Heart Attack Prediction using Neural Network and Different Online Learning Methods

Rayana Khaled Antar, **Shouq Talal ALotaibi**, **Manal AlGhamdi**

Umm Al-Qura University, Department of Computer Science, Alawali, Mecca, 24381, KSA

Summary
Heart Failure represents a critical pathological case that is challenging to predict and discover at an early age, with a notable increase in morbidity and mortality. Machine Learning and Neural Network techniques play a crucial role in predicting heart attacks, diseases and more. These techniques give valuable perspectives for clinicians who may then adjust their diagnosis for each individual patient. This paper evaluated neural network models for heart attacks predictions. Several online learning methods were investigated to automatically and accurately predict heart attacks. The UCI dataset was used in this work to train and evaluate First Order and Second Order Online Learning methods; namely Back-propagation, Delta bar Delta, Levenberg Marquardt and QuickProp learning methods. An optimizer technique was also used to minimize the random noise in the database. A regularization concept was employed to further improve the generalization of the model. Results show that a three layers' NN model with a Backpropagation algorithm and Nadam optimizer achieved a promising accuracy for the heart attach prediction tasks.

Key words:
Neural Network; Heart Attack; online learning; Optimization; Regularization.

1. Introduction

The heart is a very important organ in the human body, and life depends entirely on the effective work of the heart. One of the most dangerous diseases in the world that affects a person's life is heart disease, which is the problem of pushing the required amount of blood to the rest of the body [1]. According to a survey published by the World Health Organization (WHO), there are about 17 million people worldwide who are sick with cardiovascular disease, or 29.20\% of all deaths, most of them are in developing countries [2]. In order to reduce the burden on cardiologists, many researchers applied machine learning methods to solve the problem of prediction with high accuracy with about one-third of these methods were neural networks [3].

Neural Network (NN) is a set of biological neurons that interact with each other, where an artificial neuron is a simulation of how biological neurons interact and process information [4]. NN is a form of predictive analytic methodology that can be used to predict or classify an attribute. It is a machine learning technique, attempts to predict values in the same way as a human brain would.

Manuscript received June 5, 2021
Manuscript revised June 20, 2021
https://doi.org/10.22937/IJCSNS.2021.21.6.12

In this paper, the heart disease prediction problem is addressed using Artificial Neural Network (ANN) algorithms. Using multiple learning algorithms, optimization and regulation methods, the aim is to find the best NN model for predicting heart disease with a good performance.

The remainder of this paper is arranged as follows: Section **2** reviews the efforts made by previous studies to address the problem utilizing NN and deep learning methodologies. Section **3** provides details about the developed models, the most common techniques to improve the models' generalization as well as a brief on the first and second online learning methods. Section **4** describes the experiment setup, results with different activation functions and learning methods and comparison with other techniques. Section **5** concludes the paper by recommending the best NN model to solve the heart attack prediction problem and potential future work.

2. Related Work

Several studies have been done in medical prediction using NN. A number of techniques have been used for the identification of heart diseases including data mining techniques, this section provides an overview of related works.

The multilayer perceptron (MLP) back-propagation algorithm of the NN was used by Durairaj et al. [5] for producing efficient ANN training when combined with optimization techniques such as gradient descent. Their data was from the UCI repository and they calculated the error using MSE. Their model achieved an accuracy of 96.29%. While Awan et al. [6] used the Western Australian patient dataset and modeled an MLP network with a trial-and-error process was used to choose all of the hyper-parameters. A rectified linear activation was chosen for the hidden layer and a sigmoid activation function for the output layer. The MLP-based approach had the best accuracy of 0.64, with 48.42% sensitivity and 70.01% specificity.

Vladimir et al. [7] investigated multiple algorithms on the Cleveland dataset, which has 303 instances and 76 attributes. For training the NN, they generated two arrays with 2D that represent the training set, the second array consists of 30 elements for the test set. The output defined with 2 features to map training set with the target and achieved a recognition rate of 96.67%. Similarly, Takci et al. [8] used various machine learning techniques including MLP architecture on the UCI dataset. In their experiment, they used 90% of the data for the training and 10% for the testing and 10-fold cross-validation. Their model got an accuracy of 83.70%.

Using multiple feature selection along with MLP, Sandeepkumar et al. [9] predict chronic diseases and achieved better performance than support vector machine (SVM) and decision tree. Similarly, Le et al. [10] proposed a genetic-based feature selection in combination with a Naïve Bayes classifier to predict heart disease. The accuracy of their method was 75%. While Panday et al. [11] applied standard and improved MLP algorithms on various medical datasets to develop a decision support system for cardiovascular heart disease diagnosis. Their improved MLP achieved 82.8%, 80.73%, and 93.49% on Cleveland, Hungarian, and Switzerland datasets respectively.

Olaniyi et al. [12] employed both MLP and SVM on medical dataset to define a system that can avoid misdiagnosis in heart diseases. They achieved 85% with MLP and 87.5% with SVM. While Cheng et al. [13] employed the UCI dataset to find patterns using NN, DT, SVM, and Naive Bayes. The proposed hybrid method achieved 86.8% for F-measure, competing with the other existing methods.

3. Methodology

NN is a mathematical or computational model inspired by the functionality of biological neural networks [2]. It contains three layers: the input layer that fed the raw data to the network, the hidden layer that determines the activity of each input unit and the weights assigned to the links between them and the output layer that produces the output.

The MLP is a logistic regression classifier that uses a learned non-linear transformation to transform the input. This transformation converts the input data into a space where it can be separated linearly. MLP is distinguished from a linear perceptron by its multiple layers and non-linear activation [14]. An activation function specifies how the input weight is summed to converted it into the output from a node in a layer. In different parts of the model, different activation functions can be used [4,15]. For instance, the hidden layer can employ the rectified linear activation (ReLU), the logistic (or sigmoid) or the hyperbolic tangent (Tanh) function. While the output layer can use the linear, the sigmoid or the SoftMax function.

In addition, in order to modify the weights and the learning rate of a NN model aiming to minimize error and speed up the performance, an optimizer is used to study the behavior of different algorithms for optimizing gradients. Some of the most common optimizers are Adaptive Moment Estimation (Adam) and Stochastic Gradient Descent SGD, which improve the NN model performance by reducing the overfitting [4].

