Numerical Correspondence Using Shallow Neural Networks and Support Vector Machines

Frank Cally A. Tabuco*
University of the Philippines Diliman
789 Marcville Subd., Brgy. Parian
Calamba City, Laguna 4027
fatabuco@up.edu.ph

ABSTRACT

Since the regained popularity of Artificial Neural Network, several developments improved the classification performance of this algorithm. Techniques such as generalized back-propagation, multi-layer neurons, multiple hidden layers, and batch normalization have helped increase accuracy and helped various fields in engineering, image classification, and biology. In this paper, a shallow neural network will be constructed from scratch employing techniques such as back-propagation, normalization, and minimum average error to properly classify numerical label correspondence given a set of input features. This algorithm will then be compared to the best performing kernel trick SVM algorithm for the given dataset. SVM yielded lower training runtime without sacrificing much accuracy whereas ANN took longer traning time but provided a slight performance advantage.

Categories and Subject Descriptors

•**Computing methodologies** → **Machine learning;** *machine learning approaches;* Neural networks. *Kernel Methods;* Support Vector Machines

General Terms

Neural network, support vector machines

Keywords

Shallow networks, polynomial kernels, learning rate, minimum average error

1. INTRODUCTION

Research on Artificial Neural Networks (ANN) are primarily guided by biological characteristics and functions of a human's neuron. Neural networks utilizes information gathered from an environment (inputs) and are adjusted using synaptic weights to arrive at a particular output. These neural networks can learn the correct outputs through supervised learning, unsupervised learning, and reinforcement learning. In this paper, the focus of the experiments will be supervised learning where a labeled dataset guides the incorrect classification of the neural network to a desired output. Formally, ANN is defined as:

> *A massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two aspects:*
>
> *1. Knowledge is acquired by the network through a learning process*
> *2. Inter-Neuron connection strengths, known as synaptic weights, are used to store the knowledge.*[3]

On the topic of supervised learning, perhaps the most popular algorithm to use is Support Vector Machines (SVM). While ANN provides simplicity in construction and demonstrates a marginally better peformance in classification tasks, SVM employs various kernel tricks and optimization hyperplanes which maximizes the separation between labels of the dataset.[3]. SVM is generally believed to be a more powerful non-linear classifer compared to ANN. However, recent developments in ANN algorithms such as multilayer perceptrons, generalized backpropagation, and dropouts inevitably outperformed SVM even with a refined feature selection.[2]

A shallow ANN will be constructed to predict the numerical label correspondence of unknown data given a training dataset composed of 354 features. Along with this, SVM will be employed using polynomial kernels of degree 22 to compare the performance and runtimes of both algorithms. The rest of the paper will proceed as follows, section 2 will discuss the methodology of constructing the dataset and handling of imbalanced datasets. Section 3 will show the results of the experiments and discusses the performance measures for both algorithms. Lastly, section 4 will provide guidance for future research along with some concluding remarks.

2. METHODOLOGY

2.1 Dataset Creation and Handling of Imbalanced Datasets

There are three files provided for training and predicting the outputs of ANN and SVM. File *data.csv* contains 3486

*A new member of Computer Vision and Machine Intelligence Groupd of the Department of computer science in UP Diliman

ACM ISBN XXXXXXXX.
DOI: XXXXXXX

```
In [4]: #Checking the imbalance in data
        Counter(data_labels)

Out[4]: Counter({8: 466, 5: 287, 1: 1625, 6: 310, 4: 483, 2: 233, 3: 30, 7: 52})
```

Figure 1: Number of instances per label.

```
In [5]: #Oversampling and undersampling using SMOTE
        oversampling = SMOTE(sampling_strategy={8.0: 1400,
                 5.0: 1400,
                 6.0: 1400,
                 4.0: 1400,
                 2.0: 1400,
                 3.0: 1400,
                 7.0: 1400})
        undersampling = RandomUnderSampler(sampling_strategy={
                 1.0: 1400,})
        steps = [('over',oversampling),('u',undersampling)]
        pipeline = Pipeline(steps=steps)
        dataset, data_labels = pipeline.fit_resample(dataset, data_labels)

In [6]: #Checking the oversampled and undersampled data labels
        Counter(data_labels)

Out[6]: Counter({1: 1400,
                 2: 1400,
                 3: 1400,
                 4: 1400,
                 5: 1400,
                 6: 1400,
                 7: 1400,
                 8: 1400})
```

Figure 2: Code for oversampling and undersampling.

input instances each with 354 features while *data_labels.csv* consists of class labels (1,2,...,8) for each of the 3486 input instances. Both of these files are used to form the training, validation, and testing sets for the algorithms. The third file, *unknown_labels.csv*, is composed of 700 unlabeled instances with the same size of features. This is used by ANN and SVM to predict the unknown labels after training and testing.

