When Neurons Fail

Technical Report

El Mahdi El Mhamdi[*] EPFL

Rachid Guerraoui[†] EPFL

Abstract

Neural networks are considered robust in the sense that the failure of neurons can be compensated by additional learning phases. Nevertheless, in a critical application such as flight control, for which neural networks are now appealing solutions, one cannot afford any additional learning at run-time.

In this paper, we view a neural network as a distributed system of which neurons can fail, and we evaluate its robustness in the absence of any (recovery) learning phase. We give, for the first time, tight bounds on the number of neurons that can fail, without harming the result of a computation or making any assumption on the learning algorithm. To determine our bounds, we leverage the very fact that neuronal activation functions are Lipschitzian.

We distinguish the case of neurons that can fail and stop their activity (crash) from the case of neurons that can fail by transmitting arbitrary information (Byzantine). We show that, in the first case (crash), the bound is on a polynomial value expressed in terms of the number of neurons per layer, the Lipschitz constant of the neuronal activation function, the number of failing neurons, the synaptic weights and the depth of the layer where the crash occurred. In the second case (Byzantine), besides the parameters of the first case, the bound depends also on a polynomial term in the maximum error value of failing neurons. We also show how to extend our results to the case where synapses can fail.

In particular, in the case of multilayer (often called *deep*) neural networks, our results provide a guidance on how to distribute synaptic weights over layers, and how to tune the neuron activation function to increase robustness.

1 Introduction

Since their inception in the 1940s [25], neural networks received an oscillating amount of interest [34, 3]. They went trough two periods of excitement followed by a loss of interest. After the very first period of excitement, from the 1950s [33] to the late 1960s, came the *AI winter*, when Minksy questioned the ability of perceptrons to learn non linearly separable functions such as XOR [27]. Neural networks received a regain of interest in the late 1970s with the back-propagation algorithm [43] that overcame learning issues. Interest vanished again in the 1990s due to the lack of computing power, even-tough important practical achievements were done in the late 1980s and early 1990s in the field of image recognition [21, 20]. Neural networks are now back in the headlines for their outstanding performance [12, 35, 17, 39, 37], in tasks such as function approximation, image and speech recognition, as well as weather prediction [16]. Today, neural networks are being implemented to serve in far more critical applications than those of the last decade. New applications that seek

[*]elmahdi.elmhamdi@epfl.ch

[†]rachid.guerraoui@epfl.ch

the use of neural networks include flight control [6], radars [9] and electric cars [8]. This motivates a better understanding of the extend to which neural networks can be *robust*.

A simple approach to achieve robustness is to consider the entire neural network as a single piece of software [11], replicate this piece on several machines, and use classical state machine replication schemes to enforce the consistency of the replicas [36]. In this context, no neuron is supposed to fail independently: the unit of failure is the entire machine hosting the network. However, forcing an entire network to run on a single machine clearly hampers scalability. One could also consider strict subsets of the neural network as different pieces of software, running on one Turing machine each [7]. In this case, classical replication schemes can still be applied, but one has to face usual distributed computing problems, e.g., to handle the synchronicity of message passing between the subsets of the network [23].

Biological plausibility, together with scalability, call for going one step further and considering each neuron as a *single* physical entity that can fail *independently*, i.e., to go for genuinely distributed neural networks [28]. This approach is considered for example in the Human Brain project [24], trying to emulate the mammalian brain, the connectomics approach, searching for the nervous system topology [22], or various works on hardware-based neural networks [14, 29, 5].

In this paper, we explore this path and view a neural network as a distributed system where neurons can fail independently. We ask what is the maximum number of such failures that can be masked by the neural network, i.e., without having any impact on the overall computation. Addressing this question goes however first through precising it.

It is actually well known that the failure of neurons can be tolerated through additional learning phases [32, 41]. However, stopping a neural network and recovering its failures through a new learning phase is not an option for critical applications. One can also consider specific learning schemes a priori that makes it possible to tolerate failures a posteriori, e.g., shutting down parts of the network while learning, in order to cope with failures at run-time [30, 15, 17, 39]. We also exclude such a possibility to seek a general result. Our question can then be precised as follows: if (a) we do not make any assumption on the learning scheme and (b) we preclude the possibility of adding learning phases after computations have started, what is the maximum number of faulty neurons that can be tolerated by a neural network?

At this point, the question might sound trivial and the answer could be simply *none*. Indeed, how on earth could a neural network tolerate failures if it was not specifically devised with that purpose in mind? More specifically, if the failures of a number of neurons do not impact the overall result, then these neurons could have been be eliminated from the design of that network in the first place. In fact, the reason why the question is nontrivial is *over-provisioning* [19]. Indeed, neural networks are rarely built with the minimal number of neurons to perform a computation. To estimate exactly this minimal number, one needs to know the target function the network should approximate, which by definition is unknown[1]. In fact, it has been experimentally observed that over-provisioning [26, 4, 12, 15] leads to robustness. Yet the relation between the over-provision and the actual number of failures to be tolerated has never been precisely established. In this paper, we establish this relation for the first time[2].

In this paper, we present non-trivial tight bounds on the number of faulty neurons a neural network can support without harming the computation result. We do not require any additional learning

[1]In machine learning, we only know a finite number of the values of the target function: the learning data set

[2] Of course over-provisioning has its own cost: the need for larger learning data sets and a higher risk of over-fitting [42], two topics that are out of the scope of this paper.

phase nor make any assumption on the prior learning algorithm. In the case of multilayer, or so called *deep* networks, we formulate our results in the form of fault-per-layer distribution. The results are obtained using analytic properties of the different mathematical components of a neural network, namely the activation function, the synaptic weights, and the computation model of the network. By relying on the very fact that neuronal activation functions are bounded, and in practice Lipschitzian [38], we set precise bounds on the error propagation over the layers and therefore establish upper bounds on the number of failures a neural network can support.

Our results can directly be used as a basis to set-up constraints on the construction of a neural network. In addition to setting learning constraints using our bounds, designers can use our results as a recipe to tune the coefficient of Lipschitzness as well as the trade-off between robustness and ease of learning. Another application is to gauge when recovery-learning should be launched, for example in a hardware implementation, knowing the degradation rate (space radiation degradation of satellites for example).

For didactic reasons, we present our results in an incremental manner: we consider first a network with a single layer and focus on the crashes of neurons, then we generalize to a multilayer network with arbitrary (Byzantine [18]) failures of neurons, and finally to the arbitrary failures of synapses.

The rest of the paper is organized as follows. In Section 2, we model a neural network as a distributed system. In Section 3 we prove the upper bound on crash failures in the case of single-layer neural network. We use this proof as a starting point for the generalization given in Section 5 on Byzantine failures, first for neurons and then for synapses. We conclude the paper in Section 6 by highlighting some interesting trade-offs. For space limitations, some of the proofs are deferred to the appendices.

2 Model

In this section, we present our model, in which we view a neural network as a distributed system, then we describe the computation model and state the robustness criteria.

