[654AA] Machine Learning

Project type A

Authors:
Vlad Pandelea
Simone Spagnoli

Examiner:
Alessio Micheli

MSc in Computer Science

Pisa University
Department of Computer science

January 2018

Contents

Abstract

It's the journey that matters, not the destination. In this report we briefly describe our own journey up to the participation in the ML cup as part of the machine learning course given at Pisa University. In order to get there we developed a lightweight library to create, train and validate neural networks models. Furthermore we assessed the correctness of our implementation by running several benchmarks on standardized tasks as well as by meticulously comparing our results with those obtained using a well-known deep learning library.

Chapter 1

Introduction

The main aim of the project was that of developing some sort of NN simulator and then use it to create models that can be applied to the given task (ML cup).

To this purpose, we first developed a lightweight library, assessed the correctness of the implementation and then used the theory at our disposal to validate and find the best model for the task at hand.

The library was written in Python[1], using its efficient numpy[2] library for the basic math operations. The training of the networks is performed through the backpropagation[3] algorithm. We have also added several advanced optimization algorithms and techniques such as Momentum[4][5], Adam[6] and its variants, RMSprop[7], dropout[8], line search[9] and elastic net regularization[10].

To assess the correctness of our implementation we solved the MONK's problems[11], a dataset used for comparison of learning algorithms. However while not being able to successfully solve the MONK' problems is an indicator of buggy implementations, solving it does not guarantee the correctness of the implementation. To further convince ourselves that our code was indeed working as supposed to, we compared our results, on a number of tasks, with those obtained by building the same models and using the same hyperparameters with keras[12], a high level library relying on Tensorflow[13], Theano[14] or CNTK[15] backends. Once confident enough in our code we moved on to the ML cup task.

The ML cup dataset[16] is composed of 1016 labelled samples and 315 unlabeled ones. Each sample is characterized by 10 features and 2 outputs. The context of the data was not given, therefore we could not make use of any inside knowledge. We limited ourselves to investigate basic properties of the data and apply simple heuristic to determine possible outliers[17].

Chapter 2

Method

2.1 Design choices

As previously mentioned we decided to develop a lightweight library that we now briefly describe. The main idea was to make it highly modular: we've done this by separating the building blocks that can be freely composed. For instance, to create and train a simple neural network we would proceed as follows:

- We create a neural network object container.
- We stack layers on the neural network. Each layer has its own regularization and activation functions that can be choosen from standard implemented ones or customizable ones supplying the provided interface. Moreover each layer can use its own weights initialization method, such as fan-in (uniform ($\frac{-x}{layerinputs}$, $\frac{+x}{layerinputs}$)) or xavier[18].
- Finally we can proceed to train or run a grid search with the created neural network. In order to do so an optimizer must be selected from the provided ones or created anew, supplying the provided interface. We also need to provide a dataset object that can be either entirely used for training or partitioned by specifying a percentage to be used as validation. Another way to validate is to provide a separate validation set. Standard or customizable loss functions can be used and moreover the training and validation splits can each use their own ones.

Refer to 2.1 for a practical example:

```
def sigmoid(x):
    return 1/(1+np.exp(-x))

def sigmoiddx(x):
    return x*(1-x)

train_data_path = "data/ML-CUP17-TR.csv"
test_data_path = "data/ML-CUP17-TS.csv"
```

```
dataset = preproc.Dataset()
dataset.init_train(preproc.load_data(path=train_data_path, target=True,
    header_l=10, targets=2))
dataset.init_test(preproc.load_data(path=test_data_path, target=False, header_l=10))

optimizer = Adam(lr=0.05,b1=0.9,b2=0.999)
sigm = Activation(sigmoid, sigmoiddx)
NN = NeuralNetwork()
NN.addLayer(inputs=10,neurons=20,activation=sigm, rlambda=0.2,regularization="L2",
          dropout=0.5, weights=None, bias=0.0, weights_init="fan-in")
NN.addLayer(inputs=20,neurons=2,activation="linear",rlambda=valr,regularization="L1",
          dropout=0.0, weights=None, bias=0.0, weights_init="xavier")

(tr_loss, tr_acc, val_loss, val_acc, history)=\
   NN.fit_ds( dataset,epochs=1000, optimizer=optimizer
       ,batch_size=32,verbose=2,loss_func="mse", val_loss_fun="mee", val_set=None,
       val_split=25)

###Alternatively we can run a grid search as follows
acts=[["sigmoid","linear"]]
opts=[optimizer2]#,optimizer6,optimizer7,optimizer8]
neurs=[[25,2]]
batches = [dataset.train[0].shape[0]]
losses = ["mse"]
regs = [[regularizations.reguls["L2"],regularizations.reguls["L2"]]]
rlambdas = [[0.000,0.000]]
fg,grid_res, pred = validation.grid_search(
         dataset, epochs=[500], batch_size=batches,
         n_layers=2, val_split=0,activations=acts,
         regularizations=regs, rlambda=rlambdas,
         cvfolds=3, val_set=None, verbose=2,
         loss_fun=losses, val_loss_fun="mee",
         neurons=neurs, optimizers=opts)
```

2.2 Assessment of correctness

Validating our library has had a major role in the project. Here we briefly walk through some of the steps taken in order to do so. Other than the provided MONK datasets, which have proved to be quiet susceptible to the weights initialization but otherwise easily achieve 100% accuracy(MONK1 and MONK2), we felt the need to play with a different problem as well. To this purpose we chose another well-known problem - that is the digit recognition one[19]. Luckily, this kind of problem has been thoroughly explored and many datasets to try machine learning models on exist. Eventually the choice fell on the mnist dataset[20]. For these problem, and other randomly generated ones, we compared our results with keras. To minimize the role played by randomness, we loaded the weights generated by our library

into the keras models. This has permitted to see the evolution of the training on both libraries starting from the same point. Moreover to test the MEE function which is not available in keras, we have implemented its basic form in keras' tensors notation and then compared the result of keras backpropagation using its automatic derivation tool with ours. Some of the resulting plots as well as more details are shown in 3 and appendix A (A.1, A.2, A.3, A.4).

2.3 Data analysis and manipulation

2.3.1 MONK

In order to eliminate the bias inherent in the data representation, we followed the usual approach adopted in literature - that is the one-hot encoding. This approach equalizes the distance between the values of the attributes: this makes sense when the represented information by the value is decorrelated from the actual value itself. Moreover we have transformed the output from $\{0, 1\}$ to $\{-1, 1\}$ as to be able to use the hyperbolic tangent function that proved to be slightly better and overall faster. We did not perform a deeper analysis as this dataset is more of a benchmarking one than a real problem to investigate on.

2.3.2 ML cup

Before diving into the experiments we have first looked at the data by analyzing some of its basic properties. First off we noticed that it is characterized by a ~ 0 mean and ~ 1 variance and therefore might already be somewhat normalized. Nevertheless, we have tried the effect of applying normalization techniques as we later report in 3.

Moreover, we tried to identify the presence of outliers: to this purpose we studied the distance of the elements' features from the mean of the features in relation to the features' variance. Refer to 2.1 for a more precise idea of how the number of outliers changes in relation to the sensitivity around the mean.

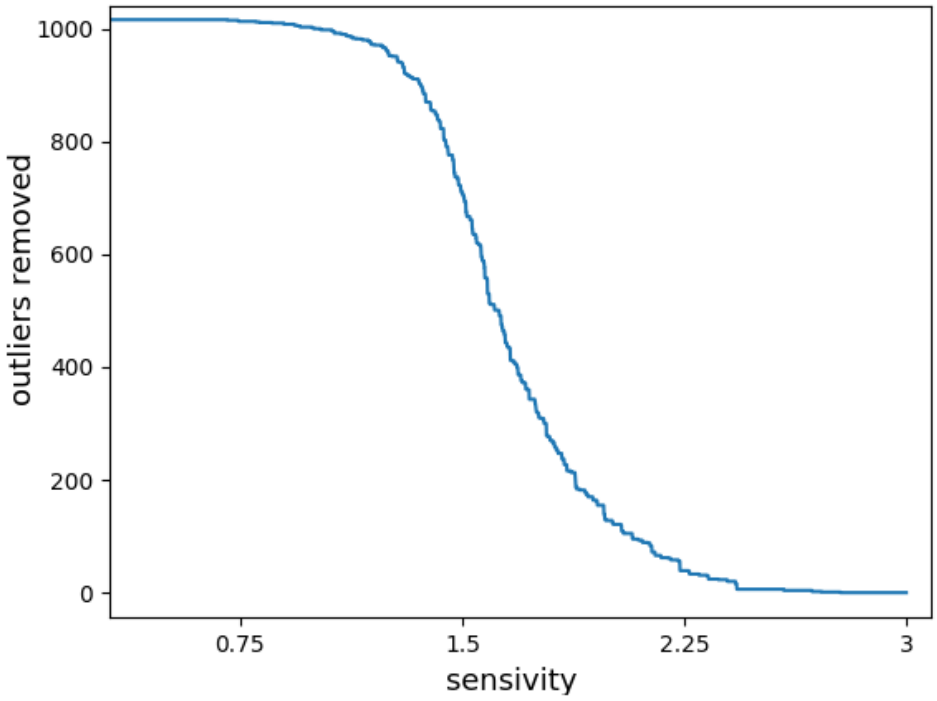


Figure 2.1: Number of outliers removed varying the sensitivity

Another simple but important data manipulation, at the training level this time, is the shuffling of the dataset before each epoch. This has several benefits and is common practice[21]. For instance it has been shown, both empirically[22] and theoretically(albeit only for the strongly convex case)[23] that shuffling helps in achieving a faster convergence. This is to be expected as shuffling the data eliminates the bias introduced by the same order of the sequential feed into the network each epoch.

Finally we also took a closer look at the labelled output, first by plotting it, and we have noticed that it was quite similar to past ML cup. From a brief analysis of the previous ML cup dataset, we noticed that the features have been remapped, the order of the samples shuffled and the output functions changed as well. We did not investigate much deeper, although it could have been prolific, as it was slightly diverting from the original scope of the project. We however also tried to train our model on the past dataset and have obtained a similar convergence rate, this could be seen as a capacity of the model of abstracting from the specifics of similar problems.

2.4 Learning techniques and novelties

As the aim of the project was that of implementing a solid NN simulator, we put our focus on mastering the most consolidated techniques and as such we left out some more advanced ones like batch normalization[24]. Nevertheless we still empirically tried the effect of some

more interesting approaches such as dropout and line search. This last method, although accelerating the convergence considered as number of iterations, has proved to be computationally expansive as each iteration requires several propagations through the network. What has instead proved to be useful were the first order methods that optimization theory proved to achieve better convergence rates than simple gradient descent in the convex setting[25]. Although our problem is not convex, these results still appear to hold. Consider for instance the Adam method in [6]. Much alike the momentum method, that tries to overcome the inherent greediness of the basic gradient descent, Adam further enhances this idea by considering biased estimates of first and second moment of the past gradients. However some uncertainty arises on the generalization capability of the models trained through these methods[26], all the more of a reason to make a good use of regularization.

Given the lack of context of the dataset, the regularization technique we have used is the elastic net regularization. This kind of regularization automatizes the selection of the best kind of regularization and is most useful when no other insight on the problem is given. However it requires a heavier search of the parameters to be fully effective and can therefore be unfeasible. In fact we had to limit ourselves to a quite reduced search.

Additional details of the results achieved with the various techniques used is given in the next chapters.

2.5 Model selection and assessment

Our model selection can be seen as composed of three phases as to make it computationally feasible. First off, we performed an initial grid search with the aim of finding the proper range of some hyperparameters such as learning rate and number of neurons. Since we tried several different optimizers, as a result of this grid search we were able to select the proper range of hyperparameters for each optimizer and consequently perform a finer grid search. Due to the fact that different optimizators have different number of parameters and even modify the given update step internally, the learning rate range turned out to be different for each one. To select an adequate range, besides considering the validation loss we also took into consideration the shape of the plots. In order to facilitate this task we ordered the plots by optimizer first and then by validation loss. The screening phase plots, among others, are available in the *screening.pdf* file attached to this report.

