Model Averaging with Online Learning Algorithms

Pang Yue

This is a report on a series of experiments conducted to compare the performance of a traditional model selection method and model averaging with online learning algorithms applied to a set of machine learning algorithms, namely, logistic regression and neural networks with different parameter settings, on the classification problem of the MNIST database of handwritten digits.

Introduction of the Problem

The MNIST database of handwritten digits contain 28x28 pixel grayscale images of handwritten digits. It poses a classification problem in the sense that, with the pixels of an image as input, an algorithm should classify whether it represents the digit 0, 1, 2, ..., or 9. A very simple performance metric is the rate of accuracy of an algorithm on a particular set of images, which is the percentage of images correctly classified out of the total number of images.

This problem particularly intrigues me because I am interested in every aspect of language processing, and recognizing handwritten digits, although a relatively old problem, seems like a good place to start.

Introduction of the Dataset

The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples (Lecun *et al.*, 1998). Details on, for example, how the data was obtained, etc., can be found on the database website that will be listed in the reference list. The datasets are in the IDX file format.

With reference to the tutorial offered by deeplearning.net (http://deeplearning.net/tutorial/gettingstarted.html#opt-sgd), I split the official training set into an actual training set of 50,000 examples and a validation set of 10,000 examples in the original sequence of the dataset.

I load the datasets into my programming environment using the mnistHelper functions offered on http://ufldl.stanford.edu/wiki/index.php/Using_the_MNIST_Dataset . I made slight modifications to these functions, changing the 0 labels to 10 to be in accordance with 1-indexing of GNU Octave for convenience.

Introduction of My Programming Environment

All my code are written in GNU Octave, version 3.8.0. The Octave syntax is largely compatible with Matlab, so it is highly probable that they can be run on Matlab as well.

Training

I trained 10 standard logistic regression models on the actual training set, with regularization parameter $\lambda = 0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10$ respectively, which roughly follows a tripling pattern. They are trained with the one-versus-all method, using the `fmincg` function (Rasmussen, 2002) as optimizer, with the maximum number of iterations as 150.

