《大数据分析中的优化算法选讲》课程大纲

课程简介

本课程聚焦于大数据分析中使用的优化算法，探讨近年来在机器学习、数据挖掘和统计计算中的新型优化方法。课程采用讨论式教学，基于前沿论文进行分析、推导和实验验证，以培养学生的阅读、理解和批判性思考能力。

课程目标

1. 了解大数据分析中的关键优化问题及挑战。
2. 学习几种主流和新兴的优化算法，并分析其理论基础和适用场景。
3. 通过论文阅读和讨论，提高学术研究能力。
4. 探索优化算法在大数据环境下的扩展与应用。

课程大纲

第一部分：优化基础与大数据挑战

1. **绪论**
   - 课程介绍与大纲概述
   - 大数据分析中的优化问题
   - 经典优化算法回顾（梯度下降、牛顿法、凸优化）
2. **大规模优化的挑战**
   - 计算复杂性与存储限制
   - 分布式与并行计算的必要性
   - 非凸优化问题的难点
   - 优化目标的设计

第二部分：前沿优化算法选讲

3. **随机优化方法**
   - 随机梯度下降（SGD）及其变体（Adam, RMSProp, AdaGrad）
   - 大规模机器学习中的优化
4. **分布式与并行优化**
   - 分布式梯度下降（DGD）与联邦优化
   - 多智能体优化与计算资源调度

5. **非凸优化与全局收敛**
    - 逃离鞍点的优化方法（随机扰动、动量方法）
    - 近期非凸优化理论突破
6. **强化学习与优化**
    - 强化学习中的优化方法
    - 自适应优化策略与博弈论方法
7. **优化中的稀疏性与低秩结构**
    - L1 罚项与稀疏学习
    - 低秩矩阵优化（核范数正则化）

第三部分：论文研讨与前沿应用

8. **论文研讨 I**
    - 近五年顶级会议（NeurIPS, ICML, AAAI）论文选读
    - 关键算法分析与实验复现讨论
9. **论文研讨 II**
    - 开源代码阅读与优化实践
    - 讨论优化算法在不同任务中的适用性
10. **优化在工业应用中的案例研究**

- 金融、医疗、推荐系统中的优化方法
- 面向超大规模数据的优化方案（Google, Meta, OpenAI等案例）

课程要求

- 阅读至少一篇前沿论文，并提交读书报告。
- 进行小组讨论，提出改进优化方法的设想。

相关文献

1. **Revocable Deep Reinforcement Learning with Affinity Regularization for Outlier-Robust Graph Matching**
    - **发表会议**：ICLR 2023（国际学习表征会议）
    - **链接**：https://blog.csdn.net/weixin_42645636/article/details/135068534
    - **摘要**：该论文提出了一种基于深度强化学习的图匹配方法RGM。该方法采用顺序节点匹配方案，自然适用于选择性内匹配策略来对抗离群点。此外，作者还设计了一个可撤销动作框

架，以提高代理在复杂约束图匹配中的灵活性，并提出了一种二次近似技术来正则化亲和度得分，以在存在离群点的情况下进行优化。

2. **Optimizing Solution-Samplers for Combinatorial Problems: The Landscape of Policy-Gradient Methods**
    - **发表会议**：NeurIPS 2023（神经信息处理系统大会）
    - **链接**：https://blog.csdn.net/weixin_42645636/article/details/135068534
    - **摘要**：本文介绍了一种新理论框架，用于分析深度神经网络和强化学习方法在解决组合问题方面的有效性。作者提出了一些问题，例如是否存在具有足够表达能力、可处理性和良性优化景观的生成模型。通过对这些问题的研究，作者得出了肯定的答案，并证明了该方法适用于多种组合问题。同时，作者还引入了一种新颖的正则化过程，有助于解决梯度消失和避免不良的静止点。

3. **SurCo: Learning Linear Surrogates For Combinatorial Nonlinear Optimization Problems**
    - **发表会议**：ICML 2023（国际机器学习会议）
    - **链接**：https://blog.csdn.net/weixin_42645636/article/details/135068534
    - **摘要**：论文介绍了一种名为SurCo的方法，用于解决具有非线性成本函数和组合约束条件的实际优化问题。作者提出了学习线性替代成本的SurCo，它可以用于现有的组合求解器中，以输出原始非线性组合优化问题的良好解决方案。作者还提出了三种SurCo变体，并通过实验表明，在现实世界的优化问题中，SurCo比最先进的方法和领域专家方法更快地找到了更好的解决方案。

