PROGRESSIVE KNOWLEDGE DISTILLATION: BUILDING ENSEMBLES FOR EFFICIENT INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

We study the problem of *progressive distillation*: Given a large, pre-trained teacher model g, we seek to decompose the model into an ensemble of smaller, low-inference cost student models f_i. The resulting ensemble allows for flexibly tuning accuracy vs. inference cost, which can be useful for a multitude of applications in efficient inference. Our method, B-DISTIL, uses an algorithmic procedure that uses function composition over intermediate activations to construct expressive ensembles with similar performance as g, but with much smaller student models. We demonstrate the effectiveness of B-DISTIL by decomposing pretrained models across a variety of image, speech, and sensor datasets. Our method comes with strong theoretical guarantees in terms of convergence as well as generalization.

1 INTRODUCTION

Knowledge distillation aims to transfer the knowledge of a large model into a smaller one (Buciluă et al., 2006; Hinton et al., 2015). While this technique is commonly used for model compression, one downside is that the procedure is fairly rigid—resulting in a single compressed model of a fixed size. In this work, we instead consider the problem of *progressive distillation*: approximating a large model via an ensemble of smaller, low-latency models. The resulting decomposition is useful for a number of applications in efficient inference. For example, components of the ensemble can be selectively combined to flexibly meet accuracy/latency constraints (Li et al., 2019; Yang & Fan, 2021), can enable efficient parallel inference execution schemes, and can facilitate *early-exit* (Bolukbasi et al., 2017; Dennis et al., 2018) or *anytime inference* (Ruiz & Verbeek, 2021; Huang et al., 2017a) applications, which are scenarios where inference may be interrupted due to variable resource availability.

More specifically, we seek to distill a large pre-trained model, g, onto an ensemble comprised of low-parameter count, low-latency models, f_i. We additionally aim for the resulting ensemble to form a *decomposition*, such that evaluating the first model produces a coarse estimate of the prediction (e.g., covering common cases), and evaluating additional models improves on this estimate (see Figure 1). There are major advantages to such an ensemble for on-device inference: 1) inference cost vs. accuracy trade-offs can be controlled on-demand at execution time, 2) the ensemble can either be executed in parallel or in sequence, or possibly a mix of both, and 3) we can improve upon coarse initial predictions without re-evaluation in response to resource availability.

While traditional distillation methods are effective when transferring information to a single model of similar capacity, it has been shown that performance can degrade significantly when reducing the capacity of the student model (Mirzadeh et al., 2020; Gao et al., 2021). Moreover, distillation of a deep network onto a weighted sum of shallow networks rarely performs better than distillation onto a single model (Cho & Hariharan, 2019; Allen-Zhu & Li, 2020). Our method exploits connections to minimax optimization and online learning (Schapire & Freund, 2013) to allow models in our ensemble to compose and reuse intermediate activation outputs of other models during inference. As long as these composition functions are resource efficient, we can increase base class capacity at roughly the same inference cost as a single model. Moreover, we show that our procedure retains the theoretical guarantees of these methods (Schapire & Freund, 2013). Concretely,

- We formulate progressive distillation as a two player zero-sum game, derive a weak learning condition for distillation and present our algorithm, B-DISTIL, to approximately solve this game. To make the search for weak learners in low parameter count models feasible, we explicitly solve

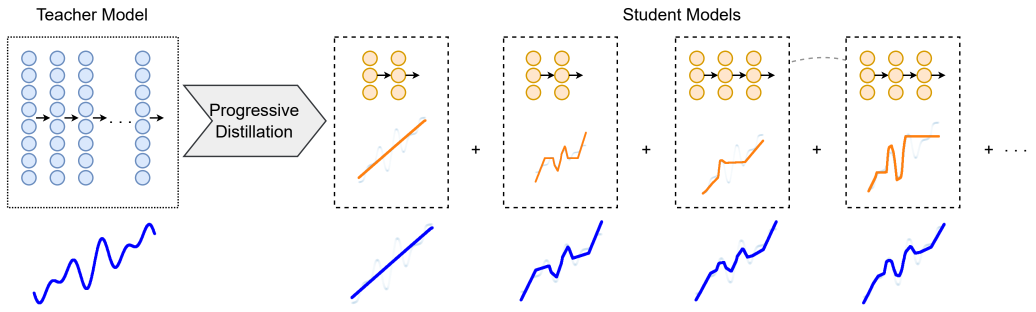


Figure 1: In progressive distillation, a large teacher model is distilled onto low inference cost models. The more student models we evaluate, the closer the ensemble's decision boundary is to that of the teacher model. Models in ensemble are allowed to depend on previously computed features.

a log-barrier based relaxation of our weak learning condition. Moreover, by allowing models to reuse computation from select intermediate layers of previously evaluated models of the ensemble, we are able to increase the model's capacity without significant increase in inference cost.

