INF8225 – Lesson 2

Slides from:

Data Mining

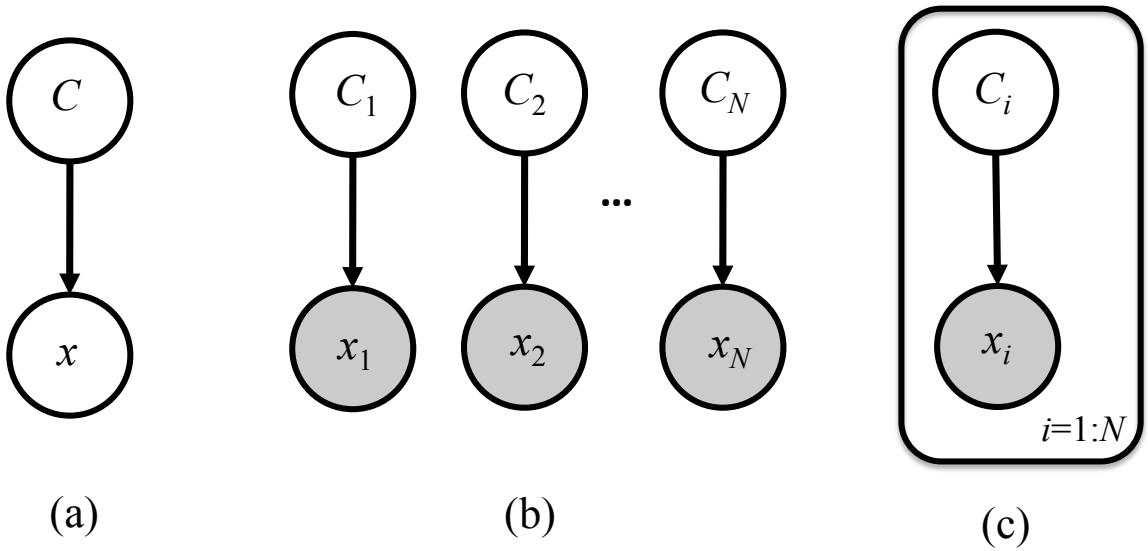
Practical Machine Learning Tools and Techniques

Slides from Chapter 9 of *Data Mining*
by I. H. Witten, E. Frank, M. A. Hall and C.J. Pal

Probability Models, Plate Notation and the Gaussian Mixture Model

Plate notation

- A “plate” is simply a box around a Bayesian network that denotes a certain number of replications of it, one for each data instance
- The plate below indicates i=1…N networks, each with an observed value for x_i and hidden variable C_i
- Plate notation captures a model for the joint probability of the entire data with a simple picture.



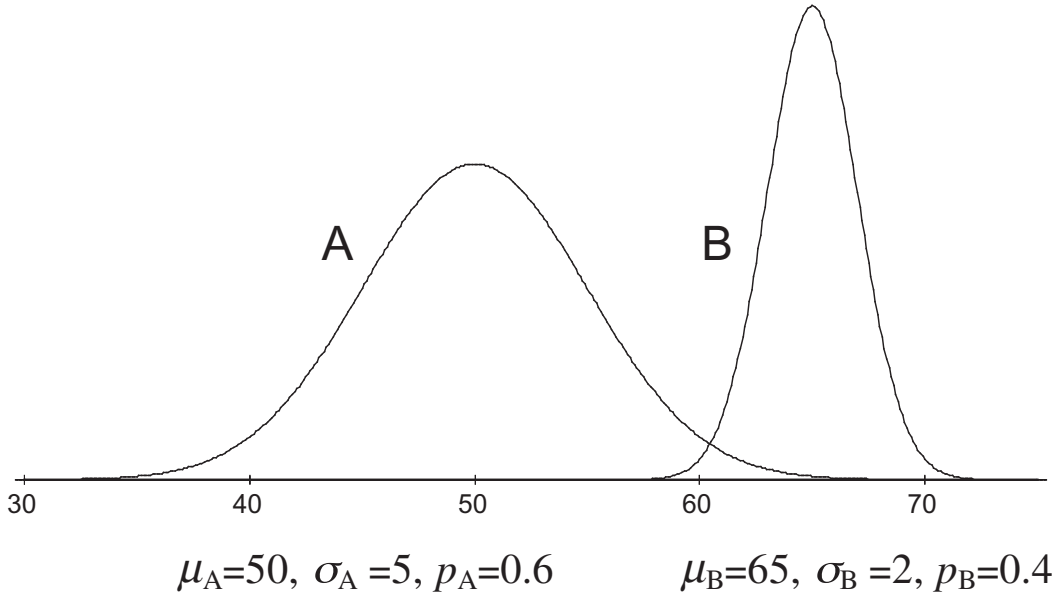
Clustering with a Gaussian Mixture

- Given the data on the left *without the labels A and B*, we wish to estimate a model for a two class Gaussian Mixture Model (GMM) on the right

Data

A 51	A 46	B 65
A 43	B 64	A 46
B 62	A 51	A 39
B 64	A 52	B 62
A 45	B 62	B 64
A 42	A 49	A 52
A 46	A 48	B 63
A 45	B 62	B 64
A 45	A 43	A 48
B 62	A 40	B 64
A 47	A 48	A 48
A 52	B 64	A 51
B 64	A 51	A 48
A 51	B 63	B 64
B 65	A 43	A 42
A 48	B 65	A 48
A 49	B 66	A 41

Model



Clustering and probability density estimation

- If we had data belonging to two known classes A and B, each having a normal distribution with means and standard deviations μ_A and σ_A for class A, and μ_B and σ_B for class B
- We could define a model whereby samples are taken from these distributions, using cluster A with probability p_A and cluster B with probability p_B (where $p_A + p_B = 1$),
- Sampling we might obtain the data in the next slide
- Now, imagine given the dataset without the classes—just the numbers—and being asked to determine the five parameters that characterize the model: μ_A, σ_A, μ_B, σ_B, and p_A (the parameter p_B can be calculated directly from p_A)

Estimating Gaussian parameters

- If we knew which of the two distributions each instance came from, finding the five parameters would be easy—just estimate the mean and standard deviation for $n=n_A$ or $n=n_B$ samples x_1, x_2, …, x_n for each cluster, A and B

$$\mu = \frac{x_1 + x_2 + \ldots + x_n}{n}$$

$$\sigma^2 = \frac{(x_1 - \mu)^2 + (x_2 - \mu)^2 + \ldots + (x_n - \mu)^2}{n-1}$$

- To estimate the fifth parameter p_A, just take the proportion of the instances that are in the A cluster, then $p_B=1-p_A$.

Motivating the EM algorithm

- If you knew the five parameters, finding the (posterior) probabilities that a given instance comes from each distribution would be easy
- Given an instance x_i, the probability that it belongs to cluster A is

$$P(\mathrm{A}|x_i) = \frac{P(x_i \mid \mathrm{A}) \cdot P(\mathrm{A})}{P(x_i)} = \frac{N(x_i;\mu_\mathrm{A},\sigma_\mathrm{A})p_\mathrm{A}}{N(x_i;\mu_\mathrm{A},\sigma_\mathrm{A})p_\mathrm{A} + N(x_i;\mu_\mathrm{B},\sigma_\mathrm{B})p_\mathrm{B}}$$

where $N()$ is the normal or Gaussian distribution

$$N(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The EM algorithm for a GMM

- Start with a random (but reasonable) assignment to the parameters
- Compute the posterior distribution for the cluster assignments for each example
- Update the parameters based on the expected class assignments - where probabilities act like weights.
- If w_i is the probability that instance *i* belongs to cluster A, the mean and std. dev. are

$$\mu_A = \frac{w_1 x_1 + w_2 x_2 + \ldots + w_n x_n}{w_1 + w_2 + \ldots + w_n}$$

$$\sigma_A^2 = \frac{w_1 (x_1 - \mu)^2 + w_2 (x_2 - \mu)^2 + \ldots + w_n (x_n - \mu)^2}{w_1 + w_2 + \ldots + w_n}$$

The EM algorithm

- Optimizes the marginal likelihood, obtained by marginalizing over the two components of the Gaussian mixture
- The marginal likelihood is a measure of the "goodness" of the clustering and it increases at each iteration of the EM algorithm.
- In practice we use the log marginal likelihood

$$\log \prod_{i=1}^{n} P(x_i) = \sum_{i=1}^{n} \log \sum_{c_i} P(x_i \mid c_i) \cdot P(c_i)$$

$$= \sum_{i=1}^{n} \log \left[N(x_i; \mu_{\mathrm{A}}, \sigma_{\mathrm{A}}) p_{\mathrm{A}} + N(x_i; \mu_{\mathrm{B}}, \sigma_{\mathrm{B}}) p_{\mathrm{B}} \right]$$

Extending the mixture Model

- The Gaussian distribution generalizes to n-dimensions
- Consider a two-dimensional model consisting of independent Gaussian distributions for each dimension
- We can transform from scalar to matrix notation for a two dimensional Gaussian distribution as follows:

$$P(x_1, x_2) = \frac{1}{\sqrt{2\pi}\sigma_1}\exp\left[-\frac{(x_1-\mu_1)^2}{2\sigma_1^2}\right]\frac{1}{\sqrt{2\pi}\sigma_2}\exp\left[-\frac{(x_2-\mu_2)^2}{2\sigma_2^2}\right]$$

$$= (2\pi)^{-1}\left(\sigma_1^2\sigma_2^2\right)^{-1/2}\exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\begin{bmatrix}\sigma_1^2 & 0\\ 0 & \sigma_2^2\end{bmatrix}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}$$

$$= (2\pi)^{-1}|\boldsymbol{\Sigma}|^{-1/2}\exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\},$$