4. **A Survey of Optimization Methods for Training DL Models: Theoretical Perspective on Convergence and Generalization**
    - **发表杂志**：Transactions on Machine Learning Research
    - **链接**：https://openreview.net/forum?id=TDujguk7NG&utm_source=chatgpt.com
    - **摘要**：As data sets grow in size and complexity, it is becoming more difficult to pull useful features from them using hand-crafted feature extractors. For this reason, deep learning (DL) frameworks are now widely popular. DL frameworks process input data using multi-layer networks. Importantly, DL approaches, as opposed to traditional machine learning (ML) methods, automatically find high-quality representation of complex data useful for a particular learning task. The Holy Grail of DL and one of the most mysterious challenges in all of modern ML is to develop a fundamental understanding of DL optimization and generalization. While numerous optimization techniques have been introduced in the literature to navigate the exploration of the highly non-convex DL optimization landscape, many survey papers reviewing them primarily focus on summarizing these methodologies, often overlooking the critical theoretical analyses of these methods. In this paper, we provide an extensive summary of the theoretical foundations of optimization methods in DL, including presenting various methodologies, their convergence analyses, and generalization abilities. This paper not only includes theoretical analysis of popular generic gradient-based first-order and second-order methods, but it also covers the analysis of the optimization techniques adapting to the properties of the DL loss landscape and explicitly encouraging the discovery of well-generalizing optimal points. Additionally, we

extend our discussion to distributed optimization methods that facilitate parallel computations, including both centralized and decentralized approaches. We provide both convex and non-convex analysis for the optimization algorithms considered in this survey paper. Finally, this paper aims to serve as a comprehensive theoretical handbook on optimization methods for DL, offering insights and understanding to both novice and seasoned researchers in the field.

5. **A comparison of optimization algorithms for deep learning**
   - **发表杂志**：International Journal of Pattern Recognition and Artificial Intelligence, 2020, 34(13): 2052013.
   - **链接**：https://www.worldscientific.com/doi/abs/10.1142/S0218001420520138
   - **摘要**：In recent years, we have witnessed the rise of deep learning. Deep neural networks have proved their success in many areas. However, the optimization of these networks has become more difficult as neural networks going deeper and datasets becoming bigger. Therefore, more advanced optimization algorithms have been proposed over the past years. In this study, widely used optimization algorithms for deep learning are examined in detail. To this end, these algorithms called adaptive gradient methods are implemented for both supervised and unsupervised tasks. The behavior of the algorithms during training and results on four image datasets, namely, MNIST, CIFAR-10, Kaggle Flowers and Labeled Faces in the Wild are compared by pointing out their differences against basic optimization algorithms.
6. **DeepSeek-V3 Technical Report**
   - **链接**：https://arxiv.org/abs/2412.19437
   - **摘要**：We present DeepSeek-V3, a strong Mixture-of-Experts (MoE) language model with 671B total parameters with 37B activated for each token. To achieve efficient inference and cost-effective training, DeepSeek-V3 adopts Multi-head Latent Attention (MLA) and DeepSeekMoE architectures, which were thoroughly validated in DeepSeek-V2. Furthermore, DeepSeek-V3 pioneers an auxiliary-loss-free strategy for load balancing and sets a multi-token prediction training objective for stronger performance. We pre-train DeepSeek-V3 on 14.8 trillion diverse and high-quality tokens, followed by Supervised Fine-Tuning and Reinforcement Learning stages to fully harness its capabilities. Comprehensive evaluations reveal that DeepSeek-V3 outperforms other open-source models and achieves performance comparable to leading closed-source models. Despite its excellent performance, DeepSeek-V3 requires only 2.788M H800 GPU hours for its full training. In addition, its training process is remarkably stable. Throughout the entire training process, we did not experience any irrecoverable loss spikes or perform any rollbacks.

A Survey of Optimization Methods for Training DL Models

1. **Optimization Challenges in Deep Learning (DL)**: As deep learning models grow in complexity and scale, optimization remains a central challenge, requiring methods that ensure fast

convergence, generalization, and scalability.