- We empirically evaluate our algorithm on synthetic as well as real-world classification tasks from vision, speech and sensor data with models suitable for the respective domains. We show that our ensemble behaves like a decomposition, allowing a run-time trade-off between accuracy and computation, while remaining competitive to the teacher model.
- We provide theoretical guarantees for our algorithm in terms of in-sample convergence and generalization performance. Our framework is not architecture or task specific and can recover existing ensemble models used in efficient inference, and we believe puts forth a general lens to view previous work and also to develop new, principled approaches for efficient inference.

2 Background and Related Work

Knowledge distillation. Machine learning inference is often resource-constrained in practice due to requirements around metrics such as memory, energy, cost, or latency. This has spurred the development of numerous techniques for model compression. A particularly popular approach is *knowledge distillation*, which considers transferring the knowledge of a larger model (or model ensemble) to a smaller one (Buciluǎ et al., 2006; Hinton et al., 2015). Despite its popularity, performing compression via distillation has several known pitfalls. Most notably, many have documented that distillation may not perform well when there is a *capacity gap* between the teacher and student, i.e., the teacher is significantly larger than the student (Mirzadeh et al., 2020; Gao et al., 2021; Cho & Hariharan, 2019; Allen-Zhu & Li, 2020). When performing distillation onto a weighted combination of ensembles, it has been observed that adding additional models into the ensemble does not dramatically improve performance over that of a single distilled model (Allen-Zhu & Li, 2020). There is also a lack of theoretical work characterizing when and why distillation is effective for compression Gou et al. (2021). Our work aims to address many of these pitfalls by developing a principled approach for progressively distilling a large model onto an ensemble of smaller, low-capacity ones.

Early exits and anytime inference. Numerous applications stand to benefit from the output of progressive distillation, which allows for flexibly tuning accuracy vs. inference cost and executing inference in parallel. Enabling trade-offs between accuracy and inference cost is particularly useful for applications that use early exit or anytime inference schemes. In on-device continuous (online) inference settings, *early exit* models aim to evaluate common cases quickly in order to improve energy efficiency and prolong battery life (Dennis et al., 2018; Bolukbasi et al., 2017). For instance, a battery powered device continuously listening for voice commands can use early exit methods to improve battery efficiency by classifying non-command speech quickly. Many methods that produce early exit models are also applicable to anytime inference (Huang et al., 2017a; Ruiz & Verbeek, 2021). In *anytime inference*, the aim is to produce a prediction even when inference is interrupted, say due to resource contention or a scheduler decision. Unlike early exit methods where the classifier chooses when to exit early, anytime inference methods have no control over when they are interrupted. We explore the effectiveness of using our method, B-DISTIL, for such applications in Section 5.

Two-player games, online optimization and boosting. The importance of equilibrium of two player zero-sum games has been recognized since the foundational work of von Neumann and Morgenstern (von Neumann & Morgenstern, 1944). Their connections to convex programming duality have lead to many developments in algorithm design, theoretical computer science and machine learning (Lugosi & Cesa-Bianchi, 2006; Goodfellow et al., 2014). One astounding application is by Schapire (1990) and Freund (1990) who showed that weak learners can be aggregated to produce strong learners. This has led to many practical boosting based learning algorithms and software such as AdaBoost (Freund & Schapire, 1997), gradient boosting (Mason et al., 1999), and XGboost (Chen & Guestrin, 2016). An application of boosting that is similar to our setup is by Trevisan et al. (2009). They show that given a target bounded function g, and class of approximating functions $\mathcal{F}$, one can approximate g arbitrarily well with respect to $\mathcal{F}$ using ideas from online learning and boosting. Although boosting has led to many theoretical and practical advances, these efforts have only recently seen success in deep learning applications. In particular, Suggala et al. (2020) propose a generalized boosting framework to extend boosting to deep networks using function compositions in feature space. The intermediate connections we (Section 3) is an extension of their approach.

3 Problem Formulation and Algorithm

We start by formulating our distillation task as a two player zero-sum game and introduce the framework of stochastic minimax optimization. Consider a class of hypotheses $\mathcal{F}$, a class of probability distributions $\mathcal{P}$ and a loss function $\mathcal{L}$ that is convex in its first argument. We consider two players whose pure strategy sets are $\mathcal{F}$ and $\mathcal{P}$ respectively, with loss (and reward) of the players given by $F(f,p) = \mathbb{E}_{x\sim p}[\mathcal{L}(f,x)]$. This naturally leads to the definition of the minimax value of the game given by

$$\max_{p\in\mathcal{P}} \min_{f\in\mathcal{F}} F\left(f,p\right). \tag{1}$$

Note that this game is convex in the hypothesis player (min-player) and concave in the distribution player (max-player) and thus amenable to algorithmic approaches. An algorithm is said to (approximately) solve the above minimax problem if it produces a distribution over hypotheses and a distribution that achieve the value in Equation (1). In order for the problem to be well-defined, we need to specify the access the algorithm has to the $\mathcal{F}$ and $\mathcal{P}$. One general notion is to allow access to a *weak gradient* vector h such that, when queried at distribution $\mu \in \mathcal{P}$ and for $\beta > 0$,

$$\langle h, \nabla_f F\left(f,p\right)\rangle \geq \beta, \tag{2}$$

The advantage of this lens is that many algorithms can be seen as implementing solutions to such minimax optimization problems. See Appendix A for a more involved discussion.

