



# SELF LEARNING DATA MODEL DESIGNED BY ADVANCED NEURAL NETWORK

Naveen Kumar N, M.Senthil Kumar,M.E,

Student, Assistant Professor ,  
Computer Science and Engineering,  
Erode Sengunthar Engineering College, Erode, India

**Abstract :** With the rapid development of deep learning in recent years, recommendation algorithm combined with deep learning model has become an important direction in the field of recommendation in the future. Personalized learning resource recommendation is the main way to realize students' adaptation to the learning system. Based on the in-depth learning mode, the time human action data are obtained and observed, and further learning analysis technology is used to construct' special learning mode. This paper proposes a futuristic learning model of deep neural networks for all modern data models. The data model proposes to satisfy the functions of futuristic applications. These things may improve or eliminate the traditional data modern training and time to taken to train the data models. A propose data model design would be a neural network, the combination of RNN and CNN (Recurrent and convolutional neural network). This is a new kind of neural network that would be designed by modified traditional prediction and conditional algorithms. It may improve the prediction precision and well trained more efficient decision making model for all purposes.

**IndexTerms –Neural Network, Adaptive Model.**

## 1. INTRODUCTION

Deep learning is a hot topic these days. But what is it that makes it special and sets it apart from other aspects of machine learning? That is a deep question (pardon the pun).

To even begin to answer it, we will need to learn the basics of neural networks. Neural networks are the workhorses of deep learning. And while they may look like black boxes, deep down they are trying to accomplish the same thing as any other model — to make good predictions. In this post, we will explore the ins and outs of a simple neural network. And by the end, hopefully you (and I) will have gained a deeper and more intuitive understanding of how neural networks do what they do.

Let's start with a really high level overview so we know what we are working with. Neural networks are multi-layer networks of neurons (the blue and magenta nodes in the chart below) that we use to classify things, make predictions, etc. Below is the diagram of a simple neural network with five inputs, 5 outputs, and two hidden layers of neurons.

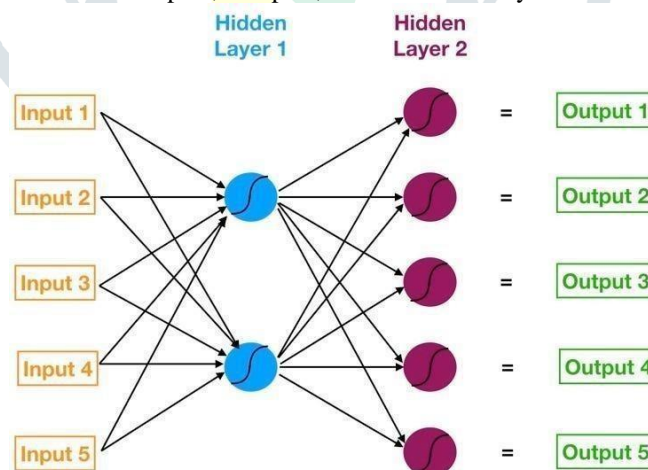


FIGURE 1: NEURAL NETWORK WITH TWO HIDDEN LAYERS

## 1.1 NEURAL NETWORK- AN OVERVIEW

Neural networks are used to mimic the basic functioning of the human brain and are inspired by how the human brain interprets information. It is used to solve various realtime tasks because of its ability to perform computations quickly and its fast responses.

Artificial Neural Network has a huge number of interconnected processing elements, also known as Nodes. These nodes are connected with other nodes using a connection link. The connection link contains weights; these weights contain the information about the input signal. Each iteration and input in turn leads to updation of these weights. After inputting all the data instances from the training data set, the final weights of the Neural Network along with its architecture is known as the Trained Neural Network. This process is called Training of Neural Networks. This trained neural network is used to solve specific problems as defined in the problem statement.

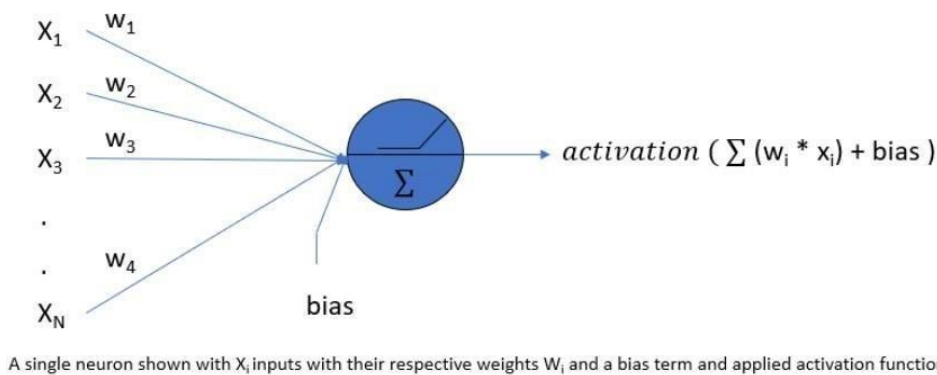


FIGURE 1.2

**Contents**

1. What is Neural Network
2. Types of Neural Networks
3. Types of Learnings
4. How does a Neural Network work
5. Endnotes

**1.2 WHAT ARE NEURAL NETWORKS?**

Neural networks are used to mimic the basic functioning of the human brain and are inspired by how the human brain interprets information. It is used to solve various realtime tasks because of its ability to perform computations quickly and its fast responses.

