1. Comparison between Multi-Layered Neural Network and Support Vector Machine

Report to the coursework project of module COMP61011 (2017-2018)
Daqian SHI - 10167702

2. Abstract

The report is written as a submission to the coursework of module Foundations of Machine Learning (COMP61011). It is aimed to accomplish the last suggested project – 'Implement and train a multi-layered neural network, evolving the weights using the difference between expected and actual outputs as a fitness function. Then compare its properties, such as accuracy or speed, to an SVM.' We constructed a neural network implemented in Python and then compared it with a SVM classifier developed in MATLAB for the aspects of accuracy and training time, on three kinds of datasets – 'attribute-fixed', 'semi-attribute-fixed', and 'attribute-floating'.

3. Introduction

Support Vector Machines (SVM) and Neural Network (NN) have been popular algorithms in the area of machine learning, and have been widely used in classification (see [1] and [2]). As a very basic implementation, NN uses Gradient Descent (GD) to find the best optimised weights of activation functions, while SVM uses Quadratic Programming (QP) techniques to find the weights of kernel functions. Normally, a standard NN minuses a derivable loss function while SVM minuses hinge loss function. They have the same logic of optimising weights – minimising loss, but usually have totally different implementations.

Comparison of the performance between the two models doesn't aim at simply confirming which one is better, but at choosing the better one for data-dependent tasks. [3] introduced a comparison among multiple classification techniques – Partial Least Squares (PLS) regression, Polynomial-PLS (Poly-PLS), Artificial NN (ANN), SVM Regression (SVR), and Least Squared SVM (LS-SVM) – on Near Infrared (NIR) spectroscopy data for analytical chemistry, in which context SVM approaches and ANN performed quite similarly in accuracy and were all better than PLS and Poly-PLS in most cases. [4] compared the performance between SVM and ANN for drug/non-drug classification system, in which SVM has very slightly better performance than ANN.

We performed a comparison on a 'feature-extracted' and 'attribute-fixed' dataset, which means, unlike pure images, the features of the data are extracted as fixed attributes. A popular way of implementing multi-class SVM is decomposition to multiple binary classifiers, as [5] and [6] introduced; hence, we focused on binary classification, which provided the most direct view of the performance of SVM. In addition, we also compared the performance between Convolutional NN (CNN) and SVM on classification of two image datasets, which are a typical form 'attribute-floating' dataset. We would like to see how NN and SVM performed on the 'attribute-fixed' and 'attribute-floating' datasets.

4. Dataset and Data Pre-Processing

The dataset with fixed attributes was from [7], which was provided for binary classification. It contains 28 attributes and records the credit card information of *bank* clients. All the attributes were standardized to real numeric values using Python package *sklearn* in NN. Furthermore, the dataset is highly unbalanced - only 5% tuples in the dataset is of class = 1. [8] introduced a solution – undersampling – to learning from unbalanced data by having the same sizes of data for all classes. To have a more sensible view of the performance, we followed the article and performed undersampling in training. For our project, the results of the classification from repeated undersampling the training data trended to average, which indicated that the accuracy loss from undersampling was ignorable.

The first image dataset was *CIFAR-10* from [9]. There are totally 10 classes of objects, but the samples of each class vary largely – the objects are not located at the same place, not facing the camera to the same angle, and are even of different sub-classes, e.g. bees and butterflies are both under the class *insects*. Without deeper learning, the attributes (features) of the dataset 'float' a lot – no matter how we convolve the image or extract Histograms of Oriented Gradients (HOG), we are not able to expect the features to be located at the same 'convolution or HOG position'; hence, this dataset was also chosen on which NN and SVM were implemented for comparison, as a contrast to the *bank* dataset. For time efficiency, we chose the first training batch of *CIFAR-10* as our training set, which contains 10,000 images and 1,000 for each class. Besides, the original RGB images were transformed to gray levels, so we didn't use the RGB features of the dataset.

The second image dataset was *MNIST from* [10]. MNIST was provided for recognizing and classifying the images of number from 0 to 9. Different from *CIFAR-10*, the position of the object – numbers – on the images are roughly fixed; so, the features of the MNIST data is somehow 'semi-fixed'. In conclusion, three types of data – attribute-fixed, attribute-floating, and semi-attribute-fixed – on which were put into implementation for comparison between NN and SVM.

