Modeling of Continuous Casting of Steel with Artificial Neural Networks

Technical report

Revision 5, September 2012
(Revision 0: March 2012)

Tadej Kodelja, Igor Grešovnik

Contents:

1 INTRODUCTION

Artificial neural networks have been successfully used to solve industrial problems for many years. The essence of the artificial neural networks is to find the rule during learning to relate output values to input parameters. A crucial advantage of artificial neural networks is the calculation speed. Once the network is trained, the approximation is very fast.

In the present report we will focus on continuous casting process of steel. Continuous casting is currently the most common process in steel production. At present, approximately 90% of all steel is produced by this technique. We will first describe the continuous casting process and the equipment necessary to perform it [1]. Then we will present its physics behind it[5].

In the second chapter we will present artificial neural networks in general. We will describe the history of neural networks, define different types of artificial neural networks (ANN), describe how to build artificial neural networks, what are their components and how to use them[5]-[9]. We will try to clearly and concisely present the basics of artificial neural networks[12].

In the third chapter we will describe the use of artificial neural networks in an industrial problem. Physics-based numerical model for continuous casting of steel was developed in the Laboratory for Multiphase processes and is now in daily use by the Štore Steel company[4]. The simulator using this model takes about 90 seconds to calculate three basic outcomes (i.e. the metallurgical length, shell thickness and billet surface temperature) of the process.

As an alternative to the physics based model, we have built a surrogate neural network-based model on the basis of data generated by the simulator. The artificial neural network calculates the process outcomes for any combination of the input parameters in less than a second. The ANN model has been obtained by training the neural network on data that is generated in advance by the physics-based simulator. After building the model, it is validated by statistical methods and by parametric studies whose results are compared to industrial results. Since calculation of the response by is very fast, such a model can be used for optimization of process parameters according to arbitrary user defined criteria[9],[10], making possible to improve the process with respect to quality, cost, productivity, or energy consumption.

2 DESCRIPTION OF THE CONTINUOUS CASTING AND ITS PHYSICS

2.1 Continuous casting of steel

Continuous casting, also called strand casting, is the process whereby molten metal is solidified into a "semi-finished" billet, bloom, or slab for subsequent rolling in the finishing mills. Prior to the introduction of continuous casting in the 1950s, steel was poured into stationary moulds to form ingots. Since then, "continuous casting" has evolved to achieve improved yield, quality, productivity and cost efficiency. It allows lower-cost production of metal sections with better quality, due to the inherently lower costs of continuous, standardized production of a product, as well as providing increased control over the process through automation. This process is used most frequently to cast steel (in terms of tonnage cast). Aluminium and copper are also continuously cast[1].

2.1.1 Equipment and process

Molten metal is tapped into the ladle from furnaces. After undergoing any ladle treatments, such as alloying and degassing, and arriving at the correct temperature, the ladle is transported to the top of the casting machine. Usually the ladle sits in a slot on a rotating turret at the casting machine. One ladle is 'on cast' (feeding the casting machine) while the other is made ready, and is switched to the casting position when the first ladle is empty.
From the ladle, the hot metal is transferred via a refractory shroud (pipe) to a holding bath called a tundish. The tundish allows a reservoir of metal to feed the casting machine while ladles are switched, thus acting as a buffer of hot metal, as well as smoothing out flow, regulating metal feed to the moulds and cleaning the metal.
Metal is drained from the tundish through another shroud into the top of an open-base copper mould. The scheme of the mould is shown in Figure 1. The depth of the mould can range from 0.5m to 2m , depending on the casting speed and section size. The mould is water-cooled to solidify the hot metal directly in contact with it; this is the primary cooling process. It also oscillates vertically to prevent the metal sticking to the mould walls. A lubricant can also be added to the metal in the mould to prevent sticking, and to trap any slag particles – including oxide particles or scale – that may be present in the metal and bring them to the top of the pool to form a floating layer of slag. Often, the shroud is set so the hot metal exits it below the surface of the slag layer in the mould and is thus called a submerged entry nozzle (SEN). In some cases, shrouds may not be used between tundish and mould; in this case, interchangeable metering nozzles in the base of the tundish direct the metal into the moulds. Some continuous casting layouts feed several moulds from the same tundish.
In the mould, a thin shell of metal next to the mould walls solidifies before the middle section, now called a strand, exits the base of the mould into a spray chamber. The bulk of metal within the walls of the strand is still molten. The strand is immediately supported by closely spaced, water-cooled

rollers which support the walls of the strand against the ferrostatic pressure of the still-solidifying liquid within the strand. To increase the rate of solidification, the strand is sprayed with large amounts of water as it passes through the spray-chamber; this is the secondary cooling process. Final solidification of the strand may take place after the strand has exited the spray-chamber[4].

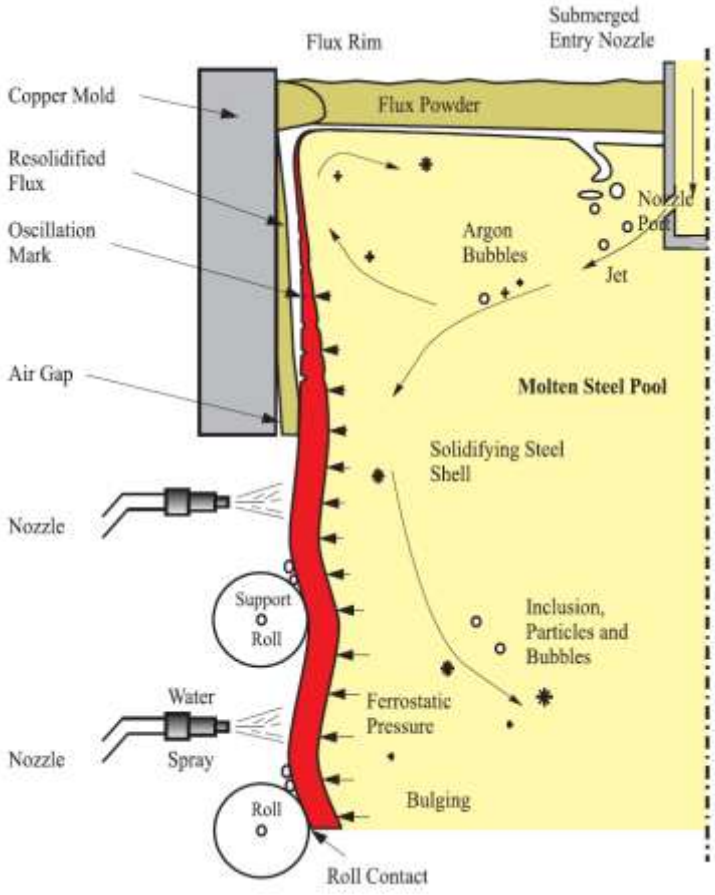

Figure 1: Physical phenomena in the mould.

It is here that the design of continuous casting machines may vary. This describes a 'curved apron' casting machine; vertical configurations are also used. In a curved apron casting machine, the strand exits the mould vertically and as it travels through the spray-chamber, the rollers gradually curve the strand towards the horizontal. In a vertical casting machine, the strand stays vertical as it passes through the spray-chamber. Moulds in a curved apron casting machine can be straight or curved, depending on the basic design of the machine.

In a true horizontal casting machine, the mould axis is horizontal and the flow of steel is horizontal from liquid to thin shell to solid (no bending). In this type of machine, either strand or mould oscillation is used to prevent sticking in the mould.

After exiting the spray-chamber, the strand passes through straightening rolls and withdrawal rolls. There may be a hot rolling stand after withdrawal to take advantage of the metal's hot condition to pre-shape the final strand. Finally, the strand is cut into predetermined lengths by mechanical shears or by travelling oxyacetylene torches, is marked for identification, and is taken either to a stockpile or to the next forming process[3]. The scheme of the casting machine is represented in Figure 2.

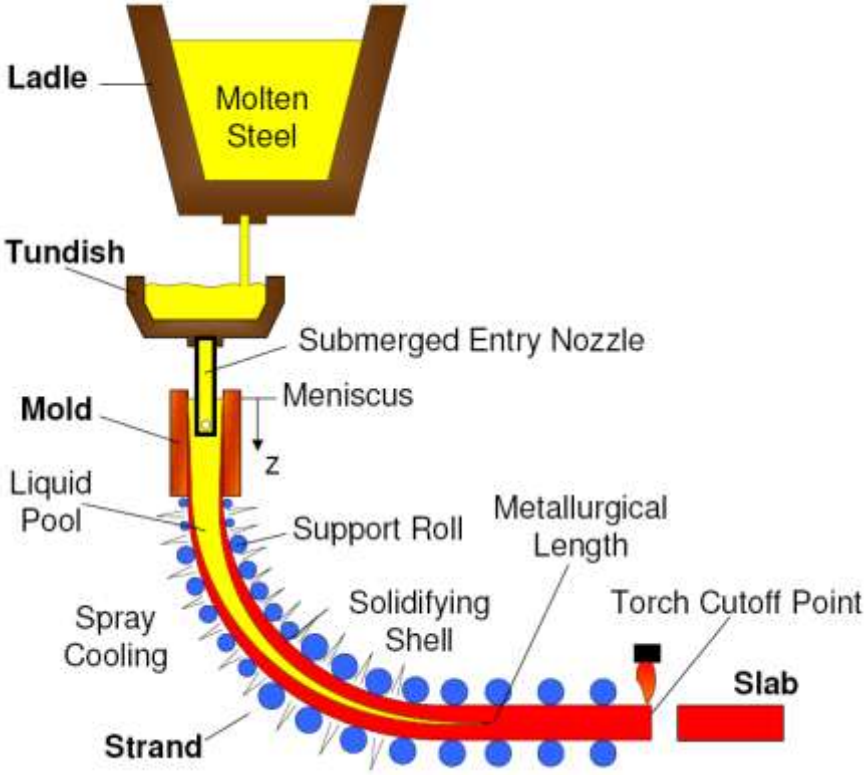

Figure 2: Scheme of the continuous casting process.

2.2 Numerical modeling in continuous casting

The macroscopic model calculates the steady temperature distribution in the strand as a function of the following process parameters: billet dimension, steel grade, casting temperature, casting velocity, primary, and two secondary cooling systems flows, pressures, temperatures, type and quantity of the casting powder, and the (non)application of the radiation shield and electromagnetic stirring. The Bennon-Incropera mixture continuum formulation is used for the physical model, solved by the recently developed meshless Local Radial Basis Function Collocation Method (LRBFCM). In this novel numerical method, the domain and boundary of interest are divided into overlapping influence areas. On each of them, the fields are represented by the multiquadrics radial basis function collocation on a related sub-set of nodes. Time-stepping is performed in an explicit way. The governing equations are solved in its strong form, i.e. no integrations are performed. The polygonisation is not present and the method is practically independent on the problem dimension[5].

2.2.1 Governing equation

Consider a connected fixed domain Ω with boundary Γ occupied by a liquid-solid phase change material described with the temperature dependent density $\rho_\wp$ of the phase $\wp$, temperature dependent specific heat at constant pressure $c_\wp$, thermal conductivity $k_\wp$, and the specific latent heat of the solid-liquid phase change h_m. The mixture continuum formulation of the enthalpy conservation for the assumed system is

$$\begin{aligned}&\frac{\partial}{\partial t}(\rho h)+\nabla\cdot(\rho\vec{v}h)=\\&\nabla\cdot(k\nabla T)+\nabla\cdot\left(\rho\vec{v}h-f_S^V\rho_S\vec{v}_S h_S-f_L^V\rho_L\vec{v}_L h_L\right)\end{aligned}\tag{1}$$

where the second term on the right-hand side is a correction term, needed to accommodate the mixture continuum formulation of the convective term. In continuation we neglect this term. In Eq.(1) mixture density and thermal conductivity are defined as

$$\rho = f_S^V\rho_S + f_L^V\rho_L,\tag{2}$$

$$k = f_S^V k_S + f_L^V k_L,\tag{3}$$

where $f_\wp^V$ represents the volume fraction of the phase $\wp$. The liquid volume fraction f_L^V might vary from 0 to 1 between solidus T_S and liquidus temperature T_L. Mixture velocity is defined as

$$\vec{v} = \left(f_S^V\rho_S\vec{v}_S + f_L^V\rho_L\vec{v}_L\right)/\rho\tag{4}$$

and mixture enthalpy is defined as

$$h = f_S^V h_S + f_L^V h_L.\tag{5}$$

The constitutive temperature-enthalpy relationships are

$$h_S = \int_{T_{ref}}^{T} c_S dT,\tag{6}$$

$$h_L = h_S(T) + \int_{T_S}^{T}(c_L - c_S)dT + h_m,\tag{7}$$

with T_{ref} standing for the reference temperature. Thermal conductivity and specific heat of the phases can arbitrarily depend on temperature[5].

