iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 9 **Issue:** X **Month of publication:** October 2021

DOI: https://doi.org/10.22214/ijraset.2021.38356

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 9 Issue X Oct 2021- Available at www.ijraset.com

Measuring accuracy of Dataset using Deep Learning Algorithm RMSProp Algorithm

Sanjana Naidu Gedela[1], Pavan Bodanki[2]
[1, 2]*Computer Science and Engineering student at Vignan's institute of Information Technology Duvvada, (Visakhapatnam)*
Electronics and communication engineering student at SRKR Engineering college Bhimavaram.

Abstract: ***Over the last few years, there have been many significant improvements in the field of AI, machine learning, deep learning are being used in various industries and research. In order to train the deep learning models learning of parameters plays a major role, here the reduction of loss incurred during the training process is the main objective. In a supervised mode of learning, a model is given the data samples and their respective outcomes. When a model generates an output, it compares it with the desired output and then takes the difference of generated and desired outputs and then attempts to bring the generated output close to the desired output. This is achieved through optimization algorithms. Though many kinds of clinical methods have been employed to detect whether patients have heart disease or not by number of features from patients. but it's still a challenging task due to the multitude of data elements involved. The motive of our project is to save human resources in medical centers and improve accuracy of diagnosis. In our project we used an RMS prop optimizer. The purpose is to decide how many hidden layers need to be selected and how many neurons need to be selected in each and every hidden layer by looking at the dataset and to give the application of deep learning to the health care sector so that we can minimize the costs of treatment and help in proactive actions. We want to show that we can increase the accuracy of the project by taking stability along with accuracy into consideration.***

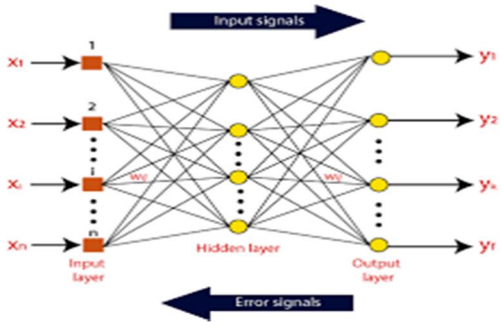
Index Terms: RMS Prop, Machine Learning, Deep Learning, number of features, proactive actions

I. INTRODUCTION

A. What Is A Neural Network?

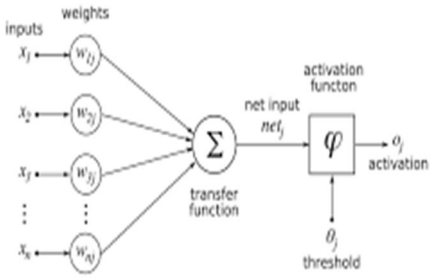
The simplest definition of a neural network, more properly referred to as an 'artificial' neural network (ANN), is provided by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielsen. He defines a neural network as: "...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs." ANNs are processing devices (algorithms or actual hardware) that are loosely modeled after the neuronal structure of the mammalian cerebral cortex but on much smaller scales. A large ANN might have hundreds or thousands of processor units, whereas a mammalian brain has billions of neurons with a corresponding increase in magnitude of their overall interaction and emergent behavior.

1) *The Basics of Neural Networks:* Basically, there are 3 different layers in a neural network :- 1) Input Layer (All the inputs are fed in the model through this layer) 2) Hidden Layers (There can be more than one hidden layers which are used for processing the inputs received from the input layers) 3) Output Layer (The data after processing is made available at the output layer) Neural networks are typically organized in layers. Layers are made up of a number of interconnected 'nodes' which contain an 'activation function'. Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'. The hidden layers then link to an 'output layer' where the answer is output as shown in the graphic below.



©IJRASET: All Rights are Reserved

2) *Activation Functions:* An activation function in a neural network defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network it decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.



B. What Applications Should Neural Networks Be Used For?

Neural networks are universal approximators, and they work best if the system you are using them to model has a high tolerance to error. One would therefore not be advised to use a neural network to balance one's cheque book! However they work very well for:

1) Capturing associations or discovering regularities within a set of patterns.
2) Where the volume, number of variables or diversity of the data is very great.
3) The relationships between variables are vaguely understood and are difficult to describe adequately with conventional approaches.

C. Literature Survey

[1]Number of works have been done related to disease prediction systems using different machine learning algorithms in medical Centres. Senthil Kumar Mohan et al,proposed Effective Heart Disease Prediction Using Hybrid Machine Learning Techniques in which strategy that objective is to find critical includes by applying Machine Learning bringing about improving the exactness in the expectation of cardiovascular malady. The expectation model is created with various blends of highlights and a few known arrangement strategies. We produce an improved exhibition level with a precision level of 88.7% through the prediction model for heart disease with hybrid random forest with a linear model (HRFLM) they likewise educated about Diverse data mining approaches and expectation techniques, Such as, KNN, LR, SVM, NN, and Vote have been fairly famous of late to distinguish and predict heart disease.

[2]Aditi Gavhane, Gouthami Kokkula, Isha Pandya, Prof. Kailas Devadkar (PhD), Prediction of Heart Disease Using Machine Learning, In this paper proposed system they used the neural network algorithm multi-layer perceptron (MLP) to train and test the dataset. In this algorithm there will be multiple layers like one for input, second for output and one or more layers are hidden layers between these two input and output layers. Each node in the input layer is connected to output nodes through these hidden layers. This connection is assigned with some weights. There is another identity input called bias which is with weight b, which is added to the node to balance the perceptron. The connection between the nodes can be feedforward or feedback based on the requirement.

Abhay Kishore et al,[3] developed Heart Attack Prediction Using Deep Learning in which This paper proposes a heart attack prediction system using Deep learning procedures, explicitly Recurrent Neural System to predict the probable prospects of heart related infections of the patient. Recurrent Neural Network is a very ground-breaking characterization calculation that utilizes Deep Learning approach in Artificial Neural Network. The paper talks about in detail the significant modules of the framework alongside the related hypothesis. The proposed model deep learning and data mining to give the precise outcomes least blunders. This paper gives a bearing and point of reference for the advancement of another type of heart attack prediction platform. Prediction stage.

Lakshmana Rao et al,[4] Machine Learning Techniques for Heart Disease Prediction in which the contributing elements for heart disease are more (circulatory strain, diabetes, current smoker, high cholesterol, etc..). So, it is difficult to distinguish heart disease. Different systems in data mining and neural systems have been utilized to discover the seriousness of heart disease among people. The idea of CHD ailment is bewildering, in addition, in this manner, the disease must be dealt with warily. Not doing early identification, may impact the heart or cause sudden passing. The perspective of therapeutic science furthermore, data burrowing is used for finding various sorts of metabolic machine learning, a procedure that causes the framework to gain from past information tests, models without being expressly customized. Machine learning makes rationale dependent on chronicled information.

