Machine Learning: Foundations

(NGCM Introductory Course)

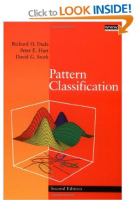
Mahesan Niranjan

School of Electronics and Computer Science
University of Southampton

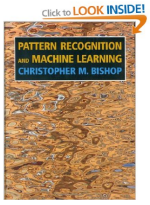
July 2019

Good Books

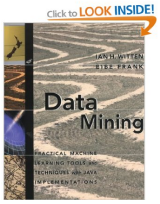
Well worth investing in books if you want to do more in this subject



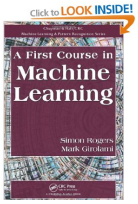
R.O.Duda, P.E.Hart & D.G.Stork
Pattern Classification



C.M. Bishop
Pattern Recognition and Machine Learning



I.H. Witten & E. Frank
Data Mining



S. Rogers & M. Girolami
A First Course in Machine Learning

"There is nothing to be learnt from a professor, which is not to be met with in books"
- David Hume (1711-1776)

(**WikiPedia:** "Hume had little respect for the professors of his time [...] He did not graduate")

Machine Learning as Data-driven Modelling

Data	$\{\boldsymbol{x}_n,\ \boldsymbol{y}_n\}_{n=1}^N \qquad \{\boldsymbol{x}_n\}_{n=1}^N$
Function Approximator	$\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{\theta}) + v$
Parameter Estimation	$E_0 = \sum_{n=1}^N \{\|\| \boldsymbol{y}_n - f(\boldsymbol{x}_n; \boldsymbol{\theta}) \|\|\}^2$
Prediction	$\hat{\boldsymbol{y}}_{N+1} = f\left(\boldsymbol{x}_{N+1}, \hat{\boldsymbol{\theta}}\right)$
Regularization	$E_1 = \sum_{n=1}^N \{\|\| \boldsymbol{y}_n - f(\boldsymbol{x}_n) \|\|\}^2 + g(\|\|\boldsymbol{\theta}\|\|)$
Modelling Uncertainty	$p\left(\boldsymbol{\theta} \mid \{\boldsymbol{x}_n, \boldsymbol{y}_n\}_{n=1}^N\right)$
Probabilistic Inference	$\boldsymbol{E}[g(\boldsymbol{\theta})] = \int g(\boldsymbol{\theta})\, p(\boldsymbol{\theta})\, d\boldsymbol{\theta} = \frac{1}{N_s} \sum_{n=1}^{N_s} g\left(\boldsymbol{\theta}^{(n)}\right)$
Sequential Estimation	$\boldsymbol{\theta}(n-1\|n-1) \longrightarrow \boldsymbol{\theta}(n\|n-1) \longrightarrow \boldsymbol{\theta}(n\|n)$ Kalman & Particle Filters; Reinforcement Learning

Gaussian Densities: Univariate and Multivariate

- Univariate Gaussian

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{1}{2}\frac{(x-m)^2}{\sigma^2}\right\}$$

- Multivariate Gaussian

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{p/2}(\det \boldsymbol{C})^{1/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{m})^t\, \boldsymbol{C}^{-1}\, (\boldsymbol{x}-\boldsymbol{m})\right\}$$

Mean $\boldsymbol{m}$ is a vector
Covariance, C, matrix: symmetric, positive definite!

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{m}, \boldsymbol{C}),\ \boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} \implies \boldsymbol{y} \sim \mathcal{N}(\boldsymbol{A}\boldsymbol{m}, \boldsymbol{A}\boldsymbol{C}\boldsymbol{A}^T)$$

Estimation

Univariate Mean	$\widehat{m} = \frac{1}{N}\sum_{n=1}^{N} x_n$
Univariate Covariance	$\widehat{\sigma} = \frac{1}{N}\sum_{n=1}^{N}(x_n - \widehat{m})^2$
Multivariate Mean	$\widehat{\boldsymbol{m}} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n$
Covariance Matrix	$\widehat{C} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}_n - \boldsymbol{m})(\boldsymbol{x}_n - \boldsymbol{m})^T$

- These are known as maximum likelihood estimates (see later).

Bayesian Decision Theory

- Classes: $\omega_i,\ i = 1, ..., K$
- Prior Probabilities: $P[\omega_1], ..., P[\omega_K]$;
  $P[\omega_i] \geq 0, \quad \sum_{i=1}^{K} P[\omega_i] = 1$
- Likelihoods (class conditional probabilities): $p(\boldsymbol{x}|\omega_i),\ i = 1, .., K$
- Posterior Probability: $P[\omega_j \,|\, \boldsymbol{x}]$

$$P[\omega_j \,|\, \boldsymbol{x}] = \frac{p(\boldsymbol{x} \,|\, \omega_j)\ P[\omega_j]}{\sum_{i=1}^{K} p(\boldsymbol{x} \,|\, \omega_i)\ P[\omega_i]}$$

- From prior knowledge: $P[\omega_i]$; From traing data: $p(\boldsymbol{x}|\omega_i)$
- Decision rule: Assign $\boldsymbol{x}$ to the class that maximizes posterior probability.
- The denominator is a constant; *i.e.* does not depend on ω_j
- Hence the decision rule becomes:

$$\boldsymbol{x} \in \max_j p(\boldsymbol{x} \,|\, \omega_j)\ P[\omega_j]$$

Bayes' Classifier for Gaussian Densities

Make assumptions, cancel common terms when making comparisons...

- Decision rule from: $p(\boldsymbol{x} \mid \omega_j)\ P[\omega_j]$
- Assume the two classes are Gaussian distributed with distinct means and identical covariance matrices
  $p(\boldsymbol{x} \mid \omega_j) = \mathcal{N}(\boldsymbol{m}_j, \boldsymbol{C})$
- Substitute into Bayes' classifier decision rule

$$\begin{aligned} P[\omega_1|\boldsymbol{x}] &\lessgtr P[\omega_2|\boldsymbol{x}] \\ p(\boldsymbol{x}|\omega_1)\ P[\omega_1] &\lessgtr p(\boldsymbol{x}|\omega_2)\ P[\omega_2] \end{aligned}$$

$$\begin{aligned} &\frac{1}{(2\pi)^{p/2}(\det(C))^{1/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{m}_1)^t \boldsymbol{C}^{-1}(\boldsymbol{x}-\boldsymbol{m}_1)\right\} P[\omega_1] \lessgtr \\ &\frac{1}{(2\pi)^{p/2}(\det(C))^{1/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{m}_2)^t \boldsymbol{C}^{-1}(\boldsymbol{x}-\boldsymbol{m}_2)\right\} P[\omega_2] \end{aligned}$$

Bayes' classifier for simple densities (cont'd)

Distinct Means; Equal, isotropic covariance matrix

- Suppose the densities are isotropic and priors are equal
  i.e. $\boldsymbol{C} = \sigma^2 \boldsymbol{I}$ and $P[\omega_1] = P[\omega_2]$
- The comparison simplifies to (see algebra on board):

$$\begin{aligned} (\boldsymbol{x}-\boldsymbol{m}_1)^t(\boldsymbol{x}-\boldsymbol{m}_1) &\lessgtr (\boldsymbol{x}-\boldsymbol{m}_2)^t(\boldsymbol{x}-\boldsymbol{m}_2) \\ |\boldsymbol{x}-\boldsymbol{m}_1| &\lessgtr |\boldsymbol{x}-\boldsymbol{m}_2| \end{aligned}$$

- The above is a simple *distance to mean* classifier
- Under the above simplistic assumptions, we only need to store one template per class (the means)!

