Reproducibility Challenge of the paper: Numerical influence of ReLU'(0) on backpropagation

Martorella Tommaso, Ramírez Contreras Héctor Manuel, Cerezo García Daniel
CS433 - Machine Learning - Project 2, EPFL, Switzerland

***Abstract*—Neural networks have become very common in machine learning, and new problems and trends arise as the trade-off between theory, computational tools and real-world problems become more narrow and complex.**

We decided to retake the influence of the ReLU'(0) on the backpropagation as it has become more common to use lower floating point precisions in the GPUs so that more tasks can run in parallel and make training and inference more efficient. As opposed to what theory suggests, the original authors shown that when using 16- and 32-bit precision, the value of ReLU'(0) may influence the result.

In this work we extended some experiments to see how the training and test loss are affected in simple and more complex models.

I. Introduction

When working with machine learning models, it becomes very important to be aware of the implications that the hardware and technical limitations can have in the model training, architecture selection, memory and the computational cost.

In the research paper Numerical influence of ReLU'(0) on backpropagation [1] that we are reproducing, one of the main arguments is the difference between the theoretical (non limited in precision) and technical capabilities of a model. In theory there's no memory or bit-precision that can alter the result, nevertheless, computational limitations must be taken into account when training the model. We also run additional experiments to further extend the authors' idea of the value of the subgradient being an hyperparameter to tune during training.

II. Scope of Reproducibility

The original paper [1] states that when using a 32 and 16 bit precision for the tensor storage and computation, the bifurcation zone (i.e. the set $S_{0,1}$ of weigths such that $backprop_0 \neq backprop_1$ for two identically initialized and trained models) is met with non-null probability. The choice of ReLU'(0) becomes computationally meaningful and influences the training and test accuracy. As the trend to lower the precision to make the model training more efficient in terms of energy, memory and resources is arising, it was very important for us to understand and make sense of the implications of changing the value of ReLU'(0) in practice.

The main takeaway from the original paper is that the arbitrary choice of mathematically negligible factors (such as the ReLU'(0)) might not be computationally negligible. Nevertheless, the main conclusion is that ReLU'(0) = 0 (which is what is conventionally used both in theory and in the major deep learning frameworks) seems to be the most robust selection for training across the variations the original authors tried. Given our goal to see the influence of such choices where low precision might be meaningful, we run our tests on both simple and more complex models, including MobileNet V3 small [2]: a model engineered to work in low resources use-cases.

One of the main motivations is that as models become bigger and more complex, errors due to numerical precision happen with relatively high frequency. Additionally, the necessity of using smaller bit-precision increases in environment with memory, time, and energy constraints. In the academic case it is also interesting to use smaller precision, because it helps to use our resources better. Our objectives then were to understand the theoretical background, the potential influence on the practical scope, and if there's room to take advantage of this peculiarity.

To achieve that, we first reproduced the experiments to support the main claims of the original paper (detect and measure the difference on fully connected networks trained with different values for ReLU'(0)), then we tried to use the subgradient as a hyperparameter during tuning to see if we could reach better performance (measured as accuracy on the validation set).

III. Methodology

Given the existing codebase from the original paper[1], our approach consisted in applying a similar approach to the authors' one and adapt part of the existing implementations to suit our goals. Nevertheless, the vast majority of the experiments and codebase has been redesigned, so that we could get a better understanding of the procedure and properly explain the results and make additional explorations (such as hyperparameter tuning). We structured the code using standard software engineering practices to achieve the flexibility and extensibility of our testing framework which allowed us to run experiments with ease.

Tools we used include Kaggle cloud services[2] as well as

[1]https://github.com/deel-ai/relu-prime
[2]https://www.kaggle.com/

our personal devices with GPUs that didn't allow us to run large models or a large number of experiments.

The first results were reproduced as we first needed to see if the code was properly implemented. Once the results were consistent, we started to think about reproducing results with different architectures, optimiser, normalization, and activation functions (that had any point with sub-gradients). The description of the models and experiments can be found in this section.

A. Model descriptions

We investigate the influence of the subgradient hyperparameter in different architectures to see if the findings could be relevant to a broader spectrum of models. In particular we run our tests on the following models:

- Fully connected NN
- ResNet-18
- MobileNet V3

We chose the fully connected architecture because of its flexibility, customizability, and because it is densely packed with ReLU activations so the effect can be easily analyzed. We are also trying to maintain enough similarities with the original paper so we can compare the results. The ResNet-18 and MobileNet V3 were chosen to investigate the effect on CNNs with residual blocks and inverted residual blocks respectively. Additionally, the MobileNet V3 mixes ReLU activations (or ReLU6) with Hardsigmoid ones, which further increases variability.

