Optimization of Deep Learning using various Optimizers, Loss functions and Dropout

S. V. G. Reddy, K.Thammi Reddy, V.Valli Kumari

Abstract: ***Deep Learning is gaining lot of prominence due to its break through results in various fields like Computer Vision, Natural Language Processing, Time Series Analysis, Health Care etc. Earlier, the Deep Learning was implemented using the batch and stochastic gradient descent algorithms and some optimizers which lead to very less performance of the models. But today, lot of work is going on for the enhancement of the performance of Deep Learning using various optimization techniques. So, in this context, It is proposed to build a Deep Learning model using various Optimizers (Adagrad, RmsProp, Adam), Loss functions (mean squared error, binary cross entropy) and Dropout concept for the Convolutional neural networks and Recurrent neural networks and verify the performance such as Accuracy and Loss of the model. The proposed model has achieved maximum Accuracy when Adam optimizer and mean squared error loss function are applied on convolutional neural networks and the model is run with minimum Loss when the same Adam optimizer and mean squared error loss function are applied on Recurrent neural networks. While performing the Regularization of the model, the maximum Accuracy is achieved when the Dropout with a minimum fraction 'p' of nodes is applied on convolutional neural networks and the model has run with minimum Loss when the same dropout value is applied on Recurrent neural networks.***

Keywords : Deep Learning, Convolutional Neural Networks, CNN, Recurrent Neural Networks, RNN, Computer Vision, Natural language processing, Time Series Analysis.

I. INTRODUCTION

Deep Learning

Deep Learning [1][2][3] is a machine learning technique which has been applied in various domains like computer vision, natural language processing, robotics, health care and artificial intelligence etc. In Deep Learning, the learning is achieved at various levels. The inputs are processed at the initial level, transform into an abstract form, at the next level, still processed into more precise form and likewise the learning is achieved more deeper. In this paper, the model is built using the Deep Learning algorithms such as Convolutional neural networks(CNN), Recurrent neural networks(RNN). Firstly, Let's understand the Convolutional neural networks. The Convolutional neural networks [4][5] are used mainly to identify or recognize the image which is termed as computer vision. Computer Vision is to make computers understand images and videos. It is used for optical character recognition, face detection, smile detection, object recognition etc. The supervised Deep Learning algorithm, Convolutional Neural Networks is a big breakthrough in the field of Computer Vision. The process of convolution is done in the following manner. Consider an input image containing pixels and its binary representation of pixels is taken. Here the binary representation of an input image is taken as a matrix of order 7 x 7 (refer fig 1). To extract few characteristics of the image such as edge detection, emboss, blur etc., various filters or feature detectors would be used. So, a feature detector or a filter is taken as a matrix of order 3 x 3 (refer fig 2) with random elements and compared with the first 3 x 3 order of input vector (which means the zeroth, first, second rows and zeroth, first, second columns). So, the logical AND operation is applied on the first element (zeroth row and zeroth column) of feature detector or filter and with the corresponding first element (zeroth row and zeroth column) of the input vector and check for the output(refer table 1) and count the total number of combinations which are producing an output of bit '1'. Here, the total number of combinations are 'zero' producing an output of 1. After the first iteration, the number of combinations '0' is written as the first element in the 5 x 5 two dimensional feature map. Next, when the comparison is done with the filter and the next 3 x 3 elements of Input vector (zeroth, first, second rows and first, second, third columns) , the total number of combinations are 'one' producing an output of 1 and so on. After the second iteration the number of combinations '1' is written as the second element in the 5 x 5 two dimensional feature map. Likewise, by mapping the total number of combinations which are producing output 1, it generates the complete two dimensional feature map. This is the process of convolution. A big size input vector would be compared with a small size filter or feature detector and produce a moderate size feature map. Similarly, the various filters or feature detectors are applied on the given input vector and produce several corresponding feature maps and the set of all these feature maps would be termed as convolution layer. Here, the Rectified linear unit (ReLU) is used as the activation function. And then the Maximum Pooling process is applied on a feature map. Here, a matrix of order 2 x 2 is considered for this process. It means, from the feature map, by reading the first 2 x 2 matrix elements (zeroth, first rows and columns), the largest element is picked and map it as the first element in the new matrix called Pooled feature map. Next, for the second iteration, the next 2 x 2 matrix elements (zeroth, first rows and first, second columns) of feature map are considered and pick the largest element and map it as the next element of pooled feature map and in the similar way

Manuscript published on 30 December 2018.
***Correspondence Author(s)**
S.V.G.Reddy, Associate Professor, Dept. of CSE, GIT, GITAM University,Hyderabad, Telangana, India.
Prof. K.Thammi Reddy, Professor, Dept. of CSE, GIT, GITAM University,Hyderabad, Telangana, India
Prof. V.ValliKumari, Professor, Dept. of CS & SE, college of engineering, Andhra University, Andhra Pradesh, India

the complete pooled feature map matrix is generated. Now, for all the feature maps of the convolution layer, the new set of pooled feature maps are generated and it is termed as pooled convolution layer. Then, every pooled feature map of pooling layer which is a two dimensional matrix is Flattened into a one dimensional matrix in a sequential way (refer Fig 3) and this matrix is given to the Input layer of Artificial neural network and then passed to the hidden layers and to the output layer and produce the final output. It means for the convolutional neural network model, the image is given as input which is passed through the convolution layer, pooling layer and the flattening is applied and passed to the Input layer of Artificial neural network, hidden layer and then to the output layer and finally the model identifies or recognizes the image(refer fig 3).