2.2.2 Spatial discretization

The temperature field of a point in the billet is prescribed by the following three-dimensional vector in the Cartesian coordinate system

$$\mathbf{p} = x\mathbf{i}_x + y\mathbf{i}_y + z\mathbf{i}_z, \tag{8}$$

where x, y, z are coordinates and $\mathbf{i}_x, \mathbf{i}_y, \mathbf{i}_z$ are base vectors. The z coordinate measures the length of the inner radius of the casting machine. This Cartesian coordinate system represents the flat geometry, which is the geometrical approximation of the real curved casting process. The origin of the z coordinate coincides with the top side of the mould, and the base vector $\mathbf{i}_z$ coincides with the casting direction. The x coordinate measures the width (west-east direction) of the billet, perpendicular to the casting direction. Its origin coincides with the centre of the billet. The y coordinate measures the thickness (south-north direction) of the billet, perpendicular to the casting direction. Its origin coincides with the inner (south) side of the billet.
According to the heat transfer phenomena of the continuous casting of steel, the heat conduction in the casting direction can be neglected. The z coordinate is then parabolic, while the x and y coordinates are elliptic. The temperature field in the billet at a given time is described by the calculation of the cross-section (called infinite slice) temperature field of the billet. In this way the temperature field at a given z coordinate depends only on the slice history at the billet and its cooling intensity as a function of time. The slices beginning at the z_{start} longitudinal coordinate of casting and travel in the direction of the $\mathbf{i}_z$ base vector with the casting speed v. For calculating the cooling intensity of the slice as a function of time, we need the connection between the z coordinate of the casting machine and slice history t, which is in general

$$z(t) = \int_{t_{start}}^{t} v(t')\,dt' + z_{start}, \quad v(t) = \vec{v}(t)\cdot\mathbf{i}_z, \tag{9}$$

where t_{start} is the initial time of a slice. In the case when the casting speed is constant, we have the following simple connection between the z coordinate of the casting machine and the slice history t

$$t(z) = \frac{z - z_{start}}{v} + t_{start}. \tag{10}$$

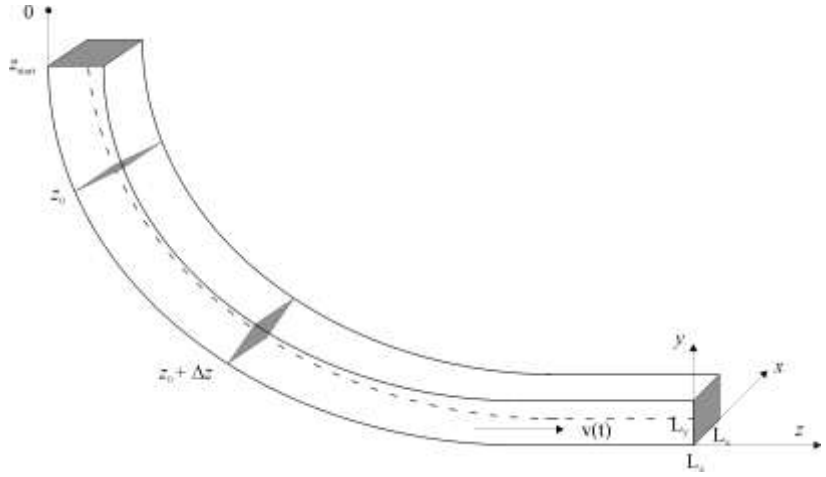

Figure 3: Slice traveling schematics at the billet.

In these paper, the simple Eq.(10) is used, since we are dealing with the steady-state solution of the casting process.
The prescribed simplified spatial discretization also simplifies the Eq.(1) by removing the convective terms. Thus the Eq. Error! Reference source not found. transforms into equation, defined on x-y plane

$$\frac{\partial}{\partial t}(\rho h) = \nabla \cdot (k \nabla T). \tag{11}$$

2.2.3 Boundary condition

The heat transport mechanisms in the mould take into account the heat transport mechanisms through the casting powder, across the air-gap (if it exists), to the mould surface, in the mould, and from the mould inner surface to the mould cooling water. The heat transport mechanisms in the secondary cooling zone take into account the effects of the casting velocity, strand surface temperature, spray nozzle type, spray water flow, temperature and pressure, radiation and cooling through the rolls contact. Different types of the rolls are considered (driving, passive, centrally cooled, externally cooled, etc.). The mentioned basic heat transfer mechanisms are modified with regard to running water and rolls stagnant water at relevant positions[5].

2.2.4 Solution procedure

We seek for mixture temperature at time $t_0 + \Delta t$ by assuming known initial temperature, velocity field, and boundary conditions at time t_0. The initial value of the temperature $T(\mathbf{p}, t)$ at a point with position vector $\mathbf{p}$ and time t_0 is defined through the known function T_0

$$T(\mathbf{p},t)=T_0(\mathbf{p});\ \mathbf{p}\in\Omega+\Gamma\,. \tag{12}$$

The boundary Γ is divided into not necessarily connected parts $\Gamma=\Gamma^D\cup\Gamma^N\cup\Gamma^R$ with Dirichlet, Neumann and Robin type boundary conditions, respectively. At the boundary point $\mathbf{p}$ with normal $\mathbf{n}_\Gamma$ and time $t_0\le t\le t_0+\Delta t$, these boundary conditions are defined through known functions T_Γ^D, T_Γ^N, T_Γ^R, $T_{\Gamma ref}^R$

$$T=T_\Gamma^D;\ \mathbf{p}\in\Gamma^D\,, \tag{13}$$

$$\frac{\partial}{\partial n_\Gamma}T=T_\Gamma^N;\ \mathbf{p}\in\Gamma^N\,, \tag{14}$$

$$\frac{\partial}{\partial n_\Gamma}T=T_\Gamma^R\left(T-T_{\Gamma ref}^R\right);\ \mathbf{p}\in\Gamma^R\,. \tag{15}$$

The numerical discretization of Eq.(11), using explicit (Euler) time discretization has the form

$$\frac{\partial(\rho h)}{\partial t}\approx\frac{\rho h-\rho_0 h_0}{\Delta t}=\nabla\cdot\left(k_0\nabla T_0\right). \tag{16}$$

From Eq.(16) the unknown function value h_l in domain node $\mathbf{p}_l$ can be calculated as

$$h_l=h_{0l}+\frac{\Delta t}{\rho_0 c_0}\left(\nabla k_{0l}\cdot\nabla T_{0l}+k_{0l}\cdot\nabla^2 T_{0l}\right). \tag{17}$$

The spatial derivatives in Eq.(17) are approximated by the LRBFCM. In the LRBFCM, the representation of unknown function value over a set of ${}_lN$ (in general) non-equally spaced nodes ${}_l\mathbf{p}_n;\ n=1,2,...,{}_lN$ is made in the following way

$$\phi(\mathbf{p})\approx\sum_{k=1}^{{}_lK}{}_l\psi_k(\mathbf{p})\,{}_l\alpha_k\,, \tag{18}$$

where ${}_l\psi_k$ stands for the shape functions, ${}_l\alpha_k$ for the coefficients of the shape functions, and ${}_lK$ represents the number of the shape functions. The left lower index on entries of expression (18) represents the influence domain (subdomain or support) ${}_l\omega$ on which the coefficients ${}_l\alpha_k$ are determined. The influence domains ${}_l\omega$ can in general be contiguous (overlapping) or non-contiguous (non-overlapping). Each of the influence domains ${}_l\omega$ includes ${}_lN$ nodes of which ${}_lN_\Omega$ can in general be in the domain and ${}_lN_\Gamma$ on the boundary, i.e. ${}_lN={}_lN_\Omega+{}_lN_\Gamma$. The total number of all nodes $\mathbf{p}_n$ is equal $N=N_\Omega+N_\Gamma$ of which N_Γ are located on the boundary and N_Ω are located in the domain. The influence domain of the node ${}_l\mathbf{p}$ is defined with the nodes having the nearest ${}_lN-1$ distances to the node ${}_l\mathbf{p}$. The five nodded ${}_lN=5$ influence domains are used in this paper. The coefficients are calculated by the collocation (interpolation).

Let us assume the known function values ${}_l\phi_n$ in the nodes ${}_l\mathbf{p}_n$ of the influence domain ${}_l\omega$. The collocation implies

$$\phi({}_l\mathbf{p}_n) = \sum_{k=1}^{{}_lN} {}_l\psi_k({}_l\mathbf{p}_n)\,{}_l\alpha_k \ . \tag{19}$$

For the coefficients to be computable, the number of the shape functions has to match the number of the collocation points ${}_lK = {}_lN$, and the collocation matrix has to be non-singular. The system of Eq.(19) can be written in a matrix-vector notation

$${}_l\underline{\boldsymbol{\Psi}}\,{}_l\boldsymbol{\alpha} = {}_l\boldsymbol{\Phi};\ {}_l\underline{\psi}_{kn} = {}_l\psi_k({}_l\mathbf{p}_n),\ {}_l\phi_n = \phi({}_l\mathbf{p}_n). \tag{20}$$

The coefficients ${}_l\boldsymbol{\alpha}$ can be computed by inverting the system

$${}_l\boldsymbol{\alpha} = {}_l\underline{\boldsymbol{\Psi}}^{-1}\,{}_l\boldsymbol{\Phi}\ . \tag{21}$$

By taking into account the expressions for the calculation of the coefficients ${}_l\boldsymbol{\alpha}$, the collocation representation of temperature $\phi(\mathbf{p})$ on subdomain ${}_l\omega$ can be expressed as

$$\phi(\mathbf{p}) \approx \sum_{k=1}^{{}_lN} {}_l\psi_k(\mathbf{p}) \sum_{n=1}^{{}_lN} {}_l\underline{\psi}^{-1}_{\mathrm{kn}}\,{}_l\phi_n \ . \tag{22}$$

The first partial spatial derivatives of $\phi(\mathbf{p})$ on subdomain ${}_l\omega$ can be expressed as

$$\frac{\partial}{\partial p_\varsigma}\phi(\mathbf{p}) \approx \sum_{k=1}^{{}_lN} \frac{\partial}{\partial p_\varsigma}\,{}_l\psi_k(\mathbf{p}) \sum_{n=1}^{{}_lN} {}_l\underline{\psi}^{-1}_{\mathrm{kn}}\,{}_l\phi_n;\ \varsigma = x, y \ . \tag{23}$$

The second partial spatial derivatives of $\phi(\mathbf{p})$ on subdomain ${}_l\omega$ can be expressed as

$$\frac{\partial^2}{\partial p_\varsigma\, p_\xi}\phi(\mathbf{p}) \approx \sum_{k=1}^{{}_lN} \frac{\partial^2}{\partial p_\varsigma\, p_\xi}\,{}_l\psi_k(\mathbf{p}) \sum_{n=1}^{{}_lN} {}_l\underline{\psi}^{-1}_{\mathrm{kn}}\,{}_l\phi_n; \quad \varsigma, \xi = x, y \ . \tag{24}$$

The radial basis functions, such as multiquadrics, can be used for the shape functions

$${}_l\psi_k(\mathbf{p}) = \left[{}_lr_k^2(\mathbf{p}) + c^2\right]^{1/2}, \tag{25}$$

where c represents the shape parameter. The explicit values of the involved first and second derivatives of $\psi_k(\mathbf{p})$ are

$$\frac{\partial}{\partial p_\varsigma}\,{}_l\psi_k(\mathbf{p}) = \frac{p_\varsigma - {}_lp_{k\varsigma}}{\left({}_lr_k^2 + c^2\right)^{1/2}}, \quad \varsigma = x, y, \tag{26}$$

$$\frac{\partial^2}{\partial p_\varsigma^2}\,{}_l\psi_k(\mathbf{p}) = \frac{{}_l r_k^2 - \left(p_\varsigma - {}_l p_{k\varsigma}\right)^2 + c^2}{\left({}_l r_k^2 + c^2\right)^{3/2}}, \quad \varsigma = x, y\,. \tag{27}$$

3 APPROXIMATIVE NUMERICAL MODELS BASED ON ARTIFICIAL NEURAL NETWORK

3.1 Neural network basics

3.1.1 What is neural network

An artificial neural network is an information-processing system that has certain performance characteristics in common with biological neural networks. Artificial neural networks have been developed as generalizations of mathematical models of human cognition or neural biology, based on the assumptions that:

- Information processing occurs at many simple elements called neurons.
- Signals are passed between neurons over connection links.
- Each connection link has an associated weight, which, in a typical neural net, multiplies the signal transmitted.
- Each neuron applies an activation function (usually nonlinear) to its net input (sum of weighted input signals) to determine its output signal.

A neural network is characterized by:

- Its pattern of connections between the neurons (architecture),
- Its method of determining the weights on the connections (training),
- Its activation function.

A neural net consists of a large number of simple processing elements called neurons. Each neuron is connected to other neurons by means of directed communication links, each with an associated weight. The weights represent information being used by the net to solve a problem. Neural nets can be applied to a wide variety of problems, such as storing and recalling data or patterns, classifying patterns, performing general mappings from input patterns to output patterns, grouping similar patterns, or finding solutions to constrained optimization problems[6].
Each neuron has an internal state, called its activation level, which is a function of the inputs it has received. Typically, a neuron sends its activation as a signal to several other neurons. It is important to note that a neuron can send only one signal at a time, although that signal is broadcast to several other neurons[6].
For example, neuron *Y*, illustrated in Figure 4, that receives inputs from neurons X_1, X_2, and X_3. The activations (output signals) of these neurons are X_1, X_2, and X_3, respectively. The weights on the

connections from X_1, X_2, and X_3 to neuron Y are w_1, w_2, and w_3, respectively. The net input, y_in, to neuron Y is the sum of the weighted signals from neurons X_1, X_2, and X_3.

$$y\_in = w_1x_1 + w_2x_2 + w_3x_3 \tag{28}$$

The activation y of neuron Y is given by some function of its net input, $y = f(y\_in)$, e.g., the logistic sigmoid function (an S-shaped curve) or any of a number of other activation functions.

