Friendly Training: Neural Networks Can Adapt Data To Make Learning Easier

Simone Marullo*,†, Matteo Tiezzi†, Marco Goš†‡, Stefano Melacci†
*Dept. of Information Engineering, University of Florence, Florence, Italy
†Dept. of Information Engineering and Mathematics, University of Siena, Siena, Italy
‡Universiti Côte d’Azur, Nice, France

Abstract—In the last decade, motivated by the success of Deep Learning, the scientific community proposed several approaches to make the learning procedure of Neural Networks more effective. When focussing on the way in which the training data are provided to the learning machine, we can distinguish between the classic random selection of stochastic gradient-based optimization and more involved techniques that devise curricula to organize data, and progressively increase the complexity of the training set. In this paper, we propose a novel training procedure named Friendly Training that, differently from the aforementioned approaches, involves altering the training examples in order to help the model to better fulfill its learning criterion. The model is allowed to “simplify” those examples that are too hard to be classified at a certain stage of the training procedure. The data transformation is controlled by a developmental plan that progressively reduces its impact during training, until it completely vanishes. In a sense, this is the opposite of what is commonly done in order to increase robustness against adversarial examples, i.e., Adversarial Training. Experiments on multiple datasets are provided, showing that Friendly Training yields improvements with respect to informed data sub-selection routines and random selection, especially in deep convolutional architectures. Results suggest that adapting the input data is a feasible way to stabilize learning and improve the generalization skills of the network.

Index Terms—Friendly training, data alteration, curriculum learning, neural networks.

I. INTRODUCTION

The outstanding results yielded by Neural Networks in several real-world tasks in the last decade [1]–[3] have strongly motivated researchers to put even more effort in improving the design of neural architectures and to introduce tools to make the learning procedure more effective. In the framework of optimization-based learning, a number of improved methods are nowadays extremely popular across different application fields: regularization techniques [4], learnable normalization functions [5], powerful adaptive methods to estimate the appropriate learning rates [6], to name a few.

This work was partly supported by the PRIN 2017 project RexLearn, funded by the Italian Ministry of Education, University and Research (grant no. 2017TWNMH2).

Accepted for publication at the IEEE International Joint Conference on Neural Networks (IJCNN) 2021 (DOI: TBA).

2021 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

In this paper we propose an approach that introduces a novel perspective in the learning dynamics of Neural Networks, and is compatible with the aforementioned techniques. In particular, we consider the possibility of extending the search space of the learning algorithm, allowing the network not only to adapt its weights and biases, but also to adapt on-the-fly the training data, in order to facilitate the fulfilment of its learning criterion. Of course, such adaptation, that we also refer to as “simplification”, must be controlled and embedded into a precise developmental plan in which the machine is progressively constrained to reduce the amount of simplification, until it ends up in handling the original training data as they are.

The key intuition behind this training strategy, that we named Friendly Training (FT), is that instead of exposing the network to an uncontrolled variety of data with heterogeneous properties over the input space, the learning can rather be guided by the information that the network has learnt to process so far. The aim of such technique is to operate on noisy data, outliers, and, more generally, on whatever falls into the areas of the input space that the network finds hard to handle at a certain stage of the learning process. Such data are modified to mitigate the impact of the information that is inconsistent with what has been learnt so far. In a sense, FT alters data so that they are just slightly more complicated with respect to what the learner is currently able to process, thus inducing the network to smoothly update the learnt function. This procedure is iterated until, at the end of training, the learnt function is consistent with the information carried by the original training data (see the toy example of Fig. 1).

This strategy echoes the pedagogical approach formulated by the psychologist Lev Vygotsky [7], according to which children learn by a progressive reduction of the so-called Zone of Proximal Development, i.e., the space between what the learner can and cannot do autonomously, which contains tasks that the learner can accomplish if appropriately guided. In our approach, the network itself decides how to adapt the data to foster its learning process, without the need of extra information on the problem or additional scoring functions.

The idea of providing data to the network following an easy-data-first plan has been popularized in the last decade by Curriculum Learning (CL) [8]. In this case, the training
set is progressively expanded by adding more difficult data, that are selected and ordered accordingly to a scoring function that might come from further knowledge on the considered problem or other heuristics. A recent study [9] on the impact of CL showed the importance of CL when dealing with noisy settings or in case of limited time or computational resources. The idea of CL has been recently applied to the case of convolutional architectures with a progressive smoothing of the convolutional feature maps [10]. A related research area is the one of Self-Paced Learning [11], in which some examples are either excluded from the training set or their impact in the loss function is kept low if some conditions, modeled by a specific regularizer, are met [12]. Such conditions depend on the current state of the classifier, and the whole process is repeated multiple times. Differently from what we propose, these techniques do not alter the training data. From the algorithmic point of view, our approach can be seen as related to the so-called Adversarial Training strategies [13], [14], although oriented toward completely different goals. Such strategies exploit the idea of providing the network with examples specifically generated to be potentially capable of fooling the classifier. Such samples are then added to the original training data to make the classifier more robust to adversarial attacks. The price to pay is that this procedure might have a negative impact on the generalization skills of the classifier [15]. In this respect, it has been recently shown that it is better to limit the data alteration during the early stages of the learning process, in order to implement a Friendly Adversarial Training policy [14]. We partially borrowed this intuition, although focusing on the opposite direction, i.e., we let the classifier help itself by altering the data, instead of creating more difficult training conditions. The aim of our approach is to improve the network generalization capability by letting the network learn in a friendly environment.

