An Overview of Double Descent and Overparameterization

Erdong Hu [*] [1]

Abstract

The double descent curve is a recently identified phenomenon that extends the classical bias-variance curve to a regime of purely monotonically decreasing loss, consistent with more recent empirical results in deep neural networks. This paper provides an overview of some of the notable developments towards understanding double descent as well as establish its effect on learning and optimizing machine learning models. We also include a discussion on the bias-variance trade-off and try to understand both its shortcomings and the absence of double descent until recently. We emphasize the critical and overparameterized regimes of the double descent curve as characterizing them helps inform the optimization and analysis of modern neural network models. Additionally, we outline some interesting future directions regarding double descent and the overall field of overparameterized machine learning.

1. Introduction

Deep neural networks (DNN) have been at the frontiers of latest machine learning developments as methods of deep learning have been employed to achieve high accuracy results in a wide range of scientific and engineering tasks. In the classical bias-variance trade off, the expectation is that larger models perform poorly due to an assumption that overparameterization inherently leads to overfitting. However, contrary to this conventional notion, these DNNs have only grown larger and larger in terms of parameter size relative to number of data samples yet still these models achieve both high training and testing accuracy. This behavior has spawned a modern notion of "larger is better" that guides the practice of deep learning.

The double descent curve, initially proposed in 2018 (Belkin et al., 2018), serves as a bridge between the classic bias-variance trade off and more recent experimental results. However, the introduced interpolation peak or critical regime that occurs between the two settings entails some non-intuitive behaviors that can impede or harm training in practice. One such example is the non-monotonicity of error with samples where at the critical regime test accuracy becomes worse with more samples (Nakkiran et al., 2019).

Additionally, because of its particular relevance to neural networks, we take special interest in the overparameterized setting and examine some of the recent works regarding it. In agreement with the more recent observations, there is a notion that the overaparameterization is harmless, namely that the overparameterized regime achieves the same test error as the underparameterized regime which would suggest overparameterization does not degrade performance. However, these analyses primarily consider generalization but not various other complementary measures such as memorization or adversarial risk where larger models do not necessarily perform better. Hence, an aim here is to assess whether overparameterization is indeed harmless.

1.1. Contents of this paper

We begin by formalizing the bias-variance decomposition and provide an analysis of the classical setting in Section 2. In Section 3, we define the double descent curve, provide a theoretical analysis based on a simple linear regression problem, and some results on the behavior near the critical regime. Section 4 discusses two methods of mitigating double descent to improve training performance, motivated by the non-monotonic behavior at the critical regime. In Section 5, we look at recent works on the overparameterized setting such as on generalization, memorization, and adversarial inputs to evaluate the notion that overparameterized models are harmless. We discuss potential future directions with regard to double descent and overparameterized models in Section 6.

2. Background

A very general description of machine learning problems is the goal is to learn a model that generalizes beyond the training data. This aim is known as *generalization*. To define generalization concretely we first establish the notation for the problem setting.

[*]Equal contribution [1]Department of Computer Science, Rice University, Houston, Texas. Correspondence to: Erdong Hu <eh51@rice.edu>.

Project report for COMP414/514, Rice CS. Copyright 2021 by the author(s).

We are given a set of n data points x_i and corresponding labels y_i $X = \{(x_i, y_i)\}_{i=1}^{n}$. Often X is partitioned between the training set and testing set, denoted as X_{train} and X_{test}. Additionally, some training algorithms further partition with a validation set, however it is used a supplement to the training set, namely as a method of regularization such as for informing early stopping algorithms.

Given a loss function $L(\theta(x_i), y_i)$ where θ is a learned model, the *empirical risk* (ER) is defined as

$$\mathbb{E}[R(\theta)] := \frac{1}{n}\sum_{i=1}^{n} L(\theta(x_i), y_i) \tag{1}$$

(1) is agnostic to whether the training or testing dataset is used so we define similarly R_{train} and R_{test} to represent the corresponding ER for each dataset. Typically the learning target is the following *empirical risk minimization* (ERM)

$$\theta = arg\min_{\theta} \mathbb{E}[R_{train}(\theta)]$$

A special case of the above is when the model is trained such that $R_{train}(\theta) \approx 0$, which is called *interpolation*. As we will discuss later interpolation, when empirical training risk at or near 0, is associated with overparameterization. Since the aim is to get the model to perform well on the test dataset as well, we therefore define *generalization* as $\Gamma(\theta) := \mathbb{E}[R_{test}(\theta)]$.

2.1. Bias-Variance Trade-off

The empirical risk can be decomposed into three terms - the bias, variance, and an irreducible error. This is a well-known result in classical statistics (Hastie et al., 2001). Let θ^* be the ground-truth,

$$\begin{aligned}\mathbb{E}[R(\theta)] &= (\mathbb{E}[\theta] - \theta^*(x))^2 \\ &\quad + \mathbb{E}[(\mathbb{E}[\theta] - \theta(x))^2] + \sigma^2\end{aligned} \tag{2}$$

where the first, second, and third terms correspond to respectively the bias, variance, and irreducible error. Note that this decomposition is specific to the mean-square error loss function. Other loss functions have similar bias-variance decompositions of their empirical risk (Pfau, 2013; Yang et al., 2020), but we will discuss them in the future works section.

The bias and variance terms play a key role in the double descent curve as the behavior in the different regimes are governed by how these two terms vary with model complexity. However, we must first establish the bias-variance trade off and demonstrate its shortcomings to illustrate why double descent emerged to describe modern machine learning models.

The Bias-Variance Tradeoff was initially proposed in 1992 and here we state the result from that initial paper

Claim 1. *(Geman et al., 1992) As model complexity increases, bias decreases and variance increases monotonically*

The intuitive reasoning behind this claim from this initial paper is as follows: If the data labels are subject to random noise and we apply a model that fits this data perfectly (Interpolating) then the variance term in (2) becomes the variance of the random noise which can be very large. On the other hand a choice of a model that is independent of the data completely eliminates the variance term but induces a large bias. In other words, bias and variance are respectively lack of and excessive dependence on the training dataset. This leads to a U-shaped empirical risk curve relative to model complexity. A corollary of the Bias-Variance Tradeoff is that larger, interpolating models are liable to have high variance and poor generalization, a problem known as *overfitting*.

However, the Bias-Variance Tradeoff proposed in the Geman et al. paper was demonstrated primarily experimentally and the neural networks considered in their experiments had hidden units on the order of roughly 30. In contrast, more modern architectures such as ResNet are magnitudes larger (He et al., 2015). Belkin et al. in their inaugural paper about double descent (2018) suggest the absence of double descent in this classical setting can be attributed to a focus on smaller and fixed set of features. Although not using the exact same datasets, the experiments in the Geman paper and the MNIST experiments in Belkin et al. (2018) demonstrate the contrast where despite similar problem settings the larger models used in the latter can exhibit the double descent curve while the former does not due to not considering complex enough models. We show the plot from Geman et al. in Figure 1 and the plot from Belkin et al. in Figure 2 for this comparison.

3. Double Descent

More recent results in deep learning have achieved good generalization even with large numbers of parameters relative to the dataset size, a characteristic of overparameterization. In particular, Advani and Saxe (2017) contradict conventional notions by showing that characteristics of overparamterization such as interpolation and the related memorization do not inherently lead to poor generalization. In their analysis, Advani and Saxe argue that gradient descent and similar optimization algorithms avoid learning many of the spurious dimensions which have zero gradient. Additionally, their experimental results demonstrate the double descent curve, though not explicitly stated, and note that the worst case error occurs when sample and parameter size are similar, which aligns with the ”critical regime” that we discuss later.

Belkin et al. (2018) describes the double descent curve as

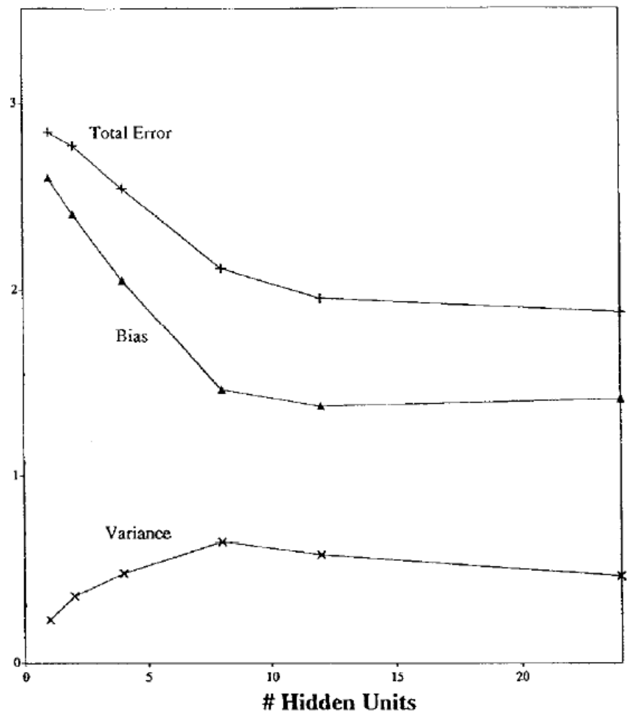


Figure 1. Plot of decreasing bias and increasing variance over number of hidden units in feedforward neural network. Trained on handwritten numeral dataset (Geman et al., 1992)

follows

- An underparameterized "classical" regime where test error aligns with the Bias-Variance Tradeoff and exhibits a U-shaped curve. Bias decreases while variance decreases. The first descent is the region where the bias term dominates where the error is initially decreasing.
- An overparameterized "modern" regime where test error decreases monotonically as model complexity increases. Here both bias and variance decrease. The second descent occurs during this monotonically decreasing error.
- An interpolation peak in between where model capacity and dataset size are similar and where the variance peaks. We refer to this region as the critical regime per Nakkiran et al. (2019) as this region also exhibits interesting behavior.

Figure 2 shows an example of the double descent curve.

3.1. Double Descent on Linear Regression

First, it is important to establish double descent for a simplified problem. Nakkiran (2019) presents an analysis of

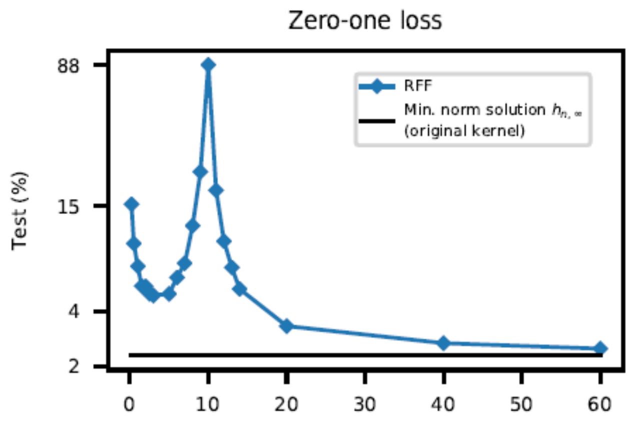


Figure 2. Plot of double descent for Random Fourier Features model with 0-1 loss over the number of random fourier features ($\times 10^3$). Trained on MNIST dataset (Belkin et al., 2018)

double descent on a simplified linear regression problem. In this analysis, the problem setting is as follows:

We are given a ground-truth $(x, y) \in \mathbb{R}^d \times \mathbb{R}$ where $x \sim \mathcal{N}(0, I_d)$ and $y = \langle x, \beta \rangle + \mathcal{N}(0, \sigma^2)$ and $\| \beta \|_2 \leq 1$ where β is the true signal and σ^2 is the variance of the random noise. In other words, x obeys a Gaussian isotropic distribution and y is linear on x with some observation noise. We have n data points $\{(x_i, y_i)\}_{i=1}^n$ from this distribution and want to learn an estimator $\hat{\beta}$ that outputs $f_{\hat{\beta}}(x) = \langle x, \hat{\beta} \rangle$. The data samples are represented as matrix $X \in \mathbb{R}^{n \times d}$ and the labels as $y \in \mathbb{R}^n$. The objective of gradient descent is the following empirical risk minimization:

$$\min_{\hat{\beta}} \| X\hat{\beta} - y \|_2^2$$

Gradient descent converges to a solution $\hat{\beta} = X^{\dagger} y$, the Moore-Penrose pseudoinverse. Nakkiran presents two claims, one of which is from Hastie et al. (2019), about this estimator's bias and variance terms, depending on the an underparamterization factor $\gamma = \frac{n}{d}$. However, for our purposes, we reframe these results in terms of the overparamterization factor $\omega = \frac{1}{\gamma}$.

Claim 2. *Overparamterized regime (Nakkiran, 2019) If $\omega > 1$, then the bias, variance, and excess risk (empirical risk without the irreducible error) are*

$$B = (1 - \frac{1}{\omega})^2 \| \beta \|_2^2$$
$$V \approx \frac{1}{\omega}(1 - \frac{1}{\omega}) \| \beta \|_2^2 + \sigma^2 \frac{1}{\omega - 1}$$
$$\mathbb{E}[R(\hat{\beta})] \approx (1 - \frac{1}{\omega}) \| \beta \|_2^2 + \sigma^2 \frac{1}{\omega - 1}$$

Claim 3. *Underparamterized regime (Hastie et al., 2019) If $\omega < 1$, then the bias and variance are*

$$B = 0$$
$$V \approx \frac{\omega\sigma^2}{1-\omega}$$

In both cases of the variance term, the terms with the noise σ^2, explodes asymptotically near $\omega = 1$, at the critical regime. This aligns with the classical intuition that high variance leads to sensitivity to random noise. For the underparameterized regime, $\omega \in [0, 1)$, the variance term is strictly increasing which again aligns with the classical case where variance is rising up to the critical interpolation peak. In the overparameterized regime $\omega \in (1, \infty)$, as ω increases, the noise term vanishes while the other term decreases to the limit of $\| \beta \|_2^2$ which is the monotonically decreasing behavior expected in double descent. Hence these theoretical results demonstrate each of the three components of the double descent curve in the linear regression setting.

3.2. Behavior near Critical Regime

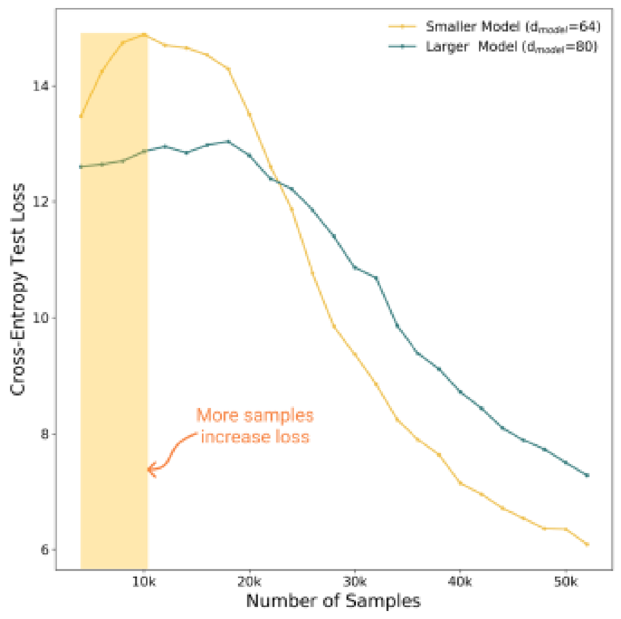


Figure 3. Plot of double descent near the critical regime for transformer models of two different sizes on IWSLT'14 German-English dataset. (Nakkiran et al., 2019)

One particularly non-intuitive behavior with the double descent curve is that error can increase by adding data samples around the critical regime. This result has been demonstrated by theoretically in the simplified linear regression setting (Nakkiran, 2019) and experimentally in CNN and transformer models (Nakkiran et al., 2019). As shown in Figure 3 and discussed in aforementioned works, this behavior is due to a sample-wise double descent behavior where varying the number of samples also exhibits the double descent curve. This behavior comes from the notion of model complexity also depending on the number of data samples in addition to the parameter width. Moreover, this result leads to potential issues in practice where even addition of independent identically distributed data can lead to worse performance as *a priori* knowledge of where in the double descent curve the training has reached is not feasible.

4. Mitigation of Double Descent

Due to the behavior near the critical regime, there is interest in mitigating this part of the double descent curve which results in a monotonically decreasing ”single-descent” curve. Various regularization methods have been found both theoretically and experimentally to achieve this effect. The classical interpretation is that regularization induces a bias in the model by adding information not directly derived from the data and thereby mitigating the variance term. Belkin et al. notes that the historical absence of double descent can be partly attributed to the common use of regularization (2018). We examine two methods: ℓ_2 regularization and optimal early stopping.

4.1. ℓ_2 Regularization

ℓ_2 or ridge regularization is a commonly used method of regularization by adding an ℓ_2 norm penalty term in the loss function. Nakkiran et al. (2021) demonstrate for the ridge regression setting that optimal choice of ℓ_2 coefficient leads to monotonically decreasing empirical risk in both model and sample size. The theoretical problem setting is the same as the one in Section 3.1. However due to the ridge regularization term, the minimization scheme becomes

$$\min_{\hat{\beta}} \| X\hat{\beta} - y \|_2^2 + \lambda \| \beta \|_2^2$$

Theorem 1. *(Nakkiran et al., 2021) For optimal choice of regularization coefficient $\lambda_{d,n}^{opt}$ and corresponding estimator $\hat{\beta}_{d,n}^{opt}$, where d is parameter width and n sample size, the following hold for the expected test risk*

$$\mathbb{E}[R(\hat{\beta}_{d,n+1}^{opt})] \leq \mathbb{E}[R(\hat{\beta}_{d,n}^{opt})]$$
$$\mathbb{E}[R(\hat{\beta}_{d+1,n}^{opt})] \leq \mathbb{E}[R(\hat{\beta}_{d,n}^{opt})]$$

While the above is only proven for the linear isotropic Gaussian case and Nakkiran et al. mention counterexamples to sample monotonicity outside of this case, they have also shown experimentally that this monotonicity with sample and model size due to ridge regularization also occurs in neural networks. As such there is interest in proving a more general result due to these empirical observations.

4.2. Epoch-wise Double Descent and Optimal Early Stopping

In addition to double descent over parameter width and sample size, it has also been observed that a double descent curve exists over training time, known as epoch-wise double descent and one explanation attributes this behavior to model complexity being linked to training time (Nakkiran et al., 2019). However, Heckel and Yilmaz (2021) show instead that this double descent curve is the result of inconsistent learning rates along different features. As shown in Figure 4, the different rates result in a superposition of misaligned bias-variance U-curves that sum to a curve resembling the double descent curve. The regularization method that comes from this is to optimally tune learning rates so that these bias-variance U-curves are in phase and the resulting single U-curve can be minimized at some optimal stopping time, namely the optimal stopping time for the single epochwise bias-variance curve, and is shown to reduce testing risk through this synchronous minimization for all features. These results were proven analytically on both linear regression and in two-layer neural networks with ReLU activations, as well as demonstrated empirically in Convolutional Neural Networks and ResNet.

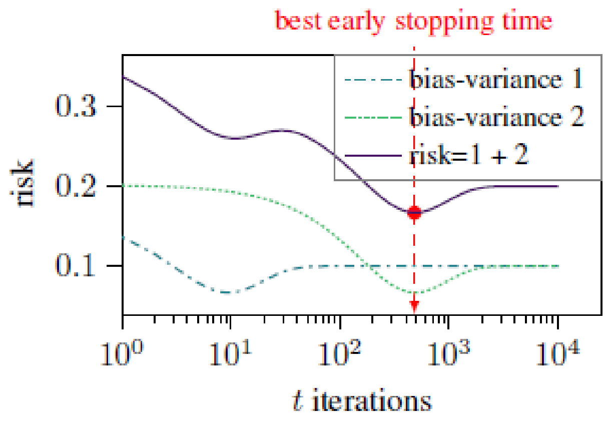


Figure 4. Plot demonstrating epoch-wise double descent and the superposition of bias-variance curves that leads to it. (Heckel & Yilmaz, 2021)

Heckel and Yilmaz (2021) note that the above method is infeasible to execute exactly in practice due to difficulty in determining the optimal learning rates for each feature. Instead they suggest a sub-optimal alternative method of tuning these learning rates as hyperparameters.

5. Overparameterization

As modern deep learning methods typically operate in the overparameterized regime, there is interest in understanding the benefits and risks of interpolating the training data. One characterization is that interpolation is *harmless*, specifically that although interpolation increases the error this interpolation penalty asymptotically becomes 0 as model overparameterization increases (Muthukumar et al., 2019). Additionally, Muthukumar et al. notes that in empirical results of the double descent phenomenon, the overparameterized regime has lower asymptotic error than the minimum of the underparameterized regime.

5.1. Memorization

One mechanism to understand the harmlessness of overparameterization is *memorization* where the model learns but does not generalize some training examples. It has been shown that overparameterized deep neural networks have both the capacity and tendency towards memorizing training data (Zhang et al., 2017; Radhakrishnan et al., 2018). In the bias-variance tradeoff perspective, memorization is a sign of high variance and overfitting. However, Feldman (2020) argues the opposite, that memorization can instead be complementary to generalization and improve model performance. Feldman notes that in practice datasets are not distributed evenly and rare examples and labels are inherently difficult to generalize. More precisely, datasets often follow a long-tailed Zipf distribution. In these distributions, Feldman shows that memorization becomes beneficial or even necessary to model performance as a result of these rare labels.

However, some security and privacy issues arise inherently from memorization. Carlini et al. (2020) show language models, primarily GPT-2, memorize training data that includes personally identifiable information and present a practical attack that specifically target these privacy vulnerabilities. In particular, larger models (And therefore overparameterized) memorize more data, and as such these privacy preservation concerns become relevant in operating in the overparameterized regime.

5.2. Adversarial inputs and Adversarial Risk

From the outset, we have emphasized generalization as a key measure of model performance on new data. However, work by Narang et al. (2021) suggests, at least in the classification problem setting, that overparameterized models are vulnerable to small, adversarially perturbed inputs that produce degraded performance. In particular, these adversarial inputs occur in the absence of other factors such as label noise and model misspecification, and where the model has good generalizability and decreasing test risk, but has high adversarial risk (Defined shortly).

In classification problems, a typical loss function is the 0-1 loss which leads to the empirical risk becoming the probability a label is incorrect. The adversarial risk, for a perturbation set $X(x, \epsilon)$ is defined as the following

$$R_{adv}(\theta) := \mathbb{E}[\max_{\tilde{x} \in X(x,\epsilon)} \mathbb{I}[sgn(\theta(\tilde{x})) \neq sgn(\theta^*(x))]]$$

Here, X and ϵ are chosen as $X(x, \epsilon) = \{\tilde{x} : |x - \tilde{x}| \leq \epsilon\}$ and $\epsilon = \frac{2}{n}$ where n is the number of samples. In other words, R_{adv} measures the local worst case loss around some neighborhood of a data point x.

The good generalization but high adversarial risk is attributed to the fragility in the classification 0-1 loss function (Narang et al., 2021) that the mean-square loss used in regression does not exhibit. Specifically, the 0-1 loss is vulnerable when the model makes predictions, prior to taking the sgn, near the decision boundary. While the results by Narang et al. currently only apply to a highly specific problem setting, they lead to a question of whether simple generalizability is an adequate measure of model performance. In addition, the framework of adversarial risk can be a useful tool to analyze local generalizability in learned models as in many problems good local behavior is desirable.

6. Conclusion and Future Directions

We have presented an overview of some recent works on double descent as well as some related topics in overparameterization. Part of our review highlighted the contrast between the classical bias-variance tradeoff perspective and more recent results. We focused primarily on works related to explaining and improving model performance at the critical and overparameterized regimes. The study of overparameterized models has great practical value in modern machine learning as many recent results have tried to characterize their benefits and risks. In particular, the double descent curve has proven to be a useful framework to understand overparameterized models, just as the bias-variance trade off has been useful in informing classical statistics and machine learning.

Here we present a few directions for future inquiry both related to the double descent phenomenon and in the general area of overparameterization.

6.1. Analysis of Double Descent on non-regression settings

Although cross-entropy loss is a commonly used loss function in machine learning problems, there is an absence of theoretical analysis on its bias and variance as typically analyses have focused on the linear regression setting instead (Nakkiran, 2019; Yang et al., 2020; Heckel & Yilmaz, 2021). Experimental results have shown that cross entropy loss also exhibits double descent behavior (Nakkiran et al., 2019; Yang et al., 2020). The generalized bias-variance decomposition (Pfau, 2013) is a potential starting point for this analysis.

6.2. Effect of data augmentation on double descent and memorization

As emphasized in section 4, regularization techniques play an important role in mitigating the undesirable effects near the critical regime as well generally improving model performance. As noted by Zhang et al. (2017), data augmentation is another form of regularization and a potential route is to examine the effects of data augmentation on the double descent behavior and on memorization versus generalization. An interesting question is whether data augmentation tools can improve the generalizability of rarer examples and labels.

6.3. Double Descent and Generalizability on Transfer Learning

Transfer learning is a task in machine learning of using an existing model for one task and applying it to a different but similar task (Dar & Baraniuk, 2021). Current analyses of double descent tend to assume the training and test data come from the same distributions (Nakkiran, 2019; Narang et al., 2021). Transfer learning is interesting because it can introduce some degree of incongruity between training and test data and we are interested in how this can affect generalization. Some work in this area has been done by Dar & Baraniuk (2021) in the case of transfer between two linear regression tasks.

References

Advani, M. S. and Saxe, A. M. High-dimensional dynamics of generalization error in neural networks. *CoRR*, abs/1710.03667, 2017. URL http://arxiv.org/abs/1710.03667.

Belkin, M., Hsu, D., Ma, S., and Mandal, S. Reconciling modern machine learning and the bias-variance trade-off. *CoRR*, abs/1812.11118, 2018. URL http://arxiv.org/abs/1812.11118.

Carlini, N., Tramèr, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., Roberts, A., Brown, T. B., Song, D., Erlingsson, Ú., Oprea, A., and Raffel, C. Extracting training data from large language models. *CoRR*, abs/2012.07805, 2020. URL https://arxiv.org/abs/2012.07805.

Dar, Y. and Baraniuk, R. G. Transfer learning can outperform the true prior in double descent regularization. *CoRR*, abs/2103.05621, 2021. URL https://arxiv.org/abs/2103.05621.

Feldman, V. Does learning require memorization? a short tale about a long tail. In Makarychev, K., Makarychev, Y., Tulsiani, M., Kamath, G., and Chuzhoy, J. (eds.), *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pp. 954–959. ACM, 2020. doi: 10.1145/3357713.3384290. URL https://doi.org/10.1145/3357713.3384290.

Geman, S., Bienenstock, E., and Doursat, R. Neural networks and the bias/variance dilemma. *Neural Comput.*, 4(1):1–58, 1992. doi: 10.1162/neco.1992.4.1.1. URL https://doi.org/10.1162/neco.1992.4.1.1.

Hastie, T., Friedman, J. H., and Tibshirani, R. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer, 2001. ISBN 978-1-4899-0519-2. doi: 10.1007/978-0-387-21606-5. URL https://doi.org/10.1007/978-0-387-21606-5.

Hastie, T., Montanari, A., Rosset, S., and Tibshirani, R. J. Surprises in high-dimensional ridgeless least squares interpolation. *CoRR*, abs/1903.08560, 2019. URL http://arxiv.org/abs/1903.08560.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Heckel, R. and Yilmaz, F. F. Early stopping in deep networks: Double descent and how to eliminate it. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=tlV90jvZbw.

Muthukumar, V., Vodrahalli, K., and Sahai, A. Harmless interpolation of noisy data in regression. *CoRR*, abs/1903.09139, 2019. URL http://arxiv.org/abs/1903.09139.

Nakkiran, P. More data can hurt for linear regression: Sample-wise double descent. *CoRR*, abs/1912.07242, 2019. URL http://arxiv.org/abs/1912.07242.

Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., and Sutskever, I. Deep double descent: Where bigger models and more data hurt. *CoRR*, abs/1912.02292, 2019. URL http://arxiv.org/abs/1912.02292.

Nakkiran, P., Venkat, P., Kakade, S. M., and Ma, T. Optimal regularization can mitigate double descent. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=7R7fAoUygoa.

Narang, A., Muthukumar, V., and Sahai, A. Classification and adversarial examples in an overparameterized linear model: A signal processing perspective. *CoRR*, abs/2109.13215, 2021. URL https://arxiv.org/abs/2109.13215.

Pfau, D. A generalized bias-variance decomposition for bregman divergences. 6 2013.

Radhakrishnan, A., Belkin, M., and Uhler, C. Downsampling leads to image memorization in convolutional autoencoders. *CoRR*, abs/1810.10333, 2018. URL http://arxiv.org/abs/1810.10333.

Yang, Z., Yu, Y., You, C., Steinhardt, J., and Ma, Y. Rethinking bias-variance trade-off for generalization of neural networks. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 10767–10777. PMLR, 2020. URL http://proceedings.mlr.press/v119/yang20j.html.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Sy8gdB9xx.

COMP514 Final Report: Literature Review For Efficient Algorithms in Distributed Optimization

Hao Liang [1] **Xinyang Song** [1] **Jiaqi Li** [1]

Abstract

Distributed stochastic gradient descent (SGD) is essential for scaling the machine learning algorithms to a large number of computing nodes. However, the infrastructures variability such as high communication delay or random node slowdown greatly impedes the performance of distributed SGD algorithm, especially in a wireless system or sensor networks. In this literature review project, we focus on algorithms that aim to address the time cost problem in distributed optimization, and analyze the pros and cons of them. Experiments on synthetic data also show the effectiveness and performance of some distributed optimization methods.

1. Overview

Classical SGD was designed to be run on a single computing node, and its error-convergence has been extensively analyzed and improved in optimization and learning theory(Dekel et al., 2012). Distributed optimization with stochastic gradient descent (SGD) is the backbone of the state-of-the-art supervised learning algorithms, especially when training large neural network models on massive datasets (Liu et al., 2019; Radford et al., 2019). The widely adopted approach now is to let worker nodes compute stochastic gradients in parallel, and average them using a parameter server (Li et al., 2014) or a blocking communication protocol ALLREDUCE (Goyal et al., 2017). Then, the model parameters are updated using the averaged gradient. This classical parallel implementation is referred as *fully synchronous SGD*. However, in a wireless system where the computing nodes typically have low bandwidth and poor connectivity, the high communication delay and unpredictable nodes slowdown may greatly hinder the benefits of parallel computation (Vogels et al., 2019; Dutta et al., 2018; Ferdinand et al., 2019; Amiri & Gündüz, 2019). It is imperative to make distributed SGD to be fast as well as robust to the system variabilities.

[*]Equal contribution [1]Rice University. Correspondence to: hl106, xs22, jl237 <@rice.edu>.

Project report for COMP414/514, Rice CS. Copyright 2021 by the author(s).

A promising approach to reduce the communication overhead in distributed SGD is to reduce the synchronization frequency among worker nodes. Each node maintains a local copy of the model parameters and performs τ local updates (only using local data) before synchronizing with others. Thus, in average, the communication time per iteration is directly reduced by τ times. This method is called Local SGD or periodic averaging SGD in recent literature (Zhou & Cong, 2018; Stich, 2019; Yu et al., 2019) and its variant federated averaging has been shown to work well even when worker nodes have non-IID data partitions(McMahan et al., 2017). However, the significant communication reduction of Local SGD comes with a cost. As observed in experiments (Wang & Joshi, 2019), a larger number of local updates τ requires less communication but typically leads to a higher error at convergence. There is an interesting tradeoff between the error-convergence and communication efficiency.

