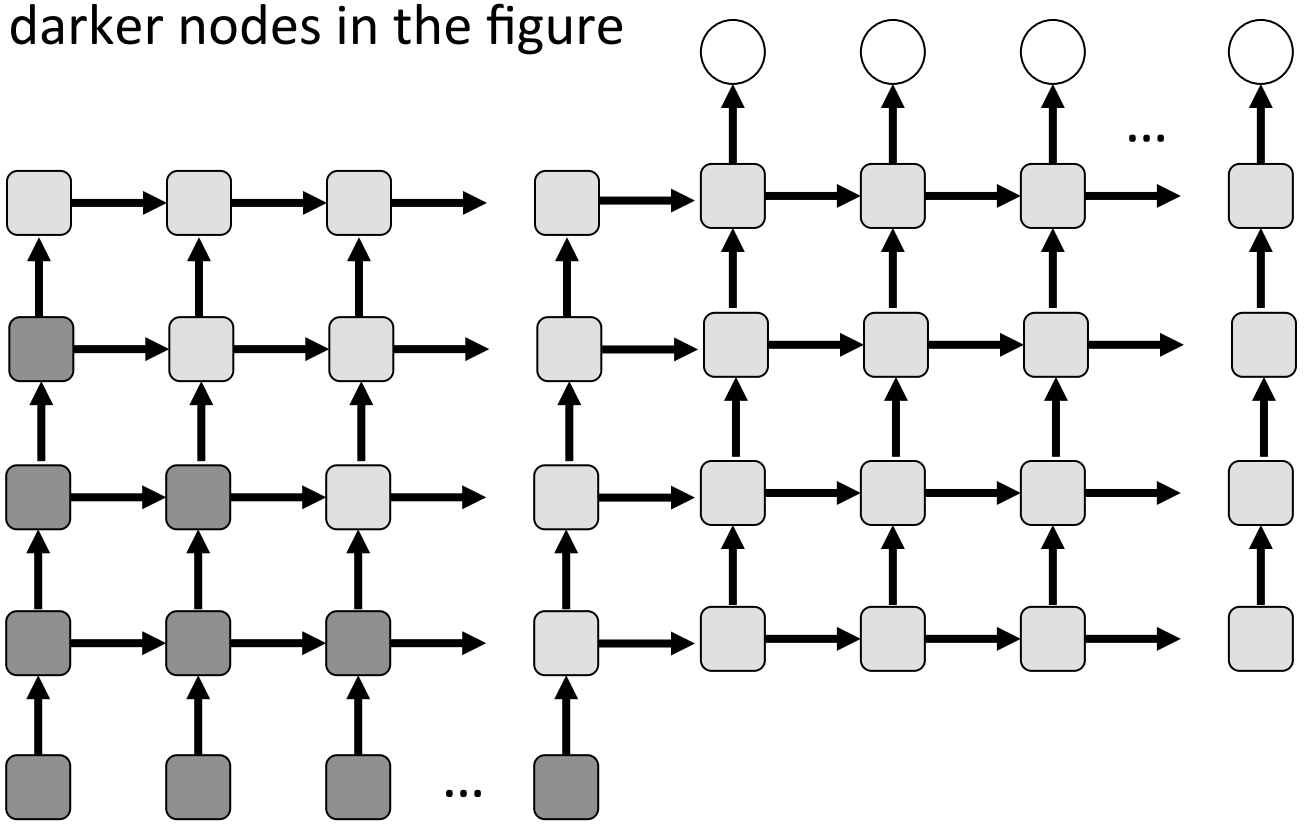
a) Recurrent networks can be made bidirectional, propagating information in both directions
   - They have been used for a wide variety of applications, including protein secondary structure prediction and handwriting recognition

b) An "encoder-decoder" network creates a fixed-length vector representation for variable-length inputs, the encoding can be used to generate a variable-length sequence as the output
   - Particularly useful for machine translation



Deep encoder-decoder architectures

- Given enough data, a deep encoder-decoder architecture such as that below can yield results that compete with translation systems that have been hand-engineered over decades of research.
- The connectivity structure means that partial computations in the model can flow through the graph in a wave, illustrated by the darker nodes in the figure



Bibliographic Notes & Further Reading

Recurrent neural networks

- Graves et al. (2009) demonstrate how recurrent neural networks are particularly effective at handwriting recognition,
- Graves et al. (2013) apply recurrent neural networks to speech.
- The form of gradient clipping presented above was proposed by Pascanu et al. (2013).
- Hochreiter and Schmidhuber (1997) is the seminal work on the "Long Short-term Memory" architecture for recurrent neural networks;
  - our explanation follows Graves and Schmidhuber (2005)'s formulation.
- Greff et al. (2015)'s paper "LSTM: A search space odyssey" explored a wide variety of variants and finds that:
  a) none of them significantly outperformed the standard LSTM architecture; and
  b) forget gates and the output activation function were the most critical components. Forget gates were added by Gers et al. (2000).