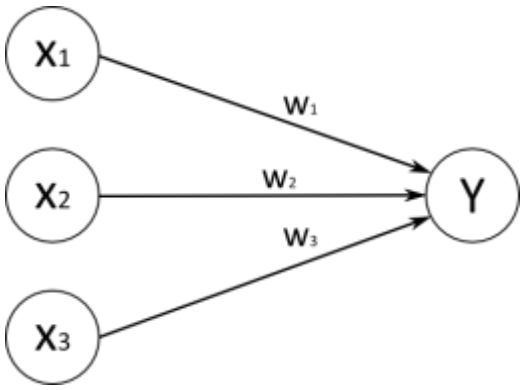

Figure 4: A simple artificial neuron

Neuron Y is further connected to neurons Z_1 and Z_2, with weights v_1 and v_2, respectively, as shown in Figure 5. Neuron Y sends its signal y to each of these units. However, in general, the values received by neurons Z_1 and Z_2 will be different, because each signal is scaled by the appropriate weight, v_1 or v_2. In a typical net, the activations z_1 and z_2 of neurons Z_1 and Z_2 would depend on inputs from several or even many neurons[6].

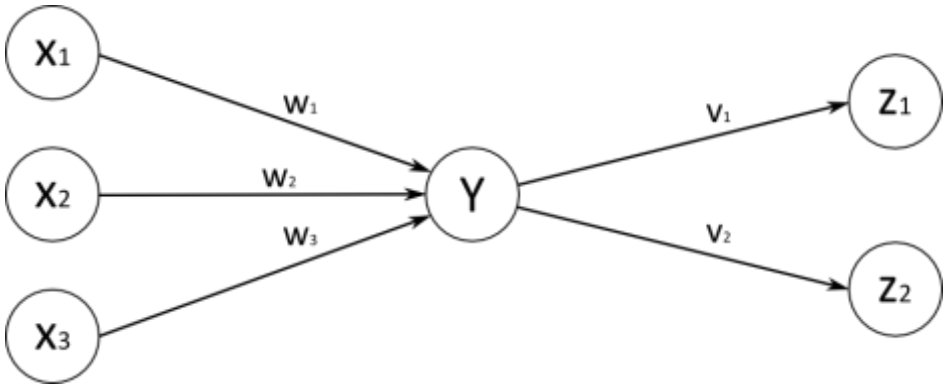

Figure 5: A simple neural network

3.1.2 History of neural networks

The history of neural networks can be traced back to the work of trying to model the neuron. The first model of a neuron was by physiologists, McCulloch and Pitts (1943). The model they created

had two inputs and a single output. McCulloch and Pitts noted that a neuron would not activate if only one of the inputs was active. The weights for each input were equal, and the output was binary. Until the inputs summed up to a certain threshold level, the output would remain zero. The McCulloch and Pitts neuron has become known today as a logic circuit

The perceptron was developed as the next model of the neuron by Rosenblatt (1958), as seen in Figure 6. Rosenblatt, who was a physiologist, randomly interconnected the perceptrons and used trial and error to randomly change the weights in order to achieve "learning." Ironically, McCulloch and Pitts' neuron is a much better model for the electrochemical process that goes on inside the neuron than the perceptron, which is the basis for the modern day field of neural networks[7].

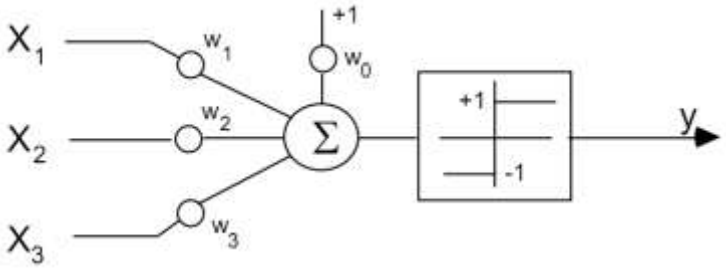

Figure 6: The perceptron

Widrow and Hoff (1960) developed a mathematical method for adapting the weights. Assuming that a desired response existed, a gradient search method was implemented, which was based on minimizing the error squared. This algorithm would later become known as LMS, or Least Mean Squares. LMS, and its variations, has been used extensively in a variety of applications, especially in the last few years. This gradient search method provided a mathematical method for finding an answer that minimized the error.

Werbos (1974) was first to develop the back propagation algorithm. It was then independently rediscovered by Parker (1985) and by Rumelhart and McClelland (1986), simultaneously. Back propagation is a generalization of the Widrow-Hoff LMS algorithm and allowed perceptrons to be trained in a multilayer configuration, thus a n-1 node neural network could be constructed and trained. The weights are adjusted based on the error between the output and some known desired output. As the name suggests, the weights are adjusted backwards through the neural network, starting with the output layer and working through each hidden layer until the input layer is reached. The back propagation algorithm changes the schematic of the perceptron by using a sigmoidal function as the squashing function. Earlier versions of the perceptron used the sign function. The advantage of the sigmoidal function over the sign function is that the sigmoidal function is differentiable. This permits the back propagation algorithm to transfer the gradient information through the nonlinear squashing function, allowing the neural network to converge to alocal minimum[8].

3.1.3 A brief description of the artificial network parameters

3.1.3.1 Learning rate

The learning rate determines the amount of correction term that is applied to adjust the neuron weights during training. Small values of the learning rate increase learning time but tend to decrease the chance of overshooting the optimal solution. At the same time, they increase the likelihood of becoming stuck at local minima. Large values of the learning rate may train the network faster, but may result in no learning occurring at all. The adaptive learning rate varies according to the amount of error being generated. The larger the error, the smaller the values and vice-versa. Therefore, if the ANN is heading towards the optimal solution it will accelerate. Correspondingly, it will decelerate when it is heading away from the optimal solution.

3.1.3.2 Momentum

The momentum value determines how much of the previous corrective term should be remembered and carried on in the current training. The larger the momentum value, the more emphasis is placed on the current correction term and the less on previous terms. It serves as a smoothing process that 'brakes' the learning process from heading in an undesirable direction.

3.1.3.3 Training epoch

A training epoch consists of one learning trial on each pattern pair in the environment. On each trial, the input is presented, the corresponding output is computed, and the weights are updated. Patterns may be presented in fixed sequential order or in permuted order within each epoch.

3.1.3.4 Input noise

Random noise is used to perturb the error surface of the neural net to jolt it out of local minima. It also helps the ANN to generalize and avoid curve fitting.

3.1.3.5 Training and testing tolerance

The training tolerance is the amount of accuracy that the network is required to achieve during its learning stage on the training data set. The testing tolerance is the accuracy that will determine the predictive result of the ANN on the test data set.

3.2 Neural network types

There are many types of artificial neural networks. An artificial neural network is a computational simulation of a biological neural network. These models mimic the real life behaviour of neurons and the electrical messages they produce between input (such as from the eyes or nerve endings in the hand), processing by the brain and the final output from the brain (such as reacting to light or from sensing touch or heat). There are other ANNs which are adaptive systems used to model things such as environments and population[11].

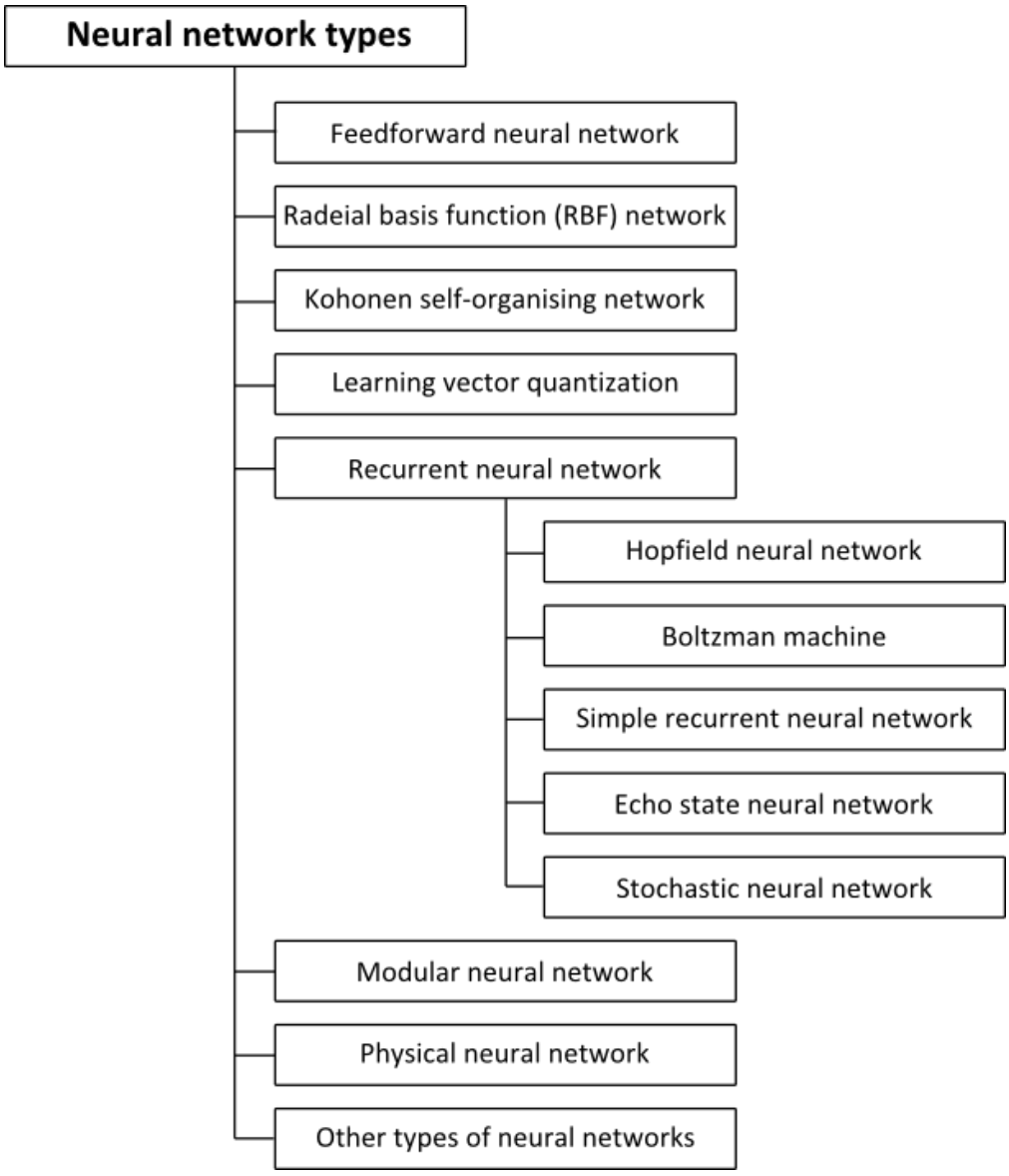

Figure 7: Neural network types

3.2.1 Feedforward neural network

The first term, "feedforward" describes how this neural network processes and recalls patterns. In a feedforward neural network, neurons are only connected foreword. Each layer of the neural network contains connections to the next layer (for example, from the input to the hidden layer), but there are no connections back. This differs from the Hopfield neural network. The Hopfield neural network is fully connected, and its connections are both forward and backward.
The term "back-propagation" describes how this type of neural network is trained. Back-propagation is a form of supervised training. When using a supervised training method, the network must be provided with both sample inputs and anticipated outputs. The anticipated outputs are

compared against the actual outputs for given input. Using the anticipated outputs, the back-propagation training algorithm then takes a calculated error and adjusts the weights of the various layers backwards from the output layer to the input layer.
The back-propagation and feedforward algorithms are often used together; however, this is by no means a requirement. It would be quite permissible to create a neural network that uses the feedforward algorithm to determine its output and does not use the back-propagation training algorithm. Similarly, if you choose to create a neural network that uses back-propagation training methods, you are not necessarily limited to a feedforward algorithm to determine the output of the neural network[12].

3.2.1.1 The structure of the feedforward neural network

Figure 8 illustrates a typical feedforward neural network with a single hidden layer.

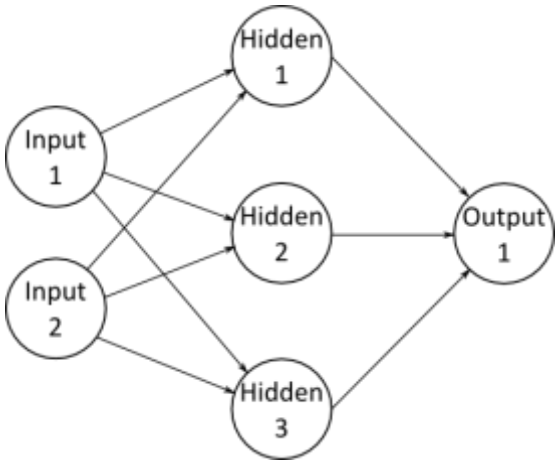

Figure 8: A typical feedforward neural network (single hidden layer)

3.2.1.2 The input layer

The input layer is the conduit through which the external environment presents a pattern to the neural network. Once a pattern is presented to the input layer, the output layer will produce another pattern. In essence, this is all the neural network does. The input layer should represent the condition for which we are training the neural network. Every input neuron should represent some independent variable that has an influence over the output of the neural network.

3.2.1.3 The output layer

The output layer of the neural network is what actually presents a pattern to the external environment. The pattern presented by the output layer can be directly traced back to the input layer. The number of output neurons should be directly related to the type of work that the neural network is to perform.
To determine the number of neurons to use in your output layer, you must first consider the intended use of the neural network. If the neural network is to be used to classify items into groups, then it is often preferable to have one output neuron for each group that input items are to be assigned into. If the neural network is to perform noise reduction on a signal, then it is likely that the number of input neurons will match the number of output neurons. In this sort of neural

network, you will want the patterns to leave the neural network in the same format as they entered[12].

3.2.2 Feedforward radial basis function (RBF) network

Feedforward neural networks with a single hidden layer that use radial basis activation function for hidden neurons are called radial basis function (RBF) networks.