- $\boldsymbol{\Sigma}$ is the covariance *matrix,* $|\boldsymbol{\Sigma}|$ is its determinant, the vector $\mathbf{x} = [x_1\ x_2]^T$, and the mean *vector* $\boldsymbol{\mu} = [\mu_1\ \mu_2]^T$

The multivariate Gaussian distribution

- Can be written in the following general form

$$P(x_1, x_2, \ldots, x_d) = (2\pi)^{-d/2} |\boldsymbol{\Sigma}|^{-1/2} \exp\left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\}.$$

- The equation for estimating the covariance matrix is

$$\boldsymbol{\Sigma} = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T.$$

- The mean is simply

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i.$$

Factor Graphs and Probability Models

Factor graphs

- Represent functions by factoring them into the product of local functions, each of which acts on a subset of the full argument set

$$F(x_1,\ldots,x_n) = \prod_{j=1}^{S} f_j(X_j)$$

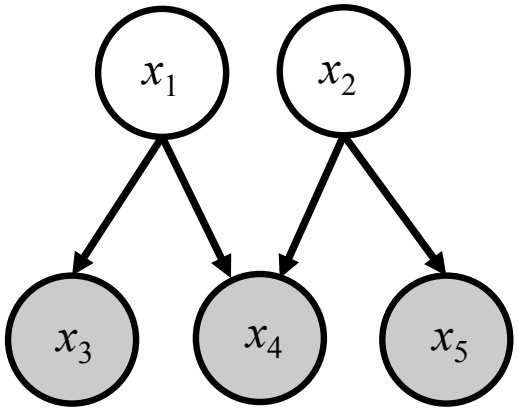
where X_j is a subset of the original set of arguments $\{x_1,\ldots,x_n\}$, $f_j(X_j)$ is a function of X_j, and j=1…S enumerates the argument subsets.

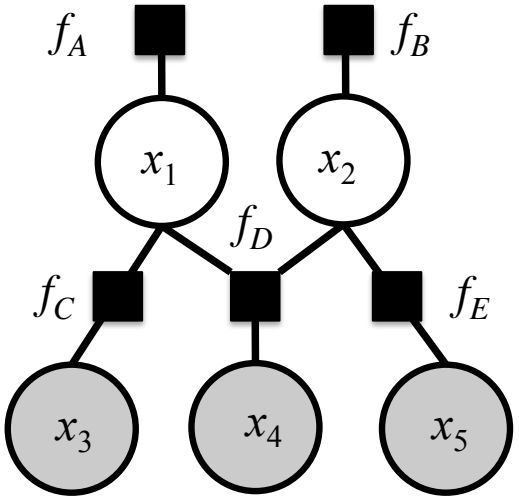
- A *factor graph* consists of variable nodes – circles – for each variable x_k and factor nodes – rectangles – for each function, with edges that connect each factor node to its variables.

Factor graph example

- A Bayesian network and its factor graph for

$$F(x_1,\ldots,x_5) = P(x_1)P(x_2)P(x_3 \mid x_1)P(x_4 \mid x_1,x_2)P(x_5 \mid x_2)$$
$$= f_A(x_1)f_B(x_2)f_C(x_3,x_1)f_D(x_4,x_1,x_2)f_E(x_5,x_2)$$

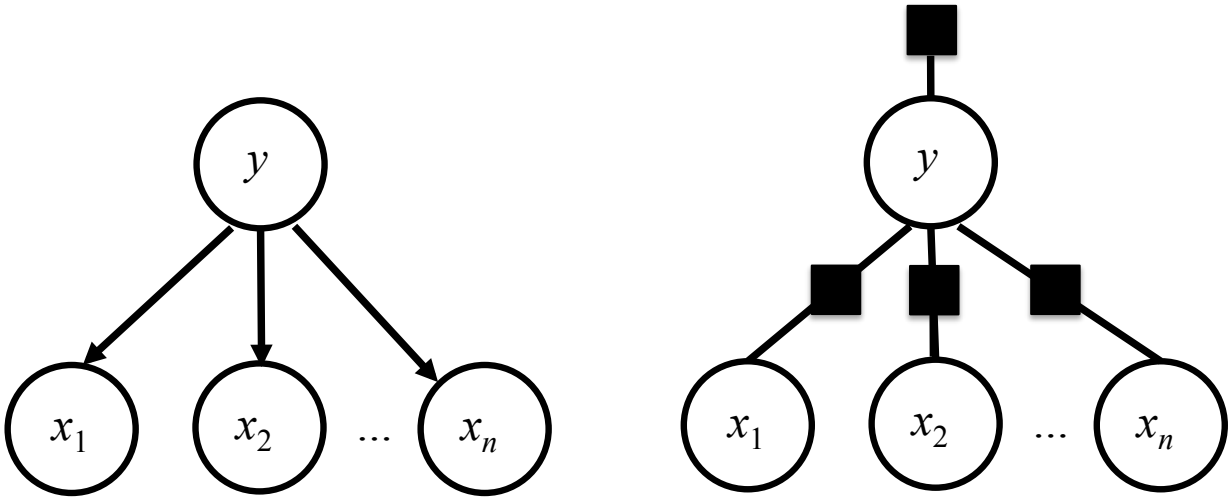




Factor graphs for naïve Bayes models

- Naïve Bayes models have simple factor graphs

$$P(y, x_1, \ldots, x_n) = P(y) \prod_{i-1}^{n} P(x_i \mid y).$$

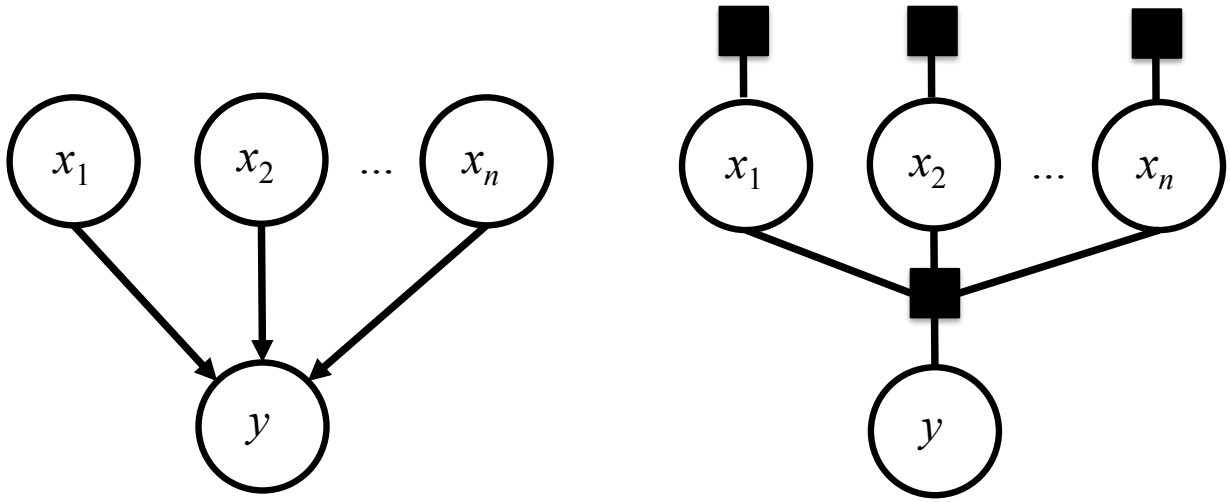


- Consider the number of parameters needed for each factor

An important observation

- Reversing the arrows in a naïve Bayes model leads to a variable that has many parents

$$P(y, x_1, \ldots, x_n) = P(y \mid x_1, \ldots, x_n) \prod_{i-1}^{n} P(x_i)$$



- Consider the # of parameters for p(y|x_1,…,x_n)

Logistic regression and factor graphs

- Rather than using a large table for the conditional distribution of a child y given many x_is, a logistic regression model can be used to reduce the number of parameters from exponential to linear
- Let's assume that all variables are binary
- Given a separate function $f_i(y,x_i)$ for each binary variable x_i, the conditional distribution defined by a logistic regression model can be expressed as shown in the next slide

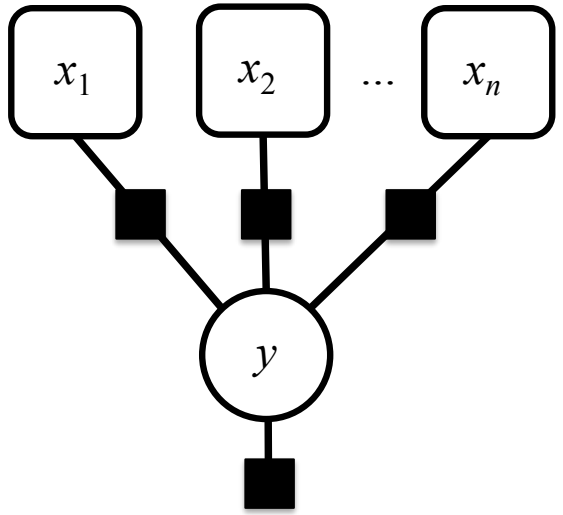
Logistic regression and factor graphs

- Logistic regression can be written as

$$P(y|x_1,\ldots,x_n)$$

$$= \frac{1}{Z(x_1,\ldots,x_n)} \exp\left(\sum_{i=1}^{n} w_i f_i(y,x_i)\right)$$

$$= \frac{1}{Z(x_1,\ldots,x_n)} \prod_{i=1}^{n} \phi_i(x_i,y).$$



where we note that the denominator *Z* is a *data dependent* a normalization term

