Assignment Report
Designing a Neural Network for Classification

Under the guidance of
Dr. Basabdatta Bhattacharya

Prepared by

Names of the Students	**ID Numbers**
Arkil Patel	2016A7PS0665G
Mehul Kasliwal	2016A7PS0043G
Saurav Pradhan	2016A7PS0061G
Shriya T P	2016A7PS0060G
Kevin Boban	2016A7PS0185G

All members contributed equally in this assignment.

Prepared in partial fulfillment of the
Course No. BITS F312
Course Title: Neural Networks and Fuzzy Logic

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE - PILANI
(FEBRUARY, 2019)

Acknowledgements

We are extremely pleased to acknowledge the Neural Networks Course Instructor-In-Charge, Dr. Basabdatta Bhattacharya, for imparting us with the knowledge to carry out this assignment successfully and for her invaluable guidance during the course of this assignment.

We are grateful to the other groups for exchanging necessary information related to the assignment.

24, Feb 2019

Arkil Patel
Mehul Kasliwal
Saurav Pradhan
Shriya T P
Kevin Boban

TABLE OF CONTENTS

1. INTRODUCTION

This report intends to give a constructive criticism of the research paper titled "Construction of Feed Forward Multilayer Perceptron Model for Genetic Dataset in Leishmaniasis Using Cognitive Computing" [1] by looking into the research methodology used and analyzing the Artificial Neural Network (ANN) model created for the purpose. The main motive of the paper is to present a new approach for diagnosing Leishmaniasis using cognitive computing. We also go on to make our own model from scratch using various preprocessing methods and following standard deep learning practices. Then, we test both models on another medical dataset to test how well they generalize. A comparative analysis of the performances of the models is also done, using accuracy as the performance metric.

2. CONSTRUCTIVE CRITICISM OF THE PAPER [1]

2.1 Strengths

The feed forward Artificial Neural Network (ANN) that is chosen in the paper is an excellent model that can suitably learn from the input data, based on the features provided to the multilayer perceptron. They are well suited for large datasets as they address the problem of identifying the right set of features for a task. Moreover, they can actually encode features across problem domains. For e.g., we can train the lower layers of the network on one dataset and for the next dataset, we won't have to start from scratch. This feature of the ANN works out particularly well in this case as they have combined around 33 different datasets for the analysis, as stated in the dataset section on page 5.

Moreover, they have applied a back-propagation algorithm for the supervised classification of the genetic datasets, which allows the network to propagate the errors of the output layer to the previous layers and suitably update the weights of all the layers. Owing to this, the model is able to learn the appropriate features from data and perform well on the 20% testing data that they have considered in the paper. The use of the gradient descent optimization along with the back propagation helps in minimizing the error further.

As indicated in the GitHub code repository, the cross-entropy loss function is applied to the model which is another favourable choice, because using a cross entropy function leads to a convex optimization problem instead of the non-convex optimization problem in the case of a mean square error function. In case of a non-convex problem, we might end up at the local minima whereas in the case of a convex optimization coupled with the gradient descent procedure, the network reaches the optimum solution.

2.2 Weaknesses

One of the major weaknesses of ANNs is that they require huge amounts of data to train i.e. a neural net is usually used with around 10,000 training samples. Added to this, they are computationally very expensive to train. It is only the advancements in Graphic Processing Units (GPU) that has made the training of neural networks viable. However, in the paper there is no mention of the use of a GPU or other similar alternatives to train the network. So, it is possible that the network hasn't been trained enough for the weights to converge optimally.

The use of ANNs implies a lot of adjustable parameters and hyperparameters. The topology of a neural network is also quite indefinite which is usually decided based on random trial and error, experience and intuition. The work in the paper doesn't highlight much of the hyperparameter tuning performed or the variations observed while doing so. Detailed experimentation should have been carried out while changing these parameters as it could yield valuable insights into the plausible causes for the disease.