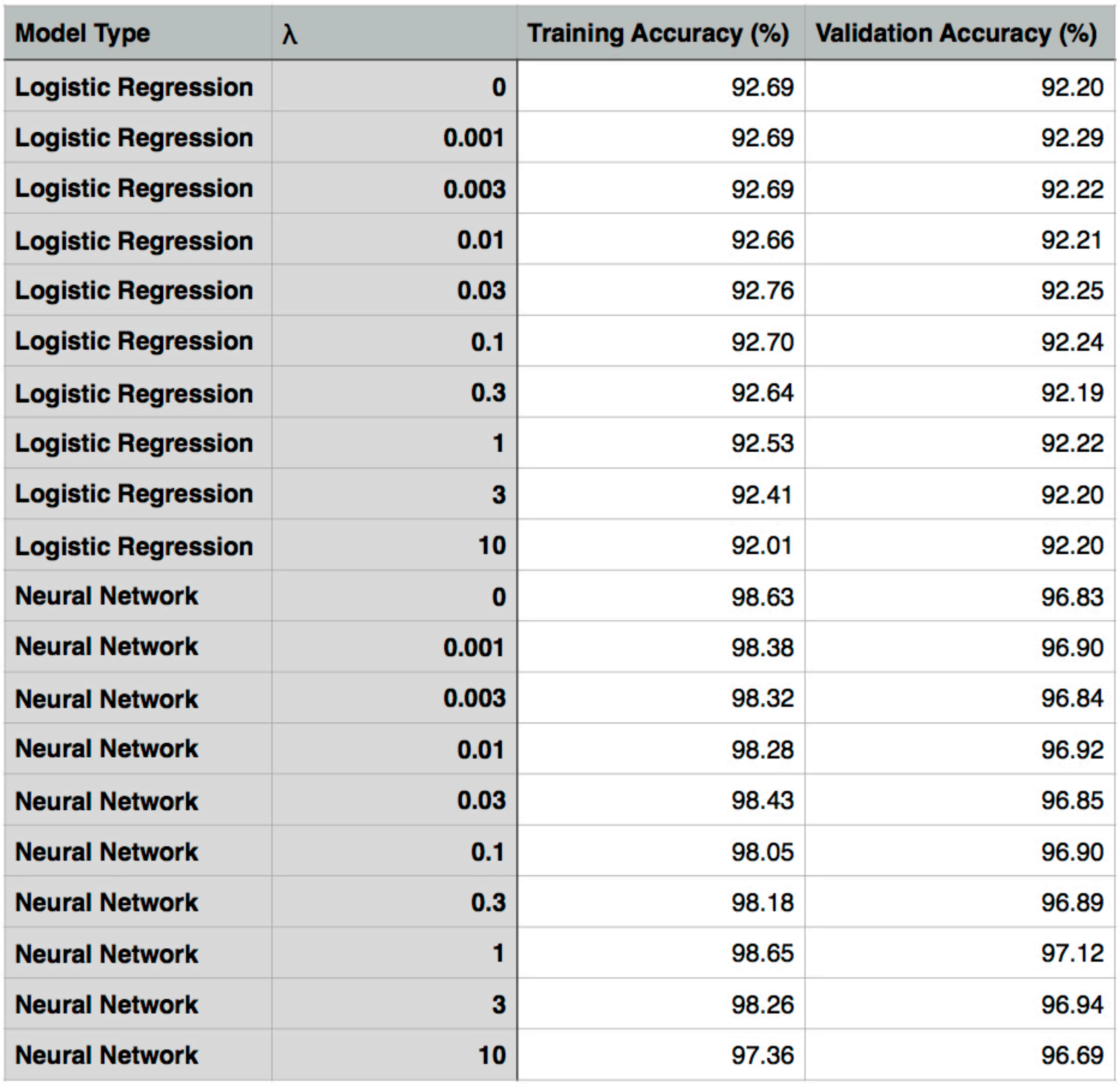
I trained 10 neural networks. They have a similar architecture: one input layer with 784 units (each accounting for one pixel in the input image), one hidden layer with 50 units, and one output layer with 10 units (each accounting for one label). This architecture is chosen based on doug's and hobs' answers on https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw : one hidden layer is sufficient for the large majority of problems;

the optimal size of the hidden layer is usually between the size of the input and the size of the output layers; and the empirical upper bound of the number of hidden units that will not result in over-fitting is $N_h = N_S/(\alpha * (N_i + N_o))$, where N_h is the number of hidden units, N_i is the number of input neurons, N_o is the number of output neurons, N_s is the number of samples in the training set, and α is an arbitrary scaling factor usually between 2 and 10. I chose 50, a relatively small number in the feasible range, to be on the safe side of the time allowed and the memory capacity of my laptop. These neural networks are trained with the forward propagation and backpropagation algorithm, and also optimized using the `fmincg` function.

After training, the algorithms are evaluated on the actual training set, and the learned parameters are saved as `.mat` files.

Validation

The validation script loads the learned parameters and evaluate the models on the validation set.



Model Type	λ	Training Accuracy (%)	Validation Accuracy (%)
Logistic Regression	0	92.69	92.20
Logistic Regression	0.001	92.69	92.29
Logistic Regression	0.003	92.69	92.22
Logistic Regression	0.01	92.66	92.21
Logistic Regression	0.03	92.76	92.25
Logistic Regression	0.1	92.70	92.24
Logistic Regression	0.3	92.64	92.19
Logistic Regression	1	92.53	92.22
Logistic Regression	3	92.41	92.20
Logistic Regression	10	92.01	92.20
Neural Network	0	98.63	96.83
Neural Network	0.001	98.38	96.90
Neural Network	0.003	98.32	96.84
Neural Network	0.01	98.28	96.92
Neural Network	0.03	98.43	96.85
Neural Network	0.1	98.05	96.90
Neural Network	0.3	98.18	96.89
Neural Network	1	98.65	97.12
Neural Network	3	98.26	96.94
Neural Network	10	97.36	96.69

From the table above it can be observed that all the models perform quite well on both the training set and the validation set, with accuracies all over 90%. There appears to be no under- or over-fitting problem for any model. All the models perform relatively poorer on the validation set than on the training set, which is reasonable since the parameters are trained to fit the training data.