B. Data sets

In order to reproduce the paper without the need of having a complex model and data set that could take much to train, we decided to work with the following datasets:

- MNIST
- FashionMNIST

The MNIST[3] (and its variation FashionMNIST[4]) dataset allowed us to run manageable experiments given our computational constraints, as it is very light and there was no need of a big storage capacity. Moreover, according to our goals, it was enough to start with it and afterwards, if possible, extend the experiments to a more complex data set.

The dataset consists in both a training and a test set. It has 60,000 samples for training, and has different hand-written numbers between 0-9. Each element is a 28x28 gray-scale image, this means that we just had to deal with one channel per image and 784 pixels in total. However, MobileNet V3 is build to deal with input with 3 channels, so before training we transformed the grayscale image into a 3-channel RGB image with the same grayscale value for each channel. The test set has 10,000 images with the same dimensions.

[3]http://yann.lecun.com/exdb/mnist/

[4]https://github.com/zalandoresearch/fashion-mnist

Each experiment has different requirements, in some of them we used a shorter set or different batch sizes. The objective was to measure the performance of the models given different computational specifics (precision) and hyperparameters.

C. Hyperparameters

As stated in the original paper and as seen in the experiments we reproduced [subsection], the value of ReLU'(0) becomes a hyperparameter when training. We run the initial experiments both in 32-bit and 16-bit precision to compare the influence of numerical precision on the size of the bifurcation zone. We decided not to run them in 64-bit precision as well because, as evidenced in the original paper as well, it is less likely to have numerical errors and therefore a rounding to exactly 0 in the forward pass, and hence to observe an influence by the choice of ReLU'(0) value. In summary, some of the most important and general hyperparameters to tune were the following:

- Precision: 16 or 32 bit precision
- Model architecture
- Activation function: ReLU vs ReLU6 vs LeakyReLU
- Value of the subgradient in the non-differentiable point(s)
- Batch Size

For each experiment we used the ADAM optimizer with learning rate 0.001 and for most experiments we used a fixed batch size of 128.

D. Experimental setup and code

The models and methods section should describe what was done to answer the research question, describe how it was done, justify the experimental design, and explain how the results were analyzed.

The model refers to the underlying mathematical model or structure which you use to describe your problem, or that your solution is based on. The methods on the other hand, are the algorithms used to solve the problem. In some cases, the suggested method directly solves the problem, without having it stated in terms of an underlying model. Generally though it is a better practice to have the model figured out and stated clearly, rather than presenting a method without specifying the model. In this case, the method can be more easily evaluated in the task of fitting the given data to the underlying model.

E. Computational requirements

This work studies the effect of the subgradient in the back-propagation step, thus we needed to compute several models or subgradiants using pytorch. We ran the experiments using Python 3.9.12 and PyTorch 1.13.1with CUDA 11.7 to use the GPU. We ran the experiments both using COLAB's GPU and local GPU's (Nvidia GeForce RTX 3060).

IV. Results

We selected some of the experiments that we found most meaningful to reproduce, cause the more complex we made it, the longer it would take and we focused mainly in the weight difference (or L_1 norm between the weight matrices), the volume of the gradient matrix in the backpropagation (comparison of differences in the gradients) and the performance of the models with both training and testing losses.

