Title

Levenberg—Marquardt multi-classification using hinge loss function

Authors

Ozyildirim, Buse Melis

Kiran, Mariam

Publication Date

2021-11-01

DOI

10.1016/j.neunet.2021.07.010

Levenberg-Marquardt Multi-Classification using Hinge Loss Function

Buse Melis Ozyildirim[a,*], Mariam Kiran[b]

[a]*Department of Computer Science Cukurova University Adana, Turkey*
mozyildirim@cu.edu.tr
[b]*Scientific Networking Division, Lawrence Berkeley National Laboratory, Berkeley, CA*
mkiran@es.net

Abstract

Incorporating higher-order optimization functions, such as Levenberg-Marquardt (LM) have revealed better generalizable solutions for deep learning problems. However, these higher-order optimization functions suffer from very large processing time and training complexity especially as training data sets become large, such as in multi-view classification problems, where finding global optima is a very costly problem. To solve this issue, we develop a solution for LM-enabled classification with, to the best of knowledge first-time implementation of hinge loss, for multiview classification. Hinge loss allows the neural network to converge faster and perform better than other loss functions such as logistic or square loss rates.

We prove our method by experimenting with various multiclass classification challenges of varying complexity and training data size. The empirical results show the training time and accuracy rates achieved, highlighting how our method outperforms in all cases, especially when training time is limited. Our paper presents important results in the relationship between optimization and loss functions and how these can impact deep learning problems.

Keywords: Neural networks, Levenberg-Marquardt, Hinge Loss, Loss functions, Classification

*Corresponding author
Email address: mozyildirim@cu.edu.tr (Buse Melis Ozyildirim)
URL: mozyildirim@cu.edu.tr (Buse Melis Ozyildirim)

1. Introduction

Neural network research is creating an impact in many engineering and real-world applications. Several methods are being explored that allow neural networks to train and converge faster to optimal solutions while achieving the best accuracies. One methodology to improve prediction accuracy is to explore multiple higher-order optimization functions, replacing single order Stochastic Gradient Descent (SGD), but these come with higher computational and memory costs. SGD has been favored as being computationally inexpensive, particularly in large-scale classification problems [1][2] or where training data streams in small samples, but its stochasticity introduces much uncertainty in the results [3]. Other optimization approaches such as Conjugate Gradient (CG) and Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L- BFGS), are also simple to train and optimally converge, sometimes improving classification accuracy [4]. In comparison to these approaches, in this paper, we focus on using a different higher-order optimization function, the Levenberg-Marquardt algorithm particularly due to the complexity of using quasi-Hessian matrix calculations in classification problems.

Higher-order approaches have proven to be more robust in finding global optima solutions [5]. However, these approaches suffer from very slow convergence and get lost in parameter space too often [6]. One way to work around this issue is to vary the loss functions being used, to allow the neural networks to converge faster. In their seminal work, Rosasco et al. [7] showed how loss functions can impact statistical learning in both classification and regression problems. They showed that when Ivanov type regularization is utilized, hinge and logistic loss functions can lead to better convergence than classic square loss.

Loss functions or error functions, like cross-entropy measures, are commonly used to train neural networks. These become very complicated in multi-view classification problems as it requires one-hot-shot encoding to parse the multiple labels. Other examples, like mean square error only applies in regression

problems where value differences are calculated to train the neural network. However, when loss functions are coupled with optimizers, in LM approaches, one cannot use cross entropy-based loss functions because it is more suited towards batch training (as in SGD). LM uses the Hessian matrix to calculate the global optima and would benefit greatly using quadratic loss functions such as Hinge loss [7]. In this paper, we investigate how this relationship of optimization and loss functions interplays, particularly when challenged in multi-view classification problems.

The importance of loss functions has been discussed in different studies. In [8], authors showed that defining good loss function may eliminate complex neural network structures and training requirement. They defined mutual channel loss and applied for one of the challenging computer vision problem, fine grained image classification. The proposed loss function uses two channels to represent diversity and discriminality. Another approach focused on loss function rather than neural network structure is [9]. Authors integrated a regularization coefficient to the cross-entropy loss and named this loss function as Dual Cross-Entropy Loss. They aimed limiting probability of the misclassified data points and eliminate gradient vanishing. They tested their method on Stanford Cars-196 dataset and CIFAR-10 datasets and proved its efficiency.

Duan et al. [10] showed how SVMs can use hinge loss used for binary classification to be extended to multi-view classification problems. In [11], use of hinge loss-SGD with convolutional neural networks was proposed and tested on German Traffic Sign Recognition Benchmark. The neural network structure had 1162284 trainable parameters and reported results showed that proposed method had achieved 99.65% accuracy. In [12], an improved Hinge loss was proposed and used as loss function of feature selection approach. The proposed approach eliminated the difficulty of optimizing multiclass Hinge loss and sparsity regularization issue. Authors reported that with their methodology, they achieved optimal convergence rate for smooth problems.

In this paper, we adopt this method and develop hinge loss with LM optimizers to develop solutions that can converge faster and produce optimal accuracy.