Model Selection

The model with the highest validation accuracy, namely, the neural network with $\lambda = 1$, is selected as the best expert. It is retrained on the official training set of 60,000 examples. With a new set of parameters, it is then evaluated on the test set. Its test accuracy is 96.77%.

Model Averaging

It is noteworthy that all models that are averaged have not been retrained. The best expert previously selected uses its old set of parameters before retraining.

Six online learning algorithms are experimented on, including four variations of the Weighted Majority Algorithm(WMA1, WMA1Norm, WMA2, WMA2Norm), a variation of the Exponential Weights Algorithm(EWA), and the Randomized Exponential Weights Algorithm(REWA).

- WMA1

  WMA1 is the standard Weighted Majority Algorithm, with unnormalized weights and zero-one loss during update: $w_{t,i} = w_{t-1,i} * \beta^{I\{f_{t,i} \neq y_t\}}$

- WMA1Norm

  WMA1Norm is the same as WMA1 except that it uses normalized weights.

- WMA2

  WMA2 is a variation of the standard Weighted Majority Algorithm, with unnormalized weights and logistic loss during update: $w_{t,i} = w_{t-1,i} * \beta^{l_t(f_{t,i})}$, with l_t being the logistic loss function.

- WMA2Norm

  WMA2Norm is the same as WMA2 except that it uses normalized weights.

