Advanced Machine Learning

Script of

PD Dr. rer. nat., Dipl. phys. Ullrich Köthe

at the Heidelberg Collaboratory for Image Processing, Interdisciplinary Center for Scientific Computing
Image Analysis and Learning Group

Transcript of: Manuel Haussmann

15. August 2017

Heidelberg University
Heidelberg Collaboratory for Image Processing
Berliner Str. 43
D-69120 Heidelberg

Contents

1 Lecture 15/04

1.1 Goals

- Find a function $Y = f(X)$, where $X \in \mathbb{R}^D$:
  - $Y \in \mathbb{R}$ or $\mathbb{R}^M$: **Regression**,
  - $Y \in \{1, ..., C\}$[1]: **Classification**[2]

  and learn the desired function f from training data:
  - $\{(X_i, Y_i)\}_{i=1}^N$: **Supervised** (correct answer known),
  - $\{X_i\}_{i=1}^N$: **Unsupervised** (must infer interesting categories).
- The function f stems from a **model class** (predefined, parameterized by θ), i.e. $f(X|\theta)$. The optimal θ are defined by the **loss function** $Loss(X_i, Y_i|\theta) \rightarrow \mathbb{R}$:

$$\text{choose } \theta \text{ such that } \sum_i Loss(X_i, Y_i|\theta) \text{ is minimized.}^3$$

- loss/gain are selected according to the application
- **generalization** vs. **overfitting**: Loss on independent data (test set) may be much bigger than loss on training data.
- predict generalization error:
  - theoretical models (e.g. (Vapnik-Chervonenkis) VC dimension)
  - empirical on independent test data or via cross-validation
- use models that generalize well:
  - simple models ($|\theta| < N$)[4]
  - regularization (restrict the search space for θ)[5]
  - ensembles (combine several classifiers)[6]
  - randomization

[1] C: class count
[2] We will mainly concentrate on classification.
[3] ...or a gain is maximized.
[4] e.g. linear regression
[5] e.g. Lasso
[6] e.g. random forests, boosting

1.2 Notation

feature matrix X of dimension $N \times D$ [7]

instance index $i(i', i_1, i_2)$, $i = 1, ..., N$ and X_i is a row of X

feature index $j(j', j_1, j_2)$, $j = 1, ..., D$ and X_j is a column of X

class index $k = 1, ..., C$ Depending on the algorithm/context we also use $k \in \{0, 1\}$ or $k \in \{-1, 1\}$.

1.3 Linear Models for Classification

- Assume that the elements contained in the training data are i.i.d[8,9], i.e. $(X_i, Y_i) \perp\!\!\!\perp (X_{i'}, Y_{i'})$, $i \neq i'$.
- in general, there are 3 approaches:
  1. Learn a **decision function**: $\hat{Y} = f(X|\theta)$, $f : \mathbb{R}^D \to \{1, ..., C\}$.

     This gives us a *hard decision on class membership* with no confidence estimate.
  2. Learn **class posterior probabilities**: $p(Y = k|X; \theta)$ for all $k : \mathbb{R}^D \to [0, 1]$, s.t. $\sum_k p(Y = k|X; \theta) = 1$[10].

     This approach always implies **1.** with

     $$f(x) = \arg\max_k \; p(Y = k|X; \theta) = \hat{Y}$$

     , "winner takes all" but with an added confidence:

     $p(\hat{Y}|X; \theta) - \max_{k \neq \hat{Y}} p(Y = k|X; \theta)$[11]

     giving us a *soft class membership*.
  3. **Generative model** (can be used to generate new data): learn the per class likelihood + prior probability with Bayes' theorem

     $$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)} \quad \text{where} \quad p(X) = \sum_k p(X|Y = k)p(Y = k)$$

     .

     - Learn $p(Y = k) = \frac{N_k}{N}$ with N_k: Number of class k instances in training data.

[7] With N the instance count and D the feature count.
[8] Independent, identically distributed
[9] If this assumption is violated probabilistic graphical models are a possibility.
[10] Special case $C = 2$: $p(Y = 1|X; \theta) = 1 - p(Y = 0|X; \theta)$
[11] for $C = 2$:

$$p(\hat{Y}|X) - (1 - p(\hat{Y}|X)) = 2p(\hat{Y}|X) - 1 \Leftrightarrow p(\hat{Y}|X) - \tfrac{1}{2}$$

– Learn data likelihood per class $\forall k : p(X|Y = k; \theta_k)$. How are the instances of class k distributed in feature space (= density estimation problem)?

This approach implies **2.** via Bayes' rule and **1.** via "winner takes all".

- **1.** and **2.** are called discriminative models.

Application to linear models: [12]

1.3.1 LDA

- Assume that $p(X|Y = k, \theta_k)$ is a Gaussian distribution for all k with a single joint covariance[13] (otherwise: QDA (see ML1)).

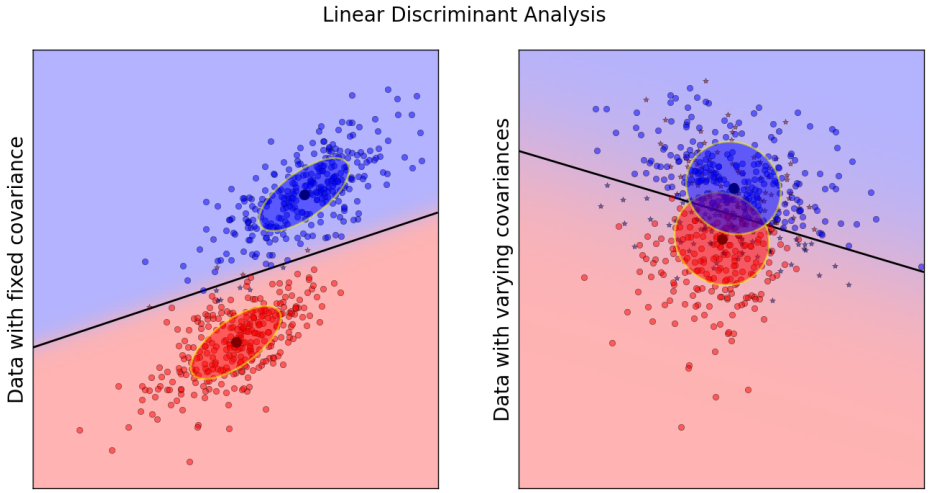


Figure 1.1: Two examples for linear decision boundaries in 2D for the 2 class case of LDA.

- Fit the Gaussians with:

  k different means: $\mu_k = \frac{1}{N_k} \sum_{i:Y_i=k} X_i$

  Total mean: $\mu = \frac{1}{N} \sum_i X_i$

  One joint covariance matrix: $\Sigma = \frac{1}{N} \sum_i (X_i - \mu_{Y_i})^\mathsf{T} (X_i - \mu_{Y_i})$.

- Per class likelihood is given by:

$$p(X|Y = k) = \frac{1}{\sqrt{(2\pi)^D |\Sigma|}} \exp\left(-\tfrac{1}{2}(X - \mu_k)\Sigma^{-1}(X - \mu_k)^\mathsf{T}\right)$$

.

[12] in the order **3.** (linear discriminant analysis (LDA)) **1.** (perceptron, linear support vector machine) **2.** (logistic regression)

[13] It means classes differ only by their location and not by their shape. In the picture we also see an example with varying covariances.

- This leads to a linear posterior if $C = 2$:

$$p(Y = 1|X) = \sigma(X\beta + b)$$

with the *logistic function* $\sigma(t) = (1 + \exp(-t))^{-1}$ and

$$\beta = \Sigma^{-1}(\mu_1^\top - \mu_{-1}^\top), \quad b = -\mu\beta.$$

$\Rightarrow$ Decision rule is defined by:

$$\arg\max_k p(Y = k|X) \Leftrightarrow \hat{Y} = f(x) = \begin{cases} 1, & X\beta + b > 0 \\ -1 & X\beta + b < 0 \end{cases}$$

where $X\beta + b = 0$ is called the *decision boundary*.

- This is a very good model if Gaussian assumption holds (approximately).

1.3.2 Directly learn the decision function

- For simplicity, we augment the data matrix X with a column of 1s $\Rightarrow$ thereby we can absorb b into β.
- we consider $C = 2$, $Y \in \{-1, 1\}$

  Perceptron (Rosenblatt, 1958)

  - If the classifier is always correct, we have $\forall i : Y_i X_i \beta > 0$. $\Rightarrow$ We should pay a penalty, when $Y_i X_i \beta < 0$

$$Loss(X_i\beta, Y_i) = \begin{cases} -Y_i X_i \beta, & Y_i X_i \beta < 0 \\ 0, & Y_i X_i \beta \geq 0 \end{cases} = (-Y_i X_i \beta)_+$$

$$Loss(\beta) = \sum_{i:Y_i \neq \hat{Y}_i} -Y_i X_i \beta = \sum_i (-Y_i X_i \beta)_+$$

    .

  - Perceptron algorithm: gradient descent on loss function

$$\frac{\partial Loss}{\partial \beta} = \sum_{i:Y_i \neq \hat{Y}_i} -Y_i X_i$$

    * choose $\beta^{(0)}$ randomly, learning rate τ
    * repeat until convergence ($t = 1, \ldots, T_{\max}$)

$$\beta(t) = \beta(t-1) + \tau \sum_{i:Y_i \neq \text{sign}(X_i \beta^{(t-1)})} Y_i X_i$$

  - Converges in finitely many steps if training data are linearly separable (zero training error).

2 Lecture 17/04

- example generative model: **LDA**
- example decision function: **Perceptron**
  - better: **linear support vector machine** (**SVM**)
  - disadvantages Perceptron:
    - solution not unique if data are separable, but most solutions generalize badly
    - may not converge if data non-separable (oscillation)
  - solution: require "safety margin" around the decision plane and maximize its size ⇒ **hinge loss**: already pay penalty if classification is correct, but with low

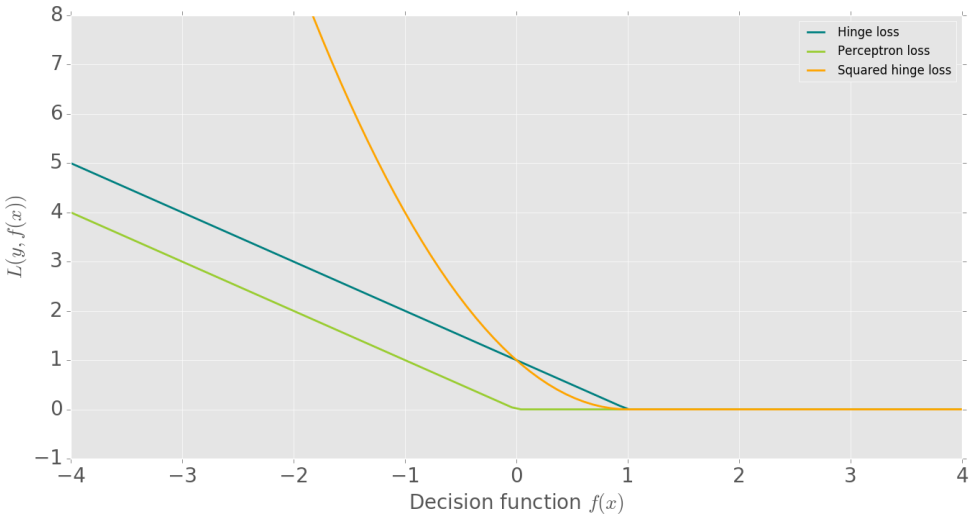


Figure 2.1: Depiction of *perceptron loss* $(\max(0, -Y_i X_i \beta))$ and *hinge loss* $(\max(0, 1 - Y_i X_i \beta))$ functions as well as the *squared hinge loss* $(\max(0, 1 - Y_i X_i \beta)^2)$ for comparison.

confidence

- The maximum margin plane is found by minimizing $||\beta||^2 = \beta^T \beta$ under hinge loss if data are separable.
- For non-separable data: minimization of loss and of $||\beta||^2$ cannot be achieved simultaneously. ⇒ control trade-off by regularization parameter λ

  * SVM objective function

$$\hat{\beta} = \arg\min_{\beta} \frac{\beta^\mathsf{T}\beta}{2} + \frac{\lambda}{N}\sum_i \max(0, 1 - Y_i X_i \beta)$$

    many algorithms

    · standard solver for quadratic programming

    · primal space algorithm: (stochastic) gradient descent, e.g. Pegasos

    · dual space algorithms: sequential minimal optimization (**SMO, LIBSVM**); dual coordinate ascent (**LIBLINEAR**)

  * Advantages:

    · good practical performance, relatively easy training (choose λ by cross-validation)

    · dual formulation can be kernelized (non-linear classifier)

  * Disadvantage: confidence is $|X_i\beta|$, but this cannot be interpreted as probability of being correct

- alternative: learn the posterior probability $p(Y|X;\beta) \Rightarrow$ if $C = 2$: confidence $2p(\hat{Y}|X) - 1$
- The posterior probability of LDA is the logistic function ($\sigma(z) = (1 + \exp(-z))^{-1}$), $p(Y_i = 1|X;\beta) = \sigma(X_i\beta)$.
- choose β according to maximum likelihood rule: maximize likelihood of training data[1]

$$p(\{X_i, Y_i\};\beta) = \prod_i p(Y_i|X_i;\beta) = \prod_i \sigma(Y_i X_i \beta)$$

  define loss negative log-likelihood:

$$-\log p(\{X_i, Y_i\};\beta) = -\sum_i \log \sigma(Y_i X_i \beta) = \sum_i \log(1 + \exp(-Y_i X_i \beta))$$

- Performance improves if we combine the loss with a regularization of β. $\Rightarrow$ regularized logistic regression (LR) objective:

$$\hat{\beta} = \arg\min_{\beta} \frac{\beta^\mathsf{T}\beta}{2} + \frac{\lambda}{N}\sum_i \log(1 + \exp(-Y_i X_i \beta))$$

  $\lambda \to \infty$ gives us the traditional LR without regularization

- Many algorithms can solve this (see *Minka 2003/2007*, *Bottou 2007*), but since there is no closed-form solution, iterative algorithms are needed.

[1] use $1 - \sigma(z) = \sigma(-z)$

1. **gradient descent-type algorithms**: work well because objective is convex

$$\begin{aligned}\frac{\partial loss(X_i, Y_i, \beta)}{\partial \beta} &= \frac{\partial}{\partial \beta} \log(1 + \exp(-Y_i X_i \beta)) \\ &= \frac{1}{1 + \exp(-Y_i X_i \beta)} \exp(-Y_i X_i \beta)(-Y_i X_i^\mathsf{T}) \\ &= \frac{(-1 + 1 + \exp(-Y_i X_i \beta))}{1 + \exp(-Y_i X_i \beta)}(-Y_i X_i^\mathsf{T}) \\ &= (1 - \sigma(Y_i X_i \beta))(-Y_i X_i^\mathsf{T})\end{aligned}$$

   - (**plain/batch**) **gradient descent**: repeat until convergence[2], $t = 1, \ldots, T_{\max}$:

$$\beta^{(t+1)} = \beta^{(t)} + \tau\left(\frac{\lambda}{N} \sum_i \underbrace{\left(1 - \sigma(Y_i X_i \beta^{(t)})\right)}_{> 0 \approx \begin{cases} 0, & \text{if correct} \\ 1, & \text{if false} \end{cases}} Y_i X_i^\mathsf{T} - \beta^{(t)}\right)$$

   - **stochastic gradient descent** (**SGD**): we want to minimize $\mathbb{E}[loss(X_i, Y_i, \beta)]$

     repeat until convergence[3], $t = 1, \ldots, T_{\max}$:

     * choose $i \in 1, \ldots, N$ at random
     * $\beta^{(t+1)} = \beta^{(t)} + \tau_t \left(\lambda(1 - \sigma(Y_i X_i \beta^{(t)}))Y_i X_i - \beta^{(t)}\right)$

   - **SGD with momentum** [4]:

$$\begin{aligned} g^{(t+1)} &= (1 - \mu) g^{(t)} + \mu \underbrace{\left(\lambda(1 - \sigma(Y_i X_i \beta^{(t)}))Y_i X_i - \beta^{(t)}\right)}_{= f^{(t)}} \\ \beta^{(t+1)} &= \beta^{(t)} + \tau_t g^{(t+1)} \end{aligned}$$

     here $\tau_t = \frac{\tau_0}{(1+t)^{3/4}}$, i.e. the rate should have a slower rate, g is initialized to zero.

     * What does the averaging mean?

$$g^{(t+1)} = \sum_{t'=0}^{T'} w_{t'} \cdot f^{(t-t')} \qquad w_{t'} = \mu^{t'} = \exp(-t'/\eta)$$

       i.e. $w_{t'}$ decays exponentially with η being the half life

       $\Rightarrow \eta \approx N$: behavior should be similar to plain GD

[2] τ represents the learning rate
[3] $\tau_t = \frac{\tau_0}{1+t}$
[4] $\mu \in (0, 1)$

– **mini-batch SGD**: choose N_B instances at random, put them into mini-batch B

$$\beta^{(t+1)} = \beta^{(t)} + \tau\left(\frac{\lambda}{N_B}\sum_{i\in B}\left(1-\sigma(Y_iX_i\beta^{(t)})\right)Y_iX_i - \beta^{(t)}\right)$$

– **averaged SGD**: similar to momentum, but smooth β instead of gradient

$$\begin{aligned}\beta^{(t+1)} &= \beta^{(t)} + \tau_t\left(\lambda(1-\sigma(Y_iX_i\beta^{(t)}))Y_iX_i^\mathsf{T} - \beta^{(t)}\right)\\ \overline{\beta}^{(t+1)} &= (1-\mu)\overline{\beta}^{(t)} + \mu\beta^{(t+1)}\end{aligned}$$

– **stochastic averaged gradient SAG**:

$$\begin{aligned}g_i^{(t+1)} &= \begin{cases} g_i^{(t)}, & i \neq i' \\ \lambda(1-\sigma(Y_{i'}X_{i'}\beta^{(t)}))Y_{i'}X_{i'} - \beta^{(t)}, & i = i'\end{cases}\\ g^{(t+1)} &= \frac{1}{N}\sum_i g_i^{(t+1)} = g^{(t)} + \frac{g_{i'}^{(t+1)} - g_{i'}^{(t)}}{N}\\ \beta^{(t+1)} &= \beta^{(t)} + \tau_t g^{(t+1)}\end{aligned}$$

3 Lecture 22/04

3.1 Algorithms for Logistic Regression

- objective:
$$\min_\beta \frac{\beta^\top \beta}{2} + \frac{\lambda}{N}\sum_i (1 + \exp(-Y_i X_i \beta))$$
- gradient descent algorithm, stochastic GD
- Newton-type algorithms in **primal** and **dual space**
- reminder: **Newton-Raphson-algorithm**: optimize nonlinear function $f(a)$, Taylor series expansion around current guess:
$$f(a^{(t)} + \Delta a) \approx f(a^{(t)}) + f'(a^{(t)})\Delta a + \frac{f''(a^{(t)})}{2}\Delta a^2 \to \min$$
$$\frac{\partial d}{\partial \Delta a} f(a^{(t)} + \Delta a) \approx f'(a^{(t)}) + f''(a^{(t)})\Delta a \stackrel{!}{=} 0$$
$\Rightarrow \Delta a = -\frac{f'(a^{(t)})}{f''(a^{(t)})}$
if a is a vector: $\Delta a = -H^{-1}|_{a^{(t)}} \nabla f|_{a^{(t)}} \Rightarrow$ update: $a^{(t+1)} = a^{(t)} + \Delta a$, need gradient
$$\min_\beta \frac{\beta^\top \beta}{2} + \frac{\lambda}{N}\sum_i \underbrace{(1 - \sigma(Y_i X_i \beta))}_{\sigma(-Y_i X_i \beta)}(-Y_i X_i^\top)$$
Hessian: $\frac{\partial^2 Loss}{\partial \beta^2} = I - \frac{\lambda}{N}(-\underbrace{\sigma'(Y_i X_i \beta))}_{\sigma(t)(1-\sigma(t))} \underbrace{Y_i^2}_{=1} X_i^\top X_i$
$$\frac{\partial^2 Loss}{\partial \beta^2} = I + \frac{\lambda}{N}\sum_i \sigma(X_i\beta)(1 - \sigma(X_i\beta)) X_i^\top X_i = I + X^\top W X$$
where $W = \frac{\lambda}{N}\text{diag}\left(\sigma(X_i\beta)(1 - \sigma(X_i\beta))\right)$ is a $N \times N$ matrix
- simplify gradient using W[1]:
$$\frac{\lambda}{N}(1 - \sigma(Y_i X_i \beta)) Y_i X_i^\top = \frac{\lambda}{N}\sum_i \sigma(X_i\beta)(1 - \sigma(X_i\beta))\frac{Y_i}{\sigma(Y_i X_i \beta)} X_i^\top$$
$$= X^\top W \tilde{Y}$$

[1] $\tilde{Y} = \frac{Y_i}{\sigma(Y_i X_i \beta)}$ is $N \times 1$

- insert into Newton-Raphson update

$$\begin{aligned}\beta^{(t+1)} &= \beta^{(t)} + (I + X^\mathsf{T} W^{(t)} X)^{-1}(X^\mathsf{T} W^{(t)} \tilde{Y}^{(t)} - \beta^{(t)}) \\ &= (I + X^\mathsf{T} W^{(t)} X)^{-1} \left((I + X^\mathsf{T} W^{(t)} X)\beta^{(t)} + X^\mathsf{T} W^{(t)} \tilde{Y}^{(t)} - \beta^{(t)}\right) \\ &= (I + X^\mathsf{T} W^{(t)} X)^{-1} \left(X^\mathsf{T} W^{(t)} (X\beta^{(t)} + \tilde{Y}^{(t)})\right) \\ &= (I + X^\mathsf{T} W^{(t)} X)^{-1} X^\mathsf{T} W^{(t)} Z^{(t)}\end{aligned}$$

  where $Z^{(t)} = X\beta^{(t)} + \tilde{Y}^{(t)}$

- this is the formal solution of the **weighted ridge regression problem**

$$\beta^{(t+1)} = \arg\min_{\beta} (Z^{(t)} - X\beta)^\mathsf{T} W^{(t)} (Z^{(t)} - X\beta) + \frac{||\beta||^2}{2}$$

  $\Rightarrow$ **Iterated Reweighted Least-Squares Algorithm** (IRLS)

  repeat until convergence, $t = 1, \ldots T_{\max}$

  - compute $W^{(t)}$ and $Z^{(t)}$
  - $V^{(t)} = (W^{(t)})^{1/2}$, $\tilde{X}^{(t)} = XV^{(t)}$, $\tilde{Z}^{(t)} = Z^{(t)}V^{(t)}$
  - use a standard solver to $\min_\beta (\tilde{Z}^{(t)} - \tilde{X}^{(t)}\beta)^2 + \frac{||\beta||^2}{2}$

- faster than GD or SGD on small datasets

- even faster: use fast approximation of the Hessian $\Rightarrow$ "quasi-Newton", e.g. BFGS (**Broyden–Fletcher–Goldfarb–Shanno**) algorithm

- Newton in **dual space**

- in the **primal space**, we approach the optimum from above: all $\beta^{(t)}$ are upper bounds of $Loss(\beta^*) \leq Loss(\beta^{(t)})$

- the **dual** problem approaches the optimum from below: $\forall \alpha^{(t)} : DualLoss(\alpha^*) \geq DualLoss(\alpha^{(t)})$

- In difficult optimization problems, one often brackets the (unknown) global optimum between a primal upper bound and a dual lower bound. The difference between the bounds is the "**duality gap**".

- If the dual bound is tight, the duality gap is zero, and primal and dual solutions agree, e.g. LR.

- requirements on dual for LD:
  - tight lower bound
  - simple in β, s.t. it can be solved in β in closed form

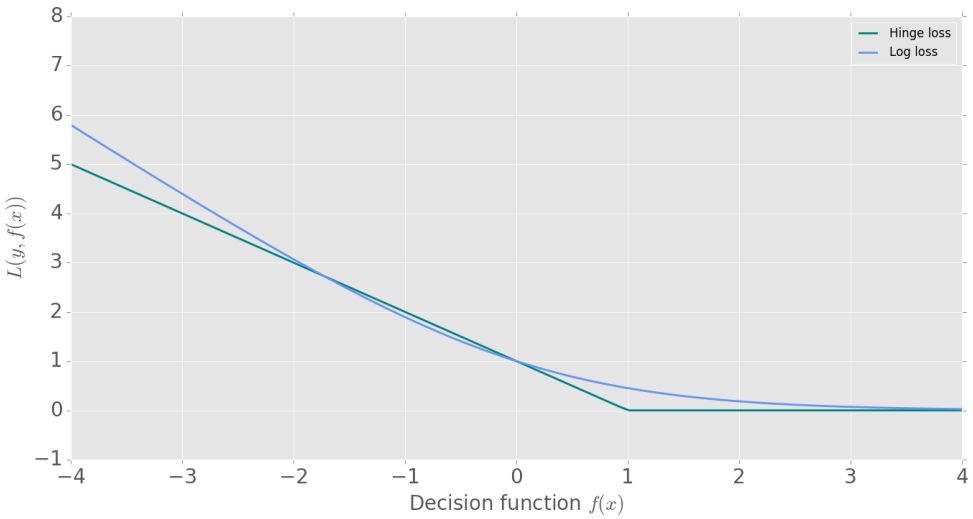


Figure 3.1: Plot showing the logistic loss function in comparison to the above introduced hinge loss.

- obvious choice: tangents of loss, parameterized by their slope

$$\log(1 + \exp(-t)) \geq -\alpha t - \alpha \log \alpha - (1 - \alpha) \log(1 - \alpha) \qquad \alpha \in [0, 1]$$

$\Rightarrow Lagrangian(\beta, \alpha) = \frac{\beta^\mathsf{T}\beta}{2} + \frac{\lambda}{N} \sum_i (-\alpha_i Y_i X_i \beta - \alpha_i \log \alpha_i - (1 - \alpha_i) \log(1 - \alpha_i))$, replace loss with its lower bound s.t. $\alpha_i \in [0, 1]$[2]

$$\frac{\partial Lagrangian}{\partial \beta} = \beta + \frac{\lambda}{N} \sum_i -\alpha_i Y_i X_i^\mathsf{T} \stackrel{!}{=} 0$$
$$\Rightarrow \beta = \frac{\lambda}{N} \sum_i \alpha_i Y_i X_i^\mathsf{T}$$

- insert into Lagrangian:

$$DualLoss(\alpha) = -\frac{\lambda^2}{2N^2} \sum_{i,i'} \alpha_i \alpha_{i'} Y_i Y_{i'} X_i X_{i'}^\mathsf{T} - \frac{\lambda}{N} \sum_i (\alpha_i \log \alpha_i + (1 - \alpha_i) \log(1 - \alpha_i))$$

- dual optimization problem: $\alpha^* = \max_\alpha DualLoss(\alpha)$ s.t. $\alpha_i \in [0, 1]$

[2] with $0 \log 0 := 0$

- solution via *coordinate-wise Newton* (one α_i at a time)

$$\begin{aligned}
\frac{\partial Dual}{\partial \alpha_i} &= -\frac{\lambda^2}{N^2} Y_i X_i \sum_{i'} (\alpha_{i'} Y_{i'} X_{i'}) - \frac{\lambda}{N} \left(\log \alpha_i + \frac{\alpha_i}{\alpha_i} - \log(1-\alpha_i) - \frac{(1-\alpha_i)}{(1-\alpha_i)} \right) \\
&= -\frac{\lambda^2}{N^2} Y_i X_i \sum_{i'} (\alpha_{i'} Y_{i'} X_{i'}) - \frac{\lambda}{N} \log \frac{\alpha_i}{1-\alpha_i} \\
\frac{\partial^2 Dual}{\partial \alpha_i^2} &= -\frac{\lambda^2}{N^2} Y_i^2 X_i X_i^\top - \frac{\lambda}{N} \frac{1}{\alpha_i} - \frac{\lambda}{N} \frac{1}{1-\alpha_i} \\
&= -\frac{\lambda^2}{N^2} X_i X_i^\top - \frac{\lambda}{N} \frac{1}{\alpha_i (1-\alpha_i)}
\end{aligned}$$

- Dual coordinate-wise Newton algorithm:
  - choose $\alpha^{(0)}$ randomly
  - repeat until convergence, $t = 1, \ldots, T_{\max}$
    - $\alpha_i^{(t+1)} = \alpha_i^{(t)} - \frac{\frac{\partial Dual}{\partial \alpha_i}}{\frac{\partial^2 Dual}{\partial \alpha_i^2}}$
    - clip at $[0, 1]$
  - LIBLINEAR implements a slightly improved version (numerically more stable)
  - seems to be the fastest algorithm for large N

3.2 Why is SGD fast for large N?

- 3 sources of error:
  1. **modeling error** ε_{mod}: How far away is the (unknown) best model in our model family from the truth [3]?
  2. **estimation/generalization error** ε_{est}: How far away is our empirical optimum (from finite training set) from the theoretical (from infinite data)[4]?
  3. **optimization error** ε_{opt}: How far away is our solution (after finitely many iterations) from the true optimum (after infinitely many iterations)[5]?
- our choice of algorithm influences 1. and 2., $\varepsilon = \varepsilon_{\text{est}} + \varepsilon_{\text{opt}}$
- Both errors should decrease at about the same rate, otherwise our efforts on minimizing one of them are useless.