2.1 Neural networks as distributed systems

We distinguish two main kinds of components of a neural network:

Processes: *Neurons* are the computing nodes of an neural network, they are unreliable, can either stop computing (crash) or send a value different from their nominal output (Byzantine faults). The failure of any node is independent from the failure of any other node. Neurons communicate via message-passing [23] through communication channels called *synapses*.

Channels: *Synapses* have a similar reliability model as the neurons: either they are correct, they stop transmitting the messages, or they transmit different messages than those they were supposed to transmit (those sent by correct neurons). The failure of a synapse is also independent from that of other synapses and neurons.

One important property of synapses is they are weighed[3]. The weight in a synapse models the importance a neuron gives to the output of the neuron at the other end of the synapse. Faults at synapses can then be modeled as errors in the value of the weight: a crashed synapse is weighed by value 0, and a Byzantine synapse exhibits any other value than the nominal value it is supposed to have.

[3]The weights are determined by the initial learning phase, when building the network.

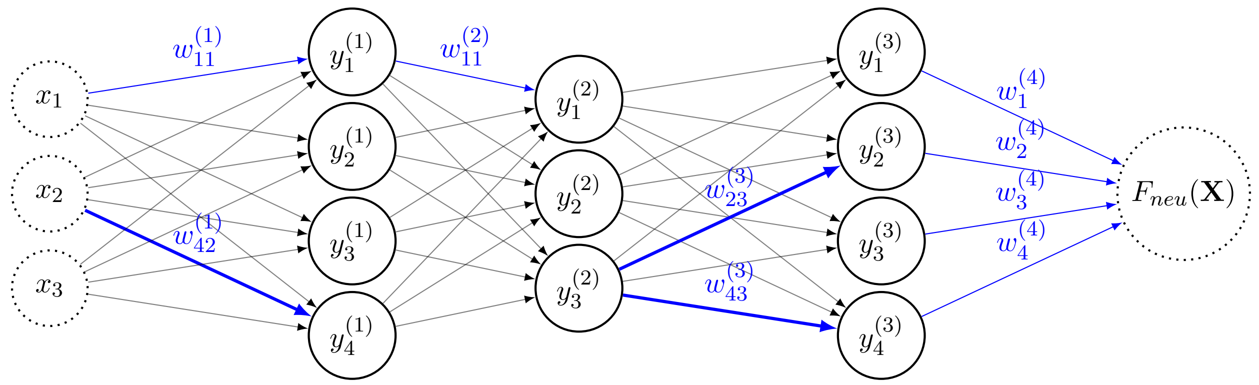

Figure 1: A (feed forward) neural network (solid nodes and edges), with $L = 3$, $N_2 = 3$ and $N_1 = N_3 = 4$. Input and output nodes (dotted) are not considered as parts of the network, but as clients. For readability, only some synaptic weights are represented (blue).

2.2 Computation

In the classical model of a multilayer perceptron [12], neurons are distributed over a series of layers. In our paper, L is the number of layers, each identified with index l and containing N_l neurons. We call $L - l$ the *depth* of the layer l. Each neuron *fires* (broadcasts) a signal to all the neurons of the next layer. Neuron j at the layer l receives the sum given by $s_j^{(l)}$ of Equation 1, where $y_i^{(l-1)}$ and $w_{ji}^{(l)}$ denotes respectively the output value at the i^{th} neuron of the $l-1^{th}$ layer, and the weight of the synapse from that same neuron to the j^{th} one of the next layer l. To define its own output $y_j^{(l)}$, the j^{th} neuron of the l^{th} layer in turn injects the sum given by Equation 1 into a non-linear activation function, often called a *squashing* function, φ, after adding a bias[4].

$$s_j^{(l)} = \sum_{i=1}^{N_{l-1}} w_{ji}^{(l)} y_i^{(l-1)} \text{ with } y_j^{(l)} = \varphi(s_j^{(l)}), \text{ for } l \geq 1 \text{ ; } y_j^{(0)} = x_j \quad \text{and } F_{neu}(\mathbf{X}) = \sum_{i=1}^{N_L} w_i^{(L+1)} y_i^{(L)} \tag{1}$$

Definition 1. *Let $A = C([0,1]^d, [0,1])$ denote the space of continuous functions mapping $[0,1]^d$ to $[0,1]$. $F_{neu}(.)$ as defined by Equation 1 is said to be a neural ϵ-approximation of a target function $F(.) \in A$ if we have: $\forall \boldsymbol{X} \in [0,1]^d$: $\|F(\boldsymbol{X}) - F_{neu}(\boldsymbol{X})\| < \epsilon$*

Universality. The effectiveness of feed-forward neural networks –which are the most common model in the literature – rely on a fundamental theorem [13] that guarantees their universal approximating power with as few as one single layer (L=1). The theorem has been successively proven and generalized [13][5].

[Universality theorem.] (We refer the reader to the proof of [13].) Let d be an integer, let $\varphi(.) : \mathbb{R} \to [0,1]$ be a strictly-increasing continuous function. Then given any function $F \in A$ and $\epsilon > 0$, there exist an integer $N(\epsilon)$, and a set of coefficients $(w_{ji}^{(1)})_{1 \leq j \leq N(\epsilon)}^{1 \leq i \leq d}$ and $(w_i^{(2)})_{1 \leq i \leq N(\epsilon)}$ such that $F_{neu}(.)$ defined in Equation 1 is a neural ϵ-approximation of $F(.)$.

[4]Following the usual notational convenience [12], we omit the bias in the computation model, cf. appendix.

[5]Note that universality for L=1 is harder to obtain than for $L > 1$ (fewer layers to approximate the target function).

Activation function. This function, is behind the non-linearity of neural networks, increasing their approximating power. The universality theorem holds for any non-constant, bounded and monotone-increasing activation function $\varphi(.)$. Yet, two main popular choices for φ in machine learning applications are the logistic function *sigmoid* given by: $sigmoid(x) = \frac{1}{1+e^{-x}}$ and the hyperbolic tangent *tanh*. In this paper, we only impose on φ the conditions of the universality theorem, and consider that φ is *Lipschitzian*, meaning that $K = \sup |\frac{\varphi(x)-\varphi(y)}{x-y}|_{(x,y)\in\mathbb{R}^2}$ exists and is a finite real number. The commonly used functions, *sigmoid* is $\frac{1}{2}$-Lipschitzian, and as any Lipschitzian function it can be tuned to be K-Lipschitzian, where K is any desired positive real value, by for example taking $x \mapsto \varphi(\frac{K}{2}x)$ as the K-tuned activation function. (The detailed derivation of the Lipschizness of φ is given in the appendix.)

