Cardiac Dysrhythmia Detection with GPU-Accelerated Neural Networks

Albert Haque AHAQUE@CS.STANFORD.EDU
Computer Science Department, Stanford University

Abstract

Cardiac dysrhythmia is responsible for over half a million deaths in the United States annually. In this work, we evaluate the performance of neural networks on classifying electrocardiogram (ECG) sequences as normal or abnormal (arrhythmia). Using neural networks as our primary learning model, we explain our model's performance and discuss hyperparameter tuning. Comparing the results of our model to SVMs, random forests, and logistic regression, we find that our neural network outperforms the other three models with a binary classification accuracy of 91.9%. For the multi-class classification task, we achieve an accuracy of 75.7%. The use of GPUs accelerates the neural network training process up to an order of magnitude over CPUs.

1. Introduction

The human heart is controlled by an electrical system which stimulates blood movement via contractions. Cardiac dysrhythmia (or arrhythmia) occurs when the electrical activity of the heart is irregular. Effects of arrhythmia range from discomfort to cardiac arrest. Although most arrhythmias are harmless, Arrhythmia is still responsible for about 500,000 deaths in the US, annually. Early detection and treatment of arrhythmia can reduce the number of deaths by 25% (Hoefman et al., 2010).

Identifying patterns in arrhythmia has been studied for several years and many statistical approaches have been attempted. These approaches can be grouped into two categories: (i) statistical learning based on explicit feature extraction and (ii) recognizing patterns from the raw time series data.

Most attempts fall into the first category of extracting features using human intuition. Many studies use classical machine learning algorithms such as support vector machines

This document is part of a term project for CS 229 Machine Learning, Stanford University (Autumn 2014) and was created on December 13, 2014.

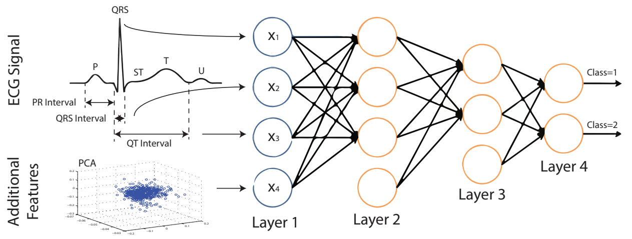


Figure 1. Proposed Neural Network Classifier. The original ECG record and hand-crafted features are used as inputs.

(Song et al., 2005). The second category of approaches are centered around time series analysis. Time series approaches use wavelet transforms and attempt to minimize the noise present in the data (Thakor & Zhu, 1991). Some models note the periodic interval between the QRS complex and PR/QT intervals (see Figure 1). Autoregressive neural networks have also been proposed for forecasting time series data (Zhang, 2001).

In this work, we are concerned with two problems. First, we want to classify a record into binary classes (normal or abnormal). Second, we want to classify a record into multiple classes, depending on the specific case of arrhythmia present (multi-class classification). We construct a neural network topology using significant hyperparameter tuning and leverage the parallel computing power of GPUs to accelerate the training process.

2. Dataset and Feature Extraction

We use the arrhythmia dataset found at the UCI Machine Learning Repository (Bache & Lichman, 2013; Guvenir, 1997). It consists of 452 records with 279 attributes per record. Each record is assigned to 1 of 16 classes: a class label of 1 indicates normal ECG patterns while a class label between 2 to 16 indicates "abnormal ECG patterns" or arrhythmia of varying types. The dataset's class distribution is shown in Figure 3.

In our experiment, we use principal component analysis to extract features from the dataset and is detailed in Section 4.1. Using these principal components, we train several models and compare their results.

Figure 2. Regularized cost function for multi-layer neural networks. Let $\Theta^{(\ell)}$ contain the network parameters for layer ℓ, $h_\Theta(x) \in \mathbb{R}^K$ be the output probabilities where K is the number of classes, $(h_\Theta(x))_i$ denotes the i^{th} output, L denotes the number of layers and s_ℓ denotes the number of neurons in layer ℓ.

