A Mathematical details for Section 2

In Section A.1, we provide some elements of proof for Theorems 1 and 2. For the sake of clarity, we start by showing how the functions $g_{i,j}$ (appearing in the composition (2)) look like in the special case of fully connected ReLU networks.

A.1 Elements of proof of Theorems 1 and 2

The proof arguments were described in [5, 6]. We simply concentrate on justifying how the results described in these works apply to Definition 1 and point the relevant results leading to Theorems 1 and 2.

It can be inferred from Definition 1 that all elements in the definition of a ReLU network training problem are piecewise smooth, where each piece is an elementary $log - exp$ function. We refer the reader to [26] for an introduction to piecewise smoothness and recent use of such notions in the context of algorithmic differentiation in [6]. Let us first argue that the results of [6] apply to Definition 1.

- We start with an explicit selection representation of $\mathrm{backprop}_s\ \mathrm{ReLU}$. Fix any $s \in \mathbb{R}$ and consider the three functions $f_1 \colon x \mapsto 0$, $f_2 \colon x \mapsto x$ and $f_3 \mapsto sx$ with the selection index $t(x) = 1$ if $x < 0$, 2 if $x > 0$ and 3 if $x = 0$. We have for all x

$$f_{t(x)} = \mathrm{ReLU}(x)$$

  Furthermore, differentiating the active selection as in [6, Definition 4] we have

$$\hat{\nabla}^t f = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ s & \text{if } x = 0 \end{cases}$$

  and the right hand side is precisely the definition of $\mathrm{backprop}_s\ \mathrm{ReLU}$. This shows that we have a selection derivative as used in [6].
- Given a ReLU network training problem as in Definition 1, we have the following property.
  - All elements in the ReLU network training problem are piecewise elementary $log - exp$.
  - Each piece can identified with an elementary $log - exp$.
  - The selection process describing the choice of active function can similarly be described by elementary $log - exp$ functions with equalities and inequalities.

Therefore, we meet the hypotheses of of [6] and all corresponding results apply to any ReLU network training problem as given in Definition 1. Fix $T \geq 1$, getting back to problem (1), using [6, Definition 5] and the selection derivative described above, for each $i = 1, \ldots, N$, there is a conservative field $D_i \colon \mathbb{R}^P \rightrightarrows \mathbb{R}^P$ such that for any $s \in [0, T]$, and $\theta \in \mathbb{R}^P$

$$\mathrm{backprop}_s l_i(\theta) \in D_i(\theta).$$

Using [5, Corollary 5] we have $D_i(\theta) = \{\nabla l_i(\theta)\}$ for all θ outside of a finite union of differentiable manifolds of dimension at most $P-1$. This leads to Theorem 1 for $s \in [0, T]$. Theorem 2 is deduced from [6, Theorem 7] which shows that with probability 1, for all $k \in \mathbb{N}$, for all $n = 1, \ldots, N$ and $s \in [0, T]$

$$\mathrm{backprop}_s[\ell(f(x_n, \theta_{k,s}), y_n)] = \nabla_\theta \ell(f(x_n, \theta_{k,s}), y_n)$$

since we have $\theta_{0,s} = \theta_0$ for all s, the generated sequences in (4) does not depend on $s \in [0, T]$. This is Theorem 2 for $s \in [0, T]$, note that a similar probabilistic argument was developped in [4]. We may repeat the same arguments fixing $T < 0$, so that both results actually hold for all $s \in [-T, T]$.

A.2 The special case of fully connected ReLU networks

The functions $g_{i,j}$ in the composition (2) can be described explicitly for any given neural network architecture. For the sake of clarity, we detail below the well-known case of fully connected ReLU networks for multiclass classification. We denote by $K \geq 2$ the total number of classes.

