Dropout Distillation

Samuel Rota Bulò ROTABULO@FBK.EU
FBK-irst, Trento, Italy

Lorenzo Porzi PORZI@FBK.EU
FBK-irst, Trento, Italy

Peter Kontschieder PKONTSCHIEDER@MAPILLARY.COM
Mapillary, Graz, Austria - Microsoft Research, Cambridge, UK

Abstract

Dropout is a popular stochastic regularization technique for deep neural networks that works by randomly dropping (*i.e.* zeroing) units from the network during training. This randomization process allows to implicitly train an ensemble of exponentially many networks sharing the same parametrization, which should be averaged at test time to deliver the final prediction. A typical workaround for this intractable averaging operation consists in scaling the layers undergoing dropout randomization. This simple rule called "standard dropout" is efficient, but might degrade the accuracy of the prediction. In this work we introduce a novel approach, coined "dropout distillation", that allows us to train a predictor in a way to better approximate the intractable, but preferable, averaging process, while keeping under control its computational efficiency. We are thus able to construct models that are as efficient as standard dropout, or even more efficient, while being more accurate. Experiments on standard benchmark datasets demonstrate the validity of our method, yielding consistent improvements over conventional dropout.

1. Introduction

Dropout is a popular regularization technique introduced in the context of neural networks that improves on the generalization capabilities of a predictor by preventing the co-adaptation of features (Hinton et al., 2012; Srivastava et al., 2014). The idea is to randomly set to zero (*a.k.a.* dropout) hidden and/or input units of a neural network during each iteration of the optimization algorithm in order to discourage the network from relying on single features, but rather consider the output of committees of features. This simple idea has been playing a pivotal role in the field for the last few years by effectively overcoming overfitting issues and pushing the performance of neural networks to new state-of-the-art levels in many application areas.

Since the introduction of dropout, several works in the literature have attempted to explain its success in avoiding overfitting. Some works supported the view of dropout performing stochastic gradient descent of some regularized loss function (Baldi & Sadowski, 2013), or as an adaptive L_2 regularizer (Wager et al., 2013), or more recently as an approximation to a well-known Bayesian model, namely the deep Gaussian process (Gal & Ghahramani, 2015a). Thanks to its regularizing effect, dropout contributes to reducing the inherent complexity, and thus the variance of the model it is applied to. This is formally stated in (Gao & Zhou, 2014), where the authors show that dropout delivers an exponential reduction in the Rademacher complexity of deep neural networks. While it is intuitive that dropout contributes to reducing the generalization error by keeping the variance of the model under control, it seems less obvious that dropout might also play a role in reducing the model bias. The latter property arises from the ability of dropout training to escape poor, local minima during the optimization of the empirical risk (Jain et al., 2015).

Dropout training intuitively trains an ensemble of exponentially many neural networks, one for each configuration of dropped units, while sharing the same parametrization. The goal is to minimize the expected empirical risk of the ensemble, which is however an intractable objective. Nevertheless, it can be effectively optimized via a stochastic gradient descent procedure, where the randomization process involves also dropping out units. At test time, the contribution from each network in the ensemble should be av-

Proceedings of the 33^{rd} International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48.

eraged to deliver the final prediction. This computation is again intractable and it is typically approximated by employing a simple heuristic, which introduces a scaling factor in each layer undergoing dropout regularization (Hinton et al., 2012). This workaround called *standard dropout* works in practice, but might suffer from a loss in the final accuracy. In (Gal & Ghahramani, 2015b), it is indeed empirically shown that improved results can be obtained by approximating the ideal averaging predictor via (computationally more intensive) Monte Carlo sampling, which gives a better control of the approximation quality.

Our work provides a novel approach, called *dropout distillation*, for inference with deep neural networks using dropout regularization. The goal is to mimic the ideal, but computationally intractable averaging predictor exhibiting improved accuracy, without sacrificing the computational efficiency of the simple scaling heuristic known from standard dropout. In our proposed solution we project the ideal predictor into a space of tractable/efficient predictors in a way to minimize a divergence measure, accounting for a specified loss function and respecting the underlying data distribution. To bypass the intractability of this objective, which stems from the necessity of evaluating the ideal predictor, we propose to adopt a stochastic optimization strategy akin to the one used for dropout training. Additionally, we characterize suitable loss functions for our approach.

The space of predictors we define for the projection allows us to control the efficiency of the final predictor, while the minimization of the divergence from the ideal predictor promotes improved accuracies. By focussing on models having the same complexity as the one trained with dropout, it is possible to preserve the computational efficiency of *e.g.* standard dropout, while aiming for better predictions. By specifying a space of smaller, more compressed models one can further reduce the computational burden. We demonstrate the effectiveness of our method in both scenarios, where improved predictors are obtained both in terms of computational efficiency and prediction accuracy compared to standard dropout.

Another aspect of our method with a practical application is that it allows us to improve any pre-trained model using dropout, without necessarily having access to the training data. Indeed, we simply require a collection of unlabelled data to carry out the dropout distillation procedure. Once the optimization is accomplished, the new and improved model can be directly deployed for future predictions.

As for the name of our approach, the word "distillation" has been used in the literature to indicate the task of transferring knowledge from some cumbersome model (typically an ensemble of deep networks) to a simpler one (Hinton et al., 2014). Since this task shares similarities with our approach, we have decided to term it *dropout distillation*. More details about this analogy will be given in paper.