Bayes' classifier for simple densities (cont'd)

Distinct Means; Common covariance matrix (but not isotropic)

- Cancel common terms and take log

$$(\boldsymbol{x}-\boldsymbol{m}_1)^t\,\boldsymbol{C}^{-1}\,(\boldsymbol{x}-\boldsymbol{m}_1) \lessgtr (\boldsymbol{x}-\boldsymbol{m}_2)^t\,\boldsymbol{C}^{-1}\,(\boldsymbol{x}-\boldsymbol{m}_2) - \log\left\{\frac{P[\omega_1]}{P[\omega_2}\right\}$$

- Also simplifies to a linear classifier!

$$\boldsymbol{w}^t\,\boldsymbol{x} + b \lessgtr 0$$

$$\begin{aligned}\boldsymbol{w} &= 2\boldsymbol{C}^{-1}(\boldsymbol{m}_2-\boldsymbol{m}_1)\\ b &= \left(\boldsymbol{m}_1^t\boldsymbol{C}^{-1}\boldsymbol{m}_1 - \boldsymbol{m}_2^t\boldsymbol{C}^{-1}\boldsymbol{m}_2\right) - \log\left\{\frac{P[\omega_1]}{P[\omega_2]}\right\}\end{aligned}$$

- Also a distance to template classifier, where the distance is

$$(\boldsymbol{x}-\boldsymbol{m}_1)^t\,\boldsymbol{C}^{-1}\,(\boldsymbol{x}-\boldsymbol{m}_1)$$

  Known as Mahalanobis distance

Posterior probabilities for simple Gaussian cases

- Bayes classifier:

$$P[\omega_1|\boldsymbol{x}] = \frac{p(\boldsymbol{x}|\omega_1)\,P[\omega_1]}{p(\boldsymbol{x}|\omega_1)\,P[\omega_1] + p(\boldsymbol{x}|\omega_2)\,P[\omega_2]}$$

- Restrictive assumptions:
  - Gaussian $p(\boldsymbol{x}|\omega_j) = \mathcal{N}(\boldsymbol{m}_j, \boldsymbol{C}_j)$
  - Equal covariance matrices: $\boldsymbol{C}_1 = \boldsymbol{C}_2 = C$
- Substitute, divide through by numerator term and cancel common terms to get

$$P[\omega_1|\boldsymbol{x}] = \frac{1}{1+\exp\{-(\boldsymbol{w}^t\boldsymbol{x}+w_0)\}}$$

- The functional form $1/(1+\exp(-\alpha))$ is known as sigmoid / logistic.

Summary:

Bayesian decision theory is fundamental to machine learning; we started from a probabilistic setting in which we described the posterior probability of membership and derived several results starting from this premise:

- Under certain assumptions...
  - The class boundary is linear
  - The posterior probability has a sigmoidal shapedefault
  - The optimal classifier reduces to a distance to template classifier
- ...and under certain other assumptions...
  - The best classifier is still a distance to template classifier, but instead of Euclidean distance we need to use Mahalanobis distance.
- ...and under certain other assumptions...
  - The optimal classifier is a quadratic classifier

Fisher Linear Discriminant Analysis

- In the $p-$ dimensional ($\mathcal{R}^p$) input space, find a direction on which projected data is maximally separable:
  - Projected means should be far apart
  - Projected scatter of each class should be small
- Projection of $\boldsymbol{x}_n$ onto direction $\boldsymbol{w}$ is $\boldsymbol{w}^t\boldsymbol{x}_n$;
  - Projected mean for class j will be at $\boldsymbol{w}^t\boldsymbol{m}_j$
  - Variance of projections is $\boldsymbol{w}^t\boldsymbol{C}_j\boldsymbol{w}$, where $\boldsymbol{C}_j$ is the covariance matrix of data in class j.
- Fisher Ratio:

$$J_F = \frac{(\boldsymbol{w}^t\boldsymbol{m}_1 - \boldsymbol{w}^t\boldsymbol{m}_2)^2}{\boldsymbol{w}^t\boldsymbol{C}_1\boldsymbol{w} + \boldsymbol{w}^t\boldsymbol{C}_2\boldsymbol{w}}$$

- We can write the numerator as $\boldsymbol{w}^t\boldsymbol{C}_B\boldsymbol{w}$, where $\boldsymbol{C}_B = (\boldsymbol{m}_1 - \boldsymbol{m}_2)(\boldsymbol{m}_1 - \boldsymbol{m}_2)^t$, the between-class scatter matrix.
- $\boldsymbol{C}_W = \boldsymbol{C}_1 + \boldsymbol{C}_2$, the within class scatter matrix.

Fisher Linear Discriminant Analysis

- Fisher criterion to maximize

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^t \boldsymbol{C}_B \boldsymbol{w}}{\boldsymbol{w}^t \boldsymbol{C}_W \boldsymbol{w}}$$

- Set gradient to zero

$$\nabla_{\boldsymbol{w}} = \frac{2\boldsymbol{C}_B\boldsymbol{w} \times (\boldsymbol{w}^t \boldsymbol{C}_W \boldsymbol{w}) - 2\boldsymbol{C}_W\boldsymbol{w} \times (\boldsymbol{w}^t \boldsymbol{C}_B \boldsymbol{w})}{(\boldsymbol{w}^t \boldsymbol{C}_W \boldsymbol{w})^2}$$

- Equate this to zero and observe
  - $\boldsymbol{w}^t\boldsymbol{C}_W\boldsymbol{w}$ and $\boldsymbol{w}^t\boldsymbol{C}_B\boldsymbol{w}$ are scalars
  - $\boldsymbol{C}_B\boldsymbol{w}$ points in the same direction as $\boldsymbol{m}_1 - \boldsymbol{m}_2$
  - We are only interested in the direction of $\boldsymbol{w}$

$$\boldsymbol{w}_F = \alpha\boldsymbol{C}_w^{-1}(\boldsymbol{m}_1 - \boldsymbol{m}_2)$$

Linear Regression & Perceptron

Formulation → derivation → solution

- Data: $\{\boldsymbol{x}_n,\ f_n\}_{n=1}^N$
  Input: $\boldsymbol{x}_n \in \mathcal{R}^p$; target / output f_n real valued
- Model: $f = \boldsymbol{w}^t\boldsymbol{x} + w_0$
  Output linear function of input (including a constant w_0)
- Work in $(p+1)$ dimensional space to avoid treating w_0 separately

$$\boldsymbol{y} = \begin{pmatrix} \boldsymbol{x} \\ 1 \end{pmatrix} \quad \boldsymbol{a} = \begin{pmatrix} \boldsymbol{w} \\ w_0 \end{pmatrix}$$

- Data: $\{\boldsymbol{y}_n,\ f_n\}_{n=1}^N$
- Model: $f = \boldsymbol{y}^t\boldsymbol{a}$
- $p+1$ unknowns held in vector $\boldsymbol{a}$

Error and Minimization

- $E = \sum_{n=1}^{N} \{\boldsymbol{y}_n^t \boldsymbol{a} - f_n\}^2$
- $E = \sum_{n=1}^{N} \left\{ \left(\sum_{j=1}^{(p+1)} a_j \, y_{nj} \right) - f_n \right\}^2$
- To find the best $\boldsymbol{a}$ we minimize E – differentiate with respect to each of the unknowns in $\boldsymbol{a}$ and set to zero.
- $$\frac{\partial E}{\partial a_i} = 2 \sum_{n=1}^{N} \left\{ \left(\sum_{j=1}^{(p+1)} a_j \, y_{nj} \right) - f_n \right\} (y_{ni})$$
- There are $(p+1)$ derivatives (with respect to each a_i)
- Equating them to zero gives $(p+1)$ equations in $(p+1)$ unknowns

Solution to Regression

- $(p+1)$ simultaneous equations to solve:
  i^{th} row, j^{th} column shown

$$\begin{pmatrix} & \cdots & \\ \cdots & \cdots & \cdots \\ \vdots & \sum_{n=1}^{N} y_{ni} y_{nj} & \cdots \\ \vdots & \cdots & \vdots \\ \cdots & \cdots & \cdots \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{(p+1)} \end{pmatrix} = \begin{pmatrix} \vdots \\ \sum_{n=1}^{N} f_n y_{ni} \\ \vdots \end{pmatrix}$$