1. Adam optimizer is an adaptive learning rate optimization algorithm that works by adjusting the step size for each parameter based on previous results. This optimizer preserves its own set of fixed hyperparameters including: the learning rate α and the two moment coefficients ß1, ß2 [16].

2. SGD where the term "stochastic" refers to a mechanism or procedure that is subject to random probability. For each iteration of SGD, a few samples are chosen at random rather than the entire sample [16]. SGD algorithm updates the weights as follow:

   $$\emptyset_j = \emptyset_j - \alpha(\hat{y}^i - y^i)x_j^i \quad (1)$$

   It finds the gradient of a single sample for the cost function instead of calculating the sum of the gradients of all [17].

Moreover, two important hyperparameters for the learning algorithm are: epoch and batch size. The Epoch specifies how long the learning algorithm will take to run through the entire training set [4]. While the Batch size refers to the total number of samples from a dataset that are used to determine the gradient for each iteration [4]. Batches are often used to group data sets. Several epochs are used in the creation of several models, where the dataset size is *d*, the number of epochs is *e*, the number of iterations is *I* and the batch size is *b*, the general relationship is $d * e = I * b$ [17].

3.1 First Order Method

In this paper, two different learning algorithms from first order are used which are Backpropagation and Adaptive Learning Rate algorithms.

3.1.1 Back-Propagation (BP)

In NN, the BP algorithm is still the approach of preference for training large networks. BP is a supervised learning method used by MLP for training. It is a gradient descent algorithm that aims to minimize the errors between the network's output and the desired outcome [14]. The most widely used training algorithm, back-propagation, is a gradient descent technique that efficiently computes the derivatives' values and modifies the weights according to a parameter known as the learning rate [14].

In the gradient descent algorithm, the weight link between the hidden layer's j neuron and the input layer's i neuron is modified according to [14]:

$$w_{ji}(t) = w_{ji}(t-1) + \Delta w_{ji}(t) \qquad b_j(t) = b_j(t) + \Delta b_j(t) \tag{2}$$

For updating the weight, $\boldsymbol{\Delta w_{ji}(t)}$ ***and*** $\boldsymbol{\Delta b_j(t)}$ given by:

$$\Delta w_{ji}(t) = \eta w P_j(t)\, x_i(t) + \alpha_w \Delta w_{ji}(t) \qquad \Delta b_j(t) = \eta b P_j(t) x_i(t) + \alpha_w \Delta b_j(t) \tag{3}$$

where w represents the weight and b represent the threshold, the momentum constants that determine the influence of the previous parameter change on the current direction movement in the space are the [Δw] and [Δb]. The learning rates represent in [ηb] and [ηw]. The [P_j(t)] is the error signal in the hidden layer's j neuron that signal is propagate backwards through the network [14].

Because the output neuron's activation function is linear, the error signal at the output node is:

$$P_j(t) = y_k(t) - \hat{y}_k(t) \tag{4}$$

where the expected output is the [y_k (t)], for the neuron in the hidden layer [14].

$$P_j(t) = F''(x_i(t)) \sum P_j^k(t) w_{jk}^2(t-1) \tag{5}$$

where the [$F''(x_i(t))$] is the first derivative with respect to $x_i(t)$.

3.1.2 Delta-Bar-Delta Learning (Adaptive Learning Rate) or TurboProp

The BP algorithm is considered as a gradient decent type of algorithm. Thus, the algorithm has a problem of a slow convergence rate. The search for the global minima may become stucked at local minima. Also, the algorithm is sensitive to the user selectable parameters. In TurboProp every weight is adjusted with each iteration. Each weight can have its own learning rate [4].

1- The learning rate is improved if the direction in which the error declines at the current stage, as shown by the error gradient, is the same as the direction in which the error has been decreasing recently [4].

2- Otherwise, the learning rate is reduced if the current direction in which the error is decreasing is opposite to the previous direction in which the error has been decreasing [4].

Up to epoch m, the direction recent history in which the error has been decreasing is defined as:

$$f_m = \emptyset f_{m-1} + (1-\emptyset)\, d_{m-1} \tag{6}$$

Thus, f_m is the average direction history and d_m is the the total gradient form epoch m, can be calculated as:

$$d_m = \sum_{n-1}^{N} [\frac{\partial E}{\partial w_m}]_n \tag{7}$$

- **With Momentum**: The fundamental concept behind momentum in ML is to speed up the training process. It's often used in NNs as the size of data causes a significant time delay when training gradients. using momentum can handle noisy gradients and even extremely tiny gradients [18].

$$\Delta w = \mu \Delta w_{m-1} - (1-\mu)\, \varepsilon_m d_m \tag{8}$$

where the Momentum part is in $\mu \Delta w_{m-1}$ [4].

3.2 Second Order Method

In this paper, two learning algorithms from second order are used which are Levenberg Marquardt and Quickprop algorithms.

3.2.1 Levenberg Marquardt algorithm

The Levenberg–Marquardt algorithm (LMA) is used to solve non-linear least squares problems. It depends on how the error function resembles a quadratic function. If the quadratic approximation is not suitable, the algorithm may diverge.

The Formula for LMA is shown (e.g., see Eq. 9).

$$f(x) = \frac{1}{2} \sum_{i=1}^{m} [\, f_i(x)\,]^2 \tag{9}$$

Let the Jacobian of $f_i(x)$ be denoted $J_i(x)$, then the LMA searches in the direction given by the solution p to the equations.

$$(J_k^T J_k + \lambda_k I)\, P_k = -J_k^T f_k \tag{10}$$

where λ_k are non-negative scalars and I is the identity matrix. The method has the nice property that, for some scalar Delta related to λ_k, the vector P_k is the solution of the constrained sub-problem of minimizing.

$$\frac{||\, J_k P + f_k ||_2^2}{2} \tag{11}$$

Subject to $||p||_2$.

