Majority Kernels: An Approach to Leverage Big Model Dynamics for Efficient Small Model Training

Hanna Mazzawi
Google Research, NY
mazzawi@google.com

Pranjal Awasthi[1]
Google Research, CA
pranjalawasthi@google.com

Xavi Gonzalvo[1]
Google Research, NY
xavigonzalvo@google.com

Srikumar Ramalingam
Google Research, NY
rsrikumar@google.com

Abstract

Recent breakthroughs and successful deployment of large language and vision models in a constrained environment predominantly follow a two phase approach. First, large models are trained to achieve peak performance, followed by a model shrinking method to meet hardware constraints; Methods like distillation, compression or quantization help leverage the highly performant large models to induce smaller performant ones. Formally, this can be seen as the problem of identifying an optimal model of size n from a larger model of size $k \cdot n$, where $k > 1$ is the overparameterization factor. This paper explores the hypothesis that a single training run can simultaneously train a larger model for performance and derive a smaller model for deployment.

Our contribution is an effective architectural change, namely, *Majority Kernels* that is compatible with the main standard architectures such as multi-layer perceptrons (MLPs), Residual networks (ResNets), and Transformers. We demonstrate that applying our technique can modify the training dynamics resulting in performance gains across architectures and tasks while maintaining the inference performance consistent. Furthermore, our approach adds minimal overhead to the cost incurred (wall clock time) at training time. The proposed approach shows strong performance on a wide variety of datasets and models, even outperforming strong baselines such as distilled ensembles as well as combinatorial optimization methods based on submodular optimization.

1 Introduction

Overparametrized models have become a cornerstone in advancing deep learning, particularly when trained with first-order methods. The empirical evidence suggests that such models often demonstrate superior performance, a trend that persists without evident saturation points, assuming that data of sufficient quality and quantity is available [Kaplan et al., 2020]. The benefits extend beyond mere accuracy improvements; these large models enhance optimization stability and enable more robust generalization in diverse scenarios. This approach aligns with ongoing efforts to push the boundaries of deep learning capabilities through continued growth in model size. This shift towards larger models is evident from the early influential works in computer vision with CNNs [Krizhevsky et al., 2012, Szegedy et al., 2015] and ResNets [He et al., 2016] to the transformer architectures in language modeling [Vaswani et al., 2017, Brown et al., 2020, Chowdhery et al., 2022, Chen et al., 2022].

---

[1]Equal contribution.

While larger models are pivotal for achieving peak performance in deep learning, their practical deployment, especially on resource-constrained devices like smartphones, necessitates consideration of the model's footprint. These devices impose limits on the number of parameters and the computational cost. This can be effectively addressed by a two-step process. First, train an overparameterized model; then, employ post-training techniques to compress and tailor the model to meet specific operational constraints. This strategy allows for the benefits of large models to be realized in environments with stringent resource limitations [Cai et al., 2019]. Techniques like model pruning and quantization are designed to create models with reduced memory needs, maintaining performance while fitting into more restricted environments [LeCun et al., 1989, Han et al., 2015, Frankle and Carbin, 2018, Cai et al., 2020, Nagel et al., 2020]. Similarly, model distillation focuses on training smaller models that can faithfully approximate a larger model (typically of the same architecture) [Hinton et al., 2006, Buciluǎ et al., 2006]. This approach of post-training optimization plays a vital role in making advanced models viable for everyday applications.

This paper introduces a novel concept, questioning whether it is possible to streamline the above two stage process into a single training run. We explore the feasibility of simultaneously conducting larger-scale training while also producing a smaller, immediately deployable model for inference. This concept involves increasing the size of a compact model in such a way that it incurs minimal additional training overhead, yet reverts to its original, smaller size for inference.

A common method to increase the model size to achieve peak performance is ensembling (see Appendix C for more related work). Given a base model $f(x; \theta)$ with $\theta \in \mathbb{R}^n$, an effective way to improve the model is by training k copies independently and average their predictions. For the ensembling method, some techniques aim to mitigate the use of extra compute and memory (without needing a second stage to reduce the size),

- Snapshot ensembling Huang et al. [2017] removes the need to train k independent copies of the base model $f(x; \theta)$, reducing the training compute to be somewhat comparable with the original model training requirements. However, this algorithm does not solve the increase in compute and memory for inference.

- Mixture of experts [Shazeer et al., 2017], An MOE creates a model with k experts each structurally equivalent to $f(x; \theta)$. This effectively creates an overparameterized model with number of parameters approximately being $k \cdot n$, and comparable compute. However, while computation cost per inference remains similar to the original model, memory requirements increase due to the larger parameter count.

- Bayesian neural networks (BNN) [Magris and Iosifidis, 2023] can be seen as having multiple models $f(x; \theta)$, where θ is sampled from a posterior distribution $p(\theta|\mathcal{D})$ given the data $\mathcal{D}$. During training and inference, we draw samples of θ to capture the uncertainty and variability in the model predictions. However, inference requires multiple runs of the model, exceeding the original compute cost. Additionally, while there is recent work about memory footprint efficiency [Dusenberry et al., 2020], regular BNNs have to store parameters and their uncertainties which can make the dimension of θ be larger than $2n$.

In this work we make the following contributions.

- We present *Majority Kernels* (MK), a novel algorithm that increases the size of training models similarly to ensembling. When applied to a base model, this approach expands its parameters while maintaining the same inference compute and memory requirements as the original model, with only a minimal increase in training computation.

- We theoretically analyze the proposed algorithm through the lenses of implicit regularization. We demonstrate how the training dynamic change when applying our technique (in this context, training dynamics from an optimization perspective, involve analyzing how the model's parameters converge towards optimal values over time, influenced by factors such as learning rate, gradient behavior, and the chosen optimization algorithm). This helps us understand how the algorithm not only naturally limits the complexity of the model class but also seeks out more stable and generalizable solutions by focusing on flatter regions of the loss landscape.

- We present an extended empirical analysis to explore the efficacy of our algorithm across different architectures and datasets, showcasing their role in facilitating implicit overparameterized training. We compare our algorithm with strong baselines such as

- Distilled ensembles - a baseline that requires orders of magnitude more compute.
- Combinatorial optimization methods - a baseline that tries to streamline the above mentioned two phases when the increase in capacity is done via the model's dimension.

Our algorithm is remarkably simple yet effective; It is particularly suitable for large language models due to its negligible compute overhead during training and its lack of impact on inference compute and memory.

2 The Majority Kernels Algorithm

In this section we introduce the Majority Kernels (MK) algorithm. This approach involves training each layer of a Deep Neural Network (DNN) with an expanded version of their internal kernels. During training, MK aggregates these expanded kernels by randomly averaging extra parameters into the layer's original dimensions. At the inference stage, the kernel reverts to the average of these expanded versions.
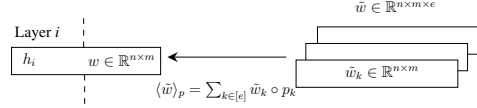

Figure 1.1: The majority kernels for the i-th layer.

It is well understood that ensembling of models often produces a model that is superior and more robust as compared to the base models [Huang et al., 2017, Fort et al., 1912]. Consider a model $f : \mathcal{X} \to \mathbb{R}^k$, where $\mathcal{X} \subseteq \mathbb{R}^d$ represents the domain of the data distribution. Define f as an ensemble of e different models: $f(x; \theta) = \frac{1}{e} \sum_{i=1}^{e} f_i(x; \theta_i)$ for $x \in \mathcal{X}$. Although typically ensembling is done in the model (function) space, if one could do ensembling in the parameter space itself then $f(x; \theta)$ would correspond to a powerful model that is also small in size. However, naive ensembling in the parameter space often performs poorly as the parameters of the different models might not be aligned along the same local optima. Our algorithm maintains parameter alignment by using stochastic weighted averaging throughout training, while using using average for inference. More formally,

Assume a multilayered model where each layer is defined as $h : \mathbb{R}^n \to \mathbb{R}^m$:

$$h(x) = \phi(xw + b),$$

where $w \in \mathbb{R}^{n \times m}$, $b \in \mathbb{R}^m$, and ϕ is a non-linear activation function like ReLU.

