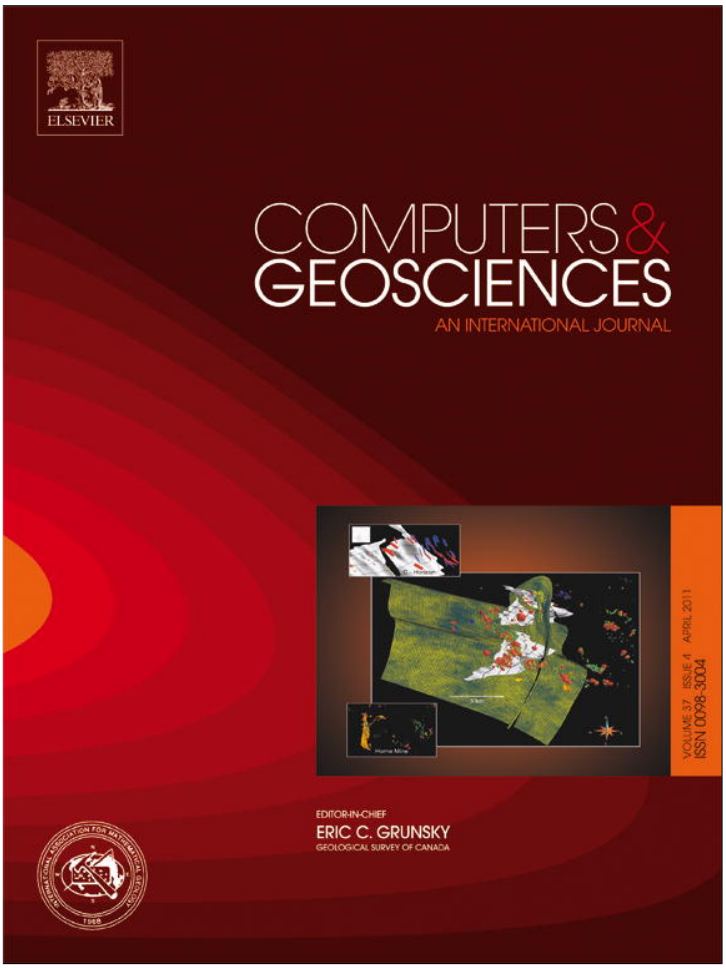
Provided for non-commercial research and education use.
Not for reproduction, distribution or commercial use.



This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier's archiving and manuscript policies are encouraged to visit:

http://www.elsevier.com/copyright

Author's personal copy

Computers & Geosciences 37 (2011) 634–644

Contents lists available at ScienceDirect

Computers & Geosciences

journal homepage: www.elsevier.com/locate/cageo

LavaNet—Neural network development environment in a general mine planning package

Ioannis Konstantinou Kapageridis *, A.G. Triantafyllou

Laboratory of Mining Information Technology and GIS Applications, Department of Geotechnology and Environmental Engineering, School of Technological Applications, Technological Educational Institute of Western Macedonia, Koila, Kozani GR-50100, Greece

ARTICLE INFO

Article history:
Received 10 December 2009
Received in revised form
3 October 2010
Accepted 29 October 2010
Available online 10 November 2010

Keywords:
Neural networks
VULCAN
SNNS
Perl
Mine planning
Estimation

ABSTRACT

LavaNet is a series of scripts written in Perl that gives access to a neural network simulation environment inside a general mine planning package. A well known and a very popular neural network development environment, the Stuttgart Neural Network Simulator, is used as the base for the development of neural networks. LavaNet runs inside VULCAN™—a complete mine planning package with advanced database, modelling and visualisation capabilities. LavaNet is taking advantage of VULCAN's Perl based scripting environment, Lava, to bring all the benefits of neural network development and application to geologists, mining engineers and other users of the specific mine planning package. LavaNet enables easy development of neural network training data sets using information from any of the data and model structures available, such as block models and drillhole databases. Neural networks can be trained inside VULCAN™ and the results be used to generate new models that can be visualised in 3D. Direct comparison of developed neural network models with conventional and geostatistical techniques is now possible within the same mine planning software package. LavaNet supports Radial Basis Function networks, Multi-Layer Perceptrons and Self-Organised Maps.

© 2010 Elsevier Ltd. All rights reserved.

1. Introduction

A number of mining planning related problems have been approached using artificial neural network (ANN) technology in the last couple of decades. These problems commonly relate to pattern classification, prediction and optimisation. ANNs have been successfully applied to these particular areas and appear suitable for similar mining and environmental problems (Kapageridis, 2002). The general trend in the mining industry for automation to the greatest degree calls for technologies such as ANNs that can utilise large amounts of data for the development of models which, otherwise, are very difficult or sometimes even impossible to construct.

One particular area where ANNs have been applied in the mine planning sector is in spatial analysis problems. Exploration and resource estimation commonly involves the prediction of various parameters characterizing a mineral deposit or a reservoir. The input data usually come in the form of samples with known positions in 3D space. The majority of the ANN systems developed for these predictive tasks are based on the relationship between the modelled parameter and sample location. The most common practice when developing training patterns sets for an ANN, is to generate input–output pairs with the input being the sample location and the desired output being the value of the modelled parameter at that location. In other words, most of the ANN systems treat the modelling of the unknown parameters as a problem of function approximation in the sample coordinates space.

Some other systems go a step further to exploit information hidden in the relationship between neighbouring samples (Kapageridis, 1999, 2003, 2005). The estimation of a parameter at a specific location in 3D space, in this case, depends on information from samples around that location. Regardless of the approach, a common obstacle in developing neural network solutions to mining problems is the development of appropriate data sets that can be used for neural network training as well as the transfer of neural network application results into the mine planning process. The neural network development environment described in this paper addresses these problems and provides a platform for fast and comprehensive development of neural network models within a general mine planning package.

1.1. VULCAN 3D software and Lava scripting

A number of general mine planning packages exist today with varying capabilities and functionality. These packages host a large number of algorithms for analysis and modelling of geological and other spatial data. They commonly use special database structures for storing data. Models produced by these packages are also stored in special file structures that cannot be opened directly by other more generic packages for further analysis and usage. VULCAN™ is

* Corresponding author. Tel.: +30 24610 40161x211; mobile: +30 6976 405545; fax: +30 24610 40876.
E-mail addresses: ioannis.kapageridis@gmail.com, ikapa@airlab.teikoz.gr (I.K. Kapageridis).

0098-3004/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cageo.2010.10.017

Author's personal copy

one of these general mine planning packages that has been around since the early 1980s produced by Maptek Pty Ltd. In its current form it has several modules covering most aspects of mine planning for any type of mining.

Envisage is the heart of the VULCAN™ software package. It provides a full 3D environment for the design and digitising of vector data and for modelling using a number of different model structures and algorithms. Extensive vector data editing and manipulating options are part of the core Envisage system. Envisage allows importing and exporting vector data from formatted data files or connected measurement equipment. Envisage is used both at mine sites and in feasibility work where geologists, engineers, surveyors and other users all work in a common environment.

