Go Wide, Then Narrow: Efficient Training of Deep Thin Networks

Denny Zhou [1] **Mao Ye** [2] **Chen Chen*** [1] **Tianjian Meng*** [1] **Mingxing Tan*** [1] **Xiaodan Song** [1] **Quoc Le** [1]
Qiang Liu [2] **Dale Schuurmans** [1]

Abstract

For deploying a deep learning model into production, it needs to be both accurate and compact to meet the latency and memory constraints. This usually results in a network that is deep (to ensure performance) and yet thin (to improve computational efficiency). In this paper, we propose an efficient method to train a deep thin network with a theoretic guarantee. Our method is motivated by model compression. It consists of three stages. First, we sufficiently widen the deep thin network and train it until convergence. Then, we use this well-trained deep wide network to warm up (or initialize) the original deep thin network. This is achieved by layerwise imitation, that is, forcing the thin network to mimic the intermediate outputs of the wide network from layer to layer. Finally, we further fine tune this already well-initialized deep thin network. The theoretical guarantee is established by using the neural mean field analysis. It demonstrates the advantage of our layerwise imitation approach over backpropagation. We also conduct large-scale empirical experiments to validate the proposed method. By training with our method, ResNet50 can outperform ResNet101, and $\text{BERT}_{\text{BASE}}$ can be comparable with $\text{BERT}_{\text{LARGE}}$, when ResNet101 and $\text{BERT}_{\text{LARGE}}$ are trained under the standard training procedures as in the literature.

1. Introduction

In many machine learning applications, in particular, language modeling and image classification, it is becoming common to dramatically increase the model size to achieve significant performance improvement (e.g., He et al., 2016; Wu et al., 2016; Devlin et al., 2019; Brock et al., 2019; Raffel et al., 2019; Brown et al., 2020). To enlarge a model, we can make it either much deeper or wider. A big deep learning model may involve millions or even billions of parameters, and be trained over a big computation cluster containing hundreds or thousands of computational nodes.

Despite their impressive performance, however, it is almost impossible to directly deploy these big deep learning models into production because of the low latency and memory constraints. To remedy this issue, there has been an increasing interest in developing compact versions of good performing big models to meet the practical constraints while without much drop of accuracy (e.g., Iandola et al., 2016; Sandler et al., 2018; Howard et al., 2019; Tan & V. Le, 2019; Wortsman et al., 2019; Sun et al., 2020).

In general, compact modeling results in networks which are both deep and thin. This is because a compact model must be sufficiently deep in order to extract hierarchical high-level representations that are impossible for shallow models to accomplish (e.g., Lee et al., 2009; Le et al., 2012; Allen-Zhu & Li, 2020), but each layer of the model does not have to be very wide since many neurons are redundant and can be pruned without hurting the performance (e.g., Han et al., 2015; Li et al., 2017; Frankle & Carbin, 2019; Liu et al., 2019; Ye et al., 2020).

Nevertheless, training deep thin networks are much more difficult than training deep wide networks. The loss surface of a deep thin network tends to be highly irregular and nonconvex (e.g., Li et al., 2018). Moreover, during the backpropagation through a deep thin network, gradients may vanish or explode (e.g., Bengio et al., 1994; Pascanu et al., 2013). On the other hand, it has been observed that increasing the width of, or "overparameterizing" the network makes it much easier to train since its loss surface becomes smoother and nearly convex (e.g., Li et al., 2018; Du et al., 2019; Allen-Zhu et al., 2019).

In this paper, we propose a generic algorithm to train deep thin networks with a theoretical guarantee. Our method is motivated by model compression. It consists of three stages (Figure 1 and Algorithm 1). In the first stage, we significantly widen the deep thin network to obtain a deep wide network, for example, twice wider, and then train it

*Equal contribution [1]Google Brain, USA [2]Department of Computer Science, The University of Texas at Austin, USA. Correspondence to: Denny Zhou <dennyzhou@google.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

until convergence. In the second stage, we use this well trained deep wide network to warm up (or initialize) the original deep thin network. In the last stage, we further fine tune this already well-initialized deep thin network. Since a deep thin network is highly nonconvex, a good initialization is almost all we need to obtain a good training result.

The key component of our method lies in its second stage. In this stage, the deep thin network is gradually warmed up by layer-to-layer imitating the intermediate outputs of the well trained deep wide network obtained in the first stage in a bottom-to-top fashion. This is analogous to curriculum learning, where a teacher breaks down an advanced learning topic to a sequence of small learning tasks, and then students learn these small tasks one by one under the teacher's stepwise guidance.

The readers may have noticed a technical issue to conduct layerwise imitation learning: the thin network and its wide version differs in the output dimension in every layer. This causes the difficulty to measure how well the thin network mimics the wide one. Obviously, such a problem does not exist in knowledge distillation since the final output dimension stays the same for either network. For example, the final outputs could be one-dimensional labels. To fix this dimension mismatch issue in our layerwise imitation learning, we insert a pair of linear transformations between any two adjacent layers in the thin network (see Figure 1): one is used to increase the dimension of its layer output to match the dimension of the corresponding layer output of the wide network, and the other to reduce the dimension back to its original size. Thus, the dimension expanded output from the thin network can be compared with the output from the wide network from layer to layer using elementwise metrics like the mean squared loss and Kullback-Leibler (KL) divergence. Since a sequence of linear layers is mathematically equivalent to a single linear layer, at the end of our algorithm, we can merge all adjacent linear layers in the thin network, including the linear transformations inside the original network modules. Thus, the thin network architecture is eventually restored to its original design.

We develop theoretical analysis for our method using the mean field analysis of neural networks (Song et al., 2018; Araújo et al., 2019; Nguyen & Pham, 2020). We show that, compared with direct gradient descent training of a deep thin network, our method allows for much simpler and tighter error bounds (see Proposition 3.3 vs. Theorem 3.5). The intuition underlying the theoretical analysis is that our layerwise imitation scheme avoids backpropagation through the deep network and consequently prevents an explosive growth of the error bound on the network depth. Similar theoretical results do not hold for the commonly used knowledge distillation and its variants (e.g., Ba & Caruana, 2014; Hinton et al., 2015), because they only modify the training target to include a distillation loss but still depend on backpropagation through the entire deep thin network.