Table 1 – The convolution process

Feature detector or filter	Input vector	Output
0	0	0
0	1	0
1	0	0
1	1	1

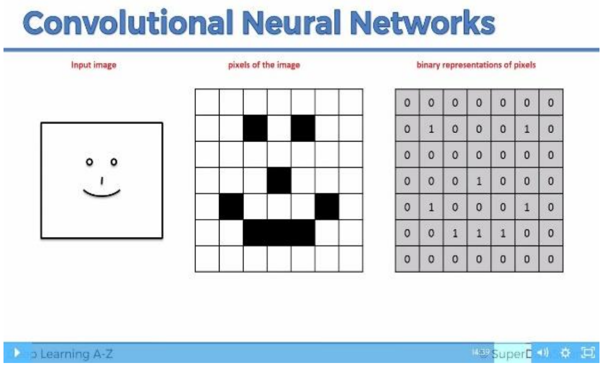
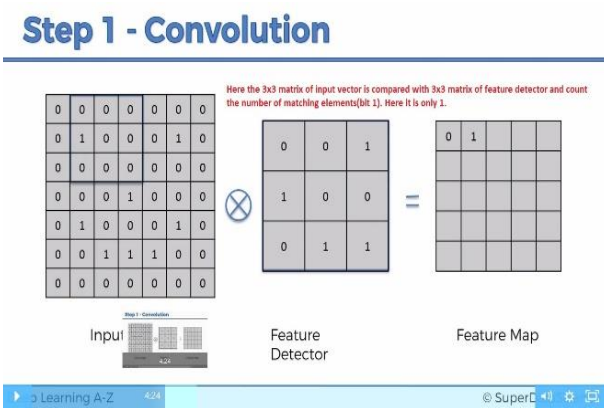

Convolutional Neural Networks
Input image
pixels of the image
binary representations of pixels
Learning A-Z
SuperDataScience

Fig 1 – The image, pixels and its binary representation



Here the 3x3 matrix of input vector is compared with 3x3 matrix of feature detector and count the number of matching elements(bit 1). Here it is only 1.

Fig 2 – The process of convolution of an image

Now, let's comprehend the Recurrent neural networks. The concept of Recurrent neural networks [6][7][8] is predominantly applied in the fields of natural language modelling, machine translation and speech recognition. The main technique behind the Recurrent neural networks is the sequence of input. The Recurrent neural network would be having series or sets of input layer, hidden layer, output layer(refer fig 4 & fig 5) and each layer containing required number of neurons.

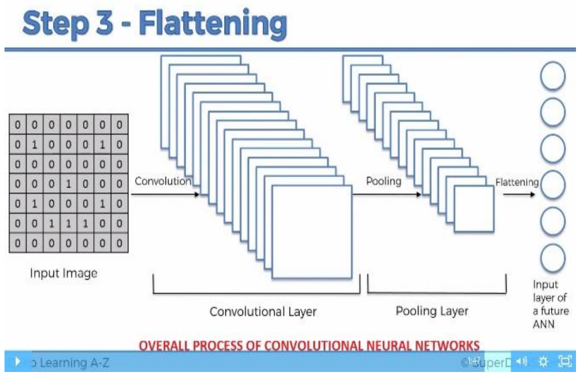


Fig 3 – The overall process of convolutional neural networks

Initially, for the first set of input, hidden layer after applying the activation function, there would be an output and this output is given as the input to the second set of input layer, hidden layer and output layer and the consequent outputs are passed to the next set of layers till the final output is achieved. The networks are termed as Recurrent neural networks as this step is carried out at each and every step for a series of input, hidden and output layers. To predict a word in a sentence, the network should remember the word of previous iteration. As the network is remembering the word of previous iteration it acts as a memory storage unit and as there are long series of input, hidden and output layers, this concept was also referred as Long short term memory networks or LSTM networks [9]. The various forms of Recurrent neural networks are 'one to many', 'many to one' and 'many to many'. The first form 'one to many' indicates if the input is 'one' image, the Recurrent neural network model produces 'many' narrations of the image. It means if the image contains various objects, then the model initially uses convolutional neural network and identifies all the objects of the image and then by using recurrent neural networks it gives the narration of all the objects as many words and hence it is 'one to many'. The second form 'many to one' indicates if there are 'many' words or sentences as the input, then the model uses purely recurrent neural networks and produces the final outcome as positive or negative feedback which is 'one' output and hence it is 'many to one'. And the third form 'many to many' indicates if there are 'many' words or sentences as input, then the model again uses recurrent neural network and translate into 'many' words or sentences as output and it is referred as 'many to many'. Here the concept of optimization is dealt in chapter 2, the optimizers Adagrad, RmsProp, ADAM and their algorithms are narrated in chapter 3, the Loss functions in chapter 4, drop out concept in chapter 5 and the total results are organized in chapter 6.

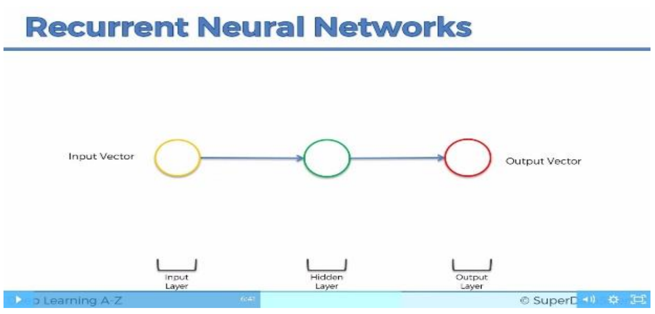


Fig 4 – The Input layer, hidden layer & output layer of Recurrent neural networks