Although, eq. (2) is a seemingly easier notion than the full optimization, we emphasize that in many problems of interest even this is challenging. In fact, in the multilabel setting that we focus on, one of the main algorithmic challenges is to construct an algorithm that reliable and efficiently finds these weak gradients. Our method FIND-WL provides a way to do this successfully as demonstrated in our experimental setup, across a variety of datasets.

For the setting of our interest, we will restrict our attention the case where $\mathcal{P}$ is the set of all distributions over a sample set drawn from a fixed distribution p and, differ the details of the general framework to the Appendix. As in traditional distillation we will work with the output of g, often referred to as logits. Our goal is to produce an ensemble of predictors from the set of hypothesis classes $\{\mathcal{F}_m\}$ to approximate g 'well'. We will do this by searching for weak learners with respect to the target g in the class $\mathcal{F}_m$. Whenever this search fails we restart the search, now allowing models in $\mathcal{F}_m$ to reuse features computed by previous models.

To cast our progressive distillation problem as a two player game, we start with a predictor (a model or an ensemble) $g : \mathcal{X} \to \mathbb{R}^L$ already provided to us along with a non empty set of low inference cost[1] hypothesis class $\{\mathcal{F}_m\}_{m=1}^{M}$. We assume that $\{\mathcal{F}_m\}$ is in increasing order of inference cost. The loss function of interest then is $\frac{1}{2}\sum_{i,j}\left(f(x_i) - g(x_i)\right)_j^2$ which is the total squared error between the

[1]Users can define 'inference cost' in terms of memory requirements or compute requirements based on their use-case. We only assume $\{\mathcal{F}_i\}$ is ordered on this metric.

two predictors. Given a training set $\{x_i\}$, we think of the role of the max player in eq. (1) to produce distributions over the training set and the role of the min player to produce a hypothesis that minimizes the loss on this distribution. In this setting, note that $\mathcal{P} = \left\{p \in \mathbb{R}^{N \times L} : p_{i,j} \geq 0 \quad \forall j \sum_i p_{i,j} = 1\right\}$ is the product of simplices in $N \times L$ dimensions, and

$$\left(\nabla_f F\left(f, p\right)\right)_j = \sum_i p_{i,j} \left(f(x_i) - g(x_i)\right)_j . \tag{3}$$

Algorithm 1 B-DISTIL: Main algorithm

```
Require: Target $g$, rounds $T$, data $\{(x_i, y_i)\}_{i=1}^N$, learning rate $\eta$, model classes $\{\mathcal{F}_m\}_{m=1}^M$
1:  $K_t^+(i,j), K_t^-(i,j) \leftarrow \frac{1}{2N}, \frac{1}{2N} \quad \forall (i,j)$
2:  $F, r, t \leftarrow \emptyset, 1, 1$
3:  while $r < R$ and $t < T$ do
4:      $f_t =$ FIND-WL$(K_t^+, K_t^-, \mathcal{F}_r)$
5:      if $f_t$ is NONE then
6:          $r \leftarrow r + 1$
7:          continue
8:      end if
9:      With $l := f_t - g$, update $K_t^+, K_t^-$. $\forall (i,j)$
            $K_{t+1}^+(i,j) \leftarrow K_t^+(i,j) \exp(-\eta \cdot l(x_i)_j)$   (4)
            $K_{t+1}^-(i,j) \leftarrow K_t^-(i,j) \exp(\eta \cdot l(x_i)_j)$   (5)
10:     Normalize $K_t^+, K_t^-$.
11:     $F, t \leftarrow F \cup \{f_t\}, t + 1$
12: end while
13: return $\frac{1}{|F|} \sum_{i=1}^{|F|} f_i$
```

Algorithm 2 FIND-WL

```
Require: Probability matrices $K^+, K^-$, model class $\mathcal{F}$ parameterized by $\theta \in \Theta$, hyper-parameters for SGD
1:  Obtain $\{\mathcal{F}_r\}_1^R$ by expanding $\mathcal{F}$ (section 3.2).
2:  for $\mathcal{F}' \in \{\mathcal{F}_r\}_{r=1}^R$ do
3:      Initialize initial parameter $\theta_0 \in \mathcal{F}'$.
4:      for $i \in \{1, \ldots, \text{max-search}\}$ do
5:          Randomly initialize $f_{\theta_i}$.
6:          Run SGD to solve Equation 6.
7:          if $f_{\theta_i}$ is a weak learner then;
8:              return $f_{\theta_i}$
9:          end if
10:     end for
11: end for
12: return NONE
```