Moreover, they could have adopted a batch learning strategy to parallelize the training of the network which could speed up the procedure to a large extent. The use of the sigmoid instead of ReLU activation function for the hidden layers of the network, as specified in the methodology section on page 3, is not a good choice. Instead, a ReLU activation does not suffer from the vanishing gradient problem and is also way more computationally efficient to calculate, as it performs a max function instead of complex exponential function as in sigmoid. Further, networks with ReLU tend to show better convergence performance than sigmoid.

2.3 Discrepancies

1. As stated in the introduction section on page 3, they have used an RNN during the pre-processing part. However, RNNs are used for variable length sequential data whereas the genetic dataset being used here has fixed length time independent samples and hence the use of recurrent neural networks doesn't seem to be justified.

2. The codebase of their model uses TensorFlow's cross entropy loss function i.e. log loss which measures the performance of a classification model whose output is a probability between 0 and 1. This cross entropy increases as the predicted probability diverges from the actual label. However, on page 9 it states that the loss function used in the paper is the mean square error function.

3. In the paper, they have claimed that they are increasing the number of neurons in the hidden layer to increase accuracy, however throughout the model the number of the neurons in the hidden layer remains constant.

4. In the dataset section on page 5, they claim to have combined 33 genetic datasets and have stated “It comprises of 5 healthy person datasets and 28 infected human datasets”, while on their GitHub repository, it states “166 datasets comprising 38 healthy person datasets and 78 infected datasets”. Apart from this mismatch, their final dataset consists of 33 rows a.k.a data samples and 45,000 columns as features of the data. However, it clearly violates the common practice of having more samples than features as a pre-emption for a good classification task.

2.4 Mismatch between claim and results

As the dataset they have used consists of 28 infected and 5 non-infected samples, it leads to a highly imbalanced dataset. For this case, they have claimed an accuracy of 85.71% and no other data analysis metrics are used. However, for a binary classification of an imbalanced dataset, it is highly possible to just predict one class as the output for all samples and still achieve an accuracy of around 90%.

Moreover, they have defined the precision and recall metrics in the “neural network parameters” section on page 7, whereas their results are just indicative of the accuracy metrics which is not a good indicator for imbalanced datasets. Instead they should have added the precision and recall scores for each epoch in the results section, so as to provide a strong basis to support their claim.

2.5 Quality of figures

The quality of figures are subpar as inbuilt libraries like matplotlib, plotly should have been used for generating better images. Further, appropriate labelling for the axes and suitable legends should have been specified. Moreover, the Figure.2. on page 11 shows that the input dimension of an individual sample is 45, however the actual value should be 45031 (number of features of an input vector).

2.6 Quality of tables

The table on page 10 is of appreciable standards as the variation in accuracy with the iteration is clearly depicted in the table. Furthermore, the change in loss and mean square error is also shown in the table.

2.7 Clarity of writing and presentation

The major workflow of the paper is clear and does not deviate. However, there is a lack of clarity in the pre-processing aspect of the paper as the way they have combined the datasets to create a larger input to the network is not at all clear. There are some shortcomings in the methodology section as well, mainly a comprehensible explanation of the working of the artificial neural network should have been presented.

There are also quite a lot of grammatical and syntactical errors throughout the paper. Another conspicuous error exists on page 6, where the title says "RNN" whereas the body of the section entirely describes the working of a multilayer perceptron and the specific details of this artificial neural network. Clearly, this is a major mistake that has been overlooked.

2.8 Further suggestions for improvement

They have used a gradient descent optimizer, whereas an Adam optimizer would have been a better choice. Foremost because the Adam optimizer uses moving averages of the parameters which allows it to use a larger effective step size and the algorithm converges to this step size without fine tuning. Hence, a gradient optimizer is suitable only when sufficient hyperparameter tuning is performed for the convergence of the model, which is not carried out in this work.

3. Description of our Neural Network

3.1 Description of the Functions

- def parameter_initialization(dimensions):
  This is a helper function which takes in input a list (dimensions) which contains information about the layers (both hidden, input and output layer included) present in the neural network and returns a parameters dictionary with the requisite size of the parameters in the given layer using a randomizer function. Each layer in the parameter dictionary is represented as Wn and bn where n indicates the nth layer of the network.