It is very common in mine planning packages to integrate a scripting language that allows automation of repetitive tasks and the extension of built-in functionality of the software. Lava is a module for a version of the popular scripting language Perl (Schwartz et al., 2008) that is built into VULCAN™. This version of Perl behaves exactly like the standard Perl version (5.6.1), except that certain functions, which can directly access Envisage internals, have been added. The syntax required to access these functions is provided through the Lava package. Lava uses many object/class structures to modularise and confine the details of the implementation of various Envisage tasks, as well as a few general functions that may be accessed without a class. There are three types of classes, Lava classes (which can only be used inside Envisage), Vulcan classes (which do not require the Envisage 3D environment to be used) and the Triangulation class, which like the Vulcan classes, can also be used outside of Envisage (Table 1). Generally, class names are prefixed with 'Lava::' or 'vulcan::' and have the first letter in each word capitalised, similarly for the general Lava functions. The class data, and functions thereon, are accessed through the member functions, which have lowercase names, and are typically called by dereferencing an instance of a class. LavaNet uses a number of these classes to gain direct access to Vulcan file and model structures. LavaNet windows and dialog boxes are also based on Lava classes.

1.2. Stuttgart Neural Network Simulator

The Stuttgart Neural Network Simulator (SNNS) is a well developed and complete environment that has been around since early 1990s. It is a multiplatform package that allows development of artificial neural network systems using a wide variety of topology architectures and training algorithms (SNNS User's Manual, 1996). It provides an X-Windows graphical user interface that is identical on all platforms. A Java based version of SNNS called JavaNNS is also available. The original X-Windows version comes complete with a batch development language called Batchman. The LavaNet interface presented in this paper utilises this tool for the development and application of neural network systems from within VULCAN™. The current (4.3) and previous versions of SNNS can be downloaded for free and be used on different operating systems including Linux and Microsoft Windows™.

2. General description of the interface

As already mentioned, LavaNet consists of a number of scripts that, once copied in VULCAN™ installation directory, can be accessed from Envisage as a menu option or through the provided toolbar. The scripts run and behave as any other Envisage option and the user has no feeling of the script interpretation process that takes place in the background. Interpretation time is normally less than a fraction of a second in modern hardware. The toolbar groups LavaNet options in separate menus according to their purpose as shown in Fig. 1. The user can determine the project name and network topology architecture through the setup options. Training, validation and application patterns can be generated through the Pattern menus using any of the data and model structures used by VULCAN™. Development of the network follows, which can be performed in multiple training stages. For example, in the case of an RBF network, there can be separate training stages for fixing the RBF centre locations, RBF receptive fields and weights to the output layer(s). These stages can be combined in a single batch development script that can be used more than once. Finally, the results from running a developed network can be imported through the Results menu.

Fig. 2 presents the way Envisage, LavaNet and SNNS work together. LavaNet reads the information stored in databases and models in Envisage and generates SNNS formatted pattern, network and Batchman development script files. Networks are developed by Batchman using the LavaNet prepared scripts and then applied to produce results using application patterns produced by LavaNet. Finally, LavaNet reads the results and writes them into a VULCAN™ database or model.

As LavaNet is further developed, more options will become available and will be added to the provided toolbar. These options will take

Table 1
Types of classes included in the Lava package for Perl.

Type	Class	Functionality
Lava	Lava::Panel	Used to create Envisage panels for acquiring specific information from the user
	Lava::Obj	Used to represent any design database object
	Lava::Links	Encapsulates an array of link records that may be associated with an Envisage object
	Lava::Coord	Encapsulates an array of coordinate records
	Lava::Text	Encapsulates an array of text data, as well as various text attributes
	Lava::Selection	Manages the prompting, selection, highlighting, loading and replacing of Envisage objects
	Lava::Layer	Used to create or select a layer, into which new objects may be added
	Lava::Point	Used to input 3D data from the user, using the standard Envisage digitising methods
	Lava::TriSelection	Used to select triangulations from the Envisage display
	Lava::Contourer	Used to contour triangulations
	Lava::gfx::window	Used to access a graphics window
	Lava::Show	Used to display messages while an operation is performed
	Lava::Message	Used to display a message panel with an OK button to close it
Vulcan	Vulcan::block_model	Used to access a new or existing block model file
	Vulcan::isisdb	Used to access a new or existing ISIS database
	Vulcan::grid	Used to access a new or existing grid model file
	Vulcan::mapio	Used to access a new or existing formatted MAP file
	Vulcan::triangulation	Used to access a new or existing triangulation model file
	Triangulation	Provides an interface to Envisage triangulation structures

Author's personal copy

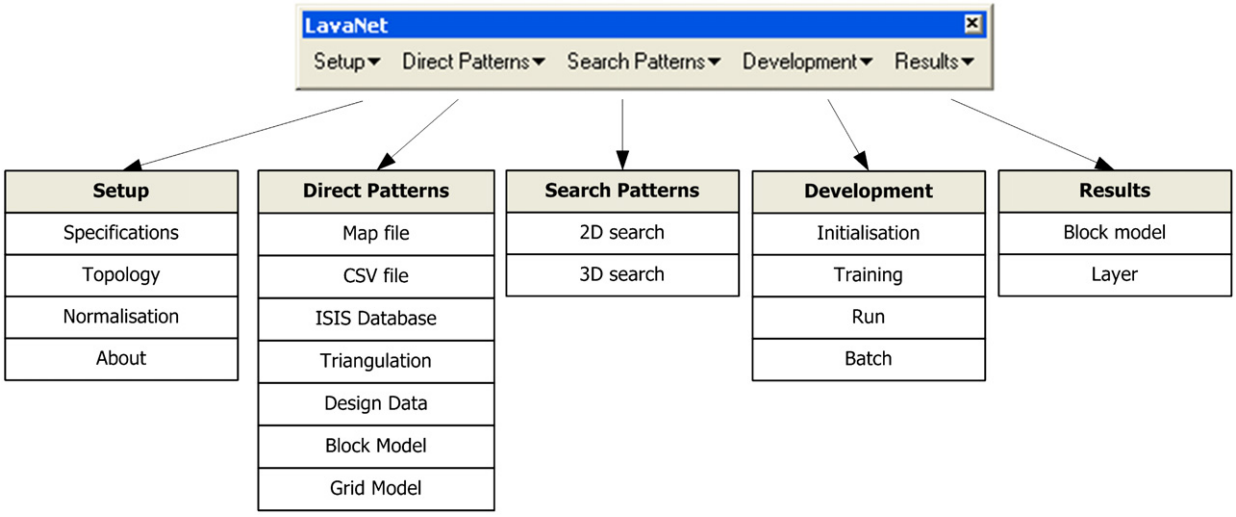


Fig. 1. LavaNet menu structure and toolbar within Envisage.