Bibliographic Notes & Further Reading

Recurrent neural networks

- IRNNs were proposed by Le et al. (2015)
- Chung et al. (2014) proposed gated recurrent units
- Schuster and Paliwal (1997) proposed bidirectional recurrent neural networks
- Chen and Chaudhari (2004) used bi-directional networks for protein structure prediction; Graves et al. (2009) used them for handwriting recognition
- Cho et al. (2014) used encoder-decoder networks for machine translation, while Sutskever et al. (2014) proposed deep encoder-decoder networks and used them with massive quantities of data
- For further accounts of advances in deep learning and a more extensive history of the field, consult the reviews of LeCun et al. (2015), Bengio (2009), and Schmidhuber (2015)

Markov Random Fields

Undirected Graphical Models

Markov random fields (MRFs)

- A clique is a group of nodes in an undirected graph where all nodes are connected to one another
- MRFs define another factorized model for a set of random variables X, where clique sets are given by X_c and a factor $\Psi_c(X_c)$ is defined for each clique such that

$$P(X) = \frac{1}{Z} \prod_{c=1}^{C} \Psi_c(X_c),$$

- The partition function Z normalizes the result to form a probability distribution

$$Z = \sum_{x \in X} \prod_{c=1}^{C} \Psi_c(X_c).$$

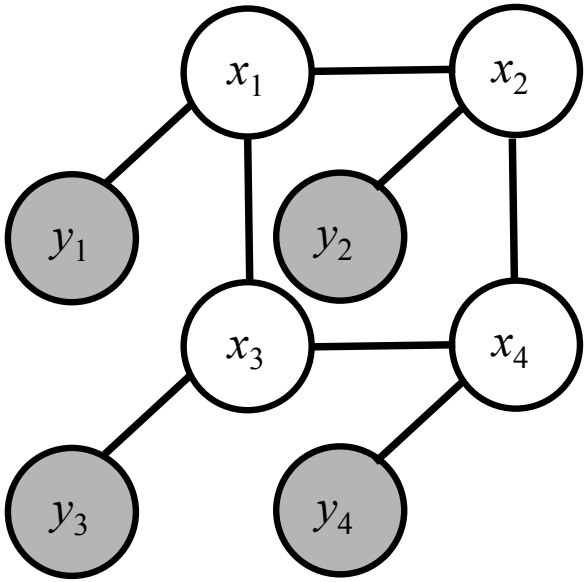
Markov random field example

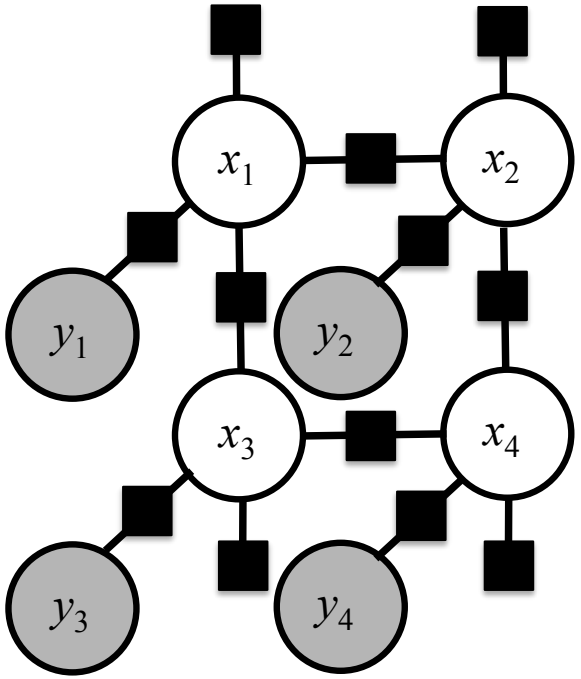
$$P(x_1, x_2, x_3, x_4) = \frac{1}{Z} \prod_{u=1}^{U} \phi_u(X_u) \prod_{v=1}^{V} \Psi_v(X_v)$$

$$= \frac{1}{Z} f_A(x_1) f_B(x_2) f_C(x_1) f_D(x_2) f_E(x_1, x_2) f_F(x_2, x_3) f_G(x_3, x_4) f_H(x_4, x_1)$$

- Note we used unary and pairwise potentials

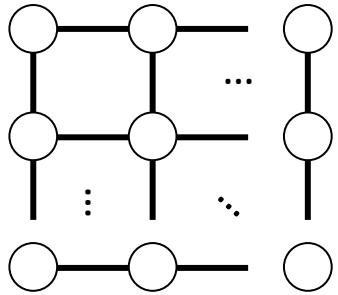
An MRF using a traditional MRF style undirected graph





An MRF as a factor graph

MRFs and energy functions



- An MRF lattice is often repeated over an imag
- MRFs can be expressed in terms of an energy function $F(X)$, where