For each optimizer we then restricted the range of hyperparameters and performed a finer grid search over 10 trials. This has permitted to select the best hyperparameters in combination with each optimizer. On the best 3 configurations we carried out additional trials in which we checked the effect of other techniques that we had not considered in the earlier phases due to the computational cost that would have otherwise been unfeasible given our resources.

Finally we selected the best configuration with the techniques that led to an improvement and we trained it on the whole training set.

Regarding the data splitting, in the various grid searches we performed a 3-fold cross validation which seemed to be a good compromise between computational cost and efficacy.

As a final note, normally in machine learning tasks we keep a separate test set in order

to have an unbiased risk estimation. In this case we have decided not to have a separate hold-out test set as the risk estimation was of no interest to us since the plain estimation risk number would have no meaning given the lack of context and means of comparison. It would have made sense to have a labelled test set, had we used different learning algorithms, in order to compare their result on an unbiased estimator.

Chapter 3

Experiments

In this chapter we present our experiments on the MONK and AA1 cup datasets.

3.1 Results on MONK

In performing our experiments on the MONK problems we have referenced to [11]. We do not deeply discuss these problems as the literature is quite vast already. All the plots shown in this section are the average over 100 trials. Additional plots can be found in appendix B B. Refer to 3.1 for the average prediction results obtained on the MONK problems and to the next subsections for task-specific discussion and plots.

3.1.1 MONK 1

The MONK1 problem proved to be the most problematic one. It appears to be quite susceptible to the initialization and in fact it was not possible to find a setting that achieves an average 100% accuracy rate over 100 trials. Intrigued by this we tried to load the weights of our network into one created with keras and obtained the same result as can be seen in 3.2. Moreover we have noticed that the results improve as the mini-batch size is reduced, probably a symptom of the tendency to fall into bad local minima. This is in line with what shown in [27] where it is suggested that mini-batches somewhat prevent from getting stuck

Table 3.1: Average prediction results obtained for the MONKs tasks

Task	#units, eta,lambda	MSE(TR/TS)	Accuracy (TR/TS)(%)
MONK 1	units=3,lr=0.2,mome=0.4, tanh-tanh,mini-batch=4,Nesterov	(0.076, 0.043)	(0.986, 0.992)
MONK 2	units=2,lr=0.2,mome=0.5, tanh-tanh,mini-batch=noNesterov	(0.001, 0.002)	(100/100)
MONK 3	units=4,lr=0.03,mome=0.25, tanh-tanh,mini-batch=no,Nesterov	(0.162, 0.196)	(0.959, 0.939)
MONK 3 reg	units=4,lr=0.03,mome=0.25, tanh-tanh,mini-batch=no, reg=Elastic 0.015,Nesterov	(0.240, 0185)	(0.934, 0.972)

into bad points. Refer to 3.1 for the learning curves with size 4 mini-batch.

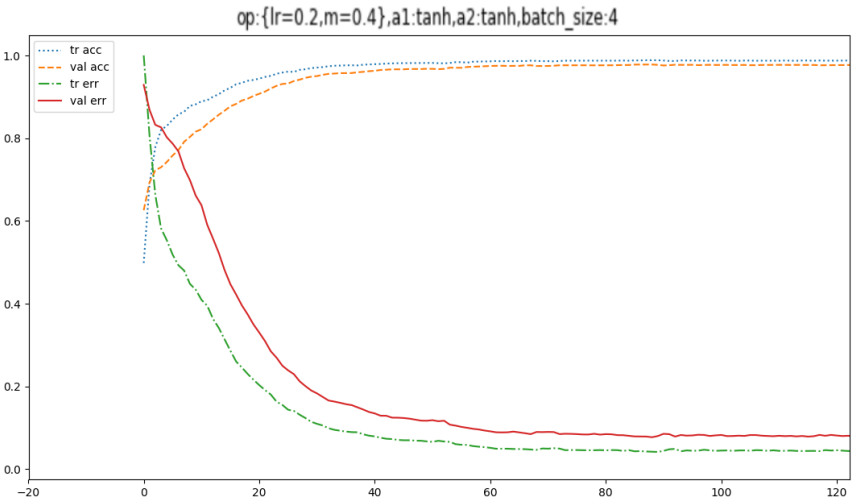


Figure 3.1: Monk1 average over 100 trials. Weights initialization: uniform distribution(-0.7/fanIn, +0.7/fanIn).

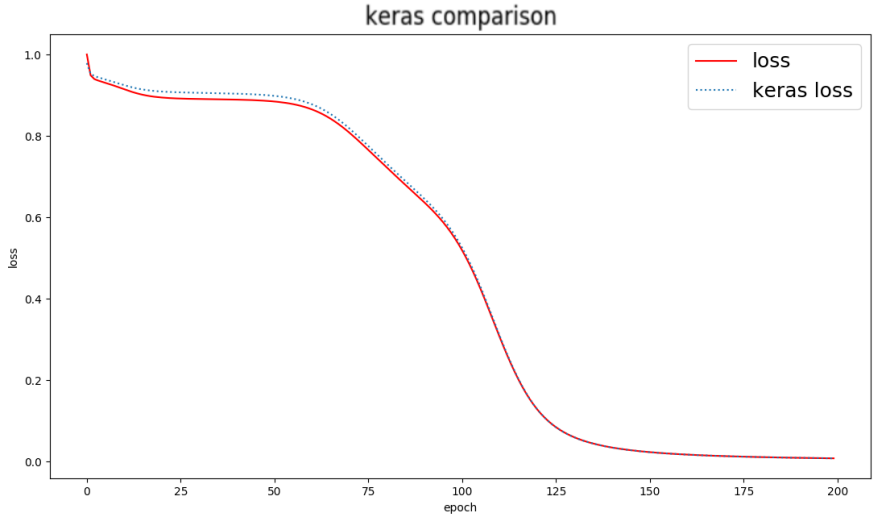


Figure 3.2: Comparison with Keras over monk1 dataset. The small differences are probably due to the different shuffling applied - with no mini-batch the plots match exactly.

3.1.2 MONK 2

The second task of the MONK problems is also susceptible to weight initialization as for instance with xavier initialization we did not obtain a 100% accuracy convergence. However with a uniform distribution (-0.7/fanIn, +0.7/fanIn), we did obtain full convergence over 100 trials as shown in 3.3.

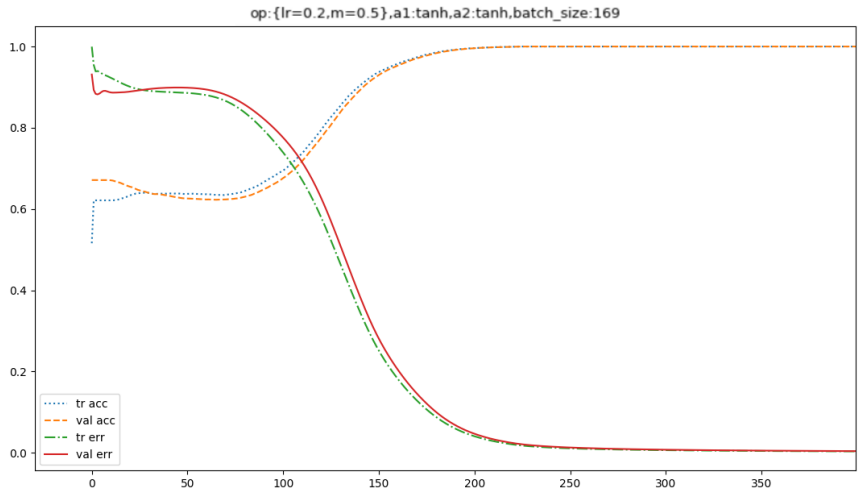


Figure 3.3: Monk2 average over 100 trials. Weights initialization: uniform distribution(-0.7/fanIn, +0.7/fanIn).

3.1.3 MONK 3

Unlike MONK1 and MONK2, here we have not been able to obtain 100% accuracy. This is to be expected as the training set for this last task contains 5% of noisy samples. Like [11], we have noticed the benefits of adding a regularization term in reducing the overfitting over the training samples. This can be better seen in 3.4.

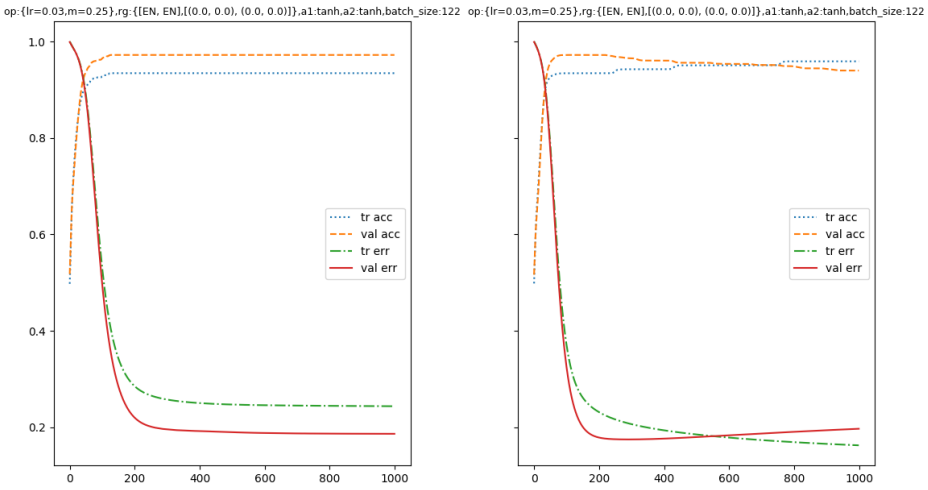


Figure 3.4: Monk3 average over 100 trials. Weights initialization: uniform distribution(-0.7/fanIn, +0.7/fanIn). In the left figure we can clearly see the effect of regularization that prevents the network from overfitting.

3.2 ML cup results

Here we report the most significant results in the various phases of the validation on the ML cup dataset, in accordance with the method described in 2.5.

3.2.1 Phase 1

In this initial screening phase we tried to cover a large range of parameters and then narrow it down to the most promising ones. Here we have decided to use a uniform weights distribution (-0.7/fanIn, 0.7/fanIn), as from short trials it appeared to perform better, and only later with a finer grid search try a different initialization method.

Overall we have noticed that the *ReLU* function led to more unstable curves and was generally outperformed by *tanh* and *sigmoid*. Typically *ReLU* proves to be useful in deeper architectures[28] as it leads to sparsity, thus improving generalization, and damps the vanishing gradient problem[29]. However our simple 3-layers architecture, coupled with the poor performance in the screening phase, could not provide any justification to keep the *ReLU* function for the subsequent grid searches and we have therefore decided to discard it. Most of the configurations with 5 neurons were not able to reach an error under 1.2 on neither training set or validation set, an indicator that such a kind of model is too simple for the problem and leads to underfitting. We have also been able to discard a set of learning rates that proved to be excessively high independently of the activation function used and at the end of this screening phase we ended up with 3 ranges of promising hyperparameters subsets combined with the momentum, adamax and RMSprop optimizers. The best results turned out to be the ones without regularization, and in the case of 80 neurons some overfitting occurred. The regularization range we have tried was probably too high as all of the configurations using it were underfitting. We have therefore decided to scale down the range for the next phases. Refer to 3.2 for the space of hyperparameters explored and to 3.3 for some of the most promising results. In the attached *screening.pdf* file all of the screening phase plots are shown. Keep in mind that some better results could have been probably achieved but our limited resources impeded the exploration of a wider variety of parameters as well as deeper architectures.