Demonstrating this, to the best of our knowledge, first implementations, our experiments in multi-view classification problems show that when the training budget is low, hinge loss with LM can produce optimal accuracy quicker than other combinations of optimizer and loss functions. Specifically, our contributions are,

- We investigate the relationship of optimization and loss functions in multi-classification problems of varying complexities.
- To the best of our knowledge, we develop and present the first-time implementation of hinge loss with Levenberg-Marquardt optimizer, showing comparisons with other loss functions.
- We present experimental results in classification problems from OpenML, showing that when the training budget is low, hinge loss and LM can provide optimal results.

The rest of the paper has been organized as follows: Section 2 describes the related work in optimizers and loss functions research. Section 3 presents the background on the preliminaries used in this paper, with Section 4 showing how the solution is implemented. Section 5 presents the experiment details and results obtained. Finally, section 6 discusses what our experiments reveal.

2. Related Work

Achieving faster convergence using a modification of the optimization algorithm has shown success [13]. Wilamowski et al. [14] modified the second-order derivatives, like the Levenberg-Marquardt algorithm, to improve the quasi-Hessian matrix calculations. Compared to SGD, methods using Gauss-Newton can help learn global convergence by quadratic derivatives [15]. These approaches utilize inverse of Jacobian or Hessian matrices to speed up convergence finding global optima quickly. Conjugate Gradient can produce solutions in n steps for n-dimensional unconstrained quadratic problems, however,

it requires estimation of Hessian vector products affecting its performance on the estimation approach. Another approach based on the Hessian matrix, L-BFGS, presents memory challenges and can only work with limited data sets. The Levenberg-Marquardt is based on inverse curvature matrix calculation and is efficient for small and medium-scaled problems but too expensive for large datasets. LM also computes local minima of multivariate functions using an iterative approach to solve nonlinear least squares functions, utilizing both gradient descent and Gauss-Newton approaches. LM have become important optimization function in recent studies. In [16], LM was used for training multilayer neural network architecture for probabilistic estimation method of brake pressure. The performance results show that LM is an efficient way for training MLP architectures. In [17], hybrid of LM and genetic algorithm was proposed for optimized automatic spectrometer design and provided better performances than that of LM and genetic algorithm individually. LM algorithm is suitable for improvements. In [18], damping factor, which is the key element of LM, was optimized and tested for identifying temperature-dependent thermal conductivities. Authors showed that their proposed method increased the accuracy and stability of LM.

The role of loss functions in training deep learning models and optimizers can impact the results. Rosasco et al. [7] showed how choosing a different loss function can impact the convergence rates. Their results showed that hinge loss was able to achieve the best results and produce bounded results for classification problems in theoretical settings. Berger [19] argued that using non-Bayesian methods such as minimax can lead to the idea of modeling loss functions as regret in decision making and develop solutions similar to how humans make decisions. Christoffersen et al. [20] showed that for any given model, the loss function used in parameter estimation and model evaluation should be the same, otherwise suboptimal parameter estimates may be affected. Hennig et al. [21] formalized the loss function as a mathematical decision problem, showing how subjective decision problems are affected. However, in all of the above papers, loss function and their impact are explained in theoretical settings.

More recent work [22] shows a variety of loss functions in distribution learning and density estimation, showing that log loss can satisfy many desired criteria. On the other hand, recent works have taken regularization factor, which provides smooth decision surface into account instead of changing loss function types. Hence, regularization have become important in machine learning models [23]. Arguably, Deming and Taleb [24] argued that one needs to pay more attention to loss functions, designed specific to the problem, can produce effects in more manners other than classical smooth, continuous, symmetric, differentials cases.

In this paper, we introduce the novel concept of using Hinge loss with the LM optimization method for classification problems. Our proposed approach takes advantage of LM for classification problems by using one of the efficient convex loss functions. Although LM is an efficient approach in itself, it does not apply to cross-entropy function which is often generally used for classification problems. On the other hand, Hinge loss is one of the efficient convex loss functions SVM models have utilized [10]. In this study, we develop multi-view classification problems as in SVM methods, but utilize optimization with the LM approach and compare results with other techniques.Main contributions of this study are to investigate loss model efficiency and to utilize Hinge loss which is generally used with SVM models, with LM optimization function. According to the test results, it can be claimed that Hinge loss with LM is one of the alternative approach to cross entropy.

3. Background

In neural network training, optimization functions play key roles in determining the best-trained weight to minimize the network prediction accuracy. The Levenberg-Marquardt algorithm provides a numerical solution to minimize a non-linear function by combining the gradient descent and Gauss-Newton method. It uses a parameter to decide the step size, taking large values first and then smaller values, to allow convergence from any initial state near-global

minima. It is known as the most efficient training algorithm [25].

3.1. Stochastic Gradient Descent

Gradient descent is a common optimization algorithm for training neural networks. Based on the idea of updating the tunable parameters to minimize the objective function, we use the learning rate to converge the loss function.

