Increasing separation in units for greater perforamnce an zero initialization

Riera-Molina C.
Universitat de Barcelona
Gran Via de les Corts Catalanes 585, Barcelona Spain, 08005
blauigris@gmail.com

Rey-Torres C.
Universitat de Barcelona
Gran Via de les Corts Catalanes 585, Barcelona Spain, 08005
camilorey@gmail.com

1. Introduction

The success of Deep Learning has been linked to its ability to learn *abstract representations* from input data in a hierarchical fashion [**?**, **?**], using Deep Neural Networks (DNN for short). However, depth is also instrumental in various DNN *issues* like the *vanishing gradient* as presented in [**?**] or [**?**]; the *exploding gradient* as presented in [**?**]; the *dead unit problem* as presented in [**?**],[**?**] or [**?**]; or the *degradation problem* as presented in [**?**].

We claim we can categorize existing methods to address these issues in two families: *architectural modifications* to the DNN and *data manipulation*. Examples of architectural modifications include (1) layer-width increase as done in [**?**, **?**]; (2) additional connections (as done in ResNets [**?**] or DenseNets [**?**]); (3) unit augmentation (as done in *leaky*-`ReLU` [**?**] or `PReLU` [**?**]). Meanwhile, the most commonly used data manipulation technique is *data normalization* on the output of layers as done under *batch normalization* [**?**].

In the case of `ReLU` based DNN, a geometric outlook allows us to conjecture on these issues differently. Under the setting defined by Rey-Torres et al. in [**?**], gradient issues (vanishing and exploding) depend on *dying-items* (e.g. dead units or points). In turn, dying items can be defined by *incorrect* positioning of the hyper-planes from the pre-activation of `ReLU` units (*relative* to the dataset). The reader can review our presentation of our rationale in Section 2 on *Separability*.

Thus, we can correct plane positioning by enforcing *separation constraints* (`Sep-Cons` for short) as detailed in Section 3) titled *Separation Constraints*. Indeed, by targeting specifically the dead unit problem (defining the unit-based separation constraint `Sep-U`); and the dead point problem (using a point-based separation constraint `Sep-P`). Intuitively, `Sep-U` constraints aim to secure unit pre-activation hyperplanes that *cut* throughout the dataset (as detailed in Section 2). Meanwhile, `Sep-P` aims to secure the existence of units with opposing pre-activation values at each dataset point.

We test our proposal in a series of experiments (Section 4) to showcase the effect of applying *Separation constraints*. We find an increase on network depth using the same width than Batch Normalization [**?**], effectively reducing the dependency between depth and width as introduced in [**?**] and [**?**].

2. `ReLU` Based Separation

A classical feed-forward DNN F can be formally approached as a multi-valued real function $F(\mathbf{x})$, that is created by composing a collection of vector *layer* functions $\ell : \mathbb{R}^{n_k} \to \mathbb{R}^{n_{k+1}}$. The hidden layers are defined as the sum of collection of scalar functions dubbed as the *units*:

$$\ell_k(\mathbf{x}) = \sum_{j=1}^{n_{k+1}} u_i^k(\mathbf{x})\hat{\mathbf{e}}_i \tag{1}$$

that depend on a weight vector $\mathbf{w}_j^k \in \mathbb{R}^{n_k}$ and a bias parameter $b_j^k \in \mathbb{R}$ affinely, featuring a truncation on negative values:

$$u_j^k(\mathbf{x}) = \max\{0, \mathbf{w}_j^k \cdot \mathbf{x} + b_j^k\} \tag{2}$$

it is trivial to see that each unit defines a partition of the space $\mathbb{R}^{n_k}$ in two sets[1]: the *upper* part of unit u_j^k and the

[1] Compare to the *acceptance* and *denial* zones in Rey-Torres et al. [**?**]

Family	Method	Examples	Intended issues
Architectural	Width increase	`Wide-ResNet` [**?**], `Inception` [**?**]	Vanishing Gradient Depth
	Additional connections	`ResNet` [**?**], `DenseNet`[**?**]	Vanishing Gradient Increase Depth
	Unit augmentation	*leaky*-`ReLU` [**?**], `PReLU` [**?**], `C-ReLU` [**?**]	Vanishing Gradient Dead Units
Data manipulation	Modify layer output	*Batch Normalization* [**?**]	Vanishing/Exploding Gradient

Table 1: Summary of techniques commonly used in deep learning to overcome issues like enhanced depth, vanishing/exploding gradient and dead neurons.

lower part of u_j^k:

$$\begin{aligned} upper(u_j^k) &= \{\mathbf{x} : \mathbf{w}_j^k \cdot \mathbf{x} + b_j^k > 0\} \\ lower(u_j^k) &= \{\mathbf{x} : \mathbf{w}_j^k \cdot \mathbf{x} + b_j^k \leq 0\} \end{aligned} \tag{3}$$

We define $upper(u_j^k)$ as the positive half-space of the plane

$$\Pi(\mathbf{w}_j^k, b_j^k) = \{\mathbf{x} : \mathbf{w}_j^k \cdot \mathbf{x} + b_j^k = 0\} \tag{4}$$

while $lower(u_j^k)$ corresponds to the closure of its negative half-space. Based on Rey-Torres et al. [**?**], we define the *affine* component of a layer function ℓ_k as the intersection of the upper parts of its units, and its zero set correspondingly:

$$A(\ell_k) = \bigcap_{j=1}^{n_{k+1}} upper(u_j^k), \quad Z(\ell_k) = \bigcap_{j=1}^{n_{k+1}} lower(u_j^k) \tag{5}$$

both $A(\ell_k)$ and $Z(\ell_k)$ are convex open polytopes in $\mathbb{R}^{n_k}$ not necessarily bounded [**?**, **?**], so that their complement is a closed non-convex set: the *interzone* of ℓ_k, that corresponds to the region of $\mathbb{R}^{n_k}$ that is neither forwarded affinely nor is mapped to zero [2]:

$$I(\ell_k) = \mathbb{R}^{n_k} \setminus (A(\ell_k) \cup Z(\ell_k)) \tag{6}$$

an important fact to keep in mind with regards to `ReLU` based DNN is that the output of layers features exclusively vectors with non-negative components. In this sense, ℓ_k maps $\mathbb{R}^{n_k}$ into the first hyperoctant of $\mathbb{R}^{n_{k+1}}$. Thus, ℓ_k defines a mapping into $\mathbb{R}_+^{n_{k+1}}$ such that:

- it maps the affine component into the *interior* of $\mathbb{R}_+^{n_{k+1}}$:
$$\ell_k[A(\ell_k)] = int(\mathbb{R}_+^{n_{k+1}})$$
- it maps the zero-set exactly to the origin of $\mathbb{R}^{n_{k+1}}$:
$$\ell_k[Z(\ell_k)] = \{\mathbf{0}\}$$
- it maps the interzone to the boundary of $\mathbb{R}_+^{n_{k+1}}$:
$$\ell_k[I(\ell_k)] = \partial\mathbb{R}_+^{n_{k+1}}$$

In addition, given a set $X \subset \mathbb{R}^n$ we can use these sets to define both *dead* units and points, as follows.

Remark 2.1 (Dead Items) *In a given `ReLU`-DNN $F : \mathbb{R}^n \to \mathbb{R}^k$, we say that the j-th unit of layer ℓ_k, u_j^k is* dead *with regards to a set $X \subset \mathbb{R}^{n_k}$ if*

$$X \subset lower(u_j^k) \tag{7}$$

on a similar note, we say that a point $\mathbf{x} \in X$ is dead *with regards to layer ℓ_k if*

$$\mathbf{x} \in Z(\ell_k) \tag{8}$$

Particularly, if $X \subset \mathbb{R}^n$, we say that point $\mathbf{x} \in X$ is dead *with regards to F if its abstract representation for each layer is dead with regards to the each layer:*

$$(\forall k | 1 < k \leq D : \ell_{k-1} \circ \ldots \ell_1(\mathbf{x}) \in Z(\ell_k)) \tag{9}$$

Set $Z(\ell_k)$ is paramount also in the *vanishing* gradient problem. As Rey-Torres et al. [**?**] within section 2 show, the loss gradient (with regards to parameters) is zero over $Z(\ell_k)$ (compare to Equations 30, 33, 38 and 43). It follows also from Rey-Torres et al. [**?**] that dead points (With regards to layers and DNN) produce zero-valued parameter gradients.

In this sense, non-zero back-propagation over `ReLU`-DNN is –from the point of view of points– dependent on that they are not dead. Meanwhile, from the point of view of units, non-zero back-propagation over them is only possible when the argument does not lie in their zero-set or worst exactly on the separating plane (notice the discontinuity described in Rey-Torres et al. [**?**]). Taking into account that points are mapped throughout the DNN conforming *abstract representations*, a point may be dead with regards to a layer but die in the next. When whole sets

[2] Compare to the *ambiguity zone* of [**?**]

are mapped through ReLU -DNN, layers may *degrade* them completely if they happen to be a subset of their zero-set.

In order to understand how units define dead points, we introduce the concept of a *separating* unit with regards to an arbitrary set X.

Definition 2.1 (Separating Unit) *Given an arbitrary set $X \subset \mathbb{R}^{n_k}$, we say that the j-th unit on layer k, u_j^k is able to* separate *through X if the following predicate is satisfied:*

$$R_X(u_j^k) \equiv \emptyset \neq upper(u_j^k) \cap X \neq X \tag{10}$$

Since $upper(u_j^k)$ and $lower(u_j^k)$ compose a partition of $\mathbb{R}^{n_k}$ it is equivalent to say that

$$R_X(u_j^k) \equiv \emptyset \neq lower(u_j^k) \cap X \neq X \tag{11}$$

Thus, by construction, a separating unit cannot be dead, and if $R_X(u_j^k)$ is valid, u_j^k cannot degrade set X to a single point.

We can extend this definition of separability to layers as follows:

Proposition 2.2 (Separating Layer) *We say that ℓ_k is able to* separate *set X if the predicate is also satisfied:*

$$R_X(\ell) \equiv \emptyset \neq A(\ell) \cap X \neq X \tag{12}$$

In terms of pre-activations, $R(u_j^k)$ can be restated as follows:

Remark 2.3 *Given a ReLU-DNN $F(\mathbf{x})$ of depth D, let k such that $1 \leq k \leq D$. If u_j^k the j-th unit on ℓ_k, $R_X(u_j^k)$ is equivalent to the following:*

$$\max_{\mathbf{x} \in X}\{\mathbf{w}_j^k \cdot \mathbf{x} + b_j^k\} > 0 \wedge \min_{\mathbf{x} \in X}\{\mathbf{w}_j^k \cdot \mathbf{x} + b_j^k\} \leq 0 \tag{13}$$

To see that both conditions are equivalent a simple geometric argument will suffice. If u_j^k separates through set X (i.e. $R(u_j^k)$ is true), then $upper(u_j^k) \cap X \neq \emptyset$, thus

$$\max_{\mathbf{x} \in X}\{\mathbf{w}_j^k \cdot \mathbf{x} + b_j^k\} > 0 \tag{14}$$

in addition, since $upper(u_j^k) \cap X \neq X$, we must have that $lower(u_j^k) \neq \emptyset$, so that

$$\min_{\mathbf{x} \in X}\{\mathbf{w}_j^k \cdot \mathbf{x} + b_j^k\} \leq 0 \tag{15}$$

conversely, assuming that both inequalities are satisfied, since X is a discrete set, and we define

$$\begin{aligned} \mathbf{x}_+ &= \arg\max_{\mathbf{x} \in X}\{\mathbf{w}_j^k \cdot \mathbf{x} + b_j^k\} \\ \mathbf{x}_- &= \arg\min_{\mathbf{x} \in X}\{\mathbf{w}_j^k \cdot \mathbf{x} + b_j^k\} \end{aligned} \tag{16}$$

we must have that

$$\begin{aligned} \mathbf{x}_+ &\in upper(u_j^k) \\ \mathbf{x}_- &\in lower(u_j^k) \end{aligned} \tag{17}$$

thus, if the inequalities are satisfied then $R(u_j^k)$ is also.

3. Separating constraints

Conceptually, we wish to enforce ReLU *separability* based on the definition provided in equation **??**. However, set theoretic functions are not differentiable [**?**, **?**, **?**].

