Machine Learning

Week 7: Artificial Neural Networks

Mahesan Niranjan

School of Electronics and Computer Science
University of Southampton

Autumn Semester 2016/17

Overview of Lecture

- Non-linear class boundaries (fixed non-linear parts)
  - Quadratic discriminant function
  - Radial basis functions
- Multi-layer perceptron
  - Error back-propagation algorithm
  - Speed-up tricks
  - Over-fitting and regularization

Radial Basis Functions

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

The Model:

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

- 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

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

- *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

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

- 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

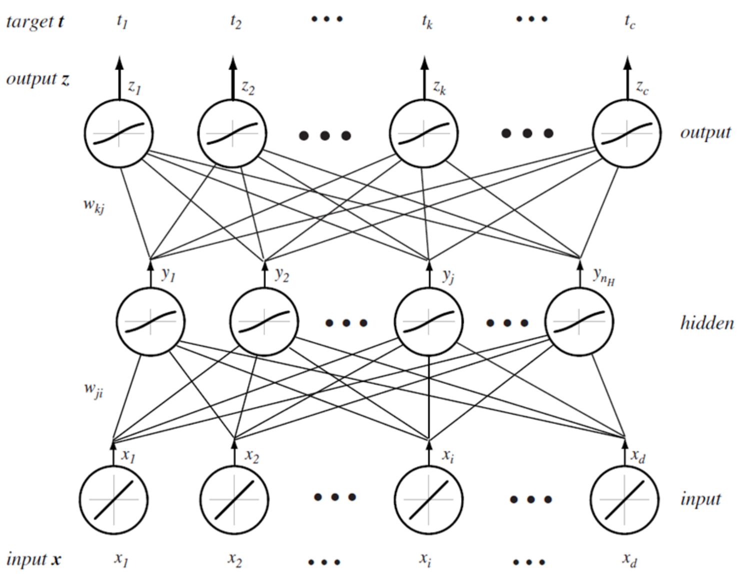
$$\widehat{\boldsymbol{\lambda}} = (Y^T Y)^{-1} Y^T \boldsymbol{f}$$

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

Lab Five: from 12 November

- Implement your own RBF
- Boston housing problem and a problem of your choice
- Compare with linear regression
- Cross validation to evaluate uncertainty
- Comparisons on out-of-sample data

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

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

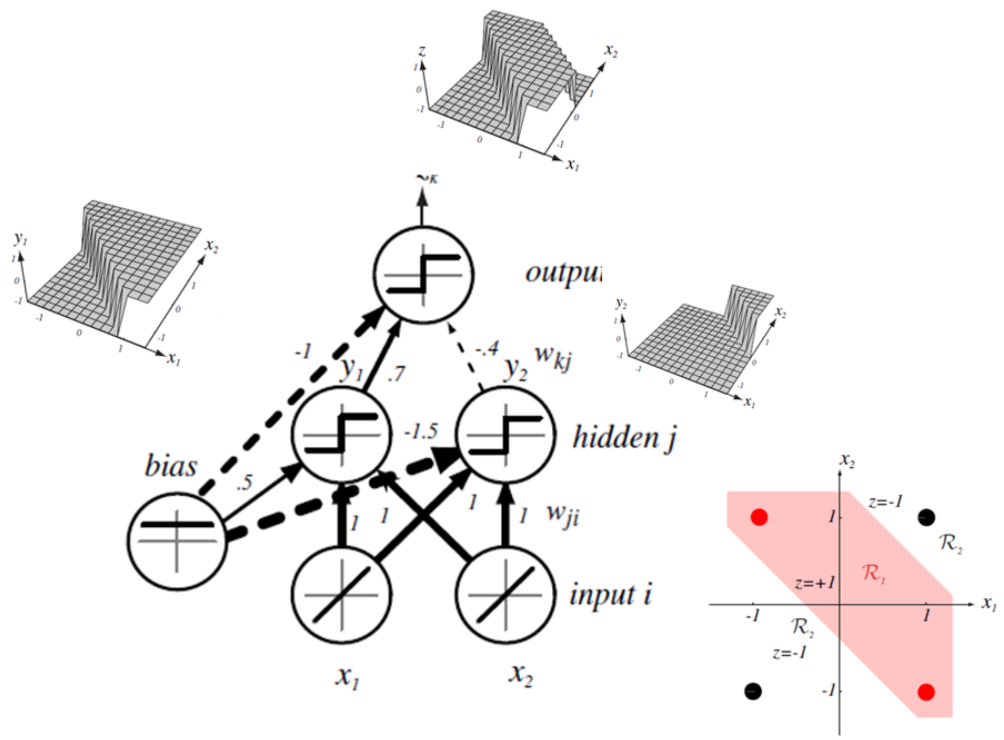
- Sigmoid (logistic)