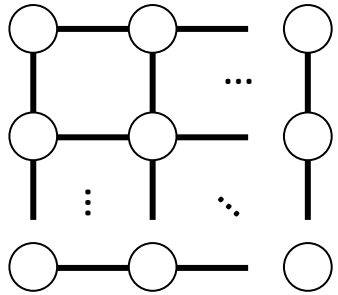
$$F(X) = \sum_{u=1}^{U} U(X_u) + \sum_{v=1}^{V} V(X_v), \qquad \text{and}$$

$$P(X) = \frac{1}{Z}\exp(-F(X)) = \frac{1}{Z}\exp\left(-\sum_{u=1}^{U} U(X_u) - \sum_{v=1}^{V} V(X_v)\right).$$

- Since Z is constant for any assignment of the variables X we have

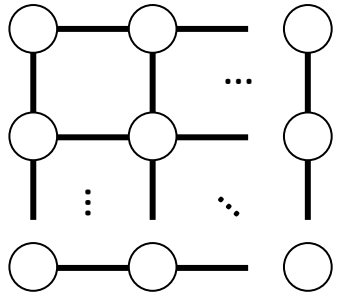
$$-\log P(x_1, x_2, x_3, x_4) \propto \sum_{u=1}^{U} U(X_u) + \sum_{v=1}^{V} V(X_v)$$

Minimizing MRF energies



- A commonly used strategy for tasks such as image segmentation and entity resolution in text documents is to minimizing an energy function of the form in the previous slide
- When such energy functions are “sub-modular”, an exact minimum can be found using algorithms based on graph-cuts; otherwise methods such as tree-reweighted message passing can be used.

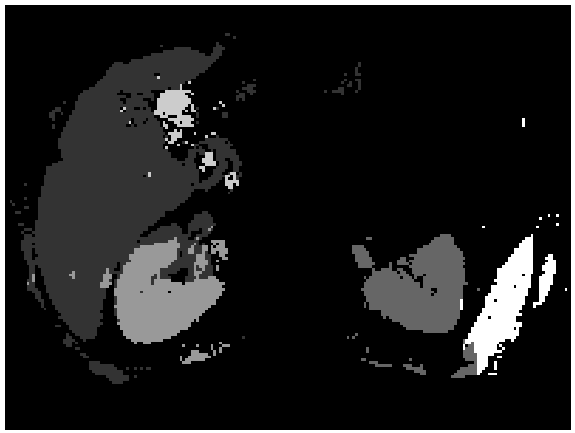
Example: Image Segmentation



- Given a labeled dataset of medical imagery, such as a computed tomography or CT image
- A well known approach is to learn a Markov Random Field model to segment that image into classes of interest, ex. anatomical structures or tumors
- Image features are combined with spatial context

Original Image

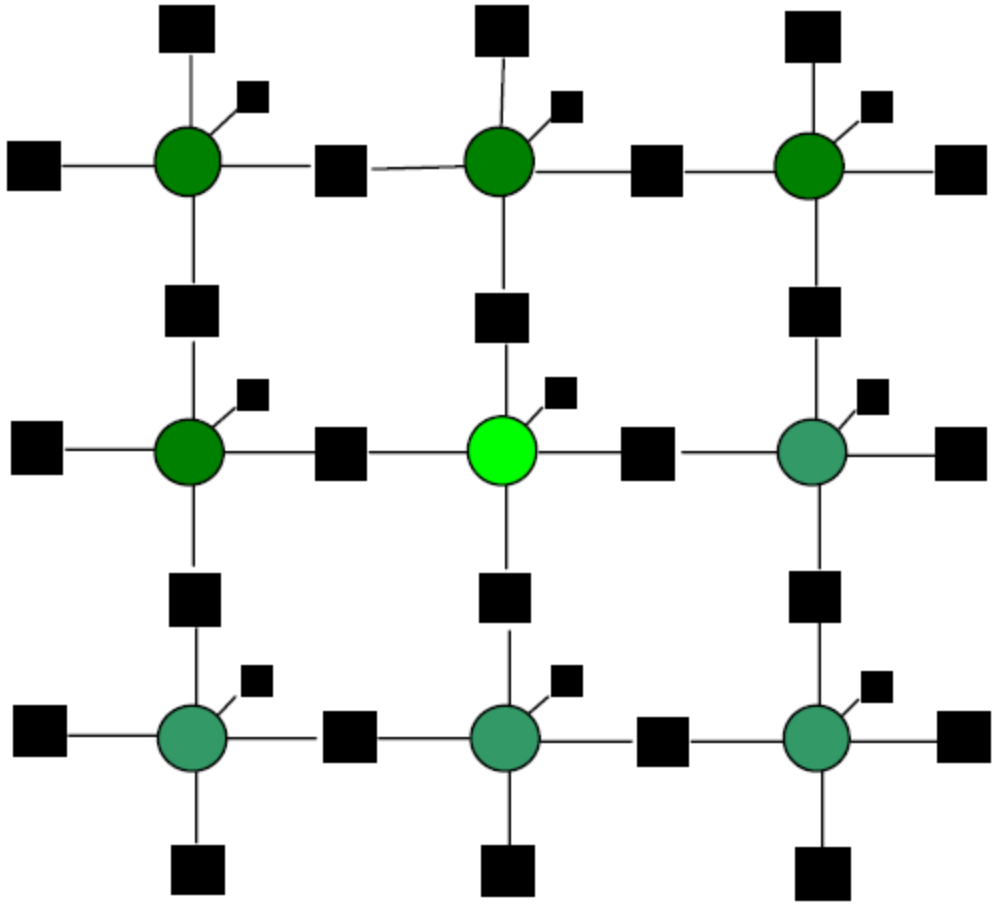
Ground truth



MRF segmentation

From Bhole et al.