2. **Gradient-Based Methods**:
    - First-order methods (SGD, SGD with momentum, Adam, etc.) are widely used due to computational efficiency.
    - Second-order methods (Newton's Method, Quasi-Newton Methods like BFGS and L-BFGS) provide faster convergence per iteration but are computationally expensive.
    - Some recent approaches modify first-order methods with perturbations and stochastic dynamics to escape saddle points more effectively.
3. **Loss Landscape-Aware Methods**:
    - These approaches focus on adapting optimization techniques to the properties of deep learning loss landscapes.
    - Methods like Sharpness-Aware Minimization (SAM), Entropy-SGD, and Low-Pass Filter SGD aim to find solutions that generalize better by locating flatter minima.
4. **Distributed Optimization**:
    - Both centralized (Downpour SGD, Elastic Averaging SGD) and decentralized (D-PSGD, MATCHA) methods were discussed to improve scalability and efficiency across multiple computing nodes.
    - Trade-offs between convergence speed and communication overhead were analyzed.
5. **Theoretical Contributions**:
    - The paper provides a rigorous theoretical foundation for optimization methods, including convergence analysis and generalization error bounds.
    - Key assumptions like convexity, smoothness, variance bounds, and step-size choices are discussed in relation to their practical implications.
6. **Bridging Theory and Practice**:
    - While theoretical guarantees provide insights into optimization performance, real-world deep learning scenarios often deviate due to practical constraints.
    - The paper highlights open challenges in making theoretical insights more applicable to large-scale deep learning problems.

Main Challenges in Optimizing Deep Learning Models

Deep learning (DL) optimization is notoriously difficult due to the highly non-convex and high-dimensional loss landscapes involved. Neural network loss surfaces contain many saddle points and local minima, making it hard to analyze or guarantee finding a global optimum (1). Theoretical understanding of these landscapes remains limited – most analytical results rely on unrealistic assumptions or apply only to simple (e.g. shallow) networks (2). On the other hand, empirical evidence suggests that in very over-parameterized networks, many minima are nearly equivalent in training loss and generalization performance (poor local minima become less problematic as model size grows) (3). Key optimization goals in DL are thus to develop methods that **converge efficiently** on such complex terrain, **generalize well** to new data, **scale to large datasets and models**, and remain **consistent/stable** in highly parallel environments (4). Achieving all these simultaneously is challenging and is considered a "holy grail" for understanding DL optimization (5).

First-Order vs. Second-Order Optimization Methods (Convergence Rates)

First-order methods – which use only gradient information – are the workhorses of DL training due to their low per-iteration cost and ease of implementation (6). This category includes plain gradient descent and stochastic gradient descent (SGD) and their variants (momentum methods, AdaGrad/Adam, etc.). Theoretically, vanilla SGD achieves **sublinear convergence** in standard settings (e.g. $O(1/K)$ in strongly convex problems and on the order of $1/\sqrt{K}$ in non-convex settings, where K is the number of iterations) (7) (8). Momentum-based gradient methods share a similar worst-case convergence order as SGD in stochastic settings (9). Adaptive learning-rate methods like Adam also have comparable sublinear rates (e.g. $O(\ln K/\sqrt{K})$ in theory (10). In practice, however, first-order optimizers often converge faster than these bounds suggest, thanks to heuristics like learning-rate schedules and data-dependent adaptation (11) (12). (Indeed, the survey notes that theoretical rates do not fully explain the empirical speedups of adaptive methods (13).) First-order methods remain popular because they are memory-efficient and straightforward to apply stochastically on huge datasets (14), even if their theoretical convergence can be relatively slow per iteration.

Second-order methods exploit curvature information (the Hessian or Hessian approximations) to accelerate convergence. In theory, they can converge in far fewer iterations – for instance, Newton's method enjoys *quadratic* convergence near an optimum (dramatically faster than the linear rate of gradient descent under similar assumptions) (15). Quasi-Newton methods like BFGS attain **superlinear** convergence while avoiding an exact Hessian computation (16). In particular, the full BFGS algorithm reduces the per-iteration complexity from cubic to quadratic time, and its limited-memory variant (L-BFGS) further brings it down to roughly linear per iteration by storing a compact approximation of the Hessian (17) (18). The trade-off is that second-order methods have much higher computational and memory costs per step. Computing or inverting a Hessian in a deep network is prohibitively expensive (cubic time in the number of parameters) (19), which is why pure second-order methods are rarely used for large-scale DL in practice. Instead, practitioners often stick to first-order methods or use very limited curvature approximations, accepting a slower iteration speed in exchange for tractability. The survey highlights this contrast: second-order methods converge in fewer iterations theoretically, but their overhead makes them challenging to deploy in modern deep learning settings (20) (21).

