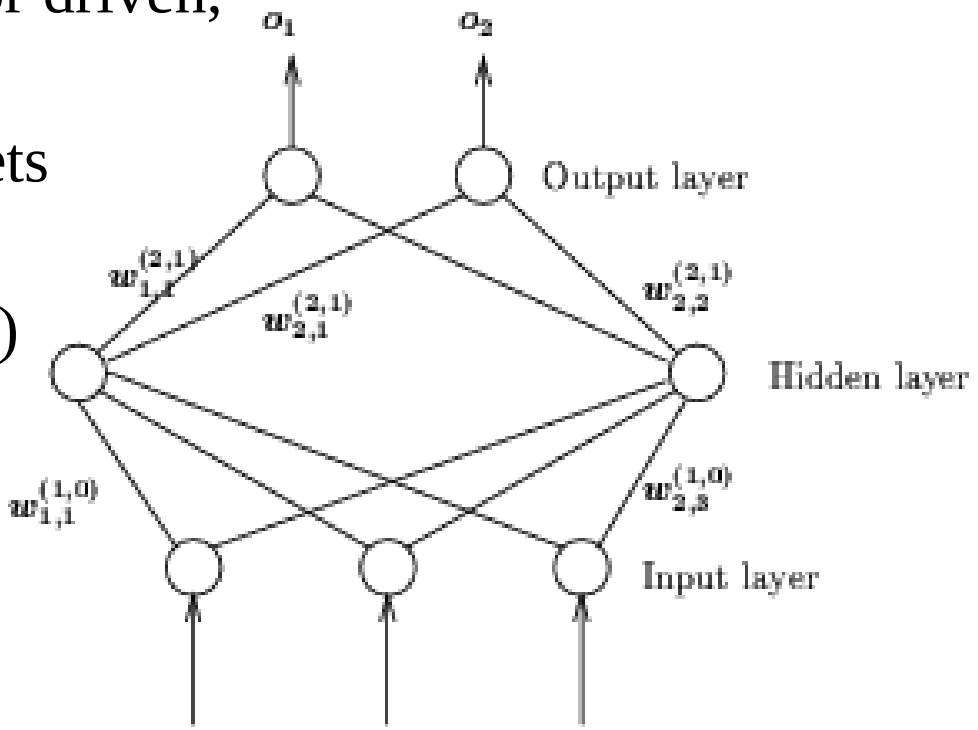
Chapter 3
Supervised learning:
Multilayer Networks I

Backpropagation Learning

- **Architecture**:
  - **Feedforward** network of at least one layer of **non-linear** hidden nodes, e.g., # of layers $L \geq 2$ (not counting the input layer)
  - Node function is differentiable

    most common: **sigmoid function** $\mathcal{S}(net) = \frac{1}{1 + e^{(-net)}}$
- **Learning**: supervised, error driven, generalized delta rule
- Call this type of nets BP nets
- The weight update rule (gradient descent approach)
- Practical considerations
- Variations of BP nets
- Applications



Backpropagation Learning

- **Notations**:
  - Weights: two weight matrices:
    $w^{(1,0)}$ from input layer (0) to hidden layer (1)
    $w^{(2,1)}$ from hidden layer (1) to output layer (2)
    $w_{2,1}^{(1,0)}$ weight from node 1 at layer 0 to node 2 in layer 1
  - Training samples: pair of $\{(x_p, d_p)\, p = 1,..., P\}$
    so it is supervised learning
  - Input pattern: $x_p = (x_{p,1},..., x_{p,n})$
  - Output pattern: $o_p = (o_{p,1},..., o_{p,k})$
  - Desired output: $d_p = (d_{p,1},..., d_{p,k})$
  - Error: $l_{p,j} = o_{p,j} - d_{p,j}$ error for output j when x_p is applied
    sum square error $= \sum_{p=1}^{P}\sum_{j=1}^{K}(l_{p,j})^2$
    This error drives learning (change $w^{(1,0)}$ and $w^{(2,1)}$)

Backpropagation Learning

- **Sigmoid function again**:
  - Differentiable:

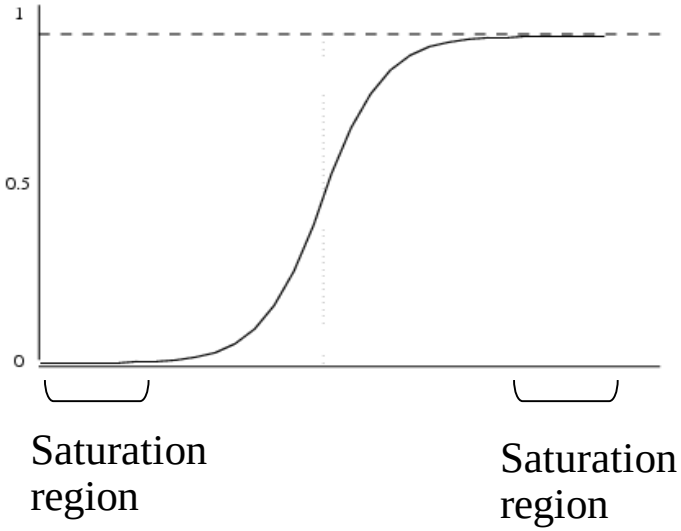
$$S(x) = \frac{1}{1+e^{-x}}$$

$$S'(x) = -\frac{1}{(1+e^{-x})^2} \cdot (1+e^{-x})'$$

$$= -\frac{1}{(1+e^{-x})^2} \cdot (-e^{-x})$$

$$= \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}}$$

$$= S(x)(1-S(x))$$



  - When $|net|$ is sufficiently large, it moves into one of the two saturation regions, behaving like a threshold or ramp function.