3.2.2 Quickprop algorithm

Quickprop is an iterative method for defining the minimum of the loss function of NN. It uses second order learning methods. The formula for updating the weights for the layer is given by (e.g., see Eq. 12).

$$\Delta^k w_i j = \Delta^{(k-1)} w_i j \left(\frac{\Delta_i j\, E^{(k)}}{\Delta_i j\, E^{(k-1)} - \Delta_i j\, E^{(k)}}\right) \tag{12}$$

Being $w_i j$ the neuron j weight of its i input and E is the loss function [4].

To further enhancement our models a regularization concept is applied. Avoiding over fitting is one of the most important aspects of training ML algorithms. If the model is over fitting, it will have a poor accuracy. This occurs because the model is attempting to capture the noise in the training dataset [4]. By noise, we mean data points that are not necessarily representative of the data's true properties but are just random chance. Learning such data points makes the model more adaptable, but it also increases the chance of over-fitting [20].

Regularization: Regularization is a method for improving the generalization of a learning algorithm by making small modifications to it [21]. Regularization methods that are widely used are:

1- **L1 Regularization**: LASSO (Least Absolute Shrinkage and Selection Operator) is a regression model that implements the L1technique. The "absolute value of magnitude" of the coefficient is applied as a compensation parameter to the loss function in Lasso Regression (L) [21].

$$||w||_1 = |w_1| + |w_2| \ldots + |w_N| \quad (13)$$

2- **L2 Regularization**: Where in Ridge regression the model that employs the L2 technique. In Ridge regression adds "squared magnitude" of coefficient as a parameter to the loss function(L) [21].

$$||w||_2 = (|w_1|^2 + |w_2|^2 \ldots + |w_N|^2)^{\frac{1}{2}} \quad (14)$$

The output function y does not change during the regularization, only the loss function is changed. The following is the output function [21]:

$$\hat{y} = w_1x_1 + w_2x_2 + .. + w_Nx_N + b \quad (15)$$

The loss function after L2 regularizatio:

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{n} |w_i| \quad (16)$$

where λ is a hyper-parameter Known as regularization constant and it is greater than zero [21].

3- Dropout Regularization: A regularization approach for approximates large number of NNs in the training stage, with various architectures in parallel. During training, certain layer outputs are dropped off randomly. This forces the layer to appear and act as a different number of nodes from the previous layer. Each update to a layer is done separately [22, 23].

In addition, there are also other strategies for improving model efficiency [4, 24], including: weight decay, early stopping and cross validation.

4. Experimental Work

In this section, framework setup is detailed as well as the UCI data overview, which includes information on each attribute, feature correlations and data pre-processing. Creating the MLP models with various layers. In addition, the model will be improved and detailed comparisons with other methods will be made. Experiments of various activation functions and online learning are shown.

4.1 Experimental Setup

Jupyter Notebook is a non-profit open-source software that has grown to support collaborative data science. The Python language was used to manage and execute the models using Jupyter [25]. The pandas and NumPy licensed libraries offer high-performance, easy-to-use data structures and data processing resources for the Python programming language. A panda's module was used for data processing, and NumPy was used to convert string values to numerical values [26] [27]. In addition, the scikit-learn, which is a Python-based ML library, was widely used to split the dataset into a training and a testing set for this library [28]. This is necessary so that a part of the patient's data can be used to train the models and the rest can be used to test their efficacy. As a result, the data was split into; 78% for training and 22% for testing.

For designing MLP model, Keras sequential layer was used to create the various NNs with different hidden layers. Keras is a Python-based NN interface that represents DL systems at a high level of abstraction. It is built on top of TensorFlow. The computing arrays are performed by TensorFlow, which is an open-source ML software library [22, 29].

4.2 Data Description

The dataset provided by "**Cleveland Heart Disease dataset**", which is publicly available online at the University of California Irvine **UCI** data mining repository. The Cleveland database, in fact, is the only one used so far by ML researchers. The data-set can be downloaded on the follow link: www.kaggle.com. The database contains a sample size of 303 patients and 14 features as shown in Table 1.

Table 1: Heart Data Information

No	*Column*	*Non-Null Column*	*Data type*
0	Age	303 non-null	Int64
1	Sex	303 non-null	Int64
2	cp	303 non-null	Int64
3	Trestbps	303 non-null	Int64
4	Chol	303 non-null	Int64

5	Fbs	303 non-null	Int64
6	Restecg	303 non-null	Int64
7	thalach	303 non-null	Int64
8	exang	303 non-null	Int64
9	oldpeak	303 non-null	float64
10	slop	303 non-null	Int64
11	ca	303 non-null	Int64
12	thal	303 non-null	Int64
13	target	303 non-null	Int64

From Table 1 there are no missing values. The essential point is to obtain a representation of whether the individual has heart attack or not. This provides data that is a 13 x 297 feature matrix [30].

The attributes consist of the following information:

1- age: age in years.

2- sex: sex (1 = male; 0 = female).

3- CP: Chest Pain Type contains (4 Values).

- Value 0: typical angina - Value 1: atypical angina - Value 2: non-anginal pain- Value 3: asymptomatic.

4- trestbps: resting blood pressure (in mm Hg on admission to the hospital).

5- chol: serum cholesterol in mg/dl.

6- (fbs): (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false).

7- restecg: Resting Electrocardiographic results contains (values 0,1,2).

- Value 0: normal - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)- Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria 20 ekgmo (month of exercise ECG reading).

8- thalach: maximum heart rate achieved.

9- exang: exercise induced angina, represent (1 = yes; 0 = no).

10- oldpeak: ST depression induced by exercise relative to rest.

11- slope: the slope of the peak exercise ST segment.

- Value 1: upsloping - Value 2: flat - Value 3: downsloping.3: asymptomatic.

12- ca: number of major vessels (0-3) colored by fluoroscopy.

13- thal: 3 = normal; 6 = fixed defect; 7 = reversable defect [30].

Table 2 depicts a sample of the last 8 rows and columns that display the listed data value.

Table 2: Sample of UCI dataset.

	age	*sex*	*cp*	*trestbps*	*Chol*	*fbs*	*restecg*	*thalach*	*exang*	*oldpeak*	*slop*	*ca*	*thal*	*target*
295	63	1	0	140	187	0	0	144	1	4.0	2	2	3	0
296	63	0	0	124	197	0	1	163	1	0.0	1	0	2	0
297	59	1	0	164	176	1	0	90	0	1.0	1	2	1	0
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

4.3 Data Visualization and Analysis

From the histogram in Fig.1, the features: target, slope, thal, sex, fbs, exang, cp and ca are represented with discrete bars indicating that they are discrete variables, which represent categorical variables. We will need to process these variables before applying the algorithms. Our target labels have two classes, 0 for no disease and 1 for disease. From the histogram, each feature has a different range of distribution. Thus, using scaling before making the prediction is important to make the target classes equal in size.