In additional to theoretic analysis, we also conduct large scale empirical experiments to validate our approach. we train the ResNet (He et al., 2016) and BERT (Devlin et al., 2019) models using our method and the baseline methods which include training deep thin networks from scratch via backpropagation and training with knowledge distillation. Experimental results show that our method significantly outperforms the baseline methods. In particular, by training with our method, ResNet50 can outperform ResNet101, and $\text{BERT}_{\text{BASE}}$ can be comparable with $\text{BERT}_{\text{LARGE}}$, when ResNet101 and $\text{BERT}_{\text{LARGE}}$ are trained under the standard training procedures as in the literature.

We organize this paper as follows. In Section 2, we present our algorithm for training deep thin networks. In Section 3, we develop theoretic results around our method using the mean field analysis for neural networks. The proof details are provided in Appendix. The work related to our algorithm and theoretic analysis are discussed in Section 3. In Section 5, we present the experimental results of training the ResNet and BERT models using different methods. Finally, we conclude this paper with discussions in Section 6.

2. Algorithm

Let us denote by $S = \{S_1, S_2, \ldots, S_n\}$ a deep thin network that we want to train, where S_i denotes the building block at the i-th layer of S. For an input x, the output of network S is given as $S(x) = (S_n \circ S_{n-1} \circ \cdots \circ S_1)(x) = S_n(S_{n-1}(\cdots S_2(S_1(x))\cdots))$.

In a feed forward network, a building block can be a hidden layer or a group of hidden layers. However, in many other neural architectures, the structure of a building block can be much more complex. Here are two typical examples. In a model for image classification, a building block usually contains convolution, pooling, and batch normalization (He et al., 2016); in a model for language modeling, a building block may include multi-head attention, feed forward network, and layer normalization (Vaswani et al., 2017).

Stage 1: Wide learning. In this stage, we first construct a deep wide network $B = \{B_1, \ldots, B_n\}$, where building block B_i is obtained by significantly *widening* building block S_i in the deep thin network S. We then train this deep wide network B until convergence. In general, the wider the network, the easier to train it. How to make a network wider may not be that straightforward as it looks like. It is actually fairly case dependent. For a feed forward network, we can widen it by introducing more neurons in its hidden layers; for a convolution network, we can widen it by introducing more filters; for a transformer like model, we can widen it from multiple dimensions, including increasing its hidden

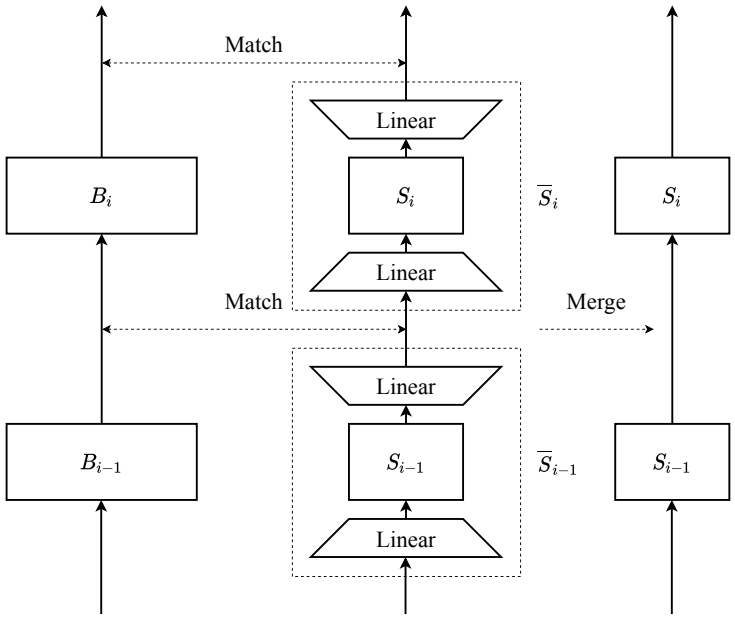

Figure 1. Illustrating "go wide, then narrow". Left panel: wide network B with building blocks B_i obtained by widening S_i in the thin network S. Middle panel: network $\bar{S}$ obtained by inserting appropriately sized linear transformation pairs between any two adjacent building blocks S_{i-1} and S_i in S. Right panel: merge all adjacent linear transformations in $\bar{S}$ to restore its architecture to S.

dimension, using more self-attention heads, or adding more hidden neurons in its feed forward network module.

Stage 2: Narrow learning. In this stage, we first construct a new network $\bar{S}$ by inserting a pair of *appropriately sized* linear transformations $\{M_{i,1}, M_{i,2}\}$ between two adjacent building blocks S_i and S_{i+1} in the thin network S (see the middle column of Figure 1 for an illustration of $\bar{S}$). The first linear transformation $M_{i,1}$ increases the output dimension of S_i to match the output dimension of B_i in the wide network B, and the second linear transformation $M_{i,2}$ reduces the dimension to its original size. Formally, the new network $\bar{S}$ can be written as $\bar{S} = \{S_1, M_{1,1}, M_{1,2}, S_2, \ldots, S_{n-1}, M_{n-1,1}, M_{n-1,2}, S_n\}$.

We group the modules in $\bar{S}$ as follows: $\bar{S}_1 = \{S_1, M_{1,1}\}$, $\bar{S}_i = \{M_{i-1,2}, S_i, M_{i,1}\}$ for $i = 2, \ldots, n-1$, and $\bar{S}_n = \{M_{n-1,2}, S_n\}$. Thus, $\bar{S} = \{\bar{S}_1, \bar{S}_2, \ldots, \bar{S}_n\}$. Next, for $i = 1, \ldots, n-1$, we sequentially train a set of subnetworks $\bar{S}^{(i)} = \{\bar{S}_1, \ldots, \bar{S}_i\}$ by minimizing the output discrepancy between $\bar{S}^{(i)}$ and subnetwork $B^{(i)} = \{B_1, \ldots, B_i\}$ in the wide network B. The instances in the training data are used as the inputs. Note that the weights of $B^{(i)}$ are fixed since the entire network B has been trained in the first stage. In addition, during this sequential training, the trained $\bar{S}^{(i)}$ is naturally served as initialization when proceeding to training $\bar{S}^{(i+1)}$. There are many ways to measure the output discrepancy between $\bar{S}^{(i)}$ and $B^{(i)}$. Typical choices include the mean squared error and KL divergence.