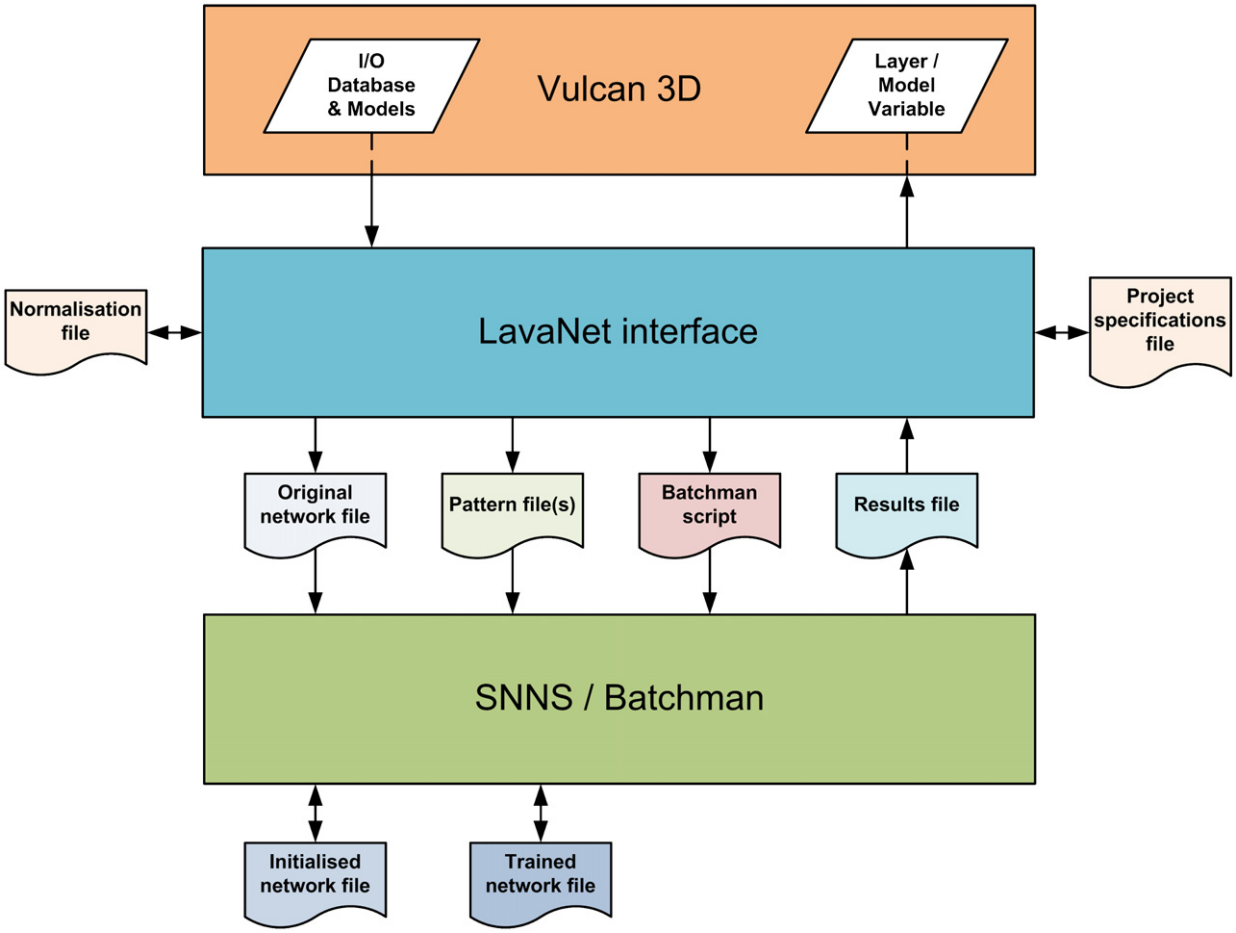


Fig. 2. Information flow between Vulcan, LavaNet and SNNS during neural network development and application.

advantage of future developments in Lava scripting and Perl as well as further developments to VULCAN™ data and model structures.

3. Network topology specification

LavaNet is based on SNNS and, therefore, uses the particular network structures and algorithms provided by this simulator. Currently three network models are implemented in LavaNet—Multi-Layer Perceptrons (MLP), Radial Basis Function Networks (RBFN) and Self-Organising Maps (SOM). Table 2 presents the various network models and functions supported by LavaNet. Networks can be designed with particular input, hidden and output nodes and specific initialisation, learning and updating functions as in SNNS. Network specification differs depending on the network model used as shown in Fig. 3. These panels are meant to be

Author's personal copy

Table 2
Network initialisation, learning and updating functions supported by the LavaNet interface.

	MLP	RBF	SOM
Initialisation functions	Randomize_Weights, Random_Weights_Perc, CPN_Weights_v3.2, CPN_Weights_V3.3, CPN_Rand_Pat	Randomize_Weights, RBF_Weights, RBF_Weights_Kohonen, RBF_Weights_Redo	Kohonen_Rand_Pat, Kohonen_Weights_v3.2, Kohonen_Const
Learning functions	BackPercolation, BackpropBatch, BackpropChunk, BackpropMomentum, BackpropWeightDecay, Counterpropagation, Quickprop, Rprop, Std_Backpropagation	RadialBasisLearning, RBF-DDA	Kohonen
Updating functions	Topological_Order, Auto_Synchronous, Random_Order, Random_Permutation, Serial_Order, Synchronous_Order	Topological_Order, Auto_Synchronous, Random_Order, Random_Permutation, Serial_Order, Synchronous_Order	Kohonen_Order

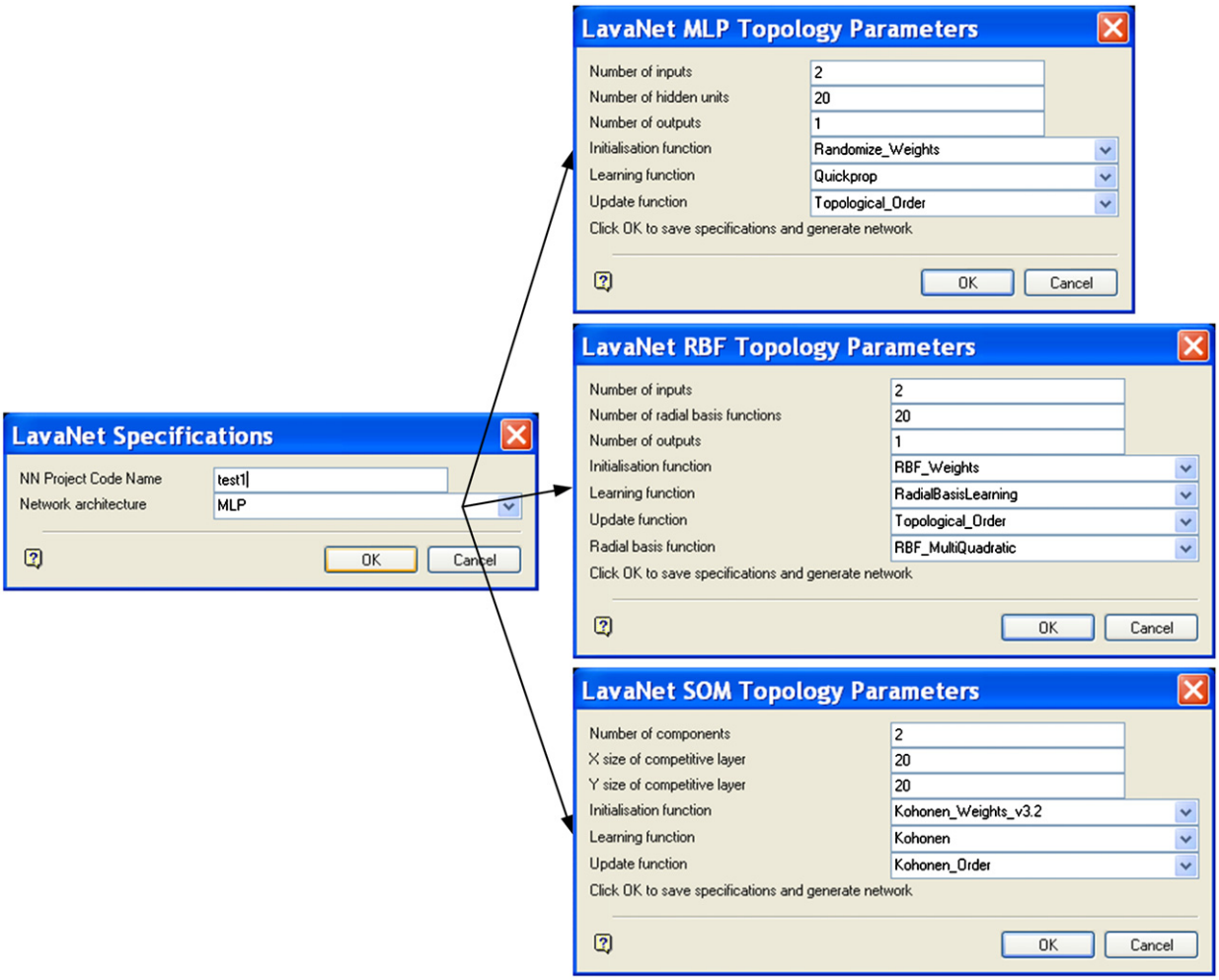


Fig. 3. Panels for choosing network architecture and functions.

complimentary to the original SNNS environment—LavaNet can receive any SNNS compatible network file and use it for development and application. Network and pattern files developed through LavaNet can also be examined, analysed and further developed in the SNNS or JavaNNS environment. A brief introduction to the network models supported by LavaNet is given in the following sections.