Instead of simply averaging the local models every τ iterations, Elastic-averaging SGD (EASGD) proposed in (Zhang et al., 2014) adds a proximal term to the objective function in order to allow some slack between the models – which is an idea coming from the Alternating Direction Method of Multipliers (ADMM) (Boyd et al., 2011) Although the efficiency of EASGD and its asynchronous and periodic averaging variants has been empirically validated (Duchi et al., 2011), its convergence analysis under general convex or non-convex objectives is an open problem. The original paper only gives an analysis of vanilla EASGD for quadratic objective functions

A novel algorithm named OverlapLocal-SGD(Wang et al., 2020) further improves the communication efficiency of Local SGD and achieves a better balance in the error-runtime tradeoff. The key idea in Overlap-Local-SGD is introducing an anchor model on each node. After each round of local updates, the anchor model use another thread/process to synchronize. Thus, the communication and computation are decoupled and happen in parallel. The locally trained models achieve consensus via averaging with the synchronized anchor model instead of communicating with others.

Overall, we organize our report as follows: we first introduce the basic ideas of distributed SGD and the reasons for the need of efficient algorithms in distributed SGD. Second, we look into the algorithms that makes up the evolution of such efficient algorithms, which includes PASGD, EASGD and OverlapSGD. Third, we show the experiments that compare those optimization methods, and also experiments designed by ourselves which are conducted on synthetic dataset and ran locally. Last, we summarize the algorithms and analyze the pros and cons of them.

2. Basic Literature review

2.1. Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is an algorithm that performs well in practice, which can accelerate the performance of gradient descent method by computing less per iteration. The iteration can be represented in the form:

$$x_{t+1} = x_t - \eta \nabla f(x_t)$$

where $\eta > 0$ is the step size.

Though SGD has great performance in many scenarios, it may have some problems when dealing with big model and big data, as well as data privacy issue. First of all, for extremely large training datasets, training on a single machine takes a lot of time. Secondly, it also has some disadvantages of data privacy. For example, for a task that to predict which photos are most likely to be viewed or shared multiple times in the future, the potential training data are the photos that a user takes and other user interactions with their photo apps. To train a general model for this task in a data center, all the photos and user interactions should be gathered from client ends to the data center, which can be privacy sensitive. These issues make executing SGD on a single machine not suitable for some kinds of work.

2.2. Distributed SGD

To solve the issues mentioned before, an alternative setting called Federated Learning for solving them has been proposed. The main idea of this model is to decouple the ability to do machine learning from the need to store the data in the cloud. One way to do so is to keep data locally in client ends and update the shared global model. (Konečný et al., 2017) The training process is made up of two parts. One part is training model on client ends and the other part is averaging model in the data center. One classic algorithm for Federated Learning is Fully synchronous SGD. We will discuss the details in the following paragraphs.

Many Federated Learning tasks have particular characters. First and foremost, Federated Learning has distinct privacy advantages compared to optimizing models by gathering training data in the data center. Since the training part is totally executed on client ends and the source of the updates is not needed for the averaging part, users' privacy is greatly protected and not accessed by the data center. Due to the specific process of Federated Learning, training data also have some typical properties. First of all, these training data are non-I.I.D. data since these data for each client end are based on each user's behavior on each device. Besides, due to the same reason, these data are unbalanced, which means the amounts of training data are different among devices (McMahan et al., 2017).

To address these key issues, (McMahan et al., 2017; Konečný et al., 2017) proposed the synchronous update scheme which proceeds in rounds of communication. One typical round of this algorithm consists of 4 steps: (1) Select a subset of existing clients and download the current model to each selected client; (2) Each selected client updated their model based on local data; (3) Send updated models to the central server; (4) The server aggregates these models (a most common way is averaging) to update the global model.

From those we discussed above, the algorithm is split into two parts, the computation part, which executes on client ends (workers), and the communication part, which executes on the central server.

Consider a network of n worker nodes, each of which only has access to its local data distribution D_i, for all $i \in \{1, ..., m\}$. Our goal is to use these n worker nodes to jointly minimize an objective function F(x), defined as follows:

$$F(x) := \frac{1}{m} \sum_{i=1}^{m} \mathbf{E}_{s \sim D_i} \ell(x; s)$$

where $\ell(x; s)$ denotes the loss function for data sample s, and x denotes the parameters in the learning model.

One worker node perform local steps can be denoted as:

$$x_k - \eta g(x_k; \xi_k^{(i)})$$

And the averaging step can be denoted as:

$$x_{k+1} = \frac{1}{m} \Sigma_{i=1}^{m} [x_k - \eta g(x_k; \xi_k^{(i)})]$$

where m is the number of worker nodes, $g(x_k; \xi_k^{(i)})$ represents the stochastic gradient evaluated on $\xi_k^{(i)} \sim D_i$, and η is the learning rate.

The wall-clock time schematic diagram and the vector illustration are separately shown as Figure 1 and Figure 2, where the blue arrows in the diagrams represent gradient computation steps and the red blocks (arrows) represent communication steps.

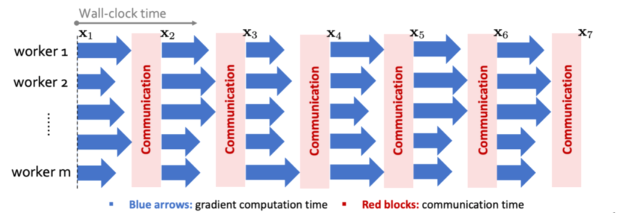

Wall-clock time
worker 1
worker 2
worker m
Communication
Blue arrows: gradient computation time
Red blocks: communication time

Figure 1. Wall-clock time of fully synchronous SGD

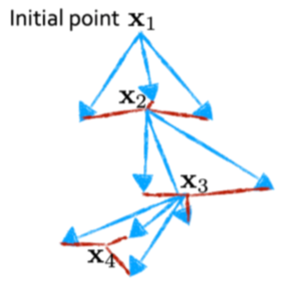


Figure 2. Vector illustration of fully synchronous SGD

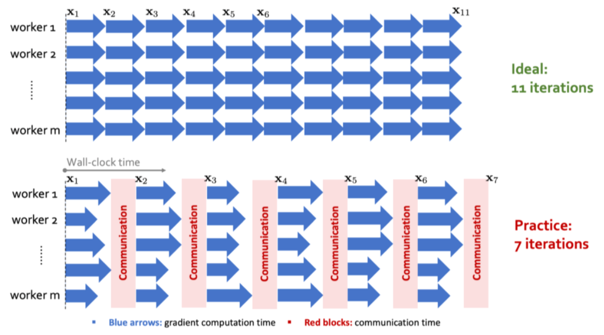


Figure 3. Comparison of Wall-clock time

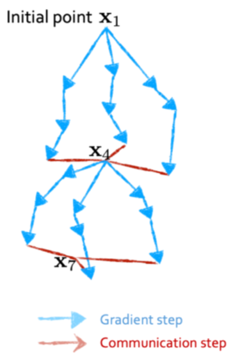


Figure 4. Vector illustration of local SGD

2.3. Existing Problems

Though the classic fully synchronous SGD algorithm has the advantages of protecting data privacy and has good performance on non-IID datasets, it still has some problems. In general, the communication efficiency of fully synchronous SGD algorithm doesn't perform well.

As we mentioned before, since data for each client end is based on each user's behavior on each device, the amounts of training data are unbalanced. Therefore, the gradient computation times for each device in each round are different. Besides, due to the character of Federated Learning that the source of the updates is not accessed for the averaging part, the communication step should execute after all gradient computation on client ends finishes. Therefore, it causes a waste of time. Moreover, the asymmetric property of Internet connection speeds between uplink and downlink also limits the communication efficiency.(Konečný et al., 2017)

As shown in Figure 3, 11 iterations without communication steps can be executed in a time period while 7 iterations with communication steps can be executed in practice in the same period.

This problem inspired some improvements to the fully synchronous SGD algorithm. In the next section, we will discuss 3 different improved algorithms on this issue.

3. Methodologies

3.1. Local SGD

As we discussed in the last section, the scheme of fully synchronous SGD often suffers from large network delays and bandwidth limits. Therefore, to improve the efficiency of communication, a new algorithm called Local SGD(Konečný et al., 2017) came out. Local SGD runs SGD on different workers (client ends) and averages the model on each worker only once in a while. This means that it first runs tau iterations on each worker, and then averages the updated model on each worker to obtain an updated global model. The vector illustration of this algorithm is shown in Figure 4. This algorithm increases communication efficiency intuitively by reducing the times of communication.

Still considering the network we mentioned in 2.2, in Local SGD, each node performs mini-batch SGD updates in parallel and periodically synchronizes model parameters. For the model at i-th worker $x^{(i)}$, we have that for $x_{k+1}^{(i)}$:

$$\begin{cases} \frac{1}{m}\sum_{j=1}^{m}[x_k^{(j)} - \gamma_j g_i(x_k^{(j)};\xi_k^{(j)})] & \text{for} \quad (k+1)\%\tau = 0 \\ x_k^{(i)} - \gamma_i g_i(x_k^{(i)};\xi_k^{(i)}) & \text{for} \quad otherwise \end{cases}$$

where $g_i(x_k^{(i)}; \xi_k^{(i)})$ represents the stochastic gradient evaluated on a random sampled mini-batch $\xi_k^{(i)} \sim D_i$, and γ is the learning rate.

3.2. Elastic SGD

Elastic Averaging SGD method(EASGD) is inspired by quadratic penalty method. The basic idea of EASGD is that each worker maintain its own local parameter while the communication and coordination of work are based on an elastic force which links to the center variable. The center variable itself is calculated as a moving average of the parameters computed by local workers.

3.2.1. EASGD UPDATE RULE

The $EASGD$ update formulas are listed as follows

$$x_{t+1}^i = x_t^i - \eta(g_t^i(x_t^i) + \rho(x_t^i - \tilde{x}_t))$$

$$\tilde{x_{t+1}} = \tilde{x}_t + \eta\Sigma_{i=1}^p \rho(x_t^i - \tilde{x}_t)$$

where $g_t^i(x_t^i)$ denotes the stochastic gradient of F in regard to x^i evaluated at iteration t, x_t^i and $\tilde{x}_t$ denote respectively the value of x^i and $\tilde{x}$ at iteration t, and η is the learning rate. The center variable $\tilde{x}$ is computed as the moving average taken over time. Let $\alpha = \eta\rho$ and $\beta = p\alpha$, then the above equation become

$$x_{t+1}^i = x_t^i - \eta g_t^i(x_t^i) - \alpha(x_t^i - \tilde{x}_t))$$

$$\tilde{x_{t+1}} = (1-\beta)\tilde{x}_t + \beta(\frac{1}{p}\Sigma_{i=1}^p x_t^i)$$

Let $\beta = p\alpha$ leads to an elastic symmetry in the update rule. There also exists an symmetric force equal to $\alpha(x_t^i - \tilde{x}_t)$ between the update of each x^i and $\tilde{x}$. Let $\alpha = \eta\rho$, where the magnitude of ρ represents the extent of exploration in the model. Small ρ means more exploration is allowed in the model because x^i could fluctuate more from the center. The main idea of $EASGD$ is to allow the local workers to perform more exploration(small ρ) and the master to perform exploitation.

3.2.2. ASYNCHRONOUS EASGD

Last section we talked about the update rule of synchronous $EASGD$. In this section, we will discuss its asynchronous variant. Same as synchronous version, the local workers are responsible for updating the local variables x_i while the master is updating the center variable $\tilde{x}$. Each worker maintains its own clock t^i, which starts from 0 and is increased by 1 after each stochastic gradient update of x^i. We define τ as the communication period, which is the frequency of communication between local worker and the center. The master will update whenever the local workers finished one communication period. From the Algorithm 1 below we can see that, whenever τ divides the local clock of the i^{th} worker, the worker communicates with the master and gets the current value of center variable. The worker then waits the server to send back the value and computes the elastic difference $\alpha(x - \tilde{x})$.

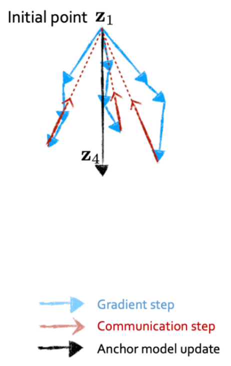


Figure 5. EASGD

Algorithm 1 Asynchronous EASGD:

```
Input: learning rate $\eta$, moving rate $\alpha$, communication period $\tau$
Initialize: $\tilde{x}$ is initialized randomly, $x^i = \tilde{x}$, $t^i = 0$
repeat
  $x \leftarrow x^i$
  if $\tau$ divides $t^i$ then
    a) $x^i \leftarrow x^i - \alpha(x - \tilde{x})$
    b) $x^i \leftarrow x^i - \alpha(x + \tilde{x})$
  end if
  $x^i \leftarrow x^i - \eta g_{t^i}^i(x)$
  $t^i \leftarrow t^i + 1$
until forever
```

The merits of $EASGD$ is that it provides fast convergent minimization and outperforming other baseline approaches in practice. Meanwhile, it reduces the communication overhead between local workers and the server. This algorithm applies to deep learning settings such as parallelized training of convolutional neural networks.

The limitation here is that $EASGD$ only converges in quadratic objective function and the choose of elastic parameter α need to be tested.

3.3. Overlap SGD

Overlap-Local-SGD introduces an anchor model on each node. After each round of local updates, the anchor model is synchronized using another thread/process. Therefore, the communication and computation are decoupled and happen in parallel. The locally trained models achieve consensus via averaging with the synchronized anchor model instead

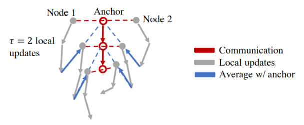


Figure 6. Overlap Local SGD

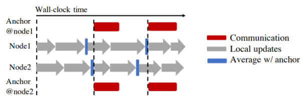


Figure 7. pipeline of Overlap Local SGD

of communicating with others.

Figure 6 and 7 presents a brief illustration of Overlap-Local-SGD. After each τ updates, the local model will be pulled towards the anchor model. We have the following update rule for the local models:

$$x^{(i)}_{k+\frac{1}{2}} = x^{(i)}_k - \gamma g_i(x^{(i)}_k; \xi^{(i)}_k)$$

$$x^{(i)}_{k+1} = \begin{cases} x^{(i)}_{k+\frac{1}{2}} - \alpha(x^{(i)}_{k+\frac{1}{2}} - z_k) & \text{for} \quad (k+1)\%\tau = 0 \\ x^{(i)}_{k+\frac{1}{2}} & \text{for} \quad otherwise \end{cases}$$

where α is a adjustable parameter. A larger α means that the local model is pulled closer to the anchor model z. After pulling back, the nodes will start the next local update, and the anchor threads on each node will concurrently synchronize the current local model and store the average into the anchor model, as shown below:

$$z^{(i)}_{k+1} = \begin{cases} \frac{1}{m}\Sigma^m_{i=1} x^{(i)}_{k+1} & \text{for} \quad (k+1)\%\tau = 0 \\ z_k & \text{for} \quad otherwise \end{cases}$$

From the formula above we can see that the anchor model $z_{\alpha\tau}$ will only be used when updating $x^i_{(\alpha+1)\tau}$. This algorithm could hide the communication latency if the parallel communication time is smaller than τ steps computation time.

Overlap Local SGD further improves the communication efficiency of Local SGD, achieving a better balance in the error-runtime trade-off. The author also provide a convergence analysis to show that this algorithm could converge to a stationary point of non-convex objectives and achieve the same convergence speed as fully synchronous SGD.

4. Experiments & Discussions

4.1. Local experiments on synthetic data

Due to the limitation of computing resources and devices, we implement a system of distributed optimization that can be ran locally on a computing mode. The difference between "local" distributed optimization and classical distributed optimization is that, without ability to run local updates simultaneously, local distributed optimization runs local updates in a prefixed order and then communicate to get global model. Also because of that, it is unreasonable to run the efficient algorithms we mentioned above. Thus in this subsection we design a new group of experiments to explore the non-iid problem in federated learning.

4.1.1. Synthetic Data

To generate synthetic data, we follow a similar setup to that in (Li et al., 2020), additionally imposing heterogeneity among devices. In particular, for each device k, we generate samples (X_k, Y_k) according to the model $y = argmax(softmax(Wx+b)), x \in R^{60}, W \in R^{10\times 60}, b \in R^{10}$. We model $W_k \sim N(u_k, 1), b_k \sim N(u_k, 1), u_k \sim N(0, \alpha); x_k \sim N(v_k, \Sigma)$ where the covariance matrix Σ is diagonal with $\Sigma_{j,j} = j^{-1.2}$. Each element in the mean vector v_k is drawn from $N(B_k, 1)$, $B_k \sim N(0, \beta)$. Therefore, α controls how much local models differ from each other and β controls how much the local data at each device differs from that of other devices. We vary α, β to generate two heterogeneous distributed datasets, denoted *Synthetic*(α, β). We also generate one IID dataset by setting the same W, b on all devices and setting X_k to follow the same distribution. Our goal is to learn a global W and b. We set (α, β) = (0,0) and (2,2) respectively to generate three non-identical distributed datasets. For all synthetic datasets, there are 8 devices in total and the number of samples on each device follows a power law.

4.1.2. Experimental details and results

We design two group of experiments, and here we present the details of the setting and hyperparameters:

- **Group 1:**Local update learning rate 0.01, mini-batch size 10, local updates step 1(which equals to fully synchronous SGD), total communication rounds 200;
- **Group 2:**Local update learning rate 0.01, mini-batch size 24, local updates step 20, total communication rounds 100;

Both experiments utilize an learning rate scheduler which decays learning rate by 10 at half and $\frac{3}{4}$ of total communication rounds, that is, for group 1, learning rate becomes 0.005 at communication round 100 and 0.0025 at communication round 150.

And the result of the first group is shown in figure 8, 9 with the second group corresponds to 10, 11. From both cases, it is easy to draw the conclusion that heterogeneous in the model and data brings larger variance in the training and

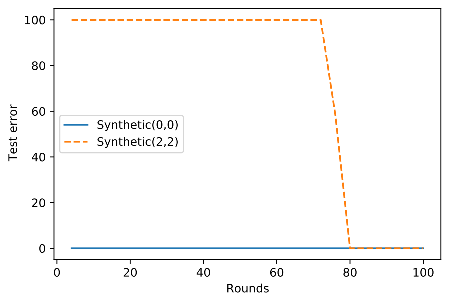


Figure 8. Round 100 test error v.s. rounds

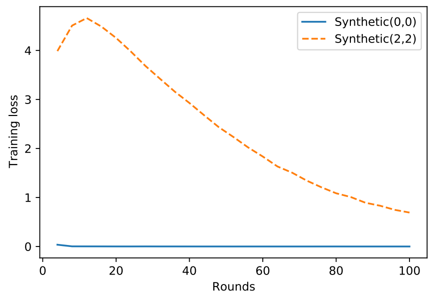


Figure 9. Round 100 training loss v.s. rounds

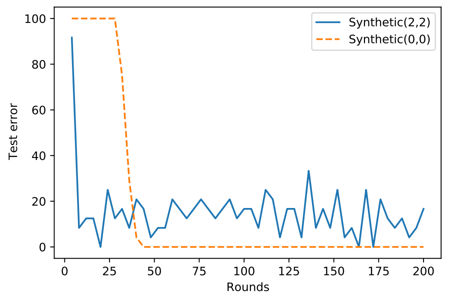


Figure 10. Round 200 test error v.s. rounds

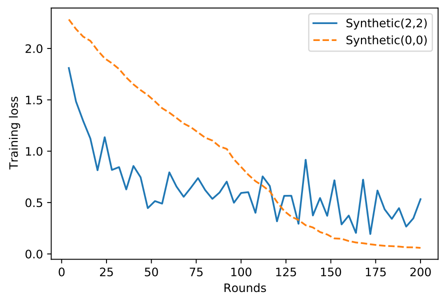


Figure 11. Round 200 training loss v.s. rounds

performance, and sometimes lead to a worse performance.

4.2. Comparison of efficient distributed optimization methods

As for the algorithms discussed in the report, we compare them through a group of experiments done in the original paper.

4.2.1. Experiment settings

In this experiment, there are four algorithm that are taken into consideration. They are: Fully synchronous SGD, LocalSGD(PASGD), PowerSGD(Vogels et al., 2019) and OverlapSGD. It is worth noting that PowerSGD is state-of-the-art efficient distributed optimization method which focuses on boosting the process by compressing the gradients communicated between the server and local nodes. The details of the experiment are: it is performed on CIFAR-10 image classification task (Krizhevsky et al., 2009). We train a ResNet-18 (He et al., 2016) for 300 epochs following the training schedule as the mini-batch size on each node is 128 and the base learning rate is 0.1, decayed by 10 after epoch 150 and 250. The first 5 epochs use the learning rate warmup schedule as described in (Goyal et al., 2017). There are 16 computing nodes in total. The training data is evenly partitioned across all nodes and not shuffled during training. And the results are shown in Fig 12, 13

4.2.2. Analysis of the results

As shown in Figure 9, Overlap-Local-SGD outperforms the rest when comparing the best test accuracy with respect to additional synchronous time per epoch. It also reveals that when the number of local update steps is limited, Overlap-Local-SGD still needs to wait for either the peer or the central server. Also, with proper techniques utilized, both Overlap-Local-SGD and Local SGD can achieve comparable or even better performance comparing to Fully synchronous SGD.

And for the results shown in Figure 10, when the x-axis is changed to Wall-clock time, Overlap-Local-SGD still reaches the minima at the fastest rate. Although other algorithms in the figure are more or less worth than Overlap-Local-SGD, one can expect that as the x-axis extends, it is possible for them to surpass Overlap-Local-SGD.

5. Conclusion

As a conclusion, in this project we explore the efficient algorithms in the distributed optimization. The reason for developing such methods is that, since the original topic focuses on a faster training regarding big model and big data, communication/computation overhead should be minimized. We mainly look into four algorithms, and below we

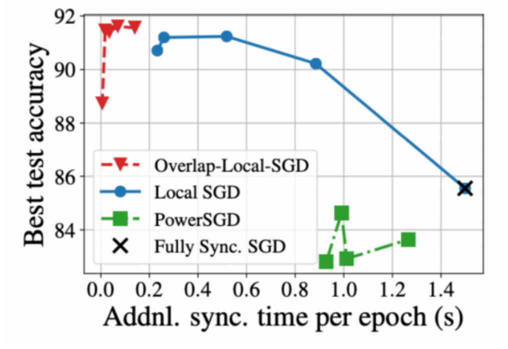


Figure 12. Test accuracy v.s. additional synchronous time per epoch. The fully synchronous methods is noted as the cross in the figure, since its local updates is always1.

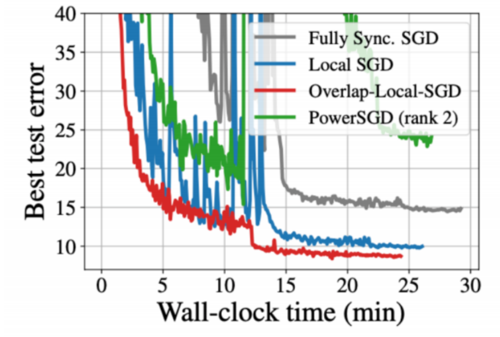


Figure 13. Test Accuracy v.s. wall-clock time.

summarize their pros and cons:

- **Fully Synchronous SGD.** As the most basic method, it brings about the idea of distribute model and/or data to local worker nodes. However, it suffers from communicating with the central server every step, especially when the communication time needed is high;
- **Local SGD(PASGD).** This algorithm comes up with one of the most important ideas in efficient methods, that is workers should allow local updates before communicating with each other. It greatly reduce the time needed on communication, however, convergence is not guaranteed and there is still waiting time for the local nodes;
- **EASGD.** To solve the convergence problem, this algorithm suggests pulling back towards an anchor model before starting the next stage. Also, momentum is finely used in it to achieve competitive performance. Nevertheless, convergence analysis is only provided when the objective function is in quadratic form, and it is hard to decide the elastic parameter α;
- **Overlap-Local-SGD.** In this method, local updates and anchor model are all utilized and it comes up with the idea of acting the communication of computation at the same time. Although it somehow sacrifices the freshness of the global model, it dramatically saves the communication time and thus lead to a more efficient solution overall.

References

Amiri, M. M. and Gündüz, D. Computation scheduling for distributed machine learning with straggling workers. *IEEE Transactions on Signal Processing*, 67(24):6270–6284, 2019.

Boyd, S., Parikh, N., and Chu, E. *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

Dekel, O., Gilad-Bachrach, R., Shamir, O., and Xiao, L. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13(1), 2012.

Duchi, J. C., Agarwal, A., and Wainwright, M. J. Dual averaging for distributed optimization: Convergence analysis and network scaling. *IEEE Transactions on Automatic control*, 57(3):592–606, 2011.

Dutta, S., Joshi, G., Ghosh, S., Dube, P., and Nagpurkar, P. Slow and stale gradients can win the race: Error-runtime trade-offs in distributed SGD. In *International Conference on Artificial Intelligence and Statistics*, pp. 803–812, 2018.

Ferdinand, N., Al-Lawati, H., Draper, S., and Nokelby, M. Anytime minibatch: Exploiting stragglers in online distributed optimization. In *International Conference on Learning Representations*, 2019.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch SGD: Training ImageNet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

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.

Konečný, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., and Bacon, D. Federated learning: Strategies for improving communication efficiency, 2017.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B.-Y. Scaling distributed machine learning with the parameter server. In *OSDI*, volume 14, pp. 583–598, 2014.

Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V. Federated optimization in heterogeneous networks, 2020.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. PMLR, 2017.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. Open AI tech. report, Feb. 2019.

Stich, S. U. Local SGD converges fast and communicates little. In *International Conference on Learning Representations*, 2019.

Vogels, T., Karimireddy, S. P., and Jaggi, M. PowerSGD: Practical low-rank gradient compression for distributed optimization. In *Advances in Neural Information Processing Systems*, 2019.

Wang, J. and Joshi, G. Adaptive communication strategies to achieve the best error-runtime trade-off in local-update sgd. *Proceedings of Machine Learning and Systems*, 1: 212–229, 2019.

Wang, J., Liang, H., and Joshi, G. Overlap local-sgd: An algorithmic approach to hide communication delays in distributed sgd. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8871–8875. IEEE, 2020.

Yu, H., Yang, S., and Zhu, S. Parallel restarted SGD with faster convergence and less communication: Demystifying why model averaging works for deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 5693–5700, 2019.

Zhang, S., Choromanska, A., and LeCun, Y. Deep learning with elastic averaging sgd. *arXiv preprint arXiv:1412.6651*, 2014.

Zhou, F. and Cong, G. On the convergence properties of a k-step averaging stochastic gradient descent algorithm for nonconvex optimization. In *International Joint Conference on Artificial Intelligence*, 2018.

Schedules in deep learning: from learning rates and momentum to quantization

Wei Ren Gan [*1] Ishtdeep Singh [*1] Liubove Orlov Savko [*1]

Abstract

Due to increased advancements in areas of deep learning, the choices in optimization techniques play a key role in training the deep neural network. The process of hyperparameter search and tuning can be prohibitive in time and costs (1). Thus, it is critical to explore and use state-of-art techniques for efficiency and accuracy in both academia and industry. In this literature review, we will dive into the different scheduling in deep learning, from learning rates (2) and momentum (1) to quantization (3).

1. Introduction

Neural networks are getting larger both in size and in number of parameters. Models exist and are being trained with billions of parameters like (4), (5). Large models cost more capital and time for training. There is a need for solutions which can help to decrease both the costs and the time of training these networks. In this paper, we will discuss a variety of techniques which leverage learning rate, momentum and quantization to reduce costs and increase performance.

The learning rate is a hyperparameter that controls how much to change the model's weights in response to the estimated error during the model evaluation. Choosing the learning rate is challenging as a value too small may result in an extremely slow convergence rate, whereas a value too large may result on non convergence. To improve neural network results and optimize the available budget, it is common practice to use learning rate schedules, with large learning rates to get faster to the local/global minima, and lower learning rates to avoid overshooting.

The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction (6). Momentum was designed to speed up learning in directions of low curvature, without becoming unstable in directions of high curvature (1).

The main motivation for quantization is to decrease the memory that is needed to train and run the network without impacting its performance and improving it if possible.

In Section 2, we will discuss REX (2) and compare widely-used learning rate schedules. In Section 3, we will discuss Demon (1) and compare widely-used learning rate and momentum schedules, along with various popular optimization algorithms. In section 4, we will discuss quantization (3), and future work in Section 5.

2. Learning Rate

Many different types of schedules have been studied:

- Constant $\eta_t = \eta_0$
- Step schedule $\eta_t = \gamma_t \cdot \eta_0$, where γ_t is piecewise linear.
- Decay on Plateau (7) is a particular case of step schedule, where the learning rate is decayed when the validation loss does not improve for a preset number of epochs.
- Linear schedule: $\eta_t = (1 - t/T) \cdot \eta_0$
- Cosine schedule (8) $\eta_t = \frac{\eta_0}{2} \cdot \left(1 + \cos(\frac{\pi \cdot t}{T})\right)$
- Exponential schedule (7) $\eta_t = \eta_0 \cdot e^{\gamma t/T}$
- OneCycle schedule (9)

$$\eta_t = \begin{cases} \eta_1 + (\eta_2 - \eta_1)\left(\dfrac{t}{T/2}\right)\eta_0, & t/T < 1/2 \\ \eta_1 + (\eta_2 - \eta_1)\left(2 - \dfrac{t}{T/2}\right)\eta_0, & t/T \geq 1/2 \end{cases} \tag{1}$$

$$\eta_t = \begin{cases} \beta_1 + (\beta_2 - \beta_1)\left(1 - \dfrac{t}{T/2}\right)\beta_0, & t/T < 1/2 \\ \beta_1 + (\beta_2 - \beta_1)\left(-\dfrac{t}{T/2} - 1\right)\beta_0, & t/T \geq 1/2 \end{cases} \tag{2}$$

- REX $\eta_t = \eta_0 \cdot \left(\frac{1-t/T}{1/2+1/2\cdot(1-t/T)}\right)$. This schedule will be explained in more detail below.

[*]Equal contribution [1]Department of Computer Science, Rice University, Houston, USA. Correspondence to: Tasos Kyrillidis <ricecomp414@gmail.com>.

Project report for COMP414/514, Rice CS. Copyright 2021 by the author(s).

All these learning rate schedules can be applied either equally to all the weights in the neural network, using, for example, Stochastic Gradient Descent, or they can be applied **adaptively**, using other types of optimizers such as Adagrad, Adadelta, RMSprop, Adam, etc.

We have not found theoretical proof of which learning rate schedule to use in which model/application. To have an idea, it can be observed empirically by training many models (ResNet, MobileNet, GPT3) etc, with different datasets, and comparing the accuracy results of different schedules. It is observed in REX. To determine which one is the best, one way to do so is to compare empirically, by training several different neural networks on different datasets, as done so in (2). The schedule REX proposed in the aforementioned algorithm is of special interest, as it outperforms in most of the experimental setups.

2.1. REX

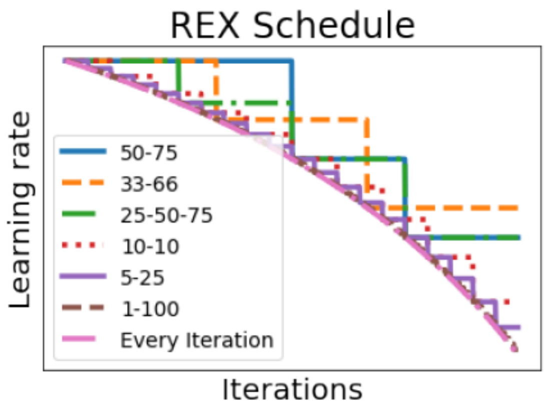
To formalize the process of identifying a good learning rate schedule, the authors decompose the learning rate schedule as a combination of a profile curve and a sampling rate on that curve.

The profile is the function that models the learning rate schedule, while the sampling rate is how frequently the learning rate is updated according to the profile. The sampling rate can be either each iteration, or it can be done only once or twice during the whole training.