Derivation in vector/matrix form

- $\boldsymbol{Y}$: $N \times (p+1)$ matrix n^{th} row is $\boldsymbol{y}_n^t$
- $\boldsymbol{f}$: $N \times 1$ vector of outputs
- Error $E = ||\boldsymbol{Y}\,\boldsymbol{a} - \boldsymbol{f}||^2$
- Gradient

$$\nabla_{\boldsymbol{a}} E = 2\boldsymbol{Y}^t(\boldsymbol{Y}\boldsymbol{a} - \boldsymbol{f})$$

- Equating the gradient to zero gives

$$\begin{aligned} \boldsymbol{Y}^t\boldsymbol{Y}\boldsymbol{a} &= \boldsymbol{Y}^t\boldsymbol{f} \\ \boldsymbol{a} &= (\boldsymbol{Y}^t\boldsymbol{Y})^{-1}\,\boldsymbol{Y}^t\,\boldsymbol{f} \end{aligned}$$

Solution by Gradient Descent

- Gradient vector: $\nabla_{\boldsymbol{a}} E = 2\boldsymbol{Y}^t(\boldsymbol{Y}\boldsymbol{a} - \boldsymbol{f})$
- Steepest descent algorithm:

```
Initialize a at random
Update a^(k+1) = a^(k) − η ∇_a E
Until Convergence
```

- Second order (Newton's) method

```
Initialize a at random
Update a^(k+1) = a^(k) − η H^{-1} ∇_a E
Until Convergence
```

- Rapid convergence with second order method, but cost of computing and inverting $\boldsymbol{H}$ can be high (more on this under Neural Networks)

Gradient and Stochastic Gradient Descent

- Error $E = \sum_{n=1}^{N} e_n^2$
- True gradient:

$$\nabla_{\boldsymbol{a}} E = 2 \sum_{n=1}^{N} \left\{ \mathbf{y_n^t a} - \mathbf{f_n} \right\} (\mathbf{y_n})$$

- Gradient computed on n^{th} data:

$$\nabla_{\boldsymbol{a}} e_n = 2 \left\{ \mathbf{y_n^t a} - \mathbf{f_n} \right\} (\mathbf{y_n})$$

Regularization

- Pseudo inverse solution: $\boldsymbol{a} = (\boldsymbol{Y}^t \boldsymbol{Y})^{-1} \boldsymbol{Y}^t \boldsymbol{f}$
- This can be ill conditioned, so we could *regularize* by

$$\boldsymbol{a} = \left(\boldsymbol{Y}^t \boldsymbol{Y} + \gamma \boldsymbol{I} \right)^{-1} \boldsymbol{Y}^t \boldsymbol{f}$$

  where γ is a small constant.
- We achieve precisely this by minimizing an error of the form

$$||\boldsymbol{Ya} - \boldsymbol{f}||^2 + \gamma ||\boldsymbol{a}||^2$$

  Here a quadratic penalty term has been included
- Homework: Differentiate this error and derive the regularized solution
- Sparse solutions are obtained by regularizing with an l_1 norm (sum of absolute values of $\boldsymbol{a}$, i.e. $\sum_{j=1}^{p} |a_j|$): Known as *Lasso*

Perceptron

- Number of misclassified examples as measure of error
  Piecewise constant (cannot differentiate)
- Suitable error measure:

$$E_P = -\sum \boldsymbol{y}_n^t \boldsymbol{a}$$

  - Summation taken over misclassified examples
  - We started with $\boldsymbol{y}_n^t \boldsymbol{a} > 0$ for positive class and $\boldsymbol{y}_n^t \boldsymbol{a} < 0$ for the negative class; we then switch the signs of negative class examples and required $\boldsymbol{y}_n^t \boldsymbol{a} > 0$ for all the training data; so for the misclassified examples $-\sum \boldsymbol{y}_n^t \boldsymbol{a}$ should be as small as possible.

Perceptron

Learning rule

- Gradient:

$$\frac{\partial E}{\partial \boldsymbol{a}} = -\sum \boldsymbol{y}_n$$

- Gradient algorithm: $\boldsymbol{a}^{(k+1)} = a^{(k)} + \sum \boldsymbol{y}_n$
- Stochastic gradient algorithm:

$$\boldsymbol{a}^{(k+1)} = a^{(k)} + \boldsymbol{y}_n$$

- Note what $\boldsymbol{y}_n$ is. It is an item of data that is taken at random and happens to be misclassified by the current value of $\boldsymbol{a}$ at iteration k.

Perceptron

Convergence of the learning rule

- Learning Rule: $\boldsymbol{a}^{(k+1)} = \boldsymbol{a}^{(k)} + \boldsymbol{y}(k)$
  where $\boldsymbol{y}(k)$ is a misclassified input.
- Training criterion
  - We start with requiring $\boldsymbol{a}^t\,\boldsymbol{y}(k) \leqslant 0$, depending on the example belonging to class 1 or class 2.
  - If we switch the signs of examples of class 2, we require $\boldsymbol{a}^t\,\boldsymbol{y}(k) > 0$ for all k.
- On misclassified data $\boldsymbol{a}^t\,\boldsymbol{y}(k) < 0$
- If $\widehat{\boldsymbol{a}}$ is a solution (separable data), for all k, $\widehat{\boldsymbol{a}}\,\boldsymbol{y}(k) > 0$
- We prove convergence by showing:
  $||\boldsymbol{a}^{(k+1)} - \widehat{\boldsymbol{a}}||^2 < ||\boldsymbol{a}^{(k)} - \widehat{\boldsymbol{a}}||^2$ for this update rule. *i.e.* the learning rule brings the guess closer to a valid solution.

Perceptron

Convergence of the learning rule (cont'd)

- For perceptron criterion, the magnitude of $\boldsymbol{a}$ is not relevant (only the direction is). Hence for some scalar α, we wish to show

$$||\boldsymbol{a}^{(k+1)} - \alpha\,\widehat{\boldsymbol{a}}||^2 < ||\boldsymbol{a}^{(k)} - \alpha\,\widehat{\boldsymbol{a}}||^2$$

- From the update formula

$$\boldsymbol{a}^{(k+1)} - \alpha\,\widehat{\boldsymbol{a}} = \boldsymbol{a}^{(k)} - \alpha\,\widehat{\boldsymbol{a}} + \boldsymbol{y}(k)$$

- Taking magnitudes

$$||\boldsymbol{a}^{(k+1)} - \alpha\widehat{\boldsymbol{a}}||^2 = ||\boldsymbol{a}^{(k)} - \alpha\widehat{\boldsymbol{a}}||^2 + 2(\boldsymbol{a}^{(k)} - \alpha\widehat{\boldsymbol{a}})^t\boldsymbol{y}(k) + ||\boldsymbol{y}(k)||^2$$

- If we drop the negative term ${\boldsymbol{a}^{(k)}}^t\boldsymbol{y}(k)$ from RHS, the equality becomes an inequality

$$||\boldsymbol{a}^{(k+1)} - \alpha\widehat{\boldsymbol{a}}||^2 < ||\boldsymbol{a}^{(k)} - \alpha\widehat{\boldsymbol{a}}||^2 - 2\alpha\widehat{\boldsymbol{a}}^t\boldsymbol{y}(k) + ||\boldsymbol{y}(k)||^2$$

Perceptron

Convergence of the learning rule (cont'd)

- Of the three terms on the right hand side, we know $\widehat{\boldsymbol{a}}^t \boldsymbol{y}(k) > 0$, because $\widehat{\boldsymbol{a}}$ is assumed to be a solution.
- If we select

$$\beta^2 = \max_i ||\boldsymbol{y}_i||^2 \quad \gamma = \min_i \widehat{\boldsymbol{a}}^t \boldsymbol{y}_i$$

i.e. largest of the positive term and smallest of the negative term, then for $\alpha = \beta^2/\gamma$,

$$||\boldsymbol{a}^{(k+1)} - \alpha\widehat{\boldsymbol{a}}||^2 < ||\boldsymbol{a}^{(k)} - \alpha\widehat{\boldsymbol{a}}||^2 - \beta^2$$

- (Note the inequality remains true when the right hand side is replaced by a quantity larger than what it previously was.)
- Every correction takes the guess closer to a true solution.
- From an initialization $\boldsymbol{a}^{(1)}$, we will find a solution in *at most* $k_0 = \frac{||\boldsymbol{a}(1) - \alpha\widehat{\boldsymbol{a}}||^2}{\beta^2}$ updates.