The rest of the paper is organized as follows. Sec. 2 reviews dropout training and emphasizes on the objective that it actually optimizes. Sec. 3 is dedicated to the inference process, where we review two state-of-the-art approaches and highlight their limitations. Sec. 4 describes our proposed approach, coined dropout distillation. Sec. 5 assesses the validity of our method on several benchmark datasets, using different network architectures. Finally, we draw conclusions and discuss future works in Sec. 6.

2. Dropout training

In this section we discuss the basic idea of dropout training. Dropout has originally been introduced to train *feed-forward* neural networks (Hinton et al., 2012). For this reason, and for the sake of clarity, we will stick to the same type of architecture in the rest of the paper. However, our proposed method can also be applied to recurrent networks exploiting dropout (Pham et al., 2014).

Consider a feed-forward neural network with layers $\{h_1, \dots, h_n\}$. Each layer is a function defined as $h_i(\boldsymbol{z}) = a(\mathtt{\Theta}_i \boldsymbol{z})$, where $a()$ is an element-wise, non-linear *activation* function, which might be different across layers, and $\mathtt{\Theta}_i$ is a matrix of layer parameters. The composition of the layers defines the neural network $f_\Theta = h_n \circ \cdots \circ h_1 : \mathcal{X} \to \mathcal{Y}$, where $\Theta = \{\mathtt{\Theta}_1, \dots, \mathtt{\Theta}_n\}$ comprises all trainable weights, and $\mathcal{X}$, $\mathcal{Y}$ denote the input and output spaces, respectively.

By applying dropout training to a network f_Θ, one implicitly introduces *gating variables* in the model, allowing to switch on/off connections within the network. Depending on whether the gating variables act on the output of network units or on network connections, we have the classic dropout setting or the so-called *drop-connect* setting (Wan et al., 2013), respectively. In the dropout case, a gated layer takes the form $h_i(\boldsymbol{z}) = a(\mathtt{\Theta}_i(\boldsymbol{\sigma}_i \bullet \boldsymbol{z}))$, where $\boldsymbol{\sigma}_i$ is a vector of gating (*i.e.* binary) variables and $\bullet$ denotes the Hadamard (or element-wise) product. Consequently, the gating variables $\boldsymbol{\sigma}_i$ determine which input dimensions from the previous layer are taken into account during the computation. In the drop-connect case, a gated layer is given by $h_i(\boldsymbol{z}) = a((\Sigma_i \bullet \mathtt{\Theta}_i)\boldsymbol{z})$, where Σ_i is a matrix of gating variables. The latter form works at a finer granularity by allowing to discard contributions from specific connections. Indeed, dropout can be seen as an instance of drop-connect under a properly-structured sampling distribution. Note that the layer expressions for both cases of dropout and drop-connect take slightly different forms in case of convolutional layers, but we omit the details here, since they are not relevant for the purposes of this section.

We denote by $f_{\Theta,\sigma}$ a *gated* neural network parametrized by Θ with a configuration of gating variables given by

$\sigma = \{\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_n\}$.[1] Different configurations of gating variables generate different network topologies, for parts of the original network will be virtually eliminated, while Θ remains shared among all representable topologies.

The goal of dropout training is to find a parametrization Θ^* that minimizes the expected risk associated with the gated neural network $f_{\Theta,\sigma}$, where the expectation is not only taken with respect to the unknown joint data distribution over $(\boldsymbol{x}, y)$, but involves also the gating variables σ. Accordingly, the risk takes the following form:

$$R(\Theta) = \mathbb{E}_{\boldsymbol{x},y,\sigma}\left[\ell(y, f_{\Theta,\sigma}(\boldsymbol{x})\right] , \tag{1}$$

where $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is a loss function penalizing wrong predictions. The expectation with respect to σ is taken by considering each gating variable an independent Bernoulli variate with parameter $1 - p$, where p is regarded as the *dropout rate*. Typically, p is set to $\frac{1}{2}$ to maximize the entropy of the sampling distribution, but in general different gating variables in the network might have different dropout rates. Regarding the expectation over $(\boldsymbol{x}, y)$, this cannot be resolved, since the underlying distribution is unknown. Therefore, one resorts to *empirical* risk approximation, replacing $\mathbb{E}_{\boldsymbol{x},y}[\cdot]$ with an empirical average computed from independent training samples $\mathcal{T} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_\mathsf{n}, y_\mathsf{n})\}$:

$$\hat{R}(\Theta) = \frac{1}{\mathsf{n}} \sum_{i=1}^{\mathsf{n}} \mathbb{E}_\sigma\left[\ell(y_i, f_{\Theta,\sigma}(\boldsymbol{x}_i)\right] , \tag{2}$$

so in practice $\hat{R}(\Theta)$ is the actual objective that dropout training optimizes, which converges to $R(\Theta)$ for $\mathsf{n} \to \infty$.

Dropout training minimizes $\hat{R}(\Theta)$ via stochastic gradient descent, where the randomization is not limited to drawing mini-batches from the set of samples in $\mathcal{T}$, but also to selecting different gating variable configurations per training data (see Alg. 1).[2]

By appropriately decreasing the learning rates, denoted by η_t in Alg. 1, the sequence of network parameters $\{\Theta^0, \Theta^1, \ldots\}$ generated by dropout training converges almost surely to a local minimum of (2), akin to other forms of stochastic gradient descent (Bottou, 1998).

The use of dropout in deep neural networks is often restricted to very few layers, typically one or two close to the terminal one, instead of applying it to the entire network. One of the reasons is that longer training schemes are necessary if dropout is extensively used, since a much larger space of network topologies has to be explored by the stochastic optimization algorithm. Another current limitation is that it is not clear how to properly tackle the inference process, which we address in our paper.