The profile chosen for REX is motivated by the empirical observation that the linear schedule sometimes performs better when keeping it constant at the beginning and delaying the linear decay after several iterations. Indeed, observing the update equation

$$\eta_t = \eta_0 \cdot \left(\frac{1 - t/T}{1/2 + 1/2 \cdot (1 - t/T)} \right)$$

we can observe that for small t, the factor is closer to one.



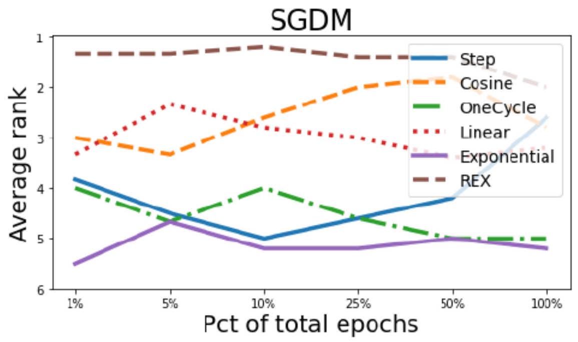
2.2. Experimental setup

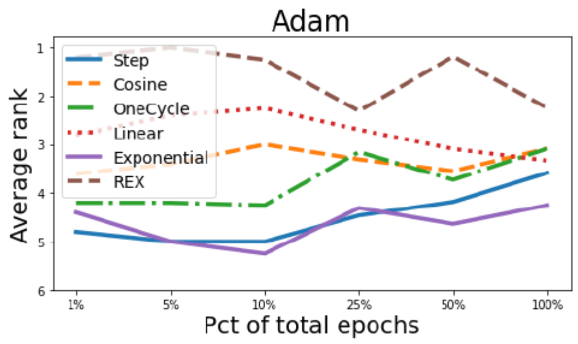
The experiments were performed on the models and dataset listed on the table. The schedules used during training are the ones listed above, excluding the constant learning rate schedule. Also, the optimizers used for training were SGD with momentum and Adam, totaling in 98 models trained.

Experiment short name	Model	Dataset
RN20-CIFAR10	ResNet20	CIFAR10
RN50-IMAGENET	ResNet50	ImageNet
VGG16-CIFAR100	VGG-16	CIFAR100
WRN-STL10	Wide ResNet 16-8	STL10
VAE-MNIST	VAE	MNIST
YOLO-VOC	YOLOv3	Pascal VOC
BERT$_{BASE}$-GLUE	BERT (Pre-trained)	GLUE (9 tasks)

2.3. Results

In all cases, REX schedule had the highest top-1 or top-3 performance for both Adam and SGD with Momentum, and all budgets, which shows a great learning rate schedule to use in applications. In the figure below the average rank across all experiments is shown, where we can see that REX has consistently the best rank.





3. Momentum

3.1. DEMON

Gradient descent based algorithms are popular for deep neural networks training (6). Numerous optimization algorithms exists, however SGDM (6) and Adam (10) remain at the forefront of both research and industry (6) (1). In order to achieve optimal performance and reduce computational costs, hyperparameters tuning of momentum, learning rate (LR), learning rate decay, weight decay, amongst others are some of the approaches (1). In DEMON (1), we will discuss the model performance and hyperparameter robustness of the decaying momentum rule on SGDM and Adam with respect to its vanilla counterparts, alternative optimization algorithms, and SGDM/Adam plus a variety of widely-used schedulers (1).

DEMON is a decaying momentum rule that decays the total contribution of the gradient to all future gradient updates (1). Consider a gradient at a particular timestamp g_t contributing $\eta \sum_i \beta^i$ of its 'energy' to all future gradient updates, where as t tends towards infinity, we get the geometric sum, $\sum_{i=1}^{\infty} \beta^i = \beta \sum_{i=0}^{\infty} \beta^i = \beta/(1-\beta)$ (1). By decaying the cumulative momentum $\beta/(1-\beta)$ linearly to 0, we get DEMON (1). Thus, the authors designed the decaying routine, $\beta_t/(1-\beta_t) = (1-t/T)\beta_{\text{init}}/(1-\beta_{\text{init}})$, with notations of current step t, total step T, and initial β as β_{init} (1). Additionally, the fraction $(1-t/T)$ refers to the proportion of iterations remaining (1). DEMON has shown to stabilize the growth of weights across training by preventing the weights from growing too quickly (1). Subsequently, DEMON speeds up in the early phases of training, but delaying the decay throughout training (1).

3.2. Experiment Setup

Numerous optimization techniques were evaluated across 28 relevant combinations of models-epochs-datasets-optimizers experiments (1). Some of the methods include AdamW (11), which decouples weight in Adam, YellowFin (12), which is an automatic tuner for learning rate and momentum in SGDM motivated by a quadratic model analysis and robustness properties, QHM (13), which decouples the momentum term from the current gradient's contribution when updating the weights, QHAdam (13), which is an adaptive learning rate extension of QHM, AMSGrad (14), which uses the maximum of the exponential moving average of squared gradients, and AggMo (15), which is a variant of momentum that combines multiple velocity vectors with different β parameters (1).

The architectures analyzed include Convolutional Networks (CNN) with Residual architecture (ResNet 20 (16), ResNet 56 (16), Wide ResNet 16-8 (17), Non-Residual architecture (VGG-16) (18), Recurrent Neural Networks (RNN) with LSTM (19), Variational AutoEncoders (VAE) (20), Capsule Network (21), Noise Conditional Score Network (NCSN) (22), and BERT (23); with datasets including MNIST, FMNIST, CIFAR-10, CIFAR-100, STL-10, Penn Treebank (PTB), Tiny ImageNet, GLUE benchmark (24) (1), figure 1.

Experiment short name	Model	Dataset
`RN20-CIFAR10`	ResNet20	CIFAR10
`RN56-TINYIMAGENET`	ResNet56	Tiny ImageNet
`VGG16-CIFAR100`	VGG-16	CIFAR100
`WRN-STL10`	Wide ResNet 16-8	STL10
`LSTM-PTB`	LSTM RNN	Penn TreeBank
`VAE-MNIST`	VAE	MNIST
`NCSN-CIFAR10`	NCSN	CIFAR10
`CAPS-FMNIST`	Capsnet	FMNIST
`BERT`$_{BASE}$`-GLUE`	BERT (Pre-trained)	GLUE (9 tasks)

Figure 1. Summary of experiment settings (1).

The authors compared DEMON with the following popular learning rate and momentum schedules: Step, OneCycle (9), Cosine (8), Linear, Exponential (7), and Decay on Plateau (7). In addition to the hyperparameters specialized to the particular schedule, the authors tuned both the learning rate in multiples of 3, the momentum $\in \{0.9, 0.95, 0.97\}$, and weight decay (1).

3.3. Results

Method	Top-1 Performance	Top-3 Performance
OneCycle Momentum	0%	0%
Linear Momentum	0%	0%
Exp Momentum decay	0%	0%
Cosine Momentum	0%	3.57%
LR Exp decay	7.14%	10.71%
LR Decay on Plateau	7.14%	21.43%
OneCycle	7.14%	25.00%
LR Linear Schedule	10.71%	39.29%
LR Step Schedule	10.71%	53.57%
LR Cosine Schedule	17.86%	60.71%
DEMON	**39.29%**	**85.71%**

Figure 2. Top-1 and Top-3 performances (%) of different schedules ranked, out of a total of 28 experiments (1).

After 28 relevant experiments, results demonstrated in figures 2, 5, and 6 show that DEMON outperforms all the learning rate and momentum schedules tested, with Top-1 finishes at 39% and Top-3 finishes at 85% (1). Further, DEMON applied optimizers performed better than other optimization techniques mentioned. Ignoring SGDM and Adam optimizers, DEMON SGDM and DEMON Adam outperformed almost all and attained competitive error against other optimization techniques including AggMo plus LR Step, QHM plus LR Step, YellowFin, AMSGrad, AdamW, and QHAdam (1). Tuning only the learning rate for both DEMON Adam and Adam plus LR linear schedule following huggingface (25) implementation in BERT$_{BASE}$-GLUE, DEMON Adam yielded a slight improvement, see figure 4

(1). In figure 4, the results of DEMON against the effective learning rate adjusted SGD demonstrated that DEMON cannot be accurately approximated with SGD ELR (1).

In figure 3, the heatmaps display optimal performance of each optimizer over the full range of possible hyperparameters (1). As opposed to its vanilla counterparts of Adam and SGDM with LR Step schedule (bottom 3 heatmaps), DEMON applied optimizers (top 3 heatmaps) have larger bands of lighter color, indicating better performance for a wide range of hyperparameters (1). The results suggest that both DEMON Adam and DEMON SGDM are more robust and less sensitive to hyperparameter tuning, which are crucial for training neural network in practice (1). DEMON can be applied to any gradient descent algorithms that has a momentum parameter (1). Thus, DEMON is easy to implement and has many benefits such as not requiring additional tuning, incurring almost no extra computational costs, and outperforming its vanilla counterparts (1).

4. Quantization

Neural networks are generally trained with FP32 or FP64 weights in order store information and to be precise. Since neural networks these days consist of billions of parameters like the GPT 3 (29), Megatron (4) using high precision numbers causes the weights to have a size of multiple GigaBytes. This in turn then requires several GPU's (30) to train a single model. This leads to increased costs of training. The idea of quantization is to use weights with less precision, which require substantially lesser space than FP32, FP64 and try to maintain the accuracy of the network if not improve it while doing so. In this paper we will discuss a method for quantization called Cyclic Precision Training (3).

Works like (31) suggest that noise can help in training of the neural networks, (32) shows that a large learning rate helps Neural Networks to learn more general patterns. The conjecture from the paper CPT (3) is that a low precision behaves like a high learning rate and helps the Neural Networks to explore, while accurate updates with high precision helps the Neural Networks to converge. The authors of CPT (3) show with experimental backing that lowering the precision has a similar effect to having a large learning rate.

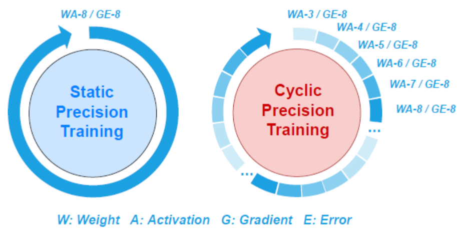

WA-8 / GE-8
Static Precision Training
WA-3 / GE-8
WA-4 / GE-8
WA-5 / GE-8
WA-6 / GE-8
WA-7 / GE-8
WA-8 / GE-8
Cyclic Precision Training
W: Weight A: Activation G: Gradient E: Error

Figure 9. Here we can see static precision training on the left side and the method proposed by the authors CPT (3) on the right side.

The main idea behind Cyclic precision training is to change the precision of the weights of the network cyclically between a lower bound and an upper bound instead of using fixed precision. The authors of CPT (3) suggest that we can implement cyclic precision with any cyclic scheduling method, but they use the cosine implementation throughout the experiments.

$$B_t^n = \lceil B_{min}^n + \frac{1}{2}(B_{max}^n - B_{min}^n)(1 - cos(\frac{t\%T_n}{T_n}\pi))\rfloor$$

B_t^n is the precision at the global step t. The bounds B_{min}^i and B_{max}^i can be found by a precision range test (3). Here $\%$ is the mod operation. Each cycle has a length of T_n. T_n is defined to be the total number of epochs divided by the N which is the total number of cycles of the precision schedule.

4.1. Results

The authors of CPT (3) performed experiments across many different tasks like CIFAR 10 (33), Imagenet (34), WikiText - 103 (35). They tested many different neural network architectures like Resnets, MobileNets (36), Tansformer(37) and LSTM (38). The precision lower bounds for each neural network are found using the precision range test though the upper bounds are fixed to be 8 (the precision of the baseline that the models are compared to). The periodic cycles (N) is set to 32.

Figure 7 explains the results of different SOTA baselines and compare it to the results of CPT. The experiment was performed on CIFAR 10 with the baselines being (DoReFa (26), WAGEUBN (27), SBM (28). We can see from the figure that across all the tests, CPT gains performance anywhere from 0.2 to 1.25 percent when compared to its baselines. While maintaining this performance, CPT also reduces the BitOps from 20 up to 37 percent. This reduction in bitOps and increased accuracy gives the CPT technique a win win situation over the SOTA baselines.

To test the scalability of CPT the authors test it on models like Resnet (16) 18,34,50 on Imagenet (34). While maintaining the accuracy it succeeds in reducing the Bitops up to 20 percent. For further testing CPT was trained on language models and was benchmarked on Wikitext-103 (35). Figure 8 shows us that CPT can be applied to language models. It again decreases the BitOps by 20 - 30 percent while maintaining or increasing the accuracy of the models. These result suggests that CPT is scalable to more complex models and data.

5. Future work

Learning rate, momentum and quantization schedules improve accuracy and efficiency. However, using them to-

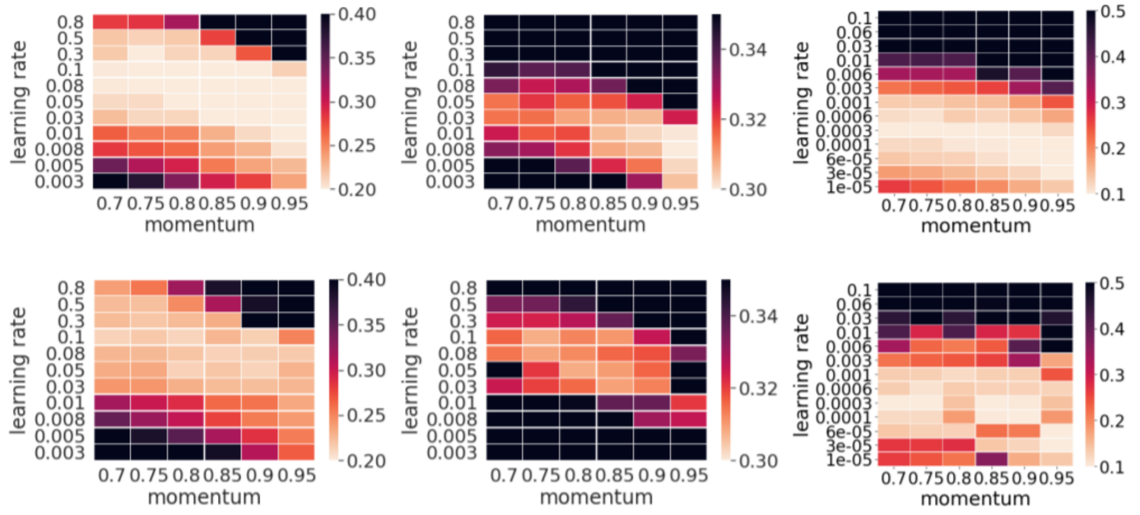


Figure 3. Error rates for WRN-STL10-DEMONSGDM-50epoch (top-left), VGG16-CIFAR100-DEMONSGDM-100epoch (top-middle) and RN20-CIFAR10-DEMONAdam-100epoch (top-right), WRN-STL10-SGDM-50epoch (bottom-left), VGG16-CIFAR100-SGDM-100epoch (bottom-middle), and RN20-CIFAR10-Adam-100epoch (bottom-right). Light-colored patches indicate better performance (1).

	Score	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
Adam + LR Linear Schedule	79.1	57.4	84.3	89.0	91.4	89.2	69.4	92.7	89.0	49.6
DEMON Adam	**79.7**	58.4	84.2	90.0	90.9	89.0	69.4	92.5	88.8	53.8

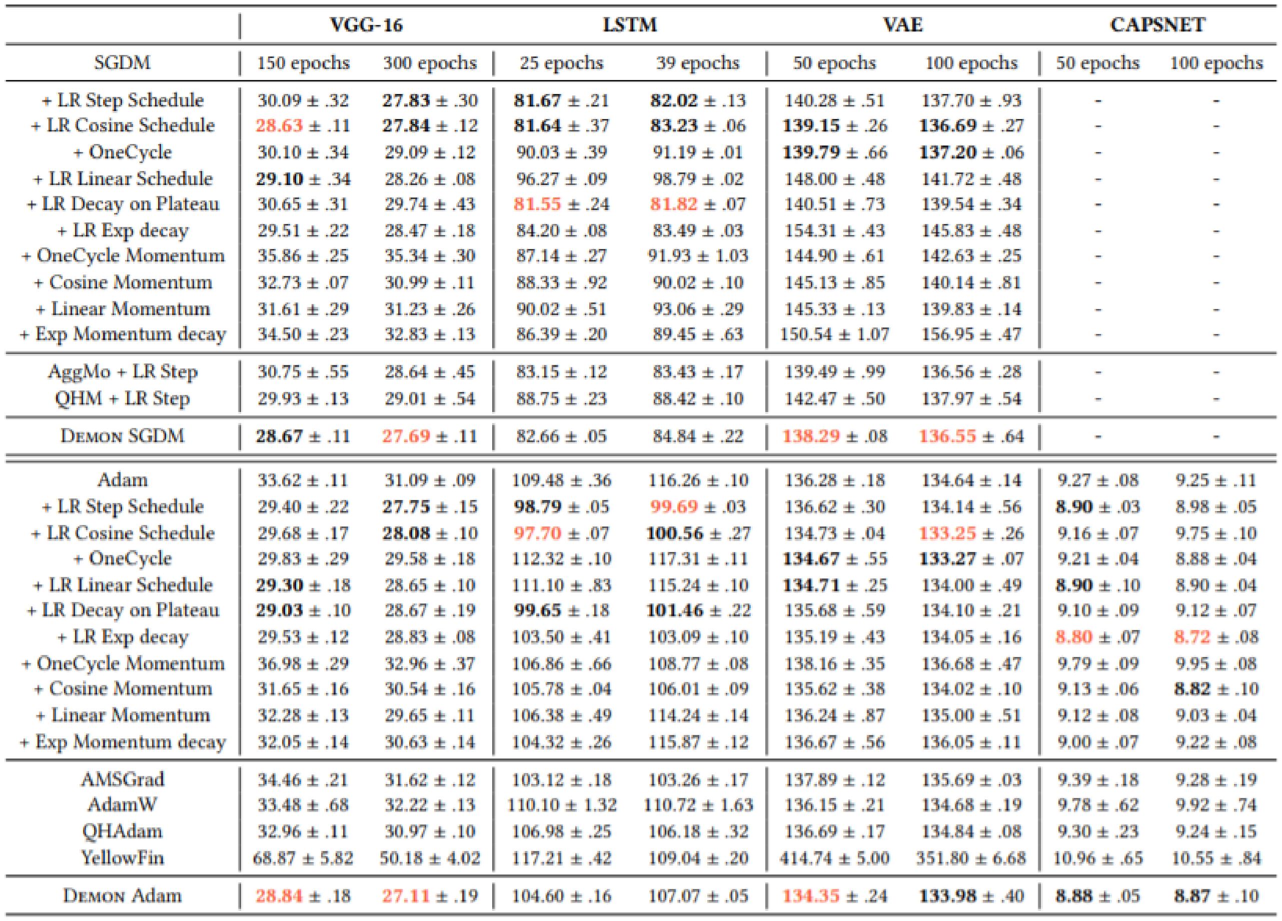
	RN-20			VGG-16			LSTM		VAE	
	75 epochs	150 epochs	300 epochs	50 epochs	100 epochs	200 epochs	25 epochs	39 epochs	50 epochs	100 epochs
SGD ELR	9.14 ± .24	8.58 ± .08	8.16 ± .15	35.71 ± .54	30.11 ± .28	29.21 ± .32	inf	inf	inf	inf
DEMON SGDM	**8.71** ± .24	**7.95** ± .15	**7.59** ± .12	**32.26** ± .21	**28.67** ± .11	**27.69** ± .11	**82.66** ± .105	**84.84** ± .22	**138.29** ± .08	**136.55** ± .64

Figure 4. Results of BERT$_{BASE}$-GLUE, Adam plus LR Linear Schedule follows the huggingface (25) implementation, and DEMON Adam (1).

gether does not always give the best performance. The correlation of quantization and momentum is still an open question, and we want to address it in future work by observing empirically its performance during training. In addition to that, and motivated by REX, we aim to find the "best" quantization schedule, as CPT only analyzes cyclic cosine schedule.

	ResNet 20			**Wide ResNet 16-8**			**ResNet 56**	
SGDM	75 epochs	150 epochs	300 epochs	50 epochs	100 epochs	300 epochs		
+ LR Step Schedule	**8.82** ± .25	8.43 ± .07	7.32 ± .14	22.42 ± .56	**17.20** ± .35	**14.51** ± .26	45.98 ± .23	41.66 ± .10
+ LR Cosine Schedule	**8.80** ± .08	8.10 ± .13	7.78 ± .14	**20.03** ± .26	**17.02** ± .24	14.66 ± .25	-	-
+ OneCycle	10.83 ± .25	9.23 ± .19	8.42 ± .12	21.67 ± .27	19.69 ± .21	19.00 ± .42	-	-
+ LR Linear Schedule	9.03 ± .24	**8.15** ± .12	**7.62** ± .12	19.54 ± .20	17.39 ± .24	**14.58** ± .18	-	-
+ LR Decay on Plateau	9.05 ± .07	8.26 ± .07	7.97 ± .14	21.05 ± .27	17.83 ± .39	15.16 ± .36	-	-
+ LR Exp decay	9.55 ± .09	9.20 ± .13	7.82 ± .05	22.65 ± .49	20.60 ± .21	15.85 ± .28	-	-
+ OneCycle Momentum	15.61 ± .39	14.02 ± .13	13.35 ± .58	29.34 ± .78	23.20 ± .39	25.42 ± .47	-	-
+ Cosine Momentum	13.57 ± .20	11.23 ± .22	10.87 ± .03	24.12 ± .34	21.66 ± .37	16.29 ± .26	-	-
+ Linear Momentum	12.31 ± .14	10.26 ± .16	10.63 ± .31	25.13 ± .28	22.74 ± .78	17.92 ± .41	-	-
+ Exp Momentum decay	14.96 ± .19	12.98 ± .15	12.35 ± .11	24.01 ± .20	19.35 ± .29	17.56 ± .21	-	-
AggMo + LR Step	8.71 ± .24	7.93 ± .15	7.62 ± .03	21.37 ± .32	17.15 ± .35	14.49 ± .26	-	-
QHM + LR Step	8.72 ± .14	7.95 ± .17	7.67 ± .10	21.75 ± .31	18.21 ± .48	14.44 ± .23	-	-
Demon SGDM	8.56 ± .10	**8.21** ± .18	**7.59** ± .12	**20.23** ± .31	16.19 ± .23	14.44 ± .53	44.87 ± .15	40.85 ± .01
Adam	13.63 ± .22	11.90 ± .06	11.94 ± .06	23.35 ± .20	**19.63** ± .26	18.65 ± .07	57.56 ± 1.50	50.89 ± .59
+ LR Step Schedule	10.47 ± .10	8.75 ± .17	**8.55** ± .05	23.85 ± .07	**19.63** ± .33	**18.29** ± .10	-	-
+ LR Cosine Schedule	**9.56** ± .12	9.15 ± .12	8.93 ± .07	22.85 ± .47	21.47 ± .31	19.08 ± .36	-	-
+ OneCycle	10.33 ± .20	9.87 ± .12	9.03 ± .18	20.02 ± .19	19.21 ± .28	19.03 ± .43	-	-
+ LR Linear Schedule	9.25 ± .12	9.20 ± .22	8.89 ± .05	**21.70** ± .11	21.53 ± .44	17.85 ± .15	-	-
+ LR Decay on Plateau	9.71 ± .39	**8.92** ± .18	8.80 ± .11	22.77 ± .33	19.91 ± .45	19.61 ± .56	-	-
+ LR Exp decay	10.48 ± .15	9.24 ± .16	**8.53** ± .07	23.30 ± .39	20.70 ± .50	19.63 ± .24	-	-
+ OneCycle Momentum	20.05 ± .91	15.60 ± .69	14.85 ± .50	24.61 ± .54	23.39 ± .39	23.54 ± .38	-	-
+ Cosine Momentum	11.08 ± .11	10.63 ± .20	10.64 ± .30	25.76 ± .22	23.58 ± .02	20.10 ± .15	-	-
+ Linear Momentum	11.91 ± .18	11.48 ± .13	11.09 ± .12	24.36 ± .31	21.93 ± .23	21.81 ± .36	-	-
+ Exp Momentum decay	15.18 ± .10	12.08 ± .16	10.63 ± .12	28.90 ± .21	25.28 ± .31	22.90 ± .41	-	-
AMSGrad	13.43 ± .14	11.83 ± .12	10.48 ± .12	21.73 ± .25	19.35 ± .20	18.21 ± .18	-	-
AdamW	12.46 ± .52	11.38 ± .21	10.50 ± .17	20.39 ± .62	18.55 ± .23	17.00 ± .41	-	-
QHAdam	15.55 ± .25	13.78 ± .08	13.36 ± .11	21.25 ± .22	19.81 ± .18	18.52 ± .25	-	-
YellowFin	13.66 ± .34	12.13 ± .41	11.39 ± .16	22.55 ± .14	20.68 ± .04	18.56 ± .33	-	-
Demon Adam	**9.68** ± .07	**8.90** ± .18	8.50 ± .12	**20.95** ± .23	**19.50** ± .32	**18.62** ± .41	48.92 ± .03	45.72 ± .31

Figure 5. Generalization Errors for RN20-CIFAR10, WRN-STL10 and RN56-TINYIMAGENET. Ignoring non SGDM and Adam optimizers, red indicates Top-1 performance and bold indicates Top-3 performance (1).



	VGG-16		LSTM		VAE		CAPSNET	
SGDM	150 epochs	300 epochs	25 epochs	39 epochs	50 epochs	100 epochs	50 epochs	100 epochs
+ LR Step Schedule	30.09 ± .32	**27.83** ± .30	**81.67** ± .21	**82.02** ± .13	140.28 ± .51	137.70 ± .93	-	-
+ LR Cosine Schedule	28.63 ± .11	**27.84** ± .12	**81.64** ± .37	**83.23** ± .06	**139.15** ± .26	**136.69** ± .27	-	-
+ OneCycle	30.10 ± .34	29.09 ± .12	90.03 ± .39	91.19 ± .01	**139.79** ± .66	**137.20** ± .06	-	-
+ LR Linear Schedule	**29.10** ± .34	28.26 ± .08	96.27 ± .09	98.79 ± .02	148.00 ± .48	141.72 ± .48	-	-
+ LR Decay on Plateau	30.65 ± .31	29.74 ± .43	81.55 ± .24	81.82 ± .07	140.51 ± .73	139.54 ± .34	-	-
+ LR Exp decay	29.51 ± .22	28.47 ± .18	84.20 ± .08	83.49 ± .03	154.31 ± .43	145.83 ± .48	-	-
+ OneCycle Momentum	35.86 ± .25	35.34 ± .30	87.14 ± .27	91.93 ± 1.03	144.90 ± .61	142.63 ± .25	-	-
+ Cosine Momentum	32.73 ± .07	30.99 ± .11	88.33 ± .92	90.02 ± .10	145.13 ± .85	140.14 ± .81	-	-
+ Linear Momentum	31.61 ± .29	31.23 ± .26	90.02 ± .51	93.06 ± .29	145.33 ± .13	139.83 ± .14	-	-
+ Exp Momentum decay	34.50 ± .23	32.83 ± .13	86.39 ± .20	89.45 ± .63	150.54 ± 1.07	156.95 ± .47	-	-
AggMo + LR Step	30.75 ± .55	28.64 ± .45	83.15 ± .12	83.43 ± .17	139.49 ± .99	136.56 ± .28	-	-
QHM + LR Step	29.93 ± .13	29.01 ± .54	88.75 ± .23	88.42 ± .10	142.47 ± .50	137.97 ± .54	-	-
Demon SGDM	**28.67** ± .11	27.69 ± .11	82.66 ± .05	84.84 ± .22	138.29 ± .08	136.55 ± .64	-	-
Adam	33.62 ± .11	31.09 ± .09	109.48 ± .36	116.26 ± .10	136.28 ± .18	134.64 ± .14	9.27 ± .08	9.25 ± .11
+ LR Step Schedule	29.40 ± .22	**27.75** ± .15	**98.79** ± .05	99.69 ± .03	136.62 ± .30	134.14 ± .56	**8.90** ± .03	8.98 ± .05
+ LR Cosine Schedule	29.68 ± .17	**28.08** ± .10	97.70 ± .07	**100.56** ± .27	134.73 ± .04	133.25 ± .26	9.16 ± .07	9.75 ± .10
+ OneCycle	29.83 ± .29	29.58 ± .18	112.32 ± .10	117.31 ± .11	**134.67** ± .55	**133.27** ± .07	9.21 ± .04	8.88 ± .04
+ LR Linear Schedule	**29.30** ± .18	28.65 ± .10	111.10 ± .83	115.24 ± .10	**134.71** ± .25	134.00 ± .49	**8.90** ± .10	8.90 ± .04
+ LR Decay on Plateau	**29.03** ± .10	28.67 ± .19	**99.65** ± .18	**101.46** ± .22	135.68 ± .59	134.10 ± .21	9.10 ± .09	9.12 ± .07
+ LR Exp decay	29.53 ± .12	28.83 ± .08	103.50 ± .41	103.09 ± .10	135.19 ± .43	134.05 ± .16	8.80 ± .07	8.72 ± .08
+ OneCycle Momentum	36.98 ± .29	32.96 ± .37	106.86 ± .66	108.77 ± .08	138.16 ± .35	136.68 ± .47	9.79 ± .09	9.95 ± .08
+ Cosine Momentum	31.65 ± .16	30.54 ± .16	105.78 ± .04	106.01 ± .09	135.62 ± .38	134.02 ± .10	9.13 ± .06	**8.82** ± .10
+ Linear Momentum	32.28 ± .13	29.65 ± .11	106.38 ± .49	114.24 ± .14	136.24 ± .87	135.00 ± .51	9.12 ± .08	9.03 ± .04
+ Exp Momentum decay	32.05 ± .14	30.63 ± .14	104.32 ± .26	115.87 ± .12	136.67 ± .56	136.05 ± .11	9.00 ± .07	9.22 ± .08
AMSGrad	34.46 ± .21	31.62 ± .12	103.12 ± .18	103.26 ± .17	137.89 ± .12	135.69 ± .03	9.39 ± .18	9.28 ± .19
AdamW	33.48 ± .68	32.22 ± .13	110.10 ± 1.32	110.72 ± 1.63	136.15 ± .21	134.68 ± .19	9.78 ± .62	9.92 ± .74
QHAdam	32.96 ± .11	30.97 ± .10	106.98 ± .25	106.18 ± .32	136.69 ± .17	134.84 ± .08	9.30 ± .23	9.24 ± .15
YellowFin	68.87 ± 5.82	50.18 ± 4.02	117.21 ± .42	109.04 ± .20	414.74 ± 5.00	351.80 ± 6.68	10.96 ± .65	10.55 ± .84
Demon Adam	28.84 ± .18	27.11 ± .19	104.60 ± .16	107.07 ± .05	134.35 ± .24	**133.98** ± .40	**8.88** ± .05	**8.87** ± .10

Figure 6. Generalization Errors for VGG16-CIFAR100 and CAPS-FMNIST, LSTM-PTB generalization perplexity, and VAE-MNIST generalization loss. Ignoring non SGDM and Adam optimizers, red indicates Top-1 performance and bold indicates Top-3 performance (1).