MK maintains the dimensionality of $h : \mathbb{R}^n \to \mathbb{R}^m$ but it uses an extended kernel $\widetilde{w} \in \mathbb{R}^{n \times m \times e}$ that allows the learning over an order of magnitude larger than the original kernel w (see Figure 1.1). The way the extended kernel $\widetilde{w}$ is used is as follows,

$$h(x) = \phi(x\langle\widetilde{w}\rangle_p + b) \quad s.t. \quad \langle\widetilde{w}\rangle_p = \sum_{k \in [e]} \widetilde{w}_k \circ p_k, \tag{2.1}$$

where $\circ$ indicates pointwise multiplication, $\widetilde{w}_k \in \mathbb{R}^{n \times m}$ refers to the k-th extended dimension of $\widetilde{w}$ and $p \in \mathbb{R}^{n \times m \times e}$ is the probability matrix used during training. The probability matrix is constructed as follows: For each pair (i, j), consider the vector $(p_{ij1}, \ldots, p_{ije})$ generated by drawing each component of p_{ijk} for $k \in [e]$ independently from an exponential distribution. We then normalize such that the sum of components equal 1 and set:

$$(p_{ij1}, \ldots, p_{ije}) \leftarrow \frac{1}{\sum_{k=1}^{e} p_{ijk}} (p_{ij1}, \ldots, p_{ije}). \tag{2.2}$$

This approach guarantees that the extra parameters are leveraged during training by exploring a ball around the mean, while during inference, the theoretical mean is used. In that case (uniform $p_{i,j} = \frac{1}{e} \cdot \mathbf{1}$)[2], and for simplicity $\langle\widetilde{w}\rangle = \frac{1}{e} \sum_{k \in [e]} \widetilde{w}_k$.

Algorithm 2.1 describes the training procedure for an architecture agnostic MK approach. The core idea is to use a stochastic version of the extended kernel $\langle\widetilde{w}\rangle_p$ during training, and a final version for inference using a uniform p (i.e., using $\langle\widetilde{w}\rangle$).

---

[2]Note that $\mathbf{1}$ is a matrix with 1's in all positions.

Algorithm 2.1 Majority kernels training algorithm.

For every layer l: Initialize $\widetilde{w} \in \mathbb{R}^{n \times m \times e}$, $b \in \mathbb{R}^m$
while $s < \text{max\_steps}$ **do**
  $B \leftarrow \text{NewBatch}()$
  $p \leftarrow \text{NormalizedExponentialRandom}()$ { Equation 2.2}
  $\text{ForwardPass}(\langle \widetilde{w} \rangle, p, b, B)$
  $\widetilde{w}, b \leftarrow \text{BackwardPass}(B)$
  $s += 1$
end while
return $\langle \widetilde{w} \rangle$, b {Return trained parameters for inference}

Experiment type	Improvement	Vanilla cost	MK cost	Ensemble cost
Fully Connected (Cifar10)	+2.07% in accuracy	1.2 CPU	1.4 CPU (+16.67%)	3.6 CPU (+300%)
Convolutional (Imagenet)	+1.01% in accuracy	11.15 TPU	12.18 TPU (+9.24%)	33.45 TPU (+300%)
Transformers (T5)	+0.75% on avg. Glue	5.9 TPU	5.95 TPU (+0.85%)	17.7 TPU (+300%)

Table 1: This table summarizes the relative performance improvements (averaged over various settings) for the various domains or architectures compared to vanilla training. Additionally, it presents the cost (in hours) of our training algorithm (with expansion factor 3, i.e., the size of the model is 3 times larger) compared to regular training and compared to the cost of vanilla ensemble of size 3. In Section 4, we present full detailed results of the above and results against comparing with strong baselines.

3 Implicit Gradient Regularization

BEA has been used in many settings to uncover inductive training biases of various optimizers (e.g. gradient descent Barrett and Dherin [2020], SGD Smith et al. [2021]; momentum Ghosh et al. [2023]; Adam and RMSProp Cattaneo et al. [2023]), various architectures (e.g., GANs Rosca et al. [2021]; diffusion models Gao et al. [2023]), and various training settings like continual-learning Dherin [2023], distributed and federated learning Barba et al. [2021] and progressively growing networks Mazzawi et al. [2024].

As an example, in the paper [Barrett and Dherin, 2020], using Backward Error Analysis (BEA), have demonstrated that for any loss function that is sufficiently differentiable (L), the process of gradient descent actually follows an adjusted loss surface, represented as $\tilde{L}$. This adjusted loss surface during training is defined by the as $\tilde{L}(\theta) = L(\theta) + \frac{\ell}{4}\|\nabla_\theta L(\theta)\|^2$ where ℓ is the learning rate.

In this equation, the additional term serves as a regularization component, promoting parameter points where the gradient is low, potentially indicating flatter minima.

In this section we analyze our algorithm using BEA to uncover inductive training biases. For MK, the parameters of the model are defined as $\bar{\theta}_p = (\langle \widetilde{w}^{(1)} \rangle_{p_1}, \ldots, \langle \widetilde{w}^{(L)} \rangle_{p_L})$ and $\bar{\theta}$ when using uniform element-wise $p = 1/e$. As in [Barrett and Dherin, 2020], we define a vector field, in this case on the extended set of parameters $f(\bar{\theta}_p) = -\nabla L(\bar{\theta}_p)$ and $f(\bar{\theta}) = -\nabla L(\bar{\theta})$. Our goal is to uncover the modified loss function in which the gradient decent algorithm follows to examine its biases.

Theorem 3.1 (Backward Error Analysis). Let L be a sufficiently differentiable function on the parameter space $\theta \in \mathbb{R}^n$. The modified loss when using Majority Kernels is,

$$\tilde{L}_{MK}(\bar{\theta}) = L(\bar{\theta}) + \frac{\ell}{4}\|\nabla L(\bar{\theta}) + \nabla^2 L(\bar{\theta}) \cdot \epsilon\|^2 + \nabla L(\bar{\theta}) \cdot \epsilon.$$

where $\epsilon \in \mathbb{R}^n$ is is the random perturbation of the virtual parameters.

See Appendix A for the proof of the theorem. Furthermore, in Appendix B, we further analyze our algorithm using the sharpness aware minimization, and the PAC frameworks. Additionally, this appendix proves Lemma B.2 showing that without the random probabilities in the algorithm, we fall back on the small model's training dynamics, and lose the leverage of the extra parameters.