- This factor graph resembles the Naïve Bayes model, but it is for a factorized *conditional* distribution
- Note: we have used rectangles for the x_i*s* because they are not being explicitly modeled as random variables

Markov random fields (MRFs)

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

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

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

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

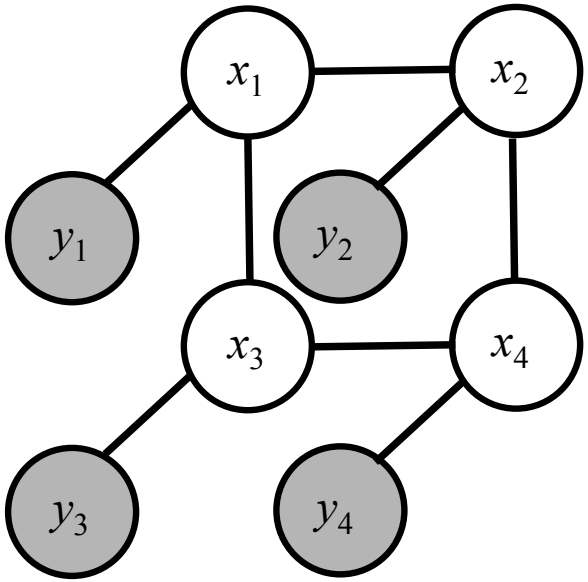
Markov random field example

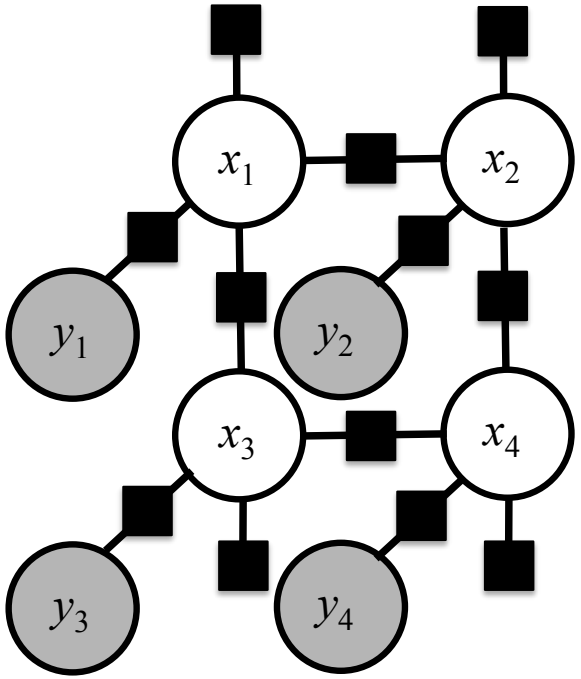
$$P(x_1, x_2, x_3, x_4) = \frac{1}{Z} \prod_{u=1}^{U} \phi_u(X_u) \prod_{v=1}^{V} \Psi_v(X_v)$$

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

- Note we used unary and pairwise potentials

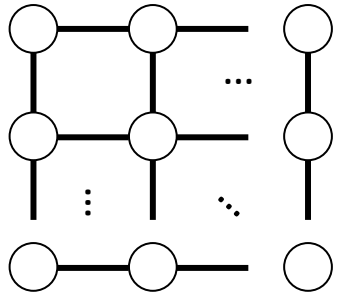
An MRF using a traditional MRF style undirected graph





An MRF as a factor graph

MRFs and energy functions



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

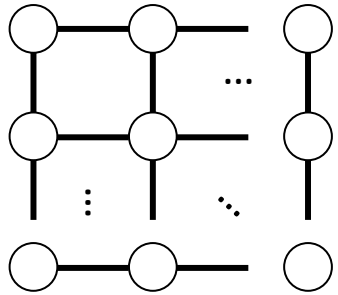
$$F(X) = \sum_{u=1}^{U} U(X_u) + \sum_{v=1}^{V} V(X_v), \qquad \text{and}$$

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

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

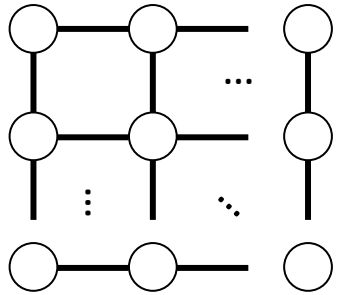
$$-\log P(x_1, x_2, x_3, x_4) \propto \sum_{u=1}^{U} U(X_u) + \sum_{v=1}^{V} V(X_v)$$

Minimizing MRF energies



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

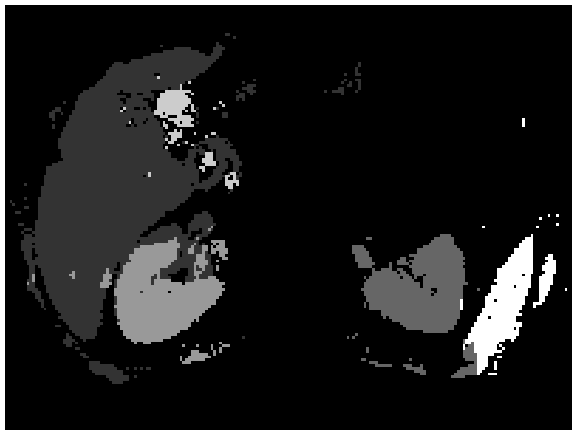
Example: Image Segmentation



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

Original Image

Ground truth



MRF segmentation

From Bhole et al. (2013)

Computing Probabilities

Computing marginal probabilities

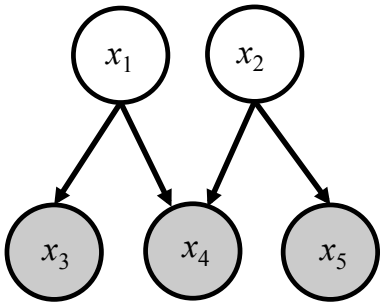
- The marginal for variable x_i is

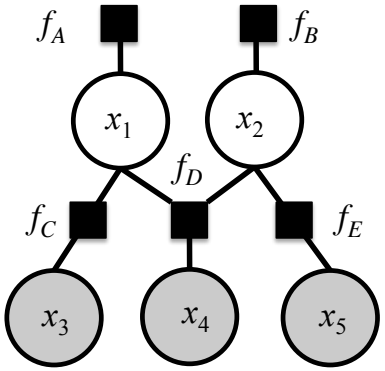
$$P(x_i) = \sum_{x_{j \neq i}} P(x_1, \ldots, x_n),$$

where the sum is over the states of all variables $x_j \neq x_i$

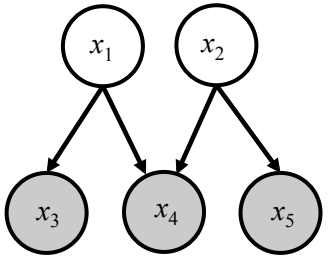
- Consider the task of computing the marginal *conditional* probability of variable x_3 given an observation for x_4 from a model where

$$P(x_1, \ldots, x_5) = P(x_1)P(x_2)$$
$$P(x_3 \mid x_1)P(x_4 \mid x_1, x_2)P(x_5 \mid x_2)$$





Marginal probabilities



- Since other variables in the graph have not been observed, they should be integrated out of the graphical model to obtain the desired result, ex.

$$P(x_3 \mid \tilde{x}_4) = \frac{P(x_3, \tilde{x}_4)}{P(\tilde{x}_4)} = \frac{P(x_3, \tilde{x}_4)}{\sum_{x_3} P(x_3, \tilde{x}_4)},$$ where the key quantity is

$$P(x_3, \tilde{x}_4) = \sum_{x_1} \sum_{x_2} \sum_{x_5} P(x_1, x_2, x_3, \tilde{x}_4, x_5)$$

$$= \sum_{x_1} \sum_{x_2} \sum_{x_5} P(x_1) P(x_2) P(x_3 \mid x_1) P(\tilde{x}_4 \mid x_1, x_2) P(x_5 \mid x_2).$$

- However, this sum involves a large data structure containing the joint probability, composed of the products over the individual probabilities