Model	Method	Precision (FW/BW)	CIFAR-10 Acc (%)	CIFAR-100 Acc (%)	GBitOPs	Latency (hour)
ResNet-74	DoReFa	8 / 8	91.16	69.31	2.67e8	44.6
	WAGEUBN	8 / 8	91.35	69.61	2.67e8	44.6
	SBM	8 / 8	92.57	71.44	2.67e8	44.6
	Proposed CPT	**3 - 8 / 8**	**93.23**	**72.35**	**1.68e8**	**35.04**
	Improv.		**+0.66**	**+0.91**	**-37.1%**	**-21.4%**
ResNet-74	DoReFa	6 / 6	90.94	69.01	1.50e8	33.2
	WAGEUBN	6 / 6	91.01	69.37	1.50e8	33.2
	SBM	6 / 6	91.15	70.31	1.50e8	33.2
	Proposed CPT	**3 - 6 / 6**	**92.4**	**70.83**	**1.05e8**	**27.5**
	Improv.		**+1.25**	**+0.52**	**-30.0%**	**-17.2%**
ResNet-164	DoReFa	8 / 8	91.40	70.90	6.04e8	101.9
	WAGEUBN	8 / 8	92.5	71.86	6.04e8	101.9
	SBM	8 / 8	93.63	72.53	6.04e8	101.9
	Proposed CPT	**3 - 8 / 8**	**93.83**	**72.9**	**3.8e8**	**80.5**
	Improv.		**+0.20**	**+0.37**	**-37.1%**	**-21.0%**
ResNet-164	DoReFa	6 / 6	91.13	70.53	3.40e8	76.7
	WAGEUBN	6 / 6	92.44	71.50	3.40e8	76.7
	SBM	6 / 6	91.95	70.34	3.40e8	76.7
	Proposed CPT	**3 - 6 / 6**	**93.02**	**71.79**	**2.37e8**	**63.5**
	Improv.		**+1.07**	**+0.29**	**-30.3%**	**-17.2%**
MobileNetV2	DoReFa	8 / 8	91.03	70.17	1.49e8	26.2
	WAGEUBN	8 / 8	92.32	71.45	1.49e8	26.2
	SBM	8 / 8	93.57	75.28	1.49e8	26.2
	Proposed CPT	**4 - 8 / 8**	**93.76**	**75.65**	**1.04e8**	**21.6**
	Improv.		**+0.19**	**+0.37**	**-30.2%**	**-17.6%**
MobileNetV2	DoReFa	6 / 6	90.25	68.4	8.39e7	18.4
	WAGEUBN	6 / 6	91.00	71.05	8.39e7	18.4
	SBM	6 / 6	91.56	72.31	8.39e7	18.4
	Proposed CPT	**4 - 6 / 6**	**91.81**	**73.18**	**6.63e7**	**15.7**
	Improv.		**+0.25**	**+0.87**	**-21.0%**	**-14.7%**

Figure 7. Results of CPT (3) with baselines of DoReFa, (26) , WAGEUBN (27) and SBM (28) for different ResNet Models and MobileNetV2 model on CIFAR 10 and 100

Model / Dataset	Method	Precision (FW/BW)	Perplexity	GBitOPs	Precision	Perplexity	GBitOPs
Transformer WikiText-103	SBM	8 / 8	31.77	1.44e6	6 / 8	32.41	9.87e5
	Proposed CPT	**4 - 8 / 8**	**30.22**	**1.0e6**	**4 - 6 / 8**	**31.0**	**7.66e5**
	Improv.		**-1.55**	**-30.2%**		**-1.41**	**-22.4%**
2-LSTM PTB	SBM	8 / 8	96.95	4.03e3	6 / 8	97.47	2.77e3
	Proposed CPT	**5 - 8 / 8**	**96.39**	**3.09e3**	**5 - 6 / 8**	**97.0**	**2.48e3**
	Improv.		**-0.56**	**-23.2%**		**-0.47**	**-10.5%**

Figure 8. Results of CPT (3) on language model tasks with baselines of SBM (28)

References

[1] John Chen and Anastasios Kyrillidis. Decaying momentum helps neural network training. *CoRR*, abs/1910.04952, 2019.

[2] John Chen, Cameron R. Wolfe, and Anastasios Kyrillidis. REX: revisiting budgeted training with an improved schedule. *CoRR*, abs/2107.04197, 2021.

[3] Yonggan Fu, Han Guo, Meng Li, Xin Yang, Yining Ding, Vikas Chandra, and Yingyan Lin. Cpt: Efficient deep neural network training via cyclic precision, 2021.

[4] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020.

[5] 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, 2020.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[7] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[8] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[9] Leslie N Smith. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

[10] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[11] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[12] Jian Zhang and Ioannis Mitliagkas. Yellowfin and the art of momentum tuning. *arXiv preprint arXiv:1706.03471*, 2017.

[13] Jerry Ma and Denis Yarats. Quasi-hyperbolic momentum and adam for deep learning. *arXiv preprint arXiv:1810.06801*, 2018.

[14] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

[15] James Lucas, Shengyang Sun, Richard Zemel, and Roger Grosse. Aggregated momentum: Stability through passive damping. *arXiv preprint arXiv:1804.00325*, 2018.

[16] 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.

[17] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[18] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[19] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[20] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[21] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. *arXiv preprint arXiv:1710.09829*, 2017.

[22] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *arXiv preprint arXiv:1907.05600*, 2019.

[23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[24] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[25] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[26] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients, 2018.

[27] Yukuan Yang, Shuang Wu, Lei Deng, Tianyi Yan, Yuan Xie, and Guoqi Li. Training high-performance and large-scale deep neural networks with full 8-bit integers, 2019.

[28] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. Scalable methods for 8-bit training of neural networks, 2018.

[29] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[30] Geoff Pleiss, Danlu Chen, Gao Huang, Tongcheng Li, Laurens van der Maaten, and Kilian Q. Weinberger. Memory-efficient implementation of densenets, 2017.

[31] Arvind Neelakantan, Luke Vilnis, Quoc V. Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks, 2015.

[32] Yuanzhi Li, Colin Wei, and Tengyu Ma. Towards explaining the regularization effect of initial large learning rate in training neural networks, 2020.

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

[34] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[35] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

[36] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[38] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

Literature Review on Lottery Ticket Hypothesis

Shijie Fan [1] **Yuchen Gu** [1] **Hannah Lei** [1]

Abstract

Deep Neural Network (DNN) is becoming popular in multiple applications due to its record-breaking predictive performance. However, this usually comes with prohibitive costs in training, storage, and inference due to the large scale of training data and model parameters. Pruning, which eliminates unnecessary weights from neural networks, is a standard method to lower costs of storage and inference by reducing parameter counts and still maintaining comparable accuracy. The lottery ticket hypothesis states that there exists a subnetwork in a randomly-initialized, dense neural network that can reach a matching accuracy of the full network in the same or fewer iterations when trained alone. Beyond pruning, this hypothesis suggests an exciting potential opportunity to further improve training performance and reduce its cost. This review will examine the Lottery Ticket Hypothesis, identify its problems and show current improvements. Finally, a practical implementation of the Lottery Ticket Hypothesis, called Early-Bird Ticket, will be introduced.

1. Introduction

Deep Neural Network (DNN) is gaining popularity in multiple applications due to its record-breaking predictive performance. However, due to the large scale of data and model parameters, the performance of Deep Neural Network (DNN) usually comes with prohibitive costs in training, storage, and inference (You et al., 2020). The recent trend of improving DNN's accuracy vs. cost efficiency involves compressing the models using pruning, a technique that eliminates unnecessary weights from neural networks. The standard approach consists of reducing parameter counts by pruning after training and retraining in order to restore comparable accuracy. Although pruning can reach a matching accuracy after removing a significant number of parameters, as it often happens late in the training or after the training, the training cost is still high. In addition, recent experiences show that the sparse architectures yielded by pruning are difficult to train from the beginning, (Frankle & Carbin, 2019) which prevents further improvement in DNN's efficiency.

But, (Frankle & Carbin, 2019) found out that a standard pruning technique naturally uncovers subnetworks whose initializations made them capable of training effectively. Based on this observation, they articulate the Lottery Ticket Hypothesis, which states that there exists a subnetwork in a randomly-initialized, dense neural network that can reach a matching accuracy of the full network in the same or fewer iterations when trained alone. This subnetwork is the winning ticket of the lottery and the hypothesis suggests a new possibility to reduce the cost of training if we can locate this winning ticket efficiently.

In the same paper that the hypothesis is proposed, the algorithm to find such a winning ticket is given and it still requires multiple rounds of training and pruning. The winning ticket is only found after fully training a dense neural network. Beyond this, the Lottery Ticket Hypothesis does not work well on every dataset and network. This review will cover current investigations into these problems and their corresponding proposed improvements such as rewinding techniques and Early-Bird Ticket.

In conclusion, the Lottery Ticket Hypothesis does present an exciting potential opportunity to further improve training performance and reduce its cost (Frankle & Carbin, 2019).

2. The Lottery Ticket Hypothesis

2.1. Definition

The Lottery Ticket Hypothesis (Frankle & Carbin, 2019) : *A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.*

The formal definition is (Frankle & Carbin, 2019):

[*]Equal contribution [1]Rice University, TX, Houston, United States. Correspondence to: Shijie Fan <sf24@rice.edu>, Yuchen Gu <yg50@rice.edu>, Hannah Lei <hyl3@rice.edu>.

Consider a dense feed-forward neural network $f(x;\theta)$ with initial parameters $\theta = \theta_0 \sim D_\theta$. When optimizing with stochastic gradient descent (SGD) on a training set, f reaches minimum validation loss l at iteration j with test accuracy a. In addition, consider training $f(x; m \odot \theta)$ with a mask $m \in \{0, 1\}$ on its parameters such that its initialization is $m \odot \theta_0$. When optimizing with SGD on the same training set (with m fixed), f reaches minimum validation loss l' at iteration j' with test accuracy a'.

The lottery ticket hypothesis predicts that $\exists m$ for which $j' \leq j$ (commensurate training time), $a' \geq a$ (commensurate accuracy), and $\|m\|_0 \ll \|\theta\|$ (fewer parameters). These trainable subnetworks $f(x; m \odot \theta)$ are called *winning tickets*.

2.2. Identification

The paper (Frankle & Carbin, 2019) shows the method to identify a winning ticket from a dense network. It is as follows:

1. Randomly initialize a neural network $f(x;\theta_0)$
2. Train the network for j iterations and have θ_j
3. Prune $p\%$ of θ_j by creating a mask m
4. Reset the remaining parameters to their values in θ_0, creating the winning ticket $f(x; m \odot \theta_0)$

Random reinitialization: instead of resetting the remaining parameters to θ_0 at step 4, random reinitialization randomly samples a new initialization θ_0' for the pruned subnetwork to be reset to.

The *one-shot* pruning technique is to prune $p\%$ of parameters and reset after training once. The iterative method will be pruning $p^{\frac{1}{n}}\%$ of parameters and reset after each of the n total rounds of training.

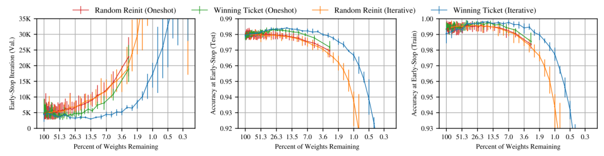


Figure 1: Early-stopping iteration and accuracy of Lenet under one-shot and iterative pruning. Average of five trials; error bars for the minimum and maximum values (Frankle & Carbin, 2019).

As in Figure 1 the iterative method produces higher train and test accuracy with a earlier stopping iteration, the paper (Frankle & Carbin, 2019) focuses on the iterative pruning method.

2.3. Performance

2.3.1. Fully-Connected Network

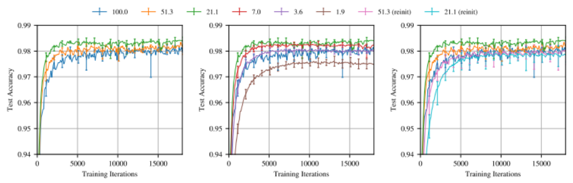


Figure 2: Test accuracy on Lenet (iterative pruning) as training proceeds. Each curve is the average of five trials. Labels are the fraction of weights remaining in the network after pruning. Error bars are the minimum and maximum of any trial (Frankle & Carbin, 2019).

Figure 2 shows how winning tickets (networks with 51.3% and 21.1% of weights remaining) arrive at higher test accuracy in fewer training iterations. It also shows that when the percentage of remaining weights is below 21.1%, the performance is deteriorating and is at the same level as the original full network when only 3.6% of weights remain. This result suggests that there exist a significant number of unnecessary parameters in a dense network, which can be removed without hurting the performance. In addition, it also implies that aggressive pruning might lead to a worse result as it might damage the structure of the subnetwork.

The right plot in Figure 2 also includes a comparison between subnetworks discovered by following steps in 2.2 and subnetworks discovered by using random reinitialization mentioned in 2.2. These two random reinitialized subnetworks use the same pruning rates that yield two winning tickets but they learn slower and produce worse accuracy than the original full network, which shows the importance of the initialization of the winning ticket.

2.3.2. Convolutional Network

The same trend also holds in the convolutional networks shown in Figure 3. Before the pruning goes too aggressive, the pruned subnetworks learn faster and produce more accurate test results. All three networks remain above their original average test accuracy when more than 2% of parameters remain. Random reinitialized subnetworks still learn slower and produce worse test accuracy.

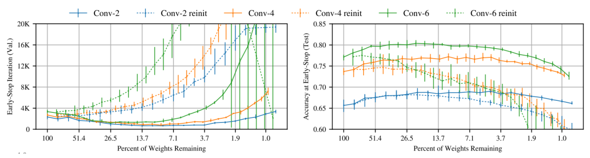


Figure 3: Early-stopping iteration and test accuracy of the Conv-2/4/6 architectures when iteratively pruned and when randomly reinitialized. Each solid line is the average of five trials; each dashed line is the average of fifteen reinitializations (three per trial) (Frankle & Carbin, 2019).

3. Problems of the Lottery Ticket Hypothesis

3.1. Performance on VGG-19

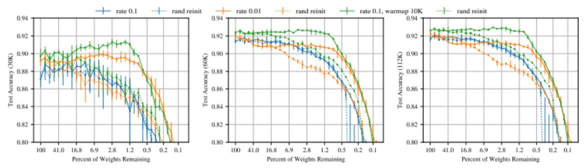


Figure 4: Test accuracy (at 30K, 60K, and 112K iterations) of VGG-19 when iteratively pruned (Frankle & Carbin, 2019).

The original method mentioned in 2.2 fails to identify any winning ticket for VGG-19 network on CIFAR-10. As Figure 4 shows, when using 0.1 as the learning rate, the subnetwork performs the same as random reinitialized ones and cannot reach a matching accuracy as the full network. When using a smaller learning rate, 0.01, the subnetwork does perform better but still cannot match the original accuracy, which means that it is not a winning ticket as well. More importantly, the early accuracy advantage disappears gradually as the training goes due to the small learning rate.

In order to locate a winning ticket for this network, (Frankle & Carbin, 2019) applied a linear learning rate warmup technique, where the learning rate goes from 0 to the initial learning rate (0.1) linearly in 10000 iterations. Although using this technique can successfully find a winning ticket, the long learning rate warmup step is still a large training cost.

3.2. Performance on Resnet-18

The performance on Resnet-18 on CIFAR-10 is similar to that on VGG-19. Notably, (Frankle & Carbin, 2019) cannot find a winning ticket even with the learning rate warmup for learning rate 0.1.

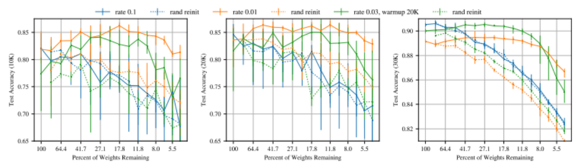


Figure 5: Test accuracy (at 10K, 20K, and 30K iterations) of Resnet-18 when iteratively pruned (Frankle & Carbin, 2019).

3.3. Explanation by SGD Instability Analysis

The stochastic gradient descent (SGD) noise is caused by SGD's inherent randomness as training data are presented to the network in a random mini-batch order within each epoch (Frankle et al., 2020a). The stability to this noise can be measured by training two copies of a network with different SGD noises at a certain iteration and comparing the similarities of the two resulting networks. (Frankle et al., 2020a) uses the line between the two resulting networks on the optimization landscape to measure this stability. If there is no increase in the error along the path, it means that two networks find the same minimum and the network is stable to the SGD noise at this point. If there is a significant increase of error ("a barrier of increased error" (Frankle et al., 2020a)), it indicates that two resulting networks are in different minimums and the network is still unstable to SGD noises.

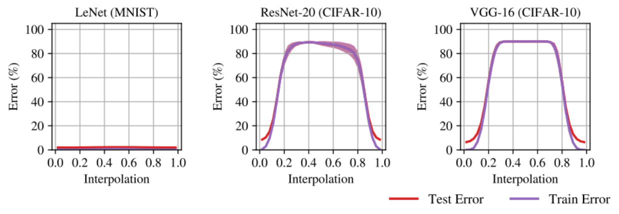


Figure 6: Error when linearly interpolating between networks trained from the same initialization with different SGD noise. Lines are means and standard deviations over three initializations and three data orders (nine samples total). Trained networks are at 0.0 and 1.0 (Frankle et al., 2020a).

Figure 6 shows that at the initialization, only Lenet on MNIST is stable as it does not see an increasing error along the line between two separately trained networks. This explains why the Lottery Ticket Hypothesis works well on Lenet but not on VGG or Resnet.

In Figure 7, both VGG and Resnet gradually become stable at around iteration 1000. Experiments by (Frankle et al.,

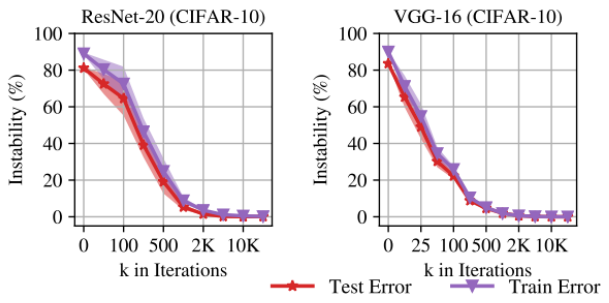


Figure 7: Linear interpolation instability when starting from step k. Each line is the mean and standard deviation across three initializations and three data orders (nine samples in total) (Frankle et al., 2020a).

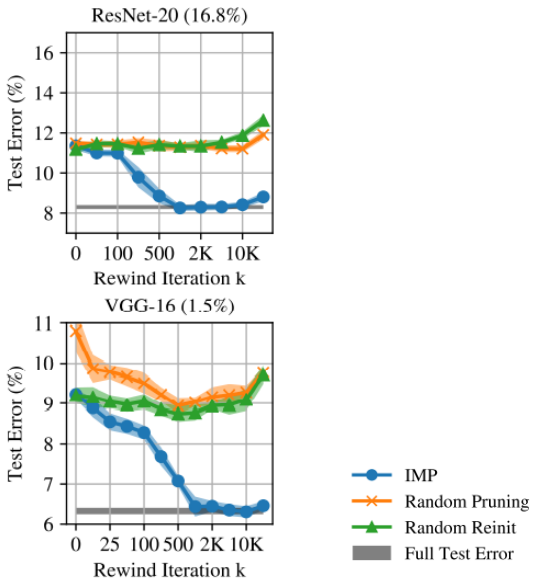


Figure 8: Test error of subnetworks created using the state of the full network at step k and applying a pruning mask. Lines are means and standard deviations over three initializations and three data orders (nine in total). Percents are weights remaining (Frankle et al., 2020a).

2020a) in Figure 8 show an interesting coincidence that if rewinding to weights at around iteration 1000 instead of resetting to the initialization as 2.2, both VGG and Resnet can produce a matching test accuracy as the full original network early in the training. As the networks from random pruning and random reinitialization show an higher test error rate at the same time, the importance of both the structure (pruning) and the initialization (rewinding) of the winning ticket is shown.

4. Improvements on the Lottery Ticket Hypothesis

The Lottery Ticket Hypothesis shows that pruning early in training with sub-networks trained in isolation can reach full accuracy. The sub-networks are as small as those found by inference-focused pruning methods after training, making it possible to maintain the sparsity level for most of the training. But this does not suggest we can find these sub-networks without first fully training the network. To find the sub-networks efficiently, several methods have been proposed.

4.1. Early-Bird Ticket

4.1.1. Overview

To close the gap between the winning ticket observation and the goal of more efficient training, (You et al., 2020) presented the **Early-Bird (EB)** ticket: the winning tickets that can be identified at an early training stage with low-cost training algorithms. These EB tickets can be detected with a mask distance without accessing the ground-truth winning tickets (i.e. the original tickets drawn after full training). After the identification, re-training these tickets can result in comparable or even better accuracies.

4.1.2. Definition

Similar to the lottery ticket hypothesis, the EB ticket states that:

Consider a dense randomly-initialized network $f(x;\theta)$ and a subnetwork $f(x;m\odot\theta)$, where $m\in\{0,1\}$ is the mask representing the pruned and unpruned connections in $f(x;\theta)$. When optimized with SGD on the same training data set, $f(x;\theta)$ reaches the minimum validation loss f_{loss} with test accuracy f_{acc} at i th iteration; $f(x;m\odot\theta_t)$ reaches the minimum validation loss f'_{loss} with test accuracy f'_{acc}, where $\theta_t)$ denotes the weights at the t th iteration. The EB tickets hypothesis states that $\exists f\ s.t.\ f'_{acc}\approx f_{acc}$ (even $\geq$) with $t\ll i$ (e.g., early stopping) and a sparse m (i.e., much reduced parameters).

4.1.3. Findings

Following the main idea of the lottery ticket hypothesis but instead pruning the networks trained at a much earlier stage, the authors observe that 1) there consistently exist EB tickets drawn at certain early epoch ranges that outperform those in later stages, even the ground-truth winning ticket. 2) some EB tickets are able to outperform even their unpruned, fully-trained models potentially due to the sparse regularization learned by EB tickets. The authors also show that large learning rates are important to the emergence of EB tickets, even extending to EB tickets. Also, they observed that low-precision training does not destroy EB tickets, which leads to cost saving in finding EB tickets, since low-precision updates can aggressively save energy compared to their full-precision baseline.

4.1.4. IMPLEMENTATION

Mask Distance By denoting the pruned channels as 0 while the kept ones as 1, the original network can be mapped to a binary mask of the drawn ticket. Hamming distance is used to calculate the *mask distance* between any two subnetworks pruned from the same dense model. Through the experiments of different models, the authors observe patterns of the pairwise mask distance matrices as shown in Figure 9.

Each (i, j)-th element in a matrix denotes the mask distance between subnetworks drawn from the i-th and j-th epochs in that corresponding experiment. The diagonal line has the lowest value since it is the epoch's mask compared with itself. Therefore, the more an element deviates from the diagonal, the more distant the two epochs are away from each other. In the matrix, a lower value (close to 0) indicates a smaller mask distance and is highlighted with a warmer color. It demonstrates that the EB tickets perform consistently: 1) the mask distances change rapidly at first, which is represented by the colors change from yellow to green from the diagonal to off-diagonal; 2) after around 10 epochs, the mask distance changes mildly, which is marked by the off-diagonal elements changes to yellow as well; 3) after around 80 epochs, the mask distance remains almost unchanged.

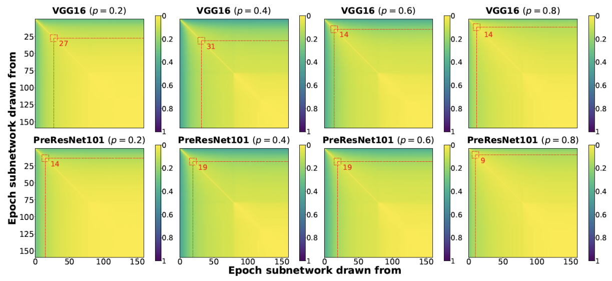


Figure 9: Visualization of the pairwise mask distance matrix for VGG16 and PreResNet101 on CIFAR-100 (You et al., 2020).

EB Train Algorithm 1 (You et al., 2020) describes the steps of searching the EB tickets. The returned EB ticket will be retrained further to reach the target accuracy. The training of EB tickets differs from the lottery ticket hypothesis in that EB train uses the unpruned weights from the drawn EB ticket instead of rewinding to the original initialization. This is marked by (Frankle et al., 2020b)'s work showing that deeper networks are not robust to reinitialization with untrained weights.

Algorithm 1 The Algorithm for Searching EB Tickets

```
Initialize the weights $W$, scaling factor $r$, pruning ratio $p$, and the FIFO queue $Q$ with length $l$;
while $t$ (epoch) $\leq t_{max}$ do
  update $W$ and $r$ using SGD training;
  Perform structured pruning based on rt towards the target ratio $p$;
  Calculate the mask distance between the current and last subnetworks and add to $Q$.
  $t = t + 1$
  if MAX $(Q) \leq \varepsilon$ then
    $t^* = t$
    Return $f(x; m_t^* \odot W)$ (EB ticket);
  end if
end while
```

4.1.5. SIGNIFICANCE

EB ticket is an actual implementation of the lottery ticket hypothesis and it is the first time showing how winning tickets can be selected at an early stage. The authors have demonstrated that EB tickets exist in both the standard training and in lower-cost schemes. Moreover, EB train with low-precision search may provide more insights in future areas of efficient training.

4.2. Stabilizing the Lottery Ticket Hypothesis

4.2.1. OVERVIEW

In this section, we will explore the idea of modifying **Iterative Magnitude Pruning (IMP)** to search for sub-networks that could have been obtained by pruning early in training rather than at iteration 0 (Frankle & Carbin, 2019).

4.2.2. CLARITY

Rewinding. The authors demonstrate that there exist subnetworks of deeper networks (i.e., Resnet-50, Squeezenet, Inception-v3) at early points in training (0.1% to 7% through) that are 50% to 99% smaller and that can complete the training process to match the original network's accuracy (Liu et al., 2018). The authors show this in Figure 10 by modifying IMP to rewind pruned sub-network weights to their former values at iteration k rather than resetting them to iteration 0.

Stability. To explain why IMP fails when resetting to iteration 0 and improves rapidly when rewinding later, they introduce sub-network stability: the distance between two trained copies of the same sub-network subjected to different noise. In particular, they focus on the noise introduced by pruning (comparing the trained weights of the full network

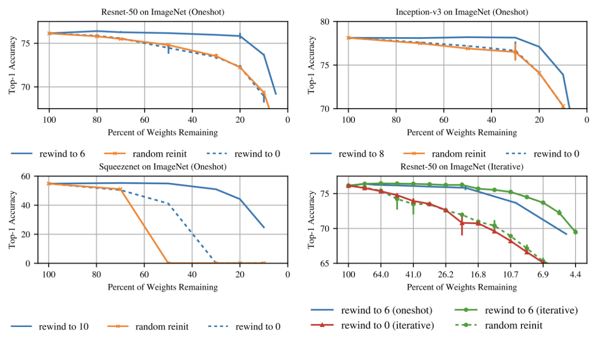


Figure 10: Rewinding to iterations early in training produces sub-networks that outperform the original networks, even when resetting to iteration 0 does not.

and sub-network) and data order (comparing the weights of two sub-networks trained with different data orders). They hypothesize that improved stability to pruning means that a sub-network comes closer to the original optimum and thereby accuracy; improvements in stability to data order mean the sub-network can do so consistently in spite of the noise intrinsic to Stochastic Gradient Descent.

4.2.3. Significance

The significance of the paper is the act of pruning earlier in training. Doing so could make it possible to reduce the cost of training networks by substantially reducing parameter-counts for most or all of training. Specifically, they find that, for many networks, there is an iteration early in training after which pruning can result in sub-networks with far higher accuracy than when pruning at initialization. The results with IMP expand the range of known opportunities to prune early in training that—if exploited—could reduce the cost of training. With better techniques, they expect this range could be expanded even further because our results are restricted by IMP's limitations. Namely, it is possible that there are equally-capable sub-networks present at initialization, but IMP is unable to find them.

4.2.4. Novelty

The novelty of the paper is that they are the first to show that it is possible to prune (1) so early in training (2) to such extreme levels of sparsity (3) on such large-scale tasks. The lottery ticket hypothesis hints at future techniques that identify small, trainable sub-networks capable of matching the accuracy of the larger networks they typically train. To date, this and other related research have focused on compressing neural networks before training. In this work, they find that other moments early in the training process may present better opportunities for this class of techniques. In doing so, they shed new light on the lottery ticket hypothesis and its manifestation in deeper networks through the lens of stability

4.3. Pruning at Initialization

4.3.1. Overview

Besides IMP, many other works have explored the possibility of pruning neural networks at initialization. By accessing the methods such as SNIP, GrasSP, SynFlow, and magnitude pruning, this paper (Frankle et al., 2021) shows how per-layer choice of the fraction of weights to prune can replace the per-weight pruning decisions. This property provides more insights into the pruning heuristics and the desire to prune at initialization.

4.3.2. Methods

A baseline used in this paper is *magnitude pruning after initialization*, which is a standard one-shot pruning that prunes the weights at the end of training. By comparing and evaluating the early pruning methods at initialization, Figure 11 shows the performance of magnitude pruning (green), SNIP (red), GraSP (purple), and SynFlow (brown) at initialization. It also includes the accuracy of magnitude pruning after training (blue), random pruning at initialization (orange), and the unpruned network (gray).

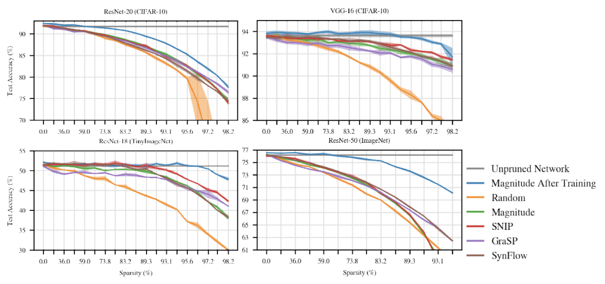


Figure 11: Accuracy of early pruning methods when pruning at initialization to various sparsities (Frankle et al., 2021).

Overall, these early-pruning methods generally outperform random pruning, but cannot match magnitude pruning after training in terms of either accuracy or the sparsities at which they match full accuracy.

4.3.3. Findings

To analyze why these methods perform differently than the baseline, magnitude pruning after training, the authors adopted ablation studies and evaluate the information that each method extracts about the network.

Through random shuffling, the authors find that all methods can maintain equal or even higher accuracy. Thus, random shuffling is not heavily affected by the weights to remove, but related to the layerwise proportion by which to prune the network. The authors then utilize reinitialization to investigate whether the methods are sensitive to the specific weights at initialization. Still, all of these early pruning techniques are insensitive to reinitialization and can maintain the same accuracy. This finding shows that it's possible to reinitialize or layerwise shuffle unpruned weights without hurting accuracy, so the useful part is the layerwise proportion instead of the specific weights. However, in broader pruning literature, such as magnitude pruning after training, shuffling and reinitialization may lead to lower accuracy. So the current finding raises the question of whether the insensitivity to shuffling or initialization may pose a limit to these early pruning techniques that restricts performance.

4.3.4. SIGNIFICANCE

While pruning at initialization leads to lower accuracy than magnitude pruning after training, this accuracy is invariant to ablations that hurt the accuracy of magnitude pruning after training. The authors seek to distinguish whether these behaviors are (1) intrinsic to the pruning methods or (2) specific to using the pruning methods at initialization. The authors eliminate (1) by showing that when pruned later in training, Magnitude, SNIP, and SynFlow improve as training progresses, so the accuracy becomes higher than pruning at initialization, and thus they are sensitive to ablations.

In conclusion, the authors show that these methods are insensitive to the specific weights and are specific to using the pruning methods at initialization. These generalized findings may be able to identify subnetworks efficiently. While there are still many challenges ahead, the future step for pruning is to continue to explore the trade-off between the training cost and training accuracy.