[1] For simplicity reasons we discuss based on the dropout case, but the results also apply to drop-connect.

[2] One can also share the same configuration of gating variables across samples in the mini-batch for the sake of computational efficiency (Graham et al., 2015).

Algorithm 1 Dropout training

Input: Training set $\mathcal{T} \subset \mathcal{X} \times \mathcal{Y}$
$t = 0$
Initialize network parameters Θ^0
repeat
  Sample $\{(\boldsymbol{x}_{\pi_1}, y_{\pi_1}), \ldots, (\boldsymbol{x}_{\pi_\mathsf{m}}, y_{\pi_\mathsf{m}})\} \subset \mathcal{T}$
  Sample gating variable configurations $\sigma^1, \ldots, \sigma^\mathsf{m}$
  Update learning rate η_t
  $\Theta^{t+1} = \Theta^t - \frac{\eta_t}{\mathsf{m}} \sum_{j=1}^{\mathsf{m}} \frac{\partial}{\partial \Theta} \ell(y_{\pi_j}, f_{\Theta^t,\sigma^j}(\boldsymbol{x}_{\pi_j}))$
  $t = t + 1$
until stopping condition is met
Output: $\Theta^* = \Theta^t$

3. Dropout inference

Dropout training aims to minimize an intractable objective as described in the previous section. The intractability stems from the expectation spanning a number of network topologies, which is in general exponential in the number of network units, or network connections, depending on whether dropout or drop-connect is employed, respectively. Nevertheless, the stochastic gradient descent procedure that is adopted for the optimization, which considers random network topologies for each update, is viable in practice and delivers good parametrizations within reasonable training times, as was demonstrated by many works that have been successfully using dropout in the last years.

Besides training, another important aspect to consider when employing dropout is how to properly perform inference. As one may expect, the same intractability issue that is faced at training time persists also at test time. Indeed, the proper way of delivering a prediction for a test sample $\boldsymbol{x} \in \mathcal{X}$ given a network with parameters Θ^* trained with dropout is by averaging the prediction of all possible gated networks. In other terms, one ideally targets the following predictor that we call *ideal predictor*:

$$f_{\text{dropout}}(\boldsymbol{x}) = \mathbb{E}_\sigma\left[f_{\Theta^*,\sigma}(\boldsymbol{x})\right] . \tag{3}$$

Again, we find that the expectation poses in general an intractable computation. The rest of this section summarizes how state-of-the-art approaches are approximating (3), before we introduce our novel solution in Section 4.

3.1. Standard dropout

The standard approach to avoid evaluating different networks at test time has been proposed in the original dropout

papers (Hinton et al., 2012; Srivastava et al., 2014). The idea is to simply scale the argument of the activation function by $1-p$ in each layer where dropout has been applied during training with rate p, *i.e.* $h_i(\boldsymbol{z}) = a((1-p)\mathtt{\theta}_i\boldsymbol{z})$ and similarly for drop-connect. At test time, all units/connections are involved in the computation. Intuitively, the scaling factor compensates for the average number of missing units during training. This simple workaround, despite being of practical use, is only weakly motivated and relies on the assumption that $\mathbb{E}_\sigma[h_i(\boldsymbol{z})] \approx h_i(\mathbb{E}_\sigma[\boldsymbol{z}])$ in each layer of the network (Baldi & Sadowski, 2014). The latter assumption does not hold in general, for the quality of the approximation also depends on the type of activation function adopted, which is instead neglected.

3.2. MC dropout

A more direct way of facing the computation of (3) is via Monte Carlo (MC) sampling, *i.e.* one generates a random sequence $\sigma^1, \dots, \sigma^{\mathtt{m}}$ of $\mathtt{m}$ gating variable configurations and averages the predictions of the corresponding gated neural networks:

$$\hat{f}_{\text{dropout}}(\boldsymbol{x}) = \frac{1}{\mathtt{m}} \sum_{i=1}^{\mathtt{m}} f_{\Theta^*,\sigma^i}(\boldsymbol{x}) \,. \tag{4}$$

This procedure called *MC dropout* has the advantage of giving a finer control on the approximation quality, since $\hat{f}_{\text{dropout}} \to f_{\text{dropout}}$ for $\mathtt{m} \to \infty$. However, it requires considerable increase of computation at test time ($\approx \mathtt{m}$ times slower on a single CPU). It has also been empirically shown in (Gal & Ghahramani, 2015b) that MC dropout exhibits a better performance than the standard scaling heuristic from the previous subsection in the presence of extensive use of dropout even at earlier convolutional layers.

4. Dropout distillation

An ideal solution for the inference phase combines the benefits from the two state-of-the-art approaches described before, *i.e.* it aims at retaining the computational efficiency of standard dropout while obtaining accuracies as good as the ones delivered from the computationally more expensive MC dropout. The method we propose is aligned with this goal and stems from the observation that one can try to project the ideal, but intractable, predictor defined as in (3) on a space of more tractable ones.