- def sigmoid(Z):
  We have coded sigmoid function which performs the following
  $$(Z) = (WA + b) = \frac{1}{(1 + e - (WA + b))}$$
  operation on the individual elements of the matrix Z.

- def relu(Z):
  This is RELU function performing the following operation
  $$A = RELU\,(Z) = max\,(0, Z)$$
  In the later sections, we have run the code using both relu and sigmoid and have drawn a comparative study of both the methods.

- def forward_prop_lin(A, W, b):

  With the parameters in place, we now perform forward propagation. First, we start of by defining forward_prop_lin module which computes the following equations:

  $$Z[l] = W[l]A[l-1] + b[l]$$

  Now with the linear forward propagation model in place, we next code up the two helper functions forward_prop_activation and forward_prop_full

- def forward_prop_activation(A_prev, W, b, activation):

  Takes in input activations of the previous layer, weights and bias and applies the requisite activation function i.e. sigmoid or ReLU activator and returns the Activation for the next layer. Also, in the process we store these values in temp_backprops because they'd be required for future back propagations.

- def forward_prop_full(X, parameters):

  This is the main function. This function sums up the entire forward pass for a training example in the dataset taking in X and parameters for the entire network

- def cost(AL, Y):

  Before coding up backpropagation we need to define some cost function to check if our model is actually learning for this purpose, we compute the cross-entropy cost J, using the following formula:

  $$-\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}\log\left(a^{[L](i)}\right) + (1-y^{(i)})\log\left(1-a^{[L](i)}\right))$$

  Now that we have calculated the cost function, using cross entropy losses. We proceed with the backpropagation of the layers, with the updation of the parameters.

- def backward_prop_lin(dZ,temp_backprop):

  Similar to forward propagation we start of by defining backward_prop_lin module which computes the following equations. Assuming that we have already calculated dZ.

  $$dW^{[l]} = \frac{\partial \mathcal{L}}{\partial W^{[l]}} = \frac{1}{m} dZ^{[l]} A^{[l-1]T}$$

  $$db^{[l]} = \frac{\partial \mathcal{L}}{\partial b^{[l]}} = \frac{1}{m}\sum_{i=1}^{m} dZ^{[l](i)}$$

  $$dA^{[l-1]} = \frac{\partial \mathcal{L}}{\partial A^{[l-1]}} = W^{[l]T} dZ^{[l]}$$

- def derivative_relu(dA,temp_backprop):
  This function calculates the derivative wrt relu

- def derivative_sigmoid(dA,temp_backprop):
  This function calculates the derivative wrt sigmoid

If f(.) is the activation function, derivative_sigmoid and derivative_relu compute:

$$dZ[l] = dA[l] * f'(Z[l])$$

Similar to the Linear forward propagation module, we now code up the two helper functions backward_prop_activation and backward_prop_full.

- def backward_prop_activation(dA,temp_backprop, activation):
  This function takes in input change in activations of the previous layer, temp_backprop and activation and computes the derivative of the weights, biases and activation of previous layer.

- def backward_prop_full(AL, Y, list_temp):
  This is the main function. This function sums up the entire backward pass for a training example in the dataset taking in X and parameters for the entire network.

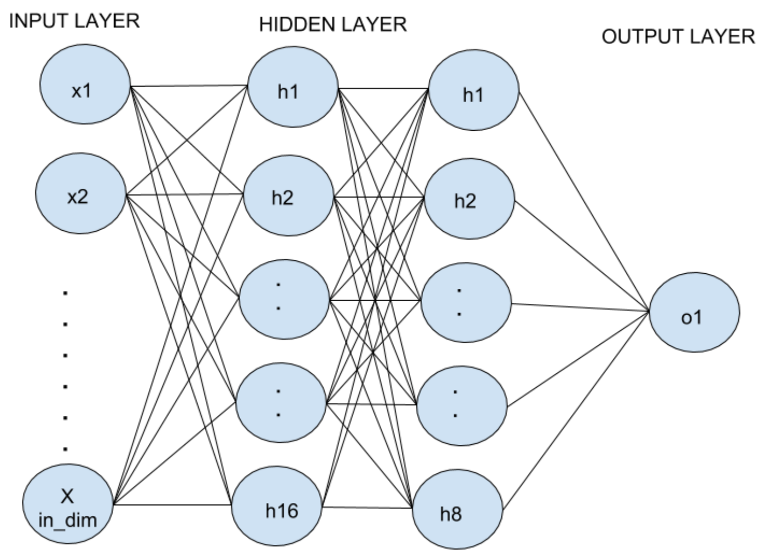
  With one pass of forward and backward completed, we conclude the pass by updating the parameters of each individual layer.