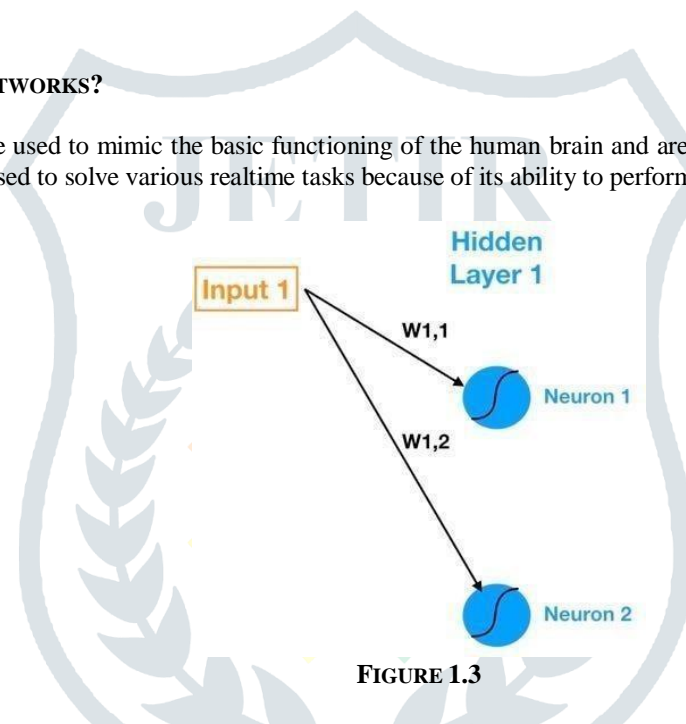


FIGURE 1.3

An Artificial Neural Network model contains various components that are inspired by the biological nervous system.

Artificial Neural Network has a huge number of interconnected processing elements, also known as Nodes. These nodes are connected with other nodes using a connection link. The connection link contains weights, these weights contain the information about the input signal. Each iteration and input in turn leads to updation of these weights. After inputting all the data instances from the training data set, the final weights of the Neural Network along with its architecture is known as the Trained Neural Network. This process is called Training of Neural Networks. This trained neural network is used to solve specific problems as defined in the problem statement.

Types of tasks that can be solved using an artificial neural network include Classification problems, Pattern Matching, Data Clustering, etc.

**1.2.1 TYPES OF NEURAL NETWORKS**

**ANN**– It is also known as an artificial neural network. It is a feed-forward neural network because the inputs are sent in the forward direction. It can also contain hidden layers which can make the model even denser. They have a fixed length as specified by the programmer. It is used for Textual Data or Tabular Data. A widely used real-life application is Facial Recognition. It is comparatively less powerful than CNN and RNN.

**CNN**– It is also known as Convolutional Neural Networks. It is mainly used for Image Data. It is used for Computer Vision. Some of the real-life applications are object detection in autonomous vehicles. It contains a combination of convolutional layers and neurons. It is more powerful than both ANN and RNN.

**RNN**– It is also known as Recurrent Neural Networks. It is used to process and interpret time series data. In this type of model, the output from a processing node is fed back into nodes in the same or previous layers. The most known types of RNN are LSTM (Long Short Term Memory) Networks

Now that we know the basics about Neural Networks, I know that Neural Networks' learning capability is what makes it interesting. There are 3 types of learnings in Neural networks, namely

1. SUPERVISED LEARNING
2. Unsupervised Learning
3. REINFORCEMENT LEARNING

## PROBLEM STATEMENT

An ANN (Artificial Neural Networks) is a system that mimics biological neurons. It processes data and exhibits intelligence by making predictions, recognizing patterns, and learning from historical data. By creating interconnected neurons, ANNs provide various benefits like organic learning, non-linear data processing, fault tolerance, and self-repair. But still, ANNs face some challenges while training. For instance, ANNs require massive volumes of data to train them. This tremendous volume of data is needed so that ANNs can be trained in all the aspects of a particular task. For instance, to train an ANN for image classification, it must be trained with labelled images of every object or living organism that it has to classify.

Training ANNs is a time-consuming process. Also, ANN models that can process large datasets become too complex to manage, maintain, and modify. Due to such challenges, many researchers were motivated to make ANNs adaptive to changes while training. Adaptive neural networks can generalize themselves according to a given problem using various adaptive strategies.

## 2 LITERATURE REVIEW

This chapter gives the overview of literature survey. This chapter represents some of the relevant work done by the researchers.

