Training of Neural Networks

Notation:

x: input of the original modeling problem or the neural network

y: output of the original modeling problem or the neural network

w: internal weights/parameters of the neural network

m: number of outputs of the model

$y = f(x , w)$: neural network model

d: data for y (e.g., trainining data)

Define Model Input-Output

Define model input-output (*x*, *y*), for example,

***x*: physical/geometrical parameters of the component**
y: S-parameters of the component

Data Generation:

(a)**Generate (*x*,*y*) samples: (x_k, y_k), k = 1, 2, …, P, such that the finished NN best (accurately) represent original *x*~*y* problem**

(b)**Data generator**

- **Measurement : for each given x_k , measure values of y_k, k=1,2,…, p**
- **Simulation: for each given x_k , use a simulator to calculate y_k, k=1,2,…, p**

Comparison of Neural Network Based Microwave Model Development Using Data from Two Types of Data Generators

Basis of Comparison	Neural Model Development Using Measurement Data	Neural Model Development Using Simulation Data
Availability of Problem Theory-Equations	Model can be developed even if the theory-equations are not known, or difficult to implement in CAD.	Model can be developed only for the problems that have theory that is implemented in a simulator.
Assumptions	No assumptions are involved and the model could include all the effects, e.g., 3D-fullwave effects, fringing effects etc.	Often involves assumptions and the model will be limited by the assumptions made by the simulator, e.g., 2.5D EM.
Input Parameter Sweep	Data generation could either be expensive or infeasible, if a geometrical parameter, e.g., transistor gate-length needs to be sampled/changed.	Relatively easier to sweep any parameter in the simulator, because the changes are numerical and not physical/manual.

Comparison of Neural Network Based Microwave Model Development Using Data from Two Types of Data Generators (continued)

Basis of Comparison	Neural Model Development Using Measurement Data	Neural Model Development Using Simulation Data
Sources of Small and Large/Gross Errors	Equipment limitations and tolerances.	Accuracy limitations and non-convergence of simulations.
Feasibility of Getting Desired Output	Development of models is possible for measurable responses only. For example, drain charge of an FET may not be easy to measure.	Any response can be modeled as long as it can be computed by the simulator.

Data Generation:

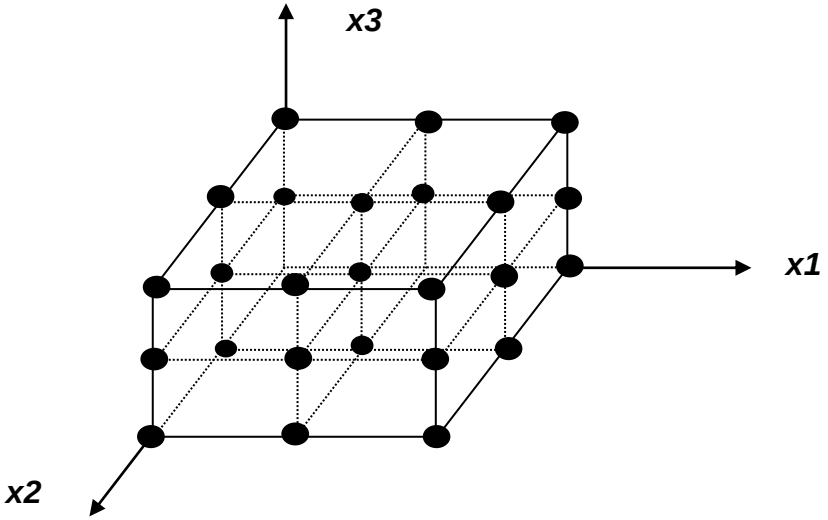
(c) **Range of x to be sampled**

- **For Testing Data and Validation data:**
  $x_{min} \leftrightarrow x_{max}$: Should represent the user-intended range in which the NN is to be used by user

- **For Training Data:**
  Default range of x samples should be equal to user-intended range, or if feasible, slightly beyond the user intended range

Data Generation

- where data should be sampled



(d) Distribution of *x* samples

- **Uniform grid distribution**
- **Non-uniform grid distribution**
- **Design of Experiments (DOE) methodology**
  - **central-composite design**
  - **2^n factorial design**
- **Star distribution**
- **Random distribution**

Data Generation (continued):

(e) Number of samples *P* -- Theoretical factor:

- **For grid distribution case: Shannon's Theorem**
- **For random distribution case: statistical confidence**

Input / Output Scaling

The orders of magnitude of various *x* and *d* values in microwave applications can be very different from one another.

Scaling of training data is desirable for efficient neural network training

The data can be scaled such that various *x* (or *d*) have similar order of magnitude

Input / Output Scaling:

Notation:

x and y -- Original x and y

$\tilde{x}$ and $\tilde{y}$ -- Scaled x and y

x_{max}, x_{min} -- Obtained from data

$\tilde{x}_{max}, \tilde{x}_{min}$ -- Dictated by NN trainer

- **Linear scale**

  Scale formula: $$\tilde{x} = \tilde{x}_{min} + \frac{x - x_{min}}{x_{max} - x_{min}}(\tilde{x}_{max} - \tilde{x}_{min})$$