5. Multi-Layered NN Construction and Optimization

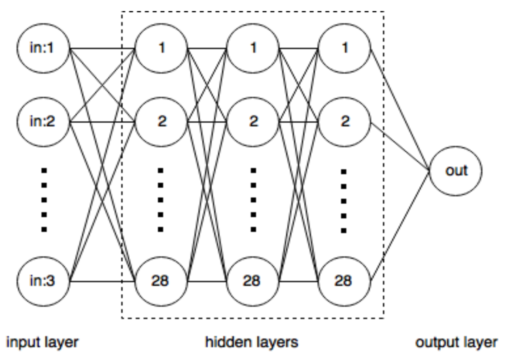

Fig. 3.1. NN topology

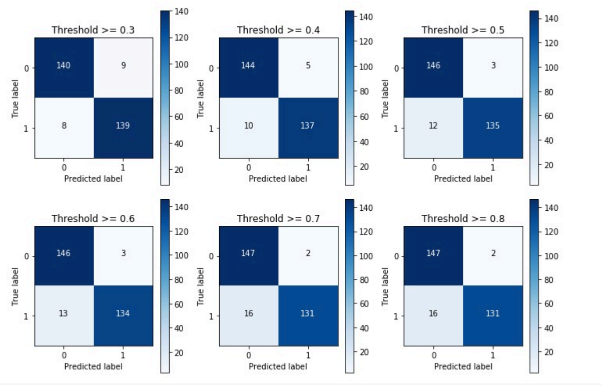

Fig. 3.2. Confusion matrix for different threshold t

The NN model was written in Python with the mean scientific computing package *Numpy*, and the model construction referred to [2], [11], and [12]. The NN is constructed and trained in 4 steps: initializing parameters, forward propagation, backward propagation and updating parameters. Furthermore, we paid further attention to certain details to optimise it. For example, every result from forward propagation should be stored in a cache, which will be used to compute the backward propagation, and every result from backward propagation should be stored in another cache then use to compute the parameters in next iteration. These details helped build the k-layer neural network more efficient without using too many loops in program.

The model we used for NN on the *bank* dataset was Restricted Boltzmann Machine with three hidden layers, and each hidden layer has 28 neurons, the size of which was the same as the number of attributes, as Fig. 3.1. shown. Every neuron had the same activation function; hence, each neuron had an output represented as $f(x_i^l, w_i^l) = \phi(\sum_i w_i^l \cdot x_i^l)$, where $\phi(\cdot)$ is the activation function, x_i^l is the i^{th} input at l^{th} layer, and w_i^l is the corresponding weight. We had tested the model with the hyperbolic tangent function $f(x) = \tanh(x)$ and the Rectifier Linear Unit (ReLu, referring to [11]) $f(x) = \log(1 + e^x)$ as activation function. These concepts were implemented via Python package *scikit-learn*.

We had implemented several strategies to optimize the performance of NN, such as K-fold cross validation (and/or its basic logic), which was widely used during construction of NN. It was not just used for avoiding over-fitting, but also used in determining the best learning rate, etc. For example, within the process of training, we assigned different learning rates to different folds, and then selected the learning rate with best performance. This procedure was repeated for multiple times. The best learning rate we found for the chosen dataset was 0.1.

Another applied optimisation was setting the best value for the parameter threshold t, which is the probability classifier of the NN model on the output neuron. After the training finished, a manual adjusting for the threshold t was performed. This was not a necessary step, but we did have a little increase of performance from it. Fig. 3.2. shows the confusion matrices with different t, and the best was $t = 0.4$, the accuracy of which is 91.7%.