2.3 Failures and robustness

Definition 2. *We say that a neuron i in layer l is dead (crashed), when the output value at that neuron $y_i^{(l)}$ is permanently 0. We say that neuron i is failing arbitrarily (Byzantine), when $y_i^{(l)}$ can take any value, however, we assume the error in our model to be bounded.*

Hypothesis 1. *(Bounded error hypothesis.) There exist a uniform upper bound $\lambda \in \mathbb{R}^{*+}$ such that for any input $\boldsymbol{X}$ any erroneous neuron where the output is $y_i(\boldsymbol{X}) + \lambda_i$, with $y_i(\boldsymbol{X})$ is the nominal value dictated by the computation model and λ_i the error, the neuron final value is bounded by λ in absolute value: $\forall i \in [0, N] : |y_i + \lambda_i| \leq \lambda$*

The latter hypothesis is a translation of the physical limits of the synapses: if an erroneous neuron corrupts the value it is supposed to send, this value is limited by the highest amount of electric charge the synapse can transport to the next neuron. Note that **without this hypothesis**, Byzantine neurons cannot be tolerated: a *crazy* neuron can send values as high as needed to fake a working network, while in fact all the other neurons are crashed (dead).

Definition 3. *We say that a neural ϵ-approximation $F_{neu}(.)$ of a target function $F(.)$ realized by N neurons supports N_{fail} faulty neurons, if for any subset of neurons $I_{fail} \subset \{1, \cdots, N\}$ of size N_{fail}, we can modify $F_{neu}(.)$ for the failing neurons consequently (0 for dead neurons, $y_i + \lambda_i$ for Byzantine ones) and still ϵ-approximate $F(.)$ by $F_{neu}(.)$.*

2.4 A corollary to universality

As a consequence of the universality theorem, we can define a minimal number of neurons $N_{min}(\epsilon)$ below which ϵ-accuracy cannot be achieved. By definition of $N_{min}(\epsilon)$, if a neural network is built with $N_{min}(\epsilon)$ neurons, it cannot support any neuron's crash.

Clearly, neural networks are not robust when built with the minimal amount of neurons. We thus evaluate N_{fail} for networks over-provisioned in neurons. We base our inquiry on the obvious fact that if a neural network contains more than $N_{min}(\epsilon)$ neurons, it must be realizing ϵ'-accuracy, with $\epsilon' \leq \epsilon$: Indeed, given $\epsilon' \leq \epsilon$ and $F_{neu}(.)$ a neural approximation realizing ϵ'-approximation of $F(.)$ with $N_{min}(\epsilon')$ neurons, $F_{neu}(.)$ also realizes an ϵ-approximation of $F(.)$.

In the rest of the paper, we set the conditions under which non-minimal networks, containing N neurons, and achieving ϵ'-accuracy ($\epsilon' \leq \epsilon$), can support N_{fail} failures and keep achieving ϵ-accuracy.

3 Failure of neurons in a single-layer neural network

This section deals with the case of the single layer neural network given by the universality theorem. To establish our bound, we translate the fact that the network supports the crash of N_{fail} neurons as given by Definition 3, which we combine with an estimation of the distance between the value of the damaged network and the nominal value of the output (which we recall is close to the target by a distance ϵ'). We end up with an upper bound on N_{fail}.

To prove that the obtained bound is tight, we look at the worst failure case. Intuitively, this corresponds to killing "key neurons": those with highest weights, and looking at an input were those same neurons were instrumental: firing the highest possible value $y_j^{(1)}$, the closest possible to 1.

Theorem 1. *It is possible for a neural network to support up to N_{fail} crashed neurons, as long as $N_{fail} \leq \frac{\epsilon-\epsilon'}{w_m}$, where $w_m = max(\|w_i^{(2)}\|, i \in [1, N])$ is the maximum norm of the weights from the hidden layer to the output node. When no additional constraints are imposed on $F(.)$ or on the network, this bound is tight.*

We can rephrase the result of Theorem 1 in terms of number of neurons. With the work of Barron [1, 2] we know that: $N_{min}(\epsilon) \approx \frac{C}{\epsilon}$ for small enough ϵ where C is a constant depending on the nature of the target function $F(.)$, and the learning data-set that we consider as exogenous to our study. If in addition the over-provisioning was done with the minimal amount to achieve ϵ':
$N_{fail} \approx \frac{1}{w_m}[\frac{C}{N_{min}(\epsilon)} - \frac{C'}{N_{min}(\epsilon')}]$ where C and C' are constants of the same nature as the parameter introduced by Barron.

For convenience, all the estimations of N_{fail} are stated in terms of ϵ and ϵ', since the latter parameters are easier to evaluate.

Proof. **Upper bound.** Let F_{neu} denotes the neural ϵ'-approximation of F, and let F_{fail} denotes the remainder of the neural function F_{neu} after N_{fail} dead neurons at the only hidden layer, let I_{fail} be the set containing the N_{fail} dead neurons, and let $\mathbf{X} \in \mathbb{R}^d$ any input vector. By triangular inequality of norms we have:

$$\|F(X) - F_{fail}(X)\| \leq \|F(\mathbf{X}) - F_{neu}(\mathbf{X})\| + \|F_{neu}(\mathbf{X}) - F_{fail}(\mathbf{X})\| \tag{2}$$

With an equality case when for an input $\mathbf{X}$ we have $F(\mathbf{X}) - F_{neu}(\mathbf{X})$ and $F_{neu}(\mathbf{X}) - F_{fail}(\mathbf{X})$ are positively proportional (**Condition 1**: equality case of the triangular inequality). We know that:

$$\|F(X) - F_{neu}(X)\| < \epsilon' \tag{3}$$

since $F_{neu}(.)$ is an ϵ'-approximation of $F(.)$. From Equation 1 applied to the output neurons in the case of both normal $F_{neu}(.)$ and $F_{fail}(.)$ neural functions, we have:

$$\|F_{neu}(X) - F_{fail}(\mathbf{X})\| = \| \underbrace{\sum_{i=1}^{N} w_i^{(2)} y_i(\mathbf{X})}_{F_{neu} \text{ signal from the hidden layer to the output.}} - \underbrace{\sum_{i=1, i \notin I_{fail}}^{N} w_i^{(2)} y_i(\mathbf{X})\|}_{F_{fail} \text{ signal from the hidden layer to the output.}}$$

$$= \| \sum_{i=1, i \in I_{fail}}^{N} w_i^{(2)} y_i(\mathbf{X})\|$$

Using another triangular inequality on norms we get

$$\|F_{neu}(\mathbf{X}) - F_{fail}(\mathbf{X})\| \leq \sum_{i=1, i\in I_{fail}}^{N} \|w_i^{(2)}\|.\| \underbrace{y_i(\mathbf{X})}_{\text{case of } F_{neu}(.)} - \underbrace{0}_{\text{case of } F_{fail}(.)} \| \tag{4}$$

With an equality case when the dead neurons are those for which the corresponding weights in the previous sum are positive. (**Condition 2**)

And by definition of w_m and the hypothesis on the bounded activation function we have: $(\forall \mathbf{X} \in \mathbb{R}^d)(\forall i \in [1, N]) : \|w_i^{(2)}\| \leq w_m$ and $y_i(\mathbf{X}) \leq 1$. Inequality 4 becomes:

$$\begin{aligned} \|F_{neu}(\mathbf{X}) - F_{fail}(\mathbf{X})\| &\leq \underbrace{\sum_{i=1, i\in I_{fail}}^{N} w_m^{(2)}}_{\text{sum over the set of dead neurons containing } N_{fail} \text{ times a constant term}} \\ &\leq N_{fail}.w_m^{(2)} \end{aligned} \tag{5}$$

With an equality case if the dead neurons are those for which the corresponding weights in the previous sum are those where the maximal value of the weights is achieved. (**Condition 3**)

Merging inequalities 2, 3 and 5 we obtain: $\|F(\mathbf{X}) - F_{fail}(\mathbf{X})\| \leq N_{fail}.w_m^{(2)} + \epsilon'$.