Consider any fully connected ReLU network architecture of depth H, with the softmax function applied on the last layer. We denote by d_h the size of each layer $h = 1, \dots, H$, and by d_0 the input dimension. In particular $d_H = K$ equals the number of classes. All the functions $f_\theta : \mathbb{R}^{d_0} \to \mathbb{R}^{d_H}$ represented by the network when varying the weight parameters $\theta \in \mathbb{R}^P$ are of the form:

$$f_\theta(x) = f(x, \theta) = \mathrm{softmax} \circ A_H \circ \sigma \circ A_{H-1} \circ \cdots \sigma \circ A_1(x) \ ,$$

where each mapping $A_h : \mathbb{R}^{d_{h-1}} \to \mathbb{R}^{d_h}$ is affine (i.e., of the form $A_h(z) = W_h z + b_h$), where $\sigma(u) = \big(\mathrm{ReLU}(u_i)\big)_i$ applies the ReLU function component-wise to any vector u, and where $\mathrm{softmax}(z) = \big(e^{z_i} / \sum_{k=1}^{d_H} e^{z_k}\big)_{1 \le i \le d_H}$ for any $z \in \mathbb{R}^{d_H}$. The weight parameters $\theta \in \mathbb{R}^P$ correspond to stacking all weight matrices W_h and biases b_h in a single vector (in particular, we have here $P = \sum_{h=1}^{H} d_h(d_{h-1} + 1)$). In the sequel, we set $P_h = \sum_{j=h}^{H} d_j(d_{j-1} + 1)$ and write $\theta_{h:H} \in \mathbb{R}^{P_h}$ for the vector of all parameters involved from layer h to the last layer H. We also write $\mathrm{concatenate}(x_1, \dots, x_r)$ to denote the vector obtained by concatenating any r vectors $x_1, \dots, x_r$. In particular, we have $\theta_{h:H} = \mathrm{concatenate}(W_h, b_h, \theta_{h+1:H})$.

Note that the decomposition above took x as input, not θ. We now explain how to construct the $g_{i,j}$ in (2). For each $i = 1, \dots, N$, the function $\theta \in \mathbb{R}^P \mapsto f(x_i, \theta)$ can be decomposed as

$$f(x_i, \theta) = \mathrm{softmax} \circ g_{i,2H-1} \circ \ldots \circ g_{i,2} \circ g_{i,1}(\theta) \ , \tag{6}$$

where, roughly speaking, the $g_{i,2h-1}$ apply the affine mapping A_h to the output $z_{h-1} \in \mathbb{R}^{d_{h-1}}$ of layer $h-1$ and pass forward all parameters $\theta_{h+1:H} \in \mathbb{R}^{P_{h+1}}$ to be used in the next layers, while the $g_{i,2h}$ apply the ReLU function to the first d_h coordinates. More formally, $g_{i,1} : \mathbb{R}^P \to \mathbb{R}^{d_1 + P_2}$ is given by

$$g_{i,1}(\theta) = \mathrm{concatenate}(W_1 x_i + b_1, \theta_{2:H}) \ ,$$

$g_{i,2} : \mathbb{R}^{d_1 + P_2} \to \mathbb{R}^{d_1 + P_2}$ maps any $(z_1, \theta_{2:H}) \in \mathbb{R}^{d_1} \times \mathbb{R}^{P_2}$ to

$$g_{i,2}(z_1, \theta_{2:H}) = \mathrm{concatenate}\big(\sigma(z_1), \theta_{2:H}\big)$$

and, for each layer $h = 2, \dots, H$, the functions $g_{i,2h-1} : \mathbb{R}^{d_{h-1} + P_h} \to \mathbb{R}^{d_h + P_{h+1}}$ and $g_{i,2h} : \mathbb{R}^{d_h + P_{h+1}} \to \mathbb{R}^{d_h + P_{h+1}}$ are given by

$$g_{i,2h-1}(z_{h-1}, \theta_{h:H}) = \mathrm{concatenate}(W_h z_{h-1} + b_h, \theta_{h+1:H})$$

and

$$g_{i,2h}(z_h, \theta_{h+1:H}) = \mathrm{concatenate}\big(\sigma(z_h), \theta_{h+1:H}\big) \qquad (\text{for } h < H).$$