In this manuscript, we provide an experimental analysis conducted on five datasets available in the related literature. Results suggest that Friendly Training is a feasible way to improve the generalization skills of the network, and to create a smooth evolution of the learning environment, avoiding to expose the network to uncontrolled information at the wrong time. The simplifications, on average, look more structured in (deep) convolutional architectures, leading to more effective improvements, while their impact in fully-connected networks is more spread over all the dimensions of the input space, resulting in less appreciable gains. In the following, we provide an in-depth analysis of the sensitivity of the system to the newly introduced hyper-parameters, and we compare Friendly Training with randomly ordered data or curricula defined by ad-hoc criteria.

The contributions of this paper are the following ones: (1) we propose a novel training strategy, named Friendly Training (FT), that allows the machine to partially simplify the data by automatically determining how to alter it; (2) we propose a developmental plan that allows the effects of FT to progressively fade out, in order to create a smooth transition from simplified data to the original one; (3) we experimentally evaluate the FT approach in convolutional and fully connected neural architectures with different number of layers and considering five different datasets, analyzing the impact of the simplification.

This paper is organized as follows. FT, including the developmental plan, is described in Section II. The relation with related work are deeply analyzed in Section III. Experiments and results are described in Section IV, while conclusions and suggestions for future work are drawn in Section V.

II. FRIENDLY TRAINING

We consider a generic classification problem in which we are given a training set $\mathcal{X}$ composed of $n$ supervised pairs, $\mathcal{X} = \{(x_k, y_k), k = 1, \ldots, n\}$, being $x_k \in \mathbb{R}^d$ a training example labeled with $y_k$. Given some input data $x$ we denote with $f(x, w)$ the function computed by a Neural Network-based classifier with all its weights and biases stored into vector $w$. When optimizing the model exploiting a mini-batch based stochastic gradient descent procedure, at each step of the training routine the following loss function $L$ is used for computing the gradient with respect to $w$ and updating the model parameters:

$$L(B, w) = \frac{1}{|B|} \sum_{i=1}^{|B|} \ell(f(x_i, w), y_i),$$

where $B \subset \mathcal{X}$ is a mini-batch of data of size $|B| \geq 1$, $(x_i, y_i) \in B$, and $\ell$ is the loss restricted to a single example. For simplicity, we assumed to aggregate the contributes of $\ell$ by averaging over the mini-batch data. The set $B$ can be sampled with different strategies. In the most common case of stochastic gradient optimization, a set of mini-batches is randomly sampled at each training epoch, in order to cover the whole set $\mathcal{X}$ without intersections among the mini-batches.

Of course, the data that populate $\mathcal{X}$ might include examples with different properties, in a task-dependent manner. To give some examples, the distribution of the training data could be multi-modal, might include outliers, might span over several disjoint manifolds, and so on. However, the aforementioned training procedure provides data to the machine independently on the state of the network and with no control on the information carried by such data. If additional information is available on the problem at hand, it might be exploited to devise curricula providing simple-examples-first, as in CL [8]. However, it is very unlikely to have information on the complexity of examples in advance and, even more importantly, the human-based criteria might not match the way in which examples are processed by the machine. The value $\ell$ could be used as an indicator to estimate such simplicity, to exclude those examples with too large loss or to reduce their contribution in Eq. (1), similarly to what is done in [11], [12].

In this paper, instead, we propose an alternative approach that, differently from what has been described so far, analyzes

---

1. We consider the case of classification mostly for the sake of simplicity. The proposed approach goes beyond classification problems.

2. We postpone to Section III a more detailed description of these strategies.
the training data according to the state of the learner, allowing
the network to modify such data, eventually discarding the
parts of information that are too complex to be handled by
the model at that moment, while preserving what sounds more
coherent with the expectation of the current classifier. Notice
that this is significantly different from deciding whether or not
to keep a training example, to give small or big weight to them
in Eq. (1), or to simply re-order the examples. Interestingly,
FT is compatible with (and not necessarily an alternative to)
the aforementioned existing strategies.