To achieve a better performance, when each training subnetwork $\bar{S}^{(i)}$, we may restart multiple times. In each restart, the most recently added building block S_i is randomly reinitialized. Finally, we choose the trained network which best mimics $B^{(i)}$ before proceeding to training $\bar{S}^{(i+1)}$.

Stage 3: Fine-tuning and merging. This is the final stage of our method. After layerwise imitation in the second stage, the network $\bar{S}^{(n)} = \bar{S}$ has been well initialized. We can further fine tune this network using the training labels. Afterwards, we merge all adjacent linear transformations in $\bar{S}$, including the native linear layers residing in its building blocks, Consequently, $\bar{S}$ is restored to the architecture of the original deep thin network S (see the illustration in Figure 1). Until then our algorithm is done. Optionally, one may restart fine-tuning several times, and then choose the model which has the minimum training error. Such a greedy choice usually works well since a thin model is supposed to have a strong regularization effect on its own.

We summarize our method in Algorithm 1. Here are several additional remarks. First, our method does not have to be more expensive than the normal knowledge distillation method since the imitation training in our method is quite light in each layer. In addition, we often can use an existing trained big model as the wide network in our method. Consequently, the wide learning stage of our method can be skipped. Moreover, the layers in our method do not have to exactly align to the layers of the trained neural models. For example, the wide network may be a 24-layer $\text{BERT}_{\text{LARGE}}$

Algorithm 1 go WIde, then Narrow (WIN)

Input: thin network $S = \{S_1, \ldots, S_n\}$; training data

Stage 1: Wide learning. Construct a wide network $B = \{B_1, \cdots, B_n\}$, where each B_i is obtained by widening S_i in network S, and train B until convergence.

Stage 2: Narrow learning. Construct another network $\bar{S} = \{\bar{S}_1, \ldots, \bar{S}_n\}$, where each $\bar{S}_i$ is obtained from wrapping up S_i by two appropriately sized linear transformations to match the output dimension of B_i.

for $i = 1$ **to** $n - 1$ **do**
  Train subnetwork $\bar{S}^{(i)} = \{\bar{S}_1, \ldots, \bar{S}_i\}$ by minimizing the output discrepancy between $\bar{S}^{(i)}$ and subnetwork $B^{(i)} = \{B_1, \ldots, B_i\}$ from the wide network B.
end for

Stage 3: Fine-tuning and merging. Use the training labels to fine tune network $\bar{S}^{(n)} = \bar{S}$, and then merge all adjacent linear layers in $\bar{S}$ to restore its architecture to S.

model, and the narrow network a 12-layer $\text{BERT}_{\text{BASE}}$ model. Then, during the imitation training, each layer in $\text{BERT}_{\text{BASE}}$ is mapped to two adjacent layers in $\text{BERT}_{\text{LARGE}}$.

3. Theoretical Analysis

In this section, we present a theoretical comparison between a layerwise imitation based scheme and the standard gradient descent training. The basic intuition underlying our analysis is that layerwise imitation breaks the learning of a deep thin network into a sequence of shallow subnetworks training, and hence avoids backpropagation through the entire deep thin network from top to bottom. This makes our method more suitable for training very deep networks, and also enables simpler theoretical analysis. Our theoretic results show that layerwise imitation yields a much tighter error bound compared with gradient descent.

3.1. Assumptions and Theoretical Results

Our analysis is built on the theory of mean field analysis of neural network (e.g., Song et al., 2018; Araújo et al., 2019; Nguyen & Pham, 2020). We start with the formulation of deep mean field network formulated by Araújo et al. (2019).

For notational conventions, let $S^m = \{S_1^m, S_2^m, \cdots, S_n^m\}$ and $B^M = \{B_1^M, B_2^M, \cdots, B_n^M\}$ be the thin and wide networks of interest, where we add the superscribes m and M to denote the number of neurons in each layer of the thin and wide networks, respectively. We assume the i-th layer of S^m and B^M are

$$S_i^m(\mathbf{z}) = \frac{1}{m}\sum_{j=1}^{m}\sigma\Big(\mathbf{z}, \boldsymbol{\theta}_{i,j}^S\Big), \quad B_i^M(\mathbf{z}) = \frac{1}{M}\sum_{j=1}^{M}\sigma\Big(\mathbf{z}, \boldsymbol{\theta}_{i,j}^B\Big),$$

where $\boldsymbol{\theta}_{i,j}^S$ and $\boldsymbol{\theta}_{i,j}^B$ are the weights of the thin and wide models, respectively. Here we also define

$$\sigma(\mathbf{z}, \boldsymbol{\theta}) = \boldsymbol{\theta}_1^\top \sigma_+(\mathbf{z}^\top \boldsymbol{\theta}_0), \quad \boldsymbol{\theta} = [\boldsymbol{\theta}_0, \boldsymbol{\theta}_1],$$

where $\sigma_+(\cdot)$ is some commonly used nonlinear element-wise mapping such as sigmoid.

In order to match the dimensions of the thin and wide models, we assume the input and output of both $S_i^m(\mathbf{z})$ and $B_i^M(\mathbf{z})$ of all the layers (except the output) have the same dimension d, so that $\mathbf{z} \in \mathbb{R}^d$ and $\boldsymbol{\theta}_{i,j}^S, \boldsymbol{\theta}_{i,j}^B \in \mathbb{R}^{(d+1)\times d}$ for all the neurons and layers. In practice, this can be ensured by inserting linear transform pairs as we have described in our practical algorithm (so that the S_i^m corresponds to the $\bar{S}_i$ in Figure1). In addition, for the sake of simplicity, the dimension of the final output is assumed to be one.

Giving a dataset $\mathcal{D} = \{\mathbf{x}_i, y\}_i$, we consider the regression problem of minimizing the mean squared error:

$$\mathcal{L}(F) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\big(F(\mathbf{x}) - y\big)^2\right], \tag{1}$$

via gradient descent with step size η and a proper random initialization. We define the output discrepancy between two models S and B to be

$$\mathbb{D}[S, B] = \sqrt{\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[\big(S(\mathbf{x}) - B(\mathbf{x})\big)^2\right]}.$$

Assumption 3.1. *Denote by B_{GD}^m and S_{GD}^M the result of running stochastic gradient descent on dataset $\mathcal{D}$ with a constant step size $\eta > 0$, for a fixed T steps.*

For both models, we initialize the parameters $\{\boldsymbol{\theta}_{i,j}^S\}_{j\in[m]}$ and $\{\boldsymbol{\theta}_{i,j}^B\}_{j\in[M]}$ in the i-th layer by drawing i.i.d. samples from a distribution ρ_i. We suppose ρ_i is absolute continuous and has bounded domain for $i \in [n]$.