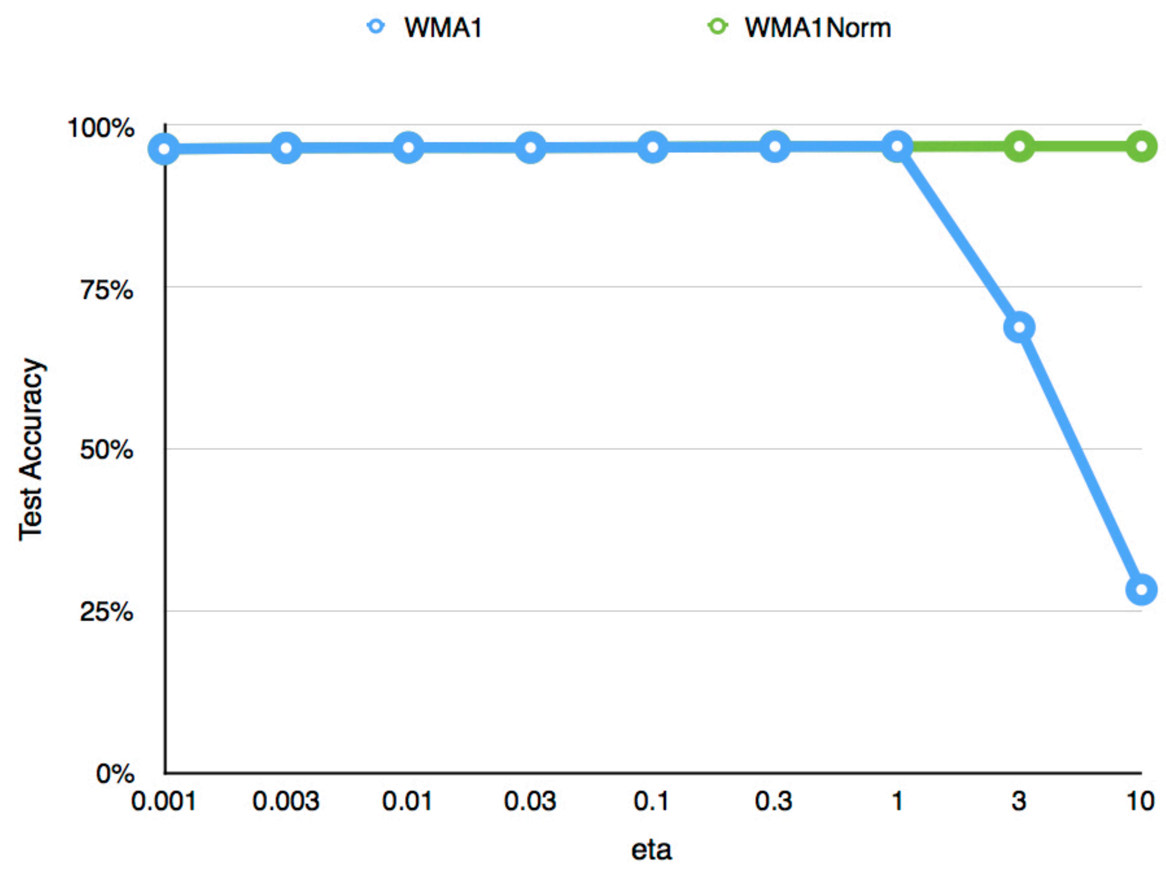
- EWA

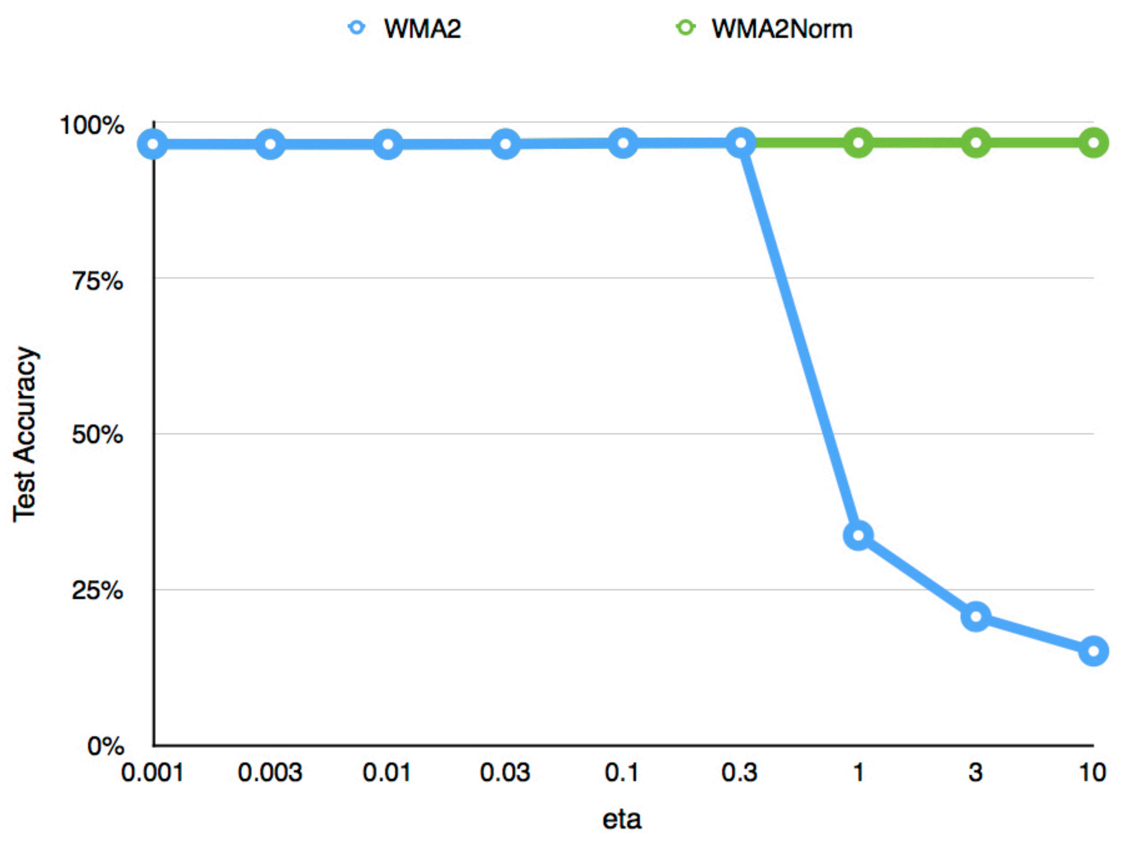
  EWA here is a variation of the standard Exponential Weights Algorithm. As the standard Exponential Weights Algorithm is only applicable to problems with a convex decision set, it is not originally applicable to the MNIST classification problem, the decision set of which is discrete. However, if the set of vectors of logistic function values for each label (from which the final prediction of a single label is drawn) is treated as the decision set, then the conditions for using the Exponential Weights Algorithm are satisfied. My implementation of EWA builds on this idea, averaging on the experts' predictions as vectors of logistic function values for each label with normalized weights, and making the final prediction by choosing the label with the highest value in this averaged vector (that is, the highest averaged predicted probability).