3.1. *Multi-Layer Perceptrons*

Beyond any doubt the most popular and widely used neural network structure, the Multi-Layer Perceptron (also known as feed-forward network) is a hierarchical design consisting of fully interconnected layers of processing elements (Haykin, 1999). Generally the operation of this network is mapping an n-dimensional input to an m-dimensional output, in other words modelling of a function F: $\Re^n \rightarrow \Re^m$. This is achieved by means of training on examples (x_1,y_1), $(x_2,y_2),\ldots,(x_k,y_k)$ of the mapping, where $y_k=f(x_k)$. MLP networks are commonly used together with an error correction algorithm such as gradient descent, or conjugate gradient descent. The structure of the MLP network comprises a number of layers of processing elements. There are three types of layers depending on their location and function: input, hidden and output. The connections between the layers are generally feed-forward during presentation of an input signal. However, during

Author's personal copy

training the network allows the back-propagation of error signals to the hidden units in order to adjust the connection weights. Feed-forward networks may have more than one hidden layers. Extra hidden layers allow more complex mappings but also require more information for training of the network. The choice is usually between an excessive number of processing elements in one hidden layer and a low number of processing elements but in more than one hidden layer.

An example of a MLP based system for ore grade/reserve estimation was developed by Wu and Zhou (1993). The Multi-Layer Perceptron in this example had four layers: an input layer, two hidden layers, and one output layer. The network received two inputs, the Easting and Northing of samples. The two hidden layers were identical and had 28 units each. It was a relatively large network considering the dimension of the input space (2D). The developers have used a fast learning algorithm called the Dynamic Quick-Propagation (Wu, 1991) that is based on the quick-propagation algorithm (Fahlman, 1988) and a system for the determination of the hidden layer size called Dynamic Node Creation (Ash, 1989). Similar to the above network, is the ANN developed by Yama and Lineberry (1999), which was based again on the MLP architecture but used the original back-propagation learning algorithm. Clarici et al. (1993) described a similar approach of a single hidden layer network. Cortez et al. (1998) presented a hybrid system combining ANN technology with geostatistics for grade/resources estimation. Their system, called NNRK ('Neural Network estimation of the drift and Residuals' Kriging'), is based on a network with 3 inputs (the sample's X, Y, Z coordinates), 6 hidden units and one output, the respective Zn assay. Garcia and Whitman (1992) have used a MLP trained using back-propagation for the inversion of lateral electrode well logs. The data used for training the network were derived from a finite difference method that simulated the lateral log. The trained network was tested using real data and the results were compared with those from an automated inversion model. In a similar fashion, Rogers et al. used a MLP network for the prediction of lithology from well logs (Rogers et al., 1992).

3.2. Radial Basis Function networks

Radial Basis Function Networks (RBFNs) are powerful network structures which construct global approximations to functions using combinations of Radial Basis Functions (RBFs) centred on weight vectors (Lowe, 1995; Moody and Darken, 1989; Park and Sandberg, 1993). The basic RBFN structure consists of non-linear basis functions centred on each hidden node weight vector. Hidden nodes have an adaptable range of influence or receptive field. The output of the hidden nodes is a radial function of the distance between each pattern vector and each hidden node weight vector.

Learning in RBF networks leads to the construction of a hyper-surface in multi-dimensional space that fits the training data in the best possible way. Function approximation and pattern classification are the main areas of application RBF networks. One of their main advantages lies in their strong scientific foundation. RBFs have been motivated by statistical pattern processing theory, regression and regularisation, biological pattern formation, and mapping in the presence of noisy data (Powell, 1987, 1992).

For an RBF network to be able to receive training samples and function as a hyper-surface reconstruction network, a number of its parameters need to be calculated. These parameters include the following:

- The number of RBFs.
- The centres of the hidden layer RBFs.
- The receptive fields or widths of RBFs.
- The weights between hidden and output layer.

Each parameter should be fixed separately through an appropriate training stage. LavaNet can be used to formulate each training stage separately using the RBF learning algorithms in Table 2.

RBF networks have been successfully used in mining and geospatial problems in the past. Caiti and Parisini (1991) have used RBF networks to interpolate geophysical properties of ocean sediments, e.g. porosity, density and grain size. Kapageridis (1999) has used RBF networks in a modular system for grade estimation. Samanta and Bandopadhyay (2009) have used RBF networks and an evolutionary algorithm for grade estimation in a placer gold deposit.

3.3. Self-organising maps

Self-organising maps (SOMs) are a special class of the unsupervised ANNs group. SOMs were developed by Kohonen (1984, 1995). The learning process applied to these networks follows the competitive learning paradigm. SOMs construct topology-preserving mappings of the input data in a way that the location of a processing element carries semantic information. The SOM can be considered as a specific type of clustering algorithm. A large number of clusters are chosen and arranged on a square or hexagonal grid in one or two dimensions. This grid is in essence a lattice of processing elements of the SOMs single computational layer. Input patterns representing similar examples are mapped to nearby nodes of the grid.

There have been multiple applications of SOMs to mining related problems in the past. A system consisting of SOMs and an expert system was developed for the evaluation of coal mine roof supports (Signer and King, 1992; King et al., 1993). Millar et al. used self-organising networks to model the complex behaviour of rock masses by classifying input variables related to the rock stability into two groups: failure or stability (Millar and Hudson, 1993). Petersen and Lorenzen (1997) applied the SOM to the modelling of gold liberation from diagnostic leaching data. Walter (1999) used SOMs for the classification of mine roof strata into one of 32 strength classes. The developed system can provide an estimate of strength within two seconds giving the drill operator a warning almost in real time when a potentially dangerous layer is reached.

4. Pattern development

4.1. Data sources

Probably the most important aspect of LavaNet is the pattern generation options. One of the common time consuming issues in neural network development is the generation of training and validation patterns from existing data sources such as drillhole databases, triangulation models, block models, formatted data files and vector data in a format compatible with neural network development software. LavaNet can handle any of the available data sources in VULCAN™ and practically covers any possible sources of information in a general mine planning package. Each data source, whether it is a database or a model, normally hosts a number of variables or parameters, such as coordinates, assay values, rock codes, etc. These can be numerical or alphanumerical depending on what the variable represents. Using LavaNet Direct Patterns options, and depending on the number of inputs and outputs defined through network setup, any of the available variables from a particular data source can be assigned as an input or an output. Fig. 4 shows a particular example where a composite map file (formatted) is used as the source for pattern information. The network architecture has three inputs and one output. The composite sample coordinates (MIDX, MIDY and MIDZ) are chosen

Author's personal copy

as the network's inputs and Au grade is used as the output. The pattern file to be generated in this example is for training purposes and all values are normalised. LavaNet reads and writes pattern files in SNNS format.