Bounds of Deep Thin Networks Trained from Scratch Analyzing deep neural networks trained with gradient descent remains a ground challenge in theoretical deep learning. The few existing bounds (Araújo et al., 2019; Nguyen & Pham, 2020) depend rather poorly on the depth of networks, due to the difficulty of controlling the errors propagated through the layers during gradient descent. Here we leverage the mean field analysis from Araújo et al. (2019) to give an estimation of the discrepancy between the thin and wide models S_{GD}^m and B_{GD}^M.

Assumption 3.2. *Suppose the data and labels in $\mathcal{D}$ are bounded, i.e. $\|\boldsymbol{x}\| \le c$ and $|y| \le c$ for some $c < \infty$. And suppose the activation function σ_+ and its first and second derivatives are bounded.*

The boundedness assumptions are typical and (almost) always required in the theoretical literature on deep learning (e.g., Song et al., 2018; Araújo et al., 2019). Relaxing the boundedness assumptions could be possible but it brings more technical issues while without bringing additional insights. In practice, given that the size of observed data is finite, we can always assume that the data and weights are properly truncated and thus bounded. Actually, for image data, they are already bounded, and the trainable weights are usually initialized from a truncated distribution.

Proposition 3.3. ***(Discrepancy between the wide and thin networks trained by stochastic gradient descent)*** *Under assumption 3.2 and 3.1, we have*

$$\mathbb{D}[S^m_{\mathrm{GD}}, B^M_{\mathrm{GD}}] = \mathcal{O}_p\left(n \exp c_1(\exp(c_2 n))\left(\frac{1}{\sqrt{m}} + \frac{1}{\sqrt{M}} + \sqrt{\eta}\right)\right),$$

where $c_1, c_2 > 0$ are some positive constant, $\mathcal{O}_p(\cdot)$ denotes the big O notation in probability, and the randomness is w.r.t. the random initialization of gradient descent, and the random mini-batches of stochastic gradient descent.

The proof of this bound is based on the proof of Theorem 5.5 in Araújo et al. (2019); see Appendix for details. Because the m is small and n is large for deep thin networks, the bound above is dominated by $n \exp(c_1 \exp(c_2 n)) m^{-1/2}$, which decreases with the width m, but grows double exponentially with the depth n. The poor dependency on n is both due to the critical gradient vanish/exploding problem when backpropagating through deep networks and the mathematical challenge for analyzing deep networks under the mean field framework.

Breaking the Curse of Depth with Layerwise Imitation Now we show how our layerwise imitation algorithm can help training deep thin networks.

Assumption 3.4. *Denote by S^m_{WIN} the result of mimicking B^M_{GD} following Algorithm 1. When training S^m_{WIN}, we assume the parameters of S^m_{WIN} in each layer are initialized by randomly sampling m neurons from the the corresponding layer of the wide network B^M_{GD}. Define $B^M_{\mathrm{GD},[i:n]} = B^M_n \circ \cdots B^M_i$.*

Theorem 3.5. ***(Main Result)*** *Assume all the layers of B^M_{GD} are Lipschitz maps and all its parameters are bounded by some constant. Under Assumption 3.1, 3.2, 3.4, we have*

$$\mathbb{D}[S^m_{\mathrm{WIN}}, B^M_{\mathrm{GD}}] = \mathcal{O}_p\left(\frac{\ell_B n}{\sqrt{m}}\right),$$

where $\ell_B = \max_{i\in[n]} \left\| B^M_{\mathrm{GD},[i+1:n]} \right\|_{\mathrm{Lip}}$ and $\mathcal{O}_p(\cdot)$ denotes the big O notation in probability, and the randomness is w.r.t. the random initialization of gradient descent, and the random mini-batches of stochastic gradient descent.

The bound above depends on linearly on n and the maximum Lipschitz constant ℓ_B. Because it is expected that the wide network B^M_{GD} is easy to train and can closely approximate the underlying true map, the Lipschitz constant ℓ_B can be mostly depended on the true map in practice (rather than how deep B^M_{GD} is) and does not explode rapidly with n like the bound on $\mathbb{D}[S^m_{\mathrm{GD}}, B^M_{\mathrm{GD}}]$. An important future work is to develop rigours bounds for ℓ_B.

4. Related Work

Our method is deeply inspired by MobileBERT (Sun et al., 2020), which is a highly compact BERT variant designed for mobile applications with extreme memory and latency constraints. In its architecture design, the original BERT building block is replaced with a thin bottleneck structure. To train it, MobileBERT is first initialized by imitating the outputs of a well trained large BERT from layer to layer and then fine tuned. The main difference between MobileBERT and the proposed method is that in MobileBERT linear transformations are introduced with the bottleneck structures so they are part of the model and cannot be cancelled out by merging as in our method. In addition, the method here is generic and can be applied to any model training.

FitNets (Romero et al., 2014) also aim at training deep thin networks. In this work, a deep student network is first partially initialized by matching the output from its some chosen layer (guided layer) to the output from another chosen layer (hint layer) of a shallow teacher network. The chosen guided and hint layers do not have to be at the same depth since the teacher network is chosen to be much shallower than the student network. After the partial initialization, the whole student network is trained via knowledge distillation. The matching is implemented by minimizing a parameterized mean squared loss in which a parameterized regressor is applied to project the student's output such that the size of its output can match the size of the teacher's output. The major difference between FitNets and our method is that the introduced regressor in FitNets is not part of the student network architecture. It is discarded after training.