Let L be the objective function and w the model parameter. At every timestep t the w is updated based on following equation,

$$w_t = w_{(t-1)} - \alpha \, \partial L / \partial w \tag{1}$$

There are two types of gradient descent implementations: stochastic (SGD) and vanilla gradient descent. While vanilla gradient descent updates parameters after processing the complete training dataset, stochastic gradient descent allows us to update the parameters sequentially after each training sample. This causes the vanilla gradient descent to converge slowly compared to stochastic gradient descent. Despite this speed, stochastic gradient descent suffers from fluctuation and causes the learning rate to decrease very slowly. This problem can be resolved using momentum techniques as it uses the past parameter updates to inform the current decisions, thereby reducing the fluctuations. If parameter updates of two sequential timesteps are in similar directions, the momentum increases and decreases the updates. Examples of these include Nesterov Accelerated Gradient, and others using adaptive learning rates such as Adagrad, Adadelta, RMSprop, and Adam. Comparative studies have revealed that Adadelta, Adagrad, and RMSprop approaches provide the best convergence rates [26]. The algorithms of compared optimizers Conjugate Gradient and BFGS are given in Appendix section. The focused approaches such as Levenberg-Marquardt, Hinge Loss are discussed in further subsections.

3.2. Levenberg-Marquardt

Both, SGD and CG are first-order approaches, that give impressive results in classification challenges [27]. Levenberg-Marquardt (LM) is an optimization

method for solving the sum of squares of non-linear functions. It is a combination of gradient descent and the Gauss-Newton method, starting with gradient descent and as it comes close to the solution, like a Gauss-Newton method, using a damping factor. While larger damping factors result in gradient descent behavior, small damping factor lead to Gauss-Newton method convergence.

Let d be a target value vector for m data points and y be the output vector of the fitting function, L is defined as:

$$L = \sum_{j=1}^{m} (d_j - y_j)^2 \tag{2}$$

According to the gradient descent approach, the update is as,

$$\Delta w_t = \alpha J^T W (d - y) \tag{3}$$

where J denotes Jacobian matrix of $\frac{\partial y}{\partial w_t}$ and W is the weight vector.

According to the Gauss-Newton approach, the update is as,

$$\Delta w_t = \left(J^T W J\right)^{-1} J^T W (d - y) \tag{4}$$

Since LM combines these methods, the update rule is,

$$\Delta w_t = \left(J^T W J + \lambda I\right)^{-1} J^T W (d - y) \tag{5}$$

where I denotes the identity matrix and λ represents the damping factor. If the new parameter results in lower errors than previous ones, the new parameter values are accepted and λ is decreased. Otherwise, the new parameter set is rejected and λ value is increased [28].

3.2.1. LM in classification problems:

LM is an optimization algorithm that works with a squared loss function. For classification problems, the most commonly used loss functions are cross-entropy and even squared loss function. One may convert the classification problem into a regression problem. This conversion requires one-hot encoding as shown in Algorithm 1.

Algorithm 1: LM in Classification Problems:

Apply one-hot encoding to the classes.

Calculate the output values of the neural network.

Calculate loss function by applying one of the methods given above.

Calculate the update on weights applying the LM approach.

Apply updates to the weights.

With N classes, there should be N outputs whose values are between {0 or 1} or {0.1 or 0.9}. Using {0.1 or 0.9} avoids getting stuck in neuron problems. These are summarized in the next section as,

- Sum of the squared distance of all outputs (opt 1).
- Squared distance of desired output (opt 2).
- Squared distance of predicted output (opt 3).
- Sum of the squared distance of both desired and predicted outputs (opt4).

3.3. Loss Functions

3.3.1. Squared Loss function or opt1:

When the squared loss function is chosen for classification problems, it is defined as a sum of square of all the output values (Eq. 6).

$$loss = \sum_{j=1}^{N} (d_j - t_j)^2 \tag{6}$$

where N is number of classes, d_j represents desired output value, and t_j represents predicted value for j^{th} class. We call this $opt1$ in this study.

3.3.2. Variations of Squared Loss function or opt2, opt3, opt4:

Other variations of the squared loss function are also used for improving classification training. For example, using squared distance of desired output as

*opt*2 (Eq. 7), squared distance of predicted output as *opt*3 (Eq. 8), or the sum of squared distance of both desired and predicted outputs as *opt*4 (Eq. 9).

$$loss = (d_j - t_j)^2 \tag{7}$$

where j^{th} class is the correct class, d_j represents desired output value for correct class, and t_j represents predicted value for the correct class.

$$loss = (d_i - t_i)^2 \tag{8}$$

where i^{th} class is the predicted class, d_i represents desired output value for predicted class, and t_i represents predicted value for chosen class.

$$loss = (d_j - t_j)^2 + (d_i - t_i)^2 \tag{9}$$

where i^{th} class is the predicted class, d_i represents desired output value for predicted class, and t_i represents predicted value for chosen class and j^{th} class is the correct class, d_j represents desired output value for correct class, and t_j represents predicted value for the correct class.

3.3.3. Hinge Loss:

Hinge loss is introduced for maximum margin classification problems such for example in Support Vector Machines [10]. Assuming that desired outputs are (-1 or 1), hinge loss is defined as,

$$hl = max\left(0, 1 - d \cdot t\right) \tag{10}$$

where d is desired value and t is the predicted value. Since hinge loss is non-differentiable, we use a smoothed version to be coupled with optimization functions. One of the frequently used variations of this is the squared hinge loss,

$$hl_2 = \left(max\left(0, 1 - d \cdot t\right)\right)^2 \tag{11}$$

For multi-view classification problems, hinge loss variations can be defined,

$$hl = max\left(0, 1 + (w_t x - w_d x)\right) \tag{12}$$

where $w_t x$ and $w_d x$ are model parameters.