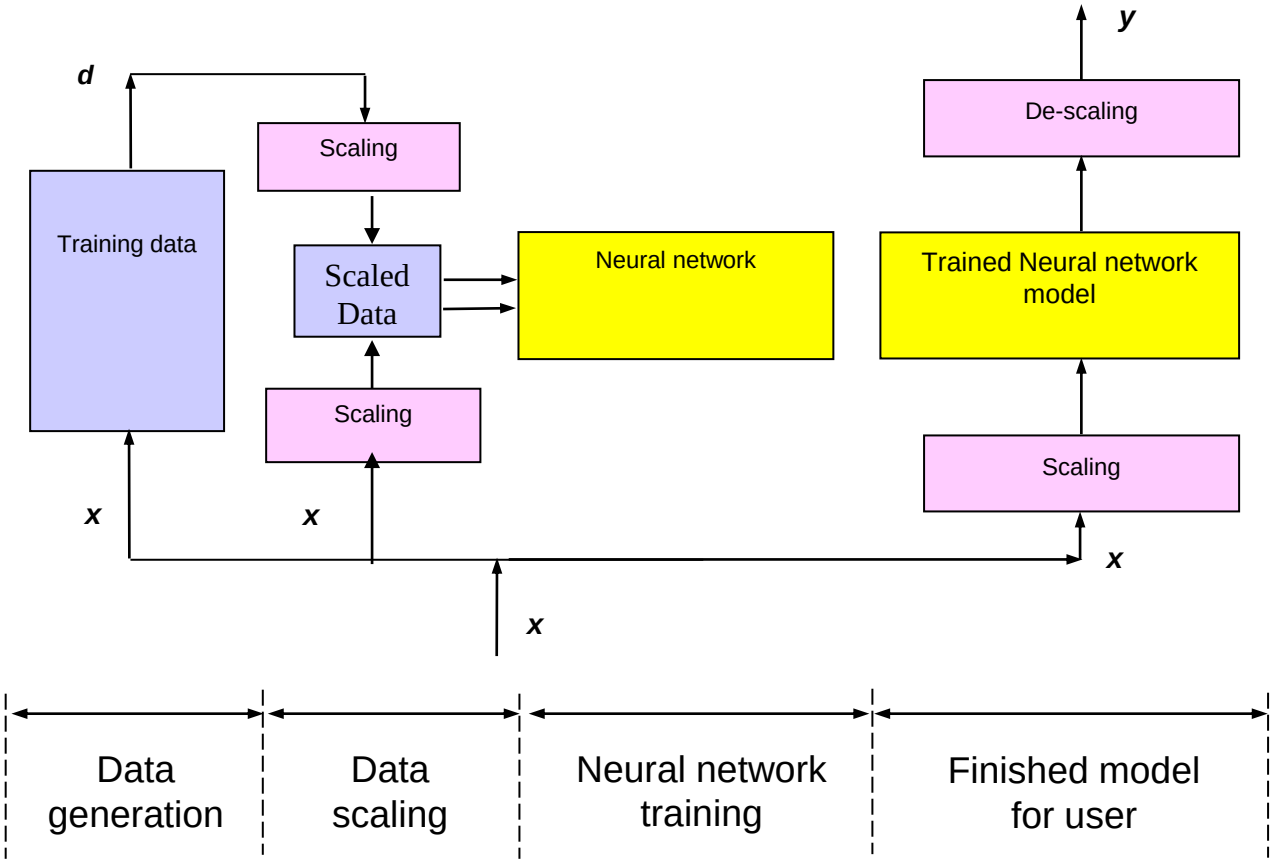
  De-scaled formula: $$x = x_{min} + \frac{\tilde{x} - \tilde{x}_{min}}{\tilde{x}_{max} - \tilde{x}_{min}}(x_{max} - x_{min})$$

- **Log scale**

  Scale formula: $$\tilde{x} = \ln(x - x_{min})$$

  De-scale formula: $$x = x_{min} + e^{\tilde{x}}$$

Illustration of Data Scaling



Divide Data into Training Set, Validation Set and Testing Set

Notation:

P – **total number of data samples generated**

D – **Set for all data,** $D = \{1, 2, \ldots, P\}$

T_r – **Training data set**

V -- **Validation data set**

T_e -- **Test data set**

Ideally: Each data set (T_r **,** V**,** T_e**) should be an adequate representation of original** $y = f(x)$ **problem in the entire** $x_{\min} \sim x_{\max}$ **range. Three sets have no overlap.**

Divide Data into Training Set, Validation Set and Testing Set

Case 1: When original data is quite sufficient, split *D* into non-overlapping sets

Case 2: When data is very limited, duplicate *D*, such that

$$T_r = V = T_e = D$$

Case 3: Split data *D* into 2 sets.

Training / Validation and Testing

Training error:

$$E_{Tr}(w) = \left[\frac{1}{size(Tr)\cdot m} \sum_{k\in Tr} \sum_{j=1}^{m} \left| \frac{y_j(x_k, w) - d_{jk}}{y_{max\,j} - y_{min\,j}} \right|^q \right]^{1/q}$$

Validation error:

$$E_V(w) = \left[\frac{1}{size(V)\cdot m} \sum_{k\in V} \sum_{j=1}^{m} \left| \frac{y_j(x_k, w) - d_{jk}}{y_{max\,j} - y_{min\,j}} \right|^q \right]^{1/q}$$

Test error:

$$E_{T_e}(w) = \left[\frac{1}{size(T_e)\cdot m} \sum_{k\in T_e} \sum_{j=1}^{m} \left| \frac{y_j(x_k, w) - d_{jk}}{y_{max\,j} - y_{min\,j}} \right|^q \right]^{1/q}$$

