Fast Back-Propagation Learning Methods For Neural Networks in Speech.

P.Haffner, A.Waibel and K.Shikano
(ATR Interpreting Telephony Research Laboratories)

1) Problem:

Optimize speech tasks in Neural Networks within a reasonable amount of computation time.

2) Back-Propagation Learning Procedure:

Iterative search for a global minimum of the Error function with a gradient descent algorithm.

3) Improving learning speed:

- **Avoid flat spots on the Error surface** : Model a sigmoid function which prevents zero learning rate.

- **Increase learning step**: Dynamically scale the gradient step size to its maximum value, with respect to overshooting.

- **Reduce learning grain:** Update connections as often as possible.

Algorithms have been selected on the basis of
- learning **speed.**
- good **generalization** on test data.
- few **parameters** to tunt.

4) Experiments:

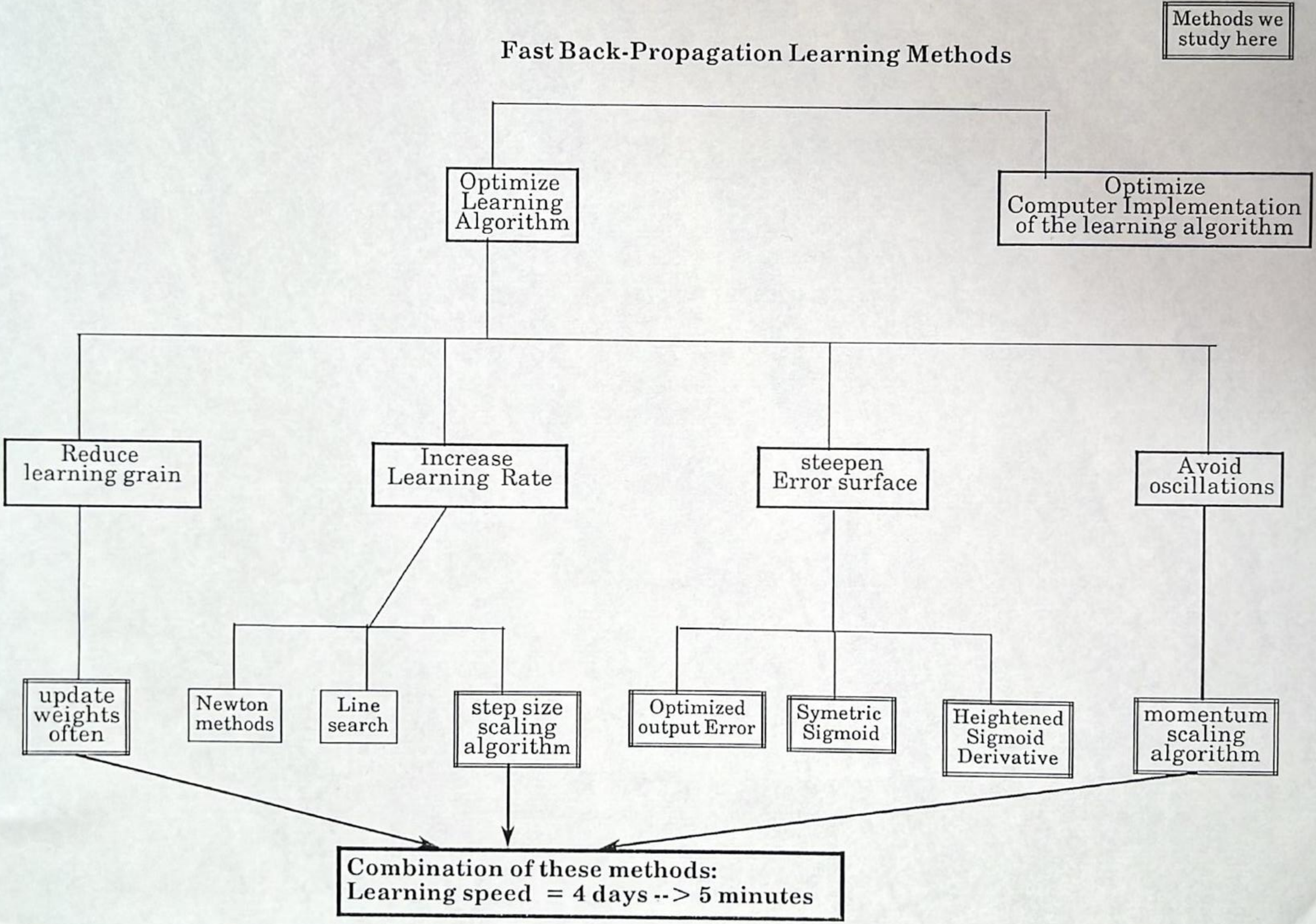
Training time for the recognition of the phonemes /b/, /d/ and /g/ has been reduced to less than **5 minutes** (Several days with standard Back-Propagation).