Formally, each example $x_i$ is altered to $\tilde{x}_i$ by adding a
learnable perturbation $\delta_i$,

$$\tilde{x}_i = x_i + \delta_i,$$

where $\delta_i \in \mathbb{R}^d$. Given a mini-batch $B$, we indicate with $\Delta$ the
matrix that collects the perturbations associated to the mini-
batch examples. In detail, the $i$-th row of $\Delta$ is the perturbation
$\delta_i$ associated with the $i$-th example in $B$. For convenience in
the notation, we avoid mentioning training epochs in what
follows, and we describe the training procedure as the iterative
processing of mini-batches of data, updating $w$ after each of
them. Let us denote with $\gamma \geq 1$ the iteration index. We
re-define the aggregated loss $L$ of Eq. (1) by providing the
network with $\tilde{x}$ instead of $x$, introducing the dependency on
$\Delta$, and by adding the iteration index $\gamma$,

$$L(B^\gamma, \Delta^\gamma, w^\gamma) = \frac{1}{|B^\gamma|} \sum_{i=1}^{|B^\gamma|} \ell(f(\tilde{x}_i, w^\gamma), y_i),$$

where $\tilde{x}_i$ is defined as in Eq. (2), $(x_i, y_i) \in B^\gamma$ (i.e., $B^\gamma$
is the mini-batch at iteration $\gamma$) and $\delta_i$ is the $i$-th row of
$\Delta^\gamma$. Jointly optimizing Eq. (3) with respect to $\Delta^\gamma$ and $w^\gamma$
allows the network not only to adapt its weights and biases
in order to better cope with the learning criterion, but also
to alter the data in $B^\gamma$ by translating them in those space
regions that can be more easily classified. However, the loss of
Eq. (3) does not introduce any constraints on each $\delta_i$. Hence,
the network is free to change the training data without any
guarantees that the simplification amount will reduce while
learning proceeds. Moreover, differently from $w^\gamma$, the set $\Delta^\gamma$
is specifically associated with the data in $B^\gamma$ (i.e, each training
example is associated with its own perturbation), meaning that
the number of variables of the optimization problem becomes
a function of the size of the training data.

We frame Eq. (3) in the context of a developmental plan
that solves both these issues. First, the system is enforced to
reduce the perturbations as long as the number of training
iterations increases. If $\gamma_{max}$ is the maximum number of
allowed iterations, we ensure that after $\gamma_{max, simp} < \gamma_{max}$
steps the data are not perturbed anymore. Secondly, we remove
the dependence of $\Delta^\gamma$ from $\gamma$, introducing an Alternate
Optimization scheme in which we decouple the optimization
of perturbations and weights. In detail, we consider a single
matrix $\Delta$ that is shared by all the mini-batches (the total
number of rows in $\Delta$ is equal to the size of a single mini-
batch). At the beginning of each training iteration, we keep
$w^\gamma$ fixed, we initialize $\Delta$ to zeros and then we estimate the
appropriate perturbations for the current data in $B^\gamma$ by gradient
descent over the variable $\Delta$ (second argument of Eq. (3)). We
indicate with $\tau^\gamma$ the number of iterative steps of such inner
optimization. The value of $\tau^\gamma$ controls the amount of alteration
on the data. For small values of $\tau^\gamma$ the network will only marginally simplify the data, while for a larger $\tau^\gamma$ the data alteration will be more aggressive. The initial value $\tau^1 \geq 1$ is a fixed hyper-parameter, while we considered a quadratic law to progressively reduce $\tau$ in function of $\gamma$,

$$\tau^\gamma = \tau^1 \cdot \max \left(1 - \frac{\gamma - 1}{\gamma_{\text{max simp}}}, 0\right)^2. \quad (4)$$

Afterwards, we update the values of $w^\gamma$, given the output $\Delta$ at the end of the just described inner optimization. This developmental plan allows the system to adapt the data in order to make the learning procedure less disruptive, especially during the earlier stages of learning, introducing a smooth optimization path driven by the evolution of $\tau^\gamma$. The detailed training procedure is formally reported in Algorithm 1 and in the following lines we provide some further details. Notice that while the weight update equation (line 10) can include any existing adaptive learning rate estimation procedure, in our current implementation $\Delta$ is updated by a fixed small learning rate (line 7). While Algorithm 1 formally returns the weights after having completed the last training iteration, as usual, the best configuration of the classifier can be selected by measuring the performance on a validation set, when available. Another important fact to mention is that when the prediction on examples perfectly matches target, line 6 will return zero gradient, hence no simplifications are performed. In our implementation we slightly relaxed this condition by zeroing the rows of $\Delta_{grad}$ (line 6) associated to those examples that are classified with a large confidence above a given threshold (see Section IV), that implements a selective early-stopping condition on the inner optimization.