Consider now any loss function $\ell : \Delta(K) \times \{1, \dots, K\} \to \mathbb{R}_+$ to compare any probability vector $q \in \Delta(K)$ of size K with any true label $y \in \{1, \dots, K\}$. For instance, ℓ can be the cross-entropy loss function, which in our case is given by

$$\ell(q, y) = -\log q(y) \ .$$

Finally, using (6), the functions $l_i : \mathbb{R}^P \to \mathbb{R}$ appearing in (1)-(2) can be decomposed as

$$l_i(\theta) = \ell\big(f(x_i, \theta), y_i\big) = \big(q \in \Delta(K) \mapsto \ell(q, y_i)\big) \circ \mathrm{softmax} \circ g_{i,2H-1} \circ \ldots \circ g_{i,2} \circ g_{i,1}(\theta) \ .$$

The last decomposition satisfies (2) with $L = 2H + 1$. When ℓ is the cross-entropy loss function, all L functions involved in this decomposition are either elementary log-exp or consist in applying ReLU to some coordinates of their input, and they are all locally Lipschitz, as required in Definition 1.

B First experiment in 64 bits precision

We reproduce the same bifurcation experiment as in Section 3 under 64 bits arithmetic precision. The results are represented in Figure 6 which is to be compared with its 32 bits counterpart in Figure 1. As mentioned in the main text, the bifurcation does not occur anymore. Indeed the magnitude of the smallest activation before application of ReLU is of the same order, but this time it is well above machine precision which is around 10^{-16}. When depicting the same neighborhood as in Figure 1, the effect of numerical error completely disappears, the bifurcation zone being reduced to a segment in the picture, which is coherent with Theorems 1 and 2.

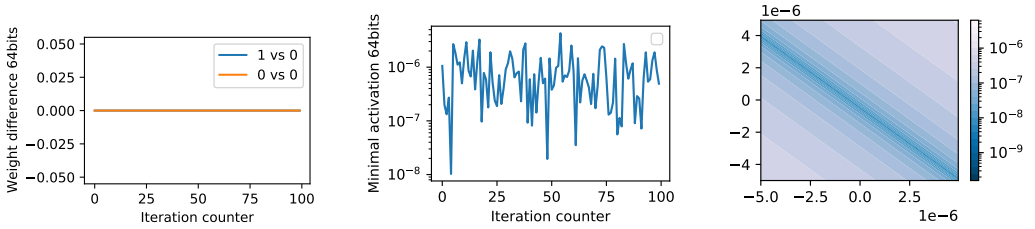

Figure 6: Same experiment as Figure 1 in 64 bits precision. Left: Difference between network parameters (L^1 norm) 100 iterations within an epoch. "0 vs 0" indicates $\|\theta_{k,0} - \tilde{\theta}_{k,0}\|_1$ where $\tilde{\theta}_{k,0}$ is a second evaluation of the same quantity (sanity check), while and "0 vs 1" indicates $\|\theta_{k,0} - \theta_{k,1}\|_1$. Center: minimal absolute activation of the hidden layers within the k-th mini-batch, before application of ReLU. At iteration 65, the jump on the left coincides with the drop in the center, corresponding to an evaluation of $\mathrm{ReLU}'(0)$ exactly. Right: illustration of the bifurcation zone at iteration $k = 65$ in a randomly generated network weight parameter plane (iteration 65 in the center). The quantity represented is the absolute value of the neuron of the first hidden layer which is found to be exactly zero within the mini-batch before application of ReLU (exact zeros are represented in white).

C Volume estimations by Monte Carlo sampling

present the empiric formulas used

D Complements on experiments

D.1 Benchmark datasets and architectures

Overview of the datasets used in this work. These are image classification benchmarks, the corresponding references are respectively [20, 19, 21].

Dataset	Dimensionality	Training set	Test set
MNIST	28×28 (grayscale)	60K	10K
CIFAR10	32×32 (color)	60K	10K
SVHN	32×32 (color)	600K	26K

Overview of the neural network architectures used in this work. The corresponding references are respectively [28, 27, 15].