The number of inputs and outputs is normally controlled by the network architecture. However, certain data sources have a very specific dimensionality, such as grid models where only three parameters are available—the two coordinates for each node and its value (Table 3). This must be considered before defining the network architecture. VULCAN™ provides a number of ways to combine information from multiple sources in a single model or file, so data source variable limitations should not be a problem for users of LavaNet.

LavaNet can generate pattern files for network development and application either by direct usage of data source parameters or through searching in 2D or 3D space for sample pairs. Search patterns can be generated by searching the data source for sample pairs in the space defined by user selectable dimensions.

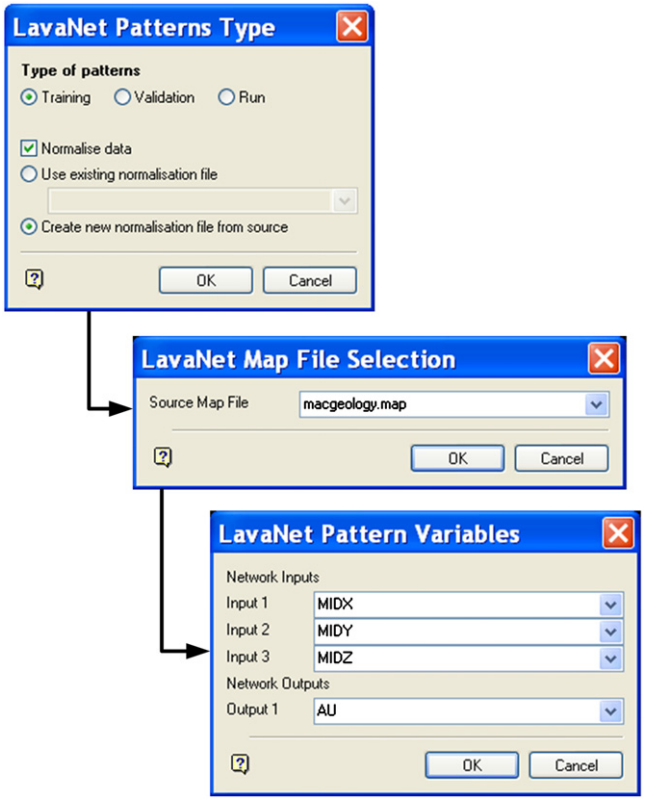


Fig. 4. Example of training pattern generation using a map file as a source of data.

4.2. Direct patterns

Direct patterns can be formed by converting the information contained in a layer, triangulation, grid or any of the data sources into patterns. In case of a triangulation, for example, each node will produce a pattern consisting of three parameters—the node coordinates. Information from multiple models or databases can be combined using already existing Envisage functionality. Typically, a database will be the source of information for the generation of training and validation patters, while a model structure can provide the desired application (run) patterns. For example, a drillhole database can provide training and validation patterns using any of the sample variables, while a block model can provide the application patterns. It is important, in this case, that both share the same input and output space.

4.3. Search patterns

In addition to direct patterns derived by straight conversion of available dimensions from each model structure or database, LavaNet can help explore the relationship between database samples and/or model nodes through the calculation of distance and direction between them, either in two or three dimensional space (Fig. 5). The actual dimensions used in calculating distance and direction between samples or nodes can be anything from standard coordinates to assay values. In other words, searching can be performed in spaces other than the 2D or 3D coordinates space. Distance and direction are included as inputs in the patterns generated. A neighbour point support variable (such as sample volume) can also become a third input in each pattern. The choice of model input variable also defines the output variable for training and validation patterns. For example, if search patterns are generated for a drillhole database and a particular assay field is chosen as model input variable, then the assay field value of a neighbour sample will be used as one of the inputs, while the assay field value at the search centre will be used as the single output. A full pattern, in this case, would be as follows:

Inputs
Distance between neighbour and search centre sample (if selected)
Direction between neighbour and search centre sample (if selected)
Neighbour sample support field value (if selected)
Neighbour sample assay field value
Output
Search centre assay field value

Therefore, search patterns can have a maximum of four inputs and one output. LavaNet also provides the facility to split search

Table 3
Data sources and available parameters for network inputs and outputs.

Data source	Inputs description	Inputs count	Outputs description	Outputs count
MAP file	Sample coordinates, sample volume, assays, codes, etc.	Unlimited	Sample coordinates, sample volume, assays, codes, etc.	Unlimited
CSV file	Sample coordinates, sample volume, assays, codes, etc.	Unlimited	Sample coordinates, sample volume, assays, codes, etc.	Unlimited
ISIS database	Sample coordinates, sample volume, assays, codes, etc.	Unlimited	Sample coordinates, sample volume, assays, codes, etc.	Unlimited
Triangulation model	Node *X*, *Y* coordinates	2	Node *Z* coordinate	1
Block model	Block centroid coordinates, volume, variables	Unlimited	Block centroid coordinates, volume, variables	Unlimited
Grid model	Node *X*, *Y* coordinates	2	Node *Z* coordinate	1
Vector data	Point *X*, *Y* or *X*, *Y*,*Z* coordinates	2 or 3	Point *Z*, *W*[a] value	1

[a] *W* is an extra parameter that Envisage stores in each point in addition to the standard *XYZ* coordinates and point name, and can be used to store numerical information.

Author's personal copy

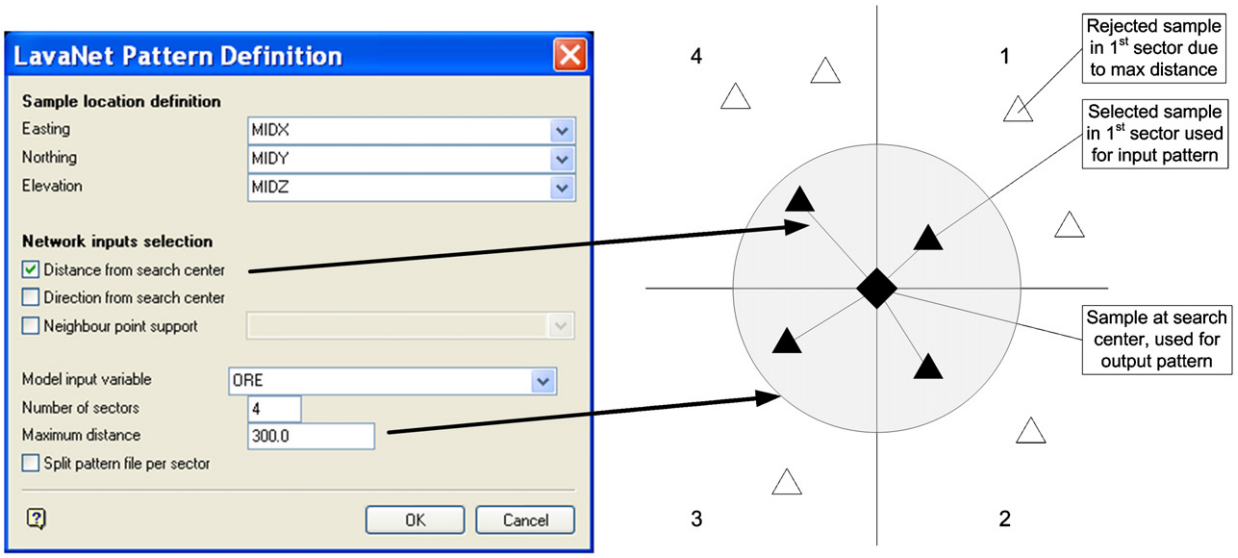


Fig. 5. Defining patterns by searching in 2D or 3D.

patterns by sectors allowing quadrant, octant or any other sector based search to be performed. Search patterns can then be split into multiple pattern files, one for each sector. This is important if a modular neural network is to be developed for modelling the selected output variable, similar to the GEMNet II (Kapageridis, 1999) neural network based grade estimation system.