2 - P - 1

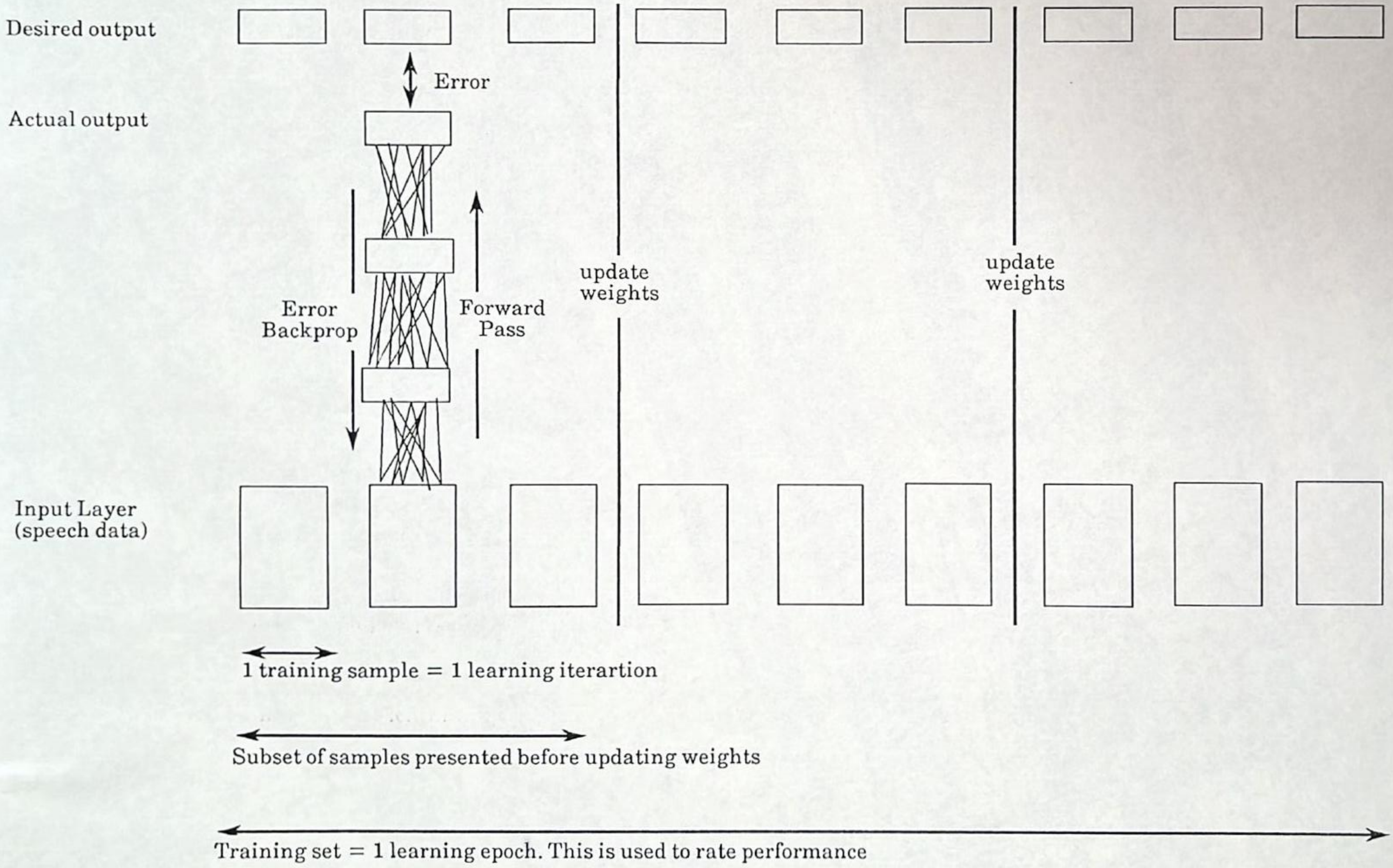
Fast Back-Propagation Learning Methods for Neural Networks in Speech.