- Chain rule of differentiation

$$\text{if } z = f(y), y = g(x), x = h(t) \text{ then } \frac{dz}{dt} = \frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dt} = f'(y)g'(x)h'(t)$$

Backpropagation Learning

- **Forward computing**:
  - Apply an input vector $\boldsymbol{x}$ to input nodes
  - Computing output vector $\boldsymbol{x}^{(1)}$ on hidden layer

$$x_j^{(1)} = S(net_j^{(1)}) = S(\sum_i w_{j,i}^{(1,0)} x_i)$$

  - Computing the output vector $\boldsymbol{o}$ on output layer

$$o_k = S(net_k^{(2)}) = S(\sum_j w_{k,j}^{(2,1)} x_j^{(1)})$$

  - The net is said to be a map from input $\boldsymbol{x}$ to output $\boldsymbol{o}$
- **Objective of learning:**
  - reduce sum square error $\sum_{p=1}^{P}\sum_{j=1}^{K}(l_{p,j})^2$ for the given P training samples as much as possible (to zero if possible)

Backpropagation Learning

- **Idea of BP learning**:
  - Update of weights in $w^{(2,1)}$ (from hidden layer to output layer): delta rule as in a single layer net using sum square error
  - Delta rule is not applicable to updating weights in $w^{(1,0)}$ (from input and hidden layer) because we don't know the desired values for hidden nodes
  - **Solution**: Propagating errors at output nodes down to hidden nodes, these computed errors on hidden nodes drives the update of weights in $w^{(1,0)}$ (again by delta rule), thus called error **BACKPROPAGATION (BP)** learning
  - How to compute errors on hidden nodes is the key
  - Error backpropagation can be continued downward if the net has more than one hidden layer
  - Proposed first by Werbos (1974), current formulation by Rumelhart, Hinton, and Williams (1986)

Backpropagation Learning

- **Generalized delta rule**:
  - Consider sequential learning mode: for a given sample (x_p, d_p)

    $$E = \sum_k (l_{p,k})^2$$

  - Update of weights by gradient descent

    For weight in $w^{(2,1)}$: $\Delta w_{k,j}^{(2,1)} \propto (-\partial E / \partial w_{k,j}^{(2,1)})$

    For weight in $w^{(1,0)}$: $\Delta w_{j,i}^{(1,0)} \propto (-\partial E / \partial w_{j,i}^{(1,0)})$

  - Derivation of update rule for $w^{(2,1)}$:

    since E is a function of $l_k = d_k - o_k$, $d_k - o_k$ is a function $net_k^{(2)}$, and $net_k^{(2)}$ is a function of $w_{k,j}^{(2,1)}$, by chain rule

$$\frac{\partial E}{\partial w_{k,j}^{(2,1)}} = \frac{\partial E}{\partial (d_k - o_k)} \frac{\partial (d_k - o_k)}{\partial net_k^{(2)}} \frac{\partial net_k^{(2)}}{\partial w_{k,j}^{(2,1)}}$$

$$= -2(d_k - o_k) \mathcal{S}'(net_k^{(2)}) x_j^{(1)}.$$

Backpropagation Learning

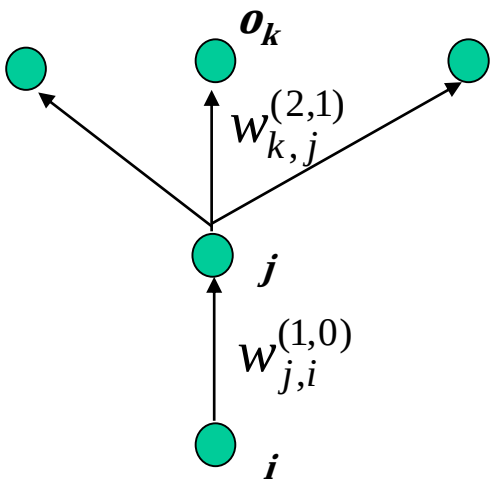
– Derivation of update rule for $w_{j,i}^{(1,0)}$

consider **hidden node *j***:

weight $w_{j,i}^{(1,0)}$ influences $net_j^{(1)}$

it sends $S(net_j^{(1)})$ to all output nodes

$\therefore$ all K terms in E are functions of $w_{j,i}^{(1,0)}$



$$E = \sum_k (d_k - o_k)^2, \quad o_k = S(net_k^{(2)}), \quad net_k^{(2)} = \sum_j x_j^{(1)} w_{k,j}^{(2,1)},$$

$$x_j^{(1)} = S(net_j^{(1)}), \; net_j^{(1)} = \sum_i x_i w_{j,i}^{(1,0)}$$

by chain rule

$$\frac{\partial E}{\partial w_{j,i}^{(1,0)}} = \sum_{k=1}^{K} \left\{ -2(d_k - o_k) S'\left(net_k^{(2)}\right) w_{k,j}^{(2,1)} S'\left(net_j^{(1)}\right) x_i \right\}$$

$\frac{\partial E}{\partial o_k}$ $\frac{\partial S(net_k^{(2)})}{\partial net_k^{(2)}}$ $\frac{\partial net_k^{(2)}}{\partial x_j^{(1)}}$ $\frac{\partial x_j^{(1)}}{\partial net_j^{(1)}}$ $\frac{\partial net_j^{(1)}}{\partial w_{j,i}^{(1)}}$