©IJRASET: All Rights are Reserved

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 9 Issue X Oct 2021- Available at www.ijraset.com

D. Problem Statement

The problem statement is to boost the speed and accuracy of a model. The RMSProp algorithm is used to reach the global minima where the cost function attains the least possible value.Other optimisation algorithms perform frequent updates with a high variance that cause the objective function to fluctuate heavily. Our aim is to propose a model to be able to converge to solutions with better quality.

II. SYSTEM ANALYSIS

A. Existing System

1) Other optimisation algorithms perform frequent updates with a high variance that cause the objective function to fluctuate heavily.
2) The fluctuation enables it to jump to new local minima. This ultimately complicates convergence to the exact minimum.
3) Drawback of other optimization techniques is including all the previous gradients and then trying to capture that mean and feed that mean in this new learning rate.
4) Disadvantage of other algorithms is that there may be a situation where the learning rate becomes very low , as a result of which weight optimisation will be minimal that is nearly equal.
5) Weights of a neural network are optimized using gradient descent.
6) If weight is 0.8 in the next iteration, weight becomes 0.7 , it depends on this formula. This formula talks about how gradient descent optimises the weight in a deep Neural Network, 0.8-0.7 is called step size---0.1 is step size .
7) Step size is minimum when we approach the global minima.
8) When we are far from global minima, step size is more.
9) Optimization in other algorithms is calculated as follows.

$$W_t = W_{t-1} - \eta \left(\frac{\partial \text{loss}}{\partial W}\right) \qquad n_t = \frac{n_{t-1}}{\sqrt{\alpha_t + \varepsilon}} \qquad \alpha_t = \sum_{i=1}^{t} \left(\frac{\partial L}{\partial w_{t-1}}\right)^2$$

B. Proposed System

1) RMS prop might be able to converge to solutions with better quality , for example better local minima.
2) In RMS prop less weightage is given to previous derivatives and more weightage is given to the recent moving average.
3) As a result of this optimization will not be closer to previous optimisations and global minima will be attained quickly.
4) Sometimes if the squares of weights is very high, denominator becomes very high
5) New learning rate becomes very low , as a result of which the optimized weight and the previous weights will be very close. That means weight optimization will be minimal.
6) Hence in RMS Prop, we replace the alpha with average weights.
7) We give less weightage to the previous derivatives and more weightage to the recent moving average , Denominator will never explode because we are giving less value to this term , Hence optimisation will be happening it will never be constant.

$$n_t = \frac{n_{t-1}}{\sqrt{W_{avg} + \varepsilon}} \qquad W_{avg(t)} = \alpha * W_{avg(t-1)} + (1-\alpha) \sum_{i=1}^{t} \left(\frac{\partial L}{\partial w_{t-1}}\right)^2$$

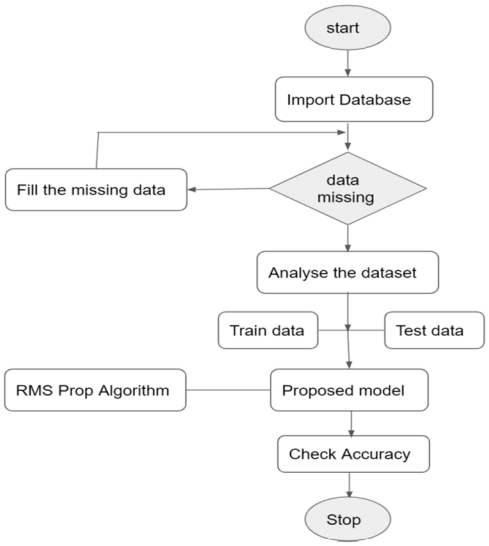
C. System Design

1) Input Design: Input Design plays a vital role in the life cycle of software development, it requires very careful attention of developers. The input design is to feed data to the application as accurately as possible. So inputs are supposed to be designed effectively so that the errors occurring while feeding are minimized. According to Software Engineering Concepts, the input forms or screens are designed to provide validation control over the input limit, range and other related validations. In general, Input design is the process of converting the user created input into a computer-based format. The goal of the input design is to make the data entry logical and free from errors. The errors in the input are controlled by the input design. Since this algorithm is a deep learning algorithm, the input would be the data set . Here we have taken a dataset that is related to Heart disease that means it consists of several columns representing the reasons for heart problems. We took the data set from a well known web site called kaggle. https://www.kaggle.com/cherngs/heart-disease-cleveland-uci
We perform the algorithm on this dataset and predict the target variable which can be a binary value that is either yes or no. The algorithm used is RMS Prop.

©IJRASET: All Rights are Reserved

2) *Output Design:* The output is presented in the form of a percentage. This percentage represents how much the developed algorithm is correct. This is nothing but the accuracy of the model. Accuracy is nothing but the how different is the actual value from the predicted value. If the predicted value is close to the actual value we will get higher accuracy and if the predicted value is no where nearer to the actual value, the accuracy would we very less. In our project we used RMSProp algorithm. It consists of hyperparameters. If we use the generic values for these hyper parameters, we will get some reasonable accuracy. But when we follow our procedure then we can increase the accuracy.

D. System Block Diagram



III. SOFTWARE REQUIREMENTS SPECIFICATIONS

A. Functional Requirements

The functional requirements are the critical requirements which should be within the system in order to overcome or at least minimize the drawbacks of the existing system. Functional requirements consist of what and what type of data should be given as input to the system, what processing should be done to the inputs in order to get the required working, functionalities and output as expected by the user and in what ways or how the output must be presented to the user. In order to make this application functional, we require the following:

1) *Finding Accuracy in a Traditional Way:* Traditionally in the sense generally we substitute the common values for hyper parameters like batch size, epocs and validation_split. Those are such as batch_size being 12, epocs being 100, validation_split being 0.2, If we do like that the accuracy would be 72%.

```
In [10]: model.fit(X_train,y_train, batch_size=12, epochs=100,verbose=1,validation_split=0.2)
16/16 [==============================] - 0s 2ms/step - loss: 0.5862 - accuracy: 0.7244 - val_loss: 0.5772 - val_accuracy: 0.6
957
Epoch 51/100
16/16 [==============================] - 0s 3ms/step - loss: 0.5329 - accuracy: 0.7112 - val_loss: 0.6037 - val_accuracy: 0.7
174
Epoch 52/100
16/16 [==============================] - 0s 3ms/step - loss: 0.5881 - accuracy: 0.6338 - val_loss: 0.5691 - val_accuracy: 0.6
957
Epoch 53/100
16/16 [==============================] - 0s 3ms/step - loss: 0.5834 - accuracy: 0.6749 - val_loss: 0.6889 - val_accuracy: 0.5
435
Epoch 54/100
16/16 [==============================] - 0s 3ms/step - loss: 0.5792 - accuracy: 0.6859 - val_loss: 0.5851 - val_accuracy: 0.6
957
Epoch 55/100
16/16 [==============================] - 0s 3ms/step - loss: 0.5369 - accuracy: 0.7653 - val_loss: 0.5707 - val_accuracy: 0.6
522
Epoch 56/100
16/16 [==============================] - 0s 3ms/step - loss: 0.5787 - accuracy: 0.7084 - val_loss: 0.5756 - val_accuracy: 0.6
739

In [11]: score = model.evaluate(X_test, y_test, verbose=1)
print("\nTest score:", score[0])
print('Test accuracy:', score[1])
3/3 [==============================] - 0s 0s/step - loss: 0.4560 - accuracy: 0.7237

Test score: 0.45598360896110535
Test accuracy: 0.7236841917037964

In [13]: scores = model.evaluate(X_test, y_test, batch_size=32, verbose=1)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
3/3 [==============================] - 0s 0s/step - loss: 0.4560 - accuracy: 0.7237

accuracy: 72.37%
```