P.Haffner, A.Waibel and K.Shikano
(ATR Interpreting Telephony Research Laboratories)

Fast Back-Propagation Learning Methods


Methods we study here
Optimize Learning Algorithm
Optimize Computer Implementation of the learning algorithm
Reduce learning grain
Increase Learning Rate
steepen Error surface
Avoid oscillations
update weights often
Newton methods
Line search
step size scaling algorithm
Optimized output Error
Symetric Sigmoid
Heightened Sigmoid Derivative
momentum scaling algorithm
Combination of these methods:
Learning speed = 4 days --> 5 minutes

Learning in Neural Networks


Desired output
Error
Actual output
Error
Backprop
Forward
Pass
update
weights
update
weights
Input Layer
(speech data)
1 training sample = 1 learning iterartion
Subset of samples presented before updating weights
Training set = 1 learning epoch. This is used to rate performance

Relations between the step size and other parameters

How do we rate performance ?

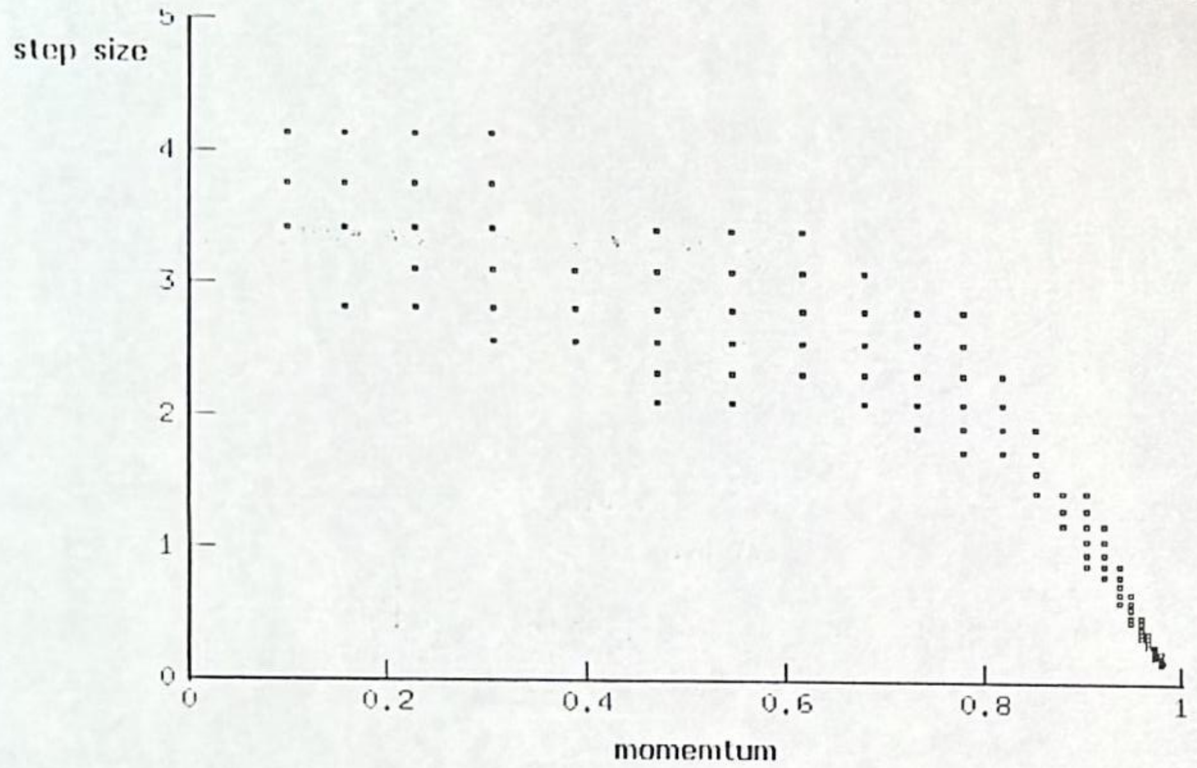
- The recognition rate on test data is the percentage of test samples that are correctly classified by the network after learning some training samples.