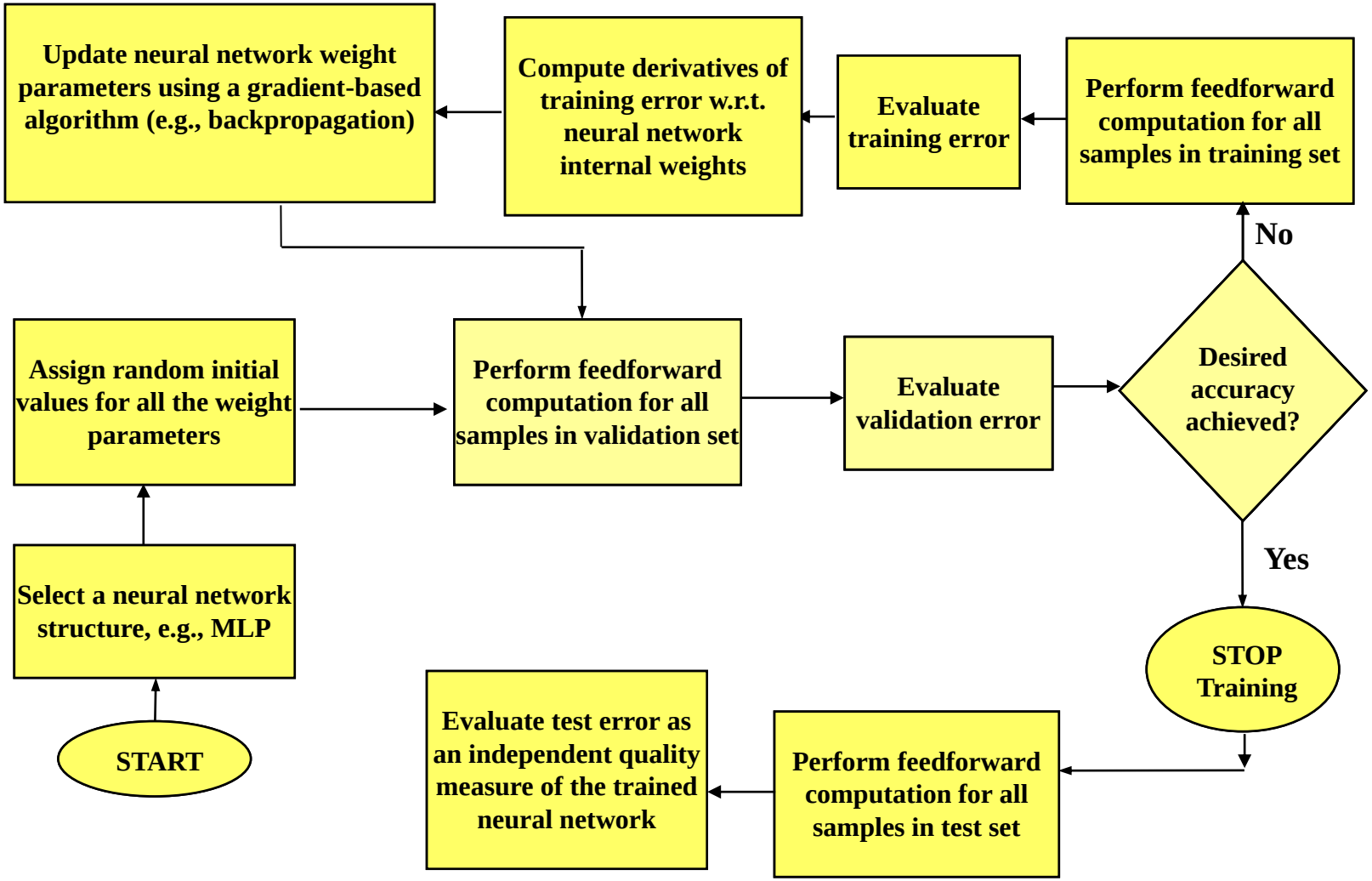
Where to Use Each Error Criteria

Training error: The training error $E_{Tr}(w)$ and its derivative $\frac{\partial E_{T_r}}{\partial w}$ are used to determine how to update w during training

Validation error: The validation error $E_V(w)$ is used as a stopping criteria during training, i.e., to determine if training is sufficient.

Test error: The test error $E_{Te}(w)$ is used after training has finished to provide a final assessment of the quality of the trained neural network. Test error is not involved during training.

Flow-chart Showing Neural Network Training, Validation and Testing



Initial Value of NN Weights Before Training

MLP: small random values

RBF/Wavelet: estimate center & width of RBF
or translation & dilation of Wavelet

Knowledge Based NN: use physical/electrical
experience

Overlearning

Definition (strict):

Math $\begin{cases} E_{T_r} \approx 0 \\ E_V >> E_{T_r} \end{cases}$

Observation: NN memorized training data, but can not generalize well

Possible reasons:

a) Too many hidden neurons

b) Not enough training data

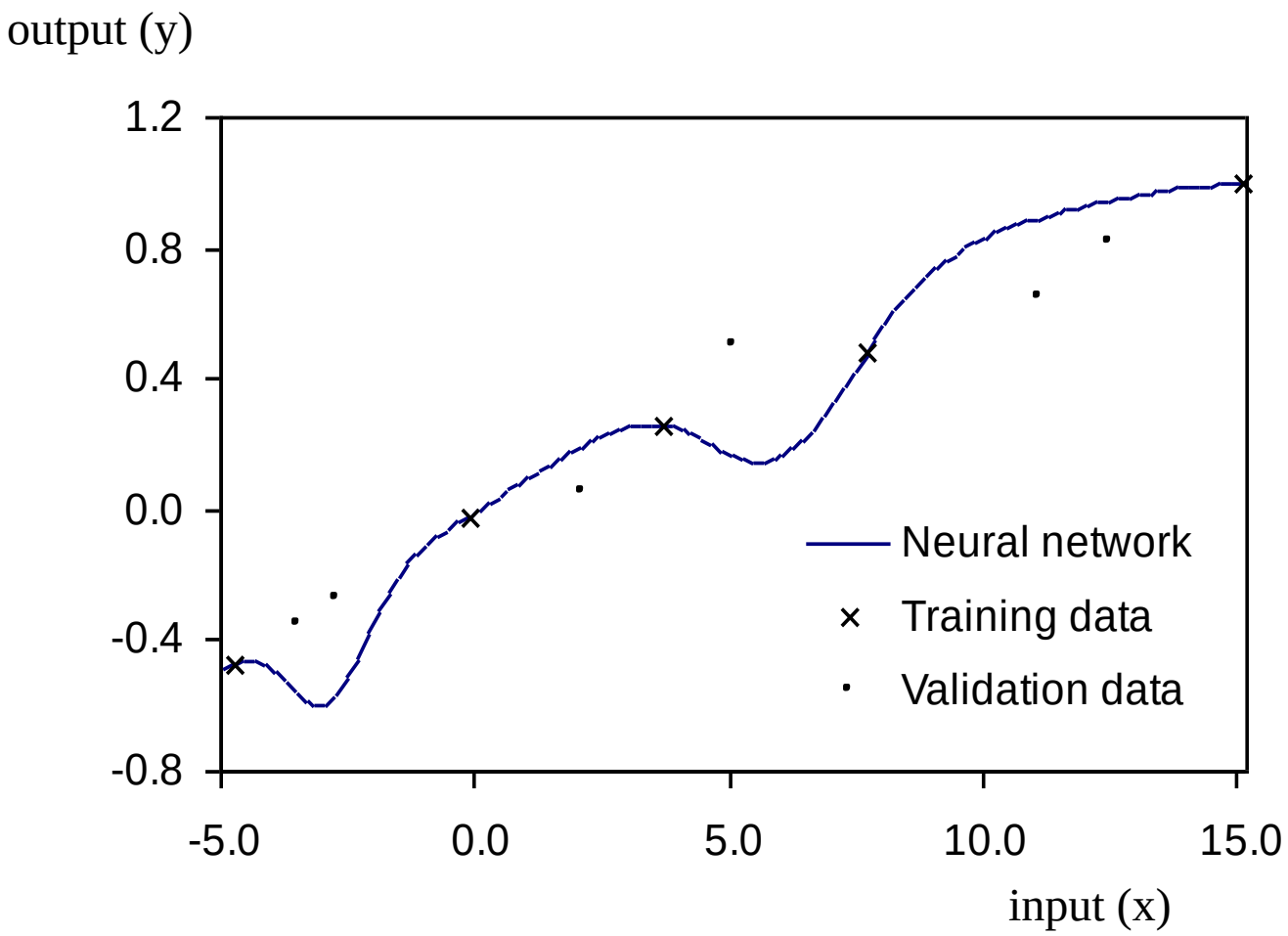
Actions:

a) Add training data

b) Delete hidden neurons

c) Backup/retrieve previous solution

Neural Network Over-Learning



Underlearning

Definition (strict):

Math $E_{T_r} >> 0$

Observation: NN can not even represent the problem at training points

Possible reasons:

a) Not enough hidden neurons

b) Training stuck at local solution

c) Not enough training

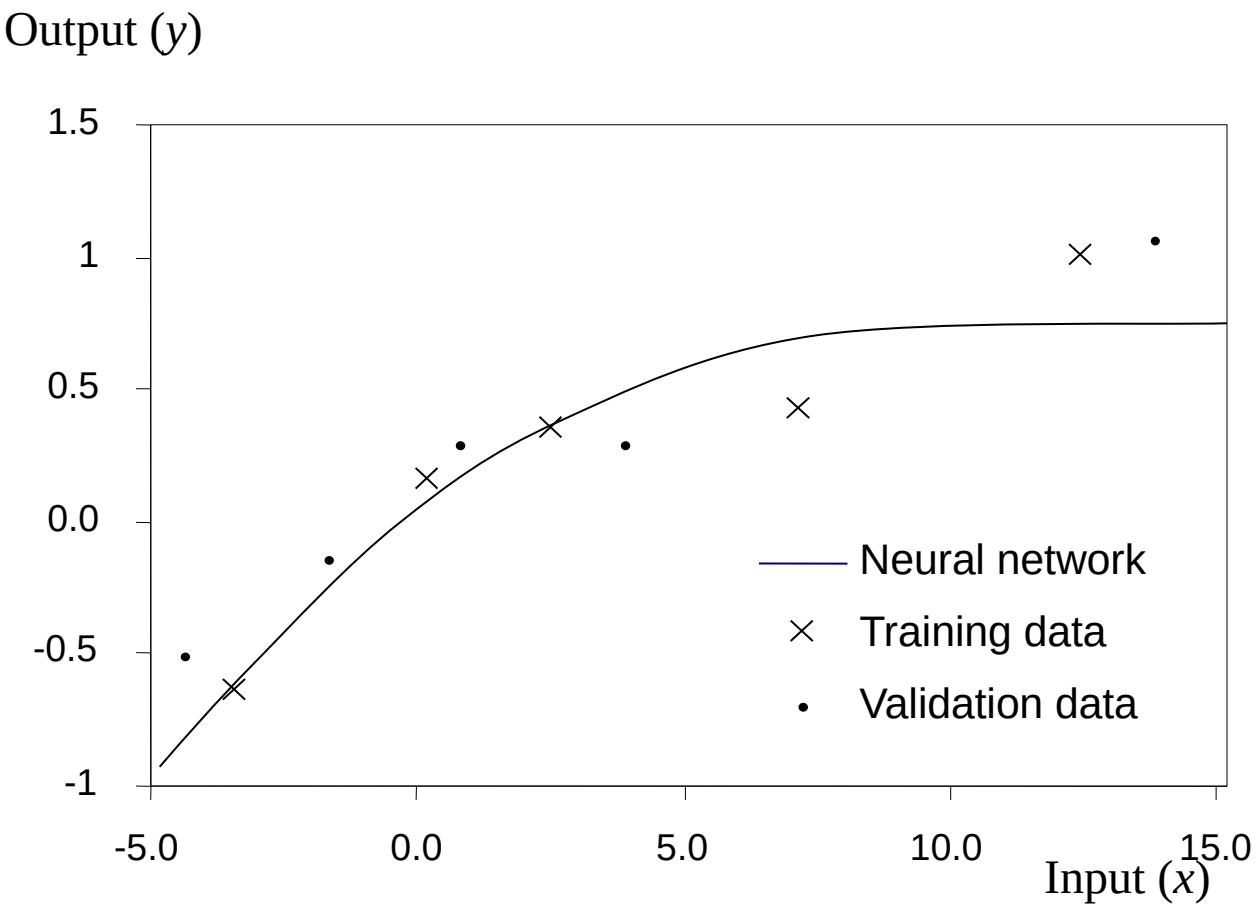
Actions:

a) Add hidden neurons

b) More training

c) Perturb solution, then train

Neural Network Under-Learning



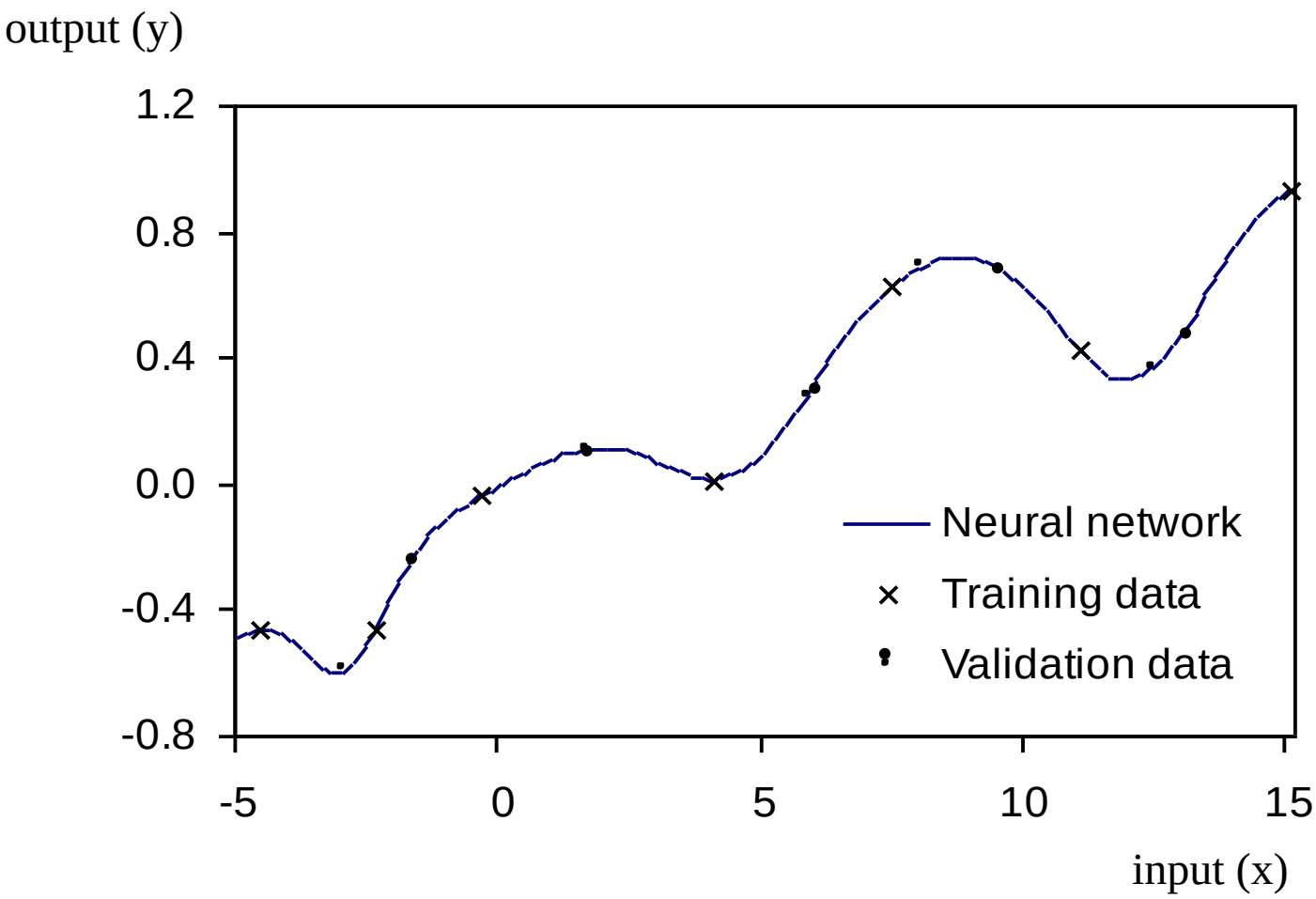
Perfect Learning:

Definition (strict):

Math $E_{T_r} \approx E_V \approx 0$

Observation: **generalized well**

Perfect Learning of Neural Networks



Types of Training

- **Sample-by-sample (or online) training: ANN weights are updated every time a training sample is presented to the network, i.e., weight update is based on training error from that sample**

- **Batch-mode (or offline) training: ANN weights are updated after each epoch, i.e., weight update is based on training error from all the samples in training data set**

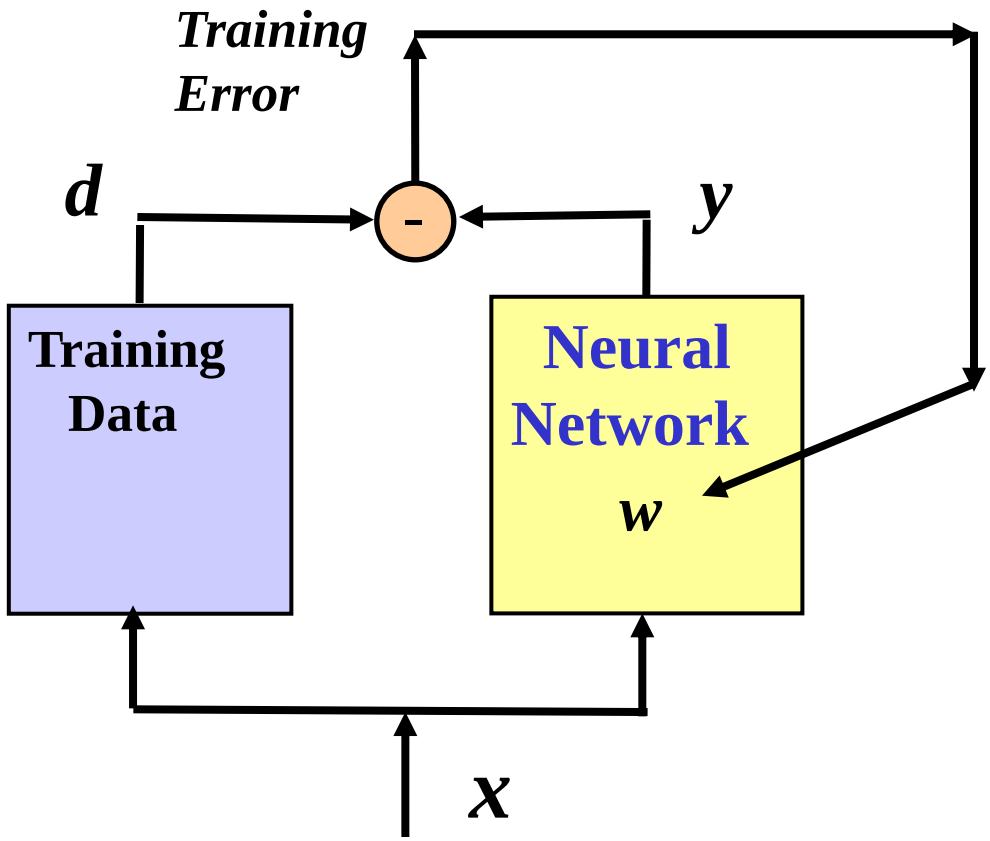
  where an epoch is defined as a stage of ANN training that involves presentation of all the samples in the training data set to the neural network once for the purpose of learning

