Genetic,Weight-Sharing and Auxiliary Loss: when MNIST

Yann Mentha
yann.mentha@epfl.ch

Abstract

In 1999, Yann LeCun et al. published a dataset which will within a few year time become what we know as the state-of-the art dataset to train image classification models on.[2] The present report aims at exploring the use of weight sharing and auxiliary losses in the context of Convolutional Neural Networks (CNN) in order to improve the performance of such networks by using GridSearch and Genetic Algorithm for parameter optimization.

1 Introduction

As CNNs demonstrate particularly good results in image recognition tasks the past decade, they quickly became the reference method way to classify visual content. In addition, innovative methods such as batch normalization or dropout are frequently developped, enhencing drastically the performance of such networks. One concern relies in the difficulty to test thoroughly such new features: indeed, this requires a method both data-independant, systematic and generalized, making performance assessment a challenging domain. Comparing the augmented network with some well-chosen baseline can still give a first hint on the real improvement brought by the method, as we will see in this report.

2 Methods

All the results and methods present in this report are displayed and computed in the jupyter notebook `explore.ipynb`

2.1 Implementation Objective

The goal of the present project consists in assessing the performance improvement in terms of test accuracy brought by weight-sharing (WS) and/or auxiliary losses (AL) techniques on a deep-CNN reference architecture for image comparison of 1000/1000 train/test set. This is performed by tuning the model parameters using two distinct meta-algorithms: Grid Search Algorithm (GSA) and Genetic Algorithm (GA). An additional and succinct comparison of these algorithms is included in the goal of this project.

2.2 Simple Network

To begin, a first CNN network (*HRTA. simple Network*) was implemented and optimized by hand in order to obtain a reasonable reference baseline for subsequent comparison purpose with the weight-sharing and auxiliary-loss improvements.

2.3 Double Network

In order to compare pairs of digits, 5 architectures of networks (all taking pairs of 14 x 14 images as input) were implemented. (*HRTA double networks*)
The first one, a naive implementation, trivially trains 2 simple networks in parallel on the pairs of the images and outputs the comparison.
The 4 other architectures implement weight sharing and/or auxiliary loss involving Linear/ReLU output layers in order to refine the output.

- Weight sharing: consists in assigning the same variables as weights for both net1 and net2 (cf Figure 2)
- Auxiliary Loss: Instead of applying the loss on the comparison result only, the 2 intermediate outputs of net1 and net2 are collected and another loss (the class loss) is applied to them in addition to the comparison loss which is applied to the comparison output and its target.

A priori, the similar nature of both inputs in the pairs (handwritten digits) motivate weight sharing: both networks might beneficiate of the training information of each other.
Figure 1 gives us an intuition why simple comparison could somehow miss precious informations: while the network 1 shows a pretty high confidency in the digit 1, there are still more chances for its digit to be greater than its paired equivalent, highlighting the importance for the network to access such probabilities rather than a discrete output. Notice that in order to train its 2 networks, the Naive implementation exclusively uses the auxiliary loss (Figure 1), since a simple comparison cannot send any gradient in the backward pass.

2.4 Testing

KFold cross validation (KFold CV) was implemented in order to obtain robust estimates of the accuracy for the multiple models. In addition, classical train/test sets were executed as well in order to limit overfitting.

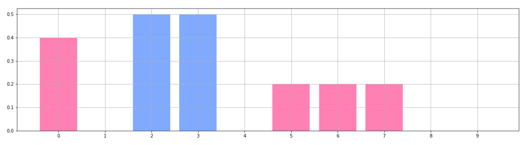

Figure 1: Example Probas

Table 1: Double Networks Architectures

Model	Weight Sharing	Auxiliary Loss
Naive	No	Yes
Model1	Yes	No
Model2	No	Yes
Model3	No	No
Model4	Yes	Yes

The key point in testing consists in evaluating fairly the augmented models and the naive one: indeed, carrying out parameter optimization for the augmented models exclusively would lead to biased results.

2.5 Optimization/Meta-algorithms

A particular accent was put on meta-algorithms rather than classical parameter tuning: indeed, as the goal of the project consists in assessing the performance improvement induced by weight/sharing and auxiliary losses rather than outperforming any pre-existing models, meta-algorithms were implemented in order to carry out fair runs between the architectures. Each combination accuracy was assessed during the GSA/GA using a KFold with relatively low K (K = 2 or 3). The best combinations were then retested with higher K (K = 10) for robustness purpose, and on classical train/test splits.