4.4. Normalisation

LavaNet has the ability to normalise data between an upper and lower bound. Each sample of data is multiplied by an *amplitude* value and shifted by an *offset* value. The amplitude and offset are referred to as *normalisation coefficients*. These coefficients are computed per input and output, meaning that there is a unique amplitude and offset value for each input and output. The coefficients are stored in a normalisation file named after the pattern file name and a .txt extension within the working area directory of Vulcan.

Normalisation coefficients are computed based on the minimum and maximum values found across all data in the selected data source. As one data source may be used for training and validation, and another be used for running the trained network, it is important to generate and use the normalisation file from the data source with the greater extents in the given input and output spaces. For example, if a drillhole database is used as the data source for training and validation, and a block model that includes spatially all drillholes is used for running the network then the block model normalisation file should be used to normalise training, validation and run pattern files. This will guarantee that all of the patterns of the selected data sources will fall between the upper and lower bounds.

The normalisation file is also used by LavaNet to de-normalise the network data, i.e. to put it in terms of the original data. The inverse of the amplitude and offset is applied to each input and output before writing the data. LavaNet normalisation files contain the following ASCII data per input/output parameter:

*amplitude_input*1 *offset_input*1 *minimum_input*1 *maximum_input*1
*amplitude_input*2 *offset_input*2 *minimum_input*2 *maximum_input*2
⋮
amplitude_outputn offset_ outputn minimum_ outputn maximum_ outputn

The number of rows in the file corresponds to the total number of inputs and outputs, i.e. for a pattern file with three inputs and one output there will be four rows of normalisation coefficients. Input coefficients are listed first. These coefficients are calculated using the following formula:

$$Amplitude(i) = (UpperBound - LowerBound)/(Maximum(i) - Minimum(i))$$

$$Offset(i) = UpperBound - Amplitude(i) * Maximum(i)$$

where $Maximum(i)$ and $Minimum(i)$ are the maximum and minimum values found within input/output i, and *UpperBound* and *LowerBound* are the values used as the limits of normalised values. Values 0.1 and 0.9 are hard-coded as the Upper and Lower bounds. The pattern generation options normalise data using the following formula:

$$Data(i) = Amplitude(i) * Data(i) + Offset(i)$$

The results options use the following formula to de-normalise the data:

$$Data(i) = (Data(i) - Offset(i))/Amplitude(i)$$

LavaNet also provides the ability to generate a normalisation file using user defined upper and lower bounds. This is important when the training, validation and application patterns have different input and output space limits. The user can find the universal minimum and maximum of all patterns (using standard Envisage functions) and use them in LavaNet to generate a normalisation file that will be used to normalise and de-normalise all pattern files.

5. Network development and application

5.1. Network initialisation

Network development is a multiple stage process in LavaNet. After the generation of the undeveloped network file, the network is initialised using one of the available initialisation functions in LavaNet (Table 2). Initialisation is important as it provides the starting point for further development of the network, by giving appropriate values to the free parameters of the network (initial weights, biases, function centres, etc.)

Author's personal copy

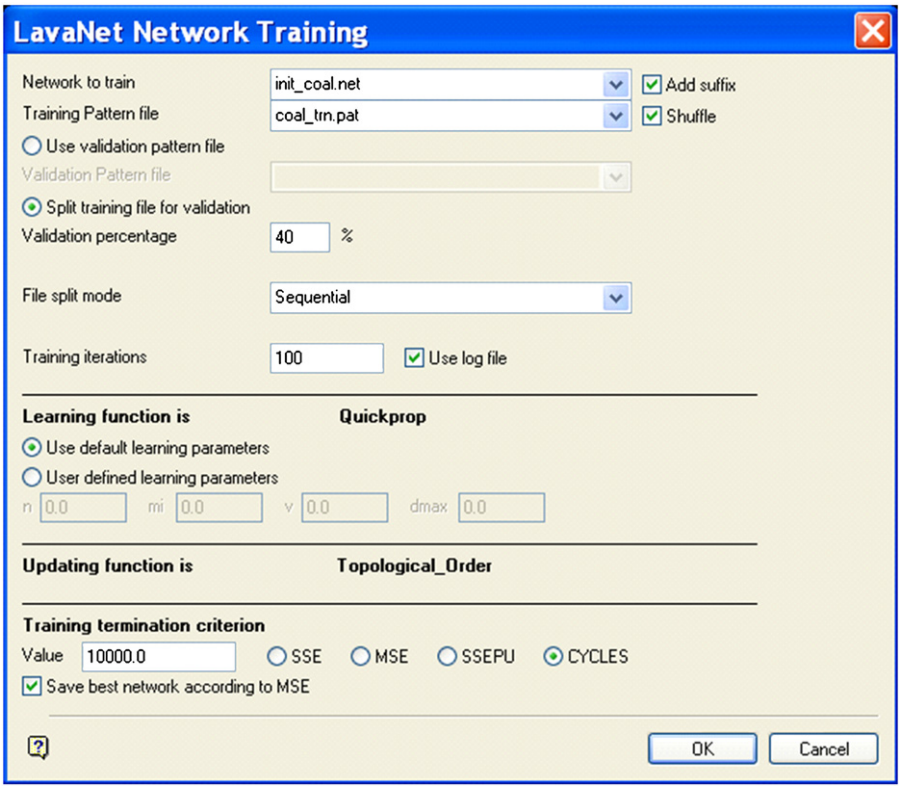


Fig. 6. Network training and validation panel with choice of pattern files, learning parameters and training termination criterion.