5. Conclusion

The Lottey Ticket Hypothesis finds the subnetworks that are comparable to the original network when retrained in isolation. However, winning tickets are harder to find when the size of the model grows larger. (Frankle & Carbin, 2019) proposed that using smaller learning rates or using learning rate warmup might help find winning tickets in large networks. Further studies (Frankle et al., 2020a) show an potential explanation using stability to SGD noise about this behavior, which also provides a starting point for further improvements such as rewinding (Renda et al., 2020), Early-Bird tickets (You et al., 2020) and different pruning heuristics (Frankle et al., 2021).

The Lottery Ticket Hypothesis speeds up the training by finding a good initialization for faster convergence. It challenges the conventional wisdom in neural network training because it shows optimal neural network structures can be learned from the start, therefore increasing the training efficiency. This finding contrasts with the previous work showing that training pruned sparse networks from the beginning often leads to lower accuracy. It also raised interest in the exploration of pruning heuristics, which may further improve the hypothesis and extend to other network discoveries.

References

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks, 2019.

Frankle, J., Dziugaite, G. K., Roy, D. M., and Carbin, M. Linear mode connectivity and the lottery ticket hypothesis, 2020a.

Frankle, J., Schwab, D. J., and Morcos, A. S. The early phase of neural network training, 2020b.

Frankle, J., Dziugaite, G. K., Roy, D. M., and Carbin, M. Pruning neural networks at initialization: Why are we missing the mark?, 2021.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

Renda, A., Frankle, J., and Carbin, M. Comparing rewinding and fine-tuning in neural network pruning. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=S1gSj0NKvB`.

You, H., Li, C., Xu, P., Fu, Y., Wang, Y., Chen, X., Baraniuk, R. G., Wang, Z., and Lin, Y. Drawing early-bird tickets: Towards more efficient training of deep networks, 2020.

Literature Review of Mixture of Experts (Project Proposal)

Anonymous Authors[1]

Abstract

Mixture of Experts (MoE) is an ensemble learning technique that implements the idea of training experts on subtasks of a predictive modeling problem. MoE solves the problems in a divide-and-conquer manner, seperating the problem space between different experts and assigning the problem under the supervision of a gating network. The first part of this paper surveys modeling and mathametical background of MoE. The second part will be focused on MoE's recent advancement in combination with deep neural networks. Finally, a MoE image segmentation model is put forward and test on neurological cell images.

1. Introduction

In 1991, Hinton and Jacob first introduced the idea of Mixture of Experts in their paper *Adaptive mixtures of local experts* (Jacobs et al., 1991). The Mixture of Experts (MoE) model is a type of ensemble learning which uses a combination of simpler learners to improve the prediction. It is usually used as a classification or regression model. The model is constructed with two parts, one is a gating network which use the same input to compute the contribution of each expert. The other part is a set of expert networks. One individual expert node can be a simple linear model or a complicated deep neural network. Later, to improve the MoE performance, Hierarchical Mixture of Experts (HMoE) came out (Jordan & Jacobs, 1994). HMoE is a tree-structured model, which is similar to the decision tree. HMoE has more than one level of experts and one gating network for each subset of expert networks. Each bottom level expert network takes the original data as input, and the upper level experts will take the combination of the expert network output with probability output from the gating network as its input. In 2012, Bishop and Svensen found a better way to do the HMoE which is applying Bayesian treatment to reduce the possibility of overfitting (Bishop & Svensén, 2012). They modeled the parameters in both the gating network and experts network with Gaussian distribution.

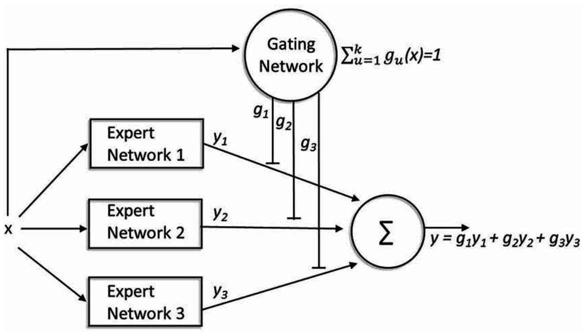


Figure 1. Structure of MoE

1.1. Ensemble Learning

The basic idea of ensemble learning algorithm is constructing a set of ensembles in different ways then using a weighted vote of the ensembles to find the best match of the data (Dietterich et al., 2002). This method works well when there is a representational problem. When each single model alone does not approximate the true function f well, it is possible that the weighted voting of the models can be a more accurate approximation(Dietterich et al., 2002).

Mixture of experts is an ensemble learning approach. It combines a set of simpler models to improve the accuracy of predictions. Some models face an issue that while the data set is too large to fit into the model, it may decrease the accuracy of the prediction. The algorithm behind the mixture of experts model is divide-and-conquer, which breaks a complex task into several simpler and smaller subtasks, and individual experts are trained for different subtasks to keep the accuracy.

This model consists of two parts, the gating network and the expert network. According to the input, the gating networks will compute a possibility for each expert with specific subtasks, then decide which expert should be assigned for a specific subtask. The expert networks are constructed with different types of models.

The goal of the gating and expert networks is helping each other to find the optimization. Each expert should find the best match gating function. Based on the gating function, train each expert to reach the best performance (Avnimelech & Intrator, 1999). With this gating network and expert network structure, a mixture of experts model train all experts at the same time and the current result will decide which dataset will be assigned to the experts in the future, which largely affect the results.

One of the important benefits of the Mixture of Experts model is, in the expert networks, it may contain various types of models, from simple linear models to deep neural networks. For example, Hu did an experiment on using MoE to improve the Electrocardiology (ECG) beat classification result. In this MoE model, they use two classifiers. Two artificial neural networks are used in the classifies, one is Self-Organization Map and Learning Vector Quantization (Hu et al., 1997). These two classifiers are combined by MoE to find a better performance.

Mixture of experts models are also frequently used in multi-modal contexts. Goyal et al. do dynamic motion prediction in movies with MoE. They used two experts in this MoE based fusion model, one is used for audio features and the other is used for video features. Then the gating function decides the contributions of each expert. They found that with MoE applied, the result is better than only using audio/video models or using Late/Early fusion model (Goyal et al., 2016).

The other benefit of MoE is that a big model may be costly, however, when decomposing the big model into several subtasks, it can decrease the cost. In addition, while the model needs some modifications, it is much easier to change it in a small sub model, rather than change one big model which is almost impossible.

2. MoE Architecture

In this section, we will introduce the detail of architecture of MoE and the architecture of Mixture of Gaussian (MoE) as well. MoG has a better cost function to help improve the prediction with mixing proportion.

2.1. Mixture of Experts

In the previous Section we have mentioned that MoE can be separated into two parts. One is individual expert networks, and the other is the gating network. In each network, it has its own parameters. We will use a mixture of N simple linear experts with input data X as an example to show the details.

In expert networks, there are N experts, each expert can compute an output y_i with its own parameter θ_{e_i}. The output of each expert model is

$$y_i = \theta_{e_i} X$$

In the gating network, it has its output p_i and this is the probability of picking the *i*th expert for this subtask. Since we are using the softmax function as the gating function in this example

$$p_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

The output of the MoE model is

$$y = \sum_{i=1}^{n} p_i(X) y_i(X)$$

The cost function we are using in this example is

$$E = \sum p_i (d - y_i)^2$$

where d is the target result we want to find.

When we do the differential with respect to y_i, which is the output of the expert, we get

$$\frac{\partial E}{\partial y_i} = p_i(d - y_i)$$

It means that if the result is small, the parameters inside that expert won't get disturbed by that training case. This parameter will wait to be used when this expert has a higher contribution.

When we do the differential with respect to p_i, which is the output of the gate, we get

$$\frac{\partial E}{\partial p_i} = p_i[(d - y_i)^2 - E]$$

This means if the expert i makes a lower probability than the average, we need to raise its probability. if the expert i makes a higher probability than the average, we need to reduce its probability. This process is called specialization (Jacobs et al., 1991).

2.2. Mixture of Guassian

In the example above, we use softmax as the cost function which is the simplest one. To improve the prediction, Jacob has a better cost function for the MoE which connects Gaussian distribution and maximum likelihood with MoE (Jacobs et al., 1991).

Hinton uses two different Gaussian distribution curve as an example to show his idea. These two Gaussian distribution curves represent each of the experts' (neural network) output. Both experts are trying to find their own predictor y_1 and y_2, so they make a Gaussian prediction around their predictors which is the maximum likelihood of these predictors.

Then the gating network will draw a curve which is the sum of these two Gaussian distribution curves, to decide the contribution of each expert. This process of scaling down Gaussian is called mixing proportion and the total contribution should equal to 1. The next step is maximizing the log probability at the target value. This model is called a mixture of Gaussian model (Jacobs et al., 1991).

The probability of the target is

$$p(d) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(d-y)^2}{2\sigma^2}}$$

. In Gaussian distribution σ should equal to 1, and we want to find the probability of the desired output in case c, so this probability becomes to

$$p(d^c|MoG) = \sum_i p_i^c \frac{1}{\sqrt{2\pi}} e^{-\frac{||d^c - o_i^c||^2}{2}}$$

In the above function, p_i^c is the mixing proportion for expert I in case c. o_i is the output of expert i.

3. Statistical inference of MoE

In this section, an overview of methods used in MoE inference step is provided. In addition, we give the algorithm outline of MoE inference with expectation maximization algorithm under mixture of Gaussian assumption. A simple example using simulated data is attached to delineate the calculation process.

3.1. Overview of MoE inference methods

Before introducing some variations of MoE framework, common approaches for inference of MoE will be introduced. (Jordan & Jacobs, 1994) and (Jacobs et al., 1991) used the expectation-maximization (EM) algorithm to compute the maximum likelihood estimates (MLEs) for MoE models, which has become a classic way of performing MoE inference. (Gormley & Murphy, 2008) deployed very similar expectation minorization maximization (EMM) algorithm in his series of studies on election. Another category of MoE inference methods uses the Bayesisan framework. (Peng et al., 1996), (Gormley & Murphy, 2010) and (Fröhwirth-Schnatter & Kaufmann, 2008) applied Markov chain Monte Carlo (MCMC) (Tanner, 2012) to do estimate the parameters; (Bishop & Svensén, 2012) use variational methods in within Bayesian paradigm to perform inference for a hierarchical mixture of experts model.

3.2. MLE with EM algorithm under mixed Gaussian assumption

The expectation maximization (EM) algorithm (Dempster et al., 1977) provides a suitable tool for maximum likelihood estimation (MLE) calculations in MoE models. The EM algorithm is most effective and most commonly used in situations where data under study has complicated distribution and when optimization of the likelihood could be explicitly expressed if an additional group of variables were known.

The EM algorithm consists of two steps: expectation (E) step followed by a maximization (M) step. The two steps are executed in a iterative manner. Generally, during the E step the conditional expectation of the complete data log likelihood is computed, given the data and current parameter values. In the M step the expected log likelihood is maximized with respect to the model parameters. The imputation of latent variables often makes maximization of the expected log likelihood more feasible. The parameter estimates produced in the M step are then used in a new E step and the cycle continues until convergence or a certain termination condition is reached. The parameter estimates yielded on convergence are the ones that achieve a stationary of the log likelihood function of the data, which is a local maximum, but could be a saddle point.

Consider the basic mixture of experts model based on the following conditional mixture:

$$P(y \mid x, \Theta) = \sum_{j=1}^{K} g_j(x, \nu) P\left(y \mid x, \theta_j\right)$$

$$\mathrm{P}(y \mid x, \theta_j) = \frac{1}{(2\pi)^{n/2}|\Gamma_j|^{1/2}} \exp\left\{-\frac{1}{2}\left[y - f_j\left(x, w_j\right)\right]^T \Gamma_j^{-1}\left[y - f_j\left(x, w_j\right)\right]\right\}$$

where $x \in R^n$, and Θ consists of ν, $\{\theta_j\}_1^K$, and θ_j consists of $\{w_j\}_1^K, \{\Gamma_j\}_1^K$. The vector $f_j\left(x, w_j\right)$ is the output of the j-th expert net. The scalar $g_j(x, \nu), j = 1, \cdots, K$ is given by the softmax function:

$$g_j(x, \nu) = e^{\beta_j(x, \nu)} / \sum_i e^{\beta_i(x, \nu)}$$

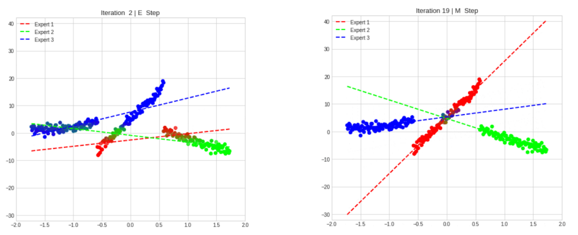
In this equation, $\beta_j(x, \nu), j = 1, \cdots, K$ are the outputs of the gating network. The parameter Θ is estimated by Maximum Likelihood (ML), where the log likelihood is given by $L = \sum_t \ln P\left(y^{(t)} \mid x^{(t)}, \Theta\right)$. The ML estimate can be found iteratively using the EM algorithm as follows. Given the current estimate $\Theta^{(k)}$, each iteration consists of two steps.

(1) E-step. For each pair $\left\{x^{(t)}, y^{(t)}\right\}$, compute $h_j^{(k)}\left(y^{(t)} \mid x^{(t)}\right) = P\left(j \mid x^{(t)}, y^{(t)}\right)$, and then form a set of objective functions:

$$\mathrm{Q}_j^e\left(\theta_j\right) = \sum_t h_j^{(k)}\left(y^{(t)} \mid x^{(t)}\right) \ln P\left(y^{(t)} \mid x^{(t)}, \theta_j\right), j = 1, \cdots, K$$

$$\mathrm{Q}^g(\nu) = \sum_t \sum_j h_j^{(k)}\left(y^{(t)} \mid x^{(t)}\right) \ln g_j^{(k)}\left(x^{(t)}, \nu^{(k)}\right)$$

(2). M-step. Find a new estimate $\Theta^{(k+1)} = \left\{\left\{\theta_j^{(k+1)}\right\}_{j=1}^{K}, \nu^{(k+1)}\right\}$



(a) EM, mixed Gaussian at iteration 2

(b) EM, mixed Gaussian, convergence

Figure 2. Inference of 3 linear experts MoE

with:

$$\theta_j^{(k+1)} = \arg\max_{\theta_j} Q_j^e(\theta_j), j = 1, \cdots, K;$$

$$\nu^{(k+1)} = \arg\max_{\nu} Q^g(\nu).$$

In certain cases where the experts $f_j(x, w_j)$ are linear in the parameter w_j, $\max_{\theta_j} Q_j^e(\theta_j)$ can be solved by setting gradient to 0; for nonlinear experts, the maximization cannot be performed analytically. Moreover, since the softmax gating function is nonlinear, $\max_\nu Q^g(\nu)$ cannot be solved analytically (Xu et al., 1995). A conventional technique used to tackle this problem is using gradient descent.

The complete algorithm starts by (usually randomly) initializing parameters of both expert nets and the gating net, and then perform E step and M step iteratively until convergence or some termination condition is reached.

Figure 2 is from an example of an MoE consisting 3 linear experts and a softmax gating network. The task is to the the model to data points generated along three lines at different input regions with Gaussian noise. With the EM algorithm scheme, this model fits the data well with few parameters and less than 20 iterations.

4. Hierarchical Mixture of Experts

Hierarchical mixture of experts (HMoE) was first introduced by Jacobs and Jordan in 1994. HMoE is a supervised learning algorithm, often considered as a "soft-division" alternative of decision trees. The input data is divided into a nested set of regions and the boundary is "soft" which means some points can lie in different regions at the same time (Jordan & Jacobs, 1994).

HMoE has a tree structure containing at least two levels of the expert networks (Bishop & Tipping, 1998). In our report, we will use a two levels HMoE as the example. The Gating network is assigned to each group of child nodes. Expert network is the leaf node. If we assume this is a binary tree, two experts will be a set under their parent node. The

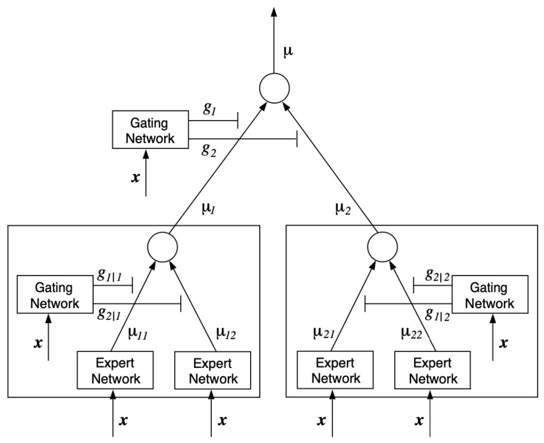


Figure 3. Tree-structured hierarchical mixture of experts.

lower-level expert still takes X as the input and produce a new parameter μ_i, and gating network also takes input X and produce a new parameter g_i

$$\xi_i = v_i^\top X$$

$$g_i = \frac{e^{\xi_i}}{\sum_k e^{\xi_k}}$$

Where the v is a weight vector. The above is the gating network parameter at lower level

$$g_i = \frac{e^{\xi_{ij}}}{\sum_k e^{\xi_{ik}}}$$

This is the parameter at higher level. Both the lower level and higher-level gating network use softmax as an error function.

$$\mu_i = \sum_j g_j | i \mu_{ij}$$

This is the output of the expert at a lower level.

$$\mu = \sum_i g_i \mu_i$$

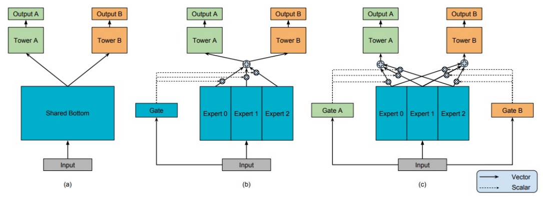
And the output of expert networks at a higher level.

4.1. Hierarchical Mixture of Experts with Expectation-Maximization

HMoE usually works along with EM algorithm to adjust the parameters(Jacobs et al., 1991). After applying EM to HMoE, the probability model, the probability distribution becomes

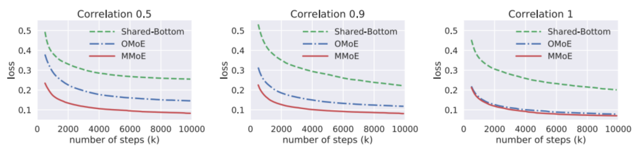
$$P(y^{(t)}, z_{ij}^{(t)} | x^{(t)}, \theta) = g_i(t) g_{j|i}^{(t)} P_{ij}(y^{(t)})$$

Where t is the target value.



(a) Shared-Bottom model. (b) One-gate MoE model. (c) Multi-gate MoE model.

Figure 4. Evolution of recommendation model structures



(a) Performance with correlation 0.5 (b) Performance with correlation 0.9 (c) Performance with two identical tasks

Figure 5. Average performance of MMoe, omoe, and shared- Bottom on synthetic data with different correlations

4.2. Bayesian Hierarchical Mixture of Experts

However, this EM algorithm may cause data overfitting issues. Furthermore, EM does not help with determining the number of experts in each branch because the maximum likelihood prefers a more complex model (Bishop & Svensén, 2012). Bishop introduces a new Bayesian hierarchical mixture of experts to solve the above problems. For each gating network node, a Gaussian prior distribution is defined independently over the parameter v, which as we mentioned before, v is the weight vector.

$$p(v_i|\beta_i) = N(v_i|0, \beta_i^{(-1)}I)$$

For each expert network, do the similar thing to its parameter θ.

$$p(\theta_j|\alpha_j) = \Pi_{k=1}^{t} N(\theta_{jk}|0, \alpha_j^{-1}I)$$

Where t is the dimension of the target value.

5. Deep learning applications of MoE

In this section, we focus the recent advancement of MoE on its ability to massively scale up the number of parameters in deep learning models without introducing too much computation overhead.

5.1. Multi-gate Mixture of Experts (MMoE)

Multi-task learning refers to the machine learning methods that utilizes a model to solve multiple tasks simultaneously. The assumption is that by learning to complete multiple correlated tasks with the same model, that the performance of each task will be higher than if we trained individual models on each task.

Figure 4 illustrates the development process of recommendation models. The shared-bottom model in Figure 4(a) is the simplest multi-task learning architecture, in which the model has a common base, and one single representation is used for different tasks. The second structure, shown in Figure 4(b), uses mixture-of-experts architecture to improve the raw shared-bottom model. However, the model towers in the upper part are still receiving weighted information from the same gate.

The MMoE architecture proposed in (Ma et al., 2018) further modifies the structure by having an individual gating network for each task, rather than a single one for the entire model. This allows the model to learn the weighting of each expert network on task and sample basis, instead of only per-sample weighting. More specifically, the MMoE learns to model the relationships between different tasks. Tasks which have little in common with each other will result in very different weight distribution in the gating networks.

The authors of the MMoE prove the theory by comparing the three architectures on synthetic datasets with varying levels of task correlation, shown in Figure 5. The performance gap between the MoE and MMoE models decreases as task correlation increases. This structure is first deployed in Youtube video recommendation system (Zhao et al., 2019).

5.2. Sparsely gated MoE layer

Based on the MMoE work, researchers found that the mixture of experts model has one other major advantage: it has highly parallelizable structure. The individual experts can be trained independently of one another. The combined ”topped” model assemblage can then be further trained and optimized (Ma et al., 2018). This is leads to further scaling up of the number of experts.

Sparsely gated MoE layer (Shazeer et al., 2017) uses MoE as a replacement layer structure for vanilla feed forward network. This work's baseline contains 2048 experts in each MoE layer, making the training extremely computational-heavy. To relieve this issue, sparsity constraint is introduced, forcing the gating network to only ”activate” a small fraction of the experts.

The gating and selecting function can be written as $y = \sum_{i=1}^{n} G(x)_i E_i(x)$, and when gate output $G(x)_i = 0$, expert E_i is not activated. The sparsity constraint is implemented by modifying the softmax gate.

$$G(x) = Softmax(KeepTopK(H(x), k))$$

Where $H(x)_i = (x \cdot W_g)_i + StandardNormal() \cdot Softplus\left((x \cdot W_{noise})_i\right)$ is a matrix multiplication on x plus an additional noise-modeling term. And function $KeepTopK(v, k)_i$ ranks vector v's elements, keeps k

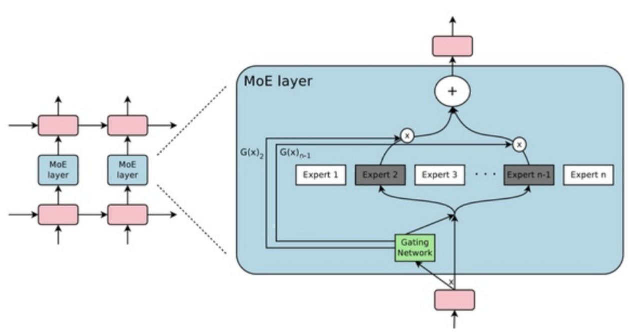


Figure 6. A MoE layers embedded with a language model

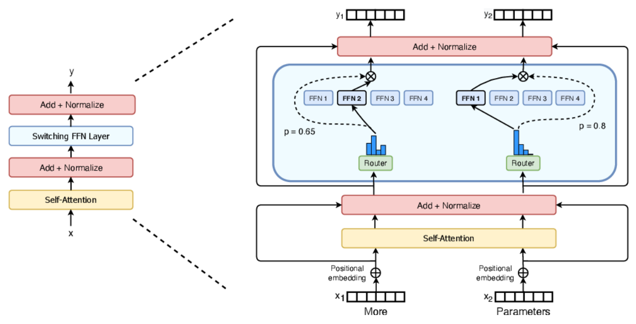


Figure 7. Illustration of a Switch Transformer encoder block

largest elements and set the others to $-\infty$ (making them 0 after softmax).

In Figure 7, the sparse gating function selects two experts to perform computations. This paper puts forward an approach to massively increase the capacity of deep networks by employing conditional computation. Although some implementation details (e.g. batching schemes, load-balancing loss) may be replaced in the future, the authors' main contribution, as they claim, is proving by example that efficient, general-purpose conditional computation with MoE in deep networks is possible and very beneficial. Training instability is resolved by adding a 0.3-0.4 ratio dropout.

5.3. Switch Transformer

Starting from the sparse gate, the Switch Transformer uses a further modified MoE algorithm called Switch Routing: instead of activating multiple experts and combining their output, Switch Routing chooses a single expert to handle a given input. This simplifies the routing computation, and reduces communication costs since individual expert models are hosted on different GPU devices.

The team used Mesh-TensorFlow (MTF) to train the model, taking advantage of data- and model-parallelism. To investigate the performance of the architecture at different scales, the team trained models of different sizes, from 223M parameters up to 1.6T parameters, finding that the ”most efficient dimension for scaling” was the number of experts. Model performance on pre-training and downstream NLP tasks was compared to T5 models requiring similar FLOPs per sample, which is shown in the table.

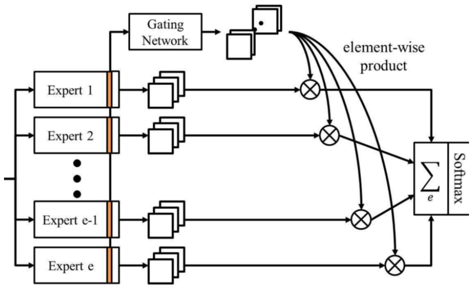


Figure 8. Mixture of image segmentation networks

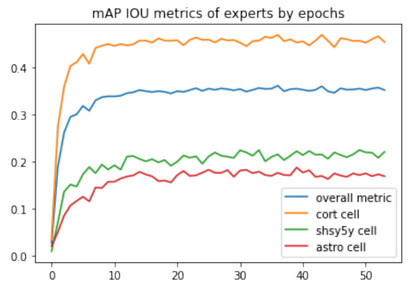


Figure 9. metrics of experts by epoch

6. Experiment: Mixture of image segmentation networks for cell instance segmentation

MoE has also been used in computer vision problems. Previously in cell image segmentation task, Hiramatsu et al. used a mixture of U-Nets and an autoencoder gating network to perform cell semantic segmentation task(Hiramatsu et al., 2018).

Recently, LIVECell, a neurological cell image dataset containing 9 different cell types has been released. Neuron cells are highly variant but has intra-class consistency, making it appropriate to use different experts to segment different cell types. We used similar workflow of (Hiramatsu et al., 2018) but with more complicated instance segmentation experts.

We can see from Figure 9 that the experts (Mask-RCNNs) are able to learn the characteristics of various cell tyes and converge at a similar pace. Quantitative result shows that under mAPIOU metric, MoE achives solid performance gain for different backbone sizes of experts, demonstrate the idea. For example, with ResNext 101 backbone, the mAPIOU of ensembling vs. single goes from 0.286 to 0.297. Our current rank on the Kaggle cell instance segmentation competition is 67/1349(∼5%).

7. Conclusion

In this review, we first introduce the structure of MoE and its inference, and survey the development to Hierachical MoE and Bayesian HMoE. In addition, we provide a review of important recent developments on using MoE as a model up-scaling technique.

The MMoE-based YouTube recommendation algorithm is a good example of modern core-infrastructure machine learning model. It is huge in size, contains an ensemble of different models, does not optimize for a single specific task but instead for a blend of different metrics, and works with sparse data inputs. Google has named this type of production-scale model ”Wide Deep Learning” models (Cheng et al., 2016).

Mixtures of Experts have distinct advantages: they can respond to particular circumstances with greater specialization, allowing the network to display a greater variety of behaviors; experts can receive a mixture of stimuli, integrating data from diverse sensors; and when the network is in operation, only a few experts are active, saving the processing power. As neural networks become more complex, there is an increasing need of integrating data from different modalities, and supplying a greater variety of responses, and Mixture of Expert models will dominate.

References

Avnimelech, R. and Intrator, N. Boosted mixture of experts: An ensemble learning scheme. *Neural computation*, 11 (2):483–497, 1999.

Bishop, C. M. and Svensén, M. Bayesian hierarchical mixtures of experts. *arXiv preprint arXiv:1212.2447*, 2012.

Bishop, C. M. and Tipping, M. E. A hierarchical latent variable model for data visualization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):281–293, 1998.

Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pp. 7–10, 2016.

Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

Dietterich, T. G. et al. Ensemble learning. *The handbook of brain theory and neural networks*, 2(1):110–125, 2002.

Fröhwirth-Schnatter, S. and Kaufmann, S. Model-based clustering of multiple time series. *Journal of Business & Economic Statistics*, 26(1):78–89, 2008.

Gormley, I. C. and Murphy, T. B. Exploring voting blocs within the irish electorate: A mixture modeling approach. *Journal of the American Statistical Association*, 103(483): 1014–1027, 2008.

Gormley, I. C. and Murphy, T. B. Clustering ranked preference data using sociodemographic covariates. In *Choice modelling: the state-of-the-art and the state-of-practice*. Emerald Group Publishing Limited, 2010.

Goyal, A., Kumar, N., Guha, T., and Narayanan, S. S. A multimodal mixture-of-experts model for dynamic emotion prediction in movies. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2822–2826. IEEE, 2016.

Hiramatsu, Y., Hotta, K., Imanishi, A., Matsuda, M., and Terai, K. Cell image segmentation by integrating multiple cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 2205–2211, 2018.

Hu, Y. H., Palreddy, S., and Tompkins, W. J. A patient-adaptable ecg beat classifier using a mixture of experts approach. *IEEE transactions on biomedical engineering*, 44(9):891–900, 1997.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Jordan, M. I. and Jacobs, R. A. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2): 181–214, 1994.

Ma, J., Zhao, Z., Yi, X., Chen, J., Hong, L., and Chi, E. H. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1930–1939, 2018.

Peng, F., Jacobs, R. A., and Tanner, M. A. Bayesian inference in mixtures-of-experts and hierarchical mixtures-of-experts models with an application to speech recognition. *Journal of the American Statistical Association*, 91(435): 953–960, 1996.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Tanner, M. A. *Tools for statistical inference: observed data and data augmentation methods*, volume 67. Springer Science & Business Media, 2012.

Xu, L., Jordan, M. I., and Hinton, G. E. An alternative model for mixtures of experts. *Advances in neural information processing systems*, pp. 633–640, 1995.

Zhao, Z., Hong, L., Wei, L., Chen, J., Nath, A., Andrews, S., Kumthekar, A., Sathiamoorthy, M., Yi, X., and Chi, E. Recommending what video to watch next: a multi-task ranking system. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pp. 43–51, 2019.

On the Use of Momentum in Stochastic Methods

Wu Li [1] **Edward Nguyen** [1]

Abstract

Deterministic momentum methods have been proven to have an exponential rate of convergence when applied to strongly convex and smooth problems (Nesterov, 2014), (Polyak, 2020). This accelerated rate cannot be attained by the standard gradient descent method. However, in many large scale optimization problems, using deterministic optimization methods is infeasible as calculating the full gradient can be computational expensive when the amount of data is very large. Hence, stochastic gradient methods have been employed to reducing the iteration complexity. A natural step would then be to modify deterministic momentum methods to use stochastic gradients resulting in stochastic momentum methods. However, proving that stochastic momentum methods have accelerated convergence rates in comparison to vanilla SGD is not straightforward. In this literature review, we provide the current state-of-the-art convergence rates of stochastic momentum methods for different function classes and also analyze the use of stochastic momentum methods for decentralized optimization and online optimization problems.

1. Overview of the project

Consider the following minimization problem:

$$\min_{\boldsymbol{x}\in\chi} f(\boldsymbol{x}), \tag{1}$$

where $\chi \subseteq \mathbb{R}^n$ is the feasible set. Denote by $\mathcal{S}$ the set of the minimizers, denoted as $\boldsymbol{x}^*$, of $f(\boldsymbol{x})$.