- Grid Search Algorithm: classical grid search

- Genetic Algorithms: a population of N=10 parameter combinations (HRTA *individuals*) is randomly created from a pool of M=6 parameters having each J_M parameter values. At every selection iteration (30 iterations),each individual is evaluated with respect to a given metric: only a given top ratio (60%) of these individuals is kept, while other are discarded. Finally these top individuals make up for the discarded ones by "breeding" in a genetic-similar manner (the child takes a random combinations of its parents values). However, the individuality (population diversity) matters as well, as this prevents convergence to a local minima where all the population share the same parameter values. The following individuality formula was therefore used:
$I_n = \frac{1}{\sum_{p \in P_n} \frac{1}{\sum_{v \in V_p} N - N_v}}$
with I_n the individuality of the individual n, P_n its parameters, V_p the actual values of parameter p, N_v the number of individuals which share the same value for parameter p. Finally, the chosed metric to segregate between combinations is $M_n = compacc_n + \lambda I_n$ with λ a hyper-hyper-parameter manually set to 0.75.[1]

The parameters and their respective values tested in GSA and GA are displayed in Table 2. Finally, a comparison between the execution time of GSA and GA was done as unlike GSA, GA does not aim at testing every possible combination, and should in that sense, execute much faster resulting in a hopefully similar performance.

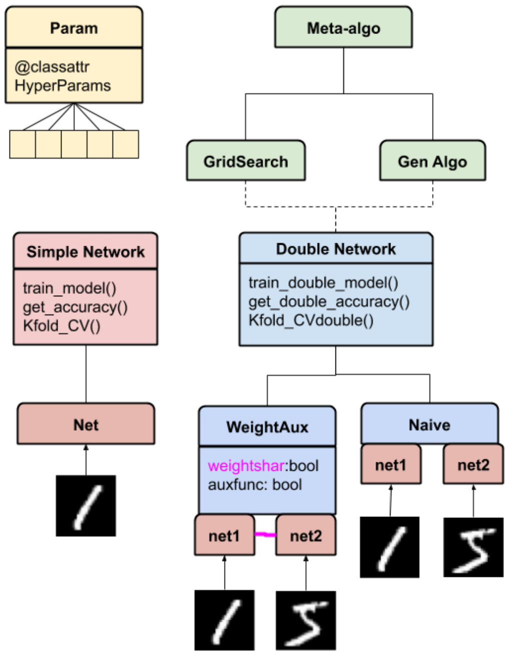

Figure 2: Project Architecture: each double network includes 2 simple networks as parameter. The Meta algorithms allow to carry on parameter selection on the double networks. The simple Network is of course able to perform digit recognition on its own.

Table 2: GSA/GA Parameter Grid

Parameter	Values
Model	Naive, Model[1-4]
CompLoss	CrossEntropyLoss, NLLLoss, MSELoss
ClassLoss	CrossEntropyLoss, NLLLoss, MSELoss
Optimizer	SGD, Adam, AdaGrad, AdamW
LR	$1^{-4}, 1^{-3}, 1^{-2}, 1^{-1}, 1$
Lambda	$0.2, 0.4, 0.7, 0.9$

3 Results

3.1 Simple Network

The simple network accomplishes a train accuracy of 100% and test accuracy of $95 \pm 0.03\%$ which was considered acceptable for subsequent analysis.

3.2 Grid Search

As displayed in Table The3, 4,5,6 and 7 the naive implementation reached the highest score (95 $\pm 2\%), overpassingthesecondbestmodelby4\%.TheGridSearch$
$5600UProcessor.$

3.3 Genetic Algorithm

Genetic Algorithm results are displayed in Table 8: the best parameter combination reaches a 10Fold accuracy of 87 %. The Genetic Algorithm took 19min and 36sec time to run on the same Intel® Core^{TM} i7-5600U Processor. T