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

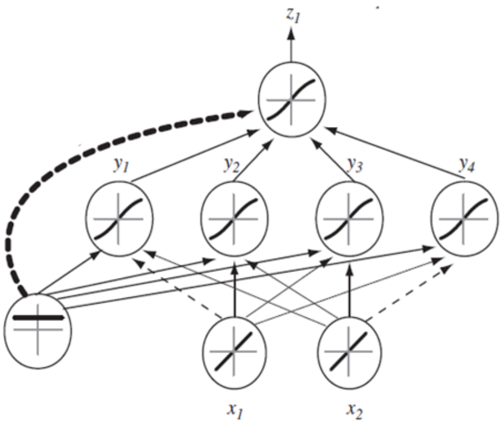
- Hyperbolic tangent

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

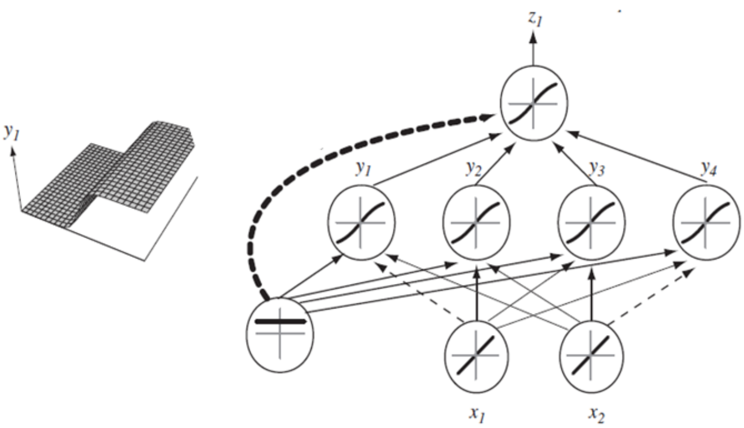
XOR Using a MLP



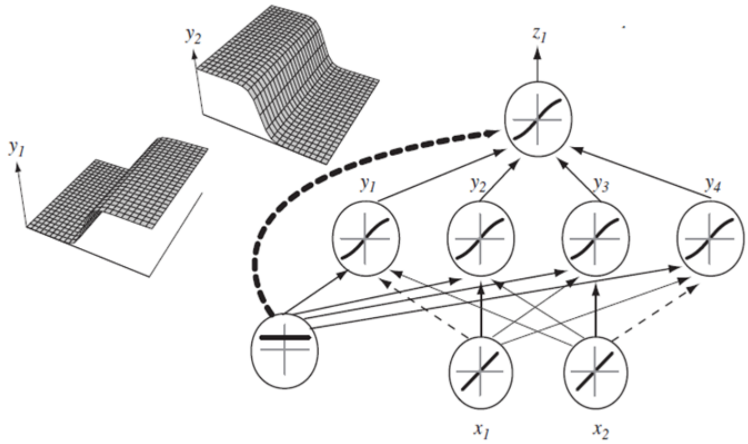
MLP Constructing Complex Functions



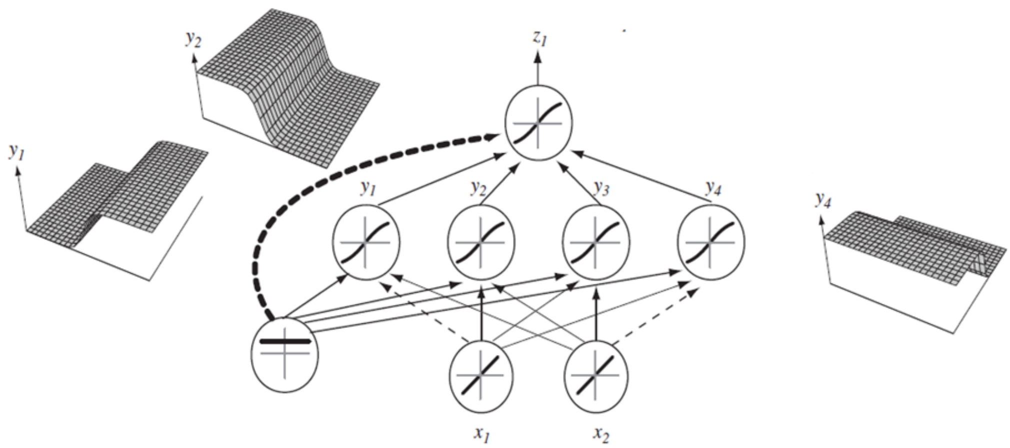
MLP Constructing Complex Functions



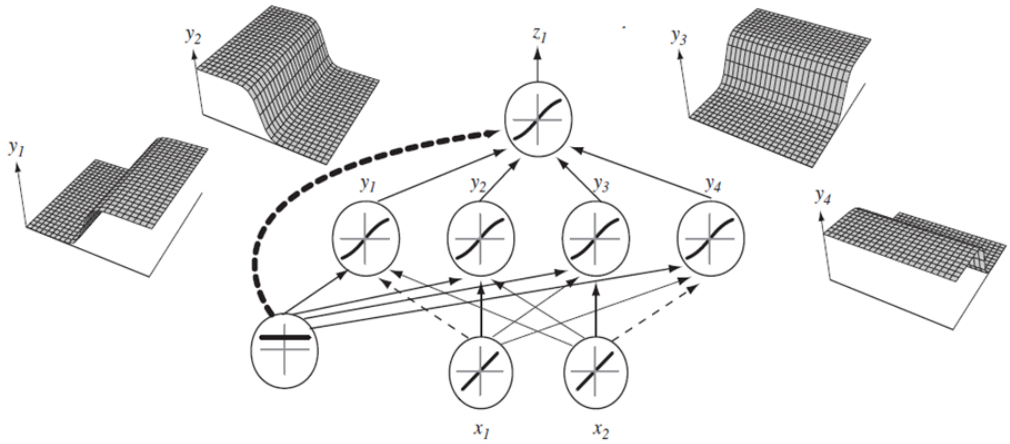
MLP Constructing Complex Functions



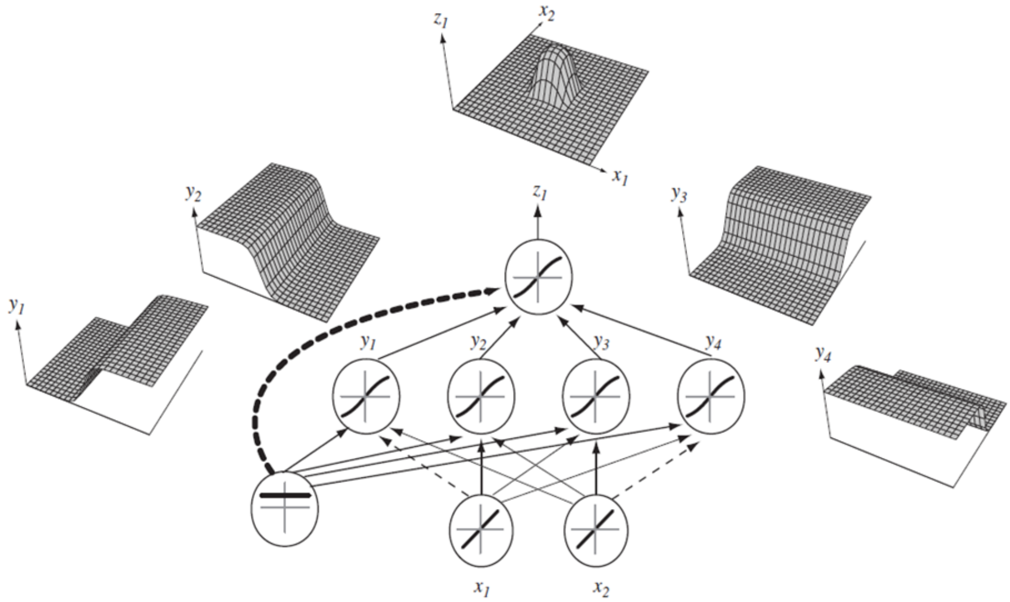
MLP Constructing Complex Functions



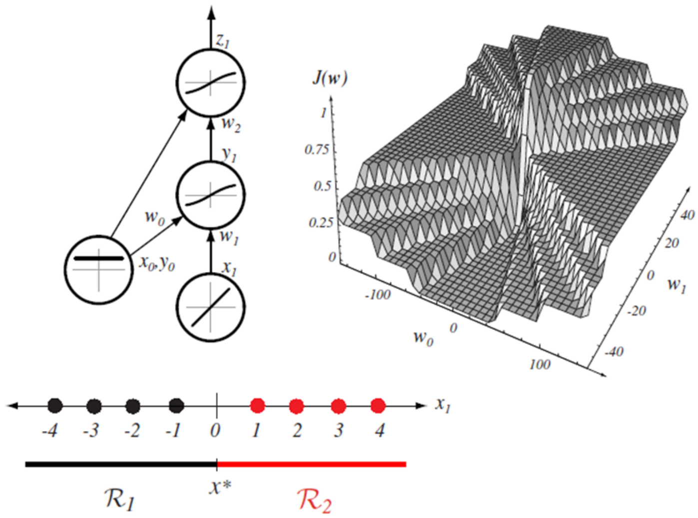
MLP Constructing Complex Functions



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 Back-propagation algorithm

- Error

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