### 2.1 TITLE - STATISTICAL MODELLING OF ARTIFICIAL NEURAL NETWORKS USING THE MULTI-LAYER PERCEPTRON

**Author - Murray Aitkin** **Journal- Cross Mark**

Multi-layer perceptrons (MLPs), a common type of artificial neural networks (ANNs), are widely used in computer science and engineering for object recognition, discrimination and classification, and have more recently found use in process monitoring and control. "Training" such networks is not a straightforward optimisation problem, and we examine features of these networks which contribute to the optimisation difficulty.

Although the original "perceptron", developed in the late 1950s (Rosenblatt 1958, Widrow and Hoff 1960), had a binary output from each "node", this was not compatible with back-propagation and similar training methods for the MLP. Hence the output of each node (and the final network output) was made a differentiable function of the network inputs. We reformulate the MLP model with the original perceptron in mind so that each node in the "hidden layers" can be considered as a latent (that is, unobserved) Bernoulli random variable. This maintains the property of binary output from the nodes, and with an imposed logistic regression of the hidden layer nodes on the inputs, the expected output of our model is identical to the MLP output with a logistic sigmoid activation function (for the case of one hidden layer).

We examine the usual MLP objective function—the sum of squares—and show its multi-modal form and the corresponding optimisation difficulty. We also construct the likelihood for the reformulated latent variable model and maximise it by standard finite mixture ML methods using an EM algorithm, which provides stable ML estimates from random starting positions without the need for regularisation or cross-validation. Overfitting of the number of nodes does not affect this stability. This algorithm is closely related to the EM algorithm of Jordan and Jacobs (1994) for the Mixture of Experts model.

### 2.2 TITLE - NEURAL NETWORKS AND STATISTICS

**Author – María del Mar Rodríguez del Aguila** **Journal- IEEE**

Multi-layer perceptron's (MLPs), a common type of artificial neural networks (ANNs), are widely used in computer science and engineering for object recognition, discrimination and classification, and have more recently found use in process monitoring and control. "Training" such networks is not a straightforward optimisation problem, and we examine features of these networks which contribute to the optimisation difficulty.

Although the original "perceptron", developed in the late 1950s (Rosenblatt 1958, Widrow and Hoff 1960), had a binary output from each "node", this was not compatible with back-propagation and similar training methods for the MLP. Hence the output of each node (and the final network output) was made a differentiable function of the network inputs. We reformulate the MLP model with the original perceptron in mind so that each node in the "hidden layers" can be considered as a latent (that is, unobserved) Bernoulli random variable. This maintains the property of binary output from the nodes, and with an imposed logistic regression of the hidden layer nodes on the inputs, the expected output of our model is identical to the MLP output with a logistic sigmoid activation function (for the case of one hidden layer).

We examine the usual MLP objective function—the sum of squares—and show its multi-modal form and the corresponding optimisation difficulty. We also construct the likelihood for the reformulated latent variable model and maximise it by standard finite mixture ML methods using an EM algorithm, which provides stable ML estimates from random starting positions without the need for regularisation or cross-validation. Over-fitting of the number of nodes does not affect this stability. This algorithm is closely related to the EM algorithm of Jordan and Jacobs (1994) for the Mixture of Experts model.

We conclude with some general comments on the relation between the MLP and latent variable models.

### 2.3 TITLE - MODEL SELECTION IN NEURAL NETWORKS

**AUTHOR – ULRICH ANDERS, OLAF KORN** **JOURNAL- GOOGLE SCHOLAR**

Recently, there has been a growing interest in the modelling of nonlinear relationships and a variety of test procedures for detecting nonlinearities has been developed. If the aim of analysis is prediction, however, it is not sufficient to uncover nonlinearities. Moreover, we need to describe them through an adequate nonlinear model. Unfortunately, for many applications theory does not guide the model building process by suggesting the relevant input variables or the correct functional form. This particular difficulty makes it attractive to consider an 'atheoretical' but flexible class of statistical models. Artificial neural networks are well suited for this purpose as they can approximate virtually any (measurable) function up to an arbitrary degree of accuracy.

(Hornik/Stinchcombe/White, 1989). This desired flexibility, however, makes the specification of an adequate neural network model even harder. Despite the huge amount of network theory and the importance of neural networks in applied work, there is still little experience with a statistical approach to model selection. The aim of this article is to develop model selection strategies for neural networks which are based on statistical concepts. Taking a statistical perspective is especially important for 'atheoretical' models like neural networks, because the reason for applying them is the lack of knowledge about an adequate functional form. Furthermore, when based on a clearly defined decision rule, model selection becomes more comprehensible and reconstructable. The concepts considered in this article are hypothesis testing, information criteria and cross validation methods. These concepts are the building blocks which constitute the basis of the different model selection strategies which we develop and evaluate in a simulation study. To our knowledge this article provides the first systematic comparison of statistical selection strategies for neural network models.