4 Experiments

In this section we present empirical results comparing our proposed method against a variety of baselines across various architectures (see the Appendix for more experimental results, including experiments on fully connected networks).
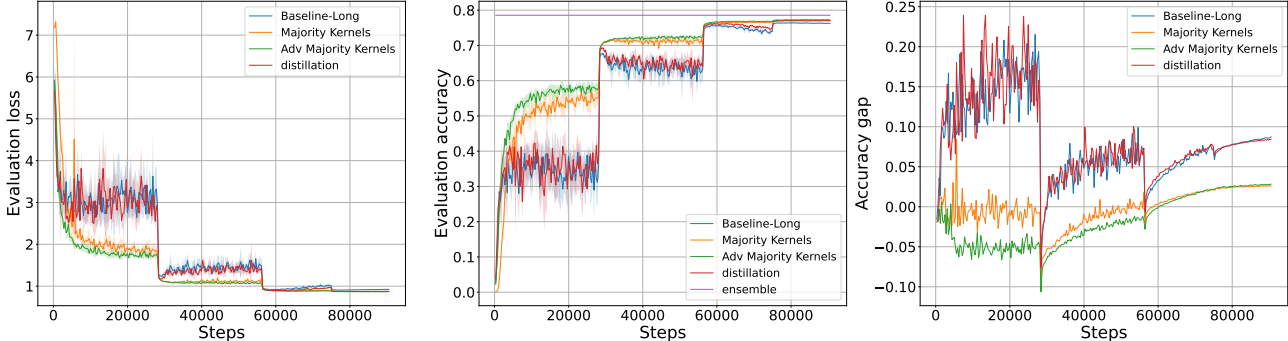

Figure 4.2: Test curves for BASELINE-LONG *(blue),* MAJORITY KERNELS *(orange),* ADV-MAJORITY KERNELS *(green),* DISTILLED-BASELINE *(red) and between (purple). It is easy to see that the generalization gap (accuracy gap between train and test above) is better in our optimization throughout training with better final performance.*

Algorithm	Test accuracy
BASELINE	76.1
BASELINE-LONG	76.43 ± 0.076
MAJORITY KERNELS	$\mathbf{77.20} \pm 0.068$
ADV-MAJORITY KERNELS	$\mathbf{77.33} \pm 0.11$
DISTILLED-BASELINE	77.05 ± 0.080
ENSEMBLE-BASELINE	78.56 ± 0.041

Figure 4.3: Results on ImageNet for the various algorithms.

4.1 Convolutional Networks

In this section, we present our results for running our experiments on Imagenet. We experiment with Imagenet on a ResNet50 architecture He et al. [2016].

In our experiment, we will compare the following algorithms,

- **BASELINE**. Training of ResNet50 based on the recipe in He et al. [2016]. Training for 90 epochs with batches of size 256, SGD with momentum as an optimizer. Our base learning rate is 0.1 and we have step decay of 0.1 every 30 epochs, we use weight decay of 0.0001. We use the standard data augmentation for ImageNet while training: we crop a random segment of the image, and scale it to standard input size of 224×224, along with random horizontal flipping of the images.

- **BASELINE-LONG**. Similar to BASELINE but trained for longer (just like majority kernels), that is, trained for 330 epochs, with learning rate drop at epochs 90, 180 and 240.

- **MAJORITY KERNELS**. The MK algorithm is implemented on the Baseline model. Due to MK's requirement for more steps to converge, we adopt the BASELINE-LONG configuration. Our expansion factor is $e = 3$.

- **ADV-MAJORITY KERNELS**. A modification of MK (with expansion factor $e = 3$), where an adversarial element is injected into the random probability at each training step. Details in Appendix D.

5

- **ENSEMBLE-BASELINE**. Evaluating true ebsemble performance by training three different BASELINE-LONG models and ensembling them.
- **DISTILLED-BASELINE**. BASELINE-LONG includes an additional knowledge distillation loss during training, where ENSEMBLE-BASELINE serves as the teacher.

Table 4.3 summarizes the results, showing that our MK algorithm outperforms others in performance. Notably, MK exhibits the most effective balance between training and test performance, highlighting the benefits of its additional regularization and its implicit stochastic sharpness optimization behavior (Section B). These effects are evident in the train and test accuracy curves during training, as depicted in Figure 4.2. Finally, our algorithm increases the training time on a TPU 4x4 by 9.24% compared to at least 400% increase of compute for the distilled baseline.

4.2 Transformer Networks

In this section, we apply MK to language tasks, focusing on fine-tuning downstream tasks using MK-enhanced pretrained T5 models. We expand a pretrained model's kernels into majority kernels by replicating them multiple times, forming our initial MK pretrained model. Using this model, we fine-tune on GLUE tasks with our MK optimization algorithm and present the results for various algorithms:

- **BASELINE**. Fine-tuning a pretrained T5 language model. This model is the T5 model with the configuration "Small", and is pretrained as described in Raffel et al. [2019].
- **MAJORITY KERNELS**. Starting with a small T5 pretrained model (refer to Raffel et al. [2019]), we convert its kernels into MK kernels with expansion factor $e = 3$. We then fine-tune it using our MK algorithm.

Table 2 summarizes the results for GLUE language tasks. A full breakdown for the various tasks and some extra details can be found in Appendix E. To make sure the performance is not tied to one pretrained checkpoint, for every run performed below we used a new pretrained checkpoint. Our training cost increase for T5 was negligable. Finally, the appendix holds comparison with Model Soups Wortsman et al. [2022] - a baseline the performs ensembling in the parameter space.

Algorithm	Glue average
BASELINE	80.3 ± 0.12
MAJORITY KERNELS	$\mathbf{80.9} \pm 0.3$

Table 2: Results on GLUE language tasks for our algorithm compared to vanilla training.

Appendix E contains figures illustrating the evaluation curves for various tasks during BASELINE and MAJORITY KERNELS training. A key observation is that, unlike previous experiments where our algorithm required extra steps, in this case, it achieves higher performance more rapidly. Regularization limits the model's complexity, discouraging it from fitting too closely to the training data of the primary task, which can include noise or irrelevant patterns. This ensures the model captures more general patterns, making it less tailored to the nuances of the primary task but more adaptable and easily fine-tuned for new, related tasks, as it has not over-committed to the specifics of the original training data.

5 Conclusions

The MK algorithm is a robust advancement in machine learning as it achieves computational efficiency during training, automates the distillation of inference parameters, and improves parameter weight optimization, which in turn smooths the training loss landscape to avoid local minima. Additionally, the MK algorithm's implicit regularization analysis reveals a beneficial second-order dependency in the modified loss, introducing stochastic regularization for smoother optimization and a bias towards flatter minima. This sharpness analysis indicates the algorithm averages out training randomness, leading to a stable and comprehensive representation of the model, particularly advantageous when capturing diverse data aspects.

References

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. CoRR, abs/2001.08361, 2020. URL https://arxiv.org/abs/2001.08361.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25, 2012.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1–9, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. Advances in neural information processing systems, 33:1877–1901, 2020.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. arXiv preprint arXiv:2204.02311, 2022.

Xi Chen, Xiao Wang, Soravit Changpinyo, AJ Piergiovanni, Piotr Padlewski, Daniel Salz, Sebastian Goodman, Adam Grycner, Basil Mustafa, Lucas Beyer, et al. Pali: A jointly-scaled multilingual language-image model. arXiv preprint arXiv:2209.06794, 2022.

Han Cai, Chuang Gan, and Song Han. Once for all: Train one network and specialize it for efficient deployment. CoRR, abs/1908.09791, 2019. URL http://arxiv.org/abs/1908.09791.

Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. Advances in neural information processing systems, 2, 1989.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149, 2015.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635, 2018.

Yaohui Cai, Zhewei Yao, Zhen Dong, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Zeroq: A novel zero shot quantization framework. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 13169–13178, 2020.

Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. In International Conference on Machine Learning, pages 7197–7206. PMLR, 2020.

Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. Neural Computation, 18:1527–1554, 2006.

Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 535–541, 2006.

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. arXiv preprint arXiv:1704.00109, 2017.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. arXiv preprint arXiv:1701.06538, 2017.

Martin Magris and Alexandros Iosifidis. Bayesian learning for neural networks: an algorithmic survey, 2023.