- Change in weight for gradient descent

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

Or in component form

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

- Gradient descent update

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

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!

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

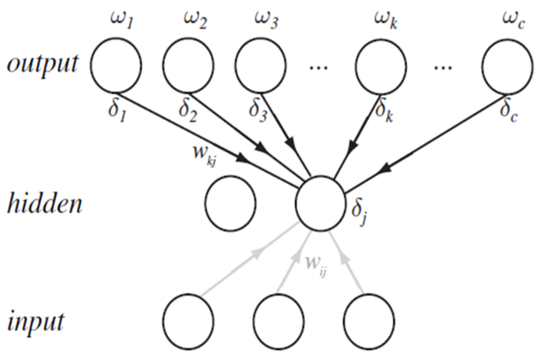
$$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right]$$
$$= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j}$$
$$= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial \mathrm{net}_k}\frac{\partial \mathrm{net}_k}{\partial y_j}$$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

Error propagation (cont'd)

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

and the update rule

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



- 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

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

- We can consider the change in objective function

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

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

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

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

... giving us the update

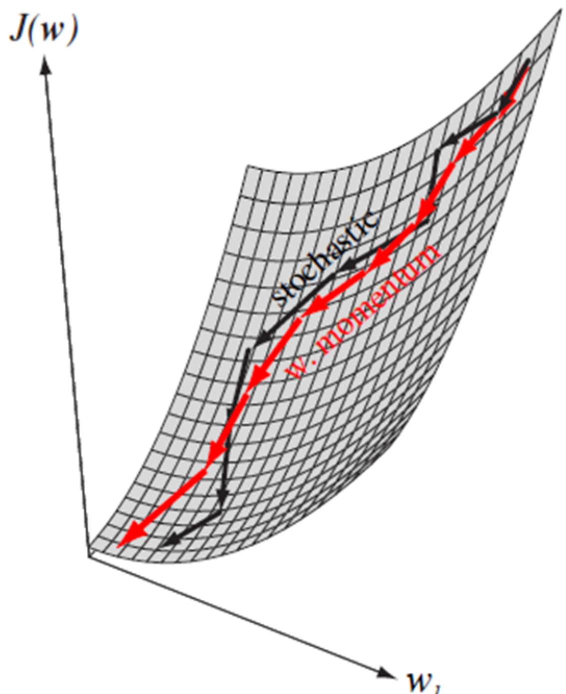
$$\begin{aligned}\mathbf{w}(m+1) \;&=\; \mathbf{w}(m) + \Delta\mathbf{w}\\ &=\; \mathbf{w}(m) - H^{-1}(m)\left(\frac{\partial J(\mathbf{w}(m))}{\partial \mathbf{w}}\right)\end{aligned}$$

