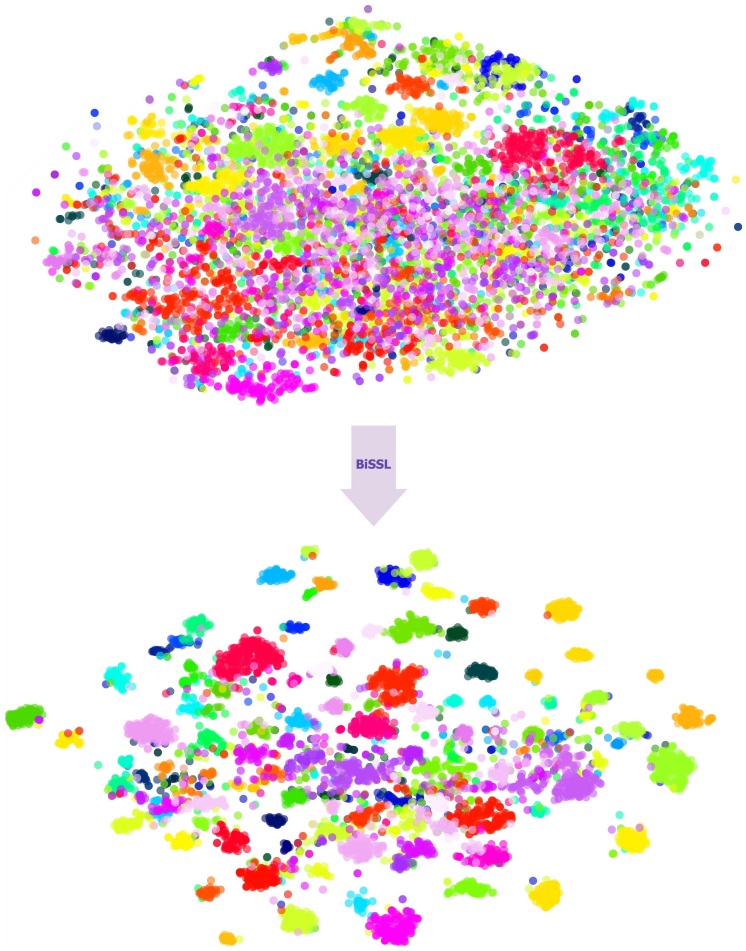

Master's Thesis
Mathematical Engineering

Revisiting Bilevel Optimization for Aligning Self-Supervised Pretraining with Downstream Fine-Tuning

Advancing BiSSL Through Systematic Variations, Novel Design Modifications, and Adaptation to New Data Domains

Author:
Gustav Wagner Zakarias

Supervisors:
Zheng-Hua Tan
Lars Kai Hansen

27th July 2025

Department of Mathematical Sciences
Thomas Manns Vej 23, DK-9220 Aalborg Ø
http://math.aau.dk

Department of Electronic Systems
Frederik Bajers Vej 7B, DK-9220 Aalborg Ø
http://es.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Revisiting Bilevel Optimization for Aligning Self-Supervised Pretraining with Downstream Fine-Tuning

Subtitle:
Advancing BiSSL Through Systematic Variations, Novel Design Modifications, and Adaptation to New Data Domains

Project Period:
Fall 2023 - Summer 2025

Author:
Gustav Wagner Zakarias

Supervisors:
Zheng-Hua Tan
Lars Kai Hansen

Copies: 1

Numbered Pages: 94

Date of Completion:
27th July 2025

Abstract:

The BiSSL framework models the pipeline of self-supervised pretraining followed by downstream fine-tuning as the lower- and upper-levels of a bilevel optimization problem. The lower-level parameters are additionally regularized to resemble the ones of the upper-level, which collectively yields a pretrained model initialization more aligned with the downstream task. This project extends the study of BiSSL by first evaluating its sensitivity to hyperparameter variations. Then, design modifications, including adaptive lower-level regularization scaling and generalized upper-level gradient expressions, are furthermore proposed and tested. Lastly, BiSSL is adapted to natural language processing tasks using the generative pretrained transformer pretext task, and then evaluated on a range of diverse downstream tasks. Results show that BiSSL is robust towards variations in most of its hyperparameters, provided that the training duration is sufficiently long. The proposed design modifications yield no consistent improvements and may even degrade performance. For natural language processing tasks, BiSSL achieves occasional gains and otherwise matches the baseline. The findings overall suggest that the original BiSSL design is robust, effective, and able to improve downstream accuracy across input domains.

Preface

This extended (60 ECTS) master's thesis was written in the period 01/09/23 to 20/06/25 by Gustav Wagner Zakarias, attending the final semesters of the masters' programme in Mathematical Engineering at Aalborg University, while concurrently enrolled in a 4+4 Ph.D. programme at the Department of Electronic Systems at Aalborg University and the Pioneer Centre for AI. For the attached code implementation utilized for the experiments of this project, `Python` 3.10.12 is used in conjunction with `numpy` 1.24.0, `pytorch` 2.1.2, `torchvision` 0.16.2, `timm` 1.0.15 and the Hugging Face libraries `transformers` 4.51.3, `datasets` 3.5.1 and `evaluate` 0.4.3. Besides this, `matplotlib` 3.10.1 is used in conjuction with Draw.io and Apple Freeform to create the figures and plots throughout the project. Overleaf v2 set with pdfLaTeX is used to write and compile the project.

The project author would like to thank supervisor Zheng-Hua Tan (AAU) and co-supervisor Lars Kai Hansen (DTU) for guidance throughout the project, and CLAAUDIA for providing the HPC resources that realized the experiments conducted for the project.

Gustav Wagner Zakarias

Contents

1 | Introduction

Supervised learning using deep neural networks [1] has emerged as a powerful technique for solving a wide array of machine learning tasks, delivering unparalleled performance within computer vision [2–7], natural language processing [8,9], and audio signal processing [10,11] among others. A common prerequisite for these successes is the availability of a substantially large pool of labeled data. However, in many real-world settings, acquiring sufficient labeled data is not feasible. This limitation often stems from high costs associated with collecting and manually annotating large datasets or from a fundamental scarcity of available data points to collect [12].

This sole reliance on explicit supervision stands in contrast to the way humans appear to learn. People can acquire new knowledge by drawing on unlabeled sensory inputs and leveraging prior knowledge to guide learning. In a sense, humans supervise themselves by constructing their own learning signals, which are acquired through interaction with the world [13,14]. In the meantime, while deep neural networks necessitate training via external supervision, they can still learn internal representations that similarly capture more general structure of the input, which can be repurposed to support learning on new tasks [15,16].

This raises the question of whether such transferable features can be learned without requiring labels at all, hence aligning more with how humans learn. Achieving this could enable a new training paradigm where unlabeled data serves as the primary resource, and labeled data is used only for final adaptation to downstream tasks. Self-supervised learning has emerged as a promising realization of this.

1.1 Self-Supervised Learning

Self-supervised learning (SSL) offers an alternative to end-to-end traditional supervised learning by leveraging unlabeled data to learn general-purpose representations. Rather than relying on task-specific labels, SSL first trains a backbone model using *pretext tasks*, which derive supervisory signals directly from the structure of the unlabeled data, guiding the model to extract general and informative features. This is followed by supervised fine-tuning on a downstream task, allowing the model to adapt its pretrained representations to improve task performance [12,15]. The general two-staged training pipeline in SSL is illustrated in Figure 1.1. As shown in the figure, auxiliary output heads are typically attached during both pretext and downstream training. They serve to align the backbone's output with the specific requirements of each task, especially when the format of the backbone output is incompatible with the task. In practice, such heads can also improve empirical performance, as some tasks benefit from solving the pretext task in a latent space separate from the backbone output space, which is further outlined in Chapter 2.

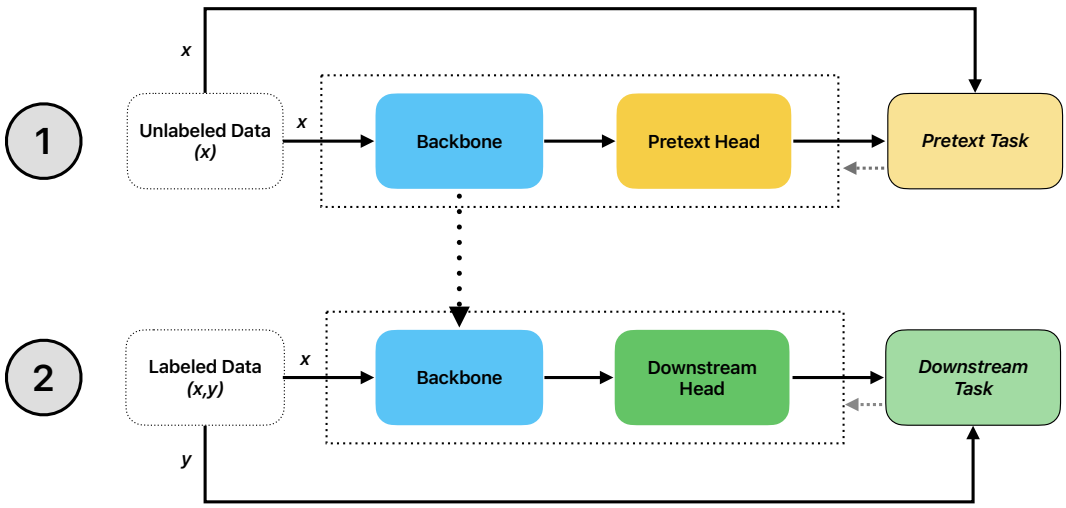

Figure 1.1: The conventional training pipeline in self-supervised learning of (1) pretext task training, followed by transmitting the backbone for subsequent (2) supervised training on the downstream task.

Designing effective pretext tasks is a central challenge in SSL. The most widely adopted approaches can be roughly grouped into four general categories: *contrastive*, *generative*, *predictive*, and *self-distillative* [17]. Figures 1.2 and 1.3 illustrate the high-level structure of these task types. However, this grouping is not strict, as many methods blur the boundaries between categories, and understanding them thus often requires a more holistic perspective. Nonetheless, the following overview presents representative examples from each category to convey their respective core design principles.

Contrastive pretext tasks learns representations by making a model distinguish between similar and dissimilar data points. The core idea is to map perceptually similar inputs to nearby points in a latent space, while also pushing apart representations of dissimilar inputs. Similar data points are typically achieved through augmentations: two augmented views of the same input are treated as a positive pair, while all other views in the batch (assumed to come from different inputs) form negative pairs [12, 18]. SimCLR [19] is a canonical example of contrastive learning in SSL. A single encoder model processes all augmented views, and a contrastive loss is used to pull together positive pairs while repelling all other instances in a batch. SimCLR is further explored in Chapter 2. Momentum Contrast (MoCo) [20] instead employs a dynamic memory bank of negative samples, achieved from a momentum encoder model whose weights are updated as an exponential moving average of a base encoder model. This design allows for a consistent set of negative samples across training steps, decoupling the number of available negative examples from the batch size. Bootstrap Your Own Latent (BYOL) [21] and SimSiam [22] avoid using negative pairs entirely. They use two separate networks to encode separate views of the same input, followed by having one network predict the output of the other. VICReg [23] shares many structural similarities with SimCLR, although it is not always classified as a contrastive method. The primary difference lies in the fact that its loss objective instead regularizes the latent space through an ensemble of three objectives: variance preservation across dimensions, decorrelation between dimensions, and invariance between paired augmented views. This enforces diversity between latent vectors within a

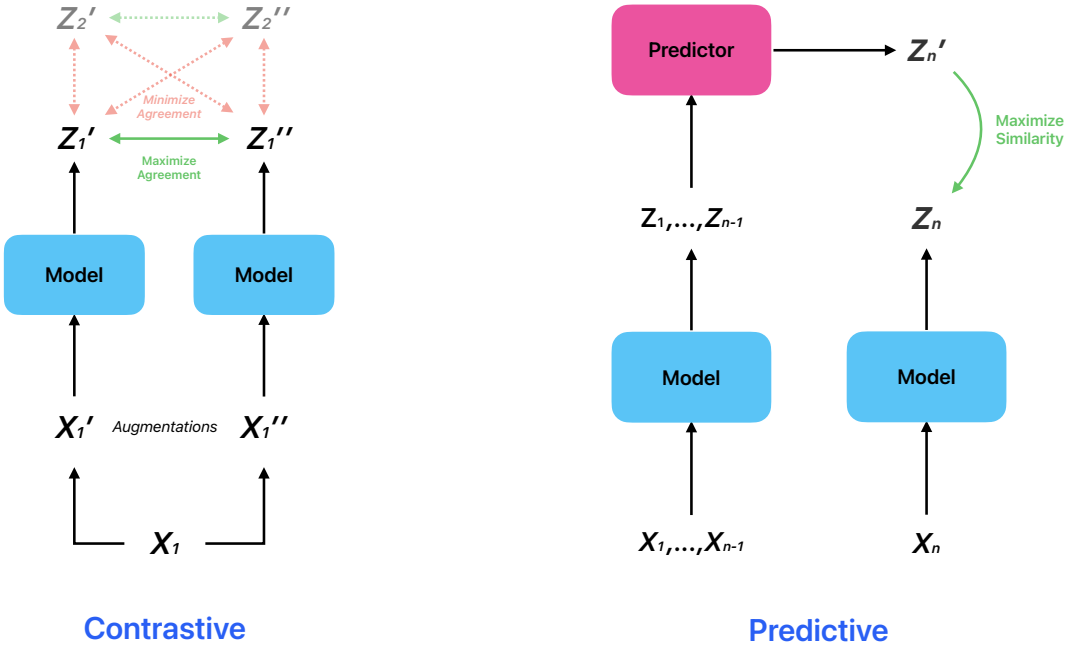

Figure 1.2: Diagrams illustrating the general setup of the contrastive (left) and predictive (right) pretext tasks. Contrastive pretext tasks aim to distinguish between similar and dissimilar data points, and predictive pretext tasks aim to predict future outcomes of sequential data.

batch and ensures that all dimensions of the latent space are non-redundant.

Predictive pretext tasks forecast future values based on past observed inputs. Hence, these tasks are naturally suited to sequential data, where temporal or structural dependencies can be leveraged for training without labels. Autoregressive predictive coding (APC) [24] generates predictions in the latent space and leverages the capabilities of deep neural networks with memory components, such as Long Short-Term Memory Networks (LSTMs) [25] and the widely popular Transformer models [26], to capture long-range dependencies. Bearing resemblance to contrastive methods, Contrastive Predictive Coding (CPC) [18] employs a contrastive loss in the latent space to distinguish between the target future value and a set of negative examples from past time steps or different sequences. In the realm of Natural Language Processing (NLP), BERT (Bidirectional Encoder Representations from Transformers) [27] uses Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). MLM randomly masks input tokens and tasks the model with predicting them based on their bidirectional context and NSP trains the model to predict whether one sentence logically follows another to learn sentence-level coherence.

Generative pretext tasks involve reconstructing either the full input or strategically corrupted portions of it, encouraging modeling of the underlying data distribution. Variational Autoencoders (VAEs) [28] utilize an encoder-decoder architecture with an intermediate bottleneck layer. Their objective is to reconstruct the input while simultaneously learning a lower-dimensional latent distribution in the bottleneck space. The generative capability arises from their ability to transform samples drawn from the latent distribution into samples resembling the input. Masked Autoencoders (MAEs) [29], originally designed

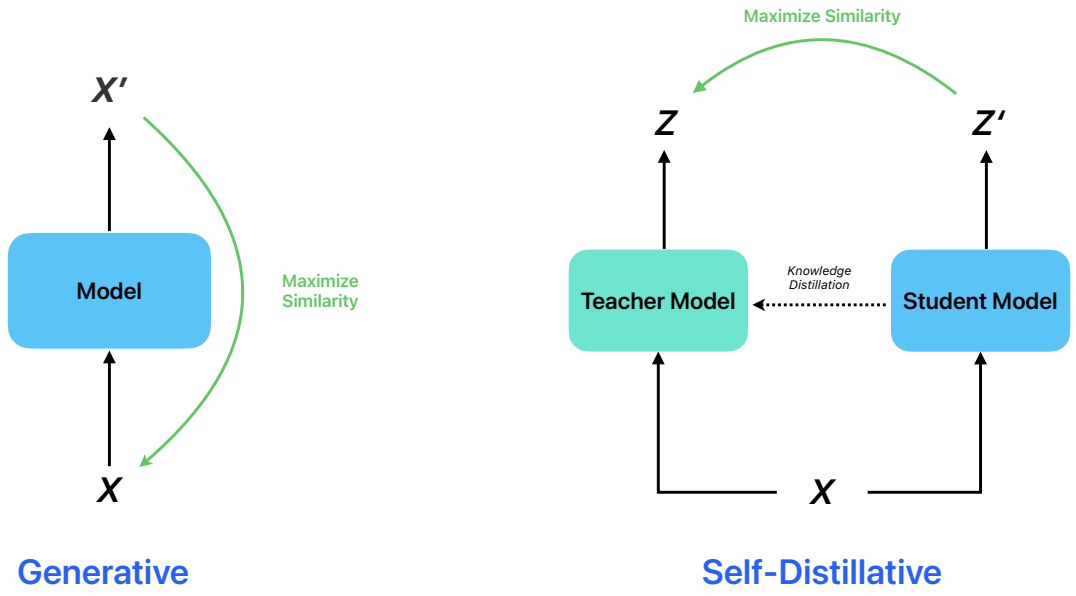

Figure 1.3: Diagrams illustrating the general setup of the generative (left) and self-distillative (right) pretext tasks. Generative pretext tasks aim to reconstruct the input, and self-distillative pretext tasks utilize a student-teacher model setup, where the student distills knowledge to the teacher, while the student is tasked with predicting the outputs of the teacher model.

for vision tasks, use a transformer-based encoder-decoder architecture. They are trained to reconstruct inputs from which large patches have been masked out before encoding. This forces the model to build semantic representations that support inferring the missing content from the unmasked patches. In the realm of NLP, the Generative Pretrained Transformer (GPT) [30–33] uses an autoregressive training objective, predicting the next token in a sequence given the preceding context. While similar in form to predictive tasks, this next-token generation objective enables coherent text synthesis, framing GPT as a generative model. GPT is further explored in Chapter 2. Lastly, BART [34] combines denoising autoencoder training with a transformer-based architecture, and is pretrained by corrupting text with various noising strategies (e.g., masking or sentence permutation), and then making the model reconstruct the original text.

Self-distillative pretext tasks commonly adopt a student-teacher setup, where the student model is trained to mimic the representations produced by the teacher. The teacher model is usually a distilled version of the student model, often derived from an exponential moving average of the student's weights during training. DINO [35] aims to learn high-quality visual representations by having the teacher model generate outputs which the student is tasked with mimicking. Unlike contrastive methods, DINO uses a consistency loss that focuses exclusively on positive pairs to encourage the model to produce consistent features for different views of the same input. The aforementioned pretext tasks of MoCo, BYOL, and SimSiam share similarities with this category as they use the output of two separate models for contrastive training. Lastly, in Data2Vec [36], the teacher processes the complete input, while the student receives a masked or partial version. The objective of the student model is then to predict the representations output

by the teacher model.

After pretext training is conducted, the task-specific head is typically discarded, and the pretrained backbone is repurposed for supervised training on a labeled downstream dataset.

1.1.1 Training on the Downstream Task

Adapting a self-supervised pretrained backbone to a downstream task typically also requires attaching a task-specific auxiliary model head to the pretrained backbone. The subsequent supervised training procedure then generally proceeds in one of two ways. During *linear probing*, the parameters of the backbone are frozen, and only the parameters of the newly attached head are updated. This approach is computationally inexpensive, as the head model is usually lightweight compared to the backbone model. It also helps prevent overfitting, particularly when the downstream dataset is small. However, its main drawback is that it may not yield optimal performance when the backbone requires adjustment to the specific downstream task. In contrast, the approach of *fine-tuning* involves additionally updating all or parts of the backbone during supervised training on the downstream task [3, 37]. This approach is typically preferred in practice over linear probing, as this approach generally leads to greater downstream performance [19, 20, 23].

A challenge with fine-tuning arises when the downstream dataset differs significantly from the pretraining data distribution. In such cases, the learned representations may be misaligned with the new task, and the fine-tuning process can degrade or overwrite useful semantic features acquired during self-supervised pretraining. This risk is further compounded when labeled data is scarce, as the fine-tuning process lacks sufficient guidance to identify which aspects of the pretrained features are task-relevant [38–41]. The conventional SSL pipeline treats pretext and downstream tasks as disjoint stages, connected only through the transferred backbone, as illustrated in Figure 1.1, which serves mainly as an initialization. One promising direction to address the aforementioned issue would be to enhance the alignment between the self-supervised pretraining and downstream fine-tuning stages. Bilevel optimization offers a framework for achieving this.

1.2 Bilevel Optimization in Self-Supervised Learning

Bilevel optimization (BLO) has emerged as a valuable tool for tackling hierarchical optimization problems in deep learning. It comprises two interdependent optimization problems, referred to as the upper and lower-level problems, respectively. The solution to the upper-level problem is constrained by the lower-level problem, whose solution, in turn, depends on the parameters of the upper-level problem. This is formally expressed by the optimization problem

$$\min_{\mathbf{x}} f(\mathbf{x}, \mathbf{y}^*(\mathbf{x})) \quad \text{s.t.} \quad \mathbf{y}^*(\mathbf{x}) \in \operatorname*{argmin}_{\mathbf{y}} g(\mathbf{x}, \mathbf{y}),$$

where f and g denote the upper and lower-level objectives respectively. This hierarchical dependency enables optimization strategies that more effectively capture complex dependencies between interrelated objectives [42]. BLO has been applied in settings such as model pruning [43], invariant risk minimization [44, 45], meta-learning [46, 47] and adversarial robustness [48]. In each case, the bilevel formulation facilitates task-aware adaptation by allowing one objective to guide the optimization of another.

Despite the appeal, solving BLO problems is challenging. The nested nature of the problem formulation leads to high computational cost and often necessitates approximations, such as unrolling gradients or using surrogate losses. These approximations, in turn, introduce potential sources of error that must be managed carefully. Consequently, most practical applications of BLO adopt problem-specific strategies, tailored to the structure and constraints of the specific problem [42]. Nevertheless, the project authors did in prior work propose a computationally feasible framework that integrated the self-supervised pretraining and downstream fine-tuning stages into a single BLO problem, named BiSSL [49].

1.2.1 The BiSSL Framework

The BiSSL framework [49] addresses the challenge of aligning pretext and downstream objectives by formulating a bilevel optimization problem in which these two stages are explicitly coupled. Specifically, the downstream task is presented in the upper-level objective, while the pretext task is represented in the lower-level problem. Chapter 4 will delve into the details, but for the sake of introduction, the mathematical formulation of BiSSL is also presented here:

$$\begin{aligned} \min_{\boldsymbol{\theta}_D,\boldsymbol{\phi}_D} \quad & \mathcal{L}^D\left(\boldsymbol{\theta}_P^*\left(\boldsymbol{\theta}_D\right),\boldsymbol{\phi}_D\right) + \mathcal{L}^D\left(\boldsymbol{\theta}_D,\boldsymbol{\phi}_D\right) \\ \text{s.t.} \quad & \boldsymbol{\theta}_P^*\left(\boldsymbol{\theta}_D\right) \in \operatorname*{argmin}_{\boldsymbol{\theta}_P} \min_{\boldsymbol{\phi}_P} \mathcal{L}^P\left(\boldsymbol{\theta}_P,\boldsymbol{\phi}_P\right) + \frac{\lambda}{2}\|\boldsymbol{\theta}_D - \boldsymbol{\theta}_P\|_2^2, \end{aligned} \tag{1.1}$$

with $\lambda \in (0,\infty)$. The objectives $\mathcal{L}^D$ and $\mathcal{L}^P$ denote the downstream fine-tuning and self-supervised pretext task training objectives respectively, with associated backbone and auxiliary head parameters $\boldsymbol{\theta}_D$, $\boldsymbol{\phi}_D$ and $\boldsymbol{\theta}_P$, $\boldsymbol{\phi}_P$, respectively. This structure models the inheritance of pretext-trained parameters into downstream training by substituting the pretext backbone parameter solution $\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D)$ directly into the downstream objective. This mutual dependency fosters a two-way interaction between pretext and downstream tasks, facilitating more nuanced information exchange that is not achievable within the conventional decoupled SSL pipeline. Operationally, BiSSL acts as an intermediate stage within the SSL training pipeline, as outlined in Figure 1.4.