Loss Landscape-Aware Optimization Strategies and Generalization

Not all minima of a neural network loss are equal – in particular, *flat minima* (where the loss landscape around the solution is broad and flat) tend to generalize better than *sharp minima* (narrow, steep basins) (22). To leverage this insight, researchers have developed optimization strategies that explicitly account for the loss landscape geometry and encourage finding flatter solutions. These techniques modify the training objective or process to favor regions of the parameter space that yield low loss *and* are robust to perturbations (indicative of flatness). Key categories of loss landscape-aware methods include:

- **Regularization-based approaches** – add explicit regularizers or penalties related to sharpness. For example, methods have used measures like minimum description length, local entropy, or "ϵ-sharpness" to penalize overly sharp minima (23). *Sharpness-Aware Minimization (SAM)* is a recent instance that optimizes a worst-case loss in an ϵ-radius neighborhood around the weights, thereby steering the optimizer toward parameters with uniformly low loss in their vicinity (i.e. flatter minima).
- **Surrogate loss smoothing** – modify the loss function itself to smooth out rugged minima. One approach is to evolve the objective via a diffusion process (24) (e.g. gradually convolving the loss with a Gaussian kernel), which can filter out sharp spikes in the landscape and make optimization trajectories less likely to get stuck in narrow minima. *Entropy-SGD* follows this idea by augmenting the loss with an entropy term, effectively optimizing a locally averaged version of the objective to find wider valleys.
- **Weight averaging strategies** – average network weights over time or across training runs to obtain flatter solutions (25). For instance, *Stochastic Weight Averaging (SWA)* averages model parameters from different epochs towards the end of training, producing a solution roughly located at the center of multiple SGD iterates. This tends to land in a broad basin of the loss surface and often improves generalization.
- **Noise injection and smoothing** – introduce randomness during training to explore flat regions. Methods like *SmoothOut* inject noise into the model weights or gradients and average the results from multiple noisy copies (26). This effectively smooths the loss landscape seen by the optimizer, helping it escape sharp, narrow minima. Some of these techniques run multiple perturbed models in parallel and aggregate their updates, which has been used in large-batch distributed training to maintain generalization when individual batch runs might converge to sharper minima (27).

The survey analyzes four representative methods – SAM, Entropy-SGD, Low-Pass Filter SGD, and SmoothOut – from the above categories (28). A common theme is that by favoring flat regions of the loss surface, these algorithms find solutions with better generalization performance (lower test error). The paper provides theoretical insight into this improvement: for example, it shows that modifying SGD with a loss-smoothing filter or noise can increase the algorithm's *uniform stability*, which directly bounds the generalization gap (29). In fact, the theoretical analysis demonstrates that these landscape-aware optimizers can achieve a provably smaller generalization error than standard SGD, under reasonable assumptions (30) (31). In summary, incorporating loss geometry into the optimization process tends to lead to flatter minima and hence models that perform better on unseen data (32). This line of research bridges the gap between pure optimization and generalization, showing how guiding the optimizer through particular regions of the landscape can yield more robust learned models.

Distributed Optimization Methods: Centralized vs. Decentralized

As datasets and models grow, it becomes infeasible to train a deep network on a single machine. **Distributed optimization** methods tackle this by parallelizing training across multiple workers. The survey distinguishes between *centralized* and *decentralized* strategies, each with its own design and theoretical considerations.

- **Centralized approaches:** These rely on a central parameter server (or coordinator) that aggregates updates from multiple worker nodes. A classic example is *Downpour SGD* (from Google's DistBelief), where each worker computes gradients on a subset of data and asynchronously sends updates to a central server which maintains the global model (33). This setup achieves data-parallel speedups but can suffer from **stale gradients** (updates arriving late while the model has already moved on) due to asynchrony (34). To improve on basic Downpour, algorithms like *Elastic Averaging SGD (EASGD)* introduce a soft coupling between each worker's local model and the global model. In EASGD, workers do not immediately overwrite the global parameters; instead, each worker's parameters are allowed to deviate from the central model to explore the landscape, while a periodic "elastic" force pulls them toward the central average (35). This encourages exploration of different minima by each worker and reduces the risk of all workers converging to the same sharp basin. An extension called *Leader SGD (LSGD)* further refines this idea by grouping workers and only periodically synchronizing with a "leader," aligning better with modern hardware architectures to reduce communication overhead (36). In general, centralized schemes benefit from a single global view of the model (making it easier to enforce consistency), but must address the communication bottleneck and potential single-point-of-failure at the parameter server.
- **Decentralized approaches:** These methods eliminate the central server and instead have workers communicate peer-to-peer according to some network topology (37). Each worker keeps a copy of the model and periodically exchanges parameters or gradients with its neighbors. A prototypical example is *Decentralized Parallel SGD (D-PSGD)* (38). In D-PSGD, after computing a gradient step on its local data, each worker averages its updated parameters with those of adjacent workers (as defined by a communication graph) (39). Over time, this neighbor-to-neighbor averaging drives all workers' models toward consensus, effectively approximating what a centralized averaging would do, but without any single coordinating node. The chief advantage is that decentralized training removes the server bottleneck, allowing better scaling when many workers are involved or network bandwidth is limited (40). However, analysis of decentralized algorithms must account for the network connectivity (e.g. the spectral properties of the graph's weight matrix) to ensure all workers converge to the same solution. The survey notes that under standard assumptions (like all workers optimizing the same objective and a connected communication graph), decentralized SGD can achieve convergence rates analogous to standard SGD, with an extra factor depending on the network's consensus speed (41) (42). Decentralized methods are thus a promising avenue to scale up training, though they require careful design of communication protocols and synchronization frequency to balance throughput and convergence accuracy.