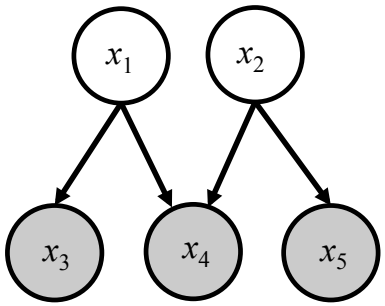
Intuition for the sum-product algorithm

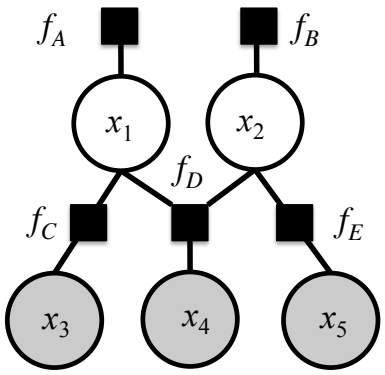
- The *sum-product algorithm* refers to a much better solution for computing marginals: simply *push the sums as far as possible to the right* before computing products of probabilities
- In our example the required marginalization can be computed by

$$P(x_3, \tilde{x}_4) = \sum_{x_1} P(x_3 \mid x_1) P(x_1) \sum_{x_2} P(\tilde{x}_4 \mid x_1, x_2) P(x_2) \sum_{x_5} P(x_5 \mid x_2)$$

$$= \sum_{x_1} P(x_3 \mid x_1) P(x_1) P(\tilde{x}_4 \mid x_1)$$

$$= \sum_{x_1} P(x_1, x_3, \tilde{x}_4).$$





The sum-product algorithm

- Computes exact marginals in tree structured factor graphs
- Begin with variable or function nodes that have only one connection (leaf nodes)
- Function nodes send the message: $\mu_{f \to x}(x) = f(x)$ to the variable connected to them
- Variable nodes send the message: $\mu_{x \to f}(x) = 1$
- Other nodes wait until they have received a message from all neighbors except the one will send a message to

Function to variable messages

- When ready, function nodes send messages of the following form to variable *x*:

$$\mu_{f\to x}(x) = \sum_{x_1,\ldots x_K} f(x, x_1, \ldots, x_K) \prod_{k\in N(f)\setminus x} \mu_{x_k\to f}(x_k),$$

- where $N(f)\setminus x$ represents the set of the function node *f*'s neighbors, excluding the recipient variable *x*
- Variables of the *K* other neighboring nodes are $x_1,\ldots,x_K$
- If a variable is observed, messages for functions involving it no longer need a sum over states of the variable, the function is evaluated with the observation
- One could think of the associated variable node as being transformed into the new modified function

Variable to function messages

- Variable nodes send messages to functions of form:

$$\mu_{x\to f}(x) = \mu_{f_1\to x}(x)\ldots\mu_{f_K\to x}(x) = \prod_{k\in N(x)\setminus f}\mu_{f_k\to x}(x),$$

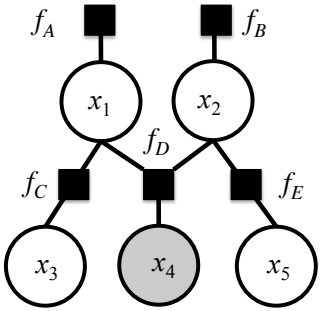
- where the product is over the (*K*) messages from all neighboring functions *N*(*x*) other than the recipient function *f, i.e.* $f_k \in N(x)\setminus f$
- The marginal for each node is obtained from the product over all *K*+1 incoming messages from all functions connected to a variable

$$P(x_i) = \mu_{f_1\to x}(x)\ldots\mu_{f_K\to x}(x)\mu_{f_{K+1}\to x}(x) = \prod_{k=1}^{K+1}\mu_{f_k\to x}(x)$$

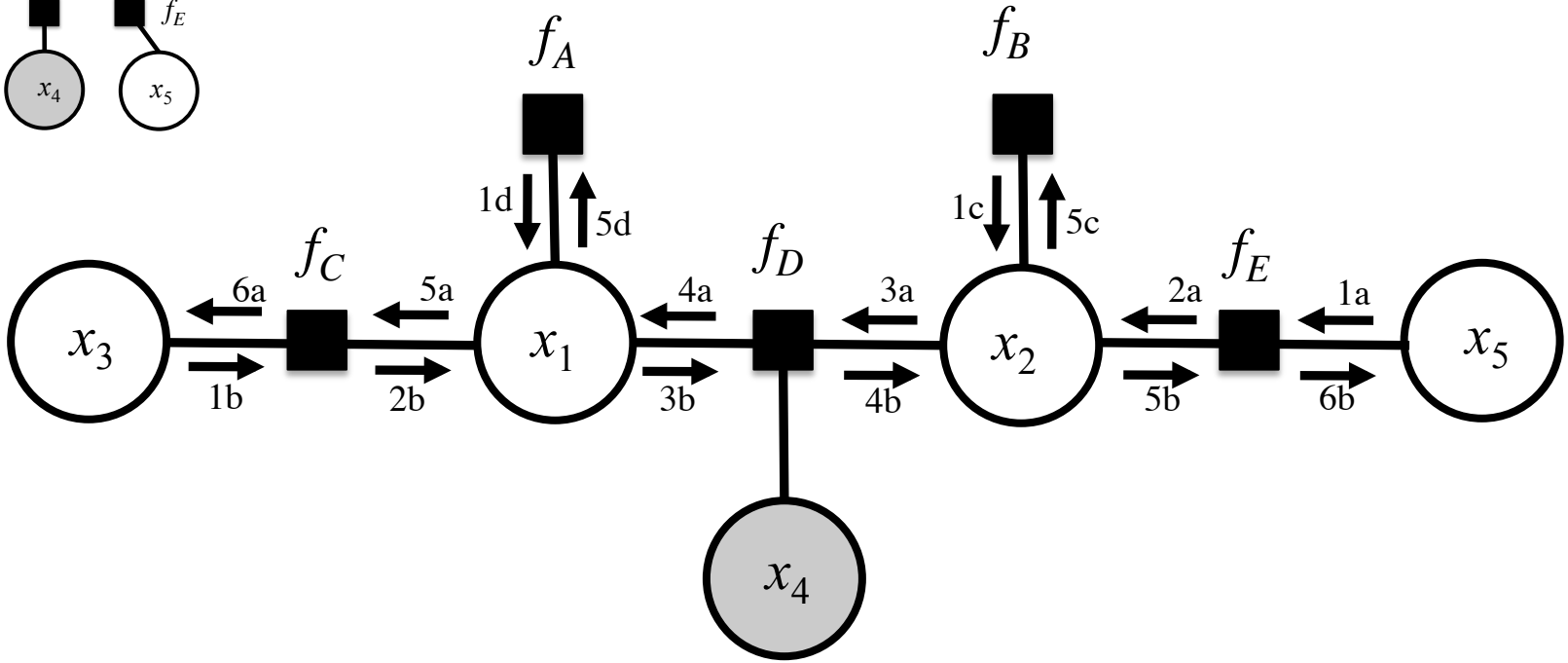
Numerical stability

- Multiplying many probabilities quickly leads to very small numbers
- The sum-product algorithm is often implemented with re-scaling
- Alternatively or additionally the computations can be performed in log space leading to computations of the form $c = \log(\exp(a) + \exp(b))$
- To help prevent loss of precision when computing the exponents, use the equivalent expression with the smaller exponent below

$$c = \log(e^a + e^b) = a + \log(1 + e^{b-a}) = b + \log(1 + e^{a-b})$$



Sum product messages



$$P(x_3, \tilde{x}_4) = \sum_{x_1} P(x_3 \mid x_1) \underbrace{P(x_1)}_{1d} \sum_{x_2} P(\tilde{x}_4 \mid x_1, x_2) \underbrace{P(x_2)}_{1c} \sum_{x_5} P(x_5 \mid x_2) \cdot \underbrace{1}_{1a}$$

$$\underbrace{\qquad\qquad}_{2a}$$

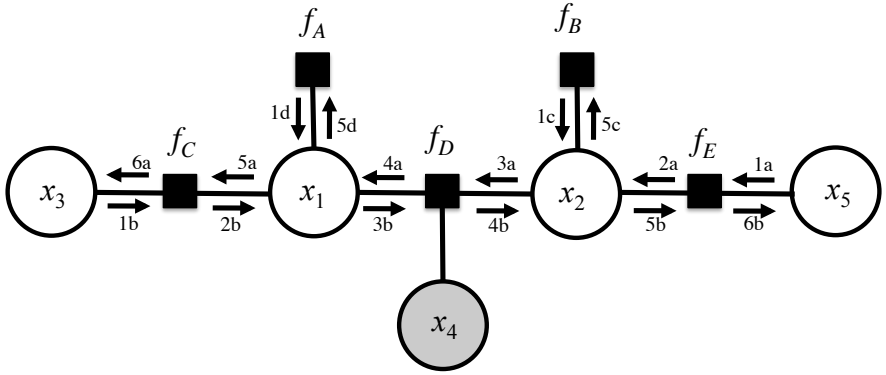
$$\underbrace{\qquad\qquad\qquad\qquad}_{3a}$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{4a}$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{5a}$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{6a}$$

The messages for our example



$$P(x_3,\tilde{x}_4)=\sum_{x_1}P(x_3\mid x_1)\underbrace{P(x_1)}_{1d}\sum_{x_2}P(\tilde{x}_4\mid x_1,x_2)\underbrace{P(x_2)}_{1c}\sum_{x_5}P(x_5\mid x_2)\cdot\underbrace{1}_{1a}$$