Conceptually, BiSSL can be viewed as refining the backbone initialization acquired through pretext training to better align with the downstream task, thereby reducing the representational mismatch usually present prior to the fine-tuning phase. Figure 1.5 illustrates this effect, showing that BiSSL yields embeddings that are more aligned with the downstream task than those produced by conventional pretext training.

1.2.2 Knowledge Gaps in the BiSSL Framework

Despite the promising previous results, the current understanding of the BiSSL framework remains incomplete. Several aspects of its formulation, design choices, and generality warrant further investigation to better assess its robustness and applicability.

Hyperparameter Influence BiSSL introduces a number of hyperparameters, whose respective influence on downstream performance has not been systematically explored. Assessing its robustness to these settings and identifying configurations that balance accuracy and computational efficiency would yield valuable insights into the framework's general applicability.

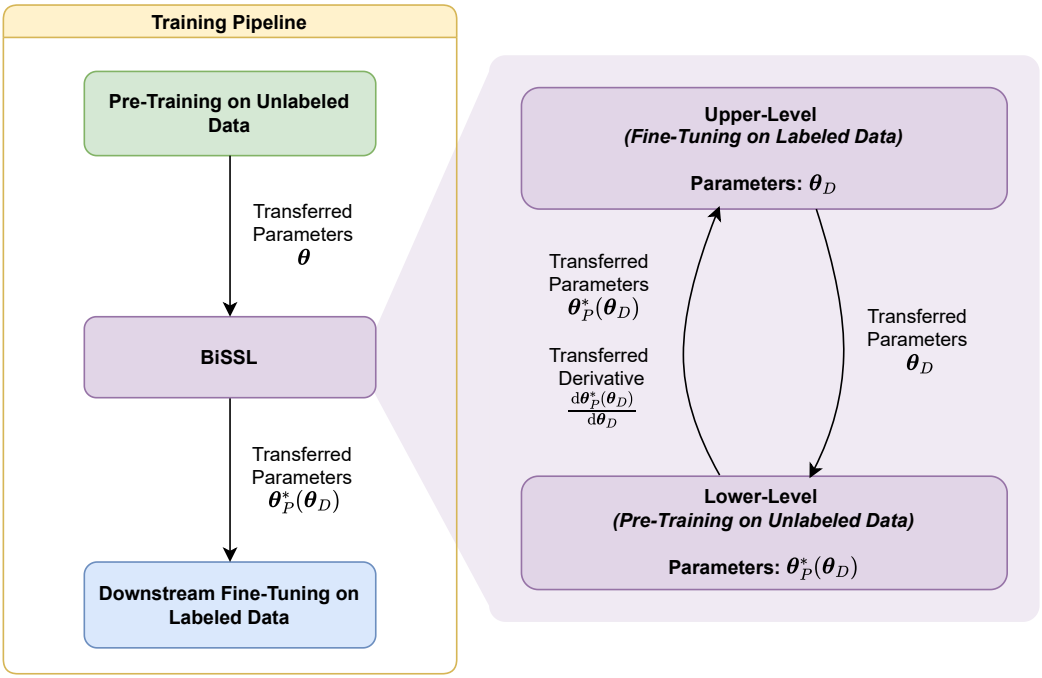

Figure 1.4: Overview of the training pipeline involving BiSSL. As seen, BiSSL introduces an intermediate training stage that solves a bilevel optimization problem, incorporating both the pretext pretraining and downstream fine-tuning training stages in order to yield a backbone $\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D)$ which is more suitable for subsequent fine-tuning. Figure 4.1 presents a more detailed illustration of the training pipeline involving BiSSL.

Design Modifications Several aspects of the overall training framework design offer opportunities for refinement that could lead to further improvements. Notably, the current version assumes the pretext head parameters $\boldsymbol{\phi}_P$ to be fixed during the upper-level optimization, primarily for simplicity in mathematical formulation and computational efficiency. However, it remains to be investigated how relaxing this assumption affects the mathematical expressions of the upper-level gradients, and consequently, whether it benefits downstream task performance in practice. Another potential generalization of the current framework is to relax the assumption that the lower-level regularization weight λ (see (1.1)) is fixed, and instead allow it to vary during training. As it will be discussed in Chapter 4, this weight plays a critical role in balancing the influence of the two levels, and allowing it to adapt dynamically during training may lead to further downstream improvements. This could be achieved through a simple scheduling scheme or by integrating it into the bilevel optimization problem itself, allowing the upper level to update it during training.

Adaptation to Natural Language Processing Tasks Although BiSSL is presented as a model and domain-agnostic framework, its empirical evaluation has so far been limited to natural image tasks. This raises the question of whether BiSSL can generalize as effectively to other modalities, and whether modifications are needed to enable such generalization. A compelling test case is NLP, where data structures, optimization regimes, and model architectures often differ from those in vision. Evaluating BiSSL on NLP benchmarks would therefore be a critical step in assessing its claimed generality.

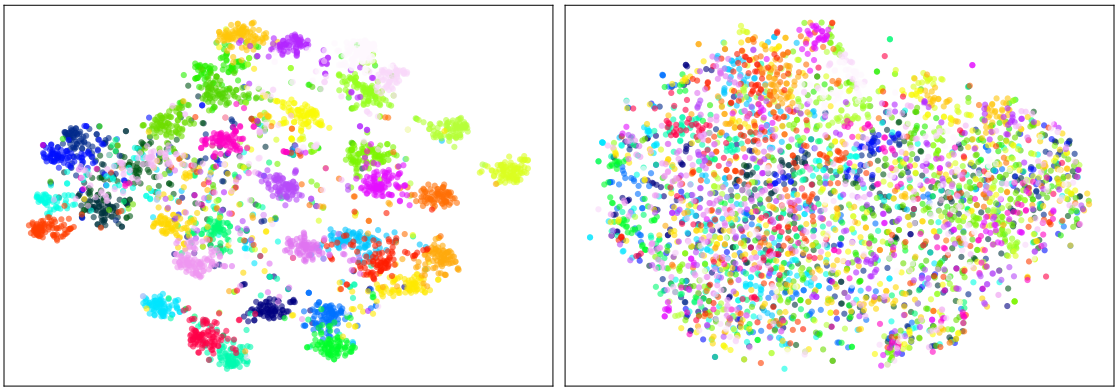

Figure 1.5: Dimensionality-reduced downstream feature representations from model backbones obtained after respectively applying BiSSL (left) and conventional self-supervised pretraining (right). Each color represents a different class. The BiSSL backbone features clearly align better with the downstream data [49].

1.3 Problem Statement

Based on the introduction made in this chapter, the overall aim of this project is to close the knowledge gaps in Section 1.2.2 through extended evaluation and design modifications of BiSSL. This is achieved by addressing the following sub-problems:

- *How do varying hyperparameter configurations and scheduling of the lower-level backbone regularization weighting in BiSSL affect downstream performance?*
- *How can adaptive lower-level backbone regularization weighting and tighter coupling of the pretext head into the upper-level objective be mathematically formulated and integrated into the BiSSL framework, and how do these modifications affect downstream performance?*
- *How can BiSSL be adapted to text-specific pretext and downstream tasks, and to what extent can it improve downstream performance?*

1.4 Project Delimitations

- This project will not provide an overview of common approaches to solve bilevel optimization in machine learning; instead, it will primarily focus on the implicit function method along with the conjugate gradient method, which will be utilized in the experiments. For an overview of the broader utility of solving bilevel optimization in machine learning, we refer to [42].
- Detailed exploration of the mechanics of pretext task design is beyond the scope of this project, with the exception of SimCLR [19] and GPT [30], which will be employed in experiments. For further background on commonly used pretext tasks in self-supervised learning, see [12, 15].
- Natural language processing tasks are limited to pretraining with GPT on a subset of the GLUE benchmark [50].

- Neural architecture design is not covered in detail. Standard, off-the-shelf architectures [4, 26] will be used.

The remainder of the project is structured as follows: Chapter 2 provides further insight into self-supervised learning, with a focus on the SimCLR and GPT pretext tasks. Chapter 3 introduces bilevel optimization and presents key results that form the basis for the BiSSL framework, which is introduced in Chapter 4. Chapter 5 then revisits BiSSL to establish the foundation for addressing the knowledge gaps identified back in Section 1.2.2. These considerations are then empirically evaluated in Chapters 6 and 7. Finally, Chapter 8 presents a discussion of the results, followed by conclusions and directions for future work in Chapter 9.

2 | Self-Supervised Learning

This chapter builds on the introduction to self-supervised learning (SSL) in Section 1.1 by exploring its underlying foundations. First, a clarification of how SSL differs from related learning paradigms is presented, followed by a detailed examination of the SimCLR and GPT pretext tasks.

2.1 Distinction from Related Learning Paradigms

To get a clearer interpretation of what classifies self-supervised learning, this section contextualizes SSL within the broader machine learning landscape by relating it to adjacent learning paradigms.

2.1.1 Supervised and Unsupervised Learning

Machine learning algorithms have traditionally been categorized as either supervised or unsupervised. The distinction, in its simplest form, lies in the presence or absence of labeled data. Supervised learning algorithms are trained on labeled datasets, where each input is paired with a corresponding target label that the model aims to predict. In contrast, unsupervised learning operates without such labels, instead seeking to uncover inherent structure in the input data, often through techniques such as clustering or dimensionality reduction [51].

This raises the question: Should SSL be regarded as supervised or unsupervised? A deliberately ambiguous answer may be: *yes*. The takeaway is that it depends largely on one's perspective.

On the one hand, SSL clearly fits within the unsupervised paradigm in that it relies solely on unlabeled data. On the other hand, SSL involves training models in a supervised fashion, as it relies on defined labels and solving (pretext) task-specific objectives. The critical nuance is that these labels are not externally provided but are instead derived from the data itself. For instance, in predictive pretext tasks, such as next-step prediction in sequential data, the "label" is simply the subsequent element in the sequence. Another example is the image-specific pretext task of RotationNet [52], where images are randomly rotated and the model is trained to predict the degree of rotation. In this case, the rotation angle serves as the label, automatically generated as part of the data transformation.

Thus, self-supervised learning can be understood as a form of unsupervised learning that leverages supervised training principles. It proceeds as if it were supervised, but without requiring any human-labeled data.

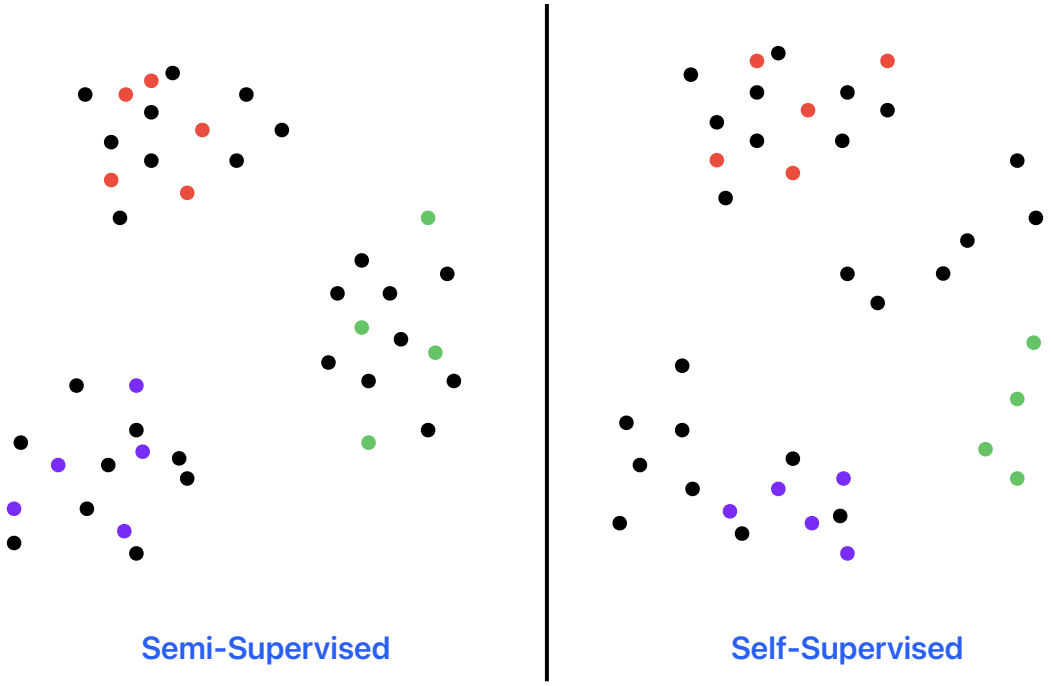

Figure 2.1: Illustrative examples of data conventionally suitable for semi-supervised (left) and self-supervised (right) learning. The black dots represent unlabeled data points, while colored dots represent labeled data, with each color corresponding to a distinct class. In the semi-supervised setting, all the unlabeled points are assumed to underlyingly belong to one of the same classes represented in the labeled data. This is typically not a requirement for the self-supervised case, as illustrated by some points not meaningfully belonging to any of the downstream classes.

2.1.2 Semi-Supervised Learning

To further complicate the taxonomy of learning paradigms, *semi-supervised learning* introduces yet another variant that shares notable similarities with SSL. Semi-supervised learning assumes access to an incompletely labeled dataset. That is, a dataset in which only a (typically tiny) subset of the instances have associated labels, while the rest remain unlabeled. This implies that both the labeled and unlabeled portions of the data are assumed to be available simultaneously, and, crucially, that the unlabeled data is expected to be sampled from the same marginal input distribution as the labeled data. By leveraging the known labels, semi-supervised learning trains models to generalize across the unlabeled examples, often through techniques such as consistency regularization or pseudo-labeling. The intuition is that, as long as the unlabeled data is semantically aligned with the labeled portion (i.e., not consisting of unrelated categories such as, e.g., pictures of cars when the labeled dataset only consists of pictures of cats and dogs), its structure can help refine decision boundaries, thereby improving performance beyond what could be achieved using the labeled subset alone [53]. The left side of Figure 2.1 illustrates an example of a simple dataset adhering to the semi-supervised learning assumptions.

In contrast, the SSL training pipeline is separated into two distinct stages as previously

illustrated in Figure 1.1: a pretraining phase on a large unlabeled dataset solving the pretext task, followed by a separate fine-tuning phase on a (usually much smaller) labeled downstream dataset. Another distinction is that SSL does not require the unlabeled pretraining data to come from the same distribution as the labeled downstream data. In fact, it often does not [15]. This distinction is made on the right side of Figure 2.1, which shows that some parts of the pretraining data may align well with the downstream data (e.g., the red colored classes), while other parts may not at all align or only align partially with the downstream classes. A representative practical example, which will be revisited in the experiments of Chapter 6, is pretraining a model on the general-purpose ImageNet dataset (decapitated of labels) [2], which contains over 1.2 million images across a diverse range of natural images, and subsequently fine-tuning it on a narrower dataset such as Oxford-IIIT Pets [54], which focuses on fine-grained image classification of cat and dog breeds. In this case, the pretraining data distribution extends beyond that of the downstream task, while some species present in the downstream dataset may also not appear in the pretraining data.

SSL pretext tasks thus enable learning general-purpose feature representations from a large-scale, heterogeneous, unlabeled dataset that are transferable across a multitude of various downstream tasks. With these distinctions in place, we now turn to a concrete type of pretext task to more closely examine how such general features can be achieved.

2.2 SimCLR: Contrastive Learning of Visual Representations

This section is based on [19] unless otherwise stated. The core focus will primarily lie on image-based examples to maintain clarity of exposition. However, the core principles of SimCLR are domain-agnostic and can be extended to any setting where meaningful data augmentations can be defined.

SimCLR trains a model to pull together different augmented views of the same image and push apart views from different images. To succeed, the model must learn representations that are invariant to superficial variations and capture the underlying semantic structure of the input, which ideally are beneficial for downstream tasks. To create these distinct views, stochastic data augmentations are applied to the input.

2.2.1 Data Augmentations

A central component in contrastive learning frameworks like SimCLR is the use of data augmentations to generate multiple distinct views of the same input image. These augmentations introduce variability while preserving the underlying semantic content, enabling models to learn representations that are invariant to such transformations. Commonly used image augmentations in contrastive learning include the following [23]:

- **Random Resized Crop:** Randomly crops a portion of the image and resizes it to a fixed size.
- **Horizontal Flip:** Flips the image horizontally.
- **Color Jittering:** Randomly alters brightness, contrast, saturation, and hue.

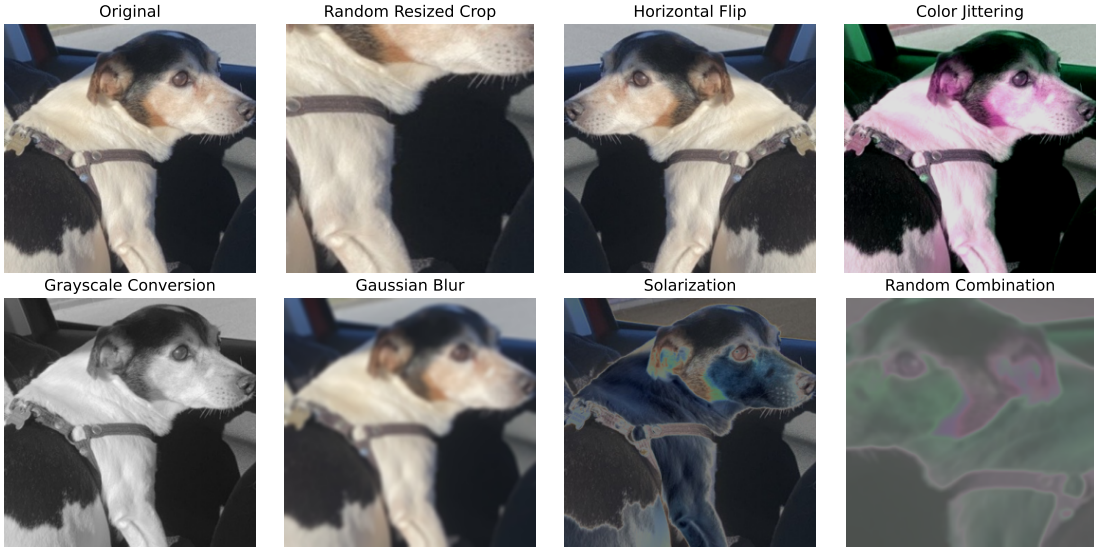

Figure 2.2: Example of augmentations. From the top left to the bottom right: Original image, random resized crop, horizontal flip, color jittering, grayscale conversion, gaussian blur, solarization, and lastly a random combination of the former augmentations.

- **Grayscale Conversion:** Converts the image to grayscale.
- **Gaussian Blur:** Applies a blur effect on the image.
- **Solarization:** Inverts all pixel values above a certain threshold.

These transformations are typically applied stochastically, meaning that each augmentation is applied sequentially to each view with a predefined probability. This stochasticity ensures that the model is exposed to a diverse set of input variations over the course of training. Figure 2.2 shows example augmentations applied to a single image. While the resulting views may differ substantially at the pixel level, they remain semantically equivalent.

2.2.2 Network Architecture and Projection Mechanism

SimCLR processes each augmented view of an input through a shared backbone encoder to extract intermediate feature representations. While SimCLR does not impose architectural constraints on the backbone, convolutional networks such as ResNets [4] have shown strong empirical performance. The resulting backbone output vectors are then passed through a separate pretext head network, commonly referred to as the projection head in this context, and is typically implemented as a multilayer perceptron (MLP) [55]. This projection head maps features into a latent space wherein the contrastive loss is applied. The projection head serves to decouple the task-specific contrastive objective from the backbone, encouraging the backbone to learn general-purpose features. After pretraining, the projection head is discarded, and only the backbone is retained for transfer to downstream tasks.

Figure 2.3 illustrates the overall framework with example pictures. The augmented views are processed through the encoder and projection head to generate embeddings used in the contrastive objective, which is discussed in the following subsection.

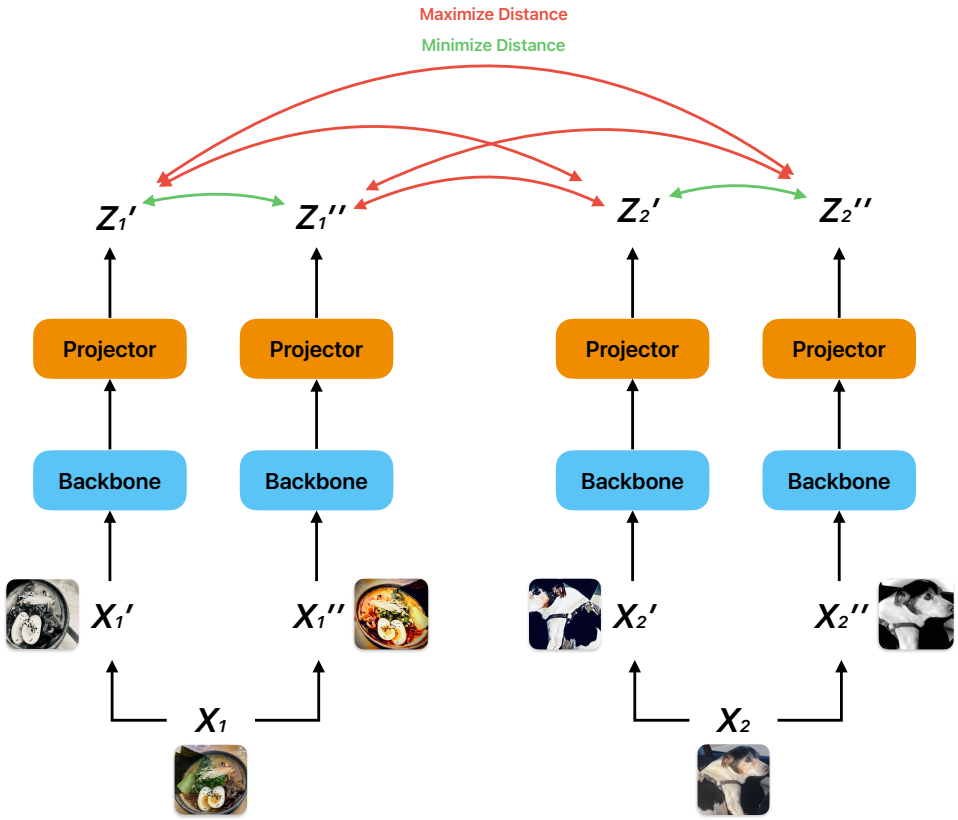

Figure 2.3: Overview of the SimCLR architecture. Two inputs $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ are each augmented twice to produce views $\boldsymbol{X}_1'$, $\boldsymbol{X}_1''$, and $\boldsymbol{X}_2'$, $\boldsymbol{X}_2''$. All views are then passed through a shared backbone encoder followed by a shared projection head to produce projected views $\boldsymbol{Z}_1'$, $\boldsymbol{Z}_1''$, $\boldsymbol{Z}_2'$, and $\boldsymbol{Z}_2''$. In the projection space, the contrastive loss encourages similarity between projected positive pairs (e.g., $\boldsymbol{Z}_1'$, $\boldsymbol{Z}_1''$) while pushing apart representations of projected negative pairs (e.g., $\boldsymbol{Z}_1'$, $\boldsymbol{Z}_2'$). After training, only the backbone is retained and used for downstream tasks.

2.2.3 The NT-Xent Loss Function

SimCLR's training objective is built on a contrastive loss that pulls together representations of augmented views from the same image (positive pairs) and pushes apart those from different images (negative pairs). This is formalized by the *normalized temperature-scaled cross entropy loss* (NT-Xent).