The gradient-based methods provide a class of efficient ways to address (1), attempting to find a minimizer, say $\boldsymbol{x}^*$, of (1) by generating a sequence $\{\boldsymbol{x}_k\}$ via an iterative procedure.

*Equal contribution [1]Rice University, Houston, Texas, USA. Correspondence to: Wu Li <awl@rice.edu>, Edward Nguyen <en18@rice.edu>.

Project report for COMP414/514, Rice CS. Copyright 2021 by the author(s).

They can be unified into:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \Pi_\chi(\boldsymbol{x}_k - \alpha_k \boldsymbol{z}_k) \\ \boldsymbol{z}_{k+1} &= \beta_k \boldsymbol{g}_{k+1} + \gamma_k \frac{\boldsymbol{x}_k - \boldsymbol{x}_{k+1}}{\alpha_k} \end{aligned} \tag{2}$$

with both $\boldsymbol{x}_0$ and $\boldsymbol{z}_0$ given as initial conditions, where all nonnegative constants α_k and β_k are the *step-sizes* and *momentum parameters*, respectively.

In (2), Π_χ denotes the orthogonal projection onto χ, $\boldsymbol{g}_k$ is an estimate or approximate of the (sub-)gradient $\nabla f(\boldsymbol{x})$ of $f(\boldsymbol{x})$ at $\boldsymbol{x} = \boldsymbol{x}_k$ and $\boldsymbol{z}_k$ is the *searching direction* vector. In fact, as to be seen the Nesterov's accelerated gradient (AG) method can also be brought into the umbrella of (2).

We note that with $\chi = \mathbb{R}$ and $\gamma_k = 1 - \beta_k$, (2) yields the standard form of (unconstrained) gradient descent method with momentum (GDM)

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \alpha_k \boldsymbol{z}_k \\ \boldsymbol{z}_{k+1} &= \beta_k \boldsymbol{g}_{k+1} + (1-\beta_k)\boldsymbol{z}_k. \end{aligned} \tag{3}$$

An alternative form of the GDM is the so-called *heavy ball* (HB) methods:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \mu_k \boldsymbol{g}_k + \nu_k(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}), \tag{4}$$

where

$$\mu_k = \alpha_k \beta_{k-1},\ \nu_k = \frac{\alpha_k(1-\beta_{k-1})}{\alpha_{k-1}}. \tag{5}$$

Comment 1.1: We note that with $\gamma_k = 0$ and $\beta_k = 1$, $\boldsymbol{z}_k = \boldsymbol{g}_k$, (3) is the standard *gradient descent* (GD) method. Thus, the GDM is a generalization of GD, which shares almost the same order of complexity as GD but yields a better performance in terms of convergence. This is due to the fact that it uses an improved searching direction vector $\boldsymbol{z}_k$ that takes into account of the gradient $\boldsymbol{g}_k$ as well as the previous estimates $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k-1}$. To visualize the use of momentum in practice, we observe that standard GD often zigzags, but with some momentum from the previous direction, the algorithm becomes "biased" and could possibly take on a more direct trajectory towards the stationary point.

In many applications, especially those related to learning from a set of J data samples, the cost function $f(\boldsymbol{x})$ is the mean of a sequence of functions $\{f_j(\boldsymbol{x})\}_{j=1}^J$. The interest in efficiently solving (1) has grown due to the significant increase in data sets, in which J can be exceedingly large. In this context, *computing the gradient* $\nabla f(\boldsymbol{x})$ *every iteration like what is done for deterministic methods can be costly and sometimes is unrealistic*. The *stochastic gradient methods* have the same form as (2) or (4) but with $\boldsymbol{g}_k$ generated randomly in a very simple way. For example, $\boldsymbol{g}_k$ can be taken as $\boldsymbol{g}_k = \nabla f_{j_k}(\boldsymbol{x}_k)$, where j_k is *randomly* taken from $[J] := \{1, 2, \cdots, j, \cdots, J\}$ with an identically independent distribution (i.i.d.). Clearly, the latter is much easier to compute as it depends only on the j_kth sample. Thus, the corresponding (2) and (4) are referred to as *stochastic gradient* method with momentum and *stochastic heavy ball* (SHB) method. These stochastic approaches have been shown empirically very effective and have become the state-of-the-art. However, analysis explicitly showing the improved performance of stochastic momentum methods compared to vanilla SGD is lacking or nonexistent.

The main objective of this project is to study a class of stochastic GDM-based algorithms by reviewing the three assigned papers (Assran & Rabbat, 2020) - (Mai & Johansson, 2021) and (Liu et al., 2020) that we selected for extra reading and determine current state-of-the-art rates for stochastic momentum based methods for different classes of functions. In addition, we will examine other variants of stochastic momentum methods applied to the decentralized stochastic optimization case (Ying et al., 2021) and the online stochastic optimization case (Yuan et al., 2016).

2. Main results in the related papers

In this section, we will summarize related papers by presenting in order

- the objective function and the algorithm;
- the assumptions;
- the main results.

After the presentation of the individual papers, we will collect and summarize the results into a table containing the convergence rates for different methods in different settings.

2.1. Assran & Rabbat: On Nesterov's Accelerated Gradient Method in Stochastic Settings

This paper (Assran & Rabbat, 2020) deals with Nesterov's accelerated gradient (AG) method with constant step-size and momentum parameters in two settings, namely, the stochastic approximation setting for quadratic functions and the finite sum setting.

2.1.1. $f(\boldsymbol{x})$ and algorithm

The objective function $f(\boldsymbol{x})$ for this paper is given by

$$f(\boldsymbol{x}) := \frac{1}{J}\sum_{j=1}^{J} f_j(\boldsymbol{x}), \tag{6}$$

where the feasible set χ is the entire $\mathbb{R}^n$.

The corresponding minimization problem is addressed with the following algorithm:

$$\begin{aligned} \boldsymbol{y}_k &= \boldsymbol{x}_k + \delta(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}) \\ \boldsymbol{x}_{k+1} &= \boldsymbol{y}_k - \varrho \boldsymbol{g}_k. \end{aligned} \tag{7}$$

where $\boldsymbol{g}_k$, an estimate of $\nabla f(\boldsymbol{y})$, is a *random vector*. In general, (7) is referred to as accelerated stochastic gradient (ASG).

Note that inserting the 1st equation of (7) into the 2nd one yields

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - [\varrho \boldsymbol{g}_k + \delta(\boldsymbol{x}_{k-1} - \boldsymbol{x}_k)] := \boldsymbol{x}_k - \boldsymbol{z}_k,$$

where $\boldsymbol{z}_k = \varrho \boldsymbol{g}_k + \delta(\boldsymbol{x}_{k-1} - \boldsymbol{x}_k)$. Comparing it with (2), where $\chi = \mathbb{R}^n$, we realize that (7) is a special case of (2) with

$$\alpha_k = 1, \quad \beta_k = \varrho, \quad \gamma_k = \delta, \quad \forall\ k. \tag{8}$$

Comment 2.1: Let $f(\boldsymbol{x})$ be $\mathcal{L}$-smooth and μ-strongly convex, denote $Q = \mathcal{L}/\mu$. Some previous results regarding the convergence of (7) include.

- *Deterministic case*:
  1. As shown in (Polyak, 2020), for GD, i.e., $\varrho = 0$, with $\delta = \frac{2}{\mathcal{L}+\mu}$, we have
  $$f(\boldsymbol{x}_k) - f(\boldsymbol{x}^*) \le \frac{\mathcal{L}}{2}(\frac{Q-1}{Q+1})^{2k}||\boldsymbol{x}_0 - \boldsymbol{x}^*||_2^2. \tag{9}$$
  2. As shown in (Nesterov, 2014), for the AG with $\delta = \frac{\sqrt{Q}-1}{\sqrt{Q}+1}$ and $\varrho = \frac{1}{\mathcal{L}}$, we have
  $$f(\boldsymbol{x}_k) - f(\boldsymbol{x}^*) \le \mathcal{L}(\frac{\sqrt{Q}-1}{\sqrt{Q}})^k||\boldsymbol{x}_0 - \boldsymbol{x}^*||_2^2. \tag{10}$$
- *Stochastic case*
  In this case, some of existing works are summarized below.
  1. It was shown in (Yang et al., 2016) that under *bounded gradient* (BG) assumption that the ASG converges with a rate of $O(\frac{1}{\sqrt{k}})$ in the smooth strongly convex setting via a diminishing step-size;
  2. In (Kulunchakov & Mairal, 2019), it was shown that with gradients that are unbiased and *bounded variance* (BV), the ASG converges to a neighborhood in the smooth strongly convex setting with constant step-size and momentum.

2.1.2. **Assumptions**

It is assumed throughout this paper that, for all $j \in [J]$, there exist $\mathcal{L} > 0$ and $\mu > 0$ such that $Q := \frac{\mathcal{L}}{\mu} < 1$ and for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$,

- $\mathbf{A}_{1.1}$ μ-strongly convex:
$$f_j(\boldsymbol{y}) \geq f_j(\boldsymbol{x}) + < \nabla f_j(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} > + \frac{\mu}{2}||\boldsymbol{x} - \boldsymbol{y}||_2^2$$
- $\mathbf{A}_{1.2}$ $\mathcal{L}$-smooth:
$$f_j(\boldsymbol{y}) \leq f_j(\boldsymbol{x}) + < \nabla f_j(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} > + \frac{\mathcal{L}}{2}||\boldsymbol{y} - \boldsymbol{x}||_2^2$$

Clearly, $f(\boldsymbol{x})$ has a unique minimizer $\boldsymbol{x}^*$ under these assumptions.

2.1.3. **The results**

I. Stochastic approximation setting:

In this setting, we have unbiased gradients with bounded variance, that is,

$$\mathbb{E}[\boldsymbol{g}_k] = \nabla f(\boldsymbol{y}_k) \quad - \quad \text{unbiased}$$

and

$$\mathbb{E}[||\boldsymbol{g}_k - \nabla f(\boldsymbol{y}_k)||_2^2] \leq \sigma^2 \quad - \quad \text{bounded variance}, \quad \forall\, k.$$

It is shown that Nesterov's method converges at an accelerated linear rate (i.e. same accelerated rate as in the deterministic setting) to a neighborhood of the optimal solution for smooth strongly-convex quadratic problems. This can be seen in the following corollary and theorem where the corollary assumes a smooth strongly-convex quadratic problem and the theorem is more general and only assume $\mathcal{L}$-smooth and μ-strongly convex functions.

Corollary 1.1 Suppose that $\alpha = \frac{1}{L}$ and $\beta = \frac{\sqrt{Q}-1}{\sqrt{Q}+1}$. Then for and all k

$$\mathbb{E}[f(y_{k+1})] - f^* \leq \frac{L}{2}(\frac{\sqrt{Q}-1}{\sqrt{Q}} + \epsilon_k)^{2k}\|x_0 - x^*\|^2 + C_\epsilon \frac{5Q^2 + 2Q^{3/2} + Q}{2L(2\sqrt{Q}-1)(\sqrt{Q}+1)^2}\sigma^2$$

where $\epsilon_k \sim (\sqrt{k}^k - 1)$ and C_ϵ is some constant that depends on $\epsilon > 0$ and choice of norm.

Theorem 1.1 Let f be $L - smooth$, μ-strongly convex, and twice continuously-differentiable (not necessarily quadratic). Suppose that $\alpha = 2/(\mu + L)$ and $\beta = 0$. Then for all k

$$\mathbb{E}[f(y_{k+1})] - f^* \leq \frac{L}{2}\left(\frac{Q-1}{Q+1}\right)^{2k}\|x_0 - x^*\|^2 + \frac{Q\sigma^2}{2L}$$

The second theorem is for the vanilla SGD case. Clearly, Stochastic Nesterov Accelerated Gradient Method not only converges at an accelerated rate for quadratic functions but also converges to a solution of less error in comparison to SGD.

II. Finite sum setting:

In this setting, we arbitrarily sample a subset of the terms of (6), which is viewed as approximating the gradient with a mini-batch gradient

$$\boldsymbol{g}_k = \sum_{i=1}^{m} \boldsymbol{\omega}_k(i)\nabla f_i(\boldsymbol{x}_k),\ \forall\, k, \tag{11}$$

where $0 < m \leq J$ and $\boldsymbol{\omega}_k \in \mathbb{R}^m$ is a *random* vector with

$$\mathbb{E}[\boldsymbol{\omega}_k(i)] = \frac{1}{m},\ \ \forall\, i, k. \tag{12}$$

The contributions of this paper in this setting include

- It is shown that Nesterov's method may diverge even if all the functions f_i are assumed to quadratic, which implies the significance of the BV assumption that holds in the stochastic approximation setting but not necessarily in the finite sum setting.
- Step-size and momentum parameters to guarantee convergence are proposed, with which, however, acceleration is not guaranteed.

2.2. Sebbouh, Gower & Defazio: On Convergence of Stochastic Heavy Ball

This paper (Sebbouh et al., 2020) intends to provides a comprehensive analysis of the stochastic heavy ball method and deals with iteration dependent step-size and momentum parameters.

2.2.1. $f(\boldsymbol{x})$ **and algorithm**

The objective function $f(\boldsymbol{x})$ considered in the 2nd paper is of the same form as (6), where the feasible set χ is the entire $\mathbb{R}^n$.

As mentioned in Section 1, the stochastic heavy ball (SHB) method is given by

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \mu_k \boldsymbol{g}_k + \nu_k(\boldsymbol{x}_k - \boldsymbol{x}_{k+1}) \tag{13}$$

where $\boldsymbol{g}_k = \nabla f_{\boldsymbol{\omega}_k}(\boldsymbol{x})$ with $f_{\boldsymbol{\omega}_k}(\boldsymbol{x})$ the an averaged version of $\{f_j(\boldsymbol{x})\}$ via the vector $\boldsymbol{\omega}_k \in \mathbb{R}^m$ randomly generated at the kth iteration as described by (11) - (12) for the finite sum setting. It is a special case of the GDM (3) with (μ_k, ν_k) related to (α_k, β_k) via (5)

In this paper, $m = J$ is used, that is

$$\nabla f_{\boldsymbol{\omega}_k}(\boldsymbol{x}) := \sum_{j=1}^{J} \boldsymbol{\omega}_k(j) \nabla f_j(\boldsymbol{x}). \tag{14}$$

Note that $\mathbb{E}[\boldsymbol{\omega}_k(j)] = 1/J, \forall\, j,\ k$. Then,

$$\begin{aligned} \mathbb{E}[\nabla f_{\boldsymbol{\omega}_k}(\boldsymbol{x})] &= \mathbb{E}[\sum_{j=1}^{J} \boldsymbol{\omega}_k(j) \nabla f_j(\boldsymbol{x})] \\ &= \sum_{j=1}^{J} \mathbb{E}[\boldsymbol{\omega}_k(j)] \nabla f_j(\boldsymbol{x}) \\ &= \frac{1}{J} \sum_{j=1}^{J} \nabla f_j(\boldsymbol{x}) = \nabla f(\boldsymbol{x}). \end{aligned} \tag{15}$$

Thus, $\nabla f_{\boldsymbol{\omega}_k}(\boldsymbol{x})$ is a unbiased estimate of $\nabla f(\boldsymbol{x})$.

2.2.2. **Assumptions**

For all $j \in [J]$, there exists $\mathcal{L}_j > 0$ such that for $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, the following holds:

- $\mathbf{A}_{2.1}$ convex

$$f_j(\boldsymbol{y}) \geq f_j(\boldsymbol{x}) + < \nabla f_j(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} >$$

- $\mathbf{A}_{2.2}$ $\mathcal{L}$-smooth

$$f_j(\boldsymbol{y}) \leq f_j(\boldsymbol{x}) + < \nabla f_j(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} > + \frac{\mathcal{L}_j}{2} ||\boldsymbol{y} - \boldsymbol{x}||_2^2$$

Let $\mathcal{S}$ denote the set of the minimizers of $f(\boldsymbol{x})$ by (6). As $f(\boldsymbol{x})$ is convex, any of its stationary points, i.e., those $\boldsymbol{x}$ such that $\nabla f(\boldsymbol{x}) = \mathbf{0}$, is a global minimizer. We have the following important results due to (Gower et al., 2018):

Lemma 1 *Let $\mathcal{S}$ be the solution set of (1) with $f(\boldsymbol{x})$ given by (6) and $\boldsymbol{\omega}$ be an averaging random vector defined above. Then, under the assumptions $\mathbf{A}_{2.1}$ and $\mathbf{A}_{2.2}$ we have*

- *There exists a $\mathcal{L} > 0$ such that*

$$\mathbb{E}[||\nabla f_{\boldsymbol{\omega}}(\boldsymbol{x}) - \nabla f_{\boldsymbol{\omega}}(\boldsymbol{x}^*)||_2^2] \leq 2\mathcal{L}[f(\boldsymbol{x}) - f(\boldsymbol{x}^*)] \quad \forall\, \boldsymbol{x}^* \in \mathcal{S}$$

and with the residual gradient noise

$$\bar{\sigma}^2 := \max_{\boldsymbol{x}^* \in \mathcal{S}} \mathbb{E}[||\nabla f_{\boldsymbol{\omega}}(\boldsymbol{x}^*)||_2^2], \tag{16}$$

we have

$$\mathbb{E}[||\nabla f_{\boldsymbol{\omega}}(\boldsymbol{x})||_2^2] \leq 4\mathcal{L}[f(\boldsymbol{x}) - f(\boldsymbol{x}^*)] + 2\bar{\sigma}^2. \tag{17}$$

2.2.3. **The results**

The key point of the analysis in this paper is to examine the SHB method from an iterative averaging viewpoint. More specifically, the convergence of (13) is analyzed via the following iterative-moving average (IMA) method (Taylor & Bach, 2020):

$$\begin{aligned} \boldsymbol{y}_k &= \boldsymbol{y}_{k-1} - \phi_k \nabla f_{\boldsymbol{\omega}_k}(\boldsymbol{x}_k) \\ \boldsymbol{x}_{k+1} &= \frac{\varphi_{k+1}}{\varphi_{k+1}+1} \boldsymbol{x}_k + \frac{1}{\varphi_{k+1}+1} \boldsymbol{y}_k \end{aligned} \tag{18}$$

with $\boldsymbol{x}_0 = \boldsymbol{z}_0$.

The authors showed that this IMA formulation is in fact equivalent to the original SHB and provided analysis based on this equivalent IMA formulation. The proof of equivalence is provided in the *supplementary material*, which states that the sequence $\{\boldsymbol{x}_k\}$ generated with (18) is exactly the same as the one generated with (13) when we set

$$\mu_k = \frac{\phi_k}{\varphi_{k+1}+1}, \quad \nu_k = \frac{\varphi_k}{\varphi_{k+1}+1}, \ \forall\, k. \tag{19}$$

Based on the above, the paper mainly provided two important results as two corollaries from a generalized theorem. The two results are

- Assuming $\mathbf{A}_{2.1}$ and $\mathbf{A}_{2.2}$, this paper is able to prove that the function values at the last iterate of SHM converges almost surely at a rate close to $o(1/\sqrt{k})$ which can be improved to $o(1/k)$ in the overparameterized case. Note that this does not require the additional assumptions of bounded gradients and bounded noise. The overparameterized case happens when $\sigma^2 = 0$
- With the proposed scheme of decreasing step sizes, exact convergence to the minimum, as opposed to convergence to a neighborhood around the minimum, is guaranteed at a rate of $o(\frac{1}{\sum_{t=0}^{k-1} \eta_t})$ where η_t is a member of the sequence of step-sizes. Note that this only requires assumptions $\mathbf{A}_{2.1}$ and $\mathbf{A}_{2.2}$. This means that in the general stochastic case, SHB has the same almost sure convergence rate as SGD with averaging (instead of the last iterate).

2.3. Mai & Johansson: On Stochastic Gradient Descent for Non-smooth and Non-convex Objectives

This paper (Mai & Johansson, 2021) deals with stochastic gradient methods with momentum and establishes the convergence rate for a class of ρ-weakly convex and constrained optimization problems.

2.3.1. $f(\boldsymbol{x})$ **and algorithm**

In this paper, the objective function $f(\boldsymbol{x})$ is defined as[1]

$$f(\boldsymbol{x}) := \mathbb{E}[f(\boldsymbol{x}, S)] = \int_{s \in \Psi} f(\boldsymbol{x}, s) \mathrm{d}F_S(s), \;\; \boldsymbol{x} \in \chi \quad (20)$$

where χ is a closed convex set, S is a random variable with a CDF $F_S(s)$ defined on the set Ψ. Furthermore, such a $f(\boldsymbol{x})$ is assumed to belong to the class of ρ-weakly convex function *but* defined over some closed and convex χ that may be different from $\mathbb{R}^n$ - this setting is more complicated than in the previous two papers analyzed.

The algorithm proposed to solve (1) with $f(\boldsymbol{x})$ given by (20) is [2]

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \Pi_\chi(\boldsymbol{x}_k - \alpha \boldsymbol{z}_k) \\ \boldsymbol{z}_{k+1} &= \beta \boldsymbol{g}_{k+1} + (1-\beta)\frac{\boldsymbol{x}_k - \boldsymbol{x}_{k+1}}{\alpha}, \end{aligned} \quad (21)$$

where $\beta \in (0, 1]$ and $\boldsymbol{g}_{k+1} = \partial f(\boldsymbol{x}_{k+1}, S_{k+1})$ with S_k the kth sample of S. Moreover, we note that (21) is a special case of (2) with $\alpha_k = \alpha$, $\beta_k = \beta, \gamma_k = 1-\beta$, $\forall\, k$.

2.3.2. **Assumptions**

Denote subdifferentials/subgradients $\partial f(\boldsymbol{x}, S)$ and $\partial f(\boldsymbol{x})$. The convergence is analyzed in the this paper under the following *assumptions*:

- $\mathbf{A}_{3.1}$ $\quad \mathbb{E}[\boldsymbol{g}_k] \in \partial f(\boldsymbol{x}_k), \;\; \forall\, k$;
- $\mathbf{A}_{3.2}$ Bounded Gradient

$$\exists\; G > 0 \;\; \text{s.t.} \quad \mathbb{E}[||\boldsymbol{g}_k||_2^2] \le G, \forall\;\; \boldsymbol{x}_k \in \chi;$$

- $\mathbf{A}_{3.3}$ Bounded Variance

$$\mathbb{E}[||\boldsymbol{g}_k - \nabla f(\boldsymbol{x}_k)||_2^2] \le \sigma^2, \;\; \forall\, k.$$

2.3.3. **The results**

I. Weakly-convex functions

Assuming $\mathbf{A}_{3.1}$ and $\mathbf{A}_{3.2}$, which is standard in analysis of stochastic optimization of non-smooth functions, it is shown that SHB for minimization of weakly convex functions has a convergence rate of $O(\frac{1}{\sqrt{K}})$, or equivalently, a sample complexity of $O(\frac{1}{\epsilon^2})$. According to the authors, this is the first complexity guarantee, obtained in a parameter-free and single time-scale fashion, for a stochastic method with momentum on non-smooth and non-convex problems.

II. Extension to smooth non-convex functions

- In the unconstrained case, under assumptions $\mathbf{A}_{3.1}$, $\mathbf{A}_{3.3}$ and that the objective function has a bounded gradient, a similar complexity of $O(\frac{1}{\epsilon^2})$ was obtained without needing to form a batch of samples at each iteration.
- In the constrained case, one only needs assumptions $\mathbf{A}_{3.1}$ and $\mathbf{A}_{3.3}$ only (without the bounded gradient assumption) to show a complexity of $O(\frac{1}{\epsilon^2})$.

[1]In practice, $f(\boldsymbol{x})$ is given by averaging a set of samples $f(\boldsymbol{x}, s_j)$: $f(\boldsymbol{x}) = \frac{1}{J}\sum_{j=1}^{J} f(\boldsymbol{x}, s_j)$.

[2]Note $\boldsymbol{x}_{k+1} = \arg\min_{\boldsymbol{x} \in \chi}\{< \boldsymbol{z}_k, \boldsymbol{x} - \boldsymbol{x}_k > + \frac{1}{\alpha}||\boldsymbol{x} - \boldsymbol{x}_k||_2^2\}$. When $\chi = \mathbb{R}^n$, $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \boldsymbol{z}_k$.

2.4. Liu, Gao & Yin: On Stochastic Gradient Descent with Momentum

This paper (Liu et al., 2020) provides new convergence analysis for *stochastic gradient descent with momentum* (SGDM) and *Multistage* SGDM.

2.4.1. $f(\boldsymbol{x})$ **and algorithm**

This paper mainly deals with minimizing the function of the same form as (6) using the standard SGDM (3), i.e.,

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \alpha_k \boldsymbol{z}_k \\ \boldsymbol{z}_{k+1} &= \beta_k \boldsymbol{g}_{k+1} + (1-\beta_k)\boldsymbol{z}_k \end{aligned} \quad (22)$$

with both $\boldsymbol{x}_0$ and $\boldsymbol{z}_0$ given as initial conditions, where $\alpha_k > 0$ is the step-size and $\beta_k \in (0, 1]$, as mentioned before, is called *momentum parameter/weight*, while $\boldsymbol{z}_k$ is the *search direction* vector, and $\boldsymbol{g}_k$ is the stochastic (sub-)gradient $\partial f(\boldsymbol{x})$ of $f(\boldsymbol{x})$ at $\boldsymbol{x} = \boldsymbol{x}_k$.

Traditionally, the parameters α_k and β_k are constant, independent of k. Nowdays, SGDM used in deep learning is a scheme, which often comes with parameter tuning rules. Essentially, the multistage SGDM requires that a constant stepsize be applied for a long period before being dropped by some constant factor, while the momentum weight is either kept unchanged or gradually increasing.

2.4.2. **Assumptions**

The following assumptions are made effective throughout the analysis, which are standard in stochastic optimization studies:

- $\mathbf{A}_{4.1}$ - unbiasedness:

$$\mathbb{E}[\boldsymbol{g}_k] \in \partial f(\boldsymbol{x}_k), \;\; \forall\, k$$

- $\mathbf{A}_{4.2}$ - independence of the samples used for computing the stochastic gradients $\{\boldsymbol{g}_k\}$ are independent.
- $\mathbf{A}_{4.3}$ - bounded variance: for some $\sigma^2 > 0$

$$\mathbb{E}[||\boldsymbol{g}_k - \nabla f(\boldsymbol{x}_k)||_2^2] \le \sigma^2, \;\; \nabla f(\boldsymbol{x}_k) \in \partial f(\boldsymbol{x}_k), \forall\, k.$$

- $\mathbf{A}_{4.4}$ - $\mathcal{L}$-smooth and differentiable with $\mathcal{L} \ge 0$:

$$f(\boldsymbol{y}) \le f(\boldsymbol{x}) + < \nabla f(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} > + \frac{\mathcal{L}}{2}||\boldsymbol{y} - \boldsymbol{x}||_2^2,$$

for all $\boldsymbol{x}, \boldsymbol{y}$.

2.4.3. **The results**

First, it is worth noting that, the analysis of the convergence of the standard SGDM is motivated by a new observation that the update direction has a controllable deviation from the current full gradient and possesses a smaller variance. Exploiting this observation, a new Lyapunov function is constructed to take advantage of the reduced variance, and the convergence results are therefore obtained.

We now list the results of this paper.

- For both strongly convex and nonconvex objectives, SGDM has the same convergence bound as SGD. This convergence bound is an improvement from previously existing bounds, and the analysis for it applies not only to least squares and does not assume uniformly bounded gradient.
- The first known convergence guarantee for the multi-stage is obtained, and it is also shown that the multi-stage setting is more beneficial than using fixed parameters.

More specifically,

- For SGDM with the 4 assumptions,
$$\mathbb{E}[||\nabla f(\boldsymbol{x}_k^{out})||^2] \leq O(\frac{1}{k\alpha} + \alpha\sigma^2);$$
- For SGDM with the 4 assumptions plus strongly convexity of $f(\boldsymbol{x})$,
$$\mathbb{E}[f(\boldsymbol{x}_k) - f(\boldsymbol{x}^*)] \leq O((1-\alpha\mu)^k + \alpha\sigma^2);$$
- For multistage SGDM the 4 assumptions,
$$\mathbb{E}[||\nabla f(\boldsymbol{x}_k^{out})||^2] \leq O(\frac{1}{nA_2} + \frac{1}{n}\sum_{l=1}^{n}\alpha_l\sigma^2),$$
where n is the number of stages, α_l is the step-size of the lth stage, and $A_2 = \alpha_l T_l$ is constant, where T_l is the length of the lth stage, for all $1 \leq l \leq n$, and hence $k = \sum_{l=1}^{n} T_l$.

2.5. Comparison of the momentum-based methods

We now summarize the results obtained in the four papers studied in this project in terms of convergence analysis. The methods involved are SGDM , SAG and SHB, specified in (22), (7), and (13), respectively, and the multi-stage SGDM. The first three methods are very easy to implement, while the pseudocode of multi-stage SGDM can be found in (Liu et al., 2020).

The bounds of convergence of the four methods under different assumptions are given in the table below.

Note that in the table η, γ and ϵ_k are all small positive constants and in the bounds of SGDM, derived in (Mai & Johansson, 2021), $\bar{k}$ is a (integer) random variable, *uniformly distributed* within $[0, k]$ and $F_\lambda(\boldsymbol{x})$ is the *Moreau envelope* of $F(\boldsymbol{x})$ is defined as $F(\boldsymbol{x}) := f(\boldsymbol{x}) + \mathrm{I}_\chi$ is used in the analysis, where I_χ is the *indicator function* of χ. Another important note is that an accelerated rate for stochastic momentum methods is actually only proven for quadratic function when using SAG. Besides that, all other rates for the various variants of stochastic momentum methods for different classes of functions can only be proven to be comparable to SGD, not accelerated.

3. Extension to Online & Decentralized Settings

In this section, we will briefly discuss the application of stochastic momentum methods in *Decentralized* and *Online settings*.

3.1. Ying, Yuan, Chen, Hu, Pan & Yin Paper: Stochastic Momentum Methods in Decentralized Setting

3.1.1. Problem Formulation

This paper (Ying et al., 2021) discusses the use of a variant of stochastic momentum methods called Decentralized Momentum Stochastic Gradient Descent (DmSGD) to solve decentralized optimization problems for two specific networks. In decentralized optimization, a set of n agents seek to collaborate to solve an optimization problem. Agents have a local objective function that are all of the same form that is modeled based on their local data. An additional restriction is that agents are not permitted to share their data and are only permitted to communicate their parameters with their neighbors. After sufficient iterations of a decentralized algorithm, agents are expected to reach a "consensus" which means they all share the same parameter. This decentralized consensus optimization problem can be formally expressed as

$$\min_{x\in\mathbb{R}^d} f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$$

where

$$f_i(x) := \mathbb{E}_{\xi_i\sim D_i} F(x;\xi_i)$$

Function $f_i(x)$ is local to agent i and random variable ξ_i denotes the local data following distribution D_i. These n agents will determine a common decision variable $x \in \mathbb{R}^d$. $f_i : \mathbb{R}^d \to \mathbb{R}$ is local and private to agent i. There are some assumptions on the loss function that are mentioned below.

- $\mathbf{A}_{5.1}$ - smoothness: Each $f_i(x)$ is L-smooth, i.e., $\|\nabla f_i(x) - \nabla f_i(y)\| \leq L\|x - y\|$ for any $x, y \in \mathbb{R}^d$.