To bridge that gap, we will make an adaptation inspired on the technique of Support Vector Machines as presented for example in [**?**] or [**?**].

That is, given a fixed choice of weight vector $\mathbf{w} \in \mathbb{R}^n$ and bias $b \in \mathbb{R}$, we define slack variables $\xi^+, \xi^- \geq 0$ for a dataset $\mathcal{T}$ (via set X) as a pair of numbers such that

$$\begin{aligned} \max\{\mathbf{w} \cdot \mathbf{x} + b | \mathbf{x} \in X\} &\geq 1 - \xi^+ \\ \min\{\mathbf{w} \cdot \mathbf{x} + b | \mathbf{x} \in X\} &\leq -1 + \xi^- \end{aligned} \tag{18}$$

This slack variables can be added to the optimization problem **??** as a multiobjective optimization problem using a technique known as scalarization [**?**] (equation 19) as

$$\arg\min_{\theta, \xi^+, \xi^-} \mathcal{L}(\mathcal{T}, \theta) + \lambda g(\xi^+, \xi^-) \tag{19}$$

where λ is a *hyper-parameter* that introduces a trade-off between the constraint fulfillment and the main task loss. Intuitively, we mean to *balance* the effect of the constraints and solving the actual problem. Also, we will require that function g is non-negative, so that in the limit (i.e. when minimized), the original loss will keep the training process going.

The intuition behind the introduction of these constraints presented in Equation 18 is as follows.

We introduce the constraints in pair according to the sign. ξ^+ intends to create at least one pre-activation value greater (or equal) than 1, in order to force the weights and biases to be chosen so that the lower part of the units is non-empty.

In symmetry, enforcing the constraint with ξ^- promotes at least pre-activation of points of X below -1. This intends to ensure that the upper parts of the units are non-empty. Notice how the inclusion of the *max* is what carries this effect of *at least one*, and what it makes different from SVM.

Thus, since neither the upper, nor the lower part of the unit is empty (i.e. the units are neither dead nor affine

in the sense of equations **??** and equation **??** respectively).

Additionally, since introducing those constraints imply the the value of the weights now depend on their output instead of only on the loss functional, this enables to use *zero-initialization*. However, notice that since values of ξ^+ and ξ^- will be equal and cancel each other, we need to introduce a mechanism to break the tie. In our case we chose a convex combination of both that we name g. We use a negligible value (0.51 in practice) so it does not bias the problem.

$$g(\xi^+, \xi^-) = \rho\xi^+ + (1-\rho)\xi^- \tag{20}$$

What remains of this section is to choose the object of the constraints among the different structures of a network: by unit, by layer, and *by point*.

3.1. Unit based separation constraints `Sep-U`

Within a `ReLU` DNN F, choose a unit u_j^k in layer $\ell_k : \mathbb{R}^{n_k} \to \mathbb{R}^{m_k}$. This unit will be defined univocally by its parameters:

$$u_j^k(\mathbf{x}) = u(\mathbf{x}; \mathbf{w}_j^k, b_j^k) \tag{21}$$

in order to enforce separation within this unit, we must substitute parameters in equation 18 and create constraints of the form:

$$\begin{array}{l} \max\{\mathbf{w}_j^k \cdot \mathbf{x} + b_j^k | \mathbf{x} \in X\} \geq 1 - \xi_{jk}^+ \\ \min\{\mathbf{w}_j^k \cdot \mathbf{x} + b_j^k | \mathbf{x} \in X\} \leq -1 + \xi_{jk}^- \end{array} \tag{22}$$

for $k = 1, \ldots, D$ and j the number of units per layer. As a consequence, we will need to introduce balancing parameters rho_{jk} on function g. Individually, we will need to minimize functions of the form

$$g(\xi^+, \xi^-) = \rho_{jk}\xi_{jk}^+ + (1-\rho_{jk})\xi_{jk}^- \tag{23}$$

so that at a global scale we will have to add to the loss functional a term of the form:

$$G = \sum_{k=1}^{D} \sum_{j=1}^{m_k} \rho_{jk}\xi_{jk}^+ + (1-\rho_{jk})\xi_{jk}^- \tag{24}$$

We name this the *separation by unit* formulation `Sep-U`. Notice that this constraint does not prevent from dead points, this is points that belong to the intersection of the lower sets of the units of a layer. That is, if $u_1, u_2, \ldots, u_m$ are the units of a layer $\ell : \mathbb{R}^n \to \mathbb{R}^m$, we say that $\mathbf{x} \in \mathbb{R}^n$ is *dead* with regards to ℓ if

$$\mathbf{x} \in \cap_{j=1})^n lower(u_j) \tag{25}$$

3.2. Point Based Separation `Sep-P`

In order to avoid the presence of dead points (recall Equation 25, we introduce the *Point Based Separation Constraint* `Sep-P` . Moreover, since in practice the entire dataset is not used for training, but *batches* of it [**?**], a small batch size may cause inconsistencies within the training process. Under our formulation, these issues would translate to unit penalization for not fulfilling `Sep-U` on a *per-batch* basis (due to the fact that the points that comply with the constraints may lie outside the batch). We need to make our constraint formulation independent of batch choice by focusing on the values that constraints take on specific points.

That is, we will constraint the slacks $\xi^\pm$ to each point on the batch. That is given $\mathbf{x} \in X$, and $u_1, \ldots, u_m$ unit functions in a layer, we define slack variables $\xi_{xk}^-, \xi_{xk}^+ \geq 0$ in the context of the following constraints:

$$\begin{array}{l} \max\limits_{j=1,\ldots,m} \{\mathbf{w}_k^j \cdot \mathbf{x} + b_k^j\} \geq 1 - \xi_{xk}^+ \\ \min\limits_{j=1,\ldots,m_k} \{\mathbf{w}_k^j \cdot \mathbf{x} + b_k^j\} \leq -1 + \xi_{xk}^- \end{array} \tag{26}$$

so that the associated term to minimize will have the following form 27:

$$G = \sum_{k=1}^{D} \rho_k\xi_{xk}^+ + (1-\rho_k)\xi_{xk}^- \tag{27}$$

Notice that `Sep-P` does not prevent dead units (as `Sep-U`). We can combine both into `Sep-P+U` , to ensure that neither dead points nor dead units are present.