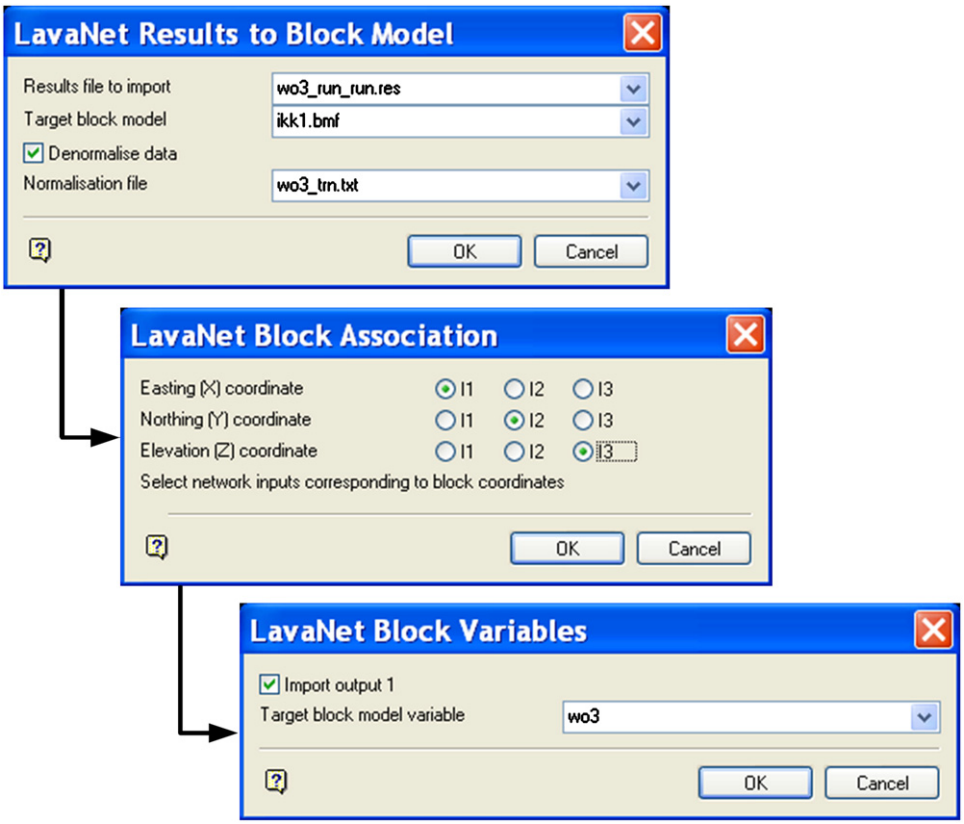


Fig. 7. Importing results of neural network application to a block model variable with de-normalisation and block centroid association.

Author's personal copy

5.2. Network training

After initialisation, the network can be trained using an appropriate learning function (Table 2), depending on the type of network. LavaNet provides the ability to split the training pattern file to two subsets that will be used for training and validation (Fig. 6). This can be done with training patterns first and validation last or evenly. The percentage used for each subset is also defined by the user.

Termination of training can be based on one of the four criteria provided by SNNS (sum or mean of squared error, sum of squared error per unit or number of training cycles). LavaNet can store the network state that produces the lowest mean squared error (MSE) when termination is based on cycles, e.g. the network can be trained for 10,000 cycles but MSE was lowest at 7500 leading to LavaNet saving the network state at that point and not at the end of training. As all network development actions in LavaNet are stored in Batchman compatible script files, the user can combine them using a text editor to further automate the development process once it is finalised.

5.3. Network application

In order to use a trained network, it is necessary to first generate an application pattern file. This file contains only inputs that can be normalised using the coefficients fixed before training. It is clearly important to use the same normalisation used during training to assure that the network receives application inputs in the same range and scale with the training examples and that the generated outputs will also be able to get properly de-normalised. The output from neural network application comes in the form of a standard SNNS results file (with a .res extension) that can be imported to a Vulcan model or layer, or be used as training pattern for further neural network development.

6. Results importing and further processing

Currently, LavaNet can import results from network application in two forms—as a vector data layer and in block model variables. Once results are imported to a vector data layer, they can be directly converted to a model using any of the modelling procedures available in Vulcan™, such as grid and triangulation modelling. The example shown in Fig. 7 demonstrates how particular block model variables can be populated using results from network application. The association of particular blocks with result file records is achieved through 3D coordinates that need to be identified in the result file input parameters. Before this association is established, the result file input parameters need to be de-normalised to bring them back to the original block model coordinate system. Outputs from the results file also need to be de-normalised before importing to a layer or block model variable.

7. Application examples

In the following sections, the flexibility of LavaNet is demonstrated using two examples: modelling in two dimensions (grid and triangulation models), and quality estimation in three dimensions (block model). However, LavaNet can be used for any samples classification (both supervised and unsupervised) or spatial interpolation problem in mine planning.

7.1. 2D modelling example

The first example is a simple 2D application of a MLP network for the construction of a mineable thickness model of a lignite deposit. Mineable thickness was evaluated in each drillhole available and a pattern file is generated with two inputs, the *X* and *Y* coordinates of the drillhole, and one output, the mineable lignite thickness. Part of the SNNS pattern file built by LavaNet using the drillhole database thickness information is shown in Fig. 8. Each pattern consists of two lines, the first giving the inputs and the second giving the output (both normalised).

The pattern file was split 60–40% for training and validation. The network was first initialised and then trained using the script that

```
SNNS pattern definition file V1.4
generated at Thu Jun 22 15:01:23 2006

No. of patterns  : 1656
No. of input units  : 2
No. of output units : 1

# Pattern 631
0.536582871292458 0.35064173900141
0.420175631174534
```

Fig. 8. Example of pattern file generated by LavaNet with normalisation of inputs and output.

```
loadNet("init_cotk.net")
loadPattern("lavanet_trn.pat")
loadPattern("lavanet_vln.pat")
setPattern("lavanet_trn.pat")
min_error = 1
setShuffle (TRUE)
setLearnFunc("BackpropMomentum", 0.01, 0.2, 0, 0.01)
setUpdateFunc ("Topological_Order")
repeat
    for i:=1 to 500 do
      trainNet ()
    end for
    setPattern ("lavanet_vln.pat")
    testNet()
    if min_error > MSE then
      min_error = MSE
      saveNet("train_init_cotk.net")
      print ("Best MSE = ", MSE)
    end if
    criterion := CYCLES
    print ("Cycles=",CYCLES,"CYCLES=",CYCLES)
    setPattern ("lavanet_trn.pat")
until criterion >= 20000
```

Fig. 9. Batchman training script generated by LavaNet.

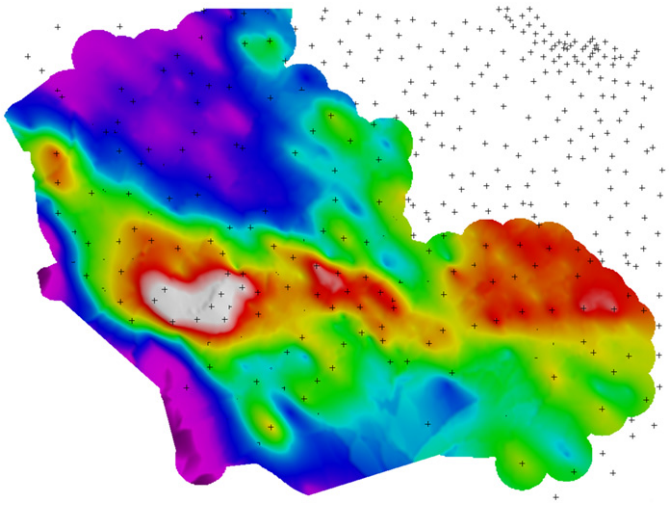


Fig. 10. Model of mineable lignite thickness built using LavaNet.

Author's personal copy

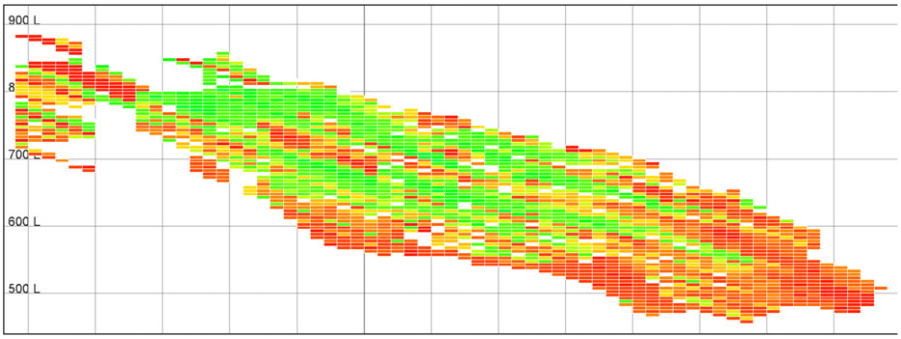


Fig. 11. Block model section showing WO_3 estimates from LavaNet RBF network.

was generated by LavaNet (Fig. 9). Training lasted 20,000 cycles while the network's performance was validated every 500. The best network state according to MSE was saved to a separate network file. This network was used to generate a grid model of mineable lignite thickness using the appropriate LavaNet options. A separate application pattern file was generated for this, using an existing grid model to provide the inputs for network application. Fig. 10 shows the produced model of lignite thickness.

