Being Friends Instead of Adversaries: Deep Networks Learn from Data Simplified by Other Networks

Simone Marullo[1,2], **Matteo Tiezzi**[2], **Marco Gori**[2,3], **Stefano Melacci**[2]

[1] Dept. of Information Engineering, University of Florence (Italy)
[2] Dept. of Information Engineering and Mathematics, University of Siena (Italy)
[3] MAASAI, Universitè Côte d'Azur, Nice (France)
simone.marullo@unifi.it, {mtiezzi,marco,mela}@diism.unisi.it

Abstract

Amongst a variety of approaches aimed at making the learning procedure of neural networks more effective, the scientific community developed strategies to order the examples according to their estimated complexity, to distil knowledge from larger networks, or to exploit the principles behind adversarial machine learning. A different idea has been recently proposed, named Friendly Training, which consists in altering the input data by adding an automatically estimated perturbation, with the goal of facilitating the learning process of a neural classifier. The transformation progressively fades-out as long as training proceeds, until it completely vanishes. In this work we revisit and extend this idea, introducing a radically different and novel approach inspired by the effectiveness of neural generators in the context of Adversarial Machine Learning. We propose an auxiliary multi-layer network that is responsible of altering the input data to make them easier to be handled by the classifier at the current stage of the training procedure. The auxiliary network is trained jointly with the neural classifier, thus intrinsically increasing the "depth" of the classifier, and it is expected to spot general regularities in the data alteration process. The effect of the auxiliary network is progressively reduced up to the end of training, when it is fully dropped and the classifier is deployed for applications. We refer to this approach as Neural Friendly Training. An extended experimental procedure involving several datasets and different neural architectures shows that Neural Friendly Training overcomes the originally proposed Friendly Training technique, improving the generalization of the classifier, especially in the case of noisy data.

1 Introduction

In the last decade, the scientific research in neural networks studied different aspects of the training procedure, leading to deep neural models of significantly increased quality (Ioffe and Szegedy 2015; Kingma and Ba 2015; Srivastava et al. 2014; Bengio et al. 2009; Li and Gong 2017; Zhang et al. 2020). Amongst a large variety of approaches, this paper considers those that are mostly oriented in performing specific actions on the available training data in order to improve the quality of the trained neural classifier. For example, Curriculum Learning (CL) pursues the idea of presenting the training data in a more efficient manner (Bengio et al. 2009; Wu, Dyer, and Neyshabur 2020; Sinha, Garg, and Larochelle 2020), exposing the network to simple, easily-discernible examples at first, and to gradually harder examples later, progressively increasing the size of the training set (Elman 1993). Self-Paced Learning (SPL) (Kumar, Packer, and Koller 2010; Li and Gong 2017) is another related research area, in which some examples are either excluded from the training set or their impact in the risk function is downplayed if some conditions are met (Li and Gong 2017).

A common property of CL and SPL is that they essentially sub-select or re-order the training examples, without altering the contents of the data. However, more recently, researches considered approaches that perform transformations of the input data within the input space of the classifier. Friendly Training (FT) (Marullo et al. 2021) is a novel approach belonging to the latter category. FT allows the training procedure not only to adapt the weights and biases of the classifier, but also to transform the training data in order to facilitate the early fulfilment of the learning criterion. Basically, data are modified to better accommodate the development of the classifier. Such transformations (also referred to as "simplifications") are controlled and embedded into a precise developmental plan in which the training procedure is progressively constrained to reduce their extent, until data are left in their original version. A key property of FT is that data are altered according to the state of the classifier at the considered stage of the training procedure, and each example is perturbed by a specific offset, obtained by an inner iterative optimization procedure that is started from scratch for each input. Similarly to CL, the benefits of FT are expected to be mostly evident in the case of noisy examples or in datasets annotated with noisy labels. These are pretty common situations of every data collection process of the real-world. In the case of CL, this has been recently discussed and evaluated in (Wu, Dyer, and Neyshabur 2020), while in the case of FT the existing evaluation is limited to artificial datasets for digit recognition (Marullo et al. 2021).

In this paper we revisit and extend the idea of FT, introducing a radically different and novel approach. The intuition behind what we propose is that the data simplification process of FT might include regularities that are shared among different training examples, and that there is an intrinsic coherence in the way data are altered in consecutive training iterations, i.e., similar simplifications might be fine

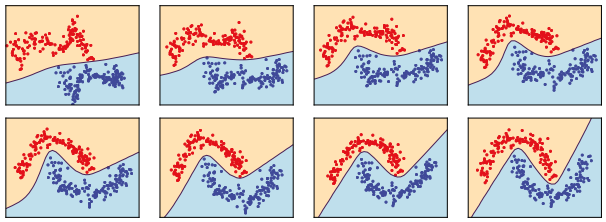


Figure 1: Left-to-right, top-to-bottom: evolution of the decision boundary developed by a single hidden layer classifier (5 neurons) in the 2-moon dataset, in Neural Friendly Training. Each plot is about a different training iteration (γ); in the last plot data are not transformed anymore.

in nearby stages of the training procedure. These considerations are not exploited by FT, which applies an independent perturbation to each example, estimated from scratch at each training step. We propose to introduce an auxiliary multilayer network, that is responsible of altering data belonging to the input space of the classifier. The auxiliary network is trained jointly with the neural classifier, and it learns how to transform the data to improve the learning process of the classifier itself. The weights of the auxiliary net represent the state of the alteration model, that is progressively updated by the training procedure, thus letting the model evolve as long as time passes. From an architectural perspective, the auxiliary network extends the classifier by adding a new set of initial layers, thus increasing the "depth" of the model. The effect of the auxiliary network is progressively reduced until the end of training, when it is fully dropped and the classifier is deployed for applications. We refer to this approach as Neural Friendly Training (NFT), and Fig. 1 illustrates the behaviour of NFT in a toy 2D classification problem.