Concretely, at each iteration t, the algorithm maintains a matrices $K_t^+ \in R^{N \times L}$ and $K_t^- \in R^{N \times L}$ of probabilities (in our setting, it turns out to be easier to maintain the positive errors and the negative error separately). Note that, the matrices K_t^+ and K_t^- are such that for all $j \in [L]$, $\sum_i K_t^+(i,j) + K_t^-(i,j) = 1$. Moreover, for all $(i,j) \in [N] \times [L]$, $0 \leq K_t^+(i,j), K_t^-(i,j) \leq 1$. The elements $K_t^+(i,j)$ and $K_t^-(i,j)$ can be thought of as the weight on the residuals $f_{t-1}(x) - g(x)$ *and* $g(x) - f_{t-1}(x)$ respectively, up-weighting large deviations from $g(x)$.

In order to make updates to this vector, we need access to a weak gradient similar to h in Equation (2). We formalize this using Definition 1, which can be seen as a natural extension of the standard weak learning assumption in the boosting literature to our setting.

Definition 1 (Weak learning condition)**.** *Given a dataset $\{(x_i, y_i)\}_{i=1}^N$, a target function $g : \mathcal{X} \to \mathbb{R}^L$ and probability matrices K_t^+, K_t^-, a function $f_t : \mathcal{X} \to \mathbb{R}^L$ is said to satisfy the weak learning condition with respect to g if for all j,*

$$\sum_i K_t^+(i,j)(f_t(x_i) - g(x_i))_j + \sum_i K_t^-(i,j)(g(x_i) - f_t(x_i))_j > 0$$

With the current probability matrices K_t^-, K_t^+, in each round t, B-DISTIL performs two steps; first, it invokes a subroutine FIND-WL that attempts to find a classifier $f_t \in \mathcal{F}_r$ satisfying the weak learning condition (Definition 1). If such a predictor is found, we add it to our ensemble and proceed to the second step – updating the probability matrices K_t^-, K_t^+ based on errors made by f_t. This is similar in spirit to the boosting algorithms such as AdaBoost (Schapire & Freund, 2013) for binary classification. If no such predictor can be found, we invoke the subroutine with the next class, $\mathcal{F}_{r+1}$ and repeat the search till a weak learner is found or we have no more classes to search in.

3.1 Finding Weak-learners

As mentioned earlier, the real difficulty in provably boosting in this setting is in finding a *single* learner f_t at round t that satisfies our weak learning condition simultaneously for all labels j. Existing boosting methods for classification treat multi-class settings ($L > 1$) as L instances of the binary classification problem (one vs all) (Schapire & Freund, 2013). They typically choose L different weak learners for each instance, which is unsuitable for resource efficient on-device inference. The difficulty is further increased by the capacity gap between the student and teacher models we consider for distillation. In our work, we focus on this aspect of the problem. Thus, along with controlling temperature for distillation, we employ two additional strategies; a) we use a log-barrier regularizer in the objective FIND-WL solves to promote weak learning and, b) we reuse limited stored activation outputs of previously evaluated models to increase the expressivity of the current base class.

3.1.1 Log-barrier regularizer

To find a weak learner, the FIND-WL method solves the optimization program in Equation 6 using stochastic gradient descent.

$$\min_{f \in \mathcal{F}} \quad \sum_i \frac{1}{N} \mathcal{L}_{\text{task}}(f(x_i), g(x_i)) - \frac{1}{\gamma} \sum_{i,j} \left(I^+_{ij} \log\left(1 + \frac{l(x_i)_j}{2B}\right) + (1 - I^+_{ij}) \log\left(1 - \frac{l(x_i)_j}{2B}\right)\right) \tag{6}$$

Here, we define $I^+_{ij} := I[K^+_t(i,j) > K^-_t(i,j)]$, $B \geq \max\{\|f(x)\|, \|g(x)\|\}$ for all x, and $l(x_i) := f(x_i) - g(x_i)$. The first term is a task specific loss between $f(x_i)$ and $g(x_i)$. For our binary and multi-class experiments, we use the standard binary/multi-class cross-entropy distillation loss function (Hinton et al., 2015), with temperature smoothening. To see the intuition behind the second term, first assume the following is true for all (i, j).

$$(K^+_t(i,j) - K^-_t(i,j))(f(x_i) - g(x_i))_j > 0. \tag{7}$$

Summing over all data points x_i, we can see that this is sufficient for f to be a weak learner with respect to g; for all j,

$$\sum_i (K^+_t(i,j) - K^-_t(i,j))(f(x_i) - g(x_i)) > 0 \tag{8}$$

$$\sum_i K^+_t(i,j)(f_t(x_i) - g(x_i))_j + \sum_i K^-_t(i,j)(g(x_i) - f_t(x_i))_j > 0. \tag{9}$$

The second term in Equation 6 is solving a soft log-barrier version of the weak learning condition of Equation 7, that penalizes those (i, j) for which Equation 7 does not hold. By tuning γ we can increase the relative importance of the regularization objective, encouraging f_t to be a weak learner potentially at the expense of task performance.