- **Supervised training: using (*x* & *y*) data for training**

- **Un-supervised training: using only *x* data for training**

Neural Network Training

The error between training data and neural network outputs is fedback to the neural network to guide the internal weight update of the network



Training Problem Statement:

Given training data $(x_k, d_k), k \in Tr$

Validation data $(x_k, d_k), k \in V$

NN model $y(x, w)$

Find values of w**, such that validation error is minimized**

$$\min_{epoch} E_V(epoch)$$

where

$$E_V(epoch) \overset{\Delta}{=} \left[\frac{1}{P_V \cdot m} \sum_{k \in V} \sum_{j=1}^{m} \left| \frac{y_j(x_k, w(epoch)) - d_{jk}}{y_{max\,j} - y_{min\,j}} \right|^q \right]^{1/q}$$

$$P_V = size(V)$$

$$w(epoch) = w(epoch-1) + \Delta w(epoch-1)$$

$$w|_{epoch=0} = \text{user's / software initial guess}$$

$\Delta w(epoch-1)$ **is the update determined by the optimization algorithm (training algorithm) which minimizes the training error**

Steps of Gradient Based Training Algorithms:

Step 1: w = initial guess

$epoch = 0$

Step 2: If $E_V(epoch) < \varepsilon$ (given accuracy criteria)

or $epoch > max\_epoch$ (max number of epochs), stop

Step 3: Calculate $E_{T_r}(w)$ and $\dfrac{\partial E_{T_r}(w)}{\partial w}$ using partial or all training Data

Step 4: Use optimization algorithm to find Δw

$w \leftarrow w + \Delta w$

Step 5: If all training data are used, then

$epoch = epoch + 1$, go to Step 2, else go to Step 3

Update w in Gradient-based Methods

$$\Delta w = \eta\, h$$

where h is the direction of the update of w
η is the step size of the update of w

Gradient based methods use information of $E_{T_r}(w)$ and $\frac{\partial E_{T_r}(w)}{\partial w}$ to determine update direction of w .

Step size η is determined by:

- **Small fixed constant set by user**
- **Adaptive constant during training**
- **Line minimization method to find best value of η**

Line Minimization Problem Statement

Let a scalar function of one variable be defined as

$$\phi(\eta) = E_{Tr}(w + \eta\, h)$$

Given present value of w and direction h
Find η such that $\phi(\eta)$ is minimized.

Solution method: (1-dimensional optimization method):

- **Sectioning methods**
  - **Golden section method**
  - **Fibonacci method**
  - **Bisection method**
- **Interpolation methods**
  - **Quadratic method**
  - **Cubic method**

Back-Propagation (BP), (Rumelhart, Hinton & Williams, 1986)

We use the negative gradient direction: $h = -\dfrac{\partial E_{T_r}(w)}{\partial w}$

for $\Delta w = \eta h$

The neural network weights are updated during training as:

$$w = w - \eta \frac{\partial E_{T_r}(w)}{\partial w}$$

or

$$w = w - \eta \frac{\partial E_{T_r}(w)}{\partial w} + \alpha \, \Delta w \,|_{\,epoch-1}$$

where η is called the learning rate

α is called the momentum factor

Determining η and α for BP :

- Set η and α as fixed constant
- η and α can be adaptive
- η = c / epoch, c is a constant

- STC (Darkens, 1992)

$$\eta = \eta_0 \frac{1 + (\frac{c}{\eta_0}) \cdot (\frac{epoch}{\tau})}{1 + (\frac{c}{\eta_0}) \cdot (\frac{epoch}{\tau}) + \tau^2 (\frac{epoch}{\tau})^2}$$

where η_0, τ, c are user defined

- Delta-bar-delta (Jacobs, 1988)

(a) A η_i for each weight w_i in w of NN, $w_i = w_i - \eta_i \frac{\partial E_{T_r}(w)}{\partial w}$

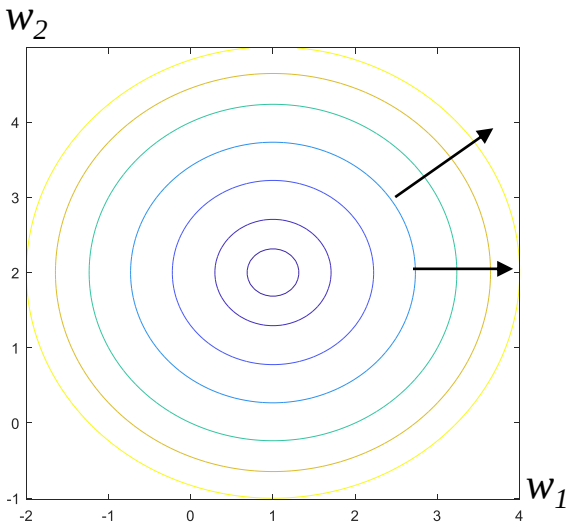
(b) η is adjusted during training using present and previous information of $\frac{\partial E_{T_r}(w)}{\partial w_i}$