Neural models to alter data samples have been proficiently exploited by the Adversarial Machine Learning community (Qiu et al. 2020; Xiao et al. 2018) with the goal of fooling a classifier. When considering how to improve a classifier exploiting another network, it is immediate to trace a connection also with Knowledge Distillation (KD) (Hinton, Vinyals, and Dean 2015; Phuong and Lampert 2019), although in KD the main network is supplied with output probability distributions obtained from a pretrained large model. The auxiliary network of NFT learns to transform the input data, closer to what is done by Spatial Transformer Networks (Jaderberg et al. 2015) (STN). However, STNs deal with image data only and estimate the parameters of a spatial transformation from a pre-defined family.

The contributions of this paper are: (1) we propose a novel training strategy that allows the machine to simplify the training data by means of an auxiliary network that progressively fades out; (2) we extend the experimental analysis of the original FT to non-artificial data, and (3) we experimentally compare it with the proposed NFT approach, using convolutional and fully connected neural architectures with different numbers of layers. Our results confirm that NFT outperforms FT, proving that NFT is a feasible and effective way to improve the generalization skills of the network and to efficiently deal with noisy training data.

2 Neural 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.[1] 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 empirical risk L measures the mismatch between predictions and the ground truths,

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

where $\mathcal{B} \subset \mathcal{X}$ is a mini-batch of data of size $|\mathcal{B}| \geq 1$, $(x_i, y_i) \in \mathcal{B}$, and ℓ is the loss function. Notice that, while we are aggregating the contributes of ℓ by averaging over the mini-batch data, every other normalization is fully compatible with what we propose. In the most common case of stochastic gradient optimization, a set of non-overlapping mini-batches is randomly sampled at each training epoch, in order to cover the whole set $\mathcal{X}$. We will refer to what we described so far as Classic Training (CT).

Friendly Training. CT provides data to the machine independently on the state of the network and on the information carried by the examples in each $\mathcal{B}$. However, data in $\mathcal{X}$ might include heterogeneous examples with different properties. For instance, their distribution could be multi-modal, it might include outliers or it could span over several disjoint manifolds, and so on and so forth. Existing results in the context of CL (Bengio et al. 2009; Wu, Dyer, and Neyshabur 2020) and SPL (Li and Gong 2017) (Section 1) show that it might be useful to provide the network with examples whose level of complexity progressively increases as long as learning proceeds. However, it is very unlikely to have information on the difficulty of the training examples and, more importantly, if the complexity is determined by humans it might not match the intrinsic difficulty that the machine will face in processing such examples. Alternatively, the value ℓ could be used as an indicator to estimate the difficulty of the data, to exclude the examples with largest loss values or to reduce their contribution in Eq. (1), more closely related to SPL (Kumar, Packer, and Koller 2010; Li and Gong 2017).

Differently from the aforementioned approach, Friendly Training (FT) (Marullo et al. 2021) *transforms* the training examples according to the state of the learner, with the aim of discarding the parts of information that are too complex to be handled by the network with the current weights, while preserving what sounds more coherent with the expectations of the current classifier.[2] FT consists in alternating two distinct optimization phases, that are iterated multiple times. In

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

[2] This is significantly different from deciding whether or not to keep a training example, to weigh its contribute in Eq. (1), or to re-

the first phase, the training data are transformed in order to make them more easily manageable by the current network. The training procedure must determine how data should be simplified according to the way the current network behaves. In the second phase, the network is updated as in CT, but exploiting the simplified data instead of the original ones. The whole procedure is framed in the context of a developmental plan in which the amount of the alteration is progressively reduced as long as time passes, until it completely vanishes. This is inspired by the basic principle of strongly simplifying the data during the early stages of life of the classifier, in order to favour its development, while the extent of transformation is reduced when the classifier improves its skills. Clearly, to deploy a trained classifier that does not rely on altered data, the impact of the simplification must vanish during the training process, exposing the classifier to the original training data after a certain number of steps. Formally, FT perturbs the training data by estimating the variation δ_i,

$$\tilde{x}_i = x_i + \delta_i, \tag{2}$$

for each example x_i. Such estimation is repeated from scratch for each training example, and at each training epoch. The terms δ_i's are obtained with the goal of minimizing L in Eq. (1), replacing x_i with $\tilde{x}_i$ of Eq. (2). Determining an accurate δ_i might require an iterative optimization procedure, and a maximum number of iterations is defined to control the strength of the perturbation, progressively reduced as long as training proceeds. [3]