[3] can be reduced by a larger model family
[4] can be reduced by a smaller model family or more training data
[5] can be reduced by more iterations

- numerical analysis: $\varepsilon_{\text{est}} \in O\left(\frac{\log N}{N}\right)$ best case, $O\left(\sqrt{\frac{\log N}{N}}\right)$ worst case

$$\varepsilon \sim \varepsilon_{\text{est}} \sim \varepsilon_{\text{opt}} \sim \frac{\log N}{N} \quad \left(\text{or } \sqrt{\frac{\log N}{N}}\right)$$

$$N \sim \frac{1}{\varepsilon} \log N$$
$$\log N \sim \log \frac{1}{\varepsilon} + \underbrace{\log \log N}_{\approx 0}$$
$$N \sim \frac{1}{\varepsilon} \log \frac{1}{\varepsilon} \quad \text{(best case)}$$
$$N \sim \frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon} \quad \text{(worst case)}$$

Algorithm	**Time per step**	**Steps to accuracy**	**Time to accuracy ε_{opt}**	**Time to total accuracy**
SGD + Dual	$O(D)$	$O(\frac{1}{\varepsilon_{\text{opt}}})$	$O(\frac{D}{\varepsilon_{\text{opt}}})$	$O(\frac{D}{\varepsilon})$
GD	$O(ND)$	$O(\log \frac{1}{\varepsilon_{\text{opt}}})$	$O(DN \log \frac{1}{\varepsilon_{\text{opt}}})$	$O(D\frac{1}{\varepsilon^2}(\log \frac{1}{\varepsilon})^2)$
Newton	$O(D^2N)$	$O(\log \log \frac{1}{\varepsilon_{\text{opt}}})$	$O(D^2N \log \log \frac{1}{\varepsilon_{\text{opt}}})$	$O(\frac{D^2}{\varepsilon^2} \log \frac{1}{\varepsilon} \log \log \frac{1}{\varepsilon})$

- Fazit: on the long run SGD wins

4 Lecture 24/04

4.1 Neural Networks

- **linear classifiers**: only work when the data are approximately linearly separable, otherwise we need a **nonlinear method**
- two approaches to construct **nonlinear methods** from **linear** ones:
  1. **augment the feature space**
     - measure more properties
     - compute new features as nonlinear functions of the existing ones (e.g. **kernel SVM**) ($\Rightarrow$ later)
  2. **non-linearly combine several linear classifiers**
     - **boosting**: $\hat{y} = \text{sign}(\sum_l \alpha_l f_l(X))$, $f_l(X)$: linear classifiers $\text{sign}(X\beta_l + b_l)$

       training: greedily add one classifier at a time, minimize exponential loss (ML1)
     - **decision tree**: hierarchy of linear classifiers

       training: greedily maximize purity (minimize Gini impurity) (ML1)
- neural networks (NN) combine the ideas in **2.** : connect linear classifiers in parallel (*"layers"*) and layers in series (*"multi-layer"* or *"deep"* if ≥ 4)
- history:
  - 1940/50s: first neuron models (Hebb, McCulloch/Pitts) and idea of multi-layered architectures inspired by brain research and meant to explain the brain
  - 1958: Perceptron and multylayer perceptron (Rosenblatt): first working training algorithm (gradient descent on centered hinge loss), but no good algorithm for multi-layer training
  - 1969: book by Papert & Minsky: proved limitations of single layered perceptron (cannot solve the XOR-problem) and they conjectured (falsly) that multi-layered architectures are not much better.

    $\Rightarrow$ first death
  - 1986: Rumelhardt & Hinton: popularized backpropagation training for multi-layer NN; first practical training algorithm for multi-layer NN $\Rightarrow$ first rebirth
  - ... 1995:

    - proof of universial approximation capability
    - solved several difficult toy problems

  but:

    - proof that exact training is difficult (NP hard in worst case)

      ⇒ need training heuristics, but they are very difficult to apply effectively ("black art")[1]
    - success on real problems was limited
    - discovery of SVM, boosting and random forests (much better on practical problems)

  ⇒ second death

- 2006:
    - much larger training sets (less overfitting)
    - GPU-based parallelization (100× speed-up)
    - Hinton: discovery of unsupervised pre-training

      ⇒ second rebirth
    - NN won several prestigious benchmark competitions
    - training was still difficult
- 2010 ...:
    - interesting ideas to simplify training (dropout, dropconnect, ReLU activation, max out, ...)
    - simpler architectures (fewer layers)

  ⇒ NN start to get interesting

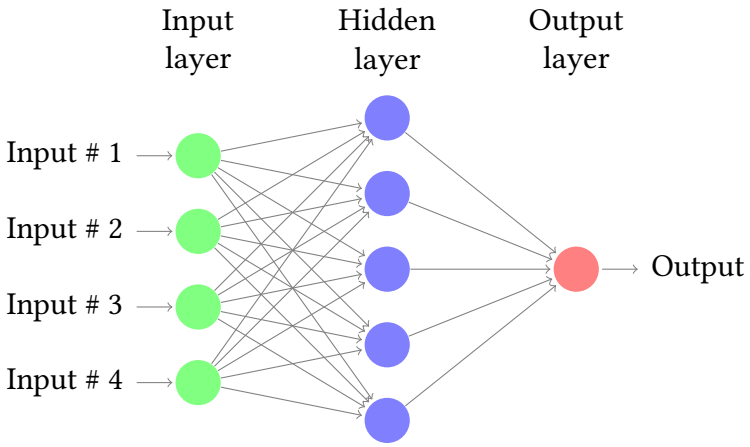
4.2 NN architecture

- each neuron has **arbitrary many inputs** and a **single output**
- originally: neuron computes a **weighted sum** of the inputs and "fires" if a threshold is exceeded (*threshold activation function*), inspired by the brain
- today: generalize for arbitrary **activation functions**:

$$Z_i = \varphi(X_i\beta + \beta_0)$$

  Z_i is the **response**, φ the **function**, X_i the **features**, β the **weights** and β_0 the **bias** [2]

[1] "Training is easy as long as you let Hinton do it."
[2] Usually, the bias is absorbed into β.



  - activation functions motivated by brain research: step function, sign function ⇒ threshold "on-off" behavior
  - activation functions motivated by training algorithms: logistic function[3], hyperbolic tangent[4]

    ⇒ smooth versions of step & sign functions
  - modern choices: piecewise linear functions: hinge function [5] (usually called *ReLU-"rectified linear unit"*); maxout activation (⇒ later)

    almost everywhere differentiable ⇒ sparse activation patterns, better generalization

- a neuron with sigmoid activation is simply logistic regression ⇒ several neurons needed
- "network architecture" = how many neurons and how to combine them (must be fixed by network designer)
- in a *"feed forward network"* all connections are directed from input → output ⇒ NN is a DAG (directed acyclic graph)
- opposite: *recurrent network*: information flows forward and backward (popular in time series analysis and not treated in this lecture)
- authors cannot agree on how to count layers:
  1. count input and output as layers ⇒ L = #hidden + 2

     popular notation: $D - H_1 - H_2 - \ldots - M$ to specify number of neurons in each layer
  2. most do not count the inputs
  3. to avoid confusion, some use different terms:

[3] $\varphi_t = \sigma(t)$
[4] $\varphi(t) = \tanh(t) = 2\sigma(2t) - 1$
[5] $\phi(t) = \max(0, t)$

  - stages = # transitions between layers
  - hidden layers

  $\Rightarrow$ we take approach **2.**, but start counting at 0, input $l = 0$, output $l = L$

- notation:

  number of layers L;

  layer index $l = 0, 1, ..., L$;

  number of inputs/features: $D, j = 0, ..., D$[6];

  number of outputs: $M, m = 1, ..., M$;

  number of hidden neurons in layer l: H_l, if there is only 1 hidden layer: H, $h = 0, ..., H$[7];

  $\mathbf{B}$[8]: 3-dimensional array of weights;

  $\mathbf{B_l}$ matrix of weights between layer $(l-1)$ and l;

  $\mathbf{B_{lh}}$: column vector [9] input weights of neuron h in layer l;

  $\mathbf{B_{lhj}}$: single weight from neuron j in layer $(l-1)$ to neuron h in layer l;

  output (row) vector of all neurons in layer l: Z_l;

  output of neuron h **in layer** l: Z_{lh};

  φ_l: activation functions in layer l (all identical)

- example: 2-layer NN with 1 output neuron:

$$\hat{Y}_i = Z_{21} = \varphi_2 \left(\sum_{h=0}^{H_1} B_{21h} \cdot \varphi_1 \left(\sum_{j=0}^{D} B_{1hj} X_{ij} \right) \right)$$

[6] 0 being the bias neuron
[7] 0 being the bias neuron
[8] actually capital β
[9] (= β in a single linear classifier)

5 Lecture 29/04

5.1 Theoretical Capabilities of NN

- A 1-layer NN is just a set of independent LR[1] instances (or linear regression).

  $\Rightarrow$ to be better, we need hidden layers

- The VC-dimension of a linear classifier with D features is $D + 1$. $N_{VC} = D + 1$ is the largest training set where zero training error is always achievable, regardless of the labels if C = 2 (ML1).

  $\Rightarrow$ we can always reduce training error by adding more hidden neurons but beware of overfitting

- consider the first stage: each hidden neuron in the first layer splits the feature space into two half-spaces.
  - their union partitions the feature space into convex cells (*"polytopes"*)
  - encode the cells by a binary number according to the side of each hyperplane
  - projects the feature space onto the corners of a H_1-dimensional hyper cube
- these 2 properties can be used to construct a 2-layer (difficult) or 3-layer (easy) network with zero training error $\Rightarrow$ exercise
- universal approximation theorems (various versions): NN can learn arbitrary functions
  - e.g. Hornik, 1991 is one of the most general
    * regression setting (includes the classification setting via regression of the posterior probabilities)
    * one hidden layer, and one neuron with linear activation
    * assume that the activation function of hidden neurons is continuous, bounded and non-constant on every compact subset of $\mathbb{R}^D$

      $\Rightarrow$ output[2]: $\hat{f}(x) = \sum_{h=0}^{H_1} B_{21h}\phi_1(B_{1h}Z_0)$
    * consider function space L_p, i.e. the set of all functions s.t.

$$L_p = \left\{ f : ||f||_p = \left(\int_{\mathbb{R}^D} |f(x)|^p dX \right)^{1/p} \right\}$$

[1] logistic regression

[2] Notation update: $Z_0 = [1\ X^\mathsf{T}]^\mathsf{T}$, i.e. a column vector

* THEOREM: for every function $f \in L_p$, there exist parameters H_1, B_1, B_2 such that

$$\left(\int_\rho |f - \hat{f}|^p dx \right)^{1/p} < \varepsilon$$

where $\rho \in \mathbb{R}^D$ arbitrary compact substract, $\varepsilon > 0$ arbitrary small

$\Rightarrow$ in principle, one hidden layer is sufficient[3]

5.2 The Practice

- intuition: Transform the data via several layers until they cluster cleanly into few easily separable clusters.
- prediction:

  input layer $Z_0 \in [(D+1) \times 1]$

  hidden layer $\tilde{Z}_l = B_l Z_{l-1} \in [H_l \times 1] = [H_l \times H_{l-1}][H_{l-1} \times 1]$ layer $(l-1)$ to l

  $Z_l = \varphi_l(\tilde{Z}_l)$ pointwise $[H_l \times 1]$

  output layer L depends on the application

  regression linear activation $\hat{Y}_i = \tilde{Z}_L^\mathsf{T} (= Z_L^\mathsf{T}) \in [1 \times M]$

  decision rule $\hat{y} = \arg\max_k \tilde{Z}_{Lk}$, $k \in 1, \ldots, C$ number of classes

  2-class posterior $p(\hat{Y}_i = 1 | X_i) = Z_L = \sigma(\tilde{Z}_L)$, $k \in \{0, 1\}$, scalar output

  multi-class posterior $(C \geq 2) \Rightarrow k \in \{1, \ldots, C\}, \forall k : p(\hat{Y}_i = k | X_i) = Z_{Lk} = \frac{e^{\tilde{Z}_{lk}}}{\sum_{k'} e^{\tilde{Z}_{Lk'}}}$[4],

5.3 Backpropagation

fancy name for gradient descent training of the weights

- Define Loss (application specific $\Rightarrow$ later) and its derivatives

$$\delta_l := \frac{\partial Loss}{\partial Z_l} = \frac{\partial Loss}{\partial Z_{l+1}} \frac{\partial Z_{l+1}}{\partial Z_l} = \delta_{l+1} \frac{\partial Z_{l+1}}{\partial Z_l}$$

- Derivatives w.r.t weights:

$$\frac{\partial Loss}{\partial B_l} = \underbrace{\frac{\partial Loss}{\partial Z_l}}_{\delta_l} \frac{\partial Z_l}{\partial \tilde{Z}_l} \frac{\partial \tilde{Z}_l}{\partial B_l} = \underbrace{\delta_l \varphi_l'(\tilde{Z}_l)}_{[H_l \times 1]} \underbrace{Z_{l-1}^\mathsf{T}}_{[1 \times H_{l-1}]} \in [H_l \times H_{l-1}]$$

[3] The problem with this theorem is, that it only gives a statement of existence with no indication on how to construct $\hat{f}$.

[4] known as "soft-max function" a generalization of the sigmoid function

- common activation functions:

  $\sigma'(\tilde{Z}_l) = Z_l(1 - Z_l)$,

  $\tanh'(\tilde{Z}_l) = 1 - Z_l^2$,

  ReLU $\varphi(t) = \max(0, t)$,

  $$\varphi'(\tilde{Z}_l) = \text{step}(\tilde{Z}_l) = \begin{cases} 1 & \tilde{Z}_l > 0 \\ 0 & \text{else} \end{cases}$$

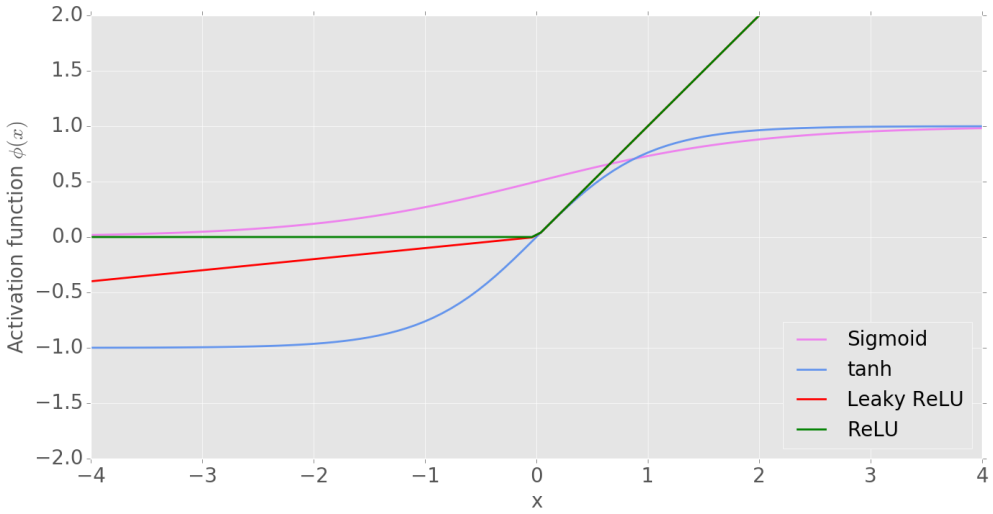


Figure 5.1: Some depictions of widely used activation functions.

- derivatives w.r.t. to previous layers:

$$\frac{\partial Z_{l+1}}{\partial Z_l} = \varphi'_{l+1}(\tilde{Z}_{l+1}) \frac{\tilde{Z}_{l+1}}{\partial Z_l} = B_{l+1}^{\mathsf{T}} \varphi'_{l+1}(\tilde{Z}_{l+1})$$

- backpropagation algorithm:
  - init $\delta_L = \frac{\partial Loss}{\partial Z_l}$, $\tilde{\delta}_L = \delta_L \varphi'_L(\tilde{Z}_L)$
  - for $l = L, \ldots, 1$:

$$\begin{aligned} \Delta B_l &= \tilde{\delta}_l Z_{l-1}^{\mathsf{T}} \\ \delta_{l-1} &= B_l^{\mathsf{T}} \tilde{\delta}_l \\ \tilde{\delta}_{l-1} &= \delta_l \varphi'_l(\tilde{Z}_l) \end{aligned}$$

  - $B_l^{(t+1)} = B_l^{(t)} - \tau \Delta B_l^{(t+1)}$

5.4 Loss functions depend on application:

regression: $Loss = \frac{1}{2}(Y - \hat{Y})^2 \Rightarrow \delta_L = \hat{Y} - Y = \tilde{\delta}_L$ (because $\varphi_L(t) = t$)

2-class posterior: $Loss = -Y \log \hat{p} - (1 - y) \log(1 - \hat{p})$

$$\delta_L = \frac{\partial Loss}{\partial Z_l} = \begin{cases} -1/Z_l, & if y = 1 \\ 1/(1 - Z_L), & if y = 0 \end{cases}, \frac{\partial Z_L}{\partial \tilde{Z}_L} = Z_L(1 - Z_L)$$

$$\tilde{\delta}_L = \delta_L \frac{\partial Z_L}{\partial \tilde{Z}_L} = \begin{cases} Z_L - 1, & if Y = 1 \\ Z_L, & if Y = 0 \end{cases}$$

- regularization: $RLoss = Loss + Regularizer$. Popular Regularizers: L_2, L_1

6 Lecture 06/05

6.1 NN training algorithm

- Initialization:
  - choose network architecture (# layers, # neurons) and learning rate (schedule)
  - init the weights $B^{(0)}$
- repeat until convergence, $t = 1, \ldots, T_{\max}$
  - $\Delta B^{(t)} = 0$
  - for i in instances$^{(t)}$ = $\begin{cases} \text{a single random } i \text{ (SGD)} \\ \text{a random mini-batch} \\ \text{full training set (GD)} \end{cases}$
    - forward step: prediction

      $$Z_0 = [1 \;\; X_i^\mathsf{T}]^\mathsf{T}$$

      for $l = 1, \ldots, L$

      $$\tilde{Z}_l = B_l^{(t-1)} - Z_{l-1}$$
      $$Z_l = \varphi_l(\tilde{Z}_l)$$

    - compute loss gradient acc. to application

      $$\tilde{\delta}_L = \frac{\partial Loss(Z_L, Y_i)}{\partial \tilde{Z}_L}$$
      $$\Delta B_L^{(t)} += \tilde{\delta}_L Z_{L-1}^\mathsf{T}$$

    - backward sweep ("error (gradient) backpropagation")
      for $l = L-1, \ldots, 1$

      $$\delta_l = \left(B_{l+1}^{(t-1)}\right)^\mathsf{T} \tilde{\delta}_{l+1}$$
      $$\tilde{\delta}_l = \delta_l * \varphi_l'(\tilde{Z}_l) \qquad \text{pointwise}$$
      $$\Delta B_l^{(t)} += \tilde{\delta}_l \mathbb{Z}_{l-1}^\mathsf{T}$$

  - weight update:

    $$B^{(t)} = B^{(t-1)} - \tau_t \Delta B^{(t)} + \mu(B^{(t-1)} - B^{(t-2)})$$

6.2 Classical Tricks to make this work

- regularization, e.g. $\frac{\lambda}{2N}||B||_F^2$, $\frac{\lambda}{L}||B||_1$, max-norm regularization B_{lh} the weights for neuron h at layer l: $B_{lh} = B_{lh}\frac{\min(C,||B_{lh}||_2)}{||B_{lh}||_2}$ ($||B_{lh}||_2 \leq C$, always $C = 3,\ldots,4$ to keep the expected input in the non-constant range of the sigmoids)

  $\sigma'(t) = \sigma(t)(1-\sigma(t)) \approx 0$ if $\sigma(t) \approx 0,1$

  $\tanh'(t) = 1 - \tanh^2(t) \approx 0$ if $t \approx \pm 1 \Rightarrow$ no gradient is propagated when the nonlinearity is saturated

- weight initialization: init such that neurons are not saturated at the beginning
  - standardize features (zero mean, unit variance)
  - assume that the neuron activations are zero mean and unit variance[1]
  - initialize weights with zero mean and variance s^2
  - input properties of next layer neuron

$$\mathbb{E}(B_{lh}Z_{l-1}) = \mathbb{E}(B_{lh})\mathbb{E}(Z_{l-1}) = 0$$

$$\text{var}(B_{lh}Z_{l-1}) = \text{var}(B_{lh})\text{var}(Z_{l-1}) = s^2 \sum_{h'=0}^{H_{l-1}} \underbrace{\text{var}Z_{l-1,h'}}_{=1} = (H_{l-1}+1)s^2$$

  - for gradient training to work, $\mathbb{E}(B_{lh}Z_{l-1}) \pm \text{std}(B_{lh}Z_{l-1})$ should not be in the saturated region.

    $\Rightarrow \sqrt{H_{l-1}+1}s \leq 1 \ (\leq 2)$, solve for $s = \frac{1}{\sqrt{H_{l-1}+1}}$

    $\Rightarrow$ init $B_{lh} \sim \mathcal{N}(0,(H_{l-1}+1)^{-1})$ or $\sim \mathcal{U}\left(-\sqrt{\frac{3}{H_{l-1}+1}}, \sqrt{\frac{3}{H_{l-1}+1}}\right)$

    always set the $B_{lh0} = 0$ (weight of bias neuron)

    variant $s = \frac{1}{\frac{\sqrt{H_{l-1}+H_{l+1}}}{2}}$

- optimization algorithms
  - plain gradient descent ("batch training")
  - stochastic gradient descent ("online training")
  - mini-batch SGD

    $\Rightarrow$ need to adjust learning rate and momentum ($\rightarrow$ later)
  - methods that automatically adjust the step size
  - usual suspects:
    - Newton, quasi Newton (BFGS), conjugate gradient with line search
    - RPROP:

[1] only assume this for the weight initialization... in general it is wrong

· idea: adjust the log of the training rate by gradient descent $\log(\theta_t) = \log(\tau_{t-1}) + \Delta_t$

⇒ multiplicative update on τ, after some math and simplifications

$$\eta_{lhj}^{(t)} = \begin{cases} \eta^+ (= 1.25), & \text{if } \nabla B_{lhj}^{(t)} \nabla B_{lhj}^{(t-1)} > 0 \\ \eta^- (= 0.7) & \text{else} \end{cases}$$

$$\tau_{t,lhj} = \min\left(\tau_{\max}, \max\left(\tau_{\min}, \tau_{t-1,lhj}\eta_{lhj}^{(t)}\right)\right)$$

$$\Delta B_{lhj}^{(t)} = \Delta B_{lhj}^{(t-1)} - \tau_{t,lhj}\text{sign}(\nabla B_{lhj}^{(t)})$$

· converges very quickly or diverges

· use with large minibatches

- termination criterion: training easily overfits ⇒ keep a separate validation set and monitor the validation error

  ⇒ stop when validation error starts to go up

- learning rate and momentum for GD and SGD [Wilson & Martinez 2003]
  - we want $\Delta B^{(t)} = -\tau_t \mathbb{E}\left[\left.\frac{\partial Loss}{\partial B}\right|_{t-1}\right]$, finite data estimate:

$$\mathbb{E}\left[\frac{\partial Loss}{\partial B}\right] = \frac{1}{N_B}\sum_{i \in \text{Batch}} \left.\frac{\partial Loss_i}{\partial B}\right|_{t-1}$$

  - for the full GD: Batch = training set, $N_B = N$ ⇒ get accurate estimate of $\mathbb{E}[grad]$ at cost $O(N)$
  - SGD: Batch = single instance, $N_B = 1$ ⇒ inaccurate estimate at cost $O(1)$
  - minibatch is between these extremes
  - rule of thumb: the more accurate $\mathbb{E}[grad]$ the bigger we can choose τ. $\tau_{GD} \approx \sqrt{N}\tau_{SGD}$ ⇒ the time for equal progress in GD is $\sqrt{N}$ longer than SGD because of $O(N)$.
  - learning rate schedules:
    * keep the learning rate constant
    * divide $\tau \to \tau/10$ when learning stalls (2x)
    * $\tau_t = \frac{\tau_0}{1+t/t_0}$

7 Lecture 08/05

7.1 RPROP

- normal update of a single weight: $B_{lhj}^{(t)} = B_{lhj}^{(t-1)} - \tau \Delta B_{lhj}^{(t)}$
- give each weight an individual training rate τ_{lhj} and train it via GD: $\log \tau_{lhj}^{(t)} = \log \tau_{lhj}^{(t-1)} + \Delta(\log \tau)_{lhj}^{(t)} \Rightarrow$ multiplicative update in τ itself

  $\Rightarrow$ update rule $B_{lhj}^{(t)} = B_{lhj}^{(t-1)} - \tau_{lhj}^{(t)} \Delta B_{lhj}^{(t)}$.
- after some math and approximations, we find: $\Delta B_{lhj}^{(t)} = \text{sign}\left(\frac{\partial Loss}{\partial B_{lhj}}\Big|_t\right)$

  $\Rightarrow$ the gradient w.r.t B only determines the step direction not the length
- step length is completely absorbed into $\tau_{lhj}^{(t)}$

$$\tau^{(t)} = \max(\tau_{\min}, \min(\tau_{\max}, \tau_{lhj}^{(t-1)} \eta_{lhj}^{(t)}))$$

$$\eta_{lhj}^{(t)} = \begin{cases} \eta^+ (= 1.25), & \text{if } \nabla B_{lhj}^{(t)} \nabla B_{lhj}^{(t-1)} > 0 \\ \eta^- (= 0.7) & \text{else} \end{cases}$$

with $\tau_{\min} = 10^{-7}$, $\tau_{\max} = 10^{-2}$

7.2 Dropout [Srivastava & Hinton 2012]

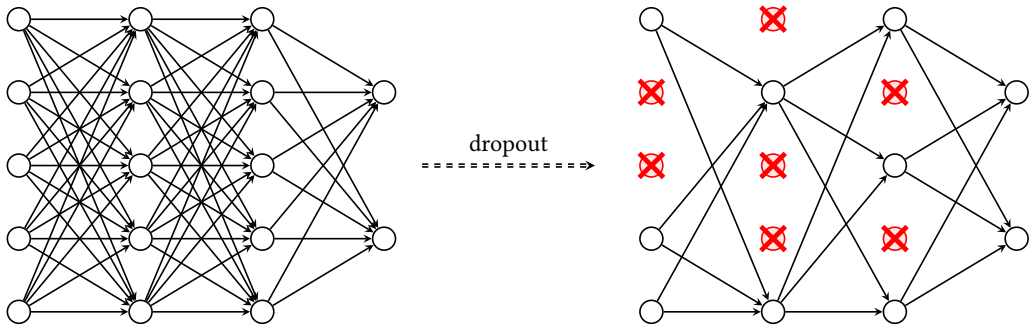


Figure 7.1: Depiction of a deep neural network demonstrating the influence of applying dropout with $p = 0.5$.

- new regularization technique (breakthrough), similar to the randomization in decision trees ⇒ random forest
- idea: randomly switch-off part of the neurons in each training step.