4 Discussion

4.1 Weight Sharing and Auxiliary Loss

It seems like weights sharing lead to overall better class classification than its extensionless counterpart, probably due to the fact that both single networks could beneficiate of twice as more training data than in the case of the naive model. However, despite better individual class accuracies, the WS-AL network performed surprisingly slightly poorer in terms of comparison accuracy than its naive counterpart: this could be due to the final linear layers which need more representational power/training in order for the comparison loss to be optimized. Finally, We can see that the AL and WS accuracy improvement lead to higher results when combined compared to the no extension baseline (+ 3% for WS,+11 % for AL,+12% for AL+WS)

4.2 Genetic Algorithm

The time taken by the genetic algorithm was significantly lower than the one of the gridsearch (19 minutes seconds for the GA vs 6 hours fot GSA), it did not manage to reach an accuracy almost as good as the best parameter of the grid-search. Finer meta-parameter tuning could lead to enhanced performance, losing somehow part of the execution time advantage.

5 Conclusion

Weight sharing proved to be a valuable extension for digit comparison by allowing each network to train faster and better: auxiliary losses has for its part demonstrated even stronger improvement by training the network for their specific task as well. Further parameter tuning could probably beat the naive baseline, by improving the simple model.
As parameter optimization turned out to be pretty expensive for such a test tube experiment, Genetic Algorithms with the right meta-parameters showed decent and usable results for practical experiments.

References

[1] Boi Faltings and Michael Schumacher. *L'intelligence artificielle par la pratique*. PPUR presses polytechniques, 2009.

[2] Y Lecun, C Cortes, and CJC Burges. *The MNIST Dataset of Handwritten Digits(Images)*. 1999.

Table 3: Grid Search parameters: Naive Model

CL	AL	Opt	LR	3F[%]	10F[%]	Acc1[%]	Acc2[%]	rank
CrossEnt	L1	AdamW	0.01	96	95 ±2	93±4	95±2	1
CrossEnt	L1	Adam	0.01	96	-	-	-	-
CrossEnt	L1	Adagrad	0.01	0.94	-	-	-	-

CL=Class Loss,AL=Auxiliary Loss,3F=3Fold CV,10F=10Fold CV,Acc1/2=accuracy of network1/2,rank= rank over the 10Fold CV accross models

Table 4: Grid Search parameters:Model 1(WS)

CL	AL	Opt	LR	Lambda	3F[%]	10F[%]	Acc1[%]	Acc2[%]	rank
CrossEnt	L1	Adagrad	0.001	0.9	90	82±5	13±6	11±4	4
CrossEnt	L1	Adagrad	0.001	0.7	81	-	-	-	-
CrossEnt	L1	Adagrad	0.001	0.4	81	-	-	-	-

Table 5: Grid Search parameters:Model 2(AL)

CL	AL	Opt	LR	Lambda	3F[%]	10F[%]	Acc1[%]	Acc2[%]	rank
L1	CrossEnt	AdamW	0.01	0.2	87	90±3	93±6	94±4	3
MSE	CrossEnt	AdamW	0.01	0.2	0.86	-	-	-	-
MSE	CrossEnt	Adagrad	0.01	0.2	0.86	-	-	-	-

Table 6: Grid Search parameters:Model 3(no extension)

CL	AL	Opt	LR	Lambda	3F[%]	10F[%]	Acc1[%]	Acc2[%]	rank
MSE	L1	Adagrad	0.01	0.9	0.82	79±5	11±4	14±9	5
MSE	L1	Adagrad	0.01	0.7	0.82	-	-	-	-
MSE	L1	Adagrad	0.01	0.4	0.82	-	-	-	-

Table 7: Grid Search parameters:Model 4(WS + AL)

CL	AL	Opt	LR	Lambda	3F[%]	10F[%]	Acc1[%]	Acc2[%]	rank
L1	CrossEnt	Adam	0.01	0.2	0.87	91±4	95±4	94±3	2
L1	CrossEnt	Adagrad	0.01	0.2	0.87	-	-	-	-
MSE	CrossEnt	Adam	0.01	0.2	0.87	-	-	-	-

Table 8: Best Genetic Algorithm parameters)

Arch rank	CL	AL	Opt	LR	Lambda	3F[%]	10F[%]	Acc1[%]	Acc2[%]
WS-AL	L1	CrossEnt	AdamW	0.001	0.7	0.85	87±4	93±4	94±4
WS-AL	CrossEnt	CrossEnt	Adam	0.001	0.7	0.83	-	-	-
WS	CrossEnt	CrossEnt	Adam	0.001	0.7	0.79	-	-	-