Neural Friendly Training. Despite the novel view introduced by FT, the instance of (Marullo et al. 2021) is mostly inspired by the basic tools used in the context of Adversarial Training (Zhang et al. 2020), with a perturbation model that requires a per-example independent optimization procedure. Here we propose to instantiate FT in a different manner, by considering that there might be some regularities in the way data samples are simplified. This leads to the introduction of a more structured transformation function that is shared by all the examples. This intuition is also motivated by recent studies in Adversarial Machine Learning that exploited perturbation models based on generative networks (Qiu et al. 2020; Xiao et al. 2018), although with the goal of fooling a classifier. Formally, a training sample $x_i \in \mathbb{R}^d$ is transformed into $\tilde{x}_i \in \mathbb{R}^d$ by means of the function $s(x_i, \theta)$,

$$\tilde{x}_i = s(x_i, \theta), \tag{3}$$

being θ a set of learnable parameters, shared by all the examples. We consider the case in which s is implemented with an additional neural network, also referred to as *auxiliary network*, whose weights and biases are collected in θ, and we talk about Neural Friendly Training (NFT). For convenience in the notation, we keep the definition of δ_i inherited from Eq. (2), i.e., $\delta_i = \tilde{x}_i - x_i$. The term *main network* refers to the network that implements f, i.e., the classifier, and we report in Fig. 2 a sketch of the proposed model.

order the examples. Interestingly, FT is compatible with (and not necessarily an alternative to) such existing strategies.

[3]Further details are available in (Marullo et al. 2021).

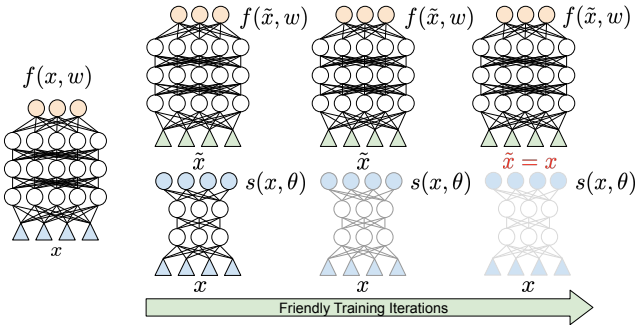


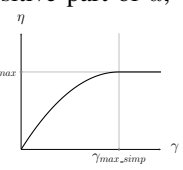
Figure 2: (a) Classic deep network. (b) Neural Friendly Training (NFT): main deep network (top) and auxiliary network (bottom). The auxiliary network learns how to simplify the data x, while the main network learns the classification task exploiting the simplified data $\tilde{x}$. As long as training proceeds, the effect of the auxiliary network is progressively reduced, until it vanishes (and it is removed).

In order to setup a valid developmental plan, we introduce an augmented criterion by re-defining the risk L of Eq. (1),

$$L(\mathcal{B}, w, \theta) = \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \Bigg(\ell\big(f(\underbrace{s(x_i, \theta)}_{\tilde{x}_i}, w), y_i\big) + \eta \big\| \underbrace{s(x_i, \theta) - x_i}_{\delta_i} \big\|^2 \Bigg), \tag{4}$$

where $(x_i, y_i) \in \mathcal{B}$, and $\eta > 0$ is the weight of the squared Euclidean norm of the perturbation δ_i. We indicate with $\gamma \geq 1$ the NFT iteration index, where each iteration consists of the two aforementioned phases. In the first phase, the auxiliary network is updated by minimizing Eq. (4) with respect to θ, keeping the main network fixed. In the second phase, the auxiliary network has the sole role of transforming the data, while the main network is updated by minimizing Eq. (4) with respect to w. If all the training data is used in this phase, then γ boils down to the epoch index (that is the case we considered in the experiments). If γ_{max} is the maximum number of NFT iterations, we ensure that after $\gamma_{max\_simp} < \gamma_{max}$ steps the data are not perturbed anymore. In order to progressively reduce the perturbation level, we increase the value of η in Eq. (4). For a large η, NFT will strongly penalize the norm of δ_i, becoming the dominant term in the optimization process of the auxiliary network, enforcing the net to keep δ_i small. We indicate with η_{max} the maximum possible value of η, and at each step γ of the developmental process we compute η using the following law, being $[a]_+$ the positive part of a,

$$\eta = \eta_{max} \left(1 - \left[1 - \frac{\gamma - 1}{\gamma_{max\_simp} - 1} \right]_+^2 \right) \tag{5}$$



where $\eta \in [0, \eta_{max}]$. At γ_{max_simp} iterations, the penalty on $\|\delta_i\|^2$ will reach its maximum weighting. While this

enforces the function $s(\cdot, \theta)$ to get closer to the identity function, we have no formal guarantees that it will effectively push the perturbation to zero. For this reason, after γ_{max_simp} iterations we drop the auxiliary network, exposing the system to the original training data. The developmental plan on η favours a smooth transition between the setting in which the auxiliary network is used and when it is removed.

The training procedure is detailed in Algorithm 1, and

Algorithm 1: Neural Friendly Training.

Input: Training set $\mathcal{X}$, initial weights and biases w, batch size b, max FT steps γ_{max}, max simplification steps γ_{max_simp}, $\eta_{max} > 0$, learning rates $\alpha > 0$ and $\beta > 0$.
Output: The final w.