Given a batch of N inputs, let $\mathbf{x}_1, \ldots, \boldsymbol{x}_{2N}$ denote the resulting augmented views such that $\mathbf{x}_{2i-1}$ and $\mathbf{x}_{2i}$ are positive pairs for $i = 1, \ldots, N$. Let $f_{\boldsymbol{\theta}} : \mathbb{R}^q \mapsto \mathbb{R}^d$ denote the model backbone with trainable parameters $\boldsymbol{\theta}$ and $p_{\boldsymbol{\phi}} : \mathbb{R}^d \mapsto \mathbb{R}^p$ the projection head with trainable parameters $\boldsymbol{\phi}$. Then, the projected representations are achieved by

$$\boldsymbol{z}_j := (f_{\boldsymbol{\theta}} \circ p_{\boldsymbol{\phi}})(\boldsymbol{x}_j), \quad j = 1, \ldots, 2N.$$

For a positive pair with indexes i and j, the NT-Xent loss is then defined as:

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_k)/\tau)}, \tag{2.1}$$

where $\tau > 0$ is refereed to as the *temperature parameter*, $\mathbb{1}_{[k \neq i]}$ is an indicator function that excludes the anchor view i from the denominator and

$$\text{sim}(\mathbf{x},\mathbf{y}) = \frac{\mathbf{x}^\intercal \mathbf{y}}{\|\mathbf{x}\|_2 \|\mathbf{y}\|_2}$$

is the cosine similarity. The expression (2.1) decreases when the considered positive pair $\mathbf{z}_i$ and $\mathbf{z}_j$ attracts in the latent space (the nominator), while increasing their distances to the remaining considered negative pairs (the denominator) as well. The cosine similarity ensures that representations are compared by their direction rather than magnitude, which avoids scale sensitivity and has been shown to improve the stability and generalizability of learned representations in contrastive learning settings [56].

The temperature parameter τ in (2.1) modulates the sensitivity of the exponential function applied to cosine similarities. Smaller values amplify differences in cosine similarities, making the loss focus more on hard negatives. This can improve feature discrimination but increases the risk of instability or representational collapse. Larger values smooth the distribution, which stabilizes training but may weaken the separation between positive and negative pairs, leading to underfitting. Choosing an appropriate τ is thus critical for balancing stability and discriminative power. Following the original work [19], $\tau = 0.5$ is used in the experiments of Chapter 6.

SimCLR evaluates each of the N positive pairs in both directions, yielding the pretext training objective to be

$$\mathcal{L}^{\text{SimCLR}} = \frac{1}{2N} \sum_{k=1}^{N} (\ell_{2k-1,2k} + \ell_{2k,2k-1}).$$

This formulation generally benefits from large batch sizes, as that provides a greater number of negative samples and thus a stronger contrastive signal.

Under some circumstances, training with large batches poses practical challenges in terms of memory and computational requirements, which has prompted subsequent research into alternative approaches. While it is outside the scope of this project to examine further contrastive pretext tasks, Appendix A outlines a selection of related pretext tasks that, while sharing core principles with SimCLR, differ in architecture, training objectives, and sampling strategies to enhance representation learning and to some extent alleviate the need for large batch sizes.

While contrastive methods like SimCLR have been extended to text (e.g., SimCSE [57]), pretraining in NLP more commonly follows alternative paradigms. We now turn to a specific approach in the context of large-scale generative pretraining.

2.3 GPT: Generative Pretrained Transformer

In recent years, large language models (LLMs) have seen widespread adoption and have driven a remarkable surge in research across natural language processing (NLP) [58]. Much of this progress can be attributed to the development of highly effective self-supervised pretraining techniques. Among the earliest and most influential of these frameworks is the GPT (Generative Pretrained Transformer) [30]. Although later versions such as GPT-2, GPT-3, and GPT-4 [31–33] have significantly increased in both scale and capability, the underlying principles remain largely unchanged. These newer models primarily differ in the

size of the architecture and the volume of training data, rather than in their fundamental design. For educational clarity and computational feasibility in the experiments of Chapter 7, the focus in this project will be on the original GPT.

First, GPT's input processing pipeline is described, including tokenization and embedding strategies. Then, it is detailed how these inputs are transformed by the model to produce an output distribution. Finally, the pretraining objective is presented, and it is discussed how GPT is adapted to downstream tasks via fine-tuning. This section is based on [30], unless otherwise stated.

2.3.1 Tokenization

Unlike image data, which can often be fed directly into neural networks with minimal preprocessing, textual data must first be converted into a compatible format. This begins with *tokenization*, the process of dividing raw text into discrete units called *tokens*, each of which can be mapped to a unique integer ID. Naive approaches such as character-level tokenization (where each character is assigned its own token) or word-level tokenization (where each word corresponds to a token) tend to be either too granular or too brittle. Character-level tokenization leads to long sequences and may struggle to capture meaning across characters, while word-level tokenization may suffer from vocabulary explosion and poor generalization to rare or unseen words.

To address this, GPT instead employs a variant of *byte pair encoding* (BPE) [59], which is a subword-level tokenization method. BPE begins with a base vocabulary consisting of all individual characters observed in a corpus. It then iteratively merges the most frequent adjacent symbol pairs into new symbols, which typically consist of character or subword pairs. This merging process continues until a predefined vocabulary size $V \in \mathbb{N}$ is reached. This approach has several advantages: frequent words or morphemes are represented as single tokens, improving efficiency, while rare or unseen words are decomposed into meaningful subunits, supporting enhanced generalization. As a result, BPE achieves a favorable trade-off between vocabulary size and expressive power.

Example: Consider the sentence:

```
"Corgis are objectively the best dog breed."
```

A word-level tokenizer might struggle with rare or out-of-vocabulary words like `corgis` and extended words like `objectively`, potentially treating them as unknown and breaking them into individual characters. In contrast, the Byte Pair Encoding (BPE) tokenizer used by GPT splits the sentence into subword units as follows:

```
[ "cor", "gis</w>", "are</w>", "objec", "tively</w>", "the</w>",
"best</w>", "dog</w>", "breed</w>" ".</w>"]
```

The special suffix `"</w>"` marks the end of a word, allowing word-level structure to be preserved during subword tokenization. This subword-based representation allows the model to reuse frequent fragments such as `"gis"` or `"tively"` across many contexts. Finally, the tokens are collected into a vector of the corresponding assigned IDs for each token, which in this case would yield the vector

$$[1055,\ 31509,\ 640,\ 7981,\ 4508,\ 481,\ 1432,\ 2585,\ 10699,\ 239]^\top.$$

2.3.2 Token and Position Embeddings

Notation: For a matrix A, the entity $(A)_i$ denotes the i'th *column vector* of A. For a vector $\mathbf{v}$, $(\mathbf{v})_i$ denotes its i'th entry.

During self-supervised pretraining, GPT processes text using a fixed-length *context window* of size $k \in \mathbb{N}$, i.e., sequences of exactly k tokens. Given a tokenized input sequence $x_1, x_2, \ldots, x_k$, where each $x_i \in \{1, \ldots, V\}$ corresponds to an index in the tokenizer vocabulary of size V, the model first maps each token to a continuous vector representation using a learned *token embedding matrix* $W_e \in \mathbb{R}^{V \times d}$, where $d \in \mathbb{N}$ is the embedding dimension.

Let the input sequence be collected into the vector $\mathbf{x} = [x_1, \ldots, x_k]^\top$. Before applying the embedding, we construct a one-hot representation of $\mathbf{x}$ using the definition below.

Definition 2.1 (One-Hot Token Mapping X)
Let a vocabulary size $V \in N$ and context window size $k \in N$ be given, and let $\mathbf{e}_j \in \{0,1\}^V$ denote the unit vector assuming the value 1 at its j'th entry and zeros elsewhere. Then the *one-hot token mapping* $X : \{1, \ldots, V\}^k \mapsto \{0,1\}^{k \times V}$ is defined such that the rows of its output are given by

$$(X(\mathbf{x})^\top)_i = \mathbf{e}_{x_i}, \quad i = 1, \ldots, k$$

for any $\mathbf{x} \in \{1, \ldots, V\}^k$.

For example, if $x_3 = 86$, then $(X(\mathbf{x})^\top)_3 = \mathbf{e}_{86}$. We then achieve the token embeddings of the entire sequence $\mathbf{x}$ through the matrix matrix product $X(\mathbf{x})W_e$.

To capture the ordering of the input, the token embeddings are further augmented with *positional embeddings*. These are learned vectors stored in a separate matrix $W_p \in \mathbb{R}^{k \times d}$, where each row $(W_p^\top)_i$ encodes the position i in the input sequence. Summing the token and positional embeddings yields the output of the first layer of GPT:

$$H_0(\mathbf{x}) = X(\mathbf{x})W_e + W_p.$$

2.3.3 Transformer Blocks and Output Layer

Given the matrix $H_0(\mathbf{x})$, it is passed through a stack of L architecturally identical transformer decoder blocks using masked multi-head attention [26, 60], denoted

$$H_l(\mathbf{x}) = \text{TransformerBlock}_l^{d,h}(H_{l-1}(\mathbf{x})) \in \mathbb{R}^{k \times d}, \quad \text{for } l = 1, \ldots, L,$$

where d is the embedding dimension and h the number of *attention heads* inside the transformer block. While delving into the details of the internal transformer architecture is beyond the scope of this project, Appendix B provides an overview of the architectural components specifically used in GPT, and Table B.1 lists the specific hyperparameter values used in the original GPT implementation, which we adopt in Chapter 7.

After the final transformer block produces $H_L(\mathbf{x})$, GPT computes its output via

$$H_{\text{out}}(\mathbf{x}) = \text{softmax}(H_L(\mathbf{x})W_e^\top), \tag{2.2}$$

where the embedding matrix W_e is reused for the output projection, which practically reduces the total number of learnable parameters. The softmax is applied row-wise, which

is further detailed in Equation (B.2) of Appendix B, mapping each row of $H_L(\mathbf{x})W_e^\top$ to a probability distribution over the tokenizer vocabulary. Hence, each row $(H_{\text{out}}(\mathbf{x})^\top)_i$ can be interpreted as a categorical distribution over possible tokens for the token x_i in $\mathbf{x}$.

Regarding the Pretext Head Unlike SimCLR in Section 2.2, which uses a dedicated pretext head (e.g., a projection MLP used only during pretraining), GPT does not separate its backbone and head architectures in this way. While the output layer (2.2) may be replaced for subsequent supervised fine-tuning (see Section 2.3.5), all parameters used during pretraining are part of the core backbone model and are typically retained. In this sense, GPT's output layer H_{out} serves as a pretext head with a fixed architecture but no additional trainable parameters.

2.3.4 Pretext Task Objective: Autoregressive Language Modeling

The self-supervised pretraining procedure of GPT is to solve the pretext task of *autoregressive language modeling*, where the model is trained to predict the next token in a sequence, given all previous ones. Specifically, the model is tasked with predicting x_t conditioned on the preceding sequence $x_1,\ldots,x_{t-1}$ without access to any future tokens. Formally, given a training corpus represented as a sequence of tokens $x_1,\ldots,x_T$, the ultimate goal is to maximize the log-likelihood:

$$\sum_{t=1}^{T} \log P(x_t \mid x_{t-k},\ldots,x_{t-1}),$$

where $k \in \mathbb{N}$ is the context window size and $P(x_t \mid x_{t-k},\ldots,x_{t-1})$ is the probability assigned to the correct token by the model.

As outlined in the previous section, the respective rows of the model output $H_{\text{out}}(\mathbf{x})$ as defined in (2.2) yield probability distributions over the tokenizer vocabulary for each token in the input sequence $\mathbf{x}$. Hence, while $H_{\text{out}}(\mathbf{x})$ can always be interpreted as a collection of arbitrary categorical distributions, the pretext task explicitly shapes this output to model the conditional probability of the next token at each position during pretraining. Let $\tilde{\mathbf{x}} = [x_2,\ldots,x_{k+1}]$ denote the right shifted sequence of a target sequence $\mathbf{x} = [x_1,\ldots,x_k]^\top$. The pretext task of GPT is to model the conditional distribution $P(\tilde{\mathbf{x}} \mid \mathbf{x})$ by minimizing the loss between predicted and true next-token distributions:

$$\mathcal{L}^{\text{GPT}}(\mathcal{B}) = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x},\tilde{\mathbf{x}}\in\mathcal{B}} \mathcal{L}^{\text{CE}}(X(\tilde{\mathbf{x}}), H_{\text{out}}(\mathbf{x})),$$

where $\mathcal{B}$ is a batch of length-k subsequence pairs from the training corpus $x_1,\ldots,x_T$ as described above, and $\mathcal{L}^{\text{CE}}$ denotes the cross-entropy loss calculated for each position in the respective matrix inputs.

2.3.5 Fine-Tuning

Following self-supervised pretraining, GPT can be fine-tuned towards a variety of downstream NLP tasks. For ease of explanation, this section will mainly focus on classification tasks. Let the downstream labeled dataset be given by the following sequence $(x_{1,1},\ldots,x_{1,q_1},y_1),\ldots,(x_{n,1},\ldots,x_{n,q_n},y_n)$, where $y_i \in \{1,\ldots,C\}$ is the class label, $q_i \in \mathbb{N}$ is the input sequence length for $i \in \{1,\ldots,n\}$ and n is the number of available sequences.

The corresponding output of the final transformer block for the i'th sequence $H_L(\mathbf{x}_i)$, $\mathbf{x}_i = [x_{i,1}, \ldots, x_{i,q_i}]^\top$ will accordingly have dimension $q_i \times d$.

The common strategy is to apply a task-specific learnable linear layer $W_d \in \mathbb{R}^{d \times C}$ to the hidden representation of $H_L(\mathbf{x})$ corresponding to the final token of the input sequence, in order to achieve a probability distribution over the downstream classes:

$$H_{\text{cls}}(\mathbf{x}_i) = \text{softmax}((H_L(\mathbf{x}_i)^\top)_{q_i}^\top W_d), \quad i = 1, \ldots, n,$$

where $(H_L(\mathbf{x})^\top)_{q_i}^\top$ then is the row of $H_L(\mathbf{x}_i)$ corresponding to the last token in the input sequence $\mathbf{x}_i$. To then model the distribution $P(y|x_1, \ldots, x_q)$, we minimize the cross-entropy loss

$$\mathcal{L}^{\text{task}} = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}^{\text{CE}}(\mathbf{e}_{y_i}, H_{\text{cls}}(\mathbf{x}_i)),$$

where $\mathbf{e}_{y_i}$ is a unit vector representing the one-hot encoded target class.

The supervised loss $\mathcal{L}^{\text{task}}$ is additionally appended with the original language modeling loss on the downstream corpus, as this has been shown to improve generalization as well as to accelerate convergence during fine-tuning. The full fine-tuning loss is then

$$\mathcal{L}^{\text{GPT-FT}} = \mathcal{L}^{\text{task}} + \nu \mathcal{L}^{\text{GPT}}, \tag{2.3}$$

where $\nu \in (0, \infty)$ is a weighting hyperparameter that controls the influence of the language modeling loss.

Sequence Control Tokens Since token sequences for downstream tasks vary in structure and length, special control tokens are introduced to delineate different parts of the input and mark boundaries such as start, end, or segment separation. Specifically, the following tokens are introduced during fine-tuning:

- `<bos>`: beginning-of-sequence token
- `<eos>`: end-of-sequence token
- `<sep>`: delimiter for separating marked segments of an input sequence, e.g., questions and answers

These tokens are embedded analogously to regular tokens. For each input sequence, the `<bos>` and `<eos>` tokens are prepended and appended, respectively. If the original vocabulary has size V, and V_D new special tokens are added during downstream adaptation, then the pretrained embedding matrix is correspondingly extended with V_D new rows, yielding the updated embedding matrix $W_e^D \in \mathbb{R}^{(V+V_D) \times d}$ during fine-tuning. The new rows are typically randomly initialized and learned jointly during fine-tuning.

3 | Bilevel Optimization

This chapter introduces the theoretical foundations of bilevel optimization along with practical considerations regarding the implementation of bilevel optimization in machine learning learning algorithms. This chapter is based on [42] unless otherwise stated.

3.1 Optimization Problem Formulation

Bilevel optimization (BLO) is a certain type of constrained optimization problem, where the constraint itself involves the solution to another optimization problem. Let f,g : $\mathbb{R}^N \times \mathbb{R}^M \mapsto \mathbb{R}$ be differentiable functions. The BLO problem is then formulated as

$$\min_{\mathbf{x}\in\mathbb{R}^N} f(\mathbf{x},\mathbf{y}^*(\mathbf{x})) \quad \text{s.t.} \quad \mathbf{y}^*(\mathbf{x}) \in \operatorname*{argmin}_{\mathbf{y}\in\mathbb{R}^M} g(\mathbf{x},\mathbf{y}), \tag{3.1}$$

where f and g are referred to as the *upper-level* and *lower-level* objectives respectively. Similarly, $\mathbf{x}$ and $\mathbf{y}$ are referred to as the upper-level and lower-level parameters or variables. While the lower-level objective g has knowledge of the parameters from the upper-level objective $\mathbf{x}$, the upper-level objective f possesses information of the lower-level objective g itself through its dependence on a solution of the lower-level problem $\mathbf{y}^*(\mathbf{x})$. Having this coupled setup of training objectives can be attractive in instances that involve solving multiple co-dependent optimization problems at once, as BLO problems incorporate this dependency.

3.2 Obtaining the Derivatives

While the optimization problem formulation (3.1) promises solutions that involve a complex interaction between the objectives, it remains ambiguous how to practically solve this optimization problem. Although often dependent on the specific application, designing training algorithms for solving the BLO problem through machine learning typically involves using gradient optimizers. This necessitates obtaining expressions for the gradients of both the lower and upper-level objectives that accurately reflect their interdependence as depicted in the BLO problem (3.1).

The gradient of the lower-level objective is straightforward to calculate, as it only depends on the upper-level objective through its parameters. However, obtaining a practical expression for the derivative of the upper-level objective is more intricate. This is due to the fact that the upper-level depends on an implicitly defined function $\mathbf{y}^*(\mathbf{x})$. To

clarify, consider the total derivative of the upper-level objective:

$$\frac{\mathrm{d}f}{\mathrm{d}\mathbf{x}} = \nabla_{\hat{\mathbf{x}}} f(\hat{\mathbf{x}}, \mathbf{y}^*(\mathbf{x}))|_{\hat{\mathbf{x}}=\mathbf{x}} + \underbrace{\frac{\mathrm{d}\mathbf{y}^*(\mathbf{x})}{\mathrm{d}\mathbf{x}}}_{\mathrm{IJ}} \nabla_{\mathbf{y}} f(\mathbf{x}, \mathbf{y})|_{\mathbf{y}=\mathbf{y}^*(\mathbf{x})}. \tag{3.2}$$

The above expression (3.2) involves the Jacobian matrix $\frac{\mathrm{d}\mathbf{y}^*(\mathbf{x})}{\mathrm{d}\mathbf{x}}$, referred to as the *implicit Jacobian* (IJ)[1]. The current setup and assumptions in (3.1) do not guarantee the existence of an explicit expression for the IJ, which is essential for calculating the upper-level derivative in practice. Moving forward, this issue can be approached from various perspectives. One common approach is gradient unrolling, which substitutes $\mathbf{y}^*(\mathbf{x})$ with the operational lower-level optimization path. While this method enables an explicit expression of the IJ, its computational complexity escalates as the number of lower-level optimization steps increases. This renders gradient unrolling unsuitable for BiSSL, described in Chapter 4, as the lower-level objective is already computationally expensive and requires a relatively large number of iterations. This approach is therefore not further explored. Instead, the implicit function method is adopted as a more practical and efficient alternative.

3.2.1 The Implicit Function Method

The implicit function method offers an alternative to gradient unrolling by instead deriving an explicit expression for the IJ, through use of the implicit function theorem [61, 62]. To achieve this, certain assumptions about the structure and properties of the lower-level objective are required. As part of this project, we introduce the following definition of the *lower-level stationary set*, which will serve as a useful tool in the derivation that follows.

Definition 3.1 (Lower-level Stationary Set)
Under the conditions of (3.1), the *lower-level stationary set* $\mathcal{G}_0$ is defined as

$$\mathcal{G}_0 = \left\{ \mathbf{x} \in \mathbb{R}^N | \exists\, \hat{\mathbf{y}}_{\mathbf{x}} \in \mathbb{R}^M : \nabla_{\mathbf{y}} g(\mathbf{x},\mathbf{y})|_{\mathbf{y}=\hat{\mathbf{y}}_{\mathbf{x}}} = \mathbf{0} \right\}.$$

In other words, the lower-level stationary set $\mathcal{G}_0$ consists of those $\mathbf{x} \in \mathbb{R}^N$ where there exists at least one corresponding stationary point $\hat{\mathbf{y}}_{\mathbf{x}} \in \mathbb{R}^M$ that causes the gradient of the lower-level to be zero when evaluated in $(\mathbf{x}, \hat{\mathbf{y}}_{\mathbf{x}})$. This construct will be used to formally state and prove the theorem below, which identifies conditions under which the IJ can be expressed in closed form.

[1] Some bilevel optimization literature refers to this quantity as the *implicit gradient*. However, this terminology can be misleading, as the IJ in this context is only an actual gradient when $M = 1$. Therefore, we instead adopt the term "implicit Jacobian" throughout this project for enhanced clarity and rigor.

Theorem 3.2 (Implicit Jacobian in Bilevel Optimization)
Under the conditions of (3.1) assume that $\nabla_{\mathbf{y}} g(\mathbf{x},\mathbf{y})$ is twice differentiable. Let $\mathbf{x} \in \mathcal{G}_0$ and $\hat{\mathbf{y}}_{\mathbf{x}} \in \mathbb{R}^M$ be given such that $\nabla_{\mathbf{y}} g(\mathbf{x},\mathbf{y})|_{\mathbf{y}=\hat{\mathbf{y}}_{\mathbf{x}}} = \mathbf{0}$ and assume that the hessian matrix $\nabla^2_{\mathbf{y}} g(\mathbf{x},\mathbf{y})|_{\mathbf{y}=\hat{\mathbf{y}}_{\mathbf{x}}}$ is invertible. Then there exists a unique implicit and **differentiable** function $\mathbf{y}^* : \mathcal{N}(\mathbf{x}) \mapsto \mathbb{R}^M$, with $\mathcal{N}(\mathbf{x})$ being a neighborhood of $\mathbf{x}$, such that $\mathbf{y}^*(\mathbf{x}) = \hat{\mathbf{y}}_{\mathbf{x}}$ and $\nabla_{\mathbf{y}} g(\tilde{\mathbf{x}},\mathbf{y})|_{\mathbf{y}=\mathbf{y}^*(\tilde{\mathbf{x}})} = \mathbf{0}$ for all $\tilde{\mathbf{x}} \in \mathcal{N}(\mathbf{x})$. Additionally the derivative of $\mathbf{y}^*$ is explicitly given by

$$\frac{\mathrm{d}\mathbf{y}^*(\mathbf{x})}{\mathrm{d}\mathbf{x}} = -\nabla^2_{\hat{\mathbf{x}}\mathbf{y}} g(\hat{\mathbf{x}},\mathbf{y})|_{\hat{\mathbf{x}}=\mathbf{x},\mathbf{y}=\mathbf{y}^*(\mathbf{x})} \Big[\nabla^2_{\mathbf{y}} g(\mathbf{x},\mathbf{y})|_{\mathbf{y}=\mathbf{y}^*(\mathbf{x})}\Big]^{-1}, \quad \mathbf{x} \in \mathcal{G}_0. \tag{3.3}$$

To summarize, the theorem states that for all values of $\mathbf{x}$ where there exists a value $\hat{\mathbf{y}}$ such that the stationary condition $\nabla_{\mathbf{y}} g(\mathbf{x},\mathbf{y})|_{\mathbf{y}=\hat{\mathbf{y}}} = \mathbf{0}$ is satisfied (i.e. for all $\mathbf{x} \in \mathcal{G}_0$), the IJ in (3.2) can be written explicitly by (3.3). We introduced Definition 3.1 for mathematical completeness, but it is worth noting that the lower-level objective g is often assumed to be strongly convex in its second argument in order ensure that $\mathcal{G}_0 = \mathbb{R}^N$ and that $\hat{\mathbf{y}}_{\mathbf{x}}$ is a unique minimizer of g, simplifying the lower-level constraint in (3.1) to $\mathbf{y}^*(\mathbf{x}) = \underset{\mathbf{y}\in\mathbb{R}^M}{\operatorname{argmin}}\, g(\mathbf{x},\mathbf{y})$. The convexity assumption will come in handy when introducing BiSSL in Chapter 4.

For the sake of conciseness, the notation $\nabla_{\boldsymbol{\xi}} h(\boldsymbol{\xi})|_{\boldsymbol{\xi}=\boldsymbol{\psi}} := \nabla_{\boldsymbol{\xi}} h(\boldsymbol{\psi})$ is employed for the rest of this chapter when it is clear from context which variables are differentiated with respect to.