- Error rate = 100 - recognition rate.

- We say that learning converges when the Error rate on training data goes below 2.0.

MOMENTUM $\alpha : \varepsilon \propto (1-\alpha)$

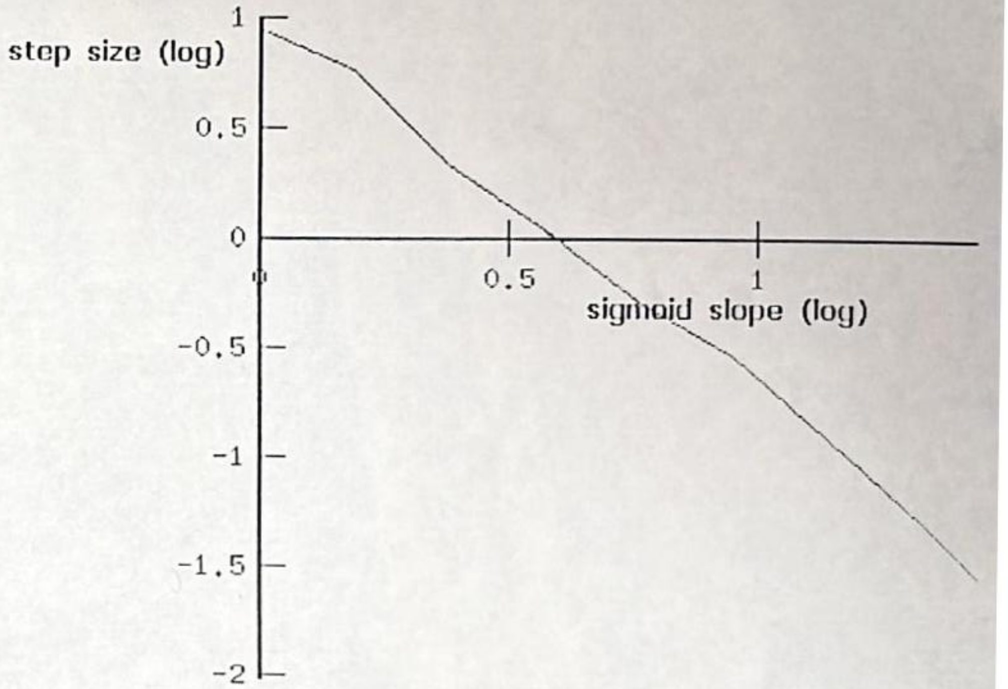
This relation based on theoretical considerations is not well verified.



838 task : optimal step size vs. momentum.

SIGMOID function $f : \varepsilon \propto 1/(f_{max}\, f'_{max})^2$

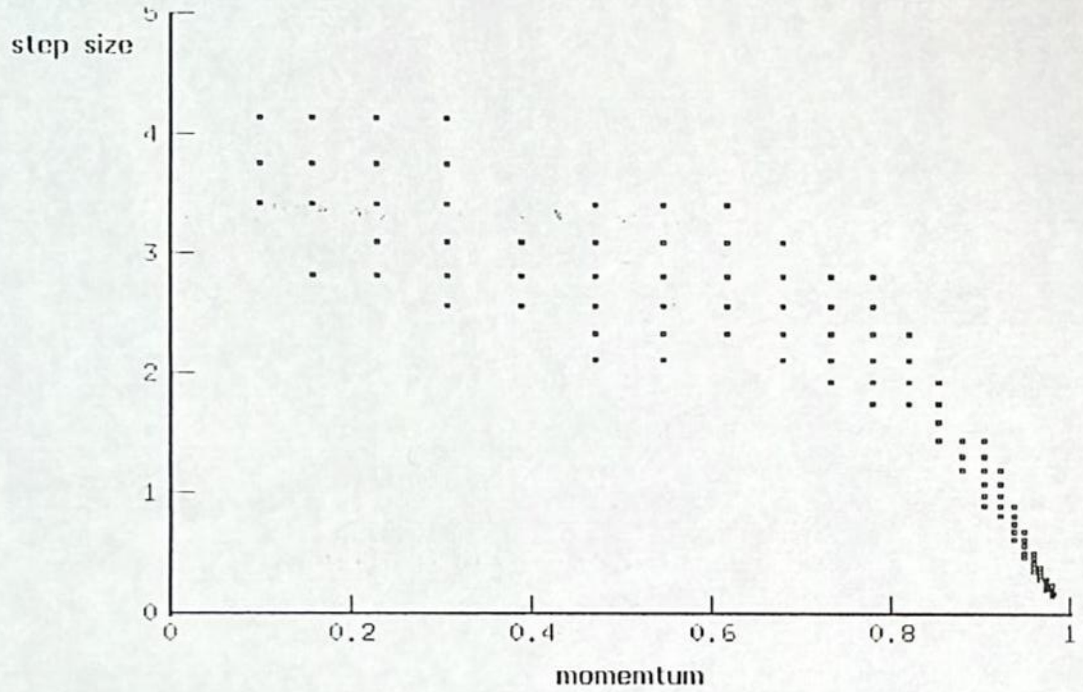
If $f(x) = s/(1 + e^{-x})$ we must have $\varepsilon \propto s^{-4}$