Estimation

Maximum likelihood on simple examples

- We have data $\boldsymbol{x}_k$
- We have a model: *e.g.* the data came from a Gaussian density
- We have parameters relating to the model: *e.g.* mean of the Gaussian
- Our task is to estimate the parameters given the data
- Frequentist thought
  - The given data is a particular realization of the underlying system
  - Repeated experiments will give different estimates
  - If each experiment uses a lot of data, the variation may be small
  - We define a probabilistic model and maximize likelihood
  - Bias and Variance in estimation
- Bayesian thought
  - We are interested in the uncertainty in parameters
  - We have a prior uncertainty
  - There is some information in the data
  - We combine these to get a posterior uncertainty

Likelihood & Log likelihood

- $p(\boldsymbol{x} \mid \omega_j)$
- Parametric form $p(\boldsymbol{x} \mid \omega_j, \boldsymbol{\theta}_j)$
  For example
$$p(\boldsymbol{x} \mid \omega_j, \boldsymbol{\theta}_j) = \mathcal{N}(\boldsymbol{m}_j, \boldsymbol{C}_j)$$

- Dataset $\mathcal{D} = \boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n$ of identical and independently distributed samples (iid)
  - All samples were drawn from this distribution
  - Independent draws (previous value does not affect the next draw)
- Likelihood of an item of data (function of the parameter!)
$$p(\boldsymbol{x}_k|\boldsymbol{\theta})$$
- Likelihood of the set of data
$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{i=1}^{n} p(\boldsymbol{x}_k|\boldsymbol{\theta})$$
- Log likelihood
$$l(\boldsymbol{\theta}) = \ln p(\mathcal{D}|\boldsymbol{\theta})$$

Maximum Likelihood

- Maximum likelihood
$$\widehat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} l(\boldsymbol{\theta})$$
- Maximize by taking derivative
$$\boldsymbol{\nabla}_{\boldsymbol{\theta}} = \begin{bmatrix} \frac{\partial}{\partial \theta_1} \\ \vdots \\ \frac{\partial}{\partial \theta_n} \end{bmatrix}$$
$$\boldsymbol{\nabla}_{\boldsymbol{\theta}} l = \sum_{k=1}^{n} \boldsymbol{\nabla}_{\boldsymbol{\theta}} p(\mathcal{D}|\boldsymbol{\theta})$$

... and equating to zero
$$\boldsymbol{\nabla}_{\boldsymbol{\theta}} l = \mathbf{0}.$$

... and solve for the unknown parameter values.

Example: Multivariate Gaussian $\mathcal{N}(\boldsymbol{m}, C)$

Mean unknown, (Covariance known)

- ... product of Gaussians; taking log removes exp and turns $\prod$ into $\sum$
- ... write it out for a single data point

$$\ln\, p\left(\boldsymbol{x}_k|\boldsymbol{m}\right) \;=\; \frac{1}{2}\ln(2\pi)^d \det C \;-\; \frac{1}{2}\left(\boldsymbol{x}_k-\boldsymbol{m}\right)^T\; C^{-1}\; \left(\boldsymbol{x}_k-\boldsymbol{m}\right)$$

- ... the derivative

$$\boldsymbol{\nabla}_{\boldsymbol{m}} \ln p\left(\boldsymbol{x}_k|\boldsymbol{m}\right) \;=\; C^{-1}\left(\boldsymbol{x}_k-\boldsymbol{m}\right)$$

- Given n data $\boldsymbol{x}_1, ..., \boldsymbol{x}_n$

$$\sum_{k=1}^{n} C^{-1}\left(\boldsymbol{x}_k-\widehat{\boldsymbol{m}}\right) \;=\; \mathbf{0}$$

- and the solution is...

$$\widehat{\boldsymbol{m}} \;=\; \frac{1}{n}\sum_{k=1}^{n} \boldsymbol{x}_k$$

Example: Univariate Gaussian $\mathcal{N}(m, \sigma^2)$

(Mean and variance unknown)

- Two parameters: $\theta_1 = m$ and $\theta_2 = \sigma^2$
- Log likelihood of a single data:

$$\ln p\left(x_k|\boldsymbol{\theta}\right) \;=\; \frac{1}{2}\ln 2\pi\theta_2 \;-\; \frac{1}{2\theta_2}(x_k-\theta_1)^2$$

- Derivative of the log likelihood

$$\boldsymbol{\nabla}_{\boldsymbol{\theta}} p\left(x_k|\boldsymbol{\theta}\right) \;=\; \begin{bmatrix} \frac{1}{\theta_2}\left(x_k-\theta_1\right)^2 \\ -\frac{1}{2\theta_2} \;+\; \frac{(x_k-\theta_1)^2}{2\theta_2^2} \end{bmatrix}$$

- Given n data $x_1, x_2, ..., x_n$, and considering the full log likelihood

$$\begin{aligned} \sum_{k=1}^{n} \frac{1}{\widehat{\theta}_2}\left(x_k-\widehat{\theta}_1\right) \;&=\; 0 \\ -\sum_{k=1}^{n} \frac{1}{\widehat{\theta}_2} + \sum_{k=1}^{n} \frac{1}{\widehat{\theta}_2^2}\left(x_k-\widehat{\theta}_1^2\right)^2 \;&=\; 0 \end{aligned}$$

Example

Univariate Gaussian, unknown mean and variance (cont'd)

- ... after some algebra

$$\widehat{m} = \frac{1}{n}\sum_{k=1}^{n} x_k$$

$$\widehat{\sigma}^2 = \frac{1}{n}\sum_{k=1}^{n}(x_k - \widehat{m})^2$$

- If we did this estimation from several datasets...

$$E\left[\frac{1}{n}\sum_{k=1}^{n}(x_k - \bar{x})^2\right] = \frac{n-1}{n}\sigma^2 \neq \sigma^2$$

  expected value of estimate is not the same as the true value!