1: **for** $\gamma = 1$ to γ_{max} **do**
2: Compute η following Eq. (5)
3: **if** $\gamma > 1$ **and** $\gamma \leq \gamma_{max\_simp}$ **then**
4: $s \leftarrow \texttt{auxiliary\_net}(\cdot, \theta)$
5: Sample a set of minibatches $B = \{\mathcal{B}_z\}$ from $\mathcal{X}$
6: **for** each mini-batch $\mathcal{B}_z \in B$ **do**
7: Compute $\nabla_\theta = \frac{\partial L(\mathcal{B}_z, w, h)}{\partial h}\Big|_{h=\theta}$, see Eq. (4)
8: $\theta = \theta - \beta \cdot \nabla_\theta$
9: **end for**
10: **else**
11: $s \leftarrow I(\cdot)$
12: **end if**
13: Sample a set of minibatches $B = \{\mathcal{B}_z\}$ from $\mathcal{X}$
14: **for** each mini-batch $\mathcal{B}_z \in B$ **do**
15: Compute $\nabla_w = \frac{\partial L(\mathcal{B}_z, h, \theta)}{\partial h}\Big|_{h=w}$, see Eq. (4)
16: $w = w - \alpha \cdot \nabla_w$
17: **end for**
18: **end for**
19: **return** w

FIRST PHASE: update $s(\cdot, \theta)$

SECOND PHASE: update $f(\cdot, w)$

in the following lines we provide some further details. The auxiliary network is not updated during the first iteration ($\gamma = 1$), since the main network is still in its initial/random state. After γ_{max_simp} iterations, the auxiliary network is replaced by the identity function $I(\cdot)$ (line 10). Notice that the weight update equations (line 8 and line 16) can include any existing adaptive learning rate estimation procedures, and in our current implementation we are using the Adam optimizer with learning rates α and β (Kingma and Ba 2015), unless differently stated. 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 (bypassing the auxiliary net at inference time).

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 classes, 300 examples), optimized by Adam with mini-batch of size 64. The auxiliary network alters the training data (from the popular 2-moon problem) in order to make them almost linearly separable during the early iterations. Then, the data distribution progressively moves toward the original configuration, and the decision boundary of the main classifier smoothly follows the data. In the last plot, the auxiliary network has been dropped and examples are located at their original positions in final stages of developmental plan.

Of course, NFT increases the complexity of each training step, due to the extra projection computed by the auxiliary network in the forward stage of the classifier and due to the first phase of Algorithm 1. The actual additional computational burden of NFT with respect to CT depends on the architecture of the auxiliary network and on the number of sampled mini-batches. Moreover, instead of Eq. 5, different developmental plans could be selected to more quickly reduce the simplification and eventually drop the auxiliary network before the end of training, even if investigating these factors goes beyond the scope of this paper. When comparing NFT and FT we can see that, from the storage point of view, NFT needs to memorize a new network and the associated intermediate variables for optimization purposes, while FT only requires a new set of variables to store the delta terms. However, from the computational point of view, for each example x_i, FT performs $\tau \geq 1$ iterations to update the perturbation δ_i, that implies τ inference steps on the main network (see Algorithm 1 of Marullo et al. (2021)). Differently, NFT does not require any inner example-wise iterative procedures (Algorithm 1, first phase). The inference time in the auxiliary network determines the concrete variations in terms of computational times with respect to FT. In our experience, on average, training with NFT took similar times to the ones of FT, since τ (in FT) gets reduced as time passes and we early stopped the inner FT iterations as suggested in (Marullo et al. 2021).

3 Experiments

We carried out a detailed experimental activity aimed at evaluating how NFT behaves when compared to FT. We considered the same experimental conditions of (Marullo et al. 2021), initially using the same datasets (Section 3.1), and then we focused on novel experiences (textual data, Section 3.2, pictures of vehicles and animals, Section 3.3), where FT was never tested before. We also performed an in-depth analysis on NFT (Section 3.4).

We considered the same four neural classifiers that were used in (Marullo et al. 2021),[4] that consist in two feed-forward Fully-Connected multi-layer perceptrons, referred to as FC-A and FC-B, two Convolutional Neural Networks, named CNN-A and CNN-B, and we also tested a ResNet18 (He et al. 2016) in one of the following experiences, motivated by related work (Wu, Dyer, and Neyshabur 2020).[5]

[4]Code available at https://sailab.diism.unisi.it/friendly.

[5]FC-A is a simple one-hidden-layer network with hyperbolic tangent activations (10 hidden neurons), while FC-B is deeper and larger model, with 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, while CNN-B is deeper (4 convolutional layers). Both of them exploit ReLU activation functions on the convolu-

The auxiliary network was selected depending on the type of data that it is expected to simplify. The output layer has the same size of the input one and linear activation. In the case of image data (Section 3.1, 3.3), the auxiliary network is inspired by U-Net (Ronneberger, Fischer, and Brox 2015). U-Net progressively down-samples the image, encoding the context information into the convolutional feature maps, and then it up-samples and transforms the data until it matches the input size, also exploiting skip connections.[6] In the case of 1-dim data (Section 3.2) we used a fully-connected auxiliary net with 256 hidden neurons.