3.1.2 Intermediate layer connections

As mentioned in Section 2, distillation onto a linear combination of low capacity student models, often offers no better performance than that of any single model in the ensemble trained independently. For boosting, empirically we see that once the first weak learner has been found in some class $\mathcal{F}_m$ of low-capacity deep networks, it is difficult to find a weak learner for the reweighed objective from the same class $\mathcal{F}_m$. To work around this we let our class of weak learners at around t to include functions that depend on output of intermediate layers of previous weak learners (Suggala et al., 2020).

As a concrete example, consider a deep fully connected network with U layers, parameterized as $f = W\phi_{1:U}$. Here $\phi_{1:u}$ can be thought of as a feature transform on x using the first u layers into $\mathbb{R}^{d_u}$ and $W \in R^{d_U \times L}$ is a linear transform. With two layer fully connected base model class $\mathcal{F}_0 := \{W^{(0)}\phi^{(0)}_{1:2} \mid W^{(0)} \in \mathbb{R}^{L \times d_2}\}$ (dropping subscript m for simplicity), define

$$\mathcal{F}_r = \{W^{(r)}\phi^{(r)}_{1:2}(\text{id} + \phi^{(r-1)}_{1:2})\} \quad \text{and} \quad \mathcal{F}'_r = \{W^{(r)}\phi^{(r)}_2(\text{id} + \phi^{(r)}_1 + \phi^{(r-1)}_1)\}$$

with $\text{id}(x) := x$. It can be seen that $\{\mathcal{F}_r\}$ and $\{\mathcal{F}'_r\}$ define a variant of residual connection based networks (Huang et al., 2017b). It can be shown that classes of function $\{\mathcal{F}_r\}$ (and $\{\mathcal{F}'_r\}$) increase in capacity with r. Moreover, when evaluating sequentially the inference cost of a model from $\mathcal{F}_r$ is roughly equal to that of $\mathcal{F}$, since each subsequent evaluation *reuses* stored activations from previous evaluations. For this reason the parameter count of each $\mathcal{F}_r$ remains the same as that of the base class.

We informally refer to the process of constructing $\{\mathcal{F}_r\}$ given a choice of base class $\mathcal{F}_0$, the parameter R and the connection type as *expanding* $\mathcal{F}_0$. In our experiments, we consider four connection functions: dense connections (He et al., 2016), residual-connections (Huang et al., 2017b), LSTM gated-connections (Hochreiter & Schmidhuber, 1997) and GRU (Cho et al., 2014) gated connections and other accumulation operations, and consider R as a hyper-parameter.

Note that while intermediate connections help with capacity, they often reduce parallelizability. For instance, weak learners from $\mathcal{F}'$ defined here depend on the output of a (fixed) learner from $\mathcal{F}_{r-1}$, and thus can only finish its inference after the previous models is evaluated. As a practical consequence dependencies on activation outputs of later layers are preferred, as this allows us more freedom in choosing execution schemes (see Appendix C).

4 THEORETICAL ANALYSIS

In this section, we provide theoretical analysis and justification for our method. We first prove in sample convergence guarantees; we will show that the ensemble output produced by algorithm 1 converges to g at $(1/\sqrt{T})$ rate, provided that the procedure FIND-WL succeeds at every time t.

Theorem 1. *Suppose the class $\mathcal{F}$ satisfies that for all $f \in \mathcal{F}$, $\|f-g\|_\infty \leq G_\infty$. Let $F = \{f_t\}$ be the ensemble after T rounds of Algorithm 1, with the final output $F_t = \frac{1}{T}\sum_{t=1}^{T} f_t$. If f_t satisfies eq. (7) for all $t \leq T$ then for $T \geq \ln 2N$ and $\eta = \frac{1}{G_\infty}\sqrt{\frac{\ln 2N}{T}}$, we have for all j*

$$\|F_{t,j} - g_j\|_\infty \leq G_\infty \sqrt{\frac{\ln 2N}{T}} \tag{10}$$

where $F_{t,j}$ and g_j are the j^{th} coordinates of the functions F_t and g respectively.

We differ the details of the proof to the Appendix B. The main idea behind the proof is to bound the rate of convergence of the algorithm towards the minimax solution. This proceeds by maintaining a potential function and keeping track of its progress through the algorithm. The bounds and techniques here are general in the sense that for various objectives and loss functions appropriate designed weak learners give similar rates of convergence to the minimax solution. Furthermore, a stronger version that shows exponential rates can be shown by additionally assuming an edge for the weak learner.