For the multivariate Gaussian, we have seen the results:

$$\widehat{\boldsymbol{m}} =$$

$$\widehat{C} =$$

Bayesian Estimation

- Data: $\mathcal{D}$: $x_1, ...x_n$
- Likelihood (as seen before) $p(x|m) \sim \mathcal{N}(m, \sigma^2)$
- Prior uncertainty over parameters: $p(m) \sim \mathcal{N}(m_0, \sigma_0^2)$
  m_0 and σ_0^2 are known.
- Posterior via Bayes' formula

$$p(m|\mathcal{D}) = \frac{p(\mathcal{D}|m)p(m)}{\int p(\mathcal{D}|m)\, p(m)\, dm}$$

  Denominator is a constant, so we deal with

$$p(m|\mathcal{D}) = \alpha \prod_{k=1}^{n} p(x_k|m)\, p(m)$$

- Two ways forward from here
  - Maximum *a posteriori* estimation
  - Inference by integrating out parameters

Bayesian Estimation: Univariate Gaussian

(Only the mean is unknown)

- Data: $\mathcal{D}$: $x_1, ...x_n$
- Likelihood (as seen before) $p(x|m) \sim \mathcal{N}(m, \sigma^2)$
- Prior uncertainty over parameters: $p(m) \sim \mathcal{N}(m_0, \sigma_0^2)$
  m_0 and σ_0^2 are known.
- Substituting gives the posterior as a product of Gaussians

$$p(m|\mathcal{D}) = \alpha \prod_{k=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x_k - m}{\sigma}\right)^2\right] \times \frac{1}{\sigma_0\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{m - m_0}{\sigma_0}\right)^2\right]$$

- Which can be reduced to...

$$p(m|\mathcal{D}) = \alpha_2 \exp\left\{-\frac{1}{2}\left\{\left(\frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}\right) m^2 - 2\left(\frac{1}{\sigma^2}\sum_{k=1}^{n} x_k + \frac{m_0}{\sigma_0^2}\right) m\right\}\right\}$$

- But then...

$$p(m|\mathcal{D}) = \frac{1}{\sigma_n} \exp\left\{-\frac{1}{2}\left(\frac{m - m_n}{\sigma_n}\right)^2\right\}$$

- Matching terms...

$$\frac{1}{\sigma_n^2} = \frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}$$

$$\frac{m_n}{\sigma_n^2} = \frac{n}{\sigma^2} \quad \frac{m_0}{\sigma_0^2} \quad \frac{1}{n}\sum_{k=1}^{n}$$

Bayesian Estimation: Univariate Gaussian (cont'd)

- Finally...

$$m_n = \left(\frac{n\sigma_0^2}{n\sigma_0^2 + \sigma^2}\right) \widehat{m}_n + \left(\frac{\sigma^2}{n\sigma_0^2 + \sigma^2}\right) m_0$$

$$\sigma_n^2 = \frac{\sigma_0^2 \sigma^2}{n\sigma_0^2 + \sigma^2}$$

- We now have an estimate that combines *prior* information about the parameter ($p(m) = m_0$) with data ($x_1, ..., x_k$) to quantify uncertainty about the parameter:
  - Before seeing any data, we have a belief
  - As we see more and more data, our belief is taken over by what the data tells us.

Unsupervised Learning

- Given: $\{\boldsymbol{x}_n\}_{n=1}^N$ (as opposed to $\{\boldsymbol{x}_n, f_n\}_{n=1}^N$)
- We might extract cluster structures
  - Notion of distance between points of data
  - Criterion to determine how many clusters (often from prior knowledge)
  - Underlying probabilistic model
- We might project data onto a subspace $\boldsymbol{x}_n \in \mathcal{R}^d \longrightarrow \boldsymbol{y}_n \in \mathcal{R}^q$
  - $q = 2$ helps visualization
  - Subspace representation useful for
    - Data compression
    - Sometimes used to reduce features

Semi Supervised Learning:

$$\{\boldsymbol{x}_n, f_n\}_{n=1}^{N_1} \text{ and } \{\boldsymbol{x}_n\}_{n=N_1+1}^{N_2}$$

Principal Component Analysis:

- N data $\boldsymbol{x}_n \in \mathcal{R}^d$ distributed with mean $\boldsymbol{m}$ and covariance matrix $\boldsymbol{C}$.
- Project onto direction $\boldsymbol{u}$; find the direction that maximizes projected variance.
- Projected variance is $\boldsymbol{u}^t \boldsymbol{C} \boldsymbol{u}$
- We are only interested in the direction; not in increasing the projected variance by choosing $\boldsymbol{u}$ with large magnitude.
- Set up a constrained optimization problem

$$\max_{\boldsymbol{u}} \boldsymbol{u}^t \boldsymbol{C} \boldsymbol{u} \quad \text{subject to } \boldsymbol{u}^t \boldsymbol{u} = 1$$

- Lagrangian

$$\mathcal{L} = \boldsymbol{u}^t \boldsymbol{C} \boldsymbol{u} - \lambda \left[\boldsymbol{u}^t \boldsymbol{u} - 1 \right]$$

- $\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}} = 0 \implies \boldsymbol{C}\boldsymbol{u} = \lambda \boldsymbol{u}$; *i.e.* principal directions are eigenvectors of covariance

Mixture Model

We write a mixture of Gaussian densities:

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

- If the mixing proportions π_k satisfy
  - $\pi_k \geq 0$
  - $\sum_{k=1}^{K} \pi_k = 1$

  $p(\boldsymbol{x})$ is a proper probability density.

- More powerful model – useful when data is multi-modal
- Parameters are: proportions, means and covariance matrices
- Parameter estimation $(\boldsymbol{\pi}_k, \boldsymbol{\mu}_k \boldsymbol{\Sigma}_k)$ is not easy.
- z_{nk}: association of n^{th} data to k^{th} mode unknown (latent)

Algorithm:

$$\begin{aligned}
\pi_k &= \frac{1}{N} \sum_{n=1}^{N} q_{nk} \\
\boldsymbol{\mu}_k &= \frac{\sum_{n=1}^{N} q_{nk} \boldsymbol{x}_n}{\sum_{n=1}^{N} q_{nk}} \\
\boldsymbol{\Sigma}_k &= \frac{\sum_{n=1}^{N} q_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^t}{\sum_{n=1}^{N} q_{nk}}
\end{aligned}$$

$$q_{nk} = \frac{\pi_k p(\boldsymbol{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j p(\boldsymbol{x}_n | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

Expectation Maximization

Auxilliary Variable as Posteriors

Interpret:

- Mixture model as a Gaussian classifier with K classes
- π_k as prior probabilities
- Each of the $\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ as class conditional densities / likelihoods.

$$\begin{aligned} p(z_{nk} = 1 | \boldsymbol{x}_n, \boldsymbol{\pi}, \Delta) &= \frac{p(z_{nk} = 1|\pi_k) p(\boldsymbol{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} p(z_{nk} = 1|\pi_k) p(\boldsymbol{x}_n | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \\ &= q_{nk} \end{aligned}$$

- Each data item has a weighted contribution to the estimation of parameters.
- Unknown assignment z_{nk}; **E**xpected value of this unknown assignment is q_{nk}, the posterior probability
- **M**aximize (the lower bound) to re-estimate parameters.

K-Means Clustering Algorithm

Input: $\boldsymbol{X} = \{\boldsymbol{x}_n^t\}_{n=1}^N$, K
Output: $\boldsymbol{C}$, `Idx`

`initialize:` $\boldsymbol{C} = \left\{\boldsymbol{c}_j^t\right\}_{j=1}^K$

`repeat`

. assign n^{th} sample to nearest $\boldsymbol{c}_j$
. `Idx(n)` $= \min_j \; ||\boldsymbol{x}_n - \boldsymbol{c}_j||^2$

. recompute $\boldsymbol{c}_j = \frac{1}{N_j} \sum_{n=j} \boldsymbol{x}_n$

`until` no change in $\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots \boldsymbol{c}_k$

K-Means as Mixture Gaussian

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_j \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

- Set $\boldsymbol{\Sigma}_k = \sigma_k^2 \boldsymbol{I}$
- At every iteration, set largest q_{nk} (largest over k) to one and others to zero. Winner take all at each datapoint.
- Computation of q_{nk} is expectation of latent variable z_{nk} – **E** step
- Re-estimation of $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ become maximum likelihood estimates from data assigned to each cluster (because q_{nk} is either one or zero) – **M** step

Radial Basis Functions

- We fix nonlinear part in a data-dependent way
- Estimate only a linear part (by linear regression)

The Model:

$$g(\mathbf{x}) = \sum_{j=1}^{M} \lambda_j \, \phi\left(|\boldsymbol{x} - \boldsymbol{m}_j| / \sigma\right)$$

- We think of *centers* $\boldsymbol{m}_j$ in the input space
- *'Radial'* comes from distances to these centers (scaled by σ)
- We refer to $\phi(.)$ as basis functions
- Note similarity to Fourier transform – signal expressed as sum of sine waves (basis functions)
- $\boldsymbol{m}_j$, $j = 1, .., M$, σ and the functional form of $\phi(.)$ are fixed by some sensible (*ad hoc*) way
- λ_j, $j = 1, .., M$ are the unknowns

Radial Basis Functions (cont'd)

- Data is $\{\boldsymbol{x}_n,\ f_n\}_{n=1}^{N}$
- Similar to linear regression, we will hope to achieve

$$\begin{aligned} f_n &= g(\boldsymbol{x}_n) \\ &= \sum_{j=1}^{M} \lambda_j \phi(|\boldsymbol{x}_n - \boldsymbol{m}_j| / \sigma) \end{aligned}$$

- *i.e.* at each input $\boldsymbol{x}_n$, the function should output our target f_n
- Usually, $M < N$ *i.e.* we have fewer basis functions than there are items of data

Radial basis functions (cont'd)

What basis functions?