  ⇒ forward sweep:
  for $l = 1, \ldots, L$:

  $$r_{l-1} \sim Bernoulli(p)^{H_{l-1}+1} \qquad [(H_{l-1} - 1) \times 1]$$
  $$\tilde{Z}_l = B_l(r_{l-1} * Z_{l-1}) \qquad \text{pointwise}$$

  keep neurons with probability p.
- backpropagation only on the subnetwork of active neurons (weights going in/out of a inactive neuron are not changed)
- at end of training: downscale all weights $B \to p \cdot B$ and use all neurons for prediction.
  - this is an approximation for the statistical interpretation of dropout
  - there are 2^H possible subnetworks ⇒ dropout trains a (small) random fraction of these
  - all subnetworks share $O(H^2)$ weights
  - at prediction time: sample M of the 2^H possible subnetworks and return the average of their prediction.
  - but: this is expensive ⇒ approximate the average of the predictions with the prediction using the average network ⇒ single prediction with average network instead of M predictions from subnetworks
  - if all activations $\varphi_l(t)$ were linear, the average network is just $B \to p \cdot B$
  - the inventors showed experimentally that this also works for nonlinear activations
  - equivalent alternative (easier to implement): upscale all active weights during training by $B_{\text{active}} \to \frac{1}{p} B_{\text{active}}$
- practical recommendations:
  - learning rate must be increased by a factor of 10…100, and we need more iterations
  - learning rate must decrease over time $\tau_t = \frac{\tau_0}{1+t/t_0}$
  - use max-norm regularization $B_{lh} \to B_{lh} \frac{\min(C, ||B_{lh}||_2)}{||B_{lh}||_2}$ with $C \approx 3..4$
- theory:
  - observation:
    * since neurons cannot rely on the presence of any input neuron during training, subtle co-adaptation effects (huge weights that cancel each other) cannot occur ⇒ strong regularization

    * weights tend to become sparse, e.g. if applied to images, weights B_l become local filters.
    * dropout reduces the Rademacher complexity of the network exponentially [Gao & Zhou, 2014]
    * reminder (ML1): Rademacher complexity $2\hat{R}$ measures the expected training success rate on nonsensical data, i.e. the features X_i are fixed and labels Y_i are random (*if the success rate here is high, the algorithm will strongly overfit*)
    * optimism: $opt = 2\hat{R} + O(\frac{1}{\sqrt{N}})$, test error ≤ train error + opt.
    * without dropout: $\hat{R} \in O\left(\prod_{l=1}^{L} ||B_l||_1\right) \Rightarrow$
        · classical regularization (=reducing $||B||$) helps
        · more layers (with total # weights fixed) increase overfitting
    * with dropout: $\hat{R} \in O\left(p^{L/2} \prod_{l=1}^{L} ||B||_1\right)$
      $\Rightarrow$ more layers reduce p^L exponentially $\Rightarrow$ we can use many layers ($L > 20$)

- variant: DropConnect ([Wan et al 2013]). Randomly drops weights (=graph edges) instead of neurons (=graph nodes)
    – small gains in performance, but a lot more complicated

7.3 Piecewise linear activation functions

- the other recent breakthrough
- **ReLU** ("rectified linear unit") [Nair & Hinton, 2010 / Glorot et al. 2011]

$$\text{ReLU}(t) = \max(0, t)$$

    – empirically works better than sigmoids
    – convex, only saturated for negative t
    – can approximate sigmoids by two ReLUs $\left(\text{ex. ReLU}(t + \theta) \text{ - ReLU}(t - \theta) - \theta\right)$
    – effect: features select a subnetwork "specialized" for that input (by driving some neurons into saturation)

      $\Rightarrow$ for each input, we select a linear subclassifier that is an "expert" for that particular region of the feature space.

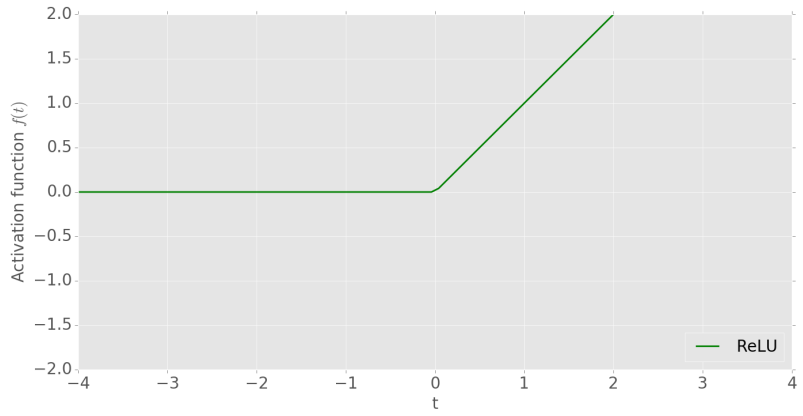


Figure 7.2: Rectified Linear Unit (ReLU) activation function, which is zero when $t < 0$ and then linear with slope 1 when $t > 0$.

7.4 MaxOut [Goodfellow et al. 2013]

- any linear function is convex, the pointwise max of a set of convex functions is also convex

  $\Rightarrow$ we can produce an arbitrary piecewise linear convex function from the max of linear functions

- any continuous function can be expressed as difference between two convex functions[1]

  $\Rightarrow$ we can arbitrarily well approximate any function by a piecewise linear function, taking the difference of two of these max(linear functions)

- maxout: alternate linear layers with maxout layers

$$l = 1, 3, 5, \ldots : Z_l = \tilde{Z}_l$$
$$l = 2, 4, 6, \ldots : Z_{lh} = \max_{S_{lh} \subset [0, H_{l-1}]} [Z_{l-1}]$$

  where S_{lh} is a subset of neurons in layer $l-1$ (typically: each neuron Z_{lh} uses k neurons of Z_{l-1} without sharing, $H_{l-1} = kH_l$)

- $k \in [2, \ldots, 20]$ is the number of linear segments after each maxout neuron.

- **main application**: convolutional neural networks: "**max pooling**": reduce a $\sqrt{k} \times \sqrt{k}$ window to a single pixel by taking the maximum.

[1] under mild assumptions

9	2	9	6	4	3
5	0	9	3	7	5
0	7	0	0	9	0
7	9	3	5	9	4

max pooling $\longmapsto$

9	9	7
9	5	9

Figure 7.3: Graphical depiction of the max pooling function used in convolutional neural networks to reduce a 2×2 window to a single pixel by taking the maximum.

7.5 PReLU “parametric ReLU” [He et al 2015]

- compromise between **ReLU** and **Maxout**: two flexible linear segments.

$$\varphi(t; a) = \begin{cases} t, & t \geq 0 \\ a \cdot t, & t < 0 \end{cases}$$

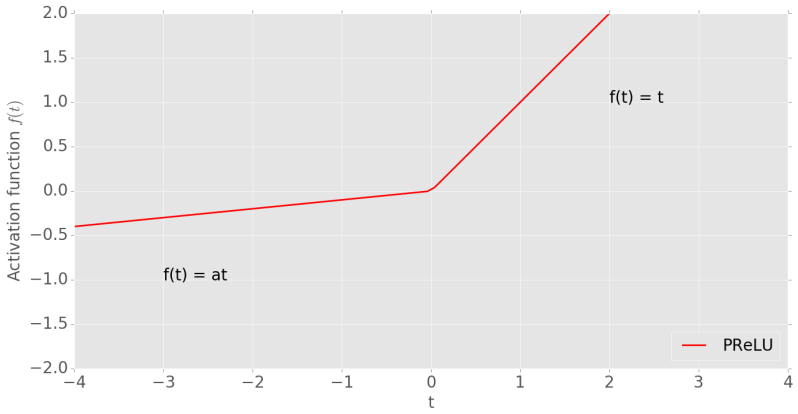


Figure 7.4: Leaky ReLUs or PReLUs are one attempt to fix the “dying ReLU” problem. Instead of the function being zero when t < 0, a leaky ReLU or PReLU will instead have a small negative slope a. For PReLUs the value of a is made into a parameter which is adaptively learned during training.

- each neuron has its own a
- the a's are trained via backpropagation:

$$\frac{\partial \varphi}{\partial Z} = \varphi'(\tilde{Z}_{lh}; a_{lh}) = \begin{cases} 1, & \tilde{Z}_{lh} \geq 0 \\ a, & \tilde{Z}_{lh} < 0 \end{cases}$$

$$\frac{\partial \varphi(\tilde{Z}_{lh}, a_{lh})}{\partial a_{lh}} = \begin{cases} 0, & \tilde{Z}_{lh} \geq 0 \\ \tilde{Z}_{lh}, & \tilde{Z}_{lh} < 0 \end{cases}$$

- weight initialization must be changed to:

$$B_l \sim \mathcal{N}\left(0, \frac{2}{(1 + a_0^2)H_{l-1}}\right)$$

$a_0 = 0.25$ recommended initial a

8 Lecture 13/05

8.1 Multi-class Classification

- **task**: assign each instance to **exactly one of** C **classes**
- distinguish from **multi-labeled problems**: **each instance** can have **several labels**. (Image: sunset, beach, surfing, ...; document: several topics)
- general approach: extend y to a "**one hot**", "**one-of-C**" vector of size C
  - hard decisions: $Y \in \{-1, 1\}^C$, contains exactly one $+1$
  - posterior probabilities: $Y \in [0, 1]^C$, $\sum_{k=1}^{C} Y_k = 1$
  - scores: $Y \in \mathbb{R}^C$
- goal: predict $\hat{Y}_i = Y_i$, we can obtain hard decisions from posteriors and scores:

$$\arg\max_k Y_k$$

- some classifiers have natural generalizations to the multi-class case
  - nearest neighbor: predict class of the nearest neighbor (or the majority of several near neighbors)
  - Naive Bayes:
    - learn $1D$ likelihoods for each feature and each class (DC likelihoods)
    - computation of posteriors via Bayes' rule is easy
  - decision trees:
    - split selection criteria are naturally multi-class
      - entropy: minimize $N_{\text{left}} \sum_k \left(-p_{\text{left,k}} \log p_{\text{left,k}}\right) + N_{\text{right}} \sum_k \left(-p_{\text{right,k}} \log p_{\text{right,k}}\right)$
      - Gini impurity: minimize $N_{\text{left}} \left(1 - \sum_k p_{\text{left,k}}^2\right) + N_{\text{right}} \left(1 - \sum_k p_{\text{right,k}}^2\right)$
    - prediction of each leaf: $\hat{y} = [p_{\text{leaf,k}}]$
  - random forest:
    - ensembles of decision trees:
      - train each tree on a random subset of the instances
      - only consider a random subset of the features when selecting a split
    - prediction: average over all tree predictions $\hat{Y} = \sum \hat{Y}_t$

- neural network:
  - define the non-linearity of the output layer via the "soft-max" function

$$Z_{Lk} = \frac{\exp(\tilde{Z}_{Lk})}{\sum_{k'} \exp \tilde{Z}_{Lk'}}$$

  - train with the "cross-entropy loss"

$$Loss = \sum_k \left[-\mathbb{I}[Y_i = k] \log Z_{Lk} - \mathbb{I}[Y_i \neq k] \log(1 - Z_{Lk})\right]$$

  - back propagation:

$$\partial_{Lk} = \frac{\partial Loss}{\partial Z_{Lk}} = \begin{cases} -1/Z_{Lk}, & k = Y_i \\ \frac{1}{1-Z_{Lk}}, & k \neq Y_i \end{cases}$$

$$\frac{\partial Z_{Lk}}{\partial \tilde{Z}_{Lk}} = ... = Z_{Lk} - Z_{Lk}^2$$

$$\frac{\partial Z_{Lk}}{\partial \tilde{Z}_{Lk''}} = ... = -Z_{Lk} Z_{Lk''}$$

$$\Rightarrow \text{Jacobian } J_{kk'} = \begin{cases} Z_{Lk}(1 - Z_{Lk}), & k = k' \\ -Z_{Lk} Z_{Lk'}, & k \neq k' \end{cases}$$

$$\tilde{\delta}_{Lk} = \frac{\partial Loss}{\partial \tilde{Z}_{Lk}} = \sum_{k''} \delta_{Lk''} J_{k''k} = \begin{cases} Z_{Lk} - 1 - \sum_{k'' \neq k} \frac{Z_{Lk''}}{1-Z_{Lk''}} Z_{Lk}, & Y_i = k \\ 2Z_{Lk} - \sum_{k'' \neq k, k'} \frac{Z_{Lk''}}{1-Z_{Lk''}} Z_{Lk}, & Y_i = k' \neq k \end{cases}$$

- logistic regression: is just the special NN with a single layer, $L = 1$
- both NN and LR are true multi-class algorithms because the prediction Z_{Lk} are not independent:
  - trained jointly and coupled via the Jacobian
  - prediction is coupled via the softmax normalization
  - all Z_{Lk} share the hidden weights
- Support vector machine 2-class objective:

$$\min_{\beta, b} \tfrac{1}{2}||\beta||_2^2 + \frac{\lambda}{N} \sum_i \text{hinge}(1 - Y_i(X_i\beta + b))$$

  equivalently

$$\min_{\beta, b} \tfrac{1}{2}||\beta||_2^2 + \frac{\lambda}{N} \sum_i \xi_i \text{ s.t. } {}^1\, Y_i(X_i\beta + b) \geq 1 - \xi_i, \;\; \xi_i \geq 0$$

  ξ_i are called slack variables

  - generalization of [Weston & Wathins, 1999]:
  - we now have a (β, b) pair for each class $\beta \to B \quad [D \times C]$, $b \to$ vector $\quad [b_k]$
  - now the constraint must hold for every pair of classes
  - predict $\hat{Y}_i = \arg\max_k (X_i B_k + b_k)$ (optional: "don't know" if too small)

    $\Rightarrow$ objective:

$$\min_{B,\underline{b}} \tfrac{1}{2}||B||_F^2 + \frac{\lambda}{N} \sum_i \sum_{k \neq Y_i} \xi_{i,k} \quad \text{s.t. } X_i B_{Y_i} + b_{Y_i} \geq X_i B_k + b_k + 2 - \xi_{ik} \text{ f.a. } k \neq Y_i$$

    $\Rightarrow N(C-1)$ slack variables
  - compute the dual and train in the dual space $\Rightarrow$ difficult

- generalization of [Crammer & Singer, 2001]:
  - it is sufficient when the constraint holds for the Y_i's closest competitor
  - also absorb the threshold parameters $\underline{b}$ into B (by adding a feature $X_{i,0} = 1$) objective

$$\min_B \tfrac{1}{2}||B||_F^2 + \frac{\lambda}{N} \sum_i \xi_i \quad \text{s.t. } X_i B_{Y_i} \geq \max_{k \neq Y_i} (X_i B_k) + 1 - \xi_i$$

  - again train in the dual $\Rightarrow$ easier, more popular than WW

- traditional belief: CS is better than WW because there are only N constraints instead of $N(C-1)$

- but: [Drogan et al. 2011] claim that the crucial difference is actually the elimination of the intercepts b because they lead to difficult equality constraints in the dual $\Rightarrow$ eliminate b in the WW and got better than CS

- if your classifier is not generalizable to $C > 2$: reduce the multi-class problem to a set of 2 class problems
  - "**one-vs-all**" or "**one-vs-rest**": train C classifiers where classifiers k' gets labels $Y_{i,k'} \begin{cases} +1, & k' = Y_i \\ -1, & k' \neq Y_i \end{cases}$ train $h_{k'}(X)$ binary classifiers for k' vs rest
    * predict: $\hat{Y}_{i,0} \arg\max_{k'} h_{k'}(X_i)$
    * this only works if the outputs of a $h_{k'}(X)$ are comparable in magnitude
      · $h_{k'}(X)$ return hard decisions: return $\hat{Y}$ if exactly one $h_{k'}(X)$ returns +1 otherwise "don't know"
      · $h_{k'}(X)$ return posteriors: just take the max
      · $h_{k'}(X)$ return scores: make the scores comparable, example: all $h_{k'}(X)$ are linear classifiers $h_{k'}(X) = XB_{k'} + b_{k'}$ are comparable when $||B_{k'}||_2 = 1$

- "**one-vs-all**" or "**all pairs**": train $\frac{C(C-1)}{2}$ classifiers for all possible pairs in k', k'' $\Rightarrow$ train $h_{k',k''}(X)$ with labels $Y_i = \begin{cases} +1, & Y_i = k' \\ -1, & Y_i = k'' \end{cases}$ and don't use the instances of the other classes
  - isn't this too expensive?

    Not always: h are kernel SVMs: training takes $\Omega(N^2)$ times, OVS takes $\Omega(CN^2)$, but OVO takes $\Omega\left(\frac{C(C-1)}{2}\left(\frac{2N}{C}\right)^2\right) = \Omega(N^2) \Rightarrow$ faster
- **prediction**:
  - variant 1: apply all classifiers and return the class with most +1 votes
  - variant 2 [Platt et al. 2000]: fill a vector with the class labels (in any order), apply the classifier for the first and last entry in the list $\Rightarrow$ pop the loosing label from the list, repeat until only one label remains $\Rightarrow \hat{Y}_i$. This can be written as a decision DAG (directed acyclic graph)

9 Lecture 20/05

9.1 Coding Matrices for multi-class problems

- we had **one vs. rest** (OVR) and **one vs. one** (OVO): We train L binary classifiers, where classes get temporary labels from $\tilde{Y}_{il} \in \{-1, 0, 1\}$[1]

  $\Rightarrow$ write labels as a $C \times L$ matrix M, s.t. $M_{kl} = \begin{cases} 1, & \text{class } k \text{ is pos in classifier } l \\ -1, & \text{class } k \text{ is pos in classifier } l \\ 0, & \text{class is not used} \end{cases}$

  e.g. $M_{\text{ovr}} = \begin{pmatrix} 1 & -1 & \dots & -1 \\ -1 & 1 & -1 & \dots \\ \dots & \dots & \dots & \dots \\ -1 & \dots & -1 & 1 \end{pmatrix}$, $M_{\text{ovo}} = \begin{pmatrix} 1 & 1\dots & 0 & 0 & \dots & \\ -1 & 0 & \dots & 1 & 1 & \dots \\ 0 & -1 & \dots & -1 & 0 & \dots \\ 0 & 0 & \dots & 0 & -1 & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \end{pmatrix}$

- in principle, M can be arbitrary, the best choice of M is an open problem (restriction, rows of M must differ)
  - OVO, OVR are still good choices
  - [Dietterich et al. 1995]: **Error-correcting output codes** (ECOC). Idea: make the rows of M pairwise as different as possible. $\Rightarrow$ classification becomes robust against errors in a few of the L classifiers. If for classes k and k', at least L' elements of M differ, we can recover from $\lfloor \frac{L'-1}{2} \rfloor$ by majority vote

- [Sun et al. 2005] : choose M at random, simple and works well

- optimize M for your data and classifiers: current research
  - [Bautista et al 2015]: compute between-class covariance, $S \in [C \times C]$, compute PCA (eigenvector matrix EV). Initial guess $M = \text{sign}(EV)$, then iterate to maximize error-correction while staying similar to EV

- stagewise optimization (various authors): add a new column to M until the overall performance is satisfying, choose the new column "optimally" with respect to existing columns.

- prediction ("decoding"):

[1] 0 means, that the instance is not used to train classifier l

  - if all $h_l(X)$ return crisp binary labels, assign X to the row of M, with minimal Hamming distance.
  - if all $h_l(X)$ return posteriors or scores: compute the loss of all rows and choose k which minimizes the loss.

- coding via M is especially critical for boosting

$$h(X) = \sum_l a_l h_l(X)$$

  $h(X)$ will be correct, when the majority of h_l is correct

- This is easy for binary classification, $h_l(X)$ just must be a bit better than guessing.
- but: for multi-class, better than guessing means $p_{\text{correct}} = \frac{1}{c} + \varepsilon \Rightarrow$ majority vote will not be correct
- We recover the "weak-learning condition" by reducing to a set of binary problems via M.

9.2 Gaussian Processes (or The Statistical Theory of Interpolation)

- so far, we always assume that training data are iid ($p(X, Y)$ is stationary, but unknown)

  $\Rightarrow$ probability of the training set factorizes: $p((X_1, Y_1), ..., (X_N, Y_N)) = \prod_{i=1}^N p(X_i, Y_i)$

  $\Rightarrow$ the NLL is a sum $-\log p(..) = -\sum_{i=1}^N -\log p(X_i, Y_i)$

  $\Rightarrow$ the loss (training error) is additive over instances $\Rightarrow$ convenient optimization of loss

- but: many applications do not fulfill the iid assumption:
  - time series
  - images: neighboring pixels usually belong to the same object $\Rightarrow$ not independent

- three ways to deal with dependent data:
  1. define features that capture a neighborhood of each instance, e.g. image filters of a window of pixels.

     $\Rightarrow$ relationship between neighboring instances (pixels) is recorded into features describing the local changes

     $\Rightarrow$ can treat the data as **approximately iid**, given ("conditional on") these new features ($\Rightarrow$ chapter "Features")

2. factorize $(p(X_1, Y_1), ..., (X_N, Y_N))$ as good as possible, (e.g. $= \frac{1}{Z} p((X_1, Y_1), (X_2, Y_2)) \cdot ... \cdot p((X_{N-1}, Y_{N-1}), (X_N, Y_N))$ "**Markov assumption**: only neighboring trace-points are related")
   - **graphical models** find such factorizations systematically ($\Rightarrow$ chapter "GM")
3. learn the full joint probability $p((X_1, Y_1), ..., (X_N, Y_N)) \Rightarrow$ **Gaussian processes**
   - this is only tractable when we use a simple model for $p(..)$
   - obvious choice: multi-variate Gaussian distribution $\Rightarrow$ everything an be computed in closed form
   - typical application is regression, classification is modeled via regression of the posterior class probability.

- Consider a vector of values $\underline{Y} = [Y_1, ..., Y_N]$. These are eventually functions $Y_i = f(X_i)$, but we ignore the X_i for the moment. Indices i are now fixed and no longer permutable.
  - model their distribution by a N-dimensional Gaussian, $p(\underline{Y}) \sim \mathcal{N}(\overline{Y}, S) = \frac{1}{Z} \exp\left(-\frac{1}{2}(\underline{Y} - \overline{Y})^\top S^{-1} (\underline{Y} - \overline{Y})\right)$ take $N \to \infty$: $\underline{Y}$ becomes a function, we still write formally $\underline{Y}_\infty \sim \mathcal{N}(\overline{Y}_\infty, S_\infty)$ "infinite-dim Gaussian"
  - in practice, we only work with finitely many points: can be interpreted as a finite dimensional marginal of the infinite dimensional Gaussian (i.e. integrate out all points we are not interested in)
  - fortunately for a Gaussian all marginals are again Gaussian.

10 Lecture 22/05

10.1 Gaussian Processes

- Generalize from a finite vector $[Y_1, ..., Y_N]$ of *dependent* variables Y_i to a function $Y = f(X)$ by taking $N \to \infty$.
- Model the probability of $f(X)$ as a Gaussian $f(X) \to \mathcal{N}(\bar{f}(X), S)$ [1].

$$f(X) \sim p(f; \bar{f}, S) = \frac{1}{Z} \exp\left(-\frac{1}{2}\left\langle f - \bar{f}, f - \bar{f}\right\rangle_{H(S)}\right)$$

  - Under our model $p(f; \bar{f}, S)$ all functions with finite norm $\left\langle f - \bar{f}, f - \bar{f}\right\rangle_{H(S)}$ have non-zero probability.[2]
  - A function f has high probability if it is similar to $\bar{f}$ and conforms to the covariance structure given by $S \Leftrightarrow ||f - \bar{f}||_{H(S)}$ is small.
  - $S(X, X + \Delta X)$ decreases slowly with $\Delta X \Rightarrow$ neighboring points are highly correlated $\Rightarrow f$ should be smooth.[3]
  - $S(X, X + \Delta X)$ decreases quickly $\Rightarrow$ noisy functions are also probable[4]
  - S must be chosen by the designer to model the properties of the application.
- in practice, we are only interested in finitely many points of f (training & test points)

  $\Rightarrow$ We create marginal distributions of $\mathbb{N}(\bar{f}, S)$ by integrating out all points we don't care about.

  $\Rightarrow$ general property: any marginal of a Gaussian is again Gaussian

  let $[X_1, ..., X_N]$ be the training locations with observed response $[Y_1, ..., Y_N]^\mathsf{T} = \underline{Y}$, $[X_{N+1}, ..., X_{N+N'}]$ the test locations where we want to find $[\hat{Y}_{N+1}, ..., \hat{Y}_{N+N'}]^\mathsf{T} = \underline{Y'}$

  $\Rightarrow$ the marginal distribution of $[Y_1, ..., Y_{N+N'}]$ is $\mathcal{N}\left(\begin{bmatrix}\underline{Y} \\ \underline{Y'}\end{bmatrix} - \overline{Y}, S_{1:N+N'}\right)$[5]

[1] $\bar{f} = \mathbb{E}(f)$, $S = \mathbb{E}[(f(X) - \bar{f}(X))(f(X') - \bar{f}(X'))]$, covariance or kernel function

[2] $\langle f, g\rangle_{H(S)} := \int \frac{\mathcal{F}(f)\mathcal{F}(g)}{\mathcal{F}(S_0)} d\omega$, where $S_0(X) = S(X, 0)$ centered kernel function at origin

[3] the plot shows a rather smooth function

[4] the plot shows a "squiggly" function

[5] specialize the kernel to the points $[X_1, .., X_{N+N'}]$, and $\bar{Y} = [\bar{f}(X_i)]$

- we simplify by setting $\bar{f} = 0$, because we can always subtract $\bar{f}$ in preprocessing, and add it after analysis $\hat{Y}_{\text{final}} = \hat{Y} + \bar{f}(X)$

$$p(\underline{Y}, \underline{Y}') \propto \exp\left(-\tfrac{1}{2}\begin{bmatrix}\underline{Y}\\ \underline{Y}'\end{bmatrix}^{\mathsf{T}} S^{-1}_{1:N+N'}\begin{bmatrix}\underline{Y}\\ \underline{Y}'\end{bmatrix}\right)$$
$$p(\underline{Y}) \propto \exp\left(-\tfrac{1}{2}\underline{Y}^{\mathsf{T}} S^{-1}_{1:N}\underline{Y}\right)$$

- we are interested in

$$p(\underline{Y}'|\underline{Y}) = p(\underline{Y}, \underline{Y}')/p(\underline{Y})$$

  we need $S^{-1} = \tilde{S}$. To compute it, we partition S according to known and unknown $S = \begin{bmatrix}A & B\\ B^{\mathsf{T}} & C\end{bmatrix}$, $\tilde{S} = \begin{bmatrix}\tilde{A} & \tilde{B}\\ \tilde{B}^{\mathsf{T}} & \tilde{C}\end{bmatrix}$, by definition $S^{-1}S = \tilde{S}S = \mathbb{1}_{N+N'}$[6]

- we get

$$p(\underline{Y}'|\underline{Y}) \sim \mathcal{N}\left(B^{\mathsf{T}}A^{-1}\underline{Y}, C - B^{\mathsf{T}}A^{-1}B\right)$$

- introduce kernel function $k(X, X')$, kernel matrix K with $K_{ij} = k(X_i, X_j)$, kernel vector $\mathbf{k}$[7] with $\mathbf{k} = k(X', X_i)$, where X' is a test point, $\kappa = K(X', X')$

- we can compute the test responses one point at a time, i.e. we can set $N' = 1$

$$p(Y'|\underline{Y}) \sim \mathcal{N}\left(\mathbf{k}(X')^{\mathsf{T}}K^{-1}\underline{Y}, \kappa - \mathbf{k}(X')^{\mathsf{T}}K^{-1}\mathbf{k}(X')\right)$$

  fundamental interpolation equation:

$$\bar{Y} = \mathbb{E}[Y' = f(X')] = \mathbf{k}(X')^{\mathsf{T}}K^{-1}\underline{Y}$$

  uncertainty of the interpolated point[8]:

$$\text{var}[\hat{Y}] = \kappa - \mathbf{k}(X')^{\mathsf{T}}K^{-1}\mathbf{k}(X')$$

- define *interpolation coefficients*: $\tilde{\underline{Y}} = K^{-1}\underline{Y}$ can be precomputed[9] because of independence from X'

- $\hat{Y} = \mathbf{k}(X')^{\mathsf{T}}\tilde{\underline{Y}}$

- example: linear interpolation, assume that X is 1D and X_i are equidistant (a grid), w.l.o.g. we set $X_i = i$ for the training points

[6] see Bishop p. 307 for the derivation of $\tilde{S}$

[7] actually a cursive k

[8] note: independent of $\underline{Y}$

[9] in practice: solve linear system $K\tilde{\underline{Y}} = \underline{Y}$... avoid computing K^{-1}. two typical algorithms:

1. if K is dense: Cholesky decomposition $K = LL^{\mathsf{T}}$
2. if K is sparse: conjugated gradients

- $k(X,X') = (1-|X-X'|)_+$, $K_{ij} = k(X_i,X_j) = k(i,j) = (1-|i-j|)_+$, $i,j \in \{1,\dots,N\} \Rightarrow$
  $K = \mathbb{1}_N$, $\kappa = 1$, $k(X',X_i) = \begin{cases} 0, & |X'-X_i| \geq 1 \\ 1-t & i = \lfloor X' \rfloor \\ t, & i = \lfloor X' \rfloor + 1 \end{cases}$ [10]

  $\hat{Y}(X') = \mathbf{k}(X')^\mathsf{T}\underline{Y} = (1-t)Y_i + tY_{i+1}$, $i = \lfloor X' \rfloor$, $t = X' - \lfloor X' \rfloor$

$$\begin{aligned} \operatorname{var}(\hat{Y}) = \kappa - \mathbf{k}(X')^\mathsf{T}K^{-1}\mathbf{k}(X') &= 1 - (\mathbf{k})(X')^\mathsf{T}\mathbf{k}(X') \\ &= 1 - (1-t)^2 - t^2 = 2t(1-t) \end{aligned}$$

[10] $t = X' - \lfloor X' \rfloor$

11 Lecture 27/05

Reminder: $\mathbb{E}(Y') = \hat{Y} = \mathbf{k}(X')^\top K^{-1}\underline{Y} = \mathbf{k}(X')^\top \tilde{Y}$

Kernel functions: A function $K(X,X')$ is a kernel iff it is positive definite (**Mercer's condition**).