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)})_k + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)})_k))\right] + \frac{\lambda}{2m}\sum_{\ell=1}^{L-1}\sum_{i=1}^{s_\ell}\sum_{j=1}^{s_{\ell+1}}(\Theta_{ij}^{(\ell)})^2$$

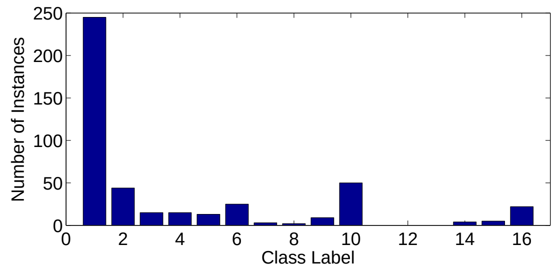


Figure 3. Class Distribution

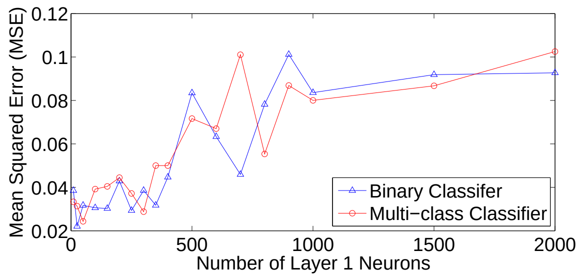
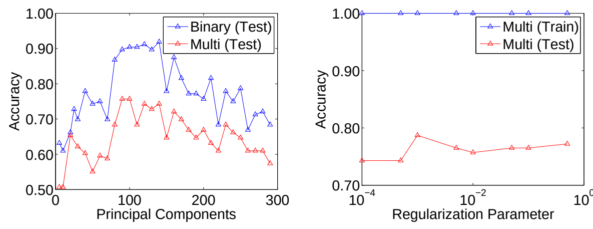


Figure 4. Number of Neurons vs MSE



(a) PCA (b) Regularization

Figure 5. Hyperparameter Tuning Results

3. Models

We compare our neural network model to three commonly used approaches: (i) support vector machines, (ii) logistic regression, and (iii) random forests. For support vector machines and logistic regression, we use a one-vs-all classifier for the multi-class classification problem.

3.1. Neural Networks

We employ a multi-layer neural network for both binary and multi-class classification. The output of each neuron is the sigmoid function:

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}},\ x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix},\ \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \quad (1)$$

We train the network using back propagation and stochastic gradient descent to minimize the cost function in Figure 2. Due to the large number of hyperparameters, a significant portion of this project was devoted to exploring the effect of hidden layer size, depth, learning rate, and regularization parameters.

Perhaps the most important hyperparameter, we analyze the effect of the number of neurons on performance. Figure 4 shows the effect of the number of neurons on mean squared error (MSE). As the number of neurons grows, the performance decreases. This is apparent for both the binary and multi-class case. Based on these results, we use 100 neurons at each layer. Although the analysis is not shown, deeper networks do not necessarily produce better results. Most often, performance suffered as the number of hidden layers increased. Based on this analysis, we select two hidden layers for our network topology.

Instead of using features from the raw dataset, we use principal components as inputs into our neural network. As shown in Figure 5(a), we analyze the number of principal components versus the classification accuracy. Too few (less than 50) or too many (greater than 200) principal components result in poor performance across all models (only NN results are shown). Based on these results, we use 100 principal components for our final model.

The regularization parameter λ attempts to prevent overfitting by reducing the magnitude of parameters in Θ. Fixing all other parameters, we vary λ for various values (see Figure 5(b)), and set $\lambda = 0.01$.

In summary, we use the following hyperparameters:

1. Regularization parameter, $\lambda = 0.01$
2. Two hidden layers $L = 2$
3. One hundred neurons at each layer $H_1 = H_2 = 100$
4. First 100 principal components are used as features

3.2. Support Vector Machines

Due to its popularity in practice, we evaluate the performance of SVMs on our dataset. The SVM attempts to find the maximum-margin hyperplane that separates the dataset in a higher dimensional feature space. Finding this optimal margin reduces to solving the following convex optimization problem:

$$\min_{\gamma,w,b} \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i \tag{2}$$

Subject to the constraints:

$$\text{s.t.} \quad y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \; i = 1, ..., m \tag{3}$$

$$\xi_i \geq 0, i = 1, ..., m \tag{4}$$

Where the ξ_i above allows for "slack" in the event the dataset is not linearly separable.

