Artificial Neural Networks

Mingmin Chi

Fudan University, Shanghai, China

November 11, 2018

Outline

Advantages of Fix Basis Functions

- Closed-form solutions to least squares problem
- Capability to modeling arbitrary nonlinearities in the mapping from inputs to targets by a suitable choice of basis functions

Limitations of Fix Basis Functions

- Curse of dimensionality

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d} \sum_{j=1}^{d} w_{ij} x_i x_j + \sum_{i=1}^{d} \sum_{j=1}^{d} \sum_{k=1}^{d} w_{ijk} x_i x_j x_k$$

Two properties to alleviating this problem

- the data vectors typically lie close to a nonlinear manifold whose intrinsic dimensionality is smaller than that of the input space, e.g., localized basis functions, which are scattered in input space only in regions containing data by rearranging, e.g., RBF-NN, SVM and RVM
- the target variables may have significant dependence on only a small number of possible directions within the data manifold: Neural networks can exploit this by choosing the directions in input space to which the basis functions respond

Single-Layer Networks

Single-layer networks have many advantages:

$$y(\mathbf{x}) = f(\mathbf{w}^\top \mathbf{x})$$

The nonlinear activation function $f(\cdot)$ is given by a step function of the form

$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases}$$

- Easy to setup and train
- Outputs are weighted sum of inputs: interpretable representation

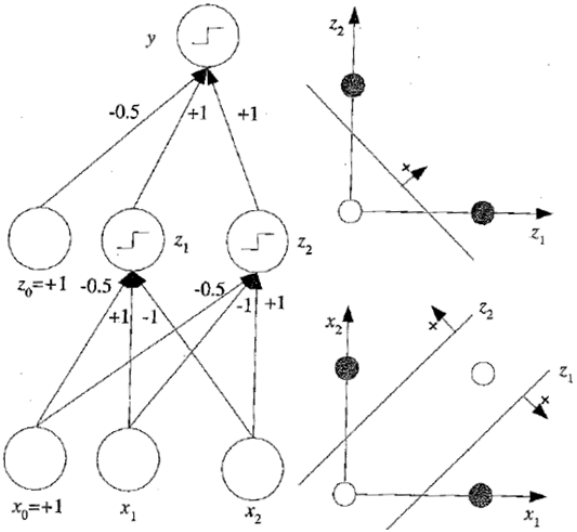
Limitations:

- Can only represent a limited set of functions
- Decision boundaries must be hyperplane
- Can only perfectly separate linearly separable data

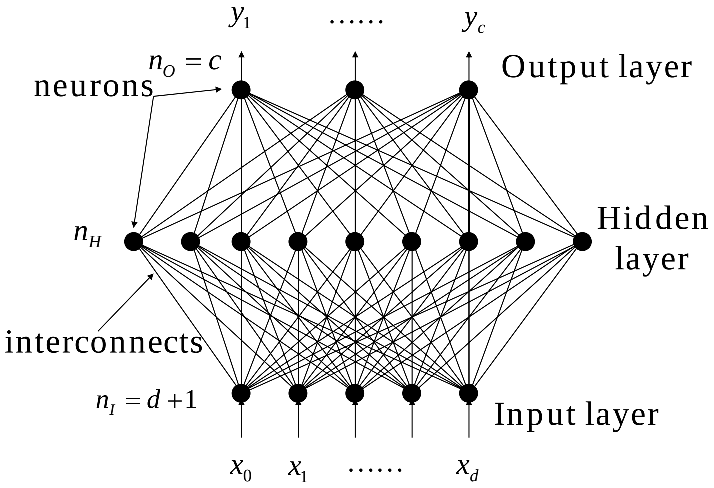
Relaxation to Other Neural Networks?

- Can we extend to three- or four-layer nets to overcome those drawbacks?
- How much multilayer networks can, at least in principle, provide the optimal solution to an arbitrary classification problem?
- They implement linear discriminants, nonlinear mapping

XoR problem



Topology of Two-layer Perceptron NNs (contd)



Following the same form of the linear models for regression and classification

- Each *d*-dimensional input vector is presented to the input layer
- Each hidden unit emits an output to the output layer or the next hidden layer
  - Each hidden unit computes the weighted sum of its inputs to form its scalar *net activation* which we denote simply as *net*:

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

Following the same form of the linear models for regression and classification

- Each *d*-dimensional input vector is presented to the input layer
- Each hidden unit emits an output to the output layer or the next hidden layer
  - Each hidden unit computes the weighted sum of its inputs to form its scalar *net activation* which we denote simply as *net*:

$$\text{net}_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji}$$

Following the same form of the linear models for regression and classification

- Each *d*-dimensional input vector is presented to the input layer
- Each hidden unit emits an output to the output layer or the next hidden layer
  - Each hidden unit computes the weighted sum of its inputs to form its scalar *net activation* which we denote simply as *net*:

$$\text{net}_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} = \mathbf{w}_j^\top \mathbf{x}$$

    w_{ji} denotes the input-to-hidden layer weights at the hidden unit j.
  - Each hidden unit emits an output that is a nonlinear function of its activation, $f(\text{net}_j)$, i.e.,

$$y_j = g(\text{net}_j)$$

- Each output unit emits the category
  - Each output unit similarly computes its net activation based on the hidden unit signals as

$$\mathrm{net}_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0}$$

- Each output unit emits the category
  - Each output unit similarly computes its net activation based on the hidden unit signals as

$$\mathrm{net}_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj}$$

- Each output unit emits the category
  - Each output unit similarly computes its net activation based on the hidden unit signals as

    $$\text{net}_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} = \mathbf{w}_k^\top \mathbf{y}$$

    at the hidden unit j.
  - Each output unit emits an output that is a nonlinear function of its activation, $f(\text{net}_k)$, i.e.,

    $$y_k = f(\text{net}_k)$$

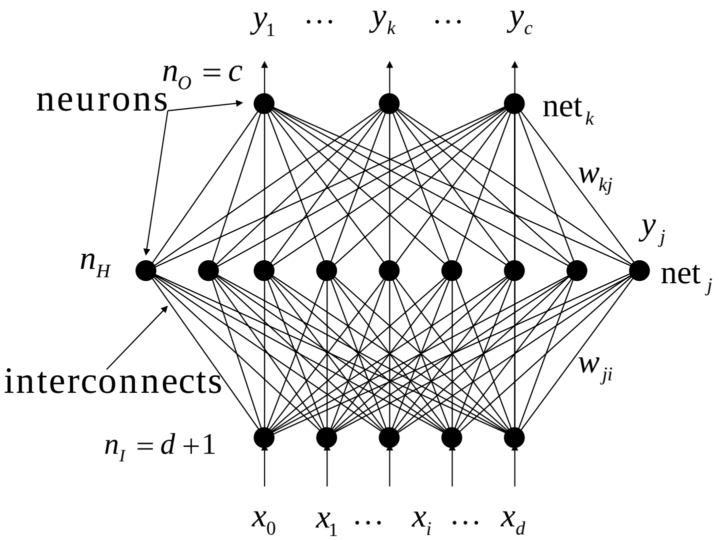
Feed-Forward Network Mappings

The mapping between inputs and outputs:

$$y_k = f\left(\underbrace{\sum_{j=0}^{n_H} f\left(\underbrace{\sum_{i=0}^{d} x_i w_{ji}}_{\text{net}_j} \right) w_{kj}}_{\text{net}_k} \right)$$

- Binary case, $y_k = \{+1, -1\}$
- Multi-category case, $y_k = f(\text{net}_k) = g_k(\mathbf{x})$

Topology of MLPNNs for Multiple Case Problem



Sum-of-Squares Error Function

Neural networks as a general class of parametric nonlinear functions by the mapping between input variables and output variables

Parametric estimation

Simply, we consider the training error on a pattern to be the sum over output units of the squared difference between the desired output **t** and the actual output **y**:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(y(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n)^2$$

Probabilistic Interpretation

Much general view of neural training by first giving a probabilistic interpretation to the network output

Advantages of using probabilistic predictions

- the choice of output unit nonlinearity
- the choice of error function

Problem Setting

- a single target variable t
- assume that t has a Gaussian distribution with an **x**-dependent mean, which is given by the output of the neural network,

$$P(t|\mathbf{x},\mathbf{w}) = \mathcal{N}(t|y(\mathbf{x},\mathbf{w}),\beta^{-1})$$

- Given i.i.d observations $\mathbf{X} = (\mathbf{x}_n)_{n=1}^N$ along with corresponding target values $\mathbf{t} = (t_n)_{n=1}^N$, the corresponding likelihood function:

Problem Setting

- a single target variable t
- assume that t has a Gaussian distribution with an $\mathbf{x}$-dependent mean, which is given by the output of the neural network,

$$P(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1})$$

- Given i.i.d observations $\mathbf{X} = (\mathbf{x}_n)_{n=1}^N$ along with corresponding target values $\mathbf{t} = (t_n)_{n=1}^N$, the corresponding likelihood function:

$$P(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} P(t_n|\mathbf{x}_n, \mathbf{w}, \beta)$$

$$\Rightarrow \frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\}^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2\pi)$$

error function

Determination of **w** by ML

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n,\mathbf{w})-\mathbf{t}_n\}^2-\frac{N}{2}\ln\beta+\frac{N}{2}\ln(2\pi)$$