- Linear: $\phi(\alpha) = \alpha$
- Gaussian: $\phi(\alpha) = \exp(-\alpha^2/\sigma^2)$
- Multi-quadric: $\phi(\alpha) = \sqrt{1+\alpha^2}$
- Thin plate splines: $\phi(\alpha) = \alpha^2 \log(\alpha)$
- Gaussian very popular in practice
  - Local basis functions
  - $\boldsymbol{m}_j$ obtained by clustering

Radial basis functions (cont'd): Estimating λ_j

- $N \times M$ matrix Y

$$\begin{bmatrix} \phi(|\boldsymbol{x}_1 - \boldsymbol{m}_1|/\sigma) & \phi(|\boldsymbol{x}_1 - \boldsymbol{m}_2|/\sigma) & \dots & \phi(|\boldsymbol{x}_1 - \boldsymbol{m}_M|/\sigma) \\ \phi(|\boldsymbol{x}_2 - \boldsymbol{m}_1|/\sigma) & \phi(|\boldsymbol{x}_2 - \boldsymbol{m}_2|/\sigma) & \dots & \phi(|\boldsymbol{x}_2 - \boldsymbol{m}_M|/\sigma) \\ \vdots & \vdots & \dots & \vdots \\ \phi(|\boldsymbol{x}_N - \boldsymbol{m}_1|/\sigma) & \phi(|\boldsymbol{x}_N - \boldsymbol{m}_2|/\sigma) & \dots & \phi(|\boldsymbol{x}_N - \boldsymbol{m}_M|/\sigma) \end{bmatrix}$$

- Targets $N \times 1$ vector

$$\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{bmatrix}$$

- Minimize error

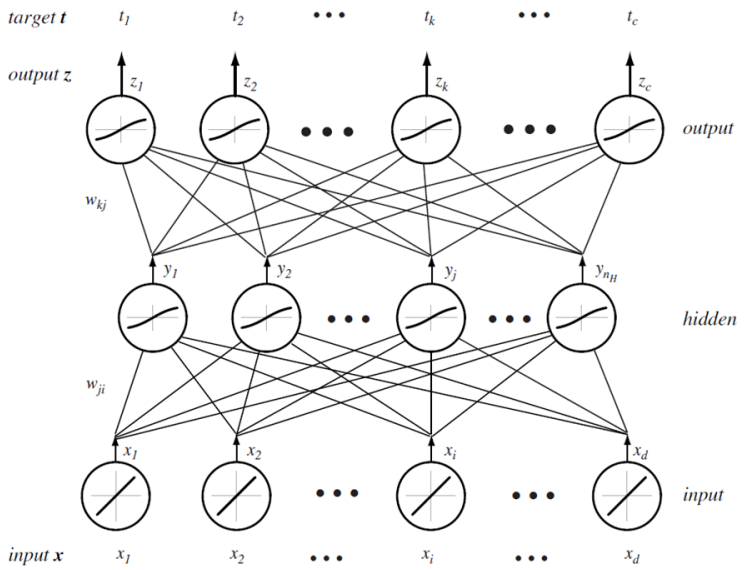
$$\widehat{\boldsymbol{\lambda}} = \min_{\lambda} |Y\,\boldsymbol{\lambda} - \boldsymbol{f}|^2$$

- Solution similar to linear regression; e.g. pseudo inverse

$$\widehat{\boldsymbol{\lambda}} = (Y^T\,Y)^{-1}\,Y^T\,\boldsymbol{f}$$

- Clustering and other rules to set $\boldsymbol{m}_j$ and σ

Multi-layer perceptron



$$g_k(\boldsymbol{x}) = f\left(\sum_{j=1}^{n_H} w_{jk}\; f\left(\sum_{i=1}^{d} w_{ji}\,x_i + w_{j0}\right) + w_{k0}\right)$$

Each unit is a logistic

- Weighted sum of inputs – net activation

$$\text{net}_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0}$$

- Squashed by a nonlinearity

$$y_j = f\left(\text{net}_j\right)$$

- Simple threshold

$$f(\text{net}) = \begin{cases} 1 & \text{if net} \geq 0 \\ -1 & \text{if net} < 0 \end{cases}$$

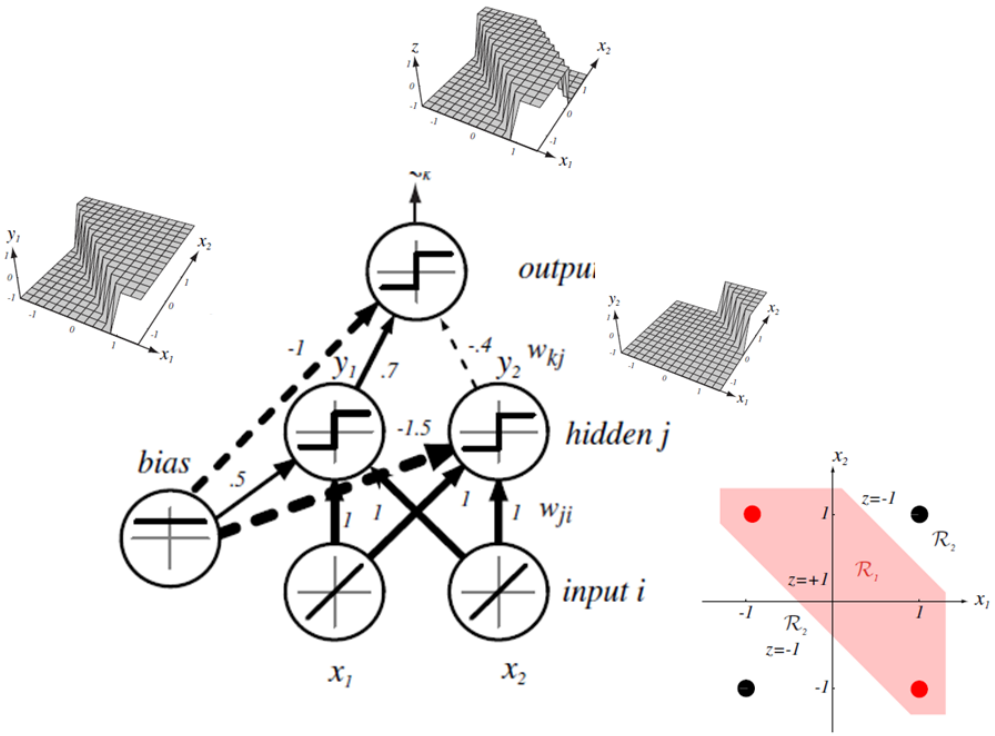
- Sigmoid (logistic)