Michael Dusenberry, Ghassen Jerfel, Yeming Wen, Yian Ma, Jasper Snoek, Katherine Heller, Balaji Lakshminarayanan, and Dustin Tran. Efficient and scalable Bayesian neural nets with rank-1 factors. In Hal Daumé III and Aarti Singh, editors, Proceedings of the 37th International Conference on Machine Learning, volume 119 of Proceedings of Machine Learning Research, pages 2782–2792. PMLR, 13–18 Jul 2020.

Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep ensembles: A loss landscape perspective. arxiv 2019. arXiv preprint arXiv:1912.02757, 1912.

David GT Barrett and Benoit Dherin. Implicit gradient regularization. arXiv preprint arXiv:2009.11162, 2020.

Samuel L Smith, Benoit Dherin, David G.T. Barrett, and Soham De. On the origin of implicit regularization in stochastic gradient descent. In ICLR, 2021.

Avrajit Ghosh, He Lyu, Xitong Zhang, and Rongrong Wang. Implicit regularization in heavy-ball momentum accelerated stochastic gradient descent. ICLR, 2023.

Matias Cattaneo, Jason Klusowski, and Boris Shigida. On the implicit bias of adam. arXiv:2309.00079, 2023.

Mihaela Rosca, Yan Wu, Benoit Dherin, and David G.T. Barrett. Discretization drift in two-player games. In ICML, 2021.

Yansong Gao, Pan Zhibong, Xin Zhou, Le Kang, and Pratik Chaudhari. Fast diffusion probabilistic model sapling through the lens of backward error analysis. arXiv:2304.11446, 2023.

Benoit Dherin. Implicit biases in multitask and continual learningfrom a backward error analysis perspective. In NeurIPS, Mathematics of Modern Machine Learning Workshop, 2023.

Luis Barba, Martin Jaggi, and Yatin Dandi. Implicit gradient alignment in distributed and federated learning. In AAAI Conference on Artificial Intelligence, AAAI'22, 2021.

Hanna Mazzawi, Xavi Gonzalvo, Michael Wunder, Sammy Jerome, and Benoit Dherin. Deep fusion: Efficient network training via pre-trained initializations. In ICML, 2024.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. CoRR, abs/1910.10683, 2019. URL http://arxiv.org/abs/1910.10683.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time, 2022.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. arXiv preprint arXiv:2010.01412, 2020.

Niladri S. Chatterji, Behnam Neyshabur, and Hanie Sedghi. The intriguing role of module criticality in the generalization of deep networks. CoRR, abs/1912.00528, 2019. URL http://arxiv.org/abs/1912.00528.

Ron Banner, Yury Nahshan, and Daniel Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. Advances in Neural Information Processing Systems, 32, 2019.

Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Rémi Gribonval, Herve Jegou, and Armand Joulin. Training with quantization noise for extreme model compression. arXiv preprint arXiv:2004.07320, 2020.

Yu Bai, Yu-Xiang Wang, and Edo Liberty. Proxquant: Quantized neural networks via proximal operators. arXiv preprint arXiv:1810.00861, 2018.

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 8612–8620, 2019.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. arXiv preprint arXiv:1412.6115, 2014.

Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jing Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. Binarybert: Pushing the limit of bert quantization. arXiv preprint arXiv:2012.15701, 2020.

Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. In Low-Power Computer Vision, pages 291–326. Chapman and Hall/CRC, 2022.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015.

Hrayr Harutyunyan, Ankit Singh Rawat, Aditya Krishna Menon, Seungyeon Kim, and Sanjiv Kumar. Supervision complexity and its role in knowledge distillation. arXiv preprint arXiv:2301.12245, 2023.

Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E Dahl, and Geoffrey E Hinton. Large scale distributed neural network training through online distillation. arXiv preprint arXiv:1804.03235, 2018.

Leo Breiman. Bagging predictors. Machine learning, 24:123–140, 1996.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. Journal of computer and system sciences, 55(1):119–139, 1997.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. Advances in neural information processing systems, 30, 2017.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1):1929–1958, 2014.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685, 2021.

Sneha Kudugunta, Aditya Kusupati, Tim Dettmers, Kaifeng Chen, Inderjit Dhillon, Yulia Tsvetkov, Hannaneh Hajishirzi, Sham Kakade, Ali Farhadi, Prateek Jain, et al. Matformer: Nested transformer for elastic inference. arXiv preprint arXiv:2310.07707, 2023.

George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions—i. Mathematical Programming, 14(1):265–294, 1978. doi: 10.1007/bf01588971.

Satoru Fujishige. Submodular functions and optimization. Elsevier, 2005.

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

A Proof of Theorem 3.1

Theorem A.1 (Backward Error Analysis). Let L be a sufficiently differentiable function on the parameter space $\theta \in \mathbb{R}^n$. The modified loss when using Majority Kernels is,

$$\tilde{L}_{MK}(\bar{\theta}) = L(\bar{\theta}) + \frac{\ell}{4}\|\nabla L(\bar{\theta}) + \nabla^2 L(\bar{\theta}) \cdot \epsilon\|^2 + \nabla L(\bar{\theta}) \cdot \epsilon.$$

where $\epsilon \in \mathbb{R}^n$ is is the random perturbation of the virtual parameters.

Proof. Our proof follows the main theorem in Barrett and Dherin [2020] (Theorem 3.1). We want a modified equation with correction terms of the form:

$$\tilde{f}(\bar{\theta}_p) = f(\bar{\theta}_p) + \ell C_1(\bar{\theta}_p). \tag{A.3}$$

The Taylor series expansion of the true solution $\theta(t + h)$ is,

$$\theta(t + h) = \theta(t) + \ell f(\theta) + \frac{\ell^2}{2}f'(\theta)f(\theta) + \mathcal{O}(\ell^3). \tag{A.4}$$

Now, replacing Eq. A.3 into $\theta(t + h)$ we get,

$$\theta_m(t + h) = \theta(t) + \ell(f(\bar{\theta}_p) + \ell C_1(\bar{\theta}_p)) + \frac{\ell^2}{2}f'(\bar{\theta}_p)f(\bar{\theta}_p) \tag{A.5}$$

$$= \theta(t) + \ell f(\bar{\theta}_p) + \ell^2\left(C_1(\bar{\theta}_p) + \frac{1}{2}f'(\bar{\theta}_p)f(\bar{\theta}_p)\right).$$

The numerical method with a first order Euler method using $f(\theta)$ for consistency is,

$$\theta_{n+1} = \theta_n + \ell f(\theta_n). \tag{A.6}$$

To get $\theta_m(n\ell) = \theta_n$ for all n, we must have $\theta_m(t + \ell)$ matching the numerical method. Comparing like powers of ℓ in equations A.5 and A.6 yields recurrent relations for the correction functions:

$$f(\theta_n) = f(\bar{\theta}_p), \quad C_1(\bar{\theta}_p) + \frac{1}{2}f'(\bar{\theta}_p)f(\bar{\theta}_p) = 0.$$

So the correction term C_1 becomes: $C_1(\bar{\theta}_p) = -\frac{1}{2}f'(\bar{\theta}_p)f(\bar{\theta}_p)$.