Backpropagation Learning

– Update rules:

for outer layer weights $w^{(2,1)}$:

$$\frac{\partial E}{\partial w_{k,j}^{(2,1)}} = -2(d_k - o_k)\mathcal{S}'(net_k^{(2)})x_j^{(1)}$$

$$\Delta w_{k,j}^{(2,1)} = \eta \times \delta_k \times x_j^{(1)} \quad \text{where} \quad \delta_k = (d_k - o_k)S'(net_k^{(2)})$$

for inner layer weights $w^{(1,0)}$:

$$\frac{\partial E}{\partial w_{j,i}^{(1,0)}} = \sum_{k=1}^{K} \left\{ -2(d_k - o_k)\mathcal{S}' \left(net_k^{(2)} \right) w_{k,j}^{(2,1)} \mathcal{S}' \left(net_j^{(1)} \right) x_i \right\}$$

$$\Delta w_{j,i}^{(1,0)} = \eta \times \mu_j \times x_i \quad \text{where} \quad \mu_j = (\sum_k \delta_k w_{k,j}^{(2,1)})S'(net_j^{(1)})$$

Weighted sum of errors from output layer

Algorithm Backpropagation;

Start with randomly chosen weights;

while MSE is unsatisfactory

and computational bounds are not exceeded, do

for each input pattern x_p, $1 \leq p \leq P$,

Compute hidden node inputs ($net_{p,j}^{(1)}$);

Compute hidden node outputs ($x_{p,j}^{(1)}$);

Compute inputs to the output nodes ($net_{p,k}^{(2)}$);

Compute the network outputs ($o_{p,k}$);

Modify outer layer weights:

$$\Delta w_{k,j}^{(2,1)} = \eta(d_{p,k} - o_{p,k})\mathcal{S}'(net_{p,k}^{(2)})x_{p,j}^{(1)}$$

Modify weights between input & hidden nodes:

$$\Delta w_{j,i}^{(1,0)} = \eta \sum_k \left((d_{p,k} - o_{p,k})\mathcal{S}'(net_{p,k}^{(2)})w_{k,j}^{(2,1)}\right) \mathcal{S}'(net_{p,j}^{(1)})x_{p,i}$$

end-for

end-while.

Note: if S is a logistic function, then $S'(x) = S(x)(1 - S(x))$

Backpropagation Learning

- **Pattern classification**:
  - Two classes: 1 output node
  - N classes: binary encoding (log N) output nodes

    better using N output nodes, a class is represented as

    (0,..., 0,1, 0,.., 0)
  - With sigmoid function, nodes at output layer will never be 1 or 0, but either $1 - \varepsilon$ or ε.
  - Error reduction becomes slower when moving into saturation regions (when ε is small).
  - For fast learning, for a given error bound ε,

    set error $l_{p,k} = 0$ if $|d_{p,k} - o_{p,k}| \leq \varepsilon$
  - When classifying a input x using a trained BP net, classify it to the k^{th} class if with $d_k > d_l$ for all $l \mathrel{!}= k$

Backpropagation Learning

- **Pattern classification**: an example
  - Classification of myoelectric signals
    - Input pattern: 3 features (NIF, VT, RR), normalized to real values between 0 and 1
    - Output patters: 2 classes: (success, failure)
  - Network structure: 2-5-3
    - 3 input nodes, 2 output nodes,
    - 1 hidden layer of 5 nodes
    - $\eta = 0.95$, $\alpha = 0.4$ (momentum)
  - Error bound $\varepsilon = 0.05$
  - 332 training samples
  - Maximum iteration = 20,000
  - When stopped, 38 patterns remain misclassified

Iteration No.	Fraction of samples misclassified	MSE
100	0.159639	0.213797
200	0.150602	0.189983
300	0.132530	0.172497
400	0.135542	0.170050
500	0.132530	0.168683
600	0.132530	0.168227
700	0.129518	0.167203
800	0.129518	0.167318
900	0.129518	0.167395
1000	0.126506	0.167376
2000	0.123494	0.166275
3000	0.129518	0.165759
4000	0.123494	0.151863
5000	0.123494	0.151121
10000	0.111446	0.149668
11000	0.114458	0.149576
12000	0.114458	0.149664
13000	0.111446	0.146705
14000	0.114458	0.149832
15000	0.111446	0.147453
16000	0.114458	0.149184
17000	0.111446	0.147182
18000	0.114458	0.147353
19000	0.114458	0.147297
20000	0.114458	0.147962

Actual Class in which network places sample:	1	2	3
Desired Target Class			
Class 1	75	5	0
Class 2	3	88	9
Class 3	2	19	131

38 patterns misclassified

Backpropagation Learning

- **Function approximation:**
  - For the given $w = (w^{(1,0)}, w^{(2,1)})$, $o = f(x)$: it realizes a functional mapping from f.
  - Theoretically, feedforward nets with at least one hidden layer of non-linear nodes are able to approximate any L2 functions (all square-integral functions, including almost all commonly used math functions) to any given degree of accuracy, provided there are sufficient many hidden nodes
  - Any L2 function $f(x)$ can be approximated by a Fourier series

    $$\hat{f}_N(x) = \sum_{k_1=-N}^{N} \cdots \sum_{k_n=-N}^{N} c_k \, e^{2\pi\sqrt{-1}(k \cdot x)}$$

    The MSE converges to 0 when N $\rightarrow \infty$

    $$\lim_{N \longrightarrow \infty} \int_{[0,1]^n} |f(x) - f_N(x)|^2 \, dx = 0$$

    It has been shown the Fourier series approximation can be realized with Feedforward net with one hidden layer of cosine node function
  - Reading for grad students (Sec. 3.5) for more discussions