Let Θ^* be the parametrization of the neural network that has been obtained with dropout training and let $\mathcal{Q} \subset \mathcal{X} \to \mathcal{Y}$ be a set of predictors with input space $\mathcal{X}$ and output space $\mathcal{Y}$ that can be *tractably*, or even efficiently, evaluated. As an example, $\mathcal{Q}$ might consist of deterministic feed-forward neural networks having the same topology as the original network f_{Θ^*}. Now, our goal is to find a predictor in $\mathcal{Q}$ that matches as close as possible f_{dropout} defined as in

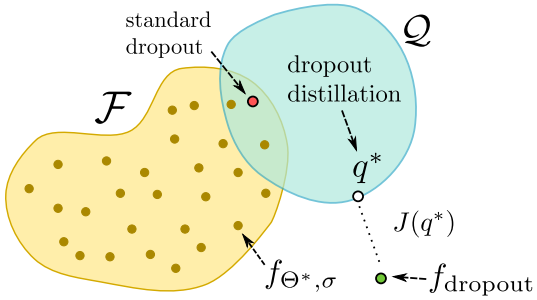

Figure 1. Visualization of dropout distillation. Set $\mathcal{F}$ includes predictors of the same type as the original network trained with dropout (with possibly different parametrizations) as well as all possible gated networks (dots illustrating $f_{\Theta^*,\sigma}$ for some σ). Set $\mathcal{Q}$ is the user-defined set of efficient predictors, which *might* have a non-empty intersection with $\mathcal{F}$. In general, the ideal dropout predictor f_{dropout}, which is intractable to compute, belongs neither to $\mathcal{F}$ nor $\mathcal{Q}$. Dropout distillation finds a predictor $q^* \in \mathcal{Q}$ that minimizes its divergence from f_{dropout} measured in terms of J. Standard dropout belongs to $\mathcal{F}$ because it produces a predictor of the same type as the original network, just with scaled parameters. It might also belong to $\mathcal{Q}$, in which case standard dropout can be used to initialize the dropout distillation optimization procedure.

(3). We achieve this by minimizing an objective function $J(q)$, which measures the dissimilarity between any predictor $q \in \mathcal{Q}$ and the ideal predictor f_{dropout} as follows (see Fig. 1 for an illustration of the procedure):

$$J(q) = \mathbb{E}_{\boldsymbol{x}}\left[\ell(f_{\text{dropout}}(\boldsymbol{x}), q(\boldsymbol{x}))\right] \,. \tag{5}$$

Here, the expectation is taken with respect to the unknown data distribution over $\mathcal{X}$.[3] The objective J is given in terms of a function $\ell(y, \hat{y})$, which measures the loss incurred by disagreements between q and the ideal predictor. Note that this loss function might differ from the one used during dropout training, *i.e.* the one in (1), but for reasons of convenience we reuse the same notation.

The benefits that come from minimizing J are as follows. We can replace the intractable predictor in (3) with a tractable one, which will target the accuracies of f_{dropout} and preserve at the same time the computational cost of standard dropout, taking a proper choice of $\mathcal{Q}$. However, optimizing (5) still requires evaluating f_{dropout}, which is not tractable, and requires also computing the expectation over $\boldsymbol{x}$. The latter problem can be pragmatically addressed by replacing the expectation with an empirical average over independent samples $\mathcal{S} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_{\mathtt{n}}\}$ drawn from the unknown distribution, *i.e.*

$$\hat{J}(q) = \frac{1}{\mathtt{n}} \sum_{i=1}^{\mathtt{n}} \ell(f_{\text{dropout}}(\boldsymbol{x}_i), q(\boldsymbol{x}_i)) \,, \tag{6}$$

which can approximate $J(q)$ arbitrarily well, since $\hat{J}(q) \to J(q)$ for $\mathtt{n} \to \infty$. The set of *unlabelled* samples $\mathcal{S}$ might

[3] Please note that no label information is needed at this point.

Algorithm 2 Dropout distillation

Input: a dataset $\mathcal{S} \subset \mathcal{X}$ and a parametrization Θ^* of a neural network trained with dropout
Assume: $\mathcal{Q}$ a parametrized family of predictors q_Ω
$t = 0$
Initialize parameters Ω^0
repeat
  Sample $\{\boldsymbol{x}_{\pi_1}, \ldots, \boldsymbol{x}_{\pi_\mathsf{m}}\} \subset \mathcal{S}$
  Sample gating variable configurations $\sigma^1, \ldots, \sigma^\mathsf{m}$
  Update learning rate η_t
  $\Omega^{t+1} = \Omega^t - \frac{\eta_t}{\mathsf{m}} \sum_{j=1}^{\mathsf{m}} \frac{\partial}{\partial \Omega} \ell(f_{\Theta^*,\sigma^j}(\boldsymbol{x}_{\pi_j}), q_{\Omega^t}(\boldsymbol{x}_{\pi_j}))$
  $t = t + 1$
until stopping condition is met
Output: $\Omega^* = \Omega^t$

e.g. come from the training set used in (1) during training or from freshly collected data samples (see Subsec. 4.2). We are then left with the issue of evaluating f_{dropout}, which in turn requires computing the expectation over σ. This leads to a similar problem as the one we have encountered when minimizing $\hat{R}(\Theta)$ during the training phase, and the workaround there was to simply randomize σ during the stochastic gradient descent steps. As we will see later, this is indeed a viable solution also for the minimization of $\hat{J}(q)$, *i.e.* we can replace f_{dropout} with a randomly-gated network $f_{\Theta^*,\sigma}$ within a stochastic gradient descent procedure, akin to Alg. 1. However, the application of this trick merits attention, because it does not guarantee convergence towards the desired solution, *i.e.* a minimizer of $\hat{J}(q)$, at least not in general, due to the position of the expectation over σ, which is nested within the loss function. It is instead possible to show (Bottou, 1998) that the aforementioned procedure, which is reported in Alg. 2 for a parametrized family of predictors, targets a local minimizer of a different objective, namely

$$\hat{J}'(q) = \frac{1}{\mathsf{n}} \sum_{i=1}^{\mathsf{n}} \mathbb{E}_\sigma \left[\ell(f_{\Theta^*,\sigma}(\boldsymbol{x}_i), q(\boldsymbol{x}_i)) \right] . \tag{7}$$