Determination of **w**

- Maximizing the likelihood function is equivalent to minimizing the sum-of-squares error function given by

$$E(\mathbf{w})=\frac{1}{2}\sum_{n=1}^{N}(y(\mathbf{x}_n,\mathbf{w})-\mathbf{t}_n)^2$$

- Is the error function $E(\mathbf{w})$ convex?

Determination of **w** by ML

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n,\mathbf{w})-\mathbf{t}_n\}^2-\frac{N}{2}\ln\beta+\frac{N}{2}\ln(2\pi)$$

Determination of **w**

- Maximizing the likelihood function is equivalent to minimizing the sum-of-squares error function given by

$$E(\mathbf{w})=\frac{1}{2}\sum_{n=1}^{N}(y(\mathbf{x}_n,\mathbf{w})-\mathbf{t}_n)^2$$

- Is the error function $E(\mathbf{w})$ convex? reason: the nonlinearity of the network function $y(\mathbf{x}_n,\mathbf{w})$

Determination of β

Having found $\mathbf{w}_{\text{ML}}$, the value of β can be found by minimizing the negative log likelihood to give

$$\frac{1}{\beta_{\text{ML}}} = \frac{1}{N}\sum_{n=1}^{N}(y(\mathbf{x}_n, \mathbf{w}_{\text{ML}}) - \mathbf{t}_n)^2$$

Binary Classification

- A single target variable t: $t = 1$ denotes $\mathcal{C}_1$ and $t = 0$ denotes $\mathcal{C}_2$
- Consider a network having a single output whose activation function is a logistic sigmoid $y = \sigma(a) \equiv \frac{1}{1+\exp(-a)}$
- Probabilistic interpretation: $y(\mathbf{x}, \mathbf{w}) = P(\mathcal{C}_1|\mathbf{x})$ and so $P(\mathcal{C}_2|\mathbf{x}) = 1 - y(\mathbf{x}, \mathbf{w})$
- Conditional distribution of targets given inputs is then a Bernoulli distribution of the form

$$P(t|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t \{1 - y(\mathbf{x}, \mathbf{w})\}^{1-t}$$

$$\Rightarrow E(\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

cross-entropy error function

K Separate Binary Classification

A network having K outputs each of which has a logistic sigmoid activation function

$$P(\mathbf{t}|\mathbf{x},\mathbf{w}) = \prod_{k=1}^{K} y_k(\mathbf{x},\mathbf{w})^{t_k} \left\{1 - y_k(\mathbf{x},\mathbf{w})\right\}^{1-t_k}$$

K Separate Binary Classification

A network having K outputs each of which has a logistic sigmoid activation function

$$P(\mathbf{t}|\mathbf{x},\mathbf{w}) = \prod_{k=1}^{K} y_k(\mathbf{x},\mathbf{w})^{t_k} \left\{1 - y_k(\mathbf{x},\mathbf{w})\right\}^{1-t_k}$$

$$\Rightarrow E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} \left\{t_{nk}\ln y_{nk} + (1-t_{nk})\ln(1-y_{nk})\right\}$$

1-of-K Coding Scheme for Multiclass Classification Problems

$$P(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \prod_{k=1}^{K} y_k(\mathbf{x}, \mathbf{w})^{t_k}$$

1-of-K Coding Scheme for Multiclass Classification Problems

$$P(\mathbf{t}|\mathbf{x},\mathbf{w}) = \prod_{k=1}^{K} y_k(\mathbf{x},\mathbf{w})^{t_k}$$

$$\Rightarrow E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_k(\mathbf{x}_n,\mathbf{w})$$

where

$$y_k(\mathbf{x},\mathbf{w}) = \frac{\exp(a_k(\mathbf{x},\mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x},\mathbf{w}))}$$

How to find **w**

Finding a weight vector **w** which minimizes the error function $E(\mathbf{w})$

- $\mathbf{w} \rightarrow \mathbf{w} + \delta\mathbf{w} \quad \Rightarrow$ the change in the error function