- New kernel functions can be constructed from existing ones with easy operations.
  - a positive linear combination of kernels is a kernel $K_{\text{new}}(X,X') = \alpha_1 K_1(X,X') + \ldots + \alpha_M K_M(X,X')$, $\alpha_1,\ldots,\alpha_M > 0$
  - a product of kernels is a kernel $K_{\text{new}}(X,X') = K_1(X,X')K_2(X,X')$
  - We may map the X into an arbitrary feature space before applying the kernel function $K_{\text{new}}(X,X') = K_1(\phi(X),\phi(X'))$.
  - the exponential of a kernel is again a kernel $K_{\text{new}}(X,X') = \exp(K_1(X,X'))$
  - [...]
- two big classes of popular kernels: "**radial basis functions**", "**tensor product kernels**"
- radial basis functions (**RBF**): $K(X,X') = K(r)$, $r = ||X - X'||$ distance between X, X' in some norm (usually Euclidean or weighted Euclidean)
  - **squared exponential** (aka Gaussian): $K(r) = \exp\left(-\frac{1}{2}\left(\frac{r}{\rho}\right)^2\right)$ where ρ = "bandwidth" of the kernel
  - γ**-exponential**: $K(r) = \exp\left(-\left(\frac{r}{\gamma}\right)^\gamma\right)$, $\gamma \in [0,2]$
  - **Mat*é*rn kernels** (less smooth than squared exponential):

$$K(r) \sim \left(\frac{r}{\gamma}\right)^\gamma k_\gamma\left(\sqrt{2\gamma}\frac{r}{\gamma}\right)^1$$

    - $\gamma = 1/2$, $K(r) = \exp(-\frac{r}{\rho})$ **Ornstein-Uhlenbeck Kernel** for Brownian motion - very rough
    - $\gamma = 3/2$, $K(r) = (1 + \sqrt{3}\frac{r}{\rho})\exp(\sqrt{3}\frac{r}{\rho})$
    - $\gamma = 5/2$, $K(r) = \left(1 + \sqrt{5}\frac{r}{\rho} + \frac{5}{3}\left(\frac{r}{\rho}\right)^2\right)\exp(-\sqrt{5}\frac{r}{\rho})$
  - **inverse quadrics**: smoother than squared exp: $K(r) = \frac{1}{\left(1+\frac{1}{2\alpha}\left(\frac{r}{\rho}\right)^2\right)^\alpha}$ $\alpha > 0$.
    - $\alpha = 1/2$ $K(r) = \frac{1}{\sqrt{1+\left(\frac{r}{\rho}\right)^2}}$

    * $\alpha = 1, K(r) = \frac{1}{1+\frac{1}{2}\left(\frac{r}{\rho}\right)^2}$
  - for 2D feature spaces: **thin plate spline** $K(r) = r^2 \log r$
    * only "conditionally positive definite" – is positive definite after performing linear regression (i.e. TPS is applied to the residuals of linear regression)
    * advantage: it's usually not necessary to optimize the bandwidth
    * TPS is the minimum energy surface of a (infinitely thin) elastic plate attached to the training points:
      · it minimizes the curvature integral $\int_{\mathbb{R}^2}(f_{xx}^2 + 2f_{xy}^2 + f_{yy}^2)dxdy$
- all kernels so far have **infinite support** $K(r) > 0$ even for very large $r \Rightarrow$ **kernel matrix** K **is dense**, i.e. expensive to invert when N is big (solution of linear system $K\tilde{Y} = \underline{Y}$ takes $O(N^3)$)
- A kernel with **compact support** (i.e. $K(r) = 0$ if $r > r_{\max}$) leads to a **sparse kernel** matrix. $\Rightarrow$ sparse solvers need $O(N)$
  - truncate a non-compact kernel — approximation
  - define compact kernels, e.g. **Wendland splines** $K(r) = \left(1 - \frac{r}{\rho}\right)_+^\gamma \text{poly}_\gamma(\frac{r}{\rho})$
    * choose γ and poly according to feature space dimension and the required # of derivatives
      · smooth, but not differentiable: $K(r) = \left(1 - \frac{r}{\rho}\right)_+^\gamma, \gamma = \lfloor \frac{D}{2} \rfloor + 1$
      · 1-times differentiable: $K(r) = \left(1 - \frac{r}{\rho}\right)_+^\gamma \left(\gamma \frac{r}{\gamma} + 1\right), \gamma = \lfloor \frac{D}{2} \rfloor + 2$
      · [...]
- radial basis functions are best if the training points X_i are irregularly arranged — "scattered data interpolation"
- if the X_i form a regular grid, tensor product kernels — allow to work in one dimension at a time
- tensor product kernel: $K(X, X') = K_1(X_1, X_1') \cdots K_D(X_D, X_D')$, each being 1D kernels
- **squared exponential**:

$$K(x, x') = \exp\left(-\tfrac{1}{2}||X - X'||^2\right) = \exp\left(-\tfrac{1}{2}(X_1 - X_1')^2\right) \cdots \exp\left(-\tfrac{1}{2}(X_D - X_D')^2\right)$$

  - only kernel that is both a RBF and a tensor product
- **B-splines** ($X_{i+1,j} - X_{i,j} = 1$ unit grid, $\rho = 1$, i.e. preprocess data accordingly)[2]:

$$k(x) = B_\gamma(x) = \int_{x-1/2}^{x+1/2} B_{\gamma-1}(x)dx = B_0 * B_{\gamma-1}, \quad B_0(x) = \begin{cases} 1, & -1/2 < x \leq 1/2 \\ 0 \end{cases}$$

[2] to simplify notation

$$B_1(x) = \begin{cases} 1 - |x|, & |x| \leq 1 \\ 0 \end{cases}$$

$$B_2(x) = \begin{cases} \frac{3}{4} - x^2, & |x| \leq \frac{1}{2} \\ \frac{1}{2}(\frac{3}{2} - |x|)^2, & \frac{1}{2} \leq |x| \leq \frac{3}{2} \\ 0 \end{cases}$$

$$B_3(x) = \begin{cases} \frac{2}{3} - x^2 + \frac{1}{2}|x|^3, & |x| \leq 1 \\ \frac{1}{6}(2 - |x|)^3, & 1 < |x| \leq 2 \\ 0 \end{cases}$$

$$B_\infty(x) \sim e^{-\frac{1}{2}\left(\frac{x}{\rho}\right)^2}$$

- kernel matrix is sparse, e.g. B_0 and B_1: $K = \mathbb{I}_N$, B_2 and B_3: K is tridiagonal. Tridiagonal systems are easy to solve:
  - Thomas algorithm
  - recursive filters
- **cardinal functions** $K(x) = \begin{cases} 1, & x = 0 \\ 0, & x \in \{\pm 1, \pm 2, \pm 3, \ldots\} \\ \neq 0 \end{cases}$
  - B_0 and B_1 are cardinal
  - ideal interpolator: $\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$
  - **Catmull-Rom spline**[3]: $C(x) = \begin{cases} 1 - \frac{5}{2}x^2 + \frac{3}{2}|x|^3, & |x| \leq 1 \\ 2 - 4|x| + \frac{5}{2}x^2 - \frac{1}{2}|x|^3, & 1 < |x| \leq 2 \\ 0 \end{cases}$
  - advantage: the kernel matrix $K = \mathbb{I}_N$, giving us $\hat{Y}(X') = \mathbf{k}(X')^\mathsf{T}\underline{Y}$
  - any $1D$ kernel defines a cardinal function: $\kappa(X')^\mathsf{T} = \mathbf{k}(X')^\mathsf{T}K^{-1}$

[3] compact support version of the sinc function

12 Lecture 29/05

- on the grid, Gaussian process interpolation is just *filtering* (convolution)
- example: Catmull-Rom spline $C(X)$, $y' = \mathbf{k}(X')^\mathsf{T} K^{-1} Y$ [1]
  To compute $\mathbf{k}(X')$, place a kernel function centered at X', $i_0 = \lfloor X' \rfloor$, $t = X' - i_0$, $\mathbf{k}(X') = [0, \ldots, 0, \underbrace{C(-t-1)}_{\text{index } -2}, \underbrace{C(-t)}_{-1}, \underbrace{C(-t+1)}_{0}, \underbrace{C(-t+2)}_{1}, 0, \ldots, 0]$. The interpolation is then the following filter

$$y' = \mathbf{k}(X')\underline{Y} = C(-t-1)Y_{i_0-1} + C(-t)Y_{i_0} + C(-t+1)Y_{i_0+1} + C(-t+2)Y_{i_0+2}.$$

- example: B-spline B_3: now $K \neq \mathbb{I}$, but a tridiagonal matrix. Define interpolation coefficients $\tilde{Y} = K^{-1}\underline{Y}$

  $\Rightarrow$ interpolation is analog to C-Rom:

$$y' = \mathbf{k}(X')\underline{Y} = B_3(-t-1)\tilde{Y}_{i_0-1} + B_3(-t)\tilde{Y}_{i_0} + B_3(-t+1)\tilde{Y}_{i_0+1} + B_3(-t+2)\tilde{Y}_{i_0+2}$$

  Computing $\tilde{Y}$ is a preprocessing of Y. Since K is tridiagonal $K^{-1}\underline{Y}$ can also be implemented by filtering (specifically, a pair of recursive filters [Unser et al. 1991]). Intuitive effect of K^{-1}: since B_3 is a smoothing filter, we would not get interpolation when $Y' = \mathbf{k}(X')^\mathsf{T}\underline{Y}$. The pre-filtering of Y with K^{-1} exactly counters the *smoothing* effect at the grid points. $\tilde{Y} = K^{-1}\underline{Y}$ is *sharpening*

12.1 Uncertainty of GP interpolation

case 1 Y_i are assumed to be noise-free $\Rightarrow$ we have to keep the values intact $\Rightarrow$ interpolation $\hat{Y}_i = \mathbf{k}(X_i)^\mathsf{T} K^{-1}\underline{Y} = Y_i$ and the variance $\mathrm{var}[Y'] = k(X', X') - \mathbf{k}(X')^\mathsf{T} K^{-1}\mathbf{k}(X') = 0$ is $X' = X_i$

case 2 Y_i are noisy: $Y_i = \underbrace{f(X_i)}_{\text{noise-free solution}} + \varepsilon_i$, $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$. The Gaussian process becomes:

$$\underline{Y} \sim \mathcal{N}(\overline{f}(=0), K + \sigma\mathbb{I})$$

where K is the uncertainty about the true function f, whereas $\sigma^2\mathbb{I}$ is the uncertainty in the measurements of Y_i.

[1] Y values at grid points, $K = \mathbb{I}$ for Catmull-Rom

- We need the conditional probability for unseen points, given the training points: $p(Y'|\underline{Y}) = p(\underline{Y}, Y')/p(\underline{Y})$[2]. The computations are almost the same as with $\sigma^2 = 0$, giving us

$$\hat{Y} = \mathbb{E}(Y'(X')) = \mathbf{k}(X')^\mathsf{T}(\underbrace{K}_{=A} + \sigma\mathbb{I}_N)\underline{Y} + \underbrace{\mathbb{E}\varepsilon'}_{=0}$$
$$\mathrm{var}(Y') = k(X', X') + \sigma^2 - \mathbf{k}(X')^\mathsf{T}(K + \sigma^2\mathbb{I}_N)^{-1}\mathbf{k}(X').$$

  This does not interpolate anymore: $\hat{\underline{Y}} = K(K + \sigma^2\mathbb{I})^{-1}\underline{Y} \neq \underline{Y}$, giving us a denoised version of $\underline{Y}$.

- σ^2 acts as a regularization parameter, it shrinks Y' towards 0 (more generally, towards $\overline{f}$).
- $\hat{Y} = \mathbf{k}(X')^\mathsf{T}(K + \sigma^2\mathbb{I})^{-1}\underline{Y}$ is exactly kernel ridge regression (see ML1), but now derived statistically.

12.2 Application [Snoek et al. 2012]: GP to optimize the hyper parameters of a learning algorithm

- Learning Algorithm 1 (LA1) solves some problem of interest, Learning Algorithm 2 (LA2) optimizes LA1.
- standard approach to hyperparameter optimization of LA1: grid search with cross-validation, but: CV is expensive, grid search is expensive, because exponentially many candidate parameter sets (in the # of parameters)
- do CV for a few hyperparameter sets θ_i and compute $Loss_i$
- use $\{(\theta_1, Loss_i)\}$ as training data for LA2
- find θ' which minimizes $Loss'$ in LA2

  $\Rightarrow$ we can try many candidates θ' ($\approx 10^6$) by cheap interpolation in LA2 (= GP)
- perform CV on LA1 only with our best candidate θ'_{best}
- repeat
- precisely, the best candidate minimizes $\frac{Loss'}{Std(Loss')}$ *"probability of improvement criterium"* or a more sophisticated criterion (better).
- suggest to use Matérn-$\frac{5}{2}$ kernel in LA2

[2] where $S = \begin{pmatrix} A + \sigma\mathbb{I}_N & B \\ B^\mathsf{T} & C + \sigma^2\mathbb{I}_{N'} \end{pmatrix}$, compare to an earlier lecture

12.3 Application: GP classification

- a standard GP learns a function $Y' = f(X') : \mathbb{R}^D \to \mathbb{R}$
- for classification, we need a posterior class probability $p(Y|X') : \mathbb{R}^D \to [0,1]$
- idea: inspired by logistic regression $p(Y|X') = \sigma(X'\beta)$, $f(X') = X'\beta$, $p(Y|X') = \sigma(f(X'))$. Now we replace $f(X')$ by a GP estimate: instead of $f(X') = X'\beta$ we use $f(X') = \mathbf{k}(X')^\mathsf{T}(K+\sigma^2\mathbb{I})^{-1}[f(X_1),...,f(X_N)]^\mathsf{T}$ but it's not so easy because f appears on the LHS and RHS
- introduce a *latent variable* Z and define $Z_i = f(X_i)$.

  $$p(Y|X) = \int p(Y|X,Z)p(Z|X)dZ$$

  simplify by making independence assumptions: $Y_i \perp\!\!\!\perp X|Z(X) \Rightarrow P(Y|X,Z) = p(Y|Z)$, and $Y \perp\!\!\!\perp Y'|Z,Z' \Rightarrow p(\{Y_i\}|\{Z_i\}) = \prod_i p(Y_i|Z_i)$ giving us

  $$p(Y|X) = \int \underbrace{p(Y|Z)}_{\sigma(Z)} \underbrace{p(Z|X)}_{\text{GP reg.}} dZ$$

- problem: how to determine Z_i
  $p(Z|X)$ actually depends on the training data $D = \{(X_i, Y_i)\}$, $p(Z|X,D)$.
  $[Z_i] = \arg\max_Z p(Z|X,D)$
- to model $p(Z|X,D)$, we make the *Laplace approximation*: in a neighborhood of the optimum $[\hat{Z}_i]$ $p(Z|X,D)$ looks like a Gaussian. $\Leftrightarrow$ we use the second order Taylor expansion of $\log p(Z|X,D)$

  $$\log p(Z|X,D) \approx \log p(\hat{Z}|X,D) - \tfrac{1}{2}(Z-\hat{Z})^\mathsf{T} H (Z-\hat{Z})$$

  [the linear term is missing, because $\hat{Z}$ is a maximum]
  H is the negative Hessian of $\log p(Z|X,D)$ at $\hat{Z}$

  $$H = -\left.\frac{\partial^2}{\partial Z^2} \log p(Z|X,D)\right|_{Z=\hat{Z}}$$

  $$p(Z|X,D) \approx \underbrace{e^{\log p(\hat{Z}|X,D)}}_{\text{const}} e^{-\frac{1}{2}(Z-\hat{Z})H(Z-\hat{Z})} = \mathcal{N}(\hat{Z}, H^{-1})$$

  $\Rightarrow$ choose H^{-1} as an appropriate kernel and estimate $\hat{Z}$

13 Lecture 03/06

13.1 GP classification

- $p(y' = \pm 1|X') = \int_{Z'} p(y|Z', X)p(Z'|X)dZ'$ latent variable Z: simplifies matters because of independence assumptions [1]
- in order to make predictions $p(Z'|X') = p(Z'|X', \mathcal{D})$ we need the Z_i values for the training points $\mathbb{E}(Z') = \mathbf{k}^\mathsf{T}(X')K^{-1}\hat{Z}$

  $\Rightarrow$ training = determine $\hat{Z}$
- $p(\hat{Z}, \{X_i, Y_i\}_{i=1}^N) = p(\hat{Z}|\{X_i, Y_i\}_{i=1}^N)p(\{X_i, Y_i\}_{i=1}^N) = p(\{Y_i\}|\hat{Z})p(\hat{Z}|\{X_i\})p(\{X_i\})$

$$\hat{Z} = \arg\max_Z p(Z|\{X_i, Y_i\}_{i=1}^N) = \arg\max_Z p(\{Y_i\}|Z)p(Z|\{X_i\})$$
$$= \arg\max_Z = \underbrace{\log p(\{Y_i\}|Z)}_{\sum_i \log p(Y_i|Z_i) = \sum_i \log \sigma(Y_i Z_i)} + \underbrace{\log \underbrace{p(Z|\{X_i\})}_{\text{GP}}}_{= -\frac{1}{2}Z^\mathsf{T}K^{-1}Z + \frac{1}{2}\log(K) + \frac{N}{2}\log 2\pi}$$

$$\Rightarrow \hat{Z} = \arg\max_Z = -\sum_i \log\left(1 + \exp(-Y_i Z_i)\right) - \tfrac{1}{2}Z^\mathsf{T}K^{-1}Z + \text{const} = \psi(Z)$$
$$\frac{\partial \psi(Z)}{\partial Z} = v - K^{-1}Z \stackrel{!}{=} 0$$

where $v = \begin{bmatrix} t_1 - \pi_1 \\ \dots \\ t_N - \pi_N \end{bmatrix}$, $t_i = 2Y_i - 1 \in \{0, 1\}, \pi_i = \sigma(Z_i)$

$$\frac{\partial^2 \psi(Z)}{\partial^2 Z^2} = W - K^{-1}$$

where $W = \begin{bmatrix} -\pi_1(1-\pi_1) & 0 \\ \dots & \dots \\ 0 & -\pi_N(1-\pi_N) \end{bmatrix}$

Update step: $Z^{(t+1)} = Z^{(t)} - (W^{(t)} - K^{-1})^{-1}(v^{(t)} - K^{-1}Z^{(t)})$, $\hat{Z} = Z^{(t\to\infty)}$

[numerically better formulation $\Rightarrow$ Rasmussen & Williams 2006]

[1] see last lecture

- predictions:
  - solve $p(Y|X') = \int_{Z'} \sigma(YZ')GP(Z'|\hat{Z},X')dZ'$, no closed form solution
    $\Rightarrow$ solve numerically or use the normal CDF instead of $\sigma(t)$ (but then we must adjust $\frac{\partial\psi}{\partial Z}, \frac{\partial^2\psi}{\partial Z^2}$ during training)
  - if we only need a decision function:

$$\hat{y} = \arg\max_Y p(Y|X') = \text{sign}(\mathbb{E}(Z')) = \text{sign}(\mathbf{k}^\top(X')K^{-1}\hat{Z})$$

13.2 The Bayesian Interpretation of GP regression (and their relation to "reproducing kernel Hilbert spaces"(RKHS))

- vector space $Y \in \mathbb{R}^N$, scalar prod. $\langle Y, Y' \rangle = Y^\top Y'$, how to visualize Y if $N > 3$, "**parallel coordinates**"

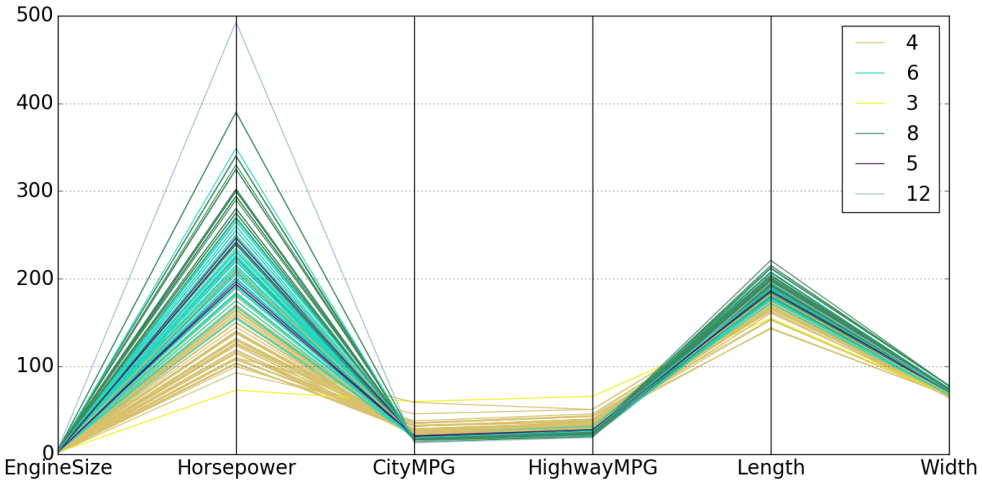


Figure 13.1: A depiction of the use of parallel coordinates as plotting technique for multivariate data. It allows one to see clusters in data and to estimate other statistics visually. When we are using parallel coordinates points are represented as connected line segments. Each vertical line represents one attribute of the car data set. One set of connected line segments represents one data point. Points that tend to cluster will appear closer together. The dataset is clustered in dependence of the number of cylinders given in the legend in the upper right (MPG-miles per gallon).

- Hilbert space: take $N \to \infty \Rightarrow$ parallel coordinates turn into a function $f(X)$, $\langle f, g \rangle = \int f(x')g(x')dx', x' \in \mathbb{R}^D$
- vector with generalized scalar product: arbitrary bilinear form $\langle Y, Y' \rangle = Y^\top AY'$ example: PCA, QDA

- doing the same in a Hilbert space gives the RKHS[2]:
  - kernel function $K(X,X') = K(X',X)$ pos.def.,
  - centered kernel function at x_0: $K_{x_0}(X) = K(X,X' = x_0)$

  H(k) is of *RKHS*:

  (i) $\forall x_0, K_{x_0} \in H(K)$

  (ii) reproducing property of the scalar product:

$$\forall x_0, \forall f(x) \in H(k) : \langle f, K_{x_0} \rangle_{H(k)} = f(x_0)$$

- to define the scalar product explicitly, we need the convolution operator

$$(f * g)(x) = \int f(x - x')g(x')dx' = \int f(x')g(x - x')dx'$$

- use convolution to define the inverse kernel function centered at $x_0 = 0$[3]

$$k_0^{-1} \Leftrightarrow (k_0^{-1} * k_0)(x) = \delta(x) = \int k_0^{-1} k(x - x')dx'$$

- to actually compute k_0^{-1}, its best to use Fourier transform: convolution theorem: $\mathcal{F}(f * g)(x) = \mathcal{F}(f)\mathcal{F}(g)$

$$k_0^{-1} \Leftrightarrow \mathcal{F}(k_0^{-1})\mathcal{F}(k_0) = 1, k_0^{-1} = \mathcal{F}^{-1}\left(\frac{1}{\mathcal{F}(k_0)}\right)$$

  but often, no closed form expression for $k_0^{-1}(x)$ exists, no problem in practice, because we can always explicitly invert the kernel matrix for our finite training set

- scalar product:

$$\langle f, g \rangle_{H(k)} := \int f(x')(k_0^{-1} * g)(x')dx' = \int (k_0^{-1/2} * f)(x') \cdot (k_0^{-1/2} * g)(x')dx'$$

- this fulfills the reproducing property: $\langle f, k_{x_0} \rangle_{H(x)} = f(x_0)$

$$\begin{aligned} \langle f, k_{x_0} \rangle_{H(x)} &= \int f(x') \underbrace{(k_0^{-1} * k_{x_0})(x')}_{(*)} dx' \\ &= \int f(x')\delta(x' - x_0)dx' = f(x_0) \end{aligned}$$

$$(*) \int k_0^{-1}(x'')k_{x_0}(x' - x'')dx'' = \int k_0^{-1}(x'')k_0((x' - x_0) - x'')dx'' = \delta(x' - x_0)$$

  if $f(x) = k_{x_1}(x)$: $\langle k_{x_1}, k_{x_0} \rangle_{H(k)} = K_{x_1}(x_0) = K_{x_0}(x_1) = K(x_0, x_1) = K_{01}$ (kernel matrix element)

[2]Note on the notation: k, K might sometimes need to be exchanged

[3]analog to inverse matrix $M^{-1} \Leftrightarrow M^{-1}M = \mathbb{1}$

- application to GP regression:
  - given training data $\mathcal{D}$, find the function $f(x)$ that has maximum a posteriori probability $p(f|\mathcal{D})$
  - expand according to Bayes $p(f|\mathcal{D}) \propto \underbrace{p(\mathcal{D}|f)}_{\text{training error}} \underbrace{p(f)}_{\text{prior for } f}$
  - data probability: squared loss $p(\mathcal{D}|f) = \exp\left(\frac{1}{2\sigma^2}\sum_i (Y_i - f(X_i))^2\right)$
  - prior: choose a Gaussian process $p(f) = \exp\left(-\frac{1}{2}\langle f, f\rangle_{H(k)}\right)$
  - prior experience encodes the expected smoothness of f in the kernel K and prefers f that conforms to this smoothness requirement. $\Leftrightarrow \langle f, f\rangle$ small $\Leftrightarrow p(f)$ high

$$\hat{f} = \arg\max_f p(f|\mathcal{D}) = \frac{1}{\sigma^2}\sum_i (Y_i - f(x_i))^2 + \underbrace{\langle f, f\rangle_{H(k)}}_{\text{regularization}} \qquad (*)$$

  - to solve this, we need the "representer theorem" [Kimeldorf & Wahba 1971]
    Thm: In any problem $(*)$, the optimal solution can be expressed as a linear combination of kernel functions *centered* at the training points $\hat{f}(x) = \sum_i \alpha_i K_{x_i}(X)$, just determine the α_i

- insert the representer theorem into $(*)$

$$\left\langle \hat{f}, \hat{f}\right\rangle_{H(k)} = \left\langle \sum \alpha_i K_{x_i}, \sum \alpha_i K_{x_i}\right\rangle = \sum_{i,j} \alpha_i \alpha_j \left\langle K_{x_i}, K_{x_j}\right\rangle$$
$$= \sum_{i,j} \alpha_i \alpha_j K(X_i, X_j) = \alpha^\mathsf{T} K \alpha$$

  Expansion of the first term of $(*)$:

$$= \frac{1}{\sigma^2}\sum_i Y_i^2 - \frac{2}{\sigma^2}\sum_i Y_i f(X_i) + \frac{1}{\sigma^2}\sum_i f(X_i)^2$$

  extending the second and third summand

$$\sum_i (f(x))^2 = \sum_i (\sum_j \alpha_j K_{x_j}(x_i))^2 = \sum_{jk} \alpha_j \alpha_k \sum_i k_{x_j}(X_i)\kappa_{x_k}(X_i) = \alpha^\mathsf{T} K^2 \alpha$$
$$\sum_i Y_i f(X_i) = \sum_i Y_i \sum_j \alpha_j k_{x_j}(X_i) = \alpha^\mathsf{T} K Y$$

  inserting into $(*)$ again gives:

$$(*) = \frac{1}{\sigma^2}\sum_i Y_i^2 - \frac{2}{\sigma^2}\alpha^\mathsf{T} K Y + \frac{1}{\sigma^2}\alpha^\mathsf{T} K^2 \alpha + \alpha^\mathsf{T} K \alpha$$
$$\frac{\partial(*)}{\partial \alpha} = -\frac{2}{\sigma^2} K Y + \frac{2}{\sigma^2} K^2 \alpha + 2K\alpha = YK\alpha + \sigma^2 \alpha \stackrel{!}{=} 0$$
$$\Rightarrow \alpha = (k + \sigma^2 \mathbb{1})^{-1} Y \quad \hat{Y} = \mathbf{k}(X')^\mathsf{T} \alpha$$

  aka **fundamental interpolation equation**

14 Lecture 10/06

14.1 Graphical Models

- **task**: model joint probability $p(X_1, ..., X_D)$, but:
  - direct modeling is intractable
  - no obvious factorization exists (e.g. for iid $p(X_1, ..., X_D) = \prod_j p(X_j)$)
- **idea**: use conditional independence between variables to factorize as good as possible, which is much weaker than unconditional independence, but our only chance
- "graphical": represent conditional independence by means of a graph[1]
- *example 1:* correct handling of independence/association is not at all obvious
  - problem: we know that Alice has two children that are not twins. What's the probability that both are boys?[2]
    1. you have no additional information: p_1
    2. we meet Alice with one of her children, who is a boy: p_2
    3. we meet Alice with one of her children, who is a boy and she says "This is my first-born": p_3
    4. we meet Alice with one of her children, who is a boy and she says "He was born on a Sunday": p_4
    5. we meet Alice with one of her children, who is a boy and she says "Today is his birthday": p_5

    $p_1 \neq p_2 \neq p_3 \neq p_4 \neq p_5$. The probabilities are $(p_1, p_2, p_3, p_4, p_5) = (\frac{1}{4}, \frac{1}{2}, \frac{1}{3}, \frac{13}{27}, \frac{729}{1459})$
    Let A be first-born, B second-born child:
    1. $p(A = \text{boy}, B = \text{boy}) = P(A = \text{boy})P(B = \text{boy}) = \frac{1}{4}$
    2. $p(A = \text{boy}, B = \text{boy} | A = \text{boy}) = \frac{P(A=\text{boy}, B=\text{boy})}{P(A=\text{boy})} = \frac{1/4}{1/2} = \frac{1}{2}$
    3. $p(A = \text{boy}, B = \text{boy} | A = \text{boy} \vee B = \text{boy}) = \frac{p(A=\text{boy}, B=\text{boy})}{P(A=\text{boy} \vee B=\text{boy})} = \frac{1/4}{3/4} = \frac{1}{3}$
    4. wrong model:

$$p\big(A{=}\text{boy} \wedge B{=}\text{boy} \wedge (A{=}\text{Sun} \vee B{=}\text{Sun}) | (A{=}\text{boy} \vee B{=}\text{boy}) \wedge (A{=}\text{Sun} \vee B{=}\text{Sun})\big)$$

[1] There will probably be a lot of plots in this chapter, which won't be reproduced here, see e.g. Barber, Koller&Friedman for those.