After mounting the two data files, it can be seen in Figure 1 that the dataset is imbalanced. Therefore, instances for each labels are skewed more one side as compared to the rest of the labels. Training on an imbalanced dataset will lead to poor classification performance for the minority. With this, a Synthetic Minority Oversampling Technique (SMOTE) is employed to oversample the minority labels and undersample the majority labels.[1] Oversampling entails creating duplicates of minority labels based on a preset value. On the other hand, undersampling refers to randomly reducing some instances within the majority set to match the new number of instances from the minority. For the experiment, minority labels are oversampled to contain 1400 instances each and the majority label is randomly undersampled to 1400 instances, also. The imblearn library is used to handle this procedure (see Figure 2).

After transforming the imbalanced dataset, training, validation, and testing sets are created with instances equal to 60%, 20%, and 20% of the dataset, respectively. It should be noted here that after using oversampling and undersampling the dataset will be ordered based on increasing value of the labels. Therefore, shuffling is necessary prior to splitting into sets. Finally, the sets are saved in a csv file with filenames matching the type of set (ie. *training_set.csv* and *training_labels.csv*).

2.2 Neural Network Construction

In this paper, only the neural network will be constructed from scratch, SVM will be constructed using the SVC class in sklearn library. Constructing the neural network requires an understanding of the different matrix sizes from the inputs, hidden layers, and outputs. First for the inputs, the *training_set.csv* is a 6720x354 matrix where each row is an input with 354 features.Therefore the input layer will have rows equal to the number of columns in the training set, a 354x1 matrix. Since the neural network to be constructed is a shallow network, only two hidden layers are constructed for this experiment. Choosing the number of nodes to be used for each hidden layer is primarily influenced by the training runtime of the neural network before it reaches a predetermined minimum average error. Below are sample runtimes for different number of nodes in the hidden layers with average error set to 0.05, and learning rate set to 0.1:

Table 1: Runtime and accuracy for different number of nodes

Node Combination	Estimated Runtime(mins)	Accuracy
>300, >300	>60	-
200, 200	16.38	99.06%
100, 100	6.30	98.21%
200, 100	14.15	98.79%
100, 200	7.13	99.20%
20, 20	2.56	98.97%

Hidden layers with nodes greater or equal to the number of features of the input tend to have longer runtimes than those with lower number of nodes. Accuracy is not sacrificed in shallow neural networks since training the network will only stop after reaching a predetermined minimum average error. By setting the minimum average error to an even smaller value, the accuracy of ANN will also increase. Additionally, each processing of codes will generate different values for the weight matrix, since the weight array is initially composed of random values which will be adjusted per training iteration. Hence, runtimes for the above table are only estimates. Another thing to consider here is the learning rate parameter. Smaller values for this parameter tend to have better performance compared to larger values. However, smaller values also entail longer training times. There is a tradeoff between runtime speed and model accuracy which researchers should consider.

Table 2: Runtime and accuracy for different learning rates using 20 nodes.

Learning Rate	Estimated Runtime(mins)	Accuracy
0.1	2.56	98.97%
0.05	4.7	99.20%
0.01	22.45	99.29%