We qualitatively show the behavior of the proposed training strategy in the toy example of Fig. 1. A very simple network with one hidden layer (5 neurons with hyperbolic tangent activation function) is trained on the popular two-moon dataset (two data distributions shaped as interleaving half-circles), optimized by Adam with mini-batch of size 16. When the network (having the same initial weights) is trained with the Friendly Training algorithm, it is less subject to oscillations in the learning process, resulting in a more controlled development of the classifier and leading to a final decision boundary that better fits the data distribution. With the Friendly Training approach, the mini-batch examples (filled dots) are altered during the first iterations (second row, first two pictures - compare their position with the corresponding pictures in the first row, which represent Classic Training). Notice that the mini-batch examples occupy their original positions during later stages (final stage of developmental plan).

### III. Relationships with Existing Work

The idea of training neural networks with a learning methodology that “gradually” changes the learning environment (which proves to be appropriate not only in humans but also in animals [16]) traces back to almost three decades ago [17] and has been known for long as Curriculum Learning (CL) [8]. Taking inspiration from the common experience of learning in humans, CL aims at designing an optimal learning plan in which the learning agent is exposed to simple, easily-discriminable examples at first, and later to gradually harder examples, also progressively increasing the size of the training set. CL proved to be successful in training deep networks on a variety of tasks [18], [19]. The concrete effect of CL has been recently revisited [9], showing how the positive impact of curriculum-based learning strategies is mostly evident in noisy setups (e.g., random permutation of labels) or large-data regimes with limited time availability. Nonetheless, recent evidence suggests that elementary curriculum-based tricks might help in vision-related tasks, at least when very deep and wide networks are used. Curriculum By Smoothing (CBS) [10] is built upon the idea of applying Gaussian-based low-pass filters on feature maps of Convolutional Neural Networks (CNNs) that process image data, with variance that progressively goes to zero as training proceeds. Even though CL remains a somewhat controversial topic and has not been widely adopted in the machine learning community, more research may be needed before ruling out this approach. In fact, there is a quite general consensus on the fact that what happens in the early stages of deep network training affects the network behavior at steady state [20]. For instance, the authors of [21] showed that inflicting visual impairments to deep CNNs in the first epochs leads to minor visual skills despite conceding recovery time.

The purpose of Friendly Training, on the opposite, is to downplay difficult, confusing and outlier examples at the beginning, while let them contribute to the generalization capability when the learner has already acquired basic skills. With respect to CL, Friendly Training does not alter the size of the dataset as a function of the number of training iterations, nor the relative ordering in which examples are presented to the network. Moreover it does not assume any predefined complexity criteria. Conversely, Friendly Training alters every single example in an adaptive way, unless it

### Algorithm 1 Friendly Training.

**Input:** Training set $X$, initial weights and biases $w^1$, batch size $b$, max learning steps $\gamma_{\text{max simp}}$, max simplification steps $\gamma_{\text{max simp}}$, $\gamma^1 \geq 1$, learning rates $\alpha > 0$ and $\eta > 0$, shared matrix $\Delta$ with $b$ rows and $d$ columns.

**Output:** The final $w^{\gamma_{\text{max simp}}+1}$.

```plaintext
1: for $\gamma = 1$ to $\gamma_{\text{max simp}}$ do
2: Sample $B^\gamma$ of size $b$ from $X$.
3: Compute $\tau^\gamma$ following Eq. (4).
4: Set all the entries of $\Delta$ to 0
5: for $\tau = 1$ to $\tau^\gamma$ do
6: Compute $\Delta_{grad} = \frac{\partial L(B^\gamma, D, w^\gamma)}{\partial D}|_{D=\Delta}$, see Eq. (3)
7: $\Delta = \Delta - \eta \cdot \Delta_{grad}$
8: end for
9: Compute $w_{grad}^\gamma = \frac{\partial L(B^\gamma, \Delta, v)}{\partial v}|_{v=w^\gamma}$, see Eq. (3)
10: $w^{\gamma+1} = w^\gamma - \alpha \cdot w_{grad}^\gamma$
11: end for
12: return $w^{\gamma_{\text{max simp}}+1}$
```
correctly predicted with a sufficiently high confidence, and only afterwards the example is used to update the network weights. In so doing, the network is exposed, at every training step, to a data population with quite large variability, although their distribution has been slightly adapted, in order to decrease the occurrence of abrupt weight changes. Differently from CBS, Friendly Training can be applied to any type of data and, interestingly, it is compatible with CBS, and not necessarily an alternative to it.