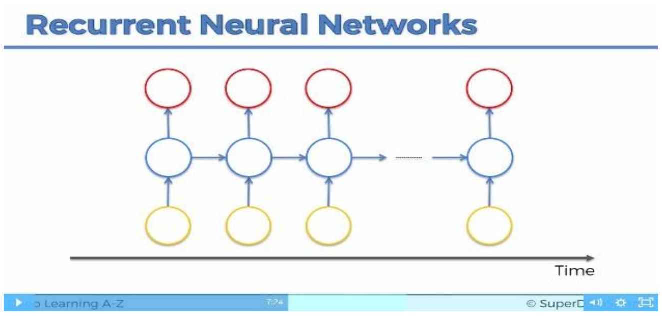


Fig 5 – The series of Input layer, hidden layer and output layer of Recurrent neural networks

II. OPTIMIZATIONIN DEEP LEARNING

The Gradient Descent [10][11][12] technique is one of the most popular algorithms to perform optimization for the neural networks. To comprehend, imagine a person standing on a top of a mountain and if he want to reach the bottom most point of the mountain with the condition that he is blind with no visibility. The Best way is to choose the surface which tends to descend as the first step. By following the similar strategy, he takes different steps to reach the bottom most point. Here, we need to find the best parameters for our learning algorithm and its corresponding cost. Here cost is nothing but the performance of the algorithm for different values of the parameters. The Gradient descent is having various variants such as Batch, Stochastic and mini Batch. The Batch Gradient descent uses the entire training data set and computes the gradient of the cost function with respect to the parameters. The Stochastic Gradient descent is the one which performs the parameter update for every training example. And the mini Batch takes the best of above two strategies by performing the parameter update for every mini batch of training examples. Initially the deep learning has used the basic gradient descent algorithms with which the models used to run with very less performance. In this paper, It is proposed to use few gradient descent optimization algorithms such as Adagrad, RmsProp and ADAM for both convolutional neural network and recurrent neural network models.

III. OPTIMIZERS

The first proposed optimizer is Adagrad. The Adagrad [13] is a Gradient descent optimization algorithm which adapts the learning rate. Here , the required parameters are Global learning rate α, mini batch size m, Initial weights θ_t, Small constant, δ, perhaps 10^{-7} , for numerical stability

```
1: Gradient accumulation variable is initialized r=0
2: while (condition !=stopping criterion) do
3: consider a mini batch of m examples from the training set
4: Compute gradient estimate $\nabla_\theta \sum_{i=1}^{m} J\left(\theta_t ;\, x^{(i)}, y^{(i)}\right)$
5: Accumulate squared gradient $r = r + (\nabla_\theta J$ □ $\nabla_\theta J)$
6: Compute update $\Delta\theta_{(t+1)} = \frac{\alpha}{\delta+\sqrt{r}}$□ $\nabla_\theta J$ (division and square root computed elementwise)
7: Apply update $\theta_{(t+1)} = \theta_t - \Delta\theta_{(t+1)}$
8: end while
```

Adagrad considers the low learning rates for the parameters occurring frequently and high learning rates for the parameters occurring infrequently. That's why it is said that it is well suitable for sparse data. Here, the mini batch of training examples are taken, compute the gradient, and square the Error (gradient), and then the weights are updated.

The second optimizer RmsProp [14] is an other optimizer algorithm which computes the learning rate with an exponential average of squared gradients. Here , the required parameters are Global learning rate α, Decay rate ρ, Mini batch size m, Initial weights θ_t , Small constant, δ, usually 10−6 , for numerical stability

```
1: Accumulation variable is initialized r=0
2: while (condition !=stopping criterion) do
3: consider a mini batch of m examples from the training set
4: Compute gradient estimate $\nabla_\theta \sum_{i=1}^{m} J\left(\theta_t ;\, x^{(i)}, y^{(i)}\right)$
5: Accumulate squared gradient $r = \rho r + (1-\rho)(\nabla_\Theta J$ □ $\nabla_\Theta J)$
6: Compute update $\Delta\theta_{(t+1)} = \frac{\alpha}{\delta+\sqrt{r}}$□ $\nabla_\theta J$ (division and square root computed elementwise)
7: Apply update $\theta_{(t+1)} = \theta_t - \Delta\theta_{(t+1)}$
8: end while
```

Here in RmsProp, the mini batch of training examples are taken, compute the gradient, and square the Error (gradient) with decay rate , and then the weights are updated.

The third optimizer ADAM[15] is one of the most efficient optimizer algorithm that computes the learning rate for each parameter. Here , the required parameters are Global learning rate α, Decay rates for moment estimates ρ1 and ρ2, Mini batch size m, Initial weights θ t, Small constant, δ, usually 10−8 , for numerical stability

```
1: Initialize 1st and 2nd moment variables r=0 and s=0
2: while (condition !=stopping criterion) do
3: consider a mini batch of m examples from the training set
4:Computegradient estimate$\nabla_\theta \sum_{i=1}^{m} J\left(\theta_t ;\, x^{(i)}, y^{(i)}\right)$
5: Update biased first moment estimate
    $s = \rho_1 s + (1 - \rho_1) \nabla_\theta J$
6: Update biased second moment estimate
    $r = \rho_2 r + (1 - \rho_2)(\nabla_\theta J$ □ $\nabla_\theta J)$
7: first moment correct bias: $\tilde{s} = \frac{s}{1-\rho_1^t}$
8: second moment correct bias: $\tilde{r} = \frac{r}{1-\rho_2^t}$
```

9: Compute update $\Delta\theta_{(t+1)} = \alpha \frac{\tilde{s}}{\delta+\sqrt{\tilde{r}}}$(division and square root computed elementwise)

10: Apply update $\theta_{(t+1)} = \theta_t - \Delta\theta_{(t+1)}$

11: end while

ADAM considers the exponentially decaying average of gradients (like momentum) and squared gradients and they are termed as first moment and the second moment respectively and hence the name Adaptive Moment (ADAM). The past gradients and squared gradients are computed and they are mainly biased towards zero. And, then the bias corrected first moment (gradient) and second moment (squared gradients) are computed and lastly the weights are updated accordingly.