©IJRASET: All Rights are Reserved

2) *Calculating most Probable Values for Hyper Parameters:* Validation_split being the first parameter, using a for loop we calculate the accuracy at every split value varying from 0.1 to 0.5. Then we finalize the value of validation split taking into consideration the value at which the accuracy is more as well as it is stable for some time which means there should not be any sudden change of raise or drop in the accuracy with respect to neighbouring splits. From this we finalize the value of validation split to be 0.2

```
In [90]: d={}
         for s in range(10,30,2):
             model.fit(X_train,y_train, batch_size=12, epochs=100,verbose=1,validation_split=s/100)
             score = model.evaluate(X_test, y_test, verbose=1)
             key=str(s/100)
             d[key]=score[1]

         plt.plot(list(d.keys()),list(d.values()))
         plt.show
Out[90]: <function matplotlib.pyplot.show(close=None, block=None)>
```

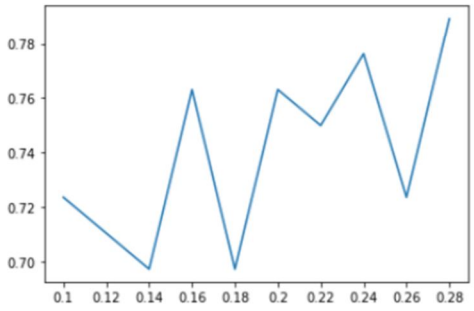
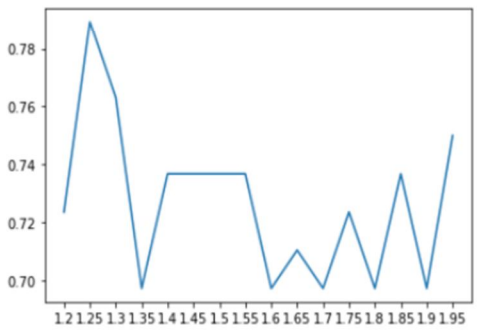


Fig.5 validation_split

Epocs being the first parameter, using a for loop we calculate the accuracy at every split value varying from 50 to 200. Then we finalize the value of epochs taking the value at which the accuracy is more as well as it is stable for some time which means there should not be any sudden change of raise or drop in the accuracy with respect to neighbouring splits. From this we finalize the value of epochs to be 120.

```
In [100]: de={}
          for e in range(120,200,5):
              model.fit(X_train,y_train, batch_size=12, epochs=e,verbose=1,validation_split=0.2)
              score = model.evaluate(X_test, y_test, verbose=1)
              key=str(e/100)
              de[key]=score[1]

          plt.plot(list(de.keys()),list(de.values()))
          plt.show
Out[100]: <function matplotlib.pyplot.show(close=None, block=None)>
```



©IJRASET: All Rights are Reserved

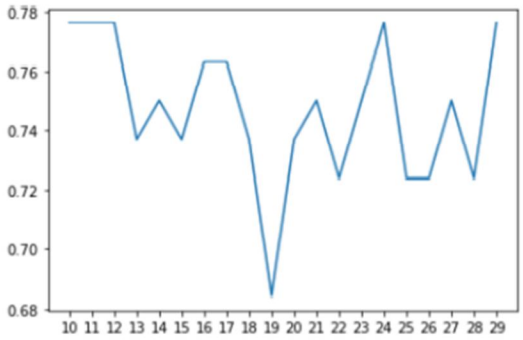
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 9 Issue X Oct 2021- Available at www.ijraset.com

Batch_size being the first parameter, using a for loop we calculate the accuracy at every split value varying from 10 to 30. Then we finalize the value of batch_size taking into consideration the value at which the accuracy is more as well as it is stable for some time which means there should not be any sudden change of raise or drop in the accuracy with respect to neighbouring splits. From this we finalize the value of batch_size to be 11.

```
In [107]: db={}
          for b in range(10,30):
              model.fit(X_train,y_train, batch_size=b, epochs=120,verbose=1,validation_split=0.2)
              score = model.evaluate(X_test, y_test, verbose=1)
              key=str(b)
              db[key]=score[1]

          plt.plot(list(db.keys()),list(db.values()))
          plt.show

Out[107]: <function matplotlib.pyplot.show(close=None, block=None)>
```



3) *Finding Accuracy after Substituting new Hyperparameters:* After substituting the above calculated values of hyperparameters, the accuracy is increased to 79%. This can be visualized as follows….

```
In [108]: history=model.fit(X_train,y_train, batch_size=11, epochs=120,verbose=1,validation_split=0.2)
          score = model.evaluate(X_test, y_test, verbose=1)

          Epoch 1/120
          17/17 [==============================] - 0s 5ms/step - loss: 3.6330e-08 - accuracy: 1.0000 - val_loss: 18.1288 - val_accurac
          y: 0.7391
          Epoch 2/120
          17/17 [==============================] - 0s 3ms/step - loss: 3.6889e-08 - accuracy: 1.0000 - val_loss: 18.7892 - val_accurac
          y: 0.7609
          Epoch 3/120
          17/17 [==============================] - 0s 3ms/step - loss: 0.0166 - accuracy: 0.9945 - val_loss: 18.2036 - val_accuracy: 0.
          7174
          Epoch 4/120
          17/17 [==============================] - 0s 3ms/step - loss: 0.0691 - accuracy: 0.9945 - val_loss: 18.0485 - val_accuracy: 0.
          8043
          Epoch 5/120
          17/17 [==============================] - 0s 3ms/step - loss: 0.1038 - accuracy: 0.9890 - val_loss: 15.9491 - val_accuracy: 0.
          7391
          Epoch 6/120
          17/17 [==============================] - 0s 3ms/step - loss: 1.3507e-04 - accuracy: 1.0000 - val loss: 15.3530 - val accurac
          y: 0.7826
          Epoch 7/120

In [109]: print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

          accuracy: 78.95%
```

©IJRASET: All Rights are Reserved

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 9 Issue X Oct 2021- Available at www.ijraset.com

IV. SYSTEM IMPLEMENTATION

A. *Organization Of Data Analysis*

To identify the most important variables and to define the best algorithm or model for predicting our target variable. Hence, this analysis will be divided into five stages:

1) *Importing Data sets*
2) *Feature Engineering:* Data is like the crude oil of machine learning which means it has to be refined into features — predictor variables — to be useful for training a model. Without relevant features, you can't train an accurate model, no matter how 9 complex the machine learning algorithm is. The process of extracting features from a raw dataset is called feature engineering.
3) *Exploratory Data Analysis:* Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypotheses and to check assumptions with the help of summary statistics and graphical representations.

a) Maximize insight into a data set
b) Uncover underlying structure
c) Extract important variables
d) Detect outliers and anomalies
e) Test underlying assumptions
f) Develop parsimonious models
g) Determine optimal factor settings

4) *Data Pre-processing:* Data Pre-processing Data pre-processing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviours or trends, and is likely to contain many errors. It is a proven method of resolving such issues. Data pre-processing prepares raw data for further processing. Data goes through a series of steps during pre-processing:

a) *Data Cleaning:* Data is cleansed through processes such as filling in missing values, smoothing noisy data, or resolving inconsistencies in the data.
b) *Data Integration:* Data with different representations are put together and conflicts within the data are resolved.
c) *Data Transformation:* Data is normalized, aggregated and generalized.
d) *Data Reduction:* This step aims to present a reduced representation of the data in a data warehouse.
e) *Data Discretization:* Involves the reduction of a number of values of a continuous attribute by dividing the range of attribute intervals.

5) *Modelling:* The outputs of prediction and feature engineering are a set of label times, historical examples of what we want to predict, and features, predictor variables used to train a model to predict the label. The process of modeling means training a machine learning algorithm to 12 predict the labels from the features, tuning it for the business need, and validating it on holdout data. The objective of machine learning is not a model that does well on training data, but one that demonstrates it satisfies the business need and can be deployed on live data. Similar to feature engineering, modeling is independent of the previous steps in the machine learning process and has standardized inputs which means we can alter the prediction problem without needing to rewrite all our code. If the business requirements change, we can generate new label times, build corresponding features, and input them into the model. Now, let's focus on the given data set we are working on.As mentioned above , the data we are working on has numerical data as independent variables(predictors) and dependent variables(Target) . So from this we can infer that we need to apply regression models to predict the sale values. Prediction is the process of predicting values of target variables by giving new testing data to the model. Accuracy is the comparison between the predicted output and actual output. Ex: if accuracy is 80% then it shows that 80 % of the data is correctly predicted and the rest 20% of the data is incorrectly predicted. 13 More the accuracy of the model, the better predictions you obtain.

B. *Introduction To Deep Learning*

Deep learning is a branch of machine learning which is completely based on artificial neural networks, as neural networks are going to mimic the human brain so deep learning is also a kind of mimic of the human brain. In deep learning, we don't need to explicitly program everything. The concept of deep learning is not new.

©IJRASET: All Rights are Reserved

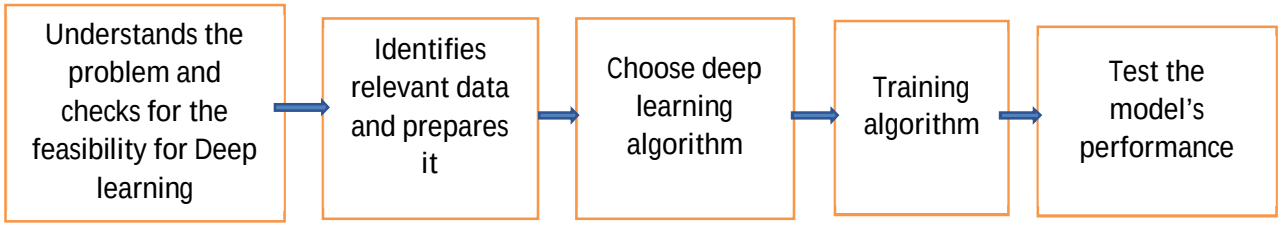
It has been around for a couple of years now. It's hype nowadays because earlier we did not have that much processing power and a lot of data. As in the last 20 years, the processing power increases exponentially, deep learning and machine learning came into the picture. A formal definition of deep learning is- neurons Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones. So, we create an artificial structure called an artificial neural net where we have nodes or neurons. We have some neurons for input value and some for output value and in between, there may be lots of neurons interconnected in the hidden layer.

1) *Architectures*

a) *Deep Neural Network:* It is a neural network with a certain level of complexity (having multiple hidden layers in between input and output layers). They are capable of modeling and processing non-linear relationships.

b) *Deep Belief Network (DBN):* It is a class of Deep Neural Network. It is a multi-layer belief network. Steps for performing DBN : a. Learn a layer of features from visible units using the Contrastive Divergence algorithm. b. Treat activations of previously trained features as visible units and then learn features of features. c. Finally, the whole DBN is trained when the learning for the final hidden layer is achieved.

2) *Working:* First, we need to identify the actual problem in order to get the right solution and it should be understood. The feasibility of Deep Learning should also be checked (whether it should fit Deep Learning or not). Second, we need to identify the relevant data which should correspond to the actual problem and should be prepared accordingly. Third, Choose the Deep Learning Algorithm appropriately. Fourth, algorithms should be used while training the dataset. Fifth, Final testing should be done on the dataset.



C. Introduction To Gradient Descent

Gradient descent is an optimization algorithm that follows the negative gradient of an objective function in order to locate the minimum of the function. A limitation of gradient descent is that it uses the same step size (learning rate) for each input variable. AdaGrad, for short, is an extension of the gradient descent optimization algorithm that allows the step size in each dimension used by the optimization algorithm to be automatically adapted based on the gradients seen for the variable (partial derivatives) over the course of the search. A limitation of AdaGrad is that it can result in a very small step size for each parameter by the end of the search that can slow the progress of the search down too much and may mean not locating the optima.

Root Mean Squared Propagation, or RMSProp, is an extension of gradient descent and the AdaGrad version of gradient descent that uses a decaying average of partial gradients in the adaptation of the step size for each parameter. The use of a decaying moving average allows the algorithm to forget early gradients and focus on the most recently observed partial gradients seen during the progress of the search, overcoming the limitation of AdaGrad.

1) *Gradient Descent:* Is an optimization algorithm.It is technically referred to as a first-order optimization algorithm as it explicitly makes use of the first order derivative of the target objective function. The first order derivative, or simply the "derivative," is the rate of change or slope of the target function at a specific point, e.g. for a specific input. If the target function takes multiple input variables, it is referred to as a multivariate function and the input variables can be thought of as a vector. In turn, the derivative of a multivariate target function may also be taken as a vector and is referred to generally as the gradient. Gradient: First order derivative for a multivariate objective function. The derivative or the gradient points in the direction of the steepest ascent of the target function for a specific input. Gradient descent refers to a minimization optimization algorithm that follows the negative of the gradient downhill of the target function to locate the minimum of the function. The gradient descent

©IJRASET: All Rights are Reserved

algorithm requires a target function that is being optimized and the derivative function for the objective function. The target function f() returns a score for a given set of inputs, and the derivative function f'() gives the derivative of the target function for a given set of inputs. The gradient descent algorithm requires a starting point (x) in the problem, such as a randomly selected point in the input space. The derivative is then calculated and a step is taken in the input space that is expected to result in a downhill movement in the target function, assuming we are minimizing the target function. A downhill movement is made by first calculating how far to move in the input space, calculated as the step size (called alpha or the learning rate) multiplied by the gradient.

