Analysis of Complex Dynamical Systems with Neural Networks

Shallow and Deep Feedforward Networks

Dr. Hans Georg Zimmermann
Senior Principal Research Scientist
Siemens AG, Corporate Technology
Email: Hans_Georg.Zimmermann@siemens.com
Neural Networks @ Siemens: 29 Years of Research, Development, Innovation
Mathematics of Artificial Intelligence

Foundations: Turing (computability), McCulloch / Pitts (brain ~ computer), Hebb (learning), Wiener (cybernetics)

Future:

**Weak AI**: Intelligent, Learning, Autonomous Systems / **Strong AI**: Search for Consciousness (but Gödel ..)

1955 / 56

**Artificial Intelligence: What To Do?**
Perception + Understanding + Action

1970

**Artificial Neural Networks: How To Do?**
Brain like Information Processing

1980

Decision Support
Logical & Fuzzy Rules for Expert Systems

Stimulus-Response Models
Feedforward Neural Nets for Regression & Classification

1990

Dynamical Systems
Recurrent Neural Nets for Forecasting & Control

2000

focus on learning instead of engineering
today
deep learning & recurrent neural networks

Perception
Processing of Heterogeneous Data

Understanding
Model Building based on Observations

Action
Decision Support, Optimization & Control

© Siemens AG, Corporate Technology
Mathematical Neural Networks

Existence Theorem:
(Hornik, Stinchcombe, White 1989)

3-layer neural networks \( y = W_2 f(W_1 x) \)
can approximate any continuous function on a compact domain.

Nonlinear Regression

Based on data identify an input-output relation

\[
y = W_2 f(W_1 x)
\]

\[
E = \sum_{t=1}^{T} (y_t - y_t^d)^2 \rightarrow \min_{w_1, w_2}
\]

Neural networks imply a Correspondence of Equations, Architectures, Local Algorithms.
Error Backpropagation - Correspondence between Architecture & Algorithm

By the forward & backward flows, \( \frac{\partial E_t}{\partial W_1}, \frac{\partial E_t}{\partial W_2} \) are efficiently computed.

Because of the local algorithm, we can easily extend the network.

In case of \( f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \) we get
\[
\tanh'(netin) = 1 - (\tanh(netin))^2 = 1 - out^2
\]

In case of \( f(z) = \text{logistic}(z) = \frac{1}{1 + e^{-z}} \) we get
\[
\text{logistic}'(netin) = \text{logistic}(netin)(1 - \text{logistic}(netin)) = out(1 - out)
\]

In case of large/sparse matrices W the use of \( W^T \) causes problems.
Feedforward Neural Networks: Learning
Local Minima versus Overparameterization in Nonlinear Regression

Observation: The experiment with 25 parallel networks shows a very diverse generalization.

Explanation: (1) could be caused by convergence of nonlinear models to different local minima.
(2) could be caused by non-unique solutions dependent on random initialization.

Dilemma: Parsimonious models are prone to type (1) problems, over parameterized models are prone to type (2) difficulties – but have less trouble with (1)!!
Steps to a Solution of Nonlinear Regression Problems

- Stochastic Optimization for Overparam. Models
- Learn Uncertainty of Data and Stiff Model Building
- Pick the Best Model of an Ensemble
- Ensemble Averaging as Best Solution

Learning Algorithms

SENN Network Architectures
Learning Structure from Data - Learning Rules as a Stochastic Search

Task: \[ E = \frac{1}{T} \sum_{t=1}^{T} E_t = \frac{1}{T} \sum_{i=1}^{T} (NN(x_t,w) - y_t^d)^2 \rightarrow \min_w \]  
Notation: \( g_t = \frac{\partial E_t}{\partial w}, \quad g = \frac{1}{T} \sum_{t=1}^{T} g_t \)

Steepest descent learning: \[ \Delta w = \eta \cdot (-g) = \text{step length} \cdot \text{search direction} \]
\[ E(w + \Delta w) = E(w) + g^T \Delta w + \frac{1}{2} \Delta w^T G \Delta w \]
\[ = E(w) - \eta g^T g + \frac{\eta^2}{2} g^T G g < E(w) \quad \text{for } \eta \text{ small} \]

Pattern by pattern learning: \[ \Delta w_t = -\eta g_t = -\eta g \quad - \eta(g_t - g) \]
steapest descent stochastic search