IV. LOSSFUNCTIONS

It is proposed to use the popular Loss functions [16] such as mean squared error and binary cross entropy along with the above optimizers. The mean squared error computation is done in the following manner. For the Input layer, the Inputs or attributes of the first training example is taken, multiply with their weights, apply the activation function and get the output for the hidden layer. Again the same process is applied on this hidden layer neuron inputs and the final output is computed. Now the difference between the final output and the Actual value (class label of data set) is computed which is termed as the Error. Now, the Error is computed for all the training examples, square all the errors, and calculate the mean of the total error. This is the mean squared error with which the initial weights are updated and again the iterations starts from the input layer till the error converges to a minimum threshold. Let us suppose that $\widehat{x_i}$be the vector denoting n number of prediction values. Also, x_i be a vector denoting n number of true values, then the mean squared error MSE is given by

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\widehat{x_i} - x_i)^2$$

The other standard loss function is binary cross entropy. The binary cross entropy computes the difference between true and predicted probability distributions. The final output is calculated in the same way as it is done for the mean squared error and this it is termed as predicted distribution and the class label of the data set is termed as true distribution. Here, the partial derivatives are computed over the predicted and true distributions and the error is computed with the difference of both the distributions. The loss function most popularly used for classification is given by

$$J = -\frac{1}{m}\sum_{i=1}^{m} y_i \log\widehat{y_i} + (1 - y_i)\log(1 - \widehat{y_i})$$

When the activation function is sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$, Then the loss function is given by

$$J = \frac{1}{n}\sum_{x} x_j(\sigma(z) - y)$$

V. REGULARIZATION USING DROPOUT

It is proposed to use the recent fascinating concept Dropout for our work. Generally the neural network model suffers from one most common problems which is Overfitting. Overfitting means, model performs very well with the training data, but could not work well with the test data. To minimize Overfitting, Regularization process is applied on the model. Regularization is a technique of doing slight modifications to the existing model and the learning algorithm, so that it performs well both in training and testing. There are several Regularization techniques in Machine learning like L1 & L2 Regularization, Dropout, Data Augmentation, Early stopping etc. One of the Recent fascinating concept to perform Regularization in Deep Learning is Dropout [17][18]. The Dropout is carried in both the Training & Test phases. In the Training phase, a random fraction 'p' of nodes and their Activations are ignored for each hidden layer, for each training example and for each iteration. In the Test Phase, all the Activations are considered, but reduce them by a factor 'p' to account for the missing Activations during training phase. So, to understand, fig 6(a) represents a particular neural network model, then fig 6(b) represents the neural network model with drop out where few nodes were dropped with 'X' symbol.

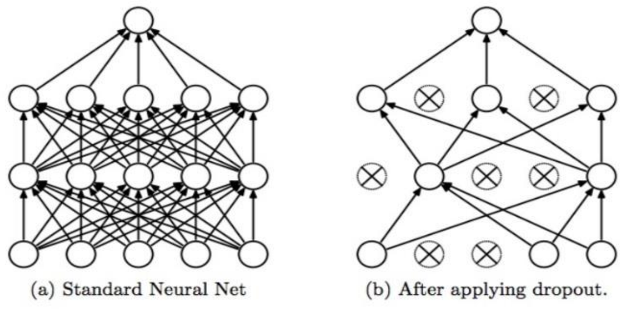


Fig 6 – The neural networks model (a) and the model after applying drop out(b)

VI. RESULTS

Implementation of optimizers

The experimentation is done in python on the CNN and RNN models with various optimizers, loss functions, drop out using the standard tensor flow, Theano and keras libraries.

Tensor Flow is an open source library which is used for high performance numerical computation. It has very flexible architecture which allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. It was developed by researchers and engineers from the Google Brain team within Google's AI organization,and it comes with strong support for machine learning, deep learning and the flexible numerical computation core is used across many other scientific domains.

Theano is a Python library which is used to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. Theano features:

Good integration with NumPy – Use numpy.ndarray in Theano-compiled functions.

Efficient use of a GPU – Perform data-intensive computations much faster than on a CPU.

symbolic differentiation – Theano performs derivatives for functions with one or many inputs.

speed& stability optimizations – Get the right answer for log(1+x) even for small values of x.

dynamic code generation of C – Evaluate expressions faster.

Deeper unit-testing and self-verification – Detect and diagnose various errors.

Keras is an efficient neural networks API, built in Python and capable of running on top of Tensor Flow, CNTK, or Theano. It was developed focusing on enabling fast experimentation. The ability from idea to result with the least possible delay is key to doing good research.

Keras may be used if you need a deep learning library that:

Allows easy and fast prototyping (because of user friendliness, modularity, and extensibility).

Works well with both convolutional networks and recurrent networks, as well as combinations of the two.

Runs extra ordinarily on CPU and GPU.

Convolutional neural networks –

Here the Adagrad, RmsProp, ADAM optimizers are applied on convolutional neural networks model and the corresponding Accuracy is verified and then the comparison is done among the three optimizers.