So that if we want to guarantee that F_{fail}, the neural function obtained with N_{fail} neurons dead is still an ϵ-approximation of $F(.)$, i.e, if we want $\|F(\mathbf{X}) - F_{fail}(\mathbf{X})\| \leq \epsilon$ a sufficient condition on the weights, the learned accuracy and the number of dead neurons is: $N_{fail}.w_m^{(2)} \leq \epsilon - \epsilon'$. Thus the upper bound on N_{fail} : $N_{fail} \leq \frac{\epsilon-\epsilon'}{w_m^{(2)}}$.

Tightness. Let $\alpha > 0$ any given positive real. If (**condition 4**) there is an input $\mathbf{X}$ such that for any neuron i in I_{fail}, we have $y_i(\mathbf{X}) > max(1 - \frac{\alpha}{2}, 1 - \frac{\alpha}{2(\epsilon-\epsilon')})$, i.e $y_i(\mathbf{X})$ close enough to 1.

In the worst case, when no constraint prevents condition 4, and conditions 1 to 3 – mentioned in the upper bound proof – to be impossible, we can write the equality case from 2, 3 and 5. $\|F(\mathbf{X}) - F_{fail}(\mathbf{X})\| = \|\sum_{i=1, i\in I_{fail}}^{N} w_i^{(2)} y_i(\mathbf{X})\| + \|F(X) - F_{neu}(X)\|$

moreover, at Inequality 3, for any $\alpha > 0$ we can exhibit an input $\mathbf{X}$ such that $\|F(X) - F_{neu}(X)\| > \epsilon' - \frac{\alpha}{2}$, otherwise ϵ' is not the best accuracy achieved by $F_{neu}(.)$. In case of an input combining this situation with the conditions of the previous paragraph, we obtain:

$\|F(\mathbf{X}) - F_{fail}(\mathbf{X})\| > max(1 - \frac{\alpha}{2}, 1 - \frac{\alpha}{2(\epsilon-\epsilon')})N_{fail}.w_m^{(2)} + \epsilon' - \frac{\alpha}{2}$

Which in case of more crashes than allowed by the upper bound of the theorem, $N_{fail} > \frac{\epsilon-\epsilon'}{w_m^{(2)}}$ leads to: $\|F(\mathbf{X}) - F_{fail}(\mathbf{X})\| > max(1 - \frac{\alpha}{2}, 1 - \frac{\alpha}{2(\epsilon-\epsilon')})(\epsilon-\epsilon') + \epsilon' - \frac{\alpha}{2} = \epsilon - \frac{\alpha}{2} + max(-\frac{\alpha}{2}, -\frac{\alpha}{2(\epsilon-\epsilon')})(\epsilon-\epsilon') > \epsilon - \frac{\alpha}{2} - \frac{\alpha}{2(\epsilon-\epsilon')}(\epsilon-\epsilon')$ which leads to: $\|F(\mathbf{X}) - F_{fail}(\mathbf{X})\| > \epsilon - \alpha$.

While α is any positive real number, we can take the previous inequality to the limit $\alpha \mapsto 0$, and we would have exhibited an input violating ϵ-accuracy for F_{fail}. Proving that the bound is tight. $\square$

4 Generalization to multilayer networks and Byzantine failures

In this section, we give a generalization of the result of Theorem 1. While the previous result shows that we can derive a tight upper bound on how many neurons can crash without losing the ϵ-accuracy, it does not say anything about a situation where some neurons can send values different from what expected. Interestingly, the proof we provided to Theorem 1 suggests a generalization to neurons with arbitrary behavior.

Assume a neuron i is broadcasting, instead of its nominal value y_i, a corrupted value: $y_i + \lambda_i$, Inequality 5 in the previous proof intuitively suggests that if we have a uniform upper bound on λ_i, we can derive a generalized version of Theorem 1 to include the case of Byzantine neurons. Using the bounded error hypothesis 1, we present a lemma that enables not only to generalize Theorem 1 to Byzantine neurons, but to go beyond and generalize the sufficient condition to a multilayer neural network, or so-called *deep neural networks*, as well as to the case of failing synapses.

4.1 Error forward propagation

The idea behind the error forward propagation lemma is that, by induction on the number of layers, when errors occurs at N_{C_l} neurons of layer l, the effect will be transmitted by all the correct neurons in the layers l' between it and the output , only the correct ones $(N_{l'} - N_{C_{l'}})$, as the failing one will be sending their own error, and Hypothesis 1 makes it possible to include the received error in the error of the failing ones. Resulting, in the worst case, in a multiplications, as many times as there are still layer to reach the output, by the Lipschitz constant, $N_{l'} - N_{C_{l'}}$, the maximum weight $w_m^{(l)}$, and finally by the bound λ of Hypothesis 1 and by N_{C_l}. The previous products are summed over the layers. Figure 2 shows error propagation in a simple neural network of 2 layer with 2 neurons each. As a calculation convention for the rest of the paper, we will consider an $(L+1)$-th layer consisting of the output node with $N_{L+1} = 1$ correct neuron and $N_{C_{L+1}} = 0$ failing neurons (though it is not part of the neural network, unlike the $(L+1)$-th synapses which are part of the network).

Lemma 1. *In a neural network containing L layers, if in each layer l, N_{C_l} neurons, among the N_l neurons of that same layer, are affected by errors such that each neuron j of layer l is broadcasting an output $y_j^{(l)} + \lambda_j^l$ to the next layer instead of the nominal $y_j^{(l)}$, then the effect on the output is bounded by:*

$$\|F_{neu}(\boldsymbol{X}) - F_\lambda(\boldsymbol{X})\| \leq \lambda \sum_{l=1}^{L} N_{C_l} K^{L-l} \prod_{l'=l+1}^{L+1} (N_{l'} - N_{C_{l'}}) w_m^{(l')} \tag{6}$$

where $F_{neu}(.)$ is the nominal neural function, $F_\lambda(.)$ the neural function accounting for the errors $\lambda_j^{(l)}$, and $w_m^{(l)} = max(|w_{ji}^{(l)}|, (j,i) \in [1, N_l][1, N_{l-1}])$ is the maximum norm of the weights of the incoming synapses to layer l.

Unless there are additional constraints on the network and on $F(.)$, this upper bound is tight.

Note that the smallest are K and the different $w_m^{(l)}$, the smallest is the propagating error. This should be kept in mind for the upcoming results.

The lemma is provable by induction: thanks to the structure of neural networks, an $L+1$-layer network can be seen as single-layer network (last layer), in which, before applying the activation function, each node is the output of an L-layer network (all the neurons to the left).