[2] We could just ask Bob.

$$= \frac{p(A = \text{boy}, B = \text{boy})}{p(A = \text{boy} \vee B = \text{boy})}$$

$\Rightarrow$ missing, that it's the same person who is a boy and born on a sunday

$\Rightarrow$ correct model:

$$p\big(A{=}\text{boy}, B{=}\text{boy} \wedge (A{=}\text{Sun} \vee B{=}\text{Sun}) \,|\, (A{=}\text{boy} \wedge A{=}\text{Sun}) \vee (B{=}\text{boy} \wedge B{=}\text{Sun})\big)$$

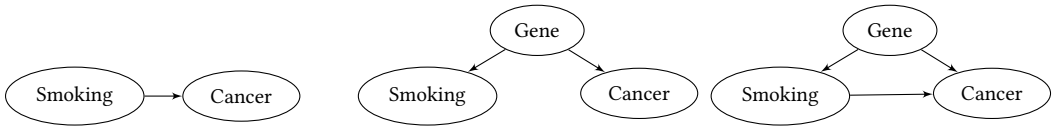
$$p_4 = \frac{13}{27} = \frac{2 \cdot 7 - 1}{4 \cdot 7 - 1}$$

5. see exercise

- *example 2*: **Simpson's paradox**: if dependencies are treated incorrectly, you can turn a statement into its opposite, using the same data
  - $\approx$ 1970 U Berkley was sued for preferring men over women

	male:app	male:adm	male:%	fem:app	fem:adm	fem:%
total	2590	1192	46	1835	557	30.4
A	825	512	62	108	89	**82**
B	560	353	63	25	17	**68**
C	325	120	**37**	593	202	34
D	417	138	33	375	131	**35**
E	191	53	**28**	393	94	24
F	272	16	6	341	24	**7**

  - 4 out of 6 departments prefer women
  - 5 out of 6 departments prefer the minority
  - in total: men are highly preferred
    $\Rightarrow$ explanation: women tend to apply for highly competitive fields
  - statistical mistakes:
    1. an association does not in general imply causality
       * to determine causality, better methods are needed
         · preferred: randomized controlled experiment (group applicants at random and force each group into a particular field $\Rightarrow$ dependency between sex & field is broken by "active intervention" ($\Rightarrow$ **interventioned dataset**))
         · often this is illegal or unethical or impossible $\Rightarrow$ have only "**observational dataset**" $\Rightarrow$ causality is a very difficult problem $\Rightarrow$ later



(a) (b) (c)

Figure 14.1: Three possible models for smoking and cancer. (a) Direct causal influence; (b) indirect causal influence via a latent common cause (Gene); (c) incorporated model with both influences.

2. omitted variable bias: apparent association could be causal, but can also have a common cause (smoking → lung cancer, gene → lung cancer and gene → smoking) or a mediating property (sex → admission, but sex→field → admission)
   if the additional variable is ignored (marginalized out), very misleading conclusions will be drawn

- graphical models are a tool to treat conditional independence systematically
  - two kinds:
    * directed (graphs): based on chain rule of probability
    * undirected (graphs): based on the Gibbs probability distribution $p(X) = \frac{1}{Z}\exp(-E(X))$
- chain rule: $p(X_1,...,X_D) = p(X_D|X_{D-1},...,X_1)\cdots p(X_2|X_1)p(X_1)$
  draw the decomposition as a directed graph[3] which is called a "Bayesian network"
- trick: can drop arcs when variables are conditionally independent (remember conditional independence does not in general imply general independence)
- goal: drop as many arcs as possible ⇒ simplest problem representation
  How many parameters are needed to specify the probability? Let $X_j \in \{1,..,C_j\}$.
  - $p(X_1,...,X_D)$ needs $\prod_j C_j - 1$ parameters
  - full factorization needs as many parameters
  - if we drop arcs, the number of parameters reduces

[3] see earlier comment

15 Lecture 12/06

15.1 Bayesian Networks (directed graphical models)

- **idea**:
  1. factorize the joint probability $p(X_1,...,X_D)$ according to the chain rule
  2. represent the factorization as a directed graph
  3. use conditional independence to remove as many edges as possible ⇒ simpler problem (reduced)
- **catch**:
  - every permutation of the variables results in a different, but equivalent factorization: $D!$ possibilities
  - but in some factorizations we can remove many more edges
    ⇒ goal: use the permutation that results in the fewest edges after step 3. the best permutation tends to be the one that results in a causal graph (i.e. arc direction = cause → effect)
- how to identify causal relationships:
  1. use **domain knowledge** (past → present→ future, property → measurement[1])
  2. perform **randomized controlled experiments**: experimenter intervenes to break potential dependencies, so that other dependencies can be analyzed in isolation (exclude the possibility of a common cause aka. confounder)
  3. when controlled experiments are impossible/illegal/unethical, **estimate causality from purely observational data**
     - this is very difficult and a hot research topic ⇒ later
- main task in BN:
  - prediction: in contrast to traditional methods, where prediction is relatively easy, here sophisticated *inference* algorithms are needed
    * compute probabilities not explicitly represented in the model:
      · marginals $p(X_j) = \sum p(X_1,...,X_D)$
      · marginals given evidence on some variables $p(X_j|X_{j'} = e_{j'})$
      · likewise for uncertain evidence

[1] can be violated in quantum mechanics

    - compute the most probable variable assignment (maximum a posteriori (MAP) solution) or several highly probable solutions (k-best)
    - support decision making ("will surgery help?")
  - training:
    - *parameter learning*: given the graph, learn the conditional probabilities of the decomposition
    - *structure learning*: identify the optimal (ideally: causal) graph

- two popular kinds of BN:
  - **temporal models**: causality is implied by time, e.g. speech recognition
  - **causal models**: give an *explanation* of the observed behavior that can be understood by domain experts

- **Pearl's** basic network construction algorithm
  1. identify all variables relevant to the problem (missing variables may lead to Simpson's paradox)
  2. arrange the variables in a useful order (ideally: causal)
  3. for $j = 1, \ldots, D$ (D = # variables)
     - add a node for X_j to the network
     - find a *minimal* subset $PA(X_j) \subseteq \{X_1, \ldots, X_{j-1}\} = S_{j-1}$ such that $X_j \perp\!\!\!\perp (S_{j-1} \backslash PA(X_j) | PA(X_j))$ $PA(X_j)$ are called the "parents" (e.g. use a statistical test like χ^2 test for conditional independence)
     - add arcs $\forall X_{j'} \in PA(X_j) : X_{j'} \rightarrow X_j$

     $\Rightarrow$ the graph represents the factorization $p(X_1, \ldots, X_D) = \prod_j p(X_j | PA(X_j))$
  4. learn the parameters of the distributions $p(X_j | PA(X_j))$ for all j ($p(X_j | PA(X_j))$), can be represented by conditional probability tables (CPT) or parametric models

  $\Rightarrow$ Bayesian or Belief Network (BN)

- there are three fundamental configurations in a BN
  - chain ("**causal chain**") $A \rightarrow B \rightarrow C$
  - diverging connection ("**common cause**") $A \leftarrow B \rightarrow C$
  - converging connection ("**common effect**") $A \rightarrow B \leftarrow C$

  $\Rightarrow$ behave interestingly when B is marginalized out or there is evidence on B

- chain:
  - if B is marginalized, we just loose information: $p(C|A) = \sum_B p(C|B)p(B|A)$ (uncertainty increases)
  - if B is known ($B = b$), then C is independent of A : $A \perp\!\!\!\perp C|B$ (dictated by the graph structure, otherwise A must be in $PA(C)$)

- common cause:
  - if B is marginalized, an association between A and C results (the arrow direction does not follow from the graph, but often from the application) example: Simpson's paradox, Berkley admission
  - if B is known: $A \perp\!\!\!\perp C|B$
- common effect:
  - if B is *not* marginalized but unknown $A \perp\!\!\!\perp C$
  - if B is marginalized: unconditional independence still holds
  - if B is known, A and C become conditionally dependent $A \not\perp\!\!\!\perp C|B$ ("**Bergson's paradox**")
- example: Burglary alarm[2]
  $p(B = 1) = 0.01$, marginal $p(A = 1) = 0.016$

B	$p(A = 1\|B)$	$p(A = 1, B)$	$p(B\|A = 1)$
0	0.007	0.0069	0.43
1	0.9	0.009	0.57

Now suppose you live in California: the alarm can be triggered by an earthquake

B	E	$p(A = 1\|B, E)$	$p(A = 1, B, E)$	$p(B, E\|A = 1)$	$p(B\|A = 1)p(E\|A = 1)$
0	0	0.001	0.00097	0.06	$\neq 0.27$
0	1	0.3	0.0059	0.37	$\neq 0.16$
1	0	0.9	0.0088	0.55	$\neq 0.35$
1	1	0.95	0.00019	0.01	$\neq 0.22$

marginalize out B:

E	$p(A = 1, E)$	$p(E\|A = 1)$
0	0.0098	0.62
1	0.0061	0.38

Bergson's paradox: given $A = 1$, we learn (e.g. from the news) that there was an earthquake $E = 1$. Compute $p(B|A = 1, E = 1) = \frac{p(A=1,B,E=1)}{p(A=1|E=1)}$

B	$p(B\|A = 1, E = 1)$
0	0.97
1	0.03

This is known as the *"explaining away effect"*

- The effect also occurs when we get evidence on any descendent of A.

[2]The tables are not in the right order. Figuring out the correct order is left as an exercise to the reader.

- The three fundamental configurations can be combined into a systematic criterion to identify all independence assumptions that are implicitly represented in a given graph. *"d-separation"*
  - directed path from $X \rightsquigarrow Y$ sequence of nodes $A_0 = X, A_1, ..., A_{k-1}, A_k = Y$ such that $A_{j-1} \rightarrow A_j$ is an arc
  - transitive closure (descendants) of X: $DE(X) = \{Y : X \rightsquigarrow Y\}$
  - ascendants of X: transitive closure of the transposed graph, nodes that can reach X: $AS(x) = \{Y | Y \rightsquigarrow X\}$
  - undirected path ($X \leftrightsquigarrow Y$): $A_0 = X, A_1, ..., A_{k-1}, A_k = Y$, such that $A_{j-1} \rightarrow A_j$ or $A_{j-1} \leftarrow A_j$ is an arc in the graph
- Consider a set $S \subset \{X_1, ..., X_D\}$ such that evidence is available for all nodes in S. Let $X, Y \notin S$. Then, an undirected path $X \leftrightsquigarrow Y$ is blocked by S if any of the following is true:
  1. $A_{i-1} - A_i - A_{i+1}$ is a chain and $A_i \in S$
  2. $A_{i-1} \leftarrow A_i \rightarrow A_{i+1}$ and $A_i \in S$
  3. $A_{i-1} \rightarrow A_i \leftarrow A_{i+1}$ and neither $A_i \notin S$ nor for $Z \in DE(A_i)$ $Z \notin S$
- **Def:** X and Y are d-separated[3] by S, if S blocks *every* path $X \leftrightsquigarrow Y$.

[3]Note: according to wiki d stands for directional

16 Lecture 17/06

- What independence assumptions does a BN encode?
  - when X and Y are associated, information must flow between them:
  - in a BN information can only flow along the arcs (in both directions!)
    $\Rightarrow$ we must consider an undirected path between X and Y ($X \nleftrightarrow Y$)
  - if information can flow along $X \nleftrightarrow Y$ the path is "active", otherwise "blocked"
  - if all paths between X and Y are blocked $\Rightarrow X \perp\!\!\!\perp Y$ (unconditionally)
  - given a set S of nodes where we have evidence (know the variable value), path activation can change[1]
- algorithm to check for d-separateness:
  Given: directed graph G, nodes S; $X, Y \notin S$
  1. Define the ancestral subgraph G' of G: remove all nodes not in $\{X, Y, S, \text{ancestors}(X, Y, S)\}$ and their arcs.
  2. Define moral graph G'' of G': for each node in G' connect all unconnected parents ("unmarried") by an undirected arc and remove all arrows.
  3. Construct G''' of G'' by removing all nodes from S in G''.
  4. X and Y are d-separated given S if they are unconnected given G'''.
- **Def:** A joint probability $p(X_1, ..., X_D)$ satisfies (directed global) Markov property w.r.t a graph, if X_j and $X_{j'}$, are d-separated by S implies $X_j \perp\!\!\!\perp X_{j'} | S$ in $p(X_1, ..., X_D)$.
- **Theorem:** If $p(X_1, ..., X_D)$ is Markov w.r.t a DAG G, then it can be factorized as $p(X_1, ..., X_D) = \prod_j p(X_j | PA(X_j))$.
- The converse is not generally true: conditional independence does not always imply d-separation.
  Example: X and Y are not d-separated but independent: $X \to Z \to Y \leftarrow X$.
  linear model:

$$
\begin{aligned}
X &= \varepsilon_X \sim \mathcal{N}(0, \sigma_X^2) \\
Z &= aX + \varepsilon_Z \\
Y &= bZ + cX + \varepsilon_Y \\
&= abX + b\varepsilon_Z + cX + \varepsilon_Y = (ab + c)X + \varepsilon'_Z
\end{aligned}
$$

  if $(ax + c) = 0 \Rightarrow X \perp\!\!\!\perp Y$

[1] see end of last lecture

- this is undesirable: define the problem away

- **Def:** A distribution $p(X_1,...,X_D)$ is *faithful* to a DAG G if $X_j \perp\!\!\!\perp X_{j'}|S$ implies d-separation of $X_j, X_{j'}$, given S.

- **Claim**: many important models for $p(X_j|PA(X_j))$ are faithful with probability 1.
  Advantage: d-separation, i.e. the structure of the graph, fully specifies all independence assumptions ⇒ we can separate the two problems
  1. define/learn the structure of the graph
  2. learn the probabilities, given the graph

16.1 Inference in BN

- **Inference**: compute interesting properties that are implicitly represented by the BN (graph structure and $p(X_j|PA(X_j))$ are known)
- basic algorithm: *variable elimination*: split the variables into 3 (disjoint & complete) sets
  - T: variables we are interested in ("targets")
  - V: variables where we have evidence ("visible")
  - U: variables we are not interested in
- when V is empty: variable elimination = marginalization over U: $p(T) = \sum_U p(T,U)$
- when V is *not* empty:
  1. define new functions $q(X_j|PA(X_j)) = \begin{cases} P(X_j|PA(X_j)), & \text{if var. assignment is comp. w/ } V \\ 0, & \text{otherwise} \end{cases}$
  2. marginalize over U $q(T|V) = \sum_U \prod_j q(X_j|PA(X_j), V)$
  3. turn into probability by normalization $p(T|V) = q(T|V) / \sum_{T'} q(T'|V)$
- example 1: last week's computations in the burglary alarm network
- example 2: Naive Bayes classifier. Assumptions:
  - class membership causes feature observations
  - features are independent, given the class $X_j \perp\!\!\!\perp X_{j'}|C$.
    Prediction:
    * $V = \{X_1 = o_1, ..., X_D = o_D\}$ what is $p(C|V)$?
    * $q(X_j|C,V) = \begin{cases} p(X_j|C), & X_j = o_j = p(X_j|C)\delta(X_j = o_j) \\ 0, & \text{otherwise} \end{cases}$

  * $p(C|X_1,...,X_D) \propto p(X_1,...,X_d|C)p(C)$
  * $p(C|X_1 = o_1,...,X_D = o_D) = q(C|X_1,...,X_D)/\sum q(C = k|X_1,...,X_D)$

  problem: variable elimination has exponential complexity in the size of U (# of variables to eliminate)

- solution, idea: use the distributive law to minimize the complexity in the sum over products
  - $ab + ac = a(b + c)$ (three operations vs 2). In a complex network, proper grouping of terms can give dramatic gains:

    $$q(X_1,X_3) = \sum_{X_2,X_4} q_1(X_1,X_2,X_3)q(X_1,X_4)$$

    assume each X_j takes b different values. Then we have in total $b^2 \cdot 2b^2 = 2b^4$, but grouping

    $$\sum_{X_2} q_1(X_1,X_2,X_3) \sum_{X_4} q_2(X_1,X_4)$$

    gives us $b^3 + 2b^2$

- but: finding the optimal grouping is in general NP-hard.
- but: it is easy for a very important special case: if the BN is a tree.
  $\Rightarrow$ use "**belief propagation algorithm**" to find the optimal computation mechanically
- why is this relevant:
  - many practical BNs are trees
  - some can be transformed into trees by duplicating and grouping variables using "junction tree algorithm"
  - belief propagation also works when the graph is not a tree ("loopy belief propagation") (relevant cycles of the undirected graph corresponding to BN) but gives approximate solution (quality is application dependent)
- belief propagation is also known as message passing.
  - it passes around (between neighboring nodes) reduced (marginalized) probability tables

17 Lecture 19/06

- marginalization in Bayesian networks is generally done by "variable elimination"
- but: VE has exponential complexity in the # of eliminated variables when applied naively
- We can take advantage of the distributive law to group sums and products such that the complexity is minimized.
- "belief propagation" finds an optimal evaluation order automatically for tree-shaped graphs.
- original algorithm [Pearl, 1988] for BN, here: use the generalization to factor graphs by [Kschischang et al., 2001]
  - factor graph:
    * two types of nodes: variables (X_j), factors (functions) f_l (small squares)

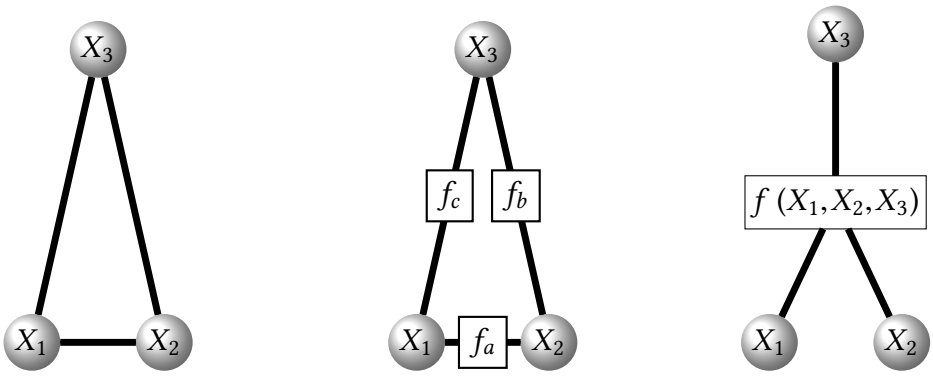


Figure 17.1: The left figure shows the undirected graph for the middle and right picture with single clique potential $\Phi\,(X_1,X_2,X_3)$. The picture in the middle is the factor graph of $\Phi\,(X_1,X_2,X_3) = f_a\,(X_1,X_2)\;f_b\,(X_2,X_3)\;f_c\,(X_3,X_1)$ and the right figure is the factor graph for $\Phi\,(X_1,X_2,X_3) = f\,(X_1,X_2,X_3)$.

    * bipartite, i.e. edges are only between nodes of different types (undirected edges)
    * edge X_j—f_l exist $\Leftrightarrow$ X_j is an argument of f_l
    * example: Burglary alarm
  - Variable elimination is implemented by "**message passing**". Each node sends and receives messages to/from its neighbors. messages = reduced probability tables = partial variable elimination

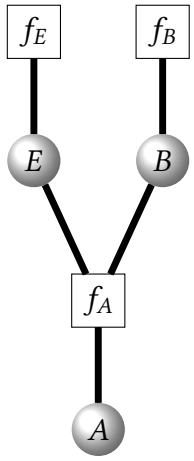


Figure 17.2: Factor graph for the Burglary alarm example

– message passing has two simple rules:

1. variable to factor:

$$\mu_{X_j \to f_l}(X_j) = \prod_{f' \in Ne(X_j) \setminus f_l} \mu_{f' \to X_j}(X_j)$$

2. factor to variable:

$$\mu_{f_l \to X_j}(X_j) = \sum_{\{X'\} \in Ne(f_l) \setminus X_i} f_l(Ne(f_l)) \prod_{\{X'\} \in Ne(f_l) \setminus X_j} \mu_{X' \to f_l}(X')$$

Note: $Ne(\cdot)$ *represents the neighborhood of* $\cdot$

– shorthand notation $\{X'\} \in \mathcal{N}(f_l) \setminus X_j =: N_{l \setminus j}$

– Summary of the principle: message sent by a node to a receiver depends on the messages coming in from all neighbors of the sender *except* the receiver.

– message scheduling: a $\underbrace{\text{message}}_{\text{LHS}}$ can be sent as soon as all required $\underbrace{\text{incoming messages}}_{\text{RHS}}$ at the sender have been received
⇒ leaf nodes in a tree can send messages to its only neighbor without waiting or prerequisites
⇒ message passing proceeds in rounds:
. round 0: send messages from leaf nodes
. round t: send messages where last prerequisites were received in round $(t-1)$
termination time T = diameter of the tree (longest path)

- finalization rule: compute the marginals from all incoming messages of the variable nodes

$$q(X_j) = \prod_{f \in Ne(X_j)} \mu_{f \to X_j}(X_j)$$
$$p(X_j) = q(X_j) / \sum_{X_j'} q(X_j')$$

- For small graphs, this is no improvement over naive variable elimination, but it is easy to implement as an algorithm for arbitrary large graphs "**sum-product algorithm**".
- If the graph has cycles (no tree) ⇒ belief propagation is generalized to "**loopy belief propagation**".
  * due to cycles, a node can receive messages through the same edge *repeatedly* (either through alternative paths or by repeated winding around a cycle)
    ⇒ Whenever this happens, the node sends *updated* messages through the other edges.
  * no hard termination condition, but converges to a fixed point (local optimal solution) = reasonable approximation of full variable elimination, but depends on the initial state
  * two possibilities to incorporate evidence (observed states)
    · set $f_l(X) = 0$ if the evidence is incompatible with the variable states $f(A,B,C), A = 1 \; \forall B,E : f(A = 0,B,E) = 0$
    · attach unary factors to the variable nodes where we have evidence
    example: Alice's children, version (2) (We know that at least one of the children is a boy.) [1]

17.1 Temporal Models/Belief Networks

- causality goes past → present → future, we know the arrow directions
  ⇒ simplest possible model: Markov chain (MC) $PA(X_j) := \{X_{j-1},...,X_{j-M}\}$, M-th order Markov chain
  $M = 1 : (X_1) \to (X_2) \to (X_3) \to ...$
  M represents how much memory the system has, for example $M = 1$ there is no memory and the future only depends on the present, not the past
  If $p(X_j|PA(X_j)) = p(X_j|X_{j-1}) = R^{(j)}$, $R_{i,i'} = p(X_j = a_i|X_{j-1=i'})$
  $p(X_D|X_1) = R^{(D)} \cdots R^{(2)} p(X_1)$

- The system is *stationary* if all X_j have the same set of states and $R^{(j)} = R^{(j')}$.

[1] there is again a graph that won't be reproduced here

- a stationary system can be represented as a *probabilistic state machine*, example: the weather homework from `exercise06.pdf`

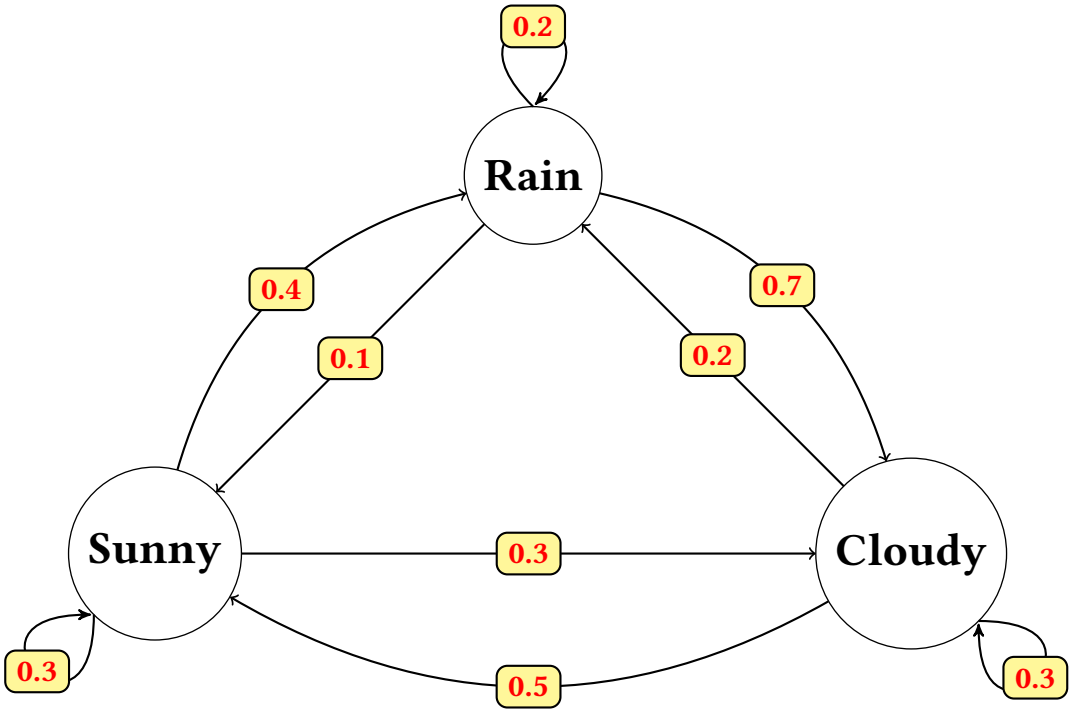


Figure 17.3: Probabilistic State Machine for weather homework.