$$f(\text{net}) = \frac{1}{1 + \exp(-\text{net})}$$

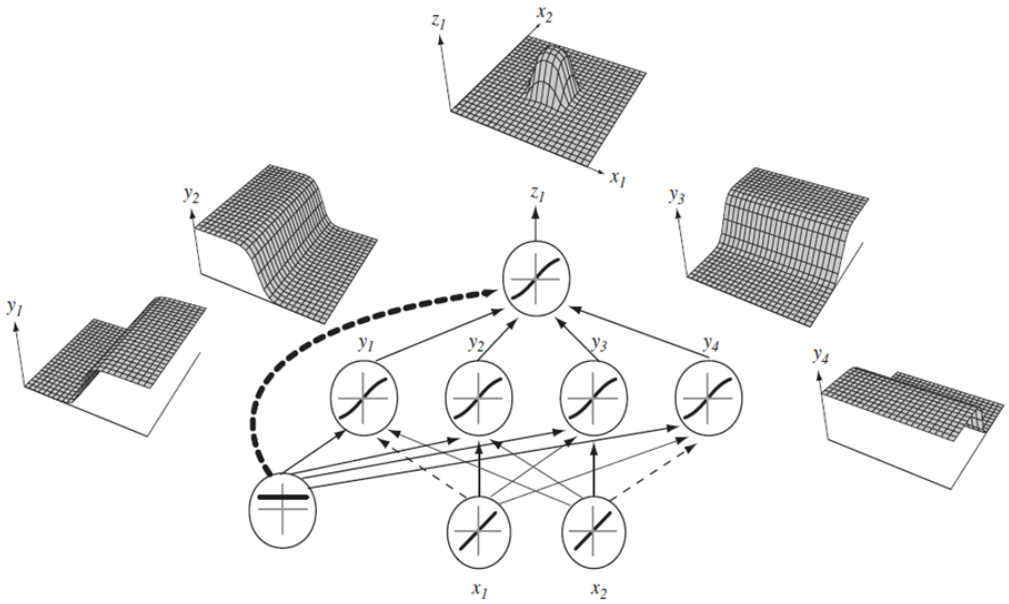
- Hyperbolic tangent

$$\begin{aligned} f(\text{net}) &= \tanh(b\,\text{net}) \\ &= \left[\frac{\exp(b\,\text{net}) - \exp(-b\,\text{net})}{\exp(b\,\text{net}) + \exp(-b\,\text{net})}\right] \end{aligned}$$

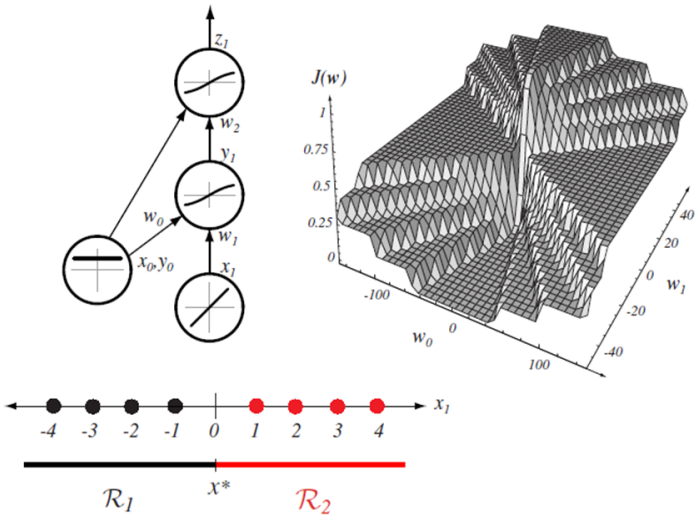
XOR Using a MLP



MLP Constructing Complex Functions



Error Function for MLP



- Not smooth and quadratic like for the linear model
- Large variation in gradient
- Local minima
- Gradient descent training with clever tricks

Training Multi-Layer Neural Networks

- Error

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}||\boldsymbol{t} - \boldsymbol{z}||^2$$

- Change in weight for gradient descent

$$\Delta \boldsymbol{w} = -\eta \frac{\partial J}{\partial \boldsymbol{w}}$$

  Or in component form

$$\Delta w_{pq} = -\eta \frac{\partial J}{\partial w_{pq}}$$

- Gradient descent update

$$\boldsymbol{w}(m+1) = \boldsymbol{w}(m) + \Delta \boldsymbol{w}(m)$$

Error Back Propagation Algorithm

- Hidden to output layer weights

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial \mathrm{net}_k}\frac{\partial \mathrm{net}_k}{\partial w_{kj}} = -\delta_k \frac{\partial \mathrm{net}_k}{\partial w_{kj}}$$

  where, change in error with respect to the activation of the unit,

$$\delta_k = -\frac{\partial J}{\partial \mathrm{net}_k}$$

  - Easy form for δ_k

$$\delta_k = -\frac{\partial J}{\partial \mathrm{net}_k} = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial \mathrm{net}_k} = (t_k - z_k)\, f'(\mathrm{net}_k)$$

    Error at output $\times$ slope of nonlinearity
  - Also with respect to weights net is differentiated easily

$$\frac{\partial \mathrm{net}_k}{\partial w_{kj}} = y_j$$

Error back propagation (cont'd)

- Units internal to the network have no explicit error signal
- Chain rule of differentiation helps!

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial w_{ji}}$$

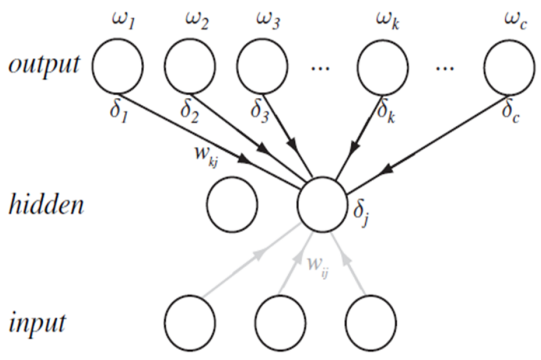
$$\begin{aligned}
\frac{\partial J}{\partial y_j} &= \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right] \\
&= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial \text{net}_k}\frac{\partial \text{net}_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}(t_k - z_k)\, f'(\text{net}_k)\, w_{kj}
\end{aligned}$$

Error propagation (cont'd)

$$\delta_j = f'(\text{net}_j)\sum_{k=1}^{c} w_{kj}\,\delta_k$$

and the update rule

$$\delta w_{ji} = \eta\, x_i\, \delta_j = \eta\left[\sum_{k=1}^{c} w_{kj}\delta_k\right] f'(\text{net}_j)\, x_i$$



- Signal y_j propagates forward
- δ_j's propagate backwards

- Having gradient enables us to update weights
- Can sum over all data and update with true gradient
- Or use sample-by-sample stochastic update
- Gradient descent can be slow (large flat regions)
- Can get stuck in local minima

Speeding-up Training: Newton's Method

Taylor expansion

$$J(\boldsymbol{w}+\Delta\boldsymbol{w}) = J(\boldsymbol{w}) + \left(\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}}\right)^T \Delta\boldsymbol{w} + \frac{1}{2}\Delta\boldsymbol{w}^T H \Delta w + \ldots$$

- We can consider the change in objective function

$$\Delta J(\boldsymbol{w}) = J(\boldsymbol{w}+\Delta\boldsymbol{w}) - J(\boldsymbol{w})$$

and ask for what $\Delta\boldsymbol{w}$ is this minimized?

$$\left(\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}}\right) + H\Delta\boldsymbol{w} = 0$$

$$\Delta\boldsymbol{w} = -H^{-1}\left(\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}}\right)$$