Strengths of BP Learning

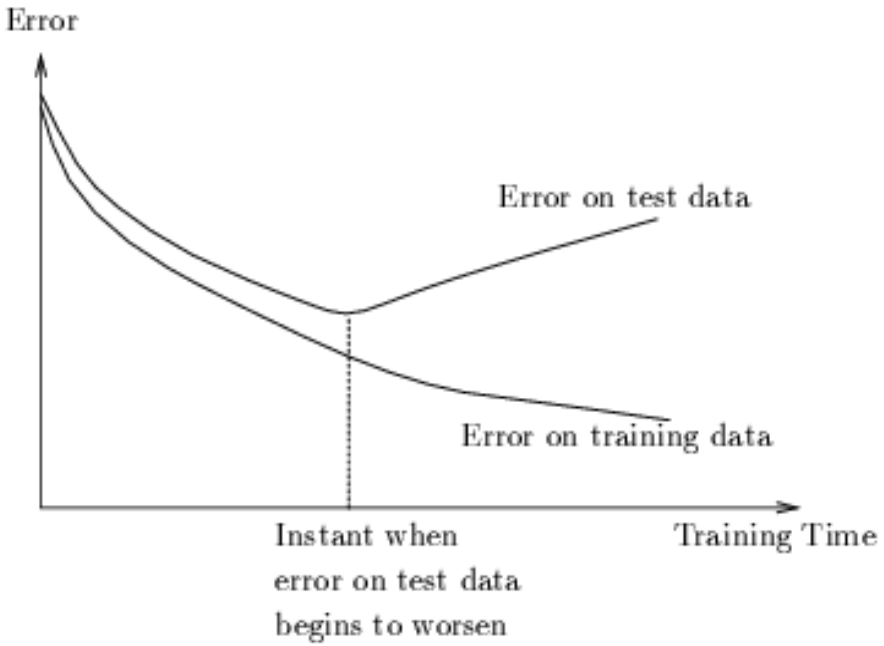
- **Great representation power**
  - Any L2 function can be represented by a BP net
  - Many such functions can be approximated by BP learning (gradient descent approach)
- **Wide applicability of BP learning**
  - Only requires that a good set of training samples is available
  - Does not require substantial prior knowledge or deep understanding of the domain itself (ill structured problems)
  - Tolerates noise and missing data in training samples (graceful degrading)
- **Easy to implement** the core of the learning algorithm
- **Good generalization power**
  - Often produce accurate results for inputs outside the training set

Deficiencies of BP Learning

- Learning often takes a **long time** to converge
  - Complex functions often need hundreds or thousands of epochs
- The net is essentially a **black box**
  - It may provide a desired mapping between input and output vectors (***x, o***) but does not have the information of why a particular ***x*** is mapped to a particular ***o.***
  - It thus cannot provide an intuitive (e.g., causal) explanation for the computed result.
  - This is because the hidden nodes and the learned weights do not have clear semantics.
    - What can be learned are operational parameters, not general, abstract knowledge of a domain
  - Unlike many statistical methods, there is no theoretically well-founded way to **assess the quality** of BP learning
    - What is the confidence level one can have for a trained BP net, with the final E (which may or may not be close to zero)?
    - What is the confidence level of *o* computed from input *x* using such net?

- Problem with gradient descent approach
  - only guarantees to reduce the total error to a **local minimum**. (***E*** may not be reduced to zero)
  - Cannot escape from the local minimum error state
  - **Not every function that is representable can be learned**
  - How bad: depends on the shape of the error surface. Too many valleys/wells will make it easy to be trapped in local minima
  - Possible remedies:
    - Try nets with different # of hidden layers and hidden nodes (they may lead to different error surfaces, some might be better than others)
    - Try different initial weights (different starting points on the surface)
    - Forced escape from local minima by random perturbation (e.g., simulated annealing)

- **Generalization** is not guaranteed even if the error is reduced to 0
  - Over-fitting/over-training problem: trained net fits the training samples perfectly (*E* reduced to 0) but it does not give accurate outputs for inputs not in the training set
  - Possible remedies:
    - More and better samples
    - Using smaller net if possible
    - Using larger error bound (forced early termination)
    - Introducing noise into samples
      - modify $(x_1,\ldots, x_n)$ to $(x_1\alpha_1,\ldots, x_n\alpha_n)$ where α_n are small random displacements
    - Cross-Validation
      - leave some (~10%) samples as test data (not used for weight update)
      - periodically check error on test data
      - Learning stops when error on test data starts to increase



- **Network paralysis** with sigmoid activation function
  - Saturation regions:
    $S(x) = 1/(1+e^{-x})$, its derivative $S'(x) = S(x)(1-S(x)) \to 0$ when $x \to \pm\infty$.
    When x falls in a saturation region, $S(x)$ hardly changes its value regardless how fast the magnitude of x increases
  - Input to an node may fall into a saturation region when some of its incoming weights become very large during learning. Consequently, weights stop to change no matter how hard you try.

$$\frac{\partial E}{\partial w_{k,j}^{(2,1)}} = -2(d_k - o_k)\mathcal{S}'(net_k^{(2)})x_j^{(1)}$$

  - Possible remedies:
    - Use non-saturating activation functions
    - Periodically normalize all weights

$$w_{k,j} := w_{k,j} / \|w_{.k}\|_2$$

- The learning (accuracy, speed, and generalization) is highly dependent of a set of learning **parameters**
  - Initial weights, learning rate, # of hidden layers and # of nodes...
  - Most of them can only be determined **empirically** (via experiments)