In all the experiments, networks were randomly initialized, providing the exact same initialization to both FT/NFT and CT, and we report results averaged over 3 runs, corresponding to 3 different instances of the initialization process. For each FT/NFT iteration, we sampled non-overlapping mini-batches until all the training data were considered, so that γ is also the epoch index. We selected a large number of epochs γ_{max} which we found to be sufficient to obtain a stable configuration of the weights in preliminary experiences (detailed below), and the reported metrics are about the model with the lowest validation error obtained during training. The error rate was selected as the main metric, since it is one of the most common and simple measure in classification problems. We performed some preliminary experiments to determine the optimal Adam learning rate in the case of CT. Then, we tuned the FT hyper-parameters (η_{max}, γ_{max_simp}, β, n_f) by grid search (detailed below). We experimented on two machines equipped with NVIDIA GeForce RTX 3090 (24GB) GPUs.

3.1 Advanced Digit and Shape Recognition

The collection of datasets presented in (Larochelle et al. 2007) is about 10-class digit recognition problems and shape-based binary classification tasks (28×28, grayscale). In detail, MNIST-ROT consists of MNIST digits rotated by a random angle, while MNIST-BACK-IMAGE features non-uniform backgrounds extracted by some random images, and MNIST-ROT-BACK-IMAGE combines the factors of variations of the first two datasets. In RECTANGLES-IMAGE we find representations of rectangles, that might be wide or tall, with the inner and outer regions eventually filled with patches taken from other images, while CONVEX is about convex or non-convex white regions on a black background. Datasets ($\approx$ 60k samples) are already divided into training, validation and test set . We compared the test error rates of the FC-A/B and CNN-A/B models in CT, FT/NFT, and also using the CL-inspired data sorting policy of (Marullo et al. 2021), named Easy-Examples First (EEF) that has the same temporal dynamics of FT. Experiments are executed for $\gamma_{max} = 200$ epochs, and we selected the model with the lowest validation error considering $\eta_{max} \in \{500, 1000, 2000\}$, $\gamma_{max\_simp} \in \{0.25, 0.5, 0.85\} \cdot \gamma_{max}$, $\beta \in \{10^{-5}, 10^{-4}, 5 \cdot 10^{-4}\}$.

Table 1 reports the test error rate of the different models, where other baseline results exploiting different types of classifier can be found in (Marullo et al. 2021) (typically overcame by FT/NFT). Our analysis starts by confirming that the family of Friendly Training algorithms (being them neural or not) very frequently shows better results than CT and EEF. Moreover, the proposed NFT almost always improves the results of FT, supporting the idea of using an auxiliary network to capture regularities in the simplification process. In the case of CNN-A and CNN-B, the error rate of NFT is lower than in FT, with the exception of RECTANGLES-IMAGE, where, however, NFT reported a pretty large standard deviation. In fully-connected architectures FC-A and FC-B, we still observe a positive impact of NFT, that usually beats FT. However, the improvement over CT can be appreciated in a less evident or more sparse manner. As a matter of fact, these architectures are less appropriate than CNNs to handle image data. However, it is still interesting to see how FC-B benefits from the auxiliary network introduced in NFT, that is indeed a convolutional architecture. Overall, results show that using an auxiliary network is better than independently estimating the perturbation offsets of each example, confirming the capability of the network to learn shared facets of the simplification process.

		mn-back	mn-rot-back	mn-rot	rectangles	convex
FC-A	CT	28.34±0.09	64.06±0.31	43.16±0.51	24.31±0.21	33.91±0.44
	EEF	**28.18**±0.47	64.27±0.19	43.91±0.73	24.48±0.11	**33.17**±0.93
	FT	28.66±0.06	64.14±0.36	43.24±0.43	24.64±0.37	34.38±0.22
	NFT	**28.15**±0.04	64.55±0.14	**42.96**±0.58	24.57±0.19	34.25±1.03
FC-B	CT	21.06±0.39	51.71±0.79	10.13±0.27	25.10±0.20	27.24±0.05
	EEF	21.38±0.18	52.95±0.63	**10.04**±0.17	**24.84**±0.32	28.21±0.96
	FT	21.74±0.26	**51.02**±0.07	11.19±0.37	**24.14**±0.53	27.49±0.07
	NFT	**20.91**±0.52	**50.20**±0.16	**10.09**±0.32	**25.09**±0.09	**26.81**±0.15
CNN-A	CT	7.25±0.16	29.05±0.45	7.48±0.14	9.86±0.32	8.24±0.09
	EEF	**7.02**±0.08	29.12±0.34	7.61±0.22	12.82±0.70	8.72±0.74
	FT	**6.80**±0.19	**28.74**±0.29	**7.36**±0.06	**9.72**±0.20	8.59±1.44
	NFT	**6.59**±0.09	**28.67**±0.35	**7.17**±0.17	10.99±1.89	**8.03**±0.23
CNN-B	CT	5.15±0.15	23.05±0.21	6.58±0.06	8.10±1.90	3.01±0.41
	EEF	**4.82**±0.19	**22.89**±0.49	7.02±0.28	8.35±1.01	3.75±0.58
	FT	**5.03**±0.11	**22.81**±0.36	6.95±0.12	**7.32**±1.31	**2.87**±0.42
	NFT	**4.96**±0.34	**22.22**±0.62	**6.48**±0.25	**6.27**±0.62	**2.78**±0.34

Table 1: Comparison of different classifiers (FC-A, FC-B, CNN-A, CNN-B) and learning algorithms (CT, EEF, FT from (Marullo et al. 2021) and our NFT) – datasets of Section 3.1 (where MN stands for MNIST and removing the suffix IMAGE). Test error and standard deviation over 3 runs are reported. For each architecture, those results that improve the CT case are in bold.

tional feature maps (32-64 filters in CNN-A, 32-48-64-64 filters in CNN-B) and on the fully connected layers (9216-128 neurons for CNN-A, 5184-128 neurons for CNN-B). Unless differently stated, learning of weights and biases is driven by the minimization of the cross-entropy loss, exploiting the Adam optimizer (Kingma and Ba 2015) with mini-batches of size 32.