XOR task : optimal step size v.s. sigmoid slope

MOMENTUM $\alpha : \varepsilon \propto (1-\alpha)$

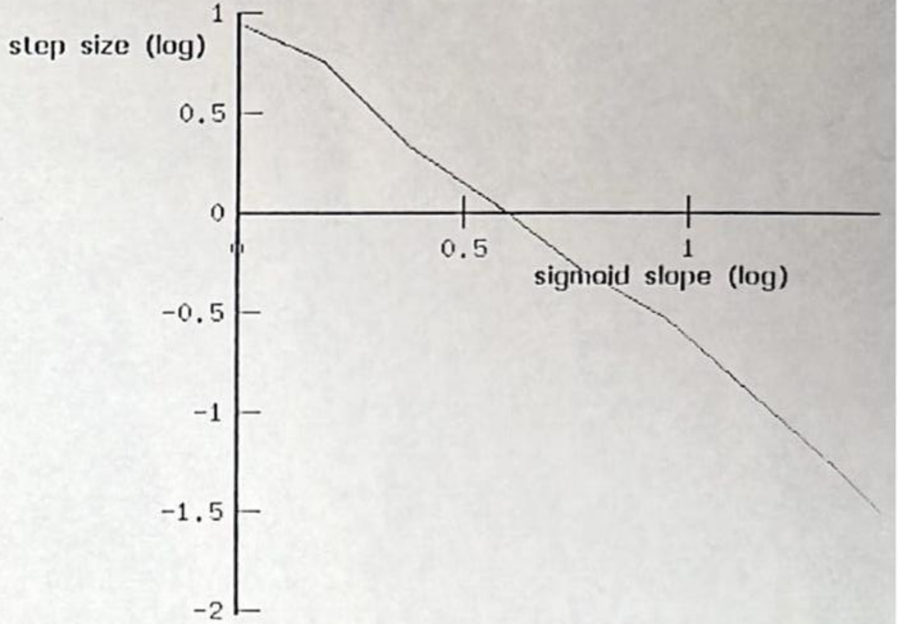
This relation based on theoretical considerations is not well verified.



838 task : optimal step size vs. momentum.

SIGMOID function $f : \varepsilon \propto 1/(f_{max} f'_{max})$:

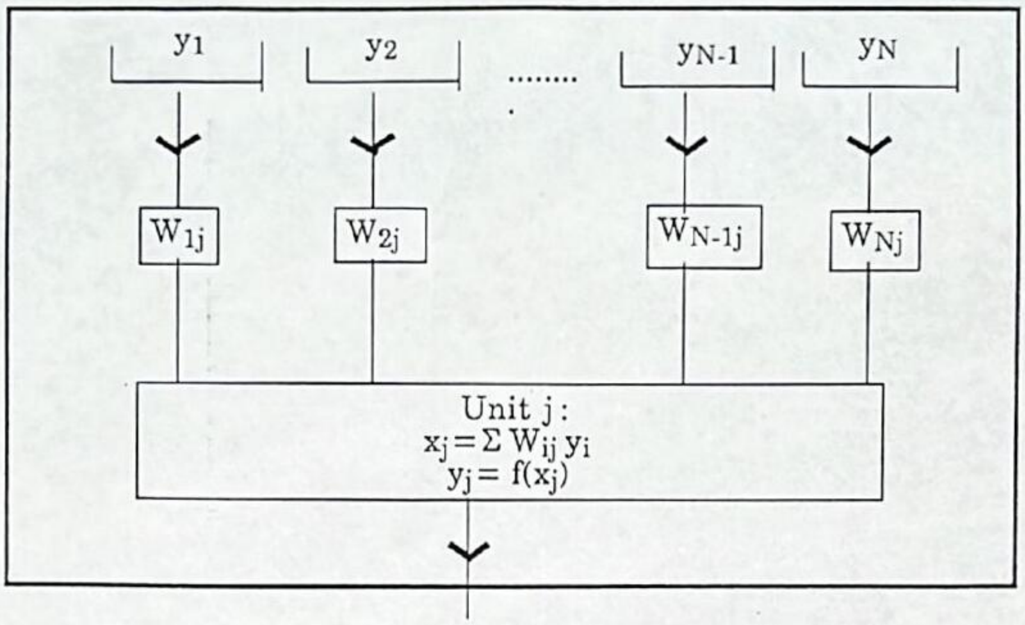
If $f(x) = s/(1 + e^{-x})$ we must have $\varepsilon \propto s^{-4}$



XOR task : optimal step size v.s. sigmoid slope

V) One unit in a Neural Network

A unit in our neural network computes a weighted sum of the states of its input units, and applies a sigmoid function to this sum.



How does this unit learn ?

Back-Propagation learning procedure is **local:**

--> the modifying parameters for the weight W_{ij} only **depend on unit j and on input** y_i

$$\partial E/\partial W_{ij}(t) = \Sigma_{\text{samples}}\, \partial E/\partial x_j \ y_i$$

$$\Delta W_{ij}(t) = \alpha_j \Delta W_{ij}(t\text{-}1) - \varepsilon_j \partial E/\partial W_{ij}(t)$$

$$W_{ij}(t) = W_{ij}(t\text{-}1) + \Delta W_{ij}(t)$$

$\partial E/\partial x_j$ is the **back-propagated signal:**

$$\partial E/\partial x_j = f'(x_j)\,(\Sigma_k W_{jk}\, \partial E/\partial x_k)$$

If we write W_j the vector $(W_{1j}, W_{2j}, ... W_{Nj})$,

Learning = trajectory of W_j in the weight space, determined by **unit j.**

α_j **and** ε_j **will be scaled to:**

- Make this trajectory as **straight** as possible.
- Minimize **oscillations.**
- Have a **fast** but controlled learning speed.

VI) The step size ε : how to scale it.

1) Initial values.

- Optimize ε with a **small** training set.
- Scale it to a **larger** training set using the relation:

$$\varepsilon = \text{Constant} / (\text{size of training set})$$

- Generally set the step size of the **output** units to **1/10** of the others.

2) Dynamical scaling.

- As a **gauge of the variations** of ε in time, we take the **cosine of the angle** between the error gradient at epoch t and the weight variation at epoch t-1:

$$\theta = \text{Angle}(\Delta W(t-1), -\nabla E(t)) .$$

- If we compute the cosine over the set of input connections to unit u, the algorithm becomes **local** to this unit:

$$\varepsilon_u(t) = \varepsilon_u(t-1).e^{(\rho.\cos(\theta))}.$$

3) Overshooting Control.

A control , at each updating iteration, limits the norm of the vector $\varepsilon_u.\nabla E$ to a fixed value $o=1.0$.

Initial ε	Fixed ε (epochs)	Scaled ε (epochs)	Initial ε	Fixed ε (epochs)	Scaled ε (epochs)
0.5	1010	70	0.01	560	100
10	30	35	0.02	360	80

XOR task BDG task.(250 samples)

VII) The momentum α : how to scale it.

1) Problems met with a fixed momentum.

When the weights are updated at each **epoch**, it is generally assumed that a good value for the momentum is 0.9.