The overall results of the simulation study are promising as they lead to neural networks which closely approximate the simulated models. Our results demonstrate that a sequence of hypothesis tests produces neural network architectures with the best overall performance. Strategies based on cross validation and information criteria are very accurate for some models although they tend to overfit or underfit others. When information criteria are to be employed, we cannot recommend the use of an estimated penalty term. The remaining part of the article is organized as follows: It is pointed out that the use of the AIC may theoretically not be justified. After that, cross validation techniques, frequently referred to in the neural networks literature, are introduced. Section 5 defines our model selection strategies.

## 2.4 TITLE- DESIGN OF NEURAL NETWORKS FOR TIME SERIES PREDICTION CASE-INITIALIZED GENETIC ALGORITHMS

**Author - Ricardo Bastos Cavalcante Prudencio** **Journal- IEEE**

Time series prediction has been successfully used to support the decision-making in several application areas. A number of techniques has been developed for modelling and predicting time series, among which we highlight the Box- Jenkins approach and the Artificial Neural Networks (ANNs). The former approach, although widespread, is only capable to construct linear models. In contrast, ANNs are able to model non-linear functions, however problems have to be faced concerning the choice of an appropriated network architecture to model the series to be predicted. In this paper, we use Genetic Algorithms (GAs) to optimize the choice of neural networks' architecture for prediction problems. To improve the GA's performance, we propose initializing its first population with well succeeded architectures previously used to predict similar series to the one being modelled. For that, we are developing a case base, in which each case associates a time series to an architecture used to predict it. After the execution of the GA, the resulting ANN architecture is ready for use, and can also be associated to the input series, to produce a new case to be inserted in the base. We expect that the choice of architectures from the case base will become better as more cases are inserted in the base. The initial prototype was tested against a random search algorithm and the results obtained were very encouraging.

A more unifying technique to optimize the architectures of ANNs is via the use of Genetic Algorithms (GAs) [Goldberg 1989]. GAs can be deployed to optimize, at the same time, different architecture's parameters, such as activation functions, hidden nodes, input variables, among others. A problem to be faced here is that, if the GA's initial population is randomly generated, the algorithm may waste time exploring poor regions in the search space of parameters before converging to the good regions.

One way to improve the GAs' performance, by providing useful information about the search space, is through 'case-initialization', in which the first GA's population consists of well succeeded solutions to problems which are similar to the one being tackled. In this paper, we use GAs to define the architecture of neural networks for time series prediction problems. To improve the results of the GA for a series, we propose initializing its first population with well-succeeded architectures previously used to predict similar series. For that, we are developing a case base in which each case associates a time series to an architecture used to predict it. Given a new time series, a query retrieves from the case base the architectures used for modeling the most similar series to the input one, and these architectures are inserted on the GA's initial population. After the execution of the GA, the resulting ANN architecture is ready for use and can also be associated to the input series, in order to form a new case to be inserted in the base. We expect that the 'case-initialization' will become better as more cases are inserted in the base. The initial prototype was tested against a random search algorithm for 25 time series. For the validation errors, the GA showed superiority in 64% of the series and for the test errors the obtained gain was observed in 60% of the series. In the following section, we discuss briefly the time series prediction problem. In section 3, we propose a methodology to design neural models for time series prediction. This is followed by the presentation of the results obtained with the initial prototype. Finally, we present the conclusion and future work.

## 2.5 TITLE- A NEURAL-STATISTICAL APPROACH TO MULTITEMPORAL AND MULTISOURCE REMOTE-SENSING IMAGE CLASSIFICATION

**Author – L. Bruzzone** **Journal- IEEE**

A data fusion approach to the classification of multisource and multitemporal remote-sensing images is proposed. The method is based on the application of the Bayes rule for minimum error to the "compound" classification of pairs of multisource images acquired at two different dates. In particular, the fusion of multisource data is obtained by using multilayer perceptron neural networks for a nonparametric estimation of posterior class probabilities. The temporal correlation between images is taken into account by the prior joint probabilities of classes at the two dates. As a novel contribution of this paper, such joint probabilities are automatically estimated by applying a specific formulation of the expectation-maximization (EM) algorithm to the data to be classified. Experiments carried out on a multisource and multitemporal data set confirmed the effectiveness of the proposed approach.

### 3 METHODOLOGY

## Unsupervised Learning in ML

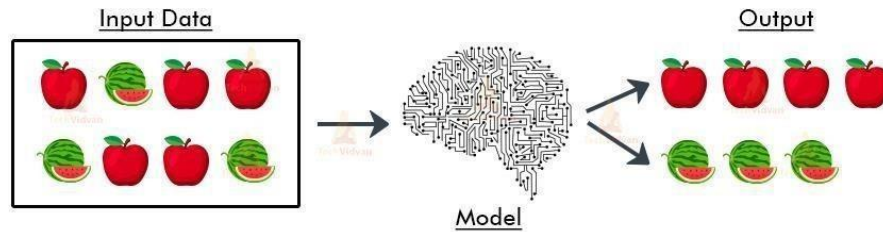


Figure 3

The documents included in this review were identified using different methods: Internet search engines, access to electronic journals and others. The search was performed using the key words "statistic", "probability" and "neural networks", and included articles published in scientific journals, conference contributions and any document published on the Internet that met the previously defined criteria. The classification of documents proposed in this review is chiefly related to the manner in which the ANNs and statistics are applied. Thus, the following first classification can be considered. Comparison between neural networks and statistical methods. This group includes documents that report theoretical comparisons, similarities and differences between neural networks and statistical procedures. Application of statistical procedures to neural networks, including documents that use statistical methods to define the neural network model.