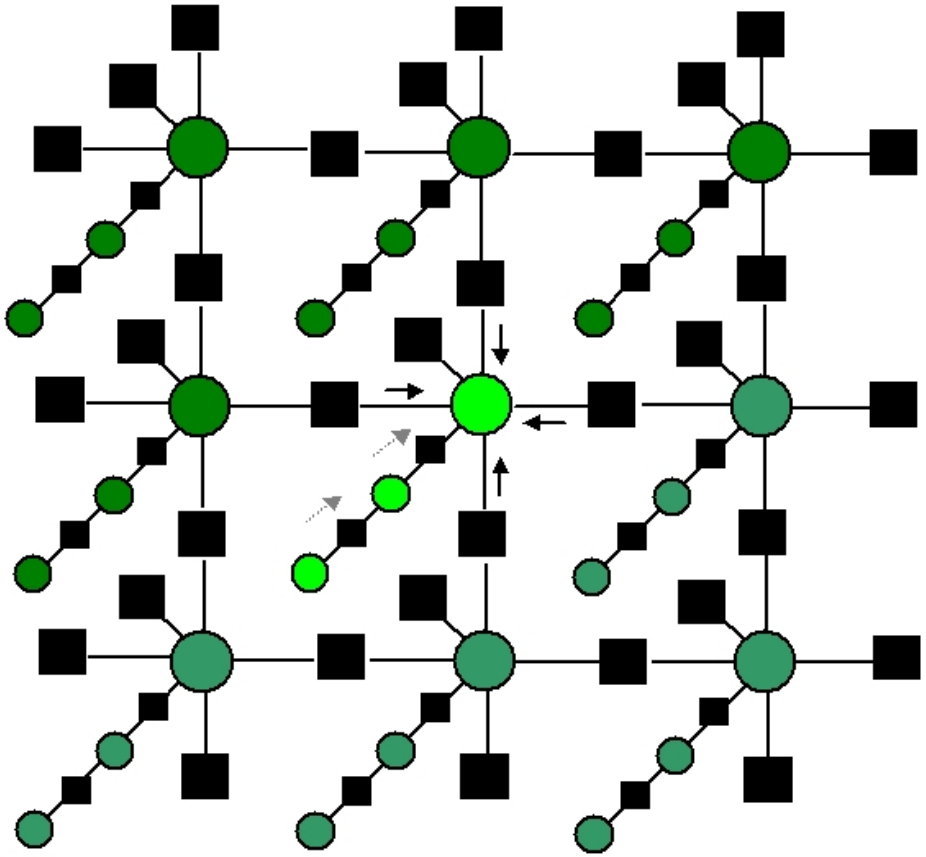
Champs aléatoires pour les images



From: Pal

- Modèles non-oriente

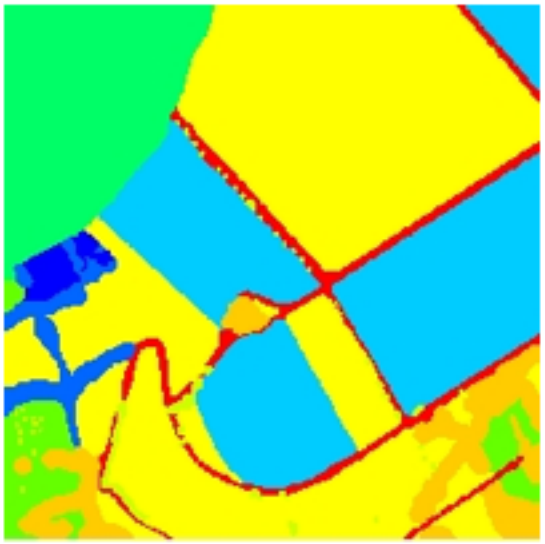
Markov Random Fields



From: Pal

- Widely applicable, ‘undirected’ graphical models

Markov Random Fields



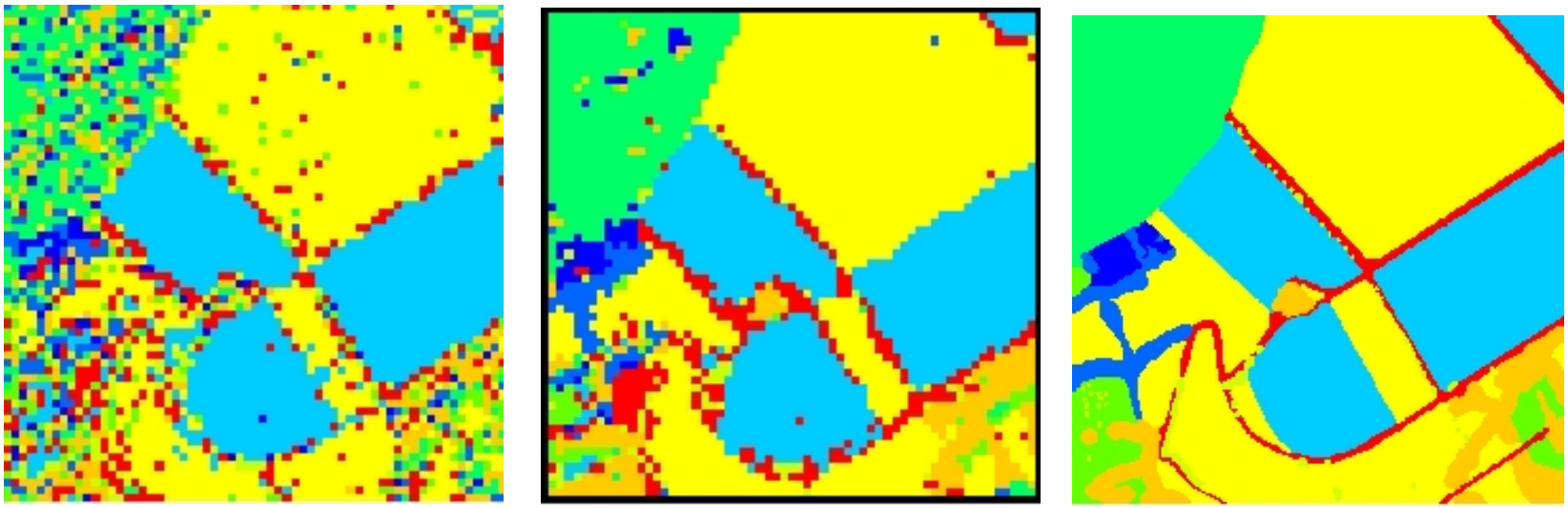
Color	Interpretation
	Field Boundaries
	Enclosed Grass Area
	Deciduous Forest
	Small Sparse Trees
	Deciduous Tree Line
	Open Field
	Dense Coniferous
	Mixed Tree line
	Crops

Figure 1. Aerial imagery and an associated coarse scale hand classification.

From: Pal

- Widely applicable, 'undirected' graphical models

Markov Random Fields



From: Pal

- Widely applicable, ‘undirected’ graphical models

Bibliographic Notes & Further Reading

Graphical Probabilty Models (GPMs) and Inference

- Plate notation has been widely used in artificial intelligence (Buntine, 1994), machine learning (Blei et al., 2003) and computational statistics (Lunn et al., 2000) to define complex probabilistic graphical models,
- GPMs form the basis of the BUGS (Bayesian inference Using Gibbs Sampling) software project (Lunn et al., 2000)
- Our presentation of factor graphs and the sum-product algorithm follows their origins in Kschischang et al. (2001) and Frey (1998)
- Bayesian networks and other models that contain cycles can be manipulated into a structure known as a *junction tree* by clustering variables, and Lauritzen and Spiegelhalter (1988)'s junction tree algorithm permits exact inference

Bibliographic Notes & Further Reading

Graphical Probabilty Models and Inference

- Ripley (1996) covers the junction tree algorithm, with practical examples