A. *Initial Experiment*

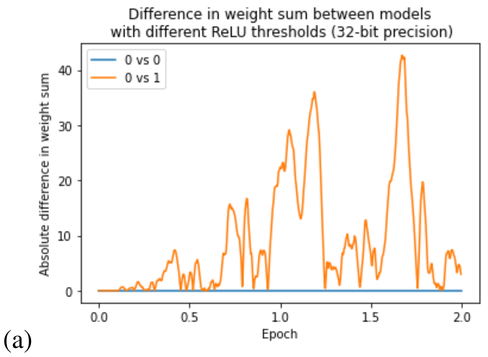
The first step was to compare the weight difference of the parameters with both 32 and 16 bit precision model and check the bifurcation zone, to see when it it starts to diverge, and if it diverges get some insights about the reasons. What we basically did here was to measure the L^1 norm between both weight matrices to have a perception of how different they became. This part does not measure weather one solution is better than the other, but it helps us to understand the magnitude of the change. This is one important point because it explains the importance of the experiments performed in this paper. We can observe that even with 32-bit precision we can notice a small difference in performance (validation accuracy), depending on the election of the ReLU'(0) value, and there is indeed a bigger effect when using 16-bit precision. Therefore, it turns reasonable and interesting to analyze how big is this difference when choosing different values in ReLU'(0) and which of the values in the interval [0,1] is more desirable. We notice, however, as can be seen in figures 4 (a) and (b) that the lower the precision is the more unstable the networks are and the weight difference between them varies rapidly because the exact activation value of zero is reached more frequently (reproducible notebook `initial_experiment.ipynb` available).

B. *Fully connected Neural Network*

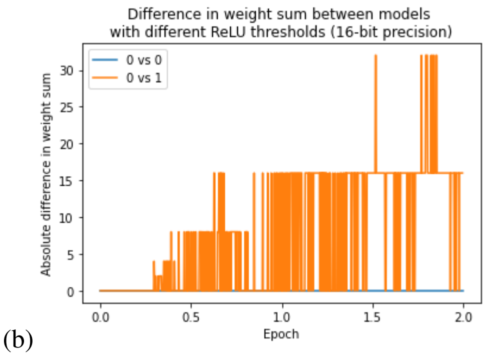
After we observed the first results with the functions we implemented, and that they were consistent with the paper, we decided to run the experiments with a fully connected neural network. Some hyperparameters were the same for all the experiments, we just changed the value of $ReLU'(0) \neq 0$ in different bit precision to observe if there were any changes in loss and accuracy of the models with this difference and to measure them on a similar conditions. The hyper parameters that we decided to fix were:

- Number of epochs: 50
- Optimizer: ADAM with our default learning rate
- Number of layers: 3
- Nodes in hidden layers: 200
- Regularization: Not used for this experiment (Identity)

With this hyperparameters fixed, the values for the back propagation we used were: $ReLU'(0) = [0, -0.5, 0.5, 0.8, 1]$. This way we could see a difference in the performance.



(a)



(b)

Figure 1. (a) $|\theta_0 - \theta_1|_1$with 32-bit precision. (b) $|\theta_0 - \theta_1|_1$ with 16-bit precision.

1) Results with 32-bit precision: When comparing the weight sum, we could immediately observe that the use of $ReLU'(0) \neq 0$ does have a big effect in the weights, they start to have different magnitudes and the difference become bigger, Nevertheless that does not mean the accuracy must be much worse, as we could be getting to a local minima that is also a solution to the problem.

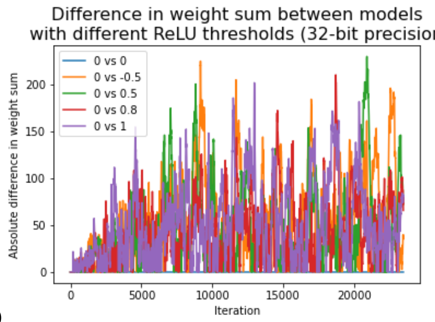
For the loss in the training we observe that with 32-bit precision there's no big difference, even though the weights are very different in magnitude, which just indicate us that even in those sub-gradients, the solution still converges, though the value of $ReLU'(0) = 0$ looks more stable, but with this number of epochs might not be enough to see if it eventually becomes more chaotic.

We also tested the models with the test or validation set to see if the solution was, indeed good, or if it was overfitting the model. The figure [???]

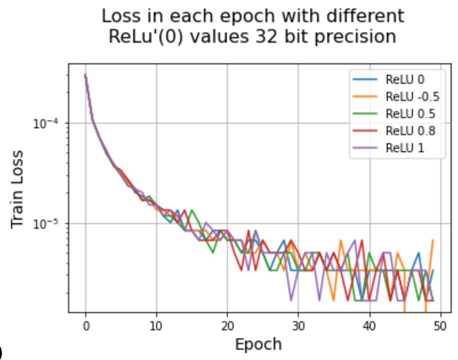
2) Results with 16-bit precision: With 16-bit precision it becomes evident that we should be considering the value of $ReLU'(0)$ when training, and it becomes important the chose of $ReLU'(0) = 0$ as a correct decision. With other values the behaviour becomes chaotic, both for training and testing.