In addition to the claim above about the in sample risk, we also show that the algorithm has a strong out of sample guarantee. We show this by bounding the generalization error of the algorithm in terms of the generalization error of the class $\mathcal{F}$. In the following theorem, we restrict to the case of binary classification for simplicity but general result follow along similar lines.

Let $\mathcal{C}_T$ denote the class of functions of the form $H(x) = \text{sign}(\frac{1}{T}\sum_{i=1}^{T} h_t(x))$, where h_t are functions in class $\mathcal{F}$. Then we have the following generalization guarantee:

Theorem 2 (Excess Risk)**.** *Suppose data D contains of N iid samples from distribution $\mathcal{D}$. Suppose that the function g has ϵ margin on data D with probability μ, that is $\Pr_{x\sim D}[|g(x)| < \epsilon] < \mu$. Further, suppose that the class $\mathcal{C}_T$ has VC dimension d. Then, for $T \geq 4G_\infty^2 \ln 2N/\epsilon^2$, with probability $1-\delta$ over the samples, the output F_T of algorithm 1 satisfies*

$$\text{err}(F_T) \leq \widehat{\text{err}}(g) + O\left(\sqrt{\frac{d\ln(N/d) + \ln(1/\delta)}{N}}\right) + \phi$$

Note that the above theorem can easily be adapted to the case of margins and VC dimension of the class $\mathcal{C}_T$ being replaced with the corresponding fat shattering dimensions. Furthermore, in the setting of stochastic minimax optimization, one can get population bounds directly by thinking of sample losses and gradients as stochastic gradients to the population objective. This is for example the view taken by Suggala et al. (2020). In our work, we separate the population and sample bounds to simplify the presentation and the proofs.

Remark 2.1. *We further emphasize that though the exact bound that appear in Theorem 1 and Theorem 2 depend on our particular algorithm, the forms and the general flavor of the guarantees of the theorems are general amongst algorithms solving stochastic minimax problems. For example, the theorems of Suggala et al. (2020) can be captured by our framework.*

5 Empirical Evaluation

In this section, we evaluate B-DISTIL on both real world and simulated datasets, and over a variety of architecture types. We consider four real world datasets across three domains—vision, speech and sensor-data—as well as two simulated datasets. This allows us to evaluate our method on five of architecture types: fully connected networks, convolutional networks, residual networks, densely connected networks and recurrent networks.

5.1 Dataset Information

For experiments with simulated data, we construct two datasets. The first dataset, referred to as *ellipsoid* is a binary classification data set. Here the classification labels for each data point $x \in \mathbb{R}^{32}$ is determined by the value of $x^T Ax$ for a random positive semidefinite matrix A. The second simulated dataset, *cube*, is for multiclass classification with 4 classes. Here labels are determined by distance to 16 randomly selected vertices from $\{-1, 1\}^{32}$ in $\mathbb{R}^{32}$, randomly partitioned into 4 classes.

We use four real world data sets for our experiments. Our image classification experiments use the *CIFAR-10* and *CIFAR-100* datasets. For spoken-audio classification tasks we use the *Google-13* speech commands dataset. We use the daily sports activities (*DSA*) dataset for experiments with sensor data. Detailed information of all datasets used, including source, size and processing steps, along with synthetic data generation steps can be found in Appendix C.

5.2 Model Architecture Details

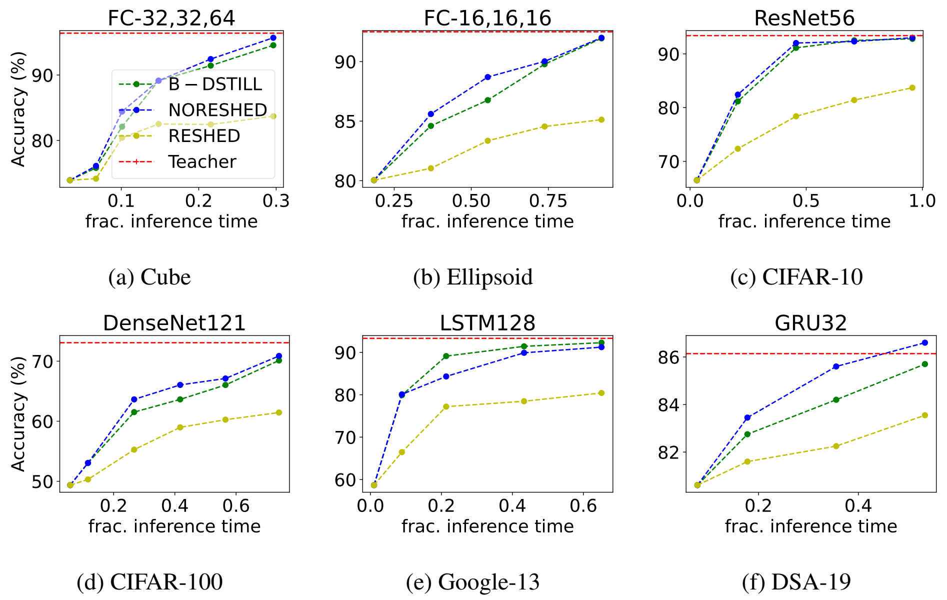
Teacher models. We use deep fully connected (FC) networks for classification on *Ellipsoid* and convolutional networks for *Cube*. For image classification on *CIFAR-10* dataset we use publicly available, pretrained ResNet models. We train reference DenseNet models for the *CIFAR-100* dataset based on publicly available training recipes. As both spoken audio data (*Google-13*) and sensor-data (*DSA-19*) are time series classification problems, we use recurrent neural networks (RNNs) for these experiments. We train LSTM (Hochreiter & Schmidhuber, 1997) based architecture on *Google-13* and a GRU (Cho et al., 2014) based architecture on *DSA-19*. Except for the pretrained ResNet models, all other teacher models are selected based on performance on validation data.