- Huang and Darwiche (1996)'s procedural guide is an excellent resource for those who need to implement the algorithm
- Probability propagation in a junction tree yields exact results, but is sometimes infeasible because the clusters become too large—in which case one must resort to sampling or variational methods

Boltzmann machines

Boltzmann machines

- Are a type of Markov random field often used for unsupervised learning
- Unlike the units of a feedforward neural network, the units in Boltzmann machines correspond to random variables, such as are used in Bayesian networks
- Older variants of Boltzmann machines were defined using exclusively binary variables, but models with continuous and discrete variables are also possible
- They became popular prior to the impressive results of convolutional neural networks on the ImageNet challenge, but have since waned in popularity because they are more difficult to work with

Boltzmann machines

- To create a Boltzmann machine we partitioning variables into ones that are visible, using a *D*-dimensional binary vector **v**, and ones that are hidden, defined by a *K*-dimensional binary vector **h**
- A Boltzmann machine is a joint probability model of the form

$$p(\mathbf{v},\mathbf{h};\theta) = \frac{1}{Z(\theta)}\exp(-E(\mathbf{v},\mathbf{h};\theta)), \quad Z(\theta) = \sum_{\mathbf{v}}\sum_{\mathbf{h}}\exp(-E(\mathbf{v},\mathbf{h};\theta)),$$

$$E(\mathbf{v},\mathbf{h};\theta) = -\frac{1}{2}\mathbf{v}^T\mathbf{A}\mathbf{v} - \frac{1}{2}\mathbf{h}^T\mathbf{B}\mathbf{h} - \mathbf{v}^T\mathbf{W}\mathbf{h} - \mathbf{a}^{\mathrm{T}}\mathbf{v} - \mathbf{b}^{\mathrm{T}}\mathbf{h},$$