Firstly, we can observe that the values of the weights become different with a logarithmic behaviour, but it was also imperative to test weather this was just getting to different but accurate solutions of the problem.

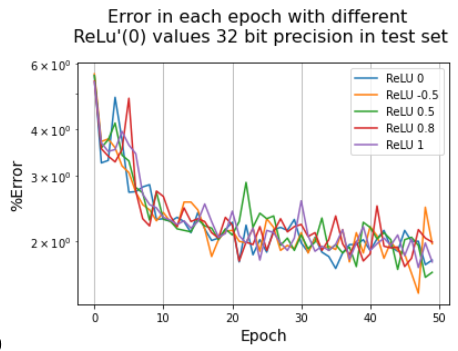
The next step was to see the training loss and check



(a)



(b)



(c)

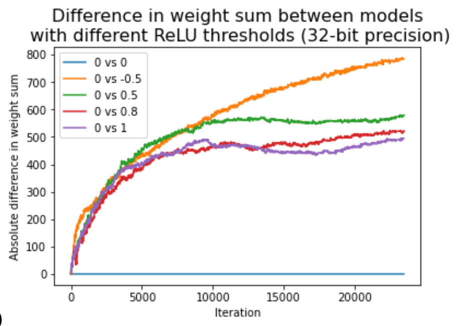
Figure 2. (a) $|\theta_0 - \theta_1|_1$ with 32 bit precision and different $ReLU'(0)$ values (b) Training loss of the model with different subgradients (c) Test Loss of the model with different subgradients.

weather the solutions were accurate, and if so, also determine if there was a big change or if it was better or worse changing the value of $ReLU'(0)$. When inspecting the plot, we could easily observe that the behaviour became very chaotic and that the higher the value we chose, the faster it would diverge from the solution. It would be also interesting to see if could change with a different learning rate or optimizer.

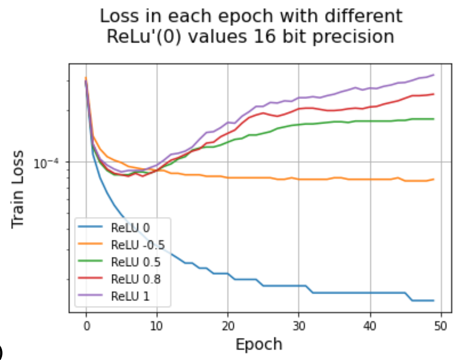
After this experiment, we proceeded to validate the model with the test data set, and the behaviour was quite close.

Comparison between precision accuracy. As last step we decided to compare the distribution of the test accuracy. This was a good way to compare the performances of each model with different values.

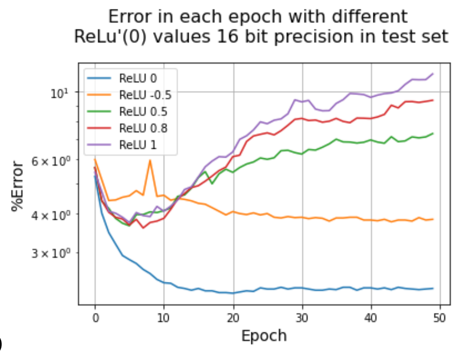
It became evident that, not only the accuracy is worse depending of the value of the subgradient, but it's also less



(a)



(b)



(c)

Figure 3. (a) $|\theta_0 - \theta_1|_1$ with 16 bit precision and different $ReLU'(0)$ values (b) Training loss of the model with different subgradients (c) Test Loss of the model with different subgradients.

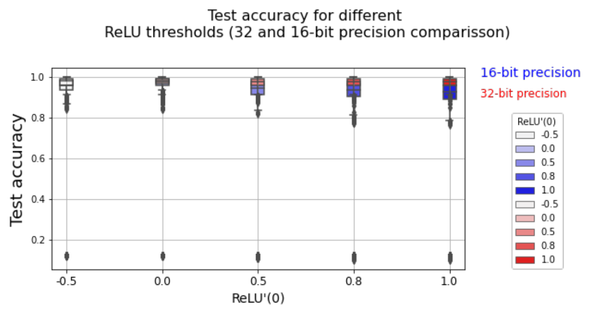


Figure 4. Comparison between model accuracies using box-plots, this allows us to look at the distribution of the accuracy through all the experiments

stable, the standard deviation gets bigger with the value of $ReLU'(0)$ which is an indicator that for this model the most robust election is the value of $ReLU'(0) = 0$.