Table 3.2: Hyperparameters explored in the screening phase

Activation	Relu,Tanh,Sigmoid
Optimizer	Adam(lr=0.007)Adam(lr=0.03)Adam(lr=0.4),Adamax(lr=0.007), Adamax(lr=0.03),Adamax(lr=0.2),SGD(lr=0.001,momentum=0.9), SGD(lr=0.008,momentum=0.9),SGD(lr=0.04,momentum=0.9), RMSprop(0.01),RMSprop(0.05),RMSprop(0.1)
Regularizer	0, L1(0.01), L2(0.01), Elastic(0.01, 0.01)
Neurons	5,20,80
Losses	MEE

Table 3.3: Some of the most promising results in the screening phase.

Val Loss(MEE)	Neurons	Optimizer	Regularization	Activation
1.1854	80	Adam lr:0.03, b1:0.9, b2:0.999	(0,0)	tanh,linear
1.1255	80	Adamax lr:0.03, b1:0.9, b2:0.999	(0,0)	tanh,linear
1.1426	20	Momentum lr:0.008 m:0.9	(0,0)	tanh,linear
1.1388	20	RMS lr:0.01 delta=0.9	(0,0)	sigmoid,linear

3.2.2 Phase 2

In this second phase we performed a finer grid searches around the 3 most promising ranges selected in the previous phase, after some adjustments. We discarded the 80 neurons parameter as architectures with fewer neurons proved to be able to achieve the same results with a better learning curve. Moreover, for our library that we did not spend time in parallelizing, 80 neurons were computationally heavy and we would have had to further reduce the range of hyperparameters explored. In table 3.4 we show the range of explored parameters in the fine grid search through which we selected the most promising model. In table 3.5 we show some of the best results obtained in said grid search. We noticed that generally Adamax was characterized by a lower in-fold variance thus exhibiting more stability. This is somewhat confirmed by the learning curve. All of the plots resulting from this phase are visible in appendix C.1.1.

On the best configurations resulting from this phase we tried the effect of some techniques such as z-score and min-max normalization, dropout, xavier initialization, different training functions and removal of outliers. The resulting plots are visible in appendix C.1.2.

Table 3.4: Phase 2 hyperparameters explored.

Neurons	Activation	Regularization	Optimizer
20/35/50	tanh/sigmoid	(1e-4,1e-4)/(1e-4,0)/(1e-3,0) (6e-5,4e-5)/(0,7e-4)	SGD (lr:0.007, m:0.9) Adamax (0.016, b1:0.9, b2:0.999) RMSprop (0.01,delta:0.9)

Table 3.5: Some of the best results from phase 2.

Neurons	Activation	Regularization	Optimizer	Val loss(MEE)	In-fold std
50	tanh	(1e-4,0)	SGD lr:0.007 , mom:0.9	1.1617	0.00415
50	sigmoid	(6e-5,4e-5)	Adamax 0.016 b1:0.9, b2:0.999	1.1327	0.00138
50	sigmoid	(1e-4,1e-4)	Adamax 0.016 b1:0.9, b2:0.999	1.1393	0.0023

3.2.3 Phase 3

We briefly discuss the effect of the techniques applied on the best models resulting from phase 3. Dropout and outlier removal did not prove to be effective. Note that this is not an absolute statement as we did not try to vary the dropout and outliers detection parameters extensively. Xavier and fan-in initialization did not seem to influence the outcome significantly, which is an indicator of low dependency of the model on the weights initialization. Finally, while min-max led to worse result, zscore did not change the outcome. This is to be

expected as zscore preserves the around 0 mean and 1 variance of the dataset while min-max changes it.

Since none of the techniques led to better results, we decided not to apply them.

As final model we then chose the Adamax model with heavier regularization in table 3.5 (3rd row) as this is more probable to be a simpler model that can therefore generalize better.

3.2.4 Hardware resources

All the experiments have been performed on i5-2520m processors [30] and 8GB of DDR3 1333MHz M471B5273CH0-YK0. The training time for a 3-fold cross validation with 3000 epochs was around 152.40 seconds, about the same as keras on cpu.

Chapter 4

Conclusions

Wrapping up our experience during these past months we can draw some conclusions. The powerful libraries available nowadays provide a high level of abstraction and while this is useful for quickly getting things done, a naive use of a user with no solid knowledge of the theoretical ground may lead to a distratrous outcome. For this reason, we strongly believe that only by implementing from the very bottom stage a machine learning model, one can really become a conscious user of said libraries. In light of this we believe that developing new and powerful models is important but even more important is not forgetting the theoretical ground that they build on. A state of art model is of no use if it is not used conscientiously. For this reason in our project we did not use complex models but rather focused on an adequate model selection, as far as our limited resources permitted.

In conclusion, by implementing our own library and comparing and benchmarking it with a combination of state of the art libraries and well-known problems, we feel that we are now much more aware of what's happening behind the scenes.

Our nickname is **RandomPred** and we agree to the disclosure and publication of our nickname, and of the results with preliminary and final ranking. The file with the results is *RandomPred_.csv*.

Appendix A

A.1 Keras comparisons

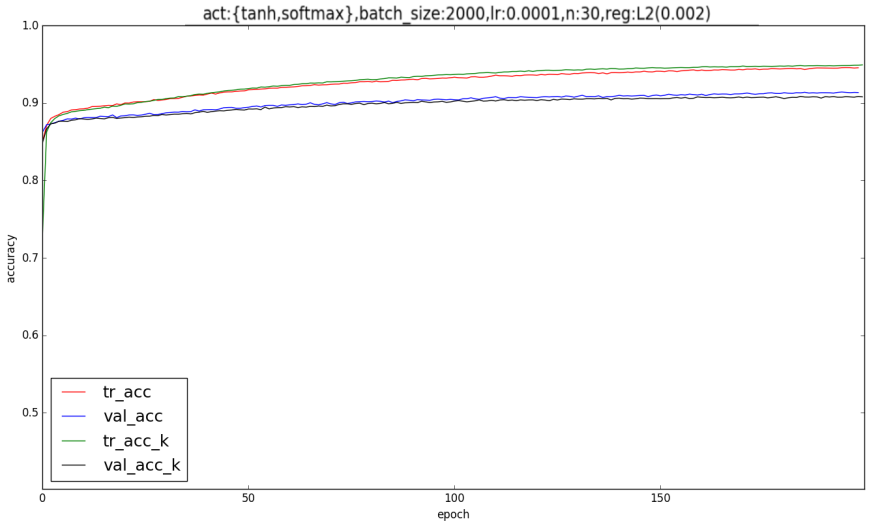


Figure A.1: Comparison with Keras over the mnist digits dataset. Again the random sampling leads to slightly different curves. Nonetheless both models reach 90% accuracy on the validation set over 150 epochs.

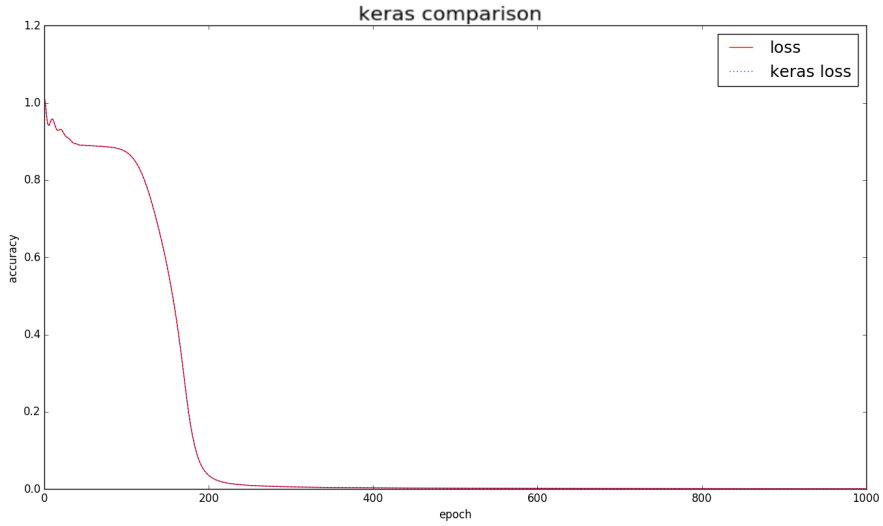


Figure A.2: Comparison with Keras on MONK2 dataset without using mini-batch.

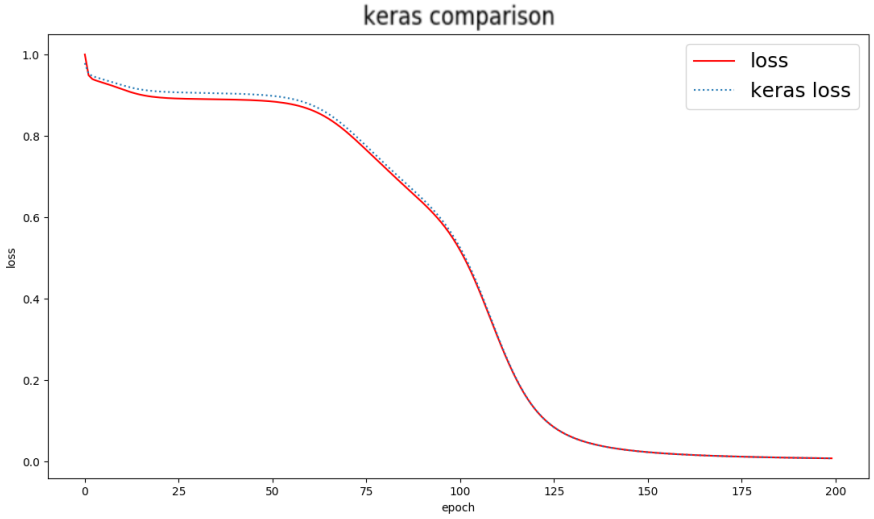


Figure A.3: Comparison with Keras over monk1 dataset. The small differences are probably due to the different shuffling applied - with no mini-batch the plots match exactly.

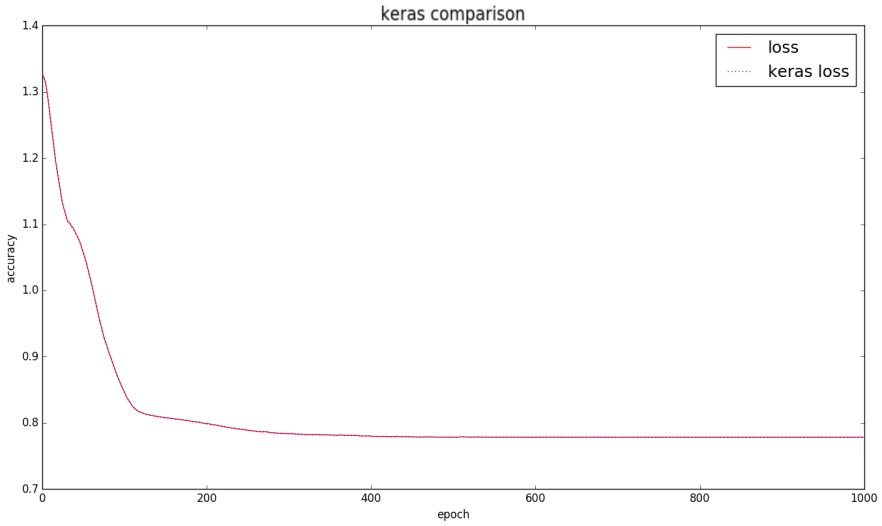


Figure A.4: Comparison with Keras on a randomly generated dataset, 2 layers network with tanh and relu.