Noise on weights act as curvature penalty \[ \langle E(w) \rangle = \frac{1}{T} \sum_i E(w + \Delta w_i) = E(w) + \sum_i \left( \frac{1}{T} \sum_{i=1}^{T} \Delta w_{it} \right) \frac{\partial E}{\partial w_i} + \frac{1}{2} \sum_i \text{var}(\Delta w_{it}) \frac{\partial^2 E}{\partial w_i^2} \]

P-by-P Learning \( \Delta w_t = -\eta g_t \) induces a local penalty on \( w \):
\[ \langle E(w) \rangle = \frac{1}{T} \sum_i E(w + \Delta w_i) = E(w) + \frac{\eta^2}{2} \sum_i \text{var}(g_{it}) \frac{\partial^2 E}{\partial w_i^2} \]
Data meet Structure: The Observer - Observation Dilemma

**Calculus of Cleaning**

\[ dev_2 = out_2 - tar \]

\[ \partial_2 = \]

\[ \frac{\partial E}{\partial w_2} \]

\[ dev_1 = \]

\[ \partial_1 = \]

\[ \frac{\partial E}{\partial w_1} \]

\[ dev_0 = \]

\[ input_t \]

\[ \partial_{0,t} = \text{residual error} \]

\[ x_t = x_t^{data} + clean(\partial_{0,t}) \]

\[ input_t = x_t^{data} + noise(clean(\partial_{0,t})) \]

**Psychological Dilemma:**

observations → world picture

&

world picture → data cleaning

**Technical Dilemma:**

data → math. model

&

math. model → data cleaning

Use the data to fit the model

Use the model to clean the data

Data cleaning implies data uncertainty

Use the data uncertainty to harden the learning
Occams Razor: Search for a Parsimonious Network

Advise: Late Stopping & Weight Pruning

- Define a criterion for the weight importance:
  \[ \text{test}_w = w^2 \]

- **Weight Pruning Procedure:**
  1. Train the Neural Network
  2. Rank weights by importance
  3. Prune lower ranked weights

- Procedure is bias free towards linear models

Pruning methods split the training data in learning data & validation data, used in the trace.
Decreasing Model Uncertainty by Averaging

The sub-networks learn different solutions of the same task. In case of large averages (m > 20) an equal weighting is superior, in case of small averages it is superior to freeze the subnets and optimize the weighting factors.

$$E_{\text{aver}} = \frac{1}{T} \sum_{T} [out_{\text{aver}} - \text{tar}]^2$$

$$= \frac{1}{T} \sum_{T} \left[ \left( \frac{1}{m} \sum_{i} \text{out}_i \right) - \text{tar} \right]^2$$

$$= \frac{1}{T} \sum_{T} \left[ \frac{1}{m} \sum_{i} (\text{out}_i - \text{tar}) \right]^2$$

$$= \frac{1}{m^2} \frac{1}{T} \sum_{T} \sum_{i} (\text{out}_i - \text{tar})^2$$

$$= \frac{1}{m} \frac{1}{m} \sum_{i} \frac{1}{T} \sum_{T} (\text{out}_i - \text{tar})^2$$

$$= \frac{1}{m} \text{aver}(E_i)$$

$$\frac{1}{T} \sum_{T} (\text{out}_i - \text{tar}) \cdot (\text{out}_j - \text{tar}) = 0 \quad \forall i \neq j$$

(covariance of the errors of the submodels)
The average ensemble model error is much better than the erratic error levels of the sub-models.
No pruning is used to stabilize the learning procedure.
The technique is of good nature regarding mis-specifications of the meta parameters.
Neural Networks are No Black Boxes

Application: Modeling of a Gas Turbine

- Inputs: 35 sensor measures and control variables of the turbine
- Output: NO$_x$ emission of the gas turbine

Sensitivity Analysis: Compute the first derivatives along the time series:

$$\frac{\partial \text{output}}{\partial \text{input}_i} > 0 \quad \frac{\partial \text{output}}{\partial \text{input}_i} < 0$$

A classification of input-output sensitivities:

- **linear relationship** (= constant first derivative)
- **monotone** (input can be used in 1dim. control)
- **non-monotone** (only multi-dim control possible)
- **~ zero** (input useless in modeling and control)
Monotonic Function Approximation with Ensemble Neural Networks

Ensemble networks are an efficient way to detect a monotonic input–output relationship. Assume, that all sub-models are good approximations of a monotonic target line. Even then, the individual sub-models might meander around this target in a non-monotonic way. Ensemble averaging smoothes out hills and valleys.

\[
\frac{\partial y(t)}{\partial x_i} \geq 0 \quad \text{but} \quad \frac{\partial y(t)}{\partial x_i} \leftrightarrow ?
\]
Deep Feedforward Neural Networks
Why Deep Feedforward Neural Networks?