The Algorithm for the optimizer implementation on convolutional neural network model is

1. Initialize the Convolutional neural network model with the sequential classifier
2. Apply the process of convolution
3. Apply the process of pooling
4. Apply the process of flattening
5. The model is built with the Input and output layers
6. Compile the CNN model using an optimizer and loss function
7. The images are made to fit for the CNN
8. The training and test data sets were generated
9. Now, Run the CNN model with some epochs

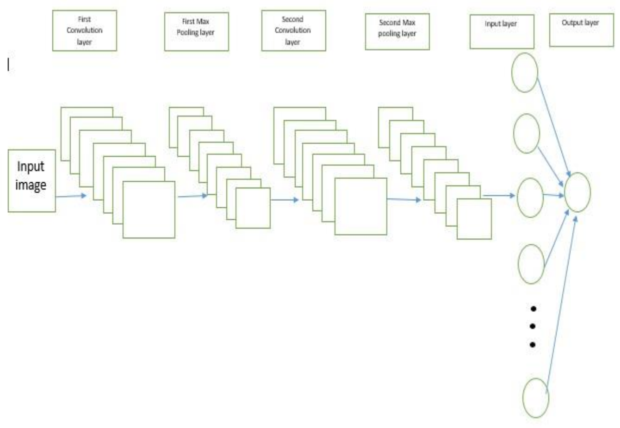


Fig 7 – The Convolutional neural networks model

The convolutional neural network model(refer fig 7) is developed, the various optimizers and loss functions are applied on the model and the Accuracy is verified. The CNN model comprises of Input image, which is passed to the first convolutional layer, then to the first max pooling layer. Then the data from this first max pooling layer is given to the second convolutional layer and then to the second max pooling layer. After this phase, the flattening process is applied and then the data is passed to the input layer of traditional artificial neural network and then to the output layer. Now, Let us elaborate the application of optimizers on convolutional neural networks model in a detailed way. The data set containing images of cats and dogs is considered and this data set is further divided into training and testing data sets. The input matrix is considered with 64x64 pixels and perform the convolution process by taking a filter of size 3x3. This convolution process is applied on the input matrix by taking various filters and produce a convolution layer by using the activation function RELU. Then the MAX Pooling of size 2x2 is applied on the convolution layer and produce the pooling layer. This pooling layer which is in two dimensional form is converted into one dimensional form which is referred as Flattening. This flattened data is given as inputs to the Input layer of Artificial neural networks. Then the input layer is designed as Dense layer with 128 neurons by using the activation function RELU and it is passed to the dense output layer comprising of 1 neuron and using the activation function Sigmoid. Then by adopting a particular optimizer, a loss function, the code is compiled by considering the needed metrics as Accuracy. Now, the data is generated from the images of training and testing data sets using a function ImageDataGenerator and the code is run for specific number of epochs. In this paper, the various optimizers Adagrad, RmsProp, Adam are applied on the convolutional neural networks model (refer table 2) and their corresponding Accuracy is verified.

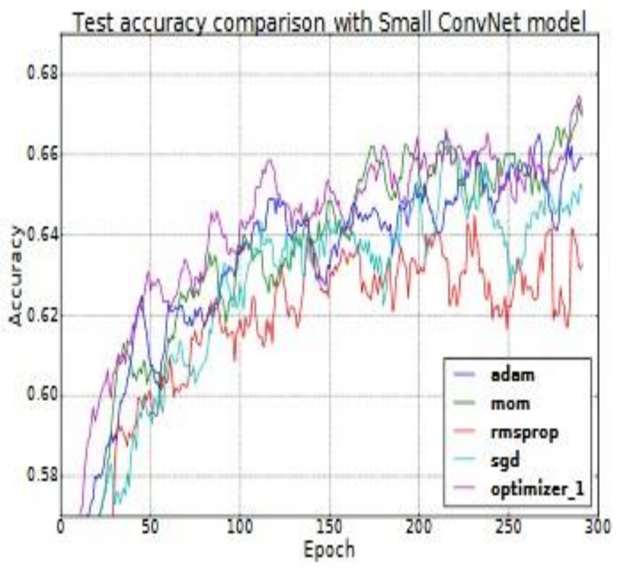
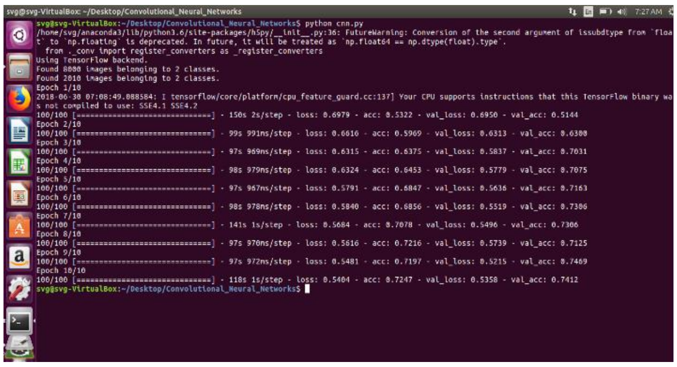