This is then subtracted from the current point, ensuring we move against the gradient, or down the target function.

a) x = x – step_size * f'(x)

The steeper the objective function at a given point, the larger the magnitude of the gradient, and in turn, the larger the step taken in the search space. The size of the step taken is scaled using a step size hyperparameter.

b) *Step Size (alpha):* Hyperparameter that controls how far to move in the search space against the gradient each iteration of the algorithm.

If the step size is too small, the movement in the search space will be small and the search will take a long time. If the step size is too large, the search may bounce around the search space and skip over the optima.

D. RMS Prop Algorithm

RMS PROP is a kind of gradient descent optimization algorithm

RMSprop lies in the realm of adaptive learning rate methods, which have been growing in popularity in recent years, but also getting some criticism. It's famous for not being published, yet being very well-known; most deep learning framework include the implementation of it out of the box. There are two ways to introduce RMSprop. First, is to look at it as the adaptation of rprop algorithm for mini-batch learning. It was the initial motivation for developing this algorithm. Another way is to look at its similarities with Adagrad and view RMSprop as a way to deal with its radically diminishing learning rates.

1) *RProp rprop:* Algorithm that's used for full-batch optimization. Rprop [3] tries to resolve the problem that gradients may vary widely in magnitudes. Some gradients may be tiny and others may be huge, which results in a very difficult problem — trying to find a single global learning rate for the algorithm. Rprop combines the idea of only using the sign of the gradient with the idea of adapting the step size individually for each weight. So, instead of looking at the magnitude of the gradient, we'll look at the step size that's defined for that particular weight. And that step size adapts individually over time, so that we accelerate learning in the direction that we need. To adjust the step size for some weight, the following algorithm is used:

a) First, we look at the signs of the last two gradients for the weight.

b) If they have the same sign, that means, we're going in the right direction, and should accelerate it by a small fraction, meaning we should increase the step size multiplicatively(e.g by a factor of 1.2). If they're different, that means we did too large of a step and jumped over a local minima, thus we should decrease the step size multiplicatively(e.g. by a factor of 0.5).

c) Then, we limit the step size between some two values. These values really depend on your application and dataset, good values that can be for default are 50 and a millionth, which is a good start.

d) Now we can apply the weight update.

2) *Rprop to RMSprop:* Rprop doesn't really work when we have very large datasets and need to perform mini-batch weights updates.The reason it doesn't work is that it violates the central idea behind stochastic gradient descent, which is when we have small enough learning rate, it averages the gradients over successive mini-batches. With rprop, we increment the weight 9 times and decrement only once, so the weight grows much larger. Rprop is equivalent to using the gradient but also dividing by the size of the gradient, so we get the same magnitude no matter how big a small that particular gradient is. The problem with mini-batches is that we divide by different gradients every time, so why not force the number we divide by to be similar for adjacent mini-batches ? The central idea of RMSprop is to keep the moving average of the squared gradients for each weight. And then we divide the gradient by the square root of the mean square. Which is why it's called RMSprop(root mean square). With math equations the update rule looks like this

©IJRASET: All Rights are Reserved

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)\left(\frac{\delta C}{\delta w}\right)^2$$

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t}} \frac{\delta C}{\delta w}$$

As you can see from the above equation we adapt the learning rate by dividing by the root of the squared gradient, but since we only have to estimate the gradient on the current mini-batch, we need instead to use the moving average of it. Default value for the moving average parameter that you can use in your projects is 0.9. It works very well for most applications.

3) *Similarity with Adagrad:* Adagrad[2] is an adaptive learning rate algorithm that looks a lot like RMSprop. Adagrad adds element-wise scaling of the gradient based on the historical sum of squares in each dimension. This means that we keep a running sum of squared gradients. And then we adapt the learning rate by dividing it by that sum. If we have two coordinates — one that always has big gradients and one that has small gradients we'll be dividing by the corresponding big or small number so we accelerate movement among small directions, and in the direction where gradients are large we're going to slow down as we divide by some large number. In such a case, Steps get smaller and smaller and smaller, because we keep updating the squared grads growing over training. So we divide by the larger number every time. In convex optimization, this makes a lot of sense, because when we approach minima we want to slow down. In non-convex cases it's bad as we can get stuck on the saddle point. We can look at RMSprop as algorithms that address that concern a little bit. With RMSprop we still keep that estimate of squared gradients, but instead of letting that estimate continually accumulate over training, we keep a moving average of it. Visualizations for different optimization algorithms that show how they behave in different situations.

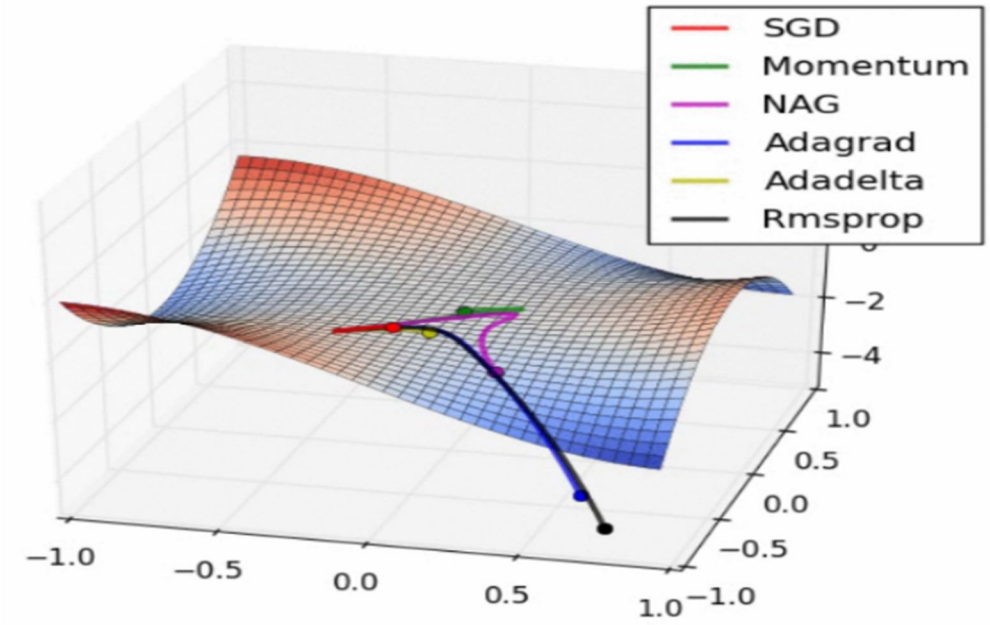


Fig.17 Speed of different algorithms

As you can see, with the case of saddle point, RMSprop(black line) goes straight down, it doesn't really matter how small the gradients are, RMSprop scales the learning rate so the algorithm goes through saddle point faster than most.