... giving us the update

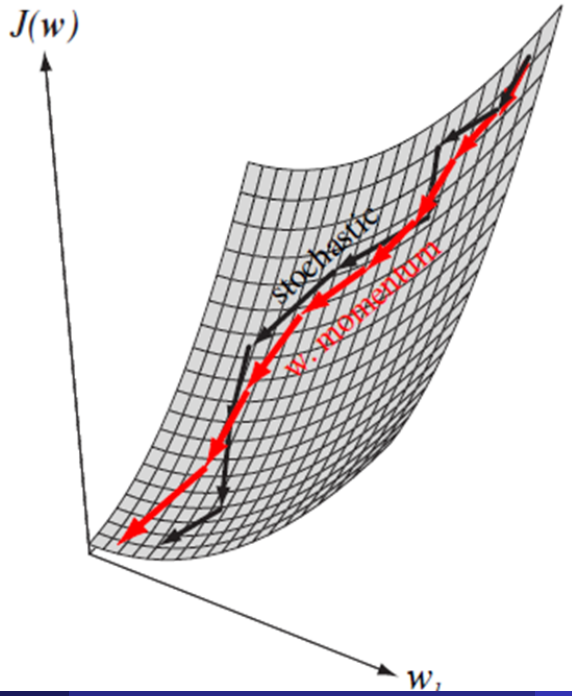
$$\begin{aligned}\boldsymbol{w}(m+1) &= \boldsymbol{w}(m) + \Delta\boldsymbol{w} \\ &= \boldsymbol{w}(m) - H^{-1}(m)\left(\frac{\partial J(\boldsymbol{w}(m))}{\partial \boldsymbol{w}}\right)\end{aligned}$$

- However
  - N weights $\rightarrow N^2$ storage for Hessian
  - Matrix inversion $\mathcal{O}(N^3)$ complexity

Momentum

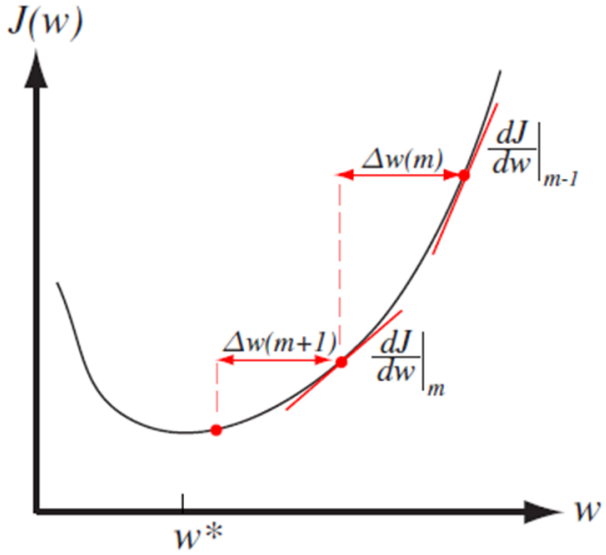
- Slow learning in regions in which gradient is small

$$\boldsymbol{w}(m+1) = \boldsymbol{w}(m) + (1-\alpha)\,\Delta\boldsymbol{w}_{\mathrm{BP}}(m) + \alpha\,\Delta\boldsymbol{w}(m-1)$$



QuickProp

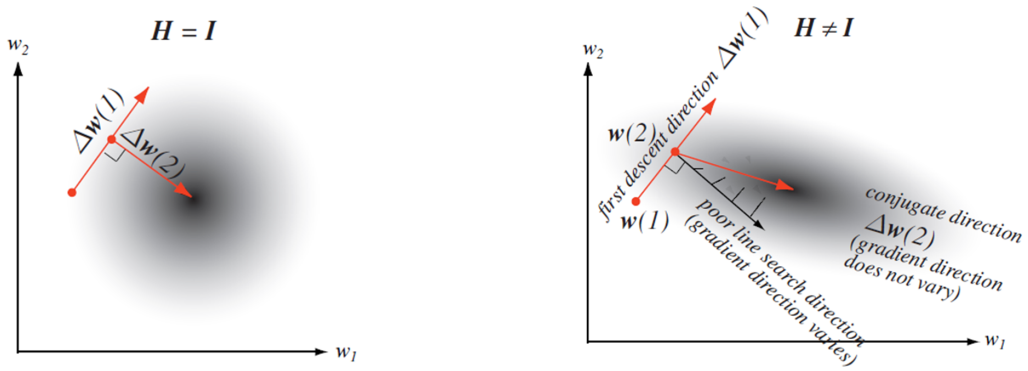
Two successive evaluations to approximate local curvature



$$\Delta w(m+1) = \frac{\partial J / \partial w\,|_m}{\partial J / \partial w\,|_{m-1} - \partial J / \partial w\,|_m}\,\Delta w(m)$$

Conjugate Gradients

- Sequence of line searches (not just gradient update)



Conjugate Gradients (cont'd)

- One dimensional line searches in multiple dimensions
- Careful choice of successive directions to search, informed by local curvature
- $\Delta \boldsymbol{w}(m-1)$ direction of line search at step $m-1$
- For new direction to search, find $\Delta \boldsymbol{w}$

$$\Delta \boldsymbol{w}^T(m-1)\, H\, \Delta \boldsymbol{w} = 0$$

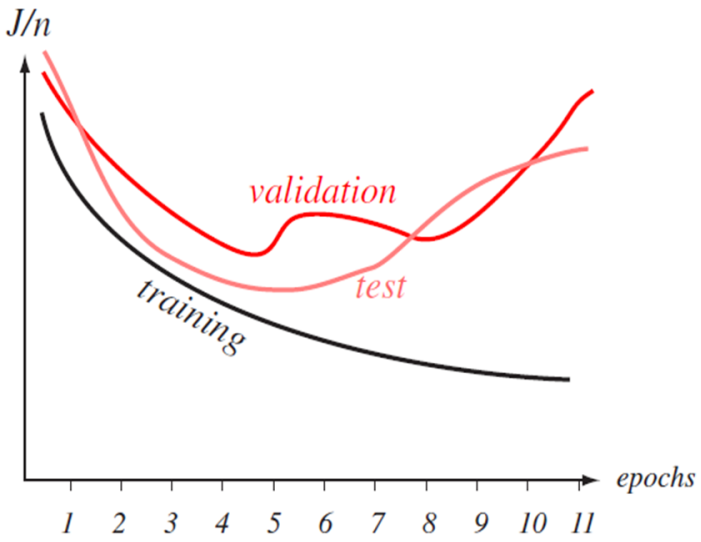
- Can be expressed as the gradient plus a component of previous search direction

$$\Delta \boldsymbol{w}(m) = \frac{\partial J(\boldsymbol{w}(m))}{\partial \boldsymbol{w}} + \beta_m \Delta \boldsymbol{w}(m-1)$$

- Need clever (approximate) way to set β_m: Fletcher-Reeves method

$$\beta_m = \frac{\nabla J^T(\boldsymbol{w}(m))\, \nabla J(\boldsymbol{w}(m))}{\nabla J^T(\boldsymbol{w}(m-1))\, \nabla J(\boldsymbol{w}(m-1))}$$

Performance on Out-of-Sample data



- Validation set to monitor error
- Stop (early) when validation error begins to increase

MLP As Posterior Probability Estimator

- Recall Bayes' decision

$$P[\omega_k|\boldsymbol{x}] = \frac{p(\boldsymbol{x}|\omega_k)\, p[\omega_k]}{\sum_{i=1}^{c} p(\boldsymbol{x}|\omega_i)\, p[\omega_i]}$$

- Suppose we train a neural net with c outputs

$$t_k(\boldsymbol{x}) = \begin{cases} 1 \text{ if } \boldsymbol{x} \in \omega_k \\ 0 \text{ otherwise} \end{cases}$$

- Mininimising the error

$$\sum_{\boldsymbol{x}} [g_k(\boldsymbol{x}, \boldsymbol{w}) - t_k]^2$$

when we have infinite data, can be shown to be the same as minimizing

$$\sum_{k=1}^{c} \int [g_k(\boldsymbol{x}, \boldsymbol{w}) - P(\omega_k|\boldsymbol{x})]\, p(\boldsymbol{x})\, d\boldsymbol{x}$$