C. *Volume measurement*

Other important point was to have a notion of how the weights were changing the more complex the model became.

For this we decided to get the volumes for fully connected models, as it was done in the paper. In this case we needed to make smaller models due to limitations in the computing resources. Nevertheless, at small scale we could observe similar tendencies.

For this experiments there were several variations. But through all of them, the two models have fixed subgradients

- Model 1 : $ReLU'(0) = 0$
- Model 2 : $ReLU'(0) = 1$

It's important to note that in this experiments we do not do the optimization step, as it's not the objective to measure the performance of the model but the difference between the gradient matrices, that can indicate us how different the models are and how many zeros we have when changing the parameters.

The first experiment was to iterate with different depths or number of layers. It had to be done with 64, 32 and 16 bit precision so that we could better compare the results. as 64 is less likely to get $ReLU(0)$, so that was our reference.

In the figure (a) we see that the deeper the model, the higher the probability of getting a value of 0 at the ReLU point with the 16 and 32 precision, at 16 bit this becomes much more relevant as we can see that the gradient matrix start to become much different even in small layers. With 32 bit (PyTorch default value) the difference is not despicable the deeper the model is.

We observe a similar behaviour with the number of neurons per layer. The more neurons, the more likely that a value can become 0. And again, this probability is much higher for the 16-bit precision.

For the Size and batch size we could observe a difference with 32 bit precision but it was not significant and there is not much variation, whereas at 16 bit precision it still very significant even with batch and sample sizes.

D. *ResNet18*

After testing the performance of the Fully Connected model, it was interesting to reproduce these experiments in a convolutional neural network, such as the ResNet18 model, in order to compare the neural networks' behaviour, and to study the differences given in how much models are affected by the election of the value ReLU'(0). The model has some layers that use Batch normalization.

In the original paper, the authors ran experiments using the batch norm, and the results were that it stabilize the weight matrices. We wanted to put this to the test and run it with different subgradient values to measure the behaviour

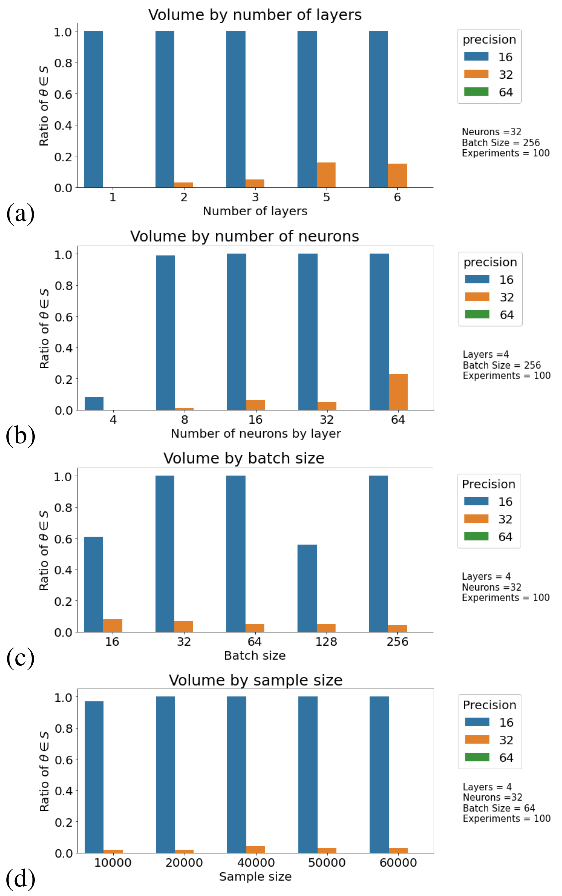


Figure 5. The volume of the model weights was a measurement of the difference between the gradient matrices (a) Comparison by varying the number of layers (b) Comparison by varying the number of neurons by layer (c) Comparison by varying the number of batch sizes for the forward and backward pass (d) Comparison by varying the number of samples from the dataset.

of the models and see if for this for more complex models this difference could be much significant.