A proof of the lemma is given in the appendix.

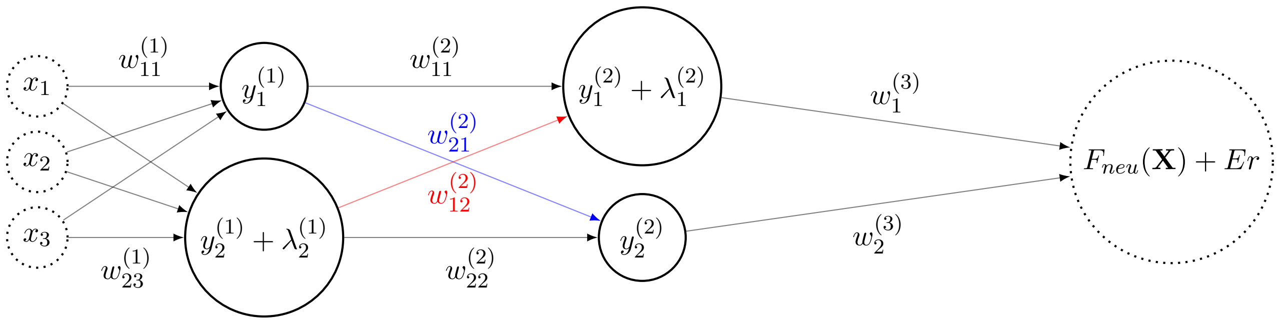

Figure 2: (Error forward propagation) Neuron 2 of layer 1 is broadcasting, in addition to its nominal value $y_2^{(1)}$ an error $\lambda_2^{(1)}$, this error, weighed with at most $w_m^{(2)}$, will affect the nominal values of correct neurons (neuron 1) from layer 2 that will inject it in the activation function and relay its effect. Failing neuron 1 of layer 2 will add another error $\lambda_1^{(2)}$), weighed with at most $w_m^{(3)}$ to the output node. Er is bounded by $\lambda(Kw_m^{(2)} + w_m^{(3)})$.

4.2 Upper bound on neuron failures

While the universal approximation theorem guarantees universal ϵ-approximation with single-layer neural network for any function on a compact subset of a finite-dimensional space of inputs, practitioners of machine learning are using more and more *deep* networks with several layers, latest reported results [40] on image-recognition tasks give numbers as high as few hundred layers.

It is therefore important, from the practical point of view, to give the multi-layer equivalent of the main result formulated in Section 3, Theorem 1. As we establish in the following, the number of crash per layer is not uniform and depends on the depths of the layer in a quasi-geometric fashion with regards to the weight distribution.

With the notations used before, if we have a network of L hidden layers, each layer l containing N_l neurons, and writing $N_{fail} = \sum_{i=1}^{L} N_{C_l}$ as the number of misbehaving neurons with N_{C_l} being the part of N_{fail} happening at layer l. Using Lemma 1, and the same reasoning as in the proof of Theorem 1, it is possible to derive a tight upper bound, not directly on N_{fail} as in monolayer case, but on the N_{C_l} than sums up to N_{fail}.

Theorem 2. *An upper bound for a multilayer neural network to support N_{fail} is given by the inequality: $\lambda \sum_{l=1}^{L} N_{C_l} K^{L-l} \prod_{l'=l+1}^{L+1} (N_{l'} - N_{C_{l'}}) w_m^{(l')} \leq \epsilon - \epsilon'$. Unless there are additional constraints on the network and on $F(.)$, such as symmetries or periodicity, this bound is tight.*

As noted after Lemma 1, small K and small weights reduce the propagating error, which translates in the previous theorem as: the smallest are K and the weights, the easier it is to satisfy the bound while N_{C_l} are getting larger. This sets the basis for the trade-off on tuning K or reducing the weights, as we stated in the introduction an as we will discuss in the conclusion.

Proof. Denote by F_{fail} the output of the network after N_{fail} failures, using Lemma 1 we have:

$\|F_{neu}(\mathbf{X}) - F_{fail}(\mathbf{X})\| \leq \lambda \sum_{l=1}^{L} N_{C_l} K^{L-l} \prod_{l'=l+1}^{L+1} (N_{l'} - N_{C_{l'}}) w_m^{(l')}$. Combining this with inequalities 2 and 3 we obtain: $\|F(\mathbf{X}) - F_{fail}(\mathbf{X})\| \leq \epsilon' + \lambda \sum_{l=1}^{L} N_{C_l} K^{L-l} \prod_{l'=l+1}^{L+1} (N_{l'} - N_{C_{l'}}) w_m^{(l')}$. Therefore, to guarantee that $\|F(\mathbf{X}) - F_{fail}(\mathbf{X})\| \leq \epsilon$, we should have: $\sum_{l=1}^{L} N_{C_l} . w_m^{(L+1)} . \prod_{l' \geq l+1}^{L} (N_{l'} - N_{C_{l'}}) w_m^{(l')} \leq \epsilon - \epsilon'$. Which proves Theorem 2. Tightness follows from the worst case corresponding to the intersection of equality situations in inequalities 2 and 3 and in Lemma 1. □

4.3 The failure of synapses

In a system where the computing nodes are physically more reliable compared to synapses, due to the physical flexibility required from the latter, as the ability to modify themselves by learning, they are much more likely to be damaged than neurons. Though the errors might have radically different nature or causes, the interconnections of synaptic role (coefficients) and neurons role (neuron value) in the working equations of a neural network makes it possible to derive results on synapses from the results on neurons of the previous sections.

We need the following lemma, establishing a link between errors at synapses and consequences on neurons. Again we use the convention that layer $L+1$ corresponds to the output node, in addition to the convention that layer 0 corresponds to input nodes (although not part of the network).

Lemma 2. *In an L-layer neural network, an error of value $\lambda_{ji}^{(l)}$ at the synapse from neuron i of layer $l-1$ to neuron j of layer l is at worst, of the same effect as an error at neuron i of value λ. Under similar absence of constraints as before, this bound is tight.*

Lemma 2 is proven in the appendix.

Theorem 3. *To guarantee robustness, we obtain the following upper bound on $N_{fail} = \sum_{l=1}^{L+1} N_{C_l}$, the number of arbitrarily failing synapses, with N_{C_l} being the number of failing ones linking layer $l-1$ to layer l: $\lambda \sum_{l=1}^{L+1} N_{C_l} K^{L+1-l} w_m^{(l)} \prod_{l'=l+1}^{L+1} (N_{l'} - N_{C_{l'}}) w_m^{(l')} \leq \epsilon - \epsilon'$. Unless there are additional constraints on the network and on $F(.)$, this upper bound is tight.*

Proof. With the convention mentioned above, applying Lemma 2 enables us to see the failure of N_{fail} synapses in an L-layer network as, in the worst case, the failure of N_{fail} neurons in an $L+1$ network, applying Theorem 2, the result follows. Tightness comes from equality cases as usually. □

5 Concluding remarks

Summing-up, when a neural network is built with just the required number of neurons to ϵ-approximate a target function, robustness to the failure of a single neuron is impossible. When built to ϵ'-approximate a target function, with $\epsilon' \leq \epsilon$ (which is the common case), our results set a tight upper bound on the number of neurons and synapses that can fail without losing ϵ-accuracy.