Practical Considerations

- A good BP net requires more than the core of the learning algorithms. Many parameters must be carefully selected to ensure a good performance.
- Although the deficiencies of BP nets cannot be completely cured, some of them can be eased by some practical means.
- **Initial weights (and biases)**
  - Random, [-0.05, 0.05], [-0.1, 0.1], [-1, 1]
  - Normalize weights for hidden layer ($w^{(1,0)}$) (Nguyen-Widrow)
    - Random assign initial weights for all hidden nodes
    - For each hidden node *j*, normalize its weight by

      $$w_{j,i}^{(1,0)} = \beta \cdot w_{j,i}^{(1,0)} / \left\| w_j^{(1,0)} \right\|_2 \quad \text{where } \beta = 0.7\sqrt[n]{m}$$

      $m = \#$ of hiddent nodes, $n = \#$ of input nodes

      $\left\| w_j^{(1,0)} \right\|_2 = \beta$ after normalization
    - Avoid bias in weight initialization:

- **Training samples**:
  - Quality and quantity of training samples often determines the quality of learning results
  - Samples must collectively represent well the problem space
    - Random sampling
    - Proportional sampling (with prior knowledge of the problem space)
  - # of training patterns needed: There is no theoretically idea number.
    - Baum and Haussler (1989): P = W/e, where

      W: total # of weights to be trained (depends on net structure)

      e: acceptable classification error rate

      If the net can be trained to correctly classify (1 – e/2)P of the P training samples, then classification accuracy of this net is 1 – e for input patterns drawn from the same sample space

      Example: W = 27, e = 0.05, P = 540. If we can successfully train the network to correctly classify (1 – 0.05/2)*540 = 526 of the samples, the net will work correctly 95% of time with other input.

- **How many hidden layers and hidden nodes per layer**:
  - Theoretically, one hidden layer (possibly with many hidden nodes) is sufficient for any L2 functions
  - There is no theoretical results on minimum necessary # of hidden nodes
  - Practical rule of thumb:
    - n = # of input nodes; m = # of hidden nodes
    - For binary/bipolar data: m = 2n
    - For real data: m >> 2n
  - Multiple hidden layers with fewer nodes may be trained faster for similar quality in some applications

- **Data representation**:
  - Binary vs bipolar
    - Bipolar representation uses training samples more efficiently

      $\Delta w_{j,i}^{(1,0)} = \eta \cdot \mu_j \cdot x_i \qquad \Delta w_{k,j}^{(2,1)} = \eta \cdot \delta_k \cdot x_j^{(1)}$

      no learning will occur when $x_i = 0$ or $x_j^{(1)} = 0$ with binary rep.
    - # of patterns can be represented with n input nodes:

      binary: 2^n

      bipolar: 2^(n-1) if no biases used, this is due to (anti) symmetry

      (if output for input x is o, output for input $-x$ will be $-o$)
  - Real value data
    - Input nodes: real value nodes (may subject to normalization)
    - Hidden nodes are sigmoid
    - Node function for output nodes: often linear (even identity)

      e.g., $o_k = \sum w_{k,j}^{(2,1)} x_j^{(1)}$
    - Training may be much slower than with binary/bipolar data (some use binary encoding of real values)

Variations of BP nets

- **Adding momentum term** (to speedup learning)
  - Weights update at time t+1 contains the momentum of the previous updates, e.g.,

    $$\Delta w_{k,j}(t+1) = \eta \cdot \delta_k(t) \cdot x_j + \alpha \cdot \Delta w_{k,j}(t)$$

    then $$\Delta w_{k,j}(t+1) = \sum_{s=1}^{t} \alpha^{t-s} \eta \cdot \delta_k(s) \cdot x_j(s)$$

    an exponentially weighted sum of all previous updates
  - Avoid sudden change of directions of weight update (smoothing the learning process)
  - Error is no longer monotonically decreasing
- **Batch mode** of weight updates
  - Weight update once per each epoch (cumulated over all P samples)
  - Smoothing the training sample outliers
  - Learning independent of the order of sample presentations
  - Usually slower than in sequential mode

- **Variations on learning rate η**
  - Fixed rate much smaller than 1
  - Start with large η, gradually decrease its value
  - Start with a small η, steadily double it until MSE start to increase
  - Give known underrepresented samples higher rates
  - Find the maximum safe step size at each stage of learning (to avoid overshoot the minimum E when increasing η**)**
  - **Adaptive learning rate** (delta-bar-delta method)
    - Each weight $w_{k,j}$ has its own rate $\eta_{k,j}$
    - If $\Delta w_{k,j}$ remains in the same direction, increase $\eta_{k,j}$ (E has a smooth curve in the vicinity of current w)
    - If $\Delta w_{k,j}$ changes the direction, decrease $\eta_{k,j}$ (E has a rough curve in the vicinity of current w)