[6]Code: https://github.com/milesial/Pytorch-UNet. In the down-sampling part, 2 initial conv. layers encode the image into n_f feature maps. Then, ν down-sampling blocks (each of them composed of maxpooling and 2 conv. layers) are followed by ν up-sampling blocks (each of them composed of bilinear upscaling and 2 conv. layers). We considered $\nu \in \{1, 2\}$, and $n_f \in \{64, 96, 128\}$.

3.2 Sentiment Analysis

We investigate how NFT behaves in Natural Language Processing considering the task of Sentiment Analysis (positive/negative polarity). We selected two datasets and considered different representations of the examples. The first dataset is IMDB (Maas et al. 2011), also known as Large Movie Review Dataset, that is a collection of 50k highly-polar reviews from the IMDB database. We considered a vocabulary of the most frequent 20k words and TF-IDF (Jones 1972) representation of each review. The second dataset, WINES (Thoutt 2017), collects 130k wine reviews scored in $[80, 100]$, that we divided into two classes, i.e., $[80, 90)$ vs. $[90, 100]$. In this case, in order to acquire a broader outlook on the effect of NFT, we chose a different text representation, exploiting a pretrained Transformer-based architecture (DistilRoBERTa (Reimers and Gurevych 2019), with average pooling to compute dense representations of size 768 for each review. We trained the deeper fully-connected architecture, FC-B, for 30 epochs. NFT hyper-parameters were selected in $\eta_{max} \in \{10, 100, 500, 1000, 2000\}$, $\gamma_{max\_simp} \in \{0.05, 0.1, 0.25, 0.85, 0.5\} \cdot \gamma_{max}$, $\beta \in \{10^{-5}, 10^{-4}, 5 \cdot 10^{-4}\}$. Concerning FT, we extended the grids of (Marullo et al. 2021) for all the new experiences of this paper, testing further parameter configurations (supplementary material at https://sailab.diism.unisi.it/friendly).

As reported in Table 2 (top), the performance of CT is consistently improved by NFT, achieving lower error rates in both the datasets (and representations). The sentence classification task appears to be slightly more difficult in WINES. This is probably due to the fact that wine reviews are less polarized, being them all highly scored. Concerning IMDB, the superiority of NFT over CT and also FT is evident. Overall, these results confirm the versatility of NFT.

	imdb		wines
FC-B CT	13.27 $\pm$0.19	FC-B CT	17.38 $\pm$0.15
FC-B FT	13.66 $\pm$0.69	FC-B FT	**17.07** $\pm$0.11
FC-B NFT	**11.93** $\pm$0.09	FC-B NFT	**17.15** $\pm$0.12
	cifar-10		**cifar-10-n10**
CNN-B CT	29.75 $\pm$0.37	ResNet CT	9.30 $\pm$0.16
CNN-B FT	30.19 $\pm$0.53	ResNet FT	**8.92** $\pm$0.23
CNN-B NFT	**29.00** $\pm$0.36	ResNet NFT	**8.10** $\pm$0.19

Table 2: Comparison of classifiers with different architectures and learning algorithms (CT, FT) – data of Section 3.2 (top) and Section 3.3 (bottom). Mean test error is reported with standard deviation. Results improving CT are in bold.

3.3 Image Classification

CIFAR-10 (Krizhevsky 2009) is a popular Image Classification dataset, consisting of 60k 32×32 color images from 10 different classes. We divided the original training data into training and validation sets (10k examples used as validation set), and we initially evaluated NFT using the previously described generic CNN-B architecture. Table 2 (bottom-left) shows that while we are not able to improve the results of CT using FT, NFT slightly improves the quality of the network, reducing the error rate and further confirming its benefits.

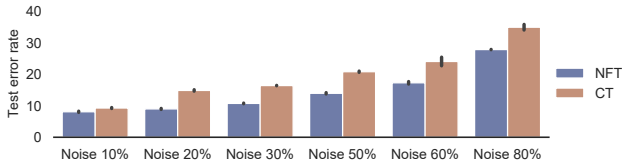


Figure 3: ResNet18 on CIFAR-10 dataset for different amounts of noisy labels. Error bars include standard dev.