The paper provides a thorough summary of both approaches, outlining their respective **advantages** and **limitations**. For centralized methods like Downpour and EASGD, it discusses the convergence guarantees (largely showing they can match the baseline SGD rate in both convex and non-convex settings under certain conditions) (43) (44). For decentralized methods, it covers theoretical results ensuring that, for example, D-PSGD converges to a near-optimal solution at a sublinear rate, with the gap to the true optimum diminishing as communication rounds proceed (45). In essence, distributed optimization allows deep learning training to be **parallelized** across compute nodes, and with the right algorithmic adjustments, one can retain provable convergence properties close to those of single-worker training. The survey highlights that practicality in this domain often lies in mitigating issues like gradient staleness (for asynchronous centralized SGD) and ensuring network-induced delays don't slow convergence (for decentralized SGD).

Theoretical Insights: Convergence and Generalization Analysis

A major contribution of the survey is compiling and elucidating the **theoretical guarantees** for a wide range of optimization methods in deep learning. This includes both **convergence rates** (how fast an algorithm approaches a minimum or stationary point) and **generalization error bounds** (how well the solution found minimizes the true risk versus the training risk). The paper presents known theoretical results (often from prior literature) in a unified framework, covering convex cases for intuition and extending to the non-convex scenarios relevant to deep networks. Below are some key theoretical findings highlighted:

- **Convergence of first-order methods:** In convex optimization, gradient descent and SGD variants have well-known convergence rates. For example, with a sufficiently small or decayed step size, SGD attains a convergence rate on the order of $1/K$ in strongly convex settings (46). In non-convex problems, one cannot guarantee convergence to a global minimum, but SGD is proven to make the gradient norm small at an $O(1/\sqrt{K})$ rate (a sublinear convergence toward a stationary point) (47). Momentum-based methods (e.g. heavy-ball or Nesterov momentum) and adaptive optimizers (Adagrad, Adam) were shown to share essentially the same order of convergence in theory (usually $O(1/\sqrt{K})$, sometimes with a logarithmic factor) (48). Notably, the survey points out that while momentum and adaptive methods often converge faster in practice, their theoretical asymptotic rates do not improve upon vanilla SGD in the worst case (49) (50). (For instance, Adam's proven rate is $O(\ln K/\sqrt{K})$, which is only a small adjustment to SGD's $O(1/\sqrt{K})$ (51).) On the other hand, the constants and conditions in these analyses differ, so direct comparisons are difficult – but at a high level, all these first-order methods achieve sublinear convergence in general non-convex problems. The survey also tabulates these results and notes that increasing the momentum hyperparameter can actually deteriorate some theoretical convergence bounds (52).

- **Convergence of second-order methods:** Classical theory shows that second-order algorithms can converge much faster *per iteration* if the cost of each iteration is ignored. Newton's method, for instance, enjoys quadratic convergence (the error decreases geometrically squared at each step) when applied to optimize a locally well-behaved objective (53). Quasi-Newton methods like BFGS are proven to have superlinear convergence – meaning they improve faster than any linear rate once close to optimum – under convex assumptions (54). The survey reiterates these points and provides the convergence rate results for Newton's method, BFGS, and L-BFGS. However, it also emphasizes that these results often assume convexity or local smoothness, and extending them to general deep learning scenarios (which are non-convex) is non-trivial. In practice, one typically doesn't see pure quadratic convergence in training deep nets because one never runs full Newton steps globally; nonetheless, these methods can reach high-accuracy optima in fewer iterations than first-order methods if they are feasible to run.