- for $D \to \infty$ $p(X_D|X_1)$ becomes the stationary distribution p_∞ which can be shown to be independent of X_1.
'stationary' means that it doesn't change anymore, $p_\infty = Rp_\infty$, i.e. p_∞ is the eigenvector of R corresponding to eigenvalue 1.

18 Lecture 24/06

This lecture is actually two lectures.

18.1 Markov Chains

$(X_1) \to (X_2) \to \ldots \to (X_D)$ if stationary: probabilistic state machine, state transition matrix R, stationary distribution $p_\infty = Rp_\infty$ eigenvector of R with eigenvalue 1.

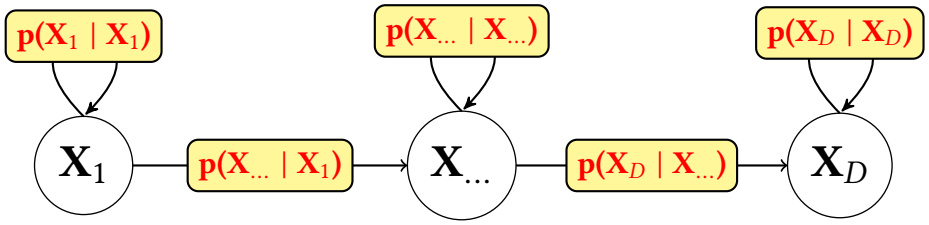


Figure 18.1: Schematic of Markov Chain

18.1.0.1 Google PageRank algorithm

- a search engine works in two steps:
  1. find pages related to the query
  2. rank these pages according to importance/relevance
- how to measure importance?
  - today: probably use actual click statistics
  - $\approx$ 1995 : no statistics available $\Rightarrow$ simulate user clicks by a random walk = monkey user clicking at random
    $\Rightarrow$ consider pages as important if they are frequently reached in the random walk $\Leftrightarrow$ high prob. in p_∞.
- define transition matrix:
  - state k = user looks at webpage k, $k = 1, \ldots, C$, C = # of pages

  1. the monkey clicks on each link on page k uniformly at random
  2. if page k contains no links the monkey goes to any page uniformly at random.

3. on any page there is a constant prob λ that the monkey goes to any other page uniformly at random instead of clicking a link.

- adjacency matrix A: $A_{k'k} = \begin{cases} 1, & \text{page } k \text{ links to page } k' \\ 0, & \text{else} \end{cases}$
- out-degree of page k: $\sum_{k'} A_{k'k}$
- from 1. and 2. we define the transition matrix R':

$$P(X_j = k' | X_{j-1} = k) = R'_{k'k} = \begin{cases} A_{kk'} / \sum_{k'} A_{k'k}, & \text{if page } k \text{ has Links} \\ 1/C, & \text{otherwise} \end{cases}$$

$$\forall k : \sum_{k'} R'_{k'k} = 1.$$

- incorporate rule 3. to define "Google matrix" R:

$$R_{k'k} = \lambda \frac{1}{C} + (1 - \lambda) R'_{k'k}$$

- importance of page k: $(p_\infty)_k$ (numerically difficult if $C >> 1$)

18.2 Hidden Markov Models (HMM)

- make Markov chain a bit more complicated: the interesting variables X_j are not observable anymore
- instead we can observe features Y_j that depend on the "**hidden**" or "**latent**" **variables** X_j (dependency is causal, but probabilistic)
  $\Rightarrow$ BN: $(X_1) \to (X_2) \to \ldots \to (X_D)$ and $(X_i) \to (Y_i), \forall i$
  probability factorizes: $p(X_1, \ldots, X_D, Y_1, \ldots, Y_D) = \prod_j p(X_j | X_{j-1}) p(Y_j | X_j)$
- example:
  - speech recognition: $X_1, \ldots, X_D$ is what the speaker said (phonemes), $Y_1, \ldots, Y_D$ is what you heard
  - wireless communication (e.g. cell phones): X_i symbol sent, Y_j symbol received
- major task:
  - compute marginals for the hidden states, given observations $\underline{Y} = \underline{O}$[1]: $p(X_j | \underline{Y} = \underline{O})$, $j = 1, \ldots, D$

[1] observed state vector

  - compute the most likely sequence of hidden states, given observations

$$\hat{\underline{a}} = \arg\max_{\underline{a}} p(\underline{X} = \underline{a} | \underline{Y} = \underline{O})$$

    (note: the global ML sequence $\hat{\underline{a}}$ generally differs from the sequence of pointwise maxima $\tilde{\alpha}_j = \arg\max_a p(X_j | \underline{Y} = \underline{O})$)

  - learn the probabilities $p(X_j|X_{j-1})$ and $p(Y_j|X_j)$ from training data

- compute pointwise marginals

$$\begin{aligned} p(X_j|\underline{Y} = \underline{O}) &= \frac{\sum_{\underline{X}\setminus X_j} p(X_1,...,X_D, Y_1 = O_1,...,Y_D = O_D)}{\sum_{\underline{X}} p(X_1,...,X_D, Y_1 = O_1,...,Y_D = O_D)} \\ &\propto \sum_{\underline{X}\setminus X_j} p(X_1,...,X_D, Y_1 = O_1,...,Y_D = O_D) = q(X_j|\underline{Y} = \underline{O}) \end{aligned}$$

- factor graph

$$\begin{aligned} f_1(X_1) &= p(X_1) && \text{prior of } X_1 \\ f_j(X_j, X_{j-1}) &= p(X_j|X_{j-1}) && \text{transition probability} \\ g_j(Y_j, X_j) &= p(Y_j|X_j) && \text{observation probability} \end{aligned}$$

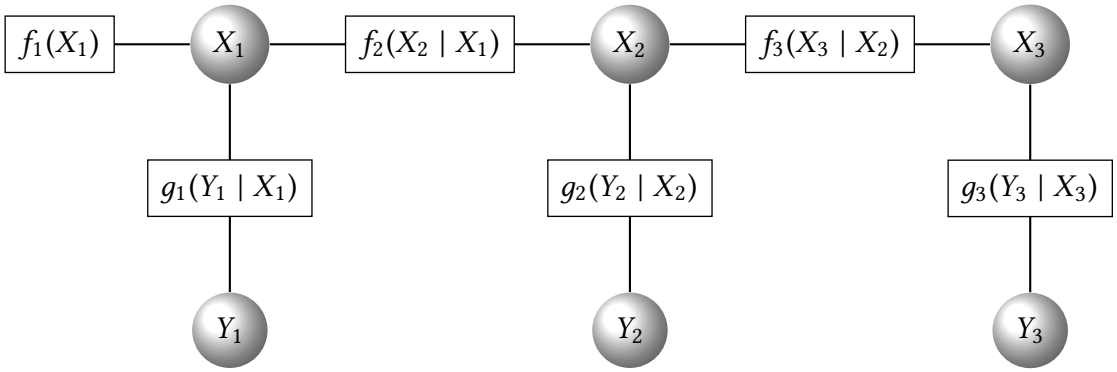


Figure 18.2: Illustration of the factor graph for a HMM.

- BP message passing rules:

$$\begin{aligned} \mu_{X\to f}(X) &= \prod_{f'\in NE(X)\setminus f} \mu_{f'\to X}(X) \\ \mu_{f\to X}(X) &= \sum_{X'\in NE(f)\setminus X} f(\underline{X}) \underbrace{\prod_{X'\in NE(f)\setminus X} \mu_{X'\to f}(X')}_{(*)} \end{aligned}$$

- all factor nodes have degree 2 ("pairwise factors") the products ($*$) contain only a single term, we can simplify message passing by concatenating two consecutive messages[2]

$$\begin{aligned}\gamma_j(X_j) &= (\mu_{Y_j \to g_j} \circ \mu_{g_j \to X_j})(X_j)\\ \alpha_j(X_j) &= (\mu_{X_{j-1} \to f_j} \circ \mu_{f_j \to X_j})(X_j)\\ \beta_j(X_j) &= (\mu_{X_{j+1} \to f_{j+1}} \circ \mu_{f_{j+1} \to X_j})(X_j)\end{aligned}$$

- message schedule:
  - round 0: send all γ messages (in parallel) and $\alpha_1(X_1)$ (these messages have no prerequisites, because f_1 and Y_j are leaves)
  - round j: send $\alpha_{j+1}(X_{j+1})$ and $\beta_{D-j}(X_{D-j})$

  $\Rightarrow$ forward-backward algorithm

- expand the message definitions:

$$\begin{aligned}\gamma_j(X_j) &= \underbrace{\sum_{X' \in NE(g_j) \setminus X_j}}_{Y_j} g_j(Y_j, X_j) \underbrace{\mu_{Y_j \to g_j}(Y_j)}_{\delta(Y_j = O_j)} = g_j(Y_j = O_j, X_j)\\ \alpha_j(X_j) &= \underbrace{\sum_{X' \in NE(f_j) \setminus X_j}}_{X_{j-1}} f_j(X_j, X_{j-1}) \underbrace{\mu_{X_{j-1} \to f_j}(X_{j-1})}_{\underbrace{\prod_{f' \in NE(X_{j-1}) \setminus f_{j-1}} \mu_{f' \to X_{j-1}}(X_{j-1})}_{\alpha_{j-1}(X_{j-1})\gamma_{j-1}(X_{j-1})}}\\ &= \sum_{X_{j-1}} f_j(X_j, X_{j-1}) \alpha_{j-1}(X_{j-1}) \gamma_{j-1}(X_{j-1})\\ \alpha_1(X_1) &= p(X_1)\\ \beta_j(X_j) &= \underbrace{\sum_{X' \in NE(f_{j+1}) \setminus X_j}}_{X_{j+1}} f_{j+1}(X_{j+1}, X_j) \mu_{X_{j+1} \to f_{j+1}}(X_{j+1})\\ &= \sum_{X_{j+1}} f_{j+1}(X_{j+1}, X_j) \gamma_{j+1}(X_{j+1}) \beta_{j+1}(X_{j+1})\\ \beta_D(X_D) &= 1\end{aligned}$$

- algorithm:
  - round 0: propagate γ_j and α_1
  - round j: propagate α_{j+1} and β_{D-j}

[2] $\circ$ represents the concatenation

  - finalization: compute marginals:

$$q(X_j|\underline{Y} = \underline{O}) = \prod_{f' \in NE(X_j)} \mu_{f' \to X_j}(X_j) = \alpha_j(X_j)\beta_j(X_j)\gamma_j(X_j)$$

- Remark: most textbooks derive the F/B algorithm directly, without factor graphs. Then, the messages α_j and γ_j are usually merged to $\tilde{\alpha}_j(X_j) = \alpha_j(X_j)\gamma_j(X_j)$.
- computing $p(X_j|\underline{Y} = \underline{O})$ is also called "smoothing", intuition: X and Y have the same statespace (symbols of an alphabet), $\underline{Y}$ is a noisy version of the true message $\underline{X} \Rightarrow p(X_j|\underline{Y} = \underline{O})$ is a denoised (smoothed) version of $\underline{Y} = \underline{O}$
- in smoothing, we condition on all observations $Y_1 = O_1, ..., Y_D = O_D$
- in an online system, we can only condition on the observations received so far $Y_1 = O_1, ..., Y_j = O_j$ (we do not yet know the values of future observations $Y_{j+1}, ..., Y_D$) $\Rightarrow$ (online) filtering
- Derive filtering from scratch and show that it gives the same results as belief propagation on factor graphs:

$$\begin{aligned}
p(X_j|Y_1 = O_1, ..., Y_j = O_j) &= p(X_j, .Y_1, ..., Y_j)/p(Y_1, ..., Y_j) \\
q(X_1|Y_1 = O_1, ..., Y_j = O_j) &= p(X_j, Y_1 = O_1, ..., Y_j = O_j) \\
p(X_j, Y_1, ..., Y_j) &= \sum_{\underline{X}\setminus X_j, \underline{Y}\setminus\{Y_1,...,Y_j\}} p(X_1, .., X_D, Y_1, ..., Y_D) \\
&= \sum_{X_{j-1}} p(X_{j-1}, X_j, Y_1, ..., Y_j).
\end{aligned}$$

- BN factorization and Bayes rule:

$$\begin{aligned}
&p(X_{j-1}, X_j, Y_1, ..., Y_j) \\
&\quad = p(Y_j|X_{j-1}, X_j, ..., Y_{j-1})p(X_j|X_{j-1}, Y_1, ..., Y_{j-1})p(X_{j-1}, Y_1, ..., Y_{j-1}) \\
&\quad = p(Y_j|X_j)p(X_j|X_{j-1})p(X_{j-1}, Y_1, ..., Y_{j-1})
\end{aligned}$$

$$\begin{aligned}
&\underbrace{q(X_j|Y_1 = O_1, ..., Y_j = O_j)}_{=:\tilde{\alpha}_j(X_j)} \\
&\quad = \sum_{X_{j-1}} p(Y_j = O_j|X_j)p(X_j|X_{j-1})q(X_{j-1}|Y_1 = O_1, ..., Y_{j-1} = O_{j-1}) \\
&\quad = p(Y_j = O_j|X_j) \sum_{X_{j-1}} p(X_j|X_{j-1}) \underbrace{q(X_{j-1}|Y_1 = O_1, ..., Y_{j-1} = O_{j-1})}_{\tilde{\alpha}_{j-1}(X_{j-1})}
\end{aligned}$$

$$\tilde{\alpha}_j(X_j) = \underbrace{p(Y_j = O_j|X_j)}_{\gamma_j(X_j)} \underbrace{\sum_{X_{j-1}} p(X_j|X_{j-1})\tilde{\alpha}_{j-1}(X_{j-1})}_{\alpha_j(X_j)}$$

translate this to our notation:

$$q(X_j|Y_1 = O_1, ..., Y_j = O_j) = \underbrace{\gamma_j(X_j)}_{\text{corrector}} \underbrace{\alpha_j(X_j)}_{\text{predictor}}$$

$$\alpha_j(X_j) = \sum_{X_{j-1}} \underbrace{p(X_j|X_{j-1})}_{f_j(X_j, X_{j-1})} \gamma_{j-1}(X_{j-1})\alpha_{j-1}(X_{j-1})$$

predictor: updated prior for X_j representing our expectations of the next X_j
corrector: noisy observation of X_j
$q(X_j|Y_1, ..., Y_j)$: compromise between our expectations and observations

- Kalman filter[3]:
  - HMM with continuous state space for X and Y. $X_j \in \mathbb{R}^N$, $Y_j \in \mathbb{R}^{N'}$
  - transitions are defined by linear (matrix) equations + additive Gaussian noise

  $\Rightarrow$ nice analytical matrix expressions for all probabilities

- task: determine the most likely sequence of $\underline{X}$, given $\underline{Y} = \underline{O}$, *"maximum likelihood detection"*

$$\hat{a} = \arg\max_a p(\underline{X} = \underline{a}|\underline{Y} = \underline{O})$$

  - "decoding the noise received message $\underline{Y}$"
  - **Viterbi-algorithm**, widely used in all digital communications

- as usual, instead of maximizing the likelihood, we minimize the negative log-likelihood
  - redefine the factors:

$$f_j(X_j, X_{j-1}) = -\log p(X_j|X_{j-1})$$
$$g_j(Y_j, X_j) = -\log p(Y_j|X_j)$$

  - objective

$$\hat{\alpha} = \arg\max_a p(\underline{X} = \underline{a}|\underline{Y} = \underline{O})$$
$$\Leftrightarrow \hat{a} = \arg\min_a \left(\sum_{j=1}^{D} g_j(Y_j = O_j, X_j = a_j) + \sum_{j=1}^{D} f_j(X_j = a_j, X_{j-1} = a_{j-1}) \right)$$

  - surprisingly, this can also be solved by a variant of belief propagation
  - crucial insight: sum-product algorithm (= standard belief propagation) automatically groups terms to minimize computations, but this grouping only relies on the *algebraic* properties of addition and multiplication

[3] might be treated later if there is enough time

$\Rightarrow$ it works for other tasks that have the required algebraic properties

- specifically addition and multiplication form a *semi-ring*
  Def of a semi-ring[4]: $(R, \oplus, \otimes)$ is a semi-ring over domain R, if

  (i) $\oplus$ is a commutative and associative operator with neutral element "0"

  (ii) $\otimes$ is a commutative and associative operator with neutral element "1"

  (iii) $\oplus$ and $\otimes$ are distributive: $(a \otimes b) \oplus (a \otimes c) = a \otimes (b \oplus c)$

  (iv) "0" annihilates $\otimes$: $a \otimes$ "0"="0"

- This obviously applies to ordinary addition and multiplication with "0" = 0, "1" = 1.

- for maximum a posteriori estimation = minimal negative log-likelihood we need the "min-sum algebra"

  (i) $a \oplus b = \min(a, b)$, "0" = ∞ ($\min(a, \infty) = a$)

  (ii) $a \otimes b = a + b$, "1" = 0, $a + 0 = a$

  (iii) $\min(a + b, a + c) = a + \min(b, c)$

  (iv) $a \otimes$ "0" $= a + \infty = \infty$

- using this algebra, belief propagation becomes the *"min-sum algorithm"*, intuition: replace all products with sums and all sums with "min" in the sum product algorithm
  $\Rightarrow$ reuse the messages α, β, γ and update scheduling

  - round 0: initialization

$$\gamma_j(X_j) = g_j(Y_j = O_j, X_j) = -\log p(Y_j = O_j | X_j)$$
$$\beta_D(X_D) = 0 = \text{"1" of min-sum}$$

[4] add inverses for a "complete" ring

– backward sweep, rounds $1, \ldots, D-1$ (compute the likelihood right to left):

$$\min_{\{X_1,\ldots,X_D\}} \left(\sum_{j=1}^{D-1} g_j(Y_j = O_j, X_j) + f_j(X_j, X_{j-1}) \right) =$$

$$= \min_{\{X_1,\ldots,X_{D-1}\}} \left(\sum_{j=1}^{D} g_j(X_j) + f_j(X_j, X_{j-1}) \right)$$

$$+ \underbrace{\min_{X_D} \left(\underbrace{g_D(X_D)}_{\gamma_D(X_D)} + f_D(X_D, X_{D-1}) + \underbrace{\beta_D(X_D)}_{=0} \right)}_{=:\beta_{D-1}(X_{D-1})}$$

$$= \min_{\{X_1,\ldots,X_{D-2}\}} \left(\sum_{j=1}^{D-2} g_j(X_j) + f_j(X_j, X_{j-1}) \right)$$

$$+ \underbrace{\min_{X_{D-1}} \left(g_{D-1}(X_{D-1}) + f_{D-1}(X_{D-1}, X_{D-2}) + \beta_{D-1}(X_{D-1}) \right)}_{\beta_{D-2}(X_{D-2})}$$

[…]

$$\beta_j(X_j) = \min_{X_{j+1}} \left(g_{j+1}(X_{j+1}) + f_{j+1}(X_{j+1}, X_j) + \beta_{j+1}(X_{j+1}) \right)$$

finally:

$$\hat{a}_1 = \arg\min_{X_1 = a_1} \left(\underbrace{\alpha_1(X_1)}_{-\log p(X_1 = a_1)} + \beta_1(X_1) + \underbrace{\gamma_1(X_1)}_{-\log p(Y_1 = O_1 | X_1 = a_1)} \right)$$

– forward sweep: propagate the solution from left to right

$$\alpha_j(X_j) = f_j(X_j, X_{j-1} = \hat{a}_{j-1})$$

$$\hat{a}_j = \arg\min_{X_j = a_j} \left(\alpha_j(X_j) + \beta_j(X_j) + \gamma_j(X_j) \right)$$

"**Viterbi algorithm**"

- remark 1: one can also do a forward sweep first, followed by a backward sweep ⇒ same result
- remark 2: in principle, one can also use this to maximize the likelihood directly (instead of negative log-likelihood)
  - use the "max-product algebra" $\oplus = \max$, "0" $= -\infty$, $\otimes = \cdot$, "1" $= 1$ to get the "*max-product algorithm*"
  - numerically not advisable, because it involves products of small numbers ⇒ loss of precision
    better: work with logarithms and min-sum algorithm

19 Lecture 26/06

Note: This lecture contains a lot of equations given in a very short amount of time. The frequency of typing errors is therefore probably higher. Proceed with caution!

- Difference between **point-wise marginals** $p(X_j|\underline{Y} = \underline{O})$ and the **global MAP solution** $\underline{\hat{a}} = \arg\min_{\underline{a}} p(\underline{X} = \underline{a}|\underline{Y} = \underline{O})$
- Example 1:
  - consider a problem where $X_j \in 1,\ldots,C$ and labels are ordinal (e.g. discretized values of a continuous phenomenon) and $Y_j \in 1,\ldots,C$ are noisy observations of the X_j.
  - Point-wise marginals describe the local uncertainty about X_j. For example $\bar{X}_j = \mathbb{E}[X_j]$, std$[X_j]$ can be computed from $p(X_j|\underline{Y} = \underline{O})$ and give local error bars[1].
  - The MAP solution is the most likely global solution, within the local error bars.
- Example 2: consider a random walk in a maze: the room entered most frequently[2] is not necessarily part of the most likely way out.

19.1 Learning the parameters (= transition probabilities) of a HMM

- case 1: supervised training: the states X_j are known in the training data → estimate the transition probabilities by counting transition frequencies
- case 2: unsupervised training: X_j are unknown
  - example: wildlife photographer wants to get footage of a interesting chimpanzee. $X_j = \begin{cases} 1, & \text{chimp is north of the river} \\ 2, & \text{chimp is south of the river} \end{cases}$ at time j.
  - photographer needs to predict X_{j+1} to set up equipment at the right location
  - observations Y_j: any evidence where chimp is/was (sightings, excrements, ...)

$$Y_j = \begin{cases} 0: & \text{no evidence or contradictory evidence on day } j \\ 1: & \text{was seen north (but may be wrong!)} \\ 2: & \text{was seen south (but may be wrong!)} \end{cases}$$

[1] example plot here
[2] max of point-wise marginals

  - task: create a HMM from N observation sequences

- stationary HMM: transition probabilities are independent of j and constant $\rho_k = p(X_1 = k)$ prior $k \in \{N, S\}$; $\pi_{k'k} = p(X_j = k' | X_{j-1} = k)$; $\mu_{mk} = p(Y_j = m | X_j = k)$, $m \in \{0, N, S\}$
  full parameter set: $\theta = \{\rho, \pi, \mu\}$

- training data:
  - N sequences, $n = 1, ..., N$, length D_n $j = 1, .., D_n$
  - Observations $\underline{Y} = \underline{O}$, $Y_j^{(n)} = O_j^{(n)} \in \{0, 1, 2\}$
  - hidden states $X_j^{(n)}$ are unknown

- maximum likelihood parameter estimation[3]:

$$\hat{\theta} = \arg\max_\theta p_\theta(\underline{Y} = \underline{O}) = \arg\max_\theta \sum_{\underline{X}} p(\underline{X}, \underline{Y} = \underline{O})$$

- no closed form solution, need an iterative algorithm: *EM* **algorithm** (known from Gaussian mixture models/clustering in *ML*1)
  given a current guess θ, try to get a better guess θ', such that $p_{\theta'}(\underline{Y} = \underline{O}) \geq p_\theta(\underline{Y} = \underline{O})$

- **Kullback-Leibler** (KL) **divergence** between two distributions $p_1(\underline{\omega})$ and $p_2(\underline{\omega})$ over the same domain $\underline{\omega} \in \Omega$

$$KL(p_2|p_1) = \sum_\omega p_1(\omega) \log \frac{p_1(\omega)}{p_2(\omega)} \geq 0$$

  We choose: $p_1(\underline{\omega}) = p_\theta(\underline{X}|\underline{Y} = \underline{O}) = \frac{p_\theta(\underline{X}, \underline{Y}=\underline{O})}{p_\theta(\underline{Y}=\underline{O})}$, $p_2(\underline{\omega}) = p_{\theta'}(\underline{X}|\underline{Y} = \underline{O}) = \frac{p_{\theta'}(\underline{X}, \underline{Y}=\underline{O})}{p_{\theta'}(\underline{Y}=\underline{O})}$

$$\begin{aligned} KL(p_2|p_1) &= \sum_{\underline{X}} \frac{p_\theta(\underline{X}, \underline{Y} = \underline{O})}{p_\theta(\underline{Y} = \underline{O})} \log \frac{p_\theta(\underline{X}, \underline{Y} = \underline{O}) p_{\theta'}(\underline{Y} = \underline{O})}{p_{\theta'}(\underline{X}, \underline{Y} = \underline{O}) p_\theta(\underline{Y} = \underline{O})} \\ &= \log \frac{p_{\theta'}(\underline{Y} = \underline{O})}{p_\theta(\underline{Y} = \underline{O})} + \frac{1}{p_\theta(\underline{Y} = \underline{O})} \sum_{\underline{X}} p_\theta(\underline{X}, \underline{Y} = \underline{O}) \log \frac{p_\theta(\underline{X}, \underline{Y} = \underline{O})}{p_{\theta'}(\underline{X}, \underline{Y} = \underline{O})} \\ &\geq 0 \end{aligned}$$

  abbreviation:

$$Q(\theta_1, \theta_2) = \sum_{\underline{X}} p_{\theta_1}(\underline{X}, \underline{Y} = \underline{O}) \log p_{\theta_2}(\underline{X}, \underline{Y} = \underline{O})$$

$$KL(p_2|p_1) = \log \frac{p_{\theta'}(\underline{Y} = \underline{O})}{p_\theta(\underline{Y} = \underline{O})} + \frac{Q(\theta, \theta) - Q(\theta, \theta')}{p_\theta(\underline{Y} = \underline{O})} \geq 0$$

[3] marginalize over all possible assignments $\underline{X}$

$$\Rightarrow \underbrace{\frac{Q(\theta,\theta') - Q(\theta,\theta)}{p_\theta(\underline{Y} = \underline{O})}}_{\text{lower bound for RHS}} \leq \log \underbrace{\underbrace{\frac{p_{\theta'}(\underline{Y} = \underline{O})}{p_\theta(\underline{Y} = \underline{O})}}_{\geq 1 \text{ desired}}}_{\geq 0 \text{ desired}}$$

$\Rightarrow$ Improve the objective $\frac{p_{\theta'}(\underline{Y}=\underline{O})}{p_\theta(\underline{Y}=\underline{O})}$ as much as possible by maximizing the lower bound.
$\Rightarrow$ define $\hat{\theta}' = \arg\max_{\theta'} Q(\theta,\theta')$

- EM algorithm[4] outline:
  1. define initial guess $\theta(0)$
  2. for $t = 1,\ldots,T$ (or until convergence)
     - **E-step:** define
     $$Q(\theta^{(t-1)},\theta') = \mathbb{E}_{\theta^{(t-1)}}[\log p_{\theta'}] = \sum_{\underline{X}} p_{\theta^{(t-1)}}(\underline{X},\underline{Y} = \underline{O}) \log p_{\theta'}(\underline{X},\underline{Y} = \underline{O})$$
     - **M-step:** find $\hat{\theta}' = \arg\max_{\theta'} Q(\theta(t-1),\theta')$
     - set $\theta(t) = \hat{\theta}'$
- thanks to the BN factorization of our HMM; the calculations simplify tremendously
$$\begin{aligned} Q(\theta,\theta') &= \sum_{\underline{X}} p_\theta(\underline{X},\underline{Y} = \underline{O}) \log \frac{\underbrace{p_{\theta'}(\underline{X},\underline{Y} = \underline{O})}}{\prod_n \prod_j p_{\theta'}(X_j|X_{j-1}) p_{\theta'}(Y_j = O_j|X_j)} \\ &= \sum_{\underline{X}} p_\theta(\underline{X},\underline{Y} = \underline{O}) \sum_{n=1}^{N} \left(\log p_{\theta'}(X_1^{(n)}) + \sum_{j=2}^{D_n} \log p_{\theta'}\left(X_j^{(n)}|X_{j-1}^{(n)}\right) \right. \\ &\quad \left. + \sum_{j=1}^{D_n} \log p_{\theta'}\left(Y_j^{(n)} = O_j^{(n)}|X_j^{(n)}\right) \right) \end{aligned}$$
- when minimizing w.r.t $\theta' = \{\rho',\pi',\mu'\}$ we also need to preserve the normalization of these probabilities $\Rightarrow$ Lagrangian
$$\begin{aligned} \mathcal{L}(\theta') &= \sum_{\underline{X}} p_\theta(\underline{X},\underline{Y} = \underline{O}) \sum_{n=1}^{N} \left(\log \underbrace{p_{\theta'}(X_1^{(n)})}_{\rho'} + \sum_{j=2}^{D_n} \log \underbrace{p_{\theta'}\left(X_j^{(n)}|X_{j-1}^{(n)}\right)}_{\pi'} \right. \\ &\quad \left. + \sum_{j=1}^{D_n} \log \underbrace{p_{\theta'}\left(Y_j^{(n)} = O_j^{(n)}|X_j^{(n)}\right)}_{\mu'} \right) + \lambda_\rho \left(1 - \sum_k \rho'_k\right) \\ &\quad + \sum_k \left[\lambda_k \left(1 - \sum_{k'} \pi'_{k'k}\right) + \eta_k \left(1 - \sum_m \mu'_{mk}\right) \right] \end{aligned}$$