$2a$, $3a$, $4a$, $5a$, $6a$

$$1\text{a}:\mu_{x_5\to f_E}(x_5)=1,\ 1\text{c}:\mu_{f_B\to x_2}(x_2)=f_B(x_2),\ \ 1\text{d}:\mu_{f_A\to x_1}(x_1)=f_A(x_1)$$

$$2\text{a}:\mu_{f_E\to x_2}(x_2)=\sum_{x_5}f_E(x_5,x_2)$$

$$3\text{a}:\mu_{x_2\to f_D}(x_5)=\mu_{f_B\to x_2}(x_2)\mu_{f_E\to x_2}(x_2)$$

$$4\text{a}:\mu_{f_D\to x_1}(x_1)=\sum_{x_2}f_D(\tilde{x}_4\mid x_1,x_2)\mu_{x_2\to f_D}(x_5)$$

$$5\text{a}:\mu_{x_1\to f_C}(x_1)=\mu_{f_A\to x_1}(x_1)\mu_{f_D\to x_1}(x_1)$$

$$6\text{a}:\mu_{f_C\to x_3}(x_3)=\sum_{x_1}f_C(x_3,x_1)\mu_{x_1\to f_C}(x_1)$$

The complete algorithm can yield all single-variable marginals in the graph using the other messages shown in the diagram

Finding the most probable configuration

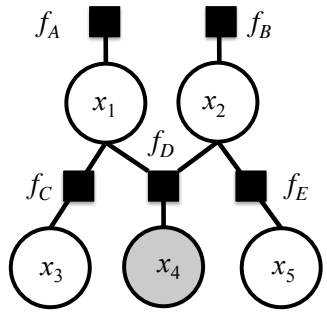
- Finding the most probable configuration of all other variables in our example given $x_4 = \tilde{x}_4$ involves searching for

$$\left\{x_1^*, x_2^*, x_3^*, x_5^*\right\} = \underset{x_1, x_2, x_3, x_5}{\operatorname{argmax}} P(x_1, x_2, x_3, x_5 \mid \tilde{x}_4),$$

for which

$$P(x_1^*, x_2^*, x_3^*, x_5^* \mid \tilde{x}_4) = \max_{x_1, x_2, x_3, x_5} P(x_1, x_2, x_3, x_5 \mid \tilde{x}_4).$$

Pushing max to the right



- Because max behaves in a similar way to sum, like in the sum-product algorithm we can push the max operations as far to the right as possible, noting that $\max(ab, ac) = a \max(b, c)$
- For our example we have

$$\max_{x_1} \max_{x_2} \max_{x_3} \max_{x_5} P(x_1, x_2, x_3, \tilde{x}_4, x_5)$$

$$= \max_{x_3} \max_{x_1} P(x_1) P(x_3 \mid x_1) \max_{x_2} P(x_2) P(\tilde{x}_4 \mid x_1, x_2) \max_{x_5} P(x_5 \mid x_2).$$

The max-sum algorithm

- A log-space version of max-product
- As in the sum-product algorithm, variables or factors that have only one connection in the graph begin by sending either :
  - a function-to-variable message $\mu_{x \to f}(x) = 0$
  - or a variable-to-function message $\mu_{f \to x}(x) = \log f(x)$
- Each function and variable node in the graph waits until it has received a message from all neighbors other than the node that will receive its message

Function to variable messages

- Each function and variable node in the graph waits until it has received a message from all neighbors other than the node that will receive its message
- Function nodes send messages of the following form to variable *x*

$$\mu_{f\to x}(x) = \max_{x_1,\ldots,x_K}\left[\log f(x,x_1,\ldots,x_K) + \sum_{k\in N(f)\setminus x}\mu_{x_k\to f}(x_k)\right],$$

where the notation $N(f)\setminus x$ is the same as for the sum-product algorithm above

Variable to function messages

- Variables send messages to functions of this form

$$\mu_{x \to f}(x) = \sum_{k \in N(x) \setminus f} \mu_{f_k \to x}(x),$$

  where the sum is over the messages from all functions other than the recipient function.

- When the algorithm terminates, the probability of the most probable configuration (MPC) can be extracted from any node using

$$p^* = \max_x \left[\sum_{k \in N(x)} \mu_{f_k \to x}(x) \right],$$

the MPC itself is:

$$x^* = \arg\max_x \left[\sum_{k \in N(x)} \mu_{f_k \to x}(x) \right].$$

Bibliographic Notes & Further Reading

Graphical Probabilty Models (GPMs) and Inference

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

Bibliographic Notes & Further Reading

Graphical Probabilty Models and Inference

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

Classical Simple Statistical Models as Conditional Probability Models

Conditional continuous probability models

- Consider a model where the conditional probability for y_i given x_i is a Gaussian with mean given by a linear function of x:

$$p(y_i \mid x_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{\left\{ y_i - (\theta_0 + \theta_1 x_i) \right\}^2}{2\sigma^2} \right],$$

- The conditional distribution for N y_is given the corresponding x_is can be defined as

$$p(y_1, \ldots, y_N \mid x_1, \ldots, x_N) = \prod_{i=1}^{N} p(y_i \mid x_i).$$

Linear regression as a probability model

- The log-likelihood of our linear model is:

$$L_{y|x} = \log \prod_{i=1}^{N} p(y_i \mid x_i) = \sum_{i=1}^{N} \log p(y_i \mid x_i).$$

$$L_{y|x} = \sum_{i=1}^{N} \log \left\{ \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{\left\{ y_i - (\theta_0 + \theta_1 x_i) \right\}^2}{2\sigma^2} \right] \right\}$$

$$= -N \log\left[\sigma\sqrt{2\pi}\right] - \sum_{i=1}^{N} \frac{\left\{ y_i - (\theta_0 + \theta_1 x_i) \right\}^2}{2\sigma^2}.$$

- To maximize the log-likelihood it suffices to find the parameters that minimize the squared error

$$\arg\max_{\theta_0\ \theta_1}\left(L_{y|x}\right) = \arg\min_{\theta_0\ \theta_1}\left(\sum_{i=1}^{N}\left\{ y_i - (\theta_0 + \theta_1 x_i) \right\}^2\right).$$

  i.e. this probability model is simply linear regression

Priors on parameters, Gaussians, ridge regression and weight decay

- Placing a zero mean Gaussian prior on the parameters **w** leads to the method of *ridge regression,* also called *weight decay*
- Consider a linear regression that uses a *D* dimensional vector **x** to make predictions
- The regression's bias term can be represented by appending a constant feature = 1 as an additional final dimension of **x** for every example
- The prior is frequently omitted from the bias
- The underlying probability model can be written

$$\prod_{i=1}^{N} p(y_i \mid x_i;\theta)p(\theta;\tau) = \left[\prod_{i=1}^{N} N(y_i;\mathbf{w}^{\mathrm{T}}x_i,\sigma^2)\right]\left[\prod_{d=1}^{D} N(w_d;0,\tau^2)\right]$$

Priors on parameters, L_2 and L_1 regularization

- Maximum a posteriori parameter estimation based on the log conditional likelihood with a zero mean Gaussian prior on weights is equivalent to minimizing a squared error loss function plus an L_2 based regularization term

$$F(\mathbf{w}) = \sum_{i=1}^{N} \left\{ y_i - \mathbf{w}^{\mathrm{T}} x_i \right\}^2 + \lambda R_{L_2}(\mathbf{w}), \quad R_{L_2}(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w} = \left\| \mathbf{w} \right\|_2^2$$

- Using a Laplace prior for the distribution over weights and taking the log of the likelihood function yields an L_1 based regularization term

$$R_{L_1}(\mathbf{w}) = \left\| \mathbf{w} \right\|_1$$

Laplace, L_1 regularization the LASSO and the Elastic Net approach

- The Laplace distribution with params μ and b is

$$P(w;\mu,b) = L(w;\mu,b) = \frac{1}{2b}\exp\left(-\frac{|w-\mu|}{b}\right)$$

- Modeling the log of the prior probability for each weight by a Laplace distribution with $\mu=0$ yields

$$-\log\left[\prod_{d=1}^{D} L(w_d;0,b)\right] = \log(2b) + \frac{1}{b}\sum_{d=1}^{D}|w_d| \propto \|\mathbf{w}\|_1$$

- The use of L_1 regularization is also know as the LASSO, "Least Absolute Shrinkage and Selection Operator".
- An alternative (convex!) approach known as the *elastic net* combines L_1 and L_2 regularization techniques using

$$\lambda_1 R_{L_1}(\boldsymbol{\theta}) + \lambda_2 R_{L_2}(\boldsymbol{\theta}) = \lambda_1 \|\mathbf{w}\|_1 + \lambda_2 \|\mathbf{w}\|_2^2$$

Matrix vector form of regression

- Linear regression can be written in matrix form

$$\sum_{i=1}^{N}\{y_i-(\theta_0+\theta_1 x_i)\}^2=\left(\begin{bmatrix}y_1\\ \vdots\\ y_N\end{bmatrix}-\begin{bmatrix}1 & x_1\\ \vdots & \vdots\\ 1 & x_N\end{bmatrix}\begin{bmatrix}\theta_0\\ \theta_1\end{bmatrix}\right)^T\left(\begin{bmatrix}y_1\\ \vdots\\ y_N\end{bmatrix}-\begin{bmatrix}1 & x_1\\ \vdots & \vdots\\ 1 & x_N\end{bmatrix}\begin{bmatrix}\theta_0\\ \theta_1\end{bmatrix}\right)$$

$$=(\mathbf{y}-\mathbf{A}\mathbf{w})^T(\mathbf{y}-\mathbf{A}\mathbf{w}),$$

- Taking the partial derivative with respect to **w** and setting the result to zero yields a *closed form* expression for the parameters:

$$\frac{\partial}{\partial \mathbf{w}}(\mathbf{y}-\mathbf{A}\mathbf{w})^{\mathrm{T}}(\mathbf{y}-\mathbf{A}\mathbf{w})=0 \quad \Rightarrow \mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{w}=\mathbf{A}^{\mathrm{T}}\mathbf{y}, \quad \mathbf{w}=(\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{y}$$