- REWA

  REWA is the standard Randomized Exponential Weights Algorithm.

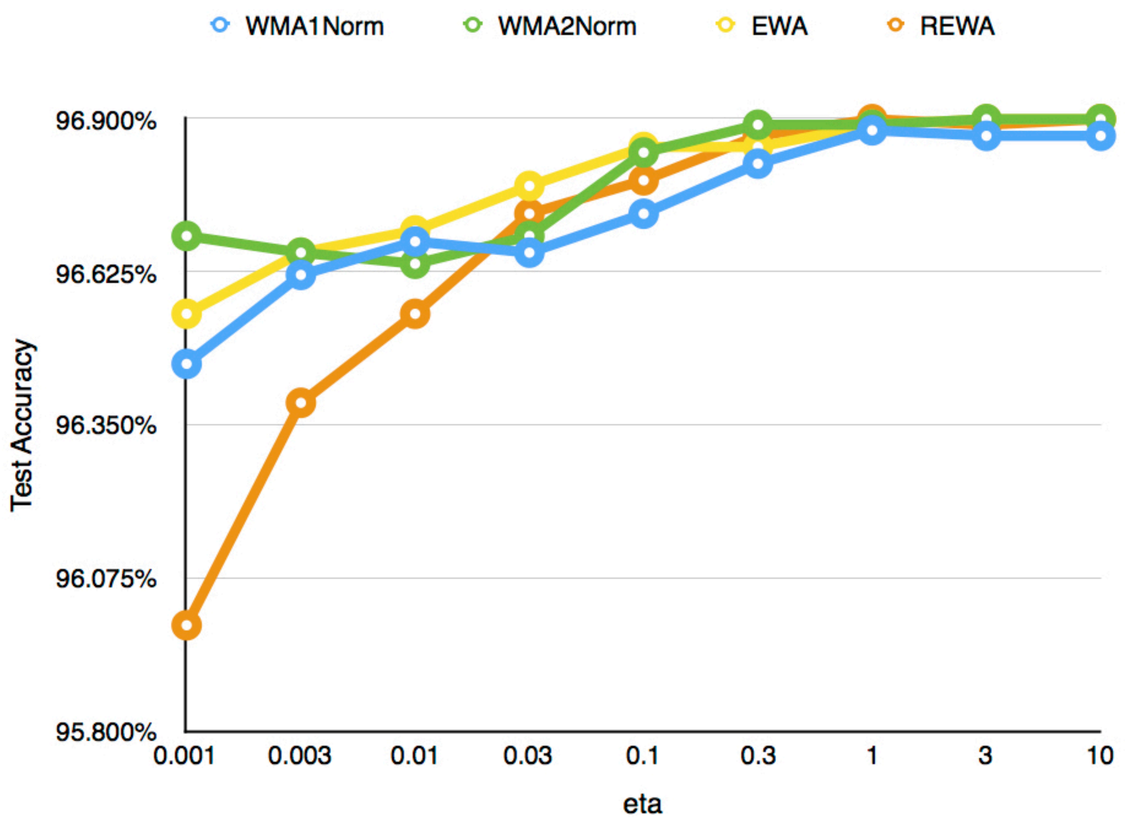
It is noteworthy that, for the sake of uniformity, I use the learning rate η for all six algorithms. For the first four algorithms, $\beta = e^{-\eta}$.