This kind of teacher-student paradigm can be traced back to knowledge distillation and its variants (Ba & Caruana, 2014; Hinton et al., 2015). The basic idea in knowledge distillation is to use both true labels and the outputs from a cumbersome model to train a small model. In the literature, the cumbersome model is usually referred to as teacher, and the small model student. The loss based on the teacher's outputs, that is, the so-called distillation loss, is linearly combined with the true labels based training loss as the final objective to train the student model. In the variants of

knowledge distillation, the intermediate outputs from the teacher model are further used to construct the distillation loss which is parameterized as in FitNets. Unlike knowledge distillation, our method uses a teacher model to initialize a student model rather than constructing a new training objective. After the initialization, the student model is trained as usual. Based on such a special initialization manner, we are able to establish a theoretic guarantee for our approach.

Our theory is built upon the mean field analysis for neural networks, which is firstly proposed by Song et al. (2018) to study two-layer neural networks and then generalized to deep networks by Araújo et al. (2019). The general idea of mean field analysis is to think of the network as an interacting particle system, and then study how the behavior of the network converges to its limiting case (as the number of neurons increases). It is shown by Araújo et al. (2019) that as the depth of a network increases, the stochasticity of the system increases at a double exponential scale with respect to its depth. This characterizes the problem of gradient explosion or vanish. On the other hand, they also establish the results which suggest that increasing the width of the network helps the propagation of gradient, as it reduces the stochasticity of the system. In our method, we first train a wide network that helps the propagation of gradients, and then force the thin network to mimic the wide network from layer to layer. Consequently, the negative influence of depth decreases from double exponential to linear.

5. Experiments

We conduct empirical evaluations by training state of the arts neural network models for image classification and natural language modeling. Our baselines include vanilla training methods for these models as shown in the literature as well as knowledge distillation. In addition, in what follows, following the convention in the literature and for the sake of convenience, we refer to the wide model in our method as teacher, and the thin model as student.

5.1. Image Classification

We train the widely used ResNet models (He et al., 2016) on the ImageNet dataset (Russakovsky et al., 2015) using our apporach and baseline methods.

5.1.1. Setup

Models. ResNet is build on a list of bottleneck layers (He et al., 2016). Each bottleneck layer consists of three modules: a projection 1x1 convolution to reduce the channel size to 1/4 of the input channels, a regular 3x3 convolution, and a final expansion 1x1 convolution to recover the channel size. The wide teacher model used in our method is constructed by increasing the channel size of the 3x3 convolution as in (Zagoruyko & Komodakis, 2016), and the remaining two 1x1 convolutions simply keep the increased channel size without projection or expansion.

Table 1. Model complexity comparison between the teacher and student models.

	Teacher		Student	
	FLOPs	Params	FLOPs	Params
ResNet50	11B	68M	4.1B	26M
ResNet50-1/2	2.9B	18M	1.1B	6.9M
ResNet50-1/4	0.75B	4.7M	0.29B	2.0M
ResNet101	23B	127M	7.9B	45M
ResNet101-1/2	5.8B	32M	2.0B	12M
ResNet101-1/4	1.5B	8.3M	0.53B	3.2M

The models that we evaluate include ResNet50, ResNet101 and their reduced versions: ResNet50-1/2, ResNet50-1/4, ResNet101-1/2 and ResNet101-1/4. For each model's reduced version, we apply the same reducing factor to all layers in that model. For example, ResNet50-1/2 means that the channel size of every layer in this model is half the channel size of the corresponding layer in ResNet50. The complexity numbers including FLOPs and parameter sizes for different models are collected in Table 1 for reference.

Vanilla training setting. We follow the training settings in (He et al., 2016). Each ResNet variant is trained with 90 epochs using SGD with momentum 0.9, batch norm decay 0.9, weight decay 1e-4, and batch size 256. The learning rate is linearly increased from 0 to 0.1 in the first 5 epochs, and then reduced by 10x at epoch 30, 60 and 80.

WIN setting. We naturally split ResNet into four big chunks or building blocks with respect to the resolution change, that is, with separations at conv2_x, con3_x, conv4_x, and con5_x. In the first stage of our method, the teacher network is constructed as 4x larger (in terms of the channel size of the 3x3 convolutions) than the corresponding student network, and trained with the vanilla setting. In the second stage, for training each building block in the student network, we run 10 epochs by minimizing the mean squared error between the output of the teacher and student network. The optimizer is SGD with momentum 0.9. The learning rate decayed from 0.1 to 0 under the cosine decay rule. After that, we fine tune the student network for 50 epochs by minimizing the Kullback-Leibler divergence from the teacher logits to student logits, with the learning rate decayed from 0.01 to 0 under the cosine decay rule. Note that the total number of training epochs here is 90, which is the same as in the vanilla training. We do not apply weight decay in the last two stages since the compact architecture of a thin network has already implied a strong regularization.

Table 2. ImageNet top-1 accuracy (%) by the models trained by the vanilla setting, knowledge distillation (KD), and our method.

Model	Vanilla	KD	WIN
ResNet50	76.2	76.8	**78.4**
ResNet50-1/2	72.2	72.9	**74.6**
ResNet50-1/4	64.2	65.1	**66.4**
ResNet101	77.5	78.0	**79.1**
ResNet101-1/2	74.6	75.5	**76.8**
ResNet101-1/4	68.1	69.1	**69.7**

Table 3. ImageNet top-1 accuracy (%) by different size teachers and their students trained with our method.

	2x		4x	
	Teacher	Student	Teacher	Student
ResNet50	78.4	78.4	78.6	78.2
ResNet50-1/2	76.0	74.6	77.6	75.0
ResNet50-1/4	70.5	66.4	74.4	67.5

5.1.2. Results

The evaluation results are collected in Table 2. The numbers listed in the table cells are the top-1 accuracy on the ImageNet dataset from the models trained by different methods: our method, the vanilla training, and knowledge distillation. The results show that our method significantly outperforms the baseline methods. Moreover, we would like to point out that ResNet50 trained by our method achieves an accuracy of 78.4% which is even higher than the accuracy of 77.5% from ResNet101 trained by the vanilla approach.

We conduct an ablation study to demonstrate the effect of the teacher model size. The results are shown in Table 3. For ResNet50, the 2x teacher performs almost equally well as the 4x teacher. The same observation holds for their students. However, for the thinner models ResNet50-1/2 and ResNet50-1/4, the models trained by the 2x teacher are worse than the models trained by the 4x teacher. We do not try an even larger teacher such as the 6x one because of the computational cost.