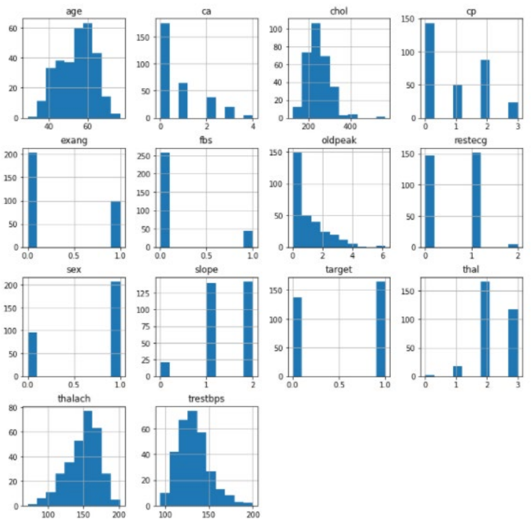

Fig. 1: Histograms of UCI dataset.

To figure out what age has a high risk of heart attacks, Fig.2 shows the bar plot where people at age 41-51 and 51-59 have the highest risk of heart attacks.

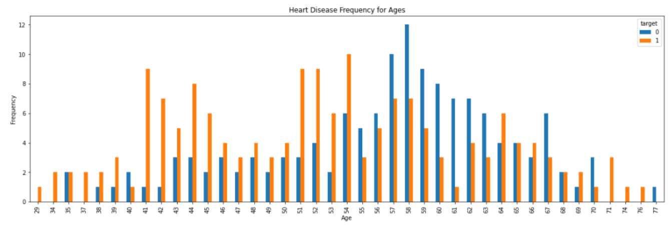

Fig. 2: Heart Disease Frequency for Ages, x-axes represent the frequency form 0 to 12 and the y-axes represent ages from 29 to 77. Ages between 41-59 have the highest frequencies of heart attack.

To understand the correlation between features which is useful for the model feature selection, a heat-map shown in Fig.3 indicating that there is no tight correction between the features.

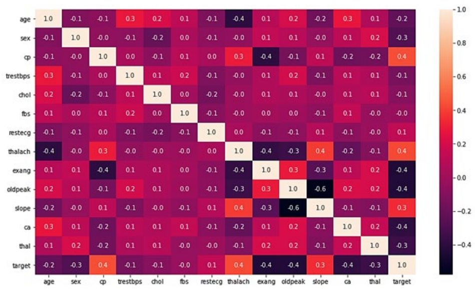

Fig. 3: Heat Map for correction between features.

Fig.4 described the number of patients with chest pain and high blood-sugar for fasting patient.

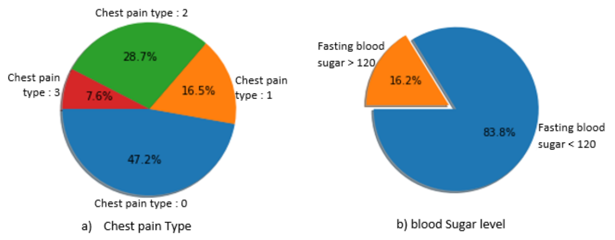

Fig. 4: Subplot for a) Chest Pain types and b) blood sugar level.

Fig.5 show the number of data in each class, indicating that the two classes are balanced (not exactly 50 percent) but the ratio is good enough to continue without dropping or increasing the data.

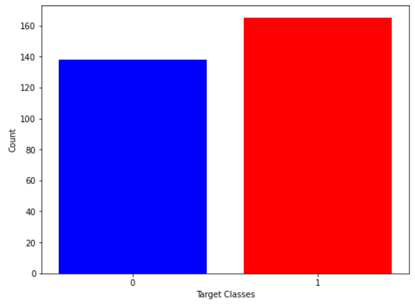

Fig. 5: Target classes count where 0 represent the absent and 1 represent the present of heart attack.

4.4 Data Pre-Processing

A strategy for testing the output of a ML algorithm is the train-test split. It can be used with any supervised learning algorithm and can be used for classification or regression problems [31]. The training part is for fitting the ML model, while the testing is for evaluating the model in order to fit the ML model [31].

There is a total of 13 features that are correlated to each patient. In this dataset, there are no needless attributes to be deleted, nor any null values to be removed. All the attribute types are considered as numeric. Then, we scaled the dataset to keep the computations more effective by using the standard scaler method to ensure that all values have a mean of zero and a unit variable. After pre-processed the data, the data were splitted into 80% training and 20% testing.

4.5 Proposed Models

The models were constructed as sequential. Then, different numbers of fully connected layers were added as shown in **Table 3** for the first order online learning methods and **Table 4** for the second order online learning methods. The inputs dimension for all models are set to 13 which corresponds to the 13 columns attributes.

Table 3: First order online learning model structure

Input No	*Hidden layer No*	*Hidden activation function*	*Neuron no*	*Output no*	*output activation function*
13	2	ReLU	6, 3 neurons	1	Sigmoid
13	3	ReLU	6, 3,3 neurons	1	Sigmoid
13	4	Tanh	26,13,6,3neurons	1	Sigmoid
13	3	ReLU	6, 3, 3 neurons	1	Sigmoid
13	3	ReLU	6, 3, 3 neurons	1	Sigmoid

Table 4: Second order online learning model structure.