The learning of deep neural networks is difficult, because…
- first layers can do nonsense, while the last hidden layer corrects everything – why to use a deep structure?
- **forward path**: relevant input information may get lost in the hierarchy of hidden layers,
- **backward path**: the error signal decays passing through many hidden layers.

**Challenge**: Define explicit tasks for the intermediate layers!
+ to allow a sequential computation of very complicated input-output relations,
+ to do a dimensionality reduction for very large input layers,
+ to exploit neighboring relationships in the inputs (images, spectra, time series).
Learning of Deep Feedforward Neural Networks

The extended network acts as a hierarchical filter

**Forward path:**
feeding the inputs to all intermediate layers avoids a loss of the input information.

**Backward path:**
learning is not only applied to the final target but to all intermediate layers.

**Remark:**
The learning is improved if we use *backward false* in the hidden backbone (hidden $\rightarrow$ hidden connections learn, but transfer no error flow)
Error Correction Learning in Deep Feedforward Neural Networks

Level 1 acts as a standard feedforward neural network. From Level 2 on, all levels have to learn only the residual error from the levels below.

\[
\begin{align*}
y^1 - y^d &= \varepsilon^1 \\
y^2 + \varepsilon^1 &= \varepsilon^2 \\
y^3 + \varepsilon^2 &= \varepsilon^3 \\
y^4 + \varepsilon^3 &= \varepsilon^4
\end{align*}
\]

The aggregation of all outputs add up to our final output.

\[
y^d = y^1 - \varepsilon^1 = y^1 - (-y^2 + \varepsilon^2) = \ldots
\]

\[
y^d = y^1 + y^2 + y^3 + y^4 - \varepsilon^4
\]
The deep network (here: 14 layers) shows the best error with 9 hidden layers (no weight pruning).

Check for the opt. number of hidden layers on a val. set & relearn on the total data set (typically >>1).

The network does not show any degradation during a long training; In no appl. 2 layers were optimal.
Design a deep feedforward neural network to realize a sequence of increasing abstract features. Finally, the output (regression or classification) is computed.

Greedy learning from **bottom to top**:

Connectors A, B, C, D have to be backward false (there is no backward error flow though the net).

At start, apply an unsupervised learning rule only to matrix A.

Repeat this procedure level by level from **bottom to top**.

Finally learn the output y with supervised learning.

For references see also the work of the groups of Yoshua Bengio and Geoffrey Hinton from 2006 on
Dimensionality Reduction with Auto-Encoders (Auto-Associators)

Auto encoders condense a large redundant input into a smaller set of variables, such that the original inputs can be reconstructed. This is a coordinate transformation onto a sub-manifold. In case of a 3-layer perceptron, this is similar to a principal component analysis.
The stacked auto-encoder squeeze out redundant input information – with a focus on the final target.

In a final step we need a general classifier to evaluate the task.

Do the learning level by level from bottom to top.

Instead of (A, A')…, one can use with pairs of (A, A^T) …
The Learning of Deep Auto-Encoders

The deep auto-encoder architecture allows a more complicated data compression or general coordinate transformations between manifolds.

To work on an initialization of the deep network we could try again to use an unsupervised greedy algorithm (e.g. Oja- or Boltzmann learning rules) in the first phase of the learning.

In a second phase we use this solution as an initialization of a back propagation learning for the overall auto-encoder.
Stacked Auto-Encoder Approach for the Learning of Deep Auto-Encoders

In the lower part of the architecture we learn a stacked auto encoder with matrices $A, A', B, B'$.

The matrices $A', B'$ are by construction exactly the back transformations we need in the upper part of the structure.

Technically speaking this is simply a shared weights network.
Images are at least 2 dim matrices (+colours = 3 dim tensors & 4 dim in videos). The neighboring should be taken into account.

With Deep Neural Networks a lot of preprocessing seems to be obsolete.