Name	Type	Layers	Loss function
Fully connected	fully connected	4	Cross-entropy
VGG11	convolutional	9	Cross-entropy
ResNet18	convolutional	18	Cross-entropy

Fully connected architecture: This architecture corresponds to the one used in [28]. We only trained this network on MNIST dataset, the resulting architecture has an input layer of size 784, three hidden layers of size 2048 and the ouput layer is of size 10.

VGG11 architecture: We used the implementation proposed in the following repository `https://github.com/kuangliu/pytorch-cifar.git` which adapts the VGG11 implementation of the module `torchvision.models` for training on CIFAR10 dataset. The only modification compared to the standard implementation is the fully connected last layers which only consists in a linear 512×10 layer. When adding batch normalization layers, it takes place after each convolutional layer.

ResNet18 architecture: We use PyTorch implementation for this architecture found in the module `torchvision.models`. We only modified the size of the output layer (10 vs 1000), the size of the

kernel in the first convolutional layer (3 vs 7) and replaced batch normalization layers by the identity (when we did not use batch normalization).

D.2 Additional Experiments with MNIST and fully connected networks

We conducted the same experiments as in Section 4.2 with a fully connected 784-2048-2048-2048-10 network on MNIST dataset. The results are represented in Figure 7 which parallels the results in Figure 3 on VGG11 with CIFAR10 dataset. We observe the same qualitative behavior. The only difference is quantitative, the fully connected architecture is less sensitive to the magnitude chosen for $\mathrm{ReLU}'(0)$.

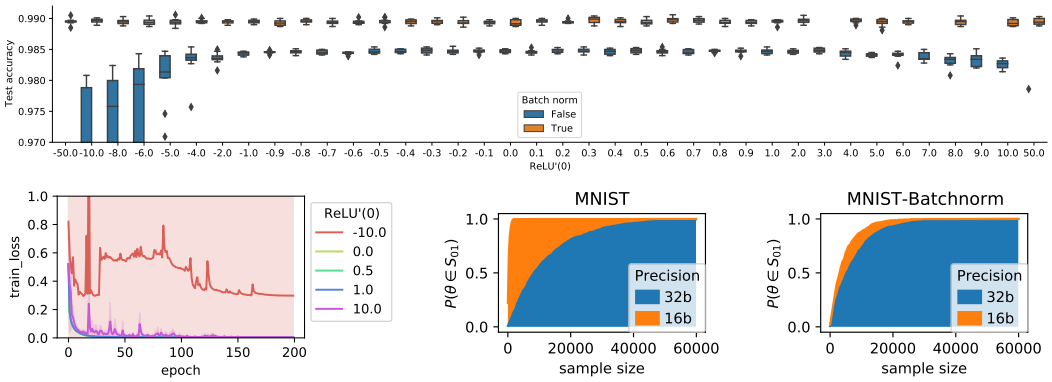

Figure 7: Top: Test error on MNIST with a fully connected 784-2048-2048-2048-10 network. The boxplots and shaded areas represent variation over ten random initializations. We recover the bell shaped curve, but the sensitivity to $\mathrm{ReLU}'(0)$ is less important. Bottom left: corresponding training loss, higher magnitude of $\mathrm{ReLU}'(0)$ induces chaotic oscillation explaining the decrease in test accuracy. Bottom center and left: relative volume estimation of the bifurcation zone with and without batch normalization. Batch normalization increases the size of the bifurcation zone with 32 bits arithmetic and decreases it under 16 bits arithmetic precision.

We investigated further the effect of combining different choices of $\mathrm{ReLU}'(0)$ with dropout [28]. Dropout is another algorithmic way to regularize deep networks and it was natural to wonder if it could have a similar effect as batch normalization. Using the same network, we combined different choices of dropout probability with different choices of $\mathrm{ReLU}'(0)$. The results are represented in Figure 8 and shows that dropout has no conjoint effect.