3.2.2.1 RBF network structure

A typical radial-basis-function neural network is shown in Figure 9. The RBF neural network has an input layer, a radial basis hidden layer, and an output layer.

Parameters c_{ij}, λ_{ij} are centers and standard deviations of radial basis activation functions. Commonly used radial basis activation functions are Gaussian and multiquadratic. The Gaussian function shown in Figure 10 is given by

$$\sigma(\gamma) = \exp\left(-\gamma^2\right). \tag{29}$$

The multiquadratic function shown in Figure 11 is given by

$$\sigma(\gamma) = \frac{1}{\left(c^2 + \gamma^2\right)}, \alpha > 0, \tag{30}$$

where c is a constant.

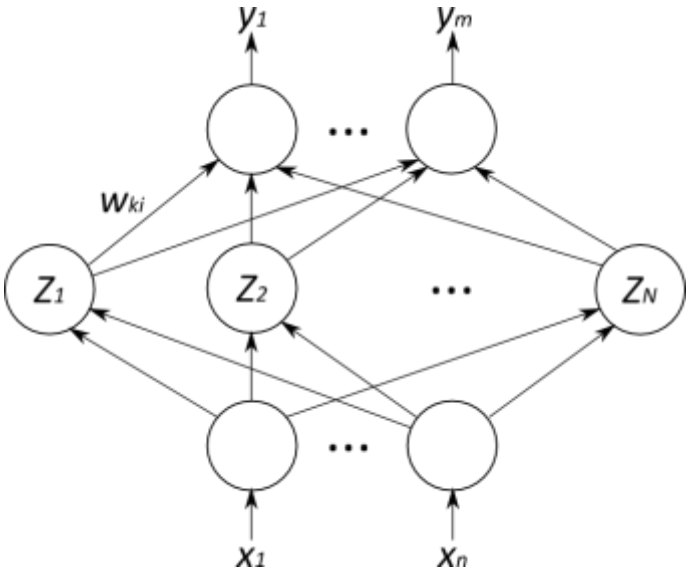

Figure 9: RBF neural network.

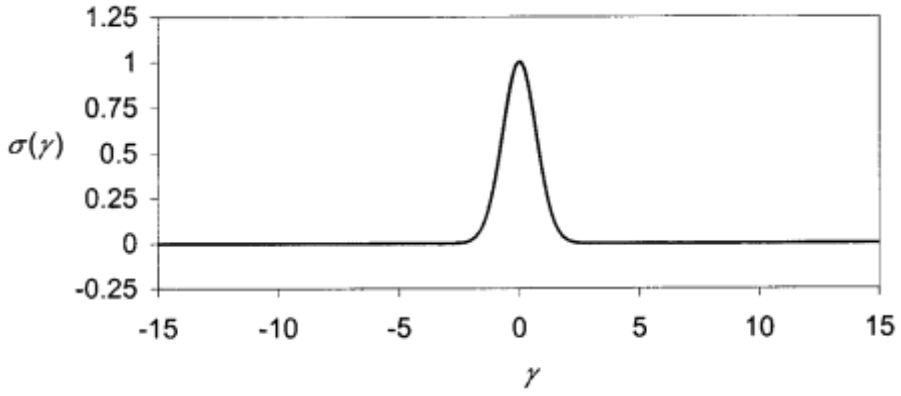

Figure 10: Gaussian function.

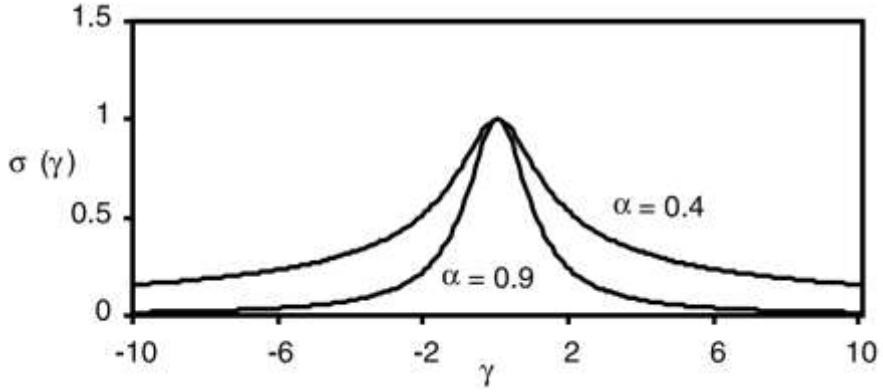

Figure 11: Multiquadratic function with $c = 1$.

3.2.3 Kohonen self-organizing map (SOM)

The self-organizing map, sometimes called a Kohonen neural network, is named after its creator, Tuevo Kohonen.

The self-organizing map differs considerably from the feedforward back-propagation neural network in both how it is trained and how it recalls a pattern. The self-organizing map does not use an activation function or a threshold value.

In addition, output from the self-organizing map is not composed of output from several neurons; rather, when a pattern is presented to a self-organizing map, one of the output neurons is selected as the "winner." This "winning" neuron provides the output from the self-organizing map. Often, a "winning" neuron represents a group in the data that is presented to the self-organizing map.

The most significant difference between the self-organizing map and the feedforward back-propagation neural network is that the self-organizing map trains in an unsupervised mode. This means that the self-organizing map is presented with data, but the correct output that corresponds to the input data is not specified.

3.2.3.1 The structure of Self organizing map

The self-organizing map works differently than the feedforward neural. The self-organizing map only contains an input neuron layer and an output neuron layer. There is no hidden layer in a self-organizing map.
The input to a self-organizing map is submitted to the neural network via the input neurons. The input neurons receive numbers that make up the input pattern to the network. A self-organizing map requires that the inputs be normalized to fall between -1 and 1. Presenting an input pattern to the network will cause a reaction from the output neurons.
The output of a self-organizing map is very different from the output of a feedforward neural network. In a self-organizing map, only one of the output neurons actually produces a value. Additionally, this single value is either true or false. Therefore, the output from the self-organizing map is usually the index of the neuron that fired. The structure of a typical self-organizing map is shown in Figure 12.

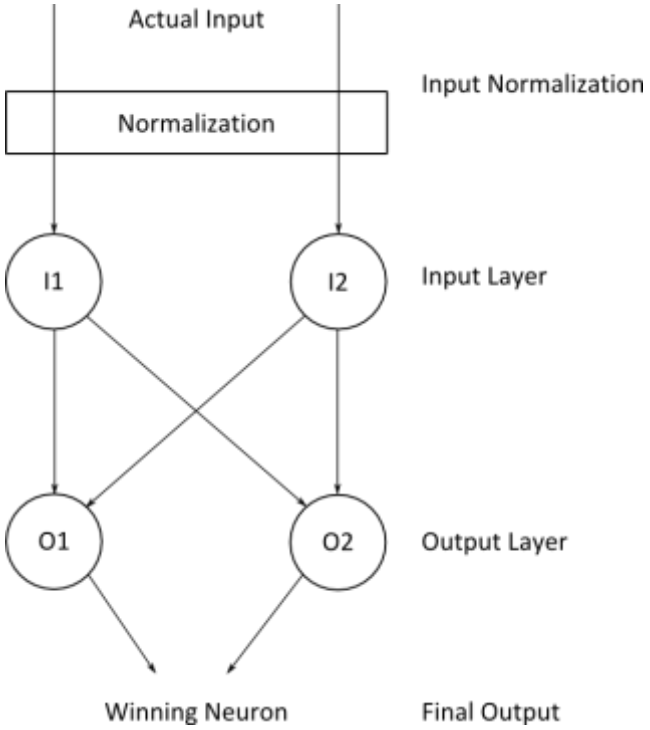

Figure 12: A self-organizing map[12].

3.2.4 Hopfield neural network

The Hopfield neural network is perhaps the simplest type of neural network. The Hopfield neural network is a fully connected single layer, autoassociative network. This means it has a single layer in which each neuron is connected to every other neuron. Autoassociative means that if the neural network recognizes a pattern, it will return that pattern[12].
A Hopfield network, with four neurons, is shown in Figure 13.

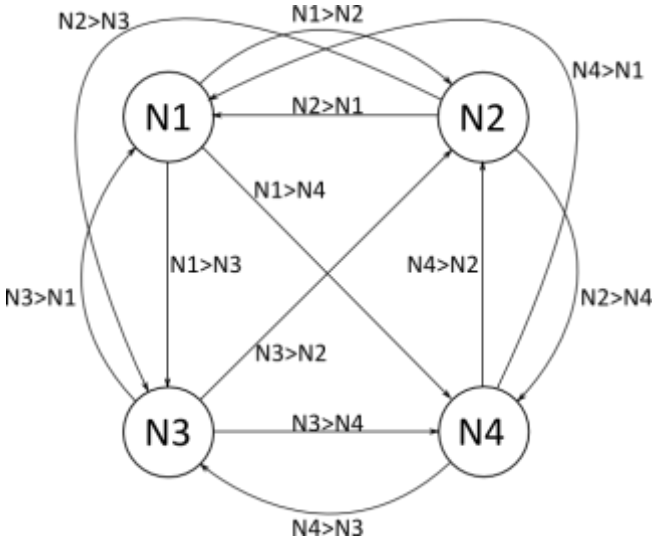

Figure 13: A Hopfield neural network with 12 connections

For an autoassociative net, the training input and target output vectors are identical. The process of training is often called storing the vectors, which may be binary or bipolar. A stored vector can be retrieved from distorted or partial (noisy) input if the input is sufficiently similar to it. The performance of the net is judged by its ability to reproduce a stored pattern from noisy input; performance is, in general, better for bipolar vectors than for binary vectors.
It is often the case that, for autoassociative nets, the weights on the diagonal (those which would connect an input pattern component to the corresponding component in the output pattern) are set to zero. Setting these weights to zero may improve the net's ability to generalize (especially when more than one vector is stored in it) or may increase the biological plausibility of the. Setting them to zero is necessary for extension to the iterative case or if the delta rule is used (to prevent the training from producing the identity matrix for the weights)[6].

3.2.5 Boltzman machine

Boltzmann machine neural nets were introduced by Hinton and Sejnowski (1983). The states of the units are binary valued, with probabilistic state transitions. The configuration of the network is the vector of the states of the units. The Boltzmann machine described in this section has fixed weights W_{ij}, which express the degree of desirability that units X_i and X_j, both be "on."
In applying Boltzmann machines to constrained optimization problems the weights represent the constraints of the problem and the quantity to be optimized.
The architecture of a Boltzmann machine is quite general, consisting of a set of units (X_i and X_j) and a set of bidirectional connections between pairs of units. If units X_i and X_j are connected, $w_{ij} \neq 0$.
The bidirectional nature of the connection is often represented as $w_{ij} = w_{ji}$.
The state x_i of unit X_i is either 1 ("on") or 0 ("off"). The objective of the neural net is to maximize the consensus function

$$C = \sum_i \left[\sum_{j \le i} w_{ij} x_i x_j \right]. \tag{31}$$

The sum runs over all units of the net.
The net finds this maximum by letting each unit attempt to change its state. The attempts may be made either sequentially or in parallel.
The change in consensus if unit X_i were to change its state (from 1 to 0 or from o to 1) is

$$\Delta C(i) = [1 - 3x_i] \left[w_{ij} + \sum_{j \ne i} w_{ij} x_j \right], \tag{32}$$

where x_i is the current state of unit X_i.
However, unit X_i does not necessarily change its state, even if doing so would increase the consensus of the net. The probability of the net accepting a change in state for unit X_i is

$$A(i,T) = \frac{1}{1 + \exp\left(-\frac{\Delta C(i)}{T} \right)}. \tag{33}$$

The control parameter T is gradually reduced as the net searches for a maximal consensus. Lower values of T make it more likely that the net will accept a change of state that increases its consensus and less likely that it will accept a change that reduces its consensus[6].

3.2.6 Simple recurrent network

Unlike the recurrent nets with symmetric weights or the feedforward nets, these nets do not necessarily produce a steady-state output. In this section, we consider a simple recurrent net that can be used to learn strings of characters. This net can be considered a "partially recurrent" net, in that most of the connections are feedforward only. A specific group of units receives feedback signals from the previous time step. These units are known as *context units*. The weights on the feedback connections to the context units are fixed, and information processing is sequential in time, so training is essentially no more difficult than for a standard back-propagation net.

3.2.6.1 The structure of simple recurrent network

The architecture for a simple recurrent net is as shown in Figure 14.

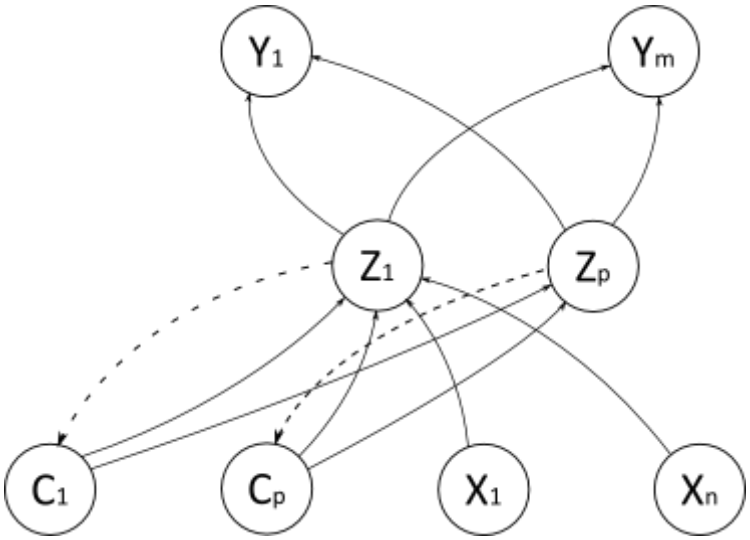

Figure 14: Simple recurrent network

3.2.6.2 Algorithm

At time *t*, the activations of the context units are the activations (output signals) of the hidden units at the previous time step. The weights from the context units to the hidden units are trained in exactly the same manner as the weights from the input units to the hidden units. Thus, at any time step, the training algorithm is the same as for standard back-propagation.