This value is not optimal when we are updating the weights more **often**.

- When the momentum is too **small** (0.9), we have oscillations.
- When it is too **large**, we have uncontrolled overshooting.

2) A scaling algorithm.

For a given unit j, the quantity we want to control is:

$$\Delta_j = \Sigma_i (W_{ij}(t))^2 - \Sigma_i (W_{ij}(t-1))^2 = 2\, \Sigma_i W_{ij} \Delta W_{ij}$$

We have $\Delta W_{ij}(t) = \alpha_j \Delta W_{ij}(t-1) - \varepsilon_j\, dE/dW_{ij}$

As our ε scaling algorithm limits the ε_j term, it is possible to limit the value of Δ by scaling α_j with the relation:

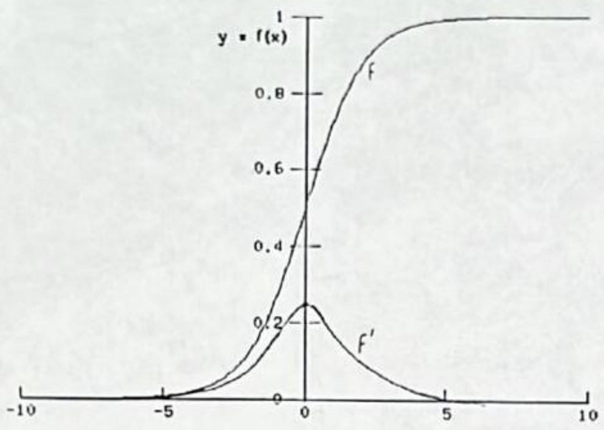
$$\alpha_j = \alpha_{max} / (1 + d\, |\Sigma_i W_{ij} \Delta W_{ij}|)$$

$\alpha_{max} = 0.99$ and $d = 1.0$ give good performance.

Epochs	10	20	30	40	50
$\alpha = 0.9$	3.3 /20	2.6 /50	2.7 /60	2.5 /60	2.4 /60
$\alpha = 0.95$	2.8 /10	1.8 /20	1.7 / 30	1.8 / 40	2.0 /50
α : dynamic	2.5 / 50	1.9 /70	1.7/100	1.7/100	1.6/100

BDG task with 780 training samples .
Error rate / % converging trials
(10 trials, Error averaged on converging trials)
(10 trials)

VIII) The sigmoid function.



Back-Propagation Learning rate is proportional to the values of the sigmoid function f and its derivative f'. But these functions flatten out at infinity.

Fig 1 : sigmoid function $f(x) = 1/(1 + e^{-x})$

Possible improvements:

1) Use a **symmetric sigmoid** whose value is never zero at infinity.

2) add a **linear function** to the sigmoid function: $f_l(x) = f(x) + l.x$.

3) only **add a small constant** l to the sigmoid derivative .

Models 2 and 3 guarantee **fast convergence to zero Error**, but overlearn the training set and may yield poor generalization on test samples.

Epochs	10	20	30	40	50
Standard Sigmoid	2.5 /50	1.9 /70	1.7 /100	1.7 /100	1.6 /100
Sigmoid 1	2.4 /50	2.1 /90	2.1 /90	2.1 /100	1.8 /100
Sigmoid 2 l=0.01	2.4 /90	1.9 /100	1.9 /100	1.9 /100	1.9 /100
Sigmoid 3 l=0.01	2.1 /100	2.0 /100	1.8 /100	2.0 /100	2.0 /100

BDG task with 780 training samples .
Error rate / % converging trials
(10 trials, Error averaged on converging trials)

IX) Influence of Initial weights:

- Different initial weights may give **very different recognition rates** after learning.
- Initial weights give good performance with a learning method,

  --> performance is still good with slightly different learning methods.

Learning method	Standard	Sigmoid derivative + 0.01	overshooting cont 10 instead of 1
3 best initial weights. (numbered from 0 to 9)	2, 0, 3	2, 0, 8	0, 3, 2

BDG task with 780 training samples

- It is possible to improve learning speed by choosing weights which are **adapted** to the problem*.