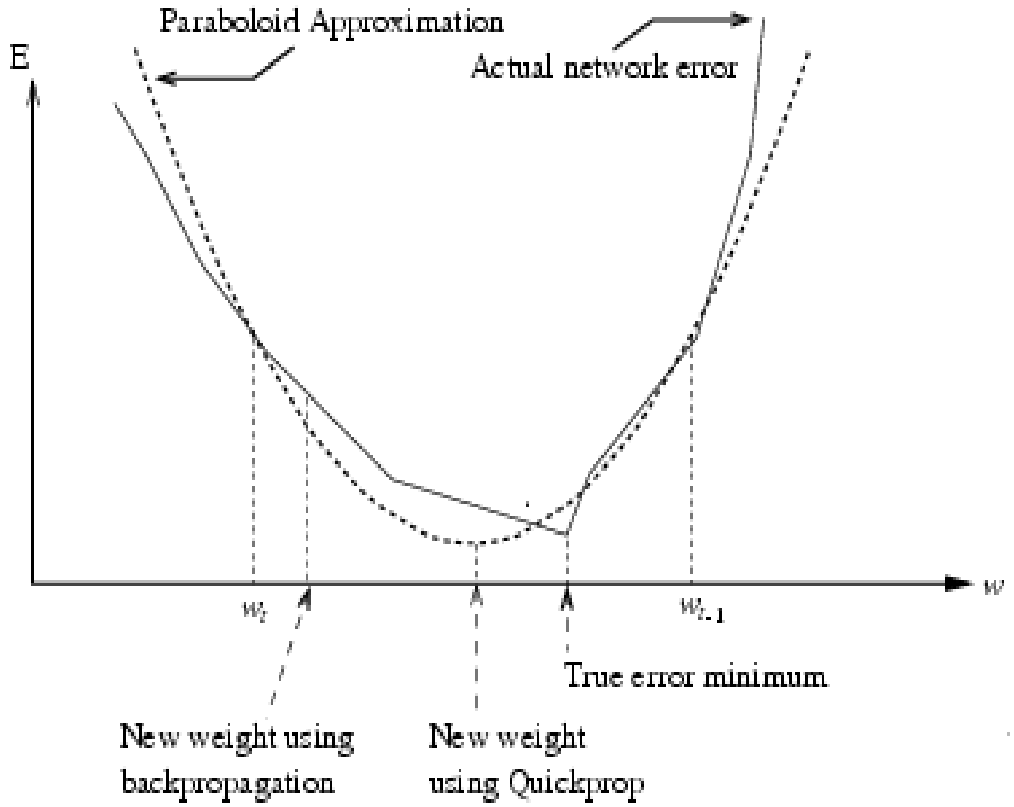
- Experimental comparison
  - Training for XOR problem (batch mode)
  - 25 simulations with random initial weights: success if E averaged over 50 consecutive epochs is less than 0.04
  - results

method	**simulations**	**success**	**Mean epochs**
BP	25	24	16,859.8
BP with momentum	25	25	2,056.3
BP with delta-bar-delta	25	22	447.3

- **Quickprop**
  - If E is of paraboloid shape

    if E does not change sign from t-1 to t nor decreased in magnitude, then its (local) minimum occurs at

$$w(t+1) = w(t) + E'(t)\Delta w(t-1)/(E'(t-1) - E'(t))$$



- **Other node functions**
  - Change the range of the logistic function from (0,1) to (a, b)

In particular, for bipolar sigmoid function (-1,1), we have

$$g(x) = 2f(x) - 1, \text{ and } g'(x) = \frac{1}{2}(1 + g(x))(1 - g(x))$$

  - Change the slope of the logistic function

$$f(x) = 1/(1 + e^{-\sigma x}),$$
$$f'(x) = \sigma f(x)(1 - f(x))$$

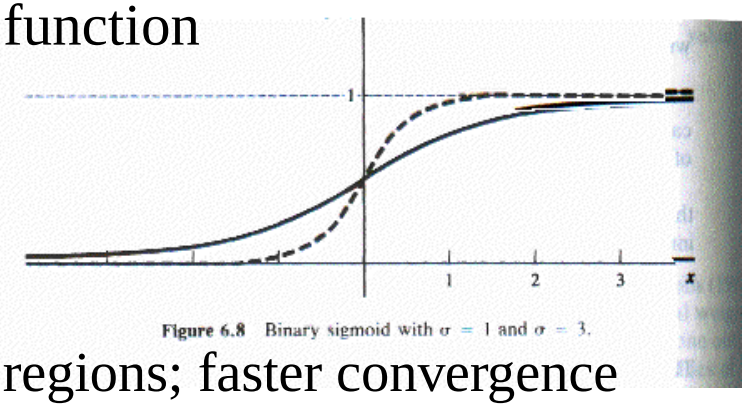


Figure 6.8 Binary sigmoid with σ = 1 and σ = 3.

    - Larger slope: quicker to move to saturation regions; faster convergence
    - Smaller slope: slow to move to saturation regions, allows refined weight adjustment
    - σ thus has a effect similar to the learning rate η (but more drastic)
  - Adaptive slope (**each node has a learned slope**)

– Another sigmoid function with slower saturation speed

$$\boldsymbol{f}(\boldsymbol{x}) = \frac{2}{\boldsymbol{\pi}}\arctan(\boldsymbol{x}),\ \boldsymbol{f}'(\boldsymbol{x}) = \frac{2}{\boldsymbol{\pi}}\frac{1}{1+\boldsymbol{x}^2}$$

For large $|x|$, $\dfrac{1}{1+x^2}$ is much larger than $\dfrac{1}{(1+e^{-x})(1+e^{x})}$

the derivative of logistic function

– A non-saturating function (also differentiable)

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

$$f'(x) = \begin{cases} \dfrac{1}{1+x} & \text{if } x \geq 0 \\ \dfrac{1}{1-x} & \text{if } x < 0 \end{cases}, \text{ then, } f'(x) \to 0 \text{ when } |x| \to \infty$$

Applications of BP Nets

- **A simple example: Learning XOR**
  - Initial weights and other parameters
    - **weights**: random numbers in [-0.5, 0.5]
    - **hidden** nodes: single layer of 4 nodes (A 2-4-1 net)
    - **biases** used;
    - **learning rate**: 0.02
  - Variations tested
    - binary vs. bipolar representation
    - different stop criteria (targets with ± 1.0 and with ± 0.8)
    - normalizing initial weights (Nguyen-Widrow)
  - Bipolar is faster than binary
    - convergence: ~3000 epochs for binary, ~400 for bipolar
    - Why?