$\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{w}=\mathbf{A}^{\mathrm{T}}\mathbf{y}$ are the famous "normal equations"

$\mathbf{A}^{+}=(\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}$ is known as the *pseudoinverse*

Linear regression with regularization

- For ridge regression our objective function becomes

$$F(\mathbf{w}) = (\mathbf{y} - \mathbf{A}\mathbf{w})^T (\mathbf{y} - \mathbf{A}\mathbf{w}) + \lambda \mathbf{w}^{\mathrm{T}} \mathbf{w}$$

- We get a closed form solution for the result

$$\frac{\partial}{\partial \mathbf{w}} F(\mathbf{w}) = 0$$

$$\Rightarrow \mathbf{A}^{\mathrm{T}} \mathbf{A}\mathbf{w} + \lambda \mathbf{w} = \mathbf{A}^{\mathrm{T}} \mathbf{y}$$

$$\mathbf{w} = (\mathbf{A}^{\mathrm{T}} \mathbf{A} + \lambda \mathbf{I})^{-1} \mathbf{A}^{\mathrm{T}} \mathbf{y}$$

- This modification to the pseudoinverse equation is very useful, allowing solutions to be found when they would otherwise not exist, often using a very small λ
- The extension to multidimensional inputs **x** can still be formulated using these matrix forms

Polynomial regression and kernels

- We can create a model for non-linear predictions using polynomials of x
- The estimation problem remains linear!

$$\sum_{i=1}^{N}\left\{y_i - (\theta_0 + \theta_1 x_i + \theta_2 x_i^2 + \ldots + \theta_K x_i^K)\right\}^2$$

$$= \left(\begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^K \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_N & x_N^2 & & x_N^K \end{bmatrix} \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_K \end{bmatrix}\right)^{\mathbf{T}} \left(\begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^K \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_N & x_N^2 & & x_N^K \end{bmatrix} \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_K \end{bmatrix}\right)$$

$$= (\mathbf{y} - \mathbf{Ac})^T (\mathbf{y} - \mathbf{Ac}).$$

- A similar trick for transforming a linear model into a non-linear model is based on using basis functions $\boldsymbol{\phi}(\mathbf{x})$, where $p(y \mid \mathbf{x}) = N(y; \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}), \sigma^2)$

Multinomial logistic regression

- A simple linear probabilistic classifier can be created using this parametric form

$$p(y \mid \mathbf{x}) = \frac{\exp\left(\sum_{k=1}^{K} w_k f_k(y,\mathbf{x})\right)}{\sum_{y} \exp\left(\sum_{k=1}^{K} w_k f_k(y,\mathbf{x})\right)} = \frac{1}{Z(\mathbf{x})} \exp\left(\sum_{k=1}^{K} w_k f_k(y,\mathbf{x})\right),$$

  where $y \in \{1,\ldots,N\}$, we use *K feature functions* $f_k(y,\mathbf{x})$ and K weights, w_k for the parameters
- We can perform learning using maximum conditional likelihood and N observations for both labels and features, $\{\tilde{y}_1,\ldots,\tilde{y}_N,\tilde{\mathbf{x}}_1,\ldots,\tilde{\mathbf{x}}_N\}$

Multiclass logistic regression and gradient computations

- The objective function (has no closed form solution)

$$L_{y|x} = \log \prod_{i=1}^{N} p(\tilde{y}_i \mid \tilde{\mathbf{x}}_i) = \sum_{i=1}^{N} \log p(\tilde{y}_i \mid \tilde{\mathbf{x}}_i)$$

- Soln: gradient descent - the derivative w.r.t. one weight is

$$\frac{\partial}{\partial w_j} p(\tilde{y} \mid \tilde{\mathbf{x}}) = \frac{\partial}{\partial w_j} \left\{ \log \left(\frac{1}{Z(\tilde{\mathbf{x}})} \exp \left(\sum_{k=1}^{K} w_k f_k(\tilde{y}, \tilde{\mathbf{x}}) \right) \right) \right\}$$

$$= \frac{\partial}{\partial w_j} \left\{ \left[\sum_{k=1}^{K} \underbrace{w_k f_k(\tilde{y}, \tilde{\mathbf{x}})}_{\text{easy part}} \right] - \underbrace{\log Z(\tilde{\mathbf{x}})}_{\text{cool part}} \right\}$$

$$= f_{k=j}(\tilde{y}, \tilde{\mathbf{x}}) - \frac{\partial}{\partial w_j} \left\{ \log \left[\sum_{y} \exp \left(\sum_{k=1}^{K} w_k f_k(y, \tilde{\mathbf{x}}) \right) \right] \right\},$$

we see the derivative breaks apart into two key terms

- The first term, the "easy part" involves terms that disappear when $w_k \neq, w_j$ leaving only the term shown on the left

Understanding gradients for logistic regression

- The second term, while seemingly daunting, has a "cool" derivative that yields an intuitive and interpretable result:

$$-\frac{\partial}{\partial w_j}\left\{\log Z(\tilde{\mathbf{x}})\right\} = -\frac{\partial}{\partial w_j}\left\{\log\left[\sum_y \exp\left(\sum_{k=1}^{K} w_k f_k(y,\tilde{\mathbf{x}})\right)\right]\right\}$$

$$= -\frac{\sum_y \exp\left(\sum_{k=1}^{K} w_k f_k(y,\tilde{\mathbf{x}})\right) f_j(y,\tilde{\mathbf{x}})}{\sum_y \exp\left(\sum_{k=1}^{K} w_k f_k(y,\tilde{\mathbf{x}})\right)} = -\sum_y p(y|\tilde{\mathbf{x}}) f_j(y,\tilde{\mathbf{x}}) = -\mathrm{E}[f_j(y,\tilde{\mathbf{x}})]_{p(y|\tilde{\mathbf{x}})},$$

  which corresponds to the expectation of the feature function under the probability distribution given by the model with the current parameter settings

- Using vectorized **w** and feature functions **f** we have

$$\frac{\partial}{\partial \mathbf{w}} L_{y|x} = \sum_{i=1}^{N}\left[\mathbf{f}(\tilde{y}_i \mid \tilde{\mathbf{x}}_i) - \mathrm{E}[\mathbf{f}(y_i \mid \tilde{\mathbf{x}}_i)]_{P(y_i|\tilde{\mathbf{x}}_i)}\right]$$

Reformulating multiclass logistic regression

- Could alternatively formulate logistic regression as:

$$p(y = c \mid \mathbf{x}) = \frac{\exp\left(\mathbf{w}_c^T \mathbf{x}\right)}{\sum_{\mathbf{y}} \exp\left(\mathbf{w}_y^T \mathbf{x}\right)},$$

  where *y* is an integer index, the weights are encoded into a vector of length *K,* and
- The features **x** have been re-defined as the result of evaluating the feature functions $f_k(y,\mathbf{x})$ in such a way that there is no difference between the features given by $f_k(y=i,\mathbf{x})$ and $f_k(y=j,\mathbf{x})$.
- This form is widely used for the last layer in neural network models, where it is referred to as the *softmax* function

Matrix vector formulation of multiclass logistic regression

- The information concerning class labels could be encoded into a *multinomial* or *one-hot vector* **y**, which is all zeros except for a single 1 in the dimension that represents the correct class label—for example, $\mathbf{y} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \end{bmatrix}^T$ for the second class.
- The weights form a matrix, $\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 & \mathbf{w}_2 & \dots & \mathbf{w}_K \end{bmatrix}^T$ and the biases form a vector, $\mathbf{b} = \begin{bmatrix} b_1 & b_2 & \dots & b_K \end{bmatrix}^T$
- The model can be formulated so as to yield *vectors* of probabilities

Logistic regression in matrix vector form

- With the previous definitions, we can now write

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{\exp\left(\mathbf{y}^T \mathbf{W}\mathbf{x} + \mathbf{y}^T \mathbf{b}\right)}{\sum_{\mathbf{y} \in Y} \exp\left(\mathbf{y}^T \mathbf{W}^T \mathbf{x} + \mathbf{y}^T \mathbf{b}\right)},$$

  where the denominator sums over each possible label,

$$\mathbf{y} \in Y, \quad Y = \{[\ 1\ \ 0\ \ 0\ \ \ldots\ \ 0\ ]^T, \ldots, [\ 0\ \ 0\ \ 0\ \ \ldots\ \ 1\ ]^T\}$$

- Re-defining $\mathbf{x}$ as $\mathbf{x} = [\mathbf{x}^T\ 1]^T$ and the parameters as a matrix of the form

$$\boldsymbol{\theta} = [\mathbf{W}\ \ \mathbf{b}] = \begin{bmatrix} \mathbf{w}_1^T & b_1 \\ \mathbf{w}_2^T & b_2 \\ \vdots & \vdots \\ \mathbf{w}_k^T & b_k \end{bmatrix},$$