3.3. Layer Based Separation `Sep-L`

As previous network architectures like `ResNet` and `DenseNet` have shown us, it is desirable sometimes to *skip* connections in between layers, for *accuracy* can improve dependent on depth [**?**, **?**]. However, this growth in accuracy is not unbounded and past some point it degrades [**?**].

Thus, it is important to provide a mechanism to forward the output of intermediate layers to the top of the network to which our constraint formulation can adapt and coalesce with existing techniques. Additionally, the number of constraints that we need to place, especially in `Sep-P+U` , is large and we would like to find a way to reduce the computational complexity.

We suggest to relax the `Sep-U` formulation for that aim and provide the reader with the `Sep-L` formulation. The intuition here is to require that at least *one* of the

pre-activations of the entire batch is greater than 1 and another is less than -1, per layer.

With this change we allow the presence of affine units, but ensure that there is at least one pre-activation greater (or lesser) than 1 (-1). That is, given a layer ℓ_k we define the *layer* margins ξ_k^- and ξ_k^+, and rewrite equation 18 as follows:

$$\begin{aligned} \max_{\mathbf{x}\in X} \max_{j=1,\dots,m_k} \{\mathbf{w}_k^j \cdot \mathbf{x} + b_k^j\} \geq 1 - \xi_k^+ \\ \min_{\mathbf{x}\in X} \min_{j=1,\dots,m_k} \{\mathbf{w}_k^j \cdot \mathbf{x} + b_k^j\} \leq -1 + \xi_k^- \end{aligned} \tag{28}$$

We will need to introduce only D constraints (one for each layer). In addition, the term to minimize alongside with the loss functional is as follows:

$$G = \sum_{k=1}^{D} \lambda_k (\rho_k \xi_k^+ + (1 - \rho_k)\xi_k^-) \tag{29}$$

we call this the `Sep-L` formulation. Notice that we have introduced additional hyperparameters λ_k to focus on the role specific layers.

4. Experimentation

Using back-propagation in `ReLU` networks can only propagate through the positive half-space

1. Force units to separate at least two points.

2. Force points to have at least one positive and negative pre-activation. .

Hypothesis at hand:

In the case of `ReLU` based DNN, we claim that these issues have a geometric undertow, that corrupts back-propagation as a training strategy. To be more precise, we contend that gradient issues (vanishing and exploding) alongside with death problems (unit and point) are intertwined with the position of the hyper-planes defined by the preactivation of `ReLU` units, that hinders error minimization.

1. correctly positioning units to make them separation units following the definitions on subsection **??**.

2. ensuring that each point in the data-set is separated contrarily by at least two units. That is, it lies on the affine component of one, and on the zero-set of another.

4.1. Performance assessment

In terms of overall performance, we chose to benchmark the separating constraints against one another, and to the very popular *batch normalization* as presented in [**?**] using the implementation from `Keras`.

The reader can find a summary of overall results in Table 2. The highest performance was obtained by the `Sep-L` with an accuracy of 100% both in validation and training, and attaining a loss value of 0 in training. While the separation constraints display superior accuracy in comparison to classical `ReLU` and batchnorm (first two rows of Table 2), it is important to notice that the accuracy drop between training and validation is also significantly lower percentage-wise. While classical `ReLU` and `ReLU + BN` experience a $20 + \%$ accuracy drop between training and validation, the `Sep-Cons` family experiences drops from less than 14%, with `Sep-P+U` experiencing only a $5,59\%$ accuracy drop.

However, Loss value variation (recorded at maximal accuracy performance) is significantly higher using for the `Sep-Cons` family ($72 + \%$ for `Sep-U` and `Sep-P` and 100% for `Sep-L`), while for `ReLU` and `ReLU + BN` loss variation remains below 5%. These measurements were taken for the `MOONS` dataset using a fixed thin network of depth $D = 50$ and a width $W = 4$.

	Accuracy		Loss	
	Train	Val.	Train	Val.
`ReLU`	0.5176	0.4	0.6925	0.6938
`ReLU + BN`	0.8117	0.6	0.6331	0.6636
`Sep-L`	1.0000	1.0	0.0000	0.0211
`Sep-P`	0.9294	0.8000	0.1765	0.6476
`Sep-U`	0.9058	0.8000	0.4161	1.5228
`Sep-P+U`	0.9882	0.9333	0.6988	1.0810

Table 2: Maximal performance experiment using the `MOONS` dataset. From left to right, accuracy and loss (for *train* and *validation* sets) for `ReLU`, `ReLU + BN`, and `Sep-Cons` in all its variants.

When varying depth and width we can see that the `Sep-Cons` Family works favorably in deeper thin networks. Notice how `ReLU`, Figure 1a, fails with networks deeper than 30 layers. In other hand, `ReLU + BN`, Figure 1b, manages to work until 70 layers deep. `Sep-P+U`, Figure 1c, works significantly better than both, up to 120 layers. Notice how all the methods suffer from depth-dependent accuracy degradation, but is compensated (partially) using width increase. This is consistent with [**?**]

and [**?**].

`Sep-P+U` however is able to delay the apparition of the issue. Specially when using *skinny* DNN (W in between 2 to 5) where `ReLU + BN` fails. We observe that `Sep-U`, Figure 1d, allows for deeper networks yet accuracy drops inversely to width, a phenomenon we blaim on the inability of the `Sep-U` constraint to address the *dead points*.

In addition, `Sep-P`, (Figure 1e), seems to perform well up to 50 layers, but it breaks down afterwards. Finally, `Sep-L` (Figure 1f) seems to suffer most with larger widths (after 25), but can also mantain acceptable performance levels well into 70 layers with a medium width.