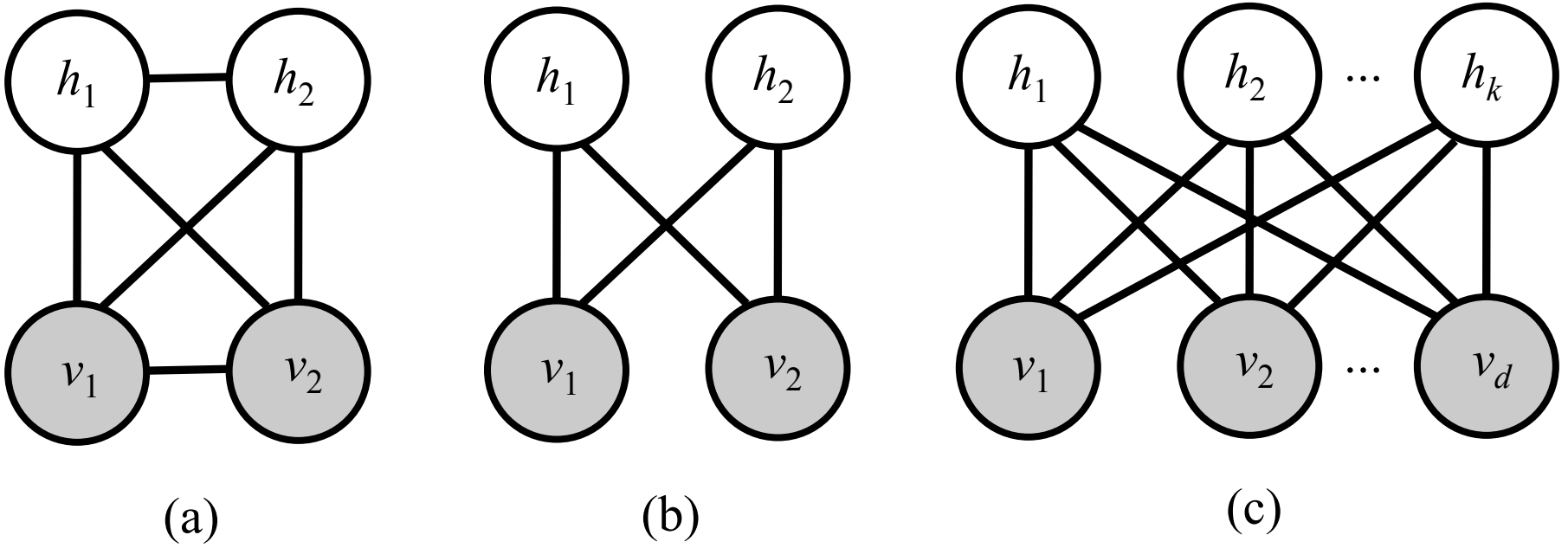
- where E(**v**,**h**;θ) is the energy function
- Z(θ) normalizes *E* so that it defines a valid joint probability
- matrices **A**, **B** and **W** encode the visible-to-visible, hidden-to-hidden and the visible-to-hidden variable interactions respectively
- vectors **a** and **b** encode the biases associated with each variable
- matrices **A** and **B** are symmetric, and their diagonal elements are 0

Boltzmann machines

(a) Boltzman Machines are binary Markov random field with pairwise connections between all variables

(b) Restricted Boltzmann machines (RBMs) do not have connections between the variables in a layer

(c) RBMs can be extended to many variables as shown



Key feature of Boltzmann machines

- A key feature of Boltzmann machines (and binary Markov random fields in general) is that the conditional distribution of one variable given the others is a sigmoid function whose argument is a weighted linear combination of the states of the other variables

$$p(h_j = 1 \mid \mathbf{v}, \mathbf{h}_{\neg j}; \theta) = \text{sigmoid}\left(\sum_{i=1}^{D} W_{ij} v_i + \sum_{k=1}^{K} B_{jk} h_k + b_j \right),$$

$$p(v_i = 1 \mid \mathbf{h}, \mathbf{v}_{\neg i}; \theta) = \text{sigmoid}\left(\sum_{j=1}^{K} W_{ij} h_j + \sum_{d=1}^{D} A_{id} v_d + c_i \right),$$

where the notation $\neg i$ indicates all elements with subscript other than *i*.