Appendix B

B.1 MONK additional plots

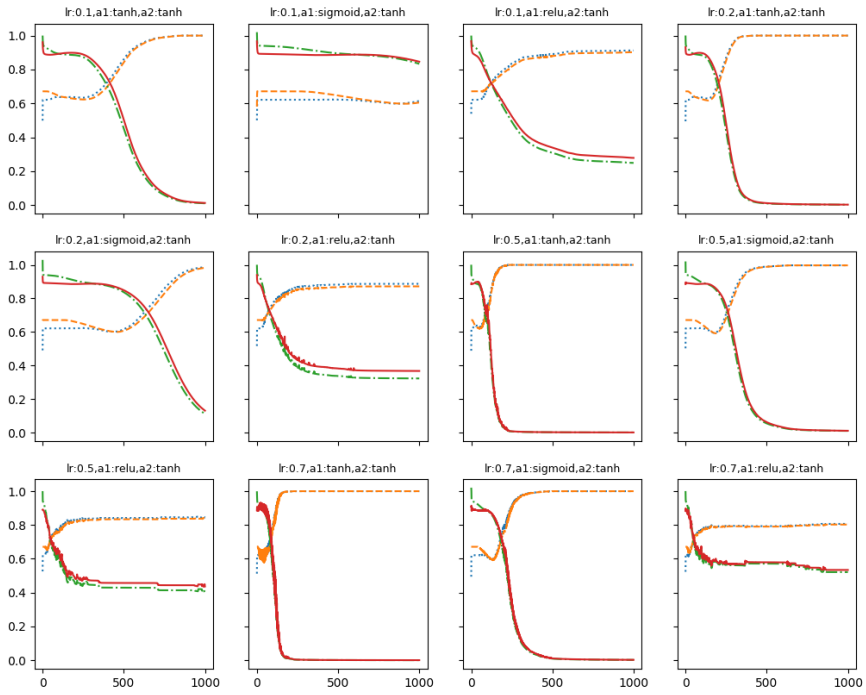


Figure B.1: Brief grid search over Monk2 dataset.

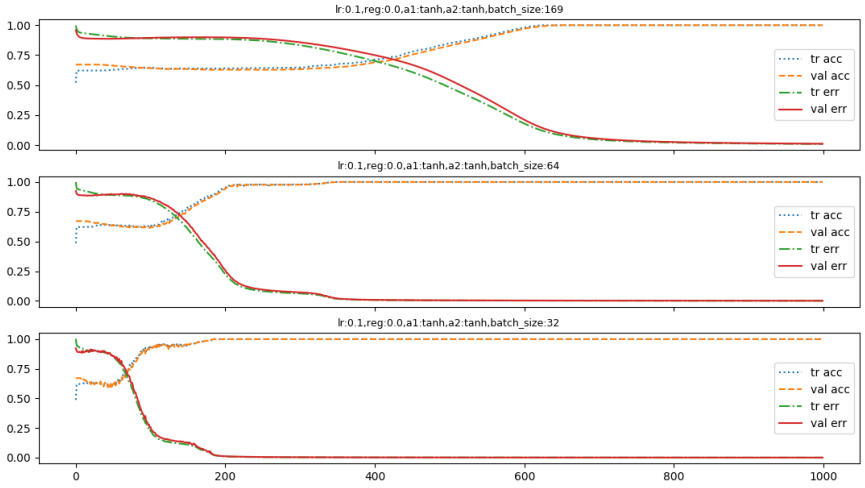


Figure B.2: Comparison between mini-batch and batch on MONK2 dataset

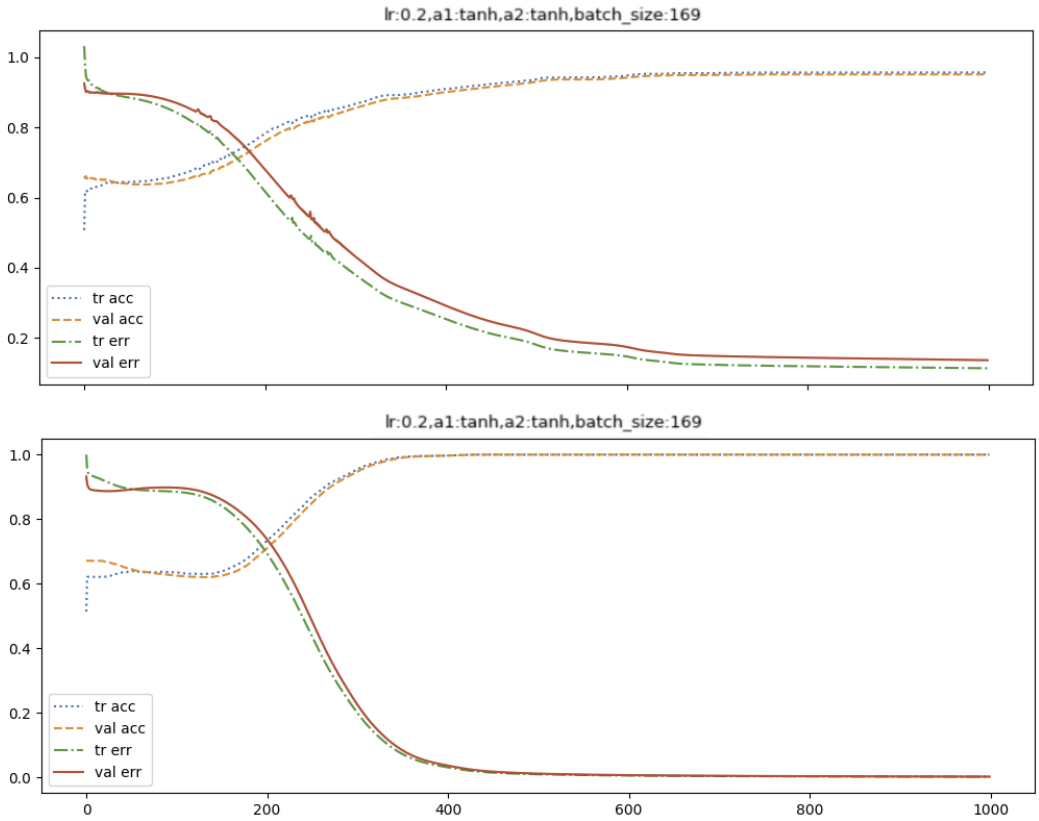


Figure B.3: Comparison between different inizializations on Monk2. In the upper figure xavier initialization is used, in the bottom one uniform(-0.7/fanIn, +0.7/fanIn) is used.

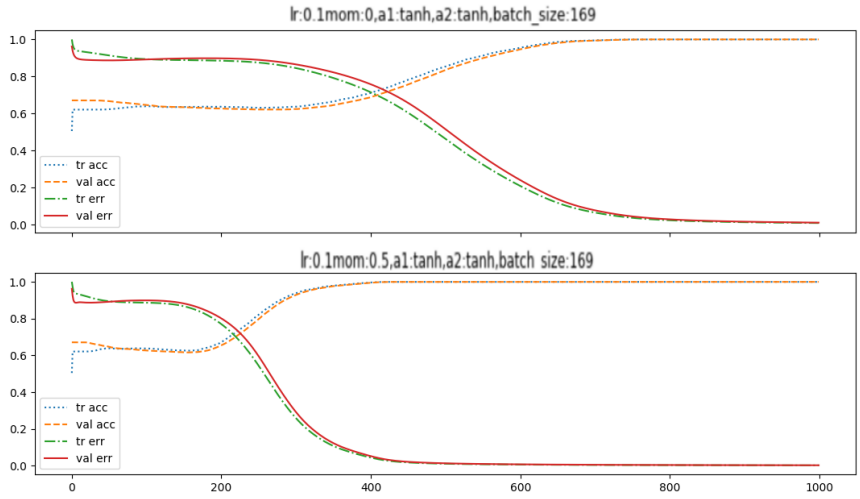


Figure B.4: Comparison between SGD with and without momentum

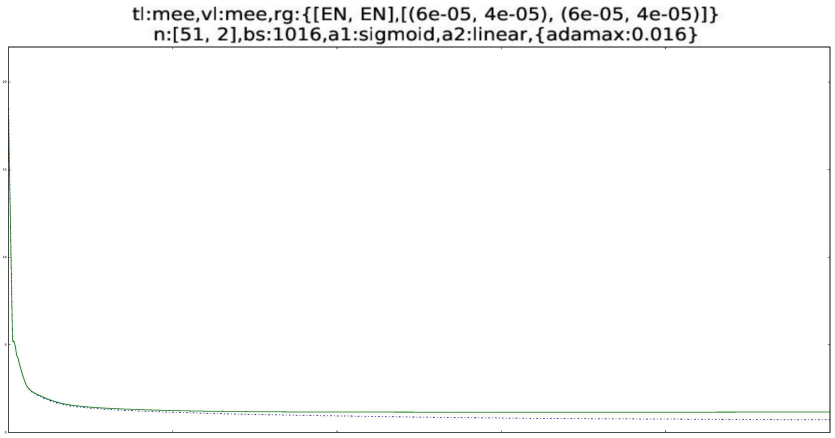
Appendix C

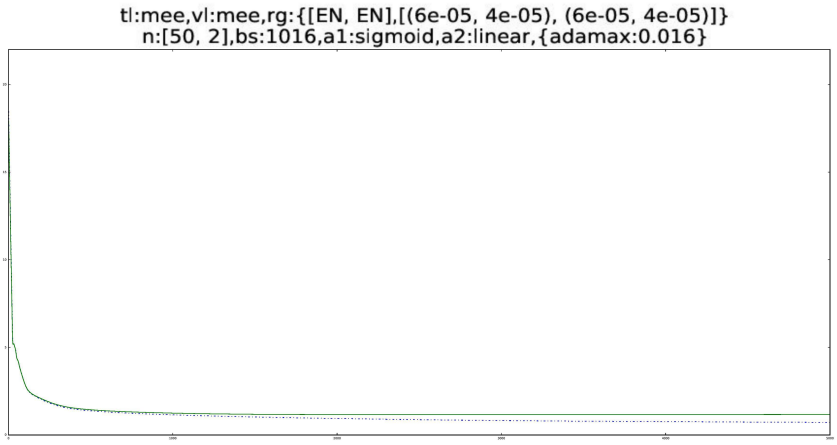
C.1 Cup additional plots

C.1.1 Phase 2: Effect of various techniques

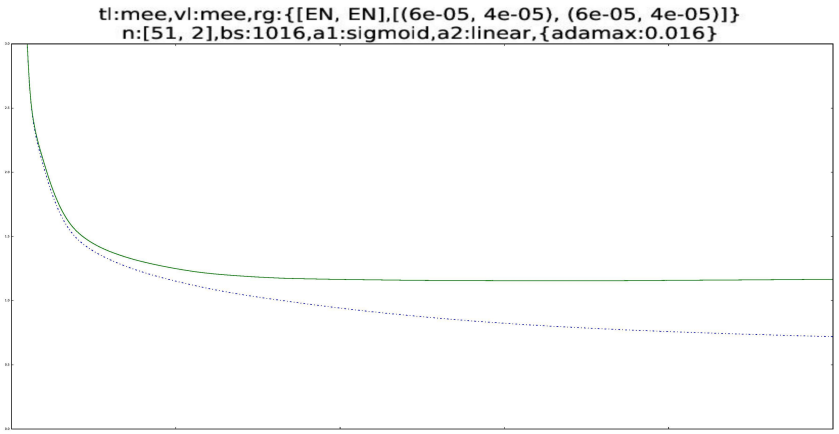
C.1.1.1 Xaver vs fan-in initialization

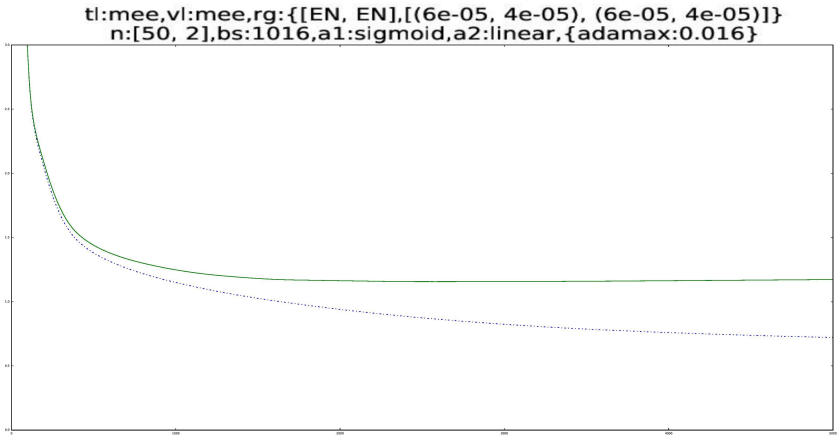
Effect of using a different weights initialization method. In the upper figure xavier initialization is used, in the bottom one uniform(-0.7/fanIn, +0.7/fanIn) is used. No significant differences in the learning appeared over 20 trials, thus indicating low dependency on the initialization method.