Implementation of statistical methods by means of neural networks. This is the inverse of the above, and includes documents that implement statistical methodology and/or probability using neural networks. Application of statistical procedures and neural networks to different problems, including documents that apply and compare ANNs and statistical techniques in relation to concrete problems in different scientific fields. Seventy-two articles on neural networks and statistics published between 1990 and 2003 were reviewed.

### 4 TRAINING A DATA

While stepping into the world of deep learning, a lot of developers try to build neural networks and face disappointing results. They may end up seeing that the training process is not able to update the weights of the network or that model is not able to find the minimum of the cost function. These are very common issues we face while training neural networks and hence, need to have a good strategy to tackle them.

This blog will help the readers understand and solve the following issues while training neural networks:

1. Overfitting vs Underfitting
2. Vanishing Gradient
3. Local Minima
4. Setting the right learning rate strategy
5. Picking the mini batch size

#### PROBLEM: OVERFITTING VS UNDERFITTING

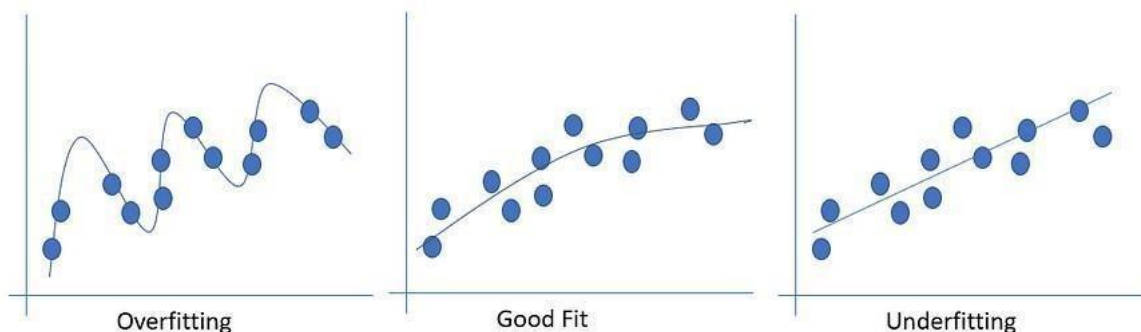


Figure 4.1

Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the model's ability to generalize. Overfitting is more likely with nonparametric and nonlinear models that have more flexibility when optimising a loss function.

On the other hand, underfitting refers to a model that can neither model the training data nor generalize to new data.

The concept of overfitting/underfitting is closely related to the problem of variance/bias tradeoff. Variance is the amount that the estimate of loss function will change if different training data was used.

- Low Variance: Suggests small changes to the estimate of the target function with changes to the training dataset.
  - High Variance: Suggests large changes to the estimate of the target function with changes to the training dataset.
- On the other hand, bias refers to the simplifying assumptions made by a model to make the loss function easier to learn.
- Low Bias: Suggests fewer assumptions about the form of the loss function.
  - High-Bias: Suggests more assumptions about the form of the loss function
- To identify the right fit of a model, we can look at the performance of a machine learning algorithm on training and test dataset over the number of training iterations/epochs.

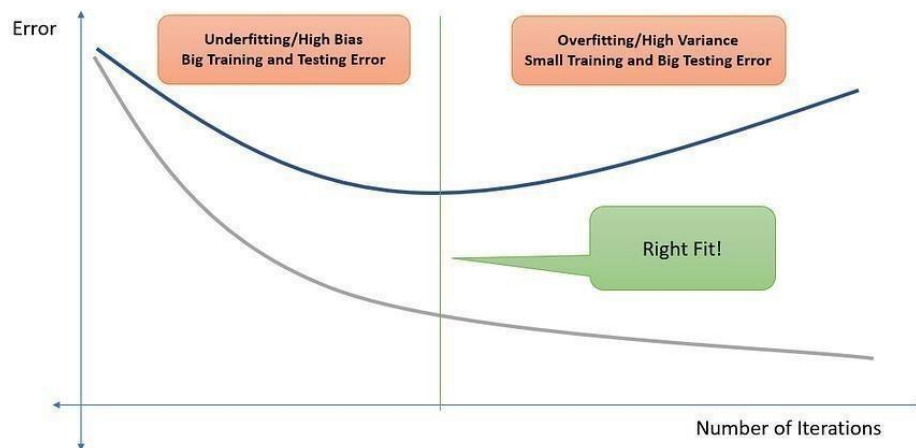


FIGURE 4.2

As the algorithm learns, the error for the model on both the training and test data falls. But after a point, the performance on the training dataset keeps decreasing but the model starts learning the noise in the training data as well. This can be seen by the rising error in the test set. The sweet spot is the point just before the error on the test data starts to jump up (as shown in the image above).