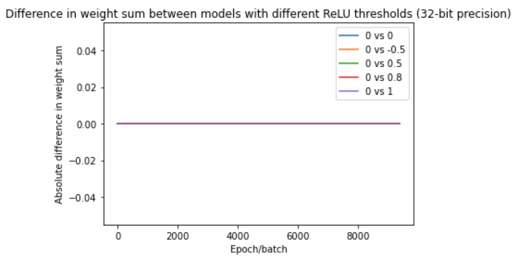
We decided to use this specific structure as it does not have an excessive amount of layers, so it is feasible to work with it, but has an excellent and representative functionality, thus it is a good option to evaluate the desired outcome.

In this case, as the architecture is fixed, the only parameter we changed was the value used at ReLU'(0), where we tested the same set we employed in the previous analysis $ReLU'(0) = [0, -0.5, 0.5, 0.8, 1]$.

The hyper parameters that we decided to fix were:

- Number of epochs:20
- Optimizer: ADAM with our default learning rate.
- Regularization: Batch norm (build up in the model)

We could run the experiments in the 32 bit precision model and the results were consistent with the paper's claims. When adding a batch normalization in each layer, the model became much more stable. This was perhaps because there were not many values getting to 0.

(a)


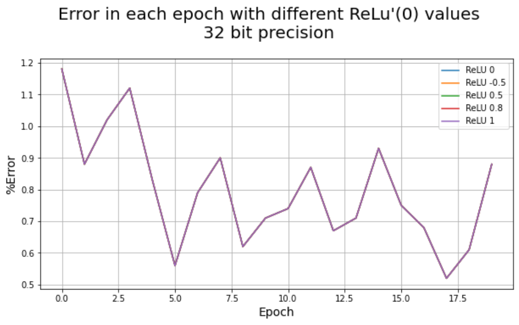
(b)


Figure 6. (a) The L_1 norm of the weights w.r.t the ReLU'(0) (b) The test error in the different models

Something interesting when ruining it with the 32 bit precision, was that the model was very stable with any subgradiant value. Which indicate us that this model is very stable thanks to the batch norm and that many weights might not get to the ReLU(0) value.

We were not able to complete the 16-bit precision tests for this model due to execution problems, though we could see how stable it was with 32-bit precision. When comparing it with a fully connected neural network it becomes evident.

E. *MobileNet V3*

Reproducing the initial experiment with MobileNet and ReLU as the activation function led to no differences detected in the first 2 training epochs as can be seen in the notebook in the GitHub repo. However, we used MobileNet to see if the idea of using the value of the subgradient could be actually used as a real hyperparameter in an hyperparameter tuning process. To do that, we used ray tune that allowed us to do the random grid search in the hyperparameter space and checkpoint the models.

To run the experiments we chose:

- Number of epochs: 6 (due to computational power limit)
- Optimizer: ADAM
- Activation function: ReLU6

ReLU6 has two non-differentiable points which we will call *alpha* and *beta*

The hyperparameter space we set for the experiment included:

- Batch size: choice [32, 64, 128]
- ADAM learning rate: loguniform [1e-4, 1e-2]
- *alpha*: uniform [0, 1]
- *beta*: uniform [0, 1]

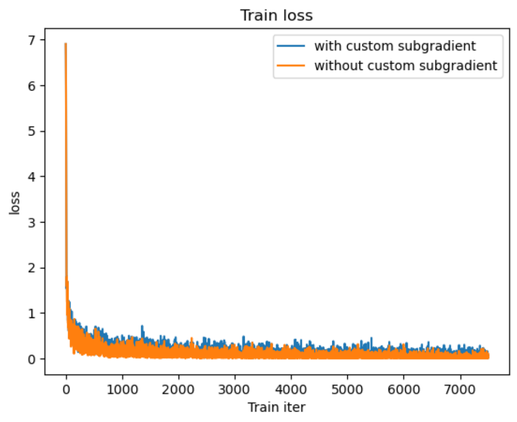
The most promising model we obtained has reached the 96.9167% accuracy with a cross-entropy loss of 0.101213 by using the following parameters:

- Batch size: choice 64
- ADAM learning rate: 0.00170239
- *alpha*: 0.496422
- *beta*: 0.455249

which is still lower than the value of the hyperparameter tuning we did using the default *alpha* and *beta* (accuracy: 97.3583%, cross-entropy loss: 0.0869832, batch size: 64, learning rate: 0.0031722)

The results are coherent with the findings above. Although the low number of epochs, a different choice for the subgradient in both non-differentiable points induces a chaotic behaviour that makes the solution less stable and precise.