Deep Convolutional Networks exploit locality in time and space and combine local features to complex patterns.
Backpropagation in Index Formulation

\[ E = \frac{1}{T} \sum_{t=1}^{T} E_t = \frac{1}{T} \sum_{t=1}^{T} \frac{1}{2} (y_t - y^d_t)^2 \]

Output: \( k = 1, \ldots, n \)
\[ \text{target} \]
\[ \text{dev}^2_k = \frac{\partial E}{\partial y_k} = \text{out}^2_k - \text{target}_k \]

Hidden: \( j = 1, \ldots, m \)
\[ \text{netin}^1_j = W^1_{ji} \text{out}^0_i \]
\[ \text{dev}^1_j = W^2_{kj} \text{dev}^2_k \]

Input: \( i = 1, \ldots, l \)
\[ \text{netin}^0_i = x_i \]
\[ \text{dev}^0_i = W^1_{ji} \text{dev}^1_j \]

Convention (Einstein): Over indices occurring twofold in a product we sum up, e.g.
\[ \text{netin}^1_j = W^1_{ji} \text{out}^0_i = \sum_{i=1}^{l} W^1_{ji} \text{out}^0_i \]

The summation over the first index of a matrix describes a transpose operation, e.g.
\[ \text{dev}^0_i = W^1_{ji} \text{dev}^1_j = \sum_{j=1}^{m} W^1_{ji} \text{dev}^1_j \]
Backpropagation for Tensor Data Flows & Computations with Multi-Indices

Tensor products with multi-indices, e.g.

\[ netin^1_j = W^1_{ji} \cdot out^0_i \]

\[ netin^{(j_1, j_2, j_3, \ldots)} = W^1_{(j_1, j_2, j_3, \ldots)} \cdot out^0_{(i_1, i_2, i_3, \ldots)} \]

\[ dev^0_i = W^1_{ji} \partial^1_j \]

\[ dev^0_{(i_1, i_2, i_3, \ldots)} = W^1_{(j_1, j_2, j_3, \ldots)} \partial^1_{(j_1, j_2, j_3, \ldots)} \]

\[ \frac{\partial E}{\partial w^1_{ji}} = \partial^1_j \cdot out^0_i \]

\[ \frac{\partial E}{\partial w^1_{(j_1, j_2, j_3, \ldots)}} = \partial^1_{(j_1, j_2, j_3, \ldots)} \cdot out^0_{(i_1, i_2, i_3, \ldots)} \]
Memory and Computational Problems with Tensors

Problem: Tensors can be extremely large, which causes memory and computational power problems:

\[ W_{(j_1, j_2, j_3 \ldots i_1, i_2, i_3 \ldots)} \in \mathbb{R}^{(|j_1| + |j_2| + |j_3| + \ldots + |i_1| + |i_2| + |i_3| + \ldots)} \]

Solution 1: Choose sparse tensors and find an appropriate list structure for the sequence of the computational steps.

Solution 2: Choose a low parameter approximation for the tensors (Parafac).

\[ W_{(i_1, i_2, i_3 \ldots)} = \sum_{k=1}^{r} u_{i_1}^k \cdot v_{i_2}^k \cdot w_{i_3}^k \ldots \]

In case of matrices this is a singular value decomposition (\( r = \text{rang}(W) \))

Solution 3: Instead of general linear- use local transformations in form of convolutions, a powerful concept from vision analytics.
Convolution parameters are learned. If $\sum A_{ii} \approx 1$ the convolution acts as a smoothing, (e.g. $y_t = \frac{1}{3} x_{t-1} + \frac{1}{3} x_t + \frac{1}{3} x_{t+1}$), if $\sum_i A_{ii} \approx 0$ it acts as an edge detection (e.g. $y_t = 1x_{t-1} - 2x_t + 1x_{t+1}$).
Deep Convolutional Networks exploit locality in time and space and combine local features to complex patterns.
For additional references see the working groups on the pages of Yann LeCun, Geoffrey Hinton and Yoshua Bengio
Simulation & Data Driven Modeling: Competitive or Cooperative Concepts?

**Competitive Concepts:**
- Simulation starts with a microscopic insight in a system (e.g. physical equations) and tries to compute it's macroscopic behavior.
- Data driven modeling starts with macroscopic observations of a system and tries to reconstruct it's microscopic causality (e.g. neural networks).

**Cooperative Concepts:**
- Neural networks can reduce residual errors of partially correct simulations (in sequential or parallel combined modeling).
- Neural networks may allow a fast approximation of simulations (e.g. if the simulation includes the solution of partial differential equations).
- Simulations allows the generation of training data, even in regions where you will not get observational data (too dangerous to run a machine in such a mode).
- Simulations are forward computations, neural nets allow forward-backward computations, providing gradient information for optimization tasks (in static / dynamic models).
- Neural nets can learn inverse models based on forward data which are generated by a simulation model (e.g. parameterizations of differential equations).
1) Analyze and optimize the measurement process

2) Building inverse models to identify the parameterization of differential equations.

3) Usage of deep feed forward networks to analyze particle traces in images???
Selected References


