Beam diagnostics at PITZ - Development of a neural network for beam recognition

Peter Spanknebel
Humboldt University of Berlin
Newtonstrasse 15, 12489 Berlin, Germany
Email: Peter.Spanknebel@web.de
DESY Zeuthen, Sept. 2004

The **P***hoto* **I***njector* **T***est Facility at DESY* **Z***euthen (PITZ) optimizes the electron source for the DESY projects on Free Electron Lasers (VUV-FEL, XFEL) and future linear colliders. Beam measurements at PITZ are mainly based on measurements of the beam profile using viewscreens (i.e. YAG) and CCD cameras. Typically the beam image recorded by a CCD camera is perturbed by noise. This noise disturbs the beam diagnostics. The goal is to find a solution for automatic beam recognition. This report demonstrates that neural networks can be used for this purpose.*

1 Beam diagnostics at PITZ

1.1 PITZ

For the FELs an intense electron source with small bunch size is needed. The **P**hoto **I**njector **T**est Facility at DESY **Z**euthen (PITZ) optimizes such an electron source for the DESY projects on Free Electron Lasers (VUV-FEL, XFEL) and on future linear colliders (e.g. TESLA, ILC).
A suitable laser produces the electrons by photoelectric effect out of the Cs_2Te cathode. The so called rf gun (**r**adio **f**requency gun) accelerates a bunch of photoelectrons with a strong electromagnetic field nearly to the speed of light. Thus the diverging effect of space charge is reduced. In addition a strong magnetic field focuses the electron bunch. The transverse beam profile, being important for the beam quality control, is to be measured with good accuracy. The beam recognition is an actual problem for the beam diagnostics.

1.2 Beam diagnostics

Besides other measurements the direct picturing of the beam (i.e. the laser or the electron beam) is an essential tool in the diagnostics. The pictures are recorded by different CCD cameras. There are two cameras for the laser beam and 5 cameras for the electron beam diagnostics. The cameras for the laser beam are situated at the *Virtual Cathode* where the optical path of the laser equals that to the real cathode.

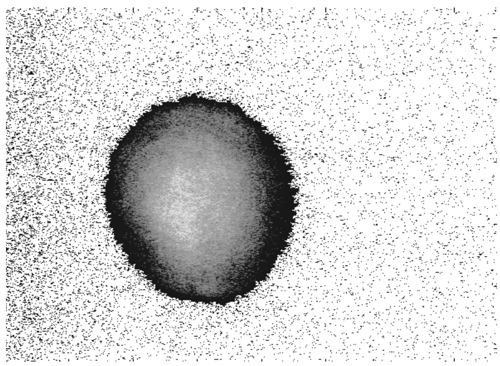

Fig. 1: *Picture from the Virtual Cathode recorded by a CCD camera. There was no background subtracted.*

The beam recorded by the cameras is typically perturbed by noise. This noise jitters from rather low to heavy and disturbs the beam diagnostics crucially. An example picture from the virtual cathode is given in Fig. 1.

Advantages	Disadvantages
- very powerful, neural networks can learn nearly everything you can put into numbers	- specialization on the operation and the range of trained values
- a net can approximate any function; no analytic expression is needed	- huge nets need high computing capacity

Tab. 1: *Advantages and Disadvantages*

2 Neural Networks

2.1 General description

Neural Networks are inspired by biological neural systems. They are composed of single units each performing a simple computation. The units are connected with links and each link carries a specific weight. Each unit gets an input value, operates on it in a particular way and passes its own output value to the successor units. A schematic view of the interactions of units is given in Fig. 2.