Even trying to further train the best models for more epochs and a smaller learning rate leads to the same result, with the default choice of ReLU6'(0) = ReLU6'(6) = 0 being the more appropriate.

(a)


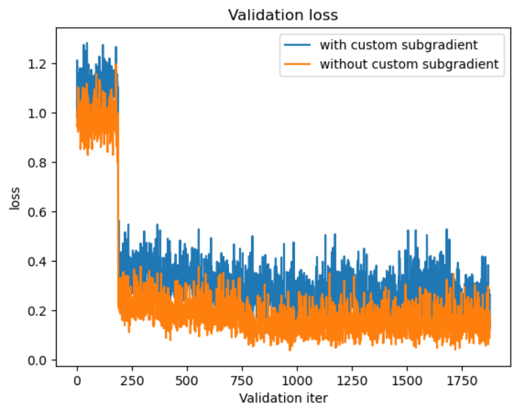
(b)


Figure 7. (a) Train loss while training the two best models (including and excluding *alpha* and *beta* as hyperparameters) (b) The validation error

We ran the same hyperparameter tuning process with the random grid search for the fully connected model (see notebook `hyperparameter_tuning_fullyconnected.ipynb`) obtaining the same results coherent with the whole analysis above.

However we noticed that when training the model with a customized value of the subgradients the non-differentiable points are reached less frequently as the training goes on, while training without the default subgradient the non-differentiable points are hit more frequently (see notebooks `hyperparameter_tuning_fullyconnected.ipynb` and `hyperparameter_tuning_mobilenet.ipynb`)

V. Discussion

The results that we got were very coherent with the ones exposed in the paper. Although we were not able to run as many experiments and with very large models, we got a big insight about the topic. It's something that is somehow negligible in theory, but when combining the computational precision, the bad election of the subgradiant can lead to a chaotic behaviour. It would have been a great breakthrough if the results lead to conclude that the best value of a subgradiant could be something different than 0, nevertheless, it gives a big support to the default values and gives a solid answer to the question: What would it be the best subgradiant election? The theory says: it doesn't matter, yet it does.

A. What was easy

As PyTorch has already build-in functions and the back-propagation method, customizing the functions was relatively easy, because we just had to rewrite some of the make some parameters customizable during the model design.

Model building was also a simple task, though time consuming. After developing the model creator for the fully connected networks, the task was much more easy. but we had to spend more time training, the larger the model became. The MobileNet implementation is a customization of PyTorch's built-in version so making it customizable was also a straightforward change.

B. What was difficult

Reproducing many experiments was perhaps one of the biggest challenges, as we need time and computational resources to train many models and get a better comparison. One approach we took was to make smaller, yet meaningful experiments to get the most of our data and time. We also used a simple dataset.

Due to the stochastic nature of neural network training, controlling the behaviour of the different random number generators used within PyTorch was essential to make run the comparisions and make the experiments reproducible. The initial experiments with MobileNet V3 were especially problematic because, its internal implementation was relying on RNGs not easily discoverable. Once we managed to do that, computational power became the limiting factor.

VI. Summary

The subgradient is a set of solutions to a non differentiable point of a non-smooth function, in theory it wouldn't matter the value it takes as long as is within the slope of the two functions that define the function, in the case of the ReLU function it is between 0 and 1 (Other case would be the $|x|_1$ which is between -1 and 1. Though in theory this is correct, when using numerical methods to perform a backpropagation, altogether with numerical bit-precision it becomes relevant as different use of decimals and rounds can lead to different solutions, as the use of 32 bit precision is widely used as a standard in neural network training and as 16 bit is becoming a trend to speed up the training in GPU's and energy saving, the chose of the subgradiant becomes relevant. Although 0 seems to be an adequate election, it becomes a hyperparameter when training and testing the model.

This experiments and results can be used as a solid base to keep the election of the subgradiant $ReLU'(0) = 0$ when training a model at 16 and 32 bit precision.

References

[1] D. Bertoin, J. Bolte, S. Gerchinovitz, and E. Pauwels, "Numerical influence of relu'(0) on backpropagation," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 468–479. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/043ab21fc5a1607b381ac3896176dac6-Paper.pdf

[2] A. Howard, M. Sandler, G. Chu, L. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," *CoRR*, vol. abs/1905.02244, 2019. [Online]. Available: http://arxiv.org/abs/1905.02244