Proof (Theorem 3.2): Let $\mathbf{x} \in \mathcal{G}_0$ be given. The existence of $\mathbf{y}^*$ then follows directly from the implicit function theorem, stated in Theorem C.1 of Appendix C. Due to the differentiability of $\mathbf{y}^*$, as well as the fact that g is assumed twice differentiable, the derivative of $\nabla_{\mathbf{y}} g(\mathbf{x}, \mathbf{y}^*(\mathbf{x})) = \mathbf{0}$ can be taken with respect to $\mathbf{x}$, i.e., the expression

$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{x}}[\nabla_{\mathbf{y}} g(\mathbf{x},\mathbf{y}^*(\mathbf{x}))] = \mathbf{0}$$

is valid. By use of the chain rule, the expression becomes

$$\nabla^2_{\hat{\mathbf{x}}\mathbf{y}} g(\mathbf{x},\mathbf{y}^*(\mathbf{x})) + \frac{\mathrm{d}\mathbf{y}^*(\mathbf{x})}{\mathrm{d}\mathbf{x}} \nabla^2_{\mathbf{y}} g(\mathbf{x}, \mathbf{y}^*(\mathbf{x})) = \mathbf{0}.$$

By leveraging the assumption that $\nabla^2_{\mathbf{y}} g(\mathbf{x}, \mathbf{y}^*(\mathbf{x}))$ is invertible, then $\frac{\mathrm{d}\mathbf{y}^*(\mathbf{x})}{\mathrm{d}\mathbf{x}}$ can be isolated, resulting in the desired expression of the implicit jacobian

$$\frac{\mathrm{d}\mathbf{y}^*(\mathbf{x})}{\mathrm{d}\mathbf{x}} = -\nabla^2_{\hat{\mathbf{x}}\mathbf{y}} g(\mathbf{x},\mathbf{y}^*(\mathbf{x})) \Big[\nabla^2_{\mathbf{y}} g(\mathbf{x}, \mathbf{y}^*(\mathbf{x}))\Big]^{-1}.$$

■

While (3.3) provides a clearer insight into how to explicitly calculate the upper-level gradient, there are still some caveats in terms of a practical setup. First of all the hessian matrices $\nabla^2_{\mathbf{x}\mathbf{y}} g(\mathbf{x},\mathbf{y}^*(\mathbf{x}))$ and $\nabla^2_{\mathbf{y}} g(\mathbf{x}, \mathbf{y}^*(\mathbf{x}))$ are infeasible to store in applications involving models containing a large number of trainable parameters. On top of this, inverting the matrix $\nabla^2_{\mathbf{y}} g(\mathbf{x}, \mathbf{y}^*(\mathbf{x}))$ is a computationally demanding task that also proves practically infeasible for very large matrices. This makes direct calculation of (3.3) infeasible in the realm of training larger deep neural networks. Therefore, the compromise is to

instead approximate the upper-level derivative using methods that are computationally and memory-wise feasible. Now, considering that the upper-level gradient (3.2) under the assumptions of Theorem 3.2 can be expressed by

$$\frac{\mathrm{d}f}{\mathrm{d}\mathbf{x}} = \nabla_{\mathbf{x}} f(\mathbf{x}, \mathbf{y}^*(\mathbf{x})) - \nabla^2_{\mathbf{xy}} g(\mathbf{x},\mathbf{y}^*(\mathbf{x}))\Big[\nabla^2_{\mathbf{y}} g(\mathbf{x}, \mathbf{y}^*(\mathbf{x}))\Big]^{-1} \nabla_{\mathbf{y}} f(\mathbf{x}, \mathbf{y}^*(\mathbf{x})),$$

it is evident that the Hessian matrices only appear in the form of a matrix-vector product. In such a setup, the conjugate gradient method can be used to approximate the inverse matrix vector product $\Big[\nabla^2_{\mathbf{y}} g(\mathbf{x}, \mathbf{y}^*(\mathbf{x}))\Big]^{-1} \nabla_{\mathbf{y}} f(\mathbf{x}, \mathbf{y}^*(\mathbf{x}))$.

Algorithm 1 The Conjugate Gradient Method

Input: Input vector $\mathbf{v}$, Matrix vector product calculator $f_M(\mathbf{z}) = M\mathbf{z}$, Number of iterations N_c.

Initialize $\widetilde{M^{-1}\mathbf{v}} \leftarrow \mathbf{0}$
Initialize $\mathbf{r} \leftarrow \mathbf{v}$ and $\mathbf{d} \leftarrow \mathbf{v}$

for $t = 0, \ldots, N_c$ **do**
    $\mathbf{r}_{\text{norm}} \leftarrow \|\mathbf{r}\|_2^2$
    $\overline{M\mathbf{d}} \leftarrow f_M(\mathbf{d})$
    $\alpha \leftarrow \mathbf{r}_{\text{norm}} / \mathbf{d}^\top \overline{M\mathbf{d}}$
    $\widetilde{M^{-1}\mathbf{v}} \leftarrow \widetilde{M^{-1}\mathbf{v}} + \alpha\mathbf{d}$
    $\mathbf{r} \leftarrow \mathbf{r} - \alpha\overline{M\mathbf{d}}$
    $\mathbf{r}_{\text{new norm}} \leftarrow \|\mathbf{r}\|_2^2$
    $\beta \leftarrow \mathbf{r}_{\text{new norm}} / \mathbf{r}_{\text{norm}}$
    $\mathbf{d} \leftarrow \mathbf{r} + \beta\mathbf{d}$
end for

Return: Inverse matrix vector product estimate $\widetilde{M^{-1}\mathbf{v}}$

The Conjugate Gradient Method

The conjugate gradient method [63, 64] is an iterative algorithm that approximates inverse matrix vector products by solving the minimization problem

$$\min_{\mathbf{z}} \frac{1}{2}\mathbf{z}^\top M\mathbf{z} - \mathbf{v}^\top \mathbf{z}, \tag{3.4}$$

which is minimized when $\mathbf{z} = M^{-1}\mathbf{v}$. From the viewpoint of approximating the second term of (3.2), the minimization problem (3.4) is

$$\min_{\mathbf{z}} \frac{1}{2}\mathbf{z}^\top \Big[\nabla^2_{\mathbf{y}} g(\mathbf{x}, \mathbf{y}^*(\mathbf{x}))\Big]\mathbf{z} - \nabla_{\mathbf{y}} f(\mathbf{x}, \mathbf{y}^*(\mathbf{x}))^\top \mathbf{z}.$$

The general conjugate gradient algorithm is described in Algorithm 1. Detailing this method further is beyond the scope of this project, and we refer to [64] for details. Notice that only the matrix vector products $M\mathbf{z}$ are necessary to run the algorithm. In the context of approximating an IJ vector product, the input vector $\mathbf{v} = \nabla_{\mathbf{y}} f(\mathbf{x}, \mathbf{y}^*(\mathbf{x}))$ and matrix vector product calculator $f_M(\mathbf{z}) = \Big[\nabla^2_{\mathbf{y}} g(\mathbf{x}, \mathbf{y}^*(\mathbf{x}))\Big]\mathbf{z}$ are used. By considering that

$$\nabla^2_{\mathbf{y}} g(\mathbf{x}, \mathbf{y}^*(\mathbf{x}))\mathbf{z} = \nabla_{\mathbf{y}}\Big[(\nabla_{\mathbf{y}} g(\mathbf{x}, \mathbf{y}^*(\mathbf{x})))^\top \mathbf{z}\Big], \tag{3.5}$$

one can first calculate the inner product, followed by calculating the outer gradient to achieve the above hessian vector product, avoiding the need to store the entire hessian at any time.

When the inverse hessian vector product $\hat{\mathbf{z}} \approx \left[\nabla^2_{\mathbf{y}} g(\mathbf{x}, \mathbf{y}^*(\mathbf{x}))\right]^{-1} \nabla_{\mathbf{y}} f(\mathbf{x}, \mathbf{y}^*(\mathbf{x}))$ have been approximated using Algorithm 1, the same strategy used in (3.5) can then be utilised for calculating the hessian vector product $-\nabla^2_{\mathbf{xy}} g(\mathbf{x},\mathbf{y}^*(\mathbf{x}))\hat{\mathbf{z}}$. This completely eliminates the need for storing any Hessian matrices at any time using the conjugate gradient method.

Formulating a training framework within a bilevel optimization problem demands careful attention to how gradients are defined and computed to support effective, task-specific optimization. For readers seeking further context on this topic before the introduction of BiSSL in the following chapter, Appendix D presents two examples from prior work illustrating how these gradients are derived.

4 | The BiSSL Framework

This chapter is based on our previous work on BiSSL [49], unless otherwise stated. First, the general bilevel optimization framework of BiSSL is introduced and justified, which is followed by derivations of the gradient expressions required for optimization. Finally, the training algorithm and pipeline for applying BiSSL in operational settings are presented.

4.1 Introducing BiSSL

Before delving into the details of the optimization problem underlying BiSSL, we first introduce some necessary notation.

4.1.1 Notation

We work within the self-supervised learning setup as introduced in Section 1.1 and Chapter 2. Accordingly, let the unlabeled and labeled datasets for pretext and downstream training be denoted as $\mathcal{D}^P = \{\mathbf{z}_k\}_{k=1}^{K_P}$ and $\mathcal{D}^D = \{\mathbf{x}_l, \mathbf{y}_l\}_{l=1}^{K_D}$ respectively with $\mathbf{z}_k, \mathbf{x}_l \in \mathbb{R}^N$. Let $f_{\boldsymbol{\theta}} : \mathbb{R}^N \mapsto \mathbb{R}^M$ denote a backbone model with trainable parameters by $\boldsymbol{\theta}$ and $p_{\boldsymbol{\phi}} : \mathbb{R}^M \mapsto \mathbb{R}^P$ a pretext head parameterized by $\boldsymbol{\phi}$. Given two models $g_{\boldsymbol{\phi}_P} \circ f_{\boldsymbol{\theta}_P} : \mathbb{R}^N \mapsto \mathbb{R}^{P_P}$ and $h_{\boldsymbol{\phi}_D} \circ f_{\boldsymbol{\theta}_D} : \mathbb{R}^N \mapsto \mathbb{R}^{P_D}$ for solving pretext and downstream tasks respectively, where $\boldsymbol{\theta}_P$,$\boldsymbol{\theta}_D \in \mathbb{R}^L$, $\boldsymbol{\phi}_P \in \mathbb{R}^{Q_P}$ and $\boldsymbol{\phi}_D \in \mathbb{R}^{Q_D}$, the pretext and downstream training objectives are denoted $\mathcal{L}^P(\boldsymbol{\theta}_P,\boldsymbol{\phi}_P;\mathcal{D}^P)$ and $\mathcal{L}^D(\boldsymbol{\theta}_D,\boldsymbol{\phi}_D;\mathcal{D}^D)$ respectively. For notational brevity, the dataset specifications are omitted, i.e., $\mathcal{L}^D(\boldsymbol{\theta}_D,\boldsymbol{\phi}_D;\mathcal{D}^D) := \mathcal{L}^D(\boldsymbol{\theta}_D,\boldsymbol{\phi}_D)$ and $\mathcal{L}^P(\boldsymbol{\theta}_P,\boldsymbol{\phi}_P;\mathcal{D}^P) := \mathcal{L}^P(\boldsymbol{\theta}_P,\boldsymbol{\phi}_P)$.

4.1.2 Optimization Problem Formulation

Recall that the conventional setup of self-supervised pretraining followed by supervised fine-tuning relies on a single backbone model with parameters $\boldsymbol{\theta}$. The pretext objective $\mathcal{L}^P(\boldsymbol{\theta},\boldsymbol{\phi}_P)$ is minimized first, yielding a solution $\boldsymbol{\theta}^*$ that serves as an initialization when subsequently minimizing the downstream task objective $\mathcal{L}^D(\boldsymbol{\theta},\boldsymbol{\phi}_D)$. At first glance, one might expect the BLO problem in Equation (3.1) to straightforwardly accommodate this structure by assigning the pretext and downstream objectives to the lower- and upper-level problems, respectively. However, directly substituting the objectives into the BLO problem introduces a key complication, as both objectives would then optimize with respect to the same set of backbone parameters $\boldsymbol{\theta}$. In a BLO framework, the lower-level problem must be fully solved before the upper-level parameters are updated. If both levels directly optimize the same parameters, this nesting becomes ill-posed, since it would imply that the solution to one level depends on a quantity that itself changes during the optimization of the other.

To resolve this, BiSSL introduces two distinct but strongly correlated sets of backbone parameters $\boldsymbol{\theta}_P$ and $\boldsymbol{\theta}_D$ for the lower-level (pretext) and upper-level (downstream) objectives, respectively. This leads to the following formulation of BiSSL.

Definition 4.1 (BiSSL)
Under the notation of Section 4.1.1, the bilevel optimization problem of *BiSSL* is given by

$$\min_{\boldsymbol{\theta}_D,\boldsymbol{\phi}_D} \quad \mathcal{L}^D\left(\boldsymbol{\theta}_P^*\left(\boldsymbol{\theta}_D\right),\boldsymbol{\phi}_D\right) + \mathcal{L}^D\left(\boldsymbol{\theta}_D,\boldsymbol{\phi}_D\right) \tag{4.1}$$

$$\text{s.t.} \quad \boldsymbol{\theta}_P^*\left(\boldsymbol{\theta}_D\right) \in \operatorname*{argmin}_{\boldsymbol{\theta}_P} \min_{\boldsymbol{\phi}_P} \mathcal{L}^P\left(\boldsymbol{\theta}_P,\boldsymbol{\phi}_P\right) + \lambda r(\boldsymbol{\theta}_D,\boldsymbol{\theta}_P), \tag{4.2}$$

where $\lambda \in (0,\infty)$ and $r : \mathbb{R}^L \times \mathbb{R}^L \mapsto [0,\infty)$ is a regularization objective that that enforces similarity between $\boldsymbol{\theta}_D$ and $\boldsymbol{\theta}_P$, such that $r(\mathbf{x},\mathbf{y}) = 0$ iff $\mathbf{x} = \mathbf{y}$.

In this formulation, the upper-level problem (4.1) optimizes the downstream objective, but with a dependency on the backbone parameter solution $\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D)$ produced by the lower-level problem (4.2), that in turn is tasked with optimizing the pretext objective. This dependency explicitly models the backbone inheritance in the self-supervised training pipeline. The regularization objective $r(\boldsymbol{\theta}_D,\boldsymbol{\theta}_P) = \|\boldsymbol{\theta}_D - \boldsymbol{\theta}_P\|_2^2$ is adopted for the implementation considered in this project, and some results derived throughout this project will employ this objective. The inclusion of the second term in (4.1) improves convergence during training and plays an important role in ensuring the lower-level optimization problem is non-trivial. For a detailed explanation on why that is, see Appendix E.

The dependency of the upper-level objective (4.1) on the lower-level solution $\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D)$ allows the pretext task objective to influence the optimization of the upper-level. Simultaneously, the regularization term r in (4.2) ensures that the learned pretext parameters remain close to those used in the upper-level, thereby encouraging the lower-level solution to occupy regions of the parameter space that are more conducive to subsequent fine-tuning. This structured coupling aims to improve the initialization quality of the backbone $\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D)$, beyond what is achieved through conventional pretraining followed by standard fine-tuning.

As discussed in Chapter 3, solving bilevel problems in practice requires careful handling of the upper-level gradient due to its implicit dependence on the lower-level optimization. As will be evident in the following section, this is no exception for BiSSL.

4.2 Expressing the Bilevel Derivatives

This section explores how to express the derivatives of the upper and lower-level objectives of BiSSL, as well as how to achieve them in practice.

4.2.1 Lower-level Gradients

As the lower-level only depends on the upper-level through its parameters, the lower-level gradients are straightforward to calculate. Defining the lower-level objective in (4.2) as

$$G(\boldsymbol{\theta}_D,\boldsymbol{\theta}_P,\boldsymbol{\phi}_P) = \mathcal{L}^P\left(\boldsymbol{\theta}_P,\boldsymbol{\phi}_P\right) + \lambda r(\boldsymbol{\theta}_D,\boldsymbol{\theta}_P), \tag{4.3}$$

its gradients are

$$\nabla_{\boldsymbol{\theta}} G(\boldsymbol{\theta}_D, \boldsymbol{\theta}, \boldsymbol{\phi}_P)|_{\boldsymbol{\theta}=\boldsymbol{\theta}_P} = \nabla_{\boldsymbol{\theta}} \mathcal{L}^P(\boldsymbol{\theta}, \boldsymbol{\phi}_P)|_{\boldsymbol{\theta}=\boldsymbol{\theta}_P} + \lambda \nabla_{\boldsymbol{\theta}} r(\boldsymbol{\theta}_D, \boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}_P},$$
$$\nabla_{\boldsymbol{\phi}} G(\boldsymbol{\theta}_D, \boldsymbol{\theta}_P, \boldsymbol{\phi})|_{\boldsymbol{\phi}=\boldsymbol{\phi}_P} = \nabla_{\boldsymbol{\phi}} \mathcal{L}^P(\boldsymbol{\theta}_P, \boldsymbol{\phi})|_{\boldsymbol{\phi}=\boldsymbol{\phi}_P}.$$

As it will be shown in the next subsection, the upper-level gradient proves more complicated to express, requiring more careful considerations both regarding the analytical expression and calculation in practice.

4.2.2 Upper-level Derivatives

Defining the upper-level objective from (4.1) as

$$F(\boldsymbol{\theta}_D, \boldsymbol{\phi}_D) := \mathcal{L}^D(\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D), \boldsymbol{\phi}_D) + \mathcal{L}^D(\boldsymbol{\theta}_D, \boldsymbol{\phi}_D), \tag{4.4}$$

then the gradients of (4.4) are

$$\frac{\mathrm{d}F}{\mathrm{d}\boldsymbol{\theta}_D} = \underbrace{\frac{\mathrm{d}\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D)}{\mathrm{d}\boldsymbol{\theta}_D}}_{\mathrm{IJ}} \nabla_{\boldsymbol{\theta}} \mathcal{L}^D(\boldsymbol{\theta}, \boldsymbol{\phi}_D)|_{\boldsymbol{\theta}=\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D)} + \nabla_{\boldsymbol{\theta}} \mathcal{L}^D(\boldsymbol{\theta}, \boldsymbol{\phi}_D)|_{\boldsymbol{\theta}=\boldsymbol{\theta}_D}, \tag{4.5}$$
$$\frac{\mathrm{d}F}{\mathrm{d}\boldsymbol{\phi}_D} = \nabla_{\boldsymbol{\phi}} \mathcal{L}^D(\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D), \boldsymbol{\phi})|_{\boldsymbol{\phi}=\boldsymbol{\phi}_D} + \nabla_{\boldsymbol{\phi}} \mathcal{L}^D(\boldsymbol{\theta}_D, \boldsymbol{\phi})|_{\boldsymbol{\phi}=\boldsymbol{\phi}_D}.$$

Due to the dependence of the lower-level solution $\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D)$ in the upper-level objective, the first term of (4.5) includes the implicit jacobian (IJ) of the (implicit) lower-level solution $\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D)$. Recalling Theorem 3.2, the IJ can, under certain conditions, be expressed explicitly.

Notation: For the remaining derivations in this Chapter, the notation $\nabla_{\boldsymbol{\xi}} h(\boldsymbol{\xi})|_{\boldsymbol{\xi}=\boldsymbol{\psi}} := \nabla_{\boldsymbol{\xi}} h(\boldsymbol{\psi})$ is employed when it is clear from context which variables are differentiated with respect to.

Proposition 4.2 (Implicit Jacobian in BiSSL)
Given the bilevel optimization problem of BiSSL in Definition 4.1, assume that the pretext head parameters $\boldsymbol{\phi}_P$ are fixed, and that the lower-level objective (4.2) fulfills the conditions of Theorem 3.2. Then the IJ defined in (4.5) is given by

$$\frac{\mathrm{d}\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D)}{\mathrm{d}\boldsymbol{\theta}_D} = -\nabla^2_{\boldsymbol{\theta}_D \boldsymbol{\theta}_P} r(\boldsymbol{\theta}_D, \boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D)) \left[\nabla^2_{\boldsymbol{\theta}_P} \left(\frac{1}{\lambda} \mathcal{L}^P(\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D), \boldsymbol{\phi}_P) + r(\boldsymbol{\theta}_D, \boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D)) \right) \right]^{-1}, \tag{4.6}$$

for all $\boldsymbol{\theta}_D \in \mathcal{G}_0$, where $\mathcal{G}_0$ is the lower-level stationary condition set as defined in Definition 3.1.

While offering an explicit expression for the IJ, the above result also underscores the importance of including a regularization term r in the lower-level that depends on both backbone parameters $\boldsymbol{\theta}_D$ and $\boldsymbol{\theta}_P$. As evidenced by the leftmost Hessian in the right-hand side of (4.6), the IJ would collapse to the zero matrix in the absence of such a term in the lower-level, thereby blocking any information flow from the lower-level solution to the upper-level gradient in (4.5).

Proof (Proposition 4.2): Given a fixed pretext head parameter configuration $\boldsymbol{\phi}_P \in \mathbb{R}^{P_P}$, we express the lower-level objective as only a function of the backbone parameters, i.e.,

$$\tilde{G}(\boldsymbol{\theta}_D,\boldsymbol{\theta}_P) = \mathcal{L}^P\left(\boldsymbol{\theta}_P,\boldsymbol{\phi}_P\right) + \lambda r(\boldsymbol{\theta}_D,\boldsymbol{\theta}_P), \tag{4.7}$$

The imposed conditions from Theorem 3.2 imply that that the objective $\tilde{G}$ is twice differentiable and that the hessian $\nabla^2_{\boldsymbol{\theta}_P}\tilde{G}(\boldsymbol{\theta}_D,\hat{\boldsymbol{\theta}}_P)$ is invertible for all $\boldsymbol{\theta}_D \in \mathcal{G}_0$ and corresponding $\hat{\boldsymbol{\theta}}_P$ that fulfills $\nabla_{\boldsymbol{\theta}_P}\tilde{G}(\boldsymbol{\theta}_D,\hat{\boldsymbol{\theta}}_P) = \mathbf{0}$. Theorem 3.2 then states that the IJ is explicitly given by

$$\frac{\mathrm{d}\boldsymbol{\theta}^*_P(\boldsymbol{\theta}_D)}{\mathrm{d}\boldsymbol{\theta}_D} = -\nabla^2_{\boldsymbol{\theta}_D\boldsymbol{\theta}_P}\tilde{G}(\boldsymbol{\theta}_D,\boldsymbol{\theta}^*_P(\boldsymbol{\theta}_D))\left[\nabla^2_{\boldsymbol{\theta}_P}\tilde{G}(\boldsymbol{\theta}_D,\boldsymbol{\theta}^*_P(\boldsymbol{\theta}_D))\right]^{-1}, \quad \boldsymbol{\theta}_D \in \mathcal{G}_0.$$

Substituting (4.7) into the above expression then yields

$$\begin{aligned}\frac{\mathrm{d}\boldsymbol{\theta}^*_P(\boldsymbol{\theta}_D)}{\mathrm{d}\boldsymbol{\theta}_D} &= -\nabla^2_{\boldsymbol{\theta}_D\boldsymbol{\theta}_P}\left(\mathcal{L}^P\left(\boldsymbol{\theta}^*_P(\boldsymbol{\theta}_D),\boldsymbol{\phi}_P\right) + \lambda r(\boldsymbol{\theta}_D,\boldsymbol{\theta}^*_P(\boldsymbol{\theta}_D))\right)\left[\nabla^2_{\boldsymbol{\theta}_P}\left(\mathcal{L}^P(\boldsymbol{\theta}^*_P(\boldsymbol{\theta}_D),\boldsymbol{\phi}_P) + \lambda r(\boldsymbol{\theta}_D,\boldsymbol{\theta}^*_P(\boldsymbol{\theta}_D))\right)\right]^{-1} \\ &= -\nabla^2_{\boldsymbol{\theta}_D\boldsymbol{\theta}_P} r(\boldsymbol{\theta}_D,\boldsymbol{\theta}^*_P(\boldsymbol{\theta}_D))\left[\nabla^2_{\boldsymbol{\theta}_P}\left(\frac{1}{\lambda}\mathcal{L}^P(\boldsymbol{\theta}^*_P(\boldsymbol{\theta}_D),\boldsymbol{\phi}_P) + r(\boldsymbol{\theta}_D,\boldsymbol{\theta}^*_P(\boldsymbol{\theta}_D))\right)\right]^{-1}, \quad \boldsymbol{\theta}_D \in \mathcal{G}_0.\end{aligned}$$

■

The form of $\mathcal{L}^P$ will in practice vary depending on the type of pretext task that is utilized, making it nearly impossible to make general claims regarding the exact structure of $\mathcal{G}_0$. To circumvent this, a regularization objective r that is strongly convex in its second argument, along with an appropriate size of λ can be chosen such that the lower-level objective (4.2) will be approximately convex, irrespective of the choice of $\boldsymbol{\theta}_D$. This makes it reasonable to assume $\mathcal{G}_0 = \mathbb{R}^L$. Hence by using the strongly convex regularization function $r(\boldsymbol{\theta}_D,\boldsymbol{\theta}_P) = \frac{1}{2}\|\boldsymbol{\theta}_D - \boldsymbol{\theta}_P\|_2^2$, the IJ in (4.6) simplifies to

$$\frac{\mathrm{d}\boldsymbol{\theta}^*_P(\boldsymbol{\theta}_D)}{\mathrm{d}\boldsymbol{\theta}_D} = \left[\frac{1}{\lambda}\nabla^2_{\boldsymbol{\theta}_P}\mathcal{L}^P(\boldsymbol{\theta}^*_P(\boldsymbol{\theta}_D),\boldsymbol{\phi}_P) + I_L\right]^{-1}, \quad \boldsymbol{\theta}_D \in \mathbb{R}^L, \tag{4.8}$$

where I_L is the $L \times L$-dimensional identity matrix.