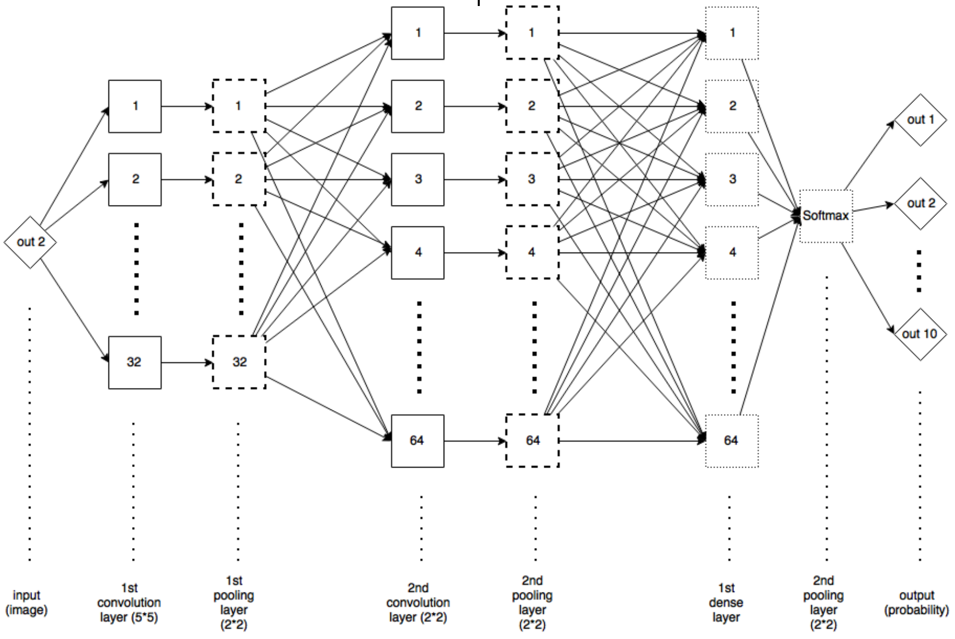

Fig. 3.3. CNN topology

The CNN model for *CIFAR-10* and *MINIST* used the library [13] and followed the ideas of [12] and [14]. There are two convolutional layers, two pooling layers, and two dense layers in the whole CNN, as Fig. 3.3. shown. The first convolutional layer has 32 5×5 neurons, while the second has 64 5×5 neurons; the two pooling layers consist 2×2 neurons; and the two dense layers are fully connected, of which the last dense layer is a single neuron. All the activation functions are ReLu, except the last dense layer, of which the activation function is Softmax $\sigma(\boldsymbol{x})_i = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}$ for $i = 1, \dots, K$ (here $K = 10$, since both *CIFAR-10* and *MNIST* has 10 classes), as [15] introduced. The learning rate is 0.01, and the number of epochs is 200.

6. SVM Construction

For the *bank* dataset, our SVM classifier is implemented through MATLAB with its internal function $fitcsvm$ [16], which is a later version of $svmtrain$ [17] that are wrapped into $MLOtools$. Since these functions are both provided by MATLAB and are replaceable with each other, plus that $fitcsvm$ has better memory management, there is no comparison with $MLOtools$. This is concluded by running SVM classifier on a 284,807*30 dataset with 8G RAM – $svmtrain$ had run out of memory and caused run-time exception while $fitcsvm$ ran correctly but took 11 hours, on Intel Core i7 chips. The kernel functions – Radial Basis Function (RBF), linear, and polynomial of order from 2 to 6 – are all tested for performance. Furthermore, we have the following setups (Tab. 4.1.) for the classifier while all other setups are as default (the default solver is Sequential Minimal Optimization (SMO), as [18] introduced):

Settings	**Arguments in MATLAB**	**Functions and Usages**
Automatic kernel scaling	‘KernelScale’ -> ‘auto’	Applying heuristic-based kernel norm to compute Gram matrix; for RBF, it normalises the σ value

Standardizing the predictor data	‘Standardize’ -> true	Centring and scaling the predictor data attributes

Tab. 4.1. SVM construction setups

Besides, 10-fold cross-validation are performed for all the datasets during the classification of SVM, to avoid the problem of over-fitting. For the extraction of desired performance data, the implementation of cross-validation is written by ourselves instead of using internal MATLAB libraries.

The strategy of the SVM classifier on *CIFAR-10* and *MINIST* followed [19] – extracting HOG features of the images and then classifying the HOG features via SVM. On MATLAB, the cells of size 4×4 was used to extract HOG features for both *CIFAR-10* and *MINIST*. The multi-class classification was implemented via the function *fitcecoc* (see [20]), into which a binary SVM classifier can be passed to perform multi-class classification. The binary classifier was built as the same as for the *bank* dataset.