[4] E for expectation, M for maximization

- optimize by setting the derivative to 0

$$\frac{\partial \mathcal{L}(\theta')}{\partial \pi'_{k'k}} = \sum_{\underline{X}} p_\theta(\underline{X}, \underline{Y} = \underline{O}) \sum_n \left[\sum_{j=2}^{D_n} \frac{\delta\left(X_j^{(n)} = k', X_{j-1}^{(n)} = k\right)}{\pi'_{k'k}} \right] - \lambda_k \stackrel{!}{=} 0$$

$$\pi'_{k'k} \lambda_k = \sum_{\underline{X}} \underbrace{p_\theta(\underline{X}, \underline{Y} = \underline{O})}_{p_\theta(\underline{X}|\underline{Y}=\underline{O}) p_\theta(\underline{Y}=\underline{O})} \sum_n \sum_{j=2}^{D_n} \delta(X_j^{(n)} = k', X_{j-1}^{(n)} = k)$$

$$= p_\theta(\underline{Y} = \underline{O}) \sum_n \sum_{j=2}^{D_n} \sum_{\underline{X}} p_\theta(\underline{X}|\underline{Y} = \underline{O}) \delta(X_j^{(n)} = k', X_{j-1}^{(n)} = k)$$

$$= p_\theta(\underline{Y} = \underline{O}) \sum_n \sum_{j=2}^{D_n} p_\theta(X_j^{(n)} = k', X_{j-1}^{(n)} = k | \underline{Y} = \underline{O})$$

$$\pi'_{k'k} \propto \sum_n \sum_{j=2}^{D_n} p_\theta(X_j^{(n)} = k', X_{j-1}^{(n)} = k | \underline{Y} = \underline{O})$$

This can be computed by a variant of the forward/backward algorithm ⇒ homework.
And then normalize: $\frac{\pi'_{k'k}}{\sum_{k'} \pi_{k'k}} \to \pi'_{k'k}$

$$\rho'_k \propto \sum_n p_\theta(X_1^{(n)} | \underline{Y} = \underline{O})$$

$$\mu'_{mk} \propto \sum_n \sum_{j=1}^{D_n} p_\theta(X_j^{(n)} = k, Y_j^{(n)} = m | \underline{Y} = \underline{O})$$

$$\propto \sum_n \sum_{j=1}^{D_n} \underbrace{p_\theta(X_j^{(n)} = k | \underline{Y} = \underline{O})}_{\text{standard F/B algorithm}} \mathbb{1}(Y_j^{(n)} = m)$$

- **Baum-Welch algorithm**: repeat until convergence:
  - compute marginals $p_\theta(X_j^{(n)} = k | \underline{Y} = \underline{O})$ and $p_\theta(X_j^{(n)} = k', X_{j-1}^{(n)} = k | \underline{Y} = \underline{O})$ under the current guess θ, using the forward-backward algorithm.
  - Update ρ', π', μ' by pretending that the marginals are the ground truth, using counting followed by normalization.
- BW converges only to a local optimum of $p_{\hat{\theta}}(\underline{Y} = \underline{O})$ ⇒ quality depends on the quality of the initial guess.
  - don't initialize with 0, unless this is a constraint of the model, because 0 probs stay 0 probs.
  - method 1:
    - random initialization: $\theta^{(0)}$ is a uninformative prior plus noise, e.g. $\pi(0)_{k'k} = (1-\lambda)\frac{1}{C} + \lambda \mathcal{U}(0,1)$

    * repeat with several initializations and keep best solution
- method 2: Viterbi counting
    * define counting matrices ρ^C, π^C, μ^C (count how often each transition occurred)
    * initialize the counting matrices $\pi^C = L_{C\times C}$, $\rho^C = L_{C\times 1}$, $\mu^C = L_{M\times C}$ (L = regularization parameter, minimum count for each transition)
    * for $t = 1, \ldots, T$
        · choose a training sequence n at random
        · define the transition matrices for the current iteration[5]:

$$\pi^{(t)} = (1-\lambda)norm(\pi^C) + \lambda norm(\mathcal{U}(0,1)_{C\times C})$$
$$\mu^{(t)} = (1-\lambda)norm(\pi^C) + \lambda norm(\mathcal{U}(0,1)_{M\times C})$$
$$\rho^{(t)} = (1-\lambda)norm(\rho^C) + \lambda norm(\mathcal{U}(0,1)_{C\times 1})$$

        · compute the MAP solution using Viterbi

$$\underline{X} = \underline{\hat{a}}, \underline{Y} \qquad = \underline{\hat{O}}, \hat{O}_j = \arg\max_{o_j} p(Y_j = O_j | X_j = \hat{a}_j)$$

        · update the counting matrices π^C, ρ^C, μ^C as if the MAP solution was the ground truth
    * init Baum-Welch with $norm(\pi^C)$, $norm(\rho^C)$, $norm(\mu^C)$

[5] $norm$ = normalization

20 Lecture 01/07

20.1 Causality

- causality is second major application of BN: (first: temporal models)
- three approaches to causality:
  - understanding the underlying mechanism (but: may be beyond our technological capabilities, too expensive, overkill, not yet possible in early stages of investigation)
  - statistical experiment: actively intervene into the system and analyze the effect statistically (but may be impossible, illegal, unethical, too expensive, too early)
  - observational analysis: measure properties
- example: cholera epidemics in London $\approx$ 1850
  - root cause (bacterium) was discovered in 1854 by Pacini (but not widely known), settled by Robert Koch in 1884
  - many hypothesis about cause (air quality, elevation of homes, social status,...)
  - Farr (head statistician): derived from data: $Y_i/Y_{i'} = (E'_i - E_0)/(E_i - E_0)$[1]
  - Snow (physician - inventor of anesthesia) believed (contrary to every one) that cholera was transmitted by contaminated water.
    $\Rightarrow$ identified highly significant association between illness and water pump
  - Farr: this hypothesis is very plausible, but not forced by the data in 1854
  - By 1866 Farr had collected enough data to confirm Snow's claim.
- Why is it difficult to derive causality here?
  - In physical systems (or standard ML) we can reset the experiment and repeat under different conditions.
    $\Rightarrow$ can identify[2] $\mathbb{E}[Y_i(X_i = a_1) - Y_i(X_i = a_0)] > 0 \Rightarrow a_1$ is better than a_0 ($\mathbb{E}$ goes over individuals i)
  - in living systems it is impossible to replay the data $\Rightarrow$ we can at best compute $\mathbb{E}_{X_i=a_1}[Y_i] - \mathbb{E}_{X_i=a_0}[Y_i]$ ($\mathbb{E}$ is over groups that received either treatment)
    How can we assure that this is $\approx \mathbb{E}[Y_i(X_i = a_1) - Y_i(X_i = a_0)]$?

[1] Y_i = prop i gets ill, E_i = elevation of i's home
[2] i is a data instance and $X_i = a_j$ are different interventions

  - we may not be able to actively intervene ⇒ the groups $\{X_i = a_1\}$ and $\{X_i = a_0\}$ are outside of our control

- goals of causality analysis:
  - prediction: What will happen if we apply treatment a or implement policy a? (e.g. raise cigarette taxes?)
  - counterfactual queries: What would have happened if X_i had been a_k instead of the actual $a_{k'}$?
  - decision making: Is it "better" to apply treatment a_k or $a_{k'}$ or no treatment at all?

- BNs are a very useful tool here if
  - they include all relevant variables (no hidden causes) → difficult to achieve
  - the arrows in the BN represent causal directions → "identifiability"
  - the probs are known → "parameter searching"

- always remember: association ("correlation") if **not** causality (but: analysis of *many* associations can reveal causality)

- Reichenbach's "common cause principle": if X and Y are associated then either X causes Y or Y causes X or there exists Z such that Z causes X and Y[3]
  - almost complete: correct when data X and Y are *not* conditioned on a common effect of X and Y ("explaining away effect")
  - example: are lectures useful? Top researchers find self-learning more efficient "selection bias"

- remember the definition of independence: $X \perp\!\!\!\perp Y \Leftrightarrow p(X,Y) = p(X)p(Y)$, $X \perp\!\!\!\perp Y|Z \Leftrightarrow p(X,Y|Z) = p(X|Z)p(Y|Z)$
  ⇒ no causation is possible between independent variables (no direct causation if conditional independent)

- *modeling of interventions* (active state changes by the investigator)
  - suppose we have a "correct" BN, i.e. $p(X_1,...,X_D) = \prod_j p(X_j|PA(X_j))$
  - Pearl's "**do**" **operator**: $\text{do}(X_j = a_j) = X_j$ was actively set into state a_j
    more general $\text{do}(X_j \sim \tilde{p}(X_j)) = X_j$ was actively drawn from $\tilde{p}(X_j)$ instead of $p(X_j|PA(X_j))$ [4]
  - "do" changes the factorization by replacing the distribution of X_j

$$p\left(X_1,...,X_D|\text{do}(X_j \sim \tilde{p}(X_j))\right) = \tilde{p}(X_j) \prod p(X_{j'}|PA(X_{j'}))$$

    graphical: incoming arcs of X_j are deleted

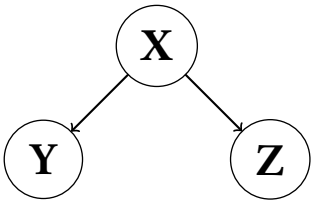


Figure 20.1: Common-Cause Model

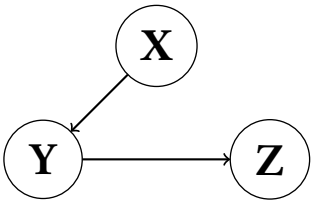


Figure 20.2: Causal-Chain Model

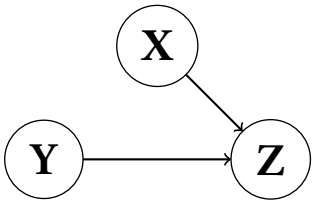


Figure 20.3: Common-Effect Model

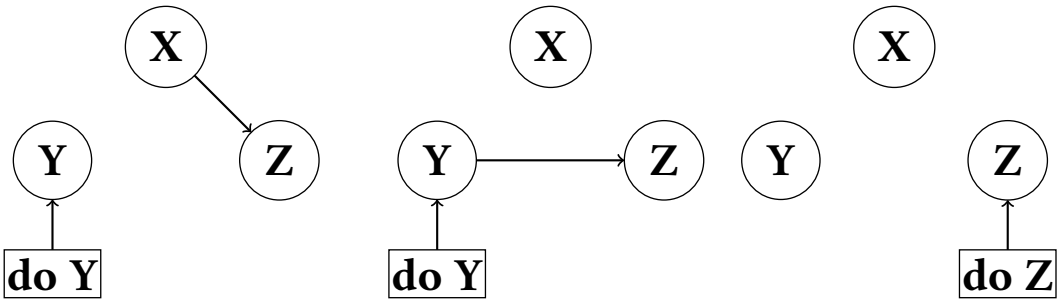


Figure 20.4: Influence of interventions on the three basic causal models

$\Rightarrow$ "structural interventions"[5]

- marginalization: (under hard intervention $\delta(X_j = a_j)$)

$$p(X_1,...,X_{j-1},X_{j+1},...,X_D|\text{do}(X_j = a_j)) = \sum_{X_j} p(X_1,...,X_D|\text{do}(X_j = a_j))$$
$$= \prod_{j' \neq j} p(X_{j'}| \underbrace{PA(X_{j'}, X_j = a_j)}_{X_j = a_j, \text{ whenever } X_j \in PA(X_{j'})})$$

- examples of the effect of "do"

$$\underbrace{p(...|X = a)}_{\text{cond prob}} = \underbrace{p(...|\text{do}(X = a))}_{\text{intervent prop}} \Leftrightarrow X \text{ is a root in the BN (no incoming arcs)}$$

- otherwise, the two probs are different:
  First example $p(X,Y) = p(Y)p(X|Y)$

$$p(Y|X = a) = \frac{p(Y)p(X = a|Y)}{\sum_Y p(Y)p(X = a|Y)}$$
$$\neq$$
$$p(Y|\text{do}(X = a)) = \sum_X p(Y)\delta(X = a) = p(Y)$$

[3] $X \to Y, Y \to X, X \leftarrow Z \to Y$
[4] hard intervention $X_j = a_j$ is a special case: $\tilde{p}(X_j) = \delta(X_j = a_j)$
[5] opposite: "parametric intervention" $p(X_j|PA(X_j)) = \tilde{p}(X_j|PA(X_j))$

Second Example[6][7] $p(X,Y,Z) = p(Z)p(X|Z)p(Y|X,Z)$

$$\begin{aligned} p(Y|X=a) = \sum_Z p(Y,Z|X=a) &= \sum_Z \frac{p(X=a,Y,Z)}{p(X=a)} \\ &=^{(*)} \sum_Z p(Z|X=a)p(Y|X=a,Z) \\ &\neq \\ p(Y|\mathrm{do}(X=a)) &= \sum_{X,Z} p(Z)\delta(X=a)p(Y|X,Z) \\ &= \sum_Z p(Z)p(Y|X=a,Z) \end{aligned}$$

$\Rightarrow$ be careful when making interventional predictions from observational probabilities ($\Rightarrow$ later)

- What's the "correct" BN?
  - Let G be a DAG with nodes $X_1,...,X_D$ and $p(X_1,...,X_D)$ a joint distribution, p is "Markov and faithful" with respect to G if:

$$[X \text{ d-separate } Y|Z]_G \Leftrightarrow [X \perp\!\!\!\perp Y|Z]_p.$$

  - in general, G is not uniquely determined:
    "Markov equivalence class": $\mathcal{M}(G) = \{G'|p \text{ is Markov and faithful w.r.t G'}\}$
  - "skeleton" of G: undirected graph obtained by removing all directions in G
  - "moral graph" of G. connect all parents by an undirected edge and remove all directions afterwards
  - **Theorem**: Two DAGs G, G' are Markov equivalent iff their skeletons and moral graphs are equal.
  - "Markov minimality": a DAG G is Markov minimal w.r.t p, if p is Markov and faithful w.r.t G but not w.r.t any subgraph of G[8]
  - "causal effect"[9]: there is a (total) causal effect from X_j to $X_{j'}$ in $p(X_1,...,X_D)$ iff $X_j \not\perp\!\!\!\perp X_{j'}$ in $p(X_1,...,X_D|\mathrm{do}(X_j \sim \tilde{p}(X_j)))$
  - "true causal graph":
    - Let G be a DAG s.t. $p(X_1,...,X_D)$ is Markov and faithful.
    - For all subsets of $S \subseteq \{X_1,...,X_D\}$ let

$$p_G(X_1,...,X_D|\mathrm{do}(S \sim \tilde{p}(S))) = \tilde{p}(S) \prod_{j \notin S} p(X_j|PA(X_j))$$

[6] $X \not\perp\!\!\!\perp Y$ partly due to direct effect $X \to Y$ partly due to common cause Z
[7] $(*)$ there are some basic calculations needed here, which are left as an exercise for the reader
[8] i.e. we cannot remove any edges from G without changing the probability
[9] intuitively: there is a path from $X_j \rightsquigarrow X_{j'}$, in the graph where all incoming arcs of X_j were removed

be the interventional distribution obtained from G and $p(X_1,...,X_D|\text{do}(S \sim \tilde{p}(S)))$ the true interventional distribution.

G is a true causal graph of p iff $\forall S, \forall \tilde{p}(S)$:

$$p_G(X_1,...,X_D|\text{do}(S \sim \tilde{p}(S)) = p(X_1,..,X_D|\text{do}(S \sim \tilde{p}(S)))$$

- Theorem: The minimal true causal graph for p is unique.

21 Lecture 03/07

21.1 Create BNs from data

- A true causal model reproduces the joint probability $p(X_1,...,X_D)$ (what you get from passive observation) and *all* interventional distributions $p(X_1,...,X_D|\text{do}(S \sim \tilde{p}(S), S \subset \underline{X}))$ (what you get from experiments).
- **Theorem**: The minimal true causal model is unique [Peters et al. 2014].
- problem: given data, infer the true model, or (weaker) a member of the Markov equivalence class
- *IC algorithm*[1]: idealized exact algorithm to identify the Markov equivalence class.
  - assumptions:
    - the nodes $X_1,...,X_D$ of the graph are known
    - the possess to a test oracle that answers (conditional) independence queries
  - steps:
    1. start with the complete graph and remove edges whose end points are (conditionally) independent
       $\Rightarrow$ skeleton of G
    2. detect "common effect" situations and orient the arrows accordingly ("v-structures")
    3. use BN constraints (e.g. cycle-free) to orient as many additional edges as possible
    4. perform experiments to obtain the orientation of the remaining edges (or orient arbitrarily)

  1. compute the skeleton (conceptual way, exponential complexity)
     - for all pairs $(X_j, X_{j'})$
       - for every subset $S \subseteq \underline{X} \backslash \{X_j, X_{j'}\}$ (including $\emptyset$)
         - ask the oracle if $X_j \perp\!\!\!\perp X_{j'}|S$, if yes
           - call $S \Rightarrow S_{jj'}$ and remember it[2]
           - remove the edge $(X_j, X_{j'})$ from graph

[1] IC = "inductive causation"
[2] remember the smallest one if there are multiple

- **Theorem**: This produces the true skeleton if the oracle is always correct. In practice, the oracle is some statistical test (⇒ later) that may be erroneous. ⇒ we get only an approximate skeleton
- *optimization 1:* PC-algorithm[3]
  - start with small conditioning sets S: CI-test ("conditional independence test") is faster and more accurate
  - showed that S only needs to include neighbors of either X_j, or $X_{j'}$ ⇒ after a few edge removals, a lot fewer S can be constructed
  - early termination: After some removals, it may become impossible to create new S.
  - algorithm:
    - start with complete graph
    - set $S = \emptyset$: for all pairs $(X_j, X_{j'})$ remove edge if $X_j \perp\!\!\!\perp X_{j'} | \emptyset$
    - work on S with $|S| = 1$: for all nodes X_j that have at least 2 neighbors
      - for all $X_{j'} \in NE(X_j)$ and all $X_{j''} \in NE(X_j) \backslash X_{j'}$ remove edge $(X_j, X_{j'})$ if $X_j \perp\!\!\!\perp X_{j'} | X_{j''}$
    - work on S with $|S| = 2$: for all X_j with at least 3 neighbors
      - for all $X_{j'} \in NE(X_j)$ and all $X_{j_1}, X_{j_2} \in NE(X_j) \backslash X_{j'}$ remove edge $(X_j, X_{j'})$ if $X_j \perp\!\!\!\perp X_{j'} | (X_{j_1}, X_{j_2})$
    - and so on, until no X_j has the required number of neighbors
  - in the worst case this is not faster than IC algorithm, but can be proven [Classen et al. 2013] that a variant of PC has worst case complexity $O(D^{2(\deg+2)})$[4]
    ⇒ if the skeleton is sparse (deg is small) ⇒ polynomial runtime; in practice this is usually the case
  - If the oracle is always correct: PC creates the correct skeleton, otherwise the result is *order-dependent* because errors lead to different subsequent tests and errors.
- *optimization 2:* stable parallel PC-algorithm: eliminate order dependence by performing all CI tests for given $|S|$ in parallel, only remove edges until each round $|S|$ is finished
  - don't remove edges in parallel but sort by confidence of the CI test (increasing p-value) and remove in that order, but skip removals whose preconditions no longer hold (i.e. the edge a particular test relies upon have already been removed) = no, this requires more thought
  - faster than PC if CI tests are performed concurrently

[3] PC = "Peter & Clark"

[4] deg: maximum degree of any node in the skeleton

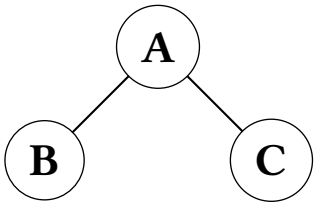


Figure 21.1: Example where applying stable-PC gives a different outcome.

2. detect "v-strutures" (common effects)
   - for all pairs $X_j, X_{j'}$, that are not connected but have a common neighbor $X_{j''}$
     if $X_{j''} \notin S_{jj'} \Rightarrow X_{j''}$ cannot be a common cause of or mediator between X_j and $X_{j'} \Rightarrow j \rightarrow j'' \leftarrow j'$
   - assume that step 2. finds *all* v-structures[5]
3. orient as many edges as possible using following constraints
   - BN must be acyclic ($\Rightarrow$ some orientations are infeasible $\Rightarrow$ use the other direction)
   - when X_j and $X_{j'}$ are not connected but have a common neighbor $X_{j''}$ this *cannot* be a v-structure[6]

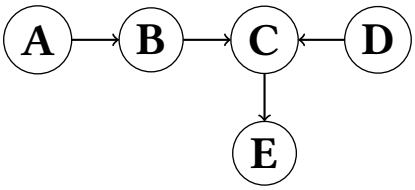


Figure 21.2: True Causal Graph of Example 2

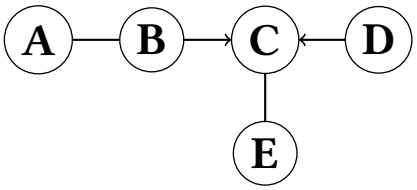


Figure 21.3: True Causal Pattern of Example 2

example[7]
example 2[Spirtes et al 2010][8]

4. get interventional data to orient remaining edges: we know that after $\text{do}(X_j = a_j)$, there can be no incoming arcs to X_j; all edges at X_j must go out in the interventional graph
   - [Eberhardt et al.2006] showed: in the worst case, two situations are needed for every edge $(X_j, X_{j'})$
     a) exp 1: intervene on X_j, but not on $X_{j'}$, exp 2: intervene on $X_{j'}$, but not on X_j

[5] excludes turning $j - j'' - j' \Rightarrow j \rightarrow j'' \leftarrow j'$
[6] if one edge is already connected, this implies the orientation of the other
[7] the lecture contains a couple of graphical examples here
[8] see `Automated Search for Causal Relations_ Theory and Practice.pdf` figure 1 for the graph

b) exp 1: intervene on neither of $X_j, X_{j'}$, exp 2: intervene on exactly one of $X_j, X_{j'}$

Figure 21.4: On the left we see the true unknown complete graph among the variables A,B,C. In one experiment, the researcher performs simultaneously and independently a parametric intervention on A and B (I_A and I_B, respectively, shown on the right). Since the interventions do not break any edges, the graph on the right represents the post-manipulation graph.

5. Theorem:
   - If one intervenes on exactly one variable per experiment, at most $D - 1$ experiments are needed to get full BN.[9]
   - If one can intervene on up to $D/2$ variables simultaneously, $\log_2 D + 1$ experiments are sufficient.[10]

   practical problems are usually not worst case

[9] assuming they do not err
[10] again assuming correctness

22 Lecture 08/07

22.1 Detecting conditional independence by statistical tests

- standard method: **G-test**
  - X can have states 1, ..., C_x
  - Y can have states 1,..., C_y

$$H(x) = -\sum_{k=1}^{C_x} p(X=k)\log(p(X=k))$$

$$H(x,y) = -\sum_{k=1,l=1}^{C_x,C_y} p(X=k,Y=l)\log(p(X=k,Y=l))$$

  - *Mutual Information*

$$MI(X,Y) = H(X) + H(Y) - H(X,Y)$$

  - usual hypothesis $H_0 : X \perp\!\!\!\perp Y$
    Under H_0 the actual observed counts N follow a multinomial distribution

$$\hat{G} = 2N\hat{MI}(X,Y)$$

    has a χ^2 distribution with $(C_x - 1)\cdot(C_y - 1)$ dof[1]
  - conditional independence:
    - repeat for every state of the conditioning set
    - take result with max p-value
    - better: Bonferroni correction[2]

- Problems:
  - Normally this test is conservative: we reject H_0 only when we are confident (high p-value). This means we assume independence in case of doubt. We would like to have a test for $X \not\perp\!\!\!\perp Y$ but this is difficult.
  - continuous variables must be discretized and MI is *very* sensitive to particular discretization $\Rightarrow$ active area of research

[1] dof = degrees of freedom
[2] see `https://xkcd.com/882/`

- if the conditioning set is large, only few samples fulfill the condition ⇒ high variance of MI, in practice $|S| = 4$ is max
- errors propagate in PC also

22.1.0.2 Kernel-based independence test

- map data into augmented feature space $\tilde{X} = \phi(X), \tilde{Y} = \psi(Y)$, where ϕ, ψ are nonlinear
- compute cross covariance matrix

$$CV = \mathbb{E}[(\tilde{X} - \mathbb{E}\tilde{X})^{\mathsf{T}}(\tilde{Y} - \mathbb{E}\tilde{Y})]$$

- compute biggest eigenvalue of CV
- perform statistical test if this ev is 0
  if yes ⇒ $X \perp\!\!\!\perp Y$ because zero correlation implies independence *after* nonlinear mapping [Fukumizu et al 2008]
- eliminate explicit mapping by Kernel trick

22.1.0.3 Approximation algorithm for BN construction

- more making algorithms
  - given our current guess of the BN, define "moves" that transform the BN into a similar one (typical ones: add arc, remove arc, reverse arc,...)
  - compute a score for all candidates produced by moves, eg:

$$BIC = -\log p(D|\theta) + \frac{\#\theta}{2}\log N$$

  - implementations vary in
    - allowed moves
    - score functions
    - amount of randomness
  - initialization: usually empty graph

22.1.0.4 Using structured equation models (SEMs)

- basic claim: Ambiguity in BN construction is caused by definitions that are too general. It may not be the best idea to allow for any possible function $p(X_j|PA(X_j))$ ⇒ restrict function class using SEMs.

$$X_j = f_j(PA(X_j), N_j)$$

  where N_j is noise

- **Theorem**: If $p(X_1,...,X_D)$ is strictly positive and Markov with respect to a DAG G then there exists a SEM on G that generates $p(X_1,...,X_d)$.
- advantage: we have control over the function class, e.g. $X_j = f_j(PA(X_j)) + N_j$
- identifiability:
  if f_j is linear and N_j is Gaussian $\Rightarrow$ cannot distinguish between $X \to Y$, $Y \to X$
  same if f_j is asymptotically constant and strictly monotone and N_j is exp. distributed[3]
- when identifiability holds:
  1. f_j is linear and N_j is nonGaussian $\Rightarrow$ LINGAM algorithm [Kons & Shimzu 2003] (similar to ICA)
  2. f_j is nonlinear N_j is Gaussian $\Rightarrow$ RESIT[4] algorithm

[3] $N_j \sim e^{-|x|}$
[4] [Peters et al. 2014]

23 Lecture 15/07

23.1 RESIT algorithm (regression with subsequent independence test)

- approximation algorithm to construct a BN
- example for hot research: find new ways to detect causality
  - traditional (PC algo): analyze dependence between variables $X \perp\!\!\!\perp Y?$, $X \perp\!\!\!\perp Y|Z?$
  - new idea: Define a SEM[1] and analyze dependency between *predictors* and *residuals* of the SEM regression.
    SEM: $X_j = f(PA(X_j)) + N_j$ (additive noise model)
    regression[2]: $\hat{f} = \arg\min_f \sum_i (X_{ij} - f(PA(X_j)_i))^2$
    residuals: $R_{ij} = X_{ij} - \hat{f}(PA(X_j)_i)$
    by definition of an additive-noise SEM, we require $N_j \perp\!\!\!\perp PA(X_j)$. If the model is correct, the same must be true for the residuals: $R_j \perp\!\!\!\perp PA(X_j)$, meaning, that there is no information in $PA(X_j)$ that could be used to reduce the residual R_j.
  - in particular $R_j \not\perp\!\!\!\perp PA(X_j)$ if the modal is not causal[3]
- algorithm:
  - phase 1 determine the optimal ordering of the variables for Bayesian factorization.
    - $S = \{X_1, \ldots, X_D\}$
    - for $t = D, \ldots, 1$ (construct order backwards)
      - for each $X_j \in S$:
        - regress X_j on $S \backslash X_j$ using a suitable regression method and compute residual R_j
        - conduct independence test for $R_j \perp\!\!\!\perp S \backslash X_j$ and store p-value p_j

[1] structured equation model
[2] (non-linear least squares, kernel regression, ...)
[3] The lecture contains an example graph here: $X_j \to X_{j'}$ via $g(X_j)$ and the anti-causal model $X_{j'} \to X_j$ via $f(X_{j'})$. Our wrong SEM predicts $\hat{x}_j = \hat{f}(g(X_j))$ if g is not invertible, information is lost $\Rightarrow R_j = X_j - \hat{X}_j$ contains information that could be used to predict $X_{j'}$ from $\hat{X}_j$ and R_j.