Table 1. Comparison of convergence behavior of different methods

Method	Assumptions	Convergence bound
SGDM	$\mathbf{A}_{4.1}$-$\mathbf{A}_{4.4}$	$\mathbb{E}[\|\|\nabla f(\boldsymbol{x}_k^{out})\|\|^2] \le O(\frac{1}{k\alpha} + \alpha\sigma^2)$
SGDM	$\mathbf{A}_{4.1}$-$\mathbf{A}_{4.4}$ plus μ-strongly convexity	$\mathbb{E}[f(\boldsymbol{x}_k) - f(\boldsymbol{x}^*)] \le O((1-\alpha\mu)^k + \alpha\sigma^2)$
Multi-SGDM	$\mathbf{A}_{4.1}$-$\mathbf{A}_{4.4}$	$\mathbb{E}[\|\|\nabla f(\boldsymbol{x}_k^{out})\|\|^2] \le O(\frac{1}{nA_2} + \frac{1}{n}\sum_{l=1}^n \alpha_l\sigma^2)$
SAG	Quadratic Functions	$\mathbb{E}[f(\boldsymbol{x}_k) - f(\boldsymbol{x}^*)] \le O((\frac{\sqrt{Q}-1}{\sqrt{Q}} + \epsilon_k)^{2k} + \eta\sigma^2)$
SHB	$\mathbf{A}_{4.4}$ plus convexity	$\mathbb{E}[f(\boldsymbol{x}_k) - f(\boldsymbol{x}^*)] \le o(\frac{1}{\sqrt{k}})$
SGDM	$\mathbf{A}_{4.1}$-$\mathbf{A}_{4.3}$ plus $\mathbf{A}_{3.2}$ & weakly convex	$\mathbb{E}[\|\|\nabla F_\lambda(\bar{\boldsymbol{x}}_{\tilde{k}})\|\|_2^2] \le O(\frac{1}{\sqrt{k+1}})$
SGDM	$\mathbf{A}_{4.1}$-$\mathbf{A}_{4.3}$ plus $\mathbf{A}_{3.2}$ & smooth non-convex	$\mathbb{E}[\|\|\nabla F_\lambda(\bar{\boldsymbol{x}}_{\tilde{k}})\|\|_2^2] \le O(\frac{1}{\sqrt{k+1}})$

- $\mathbf{A}_{5.2}$ - data heterogeneity: It holds that $\frac{1}{n}\sum_{i=1}^n \|\nabla f_i(x) - \nabla f(x)\|^2 \le b^2$ for any $x \in \mathbb{R}^d$.

Another important note is that the local objective functions $f_i(x)$ are allowed to be non-convex. The data heterogeneity assumption is important because data from different local agents can come from varying distribution. This can affect the convergence of decentralized algorithms.

Each agent i is able to calculate a noisy sample $g_i = \nabla F(x_i; \xi_i)$ of the true gradient at each iteration. There is an additional assumption made on the stochastic gradients.

- $\mathbf{A}_{5.3}$ - gradient noise: The random sample ξ_i^k is independent of each other for any k and i. We also assume $\mathbb{E}[\nabla F(x;\xi_i)] = \nabla f_i(x)$ and $\mathbb{E}\|\nabla F(x;\xi_i^k) - \nabla f_i\|^2 \le \sigma^2$.

3.1.2. Network Model

In general decentralized optimization problems, all agents are assumed to be connected according to a directed or undirected network topology. w_{ij} is the weight scaling information flowing from agent j to agent i that is defined formally as

$$w_{ij} = \begin{cases} > 0 & \text{if agent } j \text{ is connected to } i, \text{ or } i = j; \\ 0 & \text{otherwise} \end{cases}$$

$\mathcal{N}_i = \{j | w_{ij} > 0\}$ is defined as the set of neighbors of agent i which also includes agent i itself. The weight matrix $W := [w_{ij}]_{i,j=1}^n \in \mathbb{R}^{n\times n}$ is a matrix that stacks the weights of all agents. The weight matrix is also allowed to be time-varying. Another assumption is made related to the network topology.

- $\mathbf{A}_{5.4}$ - weight matrix: The weight matrix W^k is doubly-stochastic, i.e. $W^k 1 = 1$ and $1^T W^k = 1^T$. If $W^k \equiv W$, we assume $\rho(W) := \max_{\lambda_i(W)\neq 1}\{\|\lambda_i(W)\|\} \in (0,1)$ where $\lambda_i(W)$ is the i-th eigenvalue of the matrix W.

Note that $1 - \rho(W)$ is called the spectral gap and it characterizes the connectivity of the graph. $1 - \rho(W) \to 1$ means the graph is very well connected while $1 - \rho(w) \to 0$ means the graph is more sparse.

This specific work considers the specific decentralized case in which the topology can be controlled at every iteration, there is no agent failure, and each agent has similar computational ability. While this setting may seem unrealistic, it is actually a realistic setting that can be seen in high power computing. Individual GPUs or servers that hold a stack of GPUs can be abstracted as agents of a graph. The connections between individual GPUs or various servers can be changed with relatively little cost. In addition, the connection between these "agents" are high bandwidth and are tolerant to failure. Abstracting high computing resources in this manner has proven fruitful in the accelerated training of deep neural networks. According to this formulation, each agent will have the same deep neural network model but the training data set is split among multiple agents to be processed in parallel. Hence, this situation is adept in training deep neural networks using extremely large data sets.

A natural question that stems from this scenario is what is the best topology or series of topologies to use for the decentralized algorithm. If the topology is too sparse, then communication cost can be saved but this can come at the cost of additional iterations of the decentralized algorithms to reach a consensus or converge to a good solution. If the topology is too connected, there could be an extremely large communication cost that slows the overall wall clock speed of the decentralized algorithm but fewer iterations are needed to reach a consensus or converge to a good solution. Seeing this tradeoff, the paper focuses on a specific type of graph called the exponential graph.

The exponential graph is defined as a graph where each agent is assigned an index from 0 to $n-1$ and will have a one way directed communication to neighbors that are $2^0, 2^1, \cdots, 2^{\lfloor \log_2(n-1)\rfloor}$ hops away. With maximum degree $\lceil \log_2(n)\rceil$ neighbors, partial averaging over the static expo-

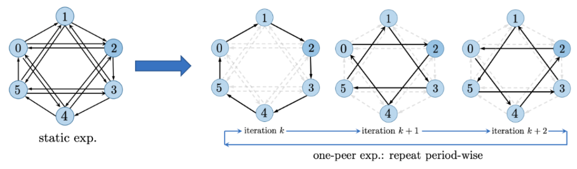


Figure 1. An example of the static exponential graph and the one-peer exponential graph for 6 agents.

nential graph requires $\Omega(\log_2(n))$ communication time per iteration. The weight matrix of the static exponential graph is defined as follows:

$$w_{ij}^{\exp} = \begin{cases} \frac{1}{\lceil \log_2(n) \rceil + 1} & \text{if } \log_2(mod(j-i,n)) \\ & \text{is an integer or } i = j \\ 0 & \text{otherwise} \end{cases} \tag{23}$$

The paper also provides a proof of this particular graph's spectral gap. This is formally stated in the following proposition.

Proposition 1. (Spectral Gap of Static Expo) The spectral gap for the static exponential graph defined by the above weight matrix is defined as

$$1 - \rho(W^{\exp}) \begin{cases} = \frac{2}{1+\lceil \log_2(n) \rceil}, & \text{when } n \text{ is even number} \\ < \frac{2}{1+\lceil \log_2(n) \rceil}, & \text{when } n \text{ is odd number} \end{cases} \tag{24}$$

In addition, we have $\|W^{exp} - \frac{1}{n}11^T\|_2 = \rho(W^{\exp})$ for exponential graphs.

Because the exponential graph incurs $\Omega(\log_2(n))$ communication overhead per iteration, the paper also covers the case where the static exponential graph is decomposed into a sequence of one-peer graphs. Each part of this sequence has each agent cycle through all its neighbors resulting in each agent communicating only to a single neighbor per iteration. These one-peer realizations have $\Omega(1)$ communication costs which matches with a ring or grid topology. The time-varying weight matrix at iteration k describing this sequence can be formally described as follows where $\tau = \lceil \log_2(n) \rceil$

$$w_{ij}^k = \begin{cases} \frac{1}{2} & \text{if } \log_2(mod(j-i),n)) = mod(k,\tau) \\ \frac{1}{2} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

Examples of the static exponential graph and the one-peer exponential graph can be seen in figure 1.

This paper then attempts to establish that applying this sequence for a long enough duration can allow agents to reach a consensus. This is formally established in the following lemma:

Lemma 1 (Periodic Exact Averaging) Suppose $\tau = \log_2(n)$ is a positive integer. If W^k is the weight matrix defined as above over the one-peer exponential graphs, it then holds that each W^k is doubly-stochastic. Furthermore, it holds that

$$W^{k+l}W^{k+l-1}\cdots W^{k+1}W^k = \frac{1}{n}11^T \tag{26}$$

for any integer $k \geq 0$ and $l \geq \tau$. Also, the consensus residue form holds that

$$(W^{k+1} - \frac{1}{n}11^T)(W^{k+1-1} - \frac{1}{n}11^T)\cdots(W^k - \frac{1}{n}11^T) = 0$$

3.1.3. ALGORITHM

The decentralized algorithm analyzed in this particular paper is Decentralized Momentum Stochastic Gradient Descent (DmSGD). The algorithm is as follows:

Algorithm 1 DmSGD

```
1: Input: $\gamma, x_i^0, m_i^0 = 0, \beta \in (0,1)$
2: for $k = 0, 1, ..., T-1$, every agent $i$ do
3:     Sample weight matrix $W^k$
4:     Update gradient $g_i^k = \nabla F(x_i^k; \xi_i^k)$
5:     $m_i^{k+1} = \sum_{j \in \mathcal{N}_i} w_{ij}^k(\beta m_j^k + g_j^k)$
6:     $x_i^{k+1} = \sum_{j \in \mathcal{N}_i} w_{ij}^k(x_j^k - \gamma m_j^k)$
7: end for
```

The intuition behind this algorithm is quite simple. First, the network topology for the current iteration is determined. Then, each agent samples from its local data distribution to calculate the stochastic gradient for the current iteration. Afterwards, this stochastic gradient is used to update the local momentum parameter. Agents communicate this new local momentum parameter with their neighbors and perform partial averaging with the momentum parameters they receive from their neighbors with their own local momentum parameter. This new partially averaged momentum parameter is used to update the local model parameters. Finally, agents communicate their newly determined local model parameters with their neighbors and perform partial averaging with the model parameters they receive from their neighbors with their own local model parameters. This process is repeated for a specified number of iterations.

Note that a key difference between traditional optimization algorithms and decentralized consensus optimization algorithms is that a stopping criteria cannot be easily established. This is because evaluating the quality of the solution depends on all the local parameters from all agents. However, aggregating all the local model parameters from all agents in one location is not always feasible due to the network structure. This means determining whether consensus has been achieved or evaluating the quality of the solution is generally difficult. Hence, the normal stopping criteria is the number of iterations.

3.1.4. The results

The paper seeks to establish a relationship between the convergence of DmSGD for the static exponential graph and the one-peer exponential graph. This is done in the following corollary and theorem.

Corollary 5.1. Under Assumptions 5.1-5.4 and for the static exponential graph, if $\gamma = \frac{\sqrt{n(1-\beta)^3}}{\sqrt{T}}$, DmSGD will converge at

$$\frac{1}{T}\sum_{k=1}^{T}\mathbb{E}\|\nabla f(\bar{x}^k)\|^2 = O\Bigg(\frac{\sigma^2}{\sqrt{(1-\beta)nT}} + \frac{n\log_2(n)(1-\beta)\sigma^2}{T} + \frac{n(1-\beta)b^2\log_2^2(n)}{T}\Bigg) \tag{27}$$

In addition, the transient iteration complexity of DmSGD over static exponential graphs is $O(n^3\log_2^2(n))$ for the data-homogeneous scenario and $O(n^3\log_2^4(n))$ for data-heterogeneous scenario.

For DmSGD with one-peer exponential graph, the following theorem is found.

Theorem 5.1. We assume $\tau = \log_2(n)$ is a positive integer, and the time-varying weight matrix is generated as according to the previous definition of the time-varying weight matrix for the one-peer exponential graph. Under Assumptions 5.1-5.4 and $\gamma = \frac{\sqrt{n(1-\beta)^3}}{\sqrt{T}}$, DmSGD will converge at

$$\frac{1}{T}\sum_{k=1}^{T}\mathbb{E}\|\nabla f(\bar{x}^k)\|^2 = O\Bigg(\frac{\sigma^2}{\sqrt{(1-\beta)nT}} + \frac{n(1-\beta)\sigma^2\tau}{T} + \frac{n(1-\beta)b^2\tau^2}{T}\Bigg) \tag{28}$$

In addition, the transient iteration complexity of DmSGD for one-peer exponential graph is $O(n^3\log_2^2(n))$ for the data-homogeneous scenario and $O(n^3\log_2^4(n))$ for data-heterogeneous scenario.

Looking at the rates determined by Corollary 1 and Theorem 1, it is apparent that DmSGD run for the static exponential graph and for one-peer exponential graph converges exactly as fast in terms of the established rate bounds. Also, both graphs will have DmSGD have the same transient iteration complexity. The transient iteration complexity is the number of iterations such that the first term of both rates given in Corollary 1 and Theorem 1 will dominate. When this first term dominates, this is called linear speedup. This can be understood as follows. The domination of the first term results in there needing to be $T = \Omega(1/(n\epsilon^2))$ iterations to reach a desired accuracy ϵ. Hence, there is an inverse relationship between the desired accuracy ϵ and n which means that the number of agents participating linearly reduces the overall number of iterations required to converge to a solution of accuracy ϵ.

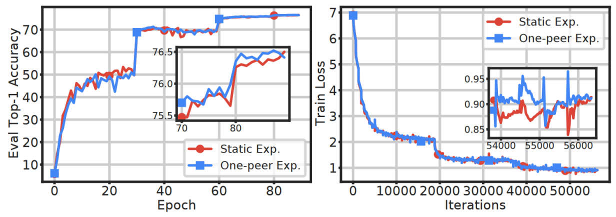


Figure 2. Convergence curves for both Top-1 Accuracy and Training Loss recorded during the training of ImageNet. The blue curve is for the one-peer exponential and the red curve is for the static exponential. The network was formed using 8 servers each with 8 GPUs.

One of the experiments performed in the paper to evaluate the theoretical claims is the training of ImageNet using 8 servers which are then abstracted as agents. Each of these 8 servers have 8 GPUs. The training is done on both the static exponential graph and the one-peer exponential graph. Results of this experiment can be seen in the figure 2.

From the extremely close similarity between the curves and the theory, it is clear that the use of one-peer exponential graphs may be more beneficial in comparison to the static exponential graph because they converge exactly as fast in terms of establish rate bounds but the one-peer exponential graph requires less communication per iteration.

3.2. Yuan, Ying, & Sayed: Stochastic Momentum Methods in Online Setting

3.2.1. Problem Formulation

This paper (Yuan et al., 2016) discusses the online case of stochastic optimization and the use to stochastic momentum methods to solve such a problem. Under these circumstances, data is constantly streaming in and the new data can change the overall data distribution. Another typical added condition is that older data that is processed is deleted or disallowed from being accessed because of memory constraints.

Formally, stochastic optimization problems can be formulated as

$$\min_{x\in\mathbb{R}^d} f(x) := \mathbb{E}_{\xi\sim D}[F(x;\xi)]$$

ξ is a random variable that denotes the data drawn from the distribution D. For stochastic optimization problems, the data distribution D is unknown and we have data samples denoted as ξ_i where i is the sample index available to estimate the data distribution. There are several assumptions made on this model. Note that gradient noise is defined by $s_i(x_{i-1}) := \nabla_x F(x_{i-1}, \xi_i) - \nabla_x \mathbb{E}[F(x_{i-1}; \xi_i)]$ and $\mathcal{F}_{i-1} := \{x_{-1}, x_0, x_1, ..., x_{i-1}\}$ which is the filtration (past history) generated by the random process x_j for $j \leq i-1$.

- $\mathbf{A}_{6.1}$ - μ-strongly convex and L-smooth: $f(x)$ is twice differentiable and its Hessian matrix satisfies $0 < \mu I_d \leq \nabla^2 f(x) \leq L I_d$
- $\mathbf{A}_{6.2}$ - conditions on gradient noise:
$$\mathbb{E}[s_i(x)|\mathcal{F}_{i-1}] = 0$$
$$\mathbb{E}\|s_i(x)\|^2|\mathcal{F}_{i-1} \leq \gamma^2 |x^* - x\|^2 + \sigma_s^2$$
- $\mathbf{A}_{6.3}$ - conditions on gradient noise:
$$\mathbb{E}[s_i(x)|\mathcal{F}_{i-1}] = 0$$
$$\mathbb{E}[\|s_i(w)\|^4|\mathcal{F}_{i-1}] \leq \gamma_4^4 \|x^* - x\|^4 + \sigma_{s,4}^4$$
where γ^2, σ_s^2, γ_4^4, and $\sigma_{s,4}^4$ are some nonnegative constants
- $\mathbf{A}_{6.4}$ - momentum parameter: The momentum parameter β is a constant that is not too close to 1, i.e., there exists a small fixed constant $\epsilon > 0$ such that $\beta \leq 1 - \epsilon$.
- $\mathbf{A}_{6.5}$: Consider the iterates z_{i-1} and x_{i-1} generated by the momentum recursion and stochastic gradient recursion. It is assumed that the noise process satisfies for some constants ϵ_1 and ϵ_2.:
$$\mathbb{E}[\|s_i(z_{i-1}) - s_i(x_{i-1}\|^2|\mathcal{F}_{i-1}] \leq \epsilon_1 \|z_{i-1} - x_{i-1}\|^2$$
$$\mathbb{E}[\|s_i(z_{i-1}) - s_i(x_{i-1}\|^4|\mathcal{F}_{i-1}] \leq \epsilon_2 \|z_{i-1} - x_{i-1}\|^4$$
- $\mathbf{A}_{6.6}$: The Hessian of the loss function $f(x)$ is Lipschitz continuous, i.e., for any two variables $x_1, x_2 \in \text{dom} f(x)$, it holds that $\|\nabla_x^2 f(x_1) - \nabla_x^2 f(x_2)\| \leq \kappa \|x_1 - x_2\|$ for some constant $\kappa \geq 0$.

Techniques such as Stochastic Gradient Descent are used to solve stochastic optimization problems. Recall that an iteration of Stochastic Gradient Descent is as follows:
$$x_i = x_{i-1} - \eta \nabla_x F(w_{i-1}; \xi_i), i \geq 0$$
where $\eta > 0$ is the step-size parameter.

Two popular choices for the step-size in stochastic methods such as SGD are decaying step-sizes and constant step-sizes. An example of a decaying step-size is $\eta(i) = \tau/i$ for some constant τ and i indicating the iteration of the algorithm. Decaying step-sizes have the advantage of ensuring asymptotic convergence towards the optimal solution. However, there is a trade-off in which the convergence rate is only of the order $O(1/i)$ for strongly-convex loss functions. This means there is a sublinear convergence rate for decaying step-sizes. On the other hand, a constant step-size allows for an exponential convergence rate of $O(\alpha^i)$ for some $\alpha \in (0, 1)$. This advantage comes at the downside of convergence towards a solution within the neighborhood of the optimal solution instead of almost-sure convergence meaning there is loss in the accuracy of the solution. To be exact, the algorithm converges, in the mean-square-error sense, to a neighborhood around the optimal solution x^* of radius on the order of $O(\eta)$. The failure to converge to the exact solution is argued to be acceptable in certain circumstances due to modeling errors when formulating the optimization problem or improving the generalization ability of a model by preventing it from overfitting to the data. In any case, constant step-sizes are the preferred choice for stochastic algorithms used for online stochastic optimization problems because it allows the algorithm to continue to adapt and correct itself based off of the incoming data which can potentially change the overall distribution of the data. In comparison, using decaying step-sizes prevents the algorithm from properly adapting and learning as more iterations pass as the step-size will approach 0 as $i \to \infty$. Hence, this paper focuses on constant step-sizes in its analysis of stochastic momentum methods.

3.2.2. Algorithm

The paper notes the benefit of acceleration that momentum methods have for solving deterministic optimization problems that are $\mu-$strongly convex and L-smooth. Hence, the paper focuses on developing stochastic momentum methods for solving online stochastic optimization problems. The stochastic momentum method they develop is as follows: Here η_m is some constant step-size. Note that this general form encompasses both Nesterov's accelerated gradient method and the heavy-ball method. When $\beta_1 = 0$ and $\beta_2 = \beta$, the method simplifies to the heavy-ball method. When $\beta_2 = 0$ and $\beta_1 = \beta$, the method simplifies to Nesterov's accelerated gradient method. β_1 and β_2 satisfy $\beta_1 + \beta_2 = \beta$ and $\beta_1 \beta_2 = 0$.

Algorithm 2 General Momentum Stochastic Gradient Method

```
1: Input: $x_{-2} = z_{-2} =$ initial states;
   $x_{-1} = x_{-2} - \eta_m \nabla_x F(x_{-2}; \xi_{-1})$;
   $\beta_1, \beta_2 \in [0, 1)$ are momentum parameters
2: for $i = 0, 1, ..., T-1$ do
3:     $z_{i-1} = x_{i-1} + \beta_1 (x_{i-1} - x_{i-2})$
4:     $x_i = z_{i-1} - \eta_m \nabla_x F(z_{i-1}; \xi_i) + \beta_2 (z_{i-1} - z_{i-2})$
5: end for
```

3.2.3. Results

The paper finds that the benefits of momentum methods found for deterministic optimization problems do not necessarily carry over when these methods are adapted for online stochastic optimization problems while employing constant step-sizes. For small enough step-sizes η and η_m, where η is the step-size parameter for SGD and η_m is the step-size parameter for the momentum method, that satisfy the relation

$$\eta = \frac{\eta_m}{1-\beta}$$

while having β not be that close to 1, any advantage by using momentum can be achieved by vanilla SGD by using a larger step-size. Formally, this can be stated as follows.

Under $\mathbf{A}_{6.1}, \mathbf{A}_{6.2}, \mathbf{A}_{6.4}$, having sufficiently small step-sizes, and having the above relation hold, the following limit holds for quadratic loss functions

$$\mathbb{E}\|x_{m,i} - x_i\|^2 = O(\eta^2), i = 0, 1, 2, ... \tag{29}$$

where $x_{m,i}$ and x_i represent the iterates at time i by the General Momentum Stochastic Gradient Method and SGD algorithms respectively

Under $\mathbf{A}_{6.1}, \mathbf{A}_{6.3}, \mathbf{A}_{6.4}, \mathbf{A}_{6.5}, \mathbf{A}_{6.6}$, having sufficiently small step-sizes, and having the above relation hold, the following limit holds for general loss functions

$$\mathbb{E}\|x_{m,i} - x_i\|^2 = O(\eta^{3/2}), i = 0, 1, 2, ... \tag{30}$$

Note that these results hold for every i and not just asymptotically. These two limits explicitly state the trajectories of stochastic momentum and vanilla SGD methods remain within $O(\mu^{3/2})$ for general loss functions and $O(\mu^2)$ for quadratic loss functions. This means that the trajectories ,in terms of mean-square-error, remain very close and are "equivalent" in that sense. In addition, it can be interpreted that the General Momentum Stochastic Gradient Method is essentially the same as running vanilla SGD with a larger step-size.

The paper further confirms their results in the following empirical result by solving the regularized logistic regression problem over the Adult Data Set. Note that in this case $\eta_m = \mu_m$ and $\eta = \mu$. ρ is the regularization parameter. The following parameters are used $\rho = 0.1$, $\eta = 0.1, \beta = 0.9$, and $\eta_m = (1-\beta)\eta = 0.01$. The green curve represents using a decaying momentum parameter where $\eta_m = 0.1$ and $\beta(i)$ decreases in a stair-wise manner, i.e., when $i \in [1, 200], \beta(i) = 0.9$; when $i \in [201, 400], \beta(i) = 0.9/200^{0.3}$; when $i \in [401, 600], \beta(i) = 0.9/400^{0.3}$; ...; when $i \in [1801, 2000], \beta(i) = 0.9/(1800^{0.3})$. The black curve represents SGD with a decaying step-size that satisfies $\eta(i) = \eta_m/[1-\beta(i)]$. The results can be seen in

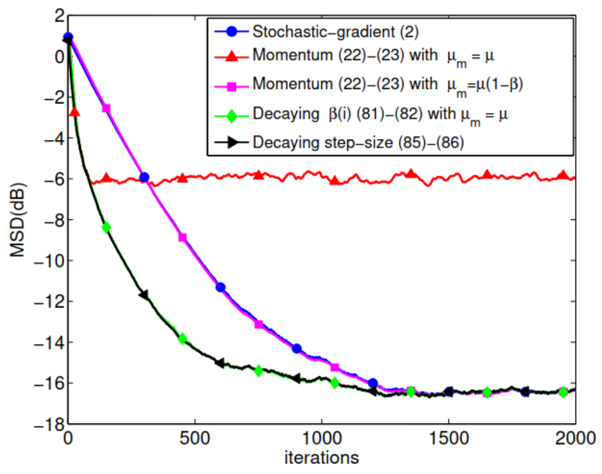


Figure 3. Empirical Results of SGD and Stochastic Momentum Methods for the regularized logistic regression problem.

Figure 3. The blue and pink curves which show SGD and the stochastic momentum method that satisfy the relation overlap almost completely. The green and black curve that show the stochastic momentum method with a decaying momentum parameter and constant step-size and SGD with a decaying step size that satisfy the relation also overlap exactly and have an expected accelerated rate. Hence, if SGD and the stochastic momentum method satisfy the relation $\eta = \frac{\eta_m}{1-\beta}$, then the algorithms run almost exactly the same.

3.3. Conclusion

It is still unproven in terms of analysis that stochastic momentum methods have definite accelerated rates in comparison to vanilla SGD. In many cases, the convergence rates of stochastic momentum methods can only be shown to be as good as those found for vanilla SGD for various classes of loss functions. Hence, there is still a significant amount of work to show that the accelerated rates seen in deterministic momentum methods carries over for stochastic momentum methods. This means that the underlying reasons of the benefits of stochastic momentum methods seen empirically are still not well understood or none. On the other hand, it has still been yet to be proven that stochastic momentum methods **do not** have accelerated rates in comparison to vanilla SGD. Lastly, the application of momentum based methods to decentralized optimization and online learning is still new and more empirical and theoretical work can be done to determine the potential benefits, lack of benefits, or drawbacks.

References

Assran, M. and Rabbat, M. G. On the convergence of nesterov's accelerated gradient method in stochastic settings. *CoRR*, abs/2002.12414, 2020. URL https://arxiv.org/abs/2002.12414.

Gower, R. M., Richtárik, P., and Bach, F. Stochastic quasi-gradient methods: Variance reduction via jacobian sketching, 2018.

Kulunchakov, A. and Mairal, J. Estimate sequences for variance-reduced stochastic composite optimization, 2019.

Liu, Y., Gao, Y., and Yin, W. An improved analysis of stochastic gradient descent with momentum, 2020.

Mai, V. V. and Johansson, M. Convergence of a stochastic gradient method with momentum for non-smooth non-convex optimization, 2021.

Nesterov, Y. *Introductory Lectures on Convex Optimization: A Basic Course*. Springer Publishing Company, Incorporated, 1 edition, 2014. ISBN 1461346916.

Polyak, B. *Introduction to Optimization*. 07 2020.

Sebbouh, O., Gower, R., and Defazio, A. On the convergence of the stochastic heavy ball method, 06 2020.

Taylor, A. and Bach, F. Stochastic first-order methods: non-asymptotic and computer-aided analyses via potential functions, 2020.

Yang, T., Lin, Q., and Li, Z. Unified convergence analysis of stochastic momentum methods for convex and non-convex optimization, 2016.

Ying, B., Yuan, K., Chen, Y., Hu, H., Pan, P., and Yin, W. Exponential graph is provably efficient for decentralized deep training, 2021.

Yuan, K., Ying, B., and Sayed, A. H. On the influence of momentum acceleration on online learning, 2016.

A Review of Recent Algorithmic Advances on Generalized Additive Models (GAMs) and Explainable AI

Franklin Zhang [1]

Abstract

Generalized additive models (GAMs) encompass a broad and powerful class of statistical algorithms. A mainstay in statistical learning toolkits since their introduction, GAMs have been extensively studied for both classification and regression tasks and are popular for both their expressibility and interpretability. This document reviews recent literature in combining GAMs with modern machine learning methods, such as deep neural nets (DNNs) and gradient-boosted trees, to both augment their efficacy while also recovering interpretability and re-opening these infamous "black boxes". We first explore the background and theory behind GAMs, explainable AI (XAI), and popular black-box models; we then examine recent proceedings in using GAMs to craft "glass-box" models; finally, we make a brief foray into efforts to explain existing black-box models using GAMs.

1. Introduction

The modern-day machine learning landscape is dominated by powerful learning techniques such as boosted or bagged trees, SVMs transformed with dimensionality-augmenting kernels, or deep neural nets (Lou et al., 2013). While excellent in predicting a variety of complex and high-dimensional problem landscapes, these "black-box" models are notorious for their complexity and lack of interpretability.

The black-box moniker stems from the fact that these machine learning models are too complicated for any human to understand. These models are difficult to troubleshoot for developers, difficult to interpret for practitioners, and thus difficult to trust for the layman user. Indeed, these models often predict the right answer for the wrong reason (the "Clever Hans" phenomenon) — taking advantage of confounding factors in datasets to achieve high performance (Schramowski et al., 2020). This phenomenon makes relying excessively on black-box machine learning models excessively dangerous, especially in fields where high-stakes, life-altering decisions are being made, such as medicine and the criminal justice system.

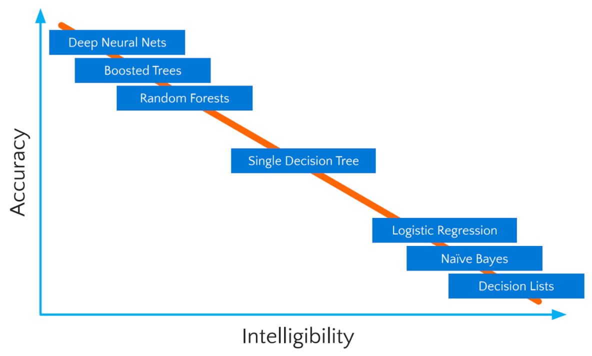


Figure 1. The trade-off between accuracy and interpretability in statistical learning models nowadays. Figure adapted from (Nori et al., 2019).

In pursuit of explainable AI (XAI), there has been a growing wave of interest in transforming these opaque machines into "glass-box" models — retaining their power while recapturing understandability. Spearheading one frontier in this field is a classic yet powerful statistical model, known as the generalized additive model (GAM), where the aim is to construct glass-box models from the get-go. In another direction, we seek to retrospectively explain black-box models using *post hoc* analyses. In this review, we will see both approaches, though we primarily focus on the former.

2. Background

2.1. Supervised Learning Problems

Supervised learning problems are a cornerstone problem of statistical machine learning. A typical supervised learning problem, defined on a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, of features $x_i \in X$ and labels $y_i \in Y$, seeks to estimate the response variable y given the set of observed features $x_1, \ldots, x_N$.

[1]Department of Computer Science, Rice University, Houston, Texas. Correspondence to: Franklin Zhang <yfz1@rice.edu>.

We do so by seeking a function f such that

$$\mathbb{E}[y|x_1, \ldots, x_N] = f(x_1, \ldots, x_N).$$

In other words, given a set of labeled observations, we seek a function f that captures the relationship between the predictor space X and the response space Y, conditioned on our observations so far. In a statistical sense, this definition captures the intuition behind teaching a model to learn some data.

It is common to formalize this notion of learning as an optimization problem: we define a *loss function*, denoted $\mathcal{L} : Y \times Y \to \mathbb{R}$, that measures how correctly our function f predicts Y given X, and look to solve the optimization problem

$$\min_{f \in \mathcal{F}} \mathbb{E}[\mathcal{L}(y_i, f(x_i))]$$

where $\mathcal{F}$ is the Hilbert space of all possible functions on X (Hastie et al., 2001; Lou et al., 2013).