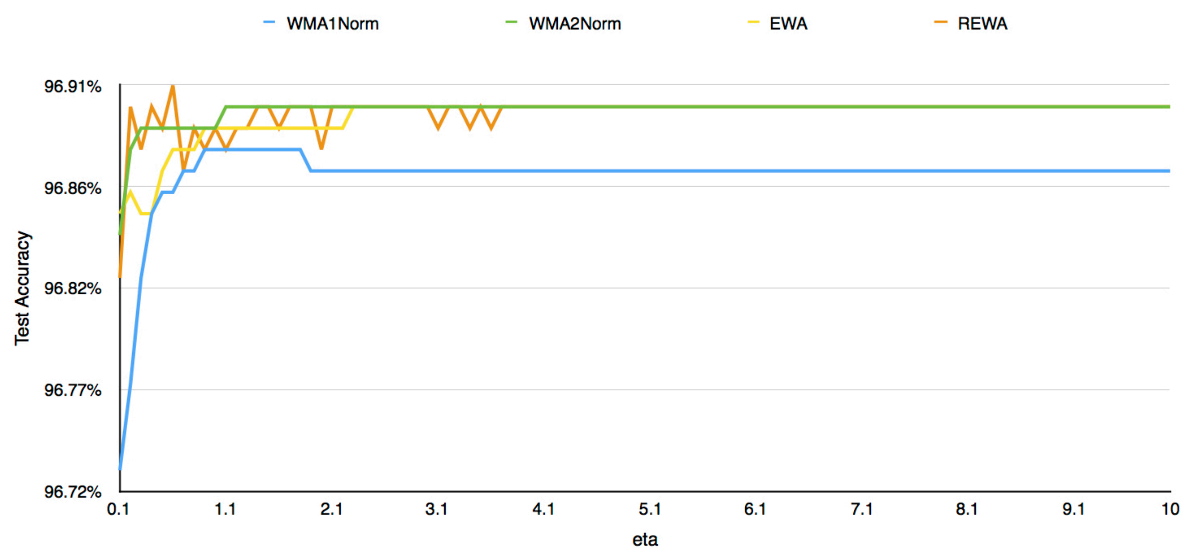
The two graphs above compares the performance of WMA1 and WMA1Norm, WMA2 and WMA2Norm. The horizontal axis is logarithmic. I originally implemented WMA1 and WMA2, and, seeing that their test accuracy drops drastically when η is relatively large, went on to implement WMA1Norm and WMA2Norm.

These two graphs clearly show that for relatively small values of η, the algorithms with unnormalized and normalized weights perform almost equally well; while for relatively large values of η, the algorithms with normalized weights retain steadily high accuracies, but those with unnormalized weights drop significantly in accuracy. One major reason is unnormalized weights are susceptible to a numerical problem. When η is relatively small, the update is not aggressive, so this problem does not show; but when η is relatively large, and the update is aggressive, the unnormalized weights will eventually become so close to zero as the number of updates accumulates that it is impossible to distinguish the label carrying the most weight with the others, even those with zero weight. Therefore, WMA1 and WMA2 are not competitive online learning algorithms for model averaging.



The graph above compares the performance of WMA1Norm, WMA2Norm, EWA and REWA on a set of values of η. The horizontal axis is logarithmic. This graph shows that there is a general tendency for the performance of each algorithm to improve as the value of η grows bigger, and all four algorithms seem comparable in performance when η is relatively large.