tl:mee,vl:mee,rg:{[EN, EN],[(6e-05, 4e-05), (6e-05, 4e-05)]}
n:[51, 2],bs:1016,a1:sigmoid,a2:linear,{adamax:0.016}

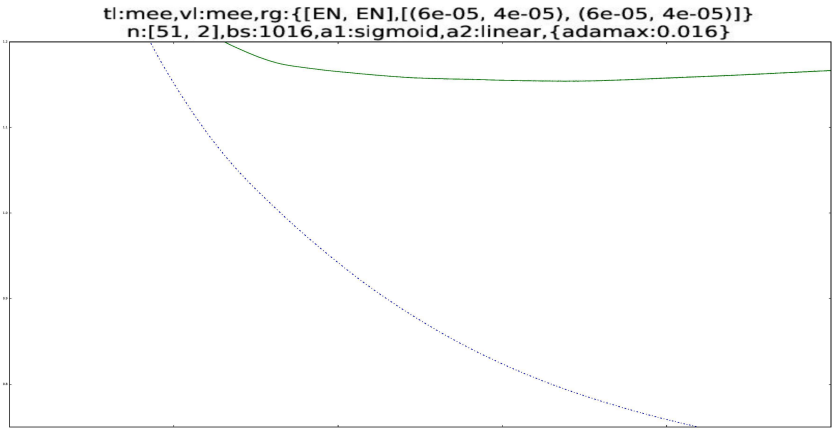

tl:mee,vl:mee,rg:{[EN, EN],[(6e-05, 4e-05), (6e-05, 4e-05)]}
n:[50, 2],bs:1016,a1:sigmoid,a2:linear,{adamax:0.016}

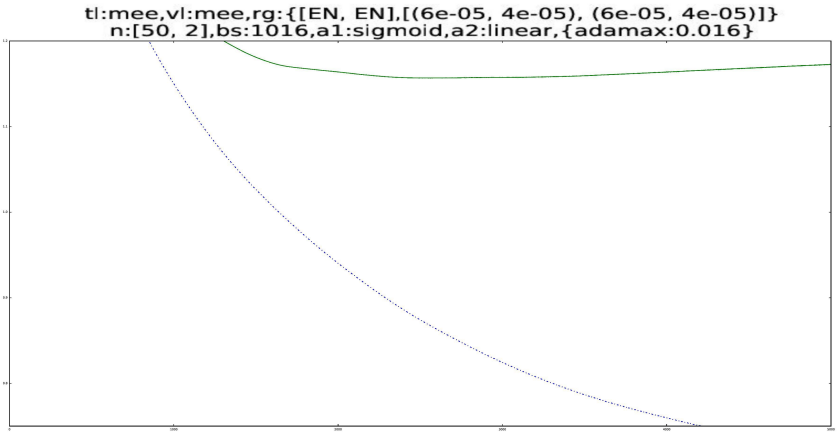
range 0-3


tl:mee,vl:mee,rg:{[EN, EN],[(6e-05, 4e-05), (6e-05, 4e-05)]}
n:[51, 2],bs:1016,a1:sigmoid,a2:linear,{adamax:0.016}


tl:mee,vl:mee,rg:{[EN, EN],[(6e-05, 4e-05), (6e-05, 4e-05)]}
n:[50, 2],bs:1016,a1:sigmoid,a2:linear,{adamax:0.016}

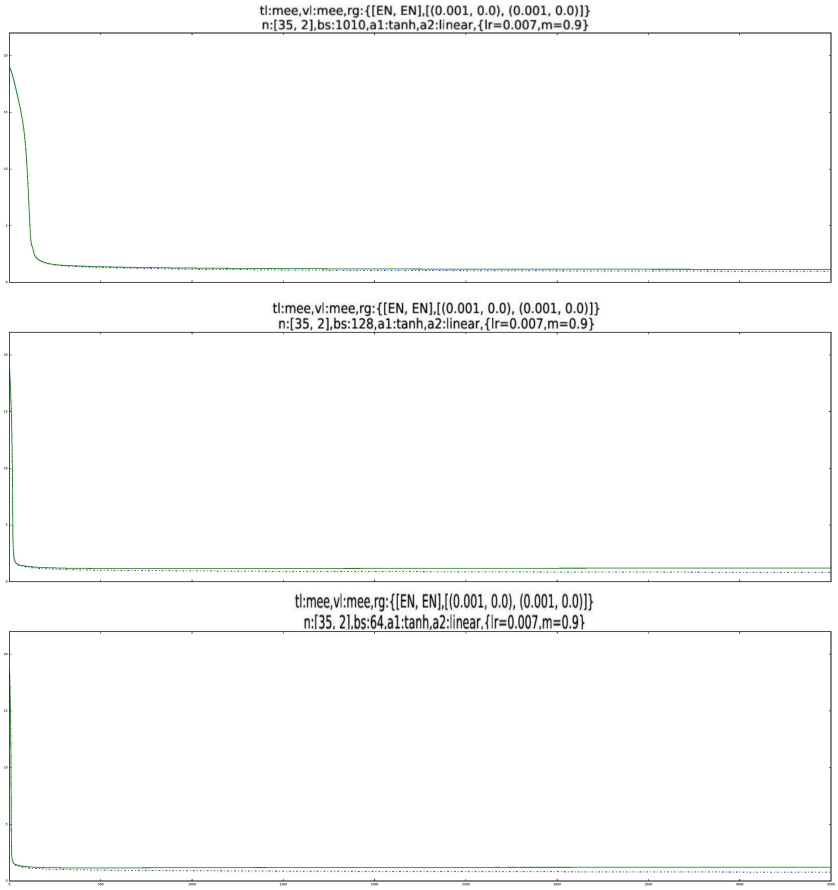
range 0.5-1


tl:mee,vl:mee,rg:{[EN, EN],[(6e-05, 4e-05), (6e-05, 4e-05)]}
n:[51, 2],bs:1016,a1:sigmoid,a2:linear,{adamax:0.016}

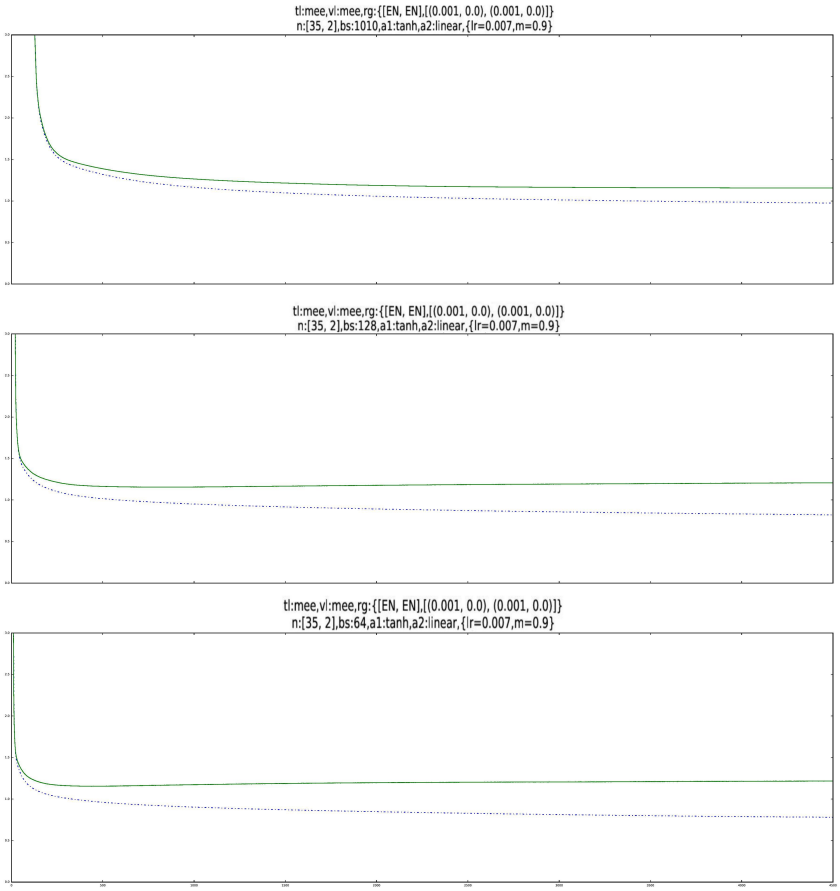

tl:mee,vl:mee,rg:{[EN, EN],[(6e-05, 4e-05), (6e-05, 4e-05)]}
n:[50, 2],bs:1016,a1:sigmoid,a2:linear,{adamax:0.016}

C.1.1.2 No batch vs mini-batch

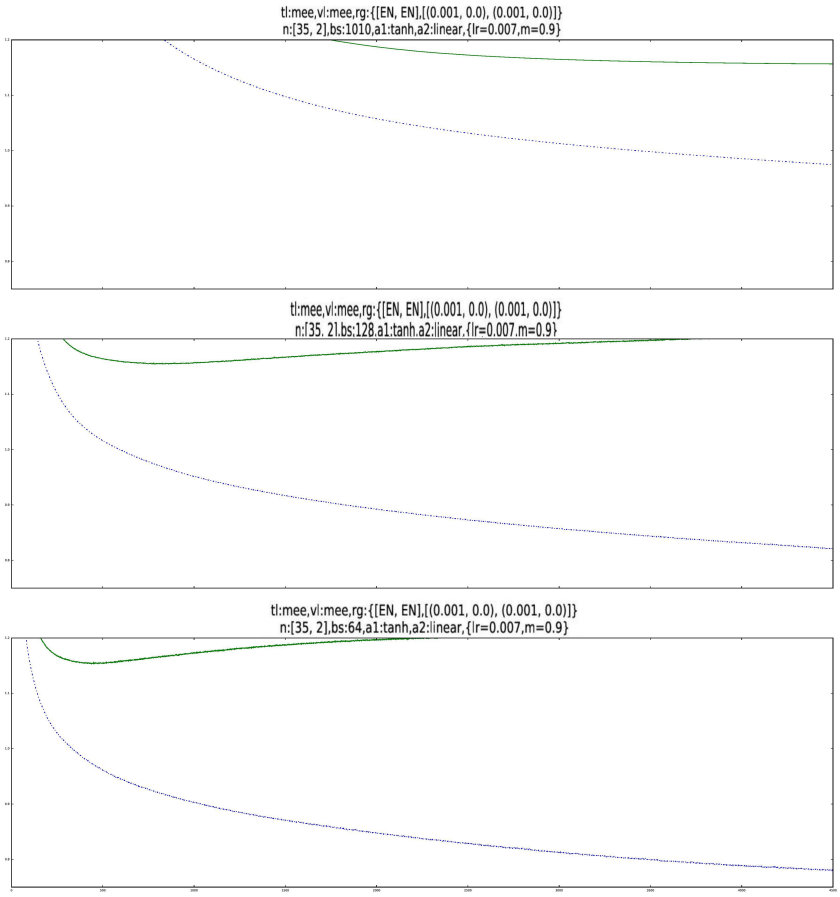
Effect of using no mini-batches, mini-batches of size 64 and 128 over 20 trials. Batches achieve a faster convergence as is to be expected but are not able to reach a better minima.