However, state-of-the art convolutional networks specifically designed/tuned for CIFAR-10 usually achieve lower error rates, so that we decided to perform a more specific experimental activity. In particular, we considered ResNet18 (He et al. 2016), inheriting all the carefully selected optimization parameters and tricks that yield state-of-the-art results in CIFAR-10.[7] Since FT/NFT bring marginal benefits over CT, we designed a more challenging condition following the setup of recently published CL activity (Wu, Dyer, and Neyshabur 2020). We introduced some noise by randomly permuting 10% of the target labels, generating what we will refer to as CIFAR-10-N10. We trained the network for 250 epochs, and reported results in Table 2 (bottom-right). NFT hyper-parameters were selected in $\eta_{max} \in \{500, 1000, 2000\}$, $\gamma_{max\_simp} \in \{0.25, 0.5, 0.7\} \cdot \gamma_{max}$, $\beta \in \{10^{-4}, 5 \cdot 10^{-4}\}$. The learning rate scheduler is applied starting from γ_{max_simp} with an initial learning rate which is $0.1 \cdot \alpha$. We observe that NFT effectively helps also when dealing with this type of network. While FT also carries a small improvement, it is far from the one obtained by NFT. We further investigated this result by varying the amount of noise injected into the training labels. Fig. 3 compares CT and NFT for different noise levels, up to 80%. Interestingly, the impact of NFT becomes more and more evident, gaining $\approx 8\%$ in strongly noisy environments, confirming that data simplification helps the main network to better discard the noisy information.

3.4 In-Depth Analysis

We qualitatively compared NFT and FT in the MNIST-BACK-IMAGE dataset of Section 3.1, in which the important information is known (the digits), since the background is uncorrelated with the target. We mostly considered the CNN-A model, for which NFT led to the most significant improvements with respect to CT (Table 1). In Fig. 4 we show how examples are affected when using an auxiliary network (bottom - NFT) or when independent transformations are estimated for each example through a gradient-based procedure (top - FT). Estimating the transformation function with a neural model leads to qualitatively different behavior. We observe that FT yields structured perturbations

[7] Stochastic Gradient Descent (learning rate 0.1 with cosine annealing learning rate scheduler) with momentum (0.9) and weight decay ($5 \cdot 10^{-4}$), mini-batches of size 128, data augmentation – see https://github.com/kuangliu/pytorch-cifar.

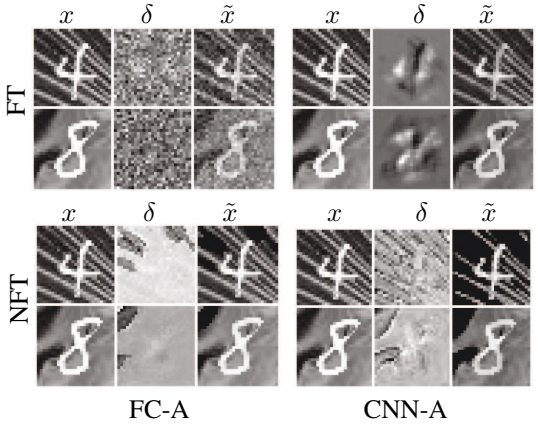


Figure 4: MNIST-BACK-IMAGE. Original data x , perturbation δ (normalized) and resulting "simplified" images $\tilde{x}$ for FC-A and CNN-A at the end of the 1st epoch. Some simplifications are hardly distinguishable. Top: FT. Bottom: NFT.

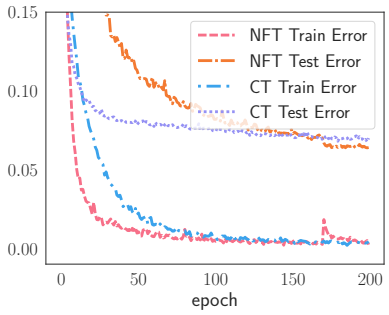


Figure 5: Training and test error rates for NFT and CT on a single run – MNIST-BACK-IMAGE, CNN-A (best viewed in colors). The auxiliary network is dropped at epoch 175. The training error of NFT is initially lower than in the case of CT since the auxiliary network simplifies the data. Differently, the test error is initially larger, since the test set is not simplified. As training proceeds, the simplification vanishes and the test data become aligned with the training ones.

only when paired with CNN-A, emphasizing the digit areas. Differently NFT shows more natural perturbation patterns, removing distracting cues (background). Basically, the convolutional auxiliary net leads to transformations with much more detailed awareness of the visual structures.

In Fig. 5, we report the evolution of test error rate during the training epochs (MNIST-BACK-IMAGE, CNN-A), comparing NFT and CT. The developmental plan reduces the impact of the perturbation until epoch 175 (afterwards, data are not altered anymore). The small bump right before such epoch is due to the final transition from altered to original data. The test error of NFT is higher than the one of CT when data are altered, as expected, while it becomes lower when the auxiliary network is dropped. On the other hand, fitting training data is easier during the early epochs in NFT, due to the simplification process.

We also evaluated the sensitivity of the system to some hyper-parameters of NFT, keeping the main network fixed. In Fig. 6, we report the test error of CNN-A, MNIST-BACK-IMAGE dataset, for different configurations of η_{max}, $\frac{\gamma_{max\_simp}}{\gamma_{max}}$, n_f, β. In particular, after having selected a sample run that is pretty representative of the general trend we observed in the experiments, we changed one of the aforementioned parameters and computed the error rate. Large values of η_{max} reduce the freedom of auxiliary network in learning the transformation function.