3.3. Logistic Regression

Closely related to our neural network, logistic regression aims to minimize the cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right) \tag{5}$$

where $h(x) = \theta^T x$ and θ is defined in (1). To learn the parameters θ, we minimize $J(\theta)$ using stochastic gradient descent.

3.4. Random Forests

Random forests apply the technique of bootstrap aggregating (i.e. bagging) to learn a model. B decision trees are created where each decision tree is trained on a random sample of the training data. Equation (6) shows how y is computed for an unseen example in a regression problem.

$$\hat{y} = \frac{1}{B}\sum_{b=1}^{B}\hat{y}_b(x) \tag{6}$$

where $\hat{y}_b$ is the prediction for decision tree b. For classification problems, the majority vote of all B trees is taken (instead of the average) to output the predicted label.

4. Experiment

We separated the dataset into a training set consisting of 316 records and a hold-out test set containing 135 randomly selected records. Both Matlab and Python were used for preprocessing, training, and testing.

Our dataset consists of 452 records. Of these, 32 records are incomplete and missing at least one feature (excluding the "vector angle to J feature"). Because we do not want to discard these records, we instead impute the values by using the feature's mean value as this tends to improve performance (Jerez, 2010).

Before we began training of our models, we performed a singular value decomposition over the dataset, extracted the first 100 principal components, and used these as input features int our models.

4.1. Results

Surprisingly, our neural network outperforms the SVM, random forest, and logistic regression model in the binary classification problem (see Table 1). In the multi-class case, our neural network still outperforms SVMs and logistic regression with a test accuracy of 75.7%.

Table 1. Classification Accuracy

	Binary		Multi-Class	
Model	Training	Test	Training	Test
Neural Network	100.0%	91.9%	100.0%	75.7%
SVM	92.4%	87.5%	96.6%	65.1%
Random Forest	99.7%	72.0%	97.0%	76.0%
Logistic Regr.	100.0%	77.6%	100.0%	69.0%

Because the cost of false diagnoses can be expensive, in monetary terms, we wanted to see our model's false positive rate. Figure 8 shows our model's ROC curves. On the test set, our network is able to achieve a true positive rate of 0.8 with a false positive rate of about 0.2. After that, higher false positive rates provide marginal gains in true positive detections.

5. Discussion

As indicated by the training accuracies, the neural network, random forest, and logistic regression tend to fit (or closely fit) the training data perfectly. However, we know these models are not memorizing the input data since the test accuracies are above 70% and 65% in the binary and multi-class case, respectively.

We wanted to further analyze the performance of our neural network by looking at the convergence rate. Figure 6 shows the mean squared error as a function of number of iterations for different training sizes.

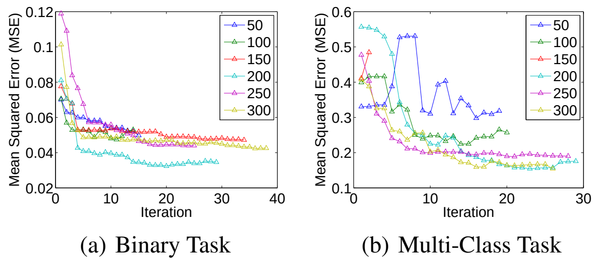


Figure 6. Convergence Rate of the Neural Network Model (Training). Because the neural network was trained on 1,000 iterations, only the first few iterations are shown above.

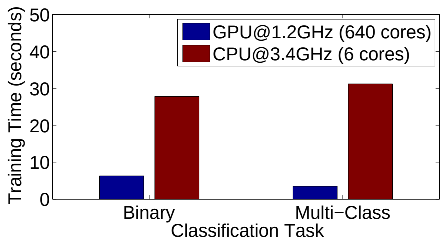


Figure 7. CPU vs GPU Runtime

As previously mentioned, we used both CPUs and GPUs for training our neural network. SVMs, logistic regression, and random forests were trained using CPUs only. Due to the large number of cores available on GPUs, GPUs are able to quickly train neural networks – especially those with a large number of neurons. We used a GeForce GTX 750 Ti GPU with 640 CUDA cores clocked at 1.2 GHz with 2 GiB of GDDR5 memory. The CPU used was an Intel i7-4930K 6 core processor clocked at 3.4 GHz and 32 GiB of DDR3 memory.