Input No	*Hidden layer No*	*Hidden activation function*	*Neuron no*	*Output no*	*output activation function*
13	2	Sigmoid	100, 20 neurons	1	Sigmoid
13	2	Sigmoid	200, 20 neurons	1	Sigmoid

4.5.1 Training the First Order Method

First Model: The Model has 4 fully connected Dense layers, 3 hidden and one output layer. The first level has dimension of 13 which corresponds to 13 columns attributes, the second layer has 6 neurons, the third layer has 3 neurons, and a ReLU is used as the activation function for all hidden layers. The output layer has a single neuron and the sigmoid activation function suited for binary classification problems. The model was trained the using Mean Square Error. This involves forward propagating the inputs, back-propagating the error, and modifying the network weights for each row of tests. The learning rate parameter was set at 0.0001. This model has an accuracy 76.12% with loss 0.1753 as represented in Fig. 6. The Adam optimizer was used. The data were tested with 16 mini-batch gradient descent over 1000 epochs.

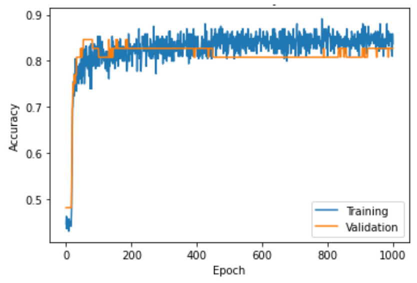

Fig. 6: First Model Accuracy

Second Model: Contains 5 fully connected Dense layers, The input layer with 13 attributes, 6 neurons in the second layer, 3 neurons in the third layer, also 3 neurons in the fourth layer and a single neuron for the output layer. All the layers used ReLU as the activation function, except the output layer where the sigmoid activation function suited for binary classification problems. The second model has an accuracy 85.07% with loss 0.1408. An optimizer has been used "Adam". The data were tested with 20 mini-batch gradient descent over 1000 epochs. We can observe that the accuracy in the test data has been improved when setting the learning rate at 0.001, adding more hidden layer and increasing the test data to 30%.

Third Model: The model architecture made up of 6 layers. The first level has dimension of 13 which corresponds to 13 columns attributes and use the ReLU activation function. The second layer has 26 neurons, the third layer has 13 neurons, the fourth layer has 6 neurons and the fifth layer has 3 neurons. All these layers use the Tanh as an activation function. The last layer is the output layer with a single neuron and a sigmoid activation function for binary classification.

As shown in Fig.7, the model achieved an accuracy of 80.60% with loss 0.1791. The Adam optimizer was used. We can observe that the accuracy in this model decreased when we changed the activation function from ReLU (in the first model) to Tanh (in this model).

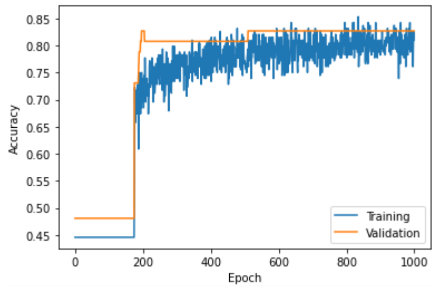

Fig. 7: Third Model Accuracy.

Fourth Model: The model consists of 5 layers as viewed in Table 5, with the same architecture of the second model. The Fourth model has an accuracy 86.89%, using Delta rule for updating weight with 0.01 learning rate and epsilon=1e-07. we can observe that the accuracy in the test data is much better among other models.

Fifth Model: Has the structure of the Fourth model with accuracy at 85.25% and loss 0.1709, using Delta rule for updating weight with 0.01 learning rate. The SGD optimizer used with momentum and decay values 0.9 and 0.01, respectively.

The models were trained the using Mean Square Error. This involves repeatedly presenting a training dataset to the network, forward propagating the inputs, back propagating the error, and updating the network weights for each row of tests. This operation can be divided into two parts:

1- Update Weights:

- **With backpropagation**: After calculated each neuron error using the above method "backpropagation", we can use it to modify the weights. Network weights are modified as follows:

  weight = weight + learning rate * error * input

  The learning rate is a parameter that we must defined, error is the error defined by the BP procedure for the neuron and input is the input value that produced the error. The learning rate was set at 0.0001.

- **With Delta rule**: updating of associated weights according to Delta rule:

$$\Delta weights_{ij} = -\eta \frac{-\eta \partial E\ weights[ij]}{\partial weights[ij]} \qquad (18)$$

Apply the weight update to each weight w_{ij} for each training pattern. The learning rate was set at 0.001.

2- Calculate the error:

- **Mean Square Error MSE:** the error was calculated using the following:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad (16)$$

Where n is the number of data point, y_i is the obtained value and $\hat{y}_i$ is the predicted values.

Train Network: After updating the network. Using MSE, looping for a fixed number of epochs and for each epoch updating the network for each row in the train set. This type of learning is known as the online learning. If errors were accumulated across an epoch before updating the weights, this is called batch learning or batch gradient descent. The number of epochs was set to 1000 and 16 batch size.

The five constructed NN models with different layers, activation functions, achieved results are shown in **Table 5**.

Table 5: Backpropagation and Delta Models Results.

Input No	***First layer***	***Second layer***	***Third layer***	***Fourth layer / Fifth layer***	***Loss function***	***Optimizer / Lrate***	***Epochs no / Patch size***	***Accuracy***
Backpropagation Method								
Model 1								
13	6 ReLU	3 ReLU	1 sigmoid	– / –	MSE	Adam / 0.0001	1000 / 16	76.12%
Model 2								
13	6 ReLU	3 ReLU	3 ReLU	1 sigmoid / –	MSE	Adam / 0.001	1000 / 20	**85.07%**
Model 3								
13	26 Tanh	13 Tanh	6 Tanh	3 Tanh / 1 sigmoid	MSE	Adam / 0.001	1000 / 20	80.60%
Delta Bar Delta Method								
Model 4								
13	6 ReLU	3 ReLU	3 ReLU	1 sigmoid / –	binary cross entropy	Adadelta / 0.01	100 / 6	**86.89%**
Model 5								
13	6 ReLU	3 ReLU	3 ReLU	1 sigmoid / –	MSE	SGD 0.01 momentum = 0.9 decay = 0.01	1000 / 16	85.25%

4.5.1 Training the Second Order Method

First Model: contains 4 fully connected layers. The input layer with 13 attributes, 100 neurons in the second layer, 20 neurons in the third layer and a single neuron for the output layer. All the layers used sigmoid as the activation function. The data were tested with 20 mini-batch gradient descent over 1000 epochs and the learning rate set at 0.01. The QuickProp validation accuracy: 52.27%.