Along with the number of epochs, the overfitting/underfitting problem is a key factor for deciding the number of hidden neurons and the number of hidden layers. They determine the capacity of the model to understand complicated relationships in the data. But similar to the number of epochs, having too many of them will lead to overfitting while too few will lead to underfitting. Solution

There are three popular approaches to overcome this:

**1. Early stopping:** Early stopping (also called “early termination”) is a method that allows us to specify a large number of training epochs and stop training once the model performance stops improving on the test dataset. It monitors the progress of training, but also stops the training when certain conditions are met. There are two predefined stopping monitors in the TensorFlow package:

- a. Stop At Step Hook: Training stops after a certain number of steps.
- b. Nan Tensor Hook: Monitors loss and stops training if it encounters NaN loss.

**2. Regularization:** This is a form of regression, that either eliminates (L1 regularisation) coefficients or shrinks (L2 regularisation) the coefficient estimates towards zero. Thus, it discourages fitting a complicated model to avoid the risk of overfitting.

**3. Dropout:** Dropout is based on the idea of training multiple neural networks on different architectures in parallel. The basic idea is that certain layer outputs are randomly dropped out during training. Thus after each iteration, the outputs are based on a different combination of neurons in the neural network.

Apart from these options, we can try out the following heuristic advice for the number of hidden layers:

“In practice it is often the case that 3-layer neural networks will outperform 2-layer nets, but going even deeper (4, 5, 6-layer) rarely helps much more. This is in stark contrast to Convolutional Networks, where depth has been found to be an extremely important component for a good recognition system (e.g. on order of 10 learnable layers).” ~ Andrej Karpathy

Finally, for selecting the number of neurons in hidden layers, we can follow the following rules:

1. Number of hidden neurons should be between size of the input layer and the output layer.
2. The most common number of hidden neurons is  $\sqrt{\text{input layer nodes} \times \text{output layer nodes}}$
3. Number of hidden neurons should keep decreasing in subsequent layers to get more and more close to the structure of output layer.

It is important to note that all of these are heuristic rules and should be tweaked as per the problem statement.

### Problem: Vanishing Gradient

Some activation functions, like sigmoid function, have a very small derivative especially when the input values are away from the centre of the function. This means that the rate at which the weights of the neural network are updated is very small. This problem gets worse as we add more hidden layers as the derivatives from the different layers get multiplied and consequently, a

large number of small numbers get multiplied leading to a final change which is close to zero. This means that the model is not able to react enough to find the optimal values.

#### SOLUTION:

The best way to handle vanishing gradient is to replace the sigmoid activation function in the hidden layers with other functions like ReLU or Tanh. Please refer to my blog on the list of activation functions available and where to use them.

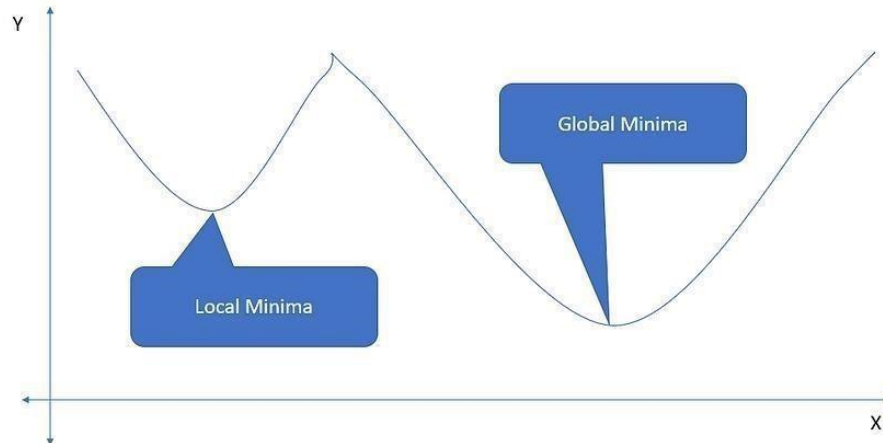


FIGURE 4.3

#### Problem: Local Minima

It is very common for the loss functions to have a number of peaks and troughs. A local minima is a point which is the minimum value within its neighbourhood but not on the loss function. As shown in the image above, once our model hits a local minima, simple gradient descent algorithm cannot find any direction to further minimize the loss function in the neighbourhood and will be stuck there. Thus, we will not be able to find the optimal weights for our neural network.