The plot shown in Figure 7 show the results of our multi-class classification task by training on a neural network with two hidden layers containing 100 neurons each. The training phase consisted of 1,000 iterations. For GPUs, the bottleneck is the data transfer to and from the device. Once the training data has been transferred to the GPU, the computation is very fast. Because our training data is so small (on the order of a few MiB), the GPU runtime does not experience long initial or post-processing delays. For networks with more hidden nodes, we can expect the GPU vs CPU runtime margin to increase (Juang et al., 2011; Steinkrau et al., 2005).

5.1. Error Analysis

To visualize the results from all models, we generated confusion matrices depicted as heat maps in Figure 9. Both logistic regression and neural networks achieved a training accuracy of 100%. This is denoted by values present only on the diagonal. For SVMs and random forests, we see incorrect classifications present on the off-diagonal entries.

Looking at the test set, we see that SVMs tend to incorrectly categorize normal heartbeats (label=1) with labels 2-16. This results in a high false positive rate. Our neural network has few errors on the test as stated in Table 1; this is confirmed in the confusion matrix above as most true/predicted labelings lie along the diagonal.

Confusion matrices are a good indicator of false positive and false negative rates. Depending on the application, one can be more expensive (in monetary terms) than the other. To analyze these rates, we plotted receiver operating characteristic (ROC) curves for the binary classification case

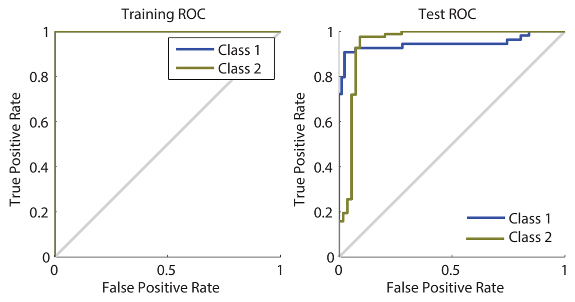


Figure 8. ROC Curves for the Binary Classification Task. Class 1 corresponds to the normal ECG case and class 2 corresponds to arrhythmia is present.

in Figure 8. We do not plot the multi-class case as some classes contain one or two training examples and this skews the ROC curves for these labels.

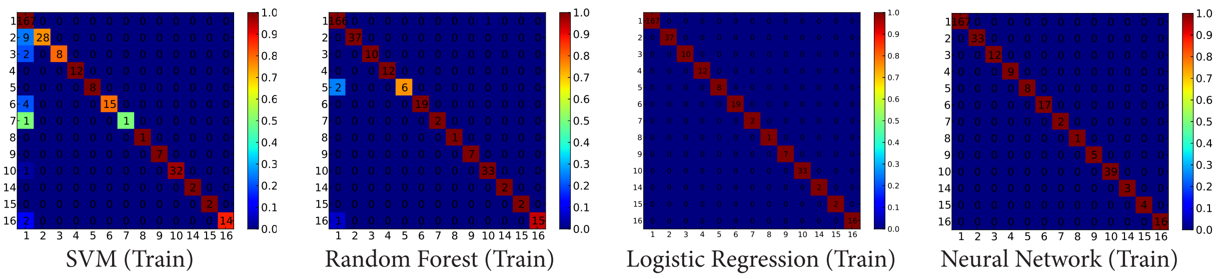
Figure 8 shows a perfect ROC curve for the training phase, as expected. Looking at the test ROC curve, if we are willing to tolerate a false positive rate of about 0.1, we are able to achieve a true positive rate of 0.9. In this particular application (in the domain of medical practice and arrhythmia detection), early detection has the potential to save lives. As a result, the cost of a false negative is greater than the cost of a false positive. If a patient is diagnosed as arrhythmia present (predicts positive), we can perform further tests to correctly label this patient.