Epochs	10	20	30	40	50
Adapted weights	2.5 / 50	1.9 /70	1.7 /100	1.7 /100	1.6 /100
Random weights	2.8 /20	2.5 / 90	2.0 /100	1.9 /100	2.0 / 100

BDG task with 780 training samples .
Error rate / % converging trials
(10 trials, Error averaged on converging trials)

* For instance, in a T.D.N.N., a physical unit is connected to another physical unit through several connections, each having a different delay. Even though their weights are different, **learning is faster if they are initially set equal.**

X) The Weight Updating Frequency.

Splitting a large and often highly redundant training set into smaller subsets is advantageous.

We assume that these subsets are **representative** of the problem .

At the beginning of the learning phase:
One subset is enough data for a network which is only acting as a **rough classifier.**

At the end of the learning phase:
The difference between two learning subsets may be considered as **noise** which prevents zero variation.

Weight updating procedure [P]

- Learns 780 samples representing the phonemes /b/, /d/ and /g/:
- Training sample: **randomly mixed**, to avoid overspecialised training subsets.
- First learning epoch, weights are updated each 3 iterations.
- Each epoch, the size of the training subset is incremented by 3, until it reaches its maximum value of 48.

Epochs	20	50	100	200	300
Epoch updating	0 /0	0 /0	2.0 /10	2.3 /100	2.0 /90
[P] Non mixed set	2.0 /10	2.1 /70	1.5 /70		
[P] Mixed set	2.5 /50	1.6 /100			

BDG task with 780 training samples .
Error rate / % converging trials
(10 trials, Error averaged on converging trials)

XI) A new Error.

This **New Error function** has been proposed by McClelland:

$$E = - \Sigma_{samples} \Sigma_j \ln(1 - (y_j - d_j)^2)$$

(y_j is the actual output and d_j is the desired output).

- **dE / dy_j is maximal when $y_j - d_j = 1$**

(it is **zero** with the standard Error $E = - \Sigma_{samples} \Sigma_j (y_j - d_j)^2$)

- This **prevents zero learning** and has roughly the same effect as adding a small constant to the sigmoid derivative.

Epochs	10	20	30	40	50
Standard Error	2.5 /50	1.9 /70	1.7 /100	1.7 /100	1.6 /100
New Error	2.4 /90	2.4 /100	2.1 /100	2.1 /100	2.1/100

BDG task with 780 training samples .
Error rate / % converging trials
(10 trials, Error averaged on converging trials)

XII) Experiments.

We have met two practical problems in our learning experiments:

1) Initial Weights.

Initial weights have a strong influence on final performance. Several methods to cope with this:

1 After several trials, we select the weights which give the best recognition rate on training samples. **But**, this does not guarantee good performance on testing data..

Trial	0	1	2	3	4	5	6	7	8	9
Training data	0.64	0.64	0.51	0.38	0.38	0.38	0.64	0.77	0.51	0.38
Test data	1.32	2.50	1.05	1.32	1.71	1.45	1.71	1.98	1.84	1.19

BDG task with 780 training samples:

Average Error rate after 50 epochs

2 We select good initial weights with smaller training set, they generally still yield better than average performance with most learning methods.

2) When to stop learning ?

What? Stop learning before reaching the zero Error on the training samples.

Why? The network becomes **overspecialized** into recognizing them and performance on test samples gets worse.

When? The solution depends on the problem. Generally when the **Error has reached a plateau.**

XIII) Our final learning procedure:

- ε and α epsilon scaling algorithm.
- Weights updating procedure [P].
- Standard sigmoid function.
- Initial weights selected for their good performance with other learning methods.
- Learning stopped when recognition rate on training data stable during 10 epochs.

Epochs	10	20	30	40	50
Time (minutes)	1.5	2.7	3.8	4.9	6.0
Training data	1.41	0.90	0.64	0.64	0.64
Test data	2.24	1.45	1.19	1.19	1.32
remarks			plateau	stop here	overshooting

BDG task with 780 training samples:

Remarks:

- Our **best recognition rate** on test data is **99.2 %**, however the probability to get such a result is less than 10 %.
- These simulations have been made on an **Alliant computer with 8 processors.**
- Our program process **1 Million connections per second** (forward + backward pass).
- With the same program, we needed **several days** to reach a 98.8 % recognition rate using standard back-propagation procedure with a momentum of 0.9.