For our algorithm, $f(\bar{\theta}_p) = f(\bar{\theta} + \epsilon)$. Doing the first order Taylor expansion of this term yields,

$$f(\bar{\theta} + \epsilon) = f(\bar{\theta}) + f'(\bar{\theta}) \cdot \epsilon,$$

where since ϵ is a vector, the product is matrix-vector product. The correction terms become:

$$f(\theta_n) = f(\bar{\theta}) + f'(\bar{\theta}) \cdot \epsilon$$

and

$$C_1(\bar{\theta}_p) = -\frac{1}{2}f'(\bar{\theta} + \epsilon)f(\bar{\theta} + \epsilon) = -\frac{1}{2}(f'(\bar{\theta}) + f''(\bar{\theta}) \cdot \epsilon)(f(\bar{\theta}) + f'(\bar{\theta}) \cdot \epsilon) =$$

$$= -\frac{1}{2}\left(f'(\bar{\theta})f(\bar{\theta}) + f''(\bar{\theta})f(\bar{\theta}) \cdot \epsilon + f'(\bar{\theta})^2 \cdot \epsilon\right) + \mathcal{O}(\epsilon)$$

Finally, the modified vector field in Eq. A.3 becomes:

$$\tilde{f}(\bar{\theta}_p) = f(\bar{\theta}) + f'(\bar{\theta}) \cdot \epsilon - \frac{\ell}{2}\left(f'(\bar{\theta})f(\bar{\theta}) + f''(\bar{\theta})f(\bar{\theta}) \cdot \epsilon + f'(\bar{\theta})^2 \cdot \epsilon\right) + \mathcal{O}(\epsilon^2).$$

Substituting the vector field definition, the first part is: $f(\bar{\theta}) + f'(\bar{\theta}) \cdot \epsilon = -\nabla L(\bar{\theta}) - \nabla^2 L(\bar{\theta}) \cdot \epsilon$, and the second part is:

$$f'(\bar{\theta})f(\bar{\theta}) + f''(\bar{\theta})f(\bar{\theta}) \cdot \epsilon + f'(\bar{\theta})^2 \cdot \epsilon = \nabla(\nabla L(\bar{\theta}))\nabla L(\bar{\theta}) + (\nabla^3 L(\bar{\theta}))\nabla L(\bar{\theta}) \cdot \epsilon + (\nabla^2 L(\bar{\theta}))^2 \cdot \epsilon$$

$$= \frac{1}{2}\nabla\|\nabla L(\bar{\theta}) + \nabla^2 L(\bar{\theta}) \cdot \epsilon\|^2 + \mathcal{O}(\epsilon^2)$$

Removing the negligible $\mathcal{O}(\epsilon^2)$ term, the modified loss for learning rate ℓ is

$$\tilde{L}_{MK}(\bar{\theta}) = L(\bar{\theta}) + \frac{\ell}{4}\|\nabla L(\bar{\theta}) + \nabla^2 L(\bar{\theta}) \cdot \epsilon\|^2 + \nabla L(\bar{\theta}) \cdot \epsilon.$$

which concludes the proof. $\square$

Corollary A.2. Our approach implicitly incorporates two new elements, the Hessian-based term $-\nabla^2 L(\bar{\theta}) \cdot \epsilon$, which introduces a second-order characteristic to the optimization process, and a distortion-based gradient term. Unlike traditional second-order methods that utilize an inverted Hessian to determine the direction of steepest descent, the direct application does not aim to pinpoint the exact descent direction; instead, it modulates the gradient update to reflect the underlying curvature of the loss surface. Stochastically adding the Hessian term in the penalizing norm should bias to a solution with not only small gradient norm, but also small Hessian norm. From the point of view of the modified loss, the MK algorithm introduces extra stochastic regularization that will offer a smoother navigation of the optimization landscape and bias toward flatter minima.

B Stochastic sharpness aware minimization

Conventional Sharpness-Aware Minimization (SAM) [Foret et al., 2020] aims to find parameters that not only minimize the training loss, $L(\theta)$, but also maintain a low loss in the vicinity of θ, thereby leading to solutions that generalize better. SAM achieves this by considering both $L(\theta)$ and $L(\theta + \epsilon)$, where ϵ is a perturbation that maximizes the loss within a defined neighborhood of θ.

In our scenario, the perturbation happens naturally via our stochastic approach. The implicit perturbation ϵ reflects the variability during training and is not necessarily the worst-case perturbation. Therefore, we can consider the expected value of the loss due to the perturbation ϵ within a defined neighborhood. The bound on the generalization loss would be more about the expected sharpness rather than the maximum sharpness.

Lemma B.1 (PAC-Bayesian Bound with Stochastic Weights). A network is parameterized by an extended set of weights $\widetilde{w}^{(l)}$ per layer, $l \in [L]$, and the parameters θ the network operates in are the result of the stochastic aggregation defined in Eq. 2.1. Let $L_D(\theta)$ and $L_S(\theta)$ denote the true and empirical loss functions, respectively. Let Q be the distribution of model parameters induced by the stochasticity in the weights, and let $\Omega_u(e)$ be the uniform distribution over the extended weight space. For any data distribution D, number of samples m, training set $S \sim D$, and prior distribution P on parameters $\tilde{\theta}$, posterior distribution Q, for any $0 < \delta$, with probability $1 - \delta$ over the draw of training data, then the expected true loss under Q can be bounded as follows [Chatterji et al., 2019]:

$$E_{\theta \sim Q}[L_D(\theta)] \leq E_{\theta \sim Q}[L_S(\theta)] + \sqrt{\frac{\mathrm{KL}(Q\|P) + \log \frac{m}{\delta}}{2(m - 1)}},$$

where $\mathrm{KL}(Q\|P)$ is the Kullback-Leibler divergence between the distribution Q and a prior distribution P.

The expected empirical loss under Q can be expressed as:

$$E_{\theta \sim Q}[L_S(\theta)] = E_{p \sim \Omega_u(e)}[L_S(\bar{\theta}_p)] \leq L_S(\bar{\theta}) + \Delta$$

where

$$\Delta = \left| E_{p \sim \Omega_u(e)}[L_S(\bar{\theta}_p)] - L_S(\bar{\theta}) \right|,$$

is the stochastic sharpness term that represents the deviation of the expected empirical loss under the random weights from the empirical loss of the mean weights. It captures the sensitivity of the empirical loss to fluctuations in the model parameters.

Finally, the stochastic approach affects the convergence behavior of the algorithm, potentially leading to more stable but slower convergence (see Lemma B.2 which also showcase the importance of the random probabilities).

Lemma B.2 (Reduced Learning Rate with uniform probabilities). Let ℓ be the standard learning rate in a conventional gradient descent algorithm. If $e > 0$ represents the extension factor and $p = \left(\frac{1}{e}, \ldots, \frac{1}{e}\right)$ the effective learning rate w.r.t the virtual layer parameters w is $\frac{\ell}{e}$, that is:

$$w \leftarrow w - \frac{\ell}{e}(\nabla_w L(w)).$$

Proof. Our algorithm optimizes for $\widetilde{w}$ instead of the conventional weight parameters w.

When performing gradient descent with learning rate ℓ parameters are updated as follows,

$$\widetilde{w} \leftarrow \widetilde{w} - \ell \nabla_{\widetilde{w}} L(\widetilde{w}).$$

Using the chain rule,

$$\nabla_{\widetilde{w}} L(\widetilde{w}) = \nabla_w L(w) \cdot \frac{\partial w}{\partial \widetilde{w}},$$

where $\nabla_w L(w)$ represents the gradient of the loss function L with respect to w, and $\frac{\partial w}{\partial \widetilde{w}}$ is the partial derivative of w with respect to $\widetilde{w}$, which in the given scenario is represented as a scaling matrix where

each element is scaled by $\frac{1}{e}$. This scaling matrix is applied element-wise to $\nabla_w L(w)$ to obtain the gradient with respect to $\widetilde{w}$.

Thus, when performing gradient descent with our algorithm with learning rate ℓ and uniform p, the implicit update of the extended kernel $\widetilde{w}$ with respect to the conventional parameters w looks as follows,

$$\widetilde{w} \leftarrow \widetilde{w} - \frac{\ell}{e}(\nabla_w L(w) \otimes \mathbf{1}_e),$$

where $\otimes$ is the Kronecker product and $\mathbf{1}_e$ is the $1 \times e$ vector of ones.