©IJRASET: All Rights are Reserved

E. Choosing Number Of Hidden Layers

- If the data is linearly separable then you don't need any hidden layers at all.
- If data is less complex and is having fewer dimensions or features then neural networks with 1 to 2 hidden layers would work.
- If data is having large dimensions or features then to get an optimum solution, 3 to 5 hidden layers can be used. It should be kept in mind that increasing hidden layers would also increase the complexity of the model and choosing hidden layers such as 8, 9, or in two digits may sometimes lead to overfitting.

1) Choosing Nodes in Hidden Layers Once hidden layers have been decided the next task is to choose the number of nodes in each hidden layer.

a) The number of hidden neurons should be between the size of the input layer and the output layer.

b) The most appropriate number of hidden neurons is sqrt(input layer nodes * output layer nodes)

c) The number of hidden neurons should keep on decreasing in subsequent layers to get more and more close to pattern and feature extraction and to identify the target class. These above algorithms are only a general use case and they can be molded according to use case. Sometimes the number of nodes in hidden layers can increase also in subsequent layers and the number of hidden layers can also be more than the ideal case.This whole depends upon the use case and problem statement that we are dealing with.

2) Based on Additional Research: In general, you cannot analytically calculate the number of layers or the number of nodes to use per layer in an artificial neural network to address a specific real-world predictive modeling problem. The number of layers and the number of nodes in each layer are model hyperparameters that you must specify. You are likely to be the first person to attempt to address your specific problem with a neural network. No one has solved it before you. Therefore, no one can tell you the answer of how to configure the network.

F. Activation Functions

In the Neural Network the activation function defines if a given node should be “activated” or not based on the weighted sum. Let’s define this weighted sum value as z. In this section I would explain why “Step Function” and “Linear Function” won’t work and talk about “Sigmoid Function” , one of the most popular activation functions. There are also other functions which I will leave aside for now. Step Function One of the first ideas would be to use so called “Step Function” (discrete output values) where we define threshold value and:

if(z > threshold) — “activate” the node (value 1)
if(z < threshold) — don’t “activate” the node (value 0)

This looks nice but it has drawbacks since the node can only have value 1 or 0 as output. In case when we would want to map multiple output classes (nodes) we got a problem. The problem is that it is possible for multiple output classes/nodes to be activated (to have the value 1). So we are not able to properly classify/decide.

1) Sigmoid Function: It is one of the most widely used activation functions today. The equation is given with the formula below. It has multiple properties which makes it so popular:

a) It’s non-linear function

b) Range values are between (0,1)

c) Between (-2,2) on x-axis the function is very steep, that cause function to tend to classify values either 1 or 0

d) Because of these properties it allows the nodes to take any values between 0 and 1. In the end, in case of multiple output classes, this would result with different probabilities of “activation” for each output class. And we will choose the one with the highest “activation”(probability) value.

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^{x}}{e^{x}+1}.$$

©IJRASET: All Rights are Reserved

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 9 Issue X Oct 2021- Available at www.ijraset.com

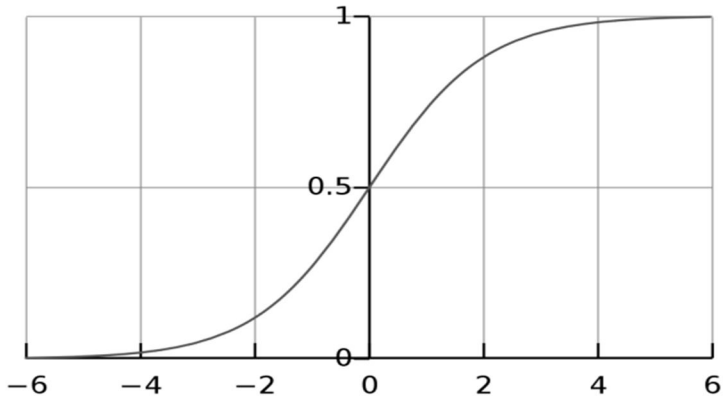
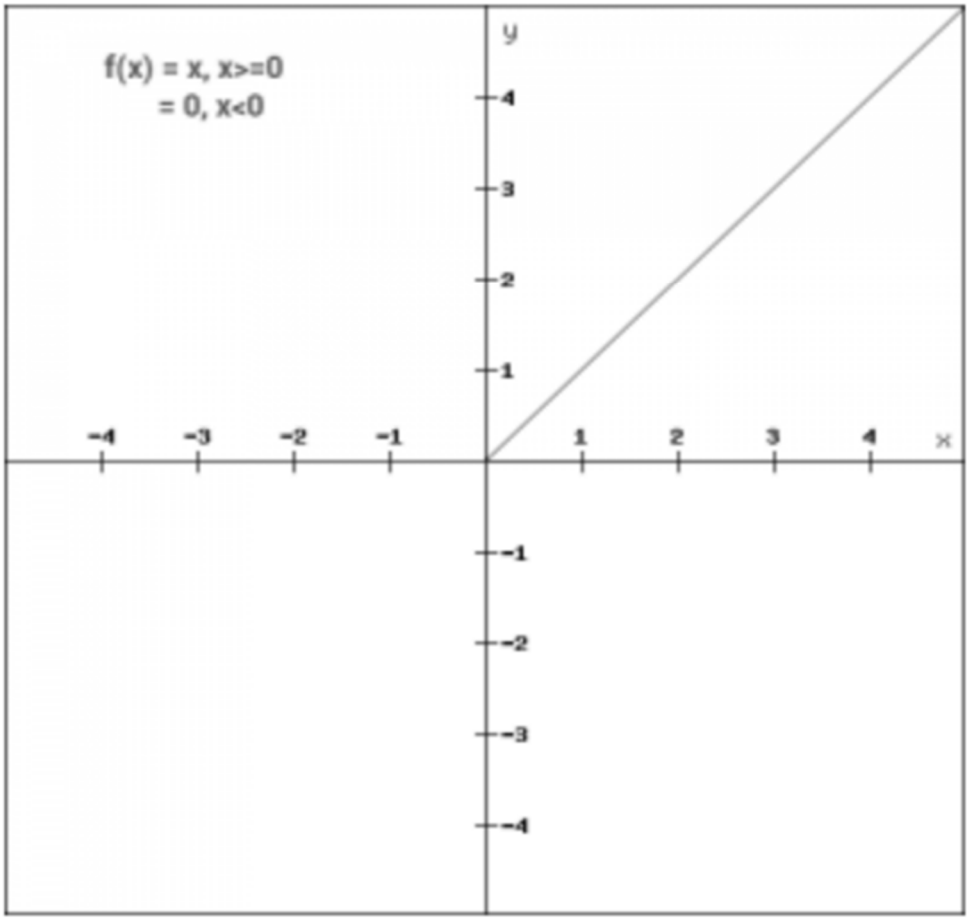


Fig.19 Sigmoid Graph

2) *ReLU - Rectified Linear Unit:* The ReLU function is another non-linear activation function that has gained popularity in the deep learning domain. ReLU stands for Rectified Linear Unit. The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time. This means that the neurons will only be deactivated if the output of the linear transformation is less than 0. The plot below will help you understand this better



For the negative input values, the result is zero, that means the neuron does not get activated. Since only a certain number of neurons are activated, the ReLU function is far more computationally efficient when compared to the sigmoid and tanh function.