3.3 Training data

3.3.1 Problems with training data

3.3.1.1 Not enough training samples

If we do not have enough training data samples, we can't sufficiently cover the space. This means, that we have some areas in that space that are not covered with data. In the Figure 15 we can see approximation of neural networks trained with enough data samples and with five samples. The difference is in the approximation in the space between training data is obvious.

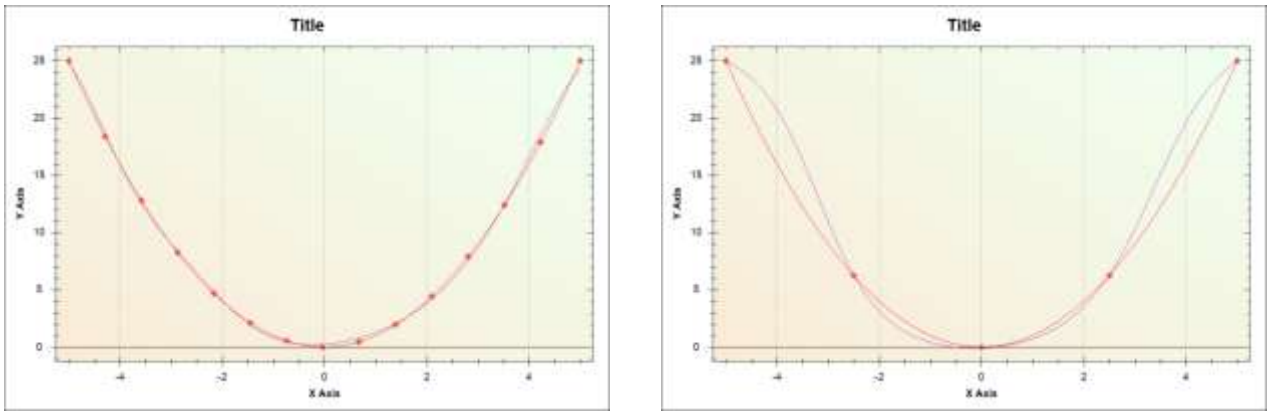

Figure 15: Approximation of neural network: (left) trained with enough training data samples, (right) trained with five samples.

3.3.1.2 Training data not equally distributed

From picture Figure 16 we can see the approximation of neural network when the data are equally distributed in space and when the data are distributed in clouds. The approximation is much better on the left picture.

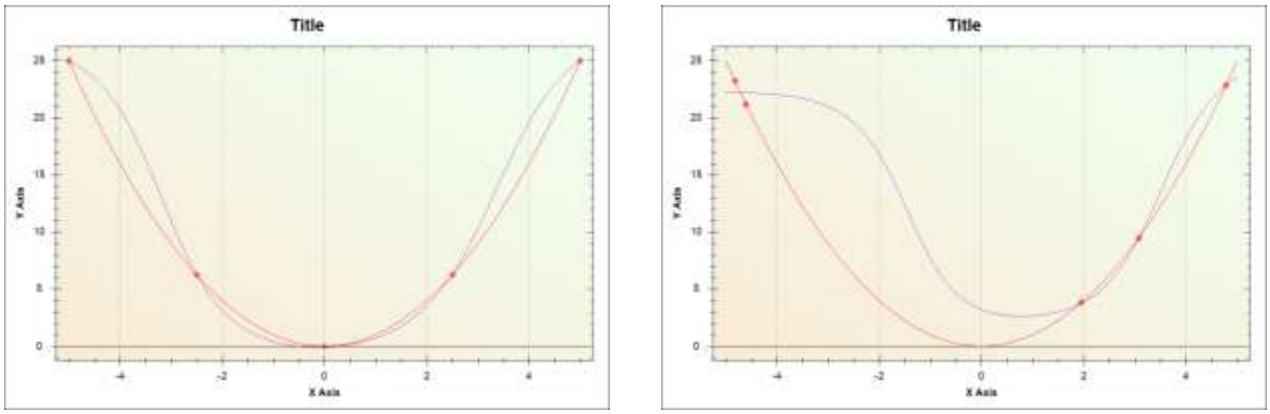

Figure 16: Approximation of neural network: (left) training data samples are equally distributed, (right) training data samples are distributed in clouds

3.3.1.3 Data duplication

When we prepare data for training we have to delete all duplicated data.

3.3.1.4 Wrong data

In the Figure 17 we can see training data samples of Boron concentration from real process. From the graph we can see, that most of these concentrations are in the small range, but some of them are out. We need to know if these overshoots are correct measurements or some errors in data. If we use these overshoots in the training procedure, the range for error calculation will be much bigger.

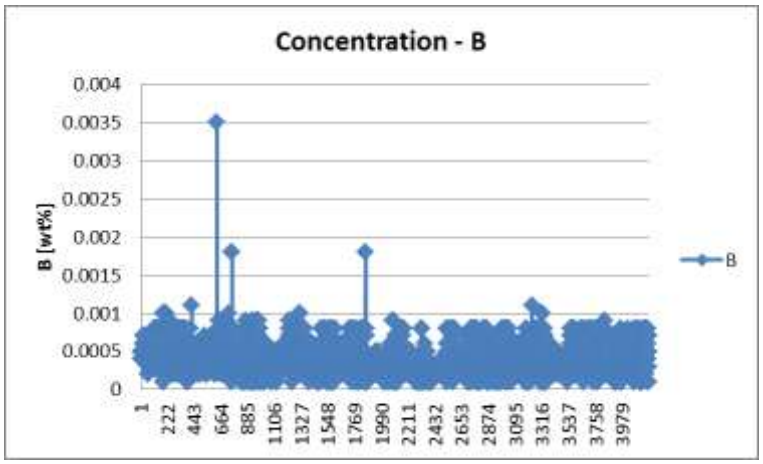

Figure 17: Wrong data: Boron concentration in training samples

3.4 Learning stage

There are many different ways that a neural network can learn; however, every learning algorithm involves the modification of the weight matrix, which holds the weights for the connections between the neurons.

3.4.1 Learning methods

Training is a very important process for a neural network. There are two forms of training that can be employed, supervised and unsupervised. Supervised training involves providing the neural network with training sets and the anticipated output. In unsupervised training, the neural network is also provided with training sets, but not with anticipated outputs.

3.4.1.1.1 Unsupervised Training

What does it mean to train a neural network without supervision? As previous mentioned, the neural network is provided with training sets, which are collections of defined input values. The unsupervised neural network is not provided with anticipated outputs.
Unsupervised training is typically used to train classification neural networks. A classification neural network receives input patterns, which are presented to the input neurons. These input patterns are then processed, causing a single neuron on the output layer to fire. This firing neuron provides the classification for the pattern and identifies to which group the pattern belongs.
Another common application for unsupervised training is data mining. In this case, you have a large amount of data to be searched, but you may not know exactly what you are looking for. You want the neural network to classify this data into several groups. You do not want to dictate to the neural network ahead of time which input pattern should be classified into which group. As the neural network trains, the input patterns fall into groups with other inputs having similar characteristics. This allows you to see which input patterns share similarities[12].
Figure 18 illustrates the flow of information through an unsupervised training algorithm.

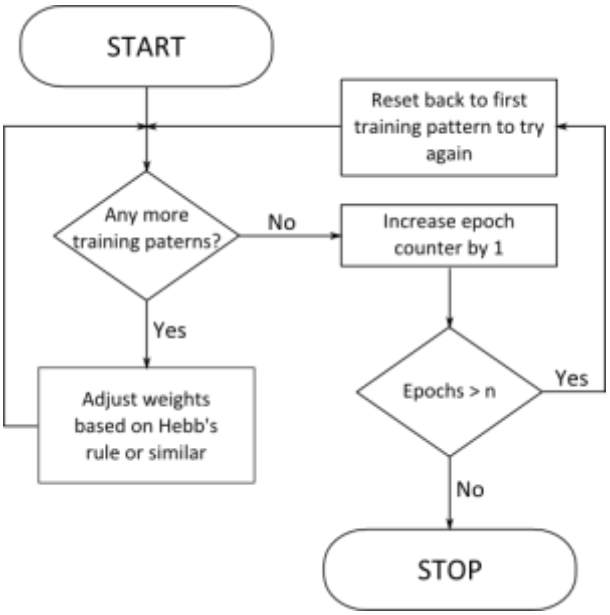

Figure 18: Unsupervized training[12].

3.4.1.1.2 Supervised Training

The supervised training method is similar to the unsupervised training method, in that training sets are provided. Just as with unsupervised training, these training sets specify input signals to the neural network. The primary difference between supervised and unsupervised training is that in supervised training the expected outputs are provided. This allows the neural network to adjust the values in the weight matrix based on the differences between the anticipated output and the actual output.

There are several popular supervised training algorithms. One of the most common is the back-propagation algorithm. It is also possible to use simulated annealing or genetic algorithms to implement supervised training[12].

Figure 19 illustrates the flow of information through a supervised training algorithm.

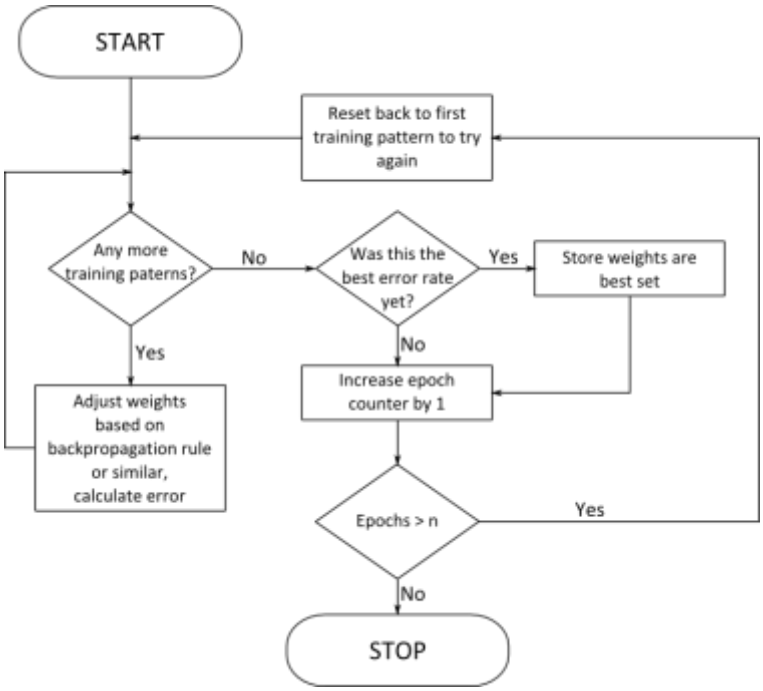

Figure 19: Supervised training[12].

3.4.1.1.3 Reinforcement learning

It is a hybrid learning method in that no desired outputs are given to the network, but the network is told if the computed output is going in the correct direction or not.

3.4.2 Error calculation

Error calculation is an important aspect of any neural network. Whether the neural network is supervised or unsupervised, an error rate must be calculated. The goal of virtually all training algorithms is to minimize the rate of error. In this section, we will examine how the rate of error is calculated for a supervised neural network. We will also discuss how the rate of error is determined for an unsupervised training algorithm. We will begin this section by examining two error calculation steps used for supervised training.

3.4.2.1 Error calculation and supervised training

There are two values that must be considered in determining the rate of error for supervised training. First, we must calculate the error for each element of the training set as it is processed. Second, we must calculate the average of the errors for all of the elements of the training set across each sample.

3.4.2.1.1 Output error

The output error is simply an error calculation that is performed to determine how different a neural network's output is from the ideal output. This value is rarely used for any purpose other than as a steppingstone in the calculation of the root mean square (RMS) error for the entire training set.

Once all of the elements of a training set have been run through the network, the RMS error can be calculated. This error acts as the global rate of error for the entire neural network.

3.4.2.1.2 Root mean square (RMS) error

The RMS method is used to calculate the rate of error for a training set based on predefined ideal results. The RMS method is effective in calculating the rate of error regardless of whether the actual results are above or below the ideal results. To calculate the RMS for a series of *n* values of *x*, consider Eg. (34)

$$x_{rms} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} = \sqrt{\frac{x_1^2 + x_2^2 + ... + x_n^2}{n}}. \tag{34}$$

The values of *x* are squared and their sum is divided by *n*. Squaring the values eliminates the issue associated with some values being above the ideal values and others below, since computing the square of a value always results in a positive number.
To apply RMS to the output of a neural network, consider Eg. (35)

$$x_{rms} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \left(actual_i - ideal_i\right)^2}. \tag{35}$$

To calculate the RMS for the arrays in the previous section, you would calculate the difference between the actual results and the ideal results, as shown in the above equation. The square of each of these would then be calculated and the results would be summed. The sum would then be divided by the number of elements, and the square root of the result of that computation would provide the rate of error.

3.4.2.2 Error calculation and unsupervised training

Most unsupervised neural networks are designed to classify input data. The input data is classified based on one of the output neurons. The degree to which each output neuron fires for the input data is studied in order to produce an error for unsupervised training. Ideally, we would like a single neuron to fire at a high level for each member of the training set. If this is not the case, we adjust the weights to the neuron with the greatest number of firings, that is, the winning neuron consolidates its win. This training method causes more and more neurons to fire for the different elements in the training set.