Fig 8 – The training Accuracy of convolutional neural networks model by Irwan Bello et al



```
svg@svg-VirtualBox: ~/Desktop/Convolutional_Neural_Networks
svg@svg-VirtualBox:~/Desktop/Convolutional_Neural_Networks$ python cnn.py
/home/svg/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of the second argument of issubdtype from `floa
t` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
Found 8000 images belonging to 2 classes.
Found 2010 images belonging to 2 classes.
Epoch 1/10
2018-06-30 07:08:49.088584: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU supports instructions that this TensorFlow binary wa
s not compiled to use: SSE4.1 SSE4.2
100/100 [==============================] - 150s 2s/step - loss: 0.6979 - acc: 0.5322 - val_loss: 0.6950 - val_acc: 0.5144
Epoch 2/10
100/100 [==============================] - 99s 991ms/step - loss: 0.6616 - acc: 0.5969 - val_loss: 0.6313 - val_acc: 0.6308
Epoch 3/10
100/100 [==============================] - 97s 969ms/step - loss: 0.6315 - acc: 0.6375 - val_loss: 0.5837 - val_acc: 0.7031
Epoch 4/10
100/100 [==============================] - 98s 979ms/step - loss: 0.6324 - acc: 0.6453 - val_loss: 0.5779 - val_acc: 0.7075
Epoch 5/10
100/100 [==============================] - 97s 967ms/step - loss: 0.5791 - acc: 0.6847 - val_loss: 0.5636 - val_acc: 0.7163
Epoch 6/10
100/100 [==============================] - 98s 978ms/step - loss: 0.5840 - acc: 0.6856 - val_loss: 0.5519 - val_acc: 0.7306
Epoch 7/10
100/100 [==============================] - 141s 1s/step - loss: 0.5684 - acc: 0.7078 - val_loss: 0.5496 - val_acc: 0.7306
Epoch 8/10
100/100 [==============================] - 97s 970ms/step - loss: 0.5616 - acc: 0.7216 - val_loss: 0.5739 - val_acc: 0.7125
Epoch 9/10
100/100 [==============================] - 97s 972ms/step - loss: 0.5481 - acc: 0.7197 - val_loss: 0.5215 - val_acc: 0.7469
Epoch 10/10
100/100 [==============================] - 118s 1s/step - loss: 0.5404 - acc: 0.7247 - val_loss: 0.5358 - val_acc: 0.7412
svg@svg-VirtualBox:~/Desktop/Convolutional_Neural_Networks$ 
```

Fig 9 – The training Accuracy of convolutional neural networks model by SVG Reddy et al

Table 2 – The Accuracy of convolutional neural networks using various optimizers

Optimizer	**Accuracy (Convolutional neural networks)**
Adagrad	69.75
RmsProp	50.75
Adam	74.12

Recently , Irwan Bello et al [19] has carried out the work on convolutional neural networks using Adam and various other optimizers and they achieved an Accuracy of nearly '68' during the training of data set(refer fig 8). And in this paper, the optimizer Adam was applied on convolutional neural networks model and the code has run with an Accuracy of 74.12 (refer fig 9). It is observed that while implementing convolutional neural networks, the optimizer Adam is running with maximum Accuracy (refer table 2) than the other optimizers Adagrad and RmsProp optimizers.

Recurrent Neural networks –

Here the Adagrad, RmsProp, ADAM optimizers are applied on Recurrent neural networks model and the corresponding Loss is verified and then the comparison is done among the three optimizers.

The Algorithm for the optimizer implementation on Recurrent neural network model is

1. Import the training data set
2. The data set is scaled and transformed to the values between 0 to 1.
3. Initialize the Recurrent neural network model with sequential as regressor
4. The model is built with Input layer, hidden layer and output layer
5. Compile the RNN model using an optimizer and loss function
6. Run the RNN model with some epochs

Now the predictions are done on the test data set and visualize the results.

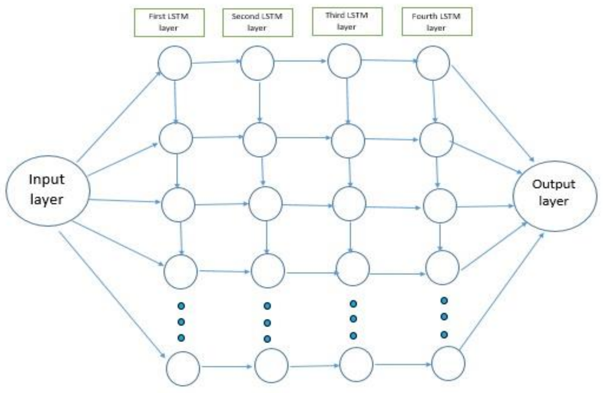


Fig 10 – The Recurrent neural networks model

The Recurrent neural network model(refer fig 10) is developed, the various optimizers and loss functions are applied on the model and the Loss is verified. The RNN model comprises of Input layer, then passed to the first LSTM layer, second, third and fourth LSTM layer and then to the output layer. Now, Let us elaborate the application of optimizers on Recurrent neural networks model in a detailed way. Here the data set is related to google stock prices wherein it contains the real stock prices and using this data the RNN model would predict the google stock price. Initially, the data set is loaded and all the data is scaled to values between 0 to 1 and generate the training and test data. The training data is processed for a batch of 60 records. Now, the RNN model is built with the sequential model as the Regressor with one input layer, three hidden layers and one output layer. The input and hidden layers consisting of 50 neurons at each layer which are referred as Long short term memory(LSTM) units and the output layer consisting of one neuron. Now, the RNN model training data is compiled with a particular optimizer, a loss function and it is run by fixing the number of epochs. After the code is run, from the test data the real stock price is retrieved and then compute the predicted price of the stock and the results are processed for visualization. So, the various optimization algorithms Adagrad, RmsProp and Adam are applied on the Recurrent neural networks model(refer table 3) and the corresponding Loss is verified.