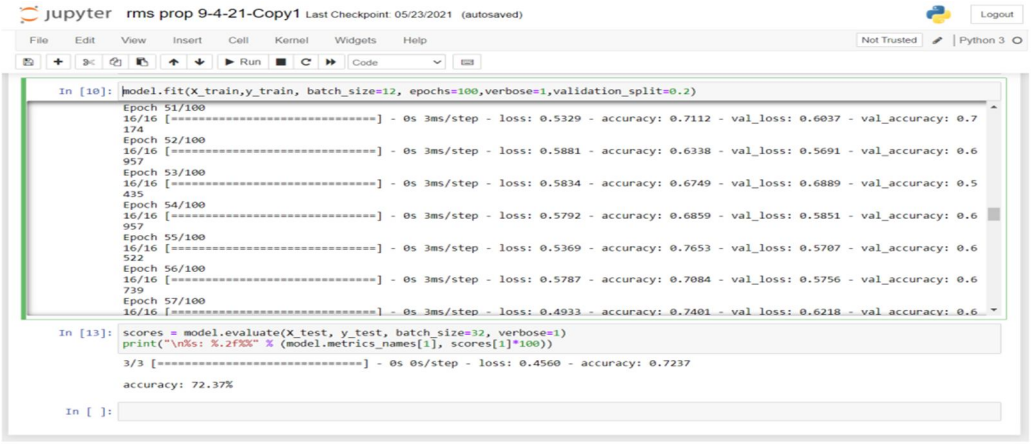
©IJRASET: All Rights are Reserved

3) *Justification for using Sigmoid and ReLU:* In our project the Activation functions which we have used are ReLU and Sigmoid - We used ReLU function for the input layers (Hidden Layers) - We used Sigmoid function for the output layers In ReLU function all the neurons are not activated at the same time. This means that the neurons will only be deactivated if the output of the linear transformation is less than zero. Sigmoid function is used in the output layer. The sigmoid function works well in the case of classification. Sigmoid and tanh should not be used as activation functions for the hidden layer. This is because of the vanishing gradient problem, i.e., if your input is on a higher side (where sigmoid goes flat) then the gradient will be near zero. This will cause very slow or no learning during backpropagation as weights will be updated with really small values. Therefore, The best function for hidden layers is thus ReLu.

V. TEST CASES

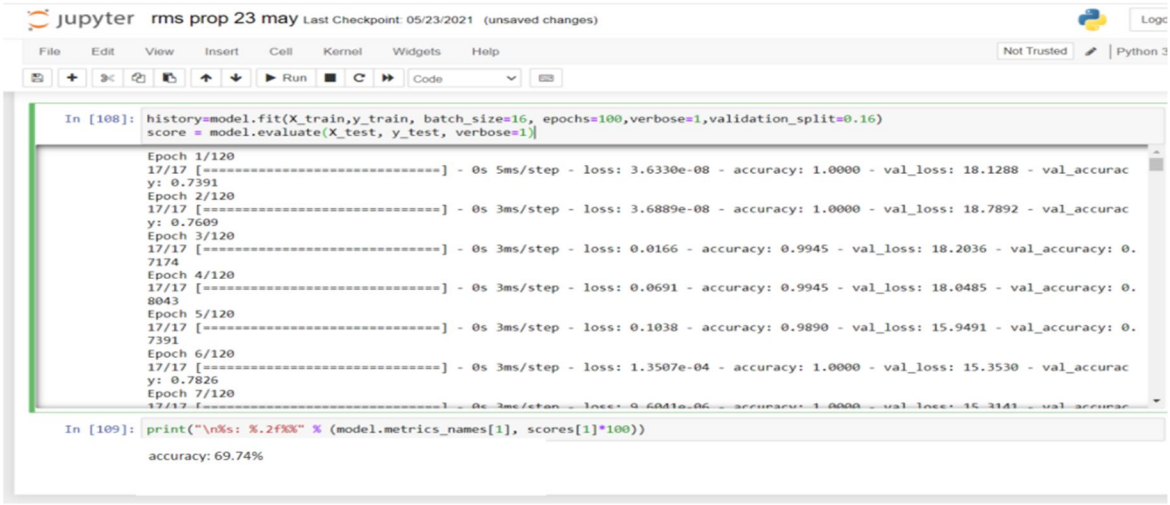
The following figures are some of the validations done in our project. By varying the values of hyper parameters, lock the values which yield higher accuracy.

1) In the below figure, it represents the accuracy of the model when the values of hyper parameters are as follows - validation_split - 0.2 , batch_size - 12 , epocs – 100



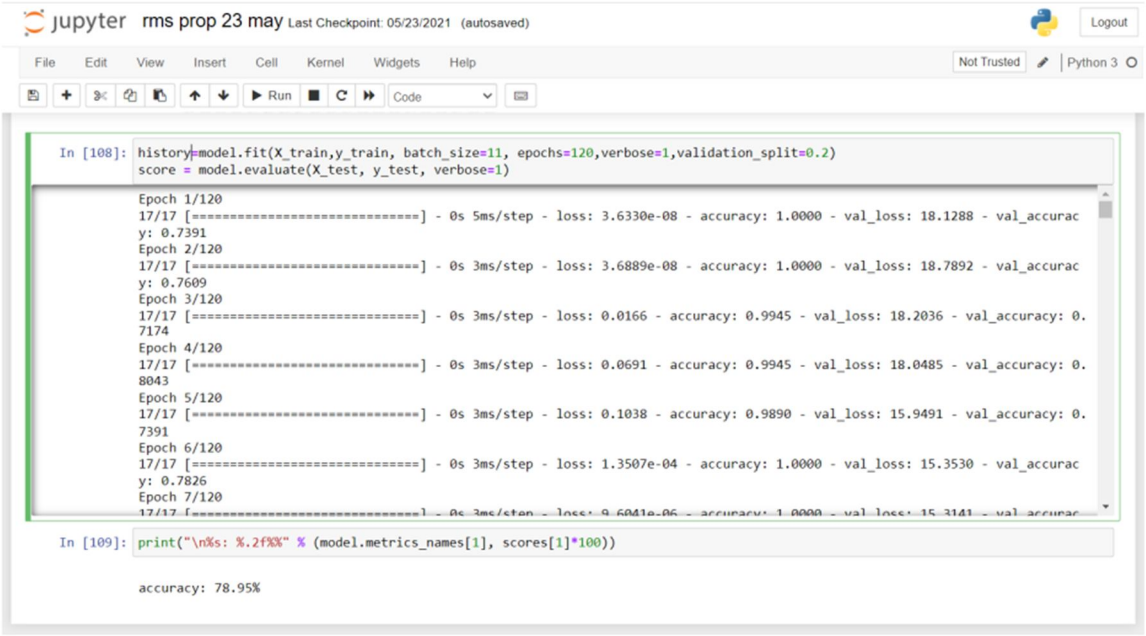
Accuracy : 72% **Fig.4 Accuracy 1**

2) In the below figure, it represents the accuracy of the model when the values of hyper parameters are as follows - validation_split - 0.16 , batch_size - 16 , epocs – 100



Accuracy : 69% **Fig.27 Accuracy 3**

3) In the below figure, it represents the accuracy of the model when the values of hyper parameters are as follows – validation_split - 0.2 , batch_size - 11 , epocs – 120.