Under the condition where the parameters p follow a uniform distribution as described in Eq. 2.1, the effective forward propagation step is implicitly using,

$$w = \frac{1}{e} \sum_{k \in [e]} \widetilde{w}_k,$$

and since moving every element in a vector by a constant $c \in \mathbb{R}$ moves the average of that vector by c, we get that

$$w \leftarrow w - \frac{\ell}{e}(\nabla_w L(w)).$$

$\square$

C Related Work

Standard approaches for obtaining a model that is amenable to inference time constraints rely on a post-training processing stage via various methods. One class of popular methods concern model compression and quantization. A popular approach to model quantization is to truncate the model weights to limited bits of precision such as 4-bit quantization or 8-bit quantization [Banner et al., 2019]. Typically quantizing the learned model weights leads to a loss in performance and one often needs an additional round of fine-tuning on the quantized weights Fan et al. [2020], Bai et al. [2018], Nagel et al. [2020]. There have also been efforts to perform post training quantization without the need for additional finetuning [Banner et al., 2019, Cai et al., 2020]. Other approaches include hardware aware quantization [Wang et al., 2019], quantization based on k-means [Gong et al., 2014] and approaches exploring extreme one-bit quantization [Bai et al., 2020]. In a similar vein, approaches based on the lottery ticket hypothesis [Frankle and Carbin, 2018] aim to prune connections within a pretrained network which amounts to zeroing out entries of the learned weight matrices. See the survey of Gholami et al. [2022] for an in-depth discussion of quantization.

An alternative to model compression is the idea of knowledge distillation [Buciluǎ et al., 2006, Hinton et al., 2015]. Given a large pretrained teacher network, distillation involves training a smaller student network, typically of the same architecture as the larger one, to mimic the behavior of the larger network. Hence the larger model acts as a source for labeled supervision and it is often the standard practice to train the smaller model over the smoothed labels (the full logit distribution of the larger model). There have also been recent works exploring the idea of online distillation [Harutyunyan et al., 2023] or co-distillation [Anil et al., 2018] where the teacher and the student models are trained simultaneously.

Our proposed majority kernels have similarities to the classical notion of model ensembling. There is a rich body of work on principled techniques such as bagging [Breiman, 1996] and boosting [Freund and Schapire, 1997] for producing an ensemble of smaller base models. In recent years it has been observed that empirically, even a simple averaging of independently trained networks produces strong ensembles [Lakshminarayanan et al., 2017]. There have also been efforts to produce an ensemble of multiple models via a single round of training [Huang et al., 2017]. While an ensemble model leads to performance benefits, applying it in inference constrained settings still requires compression techniques such as knowledge distillation. Our proposed approach can be viewed as a way to avoid that by implicitly performing model ensembling in the parameter space itself. A similar intuition underlies the standard practice of dropout regularization [Srivastava et al., 2014], but dropout does not produce a smaller model at the end of training. Our approach is complimentary to dropout, and can in fact be used in conjunction with it.

Finally, there have been recent approaches towards maintaining inference efficiency while simultaneously leveraging the capabilities of a larger model during training time. The sparse mixture-of-experts (MoE) architecture [Shazeer et al., 2017] aims to train a large model consisting of small experts and each example is routed to only a few of the experts. Another approach involves adapting a large pretrained network for many downstream tasks via adding low rank updates to the weight matrices [Hu et al., 2021]. Finally, the recent work of Kudugunta et al. [2023] aims to produce multiple models of various sizes as a result of a single training run. This is achieved by training over a loss averaged over the loss of the constituent models.

D Adversarial Probabilities

In this appendix, we will discuss a variation of the Majority Kernels algorithm presented in the paper, where we apply adversarial perturbations to the probabilities. We hope to shed light on some of the design choices, namely, the empirical reason behind sticking with simple stochastic random probability choices, which provide a simple yet effective training method.

The idea behind the adversarial probabilities is to optimize for the following loss in training,

$$\min_{\widetilde{w}} \ \max_{p} L(\langle\widetilde{w}\rangle_p).$$

We do so by making the probabilities learnable, setting them all to $1/e$ where e is the expansion factor. The full algorithm can be found below (Algorithm D.1),

---

Algorithm D.1 Adversarial Only Majority kernels.

    For every layer l: Initialize $\widetilde{w} \in \mathbb{R}^{n\times m\times e}$, $b \in \mathbb{R}^m$
    initialize: $\ell \leftarrow$ learning rate, $\epsilon \leftarrow$ small positive
    while $s <$ max_steps **do**
        $B \leftarrow$ NewBatch()
        $p \leftarrow 1/e^{m\times e}$
        gradient$_p \leftarrow$ ComputeGradient($\langle\widetilde{w}\rangle_p$,$b$,$B$)
        $p \leftarrow p + \epsilon$ gradient$_p$
        $p \leftarrow$ Normalize(p)
        gradient$_{\widetilde{w},b} \leftarrow$ ComputeGradient($\langle\widetilde{w}\rangle_p$,$b$,$B$)
        $\widetilde{w}, b \leftarrow \widetilde{w}, b - \ell$ gradient$_{\widetilde{w},b}$
        $s+ = 1$
    end while
    return $\langle\widetilde{w}\rangle$, b {Return trained parameters for inference}

Empirical results showed that while performing great at the beginning of training, this algorithm led to over-fitting later on. This was the case also when trying to learn the probabilities as part of the model, i.e., train with the following loss,

$$\min_{\widetilde{w},p} L(\langle\widetilde{w}\rangle_p).$$

We believe that the over-fitting happens from the kernels becoming equal, which leads to equal adversarial probabilities (uniform), which is equivalent to training with low learning rate (See Lemma B.2). To address this, we introduced a random element to the p to prevent the kernels from becoming equal by adding randomness if the adversarial probabilities are equal. This led to the algorithm reported in Table 4.3. The algorithm explain below.

Figure D.4 shows the difference between the learning curves with and without randomness (i.e., Algorithm D.1 vs Algorithm D.2). It is easy to see that the randomness prevents the over-fitting and may lead a slightly better algorithm than the Majority Kernels; However, we should mention that this algorithm has high overhead compared to regular training, as it is calculating the gradient twice, and thus, this algorithm is not intended as a primary contribution.

E Glue Experiment Breakdown

In this appendix, we provide details of the Glue experiment, starting with Table 3, which breaks down the performance of various tasks against vanilla training.

In addition, Figures E.5, E.6, E.7, E.8 and E.9 show eval curves on various tasks revealing that our algorithm achieves peak faster than vanilla training. In addition, our algorithm is less prune to overfitting when over trained.

Finally, in Wortsman et al. [2022], the authors discover that no alignment is needed between kernels when finetuning multiple times provided that we finetune on the same pretrained model. The authors evaluate performance on T5, with size configuration that matches ours but with higher expansion

---

Algorithm D.2 Adv-Majority kernels.