- def params_update(params, gradients, learning_rate):
  This function updates the parameters.

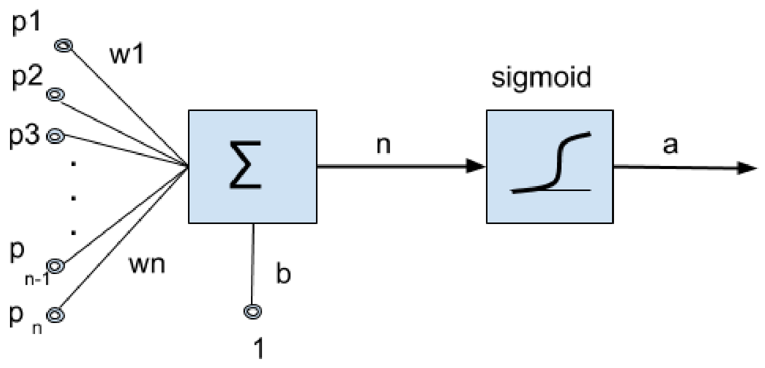
- def neural_network_batch(X, Y, dimensions, learning_rate = 0.1, num_iterations = 3000, print_cost=False,batch_size=32):
  Having coded all the pre components of the Network, finally we assemble all of them to form a full-fledged network. This function provides the implementation for the batch modeling of the Network. In this, the batch_size is set by the user. The errors are updated after the processing of a batch instead of the whole data.

- def predict(X, y, parameters):
  After training the model, this method is used to predict the results.

3.2 Structure of Neural Network



A zoomed in version of a neuron in the output layer of our model is:



(Note: The hidden layers use ReLU activation while the output layer uses sigmoid)

3.3 Description of Dataset:

Breast Cancer Diagnostic Dataset: https://www.openml.org/d/15

The dataset that we have chosen is a binary classification dataset for classifying whether the values of certain bio-markers indicate a malignant or benign case of breast cancer. It has 698 examples and 9 features (excluding class).

The primary reason we chose this dataset is because of the excellent bio-markers like chromatin count, epithelial cell size, occurence of bare and normal nuclei etc which serve as features for the binary prediction task. This is one of the only datasets available online that classifies solely on the basis of these biomarkers. Also, the dataset has 9 features which is low enough to ensure that our ReLU-based model is able to train on it without exploding the gradients and encountering NaNs. The number of examples is also moderate. It is low enough that the model can be trained within seconds and high enough to ensure the model does not underfit.

4. Results

We trained and tested our model on both datasets – the one used by the authors and the one selected by us (breast cancer). We also trained and tested their model on both these datasets.

TRAINING ACCURACY

	Paper Dataset	**Our Dataset**
Paper Model	84.6%	66.8%
Our Model	84.6%	97.6%

TEST ACCURACY

	Paper Dataset	**Our Dataset**
Paper Model	85.7%	57.7%
Our Model	85.7%	98.5%

Corresponding graphs summarizing our results and observations are included in the ipython notebooks.

5. Observations and Conclusions

1. Firstly, it was observed that among the 45032 features in the dataset used by the authors, around 19212 were filled with only zeros and contributed no new data to the model. These features were discarded before we ran the dataset on our model. Hence, dimensionality reduction was applied.
2. The given dataset has all positive examples bunched together and then all the negative examples together. That is, the data here is sorted by their target class. To make the training and testing sets more representative of the overall distribution, we shuffled the data. Shuffling the data also serves the purpose of reducing the variance and making sure that the learning done by the model remains general and doesn't overfit on the data.