        - set $\pi(t) = \arg\max_j p_j$ (place the variable with biggest p-value, i.e. highest certainty of independence at position t)
        - $PA(X_{\pi(t)}) = S \backslash X_{\pi(t)}$
        - update $S = S \backslash X_{\pi(t)}$
    - phase 2: determine SEM (remove as many edges from the graph as possible and determine the regression functions)
        - initialize G as the complete DAG for the order of phase 1 (add arcs $X_{j'} \to X_j$ for all $X_{j'} \in PA(X_j)$)
        - for $t = 2, \ldots, D$:
            - for each $X_{j'}, PA(X_{\pi(t)})$ (try to get rid of as many parents as possible)
                - regress $X_{\pi(t)}$ on $PA(X_{\pi(t)}) \backslash X_{j'}$ and compute residuals $R_{j'}$

                  if $R_{j'} \perp\!\!\!\perp PA(X_{\pi(t)})$: remove $X_{j'}$ from parents $PA(X_{\pi(t)}) = PA(X_{\pi(t)}) \backslash X_{j'}$[4]
    - output the resulting graph and regression functions

- properties:
    - only marginal dependence tests are needed (no conditional ones) → easier
    - can prove: The true causal model is identifiable when the regression functions are non-linear, provided that the regression is sufficiently powerful and the additive noise assumption is fulfilled.[5]
    - efficient: $O(D^2 Q)$ operations, where Q is complexity of the regressions and independence tests[6]

23.2 Parameter estimation in BNs

Estimate the probabilities $p(X_j | PA(X_j))$ from training data, given the structure of the BN.

- if the variables are discrete: estimate conditional probabilities by counting
- if the variables are continuous (or too many discrete states): use an SEM and regression
- if data is partially missing
    - if missing at random (the fact that a value is missing is statistically independent of the missing value): EM algorithm (replace missing data by our current guess on their expected value)
    - if systematically missing:

[4] typically: kernel-based independence tests
[5] typical regression methods: kernel SVR, Gaussian processes, generalized additive models, linear regression
[6] compare PC: $O(2^D)$

  - if the missing value is irrelevant for the instance at hand (e.g. physicians wouldn't do a useless diagnostic)
    $\Rightarrow$ introduce class "irrelevant" as an additional state of the variable
  - otherwise: non-trivial problem $\Rightarrow$ later

23.3 Drawing Conclusions from a BN

Two typical cases (in both cases it is critical to avoid omitted variable bias (Simpson's paradox)):

1. has X a direct effect on Y? (i.e. is there an arc $X \to Y$, and if yes, how strong is the association?)
2. what is the total causal effect of X on Y along all paths $X \rightsquigarrow Y$ combined?

- omitted variable bias in **1.**: Berkley admission example
  - proposed BN of the plaintiffs: sex $\to$ admission. G-test for this model is highly significant for sex $\not\perp\!\!\!\perp$ admission and discriminating women p-value $< 10^{-}5$
  - proposed BNs of the defense: sex $\to$ field $\to$ admission and maybe even a link sex $\to$ admission?
    Q: is there a direct effect of sex on admission?
    G-test: sex $\perp\!\!\!\perp$ admission | field:
    - conditional independence for 5 out of 6 fields p-value > 0.3
    - in one field: conditional dependence with p-value $= 10^{-5}$, but here women are significantly preferred (82% vs. 62%)

    $\Rightarrow$ sex has a large total causal effect, but only a small *direct* effect.
    $\Rightarrow$ rule: omitted "mediating" factors[7] (here field) cause bias when the direct effect is of interest.
- in case **2.**, omitted variable bias arises from missing common causes ("confounders") example: kidney stone data: recovery rates for two treatments A (open surgery), B (minimal invasive surgery)

A:treated	A:recovered	B:treated	B:recovered
350	273 (78%)	350	289 (83%)

  B seems to be better (but G-test says that the difference is not yet significant)
  BN: treatment $\to$ recovery. But an important confounder is missing: stone size, giving us: recovery $\leftarrow$ size$\rightarrow$ treatment $\to$ recovery

	A:treated	A:recovered	B:treated	B:recovered
	350	273 (78%)	350	289 (83%)
small	87	81(93%)	270	234(87%)
big	263	192 (73%)	80	55 (69%)

[7] variables on directed path $X \rightsquigarrow Y$

conditional on stone size, treatment A is superior. But: physicians prefer treatment B for the less severe cases[8]
problem arises due to the difference between the conditional probability $p(rec|X)$ and interventional probability $p(rec|\text{do}(treatm.))$. We derived earlier:

$$p(Y|X) = \sum_Z p(Z|X)p(Y|X,Z)$$
$$p(Y|\text{do}(X)) = \sum_Z p(Z)p(Y|X,Z)$$

the latter being the definition of the total causal effect of X on Y.

$$p(rec|\text{do}(treat = A)) = 83\%$$
$$p(rec|\text{do}(treat = B)) = 78\%$$

i.e. exactly the reverse of the conditional probability.
Computing $p(Y|\text{do}(X))$ in the presence of confounders is called *adjustment*.

[8] i.e. the BN is still not complete, because it doesn't explain the treatment choice ⇒ more hidden factors, e.g. risk factors, speed of recovery, cost,...

24 Lecture 17/07

24.1 Confounder Adjustment

- naive estimate of treatment effect $\mathbb{E}[Y|X = A] - \mathbb{E}[Y|X = B] \approx \frac{1}{N_A} \sum_{i:X_i=A} Y_i - \frac{1}{N_B} \sum_{i:X_i=B} Y_i$ is biased, when treatment decision X_i depends on features Z_i of the individual instance i, because the groups who received either treatment are not comparable
- illustration using *potential outcomes*: Imagine that for each individual, the reaction to treatment A and B is pre-determined but unknown to us.
  - By applying a treatment, we will observe one of the potential outcomes, but since we cannot rewind time, the other outcomes ("counter-factual" outcomes) are missing = *"fundamental missing data problem of causal inference"*.
  - If we have binary variables (2 treatments A and B, 2 outcomes true and false), there are 4 different types of people according to potential outcomes:

type|pot outcome	A	B	
a	1	0	A responsive
b	0	1	B responsive
c	1	1	complete responsive
d	0	0	doomed[1]

    two observation groups: N_A individuals who received treatment A: T_A[2]

  - the (unknown) proportion of the 4 types in both groups: p_a, p_b, p_c, p_d for T_A; q_a, q_b, q_c, q_d for T_B
  - we cannot distinguish types: a from c, b from d in T_A and a from d, b from c in T_B because the counter-factual outcome is unknown.
  - compute the number of recovered people: $R_A = N_A(p_a + p_c)$; $R_B = N_B(q_c + q_d)$
  - naive estimate: compare the proportions: $\frac{R_A}{N_A} - \frac{R_B}{N_B} = p_a + p_c - (q_b + q_c)$
  - we are actually interested in the treatment effect: $p_a - p_b$
  - we must make sure that the two groups T_A and T_B are comparable ("exchangeable" treatment assignment)
    $\Rightarrow p_a \approx q_a, p_b \approx q_b \Rightarrow \frac{R_A}{N_A} - \frac{R_B}{N_B} \approx p_a - p_b$ as desired.

[1] insert dark laughter here
[2] N_B, T_B analogously

  - preferred strategy to achieve these *randomized experiments* = decide about the treatment uniformly at random, without considering the features Z_i.
    $\Rightarrow$ asymptotically, the feature distributions $p(Z|X = A) = p(Z|X = B) = p(Z)$ because $X \perp\!\!\!\perp Z$ by design
    $\Rightarrow$ we have $p_a = q_a$, etc. asymptotically
    (for finite samples, this may not be achieved $\Rightarrow$ avoid this by rejection sampling, i.e. check if $p(Z|X = A) = p(Z|X = B)$ and draw a new *random* assignment if this is not the case)
  - often, randomized assignment is impossible
    example: placebo surgery $\Rightarrow$ We must explicitly adjust for the differences in T_A and T_B.

- **possibility 1. confounder adjustment:** $p(Y|\text{do}(X = A)) = \sum_Z p(Y|X = A, Z)P(Z)$
  - The confounder Z can actually be a set of features: $Z = \{Z_1, ..., Z_L\}$.
  - question: What is a valid adjustment set Z, given the BN?
  - graphical criterion: "backdoor criterion" (sufficient, but necessary):
    a) Z must not contain a descendant of X to avoid Berkson's paradox[3] and mediating variables
    b) remove the outgoing arcs of X (= remove the causal effects of X) $\Rightarrow$ All remaining associations between X and Y are spurious common cause effects of the confounders.
    If $X \perp\!\!\!\perp Y|Z$ in the modified path, such effects cannot occur if we adjust for $Z \hat{=}$ valid adjustment set.[4]
  - problem: The number of joint states in Z grows exponentially with the number of variables in Z $\#states = \Omega(2^L)$.
    $\Rightarrow$ $\#states$ is big we will be unable to estimate $P(Y|X, Z)$ from a realistic amount of training data

- **possibility 2. stratification:** define subgroups of instances with similar Z ("clusters", "strata") and estimate probabilities in each stratum separately and sum over strata
  - typically achieved by coarse quantization of the Z_l into at most 5 levels.[5]
  - still doesn't work when Z has too many variables

- **possibility 3. propensity score:**[Rosenbaum& Rubin 1983] [Austin 2011][6]
  - introduce a new variable F by splitting X (F becomes the only parent of X receiving all the arrows from Z)
    $\Rightarrow$ If Z was a valid adjustment set, so is F because it blocks exactly the same backdoor paths.

[3] `https://en.wikipedia.org/wiki/Berkson's_paradox`
[4] special case: $Z = PA(X)$
[5] age groups: 5-15, 15-25,...
[6] see `austin2011.pdf`

- define structured equation model: F is a deterministic function[7] of Z $F_A(Z) = p(X = A|Z)$, called the *propensity score for A*
  simply train a classifier that gives a posterior distribution (logistic regression, random forest, neural networks)
  $p(X = A) = Bernoulli(F_A(X))$
- surprising result: when $F_A(Z_i) \approx F_A(Z_{i'})$ then the two individuals are comparable, even if $Z_i \neq Z_{i'} \Rightarrow$ it only matters that propensity scores match
- use this in three ways:

  i) stratify on propensity score intervals

  ii) weight adjustment: consider

$$\begin{aligned}
\mathbb{E}\left[\frac{\mathbb{1}(X = A)Y}{F_A(Z)}\right] &= \sum_{X,Y,Z} \frac{\mathbb{1}(X = A)Y}{F_A(Z)} p(X, Y, Z) \\
&= \sum_{X,Y,Z} \frac{\mathbb{1}(X = A)Y}{F_A(Z)} p(Y|X, Z)p(X|Z)p(Z) \\
&= \sum_{Y,Z} \frac{Y}{F_A(Z)} p(Y|X = A, Z) \underbrace{p(X = A|Z)}_{=F_A(Z)} p(Z) \\
&= \sum_{Y} Y \sum_{Z} \underbrace{p(Y|X = A, Z)p(Z)}_{p(Y|\text{do}(X=A))} \\
&= \mathbb{E}[Y|\text{do}(X = A)]
\end{aligned}$$

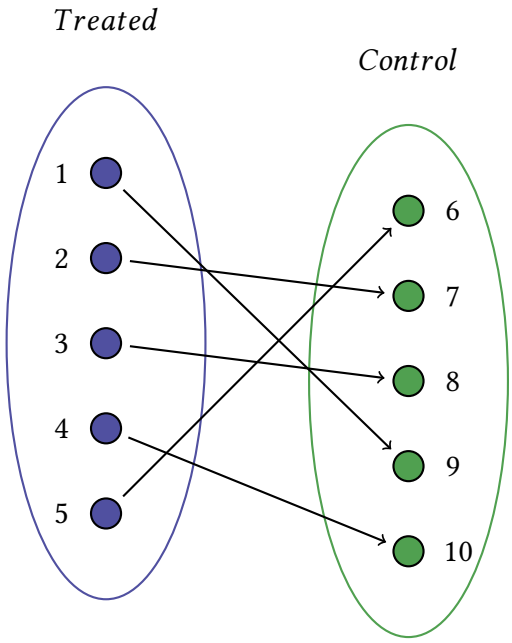
  likewise, we have $\mathbb{E}\left[\frac{\mathbb{1}(X=B)Y}{F_B(Z)}\right] = \mathbb{E}\left[Y|\text{do}(X = B)\right]$

$$\begin{aligned}
&\Rightarrow \mathbb{E}[Y|\text{do}(X = A)] - \mathbb{E}[Y|\text{do}(X = B)] \\
&\approx \frac{1}{N} \sum_i \frac{\mathbb{1}(X_i = A)Y_i}{F_A(Z_i)} - \frac{1}{N} \sum_i \frac{\mathbb{1}(X_i = B)Y_i}{F_A(Z_i)} \\
&= \frac{1}{N} \sum_{i:X_i=A} \frac{Y_i}{F_A(Z_i)} - \frac{1}{N} \sum_{i:X_i=B} \frac{Y_i}{F_B(Z_i)} \\
&\neq \frac{1}{N_A} \sum_{i:X_i=A} Y_i - \frac{1}{N_B} \sum_{i:X_i=B} Y_i
\end{aligned}$$

  iii) propensity score matching: arrange the two groups into a complete bipartite graph
  - weight edges by absolute difference $|F_A(Z_i) - F_A(Z_{i'})| = w_{ii'}$
  - remove edges whose weight is above some threshold
  - find the minimum cost by bipartite matching by Hungarian algorithm (or a greedy approximation[8])

[7] analogously for B

[8] results do not really change according to literature



* form subgroups T'_A and T'_B from all matched pairs: in T'_A and T'_B the naive formula is equivalent to the exact formula

$$\mathbb{E}[Y|\text{do}(X = A)] = \mathbb{E}_{\text{pairs}}[Y|X = A]$$

* the matched partner is our best guess at the counterfactual outcome
* ultimate pairing: twin-study

25 Lecture 22/07

25.1 Hidden Confounders

- variables we cannot measure (accurately) or don't even know that they exist/are relevant
- gold standard: randomized controlled experiment (RCE): $\text{do}(X)$ cuts all incoming arcs of X, regardless of known or hidden
- if there is no opportunity to do RCE, not all is lost:
  - sometimes, $p(Y|\text{do}(X))$ is still identifiable, e.g. front-door adjustment formula
  - It may be possible to intervene on an instrumental variable W that influences X. $\Rightarrow p(Y|\text{do}(X))$ may be identifiable
  - But: the success of these methods depends on very narrow conditions.

25.2 Transfer Learning = Domain Adaptation

- suppose we cannot get data of the desired quality in the target
- instead of asking: "How can we get away with bad data?", we ask "Can we combine the bad data with good data from a similar domain to get better results?"
- we want to be better than the naive base lines:
  - learn model from target data alone (high variance, possibly high bias if unadjusted confounding)
  - use model learned in the source domain (high bias, because domains differ)
- typical scenarios: we got high quality annotations from experts, but experts won't do this again for another dataset
- There was a carefully designed experiment in one country: are the results transferable to another country, where only observational data exists?
- Can we transfer results on a limited cohort (students) to the population at large?[1]
- Frustratingly Easy Domain Adaptation: EasyAdapt[Daumié III 2007][2], EasyAdapt++ [Daumé III et al 2010]

[1] e.g. WEIRD students in psychological studies, or transfer from lab animals to humans

[2] see `daumedomainAdapt.pdf`

- standard 2-class classification
- we have lots of annotated training data in domain $D = 0$
- we have few annotated training data in domain $D = 1$, optionally lots of unlabeled data (semi-supervised)
- idea:
  - centralize the features X
  - create an augmented feature space $\tilde{\underline{X}}$ by replicating the features:

$$\tilde{\underline{X}} = [\underline{X}, (1-D)\underline{X}, D\underline{X}] = \begin{cases} [\underline{X}, \underline{X}, \underline{O}], & \text{if } i \in \text{source} \\ [\underline{X}, \underline{O}, \underline{X}], & \text{if } i \in \text{target} \end{cases}$$

    $\Rightarrow$ treat transfer learning as a missing data problem, one of the copies is missing for each instance and replaced by the expected values $\underline{O}$.
  - the first copy should capture the common properties of both domains, the others the differences
- consider a linear classifier: training gives a weight vector: $\tilde{\beta} = [\beta_c, \beta_s, \beta_t]^\mathsf{T}$
- prediction of a source instance $\hat{Y} = \tilde{\underline{X}}\beta = \underline{X}(\beta_c + \beta_s)$
- prediction of a target instance: $\hat{Y} = \tilde{\underline{X}}\beta = \underline{X}(\beta_c + \beta_t)$
- also works for any blackbox classifier = EasyAdapt, works well in experiments
- semi-supervised version EasyAdapt++: we augment the training set also with unlabeled data
- claim: predictions of source and target classifiers should be similar on unlabeled data

$$\begin{aligned} \underline{X}_u(\beta_c + \beta_s) \approx \underline{X}_u(\beta_c + \beta_t) &\Leftrightarrow \underline{X}_u(\beta_s - \beta_t) \approx 0 \\ &\Leftrightarrow \tilde{X}_u\tilde{\beta} \approx 0 \qquad \text{where } \tilde{X}_u = [\underline{O}, \underline{X}, -\underline{X}] \end{aligned}$$

- since we don't have label 0, we add two augmented instances for each unlabeled instance, one with either label $\tilde{X}_u, \tilde{Y}_u = [\underline{O}, \underline{X}, -\underline{X}, +1], \tilde{X}_{u'm}\tilde{Y}_{u'} = [\underline{O}, \underline{X}, -\underline{X}, -1]$
- train normally and predict $\hat{Y} = \underline{X}(\beta_c + \beta_t)$ for target points
- for linear SVM, we get the rolling loss functions
- outperforms many complex methods

25.3 Data augmentation

idea: to make a ML algorithm robust against certain systematic transformations of the data X, create additional training data using these transformations without changing the outcome
$\Rightarrow$ algorithm must learn that the transformations are irrelevant

- robustness against noise: add noise,
- illumination changes: linear intensity transformations
- rotational invariance: rotate the data
- shape invariance: randomly morph the shape
- ...
- especially popular for neural networks since they need a lot of training data anyway, and can create augmented data on the fly
- preliminary: performs as well as explicitly designed invariance, but much simpler

25.4 Importance sampling by reweighting

- given a BN with fixed structure and parameterized probabilities/structural equations:
- we have a high-quality data of the BN behavior for some parameterization θ, want to know: "How does the BN behave for θ'", without redoing the (expensive) experiment.
- replace question by counterfactual question: "How would the BN have behaved had the parameters been θ' provided the hidden mechanisms didn't change."
  $\Rightarrow$ virtually replay the data to simulate BN(θ')
- example: advertisement placement on the Bing Search result [Bottou et al. 2013]
  - three conflicting goals:
    * don't annoy the user (few and relevant ads)
    * attract advertisers (high click rates at reasonable price)
    * maximize Bing's revenue
  - BN:

$$p(\underline{X}|\theta) = \prod_j p(X_j|PA(X_j),\theta)$$

  - if we change the parameters for a single white arrow, we get

$$p(\underline{X}|\theta') = P(\underline{X},\theta) \underbrace{\frac{P(X_j|PA(X_j),\theta')}{P(X_j|PA(X_j),\theta)}}_{w_j = \text{ reweighting factor for } j}$$

$$\mathbb{E}[Y|\theta'] = \mathbb{E}[Yw_j|\theta] \approx \frac{1}{N}\sum_i Y_i w_{ij}$$

This is known as 'importance sampling'

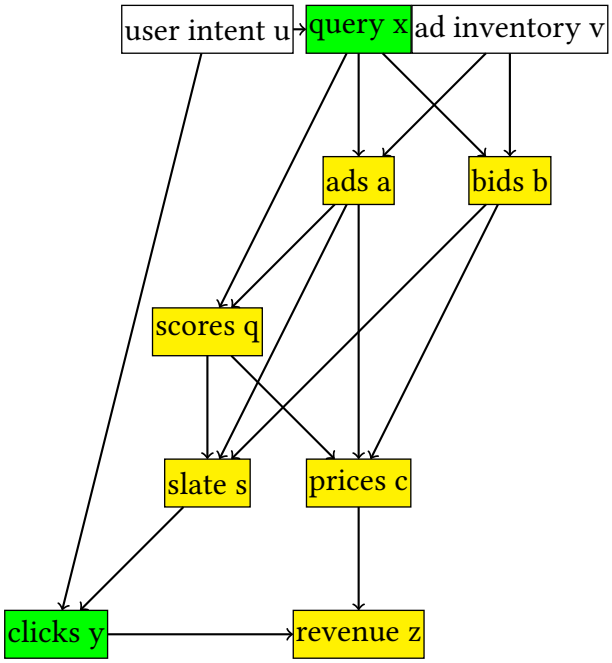


Figure 25.1: Causal graph of the network.

- critical: w_j must not diverge because $p(X_j|PA(X_j),\theta) \approx 0$
  $\Rightarrow$ researchers artificially increased the variance in the θ-experiment, so that $p(X_j|PA(X_j),\theta)$ spans a larger part of the feature space
- performs well in experiments

25.5 Causal Theory of transferability [Barenboim, Pearl, Tian, 2012-2015]

- address[3] if the causal effect $p^*(Y|\text{do}(x))$ in domain $*$ is identifiable by combining observational data from $*$ (lower quality) with high-quality (experimental) data from a related domain.
- graphical notation extensions: dotted bidirectional arrows for hidden confounding, Selection variable $S \in \{0,1\}$ for the domain
- examples: *the lecture contains some examples here, but since they aren't reproduced here, the corresponding calculations are left out as well*
- intuitive goal: transform the expressions such that no conditional probability contains $S = 1$, and $\text{do}(X)$ simultaneously
- inventors presented a complete theory (graphical criteria and algorithms) to

[3] see e.g. `AAAI-14-r425.pdf`

- decide if $p(Y|\text{do}(X), S = 1)$ is identifiable in a given graph (including dotted arrows)
- find the appropriate adjustment expression by automatic symbolic calculations

26 Lecture 24/07

26.1 The omitted chapters (aka. “Machine Learning III”)

26.1.0.5 Markov Random Fields

- undirected graphical models: decompose according to Gibb's distribution $p(X_1,...,X_D) = \frac{1}{Z}\exp\left(-\sum_c E_c(\underline{X}_c)\right)$, where E_C are the energies/potentials and c the cliques of variables in a given undirected graph (= maximal fully connected subgraphs), Z the partition function: $Z = \sum_{\underline{X}} \exp(-\sum_c E_c(\underline{X}_c))$
  The partition function is usually intractable
  $\Rightarrow$ most popular task: finding the MAP-solution [1] = minimal energy state = “best” solution

$$\hat{\underline{X}} = \arg\max_{\underline{X}} p(\underline{X}) = \arg\min_{\underline{X}} \sum_c E_c(\underline{X}_c) + \underbrace{\log Z}_{=\text{const}}$$

- typical merges
  - unary potentials $E_c(\underline{X}_c) = E(X_j)$ encode the local evidence for probable state of X_j
  - pairwise potentials $E(X_j,X_{j'})$ encode the desire of $X_j,X_{j'}$ to take similar values (= attractive potential) or different values (= repulsive potentials)
  - higher-order potentials $|C| \geq 3$: encode preferences of the structure/pattern of the variables involved (often neglected by assumption)
- typical inference algorithms for discrete X
  - exact:
    - reformulate the problem as an integer linear program $\arg\min_{\underline{X}} \underline{w}\cdot\underline{X}$ s.t. linear inequality constraints are met
      (usually NP-hard, but often tractable by heuristics in practice)
    - special cases:
      - tree-shaped models $\Rightarrow$ belief propagation gives the exact solution in one forward/backward sweep
      - sub-modular models $\sum_{X_j,X_{j'}} \mathbb{1}[X_j = X_{j'}]E(X_j,X_{j'}) \leq \sum_{X_j,X_{j'}} \mathbb{1}(X_j \neq X_{j'})E(X_j,X_{j'}) \Rightarrow$ graph-cut algorithm is exact [maximum flow in a graph = standard problem]

[1] maximum a posteriori

- approximations:
  - relaxation, i.e. allow real values for $\underline{X}$ in the linear program (rounded later)
  - move making: given a guess $\underline{X}^{(t)}$, define elementary moves (changes of few variables) and accept the best one (iterated conditional models ICM, Lazy Flipper[2], tree submodels)
  - move making: reduce the problem to a tractable subproblem (one-label-against-the-rest = α expansion, one-label-against-one = α-β swap[3])
  - loopy belief propagation: iterate message passing until convergence
  - sampling methods: randomly simulate the model and choose the best solution we have seen (Markov Chain Monte Carlo (MCMC), Svendsen-Wang, Gibbs sampling)
- learning the potentials:
  - learn them in isolation, independently of the others
  - better but much more difficult: learn potentials jointly, s.t. they reinforce each other towards a global loss function (on large patterns)
  - details: watch Fred Hamprecht's new video lecture[4]

26.1.0.6 Weak Annotations

getting annotated training data is expensive

- one-class learning: only provide annotations for the target class (assuming this is easy) ⇒ non-target = outliers in target distribution
  ⇒ generative model, one-class SVM (one contour of the PDF, e.g. 95% confidence contour, or several coupled one-class SVMs for nested contours)
- multiple instance learning: a single label for a group ("bag") of instances (e.g. one label per image for all pixels jointly) with the understanding that only some instances in each bag conform to the label ⇒ find these instances and the corresponding classifier
- similarity (metric) learning: just annotate "is A more similar to B or to C" ⇒ learn the similarity function and the clustering
- sparse annotation:[5] for each instance, only a few true labels are known (e.g. movies that someone likes) ⇒ infer the missing labels and learn a model (e.g. recommender systems)
- active learning: minimize the required training set size by actively selecting the most informative (don't waste annotator effort on the easy decisions)

[2] see lazyflipper.pdf
[3] both can be solved by graph-cut
[4] `https://www.youtube.com/playlist?list=PLuRaSnb3n4kSgSV35vTPDRBH81YgnF3Dd`
[5] related to multiple instance learning

- semi-supervised learning: combine a small labeled training set with a big unlabeled (combine supervised & unsupervised learning)
- transfer learning: combine a small labeled training set with a big labeled training set from a similar domain
- reinforcement learning: for state machines: instead of learning transition probabilities that maximize the data likelihood (Baum-Welch), learn transition probabilities that optimize an "expected reward", problem: reward can only be computed after many transitions not for each transition individually (delayed annotation): e.g. games: win or loss, robots: goal achieved or crashed

26.1.0.7 Matrix factorization

many phenomena can be explained by a linear superposition of elementary phenomena
$\Rightarrow$ (observed matrix) = $\underbrace{(\text{elementary weights})}_{\text{what could happen}} * \underbrace{\text{weights}}_{\text{what was selected}}$
general idea: use application-specific constraints to make decomposition well-posed, most popular: sparsity, each elementary entry only explains a local part of the domain, only few elementary things are active in each instance

26.1.0.8 Features

- designed features: we have to select from the infinite possibilities
- feature learning:
  - initial layers of a neural network
  - kernel approximation: $k(x, x') = \langle \phi(x), \phi(x') \rangle \Rightarrow$ use feature selection to find the important coordinates in $\phi(x)$ and compute $\tilde{\phi}(x)$ explicitly (without kernel)
- random features:
  - random projections have a lot of interesting structural properties (they are *not* chaos)
    $\Rightarrow$ use these properties (e.g. Johnson-Lindenstrauss lemma[6])
    $\Rightarrow$ multi-dimensional (randomized) hashing for similarity
    $\Rightarrow$ extreme learning machine 2 layer NN: visible$\rightarrow$ hidden random, hidden $\rightarrow$ output analytically optimized

[6] https://en.wikipedia.org/wiki/Johnson-Lindenstrauss_lemma