range 0-3

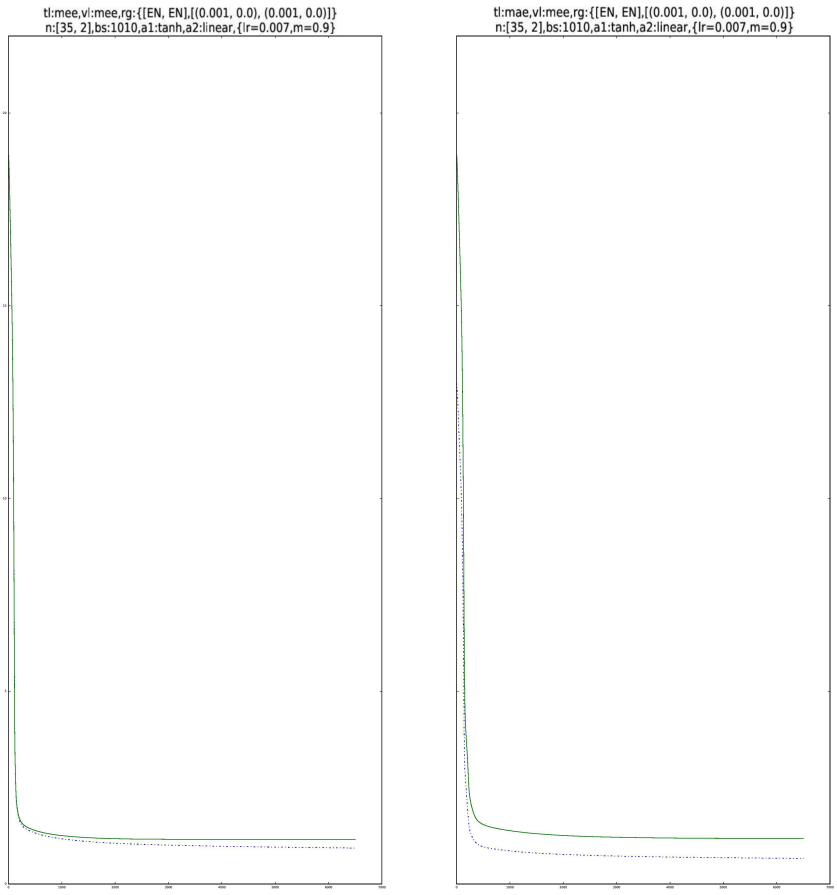

tl:mee,vl:mee,rg:{[EN, EN],[(0.001, 0.0), (0.001, 0.0)]}
n:[35, 2],bs:1010,a1:tanh,a2:linear,{lr=0.007,m=0.9}
tl:mee,vl:mee,rg:{[EN, EN],[(0.001, 0.0), (0.001, 0.0)]}
n:[35, 2],bs:128,a1:tanh,a2:linear,{lr=0.007,m=0.9}
tl:mee,vl:mee,rg:{[EN, EN],[(0.001, 0.0), (0.001, 0.0)]}
n:[35, 2],bs:64,a1:tanh,a2:linear,{lr=0.007,m=0.9}

range 0.5-1

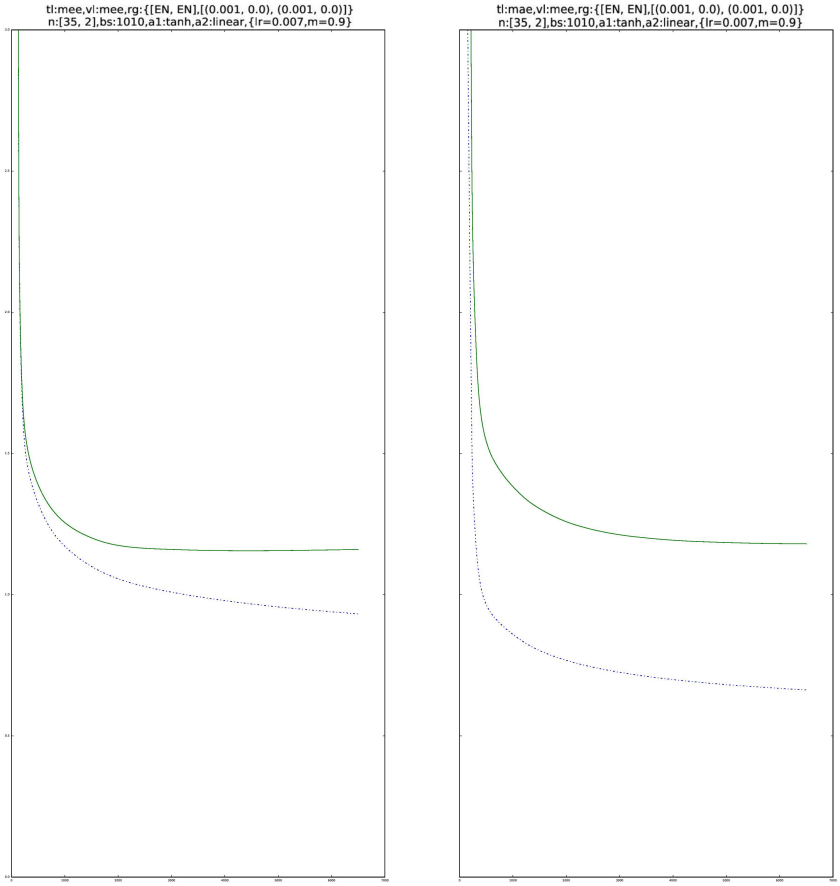

tl:mee,vl:mee,rg:{[EN, EN],[(0.001, 0.0), (0.001, 0.0)]}
n:[35, 2],bs:1010,a1:tanh,a2:linear,{lr=0.007,m=0.9}
tl:mee,vl:mee,rg:{[EN, EN],[(0.001, 0.0), (0.001, 0.0)]}
n:[35, 2],bs:128,a1:tanh,a2:linear,{lr=0.007,m=0.9}
tl:mee,vl:mee,rg:{[EN, EN],[(0.001, 0.0), (0.001, 0.0)]}
n:[35, 2],bs:64,a1:tanh,a2:linear,{lr=0.007,m=0.9}

C.1.1.3 MEE vs MAE

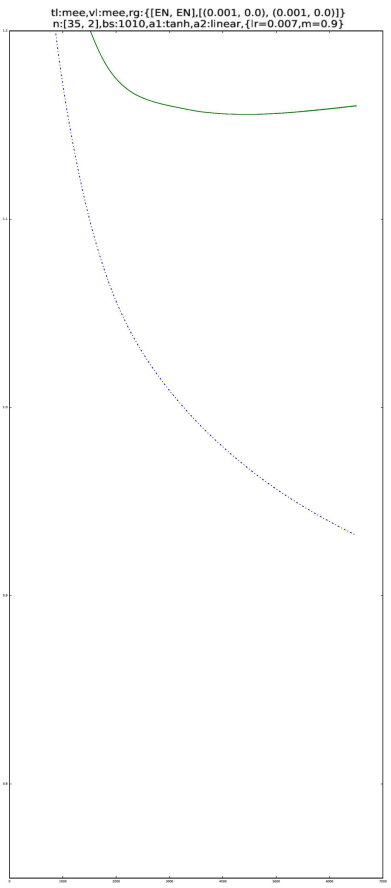
Effect of using a different training function while maintaining MEE as validation one over 20 trials. In this case Mean Absolute Error (MAE) is used for the training.

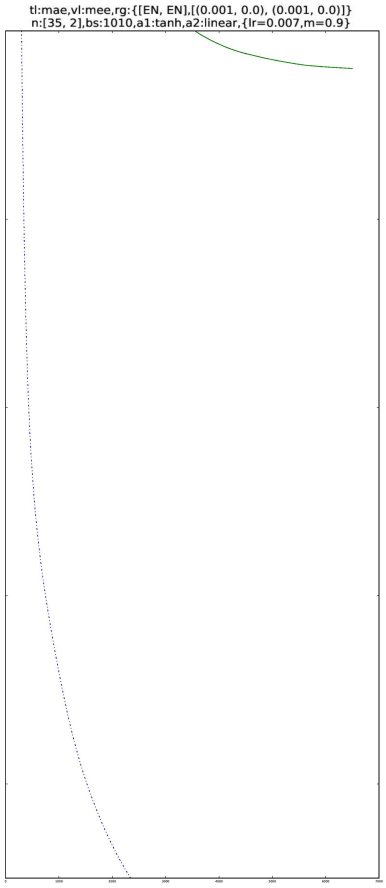


range 0-3


tl:mee,vl:mee,rg:{[EN, EN],[(0.001, 0.0), (0.001, 0.0)]}
n:[35, 2],bs:1010,a1:tanh,a2:linear,{lr=0.007,m=0.9}
tl:mae,vl:mee,rg:{[EN, EN],[(0.001, 0.0), (0.001, 0.0)]}
n:[35, 2],bs:1010,a1:tanh,a2:linear,{lr=0.007,m=0.9}

range 0.5-1

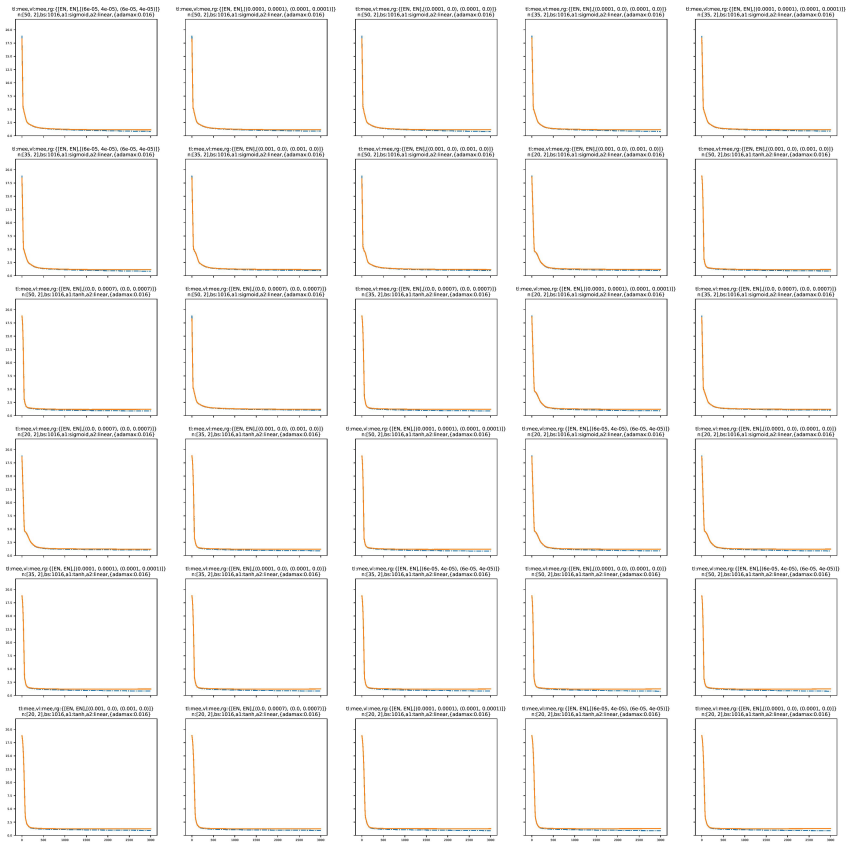

tl:mee,vl:mee,rg:{[EN, EN],[(0.001, 0.0), (0.001, 0.0)]}
n:[35, 2],bs:1010,a1:tanh,a2:linear,{lr=0.007,m=0.9}


tl:mae,vl:mee,rg:{[EN, EN],[(0.001, 0.0), (0.001, 0.0)]}
n:[35, 2],bs:1010,a1:tanh,a2:linear,{lr=0.007,m=0.9}