5.2. Language Modeling

In this task, we train BERT (Devlin et al., 2019), the state-of-the-art pre-training language model, using our method as well as the baseline methods as in the image classification tasks. Following Devlin et al. (2019), we firstly pre-train the BERT model using BooksCorpus (Zhu et al., 2015) and the Wikipedia corpus. Then we fine-tune this pre-trained model and evaluate on the Stanford Question Answering Dataset (SQuAD) 1.1 and 2.0 (Rajpurkar et al., 2016).

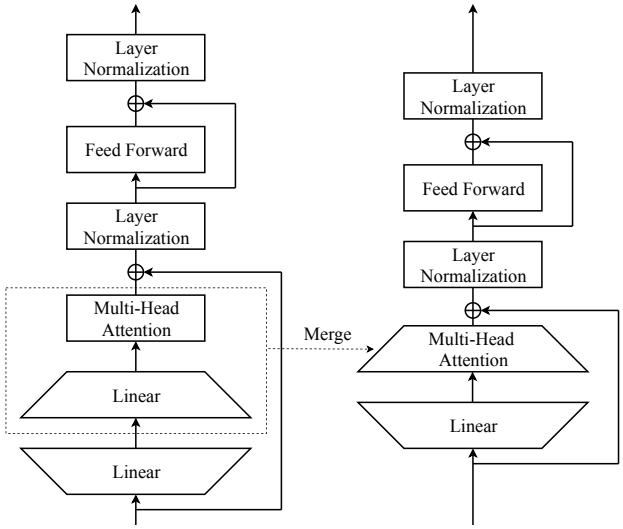

Figure 2. Adding linear transformation pairs into a thin BERT model. Left panel: a pair of linear transformation are inserted between any two adjacent transformer blocks. Right Panel: the linear transformation right next to the multi-head attention module (see the dashed box in the left panel) is merged before the training in the second stage, i.e., the stage of narrow learning, while the remaining linear transformation will be merged after fine-tuning when the whole training procedure is done. Thus, finally, the trained model has the exact same network architecture and number of parameters as the original thin model.

5.2.1. Setup

Models. The model that we are going to train here is $\text{BERT}_{\text{BASE}}$. It takes token embeddings as its inputs and contains 12 transformer blocks (Vaswani et al., 2017). Each transformer block consists of one multi-head self-attention module and one feed forward network module, which are followed by layer normalization and connected by skip connections respectively. On top of the transformer blocks, there is a classifier layer to make task-specific predictions.

The teacher model for our method is constructed by simply doubling the hidden size of every transformer block and also the width of every feed forward module in $\text{BERT}_{\text{BASE}}$. We keep the size of the teacher model's embedding the same as $\text{BERT}_{\text{BASE}}$'s and add a linear transformation right after teacher model's embedding to match its hidden size. Thus, the student model described below and the wider teacher model can share the same token embeddings as their inputs.

The way to construct the student model is illustrated in Figure 2. Specifically, taking the canonical $\text{BERT}_{\text{BASE}}$ model, we insert a pair of linear transformations between any two adjacent transformer blocks. We also put one extra linear layer over the last transformer block of $\text{BERT}_{\text{BASE}}$. The output size of the lower linear transformation is designed

Table 4. The results on the SQuAD dev datasets from the BERT models trained by our method, vanilla training and knowledge distillation (KD). †marks our runs with the official code.

Model	SQuAD 1.1 Exact Match	SQuAD 1.1 F1	SQuAD 2.0 Exact Match	SQuAD 2.0 F1
$\text{BERT}_{\text{BASE}}$ (Devlin et al., 2019)	80.8	88.5	74.2†	77.1†
$\text{BERT}_{\text{LARGE}}$ (Devlin et al., 2019)	84.1	90.9	78.7	81.9
Teacher	85.5	91.9	80.3	83.2
$\text{BERT}_{\text{BASE}}$ (Vanilla)	83.6	90.5	77.9	80.4
$\text{BERT}_{\text{BASE}}$ (KD)	84.2	90.8	78.9	81.4
$\text{BERT}_{\text{BASE}}$ (WIN)	**85.5**	**91.8**	**79.6**	**82.5**

to be the same as the output size of teacher model's transformer block, i.e., the teacher model's hidden size. To more efficiently train this student model, before the training, we merge the upper linear transformation into the fully-connected layers inside the multi-head attention module. After training, we can further merge the remaining lower linear transformation into the multi-head attention module. Similarly, we can also merge the extra linear transformation over the last transformer block into the final classifier layer. Hence, the final student model has the exact same network architecture and number of parameters as $\text{BERT}_{\text{BASE}}$.

Vanilla training setting. There are two training phrases for the BERT models: pre-training and fine-tuning. In the pre-training phrase, we train the model on the masked language modeling (MLM) and next sentence prediction (NSP) tasks using BookCorpus and Wikipedia corpus for 1 million steps with batch size of 512 and sequence length of 512. We use the Adam optimizer with the learning rate of 1e-4, β_1 = 0.9, β_2 = 0.999, weight decay of 0.01. The learning rate is linearly warmed up in the first 10,000 steps, and then linearly decayed. After pre-training, we enter the fine-tuning phrase. In this phrase, we fine tune all the parameters using the labeled data for a specific downstream task.

WIN setting. In the first stage of our method, we train the 2x wider teacher model using the vanilla method. In the second stage, we first copy the teacher model's token embeddings to the student model, and then progressively warm up the student's transformer blocks from layer to layer. In each step of this stage, we minimize the mean squared error between the output of the linear transformation after the student's transformer block, and the output of the teacher's corresponding transformer block. We train the first transformer block for 10k steps, the second for 20k steps until the 12th for 120K steps. Note that $\text{BERT}_{\text{BASE}}$ has 12 transformer block layers in total. Now we enter the fine-tuning stage. We follow the same vanilla training setting to pre-train this warmed-up model on MLM and NSP tasks. Finally, we fine tune the model for downstream tasks (no knowledge distillation is employed here).

Table 5. The results on the SQuAD 1.1 dev dataset from the $\text{BERT}_{\text{BASE}}$-1/2 models trained by our method and baselines.