3.4. LM and its Relationship with Loss functions

LM has been utilized with SVMs to develop improved accuracy of classification problems. The technique, however, cannot be used with many forms of cross-entropy or log-losses which are common approaches in training neural networks [29]. This is because of the limitations of the search space in how LM computes its convergences. The Jacobian matrix becomes too huge, requiring a lot of memory and the cross-entropy cannot compute optimal weight approximations.

On the other hand, hinge loss is based on solving a convex problem that can provide innovative solutions to the problem of the LM approach. Since on its own hinge loss is non-differentiable, we need to calculate subgradients to efficiently use it. Using the squared hinge loss function with l2-regularization, the challenge is converted into a least square problem, which can make it easy to use LM as a solver. This can be combined with multi-view classification, Eq. 13.

$$L = \lambda ||w||_2^2 + max\left(0, 1 - (w_t x - w_d x)\right)^2 \tag{13}$$

4. Proposed Approach

The proposed solution utilizes squared hinge loss function in a multilayer neural network using semi-linear hidden units.

Let d be a target value ϵ 0,1 and y be the output of the sigmoid function, L is defined as in 14:

$$L = \lambda ||w||_2^2 + max\left(0, 1 - (d - y)\right)^2 \tag{14}$$

The λ parameter plays critical role for convergence. The value of the λ determines the flexibility of the convergence. Flexibility increases the value of w. To minimize the loss function, w should be small values. If λ is zero, the model will be least squares. The w obtained with least square will be a scale

factor and it may not be solution for many problems. On the other hand, large λ values cause $w = 0$ which is also not a solution. Therefore, choosing optimal λ is critical. There are different approaches to determine best regularization parameter [30]. One of those studies uses linear combination of different ranker scores which represent different values of λ. Ranking provides solution to regression problems but it assigns labels y to the values x which is element of the input space. Ranking does not deal with the value of y, it deals with the relative ranks of the elements x. Authors took the advantage of ranking method to decide regularization parameter in [31]. They showed the advantage using linear functional strategy ranking algorithm over a fixed regularization parameters with experimental examples [31]. In this study, we applied grid search to find optimal λ. Applying swarm based approaches and linear functional strategy are aimed as future work. One limitation of using hinge loss is to have a linear activation function in the hidden units. Here, ReLU can be considered as semi-linear functions that eliminate these negative values. Another limitation of the proposed approach is that hinge loss is appropriate for binary classification problems. To solve this problem, SVM's multi-view solutions such as OneVsOne (OVO), OneVsAll (OVA) are adopted. OVO approach creates pairs from classes, iterates over all combinations and applies voting system at the end. On the other hand, OVA approach iterates N times where N is the number of class and each time one class is chosen as opponent of all other classes problem has.

Although SVM is simulated for the proposed solution, the main difference between them is that while SVM output is only {0, 1}, in our approach the output is between and including (0,1).

5. Experiment

We investigated the performance of optimizer and loss functions. For multiclass problems, a squared loss can be defined in different ways. Four different variations were used - a sum of the squared distance of all outputs (opt1), squared distance of desired output (opt2), squared distance of predicted output

Algorithm 2: Proposed Approach:

if The problem is multiclass **then**

Choose OVO or OVA methodology.

Create sub binary problems according to the chosen methodology.

end if

for Each binary problems: **do**

Assign 0.9 or 1 to the desired value to the output neuron representing the correct class and 0.1 or 0 to another output neuron.

Calculate the output values of the neural network.

Calculate loss function shown in Eq. 14.

Calculate the update on weights applying the LM approach.

Apply updates to the weights.

end for

(opt3), and a sum of the squared distance of both desired and predicted outputs (opt4). These approaches are compared with the cross-entropy loss function with the SGD optimizer and the proposed approach.

5.1. Datasets and Experimental Setup

In this study, we compared performances of hinge loss utilized, and variations of squared error loss function utilized optimization functions. Those optimization functions are conjugate gradient, LBFGS, and LM. Experiments are performed on five datasets obtained from openML. Small, medium, and large-sized datasets were chosen with details given in Table 1. We normalize the data set using 70% for training and 30% as test data. Additionally, to ensure consistency, the same seed value is used.

MLP architecture used: We use the architecture, Input Layer, Hidden Layer with 50 neurons and ReLU activation, Hidden Layer with 35 neurons, and ReLU activation, Sigmoid Layer for output. The learning rate was chosen as 0.01. For the cross-entropy tests, the momentum technique was applied with a momentum value of 0.99, using the Nesterov strategy.

Table 1: Details of the Architecture - Classification tasks.

	Iris	**Glass**	**Vehicle**	**Abalone**	**PenDigit**	**Mnist**	**IMDB**
Input	4	9	18	8	16	784	50
NN architecture	50-35-3	50-35-6	50-35-4	50-35-28	50-35-2	200-70-10	50-35-2
Size	Small	Small	Medium	Small	Small	Large	Large
Features	4	9	18	8	16	784	50
Training data set	105	150	7695	593	2924	60000	25000
Test data set	35	64	3297	253	1253	10000	25000
Classes/output	3	6	4	28	2	10	2

Variations of squared loss function are labeled opt1 - opt4. **opt1:** sum of squared distance between target and output of the model. **opt2:** squared distance between target and output of target neurons. **opt3:** squared distance between the value of output neuron and the target value of output neurons. **opt4:** sum of the squared distance between target output and an output value of the target node and model output value and value of a related node.