How to find **w**

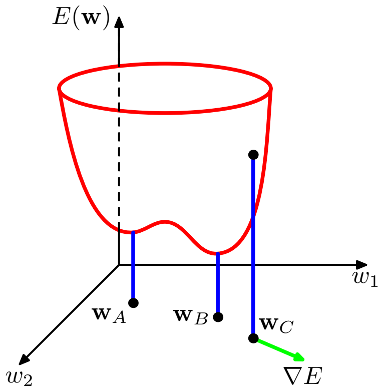
Finding a weight vector **w** which minimizes the error function $E(\mathbf{w})$

- $\mathbf{w} \to \mathbf{w} + \delta\mathbf{w} \quad \Rightarrow$ the change in the error function $\delta E \simeq \delta\mathbf{w}^\top \nabla E(\mathbf{w})$
  - the vector $\nabla E(\mathbf{w})$ points in the direction of greatest rate of increase of the error function
- our goal is to find a vector **w** such that $E(\mathbf{w})$ takes its smallest value
  - δE occurring at a stationary point, i.e.,

$$\nabla E(\mathbf{w}) = 0$$

  - minima, maxima, or saddle points
  - or reducing the error by making a small step in the direction of $-\nabla E(\mathbf{w})$

Geometrical picture of $E(\mathbf{w})$



Non-Convex Problem: Local Minima

Weight Symmetry

- Multiple distinct choices of the weight vector $\rightarrow$ the same mapping function from inputs to outputs, e.g.,
  - M hidden units with ‘tanh’ activation function
  - flipping the sign of weights
  - $\tanh(-a) = -\tanh(a)$
- Interchanging the values of all the weights $\rightarrow$ input to output function unchanged

There are typically multiple inequivalent stationary points and in particular multiple inequivalent minima

Iterative Numerical Strategy

- No hope of finding an analytical solution to $\nabla E(\mathbf{w}) = 0$
- we can resort to iterative numerical procedures

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

- usually, gradient information is widely used in many optimization algorithms

Taylor Expansion

Using Taylor expansion to consider a local approximation to the error function

- Consider the Taylor expansion of $E(\mathbf{w})$ around some point $\hat{\mathbf{w}}$:

$$E(\mathbf{w}) \simeq E(\hat{\mathbf{w}}) + (\mathbf{w} - \hat{\mathbf{w}})^\top \mathbf{b} + \frac{1}{2}(\mathbf{w} - \hat{\mathbf{w}})^\top \mathbf{H}(\mathbf{w} - \hat{\mathbf{w}})$$

  where $\mathbf{b} \equiv \nabla E|_{\mathbf{w}=\hat{\mathbf{w}}}$ and the Hessian matrix $\mathbf{H} = \nabla\nabla E$ has the elements

$$\mathbf{H}_{ij} \equiv \left.\frac{\partial E}{\partial w_i \partial w_j}\right|_{\mathbf{w}=\hat{\mathbf{w}}}$$

- the corresponding local approximation to the gradient is given by

$$\nabla E \simeq \mathbf{b} + \mathbf{H}(\mathbf{w} - \hat{\mathbf{w}})$$

Taylor Expansion at Minima

Consider a local quadratic approximation around a point $\mathbf{w}^*$ that is a minimum of the error function:

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^\top \mathbf{H}(\mathbf{w} - \mathbf{w}^*)$$

- Consider the eigenvalue equation for the Hessian matrix:

$$\mathbf{H}\mathbf{u}_i = \lambda_i \mathbf{u}_i, \mathbf{u}_i \mathbf{u}_i = \delta_{ij}$$

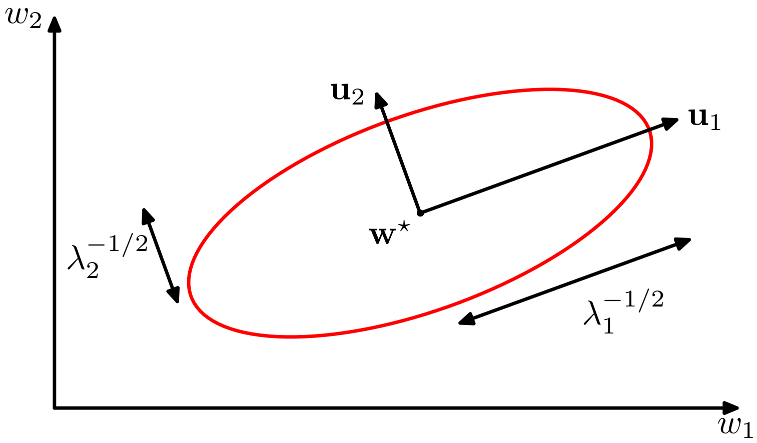
- Expand $\mathbf{w} - \mathbf{w}^*$ as a linear combination of the eigenvectors:

$$\mathbf{w} - \mathbf{w}^* = \sum_i \alpha_i \mathbf{u}_i$$

- The error function

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}\sum \lambda_i \alpha_i^2$$