Concept of Contour Plots

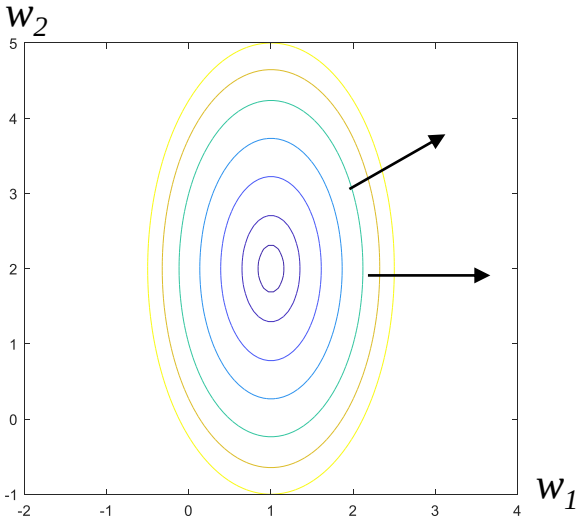
To illustrate the process of how the *w* vector change, we can use contour plots.

Simple examples of contour plots with 2 variables $w=[w_1 \ w_2]$:

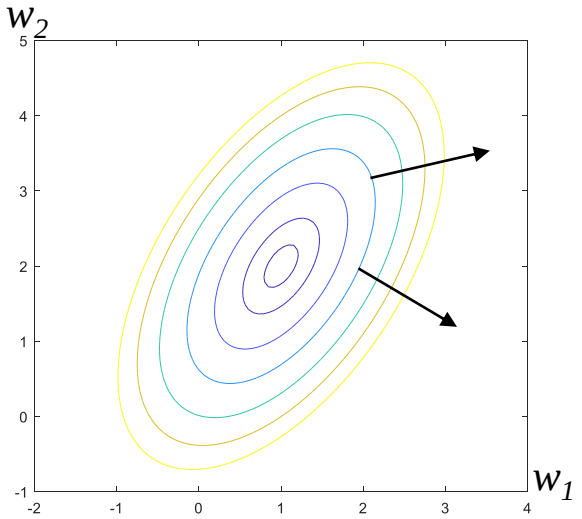
$E_{Tr}(\boldsymbol{w})=(w_1 -1)^2 + (w_2 -2)^2$



$E_{Tr}(\boldsymbol{w})=4(w_1 -1)^2 + (w_2 -2)^2$



$E_{Tr}(\boldsymbol{w})=(1.73(w_1 -1)-(w_2 -2))^2 + 0.25*((w_1 -1)+1.73(w_2 -2))^2$



Arrows show direction of gradient vector $\partial E_{Tr} / \partial \boldsymbol{w}$

The gradient vector is always perpendicular to the contour

For BP, ***w*** will move along negative direction of the gradient

Conjugate Gradient Method

$$\Delta w = \eta\, h$$

Let $\nabla E = \dfrac{\partial E_{T_r}(w)}{\partial w}$

Then $h(epoch) = -\nabla E + \gamma\, h(epoch-1)$

where $h_0 = -\nabla E$

$$\gamma = \frac{\left\|\nabla E(epoch)\right\|^2}{\left\|\nabla E(epoch-1)\right\|^2} \qquad \textbf{(Fletcher/Reeves)}$$

$$\gamma = \frac{(\nabla E(epoch) - \nabla E(epoch-1))^T \nabla E(epoch)}{\left\|\nabla E(epoch-1)\right\|^2} \qquad \textbf{(Polak – Ribiere)}$$

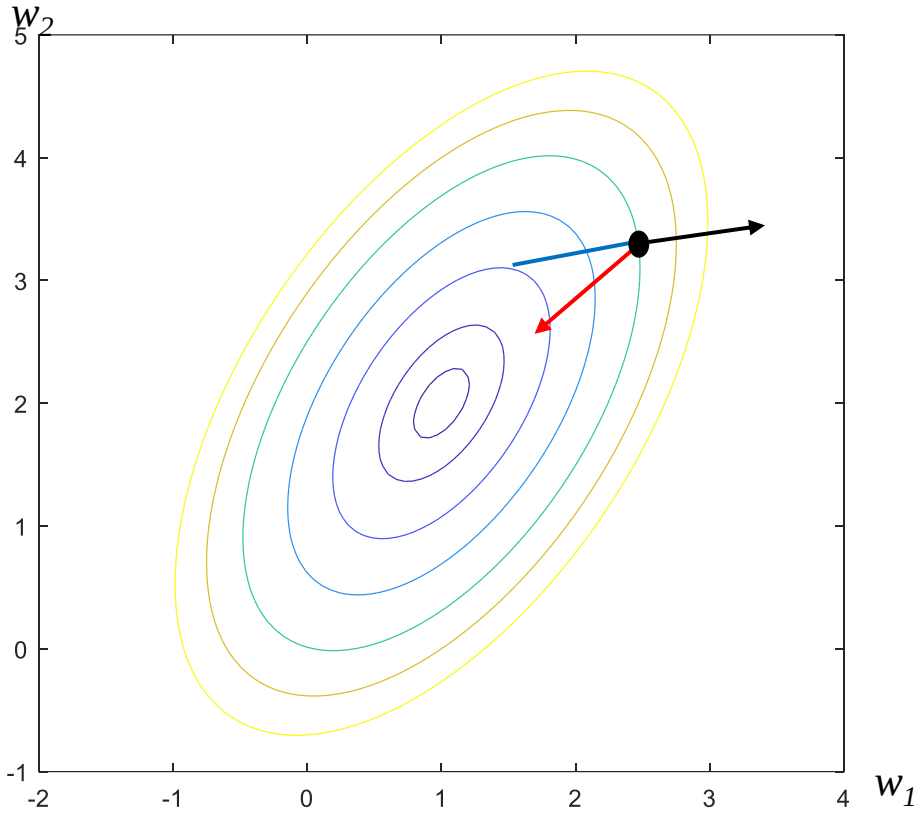
η **is determined by** $\begin{cases} \textit{Line minimization method} \\ \textit{Trust Region method} \end{cases}$