To guarantee robustness to the failure of N_{fail} neurons, the designer can use our results, precisely minimizing the left hand term in Lemma 1, during the learning scheme[6]. Satisfying this minimization guarantees that the neural network has learned the optimal weight distribution. During operations, N_{fail} neurons can fail without losing ϵ accuracy (the reasoning for synapses is similar).

Clearly, over-provisioning to guarantee ϵ'-accuracy has a cost in learning data sets. Designers using our results will have to cope with a dilemma that somehow resembles the celebrated bias/variance dilemma [10] in machine learning: should a neural network generalize well to data it did not see in the learning phase, or should it output very precise approximations for the data seen during learning, and risk over-fitting? In our case, it is a robustness/ease-of-learning dilemma. The key trade-off involves robustness, cost of learning in terms of data sets and size of the network, in two forms:

Trade-off on the Lipschitz-constant of the activation function (K). For example, choosing a low value of $K < 1$ leads to easily satisfying the inequalities of our different theorems even with high numbers of erroneous neurons or synapses. One should recall that K is an estimate of how sharp the discrimination between inputs at the level of a single neuron is (see our appendix on Lipschitzness). Therefore, for a neural network approximating a target function while using a low-K activation function, the learning time and number of necessary neurons should be significantly higher than with a high-K activation function, because the latter is much more discriminating on the input data.

Trade-off on synaptic weights. As for the Lipschitz-constant K, one can note in our inequations that imposing very low weights leaves some room for higher values of the number of erroneous components while still satisfying the bound. Achieving this requires an increase of the number of neurons. Intuitively: you need more units to sum to the desired value, if the coefficients are low.

Finally, if we know additional information about the network topology, for example if the network is convolutional[7] [20], we can use, in addition to the Lipschitz inequality, some of their periodicity properties and end up with less general bounds. Obviously, those bounds will remain specific to the convolutional topology and our bounds lie as a basis for the general case.

[6]We proved our bound independently of any learning scheme. If one particular scheme is taking the left-hand-side of our bounds as a minimization constraint to set a distribution of weights and neurons per layer, it will guarantee a minimal effect of any failure a posteriori. To our knowledge there has been one single attempt to theoretically formulate such a constraint [31], but it only minimizes the effect of the crash of a single neuron. In our case the minimization stands for a distribution of neurons and synapses.

[7]A feed-forward topology where neurons are not connected to all the neurons of the next layer and where the weights have some periodicity. These are widely used in image recognition.

References

[1] Andrew R Barron. Neural net approximation. In *Proceedings of the Seventh Yale Workshop on Adaptive and Learning Systems*, pages 69–72. New Haven, CT: Yale University, 1992.

[2] Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *Information Theory, IEEE Transactions on*, 39(3):930–945, 1993.

[3] D Cervier. Ai: The tumultuous search for artificial intelligence, 1993.

[4] Ching-Tai Chiu, Kishan Mehrotra, Chilukuri K Mohan, and Sanjay Ranka. Robustness of feedforward neural networks. In *Neural Networks, 1993., IEEE International Conference on*, pages 783–788. IEEE, 1993.

[5] Swadesh Choudhary, Steven Sloan, Sam Fok, Alexander Neckar, Eric Trautmann, Peiran Gao, Terry Stewart, Chris Eliasmith, and Kwabena Boahen. Silicon neurons that compute. In *Artificial neural networks and machine learning–ICANN 2012*, pages 121–128. Springer, 2012.

[6] Girish Chowdhary and Eric Johnson. Adaptive neural network flight control using both current and recorded data. In *AIAA Guidance, Navigation, and Control Conference, AIAA-2007-6505. Hilton Head*, 2007.

[7] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

[8] Alberto Diaz Alvarez, Francisco Serradilla Garcia, José Eugenio Naranjo, Jose Javier Anaya, and Felipe Jimenez. Modeling the driving behavior of electric vehicles using smartphones and neural networks. *Intelligent Transportation Systems Magazine, IEEE*, 6(3):44–53, 2014.

[9] Michael E Farmer, Craig S Jacobs, and Shan Cong. Neural network radar processor, April 2 2002. US Patent 6,366,236.

[10] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[11] Martin T Hagan, Howard B Demuth, Mark H Beale, and Orlando De Jesús. *Neural network design*, volume 20. PWS publishing company Boston, 1996.

[12] Simon Haykin. A comprehensive foundation. *Neural Networks*, 2, 2004.

[13] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[14] Giacomo Indiveri, Bernabé Linares-Barranco, Tara Julia Hamilton, André Van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, Piotr Dudek, Philipp Häfliger, Sylvie Renaud, et al. Neuromorphic silicon neuron circuits. *Frontiers in neuroscience*, 5, 2011.

[15] P Kerlirzin and F Vallet. Robustness in multilayer perceptrons. *Neural computation*, 5(3):473–482, 1993.

[16] Benjamin Klein, Lior Wolf, and Yehuda Afek. A dynamic convolutional layer for short range weather prediction. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[18] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[19] Steve Lawrence, C Lee Giles, and Ah Chung Tsoi. What size neural network gives optimal generalization? convergence properties of backpropagation. 1998.

[20] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10), 1995.

[21] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. *Advances in neural information processing systems 2, NIPS 1989*, 2:598–605, 1990.

[22] Jeff W Lichtman, Hanspeter Pfister, and Nir Shavit. The big data challenges of connectomics. *Nature neuroscience*, 17(11):1448–1454, 2014.

[23] Nancy A Lynch. *Distributed algorithms.* Morgan Kaufmann, 1996.

[24] Henry Markram, Eilif Muller, Srikanth Ramaswamy, Michael W Reimann, Marwan Abdellah, Carlos Aguado Sanchez, Anastasia Ailamaki, Lidia Alonso-Nanclares, Nicolas Antille, Selim Arsever, et al. Reconstruction and simulation of neocortical microcircuitry. *Cell*, 163(2):456–492, 2015.

[25] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[26] Kishan Mehrotra, Chilukuri K Mohan, Sanjay Ranka, and Ching-tai Chiu. Fault tolerance of neural networks. Technical report, DTIC Document, 1994.

[27] Marvin L Minsky and Seymour A Papert. *Perceptrons - Expanded Edition: An Introduction to Computational Geometry.* MIT press Boston, MA:, 1987.

[28] Janardan Misra and Indranil Saha. Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing*, 74(1):239–255, 2010.

[29] Dharmendra S Modha, Rajagopal Ananthanarayanan, Steven K Esser, Anthony Ndirango, Anthony J Sherbondy, and Raghavendra Singh. Cognitive computing. *Communications of the ACM*, 54(8):62–71, 2011.

[30] Cairo L Nascimento Jr. Artificial neural networks in control and optimization. *Doctor Thesis. University of Manchester. Manchester*, 1994.