Model	Exact Match	F1
$\text{BERT}_{\text{BASE}}$-1/2 (Vanilla)	78.9	86.3
$\text{BERT}_{\text{BASE}}$-1/2 (KD)	80.1	87.4
$\text{BERT}_{\text{BASE}}$-1/2 (WIN)	81.4	88.6

5.2.2. Results

We evaluate the models using the SQuAD 1.1 and 2.0 datasets. Results are shown in Table 4. Note that $\text{BERT}_{\text{BASE}}$ trained using our vanilla setting here outperforms $\text{BERT}_{\text{BASE}}$ (Devlin et al., 2019) by a large margin. The reason for the improvement is that we pre-train the model with sequence length of 512 for all steps, while Devlin et al. (2019) pre-train the model with sequence length of 128 for 90% of the steps and sequence of 512 for the rest 10% steps. The better training result establishes a stronger baseline. $\text{BERT}_{\text{BASE}}$ trained by our method further beats this stronger baseline by 1.9 exact match score and 1.3 F1 score on SQuAD 1.1, and 1.7 exact match score and 2.1 F1 score on SQuAD 2.0. Actually, $\text{BERT}_{\text{BASE}}$ trained by our method is comparable with $\text{BERT}_{\text{LARGE}}$ by vanilla training.

We also run experiments with a thinner student model called $\text{BERT}_{\text{BASE}}$-1/2 which halves the hidden size and width of the feed-forward network of $\text{BERT}_{\text{BASE}}$ in every layer. As shown in Table 5, $\text{BERT}_{\text{BASE}}$-1/2 trained by our method significantly surpasses the same model trained by the vanilla method and knowledge distillation.

In addition, we conduct an ablation study to demonstrate the effect of the timing for merging linear transformations. In our approach, we suggest to merge all adjacent linear layers after the fine-tuning stage when the whole training procedure is done. One may notice that, alternatively, we can merge the linear layers right after the narrow learning stage.

Table 6. The results on the SQuAD 1.1 dev dataset, comparing whether to merge linear transformations after the fine-tuning phrase (MAF) or pre-training phrase (MAP) in our method.

Model	Exact Match	F1
$\text{BERT}_{\text{BASE}}$ (MAF)	85.5	91.8
$\text{BERT}_{\text{BASE}}$ (MAP)	85.1	91.5
$\text{BERT}_{\text{BASE}}$-1/2 (MAF)	81.4	88.6
$\text{BERT}_{\text{BASE}}$-1/2 (MAP)	81.4	88.5

So this will be before the fine-tuning stage. By using either of these two merging methods, the network structure and model size are the same. We compare these two merging methods and present results in Table 6. From the comparison, merging after fine-tuning seems to have slightly better results. The improvement are minor but consistent.

6. Conclusion

We proposed a general method for efficiently training deep thin networks. Our method can be simply described as "go wide, then narrow". A theoretic guarantee is developed for our method by using mean field analysis for neural networks. Empirical results on training image classification and language processing models demonstrate the advantage of our method over these two baseline methods: training deep thin networks from scratch using backpropagation as in the literature, and training with the state of the art knowledge distillation method. Our method is complimentary to existing model compression techniques including quantization and knowledge distillation. One can combine our method with these techniques to obtain an even better compact model. For the future work, we are interested at searching for a different initialization or optimization method which is not teacher based while still enjoying a similar theoretic guarantee. If we can make it, we will be able to save the cost of training a large teacher model.

References

Allen-Zhu, Z. and Li, Y. Backward feature correction: How deep learning performs deep learning. *arXiv preprint arXiv:2001.04413*, 2020.

Allen-Zhu, Z., Li, Y., and Song, Z. A convergence theory for deep learning via over-parameterization. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 242–252, 2019.

Araújo, D., Oliveira, R. I., and Yukimura, D. A mean-field limit for certain deep neural networks. *arXiv preprint arXiv:1906.00193*, 2019.

Ba, J. and Caruana, R. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pp. 2654–2662, 2014.

Bengio, Y., Simard, P., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Brock, A., Donahue, J., and Simonyan, K. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

Du, S. S., Zhai, X., Poczos, B., and Singh, A. Gradient descent provably optimizes over-parameterized neural networks. In *International Conference on Learning Representations*, 2019.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al. Searching for mobilenetv3. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1314–1324, 2019.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. Squeezenet: AlexNet-level accuracy with 50$\times$ fewer parameters and $<$0.5MB model size. *arXiv preprint arXiv:1602.07360*, 2016.

Le, Q. V., Ranzato, M., Monga, R., Devin, M., Chen, K., Corrado, G. S., Dean, J., and Ng, A. Y. Building high-level features using large scale unsupervised learning.

In *Proceedings of the 29th International Conference on Machine Learning*, 2012.

Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th International Conference on Machine Learning*, pp. 609–616, 2009.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. 2017.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pp. 6389–6399, 2018.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019.

Nguyen, P.-M. and Pham, H. T. A rigorous framework for the mean field limit of multilayer neural networks. *arXiv preprint arXiv:2001.11443*, 2020.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International conference on machine learning*, pp. 1310–1318, 2013.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Romero, A., Ballas, N., Kahou, S. E., Chassang, A., Gatta, C., and Bengio, Y. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3): 211–252, 2015.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

Song, M., Montanari, A., and Nguyen, P. A mean field view of the landscape of two-layers neural networks. *Proceedings of the National Academy of Sciences*, 115:E7665–E7671, 2018.

Sun, Z., Yu, H., Song, X., Liu, R., Yang, Y., and Zhou, D. MobileBERT: a compact task-agnostic BERT for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020.

Tan, M. and V. Le, Q. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Wortsman, M., Farhadi, A., and Rastegari, M. Discovering neural wirings. In *Advances in Neural Information Processing Systems*, pp. 2684–2694, 2019.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Ye, M., Gong, C., Nie, L., Zhou, D., Klivans, A., and Liu, Q. Good subnetworks provably exist: Pruning via greedy forward selection. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Zagoruyko, S. and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pp. 19–27, 2015.

A. Proof of Proposition 3.3

This result directly follows Theorem 5.5 in Araújo et al. (2019). Let B^{∞}_{GD} denote the infinitely wide network trained by gradient descent in the limit of $M \to \infty$. By the results in Theorem 5.5 of Araújo et al. (2019), we have

$$\mathbb{D}[S^m_{\mathrm{GD}},\ B^{\infty}_{\mathrm{GD}}] = \mathcal{O}_p\left(n\exp(c_1\exp(c_2 n))\left(\frac{1}{\sqrt{m}} + \sqrt{\eta}\right)\right),$$

where we explicitly give the dependency of constant $C_{5.5}$ in Araújo et al. (2019) on the depth n, because $C_{5.5} = O(\exp(c_1 \times C_{B.16}))$, where $C_{B.16} = \mathcal{O}(\exp(c_2 n))$ and c_1 is some positive constant. See Lemma 12.2 in Araújo et al. (2019) for details.