In this implementation of ANN, each hidden layer in the neural network will contain 20 nodes and a learning rate of 0.1 to acquire the desired fast training runtime. To increase performance, minimum average error is set to 0.00095 in order to sufficiently train ANN on the given dataset. The output of the 2nd hidden layer will then be used as inputs and flattened into a 1-D array with length 8 for the output layer. The output layer is an array consisting of 1s and 0s where a value 1 in a given index is equivalent to a certain output label and only one 1-value can exist for each output. The index with value of one corresponds to the highest label

possibility computed by the neural network. For example, given an output of [0,0,0,1,0,0,0,0] since 1 is located in index 3 the label of this output is 4. Therefore the output label is equivalent to the index of the maximum label possibility plus one. The architecture and parameters of the constructed ANN are shown below.

```
In [8]: #Defining the architecture of the neural network
        nn_in = training_cols
        nn_h1 = 20
        nn_h2 = 20
        nn_out = len(classes)

In [9]: #Setting the learning rate
        lr = 0.1

        #Weight adjustment
        wa = -0.1+(0.1+0.1)

        #Number of outputs
        total_output = len(training_labels)

In [10]: #Constructing the matrices
         x_in = np.zeros((nn_in))
         w_h1 = wa*np.random.rand(nn_h1,nn_in)
         b_h1 = wa*np.random.rand(nn_h1)
         w_h2 = wa*np.random.rand(nn_h2,nn_h1)
         b_h2 = wa*np.random.rand(nn_h2)
         w_out = wa*np.random.rand(nn_out,nn_h2)
         b_out = wa*np.random.rand(nn_out)
         d_out = np.zeros((nn_out))
```

Figure 3: Architecture of the neural network.

3. RESULTS AND DISCUSSION

3.1 Testing using SVM

Classification using support vector machines is implemented using the SVC class in the sklearn library. The SVC class supports five kernel tricks namely Linear, Polynomial, Gaussian or RBF, Sigmoid, and Precomputed.[4][5] Only the first four kernels have been trained and testing, since the Precomputed kernel requires a square matrix as input. Different kernel tricks will result to different performance on any given dataset. Training the SVM classifier requires fitting the training dataset and training labels into an optimized decision hyperplane. This is done using the code *svcclassifier.fit(training_dataset, training_labels)*. After training/fitting the data, the classifier is ready to predict output labels given an unknown dataset (see Figure 4 for details).

As seen in Table 3, Sigmoid resulted in the lowest performance among all kernel tricks tested. This is mainly due to the characteristitcs of the sigmoid kernel being a classifier returning only 1s and 0s and is, therefore, better used for binary classification.[5] Polynomial kernels yielded the highest performance on the dataset. But what degree to use? It can be inferred that the peak performance occured somewhere between degree 8 and degree 30 since accuracy is lower for degree values greater than 30, e.g. degree 100 has 94.06% performance. By checking all degrees between this range, the peak performance of the polynomial kernel is with degree 14, achieving a performance score of 99.11%. This kernel trick will be used to compare the performance of the constructed ANN, and for predicting the labels of the unknown dataset.

```
In [87]: from sklearn import svm
         from sklearn.svm import SVC
         import timeit #Calculating the runtime of the SVM classifier

         start = timeit.default_timer()

         svcclassifier = SVC(kernel='poly',degree=22)
         svcclassifier.fit(input, training_labels)

         stop = timeit.default_timer()

         print('Time: ', stop - start)

         Time:  2.706052199999249

In [88]: poly = svcclassifier.predict(test_input)
```

Figure 4: Code for training and testing SVM.

Table 3: Performance of different kernel tricks.

Kernel Trick	Estimated Runtime(secs)	Accuracy
Sigmoid	18.13	83.66%
Linear	5.37	97.23%
RBF	8.82	94.73%
Polynomial, degree 8	3.43	98.62%
Polynomial, degree 14	2.69	99.11%
Polynomial, degree 15	2.91	99.06%
Polynomial, degree 30	3.22	98.44%
Polynomial, degree 50	4.45	97.05%
Polynomial, degree 100	7.09	94.06%

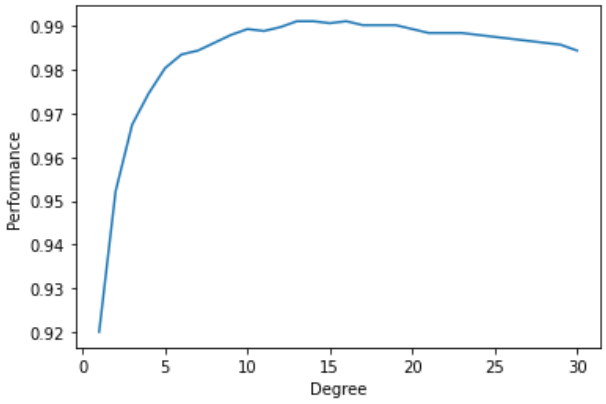

Figure 5: Performance for varying polynomial degrees.