[31] Chalapathy Neti, Michael H Schneider, and Eric D Young. Maximally fault tolerant neural networks. *Neural Networks, IEEE Transactions on*, 3(1):14–23, 1992.

[32] Vincenzo Piuri. Analysis of fault tolerance in artificial neural networks. *Journal of Parallel and Distributed Computing*, 61(1):18–48, 2001.

[33] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para.* Cornell Aeronautical Laboratory, 1957.

[34] Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27, 1995.

[35] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[36] Fred B Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[37] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press, 2014.

[38] Hava T Siegelmann and Eduardo D Sontag. Analog computation via neural networks. *Theoretical Computer Science*, 131(2):331–360, 1994.

[39] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[40] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in Neural Information Processing Systems*, pages 2368–2376, 2015.

[41] Yasuo Tan and Takashi Nanya. Fault-tolerant back-propagation model and its generalization ability. In *Neural Networks, 1993. IJCNN'93-Nagoya. Proceedings of 1993 International Joint Conference on*, volume 3, pages 2516–2519. IEEE, 1993.

[42] Igor V Tetko, David J Livingstone, and Alexander I Luik. Neural network studies. 1. comparison of overfitting and overtraining. *Journal of chemical information and computer sciences*, 35(5):826–833, 1995.

[43] Paul Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. 1974.

Appendices

A On the activation function and its Lipschitzness

A.1 Bias

In the model, we specified that we will use an non-biased activation function for notational convenience. As in most machine learning literature, this is done without the loss of generality, considering an additional constant neuron (value = 1) in each layer. During the learning phase – when building the network – instead of learning its bias value, the neuron of layer l will just learn the weight given to the constant neuron of layer $l-1$. As this weight will be always multiplied by 1, it will serve as the bias, around which the activation function will be centred.

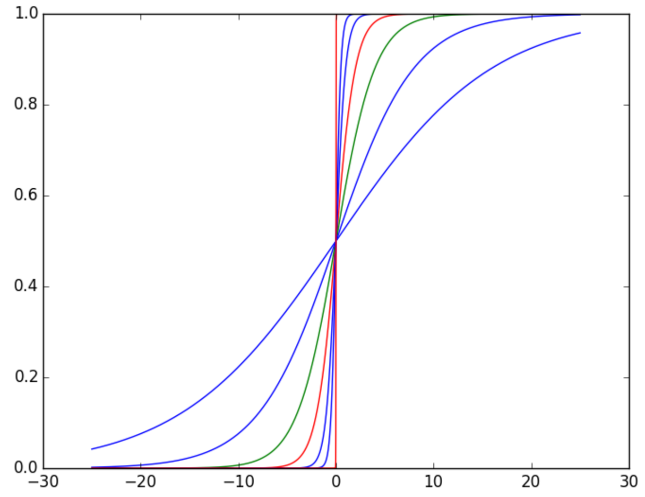

Figure 3: The profile of a sigmoid function, centred around 0 and tuned with several values of K. The larger is K, the steeper is the slope and the more discriminating is the activation function.

A.2 Lipschitzness

In the paper, we used the fact that the activation function $x \mapsto sigmoid(K.x)$ is K'-Lipschitz with K'=K/2 and K is the tuning coefficient, we prove here that this apply to *sigmoid*, which, is one of the most largely used activation functions in the literature. The reasoning generalizes to any continuous differential function, replacing 1/2 by the supremum of the first derivative of the activation function.

Lemma 3. *The activation function $\varphi(x) = sigmoid(x)$ is 1/2-Lpschitz, with 1/2 as the best Lipschitz constant on the interval $[0,1]$, and more generally, the K-tuned activation function $\varphi : x \mapsto sigmoid(K.x)$ is K'-Lipschitz with K'=K/2 as the best Lipschitz constant on the interval $[0,1]$, which is translated by the following property:*

$$\begin{aligned} &\forall (x,y) \in [0,1]^2 : |\varphi(x) - \varphi(y)| \le K'.|x-y| \\ \forall \alpha > 0, \exists (x,y) &\in [0,1]^2 : |\varphi(x) - \varphi(y)| > (K'-\alpha).|x-y| \end{aligned} \tag{7}$$

Proof of Lemma 3

Proof. We proof the lemma directly on $x \mapsto sigmoid(K.x)$

Thanks to the following inequalities:

$$\forall x \in [0,1] : |e^{-K.x}| \le 1, \ \frac{1}{|1+e^{-K.x}|} \le 1/2$$

We obtain the following bound on the first derivative of sigmoid(K.x):

$$\begin{aligned} |\frac{d\varphi}{dx}(x)| &= |\frac{K.e^{-K.x}}{(1+e^{-K.x})^2}| \\ &\le K/2 = K' \end{aligned} \tag{8}$$

Since K' is an upper bound on the absolute value of the first derivative of φ, the latter is then K'-Lipschitz on any compact subset of its input, following the mean value theorem:

$$\forall (x,y) \in [0,1]^2 : \exists c \in]x,y[: \varphi(x) - \varphi(y) = \frac{d\varphi}{dx}(c).(x-y) \tag{9}$$

Using Equation 9 and Inequality 8 applied to c, we obtain:

$$\forall (x,y) \in [0,1]^2 : |\varphi(x) - \varphi(y)| \le K'.|x-y| \tag{10}$$

Which is the first inequality of Lemma 3.

To prove that K' is the best Lipschitz constant, we look back at Equation 8, which in the case of $x = 0$ leads to $|\frac{d\varphi(x)}{dx}(0)| = K/2 = K'$

Let α be any positive real number, by continuity of the first derivative of φ, there exists $\alpha' > 0$ such that:

$$\forall c \in]0, \alpha'[: |\frac{d\varphi}{dx}(c) - \frac{d\varphi}{dx}(0)| < \alpha$$

which leads to:

$$\forall c \in]0, \alpha'[: \frac{d\varphi}{dx}(c) > \frac{d\varphi}{dx}(0) - \alpha = K' - \alpha \tag{11}$$

Using again the mean value theorem on $x = 0$ and $y = \alpha'$, similarly to Equation 9:

$$\exists c \in]0, \alpha'[: \varphi(0) - \varphi(\alpha') = \frac{d\varphi}{dx}(c).(0 - \alpha')$$

And applying Equation 11 on this latter particular c we transform the previous equation to the following inequation:

$$|\varphi(0) - \varphi(\alpha')| > (K' - \alpha')|0 - \alpha'| \tag{12}$$

Which shows that for any given $\alpha > 0$ we can exhibit two inputs, here $x = 0$ and $y = \alpha'$, such that we have the second part of Inequality 7 in Lemma 3

□

B Proof of Lemma 1

Lemma 1. *In a neural network containing L layers, if in each layer l, N_{C_l} neurons, among the N_l neurons of that same layer, are affected by errors such that each neuron j of layer l is broadcasting an output $y_j^{(l)} + \lambda_j^l$ to the next layer instead of the nominal $y_j^{(l)}$, then the effect on the output is lower or equal to $\lambda \sum_{l=1}^{L} N_{C_l} K^{L-l} \prod_{l'=l+1}^{L+1} (N_{l'} - N_{C_{l'}}) w_m^{(l')}$:*

$$\|F_{neu}(\boldsymbol{X}) - F_\lambda(\boldsymbol{X})\| \leq \lambda \sum_{l=1}^{L} N_{C_l} K^{L-l} \prod_{l'=l+1}^{L+1} (N_{l'} - N_{C_{l'}}) w_m^{(l')} \tag{13}$$

Where $F_{neu}(.)$ is the nominal neural function, $F_\lambda(.)$ the neural function accounting for the errors $\lambda_j^{(l)}$, and $w_m^{(l)} = max(|w_{ji}^{(l)}|, (j,i) \in [1, N_l][1, N_{l-1}])$ is the maximum norm of the weights of the incoming synapses to layer l.