Binary nodes

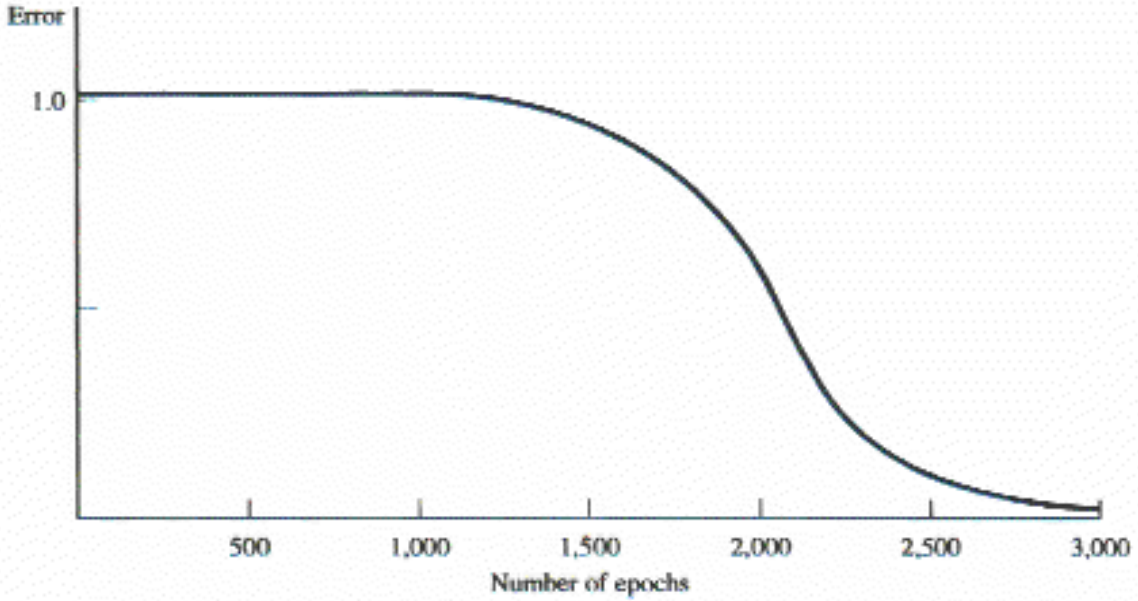


Figure 6.4 Total squared error for binary representation of Xor problem.

Bipolar nodes

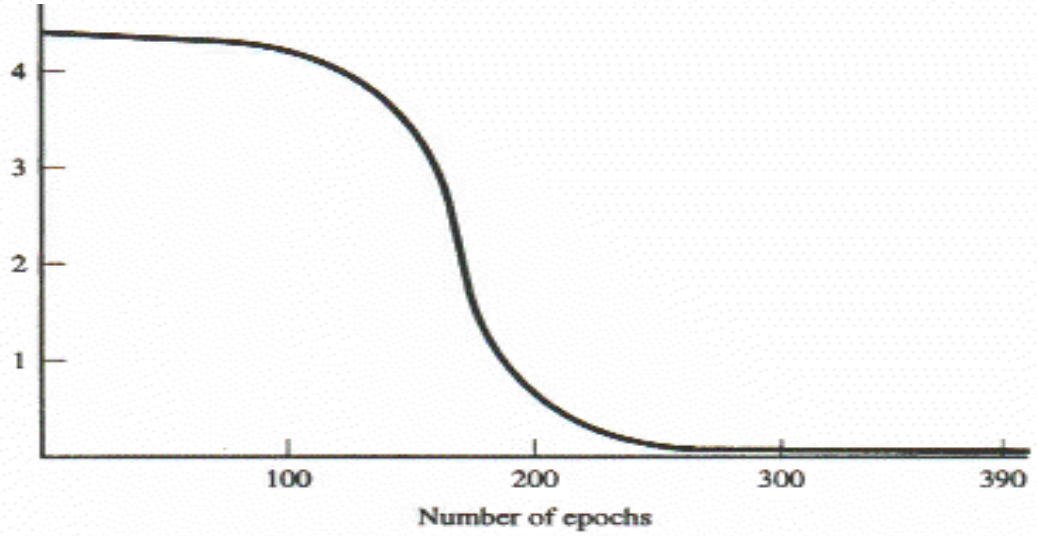
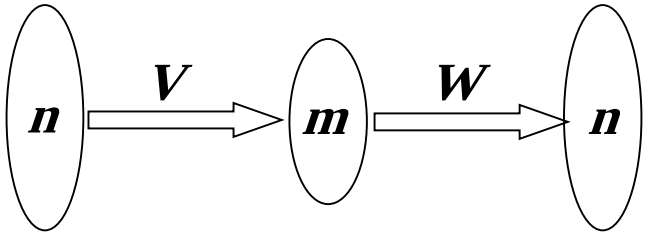


Figure 6.5 Total squared error for bipolar representation of Xor problem.