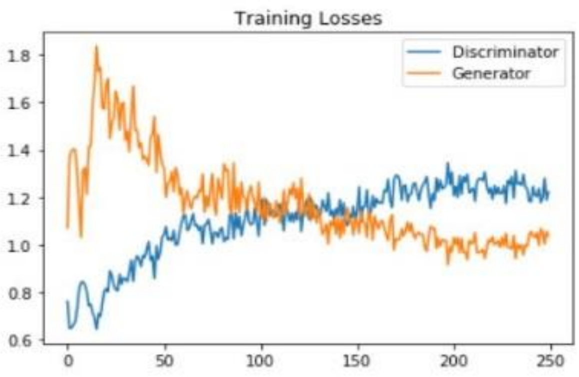
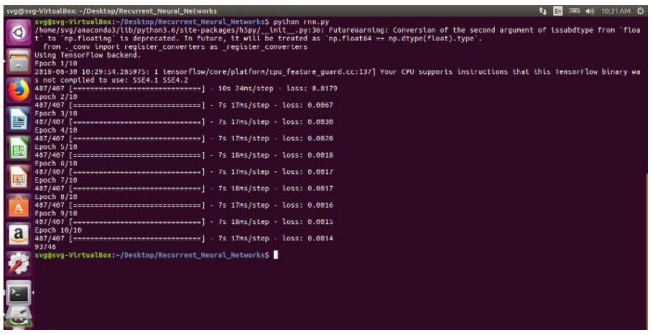


Fig 11 – The training Loss of Deep neural networks by Mohaksrivatsava et al



```
svg@svg-VirtualBox: ~/Desktop/Recurrent_Neural_Networks
svg@svg-VirtualBox:~/Desktop/Recurrent_Neural_Networks$ python rnn.py
/home/svg/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of the second argument of issubdtype from `floa
t` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
Epoch 1/10
2018-06-30 10:29:54.265975: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU supports instructions that this TensorFlow binary wa
s not compiled to use: SSE4.1 SSE4.2
487/487 [==============================] - 10s 24ms/step - loss: 0.0179
Epoch 2/10
487/487 [==============================] - 7s 17ms/step - loss: 0.0067
Epoch 3/10
487/487 [==============================] - 7s 17ms/step - loss: 0.0030
Epoch 4/10
487/487 [==============================] - 7s 17ms/step - loss: 0.0020
Epoch 5/10
487/487 [==============================] - 7s 18ms/step - loss: 0.0018
Epoch 6/10
487/487 [==============================] - 7s 17ms/step - loss: 0.0017
Epoch 7/10
487/487 [==============================] - 7s 18ms/step - loss: 0.0017
Epoch 8/10
487/487 [==============================] - 7s 17ms/step - loss: 0.0016
Epoch 9/10
487/487 [==============================] - 7s 18ms/step - loss: 0.0015
Epoch 10/10
487/487 [==============================] - 7s 17ms/step - loss: 0.0014
93746
svg@svg-VirtualBox:~/Desktop/Recurrent_Neural_Networks$
```

Fig 12 – The training Loss of Recurrent neural networks model by SVG Reddy et al

Table 3 – The Loss of Recurrent neural networks using various optimizers

Optimizer	**Loss (Recurrent neural networks)**
Adagrad	0.15
RmsProp	0.25
Adam	0.14

The Author Mohaksrivatsava et al [20] has carried out the work on deep neural networks using Adam and Adagrad optimizers and they achieved a loss of nearly '1.0' during the training of data set(refer fig 11). And in this paper, the optimizer Adam and the loss function mean squared error was applied on Recurrent neural networks and the code has run with minimum Loss '0.14' (refer fig 12). It is observed that while implementing recurrent neural networks, the optimizer Adam is running with minimum Loss than the other optimizers Adagrad and RmsProp(refer table 3).

Implementation of Loss functions

The popular Loss functions such as binary cross entropy, mean squared error are applied along with the optimizers Adagrad, RmsProp, ADAM on the convolutional neural networks model and the corresponding Accuracy is verified.

Table 4 – The Accuracy of convolutional neural networks model using various Loss functions

Loss function, Optimizer	**Accuracy (Convolutional neural networks model)**
Binary cross entropy, Adam	74.12
Mean squared error, Adam	72.12
Mean squared error, RmsProp	76.62

It is observed that while implementing convolutional neural networks model, the Loss function Mean squared error along with optimizer RmsProp has given more Accuracy of 76.62 than the other loss functions and optimizers (refer table 4).

And the Loss functions binary cross entropy, mean squared error are applied along with the optimizers Adagrad, RmsProp, ADAM on the Recurrent neural networks model and the corresponding Loss is verified.

Table 5 – The Loss of Recurrent neural networks model using various Loss functions

Loss function, Optimizer	**Loss (Recurrent neural networks)**
Mean squared error, Adam	0.14
Binary cross entropy, Adam	43.89
Binary cross entropy, Adagrad	43.89

It is observed that the loss function mean squared error and the optimizer Adam was applied on Recurrent neural networks model and the code has run with minimum Loss '0.14' compared to other loss functions and optimizers(refer table 5).

Implementation using Dropout

The Dropout concept is applied on the Convolutional neural networks model and the Accuracy is verified for different values of Drop put parameter 'p'. The 'p' values 0.1, 0.2, 0.3 are added and applied to the sequential model of the CNN and run the model.

It is observed that the Accuracy of 70 is maximum for the DropOut 'p' value of '0.1' compared to 'p' value of '0.2', '0.3' for the Convolutional neural networks model(refer table 6).

Table 6 – The Accuracy of convolutional neural networks model using Dropout

DropOut 'p' value	**Accuracy (Convolutional neural networks model)**
0.1	70.00
0.2	66.87
0.3	46.88

And the same Dropout is applied on the Recurrent neural networks model and the Loss is verified for different values of Drop put parameter 'p'. The 'p' values 0.1, 0.2, 0.3 are added and applied to the sequential model of the RNN and run the model.

Table 7 – The Loss of Recurrent neural networks using Dropout

DropOut 'p' value	**Loss (Recurrent neural networks model)**
0.1	0.12
0.2	0.14
0.3	0.16

Similarly it is observed that the Loss of 0.12 is minimum at a DropOut ‘p’ value of ‘0.1’ for the Recurrent neural networks model. It tells that ignoring the nodes should be very minimum in order to perform the Regularization which produces more Accuracy or minimum Loss (refer table 7).