Recalling the considerations made in section 3.2.1, the conjugate gradient method outlined in Algorithm 1 can be employed to feasibly approximate the second therm of the upper-level gradient (4.5) which now takes the form

$$\frac{\mathrm{d}F}{\mathrm{d}\boldsymbol{\theta}_D} = \left[\frac{1}{\lambda}\nabla^2_{\boldsymbol{\theta}_P}\mathcal{L}^P(\boldsymbol{\theta}^*_P(\boldsymbol{\theta}_D),\boldsymbol{\phi}_P) + I_L\right]^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}^D(\boldsymbol{\theta},\boldsymbol{\phi}_D)|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*_P(\boldsymbol{\theta}_D)} + \nabla_{\boldsymbol{\theta}}\mathcal{L}^D(\boldsymbol{\theta},\boldsymbol{\phi}_D)|_{\boldsymbol{\theta}=\boldsymbol{\theta}_D}. \tag{4.9}$$

4.2.3 Interpretation of λ

The expression in (4.9) enables an interpretation of the role of the regularization objective r and its scaling λ in shaping the interaction between the upper- and lower-level problems in (4.1) and (4.2). When λ is very large, the regularization dominates the lower-level objective, effectively forcing $\boldsymbol{\theta}_P \approx \boldsymbol{\theta}_D$ and disabling the influence from the pretext objective $\mathcal{L}^P$ in the lower-level. Meanwhile, the IJ in (4.8) $\approx I_M$, diminishing the influence of the lower-level objective on the upper-level gradient (4.9). This effectively makes the upper-level objective equivalent to conventional fine-tuning. Conversely, if λ is very small,

Algorithm 2 BiSSL Training Algorithm [49]

Input: Backbone and head initializations $\boldsymbol{\theta}$, $\boldsymbol{\phi}_P$, $\boldsymbol{\phi}_D$, Training objectives $\mathcal{L}^P$, $\mathcal{L}^D$, Regularization weight $\lambda \in (0,\infty)$, Optimizers opt_P, opt_D, Number of training stage alternations $T \in \mathbb{N}$ with upper and lower-level iterations N_U,$N_L \in \mathbb{N}$.

Initialize $\boldsymbol{\theta}_P \leftarrow \boldsymbol{\theta}$ and $\boldsymbol{\theta}_D \leftarrow \boldsymbol{\theta}$.

for $t = 1,\ldots,T$ **do**
  for $n = 1,\ldots,N_L$ **do** ▷ Lower-level
    Compute $\mathbf{g}_{\boldsymbol{\phi}_P} = \nabla_{\boldsymbol{\phi}} \mathcal{L}^P(\boldsymbol{\theta}_P,\boldsymbol{\phi})|_{\boldsymbol{\phi}=\boldsymbol{\phi}_P}$.
    Compute $\mathbf{g}_{\boldsymbol{\theta}_P} = \nabla_{\boldsymbol{\theta}} \mathcal{L}^P(\boldsymbol{\theta}, \boldsymbol{\phi}_P)|_{\boldsymbol{\theta}=\boldsymbol{\theta}_P} + \lambda(\boldsymbol{\theta}_P - \boldsymbol{\theta}_D)$.
    Update $\boldsymbol{\phi}_P \leftarrow \text{opt}_P(\boldsymbol{\phi}_P, \mathbf{g}_{\boldsymbol{\phi}_P})$ and $\boldsymbol{\theta}_P \leftarrow \text{opt}_P(\boldsymbol{\theta}_P, \mathbf{g}_{\boldsymbol{\theta}_P})$.
  end for

  for $n = 1,\ldots,N_U$ **do** ▷ Upper-level
    Compute $\mathbf{g}_{\boldsymbol{\phi}_D} = \nabla_{\boldsymbol{\phi}} \mathcal{L}^D(\boldsymbol{\theta}_P,\boldsymbol{\phi})|_{\boldsymbol{\phi}=\boldsymbol{\phi}_D} + \nabla_{\boldsymbol{\phi}} \mathcal{L}^D(\boldsymbol{\theta}_D,\boldsymbol{\phi})|_{\boldsymbol{\phi}=\boldsymbol{\phi}_D}$.
    Compute $\mathbf{v} = \nabla_{\boldsymbol{\theta}} \mathcal{L}^D(\boldsymbol{\theta}, \boldsymbol{\phi}_D)|_{\boldsymbol{\theta}=\boldsymbol{\theta}_P}$.
    Approximate $\mathbf{v}_{\text{IJ}} \approx \left[I_M + \frac{1}{\lambda} \nabla^2_{\boldsymbol{\theta}} \mathcal{L}^P(\boldsymbol{\theta}, \boldsymbol{\phi}_P)|_{\boldsymbol{\theta}=\boldsymbol{\theta}_P}\right]^{-1} \mathbf{v}$. ▷ Use Algorithm 1
    Compute $\mathbf{g}_{\boldsymbol{\theta}_D} = \mathbf{v}_{\text{IJ}} + \nabla_{\boldsymbol{\theta}} \mathcal{L}^D(\boldsymbol{\theta}, \boldsymbol{\phi}_D)|_{\boldsymbol{\theta}=\boldsymbol{\theta}_D}$.
    Update $\boldsymbol{\phi}_D \leftarrow \text{opt}_D(\boldsymbol{\phi}_D, \mathbf{g}_{\boldsymbol{\phi}_D})$ and $\boldsymbol{\theta}_D \leftarrow \text{opt}_D(\boldsymbol{\theta}_D, \mathbf{g}_{\boldsymbol{\theta}_D})$.
  end for
end for

Return: Backbone Parameters $\boldsymbol{\theta}_P$.

the lower-level optimization is governed primarily by the pretext loss, recovering a setup close to standard pretraining. The resulting IJ (4.8) will then contain solely near-zero entries, causing the first term in (4.1) to correspond to linear probing on the lower-level solution. Given a suitable value of λ, this reveals that BiSSL enables a more nuanced interplay between pretext and downstream objectives, facilitating a joint training regime that interpolates between standard pretraining and fine-tuning in a manner not possible within the conventional self-supervised learning pipeline.

Knowing how to explicitly express and approximate the gradients of the BiSSL problem in (4.1) and (4.2), a training algorithm and pipeline based on BiSSL can now be proposed.

4.3 Training Algorithm and Pipeline

The bilevel optimization formulation of BiSSL in Equations (4.1) and (4.2) requires solving two nested optimization problems concurrently. This is reflected in the training pipeline outlined in Algorithm 2, which alternates between updating the lower- and upper-level objectives. The lower-level (pretext) objective is optimized using standard gradient-based methods. In contrast, the upper-level (downstream) objective requires computing gradients that depend implicitly on the solution to the lower-level problem. The conjugate gradient method is employed to approximate the necessary gradient components, as discussed in Section 4.2.2.

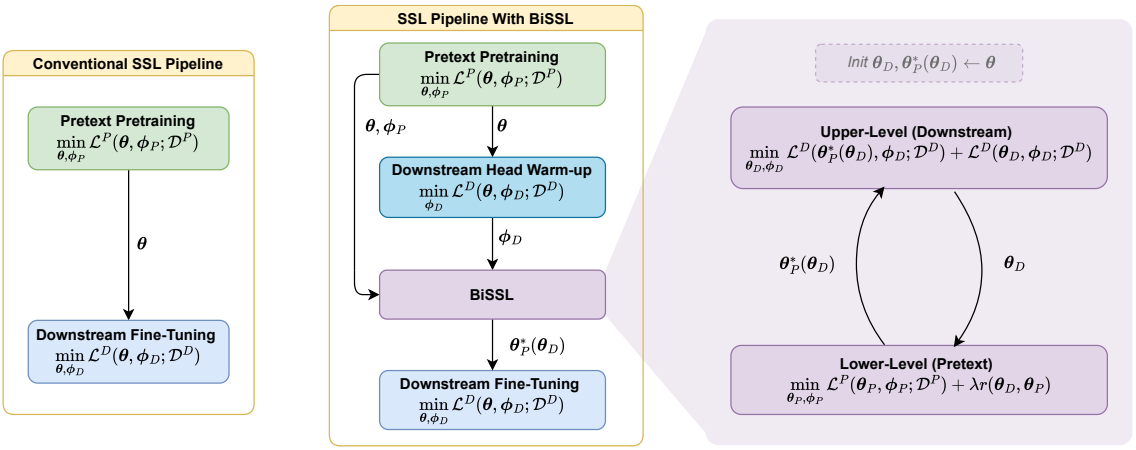

Figure 4.1: Left: Pipeline of conventional self-supervised learning (SSL). Right: Suggested training pipeline involving BiSSL [49]. The transferred parameters are used as initializations in the respective training stages they are transmitted to.

4.3.1 Training Pipeline Involving BiSSL

To apply the explicit expression of the IJ in Equation (4.8), the solution $\boldsymbol{\theta}^*_P(\boldsymbol{\theta}_D)$ must satisfy the stationary condition

$$\nabla_{\boldsymbol{\theta}} G(\boldsymbol{\theta}_D,\boldsymbol{\theta},\boldsymbol{\phi}_P)|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*_P(\boldsymbol{\theta}_D)} = \mathbf{0},$$

Starting Algorithm 2 from randomly initialized parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}_P$ is therefore unlikely to yield a configuration even remotely satisfying this condition. A similar issue arises for $\boldsymbol{\phi}_D$, as a random initialization can cause large initial updates to the upper-level backbone parameters $\boldsymbol{\theta}_D$, which in turn leads to a violation of the stationary condition through the dependence between $\boldsymbol{\theta}_D$ and $\boldsymbol{\theta}_P$ through r.

These considerations motivate placing BiSSL after an initial stage of standard self-supervised pretraining. Likewise, the downstream head $\boldsymbol{\phi}_D$ should be initialized prior to BiSSL, to reduce early training instability. The full training pipeline is summarized in the middle and right side of Figure 4.1, in contrast to the conventional SSL pipeline on the left. The proposed procedure involves four stages:

1. **Pretext training:** Standard self-supervised training is applied on the unlabeled dataset $\mathcal{D}^P$ to obtain initial values for the backbone parameters $\boldsymbol{\theta}$ and pretext head $\boldsymbol{\phi}_P$.

2. **Linear probing:** A linear classifier $\boldsymbol{\phi}_D$ is trained on top of the frozen backbone $\boldsymbol{\theta}$ using the labeled downstream dataset $\mathcal{D}^D$, providing an initialization for the downstream head.

3. **BiSSL training:** With suitable initializations for the parameters $\boldsymbol{\theta}$, $\boldsymbol{\phi}_D$ and $\boldsymbol{\phi}_D$, Algorithm 2 is executed, yielding an updated set of backbone parameters $\boldsymbol{\theta}^*_P(\boldsymbol{\theta}_D)$.

4. **Fine-tuning:** The updated backbone $\boldsymbol{\theta}^*_P(\boldsymbol{\theta}_D)$ is used to initialize standard supervised fine-tuning on $\mathcal{D}^D$.

5 | Revisiting BiSSL

Chapter 4 introduced our previous work of BiSSL, a bilevel training framework designed to improve the alignment between self-supervised pretraining and downstream fine-tuning. While the original formulation offers a compelling structure, several conceptual and empirical dimensions remain insufficiently explored. This chapter revisits these open questions, as outlined in 1.2.2, and proposes theoretical extensions and practical considerations aimed at addressing them, which will be evaluated in the experiments of Chapters 6 and 7.

5.1 Hyperparameter Impact

The empirical sensitivity of BiSSL to key hyperparameters remains underexplored. In particular:

- λ: Section 4.2.3 provides an interpretation of how different values of λ interpolate the impact of the pretext and downstream objectives on the composite bilevel optimization problem. As the original work kept $\lambda = 0.001$ during all experiments, it remains an open question how different values of λ impact downstream performance, and if the aforementioned interpretation can be reflected in the downstream performance numbers.
- T (Number of BiSSL iterations): As described in Algorithm 2, the BiSSL training procedure alternates between solving the lower- and upper-level problems for T iterations. The original work used $T = 500$ consistently. It remains unclear how longer or shorter BiSSL training affects downstream performance.
- N_U, N_L (Number of upper and lower level iterations): As described in Algorithm 2, these determine how many gradient steps are taken in solving the upper- and lower-level problems, respectively. The original experiments used $N_U = 8$ and $N_L = 20$, but also reported a minor study that implied setting $N_U = 1$ degraded performance. Besides this, no broader study was conducted.
- Fine-Tuning Epochs: The original experiments used 400 epochs for subsequent downstream fine-tuning. Given that BiSSL aims to produce better-initialized backbones, it is plausible that fewer fine-tuning epochs may suffice. Additionally, comparing fine-tuning curves between BiSSL and standard baselines may reveal differences in susceptibility to overfitting.

To assess the effect of each hyperparameter, we propose to conduct a series of controlled grid search experiments, varying one parameter at a time while keeping the others fixed. The implementation details and results are reported in Section 6.3 of Chapter 6.

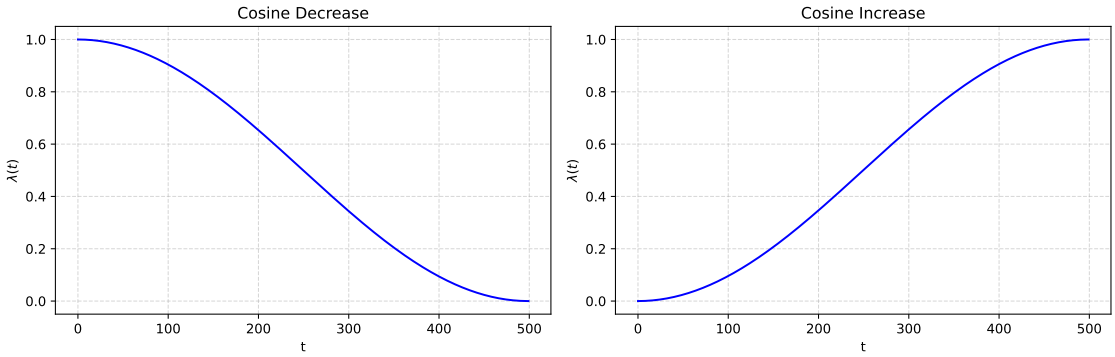

Figure 5.1: Cosine decaying (left) and cosine increasing (right) schedules for $\lambda(t)$ over $T = 500$ steps with $\lambda_{\min} = 10^{-8}$ and $\lambda_{\max} = 1$.

5.2 Adapting λ During Training

The regularization weight λ plays a central role in BiSSL, modulating the trade-off between solving the pretext task and aligning the representations for the downstream objective as outlined in Section 4.2.3. While λ previously has been treated as a fixed hyperparameter, it is natural to consider adapting its value during training. This section will propose multiple strategies for doing so.

We first turn towards examining basic scheduling strategies on λ. This first raises the question of how such a scheduler should evolve during training. Large λ forces stronger alignment between the tasks, while small λ prioritizes solving the pretext objective on its own terms. Consequently, one might prefer a larger λ early in training to guide the representation towards downstream-relevant features. Later in training, relaxing this constraint (decreasing λ) can allow the lower-level to refine its representations more flexibly. An alternative view is to treat the lower-level task as a form of initialization or preconditioning. In this case, starting with a small λ may help the model first acquire broadly useful representations, with λ gradually increased to tie them more tightly to downstream relevance.

These competing intuitions motivate opposing schedules, and we accordingly suggest scheduling strategies that support both views.

5.2.1 Scheduling Strategies

We propose employing schedulers on λ that update for every training state alternation T (see Algorithm 2). In general, we will consider schedules in the form

$$\lambda(t) = (\lambda_{\max} - \lambda_{\min}) f(t) + \lambda_{\min}, \quad t = 0, \dots, T-1,$$

where $f : \{0, \dots, T-1\} \mapsto [0,1]$ is a monotonic function. We propose three variants, each involving a different choice of f.

Cosine Decay Motivated by the intuition that stronger downstream guidance should be imposed early during BiSSL, we define a cosine decaying scheduler:

$$f^{\mathrm{CD}}(t) = \frac{1}{2}\left(1 + \cos \pi \frac{t}{T-1}\right), \quad t = 0, \dots, T-1,$$

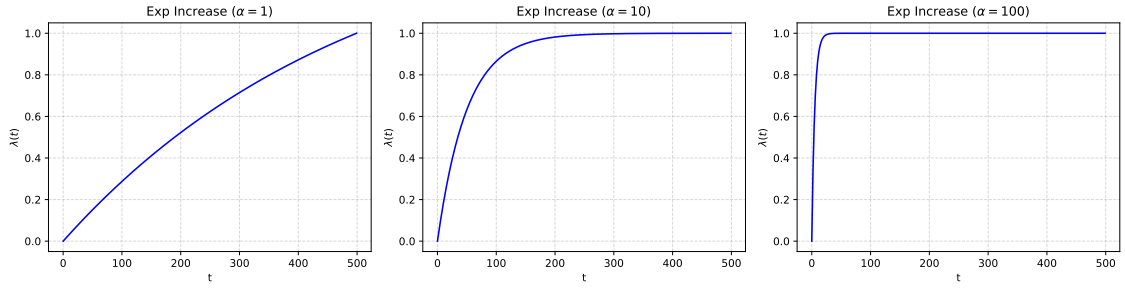

Figure 5.2: Exponential increasing schedules for $\lambda(t)$ with $\alpha = 1$ (left), $\alpha = 10$ (middle) and $\alpha = 100$ (right) over $T = 500$ steps using $\lambda_{\min} = 10^{-8}$ and $\lambda_{\max} = 1$.

with $f^{\mathrm{CD}}(0) = 1$ and $f^{\mathrm{CD}}(T-1) = 0$. This hence encourages tight coupling between the levels initially, gradually relaxing this constraint as training progresses.

Cosine Increase Appealing to the latter interpretation that an increasing scheduler would be more beneficial for the training procedure, we suggest a reversed cosine schedule that increases during training:

$$f^{\mathrm{CI}}(t) = 1 - \frac{1}{2}\left(1 + \cos \pi \frac{t}{T-1}\right), \quad t = 0, \ldots, T-1,$$

so that $f^{\mathrm{CI}}(0) = 0$ and $f^{\mathrm{CI}}(T-1) = 1$. This yields a smooth transition from pure pretext optimization toward progressively stronger downstream influence.

Figure 5.1 shows examples of plots of the two cosine suggested schedulers.

Exponential Increase The cosine increasing scheduler may rise too slowly, leading to insufficient downstream influence during much of training. To address this, we also consider an exponentially increasing schedule:

$$f^{\mathrm{EI}}(t) = \frac{1 - e^{-\alpha \frac{t}{T-1}}}{1 - e^{-\alpha}}, \quad t = 0, \ldots, T-1,$$

such that $f^{\mathrm{EI}}(0) = 0$ and $f^{\mathrm{EI}}(T-1) = 1$, where $\alpha \in (0, \infty)$ controls the sharpness of the increase. When $\alpha \to 0$, this schedule approximates a linear increase. Oppositely, as α gets larger, the rise towards 1 happens more quickly, effectively approaching a step-like transition. Figure 5.2 plots this schedule for various values of α, illustrating how the parameter governs the sharpness of the transition.

These scheduling approaches offer a simple and principled way to modulate λ without modifying the optimization structure of BiSSL. They are compatible with existing training pipelines and easy to implement. As such, they provide a strong baseline for evaluating the impact of adaptive regularization. Experiments evaluating the downstream performance using the aforementioned schedulers are presented in Section 6.4.1 of Chapter 6.

While these schedules provide coarse control, a more flexible alternative would be to instead exploit the bilevel structure of BiSSL and treat λ as an upper-level learnable parameter.

5.2.2 Learnable λ

The bilevel optimization (BLO) framework underlying BiSSL provides a natural way to adapt λ during training, avoiding the reliance on handcrafted schedules. We propose to make λ a trainable variable in the upper-level optimization problem, letting the optimization itself determine how tightly to couple the objectives. We extend the original BiSSL formulation in Definition 4.1 by including λ as a trainable upper-level parameter:

Definition 5.1 (BiSSL with Trainable λ)
Under the conditions of Definition 4.1, the bilevel optimization problem of *BiSSL* with λ as is trainable upper-level parameter is given by

$$\min_{\boldsymbol{\theta}_D,\boldsymbol{\phi}_D,\lambda} \quad \mathcal{L}^D\left(\boldsymbol{\theta}_P^*\left(\boldsymbol{\theta}_D,\lambda\right),\boldsymbol{\phi}_D\right) + \mathcal{L}^D\left(\boldsymbol{\theta}_D,\boldsymbol{\phi}_D\right) \tag{5.1}$$

$$\text{s.t.} \quad \boldsymbol{\theta}_P^*\left(\boldsymbol{\theta}_D,\lambda\right) \in \operatorname*{argmin}_{\boldsymbol{\theta}_P} \min_{\boldsymbol{\phi}_P} \mathcal{L}^P\left(\boldsymbol{\theta}_P,\boldsymbol{\phi}_P\right) + \sigma(\lambda) r(\boldsymbol{\theta}_D,\boldsymbol{\theta}_P), \tag{5.2}$$

where $\sigma : \mathbb{R} \mapsto (0,\infty)$ is a differentiable and monotonously increasing function.

This reformulation leaves the structure of BiSSL nearly unchanged, requiring no changes to the optimization procedure aside from including λ in the upper-level gradient computations. The mapping σ is introduced to enforce the constraint $\lambda > 0$ implicitly, as direct optimization over $(0,\infty)$ is usually not feasible using gradient-based optimizers. The softplus function [65] will be employed as σ for the experiments in Chapter 6, i.e., $\sigma(x) = \log\left(1 + e^x\right)$.

Deriving the Upper-Level Derivative Define the upper-level objective in (5.1) as

$$F(\boldsymbol{\theta}_D,\boldsymbol{\phi}_D,\lambda) := \mathcal{L}^D\left(\boldsymbol{\theta}_P^*\left(\boldsymbol{\theta}_D,\lambda\right),\boldsymbol{\phi}_D\right) + \mathcal{L}^D\left(\boldsymbol{\theta}_D,\boldsymbol{\phi}_D\right). \tag{5.3}$$

To optimize λ through the upper-level objective, we must compute the gradient

$$\frac{\mathrm{d}F}{\mathrm{d}\lambda} = \frac{\mathrm{d}\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda)}{\mathrm{d}\lambda} \nabla_{\boldsymbol{\theta}} \mathcal{L}^D(\boldsymbol{\theta},\boldsymbol{\phi}_D)|_{\boldsymbol{\theta}=\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda)}. \tag{5.4}$$

While the above expression seems to necessitate the approximation of yet another IJ, namely $\frac{\mathrm{d}\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda)}{\mathrm{d}\lambda}$, we show next that this can be avoided entirely. For the rest of this chapter, we use the notation $\nabla_{\boldsymbol{\xi}} h(\boldsymbol{\xi})|_{\boldsymbol{\xi}=\boldsymbol{\psi}} := \nabla_{\boldsymbol{\xi}} h(\boldsymbol{\psi})$ when it is clear from context which variables are differentiated with respect to.