Similarly, a short developmental plan with a small $\frac{\gamma_{max\_simp}}{\gamma_{max}}$ does not allow the main network to benefit from the progressively simplified data. In general, we did not experience a very significant sensitivity to the variations of n_f, and 64 features turned out to be fine in most of the experiments, with some cases in which moving to 96 was slightly preferable, as in the one we are showing in Fig. 6. Although in a fine-grained grid of values, we found that larger β helped the auxiliary network to more quickly develop meaningful transformations. As a side note, we report that NFT was $\approx 1.5\times$ slower than CT, on average–see Sec. 2; performance optimization was outside the scope of this work.

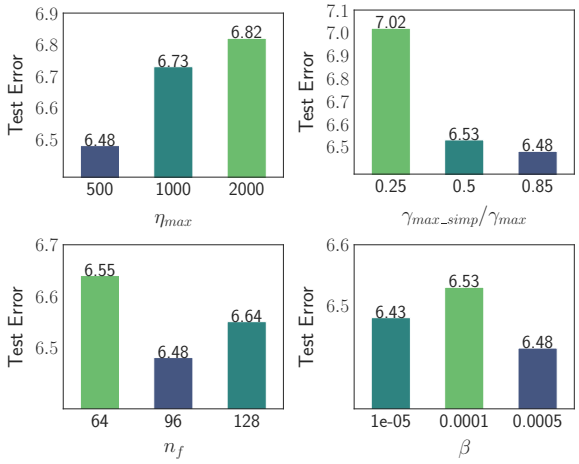


Figure 6: Test error under different configuration of the NFT hyper-parameters, CNN-A architecture.

4 Conclusions and Future Work

In this paper, we presented a novel approach to Friendly Training, according to which training data are altered by an auxiliary neural network in order to improve the learning procedure of a neural network-based classifier. Thanks to a progressive developmental plan, the classifier implicitly learns from examples that better match its current expectations, reducing the impact of difficult examples or noisy data during early training. The auxiliary neural network is dropped at the end of the training routine. An extensive experimental evaluation showed that Neural Friendly Training leads to classifiers with improved generalization skills, overcoming vanilla Friendly Training in which an example-wise perturbation is estimated in an iterative manner. Future work will focus on the investigation of different developmental plans and the evaluation of the impact of Neural Friendly Training in terms of robustness to adversarial examples.

Acknowledgements

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

References

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum Learning. In *Proc. of the International Conference on Machine Learning*, 41–48. ACM.

Elman, J. L. 1993. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1): 71–99.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 770–778.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the Knowledge in a Neural Network. In *NIPS Deep Learning and Representation Learning Workshop*.

Ioffe, S.; and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 448–456. PMLR.

Jaderberg, M.; Simonyan, K.; Zisserman, A.; and kavukcuoglu, k. 2015. Spatial Transformer Networks. In *Advances in Neural Information Processing Systems*, volume 28.

Jones, K. S. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28: 11–21.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

Krizhevsky, A. 2009. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Kumar, M. P.; Packer, B.; and Koller, D. 2010. Self-paced learning for latent variable models. In *Int. Conf. on Neural Information Processing Systems*, 1189–1197.

Larochelle, H.; Erhan, D.; Courville, A.; Bergstra, J.; and Bengio, Y. 2007. An empirical evaluation of deep architectures on problems with many factors of variation. In *International Conference on Machine Learning*, 473–480.

Li, H.; and Gong, M. 2017. Self-paced convolutional neural networks. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 2110–2116.

Maas, A. L.; Daly, R. E.; Pham, P. T.; Huang, D.; Ng, A. Y.; and Potts, C. 2011. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 142–150. Portland, Oregon, USA: Association for Computational Linguistics.

Marullo, S.; Tiezzi, M.; Gori, M.; and Melacci, S. 2021. Friendly Training: Neural Networks Can Adapt Data To Make Learning Easier. In *IEEE International Joint Conference on Neural Networks (IJCNN) (arXiv preprint arXiv:2106.10974)*.

Phuong, M.; and Lampert, C. 2019. Towards Understanding Knowledge Distillation. In *International Conference on Machine Learning*, volume 97, 5142–5151. PMLR.

Qiu, H.; Xiao, C.; Yang, L.; Yan, X.; Lee, H.; and Li, B. 2020. Semanticadv: Generating adversarial examples via attribute-conditioned image editing. In *ECCV*.

Reimers, N.; and Gurevych, I. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. ACL.

Ronneberger, O.; Fischer, P.; and Brox, T. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. volume 9351, 234–241. ISBN 978-3-319-24573-7.

Sinha, S.; Garg, A.; and Larochelle, H. 2020. Curriculum By Smoothing. In *Advances in Neural Information Processing Systems*.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56): 1929–1958.

Thoutt, Z. 2017. Wine Reviews: https://kaggle.com/zynicide/wine-reviews.

Wu, X.; Dyer, E.; and Neyshabur, B. 2020. When Do Curricula Work? *International Conference on Learning Representations*.

Xiao, C.; Li, B.; yan Zhu, J.; He, W.; Liu, M.; and Song, D. 2018. Generating Adversarial Examples with Adversarial Networks. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, 3905–3911. International Joint Conferences on Artificial Intelligence Organization.

Zhang, J.; Xu, X.; Han, B.; Niu, G.; Cui, L.; Sugiyama, M.; and Kankanhalli, M. 2020. Attacks Which Do Not Kill Training Make Adversarial Learning Stronger. In *International Conference on Machine Learning*.