7. Result and Discussion

7.1. Accuracy

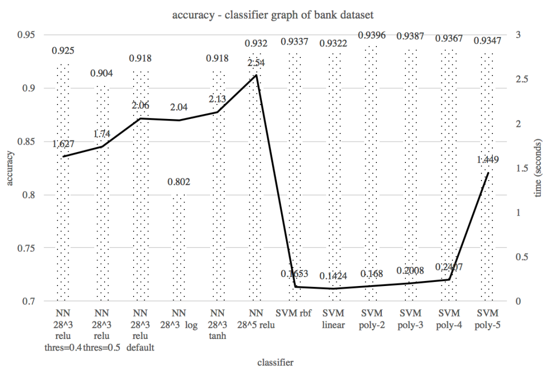

Fig.7.1. accuracies and training time of the bank dataset

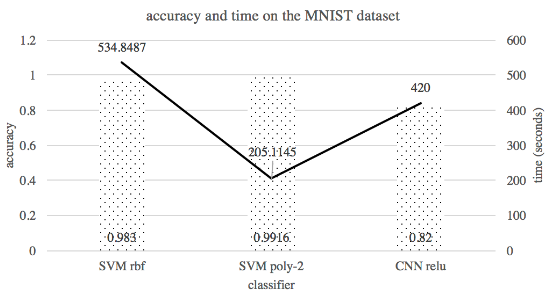

Fig.7.2. accuracies and training time of the MNIST dataset

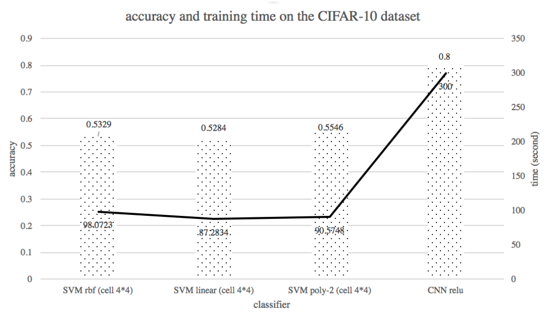

Fig.7.3. accuracies and training time of the CIFAR-10 dataset

Fig. 7.1., Fig.7.2., and Fig. 7.3. show the accuracies and training times of the implemented classifiers on the datasets of *bank*, *MNIST*, and *CIFAR-10* accordingly. The accuracy of all the NN and SVM classifiers on the *bank* dataset were quite similar, except that the NN of 28×28×28 with logit activation function had worst lowest accuracy, which is much lower than the average of other classifiers’. All the SVM classifiers performed quite stably, with around 93% accuracy, and was slightly (about 7%) better than NN classifiers. The optimization of the threshold t for ReLU-version NN gained about 0.7% increase on the performance, and a NN with 5 hidden layers had about 1.2% performance benefit than that with 3 hidden layers.

For the *MNIST* dataset, the two SVM classifiers with RBF and 2-order polynomial kernels performed at a similar level, but the CNN was not as accurate as them. The reason for the relatively poor performance of CNN might be the limit number of epochs considering the small value of learning rate. A short and slow learning (training) process might not be sufficient for large-size dataset. On the other hand, the CNN classifier with the same build produced much better classification result on the *CIFAR-10* dataset. The accuracy of SVM classifiers on it were just over 50%, but that of an insufficiently-trained CNN were around 80%. The test samples are equally distributed in the test batch, so the probability of guessing the correct class of a test object on random is 0.1, recalling that there are 10 classes. Hence, SVM classifiers had produced better classification than random guessing. The relatively worse performance of them might be caused by the large size of the cell used to extract HOG features. For the handwritten numbers in *MNIST*, a 4×4 cell extracted enough details; however, such sized cell might not extract as detailed HOG features as needed. Firstly, the images in *MNIST* are 28 by 28, while the size for that in *CIFAR-10* is 32 by 32; secondly and as mentioned before, *MNIST* has semi-fixed attributes while *CIFAR-10* is almost completely attribute floating, and considering the high accuracy on attribute-fixed and semi-attribute-fixed dataset, SVM based on the HOG features is probably not a good solution for attribute-floating dataset.

7.2. Training Time

As Fig.7.1. – Fig.7.3. shown, SVM were trained much faster than (C)NN in most cases. As introduced before, NN was developed in Python while SVM was developed in MATLAB. They were both trained and tested on the same MacOS. Furthermore, NN was trained and tested on an Intel i5-3210M CPU and SVM was trained and tested on Intel i7-7820HQ. As [21] stated, 7820HQ is about 86% faster than 3210M. Considering that both classifiers were trained in a pure training environment (no other program were running at the same time) and the speed benefit of 7820HQ, SVM on MATLAB is about 25% - 35% faster than NN for the tested conditions.