3. For the hidden layers, their model used 45 neurons per layer. In our model we preferred to use 16 and 8 units (powers of 2) as it has been suggested that keeping the number of hidden units as a power of 2 helps the model to converge faster resulting in the neural network learning faster. [6]
4. We have made two Multilayer Perceptron (MLP) models, one with activation function as sigmoid for hidden layers, and the other with activation function as ReLU (Rectified Linear Unit) for hidden layers. The activation function for final layer used for classification is sigmoid in both the models.
5. It was observed that the model with ReLU activation in the hidden layers marginally outperforms the one with sigmoid activation on small to moderate datasets. This is because the sigmoid function starts to saturate for values far from zero resulting in lesser learning, however the ReLU function keeps increasing steadily resulting in faster learning.
6. For large datasets (i.e. datasets with larger number of attributes such as the one used by the authors of the paper), it was observed that the model with ReLU activation for the hidden layers yields NaN values. This is because with large data and multiple layers, gradients tend to explode since ReLU increases monotonically, leading to NaN values. However, sigmoid activation function becomes saturated after a particular value and always generates a value in the interval (0, 1). This problem of encountering NaNs is happening because we have written a raw implementation of the neural network and have no optimised libraries to handle unstable gradients. In practice, however, ReLU is preferred over sigmoid because the dense representations generated by sigmoid activation are unfavourable due to high computation costs while the sparse representations generated by ReLU can, on the other hand, be subjected to various optimizations.
7. In their model, the authors have fixed the number of epochs. It is evident from the plot of their loss function that they had achieved the best model much before the last few epochs, hence resulting in overfitting. To avoid this we use early stopping, which means stopping prematurely before the model starts to overfit. We closely monitor the training loss and stop when we see that the loss does not decrease after some number of epochs. [7]
8. Upon running our model on the "Breast Cancer Diagnostic Dataset", we obtained an accuracy of around 98.5%, which is pretty decent considering that the model was coded from scratch without using any ML libraries with their inherent optimizations. On modifying their model slightly such that the input layer has dimension equal to the number of features on the dataset, their model generated an accuracy of 57.7% on the same dataset (breast cancer), which suggests that their model does not generalize well.
9. On running our model on the dataset used in the paper, we were able to replicate their results to a great extent. Our model also obtained an accuracy of 85.7%. However, since the dataset used in the paper is extremely small and imbalanced (there are only 33 examples), we strongly suspect that the model might have overfit on the training data.

REFERENCES

1. Sundar Mahalingam, Ritika Kabra, and Shailza Singh: "Construction of Feed Forward MultiLayer Perceptron Model For Genetic Dataset in Leishmaniasis Using Cognitive Computing", www.biorxiv.org.
2. https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0
3. https://stackoverflow.com/questions/45616191/machine-learning-using-relu-return-nan
4. https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw
5. https://towardsdatascience.com/dimensionality-reduction-does-pca-really-improve-classification-outcome-6e9ba21f0a32
6. https://ai.stackexchange.com/questions/5399/why-number-of-hidden-units-in-a-layer-are-suggested-to-be-in-powers-of-2
7. https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/
8. The breast cancer dataset was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg.
9. O. L. Mangasarian and W. H. Wolberg: "Cancer diagnosis via linear programming", SIAM News, Volume 23, Number 5, September 1990, pp 1 & 18.
10. Neural Networks and Deep Learning course on Coursera by Prof Andrew Ng.
11. Backpropagation videos on youtube by deeplizard – https://www.youtube.com/watch?v=XE3krf3CQls&feature=youtu.be
12. Haykin, S. (1994). *Neural networks* (Vol. 2). New York: Prentice hall.
13. Hagan, M. T., Demuth, H. B., Beale, M. H., & De Jesús, O. (1996). *Neural network design* (Vol. 20). Boston: Pws Pub..