Self-Paced Learning (SPL) [11] is a technique inspired by CL, originally designed for Structural Support Vector Machines (SSVMs) with latent variables. Self-paced stands for the fact that the curriculum is determined by the pupil’s abilities (the classifier’s behaviour) rather than by a teacher’s plan. Since this direction proved to be effective when compared with standard algorithms (latent variable models correspond to hard optimization problems), researchers adapted this formulation to CNNs [12]. The basic idea consists of searching for suitable example-specific weighting coefficients in the loss computation. In contrast to this approach, what we propose is not about weighting the importance of training examples, but rather temporarily altering them. However, we do embrace the idea that the state of the classifier is what can be used to determine how to deal with a particular input/target pair, gradually exposing the learner to more and more difficult examples, with an explicit temporal dynamics.

Another technique that is somehow related to Friendly Training is the so-called Adversarial Training (AT) [13], [22], which may be seen as the inverse learning technique of FT. Developed as an empirical defense strategy for adversarial attacks, which seriously affect neural networks operating on high-dimensional spaces [22], AT incorporates adversarial data into the training process. Most AT techniques rely on a minimax optimization problem [13], since the goal is to generate adversarial examples that strongly fool the classifier. Each generated example is an artificial element, lying very close to an original training data point of a certain class, but that is classified incoherently with respect to the label attached to such original point. Interestingly, in AT the system basically alters examples as in Eq. 4, even if the perturbation is computed with a different criterion and typically with no temporal dynamics.

Friendly Adversarial Training (FAT) [14] builds up on the ideas of both CL and AT. Researchers noticed [23] that the adversarial formulation sometimes hurts generalization capabilities [15]. The FAT strategy provides a more gentle learning problem, in which the generation of adversarial data is early-stopped as soon as the datapoint is misclassified. The resulting learning dynamics is such that, as learning progresses (together with accuracy and robustness), more and more iterations are needed to generate (harder) adversarial data. Our Friendly Training algorithm shares some intuitions with the FAT algorithm (consider the comments on early-stopping right after Algorithm 1), even though the direction of the iterative process which generates input data has a different goal. Moreover, FAT does not include explicit temporal dynamics.

IV. EXPERIMENTS

We describe our experimental experience by introducing the considered datasets in Section IV-A, the neural architectures, the parameter tuning procedure and the competitors in Section IV-B, and by reporting and discussing the numerical and qualitative results in Section IV-C, also including an analysis of the sensitivity of key parameters.

A. Datasets

In order to assess the performance of the proposed learning algorithm, we employed the datasets presented in [24]. Some of these datasets are about 10-class digit recognition problems (28 × 28, grayscale), and they were explicitly designed to provide harder learning conditions with respect to the well-known MNIST dataset, keeping an affordable size (62k examples per dataset, already divided into training, validation and test set). The data distribution is the product of multiple factor distributions (i.e., rotation angle, background, etc., besides factors inherent to the original data, such as the handwriting style), making them a challenging benchmark. In detail:

- *mnist-rot*: MNIST digits, rotated by a random angle $\theta$, so that $\theta \in [0, 2\pi]$.
- *mnist-back-image*: MNIST digits, with the background replaced with patches extracted by random images of public domain.
- *mnist-rot-back-image*: MNIST digits, with both the rotation and background factors of variations (combined).

We also considered other different types of data still from [24], aimed at investigating how neural networks deal with geometrical shapes and learn their properties. They share the same resolution and almost the same size of the previously mentioned datasets, but they are used for geometry-based binary classification tasks. They are:

- *rectangles-image*: white rectangles, wide or tall, with the inner and outer regions filled with patches taken from random images.
- *convex*: convex or non-convex white regions on a black background.

B. Experimental Setup

We focused on four neural network architectures, where two of them are feed-forward Fully-Connected multi-layer perceptrons, referred to as FC-A and FC-B, and the others are Convolutional Neural Networks named CNN-A and CNN-B. FC-A is a simple one-hidden-layer network with hyperbolic tangent activations (10 hidden neurons), while FC-B is inherited from [21], has 5 hidden layers (2500-2000-1500-1000-500 neurons), batch normalization and ReLU activations. CNN-A consists of 2 convolutional layers, max pooling, dropout and 2 fully connected layers; CNN-B is deeper (4 convolutional layers). Both of them employ ReLU activation functions on the convolutional feature maps (32-64 filters in CNN-A, 32-48-64-64 filters in CNN-B) and on the fully connected layers activations (9216-128 neurons for CNN-A, 5184-128 neurons for CNN-B). Cross-entropy is used as loss function, while
the Adam optimizer is exploited for the network weights and biases, using mini-batches of size 32.

We compared the test error rates of these architectures when trained under the following conditions:

- **Classic Training (CT):** training of the neural network parameters with random selection of examples (no intervention on data), as in most nowadays cases.
- **Friendly Training (FT):** the FT training procedure of Algorithm 1 is exploited.
- **Easy-Examples First (EEF):** mini-batch examples are sorted by loss value in ascending order and only the first $k$ of them are used to calculate the gradients. At the beginning, $k = 1$, so that only 1 example per mini-batch is kept, then $k$ grows with the training iterations following the same dynamics of FT. The criterion implemented by EEF can be identified as a basic instance of CL, closely related to FT (i.e., it has the same temporal dynamics of FT, but examples are not altered and only sub-selected).