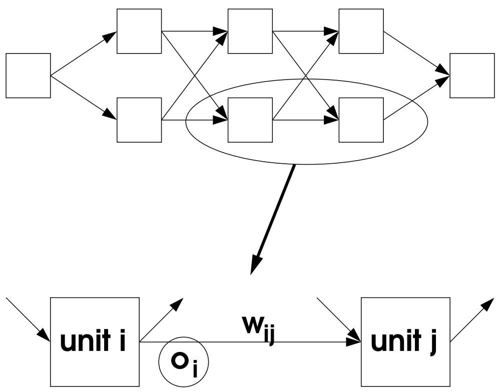

Fig. 2: *Schematic view of general units and their interaction. o_i is the output of unit i and w_{ij} the weight of the link between unit i and unit j.*

We can adjust the weights in order to teach (also called train) the network to perform a particular complex operation. Once a network is trained to perform a operation one can present input values to the network and receive the network's output. This output is an approximation of the real (unknown) output of the operation.
As neural networks are inspired by biological nervous systems the units of a net are supposed to work parallelly. Despite the fact that artifical neural networks are much faster concerning single operations of single units, the human brain beats machine computations because of the massive number of parallel operations of biological neurons [1].

Networks are organized in layers where each layer can contain several single units performing different single operations. The units of a layer are not connected among themselves but only to the units of other layers. Any neural network has to include one input layer and one output layer. In addition to that there can be several hidden layers if wanted or needed. Fig. 3 gives a schematic view of the general architecture of neural networks.

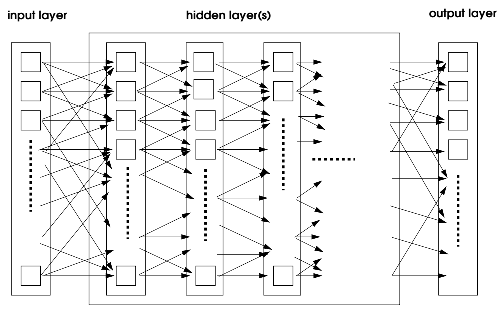

Fig. 3: *General architecture of neural networks.*

2.2 SNNS - Stuttgart Neural Network Simulator

SNNS[1] is a free software simulating neural networks. It was developed at the University of Stuttgart and is maintained at the University of Tuebingen. We chose to use this software to create and train our networks. The SNNS Software is very powerful and we did not use all features provided. The brief introduction given here is restricted on the features used for our solution of beam recognition and is based on [2].

[1] You can obtain this Software from www-ra.informatik.uni-tuebingen.de/SNNS

2.2.1 Operations in single units

In the SNNS-Software each unit j is able to perform three operations on its input. The first operation is to sum over all products of incoming outputs o_i with the corresponding weights w_{ij}. This sum is called the *net input* of the unit j.

$$\sum_i o_i \cdot w_{ij} = net_j \tag{1}$$

This net input net_j is then passed to the Activation Function $f_{act}(net_j)$ which performs a selected or self implemented operation on the net input. The activation function can also operate on the bias Θ, a unit-specific constant number. The result is the Activation Value a_j.

$$f_{act}(net_j, \Theta_j) = a_j \tag{2}$$

The activation value a_j is passed to the Output Function

$$f_{out}(a_j, \Theta_j) = o_j \tag{3}$$

leading to to final Output Value o_j of the unit j. Similar to the Activation Function the Output Function can be chosen by the user. A SNNS unit and its internal operations are sketched in Fig. 4.

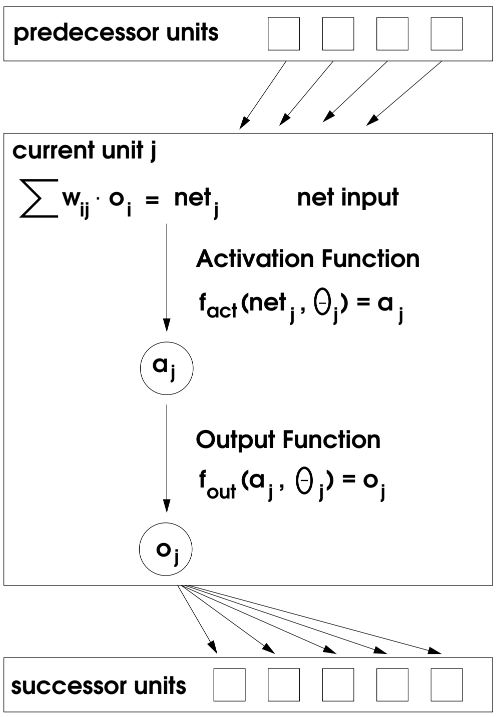

Fig. 4: *Schematic view of a unit in the SNNS.*

There are three functions for units mentioned below (see section 3) which are described in Tab. 2.