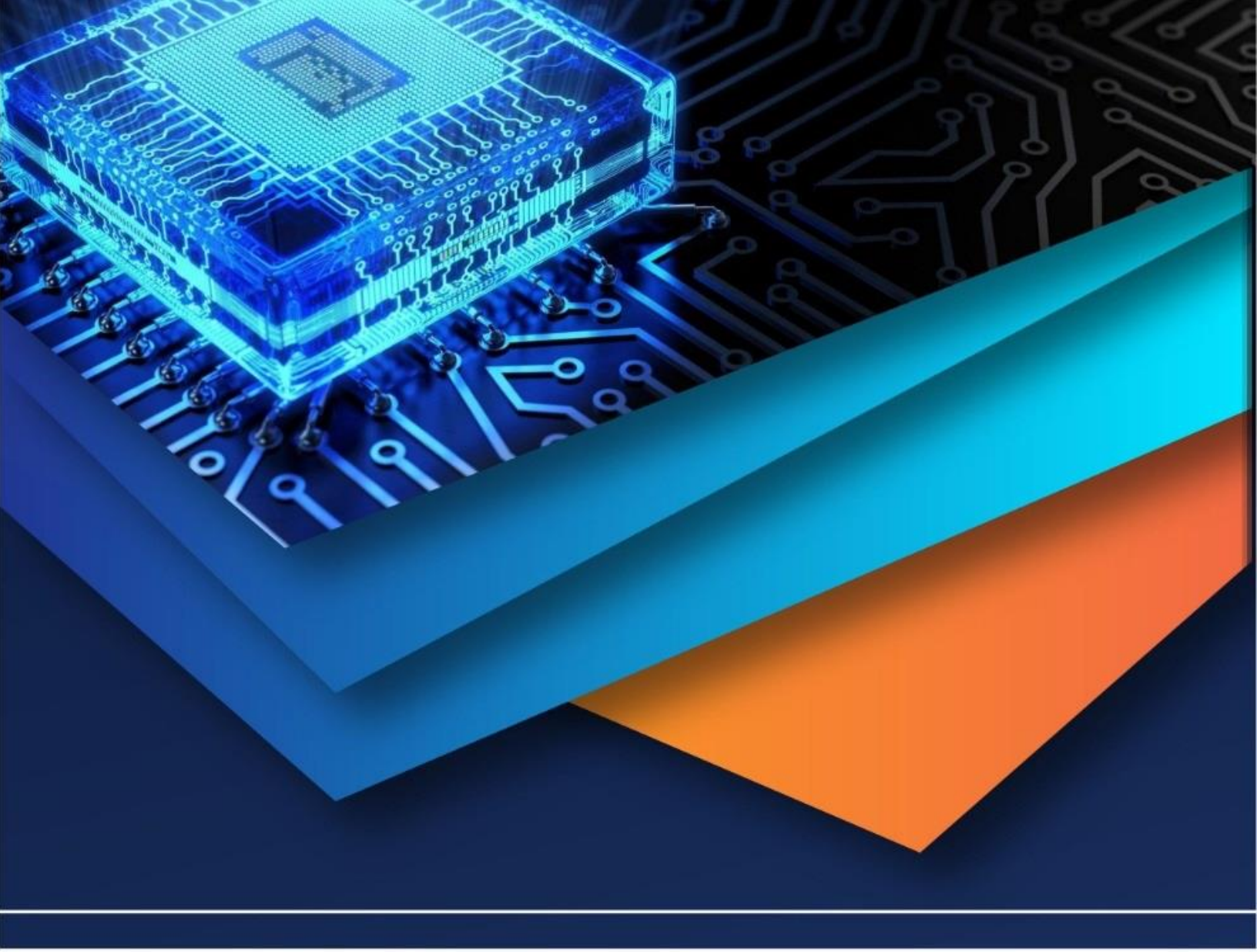
VI. CONCLUSION

RMSProp, root mean squared propagation is the optimization machine learning algorithm to train the Artificial Neural Network (ANN) by different adaptive learning rate and derived from the concepts of gradient descent and RProp. Combining averaging over mini-batches, efficiency, and the gradients over successive mini-batches, RMSProp can reach the faster convergence rate than the original optimizer, but lower than the advanced optimizer such as Adam. As knowing the high performance of RMSProp and the possibility of combining with other algorithms, harder problems could be better described and converged in the future. It is an extension of gradient descent and the AdaGrad version of gradient descent that uses a decaying average of partial gradients in the adaptation of the step size for each parameter. The use of a decaying moving average allows the algorithm to forget early gradients and focus on the most recently observed partial gradients. In this project, we have implemented RMSprop Optimization technique to find the accuracy of a dataset inorder to ensure that model is error free and to increase the performance of the proposed system. The proposed work provides the better accuracy when compared with previous works. In order to increase the accuracy further, different optimization algorithm need to be investigated.

REFERENCES

[1] Geoffrey Hinton Neural Networks for machine learning online course. https://www.coursera.org/learn/neural-networks/home/welcome.
[2] The Marginal Value of Adaptive Gradient Methods in Machine Learning Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, Benjamin Recht.
[3] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 12, 2121–2159. Retrieved from http://jmlr.org/papers/v12/duchi11a.html ↩
[4] Bengio, Y., Boulanger-Lewandowski, N., & Pascanu, R. (2012). Advances in Optimizing Recurrent Networks. Retrieved from http://arxiv.org/abs/1212.0901 ↩
[5] Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. Retrieved from http://arxiv.org/abs/1212.5701 ↩
[6] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., … Zheng, X. (2015). TensorFlow : Large-Scale Machine Learning on Heterogeneous Distributed Systems.
[7] Zhang, S., Choromanska, A., & LeCun, Y. (2015). Deep learning with Elastic Averaging SGD. Neural Information Processing Systems Conference (NIPS 2015), 1–24. Retrieved from http://arxiv.org/abs/1412.6651 ↩
[8] Ioffe, S., & Szegedy, C. (2015). Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv Preprint arXiv:1502.03167v3.

©IJRASET: All Rights are Reserved



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)