4.2. On the geometry of the separation constraints

4.3. Dynamic Behavior of the constraints

5. Conclusions

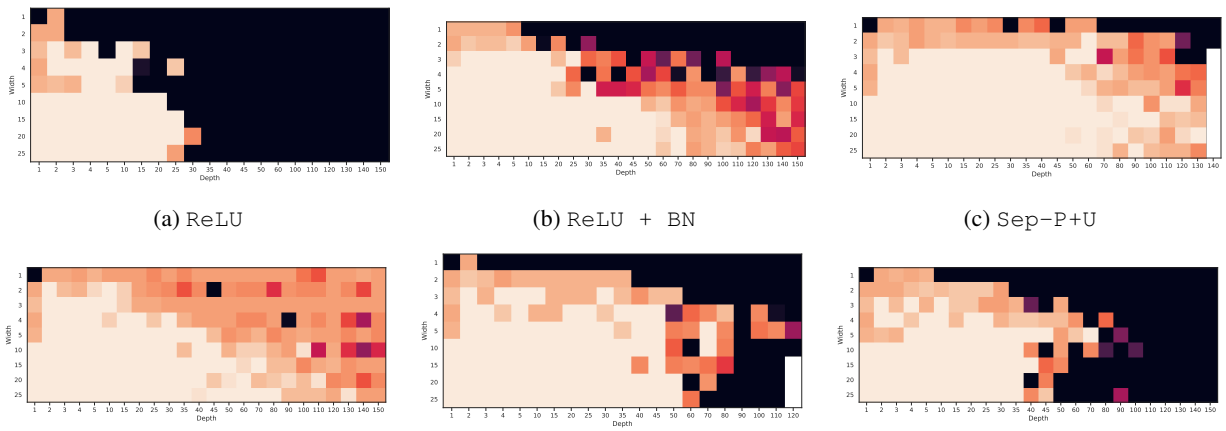
We claim that there is a difference between non-zero and *useful* activations. As presented in Figure 3, `ReLU + BN` induces a solution that maps the dataset non-trivially throughout the network. However, its performance on the `MOONS` dataset is inferior to our proposal (63% of accuracy as presented in Table 2 against any constraint based test over 90%). Moreover, comparing the output graph of `ReLU + BN` (Figure 3h) with our findings (Figures 4h, 5h. 7h and 6h) the reader can observe that the separation reached separates components of the `MOONS` dataset in intuitive manners.

However, further inquiry into the interaction between constraint loss and main loss beyond parameter λ is needed, alongside with experiments regarding other loss functionals besides the cross-entropy loss. In addition, the use of the separation constraints in other types of layers such as LSTM [**?**] or transformers [**?**][**?**] is yet to be performed, in order to explore other types of tasks like regression, unsupervised learning [**?**], graph [**?**] or generation [**?**, **?**]. Morever, more challenging datasets are definitely in need of revision.

While the separation constraints prevent the vanishing gradient, the *exploding gradient* remains at large. Notice that constraint introduction places *lower bounds* on the magnitude of the pre-activation values, but does not place upper bounds. This could be solved introducing additional constraints or limiting the norms of weight vectors.

We explore the use of separation constraint (`Sep-P+U` in this case) to enable zero initialization with good results, (recall subsection **??**). In order to make it work, we had to introduce to break symmetry in the weights, in the form of Annealed Dropout [**?**]. Since our objective in this matter (along with the entire paper) was simplicity, the use of Dropout partially defeats this purpose by moving the same stochasticity that we were removing from intialization into training. Nevertheless, our *Ansatz* is that an initialization that is based on the data (which is the result of using the constraint plus Dropout) must be superior to any random initialization which is only based in features like gradient magnitude or inputs and outputs. Further research in this matter is hereby required.

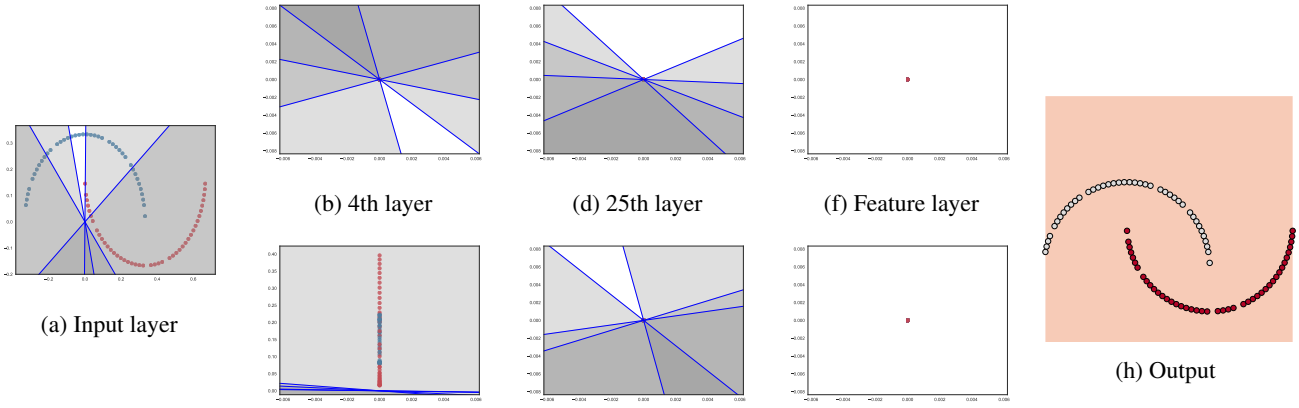
We hope that the geometric framework developed in this work will help us to delve deeper into the intricacies of deep learning so even further simplification is achieved.