All the experiments are executed for 200 epochs (trivially, $γ_{\text{max}}$ is the number of epochs multiplied by the number of mini-batches per epoch), which consistently proved to be sufficient to obtain convergence in all the learning problems. Moreover, the metrics we report are about the epoch with the lowest validation error. All the experiments referring to the same architecture share the same initialization weights; examples are presented in the same order.

The hyper-parameters of FT and EEF were selected by grid search. We indicate with $c$ the threshold above which a prediction is considered correct in order to early stop the data transformation of FT (see the description right after Algorithm 1). In detail, we considered the following ranges: $η ∈ [0.01, 0.1, 1.0, 5.0, 10.0]$, $τ^1 ∈ [10, 80, 120]$, $c ∈ [0.9, 0.95, 0.98]$, $γ_{\text{max, simp}} ∈ [0.25, 0.5, 0.85] · γ_{\text{max}}$. The above intervals proved to be suitable in preliminary investigations. Each experiment was repeated 3 times (results are averaged), varying the initialization of the weights and biases (sharing them among the competitors).

### C. Results and Discussion

We report the test error rate we obtained in Table 1 along with some reference results by other authors [12, 24] (top-portion of Table 1). Such reference results are about Fully-connected networks (NNet), Deep Belief Networks (DBNs, two different architectures), Convolutional Autoencoders (CAEs, two different architectures) – see [24] for further details. General Stochastic Network (GSN) is described in [25]. Finally, SPCN is an instance of Self-Paced Learning based on CNNs, described in [12].

Regardless of the learning algorithm, convolutional architectures CNN-A and CNN-B consistently achieve lower error rates, as expected. Several of the reference results are beaten by the architectures we experimented, also in the case of CT, due to the fact that we considered models with a larger number of parameters. Nonetheless, the general trend is coherent with reference experiments, confirming the validity of our experimental setup (e.g., compare FC-A and FC-B with NNet, or SPCN with CNN-A and CNN-B, for example).

When comparing CT, EEF, and FT, we notice that albeit EEF occasionally improves the baseline (CT) result (e.g., in the mnist-back-image task with convolutional networks), it degrades the performance in most of the datasets, confirming that simply keeping the low-loss examples of the training set, although injected in a progressive developmental plan, does not work as a trivial trick to improve performance. Differently, the FT algorithm clearly improves the test error rate when used on the convolutional architectures CNN-A and CNN-B, providing better results than the competitors in most of the cases. In the case of CNN-A, the error is systematically lowered (sometimes in a statistically significant manner), with the exception of the convex dataset. Regarding CNN-B, we still get improvements in four out of five datasets, even if not as evident as in CNN-A. These results confirm the validity of the proposed learning technique. Fully-Connected networks FC-A and FC-B, conversely, are usually not improved by FT, with a few exceptions. A further inspection on the way the system is computing the perturbation offsets revealed important elements that explain these results, pointing out at interesting facets of FT that we will describe in the following.

We qualitatively evaluated the perturbation $δ$ in Eq. (3), considering the CNN-A model of Table 1. In Fig. 2 we report the perturbations at different steps of the first epoch, applied to four randomly selected examples. The perturbation on the left concerns an example altered at the very early stages of the training procedures. The noise level decreases as long as learning proceeds (left-to-right in Fig. 2), until we see that the system stops perturbing the background and some regions of digit area are emphasized. This is coherent with the fact that, as long as the network learns to focus on the digit area to discriminate the data, there is no need to alter the background anymore.
TABLE I: Performance comparison of classifiers with different architectures (FC-A, FC-B, CNN-A, CNN-B) and learning algorithms (CT, EEF, FT). Mean test error (smaller is better) is reported along with standard deviation. The first portion of rows is about reference results taken from existing work (see the paper text for more details). For each architecture, we report in bold those results that improve the baseline (CT) case.