3.4.3 Transfer (Activation) functions

The transfer or activation function is a function that determines the output from a summation of the weighted inputs of a neuron. The transfer functions for neurons in the hidden layer are often nonlinear and they provide the nonlinearities for the network.
For the example in Figure 20, the output of neuron j, after the summation of its weighted inputs from neuron 1 to i has been mapped by the transfer function *f* can be shown as

$$O_j = f_j\left(\sum_i w_{ij}x_i\right). \tag{36}$$

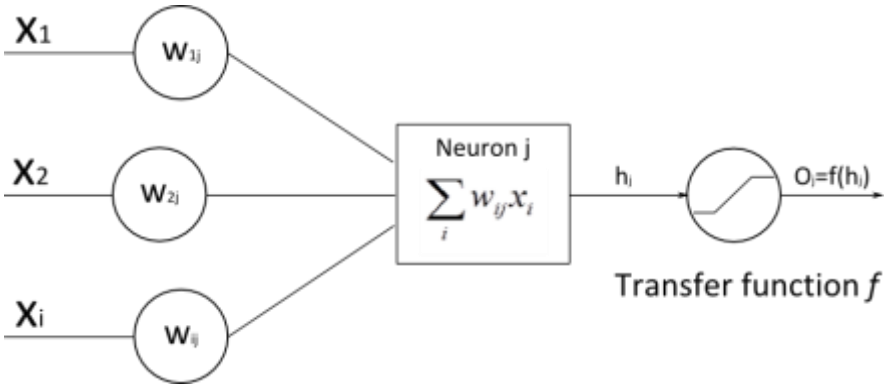

Figure 20: Diagram of neuron j

A transfer function maps any real numbers into a domain normally bounded by 0 to 1 or -1 to 1. Bounded activation functions are often called squashing functions.

Threshold: $f(x) = 0$ if $x < 0{,}1$ otherwise.

The most common transfer functions used in current ANN models are the sigmoid (S-shaped) functions. Masters in 1993 loosely defined a sigmoid function as a 'continuous, real-valued function whose domain is the real numbers, whose derivative is always positive, and whose range is bounded'. Examples of sigmoid functions are:
Logistic

$$f(x) = \frac{1}{1+e^{-x}}. \tag{37}$$

Hyperbolic tangent

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{38}$$

The logistic function remains the most commonly applied in ANN models due to the ease of computing its derivative

$$f'(x) = f(x)\big(1 - f(x)\big). \tag{39}$$

The output, O_j, of the neuron x_j of the earlier example in Figure 20 if the function f is a logistic function becomes

$$O_j = \frac{1}{1+e^{-\sum_i w_{ij}x_i - \theta_j}}. \tag{40}$$

where θ_j is the threshold on unit *j*.

If the function *f* is a threshold function instead, the output, O_j will be

$$O_j = \begin{Bmatrix} 1, \sum_i w_{ij}x_i > \theta_j \\ 0, else \end{Bmatrix}. \tag{41}$$

Sigmoid functions can never reach their theoretical limit values and it is futile to try and train an ANN to achieve these extreme values. Values that are close to the limits should be considered as having reaching those values. For example, in a logistic function where the limits are 0 to 1, a neuron should be considered to be fully activated at values around 0.9 and turned off at around 0.1. This is another reason why ANNs cannot do numerical computation as well or as accurate as simple serial computers; i.e. a calculator.

3.4.4 Training algorithms

Although there are many learning algorithms (rules) in common used, this section will only discuss the two most popular ones: the delta rule and its generalization, the back-propagation algorithm.

3.4.4.1 The Delta rule / Least mean squares (LMS) (Widrow-Hoff)

The Least Mean Square (LMS) algorithm was first proposed by Widrow and Hoff (hence, it is also called the Widrow-Hoff Rule) in 1960 when they introduced the ADALINE (Adaptive Linear), an ANN model that was similar to the perceptron model except that it only has a single output neuron and the output activation is a discrete bipolar function that produces a value of 1 or -1. The LMS algorithm was superior to Rosenblatt's perceptron learning algorithm in terms of speed but it also could not be used on networks with hidden layers.

$$\Delta w_{ij} = \eta \delta_j x_i, \tag{42}$$

where η is the learning rate ($0<\eta<1$) and δ_j is the error at neuron *j*.
The Delta rule error is

$$\delta_j = T_j - O_j. \tag{43}$$

The LMS error is

$$\delta_j = T_j - \sum W_{ij} X_j. \tag{44}$$

If we substitute output O^p with $X_p W_p$.

$$E = \frac{1}{2}\sum_p \| t^p - X_p W^p \|^2 , \tag{45}$$

where X_p is the input vector and W^p is the weight vector.

Then the gradient descent technique minimizes the error by adjusting the weights

$$\Delta W = -\eta \frac{\delta E}{\delta W}, \tag{46}$$

where η is the learning rate. The LMS can be now written as

$$\Delta W = -\eta \left(t_p - X_p W^p \right) X_p . \tag{47}$$

3.4.4.2 The Back-propagation (BP) / Generalized delta rule

The back-propagation (BP) algorithm is a generalization of the delta rule that works for networks with hidden layers. It is by far the most popular and most widely used learning algorithm by ANN researchers. Its popularity is due to its simplicity in design and implementation.

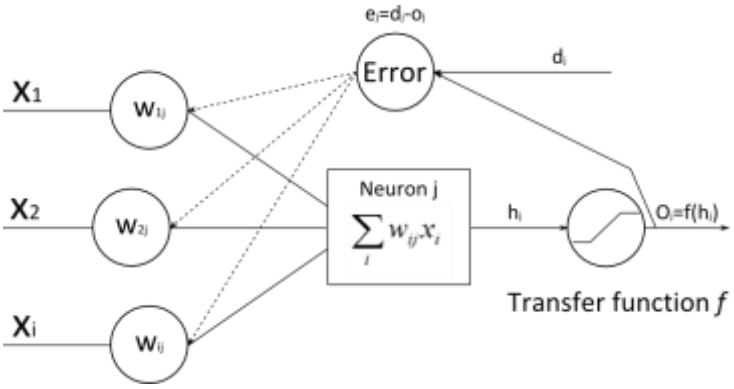

Figure 21: Back-propagation of errors for a single neuron j

The single neuron model of Figure 21 will be used to explain the BP algorithm. The BP algorithm is used mainly with MLP but a single neuron model is used here for clarity. The methodology remains the same for all models.

The BP algorithm involves a two-stage learning process using two passes: a forward pass and a backward pass. In the forward pass, the output O_j is computed from the set of input patterns, X_i

$$O_j = g\left(h_j\right) = f\left(h_j, \theta_j\right), \tag{48}$$

$$h_j = \sum_{i=1}^{i} w_{ij} x_i \,, \tag{49}$$

$$O_j = f\left(\sum_{i=1}^{i} w_{ij} x_i, \theta_j \right), \tag{50}$$

where f is a nonlinear transfer function, e.g. sigmoid function, θ_j is the threshold value for neuron j, x_i is the input from neuron i and w_{ij} is the weights associated with the connection from neuron i to neuron j.
After the output of the network has been computed, the learning algorithm is then applied from the output neurons back through the network, adjusting all the necessary weights on the connections in turn. The weight adjustment, Δw_{ij}, is as in the LMS Rule

$$\Delta w_{ij} = \eta \delta_j x_i \,, \tag{51}$$

where η is the learning rate ($0<\eta<1$) and δ_j is the error at neuron j

$$\delta_j = e_j \left(O_j\right)\left(1-O_j\right) = \left(\sum \delta_k w_k\right) O_j \left(1-O_j\right), \tag{52}$$

for hidden neurons where k is the neuron receiving output from the hidden neuron.
The adjustments are then added to the previous values

$$w_{ij} = w'_{ij} + \Delta W_{ij} \,, \tag{53}$$

where w'_{ij} is the previous weight term.

The gradient descent method is susceptible to falling of a chasm and becoming trapped in local minima. If the error surface is a bowl, imagine the gradient descent algorithm as a marble rolling from the top of the bowl trying to reach the bottom. If the marble rolls too fast, it will overshoot the bottom and swing to the opposite side of the bowl. The speed of the descent can be controlled with the learning rate term, η. On the other hand, if the learning rate is set to a very small value, the marble will descent very slowly and this translates to longer training time. An error surface of a typical problem is normally not a smooth bowl but may contain ravine and chasm where the marble could fall into. A momentum term is thus often added to the basic method to avoid the model's search direction from swinging back and forth wildly.
The weight adjustment term will then translate to

$$\Delta w_{ij} = \left(1-M\right)\eta \delta_j x_i + M\left(w'_{ij} - w''_{ij}\right), \tag{54}$$

where M is the momentum term and w''_{ij} is the weight before the previous weight w'_{ij}. The momentum term allows a weight change to persist for a number of adjustment cycles.
There are many other variations to the BP algorithm but by far, BP still proves to be the most popular and is implemented in almost all commercial ANN software packages.

3.5 Calculation of response

After training is done, some tests need to be done to verify the accuracy of ANN. Normally we verify this with verification points.

3.5.1 Training points and verification points

Verification and training points used for testing are usually a subset of historical data. Verification points are randomly chosen from datasets before training starts and are not used in training procedure. Verification points are not obligatory, but are recommended. We can use all data as training points in training. But if we don't have them, we can't verify the accuracy of the ANN. On the other hand training points are used in training and are the basis for training procedure. Better the training data are, better is the approximation. When error on training points becomes smaller than the user specified tolerance or when the number of training cycles reaches specified number, the training stops. In the Figure 22 we can see the distribution of training and verification points in complete data-set. Blue dots represent training points, while red dots represent verification points.

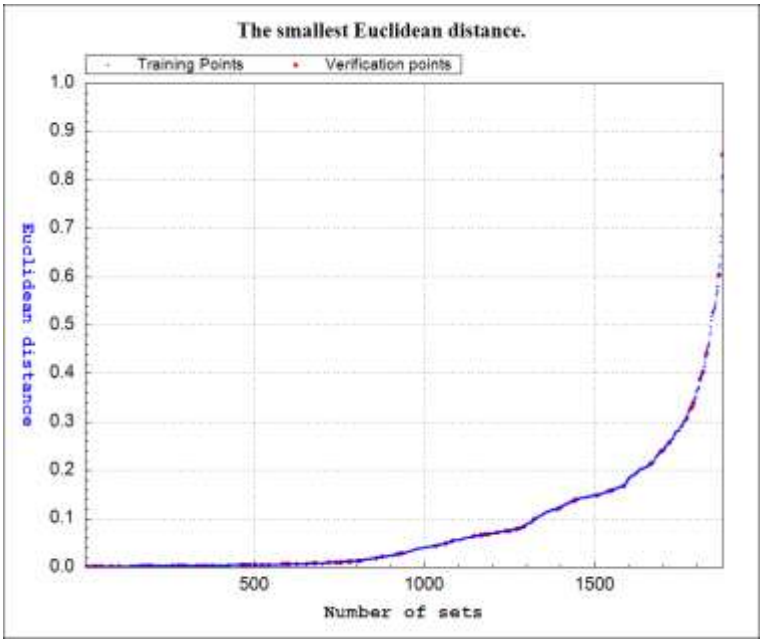

Figure 22: Minimum Euclidean distance to the closest point.

4 EXAMPLE BASED ON AN INDUSTRIAL PROBLEM

Continuous casting process is one of the processes in the steel production chain. Continuous casting enables efficient large scale production of steel. In this process, molten steel is poured vertically into a water cooled mould that is open at the bottom. Liquid steel solidifies at the contact with the mould, forming a solid shell. After partially solidified steel leaves the mould, it enters the secondary cooling system where it continues to solidify towards the core and is bent in horizontal position. This produces a steel billet that is cut and prepared for further processing[11].

For efficient production, the process must run smoothly and without defects produced in the output material, which is controlled by the main process parameters such as temperature of the molten steel, casting speed, cooling in the mould, and spray cooling at different stages. Mechanical properties of the produced material must meet requirements prescribed by customers. Obtained material properties are result of both process parameters and chemical composition that is achieved when the steel is melted, and in term chemical composition affects properties of the molt and solidified shell, as well as the solidifying process. A fair number of parameters must be properly adjusted in order to have efficient production of material with required properties.

A software framework has been developed for adjusting process parameters in material production in order to obtain the desired outcomes[9]-[10], [14]. The framework is first applied to achieve desired outcomes in the continuous casting process. Further, it is applied to approximate process parameters with ANN model.

4.1 Casting parameters generated with physical model

Table 1: Complete list of continuous casting (input) parameters.

ID	Name & units	Description	Range in the training set
1	Tcast [^{o}C]	Casting temperature	1515 - 1562
2	v [m/min]	Casting speed	1.03 - 1.86
3	DTmould [^{o}C]	Temperature difference of cooling water in the mould	5 - 10
4	Qmould [l/min]	Cooling flow rate in the mould	1050 - 1446
5	Qwreath [l/min]	Cooling flow rate in wreath spray system	10 - 39
6	Qsistem1 [l/min]	Cooling flow rate in 1st spray system	28 - 75

Table 2: Complete list of continuous casting (output) values.