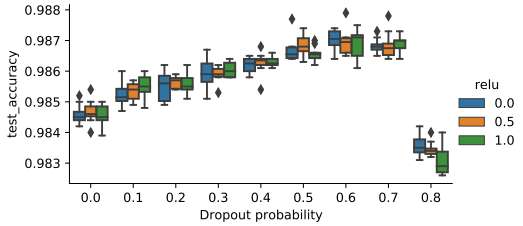

Figure 8: Experiment on combination of the choice of $\mathrm{ReLU}'(0)$ with dropout on MNIST with a fully connected 784-2048-2048-2048-10 network. The boxplots represent 10 random initializations.

D.3 Additional experiments with VGG11

As suggested by the experiment shown in Section 4.2, batch normalization stabilizes the choice of $\mathrm{ReLU}'(0)$, leading to higher test performances. We display in Figure 9 the decrease of training loss on CIFAR 10 and SVHN, for VGG11 with batch normalization. We see that the choice of $\mathrm{ReLU}'(0)$ has no impact and that the chaotic oscilations induced by this choice have completely disapeared.

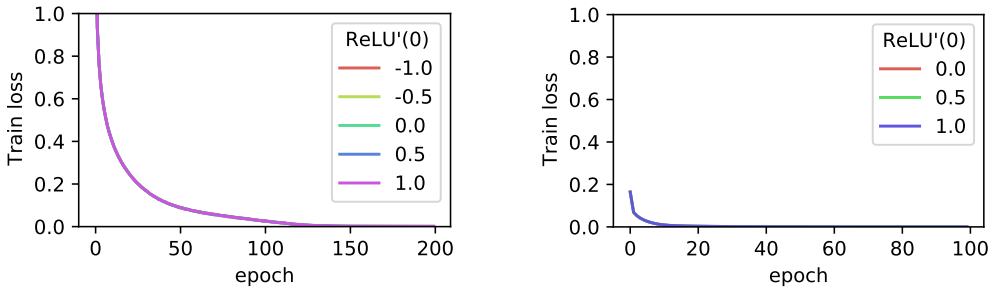

Figure 9: Training loss on CIFAR10 with VGG11 (left) and SVHN with VGG11 (right). The instability induced by the choice of $\mathrm{ReLU}'(0)$ completely disappears.

The training curves corresponding to Figure 4 are shown in Figure 10. They suggest that the Adam optimizer shows much less sensitivity to the choice of $\mathrm{ReLU}'(0)$. This is seen with a relatively efficient buffering effect on the induced oscillatory behavior on training loss decrease.

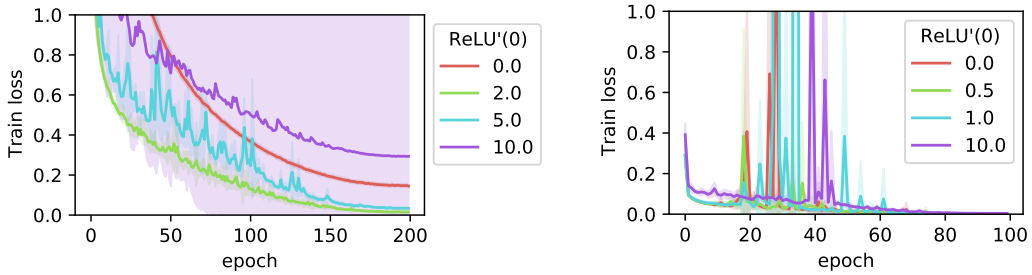

Figure 10: Training losses on CIFAR10 (left) and SVHN (right) on VGG network trained with Adam optimizer. The filled area represent standard deviation over ten random initializations.

D.4 Additional experiments with ResNet18

We performed the same experiments as the ones described in Section 4 using ResNet18 architecture trained on CIFAR 10. The test error, training loss evolution with or without batch normalization are represented in Figure 11. We have similar qualitative observations as with VGG11. We note that ResNet18 architecture is much more sensitive to the choice of $\mathrm{ReLU}'(0)$:

- Test performance degrade very fast, actually, beyond a magnitude of 0.2, we could not manage to train the network without using batch normalization.
- Even when using batch normalization, the choice of $\mathrm{ReLU}'(0)$ has an effect for relatively small variations. This is qualitatively different from what we observed with VGG11 and fully connected architectures.