3.2 Testing using ANN

Computing for the minimum average error is essentially a part of the validation procedure. This procedure uses a set independent of the training set to provide an unbiased error rate computation. Once the training and validation set have reached the minimum average error threshold, ANN has been fully trained on the given dataset and ready for testing. Intuitively, in order to reach the minimum average error, the training phase is repeated over a certain number of epochs. Having fewer epochs will limit the training phase and result to a failure in reaching the threshold required leading to poor peformance. A graph of the relationship between these two variables are shown below.

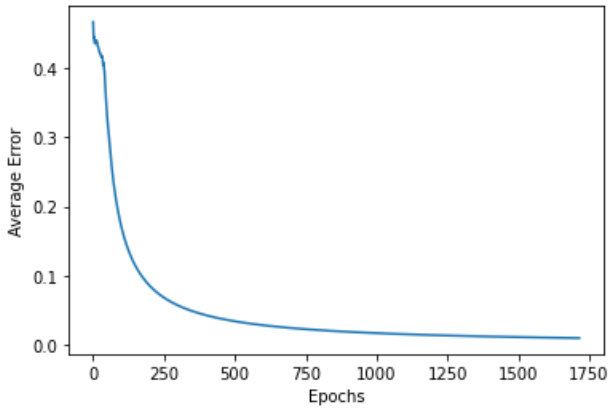

Figure 6: Average error vs. Number of Epochs.

The testing phase for the trained neural network utilizes the dataset and labels under the partitioned *test_set.csv* and *test_labels.csv*. Comparing the performance results on the testset for ANN and SVM, it can be seen that both have their own advantages. An ANN trained with multiple epochs, backpropagation, and a small minimum average error yielded a slightly better performance of 0.13 than SVM employing polynomial kernel tricks of degree 14. A major downside, however, is the long training time to fully optimize the neural network. Whereas SVM only trained for 2.69 seconds, ANN's running time is estimated at 750.11 seconds, or over 12 minutes!

Table 4: Comparison of ANN and SVM.

Algorithm	Estimated Runtime(secs)	Accuracy
ANN	750.11	99.24%
SVM	2.69	99.11%

4. CONCLUSION

Which algorithm is better? In terms of running-time, SVM is the algorithm to choose - offering higher training speeds without sacrificing performance. However, choosing the best kernel trick to use can be tedious and SVM's will peak at a certain performance level. On the other hand, despite the longer training run-times of ANN, it can actually perform even better by setting an even smaller value for the minimum average error. Other developments in improving ANN can also be employed such as dropout which could improve further its performance.[2] By doing so, the ANN classifier can yield a performance level no SVM classifier can attain. Additionally, training at better number of nodes and smaller learning rates could yield higher results but should only be considered if time is not restricted or if machines can compute faster.

As a final task, the two algorithms created are used to predict the labels for the unknown dataset and saved in files *predicted_ann.csv* and *predicted_svm.csv* for ANN and SVM, respectively.

5. REFERENCES

[1] J. Brownlee, 'SMOTE for Imbalanced Classification with Python', *Machine Learning Mastery* [web blog], 21 August 2020, https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/ (accessed 28 October 2020).

[2] M. Jeeva, 'The Scuffle Between Two Algorithsm - Neural Network vs. Support Vector Machine',*Medium* [web blog], 15 September 2018, https://medium.com/analytics-vidhya/the-scuffle-between-two-algorithms-neural-network-vs-support-vector-machine-16abe0eb4181 (accessed 1 November 2020).

[3] S. Haykin. *Neural Networks and Learning Machines* 3rd edn., New Jersey, Pearson Education Inc., 2009.

[4] sklearn.svm.SVC, *sckit-learn* [website], https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html, (accessed 1 November 2020).

[5] U. Malik, 'Implementing SVM and Kernel SVM with Python's Scikit-Learn', *Stack Abuse* [web blog], https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/ (accessed 28 October 2020).