C.1.2 Phase 2: fine grid search

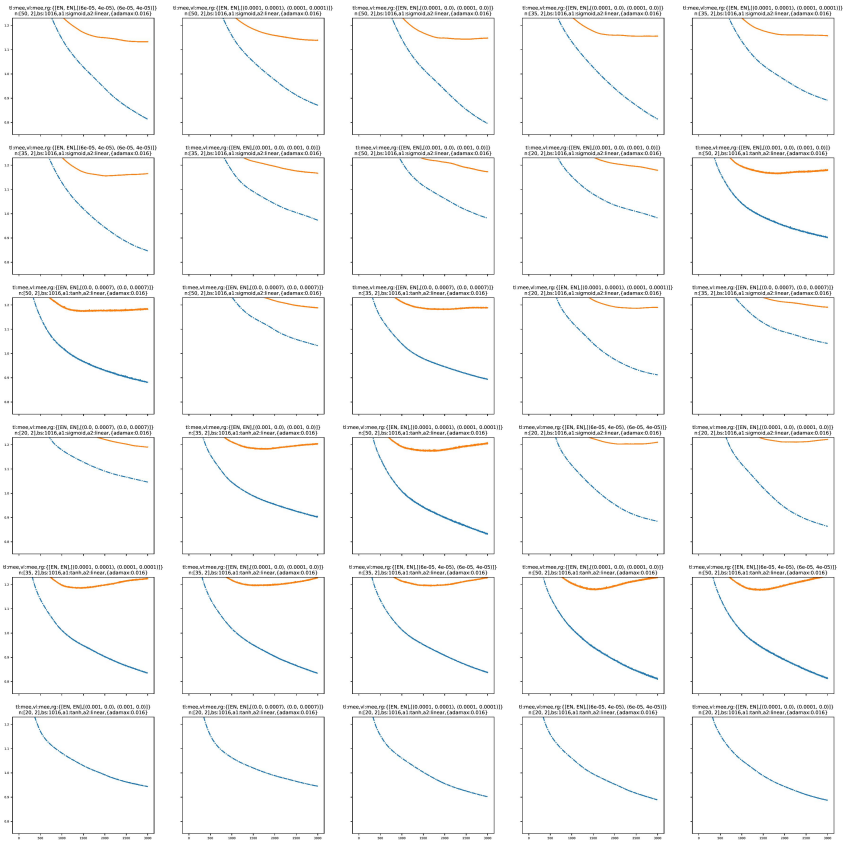
C.1.2.1 Results with adamax optimizer

Fine grid search over adamax

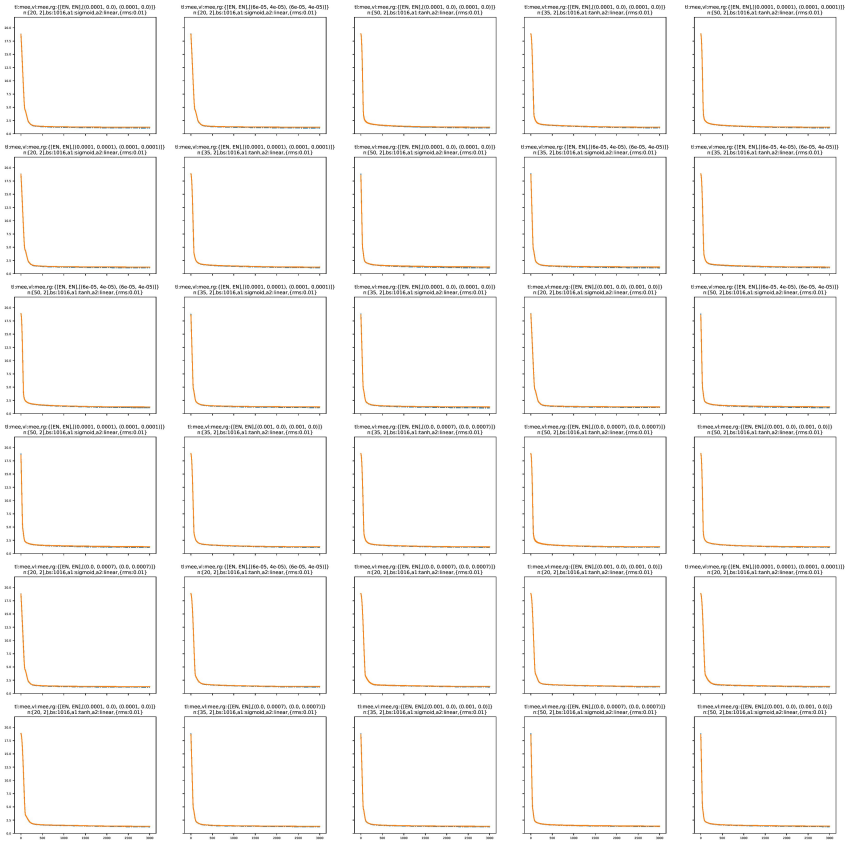


range 0-3

range 0.5-1



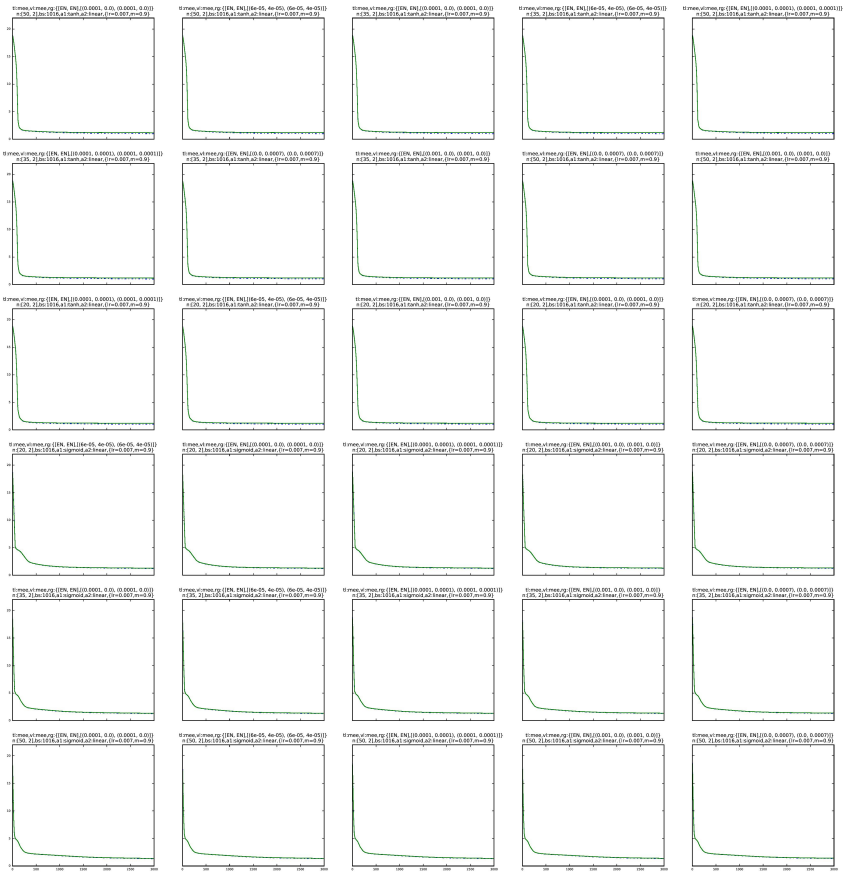
C.1.2.2 Results with RMSprop optimizer



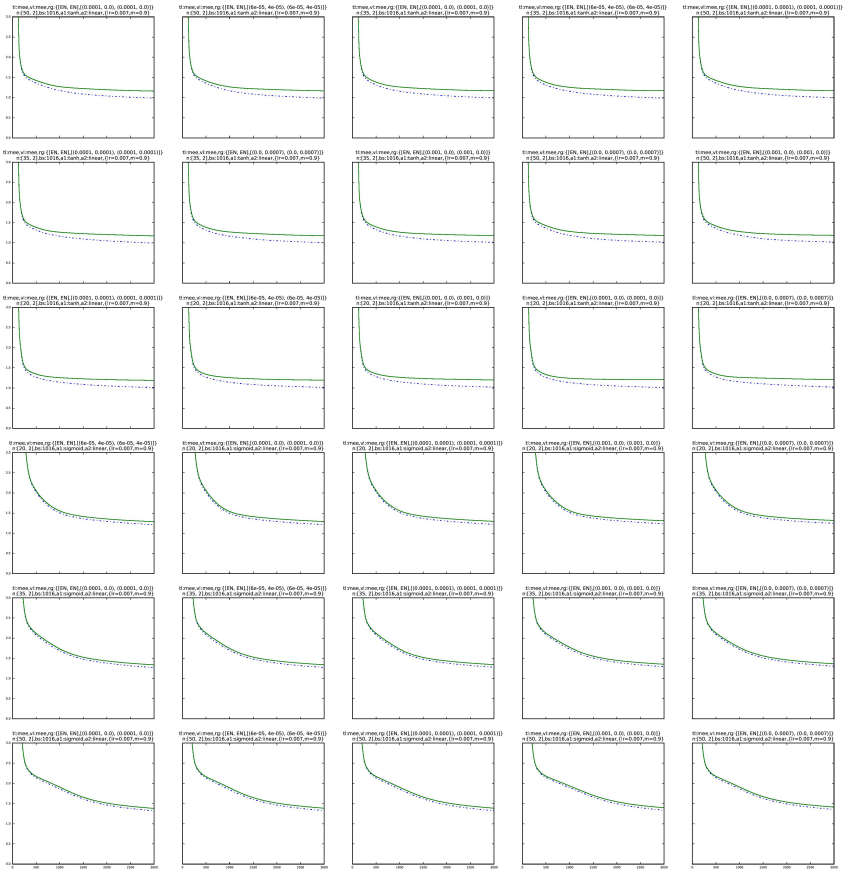
range 0-3

range 0.5-1

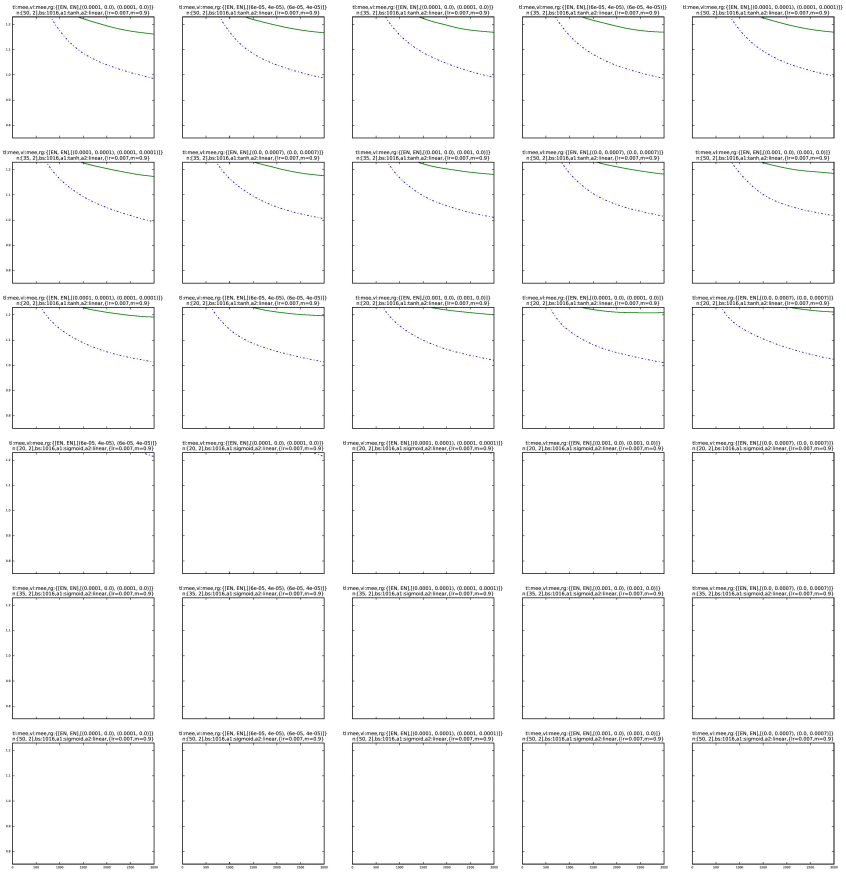
C.1.2.3 Results with SGD with momentum optimizer