To legitimate the use of the stochastic gradient descent in Alg. 2 to minimize $\hat{J}(q)$, we have to establish relations between the latter objective and the objective $\hat{J}'(q)$ that is actually optimized. How minimizers of $\hat{J}(q)$ relate to the ones of $\hat{J}'(q)$ will result from the theorem below (proof in supplementary material), providing conditions under which strong connections between $\hat{J}$ and $\hat{J}'$ can be made:

Theorem 1. *Let $\mathcal{Y} \subseteq \mathbb{R}^\mathsf{k}$ and let $J' : \mathcal{Q} \to \mathbb{R}$ be defined as*

$$J'(q) = \mathbb{E}_{\boldsymbol{x},\sigma} \left[\ell(f_{\Theta^*,\sigma}(\boldsymbol{x}), q(\boldsymbol{x})) \right] . \tag{8}$$

Then the following holds:

i) if ℓ is convex in its first argument then for all $q \in \mathcal{Q}$,

$$J(q) \leq J'(q) ;$$

ii) if there exist functions $g_1, g_2 : \mathcal{Y} \to \mathbb{R}$ and $g_3 : \mathcal{Y} \to \mathbb{R}^\mathsf{k}$ such that $\ell(y, \hat{y}) = g_1(y) + g_2(\hat{y}) + y^\top g_3(\hat{y})$ then there exists $\Delta \in \mathbb{R}$ such that for all $q \in \mathcal{Q}$,

$$J(q) = J'(q) + \Delta .$$

The theorem holds true also if we replace J and J' with $\hat{J}$ and $\hat{J}'$, respectively.

Thm. 1 shows that whenever we employ a loss function convex in its first argument – and many losses that are typically used do satisfy this property – the *dropout distillation* algorithm in Alg.2 optimizes an upper-bound of the objective in (6). The second part of the theorem provides an even more intriguing conclusion because with the given conditions for the loss function, it guarantees the correctness of the optimization procedure also for the empirical approximation $\hat{J}(q)$ in Equ. (6). Indeed, if $\hat{J}(q) = \hat{J}'(q) + \Delta$, where Δ is not depending on q, then minimizers of $\hat{J}(q)$ will coincide with minimizers of $\hat{J}'(q)$, thus legitimating our proposed dropout distillation approach.

4.1. Notes about the loss function

An important question concerns the restrictiveness of the condition that Thm. 1-*ii)* puts on the loss function. Fortunately, there is a significant class of known loss functions that satisfy the required condition. In particular, all losses that are *Bregman divergences* (Bregman, 1967) fulfill the requirements. A loss function $\ell(y, \hat{y})$ is a Bregman divergence if it takes the form $\ell(y, \hat{y}) = \phi(y) - \phi(\hat{y}) - (y - \hat{y})^\top \nabla\phi(\hat{y})$, where $\phi : \mathcal{D} \to \mathbb{R}$ is a strictly convex, differentiable function defined on a convex set $\mathcal{D}$. By setting $g_1(y) = \phi(y)$, $g_2(\hat{y}) = \hat{y}^\top \nabla\phi(\hat{y}) - \phi(\hat{y})$ and $g_3(\hat{y}) = -\nabla\phi(\hat{y})$ we have that any Bregman divergence satisfies the hypothesis of Thm. 1-*ii)*. *E.g.*, Bregman divergences are the squared Euclidean distance and the Kullback-Leibler divergence, which are often used to define losses for regression and classification tasks.[4]

4.2. Generation of the unlabelled dataset $\mathcal{S}$

The set of samples $\mathcal{S}$ that is used to define the objective in (6) can be generated in a number of different ways. A naive solution consists in generating random samples according to some noise distribution (*e.g.* Gaussian noise). This allows us to generate unlimited amount of data, but it jeopardizes the effectiveness of the dropout distillation algorithm, due to the large discrepancy between the sampling distribution and the true data distribution. Another solution consists in retaining the same samples used during training if still available, *i.e.* $\mathcal{S} = \{\boldsymbol{x}_i \in \mathcal{X} :$

[4] The squared Euclidean distance is generated by $\phi(\boldsymbol{y}) = \|\boldsymbol{y}\|^2$, while the KL-divergence by the entropy function.

$(\boldsymbol{x}_i, y) \in \mathcal{T}$ for some $y \in \mathcal{Y}\}$. This yields a more significant outcome from the algorithm, but it might be prone to overfitting, since it insists on fitting the target predictor f_{dropout} only on samples used to train it.[5] Another possibility considers using perturbed versions of the training samples. This allows us to artificially generate a large number of samples, which will be close to the true data manifold, while counteracting the aforementioned overfitting issue. The preferred solution is however to construct $\mathcal{S}$ based on previously unseen samples, drawn from the data distribution. This is however not always possible, in which case the use of perturbed training samples yields a good compromise.