Taylor Expansion at Minima (contd)



Use of Gradient Information

Without using gradient

Taylor expansion of $E(\mathbf{w})$ around some point $\hat{\mathbf{w}}$:

$$E(\mathbf{w}) \simeq E(\hat{\mathbf{w}}) + (\mathbf{w} - \hat{\mathbf{w}})^\top \mathbf{b} + \frac{1}{2}(\mathbf{w} - \hat{\mathbf{w}})^\top \mathbf{H}(\mathbf{w} - \hat{\mathbf{w}})$$

- $W(W+3)/2$ independent elements
- $\mathcal{O}(W^2)$ function evaluation with $\mathcal{O}(W)$ step
- Totally $\mathcal{O}(W^3)$ computation complexity

Using gradient

- ∇E brings W items of information
- $\mathcal{O}(W)$ gradient evaluation
- Totally $\mathcal{O}(W^2)$ computation complexity

Gradient-based Optimization

- gradient descent or steepest descent: at each step the weight vector is moved in the direction of the greatest rate of decrease of error function (batch mode)

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

- conjugate gradients or quasi-Newton: the error function always decreases at each iteration unless the weight vector has arrived at a local or global minimum
- stochastic gradient descent or sequential gradient descent: an on-line version of gradient descent

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)}), \; E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

Intuition

- The goal of the network training is to find an efficient technique for evaluating the gradient of an error function for a feed-forward neural network

Intuition

- The goal of the network training is to find an efficient technique for evaluating the gradient of an error function for a feed-forward neural network
- Could we calculate an effective error for each hidden unit, and thus derive a learning rule for the input-to-hidden weights?

In error backpropagation using a local message passing scheme information is sent alternately forwards and backwards through the network

Training Error

In general

- We consider the training error on a pattern to be the sum over output units of the squared difference between the desired output t_k and the actual output y_k:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{c}(t_k - y_k)^2 = \frac{1}{2}||\mathbf{t} - \mathbf{y}||^2$$

Training Error

In general

- We consider the training error on a pattern to be the sum over output units of the squared difference between the desired output t_k and the actual output y_k:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{c}(t_k - y_k)^2 = \frac{1}{2}||\mathbf{t} - \mathbf{y}||^2$$

- The back-propagation learning rule is based on gradient descent.

Gradient Descent

The back-propagation learning rule is based on gradient descent

- The weights are initialized with random values

Gradient Descent

The back-propagation learning rule is based on gradient descent

- The weights are initialized with random values
- The weights are changed in a direction that will reduce the error:

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

or in component form $\Delta \mathbf{w}_{pq} = -\eta \frac{\partial E}{\partial w_{pq}}$

Gradient Descent

The back-propagation learning rule is based on gradient descent

- The weights are initialized with random values
- The weights are changed in a direction that will reduce the error:

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

  or in component form $\Delta \mathbf{w}_{pq} = -\eta \frac{\partial E}{\partial w_{pq}}$

- This iterative algorithm requires taking a weight vector at iteration τ and updating it as

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

The Chain Rule for Differentiation

$$\text{net}_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj}$$

- Since the error is not explicitly dependent upon w_{kj}, we must use the chain rule for differentiation:

$$\frac{\partial E}{\partial w_{kj}}$$

The Chain Rule for Differentiation

$$\text{net}_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj}$$

- Since the error is not explicitly dependent upon w_{kj}, we must use the chain rule for differentiation:

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial \text{net}_k} \frac{\partial \text{net}_k}{\partial w_{kj}}$$

The Chain Rule for Differentiation

$$\text{net}_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj}$$

- Since the error is not explicitly dependent upon w_{kj}, we must use the chain rule for differentiation:

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial \text{net}_k} \frac{\partial \text{net}_k}{\partial w_{kj}} = -\delta_k \frac{\partial \text{net}_k}{\partial w_{kj}}$$

The Chain Rule for Differentiation

$$\text{net}_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj}$$

- Since the error is not explicitly dependent upon w_{kj}, we must use the chain rule for differentiation:

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial \text{net}_k} \frac{\partial \text{net}_k}{\partial w_{kj}} = -\delta_k \frac{\partial \text{net}_k}{\partial w_{kj}}$$