range 0-3

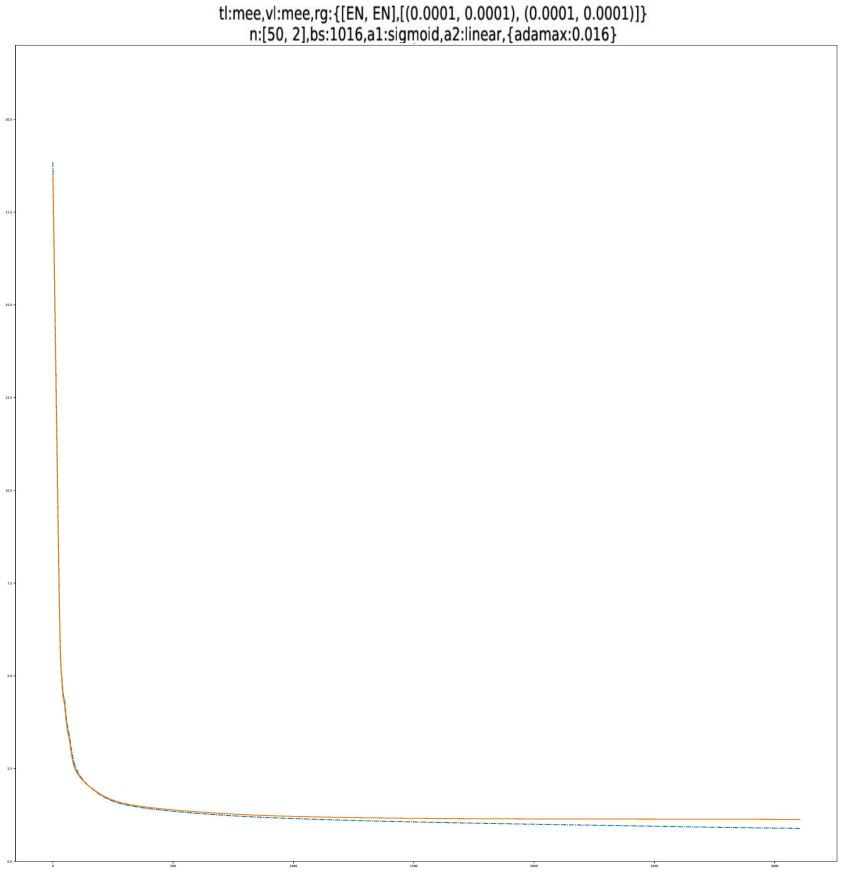


range 0.5-1

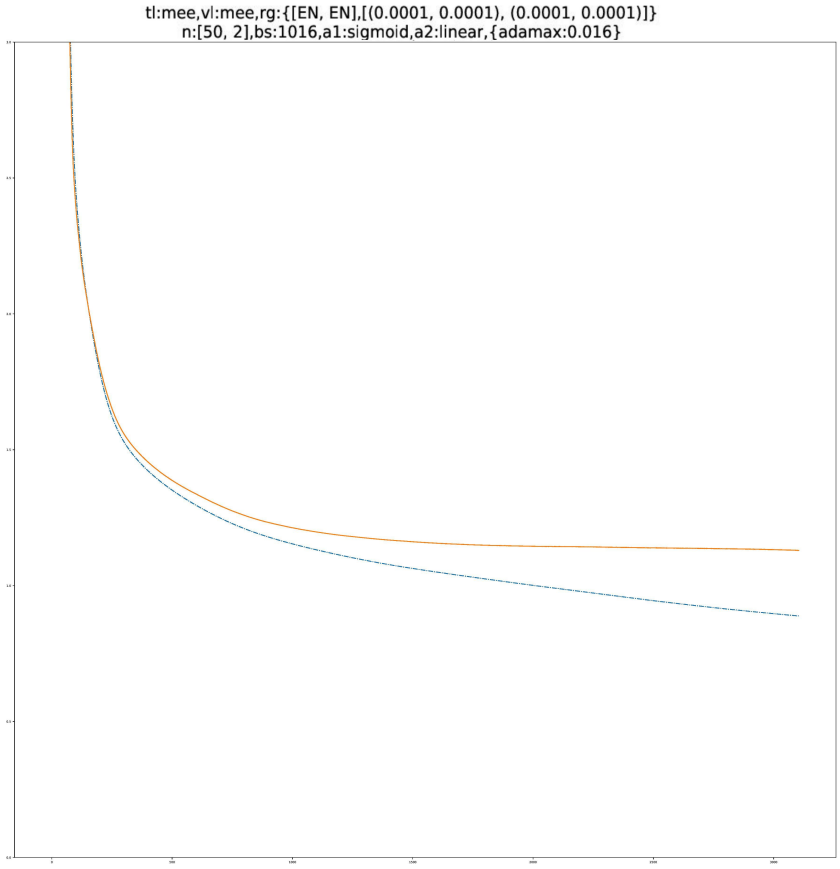


C.1.2.4 ML cup 2016 Dataset

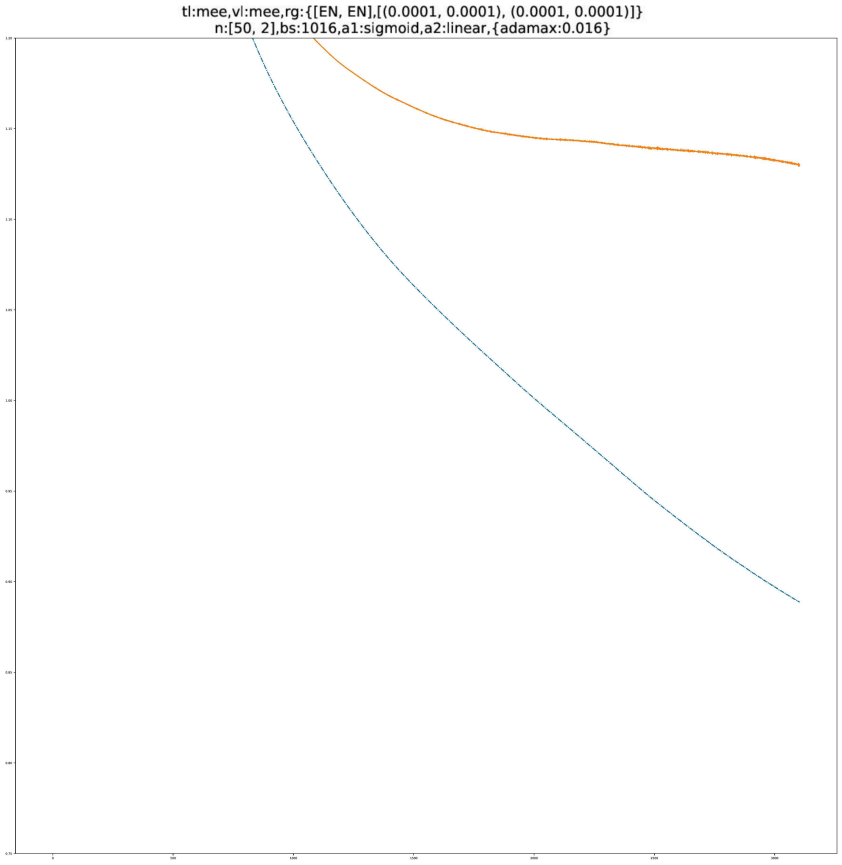
Just out of curiosity we tried the performance of the same model on the previous ML cup over 10 trials, and as expected obtained a pretty similar result.



range 0-3


tl:mee,vl:mee,rg:{[EN, EN],[(0.0001, 0.0001), (0.0001, 0.0001)]}
n:[50, 2],bs:1016,a1:sigmoid,a2:linear,{adamax:0.016}

range 0.5-1


tl:mee,vl:mee,rg:{[EN, EN],[(0.0001, 0.0001), (0.0001, 0.0001)]}
n:[50, 2],bs:1016,a1:sigmoid,a2:linear,{adamax:0.016}

Bibliography

(1) G. Rossum, *Python Reference Manual*, tech. rep., Amsterdam, The Netherlands, The Netherlands, 1995.

(2) S. van der Walt, S. C. Colbert and G. Varoquaux, "The NumPy Array: A Structure for Efficient Numerical Computation", *Computing in Science & Engineering*, 2011, **13**, 22–30.

(3) H. Leung and S. Haykin, "The complex backpropagation algorithm", *IEEE Transactions on Signal Processing*, 1991, **39**, 2101–2104.

(4) B. Polyak, "Some methods of speeding up the convergence of iteration methods", 1964, **4**, 1–17.

(5) Y. Nesterov, "A method of solving a convex programming problem with convergence rate O (1/k2)".

(6) D. Kingma and J. Ba, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

(7) T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude", *COURSERA: Neural networks for machine learning*, 2012, **4**, 26–31.

(8) N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting.", *Journal of machine learning research*, 2014, **15**, 1929–1958.

(9) G. Gordon and R. Tibshirani, "Accelerated first-order methods", *Optimization*, 2012, **10**, 725.

(10) H. Zou and T. Hastie, "Regularization and variable selection via the elastic net", *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2005, **67**, 301–320.

(11) S. B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. D. Jong, S. Dzeroski, S. E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R. Michalski, T. Mitchell, P. Pachowicz, Y. Reich, H. Vafaie, W. V. D. Welde, W. Wenzel, J. Wnek and J. Zhang, *The MONK's Problems A Performance Comparison of Different Learning Algorithms*, tech. rep., 1991.

(12) F. Chollet et al., *Keras*, `https://github.com/keras-team/keras`, 2015.

(13) M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu and X. Zhang, "TensorFlow: A system for large-scale machine learning", *CoRR*, 2016, **abs/1605.08695**.

(14) Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions", *arXiv e-prints*, 2016, **abs/1605.02688**.

(15) F. Seide and A. Agarwal, *CNTK: Microsoft's Open-Source Deep-Learning Toolkit*, 2016.

(16) *AA1 CUP*, `http://pages.di.unipi.it/micheli/DID/CUP-AA1/`.

(17) V. Hodge and J. Austin, "A Survey of Outlier Detection Methodologies", *Artificial Intelligence Review*, 2004, **22**, 85–126.

(18) X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", 2010.

(19) Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard and L. D. Jackel, Advances in neural information processing systems, 1990, pp. 396–404.

(20) Y. LeCun, "The MNIST database of handwritten digits", *http://yann. lecun. com/exdb/mnist/*, 1998.

(21) I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, `http://www.deeplearningbook.org`, MIT Press, 2016.

(22) L. Bottou, Proceedings of the symposium on learning and data science, Paris, 2009.

(23) M. Gürbüzbalaban, A. Ozdaglar and P. Parrilo, "Why random reshuffling beats stochastic gradient descent", *arXiv preprint arXiv:1510.08560*, 2015.

(24) S. Ioffe and C. Szegedy, International Conference on Machine Learning, 2015, pp. 448–456.

(25) E. Ghadimi, H. R. Feyzmahdavian and M. Johansson, "Global convergence of the Heavy-ball method for convex optimization", 2015, 310–315.

(26) N. S. Keskar and R. Socher, "Improving Generalization Performance by Switching from Adam to SGD", *arXiv preprint arXiv:1712.07628*, 2017.

(27) C. Zhang, Q. Liao, A. Rakhlin, K. Sridharan, B. Miranda, N. Golowich and T. Poggio, *Theory of Deep Learning III: Generalization Properties of SGD*, tech. rep., Center for Brains, Minds and Machines (CBMM), 2017.

(28) M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean et al., Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on, 2013, pp. 3517–3521.

(29) R. Pascanu, T. Mikolov and Y. Bengio, International Conference on Machine Learning, 2013, pp. 1310–1318.

(30) *Intel i5 2520m processor*, `https://ark.intel.com/it/products/52229/Intel-Core-i5-2520M-Processor-3M-Cache-up-to-3_20-GHz`.