Proposition 5.2 (Upper-Level Derivative With Respect To λ)
Let the bilevel optimization problem in Definition 5.1 with $r(\boldsymbol{\theta}_D,\boldsymbol{\theta}_P) = \frac{1}{2}\|\boldsymbol{\theta}_D - \boldsymbol{\theta}_P\|_2^2$ be given. Assuming that the lower-level (5.2) fulfills the conditions of Theorem 3.2, the derivative of the upper-level objective as defined in (5.3) with respect to λ is given by

$$\frac{\mathrm{d}F}{\mathrm{d}\lambda} = \frac{1}{\sigma(\lambda)} \frac{\partial\sigma}{\partial\lambda} \left\langle \boldsymbol{\theta}_D - \boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda), \frac{\mathrm{d}\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda)}{\mathrm{d}\boldsymbol{\theta}_D} \nabla_{\boldsymbol{\theta}} \mathcal{L}^D(\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda),\boldsymbol{\phi}_D) \right\rangle, \tag{5.5}$$

where $\langle\cdot,\cdot\rangle$ denotes the euclidean inner product.

We observe that the rightmost term inside the inner product of (5.5) corresponds to the gradient of the upper-level loss with respect to $\boldsymbol{\theta}_D$ as in Equation (4.5). Since this gradient is already computed as part of the standard BiSSL training loop, evaluating the above derivative hence requires only the additional calculation of a single dot product and a few elementary operations. Consequently, introducing a learnable λ as in Definition 5.1 adds virtually no computational overhead.

Proof (Proposition** (5.2)**): Theorem 3.2 implies that the IJ present in (5.4) is given by

$$\frac{\mathrm{d}\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda)}{\mathrm{d}\lambda} = -\nabla^2_{\lambda\boldsymbol{\theta}_P}\sigma(\lambda)r(\boldsymbol{\theta}_D,\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda))\Big[\nabla^2_{\boldsymbol{\theta}_P}\Big(\mathcal{L}^P(\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D),\boldsymbol{\phi}_P) + \sigma(\lambda)r(\boldsymbol{\theta}_D,\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D))\Big)\Big]^{-1}.$$

Using the lower-level regularization objective $r(\boldsymbol{\theta}_D,\boldsymbol{\theta}_P) = \frac{1}{2}\|\boldsymbol{\theta}_D - \boldsymbol{\theta}_P\|_2^2$, the equation above simplifies to

$$\begin{aligned}\frac{\mathrm{d}\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda)}{\mathrm{d}\lambda} &= \nabla_\lambda\sigma(\lambda)(\boldsymbol{\theta}_D - \boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda))\Big[\nabla^2_{\boldsymbol{\theta}_P}\mathcal{L}^P(\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D),\boldsymbol{\phi}_P) + \sigma(\lambda)I_L\Big]^{-1}\\ &= \frac{\partial\sigma}{\partial\lambda}(\boldsymbol{\theta}_D - \boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda))^\top\Big[\nabla^2_{\boldsymbol{\theta}_P}\mathcal{L}^P(\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D),\boldsymbol{\phi}_P) + \sigma(\lambda)I_L\Big]^{-1}.\end{aligned}$$

Substituting the above into the expression of the upper-level gradient with respect to λ in (5.4) then provides the expression

$$\begin{aligned}\frac{\mathrm{d}F}{\mathrm{d}\lambda} &= \frac{\partial\sigma}{\partial\lambda}(\boldsymbol{\theta}_D - \boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda))^\top\big[\nabla^2_{\boldsymbol{\theta}_P}\mathcal{L}^P(\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D),\boldsymbol{\phi}_P) + \sigma(\lambda)I_L\big]^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}^D(\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda),\boldsymbol{\phi}_D)\\ &= \frac{1}{\sigma(\lambda)}\frac{\partial\sigma}{\partial\lambda}(\boldsymbol{\theta}_D - \boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda))^\top\underbrace{\left[\frac{1}{\sigma(\lambda)}\nabla^2_{\boldsymbol{\theta}_P}\mathcal{L}^P(\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D),\boldsymbol{\phi}_P) + I_L\right]^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}^D(\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda),\boldsymbol{\phi}_D)}_{\text{Derivative of } \mathcal{L}^D(\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda),\boldsymbol{\phi}_D) \text{ wrt. } \boldsymbol{\theta}_D}.\end{aligned}$$

The under-braced part above is exactly equal to the gradient of the first term of the upper-level objective with respect to $\boldsymbol{\theta}_D$ from (4.9). This finally leads to

$$\begin{aligned}\frac{\mathrm{d}F}{\mathrm{d}\lambda} &= \frac{1}{\sigma(\lambda)}\frac{\partial\sigma}{\partial\lambda}(\boldsymbol{\theta}_D - \boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda))^\top\frac{\mathrm{d}\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda)}{\mathrm{d}\boldsymbol{\theta}_D}\nabla_{\boldsymbol{\theta}}\mathcal{L}^D(\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda),\boldsymbol{\phi}_D)\\ &= \frac{1}{\sigma(\lambda)}\frac{\partial\sigma}{\partial\lambda}\left\langle\boldsymbol{\theta}_D - \boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda), \frac{\mathrm{d}\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda)}{\mathrm{d}\boldsymbol{\theta}_D}\nabla_{\boldsymbol{\theta}}\mathcal{L}^D(\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda),\boldsymbol{\phi}_D)\right\rangle.\end{aligned}$$

■

In light of the derivations in the proof above, it is noteworthy to mention that the simple expression achieved in Proposition 5.2 is only achievable with the specific choice of regularization objective for which the mixed Hessian $\nabla^2_{\boldsymbol{\theta}_D\boldsymbol{\theta}_P}r(\boldsymbol{\theta}_D,\boldsymbol{\theta}_P)$ equals the identity matrix (as discussed prior to Equation (4.8)). In the general case where no specific structure is imposed on r, computing the derivative with respect to λ would require additional Hessians not already available from the BiSSL gradient flow, thereby incurring a substantial increase in computational cost. As further exploration of these generalizations lies outside the scope of this work, we leave this here as a potential avenue for future work.

Dampening The factor $\frac{1}{\sigma(\lambda)}$ in (5.5) implicitly acts as a dampening mechanism: when $\sigma(\lambda)$ is large, its gradient contributions are downscaled, potentially stabilizing training. However, this also poses a risk, namely if $\sigma(\lambda)$ grows too large early in training, its gradient becomes vanishingly small, effectively freezing at a large value. Conversely, small values of $\sigma(\lambda)$ will produce large and potentially unstable updates.

To avoid such brittle behavior, we suggest a minor modification to the gradient in Proposition 5.2, by introducing a dampening term $\lambda_{\text{damp}} \in (0,\infty)$ into the denominator:

$$\frac{\mathrm{d}F}{\mathrm{d}\lambda} = \frac{1}{\sigma(\lambda) + \lambda_{\text{damp}}} \frac{\partial \sigma}{\partial \lambda} \left\langle \boldsymbol{\theta}_D - \boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda), \frac{\mathrm{d}\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda)}{\mathrm{d}\boldsymbol{\theta}_D} \nabla_{\boldsymbol{\theta}} \mathcal{L}^D(\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda), \boldsymbol{\phi}_D) \right\rangle, \tag{5.6}$$

This prevents division by values very close to zero and adds a stable floor to the learning rate of the optimization of λ. Experiments with λ being learnable are presented in Section 6.4.2 of Chapter 6.

5.3 Non-Fixed Pretext Head Doing IJ Approximation

In earlier derivations of the IJ back in Proposition 4.2, the pretext head parameters $\boldsymbol{\phi}_P$ were assumed to be fixed. While this simplification made the derivations more tractable, it was not without loss of generality. Specifically, it neglects how the backbone and pretext head parameters $\boldsymbol{\theta}_P$, $\boldsymbol{\phi}_P$ interact through the pretext objective $\mathcal{L}^P$, which in turn potentially can reduce the fidelity of the IJ approximation. This section examines how lifting the fixed $\boldsymbol{\phi}_P$-assumption affects the IJ and the upper-level gradient, and concludes by proposing a practical strategy for approximating the resulting modified IJ.

At first glance, one might attempt to directly adapt the original BiSSL structure in Definition 4.1. However, this introduces a technical obstacle. The fundamental result in Theorem 3.2 is tailored towards BLO problems where the lower-level objectives depend solely on two distinct sets of adjustable parameters. Since $\boldsymbol{\phi}_P$ appears as a third variable in the lower-level objective (4.3), which in turn is not present in the upper-level, it hence falls outside the structure assumed by this result.

Rather than trying to shoehorn the current formulation of BiSSL into the structure required by Theorem 3.2, we instead propose a reformulation of BiSSL. Specifically, we suggest concatenating the lower-level backbone and pretext head parameters into a single composite vector, treating them as a unified set of lower-level variables. This enables the problem to be cast in a more standard bilevel form, where the lower-level optimization operates over a single parameter vector. We now proceed with this reformulation, which first involves a few structural adjustments to ensure compatibility within the BiSSL framework.

5.3.1 Reformulated BiSSL

We employ the same notation from Section 4.1.1, and now collect the lower-level parameters $\boldsymbol{\theta}_P$ and $\boldsymbol{\phi}_P$ into a single concatenated vector $\boldsymbol{\omega} = [\boldsymbol{\theta}_P^\top, \boldsymbol{\phi}_P^\top]^\top \in \mathbb{R}^{L+P_P}$, such that its individual components are given by

$$\boldsymbol{\omega}_i = \begin{cases} \boldsymbol{\theta}_i & \text{if } i \leq L \\ \boldsymbol{\phi}_{i-L} & \text{otherwise} \end{cases} \tag{5.7}$$

for $i = 1, \ldots, L + P_P$. To selectively operate on the subcomponents of $\boldsymbol{\omega}$, we define the sub-parameter mappings.

Definition 5.3 (Sub-Parameter Mappings)
Given $\boldsymbol{\omega} \in \mathbb{R}^{L+P_P}$ with entries as in (5.7), the *sub-parameter mappings* $\gamma_{\boldsymbol{\theta}_P} : \mathbb{R}^{L+P_P} \mapsto \mathbb{R}^L$ and $\gamma_{\boldsymbol{\phi}_P} : \mathbb{R}^{L+P_P} \mapsto \mathbb{R}^{P_P}$ are defined such that they respectively extract the backbone and pretext head parameters $\boldsymbol{\theta}_P$, $\boldsymbol{\phi}_P$ from $\boldsymbol{\omega}$, i.e., $\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}) = \boldsymbol{\theta}_P$ and $\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}) = \boldsymbol{\phi}_P$.

The sub-parameter mappings can be written explicitly as the matrix-vector products

$$\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}) = [I_L \quad \mathbf{O}_{L\times P_P}]\boldsymbol{\omega} = \boldsymbol{\theta}_P, \tag{5.8}$$

and

$$\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}) = [\mathbf{O}_{P_P\times L} \quad I_{P_P}]\boldsymbol{\omega} = \boldsymbol{\phi}_P. \tag{5.9}$$

where $\mathbf{O}_{X\times Y}$ denote the $X \times Y$-dimensional zero matrix. The notation $[A_{X\times Y} \quad B_{X\times Z}] \in \mathbb{R}^{X\times Y+Z}$ is used to denote the horizontal concatenation of matrices $A_{X\times Y} \in \mathbb{R}^{X\times Y}$ and $B_{X\times Z} \in \mathbb{R}^{X\times Z}$ such that its first Y columns are composed of A and its last Z columns are composed by B. *Similar composed block structures of matrices and vectors will recur throughout the remainder of this section. We will omit further explicit definitions of this notation as the meaning should be clear from context.*

This simple structure for expressing the sub-parameter mappings will prove beneficial when deriving the IJ in the subsequent section. We are now equipped to state the reformulated BiSSL problem.

Definition 5.4 (BiSSL with Concatenated Lower-Level Parameters)
Under the conditions of Definition 4.1, the bilevel optimization problem of BiSSL with concatenated lower-level parameters is defined as

$$\min_{\boldsymbol{\theta}_D,\boldsymbol{\phi}_D} \quad \mathcal{L}^D(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)),\boldsymbol{\phi}_D) + \mathcal{L}^D(\boldsymbol{\theta}_D,\boldsymbol{\phi}_D) \tag{5.10}$$

$$\text{s.t.} \quad \boldsymbol{\omega}^*(\boldsymbol{\theta}_D) \in \operatorname*{argmin}_{\boldsymbol{\omega}} \mathcal{L}^P(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega})) + \lambda r(\boldsymbol{\theta}_D, \gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega})), \tag{5.11}$$

where $\gamma_{\boldsymbol{\theta}_P}$, $\gamma_{\boldsymbol{\phi}_P}$ are the sub-parameter mappings defined in Definition (5.3).

This reformulation is effectively equivalent to the original bilevel problem in Definition 4.1. As noted, this alternative phrasing will prove convenient when relaxing the fixed pretext head assumption in the derivation of the IJ.

5.3.2 Implicit Jacobian of Concatenated Lower-Level Solution

The lower-level objective (5.11) can now more compactly be expressed in terms of the aggregated parameter vector $\boldsymbol{\omega}$ as

$$G(\boldsymbol{\theta}_D,\boldsymbol{\omega}) := \mathcal{L}^P(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega})) + \lambda r(\boldsymbol{\theta}_D, \gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega})).$$

Additionally, the gradient of the first term of the upper-level objective in (5.10) with respect to $\boldsymbol{\theta}_D$ is now

$$\frac{\mathrm{d}\mathcal{L}^D(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)),\boldsymbol{\phi}_D)}{\mathrm{d}\boldsymbol{\theta}_D} = \frac{\mathrm{d}\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))}{\mathrm{d}\boldsymbol{\theta}_D} \nabla_{\boldsymbol{\theta}} \mathcal{L}^D(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)), \boldsymbol{\phi}_D).$$

Rather than attempting to express the above IJ directly, we will first derive the expression for $\frac{\mathrm{d}\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)}{\mathrm{d}\boldsymbol{\theta}_D}$, which will serve as a key intermediate step. For notational simplicity, we again use the notation $\nabla_{\boldsymbol{\xi}} h(\boldsymbol{\xi})|_{\boldsymbol{\xi}=\boldsymbol{\psi}} := \nabla_{\boldsymbol{\xi}} h(\boldsymbol{\psi})$ when it is clear from context which variables are differentiated with respect to.

Proposition 5.5 (IJ of Concatenated Lower-Level Solution)
Let the bilevel optimization problem in Definition 5.4 be given. Assuming the lower-level (5.11) satisfies the conditions of Theorem 3.2, then the following IJ is given by

$$\frac{\mathrm{d}\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)}{\mathrm{d}\boldsymbol{\theta}_D} = M_{\boldsymbol{\theta}_D,\boldsymbol{\omega}}(\boldsymbol{\theta}_D) M_{\boldsymbol{\omega}}(\boldsymbol{\theta}_D), \quad \boldsymbol{\theta} \in \mathcal{G}_0 \tag{5.12}$$

where $\mathcal{G}_0$ is the lower-level stationary set as given in Definition 3.1, and

$$M_{\boldsymbol{\theta}_D,\boldsymbol{\omega}}(\boldsymbol{\theta}_D) = \begin{bmatrix} \nabla^2_{\boldsymbol{\theta}_D\boldsymbol{\theta}_P} r(\boldsymbol{\theta}_D, \gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) & \mathbf{O}_{L\times P_P} \end{bmatrix},$$

$$M_{\boldsymbol{\omega}}(\boldsymbol{\theta}_D) = \left[\frac{1}{\lambda} \frac{\mathrm{d}^2}{\mathrm{d}^2\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)} \mathcal{L}^P(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)), \gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) + \begin{bmatrix} \nabla^2_{\boldsymbol{\theta}_P} r(\boldsymbol{\theta}_D, \gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) & \mathbf{O}_{L\times P_P} \\ \mathbf{O}_{P_P\times L} & \mathbf{O}_{P_P\times P_P} \end{bmatrix} \right]^{-1}.$$

As the proof transpires in a manner similar to that of Proposition 4.2 and involves extensive mathematical notation, it is defered to Appendix C. The structure of the IJ in (5.12) remains largely consistent with that in Proposition 4.2. The primary difference lies in the Hessian block

$$\begin{aligned} &\frac{\mathrm{d}^2}{\mathrm{d}^2\boldsymbol{\omega}} \mathcal{L}^P(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)), \gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) \\ &= \begin{bmatrix} \nabla^2_{\boldsymbol{\theta}_P} \mathcal{L}^P(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)), \gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) & \nabla^2_{\boldsymbol{\theta}_P\boldsymbol{\phi}_P} \mathcal{L}^P(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)), \gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) \\ \nabla^2_{\boldsymbol{\phi}_P\boldsymbol{\theta}_P} \mathcal{L}^P(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)), \gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) & \nabla^2_{\boldsymbol{\phi}_P} \mathcal{L}^P(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)), \gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) \end{bmatrix}, \end{aligned} \tag{5.13}$$

which follows by application of Lemma C.3. This matrix introduces second-order derivatives with respect to the pretext head parameters $\boldsymbol{\phi}_P$. Since this block appears inside the inverted matrix $M_{\boldsymbol{\omega}}(\boldsymbol{\theta}_D)$ of Proposition 5.5, it couples $\boldsymbol{\phi}_P$ into the computation of the IJ. That is, although $\boldsymbol{\phi}_P$ does not appear in the lower-level regularization objective r, it influences the IJ through its role in the pretext objective $\mathcal{L}^P$.

Now, returning to the case of using the specific regularization objective $r(\boldsymbol{\theta}_D, \boldsymbol{\theta}_P) = \frac{1}{2}\|\boldsymbol{\theta}_D - \boldsymbol{\theta}_P\|_2^2$, we follow the same reasoning as in Section 4.2.2 to assume that $\mathcal{G}_0 = \mathbb{R}^L$, leading to the IJ in Proposition 5.5 simplifying to

$$\begin{aligned} \frac{\mathrm{d}\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)}{\mathrm{d}\boldsymbol{\theta}_D} &= [I_L \quad \mathbf{O}_{L\times P_P}] \underbrace{\left[\frac{1}{\lambda} \frac{\mathrm{d}^2}{\mathrm{d}^2\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)} \mathcal{L}^P(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)), \gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) + \begin{bmatrix} I_L & \mathbf{O}_{L\times P_P} \\ \mathbf{O}_{P_P\times L} & \mathbf{O}_{P_P\times P_P} \end{bmatrix} \right]^{-1}}_{A^{-1}(\boldsymbol{\theta}_D)} \\ &:= [I_L \quad \mathbf{O}_{L\times P_P}] A^{-1}(\boldsymbol{\theta}_D), \end{aligned} \tag{5.14}$$

for $\boldsymbol{\theta}_D \in \mathbb{R}^L$, where we introduce the shorthand $A^{-1}(\boldsymbol{\theta}_D) \in \mathbb{R}^{L+P_P \times L+P_P}$ for later reference.

As a sanity check, if the pretext head parameters ϕ_P are assumed fixed, then the dependence of $A(\boldsymbol{\theta}_D)^{-1}$ on these parameters disappears in the above equation, yielding only the upper-leftmost sub-matrix in (5.13) to assume non-zero values. In that case, apart from the increased dimensionality, we recover the original IJ expression from Proposition (4.2).

5.3.3 Updated Upper-Level Derivative

Now that the sub-parameter mapping is applied to the solution variable in the upper-level objective, i.e., $\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))$, a potential complication arises: taking the derivative of $\mathcal{L}^D(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)),\boldsymbol{\phi}_D)$ with respect to $\boldsymbol{\theta}_D$ could introduce third-order derivatives, due to the nested composition through $\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))$. However, the following corollary shows that this can be circumvented.

Corollary 5.6 (Derivative of First Term in Equation (5.10)**)**
Under the conditions of Proposition 5.5, and assuming $r(\boldsymbol{\theta}_D, \boldsymbol{\theta}_P) = \frac{1}{2}\|\boldsymbol{\theta}_D - \boldsymbol{\theta}_P\|_2^2$, the derivative of the first term of the upper-level objective (5.10) is given by

$$\frac{\mathrm{d}\mathcal{L}^D(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)),\boldsymbol{\phi}_D)}{\mathrm{d}\boldsymbol{\theta}_D} = \Big(A^{-1}(\boldsymbol{\theta}_D)\Big)_{1:L} \nabla_{\boldsymbol{\theta}} \mathcal{L}^D(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)), \boldsymbol{\phi}_D),$$

where $\big(A^{-1}(\boldsymbol{\theta}_D)\big)_{1:L}$ denotes the $L \times L$-dimensional upper-left submatrix of

$$A^{-1}(\boldsymbol{\theta}_D) = \left[\frac{1}{\lambda}\frac{\mathrm{d}^2}{\mathrm{d}^2\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)}\mathcal{L}^P(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) + \begin{bmatrix} I_L & \mathbf{O}_{L\times P_P} \\ \mathbf{O}_{P_P\times L} & \mathbf{O}_{P_P\times P_P} \end{bmatrix}\right]^{-1}.$$

Proof (Corollary 5.6): Recall that $\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega})$ can be expressed by the simple linear projection in (5.8), and how (5.14) expresses the IJ with the specified regularization objective. We then compute

$$\begin{aligned}
\frac{\mathrm{d}\mathcal{L}^D(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)),\boldsymbol{\phi}_D)}{\mathrm{d}\boldsymbol{\theta}_D} &= \frac{\mathrm{d}\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))}{\mathrm{d}\boldsymbol{\theta}_D} \nabla_{\boldsymbol{\theta}} \mathcal{L}^D(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)), \boldsymbol{\phi}_D) \\
&= \frac{\mathrm{d}[I_L\ \mathbf{O}_{L\times P_P}]\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)}{\mathrm{d}\boldsymbol{\theta}_D} \nabla_{\boldsymbol{\theta}} \mathcal{L}^D(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)), \boldsymbol{\phi}_D) \\
&= \frac{\mathrm{d}\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)}{\mathrm{d}\boldsymbol{\theta}_D} \begin{bmatrix} I_L \\ \boldsymbol{O}_{P_P\times L} \end{bmatrix} \nabla_{\boldsymbol{\theta}} \mathcal{L}^D(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)), \boldsymbol{\phi}_D) \\
&= [I_L \quad \mathbf{O}_{L\times P_P}] A^{-1}(\boldsymbol{\theta}_D) \begin{bmatrix} I_L \\ \boldsymbol{O}_{P_P\times L} \end{bmatrix} \nabla_{\boldsymbol{\theta}} \mathcal{L}^D(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)), \boldsymbol{\phi}_D) \\
&= \Big(A^{-1}(\boldsymbol{\theta}_D)\Big)_{1:L} \nabla_{\boldsymbol{\theta}} \mathcal{L}^D(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)), \boldsymbol{\phi}_D).
\end{aligned}$$

■

This expression at first appears to resemble the one obtained in the setting without a sub-parameter map in (4.9). However, there is a critical difference: the matrix $A^{-1}(\boldsymbol{\theta}_D)$ encodes second-order derivatives involving **both** the backbone and pretext head parameters (as seen in (5.14)). So, although only the first L rows of $A^{-1}(\boldsymbol{\theta}_D)$ are used in the above expression, these rows depend implicitly on all its entries through the inversion, including those involving $\boldsymbol{\phi}_P$.