Logistic regression in matrix vector form

- We can now write logistic regression very compactly as

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{\exp\left(\mathbf{y}^T \boldsymbol{\theta} \mathbf{x}\right)}{\sum_{\mathbf{y} \in Y} \exp\left(\mathbf{y}^T \boldsymbol{\theta} \mathbf{x}\right)} = \frac{1}{Z(\mathbf{x})} \exp\left(\mathbf{y}^T \boldsymbol{\theta} \mathbf{x}\right)$$

- The gradient of the log-conditional-likelihood with respect to the parameter *matrix* $\boldsymbol{\theta}$ is

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\theta}} \log \prod_i p(\tilde{\mathbf{y}}_i \mid \tilde{\mathbf{x}}_i; \boldsymbol{\theta}) &= \sum_{i=1}^{N} \left[\frac{\partial}{\partial \boldsymbol{\theta}} (\tilde{\mathbf{y}}_i^T \boldsymbol{\theta} \tilde{\mathbf{x}}_i) - \frac{\partial}{\partial \boldsymbol{\theta}} \log Z(\tilde{\mathbf{x}}_i) \right] \\
&= \sum_{i=1}^{N} \tilde{\mathbf{y}}_i \tilde{\mathbf{x}}_i^T - \sum_{i=1}^{N} \sum_{\mathbf{y} \in Y} P(\mathbf{y} \mid \tilde{\mathbf{x}}_i) \mathbf{y} \tilde{\mathbf{x}}_i^T \\
&= \sum_{i=1}^{N} \tilde{\mathbf{y}}_i \tilde{\mathbf{x}}_i^T - \sum_{i=1}^{N} \mathrm{E}[\mathbf{y} \tilde{\mathbf{x}}_i^T]_{P(\mathbf{y} \mid \tilde{\mathbf{x}}_i)}
\end{aligned}$$

- The second term transforms into a vector of probabilities for the classes of **y** under the model, multiplied by the observed **x** transpose

Gradient descent

- Given a conditional probability model $p(y \mid \mathbf{x}; \boldsymbol{\theta})$
- Parameter vector $\boldsymbol{\theta}$, data $\tilde{y}_i, \tilde{\mathbf{x}}_i, i = 1 \ldots N$
- Prior on parameters, $p(\boldsymbol{\theta}; \lambda)$ with hyperparameter, λ
- Gradient descent with learning rate η can be written as:

$$\boldsymbol{\theta} = \boldsymbol{\theta}_o \quad \text{// initialize parameters}$$

$$\text{while converged} \;==\; \text{FALSE}$$

$$\mathbf{g} \;=\; \frac{\partial}{\partial \boldsymbol{\theta}} \left[-\sum_{i=1}^{N} \log p(\tilde{y}_i \mid \tilde{\mathbf{x}}_i; \boldsymbol{\theta}) - \log p(\boldsymbol{\theta}; \lambda) \right]$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \mathbf{g}$$

- For convex models the change in the loss or the parameters is often monitored and the algorithm is terminated when it stabilizes

Second order methods

- Alternatively, gradient descent can be based on the second derivative by computing the Hessian matrix **H** at each iteration and using updates

$$\mathbf{H} = \frac{\partial^2}{\partial \boldsymbol{\theta}^2}\left[-\sum_{i=1}^{N} \log p(\tilde{y}_i \mid \tilde{\mathbf{x}}_i; \boldsymbol{\theta}) - \log p(\boldsymbol{\theta}; \lambda)\right],$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \mathbf{H}^{-1}\mathbf{g}.$$

- To understand why this is possible and the relationship to learning rates, see Appendix A.1

Generalized linear models

- Linear regression and logistic regression are special cases of a family of conditional probability models known in statistics as "generalized linear models" (GLMs)
- Idea: unify and generalize linear and logistic regression
- Data can be thought of in terms of response variables y_i and explanatory variables organized as vectors $\mathbf{x}_i$, i=1,…,n.
- Response variables could be expressed in different ways, ranging from binary to categorical or ordinal data
- A model is then defined where μ or the expected value of the distribution used for the response variable consists of an initial linear prediction which is then subjected to a smooth, invertible and potentially non-linear transformation using the *mean function* g^{-1}, i.e.

$$\mu_i = \mathrm{E}[y_i] = g^{-1}(\boldsymbol{\beta}^T \mathbf{x}_i)$$

- The mean function is the inverse of the *link function, g*

GLMs

- In GLMs the entire set of explanatory variables for all the observations is often arranged as a nxp matrix **X**, so that a vector of linear predictions for the entire data set is $\boldsymbol{\eta} = \mathbf{X}\boldsymbol{\beta}$
- The variance of the underlying distribution can also be modeled; typically as a function of the mean
- Different distributions, link functions and corresponding mean functions give a great deal of flexibility in defining probabilistic models, see examples below

Link Name	**Link Function** $\eta = \beta^T \mathbf{x} = g(\mu)$	**Mean Function** $\mu = g^{-1}(\beta^T \mathbf{x}) = g^{-1}(\eta)$	**Typical Distribution**
Identity	$\eta = \mu$	$\mu = \eta$	Gaussian
Inverse	$\eta = \mu^{-1}$	$\mu = \eta^{-1}$	Exponential
Log	$\eta = \log_e \mu$	$\mu = \exp(\eta)$	Poisson
Log-log	$\eta = -\log(-\log_e \mu)$	$\mu = \exp(-\exp(-\eta))$	Bernoulli
Logit	$\eta = \log_e \frac{\mu}{1-\mu}$	$\mu = \frac{1}{1+\exp(-\eta)}$	Bernoulli
Probit	$\eta = \Phi^{-1}(\mu)$	$\mu = \Phi(\eta)$	Bernoulli

Note: $\Phi(\cdot)$ *is the cumulative normal distribution.*

Predictions for ordered classes

- To define a model with *M* ordinal categories, *M*–1 *cumulative* probability models of the form $P(Y_i \leq j)$ can be used where the random variable Y_i represents the category of a given instance *i*
- Models for $P(Y_i = j)$ can then be obtained using differences between the cumulative distribution models
- Here we will use complementary cumulative probabilities, known as *survival functions,* of the form $P(Y_i > j) = 1 - P(Y_i \leq j)$
- They sometimes simplify the interpretation of parameters
- The class probabilities can be obtained from:

$$P(Y_i = 1) = 1 - P(Y_i > 1)$$

$$P(Y_i = j) = P(Y_i > j - 1) - P(Y_i > j)$$

$$P(Y_i = M) = P(Y_i > M - 1).$$

- For binary predictions the following model is popular

$$\text{logit}(\gamma_{ij}) = \log \frac{\gamma_{ij}}{1 - \gamma_{ij}} = b_j + \mathbf{w}^T \mathbf{x}_i$$

Conditional probability models based on kernels

- Linear models can be transformed into non-linear ones by applying the "kernel trick"
- Suppose the features **x** are replaced by the vector **k**(**x**) whose elements are determined using a kernel function $k(\mathbf{x},\mathbf{x}_j)$ for every training example:

$$\mathbf{k}(\mathbf{x}) = \begin{bmatrix} k(\mathbf{x},\mathbf{x}_1) \\ \vdots \\ k(\mathbf{x},\mathbf{x}_V) \\ 1 \end{bmatrix}$$

- A "1" has been appended to this vector to implement the bias term in the parameter matrix
- Kernelized regression

$$p(y \mid \mathbf{x}) = N(y; \mathbf{w}^{\mathrm{T}}\mathbf{k}(\mathbf{x}), \sigma^2)$$

- Kernelized classification

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{\exp\left(\mathbf{y}^T \boldsymbol{\theta}\mathbf{k}(\mathbf{x})\right)}{\sum_{\mathbf{y}} \exp\left(\mathbf{y}^T \boldsymbol{\theta}\mathbf{k}(\mathbf{x})\right)}$$

Training and evaluating deep networks

Data Mining

Practical Machine Learning Tools and Techniques

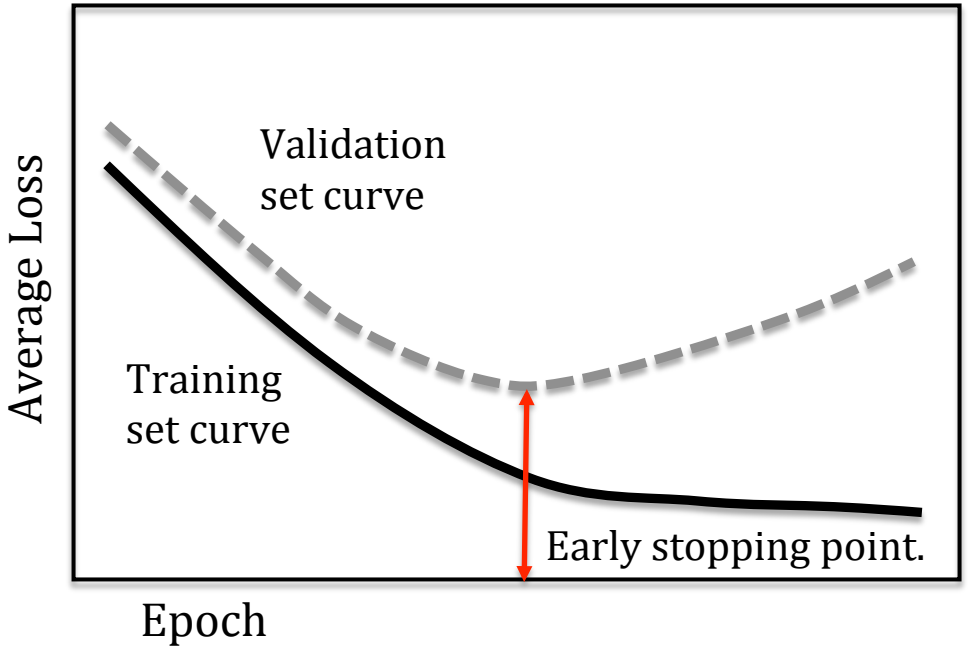
A Selection of Slides from Chapter 10, Deep learning