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

Momentum

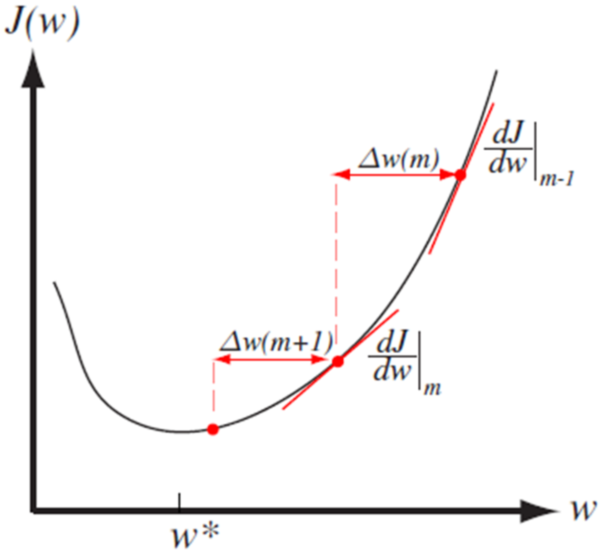
- Slow learning in regions in which gradient is small

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



QuickProp

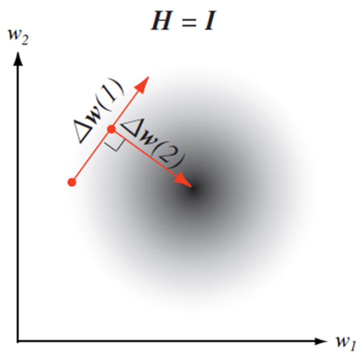
Two successive evaluations to approximate local curvature

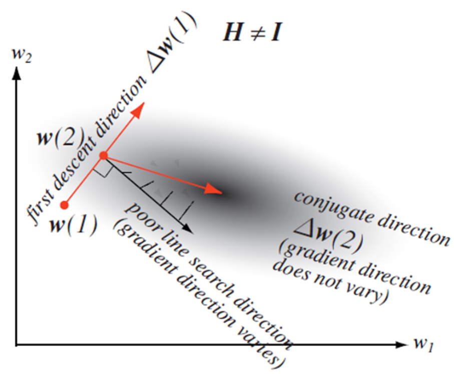


$$\Delta w(m+1) = \frac{\partial J / \partial w \,|_m}{\partial J / \partial w \,|_{m-1} - \partial J / \partial w \,|_m} \Delta w(m)$$

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
- $\Delta \mathbf{w}(m-1)$ direction of line search at step $m-1$
- For new direction to search, find $\Delta \mathbf{w}$

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

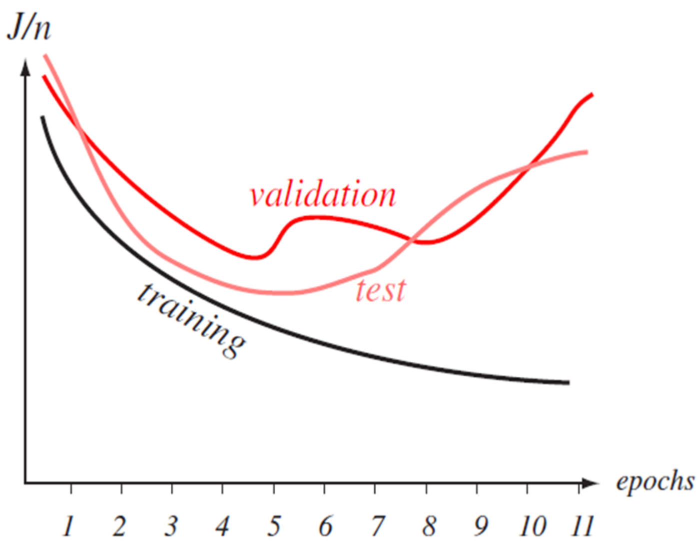
- Can be expressed as the gradient plus a component of previous search direction

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

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

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

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

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

- 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

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

Summary

- Linear discriminant functions / regression (limited scope)
- Nonlinear models
  - More powerful
  - Difficult to train
- Some clever tricks
  - Variants of gradient descent
  - Fixed non-linear, variable linear models
- Estimators of $P\,[\,\omega_j\,|\,\boldsymbol{x}\,]$

What next:

- Coursework on MLP
  - How well does it approximate posterir probabilities?
  - How do we model time series?
- Guest lecture: Deep Neural Networks