5.2. Simulation Results

The accuracy results are summarized in Tables 2, 3. The loss performances of the various optimization functions are shown in Figures 1 - 7.

It is observed with the different squared loss functions, the predicted output (opt3) provides the worst results in general. Here, the optimizer tries to decrease the value of incorrect classes, not considering the actual output, leading to incorrect assumptions of the winning classes. However, Figures 6, 7 show that it performs better for larger datasets and models. Opt4 provides some better results, using two objectives - while it increases the output value of the correct class, it decreases the output of the winner. Overall, with increased objectives opt1, obtains worse results for overall accuracy.

When opt1 results are compared to cross-entropy results, there is a dramatic

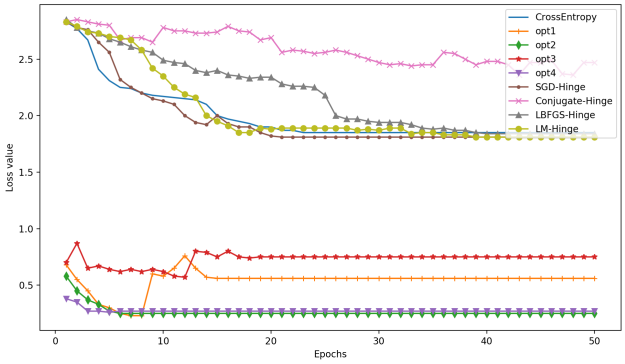

Figure 1: Abalone.

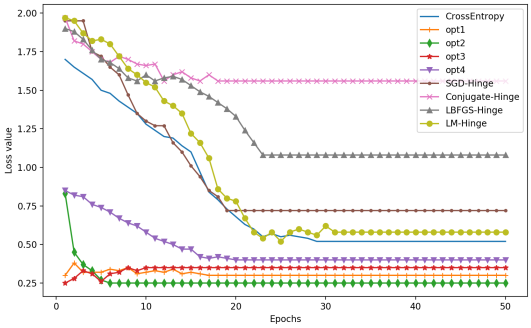

Figure 2: Glass.

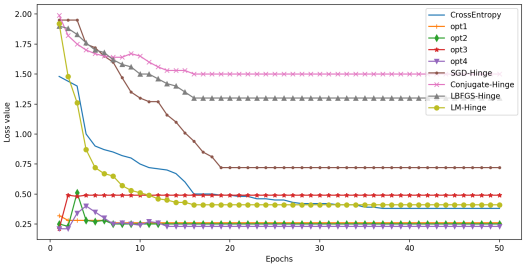

Figure 3: Iris.

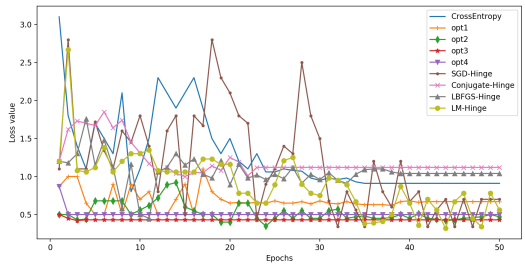

Figure 4: PenDigit.

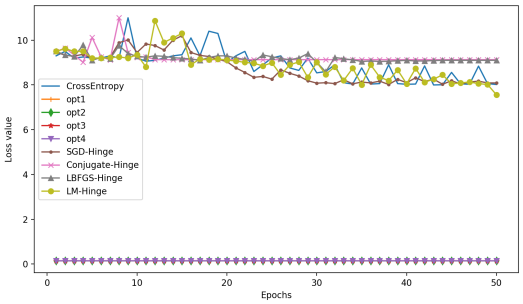

Figure 5: Vehicle.

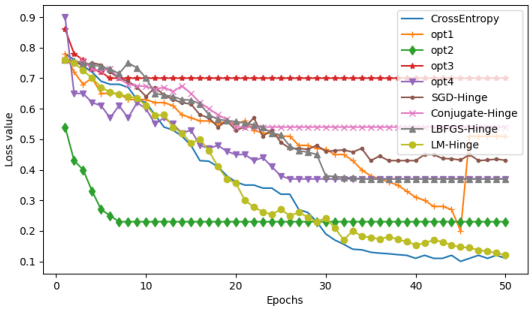

Figure 6: IMDB.

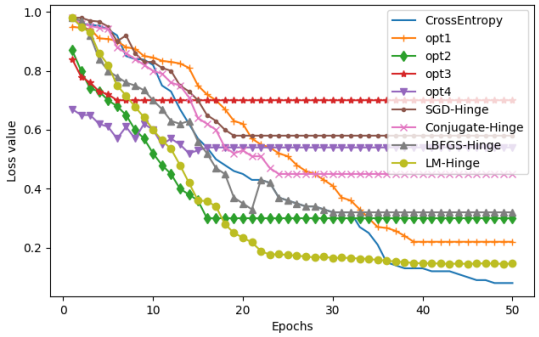

Figure 7: MNIST.

decrease in model accuracy. It is also observed that when the number of classes increases, the decrease in accuracy becomes more dramatic.