6. Conclusion

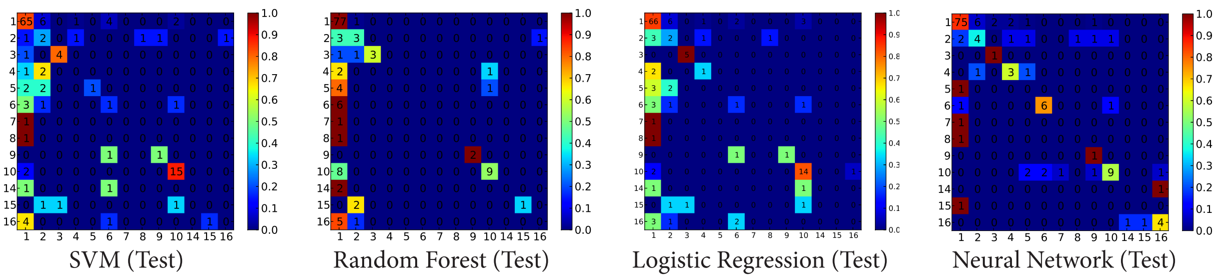
Due to the time-series nature of ECG data, future work can explore recurrent and autoregressive neural networks (Zhang, 2001). These networks are well suited for predicting future time series and can be applied to ECG signals. Additionally, as with any learning problem, more (training) data would benefit our research. A longer-term project would aim to utilize smart health sensors (FitBit, Jawbone, Microsoft Band, etc.) for real-time cardiac monitoring.

In this work, we used principal component analysis to transform the original 279 features from the dataset. We then analyzed the effect of number of principal components, hyperparameter tuning, and the runtime effect of GPUs. Results show that our neural network outperforms other methods on the binary classification task. For multi-class Arrhythmia classification, our neural network performs well and is beaten only by random forests. Future work can explore the use of ECG time series data and let the neural network extract features from the raw time series data on its own.

As previously mentioned, the cost of false negatives is very high. Building a classifier to minimize the false negative rate can bring many benefits to the medical and healthcare industry. This work allowed us to deeply analyze learning



(a) Confusion Matrices (Training Set)



(b) Confusion Matrices (Test Set)

Figure 9. Confusion matrices for all models for the multi-class classification problem. Predicted labels are on the y-axis and true labels are on the x axis. Numbers indicate the number of records with a specific true/predicted labeling. Labels 11, 12, and 13 are not shown because these labels do not appear in the dataset.

algorithms using real data. We hope future researchers and students can build on this work.

References

Bache, K. and Lichman, M. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

Guvenir, H Altay et al. A supervised machine learning algorithm for arrhythmia analysis. In *Computers in Cardiology 1997*, pp. 433–436. IEEE, 1997.

Hoefman, Emmy, Bindels, Patrick JE, and van Weert, Henk CPM. Efficacy of diagnostic tools for detecting cardiac arrhythmias: systematic literature search. *Netherlands Heart Journal*, 18(11):543–551, 2010.

Jerez, José M et al. Missing data imputation using statistical and machine learning methods in a real breast cancer problem. *Artificial intelligence in medicine*, 50(2):105–115, 2010.

Juang, Chia-Feng, Chen, Teng-Chang, and Cheng, Wei-Yuan. Speedup of implementing fuzzy neural networks with high-dimensional inputs through parallel processing on graphic processing units. *Fuzzy Systems, IEEE Transactions on*, 19(4):717–728, 2011.

Song, Mi Hye, Lee, Jeon, Cho, Sung Pil, Lee, Kyoung Joung, and Yoo, Sun Kook. Support vector machine based arrhythmia classification using reduced features. *International Journal of Control Automation and Systems*, 3(4):571, 2005.

Steinkrau, Dave, Simard, Patrice Y, and Buck, Ian. Using gpus for machine learning algorithms. In *Proceedings of the Eighth International Conference on Document Analysis and Recognition*, pp. 1115–1119. IEEE Computer Society, 2005.

Thakor, Nitish V and Zhu, Yi-Sheng. Applications of adaptive filtering to ecg analysis: noise cancellation and arrhythmia detection. *Biomedical Engineering, IEEE Transactions on*, 38(8):785–794, 1991.

Zhang, Guoqiang Peter. An investigation of neural networks for linear time-series forecasting. *Computers & Operations Research*, 28(12):1183–1202, 2001.