VII. CONCLUSION

The two Deep Learning algorithms such as Convolutional neural networks and Recurrent neural networks are implemented using optimizers (Adagrad, RmsProp, ADAM), loss functions(Mean squared error, binary cross entropy) and Dropout concept. Using the Adam Optimizer, maximum Accuracy of 74.12 is achieved with the Convolutional neural networks model and it has run with minimum loss of 0.14 with the Recurrent neural networks model. Using the Mean squared error loss function and RmsProp optimizer, maximum Accuracy is observed for the convolutional neural networks model and by using the Mean squared error loss function and Adam optimizer the algorithm is run with the minimum Loss for the Recurrent neural networks model. While performing the Regularization, the Dropout is applied and it is observed that the maximum Accuracy is achieved for the Convolutional neural networks model with a Drop out parameter ‘p’ value of ‘0.1’ and with this same value of Dropout parameter ‘p’ value of ‘0.1’, the Recurrent neural networks model has run with the minimum loss. As a future work, the above two Deep Learning algorithms CNN and RNN may be implemented using various fuzzy functions, and by initializing the weights mathematically instead of Random weights.

ACKNOWLEDGEMENT

I am very much thankful to my Supervisor Prof. K.Thammi Reddy, Co Supervisor Prof. V.ValliKumari for their guidance and constant monitoring of my PhD work and heartfelt gratitude to my parents and the Almighty.

REFERENCES

1. B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, And A. Torralba, “Learning Deep Features For Discriminative Localization,” In Proceedings Of The Ieee Conference On Computer Vision And Pattern Recognition , 2016, Pp. 2921–2929
2. Https://Www.Superdatascience.Com/Deeplearning/
3. https://bhatsnotes.com/2016/12/23/artificial-intelligence-t-hub/
4. M. D. Zeiler And R. Fergus, “Visualizing And Understanding Convolutional Networks,” In European Conference On Computer Vision Springer, 2014, Pp. 818–833
5. A.Negi, C.Bhagvati, B.Krishna, An OCR system for Telugu, IEEE Proceedings of Sixth International conference on Document Analysis and Recognition, DOI: 10.1109/ICDAR.2001.953958
6. Ruben Tolosana, Ruben Vera-Rodriguez, Julian Fierrez, (Member, Ieee), And Javier Ortega-Garcia, (Fellow, Ieee), “ Exploring Recurrent Neural Networks For On-Line Handwritten Signature Biometrics”, Ieee Access, Volume 6, P 5128-5138, 2018,
7. A.Graves, A. R. Mohamed, and G. Hinton, ‘‘Towards end-to-end speech recognition with recurrent neural networks,’’ in Proc. Int. Conf. Mach. Learn. , vol. 14. 2014, pp. 1764–1772.
8. S.Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, ‘‘Gradient flow in recurrent nets: The difficulty of learning long-term dependencies,’’ in A Field Guide to Dynamical Recurrent Networks, S. C. Kremer and J. F. Kolen, Eds. 2001.
9. S. Hochreiter and J. Schmidhuber, ‘‘Long short-term memory,’’ Neural Comput. , vol. 9, no. 8, pp. 1735–1780, 1997.
10. Sebastian Ruder, “An Overview Of Gradient Descent Algorithms”, Cornell University Library, Arxiv: 1609.04747[Cs. Lg]
11. AnirbanSarkar, AdityaChattopadhyay, PrantikHowlader, V. Balasubramanian, Grad-Cam++: “Generalized Gradient-Based Visual Explanations For Deep Convolutional Networks”, Proceedings Of Ieee Winter Conference On Applications Of Computer Vision (Wacv'18), Mar 2018. [Arxiv]
12. R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, And D. Batra, “Grad-Cam: Why Did You Say That? Visual Explanations From Deep Networks Via Gradient-Based Localization,” ArxivPreprint Arxiv:1610.02391 , 2016.
13. AsmelashTekaHadgu; Aastha Nigam ; Ernesto Diaz-Aviles, “Large-scale learning with Adagrad on Spark, 2015 IEEE International Conference on Big Data (Big Data), DOI: 10.1109/BigData.2015.7364091
14. Mahesh Chandra Mukkamala, Matthias Hein , “Variants of RmsProp and Adagrad with Logarithmic Regret Bounds”, Proceedings of the International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017, arXiv:1706.05507v2[cs.LG]
15. Zijun Zhang, “Improved Adam Optimizer for Deep Neural Networks”, 978-1-5386-2542-2/18/ ©2018 IEEE
16. KatarzynaJanocha, Wojciech Marian Czarnecki, “On Loss Functions for Deep Neural Networks in Classification”, Theoretical foundations of machine learning, Vol. 25 (2016): 49–59 doi: 10.4467/20838476SI.16.004.6185
17. NitishSrivastava, Geoffrey Hinton, Alex Krizhevsky, IlyaSutskever, RuslanSalakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, Journal of Machine Learning Research 15 (2014), 1929-1958
18. Vu Pham ; ThéodoreBluche ; Christopher Kermorvant ; JérômeLouradour, “Dropout Improves Recurrent Neural Networks for Handwriting Recognition”, 2014 IEEE 14th International Conference on Frontiers in Handwriting Recognition, DOI: 10.1109/ICFHR.2014.55
19. Irwan Bello, BarretZoph, vijayvasudevan, QuocV.Le, “Neural optimizer search with Reinforcement learning”, Proceedings of the 34th International Conference on Machine Learning , Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).
20. Mohaksrivatsava, s.pallavi, srijita Chandra, G.Geetha, “ Comparison of optimizers implemented in generative adversarial network(GAN)”, International journal of Pure and Applied mathematics, vol. 119, no 12, 2018.