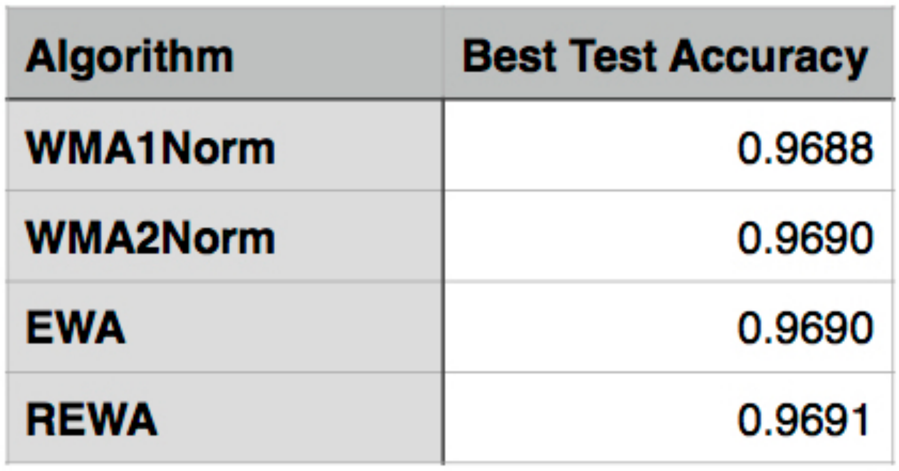
This comparison, though stretching over a satisfying range of values of η ($e^{-0.01} = 0.99900$, $e^{-10} = 4.5400 * 10^{-5}$), is not fine-grained enough to display each algorithm's properties.



The graph above compares the performance of WMA1Norm, WMA2Norm, EWA and REWA in a more fine-grained fashion. I divided the interval [0, 10] into 100 intervals with equal length of 0.1, and experimented with η as the value of each endpoint (except 0). The horizontal axis here is linear.

A number of interesting conclusions can be drawn from this graph.

- There is indeed a general tendency for the performance of each algorithm to improve as the value of η grows bigger.
- WMA1Norm performs significantly worse than the other three algorithms, presumably because it updates with zero-one loss instead of logistic loss. (It is noteworthy, however, that WMA1Norm is still a very competitive algorithm, noting that the vertical axis of the graph presents as narrow a range as 0.19%. From the table below it can be observed that the best test accuracy of WMA1Norm is very close to those of the other three algorithms.)



Algorithm	Best Test Accuracy
WMA1Norm	0.9688
WMA2Norm	0.9690
EWA	0.9690
REWA	0.9691

- Although it has the highest of all best test accuracies, REWA performs relatively unstably due to randomization, which is apparent from its zigzagging performance curve. (Again, because the vertical axis of the graph has a very narrow range, this slight instability can be safely ignored for general purposes.)
- WMA2Norm, EWA and REWA converge to an equally good test accuracy (96.90%) after η grows bigger than 3.6.
- Quite surprisingly, WMA2Norm performs better than EWA with relatively small values of η before their convergence, which is yet inexplicable to me with my current knowledge.

- All four online learning algorithms perform better than the single best expert selected on values of η that are not "too small" (i.e. the updates are reasonably aggressive and thus manifest the property of model averaging), apparant from the fact that all four performance curves are higher than the 96.77% reference line, except for WMA1Norm with very small values of η.

Future Work

In the series of experiments reported here, I only experimented with neural networks with a single hidden layer of 50 units, which is chosen from a range intuitively. In the future, I would like to experiment with more architectures. Moreover, I would like to research deeper into the pruning technique for determining the optimal number of hidden units, and try implementing it myself.

Simply assigning the first 50,000 examples in the official training set to the actual training set and the last 10,000 to the validation set, though it seems to be common practice, may not be convincing enough. I would like to implement K-fold cross validation and see if it yields the same results.

I would also like to conduct further research and experiments to determine the factors contributing to WMA2Norm performing better than EWA with relatively small values of η.

Reference List

LeCun, Y., Cortes, C., and Burges, C.J.C., 1998, The MNIST database of handwritten digits, http://yann.lecun.com/exdb/mnist/ (July 31, 2017)

Rasmussen, C.E., 2002, fmincg[Source Code]