2.2. Generalized Additive Models

Generalized Additive Models (GAMs) are a family of supervised learning models that takes the form

$$g(\mathbb{E}[y|x_1, \ldots, x_N]) = \beta + s_1(x_1) + s_2(x_2) + \cdots + s_N(x_N),$$

where $s_i(\cdot)$ are arbitrary smooth functions on X, colloquially termed *shape functions*, and $g(\cdot)$ is another smooth, usually invertible function called the *link function* (Hastie & Tibshirani, 1986). Intuitively, we can understand the $s_i(\cdot)$ as either learned or pre-crafted functions that capture some complex behavior in feature x_i, while the link function $g(\cdot)$ can be thought of as one last transformation of the model output into something that fits our problem domain. For example, it is common for g to be the identity function for regression problems, where $Y \subseteq \mathbb{R}$, or the logistic function for binary classification problems, where $Y = \{-1, 1\}$.

To build some quick intuition: let us explore how we recover other statistical models through the massaging the definition of a GAM.

- Taking s_i to be linear and g to be identity, we recover ordinary linear regression.
- Taking s_i to be linear and g to be logistic, we recover logistic regression.
- Taking s_i to be classification or regression trees (CARTs), we recover a boosted or bagged tree model.
- Taking g to be (the inverse of) a smooth activation function (e.g. $\tanh$, sigmoid, etc.), we recover a hidden-layer node in a feed-forward neural network.

Now that we are experts, we proceed to the literature review.

3. Literature Review

3.1. Explainable Boosting Models

To the author's knowledge, the introduction of Explainable Boosting Models (EBMs) by (Lou et al., 2012; 2013), and implemented by Microsoft Research in (Nori et al., 2019) as the `InterpretML` framework, reinvigorated confidence in GAMs as a competitive modern machine learning model and popularized the rush of GAM-based XAI models.

An EBM is a GAM — that is, it takes the form

$$g(\mathbb{E}[y|X]) = \beta + \sum s_i(x_i) \tag{1}$$

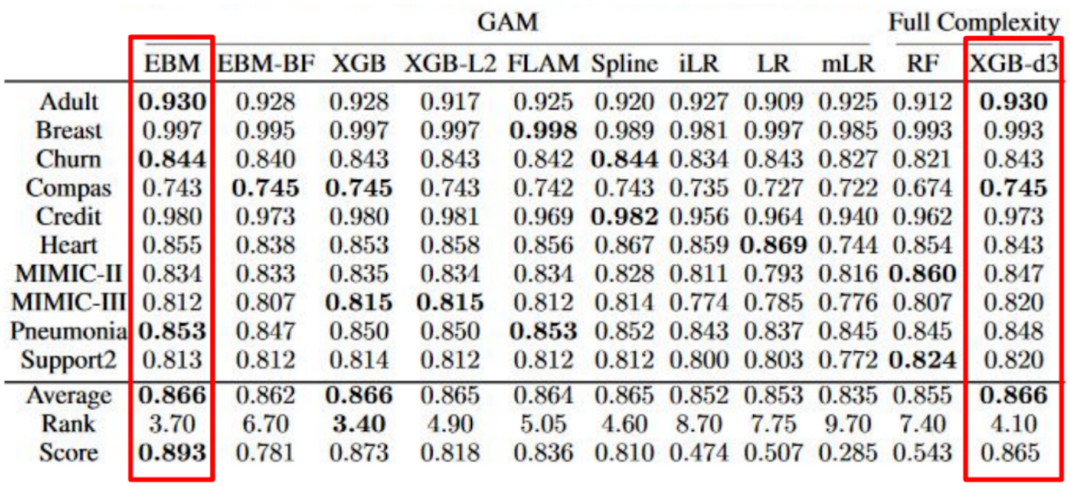
where g is the link function that adapts the GAM to the problem domain, such as regression or classification. The value of the EBM, however, draws from the modernization in formulation, fitting, and analysis.

Firstly, in contrast to using low-order polynomial splines or analytically-derived "scatterplot smoothers," as traditionally recommended by (Hastie & Tibshirani, 1986), EBMs use modern-day non-linear methods, such as classification and regression trees (CARTs), for their shape functions $s_i(\cdot)$ on the features x_i. With this heightened complexity, the traditional fitting algorithm of (penalized) iterative re-weighted least squares is exchanged for modern machine learning techniques as well, such as gradient boosting and bagging, to ensemble the shape functions into a GAM.

EBMs are constructed from a tree-ensemble boosting procedure (Nori et al., 2019; Lou et al., 2012). The fitting procedure is as follows:

1. Construct a GAM by fitting and boosting boosting CARTs on one feature at a time, in a round-robin fashion to mitigate the effects of co-linearity; the learning rate is set very low, so that feature order does not matter.
2. Repeat a large (e.g. 10,000-1,000,000) amount of times, collecting a variety of boosted shape functions to learn the full feature landscape.
3. Bag all learned shape functions together to obtain the final EBM — each shape function s_j thus becomes a gradient-boosted ensemble of bagged trees (Chang et al., 2021).

The intuition behind the explainability of an EBM comes from the simple observation that correlational relationships are aptly summarized by 2D-plots or heatmaps. Shape functions can be plotted to summarize local feature effects on the response, and after initial fitting, each function f_j can act as a lookup table per feature, returning a term contribution (Nori et al., 2019). Thus, after making use of black-box



	GAM									Full Complexity	
	EBM	EBM-BF	XGB	XGB-L2	FLAM	Spline	iLR	LR	mLR	RF	XGB-d3
Adult	**0.930**	0.928	0.928	0.917	0.925	0.920	0.927	0.909	0.925	0.912	**0.930**
Breast	0.997	0.995	0.997	0.997	**0.998**	0.989	0.981	0.997	0.985	0.993	0.993
Churn	**0.844**	0.840	0.843	0.843	0.842	**0.844**	0.834	0.843	0.827	0.821	0.843
Compas	0.743	**0.745**	**0.745**	0.743	0.742	0.743	0.735	0.727	0.722	0.674	**0.745**
Credit	0.980	0.973	0.980	0.981	0.969	**0.982**	0.956	0.964	0.940	0.962	0.973
Heart	0.855	0.838	0.853	0.858	0.856	0.867	0.859	**0.869**	0.744	0.854	0.843
MIMIC-II	0.834	0.833	0.835	0.834	0.834	0.828	0.811	0.793	0.816	**0.860**	0.847
MIMIC-III	0.812	0.807	**0.815**	**0.815**	0.812	0.814	0.774	0.785	0.776	0.807	0.820
Pneumonia	**0.853**	0.847	0.850	0.850	**0.853**	0.852	0.843	0.837	0.845	0.845	0.848
Support2	0.813	0.812	0.814	0.812	0.812	0.812	0.800	0.803	0.772	**0.824**	0.820
Average	**0.866**	0.862	**0.866**	0.865	0.864	0.865	0.852	0.853	0.835	0.855	**0.866**
Rank	3.70	6.70	**3.40**	4.90	5.05	4.60	8.70	7.75	9.70	7.40	4.10
Score	**0.893**	0.781	0.873	0.818	0.836	0.810	0.474	0.507	0.285	0.543	0.865

Figure 2. Test set AUCs (%) across ten satasets, averaged over five runs. Best number in each row is in bold. Figure adapted from (Chang et al., 2021).

methods to construct each s_j, predictions can be made by simply querying the lookup for term contributions, adding them up, then passing them through the link function g to compute the final prediction. In this way, we transform the model into a glass-box EBM.

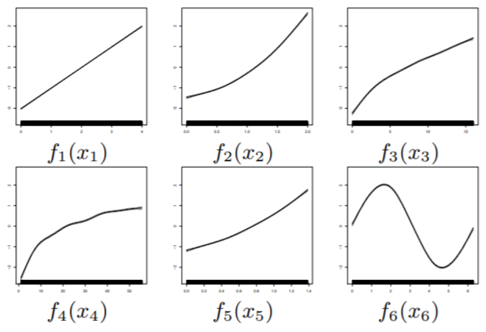


Figure 3. A synthetic example of visualizing univariate shape functions. Figure adapted from (Lou et al., 2012).

However, pairwise interactions can also be rendered as heatmaps of $s_{ij}(x_i, x_j)$ on the two-dimensional x_i, x_j-plane (Lou et al., 2013). One drawback of Equation (1) is that it does not model any interactions between features, since each $s_i(\cdot)$ is univariate. Expanding the model to the form

$$g(\mathbb{E}[y|X) = \beta + \sum s_i(x_i) + \sum s_{ij}(x_i, x_j) \quad (2)$$

gives us Generalized Additive Models plus Interactions (GA^{2}M). Efficient algorithms exist to detect non-spurious pairwise interactions without the quadratic explosion of brute-force comparisons (Lou et al., 2013).

The simplistic, additive nature of GAMs aids us in interpretability, but it has also historically limited the power of GAMs to model high-dimensional problems with complex, non-linear interactions. Empirical results tell us that the performance of GAMs, using traditional low-order shape functions, lie somewhere in-between simple models, such as linear or logistic regression, and full-complexity models, such as gradient-boosted trees, random forests, or deep neural nets (Lou et al., 2012).

However, EBMs escape this problem, likely by encapsulating model complexity in the shape functions while retaining interpretability through additivity. Recent benchmarks show that EBMs perform approximately as well as the best black-box models (e.g. XGBoost for gradient-boosted trees) (Wang et al., 2021; Chang et al., 2021).

3.2. Neural Additive Models

We alluded above how the shape functions s_j can be arbitrarily complex. By choosing each s_j to be a deep neural net (DNN), we dial this notion to the max. In Neural Additive Models (NAMs), introduced for classification and regression in (Agarwal et al., 2021) and learning-to-rank (LTR) in (Zhuang et al., 2020), we create a GAM where each individual feature is trained on a separate DNN. Each network is trained in parallel using backpropagation; since neural networks are universal approximators, these individual feature networks can learn arbitrarily complex shape functions (Hornik et al., 1989). The final model becomes a linear combination of the individual feature networks.

Model	MIMIC-II (AUC)	Credit (AUC)	CA Housing (RMSE)	FICO (RMSE)
Log./Linear Reg.	0.791 ± 0.007	0.975 ± 0.010	0.728 ± 0.015	4.344 ± 0.056
CART	0.768 ± 0.008	0.956 ± 0.004	0.720 ± 0.006	4.900 ± 0.113
NAMs	0.830 ± 0.008	0.980 ± 0.002	0.562 ± 0.007	3.490 ± 0.081
EBMs	0.835 ± 0.007	0.976 ± 0.009	0.557 ± 0.009	3.512 ± 0.095
XGBoost	0.844 ± 0.006	0.981 ± 0.008	0.532 ± 0.014	3.345 ± 0.071
DNNs	0.832 ± 0.009	0.978 ± 0.003	0.492 ± 0.009	3.324 ± 0.092

Figure 4. Comparison of NAM vs. other popular models on single-task learning tasks. Means and standard deviations are reported from 5-fold cross validation. Higher AUCs (classification) and lower RMSEs (regression) are better. Figure adapted from (Agarwal et al., 2021).

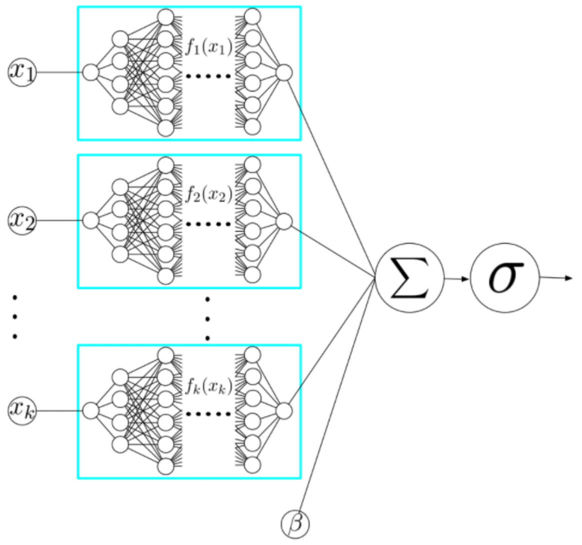


Figure 5. NAM architecture for classification models with sigmoid link function. Each input variable is handled by a different DNN. Figure adapted from (Agarwal et al., 2021).

As with EBMs, after fitting the shape functions transform into lookup tables, and we toss away the bulky DNNs in favor of these interpretable feature-to-response graphs. These graphs are not just heuristic; they are, in fact, exact descriptions of how the NAM computes its prediction, as the univariate shape functions map out the entirety of each network's feed-forward range (Agarwal et al., 2021).

Benchmarks show that NAMs remain competitive against full-complexity models, while also matching EBMs in both regression and classification tasks (Agarwal et al., 2021). However, there are a few key differences between the original boosted-tree EBMs and DNN-based GAMs, where NAMs distinguish themselves:

- The deep-learning community is much larger than the one for tree-based GAMs.
- NAMs are applicable to various settings that prove problematic for boosted CARTs. In particular, it has been shown that NAMs can be extended to multitask, multiclass, or multi-label learning without any changes to model formulation, training, or composability.
- Boosted-tree EBMs require a huge (e.g. up to millions) ensemble of decision trees to accurately map out each shape function (Nori et al., 2019; Lou et al., 2012); NAMs reduce this footprint to only a handful (2 - 100) of NNs (Agarwal et al., 2021).
- By taking advantage of GPUs, TPUs, and other specialized hardware for deep-learning, and in combination with the above point, NAMs become much more scalable than EBMs (Agarwal et al., 2021).
- Because NNs are *differentiable* via backpropagation, NAMs can be encoded as components of *contextual parameter generators*, enabling context-sensitive parameterized models (Agarwal et al., 2021; Zhuang et al., 2020). Agarwal et al. apply this for estimating personalized treatment benefits for Covid-19 patents under patient context.

In particular, the last point above enables NAMs — as a differentiable, nonlinear additive model — to be especially suited for the learning-to-rank problem (Zhuang et al., 2020).

3.3. GAMs for Learning-to-Rank

The supervised learning-to-rank (LTR) problem is defined a bit differentially than traditional classification or regression tasks. In the LTR problem, a dataset becomes a set of tuples $\mathcal{D} = \{(q, X, y)\}_{i=1}^{N}$, where for any given tuple (q, X, y):

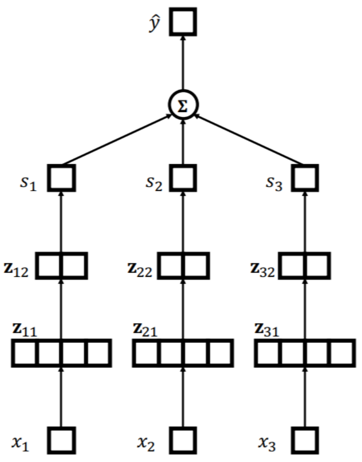


Figure 6. Architecture for a context-absent NAM. Note the similarities with Figure 5. Figure adopted from (Zhuang et al., 2020).

- $q = (q_1, \dots, q_m)^\top$ is an m-vector of contextual feature signals (e.g. query features in search tasks,
- $X = (x_1, \dots, x_p)^\top$ is a p-vector of data items. If x_j is also a vector, as is commonly the case, then X is a matrix; and so on for higher-order tensors.
- $y = (y_1, \dots, y_p)^\top$ is a p-vector of relevant labels, where $y_i \in \mathbb{R}$ labels x_i. A higher y_i indicates that item x_i is more relevant.

In the supervised LTR problem, we wish to learn a ranker φ on $\mathcal{D}$ to estimate a ranking $\hat{\pi} \in \Pi$ on X, where Π is the set of all permutations of X. As with other supervised learning problems, we would like the estimate $\hat{\pi}$ to be as close as possible to the ground-truth ranking $\pi^\star \in \Pi$, where the optimal ranking can always be obtained by sorting x_i on relevance labels y_i from highest to lowest (Zhuang et al., 2020). Because we would like φ to predict on unseen items X and context q, where we do not have the luxury of access to labels y, we need a way of estimating the rankings $\hat{y}_i$.

A common construction of φ has the ranker learn a scoring function F to estimate $\hat{y}_i \in \mathbb{R}$. If F takes both context features q and individual data items X, then φ provides an *context-present* ranking. If F only takes individual data items X, then φ provides a *context-absent* ranking.

In (Zhuang et al., 2020), a ranking GAM model is proposed for F, where context features encode importance weights for each item-level feature:

$$\hat{y}_i = F(q, x_i) = \sum_{j=1}^{p} \sum_{k=1}^{m} w_{jk}(q_k) \cdot f_j(x_{ij}) \tag{3}$$

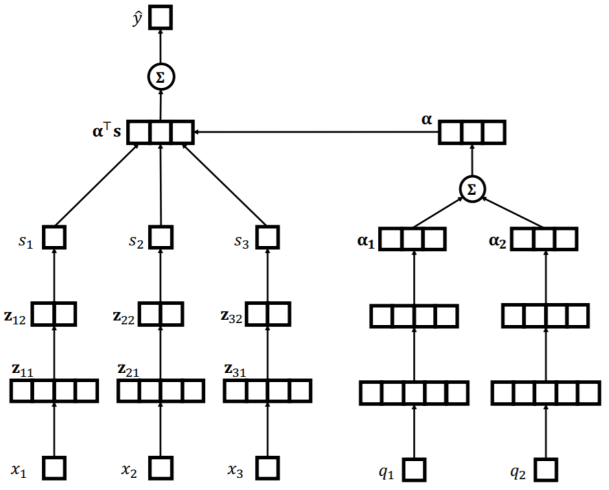


Figure 7. Architecture for a context-present NAM, where contexts q are encoded as importance weights. Figure adopted from (Zhuang et al., 2020).

where $w_{jk}(\cdot)$ and $f_j(\cdot)$ are both arbitrary univariate functions to be learned. For constructing importance functions w_{jk} and shape functions f_j, Zhuang et al. compared tree-based GAMs against neural-based GAMS (i.e. a NAM). Their results are reproduced in Figure 8.

Data set	Method	NDCG$_1$	NDCG$_5$	NDCG$_{10}$
YAHOO	Tree GAM	67.61	69.46	73.89
	Neural GAM	67.63	69.62	73.98
	Tree RankGAM	69.12	71.03	75.04
	Neural RankGAM	**69.36**	**71.32**	**75.33***
WEB30K	Tree GAM	29.79	32.79	35.96
	Neural GAM	30.59	33.55	36.54
	Tree RankGAM	41.90	42.04	44.37
	Neural RankGAM	**44.31***	**43.29***	**45.09***
CWS	Tree GAM	19.74	32.91	36.72
	Neural GAM	20.09	34.01	38.60
	Tree RankGAM	20.16	35.06	39.27
	Neural RankGAM	20.35	34.94	38.93
	Neural RankGAM+	**24.43***	**39.88***	**42.84***

Figure 8. Performance comparison (%) on three ranking tasks, where the evaluation metric is Normalized Discounted Cumulative Gain (NDCG) parameterized on $k = 1, 5, 10$. The best result per column is bolded; results that are statistically significantly better ($p < 0.01$) than Tree RankGAM are starred. Figure adapted from (Zhuang et al., 2020).

As with EBMs and NAMs, the novelty of using GAMs for crafting interpretable LTR models comes from our ability to extract feature relationships from additivity and visualize them directly. For context-absent rankings, these visualizations are identical to those described for EBMs and NAMs — feature-response shape functions and pairwise feature interactions. For context-present rankings, we can also describe how context ascribes importance to each feature. In Figure 9, we see an example of how users of a ranking GAM can visualize the patterns between item features and context-attributed importance. In particular, it's clearly apparent that the model has learned geographical and/or cultural similarities between the contexts — for example. regions US and UB share similar importance weight distributions for the same set of features, as do AU and NZ.

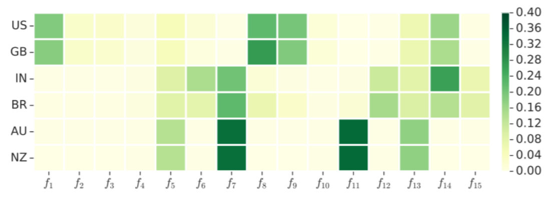


Figure 9. A heatmap of context feature importance, as learned and explained by GAMs. Each row is a specific context feature value; each column is a learned shape function an item $f_j(x_j)$. We see context clusters on geographic and/or cultural proximity. Figure adapted from (Zhuang et al., 2020).

3.4. SHapley Additive exPlanations (SHAP)

So far, we have primarily explored how GAMs enable us to construct "glass-box", informative models for supervised learning tasks. An orthogonal pursuit in the field of XAI, however, is the attempt to explain existing black-box models via *post hoc* methods. In this last section, we venture briefly in this direction and explore the notion of SHAP values for feature attribution in black-box models.

The SHAP framework was conceived and introduced by Lundberg & Lee for interpreting predictions by attributing feature importance in a black-box model. As the name suggests, SHAP values borrow the concept of Shapley values from game theory, devised by Shapley. SHAP is a unification of prior black-box explanatory methods (Bach et al., 2015; Datta et al., 2016; Lipovetsky & Conklin, 2001; Ribeiro et al., 2016; Shrikumar et al., 2017; Aas et al., 2020), many also using Shapley values.

In the game-theoretic definition (Shapley, 1953), we are interested in attributing the contributions of a set of players P towards the payoff of a coalitional or cooperative game. For the set of players P, we define a characteristic function $v : 2^P \to \mathbb{R}$, with $v(\varnothing) = 0$, that measures the expected sum of payoffs that a coalition $S \subseteq P$ can obtain by cooperating in this game. The Shapley value for player p is thus defined:

$$\phi_p(v, P) = \sum_{S \subseteq P} \frac{|S|!\,(N - |S| - 1)!}{N!}(v(S) - v(S \setminus \{p\})) \tag{4}$$

Intuitively, we read Equation 4 as the marginal contribution an actor p adds when joining coalition S, averaged over all possible permutations of forming S.

To transpose from a game-theoretic to statistical setting, we simply define the set of "players" as the set of features $\{x_1, \ldots, x_N\}$, and the "coalitional game" as the supervised learning problem. Thus, feature attribution becomes intuitively mapped to the marginal contribution of these "players" towards our cooperative game objective.

Before we arrive at the SHAP framework, we briefly discuss one crucial result that makes Shapley values especially desirable as an explanatory method. When explaining black-box models, it is usually the case that the original model does not explain itself well given its complexity; thus, we must use a simpler explanation model, defined as an interpretable approximation of the original model (Lundberg & Lee, 2017). We say an explanation model is an additive feature attribution method, when it is a linear combination of binary variables

$$g(z') = \phi_0 + \sum_{i=1}^{N} \phi_i z'_i, \tag{5}$$

where $z'_i \in \{0, 1\}$, N is the number of (simplified) input features, and $\phi_i \in \mathbb{R}$ is an attribution score. In other words, an additive feature attribution method is precisely we use a GAM as our explanatory model — note the similarity with Equation 1.

It is desirable for explanatory models to satisfy the following properties:

1. **Local accuracy.** When approximating the original black-box model f for specific input x, the explanatory model g should at least match the output of f for simplified input x.

2. **Missingness.** All features missing in the original input should also have no impact in the explanatory model.

3. **Consistency.** The attribution for a feature should be monotonically increasing with respect to its contribution (i.e. the explanation should not penalize features for improving the model).

An astonishing result is that the Shapley value is the only additive feature attribution method that satisfies all three properties above (Young, 1985)! That is — when explaining black-box models using GAMs, it is uniquely correct to

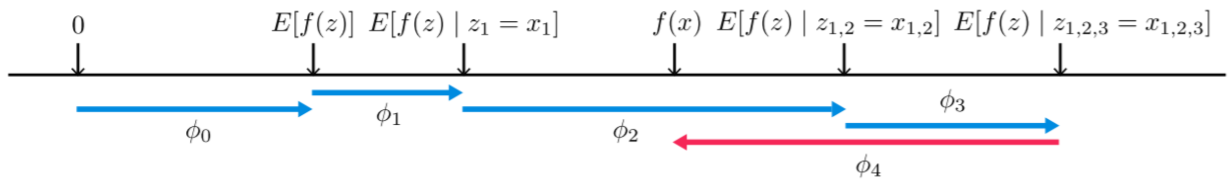


Figure 10. An illustration of how SHAP values attribute contribution scores to features. They score each feature by the marginal change in expected model prediction when conditioned on adding a new feature to the existing feature set. The diagram above shows a single ordering of adding features. Figure adapted from (Lundberg & Lee, 2017).

attribute feature contributions with Shapley values. Unfortunately, it has been shown that computing Shapley values is NP-Complete, in general (Deng & Papadimitriou, 1994). However, model-agnostic methods exist to approximate local values via sampling methods (Ribeiro et al., 2016; Lundberg & Lee, 2017), as do model-specific methods leveraging extra knowledge about our model, such as the recursive compositional nature of DNNs, to improve Shapley value computation (Lundberg & Lee, 2017).

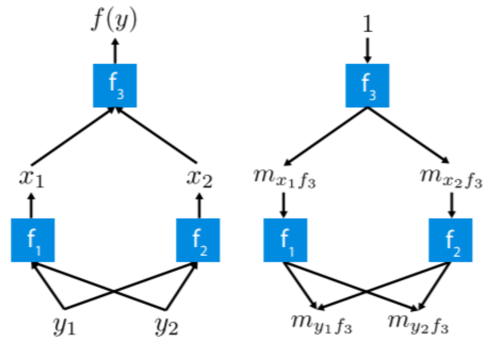


Figure 11. A visualization of the DeepSHAP algorithm for computing Shapley values on compositional models, such as deep neural nets. Figure adapted from (Lundberg & Lee, 2017).

Finally, we describe SHAP values. In definition, SHAP values are precisely Shapley values — that is, an attribution metric over the powerset of inputs $\{x_1, \ldots, x_N\}$. The novel contribution comes in the unification of existing methods for computing SHAP values. Existing methods either do not use Shapley values (thus violating correctness, if they are additive models), or approximate Shapley values by Monte Carlo or sampling methods (Lundberg & Lee, 2017). The SHAP framework simply adapts insights from these previous additive feature attribution methods to approximate SHAP values under both model-agnostic (Shapley sampling, Kernel SHAP) and model-specific settings (Max SHAP, Deep SHAP) (Lundberg & Lee, 2017). For example, Deep SHAP combines the backpropagation-based linearization of the DeepLIFT algorithm to recursively compute SHAP values on neural networks (see Figure 11).

The theoretical correctness of Shapley values as the only additive feature attribution method satisfying our intuition matches empirical benchmarks. A small-scale study comparing SHAP against previous explanatory methods (LIME, DeepLIFT) showed SHAP aligned most strongly with human intuition (Lundberg & Lee, 2017). In a larger-scale study, Lundberg et al. also showed that SHAP aligns with anaesthesiologist explanations for predicting hypoxaemia during surgery (Lundberg et al., 2018).

4. Conclusion

The tension between accuracy and interpretability of machine learning models has only widened as models grow more complex and powerful. In this work, we investigated two pursuits in the field of XAI for developing explainable model predictions: (1) constructing competitive glass-box models, and (2) attributing feature explanations to existing black-box models. In both frontiers, we saw the ubiquity and importance of GAMs: for constructing the intrinsically interpretable glass-box models, GAMs keep complexity understandable by additivity, while maintaining competitive performance against black-box models via arbitrary shape functions; for attributing feature contribution, Shapley values prove uniquely correct as GAM-based feature explanation models that align with human intuition.

It is important to mention the relative immaturity of XAI compared to mainstream black-box pursuits. While GAMs remain promising for achievable glass-box models, they suffer from limitations as well. For example, many glass-box models focus on tabular data only (Agarwal et al., 2021). While there exist efforts to explain black-box models on non-tabular data (e.g. pixel images) (Bach et al., 2015), to the author's knowledge, little progress is made in generalized interpretable models in this domain. Further, recent criticisms of explanatory models draw attention to failures in methods to accurately capture model behaviors, or failing to generalize explanations beyond the initial database (Rudin, 2019). Ultimately, the field of XAI remains promising yet nascent — more work is required to truly achieve explainable artificial intelligence.

References

Aas, K., Jullum, M., and Løland, A. Explaining individual predictions when features are dependent: More accurate approximations to shapley values, 2020.

Agarwal, R., Melnick, L., Frosst, N., Zhang, X., Lengerich, B., Caruana, R., and Hinton, G. Neural additive models: Interpretable machine learning with neural nets, 2021.

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS ONE*, 10, 2015.

Chang, C.-H., Tan, S., Lengerich, B., Goldenberg, A., and Caruana, R. How interpretable and trustworthy are gams?, 2021.

Datta, A., Sen, S., and Zick, Y. Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 598–617, 2016.

Deng, X. and Papadimitriou, C. H. On the complexity of cooperative solution concepts. *Math. Oper. Res.*, 19(2): 257–266, 1994. URL http://dblp.uni-trier.de/db/journals/mor/mor19.html#DengP94.

Hastie, T. and Tibshirani, R. Generalized additive models. *Statistical Science*, 1(3):297–318, 1986.

Hastie, T., Tibshirani, R., and Friedman, J. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

Lipovetsky, S. and Conklin, M. Analysis of regression in game theory approach. *Applied Stochastic Models in Business and Industry*, 17(4):319–330, 2001. doi: https://doi.org/10.1002/asmb.446. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/asmb.446.

Lou, Y., Caruana, R., and Gehrke, J. Intelligible models for classification and regression. In *KDD*, 2012.

Lou, Y., Caruana, R., Gehrke, J., and Hooker, G. Accurate intelligible models with pairwise interactions. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '13*. ACM Press, 2013. doi: 10.1145/2487575.2487579.

Lundberg, S. and Lee, S.-I. A unified approach to interpreting model predictions, 2017.

Lundberg, S. M., Nair, B., Vavilala, M. S., Horibe, M., Eisses, M. J., Adams, T., Liston, D. E., Low, D. K.-W., Newman, S.-F., Kim, J., et al. Explainable machine-learning predictions for the prevention of hypoxaemia during surgery. *Nature Biomedical Engineering*, 2(10): 749, 2018.

Nori, H., Jenkins, S., Koch, P., and Caruana, R. Interpretml: A unified framework for machine learning interpretability. *arXiv preprint arXiv:1909.09223*, 2019.

Ribeiro, M. T., Singh, S., and Guestrin, C. "Why Should I Trust You?": Explaining the predictions of any classifier, 2016.

Rudin, C. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, May 2019. doi: 10.1038/s42256-019-0048-x. URL https://doi.org/10.1038%2Fs42256-019-0048-x.

Schramowski, P., Stammer, W., Teso, S., Brugger, A., Shao, X., Luigs, H.-G., Mahlein, A.-K., and Kersting, K. Making deep neural networks right for the right scientific reasons by interacting with their explanations, 2020.

Shapley, L. S. *17. A Value for n-Person Games*, pp. 307–318. Princeton University Press, 1953. doi: doi:10.1515/9781400881970-018. URL https://doi.org/10.1515/9781400881970-018.

Shrikumar, A., Greenside, P., Shcherbina, A., and Kundaje, A. Not just a black box: Learning important features through propagating activation differences, 2017.

Wang, C., Han, B., Patel, B., and Rudin, C. In pursuit of interpretable, fair and accurate machine learning for criminal recidivism prediction. 2021.

Young, H. P. Monotonic solutions of cooperative games. *International Journal of Game Theory*, 14(2):65–72, 1985.

Zhuang, H., Wang, X., Bendersky, M., Grushetsky, A., Wu, Y., Mitrichev, P., Sterling, E., Bell, N., Ravina, W., and Qian, H. Interpretable learning-to-rank with generalized additive models, 2020.