#### SOLUTION:

We have two options for handling local minima problem:

1. Random Restart: It basically means that we can start from different points of the loss function and perform gradient descent from all of them. This approach significantly increases our chances of finding the global minima.
2. Momentum: This approach is inspired by the idea of an object moving very fast on an uneven surface such that its momentum pushes it over small humps. The basic idea is that the gradient at a local minima is zero but the gradient in the previous epochs will have non-zero values and taking their average will help us get out of the hump at the local minima. Formally, momentum is a weighted average of past gradients and greater weight is applied to recent gradients, leading to an exponentially decaying average of the gradients. The weight is usually called velocity and varies between 0 and 1. The best way to find the value of velocity is to use cross-validation.

#### PROBLEM: MINIBATCH SIZE

Mini-batch gradient descent is a variation of the gradient descent algorithm that splits the training dataset into small batches that are used to calculate model error and update model coefficients. It is the most common form of gradient descent algorithm (the other options being batch and stochastic gradient descent). The batch size is determined by Mini batch size parameter and the way we select it impacts the resource requirements along with the training speed of neural networks. Too little would lead to a slow training progress but helps in converging to global minima.

Selecting a high Mini batch size would lead to faster training but requires more memory and computational resources. It also has the risk of being stuck at local minima. Solution:

A Mini batch size of 32 is a good place to start and we can try out 64, 128 and 256 as well if that doesn't work. The most common set of values used for most problems are: 1, 2, 4, 16, 32, 64, 128 and 256.

#### PROBLEM: HOW TO SET THE LEARNING RATE

Learning rate refers to the amount by which the weights are updated during training (also known as step size) of machine learning models. It is one of the important hyper parameters used in the training of neural networks and the usual suspects are 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001 and 0.000001. Setting a very low learning rate, makes our model very slow in terms of identifying the minimum point on the cost function while picking a high value will make us miss the optimal point as the model will keep taking big strides. Looking at the extreme version of this, a high learning rate can result in the performance of the model oscillating between training epochs which indicates that weights are diverging from the optimal value. At the other extreme end, a very small value may prevent the model from converging or be stuck at a local minima. There are typically 3 scenarios we will be facing:

1. Validation error decreasing fast during training: Indicates good choice of learning rate.
2. Validation error decreasing very slowly during training: Indicates learning rate needs to be increased
3. Validation error increasing slowly during training: Indicates learning rate needs to be decreased

**SOLUTION:**

Some modern versions of stochastic gradient descent algorithms, such as Adam, allow for adaptive learning rates where the performance of the model on the training dataset is monitored by the algorithm and the learning rate is adjusted in response. The learning rate is gradually reduced if the error rate starts to fall and increased if the error rate doesn't improve for a number of epochs. If that doesn't work then we can try other optimisers like AdaGrad or RMSProp (which also have adaptive learning rate mechanism). In a separate blog, I will discuss the various optimisers available for training neural networks and their pros and cons.

If we want to manually set the parameter value then we should check the learning dynamics of the model by creating a line plot of loss over training epochs. We can then check the following:

- Is the rate of learning fast or slow?
- Have there been very small changes in the error rate indicating low learning rate value?
- Are we seeing oscillations in loss indicating high learning rate value?

We can also manually set a learning rate schedule where the learning rate is decreased linearly/exponentially from a maximum to a minimum value after a certain number of epochs.

**5 DISCUSSION**

Many edge-based stream analytics systems focus on deadline driven processing, bandwidth limited scheduling and real-time processing. Video analytics using edge and in-transit network nodes with a deadline-time for each job and priority based scheduling.

Gig sight saves bandwidth by running computer vision algorithms on a cloudlet and sending the resulting reduced data (recognized objects) to the cloud. Vigil is an edge-based wireless surveillance system which saves bandwidth by scheduling techniques to support intelligent tracking and surveillance.

In existing work, edge computing used to perform basic processing by rescaling the frame with increasing algorithm complexity to achieve high-performance.

**5.1 DRAWBACKS**

1. Artificial Neural Networks require lots of computational power.

• Neural networks are modeled after the brain and are composed of many interconnected processing nodes. Each node computes based on its weight parameters and adjusts them through backpropagation. Due to many parameters, ANN also needs more extensive datasets for training. ANN requires high computation power for these reasons.

• Besides upgrading your hardware, you could also play around with the hyperparameters of your neural network. This could potentially help you reduce the amount of computational power required.

2. Neural network models are hard to explain.

• It's relatively straightforward to explain traditional machine learning models. For instance, a linear regression model is easy to interpret because it's just a straight line. Coefficients of the line can be interpreted as the relationship between predictor variables and the response variable.

• But, neural networks are much more complex. They are composed of many interconnected processing nodes. It isn't easy to understand how the node weights result in the predicted output.

3. Neural network training requires lots of data.

• Neural networks are very flexible and can learn to recognize input data patterns.

• A simpler machine learning model will perform better than a neural network on small datasets.

4. Data preparation for neural network models needs careful attention

• Data preparation is a crucial step in machine learning. It's essential for neural network models.

• This is because neural networks are susceptible to input data.