For some datasets such as PenDigit and Vehicle, most of the approaches fluctuated until epoch 48 or 49. For those datasets algorithms ran 5 more epochs and decreases in fluctuation frequencies were observed. To provide consistency, Figures are cropped at fiftieth epoch.

Using Levenberg-Marquardt as Optimizer: Although it is an efficient optimizer utilizing second-order derivatives, for classification problems takes a long time and may cause reductions inaccuracy. The reason behind this is that LM is directly applied to MSE and applying it to classification problems requires additional efforts. Moreover, it is known that LM is utilized to solve SVM problems with some modifications. If the squared hinge loss function with l2-regularization is used for classification purposes, the problem is converted into a least square problem, allowing one to use LM as a solver.

The multiclass Hinge loss function provides the closest results to cross-entropy results. Performances of different optimization functions utilizing the

Table 2: Final accuracy results (shown in (%)) across all loss and optimizer functions combinations after training for 50 epochs.

Loss Functions	**CrossEntropy**	**opt1**	**opt2**	**opt3**	**opt4**
with Optimizer	*SGD*	*SGD*	*LM*	*LM*	*LM*
Data set					
Iris	**95.45**	45.45	51.11	35.55	62.22
Glass	**67.74**	9.68	30.77	8.88	46.15
PenDigit	**89.62**	65.88	78.83	57.3	64.9
Vehicle	**27.27**	14.88	20.85	14.88	26.37
Abalone	**31.83**	1.17	28.93	1.17	29.52
Mnist	**98.2**	82.9	79.6	67.5	71.36
IMDB	**87.9**	48.54	57.4	26.4	52.65

Hinge loss function are compared in Table 3. This table aims to answer two research questions we have. First one is which Hinge Loss - Optimizer variation will catch the baseline Cross Entropy results, second one is which optimizer is more beneficial for using with Hinge Loss. According to the test results, LM and SGD performances are very close to cross-entropy results. Classification results of PenDigit and Vehicle datasets are better than cross-entropy classifications. It is also proved that Hinge Loss-LM combination performed better than Hinge Loss -SGD variation for larger data sets. Moreover, the Hinge loss function provides faster convergence than cross-entropy and other MSE options.

Computational time requirements for Hinge loss utilized methods are given in Table 4. All tests were run on the same hardware architecture. Computational time was measured by utilizing time.process_time() function from Python's time library.

According to the Table 4, while LBFGS takes a quite a long time, SGD takes the least time. LM takes more time than SGD. However, computational times given in Table 4 show time need for 50 epoch and for most of the datasets LM converges to optimal values before 50 epoch. Since LM converges before SGD,

Table 3: Final accuracy results (shown in (%)) across all loss and optimizer functions combinations after training for 50 epochs.

Loss Functions	**Hinge-L2**	**Hinge-L2**	**Hinge-L2**	**Hinge-L2**
with Optimizer	*SGD*	*Conjugate*	*LBFGS*	*LM*
Data set				
Iris	**77.45**	43.27	62.73	75.55
Glass	62.74	23.68	35.77	**66.15**
PenDigit	**93.62**	74.46	78.83	89.46
Vehicle	25.2	12.66	14.85	**31.27**
Abalone	**31.83**	5.97	**31.83**	**31.83**
Mnist	70.42	75.56	82.34	**97.34**
IMDB	45.28	47.34	51.34	**86.45**

the time requirement will be the same or less. The main reason behind LM's time requirement is that time requirement for Hessian calculation is proportional to both number of classes and the number of parameters defined in neural network architecture.

6. Conclusions

In this study, the applicability of hinge loss and LM to classification problems are analyzed.

In this paper, the Hinge loss function is converted to the squared multiclass Hinge loss function, with l2-regularization added to it. All the MSE options and new forms of the multiclass squared hinge loss function with l2-regularization were implemented and compared with a cross-entropy loss function. Moreover, squared hinge loss function with l2-regularization is applied to other optimization functions. According to the test results, while opt3 and opt1 perform worse, opt4 was the optimal solution for applying MSE to classification problems. Multiclass Hinge loss function with l2 regularization provided the closest results to cross-entropy results. Moreover, the squared multiclass Hinge loss function with

Table 4: Computational time comparison (sec) for 50 epochs training

Loss Functions	**Hinge-L2**	**Hinge-L2**	**Hinge-L2**	**Hinge-L2**
with Optimizer	*SGD*	*Conjugate*	*LBFGS*	*LM*
Data set				
Iris	**0.00194**	0.0087	0.127	0.00573
Glass	**0.00453**	0.0675	0.189	0.167
PenDigit	**0.245**	2.453	16.78	1.976
Vehicle	**1.687**	5.674	25.46	12.34
Abalone	**0.00786**	0.0885	0.276	0.0674
Mnist	**196.645**	588.363	1087.463	323.234
IMDB	**123.645**	472.34	956.34	285.45

l2 regularization performed well with other optimization functions.