Speed: generally fast than BP

Memory: A few vectors of N_W long, where N_W is the total # of NN weights/parameters in w

Illustration of Conjugate Direction

Simple examples of contour plots with 2 variables $w=[w_1 \ w_2]$:



Location of $\boldsymbol{w}$

Gradient direction

Negative gradient direction

Conjugate gradient direction

Quasi-Newton Method

Let H be the Hessian matrix of E_{Tr} w.r.t. w

B be the inverse of H

Weight update: $\Delta w = \eta h$, where $h = -B\nabla E$

Use information of Δw and Δg to approximate B:

$$B\big|_{epoch=0} = I$$

$$B\big|_{epoch} = B\big|_{epoch-1} + \frac{\Delta w \Delta w^T}{\Delta w^T \Delta g} - \frac{B\big|_{epoch-1} \Delta g \Delta g^T B\big|_{epoch-1}}{\Delta g^T B\big|_{epoch-1} \Delta g}$$

(DFP formula)

where $\Delta w = w(epoch) - w(epoch\text{-}1)$

$$\Delta g = \nabla E(epoch) - \nabla E(epoch-1)$$

Speed: fast

Main Memory Needed: N_W^2 (Large)

Levenberg-Marquardt Method

Obtain Δw from solving linear equations, e.g.

$$-(J^T J + \mu I)\Delta w = J^T e$$

where $e = [\, e_1, e_2, \ldots, e_{Ne} \,]^T$, $\quad e_i = \delta_{jk} = \dfrac{y_j(x_k, w) - d_{jk}}{y_{\max,j} - y_{\min,j}}$

J **is Jacobian,** $J = \left(\dfrac{\partial e^T}{\partial w}\right)^T$

$$\mu \begin{cases} > 0 & \textit{Typical Levenberg Marquardt} \\ = 0 & \textit{Gauss Newton} \end{cases}$$

This method is good if $\|e\|$ can be very small, e.g., small residue problems.

Computation needs LU decomposition
Main Memory Needed: N_W^2, (Large)

Other Training Methods

Huber-Quasi-Newton

Similar to Quasi-Newton, except that the error function for training is based on Huber function, and not the conventional least square error function.

The Huber formulation allows the training algorithm to robustly handle both small random errors and accidental large error in training data.

Other Training Methods (continued)

Simplex Method using information $E_{T_r}(w)$ only

The method starts with several initial guesses of *w*. These initial points form a simplex in the *w* space.

The method then iteratively updates the simplex using basic moves such as reflection, expansion, and contraction, according to the information of $E_{T_r}(w)$ at vertices of the simplex

The error $E_{T_r}(w)$ generally decreases as the simplex is updated.

Other Training Methods (continued)

Genetic Algorithm using information $E_{Tr}(w)$ only, searching for global minimum

The algorithm starts with several initial points of w called a population of w

A fitness value is defined for each w such that w with lower error $E_{Tr}(w)$ has a high fitness

w points with high fitness values are more likely selected as parents from whom new points of w called offspring are produced

This process continues, and fitness among the population improves

Other Methods (continued)

Particle Swarm Optimization (PSO) using information $\boldsymbol{E_{Tr}(w)}$ only, searching for global minimum

The algorithm starts with several initial points of $\boldsymbol{w}$. Each point of $\boldsymbol{w}$ is called a particle, and all the particles together is called a swarm of $\boldsymbol{w}$.

Let $\boldsymbol{p}_b$ represent the historical best of a particle $\boldsymbol{w}$. Let $\boldsymbol{g}_b$ represent the historical best of all particles $\boldsymbol{w}$ in the swarm.

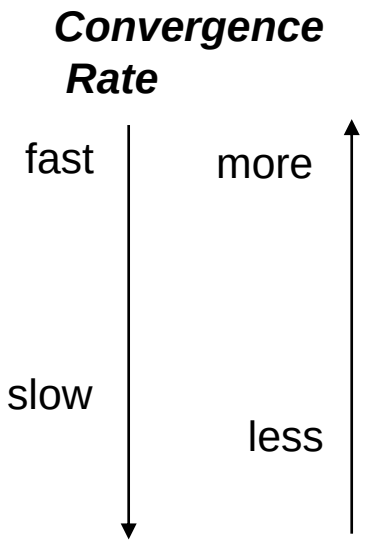
Let $\boldsymbol{v}$ be defined as the velocity of a particle $\boldsymbol{w}$, computed as:

$$\boldsymbol{v} = c_0 \boldsymbol{v}_{old} + c_1 r_1 (\boldsymbol{p}_b - \boldsymbol{w}) + c_2 r_2 (\boldsymbol{g}_b - \boldsymbol{w})$$

where c_0, c_1 and c_2 are constant weight parameters, r_1 and r_2 are random values between 0 and 1, and $\boldsymbol{v}_{old}$ represent the $\boldsymbol{v}$ of the particle in the previous iteration.

Each particle is updated by $\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{v}$

Qualitative Comparison of Different Algorithms



Levenberg Marquardt (for small-residue problems)
Quasi-Newton
Conjugate Gradient
BP

Comparison of Training Algorithms for 3-Conductor Microstrip Line Example (5 input neurons, 28 hidden neurons, 5 output neurons)

Training Algorithm	No. of Epochs	Training Error (%)	Avg. Test Error (%)	CPU (in Sec)
Adaptive Backpropagation	10755	0.224	0.252	13724
Conjugate-gradient	2169	0.415	0.473	5511
Quasi-Newton	1007	0.227	0.242	2034
Levenberg-Marquardt	20	0.276	0.294	1453

Comparison of Training Algorithms for MESFET Example

(4 input neurons, 60 hidden neurons, 8 output neurons)

Training Algorithm	No of Epochs	Training Error (%)	Ave. Test Error (%)	CPU (in Sec)
Adaptive Backpropagation	15319	0.98	1.04	11245
Conjugate Gradient	1605	0.99	1.04	4391
Quasi-Newton	570	0.88	0.89	1574
Levenberg-Marquardt	12	0.97	1.03	4322