ID	Name & units	Description & units	Range in the training set
1	ML [m]	Metallurgical length	8.6399 - 12.54
2	DS [m]	Shell thickness at the end of the mould	0.0058875 - 0.0210225
3	T [^{o}C]	Billet surface temperature at straightening start position	1064.5 - 1163.5

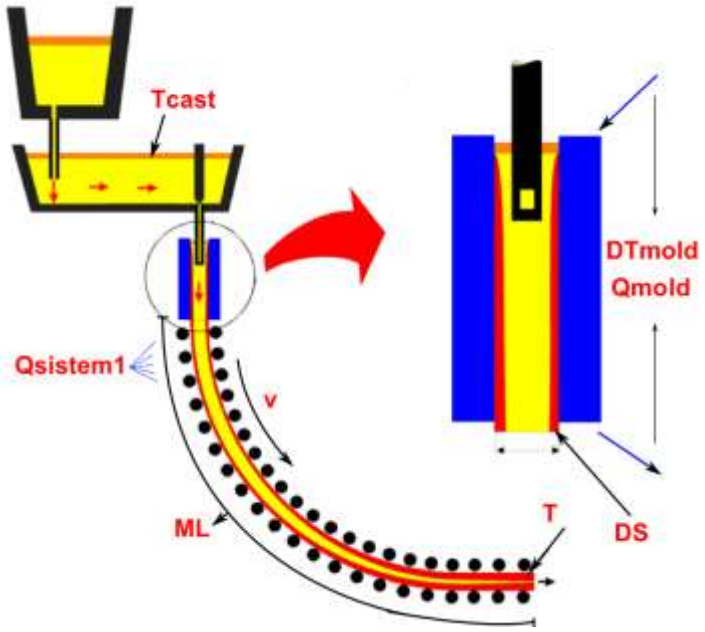

Figure 23: Measurement position on continuous casting machine.

4.2 Training the artificial neural network

We trained artificial neural network with data generated from continuous casting simulator developed in Laboratory for multiphase processes. Process is defined with 6 process parameters (Table 1).
24804 Training sets were automatically generated with physical simulator. The main goal of the study is to train the artificial neural network in order to be capable of predicting metallurgical length, shall thickness and billet surface temperature for one chemical composition, while changing the process parameters accounted for in training procedure. For practical set-up of the relevant artificial neural network we used Aforge.Net[15] and NeuroDotNet[16] libraries, implemented in our software module written in C-sharp. Datasets were stored in predefined JSON-based format and imported from file before training. Our module allows us to check training and verification errors during the training procedure. The procedure consist of five steps: reading data from a file, data preparation, training, testing and prediction of unknown output values based on different combinations of 6 input parameters listed in Table 1. During training, state of the artificial neural network is adjusted to data sets with known output values. These comprise historical cases of steel production in the past. During training, the ANN response in training and verification points is checked in order to see how well it does at predicting known and unknown output values. Verification and training points used for testing are usually a subset of historical data. Verification points are randomly chosen from datasets before training starts and are not used in training procedure, while training points are. When error on training points becomes smaller than the user specified tolerance or when the number of training cycles reaches specified number, the training stops. We tried different combinations of layouts and training parameters. We made more than 20 trainings with both, Aforge[15] and NeuronDotNet[16] libraries. The best results were achieved by using NeuronDotNet library, one hidden layer with 15 neurons. The learning rate was set to 0.3,

momentum to 0.6 and maximum number of epochs was set to 100000. Training procedures were always performed on HP workstation HPDL380G7 with 12 Intel Xenon 2.0GHz processors, 24GB installed RAM. Trained neural network which gave us the best results was trained in approximately 18 hours.

4.3 Results (parametric studies)

After the training of the ANN was done, the errors of the approximated outputs in verification sets were calculated, and some parametric studies were performed. The accuracy of the ANN and the dependences on parameters have been verified through the parametric tests.
The relative errors of the obtained approximation in all verification points were checked in the first study. These errors are a good indicator of accuracy of the obtained neural network-based approximation, and are defined as follows

$$\delta v_i = \left| \frac{v^{(m)}(\mathbf{p}_i) - v(\mathbf{p}_i)}{\max_{j \in I_T}\left(v^{(m)}(\mathbf{p}_j)\right) - \min_{j \in I_T}\left(v^{(m)}(\mathbf{p}_j)\right)} \right|; \quad i \in I_V, \tag{55}$$

where $v^{(m)}(\mathbf{p}_i)$ is the actual (measured) value of the output quantity v at parameters $\mathbf{p}_i$, $v(\mathbf{p}_i)$ is the approximated value of this quantity at the same vector of parameters, and denominator contains the range of the considered quantity over all training sets. Division by the range is performed for normalization and easier comparison of the results for different quantities that may typically differ by several orders of magnitude. $\mathbf{p}_i$ is the vector of input parameters of the verification set i, in which the actual values of the output quantities are known, since verification sets are taken out of the data provided by industrial measurements. Index i is an element of the index set I_V that enumerates the verification points. In our case, $I_V \in \{1, 2, ..., N_V\}$.
Verification points in our test represented five percent of the complete data-set provided. There were 24808 points with corresponding values of output quantities in the data-set, of which 1240 were randomly selected as verification sets and were excluded from training procedure. This preparation procedure is done automatically before training starts.
Actual and predicted values of metallurgical length are shown in Figure 24. Maximum relative error in verification points for metallurgical length is 11 percent. 98 percent of errors in verification points are under five percent.

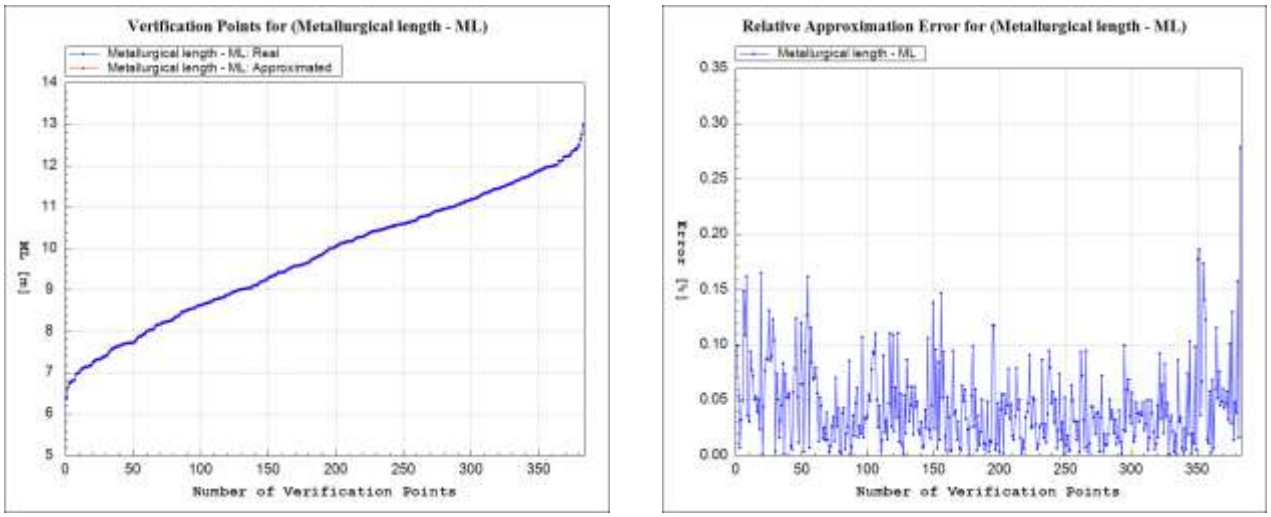

Figure 24: Approximation of metallurgical length (ML) in verification points. Left: comparison between actual and approximated values. Right: relative error in verification points. On both graphs, verification points are ordered according to the magnitude of metallurgical length.

Actual and predicted values of shell thickness at the end of the mould are shown in Figure 24. Maximum relative error in verification points for shell thickness is 11 percent. 98 percent of errors in verification points are under five percent.

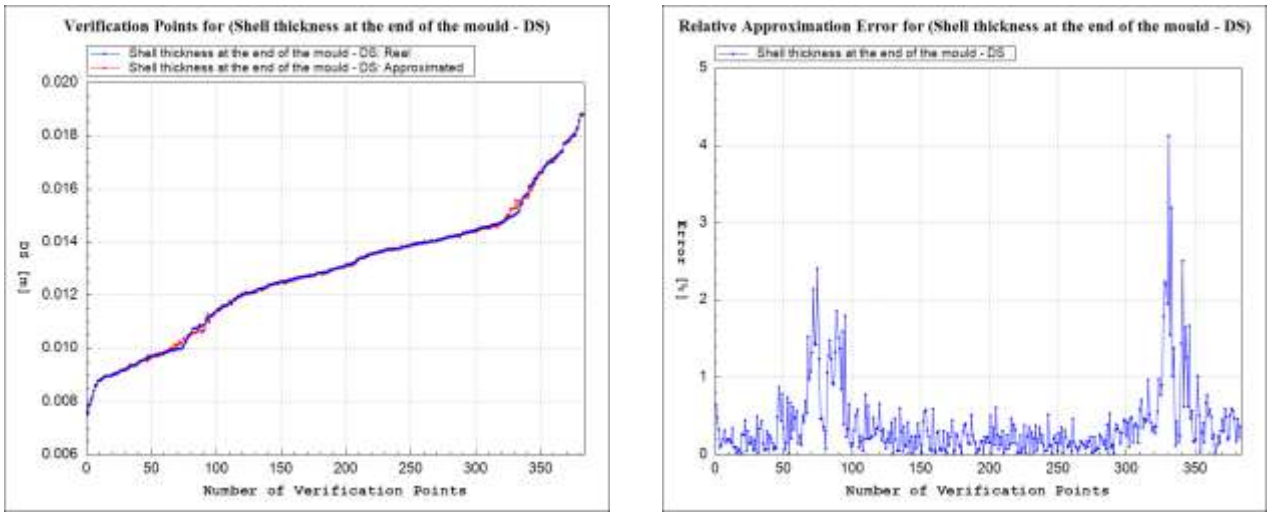

Figure 25: Approximation of shell thickness (DS) in verification points. Left: comparison between actual and approximated values. Right: relative error in verification points. On both graphs, verification points are ordered according to the magnitude of shell thickness.

Actual and predicted values of billet surface temperature at straightening start position are shown in Figure 24. Maximum relative error in verification points for billet surface temperature is 11 percent. 98 percent of errors in verification points are under five percent.

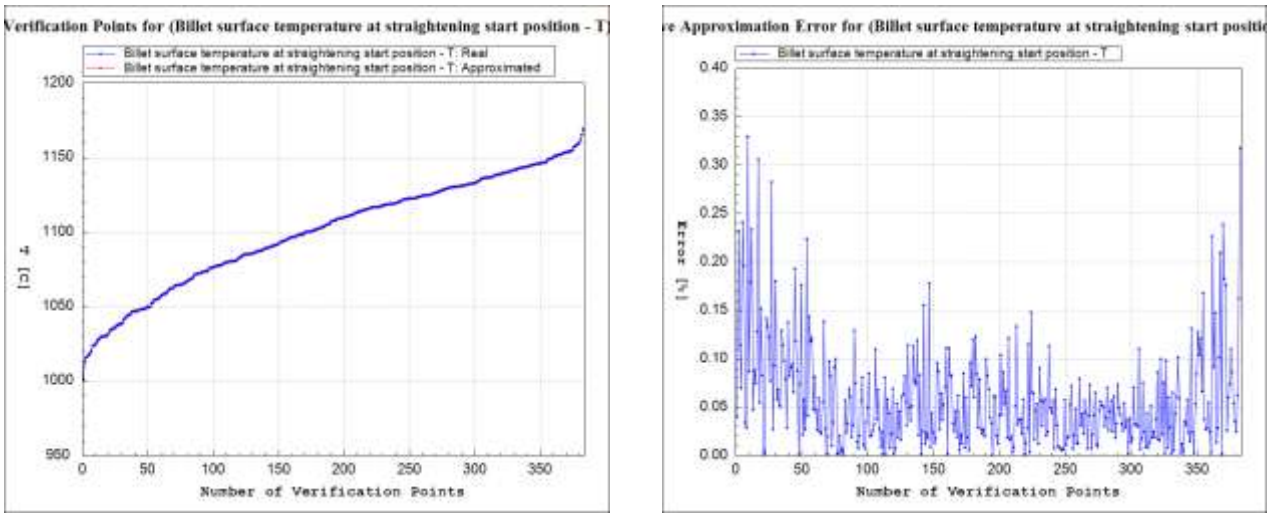

Figure 26: Approximation of billet surface temperature (T) in verification points. Left: comparison between actual and approximated values. Right: relative error in verification points. On both graphs, verification points are ordered according to the magnitude of shell thickness.

4.3.1 Parametric test 1 (verification and training points)

In the next study we randomly take 4 data sets from the entire data. 2 sets were chosen among verification points and other 2 from training points. In each chosen set we varied one parameter, for example, casting speed (v), while other parameters were fixed. Parameter was varied within the range defined by the minimum and maximum value of that parameter over all dataset used in training. These kind of tests help us find out how the change of one parameter, for example, casting speed, influences process output quantities of interest such as metallurgical length, shell thickness and billet surface temperature. We performed these tests for all 6 input parameters. The influence of variation in casting speed on billet surface temperature is shown in Figure 27.