7.2. Block model estimation example

In this second example, an RBF network is used to estimate WO_3 grade of a tungsten deposit in 3D (*X*, *Y*, *Z* coordinates as inputs). Drillhole composite data are used to generate the pattern files to train and validate the network, while a block model is used to generate the application pattern file. The development of the network is similar to the previous example—the initialisation, learning and updating functions differ as a different type of network is now used. The trained network is applied on the block model pattern file and the results are de-normalised and imported back into the block model. Fig. 11 shows a section through the block model coloured by WO_3 estimates of the LavaNet RBF network.

8. Conclusions and further work

LavaNet is an interface to a neural network development environment, built inside a general mine planning package, which allows quick and effortless development and application of artificial neural networks for mining and environmental problems. LavaNet is constantly under development with further options and functionality being added that take advantage of new capabilities provided by new versions of Vulcan's LAVA scripting language, including extensions to the graphical user interface, model structures and data sources.

References

Ash, T., 1989. Dynamic Node Creation in Backpropagation Networks. ICS Report 8901. Institute of Cognitive Science, University of California, San Diego, California.

Caiti, A., Parisini, T., 1991. Mapping of ocean sediments by networks of parallel interpolating units. In: proceedings of the IEEE Conference on Neural Networks for Ocean Engineering, Washington, DC, pp. 231–238.

Clarici, E., Owen, D., Durucan, S., Ravenscroft, P., 1993. Recoverable reserve estimation using a neural network. In: proceedings of the 24th International Symposium on the Application of Computers and Operations Research in the Minerals Industries (APCOM), Montreal, Quebec.

Cortez, L.P., Sousa, A.J., Durao, F.O., 1998. Mineral resources estimation using neural networks and geostatistical techniques. In: proceedings of the 27th International Symposium on the Application of Computers and Operations Research in the Minerals Industries (APCOM). The Institution of Mining and Metallurgy (IMM), London.

Fahlman, S.E., 1988. Fast learning variations on backpropagation: an empirical study. In: Touretzky, D.S., Hinton, G., Sejnowski, T. (Eds.), Proceedings of the 1988 Connectionist Models Summer School. Morgan Kaufmann Publishers, San Mateo, California.

Garcia, G., Whitman, W.W., 1992. Inversion of a lateral log using neural networks. SPE 24454, Society of Petroleum Engineers.

Haykin, S., 1999. Neural Networks—A Comprehensive Foundation. Prentice-Hall, New Jersey 823 pp.

Kapageridis, I., 1999. Application of artificial neural network systems in grade estimation from exploration data. Ph.D. Dissertation, Department of Mineral Resources Engineering, University of Nottingham, Nottingham, United Kingdom, 260pp.

Kapageridis, I., 2002. Artificial neural network technology in mining and environmental applications. In: proceedings of the 11th International Symposium on Mine Planning and Equipment Selection (MPES 2002). VŠB—Technical University of Ostrava, Prague.

Kapageridis, I., 2003. Investigation of dimensionality effects on the performance of artificial neural networks applied to grade estimation from exploration data. Post Doctorate Dissertation, Department of Geotechnology and Environment, Technological Educational Institute of Western Macedonia, Kozani, Greece, 81pp.

Kapageridis, I., 2005. Input space configuration effects in neural network based grade estimation. Computers and Geosciences 31 (6), 704–717.

King, R.L., Hicks, M.A., Signer, S.P., 1993. Using unsupervised learning for feature detection in a coal mine roof. Engineering Applications of Artificial Intelligence 6 (6), 565–573.

Kohonen, T., 1984. Self-Organisation and Associative Memory. Springer-Verlag, Berlin, 312pp.

Kohonen, T., 1995. Self-Organising Maps, second ed. Springer-Verlag, Berlin, 534pp.

Lowe, D., 1995. Radial basis function networks. In: Arbib, M.A. (Ed.), The Handbook of Brain Theory and Neural Networks. MIT Press, Cambridge, pp. 930–934.

Millar, D.L., Hudson, J.A., 1993. Rock engineering system performance monitoring using neural networks. Preprints of the 'Artificial Intelligence in the Minerals Sector' (one day symposium held at the University of Nottingham).

Moody, J., Darken, C., 1989. Fast learning in networks of locally-tuned processing units. Neural Computation 1, 281–294.

Park, J., Sandberg, I., 1993. Approximation and radial-basis-function networks. Neural Computation 5, 305–316.

Petersen, K.R.P., Lorenzen, L., 1997. Gold liberation modelling using neural network analysis of diagnostic leaching data. In: Hoberg, H., von Blottnitz, H. (Eds.), Proceedings of the XX International Mineral Processing Congress, vol. 1. Aachen, Germany, pp. 391–400.

Powell, M.D., 1992. The theory of radial basis function approximation in 1990. In: Light, W. (Ed.), Advances in Numerical Analysis Vol. II: Wavelets, Subdivision Algorithms, and Radial Basis Functions. Oxford Science Publications, Oxford, pp. 105–210.

Powell, M.D., 1987. Radial basis functions for multivariate interpolation: a review. In: Mason, J., Cox, M. (Eds.), The Approximation of Functions and Data. Clarendon Press, Oxford, 694pp.

Rogers, S.J., Fang, J.H., Karr, C.L., Stanley, D.A., 1992. Determination of lithology from well logs using neural networks. Bulletin of the American Association of Petroleum Geologists, 731–739.

Samanta, B., Bandopadhyay, S., 2009. Construction of a radial bases function network using an evolutionary algorithm for grade estimation in a placer gold deposit. Computers and Goesciences 35 (8), 1592–1602.

Schwartz, R.L., Phoenix, T., Foy, B.D., 2008. Learning Perl, fifth ed. O'Reilly Media, Inc., 348pp.

Signer, S.P., King, R.L., 1992. Evaluation of coal mine roof supports using artificial intelligence. In: proceedings of the 23rd International Symposium on the Application of Computers and Operations Research in the Minerals Industries (APCOM), Arizona, USA.

SNNS, 1996. Stuttgart Neural Network Simulator Version 4.2 User's Manual, Report No. 6/95. Institute for Parallel and Distributed High Performance Systems (IPVR), University of Stuttgart, 350pp.

Walter, K.U., 1999. Neural network technology for strata strength characterisation. In: proceedings of the International Joint Conference on Neural Networks (IJCNN '99),

Author's personal copy

International Neural Network Society and The Neural Networks Council of IEEE, Washington, DC.
Wu, X., 1991. Neural network-based material modelling. Ph.D. Thesis, Department of Civil Engineering, University of Illinois, Urbana, Illinois.
Wu, X., Zhou, Y., 1993. Reserve estimation using neural network techniques. Computers and Geosciences 9 (4), 567–575.
Yama, B.R., Lineberry, G.T., 1999. Artificial neural network application for a predictive task in mining. Mining Engineering 51 (2), 59–64.