- **Loss landscape-aware methods' theory:** For the newer optimization strategies designed to find flat minima (SAM, Entropy-SGD, LPF-SGD, SmoothOut, etc.), the survey offers theoretical analysis mainly in terms of **uniform stability** and related generalization measures. A central theoretical tool is *algorithmic stability*: if an optimization algorithm's output does not change much when a single training example is modified, then the algorithm is said to be stable and one can bound its generalization error (55). The paper shows that techniques which smooth or regularize the loss landscape can increase the algorithm's stability. For example, adding noise (as in SmoothOut or stochastic Langevin-based methods) effectively smooths the loss function and can be shown to reduce the sensitivity of the final weights to any one data point (56). Consequently, one can prove a smaller generalization gap for such methods compared to standard SGD. In one analysis, the authors note that an SGD variant with a gradient low-pass filter yields a smaller stability constant and thus a lower generalization error bound than plain SGD (57). Similarly, SAM's update (which seeks parameters that minimize loss even under an adversarial perturbation) can be seen as optimizing a uniformly worse-case loss, which intuitively should lead to a solution that is stable against input perturbations and noise. While the survey primarily summarizes known results, it provides a cohesive view that these landscape-aware algorithms not only find minima with low training loss but also have **provable generalization advantages** by virtue of the flatter, smoother loss basin they converge to (58).

- **Generalization error analysis:** Beyond stability, the survey touches on other theoretical frameworks for generalization in the context of optimization. It references classical results (e.g. those of Hardt et al. 2016) which give generalization guarantees for SGD under certain conditions (59) (60). In particular, for convex loss functions, one can bound the generalization error of SGD in terms of the number of passes and learning rate schedule (61). For non-convex cases, the results are weaker but still indicate that, as the number of training samples grows, the generalization error of SGD tends to decrease (62). The survey also compares algorithms: interestingly, one finding is that adding heavy momentum (as in the stochastic heavy-ball method) may hurt generalization compared to vanilla SGD – the analysis showed SGD (with no momentum) had a smaller generalization bound than the momentum method under their studied setup (63). This aligns with some empirical observations that adaptive or accelerated optimizers can sometimes lead to worse test performance even if they optimize training loss faster. Overall, the theoretical contributions of the paper lie in systematically presenting these convergence proofs and generalization error bounds for the gamut of optimization methods, thereby serving as a **comprehensive reference**. It bridges results from the optimization literature and generalization theory to provide a clearer picture of why and how certain methods work for deep learning (64) (65).

Practical Challenges in Applying Theoretical Results to Real-World DL

While the paper provides a rich theoretical perspective, it also implicitly highlights the difficulties in translating theory directly into practice for deep learning:

- **Mismatch between theory assumptions and reality:** Many convergence proofs require assumptions like convexity, Lipschitz continuity of gradients, or IID data that are not strictly true for modern deep nets (66). Deep learning landscapes are non-convex and complex, and data can be non-identically distributed (especially in distributed or federated settings). This means theoretical guarantees (e.g. global convergence or strong stability bounds) often do not directly hold in the realistic training scenario or are derived under conditions that practitioners can't verify. Thus, there is a gap between the idealized models used in theory and the behavior of actual neural network training.
- **Computational feasibility vs. theoretical optimality:** Methods that are theoretically fast (like second-order Newton-type algorithms) or provably find flatter minima (like certain regularized objectives) can be computationally impractical at scale. For example, Newton's method requires inverting a Hessian matrix, an $O(n^3)$ operation in the number of parameters, which is intractable for modern networks (67). Even storing a full Hessian or its approximation (BFGS) is memory-intensive (68). As a result, practitioners rarely use pure second-order methods despite their superior convergence properties. Instead, they rely on simpler first-order methods, accepting slower theoretical convergence for the sake of efficiency and scalability. This indicates a challenge: the algorithms that theory tells us are optimal often cannot be deployed on real problems without modifications. Similarly, some "loss landscape-aware" techniques require heavy computation (e.g. SAM requires two forward-backward passes per step), which can be a barrier despite their generalization benefits.
- **Empirical methods outpacing theory:** In some cases, techniques that work remarkably well in practice lack a satisfying theoretical explanation or guarantee. The survey points out the example of Hessian-free optimization – an approach that uses iterative methods to approximate second-order steps without explicit Hessian construction – which has enabled training of certain deep networks (notably by Martens et al.) but remains largely empirically justified. There are **no strong theoretical convergence guarantees** for these Hessian-free methods. Researchers often deploy such methods based on intuition and empirical results, with theory catching up only later if at all. This reflects a broader reality that practical innovation in DL optimization can get ahead of what current theory can explain, leaving a gap in our formal understanding.
- **Discrepancies in observed vs. predicted performance:** Theoretical bounds on convergence or generalization are often *worst-case* and may not capture the typical behavior of an algorithm on real data. For instance, according to theory, adaptive optimizers like Adam have essentially the same asymptotic order of convergence as SGD (69), yet empirically they often reach a good solution much faster in wall-clock time or require fewer epochs on complex problems (70). This suggests that our theoretical analysis is missing some facets of why these methods work well (e.g. how they adapt to the data geometry in practice). Likewise, generalization bounds for deep learning (when obtained) are frequently too loose to be practical – they might guarantee, say, that the test error is within 5% of training error only when the number of samples is astronomically large, whereas in reality deep networks already achieve low test error with far