Our proposed approach investigates the relationships between optimization and loss functions and how it affects the accuracy and training time of complex problems. From the results, we can conclude that using Hinge loss (with LM) outperforms the much-favored cross-entropy (with SGD), by reaching optimal loss function very quickly in the same computational cost of the training epochs. However, when trained for longer, such as 50 epochs, we found that cross-entropy can produce much better accuracy results. The hinge loss function converges faster than other loss functions. It is also applicable to other optimization functions. In terms of computation, the Hessian function takes much longer than the other simple order optimization functions. But with the added advantage of taking lesser training epochs to reach equilibrium, it warrants further research in investigating how higher-order Hessian matrix calculations can be made quickly to improve the performance of these techniques. Furthermore, in future experiments, we will analyze Levenberg-Marquardt's performance in more complex neural network architectures such as in problems of reinforcement learning to determine the convergence properties of the algorithms and its comparison with more commonly used optimization and loss approaches.

Contribution and novelty of this work can be summarized as follows: investigating the effect of loss function calculation types for multi-classes, use of Hinge loss and l2-regularization with different optimization functions, investigating applicability of LM-Hinge loss on multi-class classification problems and its performance. According to the test performances, although computational complexity of LM is higher than that of SGD, LM-Hinge loss combination converges to optimum before SGD converges and can provide almost the same classification results.

7. Acknowledgements

This material is based upon work supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research, Office of Science Early Career Research Program for 'Large-scale Deep Learning for Intelligent Networks' Contract no FP00006145.

References

[1] E. Bottou, Stochastic gradient learning in neural networks, in proceeding in neural networks (1991).

[2] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324. doi:10.1109/5.726791.

[3] S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, H. Asai, A stochastic quasi-newton method with nesterov's accelerated gradient (2019). arXiv:1909.03621.

[4] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, A. Y. Ng, On optimization methods for deep learning, in: Proceedings of the 28th International Conference on International Conference on Machine Learning, ICML'11, Omnipress, USA, 2011, pp. 265–272.
URL http://dl.acm.org/citation.cfm?id=3104482.3104516

[5] O. Nelles, Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models, Springer, 2001.

[6] M. K. Transtrum, J. P. Sethna, Improvements to the levenberg-marquardt algorithm for nonlinear least-squares minimization (2012). arXiv:1201.5885.

[7] L. Rosasco, E. De Vito, A. Caponnetto, M. Piana, A. Verri, Are loss functions all the same?, Neural Comput. 16 (5) (2004) 1063–1076. doi:10.1162/089976604773135104.
URL https://doi.org/10.1162/089976604773135104

[8] D. Chang, Y. Ding, J. Xie, A. K. Bhunia, X. Li, Z. Ma, M. Wu, J. Guo, Y.-Z. Song, The devil is in the channels: Mutual-channel loss for fine-grained image classification, IEEE Transactions on Image Processing 29 (2020) 4683–4695. doi:10.1109/tip.2020.2973812.
URL http://dx.doi.org/10.1109/TIP.2020.2973812

[9] Dual cross-entropy loss for small-sample fine-grained vehicle classification, Li Xiaoxu and Yu, Liyun and Chang, Dongliang and Ma, Zhanyu and Cao, Jie 68 (2019) 4204–4212.

[10] K.-B. Duan, S. S. Keerthi, Which is the best multiclass svm method? an empirical study, in: N. C. Oza, R. Polikar, J. Kittler, F. Roli (Eds.), Multiple Classifier Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 278–285.

[11] J. Jin, K. Fu, C. Zhang, Traffic sign recognition with hinge loss trained convolutional neural networks, IEEE Transactions on Intelligent Transportation Systems 15 (5) (2014) 1991–2000. doi:10.1109/TITS.2014.2308281.

[12] H. Peng, C.-L. Liu, Discriminative feature selection via employing smooth and robust hinge loss, IEEE transactions on neural networks and learning systems 30 (3) (2019) 788—802. doi:10.1109/tnnls.2018.2852297.
URL https://doi.org/10.1109/TNNLS.2018.2852297

[13] J. Shawe-Taylor, D. Cohen, Linear programming algorithm for neural networks, Neural Networks 3 (1990) 575–582. doi:10.1016/0893-6080(90)90007-8.

[14] B. M. Wilamowski, H. Yu, Improved computation for levenberg–marquardt training, IEEE Transactions on Neural Networks 21 (6) (2010) 930–937. doi:10.1109/TNN.2010.2045657.

[15] M. Bun, Applications of the levenberg-marquardt algorithm to the inverse problem (2009).
URL https://sites.math.washington.edu/~reu/papers/2009/mark/reupaper.pdf

[16] C. Lv, Y. Xing, J. Zhang, X. Na, Y. Li, T. Liu, D. Cao, F. Wang, Levenberg–marquardt backpropagation training of multilayer neural networks for state estimation of a safety-critical cyber-physical system, IEEE Transactions on Industrial Informatics 14 (8) (2018) 3436–3446. doi:10.1109/TII.2017.2777460.

[17] K. H. Cheong, J. M. Koh, A hybrid genetic-levenberg marquardt algorithm for automated spectrometer design optimization, Ultramicroscopy 202 (2019) 100 – 106. doi:https://doi.org/10.1016/j.ultramic.2019.03.004.
URL http://www.sciencedirect.com/science/article/pii/S0304399118303693

[18] M. Cui, Y. Zhao, B. Xu, X. wei Gao, A new approach for determining damping factors in levenberg-marquardt algorithm for solving an inverse heat conduction problem, International Journal of Heat and Mass Transfer 107 (2017) 747 – 754. doi:https://doi.org/10.1016/j.ijheatmasstransfer.2016.11.101.
URL http://www.sciencedirect.com/science/article/pii/S0017931016331568

[19] J. O. Berger, Statistical Decision Theory and Bayesian Analysis, Springer, 1985.