(a) `ReLU` (b) `ReLU + BN` (c) `Sep-P+U`

(d) `Sep-U` (e) `Sep-P` (f) `Sep-L`

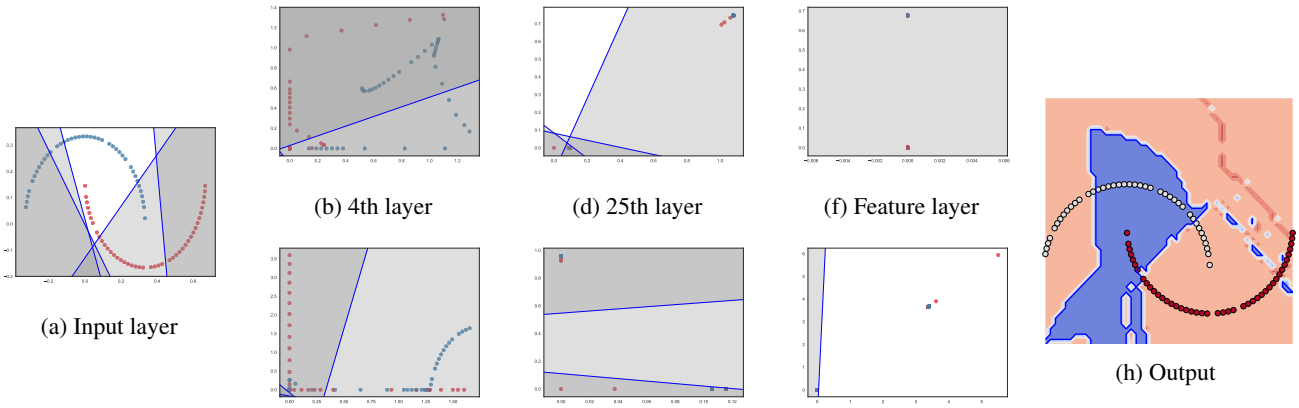
Figure 1: Depth vs Width accuracy plot for a rectangular network of depth W and depth D varying both parameters: $2 \leq W \leq 25$ and $2 \leq D \leq 150$, over the `MOONS` dataset. This network were trained using an Adam scheme, and learning rate of $\gamma = 0.01$. The color map varies according to accuracy attained of each of the combinations of W and D (black is trivial accuracy and light cream is 1.0)



(a) Input layer (b) 4th layer (d) 25th layer (f) Feature layer

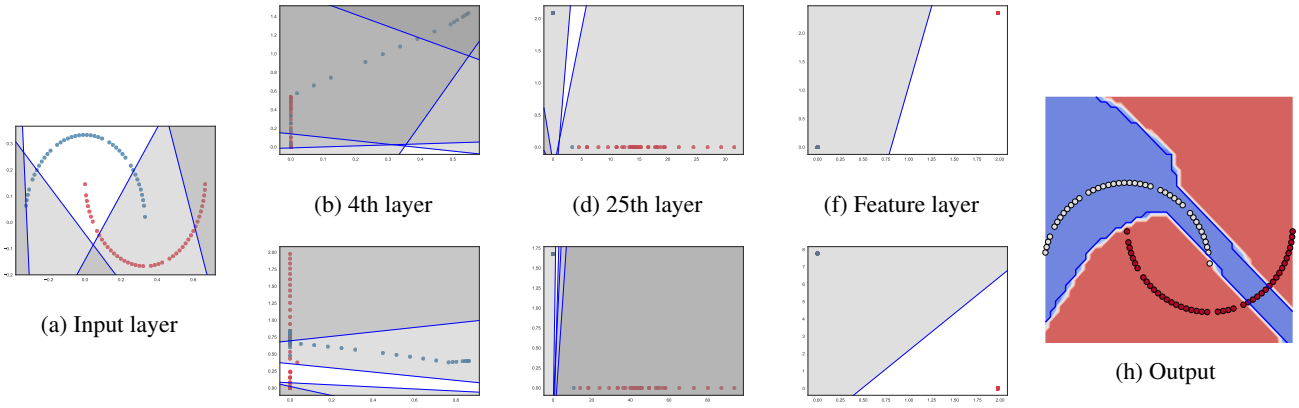
(c) 4th layer (e) 25th layer (g) Feature layer (h) Output

Figure 2: Data transformed across a 50x4 `ReLU` classification network. Notice how the the dataset is progressively mapped to zero as it traverses the network. This renders the output layer unable to solve the problem.



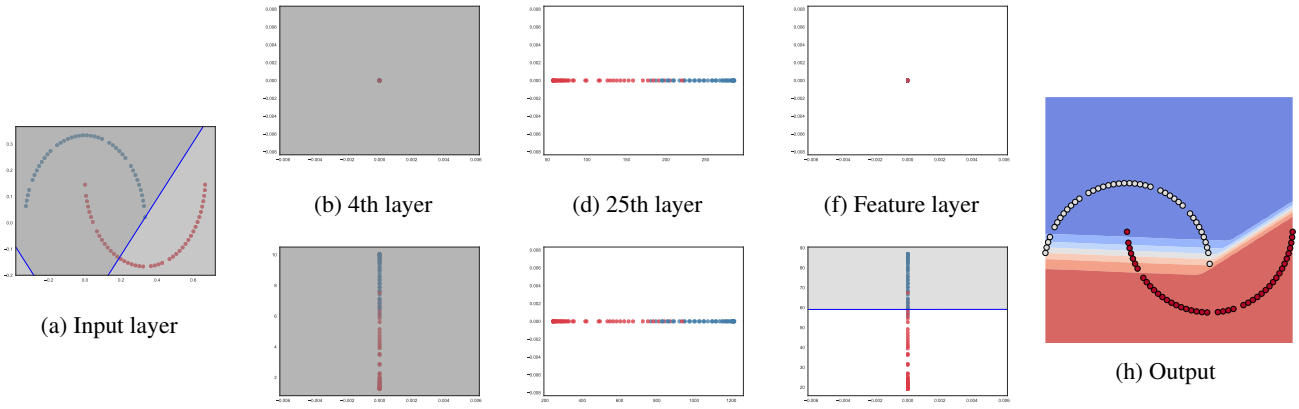
(a) Input layer (b) 4th layer (c) 4th layer (d) 25th layer (e) 25th layer (f) Feature layer (g) Feature layer (h) Output

Figure 3: Data transformed across a 50x4 `ReLU + BN` network. The dataset is collapsed in few points at the feature layer. As the gradient cannot be backpropagated across the truncation after the affine transform of γ and β despite the standarization, failing in the same manner than `ReLU` only that with non-zero activations. This results in *topological mixing* of the datasets. Therefore, the representational capability of the network is hindered to such extent that the resulting output, although non-trivial, is totally arbitrary.