4.3. Why "distillation"?

The terminology "distillation" has been introduced in (Hinton et al., 2014) to indicate the task of transferring knowledge that resides within some cumbersome models to a small model for the sake of easier deployment. The same concept was pioneered in (Bucilă et al., 2006) under a different name, namely model compression. The underlying idea is to use a trained, complex model, or ensemble of models, to label a large set of unlabelled data, thus generating de facto a new training set, which is then used to train another smaller model. If we re-interpret the same idea within a Bayesian scenario, where the ensemble is generated by a Bayesian posterior distribution, then an even older precursor of distillation is (Snelson & Ghahramani, 2005), where the purpose was to construct compact approximations of Bayesian predictive distributions. Another recent work that that aims at distilling a Monte Carlo approximation to the Bayesian posterior predictive distribution is (**?**). Here, the authors propose an algorithm that is close in spirit to ours, even though it was not related to dropout.

We refer to our approach as *dropout distillation* because it follows to some extent the general scheme of the distillation paradigm: the stochastic regularization of dropout yields a large ensemble of possible predictors (exponential in the number of gating variables), which we want to mimic in the best possible way via a *single* predictor from some hypothesis space $\mathcal{Q}$. Our main focus is to construct a predictor having the same complexity of standard dropout, while targeting the better accuracies of the averaged prediction in (3), and at the same time provide some theoretical guarantees that legitimate the use of the proposed algorithm. Nevertheless, nothing prevents our approach from performing model compression by restricting $\mathcal{Q}$ to smaller models, as we will also demonstrate with some experiments in the following section.

[5] f_{dropout} yields crisper predictions on the data it was trained on, thus limiting the amount of transferable dark knowledge.

5. Experiments

In this section we assess the effectiveness of dropout distillation when applied to a variety of neural network architectures for classification tasks. We start with experiments in Subsect. 5.1 that validate our core contribution, *i.e.* comparing our proposed dropout distillation approach against state-of-the-art inference with dropout. In Subsect. 5.2 we illustrate the impact of using different datasets for training of dropout distillation. Finally, Subsect. 5.3 demonstrates the ability of our approach to jointly perform dropout distillation and target model compression, *i.e.* considering target predictor architectures with lower model complexity. In all our experiments we have considered the KL-divergence as loss function for both training and distillation phases.

5.1. Dropout distillation vs. standard- and MC dropout

In a first series of experiments we compare *Standard* dropout (see Sec. 3.1) and *Monte Carlo* (MC) dropout (see Sec. 3.2) with the proposed dropout *Distillation* algorithm. In all cases, we first train a baseline network and evaluate its performance using standard and MC dropout inference. In the case of MC dropout, we average $\mathsf{m} = 100$ predictions using randomly sampled configurations of the gating variables. Next, we use Alg. 2 to train a new deep neural network having the same architecture as the baseline net. We did not experience significant reduction of test errors when increasing m. The parameters Ω are initialized from those of the baseline, while the dataset $\mathcal{S}$ is obtained by randomly perturbing the training images (pixels of the training images are randomly set to zero with probability 0.2).

The results are summarized in Table 1, including mean test errors with corresponding standard deviations over 5 repetitions for MC and our method. Dropout distillation performs better than standard dropout in almost all experiments (and equally well on the MNIST LeNet). We observe that in most cases MC dropout outperforms our approach, at the cost of a $\approx$100 fold increase in computational complexity and higher error standard deviation. Interestingly, the performance delta between standard and MC dropout is consistently higher for network architectures with multiple dropout layers (*e.g.* CIFAR10 Quick, AllConv). Next, we provide a description of each dataset used along with corresponding training settings.

CIFAR10 experiments. The CIFAR10 dataset (Krizhevsky & Hinton, 2009) consists of 60.000 tiny color images of size 32×32, each belonging to one of 10 semantic classes like airplane, cat, *etc*. 50.000 images are meant to be used as training data and the rest for testing. We consider two recent network architectures using dropout at several layers: Network in Network (Lin et al., 2014) (NiN) and All Convolutional Network (Springenberg et al., 2014) (AllConv). We also consider a modified

Network	Standard	Monte Carlo	Distillation
MNIST LeNet	0.67 %	0.67±0.01 %	0.67±0.01 %
MNIST LeNetAll	0.51 %	0.49±0.01 %	0.49±0.01 %
CIFAR10 Quick	18.15 %	16.78±0.14 %	17.20±0.06 %
CIFAR10 NiN	11.16 %	11.04±0.06 %	11.14±0.04 %
CIFAR10 AllConv	11.20 %	10.60±0.10 %	10.80±0.04 %
CIFAR100 AllConv	32.46 %	30.99±0.18 %	32.07±0.05 %
CIFAR100 NiN	35.28 %	35.05±0.08 %	35.20±0.05 %

Network	Train	Pert. Train	Test
CIFAR10 Quick	17.15 %	17.20 %	17.23 %
CIFAR10 NiN	11.20 %	11.14 %	11.14 %
CIFAR10 AllConv	10.59 %	10.80 %	10.64 %
CIFAR100 AllConv	31.88 %	32.07 %	31.75 %
CIFAR100 NiN	35.21 %	35.20 %	35.18 %

Table 1. Left: Comparison of test data errors using standard dropout, Monte Carlo dropout and our proposed dropout distillation on the CIFAR10, CIFAR100 and MNIST datasets using several different network architectures (with standard deviations for 5 repetitions of MC and our distillation approach). Right: Dropout distillation results obtained by training on: original training samples (Train); perturbed training samples (see Sec. 5.1, Pert. Train); unlabelled test samples (Test).