fewer samples. The survey acknowledges, for example, that even though we can bound generalization error via stability, it “might not fully capture” the differences between optimizers in practice (71). In short, the **theory often doesn’t tightly reflect the empirics**, making it hard for practitioners to rely solely on theoretical guidance.

- **Hyperparameter tuning and other practical nuances:** Real-world training involves many hyperparameters (learning rate schedules, batch size, momentum coefficients, etc.) that theoretical analyses usually fix or assume optimally set. In practice, finding the right settings is an art informed by experimentation. The survey notes that adaptive methods are more robust to learning rate choice than SGD (72), which is an important practical advantage not obvious from theory. Moreover, issues like numerical stability, hardware-specific optimizations, and parallel synchronization all influence training outcomes but are abstracted away in most theoretical models. Bridging this gap requires additional theoretical work or guidelines that take these factors into account.

In summary, while significant progress has been made in understanding DL optimization theoretically, applying these insights to improve real-world training remains non-trivial. The paper serves as a “theoretical handbook” (73), consolidating what is known about convergence and generalization for various optimization methods. However, it also underlines that developing a *fundamental understanding* of deep learning training is still an ongoing challenge (74). Bridging the divide between theory and practice will require refining theoretical models to better reflect practical conditions, and conversely, continuing to derive new theory that can explain and predict the empirical successes of modern optimization techniques. The hope is that such efforts will eventually yield optimization methods that are not only provably efficient and generalizable, but also practically effective for training the next generation of deep neural networks (75) (76).

Most widely used optimization methods for DL

1. Stochastic Gradient Descent (SGD)

💡 *SGD updates parameters using a noisy estimate of the gradient, computed on small batches rather than the entire dataset.*

Advantages:

✓ **Efficient for large-scale datasets** – Avoids computing full gradients over the entire dataset, making it feasible for deep learning.
✓ **Converges to good minima** – Empirically, SGD often finds flatter minima, which generalize better to unseen data.
✓ **Memory-efficient** – Requires storing only gradients and model parameters, making it feasible for large models.

Disadvantages:

✗ **Slow convergence** – SGD exhibits high variance in updates, leading to slow progress.
✗ **Sensitive to learning rate** – Improper learning rate settings can cause divergence or slow convergence.
✗ **Difficult to escape saddle points** – Vanilla SGD struggles in loss landscapes with many saddle points, slowing down optimization.

---

2. SGD with Momentum

💡 *Momentum-based SGD accumulates a moving average of past gradients to accelerate convergence and reduce oscillations.*

Advantages:

✓ **Faster convergence than vanilla SGD** – Helps accelerate updates in the right direction.
✓ **Dampens oscillations** – Reduces the zig-zag movement of SGD, especially in ravines of the loss landscape.
✓ **Improves stability** – Works well for training deep networks with complex loss surfaces.

Disadvantages:

✗ **May overshoot** – Large momentum values can cause divergence, requiring careful tuning.
✗ **Less adaptive to sudden changes** – If the loss landscape shifts, the accumulated momentum can delay adaptation.

---

3. Nesterov Accelerated Gradient (NAG)

💡 *An improvement over momentum-based SGD that computes the gradient at a "look-ahead" position to improve optimization speed.*

Advantages:

✓ **More responsive than standard momentum** – Corrects the update direction before taking a full step.
✓ **Improves convergence speed** – Especially useful in convex and near-convex problems.
✓ **Can generalize better** – Helps avoid sharp minima, similar to momentum-based approaches.