8. Conclusion and Discussion

According to our test, SVM and multi-layered NN have similar performance on attribute-fixed datasets, and SVM is very slightly more accurate. For classifying semi-attribute-fixed data, SVM is better than an insufficiently trained CNN, but for attribute-floating dataset, even an insufficiently trained CNN is much better than SVM on HOG features.

9. References

[1] B. Schölkopf, Learning with kernels, Cambridge, Mass: MIT Press, 2009.

[2] J. Heaton, AIFH, Volume 3: Deep Learning and Neural Networks, St. Louis (Mo.): Heaton Research, Inc., 2015.

[3] R. M. Balabin and E. I. Lomakina, "Support vector machine regression (SVR/LS-SVM)—an alternative to neural networks (ANN) for analytical chemistry? Comparison of nonlinear methods on near infrared (NIR) spectroscopy data," *The Analyst,* vol. 136, no. 8, p. 1703, 2011.

[4] E. BYVATOV, U. FECHNER, J. SADOWSKI and G. SCHNEIDER, "Comparison of Support Vector Machine and Artificial Neural Network Systems for Drug/Nondrug Classification," *Journal of Chemical Information and Computer Sciences,* vol. 43, no. 6, pp. 1882-1889, 2003.

[5] J. Weston and C. Watkins, "Multi-Class Support Vector Machine," ResearchGate GmbH, March 1999. [Online]. Available: https://www.researchgate.net/publication/221997121_Multi-Class_Support_Vector_Machine. [Accessed 25 October 2017].

[6] V. Franc and V. Hlavac, "Multi-class support vector machine," in *Object recognition supported by user interaction for service robots*, Quebec, 2002.

[7] Y. Tang, "Machine Learning & Deep Learning Practical," CSDN.NET, 2017. [Online]. Available: http://edu.csdn.net/course/play/5084/94276. [Accessed 25 October 2017].

[8] X. Liu, J. Wu and Z. Zhou, "Exploratory Undersampling for Class-Imbalance Learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics),* vol. 39, no. 2, pp. 539-550, 2009.

[9] A. Krizhevsky, "The CIFAR-10 dataset," Department of Computer Science, University of Toronto, 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html. [Accessed 26 October 2017].

[10] Y. LeCun, C. Cortes and C. J. Burges, “THE MNIST DATABASE of handwritten digits,” [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed 26 October 2017].

[11] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proc. 27th International Conference on Machine Learning*, 2010.

[12] A. Krizhevsky, I. Sutskever and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, Stateline, 2012.

[13] I. Google, “All symbols in TensorFlow | TensorFlow,” Google, Inc, 2017. [Online]. Available: https://www.tensorflow.org/api_docs/python/. [Accessed 26 October 2017].

[14] U. o. S. Stanford Computer Science, “ConvNetJS CIFAR-10 demo,” University of Stanford, [Online]. Available: http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html. [Accessed 26 October 2017].

[15] C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), Newer (Colored) edition ed., Singapore: Springer, 2007.

[16] MathWorks, “Train binary support vector machine classifier - MATLAB fitcsvm - MathWorks United Kingdom,” MathWorks, Inc, 2017. [Online]. Available: https://uk.mathworks.com/help/stats/fitcsvm.html. [Accessed 25 10 2017].

[17] MathWorks, “Train support vector machine classifier - MATLAB svmtrain - MathWorks United Kingdom,” MathWorks, Inc, 2017. [Online]. Available: https://uk.mathworks.com/help/stats/svmtrain.html. [Accessed 25 10 2017].

[18] R. Fan, P. Chen and C. Lin, “Working set selection using second order information for training support vector machines,” *Journal of Machine Learning Research,* vol. 6, p. 1889–1918, 2005.

[19] N. Dalal and B. Triggs, “Histograms of Oriented Gradients for Human Detection,” in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2005.

[20] Mathworks, “Fit multiclass models for support vector machines or other classifiers - MATLAB fitcecoc - MathWorks United Kingdom,” Mathwork, Inc, 2017. [Online]. Available: https://uk.mathworks.com/help/stats/fitcecoc.html. [Accessed 26 October 2017].

[21] UserBenchmark, “UserBenchmark: Intel Core i5-3210M vs i7-7820HQ,” 2017. [Online]. Available: http://cpu.userbenchmark.com/Compare/Intel-Core-i7-7820HQ-vs-Intel-Core-i5-3210M/m185229vs2719. [Accessed 27 October 2017].