Name	Formula
Identity	$f(x_j) = x_j$
IdentityPlusBias	$f(x_j, \Theta_j) = x_j + \Theta_j$
Logistic	$f(x_j, \Theta_j) = \frac{1}{1+e^{-(x_j+\Theta_j)}}$

Tab. 2: *Some Activation and Output Functions of SNNS. Here the x_j stands either for net_j or a_j.*

2.2.2 Training and data preparation

In order to train the network we have to give sets of possible input values and the corresponding desired output values. A set of input and desired output (teaching output) values is called pattern. The available patterns are summarized in pattern files e.g. *training.pat*. A pattern file has a syntax sensitive header. Below this header the input and teaching output values are simply listed.

As the input values have propagated through the network the network's output is compared to the teaching output. The difference is called error. A routine searches the (in optimal case: global) minimum of the network's error function and adjusts the weights of the links accordingly. These routines are called learning functions.
A very fundamental learning function is the so called Backpropagation [2] (also known as *generalized delta-rule*), where the error of the network's output is sequentially propagated backwards through the network i.e. from the output units to the input units. Less general but very powerful in several situations is the Resilent Backpropagation (see [1] and [2]).

2.2.3 Advices for the training

For the training of a network one should divide the available data sets (i.e. input and teaching output) in three equal sets. One is the *training set*, which is used to adjust the weights of the network's links. Another set which is used during the training is called *validation set*. This set is not used to adjust the weights, but to validate how the network handles untrained data. The third set is used to control the fully trained network and therefore can be called *test set*. In order to keep the network's property to generalize in case of unlearned data values, the training (learning) should be stopped at the minimum error of the validation set (see Fig. 5). The error of the training set might decrease beyond

the minimum of the validation set (again see Fig. 5), but beyond the minimum the network is learning to reproduce the training set and not to approximate the corresponding operation.
One should specify the fact that the weights and biases of a network have to be initialized before the learning routine starts. The success of the learning process depends crucially on the starting configuration. A disadvantageous starting configuration of weights and biases may lead the learning routine in a suboptimal minimum of the error function.

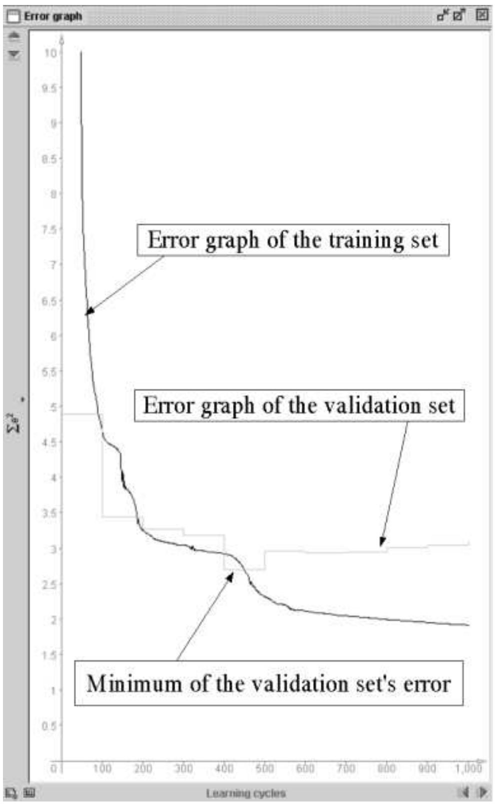

Fig. 5: *Error graph in SNNS.*

3 Networks and their performance

The units of the presented networks were all set to Identity for Output Function and in case of in- or output units even Identity for Activation Function. Ergo only the Activation Functions of units of hidden layers were changed to be different from Identity.
All units were connected feed forward, i.e. any unit of a layer was connected to all units of the successor layer. So called shortcut connections (see [2] for details) were not created.