Similar Monte Carlo relative volume experimentation were carried out for this network architecture, the results are presented in Figure 12. The results are qualitatively similar to what we observed for VGG11 architecture: the bifurcation zone is met very often for 16 bits precision, and the addition of batch normalization increases this frequency in 32 bits precision. Note that we did not observe a significative variation in 16 bits precision.

E Complimentary information total amount of compute and resources used

All the experiments were run on a 2080ti GPU. The code corresponding to the experiments and experiments results are available at `https://github.com/AnonymousReLU/ReLU_prime` Details about each test accuracy experiments are reported on Table 2.

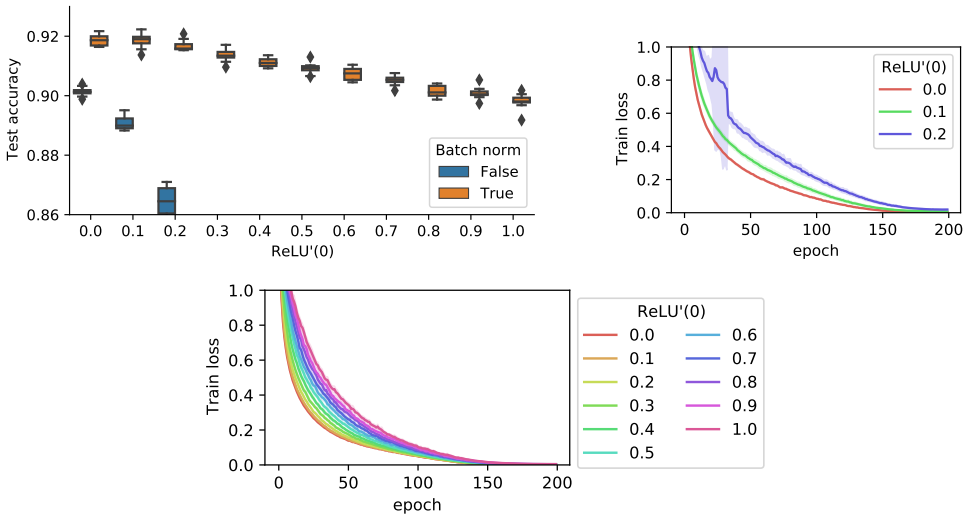

Figure 11: Training experiment on CIFAR10 with Resnet18 and SGD optimizer. Top Left: test accuracy with and without batch normalization. Top right: training loss during training without batch normalization. Bottom: training loss during training with batch normalization.

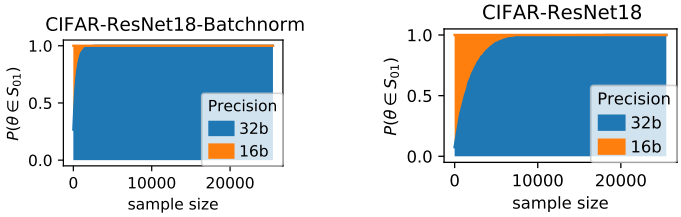

Figure 12: Relative volume Monte Carlo estimtion on CIFAR10 with Resnet18 with and without batch normalization under 16 bits or 32 bits precision.

Dataset	Network	Optimizer	Batch size	Epochs	Time by epoch	Repetitions
CIFAR10	VGG11	SGD	128	200	9 seconds	10 times
CIFAR10	VGG11	Adam	128	200	10 seconds	10 times
CIFAR10	ResNet18	SGD	128	200	13 seconds	10 times
SVHN	VGG11	Adam	128	64	85 seconds	10 times
MNIST	MLP	SGD	128	200	2 seconds	10 times

Table 2: Experimental setup