- We define δ_k as the *sensitivity* of the unit k:

$$\delta_k = -\frac{\partial E}{\partial \text{net}_k}$$

The Chain Rule for Differentiation (contd)

- Assuming that the activation function $f(\cdot)$ is differentiable, δ_k can be simplified as follows:

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

The Chain Rule for Differentiation (contd)

- Assuming that the activation function $f(\cdot)$ is differentiable, δ_k can be simplified as follows:

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

The Chain Rule for Differentiation (contd)

- Assuming that the activation function $f(\cdot)$ is differentiable, δ_k can be simplified as follows:

$$\delta_k = -\frac{\partial E}{\partial \mathrm{net}_k} = -\frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial \mathrm{net}_k} = (t_k - y_k)f'(\mathrm{net}_k)$$

The Chain Rule for Differentiation (contd)

- Assuming that the activation function $f(\cdot)$ is differentiable, δ_k can be simplified as follows:

$$\delta_k = -\frac{\partial E}{\partial \text{net}_k} = -\frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial \text{net}_k} = (t_k - y_k)f'(\text{net}_k)$$

- As $\text{net}_k = \sum_{j=0}^{n_H} w_{kj} y_j$, we have $\frac{\partial \text{net}_k}{\partial w_{kj}} = y_j$
- Learning rule for the hidden-to-output weights:

$$\Delta w_{kj} = \eta \delta_k y_j$$

The Chain Rule for Differentiation (contd)

- Assuming that the activation function $f(\cdot)$ is differentiable, δ_k can be simplified as follows:

$$\delta_k = -\frac{\partial E}{\partial \text{net}_k} = -\frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial \text{net}_k} = (t_k - y_k) f'(\text{net}_k)$$

- As $\text{net}_k = \sum_{j=0}^{n_H} w_{kj} y_j$, we have $\frac{\partial \text{net}_k}{\partial w_{kj}} = y_j$
- Learning rule for the hidden-to-output weights:

$$\Delta w_{kj} = \eta \delta_k y_j = \eta \underbrace{(t_k - y_k) f'(\text{net}_k)}_{\delta_k} y_j$$

- In the case of the output unit is linear, i.e., $f(\text{net}_k) = \text{net}_k$ and $f'(\text{net}_k) = 1$, then the above equation is simply the LMS rule

The Chain Rule for Differentiation

The learning rule for the input-to-hidden units is more subtle, and it is the crux of the solution to the credit assignment problem

The Chain Rule for Differentiation

The learning rule for the input-to-hidden units is more subtle, and it is the crux of the solution to the credit assignment problem

$$\text{net}_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji}$$

With the chain rule, we calculate

$$\frac{\partial E}{\partial w_{ji}}$$

The Chain Rule for Differentiation

The learning rule for the input-to-hidden units is more subtle, and it is the crux of the solution to the credit assignment problem

$$\text{net}_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji}$$

With the chain rule, we calculate

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

The Chain Rule for Differentiation(contd)

$$\text{net}_j = \sum_{i=0}^{d} x_i w_{ji}, \ \frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial \text{net}_j}\frac{\partial \text{net}_j}{\partial w_{ji}}$$

- The last term on the right-hand side involves all of the weights w_{ji}:

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

- The second term on the right-hand side:

$$\frac{\partial y_j}{\partial \text{net}_j} = f'(\text{net}_j)$$

The Chain Rule for Differentiation(contd)

- The first term on the right-hand side involves all of the weights w_{kj}:

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

The Chain Rule for Differentiation(contd)

- In analogy with the case for hidden-to-output, we can also define the sensitivity for a hidden unit as

The Chain Rule for Differentiation(contd)

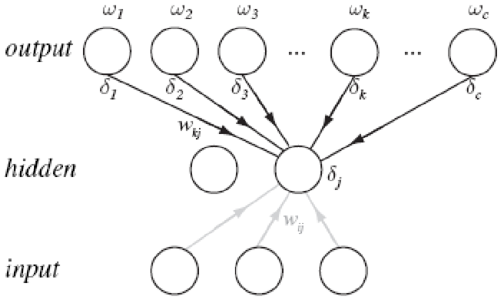
- In analogy with the case for hidden-to-output, we can also define the sensitivity for a hidden unit as

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

  This is the core of the solution to the credit assignment problem
- The learning rule for the input-to-hidden weights is

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

BP Algorithm



$$\Delta w_{kj} = \eta \underbrace{(t_k - y_k) f'(\text{net}_k)}_{\delta_k} z_j$$