Student models. For all distillation tasks, for simplicity we design the student base model class from the same architecture type as the teacher model, but starting with significantly fewer parameters and resource requirements. We train for at most $T = 7$ rounds, keeping $\eta = 1$ in all our experiments. Whenever FIND-WL fails to find a weak learner, we expand the base class $\mathcal{F}$ using the connection specified as a hyperparameter. Since we need only at-most $T = 7$ weak learners, we can pick small values of R (say, 2). The details of the intermediate connections used for each data-set and hyperparameters such as γ, hyper-parameters for SGD can be found in Appendix C and D.

5.3 Experimental Evaluation and Results

First, we present the trade-off between accuracy and inference time offered by B-DISTIL in the context of anytime inference and early prediction. We compare our models on top-1 classification accuracy and total floating point operations (FLOPs) required for inference. We use a publicly available profiler (Rasley et al., 2020) to measure floating point operations. For simplicity of presentation, we convert these to the corresponding inference times (τ) on a reference accelerator (NVIDIA 3090Ti).

Anytime inference. As discussed previously, in the anytime inference setting a model is required to produce a prediction even when its execution is interrupted. Standard model architectures can only output a prediction once the execution is complete and thus are unsuitable for this setting. We instead compare against the idealized baseline where we assume *oracle* access to the inference budget ahead of *each* execution. Under this assumption, we can train a set of models suitable various inference time constraints, say by training models at various depths, and pick the one that fits the current inference budget obtained by querying the oracle. We refer to this baseline as NO-RESHED and compare B-DISTIL to it on both synthetic and real world datasets in Figure 2. This idealized baseline establishes an upper bound on the accuracy of B-DISTIL.



(a) Cube (b) Ellipsoid (c) CIFAR-10

(d) CIFAR-100 (e) Google-13 (f) DSA-19

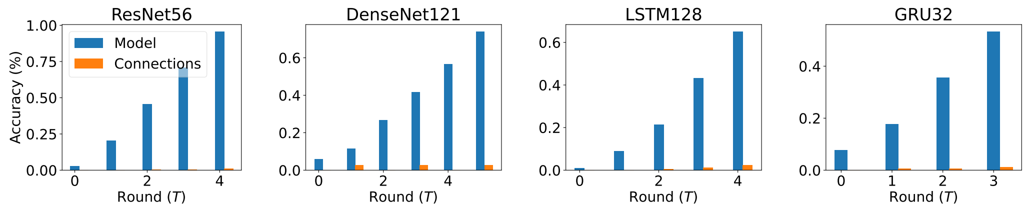
Figure 2: Accuracy vs inference-time trade-off. Inference time is reported as fraction of teacher's inference time. B-DISTIL performs this trade-off at runtime. The baseline NO-RESHED at inference time τ_w (x-axis) is the accuracy of a single model that is allowed $|\tau_w - 0|$ time for inference. Similarly the baseline RESHED at τ_w is the accuracy of an ensemble of models, where the ensemble is such that model w requires $|\tau_w - \tau_{w-1}|$ time to perform its inference. We see that B-DISTIL remains competitive to the oralce baseline (NO-RESCHED) and outperforms weighted averaging (RESCHED).

Dataset	**Algorithm**	**Early-prediction Acc**				
		$T = 50\%$		$T = 75\%$		$T = 100\%$
		Acc (%)	Frac	Acc (%)	Frac	Acc (%)
Google-13	E-RNN	88.31	0.48	88.42	0.65	92.43
	B-DISTIL	87.41	0.49	89.31	0.71	92.25
DSA-19	E-RNN	83.5	0.55	83.6	0.56	86.8
	B-DISTIL	82.1	0.53	84.1	0.58	87.2

Table 1: Early prediction performance. Performance of the ensemble produced by B-DISTIL to the E-RNN algorithm (Dennis et al., 2018). The accuracy and the cummulative fraction of the data early predicted at $50\%, 75\%$ and 100% time steps are shown. At $T = 100$, frac. evaluated is 1.0. The ensemble output by B-DISTIL with the early-prediction loss is competitive to the E-RNN algorithm, a method developed specifically for early prediction of RNNs.