Disadvantages:

✗ **More computational overhead** – Requires computing an additional gradient evaluation.
✗ **Still requires tuning hyperparameters** – Momentum and learning rate must be carefully adjusted.

---

4. AdaGrad (Adaptive Gradient Algorithm)

💡 *Adjusts the learning rate individually for each parameter based on past gradient magnitudes.*

Advantages:

✓ **Good for sparse data** – In NLP or recommender systems, it adapts well to infrequent updates.
✓ **No need for manual learning rate tuning** – Learning rates decrease automatically over time.

Disadvantages:

✗ **Aggressive learning rate decay** – As training progresses, learning rates shrink too much, leading to early stagnation.
✗ **Not ideal for deep learning** – Works well in convex settings but struggles with large deep networks.

---

5. RMSprop (Root Mean Square Propagation)

💡 *A modification of AdaGrad that prevents the learning rate from shrinking too quickly by using an exponential moving average of past gradients.*

Advantages:

✓ **Works well in non-stationary settings** – Adapts to changes in gradient magnitudes.
✓ **Prevents learning rate decay issues** – Unlike AdaGrad, it maintains a reasonable learning rate.
✓ **Suitable for recurrent networks (RNNs, LSTMs)** – Performs well in training deep and sequential models.

Disadvantages:

✗ **Hyperparameter tuning required** – The decay rate must be chosen carefully for best results.
✗ **Can still get stuck in sharp minima** – Does not explicitly encourage flat minima for better generalization.

---

6. Adam (Adaptive Moment Estimation)

💡 *Combines ideas from momentum-based SGD and RMSprop, maintaining separate learning rates for each parameter and incorporating adaptive updates.*

Advantages:

✓ **Fast convergence** – Generally converges faster than vanilla SGD.
✓ **Resilient to poor hyperparameter choices** – Works well with minimal tuning.
✓ **Effective for deep networks** – Used widely in CV (Convolutional Neural Networks) and NLP (Transformer models).

Disadvantages:

✗ **May lead to poor generalization** – Often finds sharp minima, which can result in worse test accuracy.
✗ **Not always stable in very large-scale training** – Sometimes, Adam struggles with very large datasets or architectures.

---

7. AdamW (Decoupled Weight Decay in Adam)

💡 *A modification of Adam that improves regularization by decoupling weight decay from the gradient update.*

Advantages:

✓ **Better generalization than Adam** – Reduces the tendency of Adam to overfit.
✓ **Improved stability** – Helps maintain good convergence properties without harming performance.

Disadvantages:

✗ **Still computationally expensive** – Requires additional parameter tracking like Adam.
✗ **Requires tuning weight decay** – Must balance decay and learning rate effectively.

---

8. L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno)

💡 *A quasi-Newton optimization method that approximates the Hessian matrix using past gradients.*

Advantages:

✓ **Very fast convergence** – Superlinear convergence in some cases.
✓ **Useful for smaller networks or convex problems** – Works well in settings with well-behaved loss surfaces.

Disadvantages:

✗ **High memory requirements** – Stores past gradients, making it less feasible for deep learning.
✗ **Not suitable for online or stochastic updates** – Best for batch optimization rather than mini-batch training.

Which Optimizer to Choose?

Optimizer	Best for	Strengths	Weaknesses
SGD	General deep learning	Simple, memory-efficient	Slow convergence, sensitive to learning rate

Optimizer	Best for	Strengths	Weaknesses
SGD with Momentum	Large-scale deep learning	Reduces oscillations, faster than SGD	May overshoot, hard to tune
NAG	Similar to Momentum but more refined	Faster, more stable	Higher computational cost
AdaGrad	Sparse data, NLP	Good for rare updates	Learning rate shrinks too fast
RMSprop	RNNs, LSTMs	Adaptive learning rate	Requires tuning decay factor
Adam	General-purpose deep learning	Fast convergence, widely used	Can overfit, struggles with large-scale optimization
AdamW	Large-scale models	Better generalization than Adam	Still computationally expensive
L-BFGS	Small networks, convex problems	Superlinear convergence	High memory usage, not suited for mini-batch training

Final Thoughts

- **For large-scale deep learning** (CNNs, NLP models), **Adam** or **SGD with momentum** are the go-to choices.
- **For small-scale or convex problems**, **L-BFGS** or **SGD** works well.
- **For recurrent models (RNNs, LSTMs)**, **RMSprop** or **Adam** are often used.
- **For better generalization**, **SGD** (possibly with momentum) is still considered superior to adaptive methods like Adam.