| Classifier | mnist-back-image | mnist-rot-back-image | mnist-rot | rectangles-image | convex |
|------------|------------------|-----------------------|-----------|------------------|-------|
| NNet       | 27.41 ± 0.39     | 62.16 ± 0.43          | 18.11 ± 0.34 | 33.20 ± 0.41     | 32.25 ± 0.41 |
| DBN-1      | 16.15 ± 0.32     | 52.21 ± 0.44          | 14.69 ± 0.31 | 23.69 ± 0.37     | 19.92 ± 0.35 |
| DBN-3      | 16.31 ± 0.32     | 47.39 ± 0.44          | 10.30 ± 0.27 | 22.50 ± 0.37     | 18.63 ± 0.34 |
| CAE-1      | 16.70 ± 0.33     | 48.10 ± 0.44          | 11.59 ± 0.28 | 21.86 ± 0.36     | n.a.   |
| CAE-2      | 15.50 ± 0.32     | 45.23 ± 0.44          | 9.66 ± 0.26  | 21.54 ± 0.36     | n.a.   |
| GSN        | 16.04 ± 0.07     | 43.86 ± 0.05          | 8.66 ± 0.08  | 22.10 ± 0.03     | n.a.   |
| SPCN       | 9.55 ± 0.06      | 35.26 ± 0.05          | 9.81 ± 0.07  | 10.60 ± 0.03     | n.a.   |

| Classifier | mnist-back-image | mnist-rot-back-image | mnist-rot | rectangles-image | convex |
|------------|------------------|-----------------------|-----------|------------------|-------|
| FC-A / CT  | 28.34 ± 0.09     | 64.06 ± 0.31          | 43.16 ± 0.51 | 24.31 ± 0.21     | 33.91 ± 0.44 |
| FC-A / EEF | 28.18 ± 0.47     | 64.27 ± 0.19          | 43.91 ± 0.73 | 24.48 ± 0.11     | 33.17 ± 0.93 |
| FC-A / FT  | 28.66 ± 0.06     | 64.14 ± 0.36          | 43.24 ± 0.43 | 24.64 ± 0.37     | 34.38 ± 0.22 |
| FC-B / CT  | 21.06 ± 0.39     | 51.71 ± 0.79          | 10.13 ± 0.27 | 25.10 ± 0.20     | 27.24 ± 0.05 |
| FC-B / EEF | 21.38 ± 0.18     | 52.95 ± 0.63          | 10.04 ± 0.17 | 24.84 ± 0.32     | 28.21 ± 0.96 |
| FC-B / FT  | 21.74 ± 0.26     | 51.02 ± 0.07          | 11.19 ± 0.37 | 24.14 ± 0.53     | 27.49 ± 0.07 |
| CNN-A / CT | 7.25 ± 0.16      | 29.05 ± 0.45          | 7.48 ± 0.14  | 9.86 ± 0.32      | 8.24 ± 0.09  |
| CNN-A / EEF| 7.02 ± 0.08      | 29.12 ± 0.34          | 7.61 ± 0.22  | 12.82 ± 0.70     | 8.72 ± 0.74  |
| CNN-A / FT | 6.80 ± 0.19      | 28.74 ± 0.29          | 7.36 ± 0.06  | 9.72 ± 0.20      | 8.59 ± 1.44  |
| CNN-B / CT | 5.15 ± 0.15      | 23.05 ± 0.21          | 6.58 ± 0.06  | 8.10 ± 1.90      | 3.01 ± 0.41  |
| CNN-B / EEF| 4.82 ± 0.19      | 22.89 ± 0.49          | 7.02 ± 0.28  | 8.35 ± 1.01      | 3.75 ± 0.58  |
| CNN-B / FT | 5.03 ± 0.11      | 22.81 ± 0.36          | 6.95 ± 0.12  | 7.32 ± 1.31      | 2.87 ± 0.42  |

networks, and suggests directions for future improvements. As a matter of fact, the simplified examples get very artificial and far from the distribution of the original training data. In Fig. 4, we report the evolution of the error rate during the training epochs (mnist-back-image, CNN-A), comparing FT and CT. We also report a black curve that is proportional to $\tau^\gamma$, thus showing how the developmental plan reduces the impact of the perturbation (when it becomes zero, data are not altered anymore). The small bump right before 100 epochs is due to

![Fig. 3: Original data $x$ (first column), perturbation $\delta$ (second column - emphasized by normalization to make it more visible) and resulting “simplified” images $\tilde{x}$ (third column) for different networks (Table I) at the end of the first epoch. Some simplifications are hardly distinguishable by a human.](image1)

![Fig. 4: Training and test error rates for FT and CT on a single run (mnist-back-image, CNN-A architecture). The black curve (plan) is proportional to $\tau^\gamma$.](image2)

features in salient regions covered by the digits. On the other hand, $\delta$’s obtained from the FC-A and FC-B models (Fig. 3a, 3c) are very noisy, with a limited emergence of visual structure in the deeper model FC-B (3c). However, still most of the pixels get altered. This rowdy perturbation might be the reason behind the limited success of FT in the fully-connected networks, and suggests directions for future improvements.
as expected, while it becomes lower when the simplification rate vanishes.