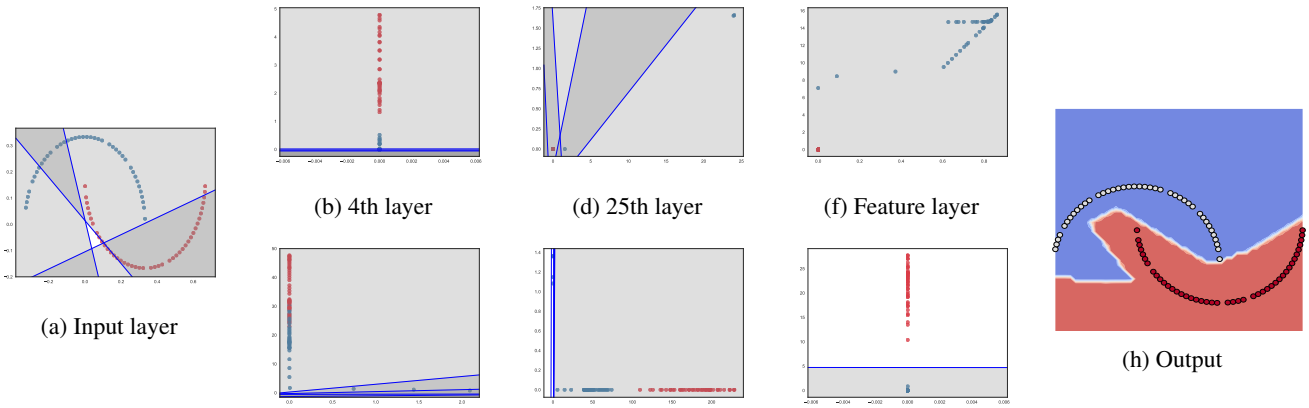
(a) Input layer (b) 4th layer (c) 4th layer (d) 25th layer (e) 25th layer (f) Feature layer (g) Feature layer (h) Output

Figure 4: Data transformed across a 50x4 `Sep-U` network. Notice how dead and affine units have been reduced. Despite collapsing the dataset in two points at the feature layer, the classification performed in the output layer is approximately correct. We conjecture that this is due the dead point addressed with `Sep-P` .



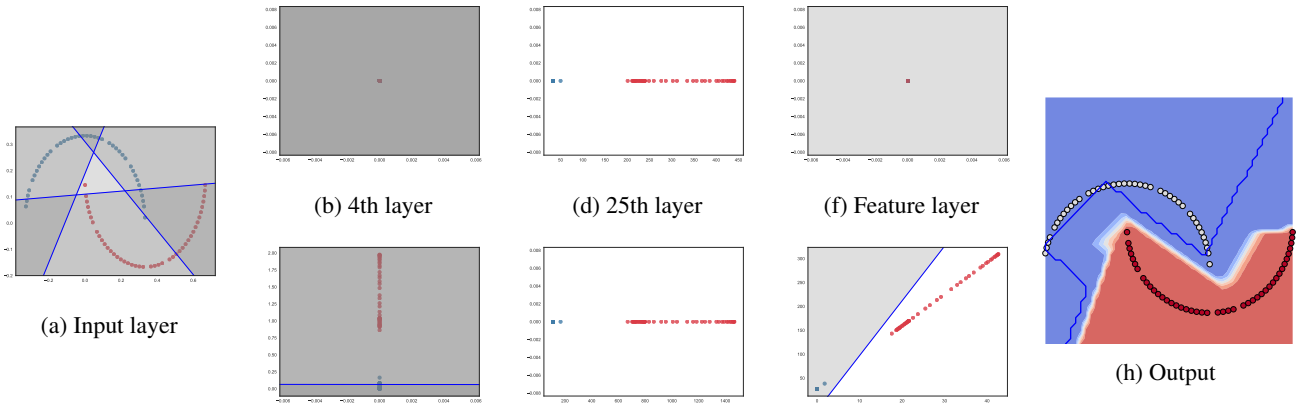
(a) Input layer (b) 4th layer (c) 4th layer (d) 25th layer (e) 25th layer (f) Feature layer (g) Feature layer (h) Output

Figure 5: Data transformed across a 50x4 `Sep-P` network. The network displays a richer internal representation without collapsing the dataset like `Sep-U` or `ReLU + BN`. However, plenty of dead and affine units appear since they are not penalized, causing underfitting.



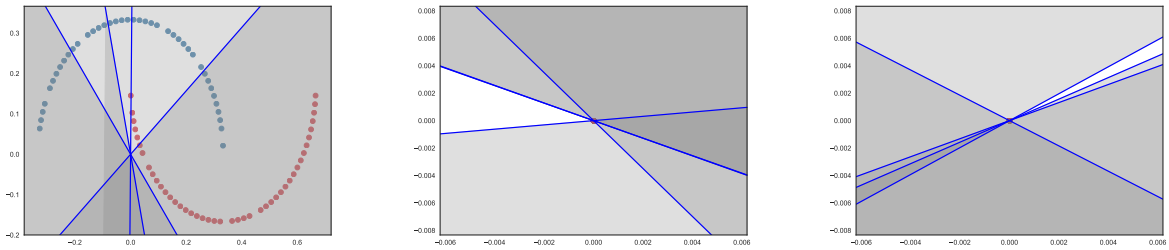
(a) Input layer (b) 4th layer (c) 4th layer (d) 25th layer (e) 25th layer (f) Feature layer (g) Feature layer (h) Output

Figure 6: Data transformed across a 50x4 `Sep-P+U` network. The network displays internal representations without collapsing the dataset like `Sep-P` while retaining classification power like `Sep-U`.

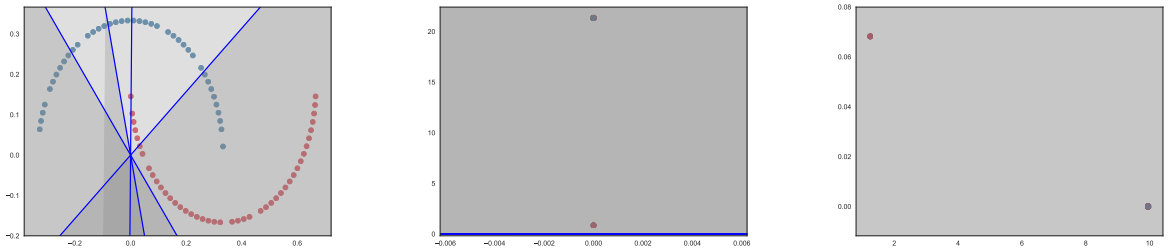


(a) Input layer (b) 4th layer (c) 4th layer (d) 25th layer (e) 25th layer (f) Feature layer (g) Feature layer (h) Output