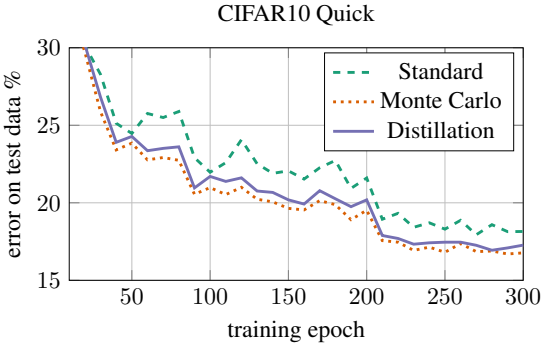

Figure 2. Test data misclassification rate for CIFAR10 dataset using described "Quick" network architecture. Comparison of error rates for Standard, Monte Carlo and our proposed dropout distillation inference.

version of the "Quick" architecture of cuda-convnet[6], where we perform dropout after every convolutional or fully connected layer, except for the final one.

All baseline networks are trained using stochastic gradient descent with momentum 0.9 and L_2 regularization. For NiN and AllConv we adopt the originally reported meta-parameters, while for Quick we perform 300 training epochs, with a learning rate of 0.05 for the first 200 and 0.005 for the last 100. For dropout distillation we use the same training schedule for all networks: first, we perform 20 epochs using a learning rate equal to the one used in the last iteration of the baseline network, then we reduce the learning rate by a factor of 10 and run the training for 10 additional epochs. For each network, during training we vertically flip each input image with probability 0.5.

CIFAR100 experiments. The CIFAR100 dataset (Krizhevsky & Hinton, 2009) is almost identical to CIFAR10, except for the number of classes which is increased to 100. Each class contains 600 images, where 500 are reserved for training and 100 for testing. We consider the same Network in Network and All Convolutional architectures as used for the CIFAR10 experiments. Also, the training procedures for the baseline and distilled networks remain the same.

MNIST experiments. As a final benchmark, we consider the MNIST (LeCun et al., 1998) handwritten digits recognition dataset. Here we adopt two variations of the well known LeNet-5 (LeCun et al., 1998) architecture: one using dropout after the penultimate layer (LeNet) and one using dropout after every convolutional or fully connected layer (LeNetAll). The baseline networks are trained using the following schedules: 20 epochs with a learning rate of 10^{-3}, then 10 epochs with a learning rate of 10^{-4}, while dropout distillation is performed in the same way as for CIFAR10 and CIFAR100.

5.2. Performing distillation with different training sets

As mentioned in Section 4.2, the unlabelled dataset $\mathcal{S}$ can be generated in a number of different ways. Here, we compare the results obtained from three different strategies: i) using the original training samples (Train), ii) using perturbed training samples as described in Section 5.1 (Pert. Train) and iii) using the test data samples (Test). Please note that in none of the aforementioned strategies the corresponding ground truth labels are used, thus legitimating also iii) as a potential, though somewhat limited data generation scenario. The results obtained on the different CIFAR networks are reported in the right Table 1. We observe that our approach is largely insensitive *w.r.t.* the proposed strategies of generating training data. Overall, we experience a gap of at most $\approx 0.3\%$ for different ways to generate $\mathcal{S}$, confirming the effectiveness of our approach to handle different types of training data.

5.3. Distilling into smaller target networks

Another potential application for our proposed dropout distillation is related to model compression (Bucilă et al., 2006) or network distillation (Hinton et al., 2014). We can control the target model capacity, depending on the choice of the target hypothesis space $\mathcal{Q}$ (where we eventually select our dropout distillation predictor from). Consequently, we can perform dropout distillation *jointly* with distilling the networks knowledge into a less complex target model,

[6] https://code.google.com/archive/p/cuda-convnet/

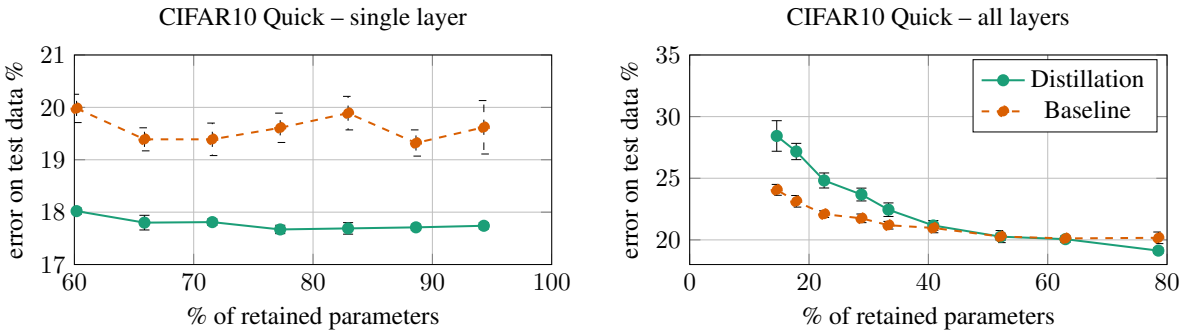

Figure 3. Test data misclassification rate for the distilled CIFAR10 Quick network, as a function of percentage of retained parameter. Left: changing the number of neurons in the first fully connected layer. Right: changing the size of all layers in the network.

such that *e.g.* the resulting predictors can be deployed on less powerful hardware platforms. While in the previously described experiments the target network topology was always the same as the source network (using its parameters for initialization), we are now demonstrating results on the CIFAR10 dataset using modified target networks.