Before starting with beam recognition some simple networks were created, trained and tested to understand the SNNS.

3.1 2D vector input and output

The goal was to create and train a network with vector input and vector output. Therefore a possible task was to train a network to compute two different functions. The input was chosen to be $x \,\epsilon\, [0, 3]$ and $y \,\epsilon\, [3, 6]$. The desired output was $sin(x)$ on the one hand and $\frac{sin(x) \cdot e^y}{8000}$ on the other hand. The network has two input and two output units and two hidden layers of 4 units each. The units of the first hidden layer were fixed to IdentityPlusBias for Activation Function, whereas the second hidden layer was completely Logistic.
Fig. 6 and Fig. 7 show the performance of the trained network.

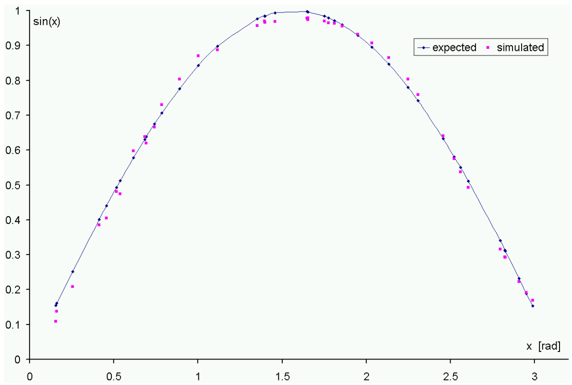

Fig. 6: *Performance of the network's sinus output.*

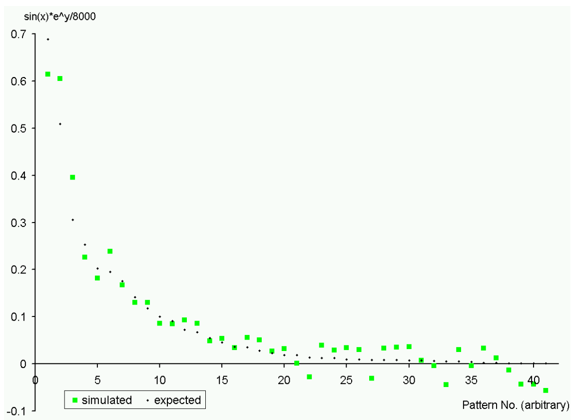

Fig. 7: *Plot of* $\frac{sin(x) \cdot e^y}{8000}$ *versus arbitrary pattern numbers.*

This network's performance is good enough to step towards a network for beam recognition.

3.2 Network for beam recognition

The task for this network was to define a rectangle (**R**egion **Of** **I**nterest) surrounding the beam spot. The network should handle the plain data of the CCD picture, i.e. no foregoing distinction between noise and beam should be needed.

3.2.1 Input and output values

An existing program called *Video Client* analyses the whole picture from the CCD camera. It computes the mean X and mean Y position of all entries ($< X >$ and $< Y >$) and the corresponding root mean square (X_{RMS} and Y_{RMS}) of these values. Besides this many other values are computed and may also be included to future networks. To show that the network can even ignore special input units if created adequately, another value ($Cov(X, Y)$) of the original CCD picture was included as input value.
The output was chosen to be the upper left edge (coordinates X_1 and Y_1) and the lower right edge (coordinates X_2 and Y_2) of the wanted rectangle (ROI).
For the training of this network 171 patterns were used.

3.2.2 Architecture

The network has 5 input units and 4 output units and 3 hidden layers.
The first hidden layer includes 19 Logistic units and one *LogisticNotInhibit* unit. The latter is a Logistic unit that skips any input of units named *Inhibit*. This is a possible way to exclude unwanted input such as the 5th input value ($Cov(X, Y)$) of this network. The unit of the 5th input value is therefore named *Inhibit*.
The second hidden layer includes 10 and the third hidden layer 20 Logistic units. The SNNS allows to display networks, Fig. 8 shows the picture of the specified network.

3.2.3 Performance

The performance of the trained network is easy to rate if you look at some tested CCD pictures and the network's rectangles.