Unless there are additional constraints on the network and on $F(.)$, this upper bound is tight.

Proof. We proceed by induction on L.

Initiation. Let $N_{fail} = N_{C_1}$ be the number of neurons failing in the single layer of the network, $let I_{fail}$ be the set containing those neurons, we have:

$$\|F_{neu}(\mathbf{X}) - F_\lambda(\mathbf{X})\| = \| \sum_{i \in I_{fail}} w_i^{(2)} (y_i^{(1)} + \lambda_i^{(1)}) \| \tag{14}$$

Which, by the triangular inequality leads to:

$$\|F_{neu}(\mathbf{X}) - F_\lambda(\mathbf{X})\| \leq \sum_{i \in I_{fail}} \|w_i^{(2)} (y_i^{(1)} + \lambda_i^{(1)})\| \tag{15}$$

With equality cases when the terms are positively proportional (**Condition 1**). Applying Hypothesis 1 and the definition of $w_m(2)$ gives us:

$$\|F_{neu}(\mathbf{X}) - F_\lambda(\mathbf{X})\| \leq N_{C_1} w_m^{(2)} \lambda \tag{16}$$

With equality case when for an input $y_i^{(1)} + \lambda_i^{(1)} = \lambda$ (**Condition 2**) and when the failing neurons are all linked to the output with the maximal weight $w_m^{(2)}$ (**Condition 3**).

We observe that Inequation 16 is the $L = 1$ version of the Lemma, and that in the worst case of failure, when no additional constraint on the network to avoid that we have Conditions 1 to 3 simultaneously occurring, the bound is tight.

Induction step. Assume that the proposition of Lemma 1 holds for networks with to some number of layers $L \geq 1$.

Now consider a network consisting of $L + 1$ layers. The layered structure of the network enables us to see each of N_{L+1} neurons of the $(L + 1)$-th layer, first as an output to an L-layer network (all the nodes the left of that neuron), and second, after applying the activation function, as a neuron in a single-layer neural network (consisting in the $L + 1$ layer alone).

In this last $(L + 1)$-th layer, we can distinguish two subsets of neurons:

1. (Failing neurons at layer L+1) A subset of $N_{C_{L+1}}$ failing neurons, that will yield, as in the initiation step (sigle layer), an error of at most $N_{C_{L+1}} w_m^{(L+2)} \lambda$
2. (Correct neurons at layer L+1) A subset of $N_{L+1} - N_{C_{L+1}}$ correct neurons, that will just forward, in addition to their nominal value, the error Er of the L-layer neural network *behind them*, multiplying it by at most the maximum synaptic weight from layer L to layer L+1, $w_m^{(L+1)}$ and the Lipschitz constant K, yielding an error of at most $Er(N_{L+1} - N_{C_{L+1}})K$

By the induction hypothesis we have:
$Er \leq \lambda \sum_{l=1}^{L} N_{C_l} K^{L-l} \prod_{l'=l+1}^{L+1} (N_{l'} - N_{C_{l'}}) w_m^{(l')}$
As the output node is linear (not part of the neural network, not performing any non-linear activation function), the errors mentioned in 1 and 2 will be added and yield a total error bounded as follow:

$$\begin{aligned} \|F_{neu}(\mathbf{X}) - F_\lambda(\mathbf{X})\| &\leq N_{C_{L+1}} w_m^{(L+2)} \lambda + (N_{L+1} - N_{C_{L+1}}) K \lambda \sum_{l=1}^{L} N_{C_l} K^{L-l} \prod_{l'=l+1}^{L+1} (N_{l'} - N_{C_{l'}}) w_m^{(l')} \\ &= \lambda \sum_{l=1}^{L+1} N_{C_l} K^{L+1-l} \prod_{l'=l+1}^{L+2} (N_{l'} - N_{C_{l'}}) w_m^{(l')} \end{aligned} \tag{17}$$

Which is the desired bound for an $L + 1$-layer network. The equality case follows from considering the inter-occurrence of the equality cases at all the contributing parts, in case no constraint on the network is set to avoid it.

By induction, the proposition of Lemma 1 is true for any integer $L \geq 1$. □

C Proof of Lemma 2

Lemma 2. *In an L-layer neural network, an error of value $\lambda_{ji}^{(l)}$ in a synapse linking neuron i from layer $l - 1$ to neuron j at layer l is equivalent to an error at neuron j of at most $K|\lambda_{ji}^{(l)} w_{ji}^{(l)}|$ for*

$l \leq L$, and to an error or at most $|\lambda_{ji}^{(l)} w_{ji}^{(l)}|$ if the synapse is linking the last layer $l-1 = L$ to the output node $l = L+1$. Under similar absence of constraints as before, this bound is tight.

Proof. Let l be a hidden layer in the neural network, and let i and j any neurons from $l-1$ and l respectively.

An error of value λ in neuron i yields a received sum at neuron j, noted $s_{\lambda,j}^{(l)}$ and given by Equation 1 as follow:

$$\begin{aligned} s_{\lambda,j}^{(l)} &= \sum_{k=1,k\neq i}^{N_{l-1}} w_{jk}^{(l)} y_k^{(l-1)} + w_{ji}^{(l)}(y_i^{(l-1)} + \lambda_{ji}^{(l)}) \\ &= \sum_{k=1}^{N_{l-1}} w_{jk}^{(l)} y_k^{(l-1)} + w_{ji}^{(l)} \lambda_{ji}^{(l)} \end{aligned} \tag{18}$$

Therefore, by K'-Lipschitzness of the activation function, in the case of $l < L+1$, the output error of neuron j is given by the following equation:

$$\begin{aligned} |error| &= |\varphi(s_{\lambda,j}^{(l)}) - \varphi(s_j^{(l)})| \\ &\leq K'.|s_{\lambda,j}^{(l)} - s_j^{(l)}| \\ &= K'.|\lambda_{ji}^{(l)} w_{ji}^{(l)}| \end{aligned} \tag{19}$$

and in the case of the last layer (no activation function at the output node):

$$\begin{aligned} |error| &= |s_{\lambda,j}^{(l)} - s_j^{(l)}| \\ &= |\lambda_{ji}^{(l)} w_{ji}^{(l)}| \end{aligned} \tag{20}$$

□