For every layer l: Initialize $\widetilde{w} \in \mathbb{R}^{n \times m \times e}$, $b \in \mathbb{R}^m$
initialize: $\ell \leftarrow$ learning rate, $\epsilon \leftarrow$ small positive
while $s < \max\_steps$ **do**
   $B \leftarrow$ NewBatch()
   $p \leftarrow 1/e^{m \times e}$
   $\text{gradient}_p \leftarrow$ ComputeGradient($\langle \widetilde{w} \rangle_p$,$b$,$B$)
   $p \leftarrow p + \epsilon \, \text{gradient}_p$
   $p \leftarrow$ Normalize(p)
   $u \leftarrow$ KL_Divergence($p, 1/e^{m \times e}$) / $\log e$
   $random_p \leftarrow$ NormalizedExponentialRnd()
   $p \leftarrow u \cdot p + (1 - u) \cdot random_p$
   $\text{gradient}_{\widetilde{w},b} \leftarrow$ ComputeGradient($\langle \widetilde{w} \rangle_p$,$b$,$B$)
   $\widetilde{w}, b \leftarrow \widetilde{w}, b - \ell \, \text{gradient}_{\widetilde{w},b}$
   $s + = 1$
end while
return $\langle \widetilde{w} \rangle$, b {Return trained parameters for inference}
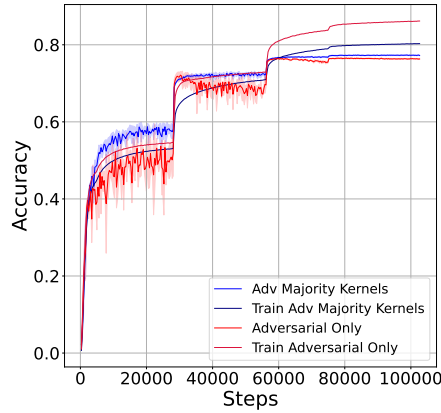

Figure D.4: Train and eval curves for algorithms D.1 and D.2

scale (more kernels to average). Table 4 below shows performance comparison on GLUE tasks in which the authors publish performance.

Our algorithm have greater boost on most tasks with only averaging three kernels. While the algorithm mentioned above share similarity with ours in that it averages multiple kernels to create one inference one, one substantial difference is that our algorithm does not require the compute of finetuning multiple times, nor the additional engineering complexity of finding the right step where performance peaks for early stopping. We require one run with a slight overhead per step, and our evaluation is continuous.

Model	Glue avg	COLA Matthew's	SST acc	MRPC f1	MRPC acc	STS-b pearson	STS-b spearman	qqp acc	qqp f1	MNLI-m	MNLI-mm	QNLI	RTE
BASELINE	80.3±0.1	36.88±1	92.43±0.2	90.85±0.2	87.58±0.3	88.17±0.3	88.03±0.3	88.02±0.1	91.16±0.1	83.96±0.1	83.34±0.1	**90.13**±0.2	72.44±0.9
MAJORITY KERNELS	**80.9**±0.3	**39.69**±1.7	**92.77**±0.2	**91.39**±0.2	**88.23**±0.2	**88.73**±0.4	**88.71**±0.1	88.03±0.1	91.17±0.1	83.9±0.2	**83.66**±0.3	89.75±0.5	73.04±0.9

Table 3: Performance of the various models on downstream tasks (Glue tasks).
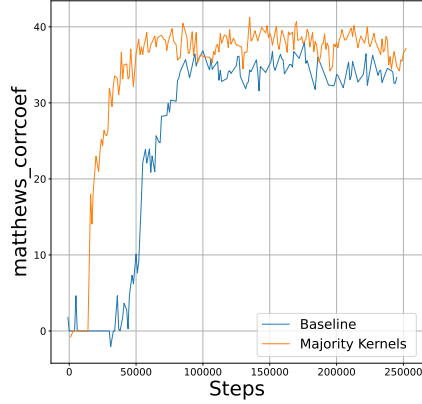
16

Figure E.5: Eval curves for our algorithm compared to vanilla training on Glue Cola
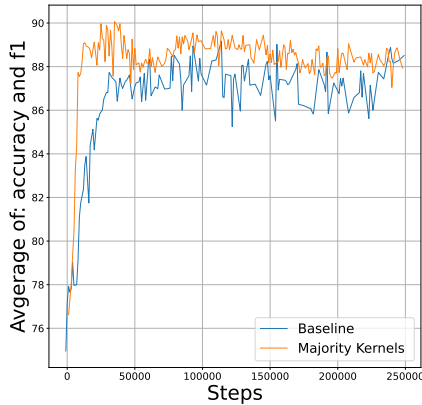

Figure E.6: Eval curves for our algorithm compared to vanilla training on Glue MRPC

F The Subset Selection Baseline

We can view the shrinking of model dimension overparameterization as a combinatorial subset selection algorithm, and one of the predominantly used subset selection method is based on submodular maximization [Nemhauser et al., 1978, Fujishige, 2005]. To describe our method we first focus on a 1-layer network, i.e., $f(x) = v^T \sigma(W \cdot x)$ where $x \in \mathbb{R}^d, W \in \mathbb{R}^{m \times d}$ and $v \in \mathbb{R}^m$. For a given overparameterization factor $e > 1$, we initialize the network with parameters $\{v_0, U_0\}$ where $v_0 \in \mathbb{R}^m$ and $U_0 \in \mathbb{R}^{e \cdot m \times d}$. At time t, before each step of gradient update, i.e., a forward and backward pass, we first invoke a combinatorial subset selection procedure to select the best m rows (neurons) out of the $e \cdot m$ rows in U_t. At the end of training we again invoke the subset selection procedure to select the best m rows to output the final network. The above approach can be easily extended to deeper networks by independently invoking the subset selection procedure for each hidden layer in the network.

We next describe the subset selection procedure. The core idea stems from the fact that we should aim to select neurons that have the most utility, i.e., achieve low loss overall and at the same time aim to avoid selecting redundant neurons, i.e., keep the selected network small. Hence we need to balance notions of utility and diversity, a setting tailor made for submodular optimization. Given $U \in \mathbb{R}^{e \cdot m \times d}$, for each row $i \in [e \cdot m]$ let $u(i)$ denote it's perceived utility. Furthermore let $s(i, j)$ denote the cosine similarity between rows i and j. Then for a given subset S of the rows we consider the following pairwise submodular objective that evaluates the effectiveness of S

$$f(S) = \alpha \sum_{i \in S} u(i) - \beta \sum_{(i,j) \in S} s(i, j) \tag{F.7}$$
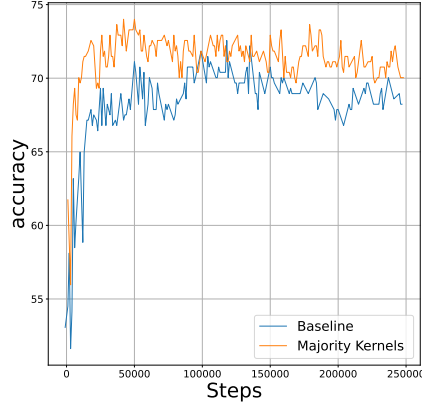
17

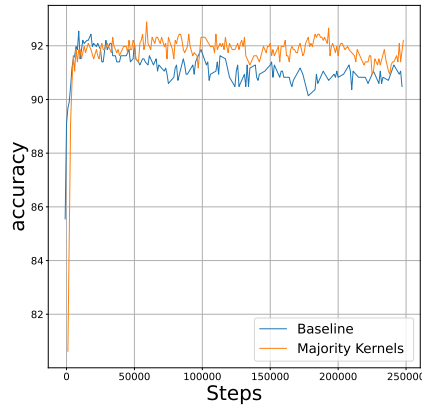*Figure E.7: Eval curves for our algorithm compared to vanilla training on Glue RTE*

Figure E.8: Eval curves for our algorithm compared to vanilla training on Glue SST2

where α, β are hyperparameters. By appropriate choices of the parameters and the similarity functions, it can be shown that the above objective is both submodular and monotonically non-decreasing. Note that evaluating $f(S)$ involves $O(|S|^2)$ computation due to the presence of pairwise terms. This can be computationally prohibitive for layers that have thousands of neurons. Hence as a practical approximation we first consider a t-nn graph $G = (V, E)$ over the $k \cdot m$ rows where each row is only connected to its t nearest neighbors. In our experiments we pick t to be a small value ($t = 10$). Furthermore, we use the norm of row i as a proxy for the utility of neuron i. Let c be the constant term defined as $c = \max_\ell \sum_{j:(\ell,j)\in E} s(\ell, j)$. Then we define the utility as $u(i) = \|U_i\| + c$. Hence our final objective is as follows

$$S^* = \operatorname{argmin}_{S:|S|=m} f(S) \tag{F.8}$$

$$f(S) = \alpha \sum_{i \in S} u(i) - \beta \sum_{\substack{(i,j)\in S \\ (i,j)\in E}} s(i,j). \tag{F.9}$$