5.3.4 Approximating the Updated IJ

To approximate the upper-level gradient in Corollary 5.6, we suggest approximating the full matrix-vector product

$$A^{-1}(\boldsymbol{\theta}_D)\begin{bmatrix}\nabla_{\boldsymbol{\theta}}\mathcal{L}^D(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)),\boldsymbol{\phi}_D)\\ \mathbf{0}_{P_P}\end{bmatrix},$$

using the conjugate gradient method as outlined in Algorithm 1, where $\mathbf{0}_{P_P}$ denotes the P_P-dimensional zero vector. This approach ensures that the CG method captures interactions for all entries in $A(\boldsymbol{\theta}_D)$, including those involving the pretext head $\boldsymbol{\phi}_P$. After the CG method has converged towards an approximate solution, the final step is to discard the last P_P entries of the output vector, yielding the desired derivative with respect to the backbone parameters solely.

A core advantage of this approach is that it integrates almost seamlessly with the existing BiSSL setup. Aside from modifying the CG approximation to include additional components as described above, no additional structural changes are required. Experiments using this updated setup are documented in Section 6.5 of Chapter 6.

5.4 Applying BiSSL on GPT

Although the BiSSL framework is presented as domain-agnostic, its evaluation has so far been limited to computer vision tasks. Given the demonstrated success of large-scale self-supervised pretraining and downstream fine-tuning in NLP [27, 30, 34], it is of clear interest to investigate whether BiSSL can offer similar benefits in this domain. In this project, we apply BiSSL on the original GPT [30], as detailed in Chapter 2.

The absence of an explicit pretext head with dedicated parameters (see the last paragraph of Section 2.3.3) may make it appear as if BiSSL is incompatible with GPT. However, BiSSL actually imposes no requirement that the pretext head contain separate trainable parameters. It only assumes that a separation between the backbone and head *parameters* can be made based on how training objectives are defined over parameterized models. Hence, as long as the loss function $\mathcal{L}^P$ captures the complete pretext task, the vector $\boldsymbol{\phi}_P$ representing the pretext head parameters may simply be empty. In this case, the lower-level optimization problem in (4.2) reduces to

$$\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D) \in \operatorname*{argmin}_{\boldsymbol{\theta}_P} \mathcal{L}^P(\boldsymbol{\theta}_P) + \lambda r(\boldsymbol{\theta}_D, \boldsymbol{\theta}_P),$$

which does not affect the validity of Proposition 4.2 concerning how to express the IJ.

5.4.1 Embedding Matrix and Token-Specific Parameter Partitioning

A further consideration in adapting BiSSL to NLP involves how the token embedding matrix W_e^D is treated by the upper-level. In GPT, this matrix acts both as the input embedding and as the output projection layer. However, downstream tasks may introduce new tokens that are not present during pretraining, as outlined back in Section 2.3.5. To accommodate this, we partition the embedding matrix $W_e^D \in \mathbb{R}^{V+\tilde{V}}$ as follows:

- The first V rows, corresponding to tokens seen during pretraining, are assigned to the backbone parameters $\boldsymbol{\theta}$.

- The remaining $\tilde{V}$ rows, associated with new downstream-specific tokens, are assigned to the downstream head $\boldsymbol{\phi}_D$.

This setup at first appears to violate the original assumptions stated in Section 4.1.1 that the downstream head $h_{\boldsymbol{\phi}_D}$ is applied to the output of the backbone model, i.e., $h_{\boldsymbol{\phi}_D} \circ f_{\boldsymbol{\theta}}$, since the token-embeddings are applied prior to the backbone model. Fortunately, this structural assumption is actually not required for the theoretical results underpinning BiSSL. Specifically, the downstream head plays no role in the derivation of the IJ, which is evident in the proof of Proposition 4.2. Hence, redefining the downstream head to include components that are placed in advance of the backbone, as done with token embeddings, does not violate any theoretical foundations of the framework. Consequently, BiSSL can be applied directly to this setting without the need of any modification of theoretical formulations and assumptions.

6 | Experiments and Results: Ablations and Design Modifications

This chapter presents the experimental setup and results corresponding to the hypotheses and framework modifications discussed in Sections 5.1 to 5.3 of Chapter 5. We begin by outlining the implementation details, which are followed by a series of extensions and ablation studies that test the sensitivity of BiSSL to its key hyperparameters, as discussed in Section 5.1. We then present experiments evaluating strategies for adapting the lower-level weight regularization scaling λ dynamically during training as discussed in Section 5.2. Finally, we examine the effect of modifying the implicit Jacobian (IJ) approximation to include pretext head parameters, as proposed in Section 5.3.

Throughout, we briefly comment on the empirical results and their immediate implications. However, a more comprehensive discussion is deferred to Chapter 8.

6.1 Default Implementation Details

This section outlines the default experimental setup, which largely mirrors that of our previous work [49] to ensure comparability with its reported baselines. Subsequent sections introduce targeted deviations, each detailed in context. We begin by describing the vision datasets used in the experiments, then the training procedure for the conventional self-supervised pipeline baseline, followed by the BiSSL implementation. Finally, we present a reduced-scale variant of the default setup, used in cases where computational or time constraints prevent full-scale experiments.

6.1.1 Datasets Overview

We adopt the same general data pipeline as the original BiSSL work [49], with a few simplifications due to computational constraints. This includes choices of datasets for both pretraining and downstream evaluation.

Pretext Task Data

For self-supervised pretraining, we use the ImageNet-1K dataset [2], consistent with the original setup. When the full-scale setup is unnecessary or computationally prohibitive, we substitute a reduced variant of STL-10 [66], detailed in Section 6.1.4. The ImageNet dataset is used for pretraining unless otherwise stated.

We use a data augmentation pipeline closely aligned with that of BYOL [21] and VICReg [23]. The augmentation sequence is as follows:

1. Random cropping of a uniformly sampled area with a size ratio between 0.5 and 1, followed by resizing the image to size 96×96 for both inputs.
2. Horizontal flip with probability 0.5 for both inputs.
3. Color jittering of brightness 0.4, contrast 0.4, saturation 0.2 and hue 0.1 done in a random order, with probability 0.8 for both inputs.
4. Grayscale with probability 0.2 for both inputs.
5. Gaussian blur with kernel size 23 and probability 1.0 for one input and 0.1 for the other.
6. Solarization with probability 0.0 for the first augmentation and 0.2 for the other.
7. Color normalization with the ImageNet mean (0.485,0.456,0.406) and standard deviation (0.229,0.224,0.224) for both inputs.

Downstream Task Data

For downstream evaluation, we employ a subset of four classification tasks from the twelve originally used in [49]. This reduction is motivated both by resource and time constraints and by the project's focus on analyzing variations within BiSSL rather than benchmarking against other baselines.

This selection includes datasets previously used in ablation studies of the original paper, enabling easier comparison. The selected datasets are:

- PASCAL VOC 2007 (VOC07) [67]
- Describable Textures Dataset (DTD) [68]
- Oxford-IIIT Pets (Pets) [54]
- Oxford 102 Flowers (Flowers) [69]

This subset offers a representative mix of both coarse and fine-grained natural image classification tasks.

A simpler augmentation scheme is applied for downstream training. Images are center cropped to size 96×96 with a minimal crop ratio of 0.5, followed by the image being randomly flipped horizontally. Lastly, the image is normalized with the same mean and standard deviations used for the pretext data augmentations.

For DTD and Flowers, we use the official validation partitions. For VOC07 and Pets, we reuse the splits from [49], which reserves approximately 20% of the original training data as a validation set.

6.1.2 Baseline Setup: Conventional Self-Supervised Training Pipeline

This section describes the baseline setup that will be compared with BiSSL. The setup mirrors the training pipeline previously outlined in the left-side diagram of Figure 4.1.

Pretext Training

We adopt SimCLR [19] as the pretext task, introduced in Section 2.2 of Chapter 2. A ResNet-50 [4] backbone is pretrained using the NT-Xent loss (2.1) with a temperature parameter $\tau = 0.5$. The projection head comprises three linear layers, each with 256 units and batch normalization layers in between.

Optimization is performed using the LARS optimizer [70] with a trust coefficient of 0.001, momentum 0.9, and weight decay 10^{-6}. The learning rate linearly warms up from 0 to 4.8 over the first 10 epochs, followed by a cosine decay to 0 with no restarts [71]. Training is done for 500 epochs with a batch size of 1024.

To reduce computational cost and environmental impact, we reuse the pretrained weights from the original BiSSL implementation for all experiments.

Downstream Fine-Tuning

For downstream tasks, a linear classification head is appended to the pretrained backbone, consisting of a single fully connected layer with output dimensionality equal to the number of classes in the task. Training uses the cross-entropy loss, stochastic gradient descent (SGD) [72] with momentum 0.9, and a cosine decaying learning rate schedule without restarts [71]. The batch size is set to 256, and models are fine-tuned for 400 epochs.

Hyperparameter optimization is conducted via random sampling of H values over a predefined space. Namely, each trial samples the learning rate and weight decay from log-uniform distributions ranging from 10^{-4} to 1 and from 10^{-5} to 10^{-2}, respectively. Though we used $H = 100$ in [49], we will either use $H = 50$ or $H = 25$ for the experiments in this project. Validation performance is assessed after every epoch using the top-1 and top-5 accuracy (or the 11-point mean average precision (mAP) for VOC07 [67]). The configuration yielding the best overall trade-off between validation loss and accuracy is selected.

For test-time evaluation, we retrain 10 models using the selected hyperparameter configuration. Model checkpoints are stored only when top-1 validation accuracy (or mAP for VOC07) increases its all-time previous maximal accuracy. Final test results are reported as means and standard deviations of top-1 and top-5 accuracy (or solely mAP for VOC07) across these 10 runs.

6.1.3 Training Setup Using BiSSL

The BiSSL-based training follows the pipeline depicted on the right side of Figure 4.1. This section outlines the training procedure for each stage in that pipeline.

Pretext Task

The pretext training phase is identical to the baseline setup described in Section 6.1.2. As such, we again reuse the same pretrained weights.

Downstream Head Warm-up

This stage mirrors the downstream fine-tuning setup described in Section 6.1.2, with one key difference: only the classification head is trained while the backbone remains frozen. Rather than performing a separate hyperparameter search, we use the learning rate and weight decay values identified as optimal in the baseline fine-tuning setup as a starting point, where minor adjustments are occasionally made based on preliminary experimentation. The warm-up is run for 20 epochs with no scheduling on the learning rate.

BiSSL

The implementation of the BiSSL algorithm, generally outlined in Algorithm 2, is specified by first summarizing configurations specific to the lower and upper training stages, followed by a description of the composite configuration for the BiSSL implementation.

Lower-Level The configuration of the lower-level broadly aligns with the pretext training procedure described in Section 6.1.2, with a few key modifications. As per formulation in the lower-level problem (4.2), its objective is composed of the original pretext task loss $\mathcal{L}^P$ (the NT-Xent loss from (2.1) in this case) and a regularization term that penalizes the distance between the upper- and lower-level backbone parameters. We use $r(\boldsymbol{\theta}_D, \boldsymbol{\theta}_P) = \frac{1}{2}||\boldsymbol{\theta}_D - \boldsymbol{\theta}_P||_2^2$, with the weighting fixed at $\lambda = 0.001$. Differing from the original pretext training, the linear learning rate warm-up now extends across $N_L \cdot 10$ steps, where N_L denotes the number of gradient iterations in the lower-level training stage.

Upper-Level The upper-level shares most of its configuration with the downstream fine-tuning setup in Section 6.1.2, except where noted here. The learning rate and weight decay are inherited from the downstream head warm-up stage. To approximate the first term of the upper-level gradient in Equation (4.9), we use the conjugate gradient method outlined in Algorithm 1, with the Hessian-vector product calculator presented in Algorithm 3. It utilizes a randomly sampled lower-level batch $\mathbf{z}$. The input vector $\mathbf{v}$ in Algorithm 1, is this case the gradient of the downstream loss with respect to the lower-level backbone, evaluated at $\boldsymbol{\theta} = \boldsymbol{\theta}_P$, i.e., $\mathbf{v} = \nabla_{\boldsymbol{\theta}} \mathcal{L}^D(\boldsymbol{\theta}, \boldsymbol{\phi}_D)|_{\boldsymbol{\theta}=\boldsymbol{\theta}_P}$. The conjugate gradient procedure uses $N_c = 5$ iterations and includes a damping term $\lambda_{\text{damp}} = 10$ to improve convergence and numerical stability.

Algorithm 3 Hessian Vector Product Calculation f_H (To use in Algorithm 1)

1: **Input:** Input vector $\mathbf{x}$, Model parameters $\boldsymbol{\theta}_P$, $\boldsymbol{\phi}_P$, Training Objective $\mathcal{L}^P$, lower-level data batch $\mathbf{z}$, Regularization Weight λ and dampening λ_{damp}.

2: $\pi(\boldsymbol{\theta}_P) \leftarrow \left(\nabla_{\boldsymbol{\theta}} \mathcal{L}^P\left(\boldsymbol{\theta}, \boldsymbol{\phi}_P; \mathbf{z}\right)|_{\boldsymbol{\theta}=\boldsymbol{\theta}_P}\right)^T \mathbf{x}$

3: $\mathbf{g} \leftarrow \nabla_{\boldsymbol{\theta}} \pi(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_P}$ ▷ Memory efficient calculation of $\nabla^2_{\boldsymbol{\theta}} \mathcal{L}^P(\boldsymbol{\theta}, \boldsymbol{\phi}_P; \mathbf{z})|_{\boldsymbol{\theta}=\boldsymbol{\theta}_P} \mathbf{x}$.

4: $\mathbf{y} \leftarrow \mathbf{x} + \frac{1}{\lambda + \lambda_{\text{damp}}} \mathbf{y}$

5: **Return:** $f_H(\mathbf{x}) := \mathbf{y}$

Composite Configuration Details Both backbones $\boldsymbol{\theta}_P$ and $\boldsymbol{\theta}_D$ are initialized using the pretrained backbone from the original self-supervised training. The lower-level conducts

$N_L = 20$ gradient steps before alternating to the upper-level, which then conducts $N_U = 8$ gradient steps. This cycle is repeated for a total of $T = 500$ alternations. Given that the ImageNet training set consists of 1251 batches (with the current batch size of 1024), these $T = 500$ alternations correspond to roughly $\frac{TN_L}{1251} = \frac{500 \cdot 20}{1251} \approx 8$ conventional pretext epochs. This is considered a negligible overhead relative to the 500 pretraining epochs used in the baseline.

To prevent data from being reshuffled between successive training stage alternations, we maintain separate stacks for the batched lower- and upper-level datasets. These stacks are regenerated only when remaining batches are insufficient to complete the scheduled number of gradient steps (i.e., $N_L = 20$ for the lower-level). For smaller downstream datasets where fewer than $N_U = 8$ distinct batches can be formed, the data is reshuffled and the stacks are remade once the remaining number of examples falls below the upper-level batch size (in this case 256). Gradients with norms exceeding 10 are clipped to ensure numerical stability.

Downstream Fine-Tuning

To ensure comparability, the final downstream fine-tuning stage mirrors the fine-tuning procedure described in Section 6.1.2, except for the fact that the backbone is initialized instead with the lower-level backbone obtained from the previous BiSSL stage.

6.1.4 Small-scale Configuration

For certain experiments where the default setup is excessive or computationally infeasible, we adopt a reduced configuration referred to as the small-scale setup. The small-scale setup mirrors the structure of the default setup as closely as possible, with a few modifications aimed at reducing computational demands, namely:

- **Backbone model:** A smaller ResNet-18 [4] architecture is used as the backbone instead of a ResNet-50.
- **Pretraining dataset:** The unlabeled portion of the smaller-scale STL10 dataset [66] is used in place of ImageNet for self-supervised pretraining and lower-level optimization of BiSSL. This dataset consists of 100,000 natural images of resolution 96×96.
- **Training duration adjustment:** With the current batch size of 1024, the STL10 dataset yields a total of 98 training batches. Thus, conducting $T = 500$ BiSSL training stage alternations is approximately equivalent to $\frac{TN_L}{98} = \frac{500 \cdot 20}{98} \approx 100$ epochs of conventional pretraining in this setting. To ensure comparability, the baseline models used for evaluation are pretrained for a total of 600 epochs by adding 100 extra epochs to their original schedule, ensuring that the number of effective pretext training steps roughly matches.

Apart from these adjustments, all other configurations remain consistent with the default setup described in Section 6.1.1, 6.1.2, and 6.1.3. For clarity, we emphasize that *the small-scale setup is only employed when explicitly stated.*

Table 6.1: Baseline results for the default setup. Top-1 accuracy is reported, except for VOC07, where the 11-point mAP is used. Significant improvements are shown in bold.

	VOC07	DTD	Pets	Flowers
Only FT [49]	71.0 ± 0.1	60.3 ± 0.9	73.2 ± 0.3	82.6 ± 0.3
BiSSL + FT	$\mathbf{71.4 \pm 0.1}$	$\mathbf{63.8 \pm 0.3}$	$\mathbf{77.7 \pm 0.5}$	$\mathbf{84.2 \pm 0.3}$
Avg Diff	**+0.4**	**+3.5**	**+4.5**	**+1.6**

Table 6.2: Original and sweep values for each hyperparameter. Bolded entries indicate the default/original values. See Algorithm 2 for details on how the parameters λ, T, N_U and N_L are used in BiSSL. *: For $N_U = 1$, we'll reuse the result from [49].

Parameter	**Original**	**Sweep**
λ	0.001	$\{1, 0.1, 0.01, \mathbf{0.001}, 0.0001\}$
T	500	$\{100, 200, 300, 400, \mathbf{500}, 600, 800, 1000\}$
N_U	8	$\{1^*, 2, 4, \mathbf{8}, 10, 12\}$
N_L	20	$\{1, 2, 5, 10, \mathbf{20}, 30, 40, 50\}$
FT Epochs	400	$\{5,10, 25, 50, 100, 200, 300, \mathbf{400}\}$

6.2 Baselines

We reuse the standard fine-tuning baseline from [49], as our default experimental setup is identical in all relevant aspects. The standard BiSSL configuration results are reproduced to ensure full control and comparability across experiments, accounting for any potential changes in the implementation that may have occurred in the meantime. We conduct the hyperparameter grid search for $H = 50$ combinations.

Table 6.1 summarizes the baseline results for the default setup. BiSSL consistently outperforms standard fine-tuning across all four datasets.

6.3 Hyperparameter Influence

This section presents the experimental setup and results of the hyperparameter studies proposed in Section 5.1. The goal is to better understand how key hyperparameters involved in BiSSL affect downstream performance when varied. For each experiment in this section, we perform a grid search over learning rates and weight decay values using $H = 25$ combinations and document top-1 classification accuracies.

For each hyperparameter studied, we vary it while keeping all other hyperparameters fixed and identical to those outlined in the default setup. Table 6.2 lists the sweep values to be evaluated for each hyperparameter. As each variation of a hyperparameter involves conducting BiSSL and subsequent fine-tuning from scratch, we only document results regarding BiSSL-specific hyperparameters (λ, T, N_U, and N_L) on the Pets dataset to spare computational resources. For the experiments involving the number of fine-tuning epochs, we report results across all four downstream datasets (Pets, VOC07, DTD, and Flowers). Results in this section are presented in figures, but full tabulated values can be found in Section F.1.1 of Appendix F.

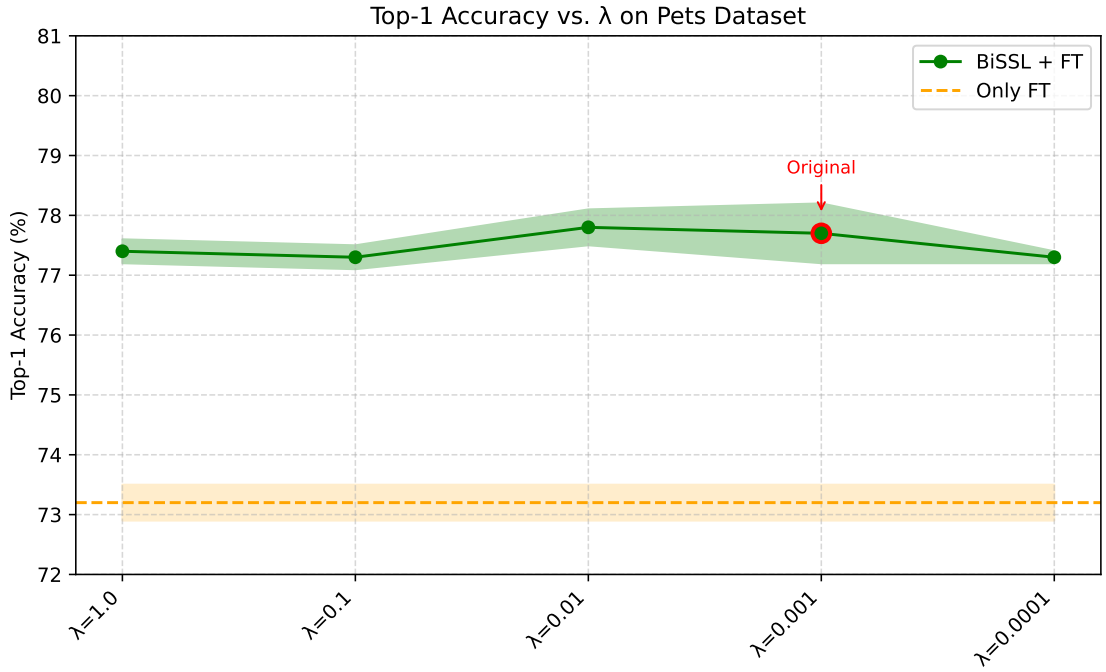

Figure 6.1: Impact of varying the lower-level regularization strength λ on downstream top-1 classification accuracy (Pets dataset). See Table F.1 in Appendix F for tabulated results.

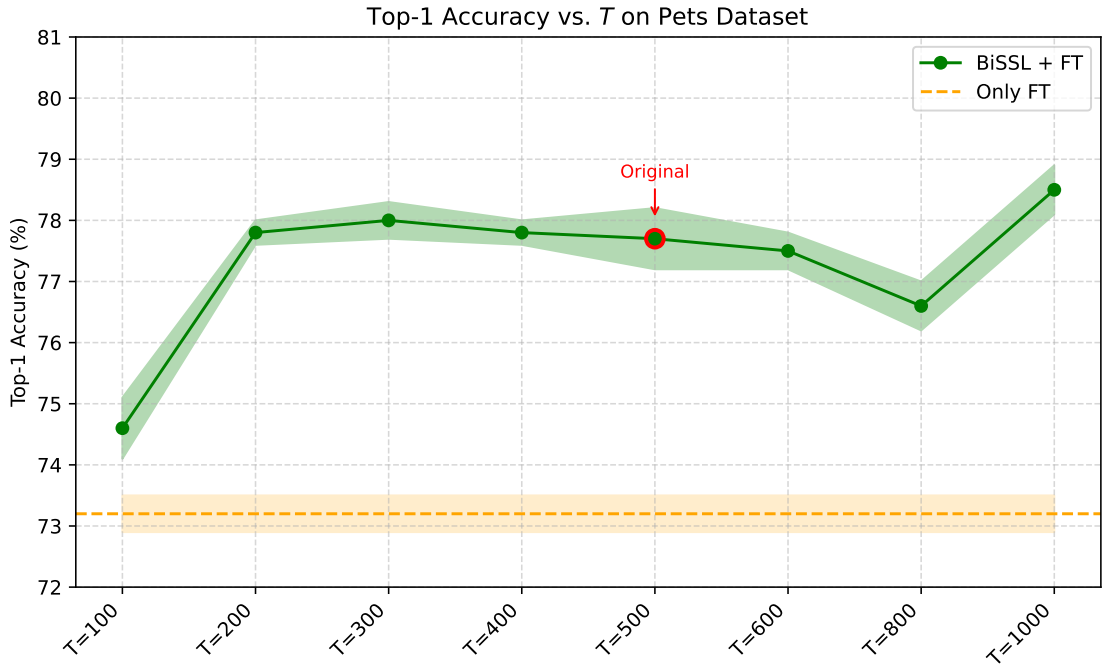

Figure 6.2: Impact of varying the number of training stage alternations T on downstream top-1 classification accuracy (Pets dataset). See Table F.2 in Appendix F for tabulated results.