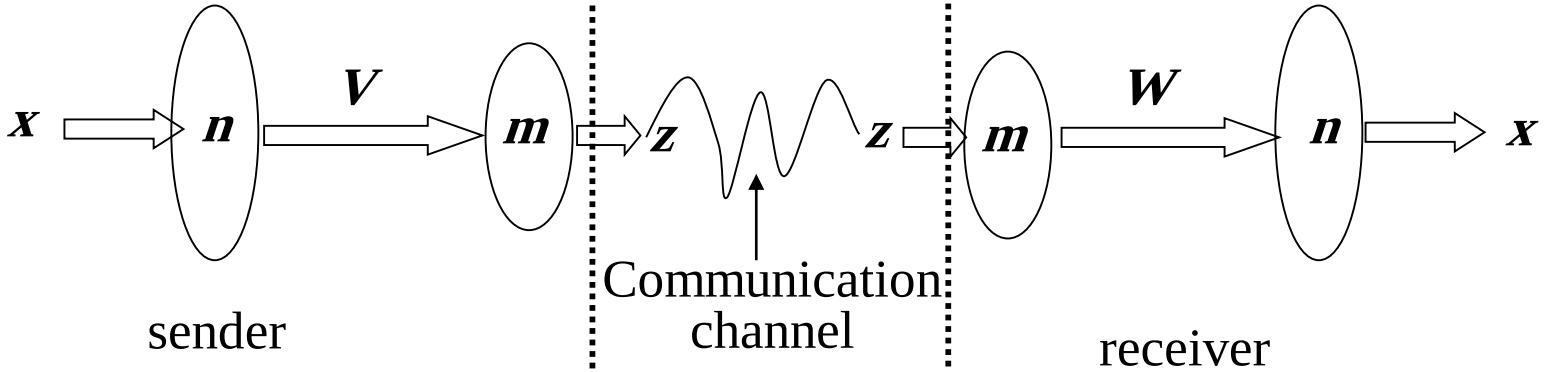
- Relaxing acceptable error range **may** speed up convergence
  - ± 1.0 is an asymptotic limits of sigmoid function,
  - When an output approaches ± 1.0, it falls in a saturation region
  - Use $\pm \boldsymbol{a}$ where $0 < \boldsymbol{a} < 1.0$ (e.g., ± 0.8)
- Normalizing initial weights **may** also help

	Random	**Nguyen-Widrow**
Binary	2,891	1,935
Bipolar	387	224
Bipolar with targets $= \pm 0.8$	264	127

- **Data compression**
  - **Autoassociation** of patterns (vectors) with themselves using a small number of hidden nodes:
    - **training samples:**: $\boldsymbol{x}$:$\boldsymbol{x}$ ($\boldsymbol{x}$ has dimension n)
      hidden nodes: $m < n$ (A n-m-n net)



    - If training is successful, applying any vector $\boldsymbol{x}$ on input nodes will generate the same $\boldsymbol{x}$ on output nodes
    - Pattern $\boldsymbol{z}$ on hidden layer becomes a compressed representation of $\boldsymbol{x}$ (with smaller dimension $m < n$)
    - Application: reducing transmission cost



- **Example:** compressing character bitmaps.
  - Each character is represented by a 7 by 9 pixel bitmap, or a binary vector of dimension 63
  - 10 characters (A – J) are used in experiment
  - Error range:

    tight: 0.1 (off: 0 – 0.1; on: 0.9 – 1.0)
    loose: 0.2 (off: 0 – 0.2; on: 0.8 – 1.0)
  - Relationship between # hidden nodes, error range, and convergence rate
    - relaxing error range may speed up
    - increasing # hidden nodes (to a point) may speed up

    error range: 0.1 hidden nodes: 10 # epochs 400+
    error range: 0.2 hidden nodes: 10 # epochs 200+
    error range: 0.1 hidden nodes: 20 # epochs 180+
    error range: 0.2 hidden nodes: 20 # epochs 90+
    no noticeable speed up when # hidden nodes increases to beyond 22

- **Other applications**.
  - Medical diagnosis
    - Input: manifestation (symptoms, lab tests, etc.)
      Output: possible disease(s)
    - Problems:
      - no causal relations can be established
      - hard to determine what should be included as inputs
    - Currently focus on more restricted diagnostic tasks
      - e.g., predict prostate cancer or hepatitis B based on standard blood test
  - Process control
    - Input: environmental parameters
      Output: control parameters
    - Learn ill-structured control functions

- Stock market forecasting
  - Input: financial factors (CPI, interest rate, etc.) and stock quotes of previous days (weeks)

    Output: forecast of stock prices or stock indices (e.g., S&P 500)
  - Training samples: stock market data of past few years
- Consumer credit evaluation
  - Input: personal financial information (income, debt, payment history, etc.)
  - Output: credit rating
- And many more
- Key for successful application
  - Careful design of input vector (including all **important** features): some domain knowledge
  - Obtain good training samples: time and other cost

Summary of BP Nets

- **Architecture**
  - Multi-layer, feed-forward (full connection between nodes in adjacent layers, no connection within a layer)
  - One or more hidden layers with non-linear activation function (most commonly used are sigmoid functions)
- **BP learning algorithm**
  - Supervised learning (samples (x_p, d_p))
  - Approach: gradient descent to reduce the total error (why it is also called generalized delta rule)
  - Error terms at output nodes
    error terms at hidden nodes (why it is called error BP)
  - Ways to speed up the learning process
    - Adding momentum terms
    - Adaptive learning rate (delta-bar-delta)
    - Quickprop
  - Generalization (cross-validation test)

- **Strengths of BP learning**
  - Great representation power
  - Wide practical applicability
  - Easy to implement
  - Good generalization power
- **Problems of BP learning**
  - Learning often takes a long time to converge
  - The net is essentially a black box
  - Gradient descent approach only guarantees a local minimum error
  - Not every function that is representable can be learned
  - Generalization is not guaranteed even if the error is reduced to zero
  - No well-founded way to assess the quality of BP learning
  - Network paralysis may occur (learning is stopped)
  - Selection of learning parameters can only be done by trial-and-error
  - BP learning is non-incremental (to include new training samples, the network must be re-trained with all old and new samples)