We also evaluated the sensitivity of the system to the main hyper-parameters of FT. In Fig. 5 we report the test error of CNN-A, mnist-back-image dataset, for different configurations of \( c \), \( \eta \), \( \tau^1 \) and \( \frac{\gamma_{\text{max simp}}}{\gamma_{\text{max}}} \) in a sample run that is pretty representative of the general trend we observed in the experiments. Larger values of \( c \) reduce the frequency of early-stops in computing the perturbations, thus allowing FT to alter the data more extensively and improve the performance. The preference on the largest value of \( \tau^1 \) suggests that an aggressive perturbation at the early stages of learning helps. Small learning rates \( (\eta) \) allow the system to avoid extreme alterations of the data, while a developmental plan covering at least 50% of the training steps seems more effective than the shorter-term plans, \( \frac{\gamma_{\text{max simp}}}{\gamma_{\text{max}}} = 0.5 \).

Fig. 5: Test error (single run) of CNN-A (mnist-back-image dataset) under different configurations of the FT hyper-parameters (see the paper text for a description).

V. CONCLUSION AND FUTURE WORK

We presented a novel training procedure named Friendly Training, that allows a network to alter the training data accordingly to a developmental plan, in order to implicitly learn from more manageable data. Differently from related work, the network decides what information to discard from the data at different stages of the training procedure, leading to improved generalization skills and a smoother development of the decision boundaries. We plan to extend this approach introducing a separate neural model to estimate the simplification that should be applied to the data, instead of directly optimizing the perturbations.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al., “Mastering the game of go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell et al., “Language models are few-shot learners,” in *Advances in Neural Information Processing Systems*, 2020.

[4] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[5] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[6] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *International Conference on Learning Representations*, 2015. [Online]. Available: https://arxiv.org/abs/1412.6980

[7] L. S. Vygotsky, *Mind in society: The development of higher psychological processes*. Cambridge, MA: Harvard University Press., 1987.

[8] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in *International conference on machine learning*. PMLR, 2009, p. 41–48.

[9] X. Wu, E. Dyer, and B. Neyshabur, “When do curricula work?” in *International Conference on Learning Representations*, 2021. [Online]. Available: http://arxiv.org/abs/2012.03107

[10] S. Sinha, A. Garg, and H. Larochelle, “Curriculum by smoothing,” in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.

[11] M. P. Kumar, B. Packer, and D. Koller, “Self-paced learning for latent variable models,” in *Proceedings of the 23rd International Conference on Neural Information Processing Systems-Volume 1*, 2010, pp. 1189–1197.

[12] H. Li and M. Gong, “Self-paced convolutional neural networks,” in *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 2017, pp. 2110–2116.

[13] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” in *International Conference on Learning Representations*, 2018. [Online]. Available: http://arxiv.org/abs/1706.06083

[14] J. Zhang, X. Xu, B. Han, G. Niu, L. Cui, M. Sugiyama, and M. Kankanhalli, “Attacks which do not kill training make adversarial learning stronger,” in *International conference on machine learning*. PMLR, 2020.

[15] A. Raghunathan, S. M. Xie, F. Yang, J. C. Duchi, and P. Liang, “Adversarial training can hurt generalization,” in *International Conference on Machine Learning 2019 - Workshop Deep Phenomena*, 2019.

[16] G. Peterson, “A day of great illumination: B. f. skinner’s discovery of shaping,” *Journal of the experimental analysis of behavior*, vol. 82, pp. 317–28, 12 2004.

[17] J. L. Elman, “Learning and development in neural networks: The importance of starting small,” *Cognition*, vol. 48, no. 1, pp. 71–99, 1993.

[18] C. Gong, D. Tao, S. J. Maybank, W. Liu, G. Kang, and J. Yang, “Multimodal curriculum learning for semi-supervised image classification,” *IEEE Transactions on Image Processing*, vol. 25, no. 7, pp. 3249–3260, 2016.

[19] G. Hacohen and D. Weinshall, “On the power of curriculum learning in training deep networks,” in *International Conference on Machine Learning*. PMLR, 2019, pp. 2535–2544.

[20] S. Jastrzebski, M. Szymczak, S. Fort, D. Arpit, J. Tabor, K. Cho, and K. Geras, “The break-even point on optimization trajectories of deep neural networks,” in *International Conference on Learning Representations*, 2019.

[21] A. Achille, M. Rower, and S. Soatto, “Critical learning periods in deep networks,” in *International Conference on Learning Representations*, 2019. [Online]. Available: http://arxiv.org/abs/1711.08856

[22] I. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” in *International Conference on Learning Representations*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6572

[23] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, “Robustness may be at odds with accuracy,” 2019.
[24] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *International conference on machine learning*. PMLR, 2007, pp. 473–480.

[25] M. Zöhrer and F. Pernkopf, “General stochastic networks for classification,” *Advances in Neural Information Processing Systems*, vol. 27, pp. 2015–2023, 2014.