Second Model: also contains 4 fully connected layers. The input layer with 13 attributes, 200 neurons in the second layer, 20 neurons in the third layer and a single neuron for the output layer. All the layers used sigmoid as the activation function. The data were tested with 20 mini-batch gradient descent over 1000 epochs and the learning rate set at 0.1. LMA accuracy is: 50.00%.

After applying LMA and QuickProp, the validation accuracies of both methods are shown in **Table 6**.

Table 6: Levenberg Marquardt and QuickProp Models Results.

Input No	***First layer***	***Second layer***	***Loss function***	***Algorithm***	***Accuracy***
Model 1 - QuickProb					
13	100 Sigmoid	10 Sigmoid	Binary cross entropy	QuickProp	52.27%
Model 2 - Levenberg Marquardt					
13	200 Sigmoid	20 Sigmoid	MSE	LMA	50.00%

4.6 Further Enhancement to the Models

Deep Learning requires a lot of hyperparameter optimization. NNs are extremely complex to build, with numerous of parameters to configure. Furthermore, individual models will take a long time to train.

4.6.1 First Order Method

The defined MLP model that expects 13 inputs variables, has four hidden layers with 32, 13, 6, and 3 nodes respectively, and an output layer with one node to estimate the probability of each class. Nodes in the hidden layer will employ the ReLU, while nodes in the output layer will employ the sigmoid activation function. Different optimizer has been employed such as (Adam, RMSprop, adadelta ..etc), where the learning rate needs to be defined for evaluating different rates values. The model will be trained to reduce the cross-entropy error with 4000 epochs over 62 batch size. The test set will be used as to evaluate the generalization of the model. Further, the following evaluations will be conducted:

- **First**, checking the effect of multiple learning rates on the accuracies of both training and testing sets. Fig.8 represents line plots for each learning rate including 8-line plots for the eight different learning rates that have been tested. The training dataset's classification accuracy is shown in blue, while the test dataset's accuracy is shown in orange.

  By comparing the average results, the plots show oscillations in behavior for the high learning rate of 1.0 as in Fig. 8 a) and the model failure to learn anything with the low learning rates of 1E-6 and 1E-7 as in Fig. 8 g) and h). While in Fig. 8 b), c) and d) the model was able to learn the problem well. The results indicate that a modest learning rate of 0.001 results in a good accuracy using both train and test sets for the selected model configuration.

- **Second**, the effect of various "patience" values that represent the number of epochs to wait for a transition prior to reduce the learning rate, is being examined. The default learning rate of 0.01 will be used, and the learning rate will be dropped. According to theses graphs in **Fig. 9**, the patience value of 200 for this model on this issue should result in improved results because it allows the higher learning rate to be used for a longer period of time before the rate is lowered to optimize the weights.

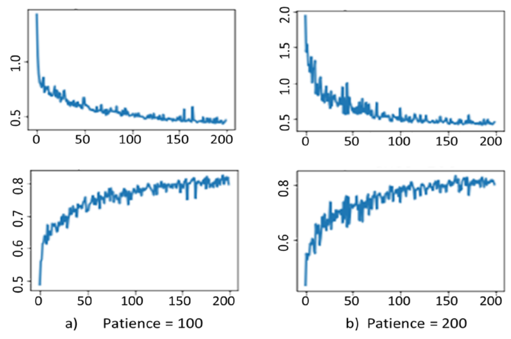

Fig. 8: Training Accuracy Over Epochs for Different Patience Values. a) patience value of 100 result in higher learning rate to be used for a longer period before the rate is lowered. b) patience value of 200 result in lower learning rate to be used for a shorter period.

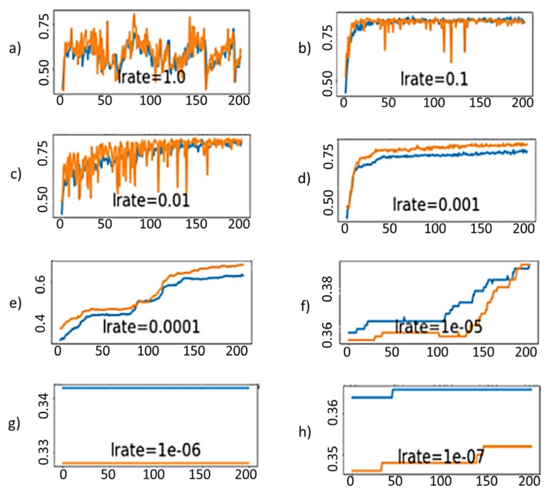

Fig. 9: the effect of different Learning Rates. a) too large lrate result in failure to learn, b) lrate 0.1 result in ability to learn, c) lrate 0.01 has showed good result, d) lrate 0.001 results in strong model performance on train and test set, e) lrate 0.0001 poor performance, g) and h) too small lrate result in inability to learn.

- **Third**, the effect of Adaptive Learning Rates, the dynamics of different adaptive learning rate methods. By explore the most popular methods of Nesterov-accelerated Adaptive Moment Estimation (Nadam), Root Mean Square Propagation (RMSprop), adaptive gradient algorithm (AdaGrad) and Adam and compare their behavior with a static learning rate. The Fig. 10 shows four-line plots for the multiple optimization algorithms that have been tested. The training dataset's classification accuracy is shown in blue, while the test dataset's accuracy is shown in orange.

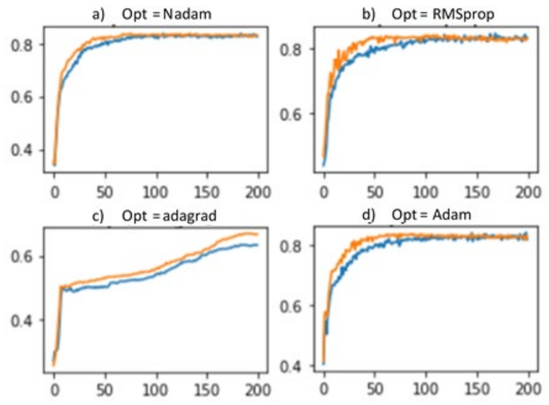

Fig. 10: Train and Test accuracy Over Multiple optimizations. a) Nadam optimizer. b) RMSprop Optimizer and d) Adam optimizer demonstrate similar performance but a) Namdam showed best result. d) adagrad takes all 200 epochs and result in unstable accuracy.