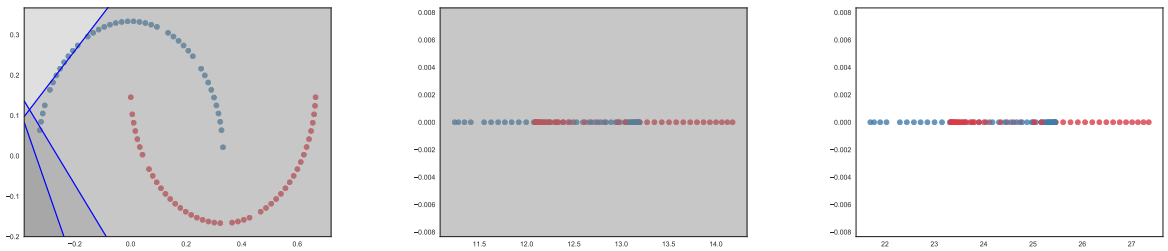
Figure 7: Data transformed across a 50x4 `Sep-L` network. Although being a relaxation of `Sep-P+U` showing an intuitive solution with several affine and dead units. Indeed, they not only do not hinder the network but forward the solution 4th layer to the output.



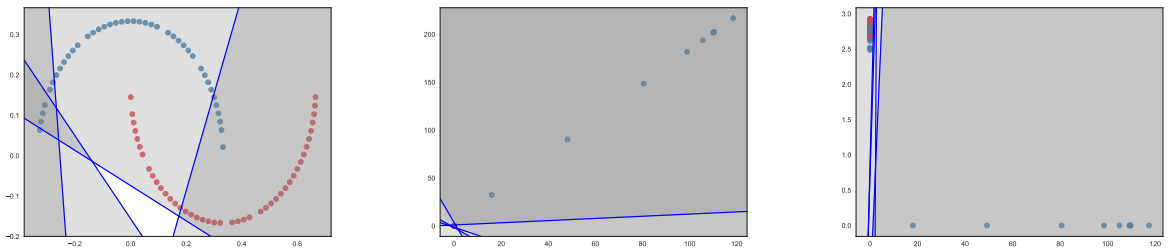
(a) `ReLU` input layer (b) `ReLU` 50th layer (c) `ReLU` 50th layer



(d) `ReLU + BN` input layer (e) `ReLU + BN` 50th layer (f) `ReLU + BN` 50th layer

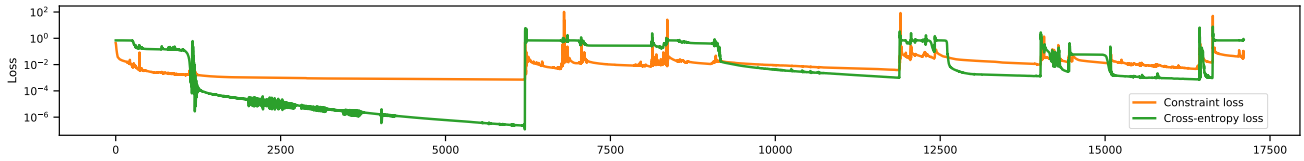


(g) `Sep-L` input layer (h) `Sep-L` 50th layer (i) `Sep-L` 50th layer

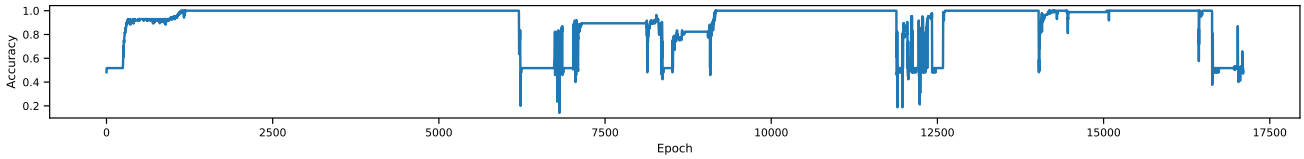


(j) `Sep-P+U` Input (k) `Sep-P+U` 50th layer (l) `Sep-P+U` 50th layer

Figure 8: Data transformed across a 50x4 network with no main loss (cross-entropy) with constraints `Sep-L` and `Sep-P+U`, versus `ReLU` and `ReLU + BN`. Notice how effectively `ReLU` and `ReLU + BN` collapse the dataset into few points whereas `Sep-L` and `Sep-P+U` force the network to learn representations that preserve geometrical structure useful for back-propagation.

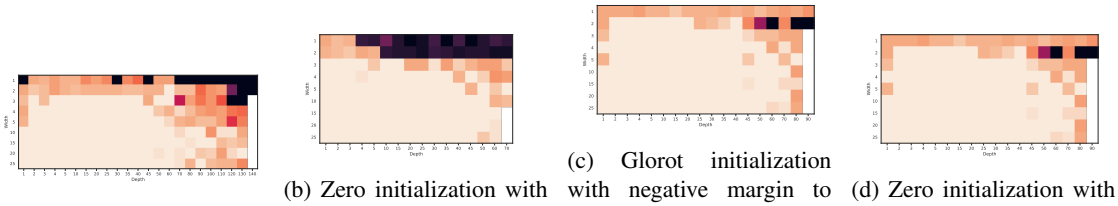


(a) Evolution of cross-entropy and constraint loss during training.



(b) Evolution of accuracy during training.

Figure 9: Evolution of training throughout epochs (cross-entropy, constraint loss, and accuracy). Left-hand axis show the accuracy metric (blue line) against the cross-entropy, and constraint loss in the right axis (orange line), for each epoch of the training phase in the horizontal axis.



(a) Glorot initialization

(b) Zero initialization with $\rho = 0.51$

(c) Glorot initialization with negative margin to zero

(d) Zero initialization with negative margin to zero

Figure 10: Glorot vs Zero initialization in a series of Depth D vs Width W accuracy plots a for rectangular network using a grid ($2 \leq W \leq 25$ and $2 \leq D \leq 150$), trained over the MOONS dataset using an Adam learning rate of 0.01, annealed dropout for 1000 epoch with an initial rate of 0.01 in the. The color scheme depicts accuracy attained for each grid location, from dark to light. (a) classical Glorot, (b) Zero intialization using $\rho = 0.51$, (c) Glorot initialization using $\rho = 0$ or enforcing only non-negative constraints and (d) Zero initialization using $\rho = 0$ or non-negative constraints.