[20] P. Christoffersen, K. Jacobs, The importance of the loss function in option valuation (2003).

[21] C. Hennig, M. Kutlukaya, Some thoughts about the design of loss functions (2007).

[22] N. Haghtalab, C. Musco, B. Waggoner, Toward a characterization of loss functions for distribution learning (2019).

[23] G. Gnecco, M. Sanguineti, Regularization techniques and suboptimal solutions to optimization problems in learning from data 22(3) (2010) 793–829.

[24] Deming, W. Edwards, Out of the Crisis, The MIT Press. ISBN 9780262541152, 2000.

[25] P. Wilson, H. A. Mantooth, Model-Based Engineering for Complex Electronic Systems, Elsevier, 2013.

[26] S. Ruder, An overview of gradient descent optimization algorithms (2016). URL https://ruder.io/optimizing-gradient-descent/

[27] S. Sun, Z. Cao, H. Zhu, J. Zhao, A survey of optimization methods from a machine learning perspective (2019). arXiv:1906.06821.

[28] H. P. Gavin, The levenberg-marquardt algorithm for nonlinear least squares curve-fitting problems.
URL http://people.duke.edu/~hpgavin/ce281/lm.pdf

[29] C. C. Aggarwal, Neural Networks and Deep Learning, Springer International Publishing, 2018.

[30] U. Tautenhahn, U. Hämarik, The use of monotonicity for choosing the regularization parameter in ill-posed problems, Inverse Problems 15 (6) (1999) 1487–1505. doi:10.1088/0266-5611/15/6/307.
URL https://doi.org/10.1088/0266-5611/15/6/307

[31] G. Kriukova, O. Panasiuk, S. V. Pereverzyev, P. Tkachenko, A linear functional strategy for regularized ranking, Neural Networks 73 (2016) 26–35. doi:https://doi.org/10.1016/j.neunet.2015.08.012.
URL https://www.sciencedirect.com/science/article/pii/S0893608015001756

[32] Y.-H. Dai, Nonlinear conjugate gradient methods.
URL ftp://lsec.cc.ac.cn/pub/home/dyh/papers/CGoverview.pdf

[33] Y.-H. Dai, A perfect example for the bfgs method.
URL ftp://www.cc.ac.cn/pub/dyh/papers/bfgs-example.pdf

Appendix A. Conjugate Gradient

As discussed above, using previous parameter updates is a way to avoid fluctuations. However, we can also use equations to inform when to perform updates as

$$\Delta w_t = -\partial L/(\partial w_t) + \beta \Delta w_{(t-1)} \tag{A.1}$$

Conjugate gradient works on finding an optimal coefficient β to prevent fluctuation. There are several methods for calculating the coefficient β in literature such as Fletcher-Rieves, Polak–Ribière, Hestenes-Stiefel, and Dai–Yuan. The formula below shows the β calculation of the methods, respectively.

$$\beta = \frac{\partial L/(\partial w_t)^T (\partial L/(\partial w_t))}{(\partial L/(\partial w_{t-1}))^T (\partial L/(\partial w_{t-1}))} \tag{A.2}$$

$$\beta = \frac{\partial L/(\partial w_t)^T (\partial L/(\partial w_t) - \partial L/(\partial w_{t-1}))}{(\partial L/(\partial w_{t-1}))^T (\partial L/(\partial w_{t-1}))} \tag{A.3}$$

$$\beta = \frac{\partial L/(\partial w_t)^T (\partial L/(\partial w_t) - \partial L/(\partial w_{t-1}))}{(\Delta w_(t-1))^T (\partial L/(\partial w_t) - \partial L/(\partial w_{t-1}))} \tag{A.4}$$

$$\beta = \frac{\partial L/(\partial w_t)^T \partial L/(\partial w_t)}{(\Delta w_(t-1))^T (\partial L/(\partial w_t) - \partial L/(\partial w_{t-1}))} \tag{A.5}$$

Although CG provides fast convergence, it often results in poor performance [32].

Appendix B. Broyden-Fletcher-Goldfarb-Shanno Algorithm

Quasi-Newton methods approximate the Hessian value to solve unconstrained optimization problems. Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm is one of the efficient Quasi-Newton approaches [33].

Let L be a continuously twice differentiable loss function, an approximation of the inverse Hessian matrix as given below.

$$H_{t+1} = H_t - \frac{H_t s_t s_t^T H_t}{s_t^T H_t s_t} + \frac{y_t y_t^T}{s_t^T y_t} \tag{B.1}$$

$$s_t = w_{t+1} - w_t \tag{B.2}$$

$$y_t = (\partial L/\partial w_{t+1}) - (\partial L/\partial w_t) \tag{B.3}$$

Parameter update are as follows,

$$\Delta w_t = H_t^T (\partial L/\partial w_t) \tag{B.4}$$

$$w_{t+1} = w_t + \alpha \Delta w_t \tag{B.5}$$