Observation showed that AdaGrad in Fig. 10 c) with a learning rate of 0.001 and epsilon=1e-07 does learn the problem, but it takes roughly about 200 epochs and produces unstable accuracy for both training and testing sets. Whereas Nadam in Fig. 10 a), RMSprop in Fig. 10 b), and Adam in Fig. 10 d) showed comparable results, learning the problem in less than 50 training epochs and using the rest of the time performing small weight changes.

- **Lastly**, NN requires a lot of hyperparameter optimization. By trying different values of the regularization parameter, weight regularization can help to improve the overfit model. Initially grid search through certain orders of magnitude between 0.0 and 0.1, then grid search on that level until found it. By specifying the values to measure, looping through each, and saving the results from train and test sets, we can grid search through the orders of magnitude.

  Fig. 11 shows a line plot of the effects, demonstrating that greater weight regularization parameter values improve test accuracy. It is obvious that using the peak value of 0.1 produces a major decrease in both training and testing accuracies.

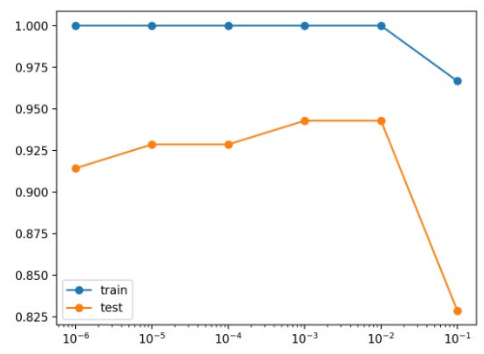

Fig. 11: Accuracy on Train and Test with different weight regularization parameters.

4.6.2 Second Order Method

There are various factors that are limiting the accuracy of the built model. First is the small size of the dataset, second is the absence of optimizers in second-order methods compared to first-order methods, third is the time taken by the model to train for second-order methods increases exponentially as the model depth is increased. Thus, after trying various combinations of the network and hyperparameters, therefore, the model's performance was not able to be improved.

4.7 Comparison with Related Work

Extensive comparison with other techniques of the same methods (See Table 7) shows the results. Durairaj et al. [5] and Vladimir et al. [7] outperform others due to determine the proper parameters setting for the MLP algorithm in order to predict heart disease with higher accuracy.

The **Table 6** shows a comparison of QuickProp and LMA with accuracy around 52.27% and 50% respectively. According to the loss function has limitation since it can only use the MSE loss function, which reduces its accuracy, due to the limit number of records. Usually, in second order methods understanding how the gradient is normally measured and added to network parameters to attempt to iteratively achieve convergence of the loss to a global minimum should be considered.

Table 7: Comparison with Related Work.

Authors	*Techniques*	*Dataset*	*Accuracy*
Durairaj et al. [5]	MLP – Backpropagation	UCI	**96.29%**
Awan et al. [6]	MLP	Western Australian data system	64.93%
Vladimir et al. [7]	MLP	Cleveland	**96.67%**
Tackci et al. [8]	MLP	UCI	83.70%
Sandeepk umaret al. [9]	Feature selection, SVM, Decision Tree and MLP	UCI	80.1%
Le et al. [10]	Genetic feature selection Naive based classifier	UCI	75.0%
Pandy et al. [11]	Applied ordinary improved MLP	Cleveland	93.49%
		Hungarian	80.73%
		Switzerland	82.8%
Olaniyi et al. [12]	MLP	UCI	85.0%
	SVM		87.5%
Cheng et al. [13]	NN, DT, SVM and naive Bayes	UCI	86.8%

In order to fully confirm good performances that MLP has achieved, we have also applied activation function for hidden and output layers, the MLP details and results are shown in **Table 8**, where the models were tested with different optimizer, learning rate, loss function and regulizers.

Table 8: Comparison with Other First order methods.

Input No	***Hidden layer No***	***Hidden activation function***	***Output No***	***Output activation function***	***Loss function***	***Optimizer / Lrate***	***Regulizer***	***Accuracy***
13	3	ReLU	1	sigmoid	binary cross entropy	Adam / 0.001	0.000001	**97.8%**

13	3	ReLU	1	sigmoid	binary cross entropy	RMS prop 0.001	0.000001	95.60%
13	4	Tanh	1	sigmoid	binary cross entropy	Adagrad 0.01	-	86.89%
13	3	ReLU	1	sigmoid	binary cross entropy	Adadelta 0.001	0.000001	56.89%
13	3	ReLU	1	sigmoid	binary cross entropy	Adamax 0.001	0.000001	85.20%
13	3	ReLU	1	sigmoid	binary cross entropy	Nadam 0.001	0.000001	**99.3%**
13	4	ReLU	1	sigmoid	Mean Square Error	SGD 0.01	0.000001	85.25%

As a result, the NADAM optimizer updating the parameters with the momentum step before calculating the gradient, we can perform a more effective step in the gradient direction. NADAM optimizer had an accuracy about 99.30%.

5. Conclusion and Discussion

In this paper, we addressed the problem of heart disease diagnosis using NN. We employed the UCI dataset, which has 303 instances and 14 attributes to test multiple NN models.

Our experiments show that the best NN architecture for prediction heart attack consists of three hidden layers with 13 features as an input and one output layer. The BP algorithms was used as online learning method with Nadam optimizer. The best learning rate was 0.001 with a binary cross entropy loss function for testing/validation among 62 patch size and over 4000 epochs. Best activation function for all layers was ReLU for hidden layers and sigmoid for output layer. Further experiments with more advanced techniques is part of the authors' ongoing research.

References

[1] Haq, A.U., Li, J.P., Memon, M.H., Nazir, S., Sun, R.. A hybrid intelligent system framework for the prediction of heart disease using machine learning algorithms. *Mobile Information Systems*2018;2018.

[2] Malav, A., Kadam, K., Kamat, P.. Prediction of heart disease using k-means and artificial neural network as hybrid approach to improve accuracy. *International Journal of Engineering and Technology*2017;9(4):3081–3085.