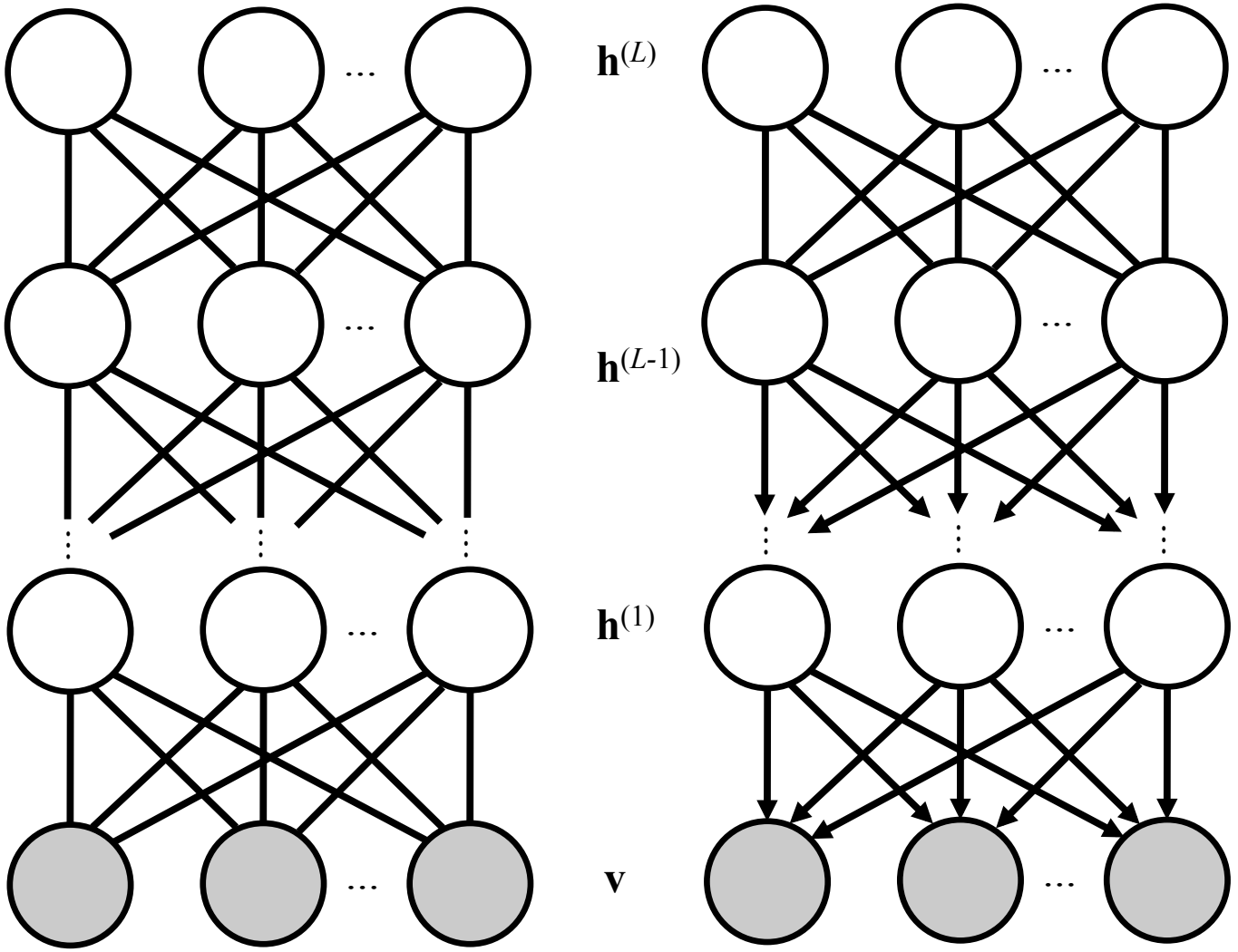
Contrastive divergence

- Running a Gibbs sampler for a Boltzmann machine often requires many iterations,
- A technique called “contrastive divergence” is a popular alternative that initializes the sampler to the observed data instead of randomly and performs a limited number of Gibbs updates.
- In an RBM a sample can be generated from the sigmoid distributions for all the hidden variables given the observed; then samples can be generated for the observed variables given the hidden variable sample
- This single step often works well in practice, although the process of alternating the sampling of hidden and visible units can be continued for multiple steps.