It is easy to see that the above is a monotone submodular objective for which a simple greedy algorithm achieves a $1 - \frac{1}{e}$-approximation [Nemhauser et al., 1978]. The full training procedure based on the above approach is described in Algorithm F.1.
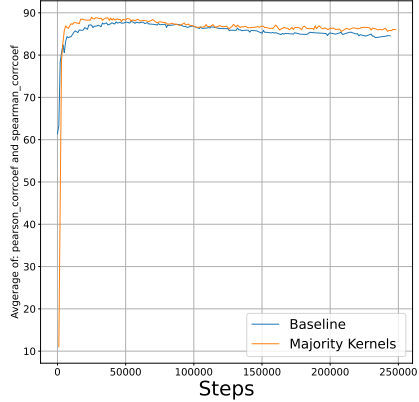
18

Figure E.9: Eval curves for our algorithm compared to vanilla training on Glue STSb

Algorithm	MRPC	RTE	CoLA	SST-2
BASELINE	89.216	72.443	36.883	92.433
MAJORITY KERNELS	**89.816**	**73.043**	39.696	**92.776**
MODEL SOUPS - UNIFORM	82.7	61.7	10.4	91.1
MODEL SOUPS - GREEDY	89.7	70	**43**	91.7

Table 4: Results on GLUE language tasks for our algorithm compared Model Soups (Table J.1 in Wortsman et al. [2022]).

G   Experiments on Fully Connected Networks

We consider training of vanilla feedforward networks on the CIFAR-10 dataset [Krizhevsky et al., 2009]. In this setting the overparameterization is in terms of the expanded width of each hidden layer. In our experiments the expansion factor for overparameterization is set to $e = 3$. We will compare the following algorithms,

- **BASELINE**. A standard feedforward network training.

- **MAJORITY KERNELS**. Training of an overparameterized network via MK with $e = 3$.

- **ENSEMBLE-BASELINE**. This baseline assesses true ensemble performance, setting the achievable performance ceiling. We train and ensemble three ($e = 3$) independent models.

- **DISTILLED-BASELINE**. Assesses standard knowledge distillation to compress the model produced by the ensemble baseline to the original model architecture.

- **SUBSET-BASELINE**. A baseline based on discrete optimization. The method treats model dimension reduction as a subset selection problem, commonly addressed through submodular maximization [Nemhauser et al., 1978, Fujishige, 2005]. This baseline is described in detail in Appendix F. Note that due to the invocation of a combinatorial procedure for each layer

---

Algorithm F.1 The subset selection algorithm for training.

For every layer r: Initialize $W_r \in \mathbb{R}^{n_r \times m_r \times e}$, $b_r \in \mathbb{R}^{m_r \times e}$.
while $step < \max\_steps$ **do**
  $B \leftarrow$ New Batch
  $\forall r$, compute the subset S_r of m_r rows via the greedy algorithm for the objective in Equation (F.8).

  Train one step with parameters $W_r[S_r]$, $b_r[S_r]$ and batch B.
  $\forall r$: update W_r, b_r
  $step = step + 1$
end while
return W^m by again invoking the greedy algorithm for submodular optimization for each layer.

19

this baseline is computationally much more expensive and is impractical beyond simple architectures.

For each algorithm, we hypertune the learning rate by training with learning rates in 0.001×1.5^i for $i \in [-4, 5]$ when the optimizer is the Adam optimizer. Similarly, when the optimizer is SGD we consider the range of learning rates to be in 0.025×1.5^i for $i \in [-4, 5]$. In each case we pick the best performing learning rate on a separate validation set. Finally, we report the test accuracy for the chosen learning rate.

We run the experiment on various architecture topologies:

- $\mathcal{A}_1$ One hidden layer with 100 neurons.
- $\mathcal{A}_2$ Two hidden layers with $\{200, 100\}$ neurons.
- $\mathcal{A}_3$ Three hidden layers with $\{400, 200, 100\}$ neurons.

Results: The results are presented in Table 5. We see that across the three topologies, MK consistently outperforms the baseline and distilled baselines. Furthermore, it achieves performance comparable or better than the much more expensive subset selection based combinatorial approach. Our algorithm increases the training time on CPU by 16.67% which is a negligible increase compared to the distilled baseline and the subset one.

Model	Architecture	Optimizer	Test accuracy	Optimizer	Test accuracy
$\mathcal{A}_1$	[100]	Adam	51.79 ± 0.21	SGD	51.24 ± 0.16
ensemble-$\mathcal{A}_1$	[100]	Adam	52.27 ± 0.12	SGD	52.55 ± 0.52
distilled-$\mathcal{A}_1$	[100]	Adam	48.68 ± 0.86	SGD	48.69 ± 0.33
Subset-$\mathcal{A}_1$	[100]	Adam	52.09 ± 0.16	SGD	52.10 ± 0.20
Majority-$\mathcal{A}_1$	[100]	Adam	52.01 ± 0.10	SGD	51.82 ± 0.15
$\mathcal{A}_2$	[200, 100]	Adam	52.99 ± 0.09	SGD	52.64 ± 0.34
ensemble-$\mathcal{A}_2$	[200, 100]	Adam	54.36 ± 0.08	SGD	54.87 ± 0.15
distilled-$\mathcal{A}_2$	[200, 100]	Adam	52.94 ± 0.26	SGD	51.87 ± 0.89
Subset-$\mathcal{A}_2$	[200, 100]	Adam	53.69 ± 0.20	SGD	53.90 ± 0.15
Majority-$\mathcal{A}_2$	[200, 100]	Adam	54.29 ± 0.20	SGD	54.21 ± 0.24
$\mathcal{A}_3$	[400, 200, 100]	Adam	53.79 ± 0.18	SGD	53.30 ± 0.17
ensemble-$\mathcal{A}_3$	[400, 200, 100]	Adam	55.62 ± 0.15	SGD	56.56 ± 0.27
distilled-$\mathcal{A}_3$	[400, 200, 100]	Adam	55.26 ± 0.32	SGD	54.21 ± 0.31
Subset-$\mathcal{A}_3$	[400, 200, 100]	Adam	54.58 ± 0.15	SGD	54.57 ± 0.34
Majority-$\mathcal{A}_3$	[400, 200, 100]	Adam	55.04 ± 0.12	SGD	54.97 ± 0.31

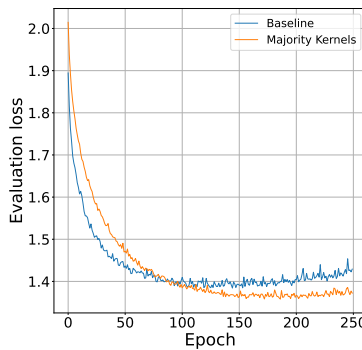*Table 5: Results of the various algorithms trained on CIFAR-10 with batch size 256.*

Figure G.10: Test loss curves for training CIFAR-10 on Majority Kernels vs Baseline on the $\mathcal{A}_1$ architecture.

In the Figure G.10 below, we present the loss curves throughout training of the vanilla training vs the majority kernels on the $\mathcal{A}_1$ architecture. Notice that with the added regularization, MK often needs more steps to converge.