Similarly,

$$\mathbb{D}[S^m_{\mathrm{GD}},\ B^{\infty}_{\mathrm{GD}}] = \mathcal{O}_p\left(n\exp(c_1\exp(c_2 n))\left(\frac{1}{\sqrt{M}} + \sqrt{\eta}\right)\right).$$

Combining this, we have

$$\begin{aligned}\mathbb{D}[B^M_{\mathrm{GD}},\ B^M_{\mathrm{GD}}] &\le \mathbb{D}[S^m_{\mathrm{GD}},\ B^{\infty}_{\mathrm{GD}}] + \mathbb{D}[B^M_{\mathrm{GD}},\ B^{\infty}_{\mathrm{GD}}] \\ &= \mathcal{O}_p\left(n\exp(c_1\exp(c_2 n))\left(\frac{1}{\sqrt{m}} + \frac{1}{\sqrt{M}} + \sqrt{\eta}\right)\right).\end{aligned}$$

B. Proof of Theorem 3.5

Assumption 3.4 *Denote by S^m_{WIN} the result of mimicking B^M_{GD} following Algorithm 1. When training S^m_{WIN}, we assume the parameters of S^m_{WIN} in each layer are initialized by randomly sampling m neurons from the the corresponding layer of the wide network B^M_{GD}. Define $B^M_{\mathrm{GD},[i:n]} = B^M_n \circ \cdots B^M_i$.*

Theorem 3.5 *Assume all the layers of B^M_{GD} are Lipschitz maps and all its parameters are bounded by some constant. Under the assumptions 3.1, 3.2, 3.4, we have*

$$\mathbb{D}[S^m_{\mathrm{WIN}}, B^M_{\mathrm{GD}}] = \mathcal{O}_p\left(\frac{\ell_B n}{\sqrt{m}}\right),$$

where $\ell_B = \max_{i\in[n]}\left\|B^M_{\mathrm{GD},[i+1:n]}\right\|_{\mathrm{Lip}}$ and $\mathcal{O}_p(\cdot)$ denotes the big O notation in probability, and the randomness is w.r.t. the random initialization of gradient descent, and the random mini-batches of stochastic gradient descent.

Proof. To simply the notation, we denote B^M_{GD} by B^M and S^m_{WIN} by S^m in the proof. We have

$$\begin{aligned}B^M(\mathbf{x}) &= (B^M_n \circ B^M_{n-1} \circ ... \circ B^M_1)(\mathbf{x}) \\ S^m(\mathbf{x}) &= \left(S^m_n \circ S^m_{n-1} \circ ... \circ S^m_1\right)(\mathbf{x}).\end{aligned}$$

We define

$$B^M_{[k_1:k_2]}(\mathbf{z}) = (B^M_{k_2} \circ B^M_{k_2-1} \circ ... \circ B^M_{k_1})(\mathbf{z}),$$

where $\mathbf{z}$ is the input of $B^M_{[k_1:k_2]}$. Define

$$\begin{aligned}F_0(\mathbf{x}) &= \left(B^M_n \circ ... \circ B^M_3 \circ B^M_2 \circ B^M_1\right)(\mathbf{x}) \\ F_1(\mathbf{x}) &= \left(B^M_n \circ ... \circ B^M_3 \circ B^M_2 \circ S^m_1\right)(\mathbf{x}) \\ F_2(\mathbf{x}) &= \left(B^M_n \circ ... \circ B^M_3 \circ S^m_2 \circ S^m_1\right)(\mathbf{x}) \\ &\cdots \\ F_n(\mathbf{x}) &= \left(S^m_n \circ ... \circ S^m_3 \circ S^m_2 \circ S^m_1\right)(\mathbf{x}),\end{aligned}$$

following which we have $F_0 = B^M$ and $F_n = S^m$, and hence

$$\mathbb{D}[S^m, B^M] = \mathbb{D}[F_n, F_0] \le \sum_{k=1}^{n}\mathbb{D}[F_k, F_{k-1}].$$

Define $\ell_{i-1} := \left\| B^M_{[i:n]} \right\|_{\text{Lip}}$ for $i \in [n]$ and $\ell_n = 1$. Note that

$$\begin{aligned}\mathbb{D}[F_1, F_0] &= \sqrt{\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[\left(B^M_{[2:n]} \circ B^M_1(\mathbf{x}) - B^M_{[2:n]} \circ S^m_1(\mathbf{x})\right)^2\right]} \\ &\leq \ell_1 \sqrt{\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[\left(B^M_1(\mathbf{x}) - S^m_1(\mathbf{x})\right)^2\right]}\end{aligned}$$

By the assumption that we initialize $S^m_1(\mathbf{x})$ by randomly sampling neurons from $B^M_1(\mathbf{x})$, we have, with high probability,

$$\sqrt{\mathbb{E}_{x\sim\mathcal{D}}\left[\left(B^M_1(\mathbf{x}) - S^m_1(\mathbf{x})\right)^2\right]} \leq \frac{c}{\sqrt{m}},$$

where c is constant depending on the bounds of the parameters of B^M. Therefore,

$$\mathbb{D}[F_1, F_0] = \mathcal{O}_p\left(\frac{\ell_1}{\sqrt{m}}\right).$$

Similarly, we have

$$\mathbb{D}[F_k, F_{k-1}] = \mathcal{O}\left(\frac{\ell_k}{\sqrt{m}}\right), \quad \forall k = 2, \ldots, n.$$

Combine all the results, we have

$$\mathbb{D}[B^M, S^m] = \mathcal{O}\left(\frac{n \max_{k\in[n]} \ell_k}{\sqrt{m}}\right).$$

□

Remark Since the wide network B^M_{GD} is observed to be easy to train, it is expected that it can closely approximate the underlying true function and behaves nicely, hence yielding a small ℓ_B. An important future direction is to develop rigorous theoretical bounds for controlling ℓ_B.