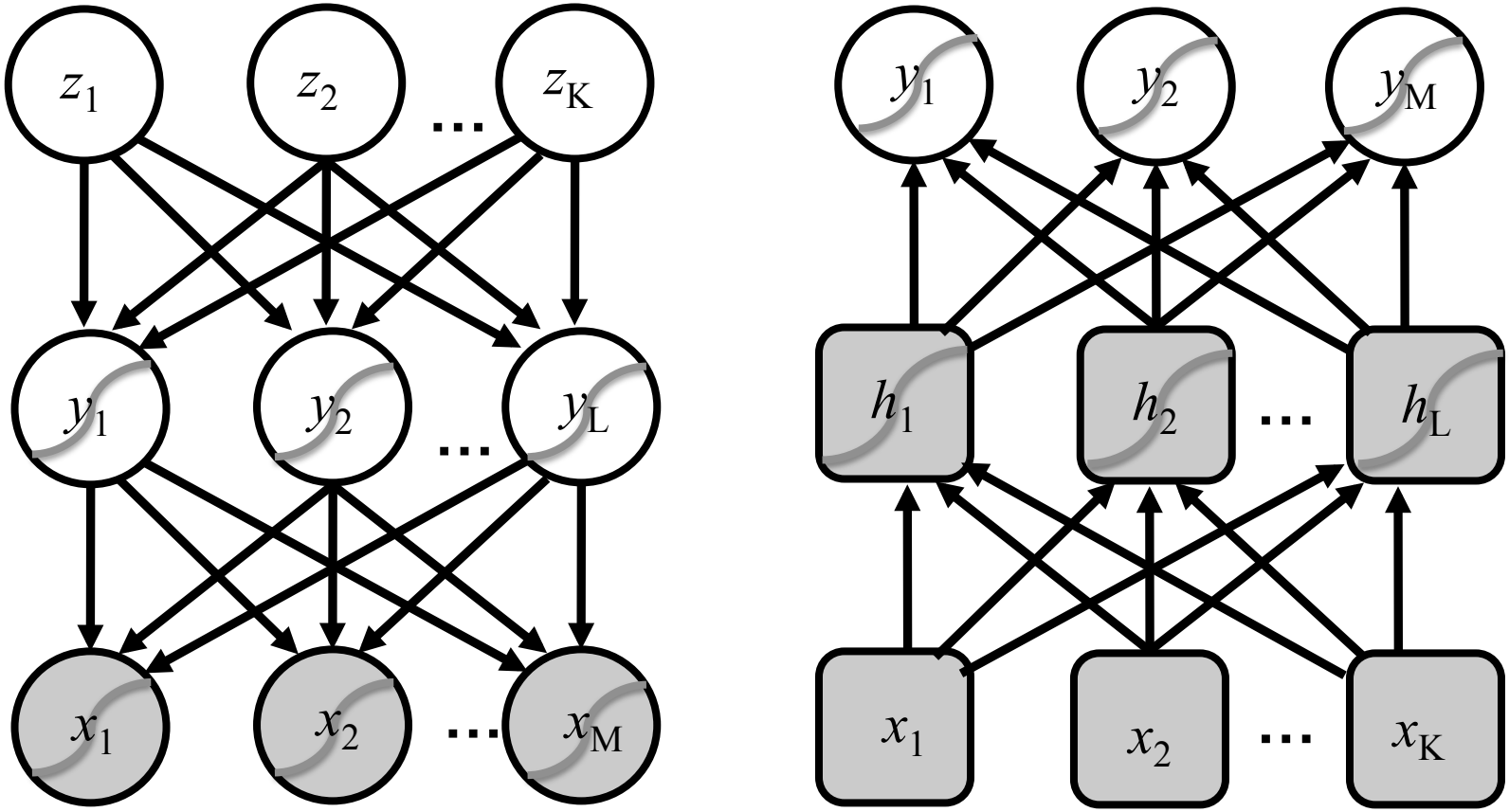
Deep RBMs and deep belief networks

- Deep Boltzmann machines involve coupling layers of random variables using restricted Boltzmann machine connectivity
- While any deep Bayesian network is technically a deep belief network, the term “deep belief network” has become strongly associated with a particular type of deep architecture that can be constructed by training restricted Boltzmann machines incrementally.
- The procedure is based on converting the lower part of a growing model into a Bayesian belief network, adding an RBM for the upper part of the model, then continuing the training, conversion and stacking process.

A deep RBM vs a deep belief network



A sigmoidal belief network vs a neural network



Bibliographic Notes & Further Reading

Stochastic methods – Boltzmann machines

- The history of Markov random fields has roots in statistical physics in the 1920s with so-called "Ising models" of ferromagnetism
- Our presentation of Boltzmann machines follows Hinton and Sejnowski (1983), but we use matrix-vector notation and our exposition more closely resembles formulations such as that of Salakhutdinov and Hinton (2009)
- Harmonium networks proposed in Smolensky (1986) are essentially equivalent to what are now commonly referred to as restricted Boltzmann machines
- Contrastive divergence was proposed by Hinton (2002)

Bibliographic Notes & Further Reading

Stochastic methods – Boltzmann machines

- The idea of using unsupervised pre-training to initialize deep networks using stacks of restricted Boltzmann machines was popularized by Hinton and Salakhutdinov (2006)
- Salakhutdinov and Hinton (2009) give further details on the use of deep Boltzmann machines and training procedures for deep belief networks, including other nuances for greedy training of deep restricted Boltzman machines
- Neal (1992) introduced sigmoidal belief networks
- Welling et al. (2004) showed how to extend Boltzmann machines to categorical and continuous variables using exponential-family models
- A greedy layer-wise training procedure for deep Boltzmann machines was proposed by Hinton and Salakhutdinov (2006) and refined by Murphy (2012)

Bibliographic Notes & Further Reading

Stochastic methods – Boltzmann machines

- Hybrid supervised and unsupervised learning procedures for restricted Boltzman machines were proposed by McCallum et al. (2005) and further explored by Larochelle and Bengio (2008).
- Vincent et al. (2010) proposed the autoencoder approach to unsupervised pre-training; they also explored various layer-wise stacking and training strategies and compared stacked restricted Boltzmann machines with stacked autoencoders.