[3] Peng, C.C., Huang, C.W., Lai, Y.C.. Heart disease prediction using artificial neural networks: A survey. In:2020 *IEEE 2nd Eurasia Conference on Biomedical Engineering, Healthcare and Sustainability (ECBIOS)*. IEEE; 2020, p. 147–150.

[4] Samarasinghe, S.. *Neural networks for applied sciences and engineering: from fundamentals to complex pattern recognition*. Crc Press; 2016.

[5] Durairaj, M., Revathi, V. Prediction of heart disease using back propagation mlp algorithm. *International Journal of Scientific & Technology Research* 2015;4(8):235–239.

[6] Awan, S.E., Bennamoun, M., Sohel, F., Sanfilippo, F.M., Dwivedi, G.. Machine learning-based prediction of heart failure readmission or death: implications of choosing the right model and the right metrics. *ESC heart failure* 2019;6(2):428–435.

[7] Mati ́c, V., et al. Effective diagnosis of heart disease presence using artificial neural networks. In: *Sinteza 2017- International Scientific Conference on Information Technology and Data Related Research*. Singidunum University; 2017, p. 3–8.8.

[8] Takci, H.. Improvement of heart attack prediction by the feature selection methods. *Turkish Journal of Electrical Engineering & Computer Sciences* 2018;26(1):1–10.

[9] Hegde, S., Hedge, R.. Symmetry based feature selection with multilayer perceptron for the prediction of chronic disease. *International Journal of Recent Technology and Engineering* 2019;8(2):3316–3322.10.

[10] Le, H.M., Tran, T.D., Van Tran, L.. Automatic heart disease prediction using feature selection and data mining technique. *Journal of Computer Science and Cybernetics* 2018;34(1):33–48.

[11] Panday, P., Godara, N.. Decision support system for cardiovascular heart disease diagnosis using improved multilayer perceptron. *International Journal of Computer Applications* 2012;45(8).12.

[12] Olaniyi, E.O., Oyedotun, O.K., Adnan, K.. Heart diseases diagnosis using neural networks arbitration. *International Journal of Intelligent Systems and Applications* 2015;7(12):72.

[13] Cheng, C.A., Chiu, H.W.. An artificial neural network model for the evaluation of carotid artery stenting prognosis using a national-wide database. In: *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE; 2017, p.2566–2569.14.

[14] Adnan, J., Daud, N.N., Ahmad, S., Mat, M., Ishak, M., Hashim, F., et al. Heart abnormality activity detection using multilayer perceptron (mlp) network. In: *AIP Conference Proceedings*; vol. 2016. AIP Publishing LLC; 2018, p. 020013.15.

[15] Yosi Taguri, S.E., Lussato, R.. 7 types of neural network activation functions: How to choose? Available at https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/ (2016).

[16] Chandra, K., Meijer, E., Andow, S., Arroyo-Fang, E., Dea, I., George, J., et al. Gradient descent: The ultimate optimizer. *arXiv preprintarXiv:190913371* 2019.

[17] geeksforgeeks.org. Ml —stochastic gradient descent (sgd). Available at https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/ (2020/05/16).

[18] Mahapatra, A.. Momentum in machine learning by medium. Available at https://medium.com (2019/06/14).

[19] Xiao, T., Zhang, L., Ma, S.. System Simulation and Scientific Computing*: International Conference, ICSC 2012, Shanghai, China, October27-30, 2012. Proceedings, Part I*; vol. 326. Springer; 2012.20.

[20] Farahmand, A.m.. Regularization in reinforcement learning 2011.
[21] AlindGupta, g.. Regularization in machine learning. Available at https://www.geeksforgeeks.org/regularization-in-machine-learning/amp/ (2020/08/21).
[22] Aakash Nain Sayak Paul, M.M.R.. Keras. Available at https://keras.io/about// (2020/08/26).
[23] Brownlee, J.. A gentle introduction to dropout for regularizing deep neural networks. Available at https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/ (2019/08/06).
[24] Gupta,P..Cross-validation in machine learning. Available at https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f (2020/08/21).
[25] Kluyver, T., Ragan-Kelley, B., Perez, F., Granger, B., Bussonnier, M., Frederic, J., et al. Jupyter notebooks – a publishing format for reproducible computational workflows. In: Loizides, F., Schmidt, B., editors. *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. IOS Press; 2016, p. 87 – 90.26.
[26] Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., et al. Array programming with NumPy. *Nature* 2020;**585**(7825):357–362. doi: \bibinfo{doi}{10.1038/s41586-020-2649-2}. URL https://doi.org/10.1038/s41586-020-2649-2.
[27] pandas development team, T.. pandas-dev/pandas : Pandas. 2020. doi:\bibinfo{doi}{10.5281/zenodo.3509134}. URL https://doi.org/10.5281/zenodo.3509134.28.
[28] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., et al. Api design for machine learning software: experiences from the scikit-learn project. *ArXiv preprint arXiv* :13090238 2013.
[29] Chollet, F., et al. Keras. https://github.com/fchollet/keras; 2015.
[30] Inc., K.. Heart disease uci. 2019.
[31] Brownlee, J..Train-test split for evaluating machine learning algorithms. Available at https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/ (2020/08/26).

Rayana Elmustapha Antar received the B.S. degree in Information Technology and Computing from Arab Open University in 2018 with first class honor. In 2007, she trained as an Executive Secretary at Finance department in Dr. Abdulrahman Taha Bakhsh Hospital. In 2010, she trained as a system analyst in Global Tracks Company. Since 2018 she has been working as an IT Manager in Advanced Smart Integration Company, Jeddah, KSA.

Shouq Talal ALotaibi obtained a BA in Computer Science from Umm AlQura University in 2019 with first class honors. At the end of 2019, she trained in Egypt at an application company called PITECHNOLOGY. She volunteered at the Rafa Charity Association for 3 months, Mecca, KSA.

Manal Algamdi received her Ph.D. degree in Computer Vision from University of Sheffield, United Kingdom in 2015. Her study involved video representation and video similarity measurements. Currently, she is an Associate Professor at the Department of Computer Science, UQU, Saudi Arabia. Manal's research interests include deep learning and its application in different areas of computer vision including HealthCare applications.