The Figures 9 and 10 show that the network's output rectangles are not optimal positioned but at least surrounding the beam. Sometimes a rectangle even cuts the fringe of the beam's spot

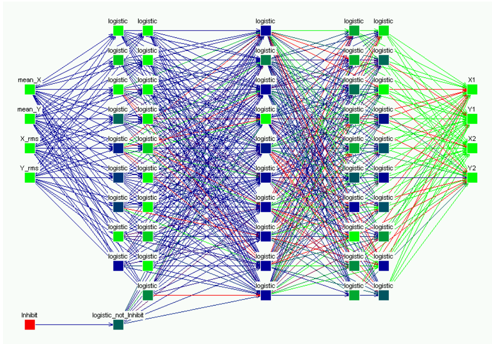

Fig. 8: *The trained network for beam recognition. Different colors indicate the value of weights resp. output values of units.*

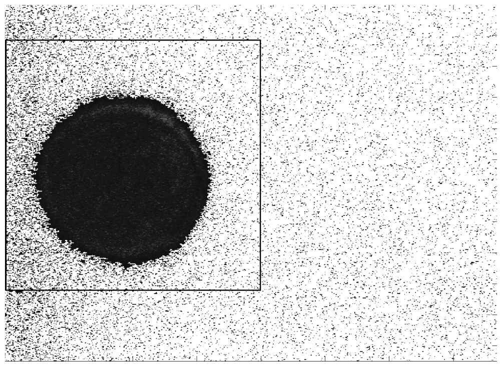

Fig. 9: *Performance of the network for beam recognition - example 1. The above picture was not seen by the network during training.*

(see Fig. 10).
Nevertheless it was possible to demonstrate, that neural networks can be used for beam recognition during the beam diagnostics at PITZ. As the network is very fast on computing its output, it might be used for online diagnostic during the measurements.

3.2.4 Outlook

Having the coordinates of the ROI, one can either apply the existing program *Video Client* directly on this region or apply annother network to improve the ROI before further diagnostics take place.
A completely different approach would be to train a network to classify between noise and signal. In general one of the great feautures of neural networks is their ability to classify.
Further on the presented network is not optimized concerning the number of units. A reduction of units would increase the speed of computation and might only slightly affect the

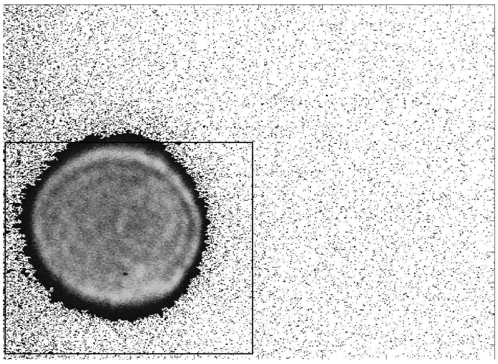

Fig. 10: *Performance of the network for beam recognition - example 3. The above picture was not seen by the network during training.*

network's output.

Acknowledgments

My deep thanks go to Janine Braeuer, my girlfriend who supported me greatly during the summerstudent programme. I appologise to my children Florian and Luca for the lack of patience and shared time.
I thank Velizar Miltchev who was my tutor and supported me at PITZ. Thanks to Johannes Vetter and Michael Beimforde for conspirative discussions at the lake and many helpful hints on Latex. Thanks for a wonderful time to Bagrat Petrosyan, Jang Hui Han and Karen Abrahamyan who welcomed me in their office and talked and laughed with me.
Further on I have to thank Mikhail Krasilnikov and Michael Beimforde for reading this report and giving helpful suggestions.
Finally I have to thank DESY to enalbe the summerstudent programme and an unknown person who resigned her or his position in the programme.

References

[1] Andreas Zell, Simulation Neuronaler Netze, Addison-Wesley, 1994, German

[2] SNNS-Stuttgart Neural Network Simulator, User Manual Version 4.2, A.Zell et al., available from http://www-ra.informatik.uni-tuebingen.de/downloads /SNNS/SNNSv4.2.Manual.pdf