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

Early stopping

- Deep learning involves high capacity architectures, which are susceptible to overfitting even when data is plentiful,
- Early stopping is standard practice even when other methods to reduce overfitting are employed, ex. regularization and dropout
- The idea is to monitor learning curves that plot the average loss for the training and validation sets as a function of epoch
- The key is to find the point at which the validation set average loss begins to deteriorate

Early stopping



- In practice the curves above can be more noisy due to the use of stochastic gradient descent
- As such, it is common to keep the history of the validation set curve when looking for the minimum – even if it goes back up it might come back down

Validation sets and hyperparameters

- In deep learning hyperparameters are tuned by identifying what settings lead to best performance on the validation set, using early stopping
- Common hyperparameters include the strength of parameter regularization, but also model complexity in terms of the number of hidden units and layers and their connectivity, the form of activation functions, and parameters of the learning algorithm itself.
- Because of the many choices involved, performance monitoring on validation sets assumes an even more central role than it does with traditional machine learning methods.

Test sets

- *Should be set aside for a truly final evaluation*
- Repeated rounds of experiments using test set data give misleading (ex. optimistic) estimates of performance on fresh data
- For this reason, the research community has come to favor public challenges with hidden test-set labels, a development that has undoubtedly helped gauge progress in the field
- Controversy arises when participants submit multiple entries, and some favor a model where participants submit code to a competition server, so that the test data itself is hidden

Validation sets vs. cross-validation

- The use of a validation set is different from using *k*-fold cross-validation to evaluate a learning technique or to select hyperparameters.
- Cross-validation involves creating multiple training and testing partitions.
- Datasets for deep learning tend to be so massive that a single large test set adequately represents a model's performance, reducing the need for cross-validation
  - Since training often takes days or weeks, even using GPUs, cross-validation is often impractical anyway.
- If you do use cross validation you need to have an intern validation set *for each fold* to adjust hyperparameters or perform cross validation only using the training set

Validation set data and the ‘end game’

- To obtain the best possible results, one needs to tune hyperparameters, usually with a single validation set extracted from the training set.
- However, there is a dilemma: omitting the validation set from final training can reduce performance in the test.
- It is advantageous to train on the combined training and validation data, but this risks overfitting.
- One solution is to stop training after the same number of epochs that led to the best validation set performance; another is to monitor the average loss over the combined training set and stop when it reaches the level it was at when early stopping was performed using the validation set.
- One can use cross validation within the training set, treating each fold as a different validation set, then train the final model on the entire training data with the identified hyperparameters to perform the final test

Hyperparameter tuning

- A weighted combination of L_2 and L_1 regularization is often used to regularize weights
- Hyperparameters in deep learning are often tuned heuristically by hand, or using grid search
- An alternative is random search, where instead of placing a regular grid over hyperparameter space, probability distributions are specified from which samples are taken
- Another approach is to use machine learning and Bayesian techniques to infer the next hyperparameter configuration to try in a sequence of experimental runs
- Keep in mind even things like the learning rate schedule (discussed below) are forms of hyperparameters and you need to be careful not to tune them on the test set

Mini-batch based stochastic gradient descent (SGD)

- Stochastic gradient descent updates model parameters according to the gradient computed from one example
- The mini-batch variant uses a small subset of the data and bases updates to parameters on the average gradient over the examples in the batch
- This operates just like the regular procedure: initialize the parameters, enter a parameter update loop, and terminate by monitoring a validation set
- Normally these batches are randomly selected disjoint subsets of the training set, perhaps shuffled after each epoch, depending on the time required to do so

Mini-batch based SGD

- Each pass through a set of mini-batches that represent the complete training set is an *epoch*
- Using the empirical risk plus a regularization term as the objective function, updates are

$$\theta^{\text{new}} \leftarrow \theta - \eta_t \left[\frac{1}{B_k} \sum_{i \in \mathrm{I}} \left[\frac{\partial}{\partial \theta} L(f(\mathbf{x}_i;\theta), \mathbf{y}_i) \right] + \frac{B_k}{N} \lambda \frac{\partial}{\partial \theta} R(\theta) \right]$$

- η_t is the learning rate and may depend on the epoch t
- The batch is represented by a set of indices I=I(t,k) into the original data; the kth batch has B_k examples
- N is the size of the training set
- $L(f(\mathbf{x}_i;\theta),\mathbf{y}_i)$ is the loss for example $\mathbf{x}_i$, label $\mathbf{y}_i$, params θ
- R(θ) is the regularizer, with weight λ

Mini-batches

- Typically contain two to several hundred examples
  - For large models the choice may be constrained by resources
- Batch size often influences the stability and speed of learning; some sizes work particularly well for a given model and data set.
- Sometimes a search is performed over a set of potential batch sizes to find one that works well, before doing a lengthy optimization.
- The mix of class labels in the batches can influence the result
  - For unbalanced data there may be an advantage in pre-training the model using mini-batches in which the labels are balanced, then fine-tuning the upper layer or layers using the unbalanced label statistics.

Momentum

- As with regular gradient descent, 'momentum' can help the optimization escape plateaus in the loss
- Momentum is implemented by computing a moving average: $\Delta\theta = -\eta\nabla_\theta L(\theta) + \alpha\Delta\theta^{\text{old}}$
  - where the first term is the current gradient of the loss times a learning rate
  - the second term is the previous update weighted by $\alpha \in [0,1]$
- Since the mini- batch approach operates on a small subset of the data, this averaging can allow information from other recently seen mini-batches to contribute to the current parameter update
- A momentum value of 0.9 is often used as a starting point, but it is common to hand-tune it, the learning rate, and the schedule used to modify the learning rate during the training process

Learning rate schedules

- The learning rate is a critical choice when using mini-batch based stochastic gradient descent.
- Small values such as 0.001 often work well, but it is common to perform a logarithmically spaced search, say in the interval [10^{-8}, 1], followed by a finer grid or binary search.
- The learning rate may be adapted over epochs t to give a learning rate schedule, ex. $\eta_t = \eta_0 (1 + \varepsilon t)^{-1}$
- A fixed learning rate is often used in the first few epochs, followed by a decreasing schedule
- Many other options, ex. divide the rate by 10 when the validation error rate ceases to improve

Mini-batch SGD pseudocode

$\theta = \theta_o$ // initialize parameters

$\Delta\theta = 0$

$t = 0$

while converged == FALSE

$\{I_1, \ldots, I_K\} = \text{shuffle}(X)$ // create K mini-batches

for $k = 1 \ldots K$

$$\mathbf{g} = \frac{1}{B_k} \sum_{i \in I_k} \left[\frac{\partial}{\partial \theta} L(f(\mathbf{x}_i; \theta), \mathbf{y}_i) \right] + \frac{B_k}{N} \lambda \frac{\partial}{\partial \theta} R(\theta)$$

$$\Delta\theta \leftarrow -\eta_t \mathbf{g} + \alpha \Delta\theta$$

$$\theta \leftarrow \theta + \Delta\theta$$

end

$t = t + 1$

end

Bibliographic Notes & Further Reading

Logistic Regression, GLMs and Regularized Regression

- Logistic regression is sometimes referred to as the workhorse of applied statistics; Hosmer and Lemeshow (2004) is a great reference
- Nelder and Wedderburn (1972)'s work led to the generalized linear modeling framework
- McCullagh (1980)'s developed proportional odds models for ordinal regression, which are sometimes called ordered logit models because they use the generalized logit function
- Frank and Hall (2001) showed how to adapt arbitrary machine learning techniques to ordered predictions
- McCullagh and Nelder (1989)'s widely cited monograph is another good source for details on the framework of GLMs
- Tibshirani (1996) developed the famous "Least Absolute Shrinkage and Selection Operator," also known as the LASSO
- Zou and Hastie (2005) developed the "elastic net" regularization approach which combines L_1 and L_2 regularization

Bibliographic Notes & Further Reading

Kernel Logistic Regression, and Various Vector Machines

- Kernel logistic regression transforms a linear classifier into a non-linear one, and probabilistic sparse kernel techniques are attractive alternatives to support vector machines
- Tipping (2001) proposed a "relevance vector machine" that manipulates priors on parameters in a way that encourages kernel weights to become zero during learning
- Lawrence, Seeger, and Herbrich (2003) proposed an "informative vector machine," which treats the problem as a fast, sparse Gaussian process method in the sense of Williams and Rasmussen (2006)
- Zhu and Hastie (2005) formulated sparse kernel logistic regression as an "import vector machine" that uses greedy search methods
- None of these methods approach the popularity of Cortes and Vapnik (1995)'s support vector machines, perhaps because their objective functions are not convex, in contrast to the underlying SVM (and L2 regularized kernel logistic regression)
- When probabilities are needed from an SVM, Platt (1999) shows how to fit a logistic regression to the classification scores.