Reduced CIFAR10 Quick network – single layer. In this experiment we are gradually reducing the number of output units of the penultimate, fully-connected layer of the CIFAR10 Quick network while monitoring the prediction error on test data. To this end, we are decreasing the original number of hidden units from 64 by multiples of 8, until we are finally down to only 8 output units. Both affected layers (*i.e.* the penultimate and terminal one) are initialized randomly by sampling from a gaussian distribution with standard deviation 0.01, while all preceding layer parameters are initialized from the original network. Fig. 3 shows the test error as a function of the remaining network parameters (in percent). We compared the distilled network, trained with our method, against the same network trained from the original training data (Baseline in the figure). We are able to drop $\approx 40\%$ of the network parameters, still obtaining a test error of $\approx 18.02\%$. This is still lower than the results obtained by standard dropout inference with all parameters (18.15%) and significantly better than the baseline network. Similar to the results presented in Tab. 1, we find low standard deviations when performing model compression.

Reduced CIFAR10 Quick network – all layers. In this experiment we reduce the size of the whole network by proportionally shrinking all the network's layers. We are again reducing the number of parameters and therefore the model capacity, however, this time by gradually removing filters from the convolutional layers and neurons from the fully connected layers. Fig. 3, right, shows the test error with corresponding standard deviations when randomly decreasing the number of filters and neurons. Obviously, reducing all layers at once has a larger impact on the overall performance for a given model size. Indeed, we can see that the distilled network performs better than the baseline with compression levels up to 40%. However, retraining the small network from the original training data yields lower erros than our distilled network at higher compression rates. Indeed, if the model capacity shriks too much, we incur an underfitting issue caused by the regularizing effect that the dropout distillation introduces, which in turn prevents overfitting when the model complexity is large.

6. Conclusion and future works

Dropout has proven to be an effective way for regularizing neural networks and its use contributed to delivering state-of-the-art results in many application areas. It implicitly allows to train an ensemble of exponentially many neural networks sharing the same parametrization, where all outputs should be averaged at test time to deliver the final prediction. This intractable computation is typically approximated with an efficient heuristic, by simply scaling the affected layers during inference. In this work we have introduced dropout distillation, an approach to better approximate the intractable average predictor, without sacrificing the computational efficiency of standard dropout. To this end, we find a predictor within a family of efficient predictors in a way to minimize the divergence from the ideal, but intractable, averaging predictor. Although the divergence itself is intractable to compute, we legitimated the use of stochastic gradient descent to carry out the optimization akin to dropout training, and we showed the correctness of our algorithm for a widely used class of loss functions. Experiments on standard benchmark datasets confirmed the validity of our method with consistent improvements over conventional dropout inference. We have also shown that by confining the hypothesis space to smaller neural networks, we were able to construct predictors that can be evaluated more efficiently than standard dropout inference. As a future direction, we will investigate applications of dropout distillation to neural networks with recurrent connections trained with dropout, as *e.g.* shown in (Pham et al., 2014), as our approach makes no assumption about the form of the predictor.

References

Baldi, P. and Sadowski, P. Understanding dropout. In *Advances in Neural Information Processing Systems*, volume 26, pp. 2814–2822. 2013.

Baldi, Pierre and Sadowski, Peter. The dropout learning algorithm. *Artif. Intell.*, 210:78–122, 2014.

Bottou, L. Online algorithms and stochastic approximations. In Saad, David (ed.), *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

Bregman, L. M. The relaxation method of finding the common points of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7:200–217, 1967.

Bucilă, C., Caruana, R., and Niculescu-Mizil, A. Model compression. In *Int. Conf. on Knowledge Discovery and Data Mining*, pp. 535–541, 2006.

Gal, Y. and Ghahramani, Z. Dropout as a Bayesian approximation: Insights and applications. In *Deep Learning Workshop, ICML*, 2015a.

Gal, Y. and Ghahramani, Z. On modern deep learning and variational inference. In *Advances in Approximate Bayesian Inference workshop, NIPS*, 2015b.

Gao, W. and Zhou, Z.-H. Dropout rademacher complexity of deep neural networks. *ArXiv*, abs/1402.3811, 2014.

Graham, B., Reizenstein, J., and Robinson, L. Efficient batchwise dropout training using submatrices. *ArXiv*, abs/1502.02478, 2015.

Hinton, G., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv*, abs/1207.0580, 2012.

Hinton, G. E., Vinyals, So., and Dean, J. Distilling the knowledge in a neural network. In *Deep Learning Workshop, NIPS*, 2014.

Jain, P., Kulkarni, V., Thakurta, A., and Williams, O. To drop or not to drop: Robustness, consistency and differential privacy properties of dropout. *ArXiv*, abs/1503.02031, 2015.

Krizhevsky, Alex and Hinton, Geoffrey. Learning multiple layers of features from tiny images, 2009.

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Lin, M., Chen, Q., and Yan, S. Network in network. In *Int. Conf. on Learning Representations*, 2014.

Pham, V., Bluche, T., Kermorvant, C., and Louradour, J. Dropout improves recurrent neural networks for handwriting recognition. In *Int. Conf. on Frontiers in Handwriting Recognition*, pp. 285–290, 2014.

Snelson, E. and Ghahramani, Z. Compact approximations to bayesian predictive distributions. In *Int. Conf. on Machine Learning*, pp. 840–847, 2005.

Springenberg, Jost Tobias, Dosovitskiy, Alexey, Brox, Thomas, and Riedmiller, Martin. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. of Machine Learning Research*, 15:1929–1958, 2014.

Wager, S., Wang, S., and Liang, P. S. Dropout training as adaptive regularization. In *Advances in Neural Information Processing Systems*, volume 26, pp. 351–359. 2013.

Wan, L., Zeiler, M. D., Zhang, S., LeCun, Y., and Fergus, R. Regularization of neural networks using dropconnect. In *Int. Conf. on Machine Learning*, pp. 1058–1066, 2013.