B-DISTIL can improve on its initial prediction whenever inference jobs are allowed to be rescheduled. To contextualize this possible improvement, we consider the case where the execution is interrupted and rescheduled (with zero-latency, for simplicity) at times $\{\tau_1, \tau_2, \ldots, \tau_W\}$. We are required to output a prediction at each τ_w. Assuming we know these interrupt points in advance, one possible baseline can be as follows: select models with inference budgets $|\tau_1|, |\tau_2 - \tau_1|, \ldots, |\tau_w - \tau_{w-1}|$. Sequentially evaluate them and at at each interrupt τ_w, output the (possibly weighted) average prediction of the w models. We call this baseline RESCHED. Since the prediction at τ_w is a simple average of models, we expect its performance to serve as a lower-bound for the performance of B-DISTIL. In the same Figure (Figure 2) we compare B-DISTIL to the RESHED baseline. Note that both baselines require access to inference budget ahead of time.

Early prediction. To evaluate the applicability of our method for early prediction in online time-series inference, we compare the performance of B-DISTIL to that of E-RNN from Dennis et al. (2018). Unlike B-DISTIL, which can be applied to generic architectures, E-RNN is a state-of-the-art method for early prediction that was developed specifically for RNNs. When training, we set the task loss



(a) CIFAR-10 (b) SVNH-10 (c) Google-13 (d) DSA-19

Figure 3: Overhead of connections. The floating point operations required to evaluate the model added in round T, compared to that required to evaluate just the connections used by this model, for each of the real-world dataset. FLOPs are reported as a fraction of teacher model's FLOPs. We see that the connections add relatively little overhead.

$\mathcal{L}_{\text{task}}$ in Equation (6) to the early-classification loss used in E-RNN training. We evaluate our method on the time-series datasets *GoogleSpeech* and *DSA*. The performance in terms of time-steps evaluated is compared in Table 1. B-DISTIL remains competitive to E-RNN algorithm for early-prediction. Unlike E-RNN, B-DISTIL also offers early prediction for offline evaluation of time-series data — when all of the data arrives at once. For such cases, a threshold can be tuned similar to E-RNN and B-DISTIL can evaluate the models in its ensemble in order of increasing inference cost, exiting as soon as the confidence-score crosses this threshold.

Overhead of connections. Our method uses intermediate connections to improve its performance. Although these connections are designed to be efficient, they still have an overhead cost over an averaging based ensemble. The FLOPs required to evaluate intermediate connections corresponding to the distillation tasks in Figure 2 is shown in Figure 3. Here, we compare the FLOPs required to evaluate the model that was added in a round T to the FLOPs required evaluate the intermediate connections used by this model. Note that summing up all the FLOPs up to a round T, in Figure 3 gives the total FLOPs required to for the ensemble constructed at the end of round T. For all our models, this overhead is negligible when compared to the inference cost of the corresponding model.

To evaluate the benefits offered by the intermediate connections, we can compare the results of B-DISTIL run with connections and B-DISTIL without connections. The later case can be thought of as running the AdaBoost algorithm for distillation. Note that this is the same as the RESCHED baseline (simple averaging). Thus comparing the B-DISTIL plot in Figure 2 to the plot of RESCHED highlights the benefit of using intermediate connections. As in this work our focus is on finding weak learners in the presence of capacity gap, and we do not explore additional compression strategies like quantization, hard thresholding, low-rank projection that can further reduce inference cost.

6 Conclusion and Future Work

In this paper, we propose B-DISTIL, an algorithm for progressive distillation. B-DISTIL produces an ensemble of learners that provide efficient inference, with progressively better results as the ensemble size increases: this allows for a straightforward way to trade off accuracy and compute at inference time, in cases where inference may be interrupted abruptly, or when variable levels of accuracy are expected. We experimentally demonstrate the effectiveness of B-DISTIL by decomposing well established deeper models onto ensembles over a variety of problem domains. The procedure leverages a stochastic solver combined with log barrier regularization for finding weak learners during distillation. Further, we circumvent the issue of model capacity by using intermediary connections.

A key insight in this work is that posing distillation as a two player zero-sum game allows us to abstract away model architecture details into base class construction $\mathcal{F}$. This means that, conditioned on us finding a 'weak learner' from the base class, we retain the guarantees of the traditional boosting setup. A caveat of this abstraction is that the user must design $\mathcal{F}$. In future work, we would like to extend these ideas by a running boosting-like procedures on model and hyperparameters taken together. Similar methods have recently found success in applications such as hyperparameter tuning for federated learning (Khodak et al., 2021).