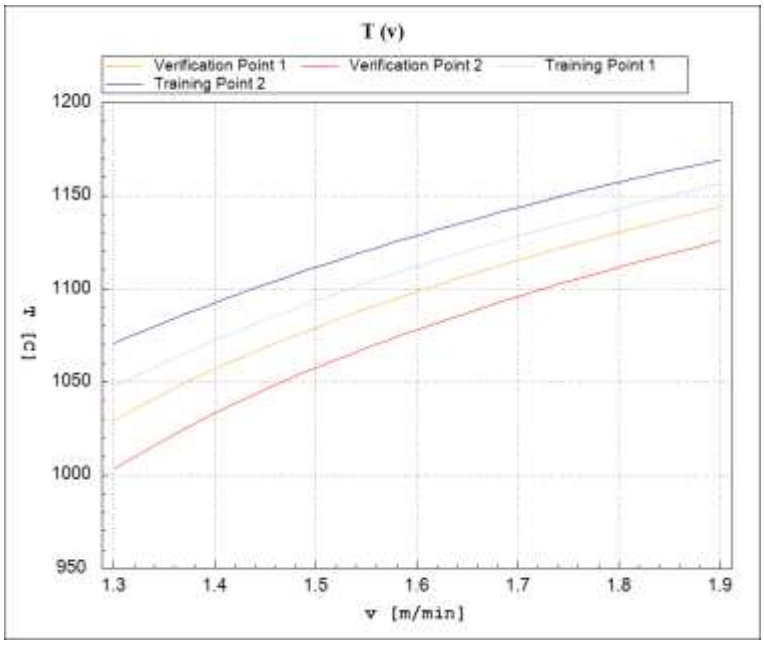

Figure 27: Billet surface temperature dependent on casting speed, calculated by the ANN model on two verifications and two training sets.

From the graph we can see that if we casting speed (v), billet surface temperature also increases. This trend is well known in metallurgy.

4.3.2 Parametric test 2 (points on line between two points)

In this parametric test we perform a series of tests where carbon mass fraction is varied and each parametric test is based on a vector of input parameters picked out of a spatially related set. This set is constructed in such a way that the contained parameter vectors lie on the line connecting the two chosen endpoints $\mathbf{p}_I$ and $\mathbf{p}_F$ from the space of input parameters. The effects of variation of the carbon mass fraction are plotted around the constructed points (including the chosen endpoints). The intermediate input parameter vectors $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$ are chosen equidistantly such that

$$\mathbf{p}_j = \mathbf{p}_I + (\mathbf{p}_F - \mathbf{p}_I)\frac{j}{n+1};\ j = 0,1,2,...,n+1\ , \tag{56}$$

where n is the number of the intermediate input vectors. The endpoints $\mathbf{p}_I$ and $\mathbf{p}_F$ are in our case taken from the data used for training the ANN. The arrangement of the input vectors is schematically shown in Figure 28.

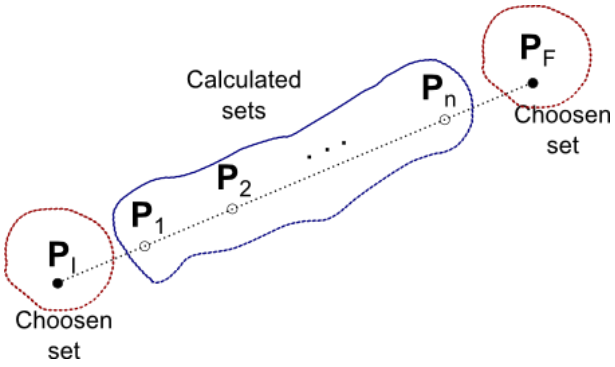

Figure 28: Points for parametric studies chosen on the line between two points chosen from training data.

In each of the points from Figure 28 we varied one parameter (casting speed in this case) over the whole range of values, while other parameters remained unchanged. Influence on billet surface temperature is shown in Figure 29.

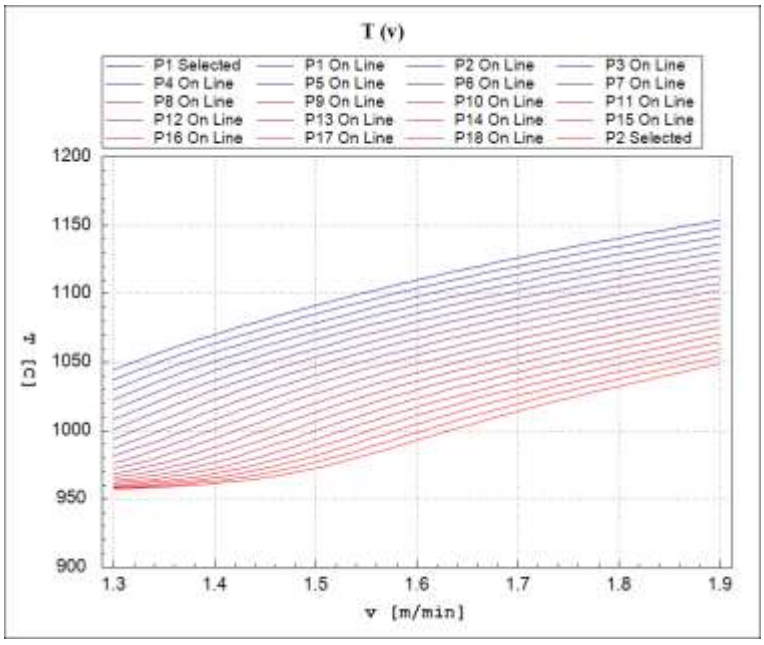

Figure 29: Billet surface temperature dependent on casting speed, calculated by the ANN model in 2 points from the training data and 18 other points on the line between them.

In this test we can find out how smoothly the curves on the graph pass from the parameter vector $\mathbf{p}_I$ to $\mathbf{p}_F$. Because the parameter vectors $\mathbf{p}_j$ that lie between $\mathbf{p}_I$ and $\mathbf{p}_F$ were not included in the training, one could expect lower accuracy of the approximated response at these parameters.

4.3.3 Parametric test 3 (weighted Euclidean distances)

In the next study we examine how uniformly the parametric space is covered by the training data. We first calculate for each vector of input parameters from the training set the smallest weighted Euclidean distance from this vector to any other input parameters vector from the training set. We define the weighted Euclidean distance d_l as

$$d_l(\mathbf{x},\mathbf{y}) = \sum_{k=1}^{N_p} \sqrt{w_k^2 (x_k - y_k)^2}\ ,$$
$$w_k = \frac{1}{l_k}\ , \tag{57}$$
$$l_k = \max_i (p_{ik}) - \min_i (p_{ik});\ i \in I_M\ ,$$

where l_i represents the range of the corresponding input parameter over the complete measured data obtained from the industrial line, and **x** and **y** are arbitrary vectors in the space of input parameters and N_p is the dimension of the space. We denote by I_M the index set used to enumerate all data sets obtained form industrial measurements, thus p_{ik} represents the *k*-th element of the vector of input parameters $\mathbf{p}_i$.

Figure 30 and Figure 31 show the distribution of the training and the verification points according to their distances to the closest training point, and to the *m*-th closest training point respectively, for *m*=7. Since 7 is one more than the number of process parameters, this is the minimal number of data-sets necessary for linear regression. Points on the graphs correspond to the provided data sets that are ordered by the distance to the *m*-th closest point. We can see from the graph that the distance to the closest points varies a lot, and a large portion of points don't have close neighbors. This means that distribution of data points in parameter space is not very uniform. This could be expected with respect to the fact that data was obtained from the actual industrial line. In steel production, a number of standardized steel qualities are used with narrowly defined chemical compositions, for which process is adjusted according to expert knowledge generated by past experience. Clusters of data points are therefore formed around vicinity of parameter settings that are commonly in use.

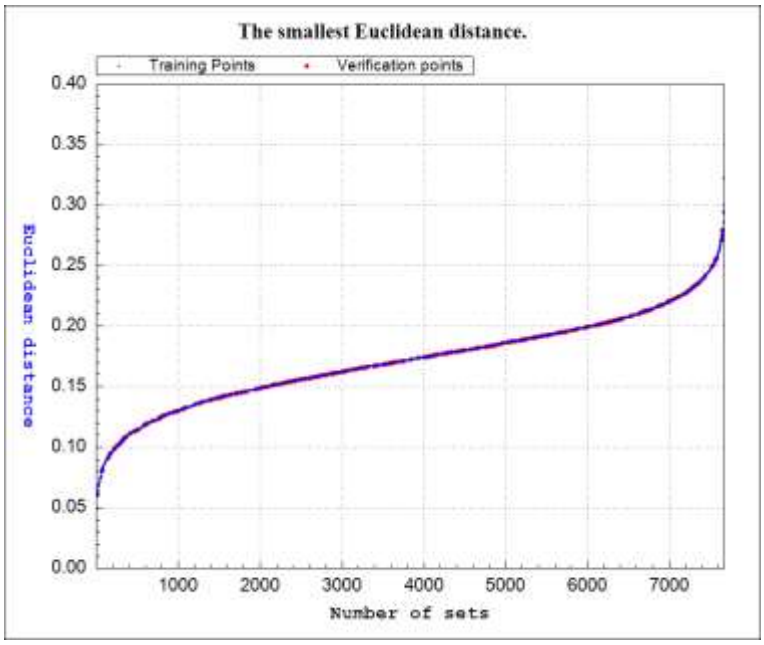

Figure 30: Minimum weighted Euclidean distance to the closest point.

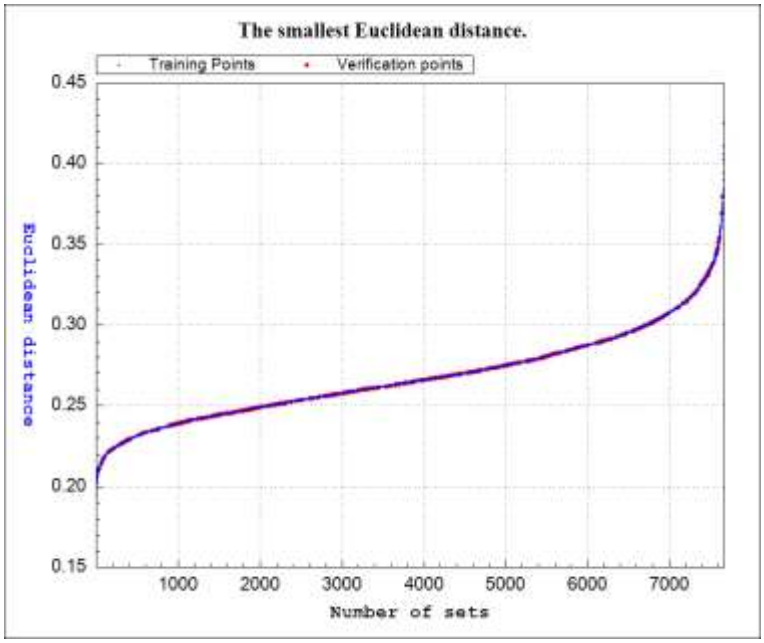

Figure 31: Minimum weighted Euclidean distance to the 7^{th} closest point.

Acknowledgment

The presented work was partially supported by The Centre of Excellence for Biosensors, Instrumentation and Process Control (COBIK), which is an operation financed by the European Union, European Regional Development Fund and Republic of Slovenia, Ministry of Higher Education, Science and Technology. The financial support is gratefully acknowledged.

References:

[1] **Irwing, W. R.** (1993). *Continuous Casting of Steel*. The Institute of Materials, London.

[2] **Continuous Casting of Steel** (2012): Available at: http://en.wikipedia.org/wiki/Continuous_casting.

[3] **Continuous casting** (2012): Available at: http://ccc.illinois.edu/introduction/concast.html

[4] **Vertnik, R.** (2010): Heat and fluid flow simulation of the continuous casting of steel by a meshless method. Dissertation. University of Nova Gorica.

[5] **Lorbiecka, A.; Vertnik, R.; Gjerkeš, H.; Manojlovič, G.; Cesar, J.; Šarler, B.** (2009): Numerical modeling of gain structure in continuous casting of steel. Computers, materials & continua, vol. 8, pp. 196-208

[6] **Fausett, L.** (1994): *Fundamentals of Neural Networks*. Florida Institute of Technology: Prentice-Hall,

[7] **Anderson, J. A.; and Rosenfeld, E.** (1987). *Neurocomputing: Foundations of Research*, MIT Press, Cambridge, MA

[8] **Smith, B. R.** (1997): Neural Network Enhancement of Closed-Loop Controllers for I11 Modeled Systems with Unknown Nonlinearities. Faculty of the Virginia Polytechnic: Institute and State University.

[9] **Grešovnik, I.; Kodelja, T.; Vertnik, R.; Šarler, B.** (2012): A software framework for optimization parameters in material production. *Applied Mechanics and Materials*, vol. 101, pp. 838-841.

[10] **Grešovnik, I.; Kodelja, T.; Vertnik, R.; Šarler, B.** (2012): Application of artificial neural networks to improve steel production process. Bruzzone, A. G.; Hamza, M. H. *Proceedings of the 15th International Conference on Artificial Intelligence and Soft Computing*, Napoli, Italy, pp. 249-255.

[11] **ANN Types** (2012): Available at: http://en.wikipedia.org/wiki/Types_of_artificial_neural_networks

[12] **Heaton, J.** (2008): *Introduction to Neural Networks for C#*. Heaton Research inc: St Luis, vol. 2.

[13] **Salakhutdinov, R.; Hinton, G. E.** (2009). Deep Boltzmann machines. The Twelfth International Conference on Artificial Intelligence and Statistics. *AISTATS'09*, vol. 5, pp. 448-455.

[14] **Grešovnik, I.** (2012): Iglib.net - investigative generic library. Available at: http://www2.arnes.si/ ljc3m2/igor/iglib/.

[15] **Aforge.net** (2011): Framework. Available at: http://www.aforgenet.com/.

[16] **NeuronDotNet** (2011): Neural networks in C#. Available at: http://neurondotnet.freehostia.com/.