5. Optimizing neural network models for production can be challenging.

• You can build neural networks quickly with libraries such as Keras. But, once you've created the model, you need to think about how to deploy it in production.

• This can be challenging because neural networks can be computationally intensive.

**5.2 PROPOSED SYSTEM**

5.2.1 The model running in a local or could machine based on the requirements, that poses higher precision on output.

5.2.2 After basic processing, neurons passed into an empty neuron structure to store and retrieve the data as like humans.

### 5.2.3 ADVANTAGES

#### 1) Store information on the entire network

Just like it happens in traditional programming where information is stored on the network and not on a database. If a few pieces of information disappear from one place, it does not stop the whole network from functioning.

#### 2) THE ABILITY TO WORK WITH INSUFFICIENT KNOWLEDGE:

After the training of ANN, the output produced by the data can be incomplete or insufficient. The importance of that missing information determines the lack of performance.

#### 3) GOOD FAULT TOLERANCE:

The output generation is not affected by the corruption of one or more than one cell of artificial neural network. This makes the networks better at tolerating faults.

#### 4) DISTRIBUTED MEMORY:

For an artificial neural network to become able to learn, it is necessary to outline the examples and to teach it according to the output that is desired by showing those examples to the network. The progress of the network is directly proportional to the instances that are selected.

#### 5) GRADUAL CORRUPTION:

Indeed a network experiences relative degradation and slows over time. But it does not immediately corrode the network.

#### 6) ABILITY TO TRAIN MACHINE:

ANN learn from events and make decisions through commenting on similar events.

#### 7) THE ABILITY OF PARALLEL PROCESSING:

These networks have numerical strength which makes them capable of performing more than one function at a time.

## 6 SUMMARY

Artificial neural networks are inspired from their biological counterparts. Adaptation is one of the most important features of both types of networks. Adaptive artificial neural networks are a class of networks used in dynamic environments. They are characterized by online learning. A number of techniques are used to provide adaptability to neural networks: adaptation by weight modification, by neuronal property modification, and by network structure modification. A brief review of various types of implementations is provided.

## REFERENCE

- [1] Rumelhart, D. E. and J. L. McClelland, 1986, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, MIT Press, Cambridge, MA
- [2] McCulloch, W. S. and W. Pitts, 1943, "A logical calculus of ideas immanent in nervous activity," Bulletin of Mathematical Biophysics, vol. 5 pp. 115-133 Neural Networks, pp. 2476-2481
- [3] Rosenblatt, F., 1958, "The perceptron: a probabilistic model for information storage and organization in the brain," Psychological Review, vol. 65, pp. 386-408
- [4] Selfridge, O. G., 1958, "Pandemonium: a paradigm for learning," Mechanisation of Thought Processes: Proceedings of a Symposium Held at the National Physical Laboratory, London: HMSO, pp. 513-526
- [5] Nguyen, D. and B. Widrow, 1990, "Improving the Learning Speed of 2-layer Neural Networks by Choosing Initial Values of the Adaptive Weights," Proceedings of the IEEE International Joint Conference on Neural Networks, Vol. 3, pp. 21-26
- [6] Minsky, M. and S. Papert, 1969, Perceptrons, MIT Press, Cambridge, MA
- [7] Parker, D., 1985, "Learning Logic," Technical Report TR-87, Center for Computational Research in Economics and Management Science, MIT, Cambridge, MA
- [8] Anderson, J. A. and E. Rosenfeld, 1987, Neurocomputing: Foundations of Research, MIT Press, Cambridge, MA
- [9] Narendra, K. S. and K. Parthasarathy, 1990, "Identification and Control of Dynamical Systems using Neural Networks," IEEE Transactions on Neural Networks, Vol. 1, No. 1, pp. 4-27
- [10] Selinsky, J. W. and A. Guez, 1989, "The Role of A Priori Knowledge of Plant Dynamics in Neurocontroller Design,"

Proceedings of the 28th Conference on Decision and Control, Vol. 2, pp. 1754- 1758

[11] Joerding, W. H. and J. L. Meador, 1991, "Encoding A Priori Information in FeedForward Networks," Neural Networks, Vol. 4, pp.

847-856 Proceedings of the IEEE International Joint Conference on Neural Networks, Vol. 3, pp. 21-26

[12] Nordgren, R. E. and P. H. Meckl, 1993, "An Analytical Comparison of a Neural Network and a Model-Based Adaptive Controller," IEEE Transactions on Neural Networks, Vol. 4, No. 4, pp. 685-694

[13] Miller, W. T., R. S. Sutton, and P. J. Werbos, 1990, Neural Networks for Control, MIT Press, Cambridge, MA

[14] Pao, Y. H., 1989, Adaptive Pattern Recognition and Neural Networks

[15] Brown, R. H., T. L. Ruchti, and X. Feng, 1993, "Artificial Neural Network Identification of Partially Known +Dynamic Nonlinear Systems," Proceedings of the 32nd Conference on Decision and Control, vol. 4, pp. 3694-3699