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

BP Algorithm

```
begin initialize network topology (# hidden units), $\mathbf{w}$, criterion $\theta, \eta, r \leftarrow 0$
    do $r \leftarrow r + 1$ (increment epoch)
        $m \leftarrow 0;\ \Delta w_{ij} \leftarrow 0;\ \Delta w_{jk} \leftarrow 0$
        do $m \leftarrow m + 1$
            $\mathbf{x}^m \leftarrow$ select pattern
            $\Delta w_{ij} \leftarrow \Delta w_{ij} + \eta \delta_j x_i;\quad \Delta w_{jk} \leftarrow \Delta w_{jk} + \eta \delta_k y_j$
        until $m = n$
        $w_{ij} \leftarrow w_{ij} + \Delta w_{ij};\quad w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$
    until $\nabla J(\mathbf{w}) < \theta$
return $\mathbf{w}$
end
```

- Input units include a bias unit

- Input units include a bias unit
- Input units are connected directly to output units as well as hidden units

- Input units include a bias unit
- Input units are connected directly to output units as well as hidden units
- Three are more than two-layer of units

- Input units include a bias unit
- Input units are connected directly to output units as well as hidden units
- Three are more than two-layer of units
- There are different nonlinearities for different layers

- Input units include a bias unit
- Input units are connected directly to output units as well as hidden units
- Three are more than two-layer of units
- There are different nonlinearities for different layers
- Each unit has its own nonlinearity

- Input units include a bias unit
- Input units are connected directly to output units as well as hidden units
- Three are more than two-layer of units
- There are different nonlinearities for different layers
- Each unit has its own nonlinearity
- Each unit has a different learning rate

- MLPNN computes a non-linear function of the scalar product of the input vector and a weight vector
- The other major class of neural network model can be considered in which the activation of a hidden unit is determined by the distance between the input vector and a prototype vector, known as Radial Basis Function neural networks, RBF-NN
- The training for RBF networks can be substantially faster than the methods used to train multi-layer perceptron networks

Exact Interpolation

- Origins in techniques for performing exact interpolation of a set of data points in a multi-dimensional space
- The exact interpolation problem requires every input vector to be mapped exactly onto the corresponding target vector, e.g., if the data set $(\mathbf{x}_i, t_i), \mathbf{x}_i \in \mathbb{R}^d, t^i \in \mathbb{R}, , i = 1, \cdots, n$, the goal is to find a function $h(\mathbf{x})$ such that $h(\mathbf{x}_i) = t_i$

Exact Interpolation

- Origins in techniques for performing exact interpolation of a set of data points in a multi-dimensional space
- The exact interpolation problem requires every input vector to be mapped exactly onto the corresponding target vector, e.g., if the data set $(\mathbf{x}_i, t_i), \mathbf{x}_i \in \mathbb{R}^d, t^i \in \mathbb{R}, , i = 1, \cdots, n$, the goal is to find a function $h(\mathbf{x})$ such that $h(\mathbf{x}_i) = t_i$
- The radial basis function approach (Powell, 1987) introduce s a set of n basis functions, one for each point, which take the form $\phi(||\mathbf{x} - \mathbf{x}_i||)$
- The ith such function depends on the distance $||\mathbf{x} - \mathbf{x}_i||$
- The output of the mapping:

$$h(\mathbf{x}) = \sum_i w_i \phi(||\mathbf{x} - \mathbf{x}_i||)$$

Exact Interpolation (contd)

- The interpolation conditions can then be written in matrix form as

$$\Phi \mathbf{w} = \mathbf{t}$$

- Provided the inverse matrix Φ^{-1} exists we can solve the above function to give

$$\mathbf{w} = \Phi^{-1}\mathbf{t}$$

- The most commonly used the function is Gaussian:

$$\phi(\mathbf{x}) = \exp\left(-\frac{\mathbf{x}^2}{2\sigma^2}\right)$$

RBF networks

- Limitation of the exact interpolation
- By introducing a number of modifications to the exact interpolation procedure we obtain the radial basis function neural network model (Broomhead and Lowe, 1988; Moody and Darken, 1989)

RBF networks

- Limitation of the exact interpolation
- By introducing a number of modifications to the exact interpolation procedure we obtain the radial basis function neural network model (Broomhead and Lowe, 1988; Moody and Darken, 1989)
- This provides a smooth interpolating function in which the number of basis functions is determined by the complexity of the mapping to be represented rather than by the size of the data set. The modification which are required are as follows:

RBF networks

- Limitation of the exact interpolation
- By introducing a number of modifications to the exact interpolation procedure we obtain the radial basis function neural network model (Broomhead and Lowe, 1988; Moody and Darken, 1989)
- This provides a smooth interpolating function in which the number of basis functions is determined by the complexity of the mapping to be represented rather than by the size of the data set. The modification which are required are as follows:
  1. the number M of basis functions need not equal to the number n of data points, typically $M \ll n$

RBF networks

- Limitation of the exact interpolation
- By introducing a number of modifications to the exact interpolation procedure we obtain the radial basis function neural network model (Broomhead and Lowe, 1988; Moody and Darken, 1989)
- This provides a smooth interpolating function in which the number of basis functions is determined by the complexity of the mapping to be represented rather than by the size of the data set. The modification which are required are as follows:
    1. the number M of basis functions need not equal to the number n of data points, typically $M \ll n$
    2. The centers of the basis functions

RBF networks

- Limitation of the exact interpolation
- By introducing a number of modifications to the exact interpolation procedure we obtain the radial basis function neural network model (Broomhead and Lowe, 1988; Moody and Darken, 1989)
- This provides a smooth interpolating function in which the number of basis functions is determined by the complexity of the mapping to be represented rather than by the size of the data set. The modification which are required are as follows:
  1. the number M of basis functions need not equal to the number n of data points, typically $M \ll n$
  2. The centers of the basis functions
  3. The width σ_j of the basis functions

RBF networks

- Limitation of the exact interpolation
- By introducing a number of modifications to the exact interpolation procedure we obtain the radial basis function neural network model (Broomhead and Lowe, 1988; Moody and Darken, 1989)
- This provides a smooth interpolating function in which the number of basis functions is determined by the complexity of the mapping to be represented rather than by the size of the data set. The modification which are required are as follows:
  1. the number M of basis functions need not equal to the number n of data points, typically $M \ll n$
  2. The centers of the basis functions
  3. The width σ_j of the basis functions
  4. Bias parameters are included in the linear sum

RBF Networks

- When all the above change are made to the exact interpolation, we arrive at the following form for the RBF-NNs:

$$y_k(\mathbf{x}) = \sum_{j=1}^{M} w_{kj}\phi_j(\mathbf{x}) + w_{k0}$$

RBF Networks

- When all the above change are made to the exact interpolation, we arrive at the following form for the RBF-NNs:

$$y_k(\mathbf{x}) = \sum_{j=1}^{M} w_{kj}\phi_j(\mathbf{x}) + w_{k0}$$

- For the case of Gaussian basis functions we have

$$\phi_j(\mathbf{x}) = \exp\left(-\frac{||\mathbf{x} - \boldsymbol{\mu}_j||^2}{2\sigma_j^2}\right)$$

RBF Network Training

Two stage training procedure

1. The input dataset **X** alone is used to determine the parameters of the basis functions (e.g., μ_j and σ_j for the spherical Gaussian basis functions considered above)

RBF Network Training

Two stage training procedure

1. The input dataset **X** alone is used to determine the parameters of the basis functions (e.g., $\boldsymbol{\mu}_j$ and σ_j for the spherical Gaussian basis functions considered above)
2. The second stage training
   - We begin by absorbing the bias as

$$y_k(\mathbf{x}) = \sum_{j=0}^{M} w_{kj}\phi_j(\mathbf{x})$$

   - This can be written in matrix notation as

$$\mathbf{y} = \mathbf{W}\Phi$$

The Second-Stage RBF Network Training

- We can optimize the weights by minimization of a suitable error function, conveniently, to consider a sum-of-squares error function given by

$$E = \frac{1}{2} \sum_{i} \sum_{k} \{ (y_k(\mathbf{x}_i) - t_k^i)^2 \}$$

The Second-Stage RBF Network Training

- We can optimize the weights by minimization of a suitable error function, conveniently, to consider a sum-of-squares error function given by

$$E = \frac{1}{2}\sum_i \sum_k \{(y_k(\mathbf{x}_i) - t_k^i)^2\}$$

- We can obtain the normal equation for the least squares problem:

$$(\Phi^\top \Phi)\mathbf{W}^\top = \Phi^\top \mathbf{T}$$
$$\Rightarrow \mathbf{W}^\top = \Phi^\dagger \mathbf{T}$$

where $\Phi^\dagger$ denotes the pseudo-inverse of Φ.

- Activation function: nonlinear, saturation, monotonicity and continuity and smoothness
- Scaling input
- Number of hidden units n_H, roughly, $n/10$
- Initializing weights
- Learning rate, $\eta = 0.1$
- Weight decay
- Number of hidden layers
- Criterion (or objective) function