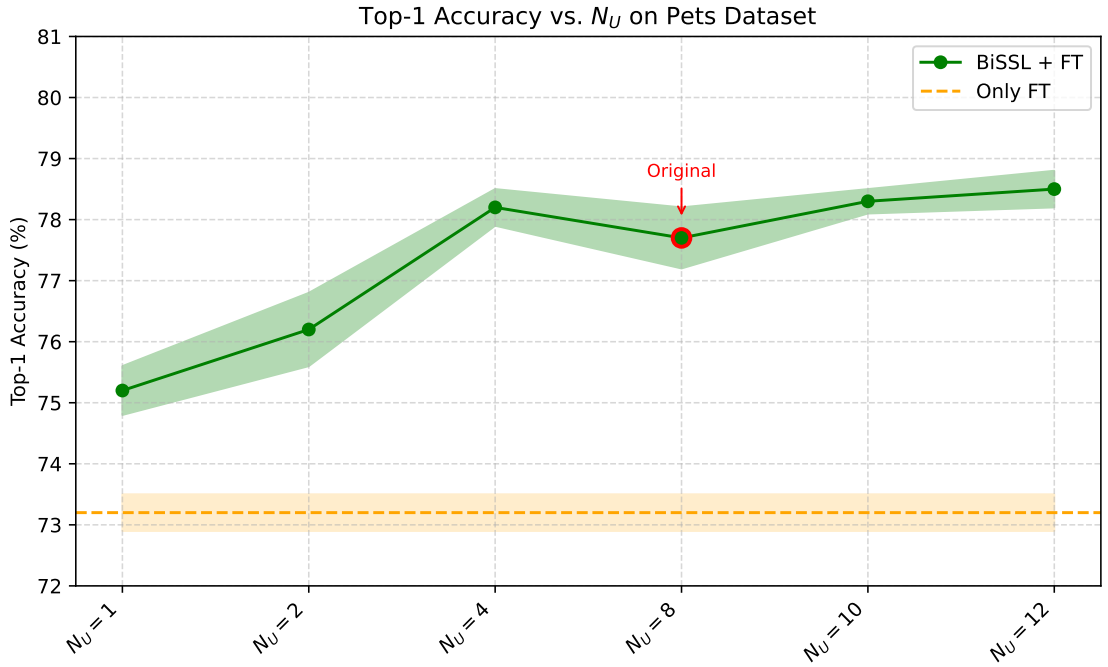

Figure 6.3: Impact of varying the number of upper-level iterations N_U on downstream top-1 classification accuracy (Pets dataset). See Table F.3 in Appendix F for tabulated results.

BiSSL Specific Hyperparameters Figures 6.1, 6.2, 6.3 and 6.4 display the Top-1 accuracies on the Pets dataset when varying the hyperparameters λ, T, N_U, and N_L respectively, compared with the conventional SSL pipeline baselines documented in Table 6.1. Adjusting λ shows no significant effect on downstream performance, suggesting the method is robust to the choice of regularization strength within the tested range.

For the temperature parameter T in Figure 6.2, we observe that using $T = 200$ yields comparable performance to the default value of $T = 500$, which indicates that BiSSL can achieve similar results with substantially less computational effort. Overall, these results imply that as long as T is chosen sufficiently large, BiSSL attains improved performance.

Regarding the number of upper-level optimization steps N_U in Figure 6.3, improvements plateau once N_U reaches 4. Since the upper-level optimization is the most computationally intensive part of the process, reducing N_U from 8 to 4 offers a prominent efficiency gain without sacrificing accuracy in this example.

Varying the number of lower-level optimization steps N_L as seen in Figure 6.4 shows only a small performance increase from $N_L = 1$ to $N_L = 5$, with no improvements afterwards. This suggests that $N_L = 5$ or even $N_L = 2$ is a reasonable choice, and calls into question whether the original setting of $N_L = 20$ is excessive.

Number of Downstream Fine-Tuning Epochs Figure 6.5 presents the final ablation study, showing downstream performance over a varying number of fine-tuning epochs. Since these experiments are relatively inexpensive to run, we include results for all four datasets. For each configuration, both BiSSL and the conventional baseline were trained under the default settings, varying only the number of fine-tuning epochs, using $H = 25$.

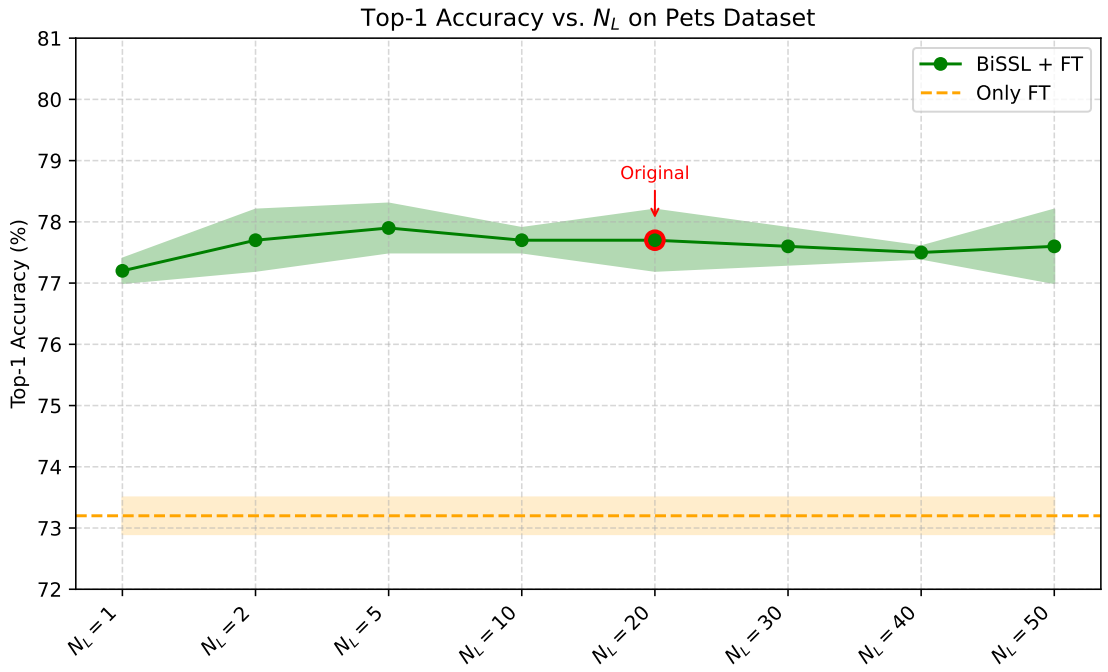

Figure 6.4: Impact of varying the number of lower-level iterations N_L on downstream top-1 classification accuracy (Pets dataset). See Table F.3 in Appendix F for tabulated results.

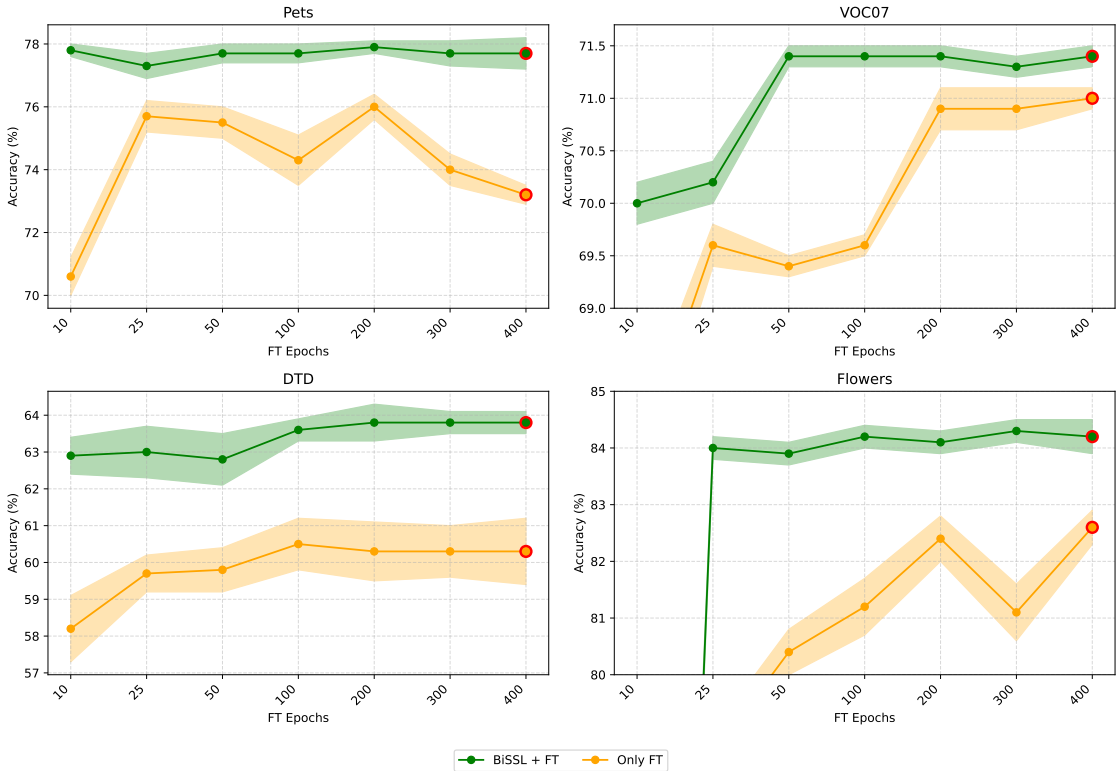

Figure 6.5: Impact of varying the number of fine-tuning epochs on downstream top-1 classification accuracy across the Pets, VOC07, DTD, and Flowers datasets. For VOC07 and Flowers, the accuracy-axes are cropped to better highlight differences among the highest achieved accuracies. See Table F.5 in Appendix F for tabulated results.

While each dataset exhibits its own characteristic behavior, BiSSL consistently yields robust downstream accuracy once a sufficient number of fine-tuning epochs is reached, and the accuracy remains stable thereafter. In contrast, the Only FT baseline is more sensitive to this parameter, in some cases showing less consistent improvements.

The baseline on the Pets dataset in the upper-leftmost part achieves its best performance with very few fine-tuning epochs and degrades as training continues, implying possible overfitting. On the other hand, BiSSL attains its performance consistently and outperforms the baseline regardless of the epoch count.

For the DTD dataset in the lower-leftmost part, a similar pattern emerges, though the baseline does not deteriorate with more fine-tuning. Both methods improve with longer training up to about 100 epochs, after which performance saturates. Again, BiSSL remains consistently superior.

BiSSL converges earlier for the VOC07 dataset in the upper-rightmost part, reaching peak performance around 50 epochs. The baseline also improves with more epochs until plateauing at 200 epochs.

Finally, for the Flowers dataset in the lower-rightmost part of the figure, BiSSL reaches strong performance quickly and sees negligible gains after 25 epochs. The baseline shows substantial variance across configurations, however. While it's less clear whether performance has stalled, the fluctuations suggest that more extensive hyperparameter tuning might yield improvements for the baseline in this case.

6.4 Adaptive Scaling of λ

This section investigates a modified BiSSL configuration where the lower-level regularization weight λ is allowed to vary during training. We evaluate the strategies for adaptively setting λ described in Section 5.2 of Chapter 5, including both scheduling and making it learnable.

6.4.1 Scheduling λ

We examine the effect of scheduling λ using the suggested schedulers from Section 5.2.1 with a fixed minimum value $\lambda_{\min} = 10^{-8}$ and varying the maximum value $\lambda_{\max}$. Since this involves sweeping over several combinations of schedulers and varying $\lambda_{\max}$, we conduct these experiments using the small-scale setup to reduce computational cost. $H = 25$ is used for the hyperparameter search.

Cosine Schedulers As visualized in Figure 5.1, we consider both increasing and decreasing cosine schedules for λ, with values developing between a fixed $\lambda_{\min} = 10^{-8}$ and varying $\lambda_{\max}$. Figure 6.6 presents the downstream accuracy under these schedulers.

The results suggest that the performance is highly sensitive to the choice of $\lambda_{\max}$. Poorly chosen values can lead to substantial drops in accuracy, in many cases even underperforming the conventional fine-tuning baseline. The best results are achieved with $\lambda_{\max} = 0.002$, which slightly (though not significantly) outperforms the standard BiSSL configuration. This value is particularly notable because the average value of $\lambda(t)$ under the cosine schedule equals 0.001, which corresponds to the default value when λ is fixed in BiSSL.

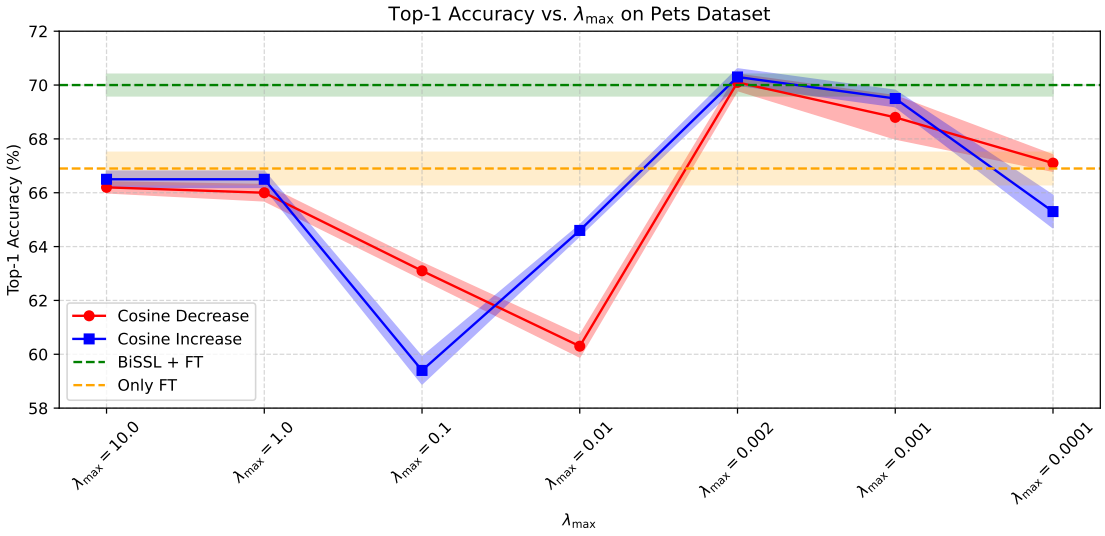

Figure 6.6: Impact of applying cosine scheduling strategies to the lower-level regularization strength λ on downstream Top-1 classification accuracy (Pets dataset and small-scale setup). Results are shown for both increasing and decreasing cosine schedules across different values of $\lambda_{\max}$, with $\lambda_{\min}$ fixed at 10^{-8}. The conventional Only FT, as well as the BiSSL baseline (fixed $\lambda = 0.001$), is included for comparison. See Table F.6 in Appendix F for tabulated results.

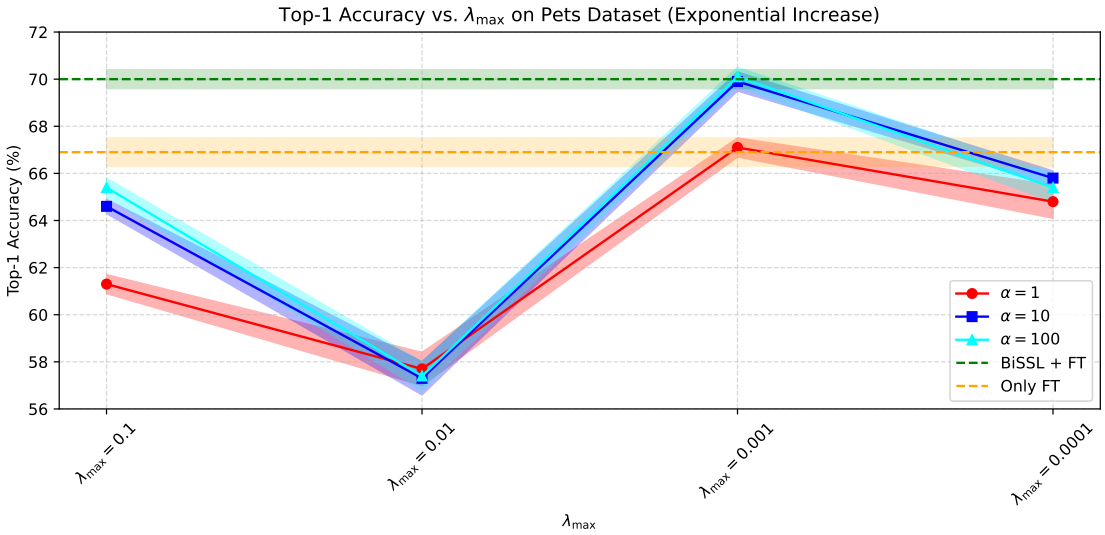

Figure 6.7: Impact of applying exponentially increasing scheduling to the lower-level regularization strength λ on downstream Top-1 classification accuracy (Pets dataset and small-scale setup). Results are shown for $\alpha \in \{1,10,100\}$ across different values of $\lambda_{\max}$, with $\lambda_{\min}$ fixed at 10^{-8}. The conventional Only FT, as well as the BiSSL baseline (fixed $\lambda = 0.001$), is included for comparison. See Table F.7 in Appendix F for tabulated results.

Table 6.3: Downstream top-1 classification accuracy for BiSSL using a cosine-increasing schedule with $\lambda_{\max} = 0.002$ and an exponential-increasing schedule with $\alpha = 100$ and $\lambda_{\max} = 0.001$, evaluated on the full-scale default setup across the VOC07, DTD, Pets, and Flowers datasets.

	VOC07	DTD	Pets	Flowers
Only FT [49]	71.0 ± 0.1	60.3 ± 0.9	73.2 ± 0.3	82.6 ± 0.3
BiSSL+FT (Original, Fixed $\lambda = 0.001$)	71.4 ± 0.1	63.8 ± 0.3	77.7 ± 0.5	84.2 ± 0.3
BiSSL+FT (Exp.Inc., $\alpha = 100$, $\lambda_{\max} = 0.001$)	71.2 ± 0.1	62.9 ± 0.2	77.3 ± 0.2	84.3 ± 0.1
BiSSL+FT (Cos.Inc., $\lambda_{\max} = 0.002$)	70.2 ± 0.1	61.0 ± 0.3	76.5 ± 0.4	80.9 ± 0.2

We furthermore observe that setting $\lambda_{\max} = 1$ or $\lambda_{\max} = 10$ results in performance matching that of the Only FT baseline. This aligns well with the interpretation back in Section 4.2.3, implying that large values of λ essentially force BiSSL to conduct conventional fine-tuning. Note that this appears contradictory to the previous results for setting a fixed $\lambda = 1$ in Figure 6.1. We provide further insights into this in the discussion of Chapter 8.

All in all, no clear advantage of increasing over decreasing cosine schedules is observed from the results. Hence, these findings overall suggest that cosine scheduling provides no clear benefit over keeping λ fixed.

Exponential Schedulers As visualized in Figure 5.2, we consider exponential scheduling strategies for $\lambda(t)$ with $\lambda_{\min} = 10^{-8}$, sharpness $\alpha \in \{1, 10, 100\}$ and varying $\lambda_{\max} \in \{0.1, 0.01, 0.001, 0.0001\}$. Figure 6.7 presents downstream performance using these exponential schedules on the Pets dataset.

Similar to the cosine schedules, the exponential schedulers appear highly sensitive to the choice of $\lambda_{\max}$. When setting $\lambda_{\max} = 0.001$, which roughly aligns the average λ for the exponential *increasing* scheduler with the default constant value of 0.001, performance remains comparable to the BiSSL baseline. However, in all other tested configurations, scheduling adversely impacts performance to the extent that it underperforms even the conventional fine-tuning baseline.

Scheduling on Default Setup The earlier scheduling experiments were run on the small-scale setup, leaving it unclear whether their conclusions carry over to the default configuration. Based on validation accuracy, the exponential increase schedule with $\alpha = 100$ and $\lambda_{\max} = 0.001$ arose as the best-performing choice, which motivated us to re-evaluate this configuration on the full default setup. Since this schedule quickly approaches an almost fixed value, mirroring the original setup, we also wanted to include a slower-rising alternative for contrast. For this, we selected the cosine-increasing schedule with $\lambda_{\max} = 0.002$.

Table 6.3 shows the results. In general, the scheduled variants perform either worse or on par with the standard BiSSL configuration. For the cosine-increasing schedule in particular, the performance drop is substantial enough that in two out of four datasets, it even falls below the Only FT baseline. These findings reinforce the conclusion that scheduling λ increases downstream volatility without offering consistent benefits. In some cases, scheduling may even negate the advantages of BiSSL when compared to both fixed-λ and conventional fine-tuning.

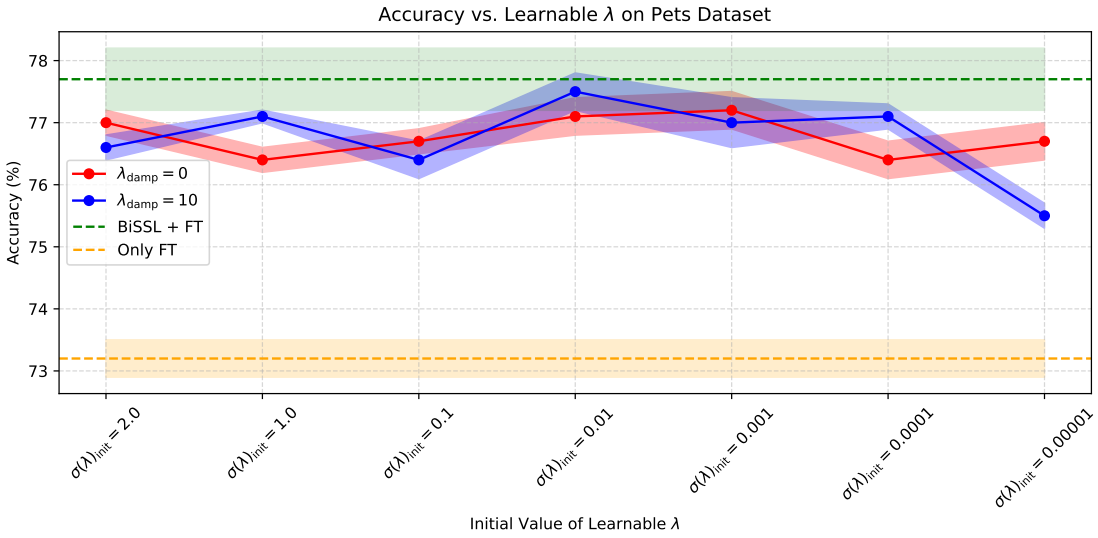

Figure 6.8: Downstream top-1 accuracy on the Pets dataset using learnable λ with $\lambda_{\text{damp}} = 0$ and $\lambda_{\text{damp}} = 10$ for different initializations. See Table F.8 in Appendix F for tabulated results.

6.4.2 Learnable λ via Upper-Level Optimization

The earlier attempts to adapt λ did not yield clear improvements. As a final approach, we instead allow BiSSL to learn λ directly by treating it as an upper-level parameter, following the formulation introduced in Definition 5.1 of Section 5.2.2. We use the softplus function $\sigma(x) = \log(1 + e^x)$, and update λ using the upper-level gradient in (5.6). The same optimizer is used as for other upper-level parameters. We test two configurations, using $\lambda_{\text{damp}} = 0$ and $\lambda_{\text{damp}} = 10$, to also evaluate the effect of dampening. Experiments are conducted on the default setup, varying the initial value of $\sigma(\lambda)$.

Figure 6.8 shows the downstream accuracy on the Pets dataset for different initializations. The results indicate that learning λ does not lead to any clear improvement in downstream performance. Initial values around $\sigma(\lambda)_{\text{init}} = 0.01$ or $\sigma(\lambda)_{\text{init}} = 0.001$ appear to provide the best results, but the variation is minor. Unlike the scheduler-based strategies, however, the performance never drops below that of the Only FT baseline. All configurations yield consistently stronger results, suggesting that while learning λ may not improve performance, it also does not introduce the same instability. Figures F.1 and F.2 in Appendix F illustrate how λ evolves during training for the two dampening settings.

6.5 Non-Fixed Pretext Head During IJ Approximation

This section presents experiments using the alternative setup introduced i Section 5.3, where the pretext head parameters are not assumed fixed during the approximation of the IJ. The overall setup remains consistent with the default configuration, using $H = 50$. The main difference lies in the estimation of the upper-level gradient, where we instead approximate the gradient from Corollary 5.6. We follow the procedure suggested in Section 5.3.4 and update the implemented conjugate gradient (CG) solver accordingly. All other experimental settings remain identical to the default.

Table 6.4: Downstream accuracies comparing the default BiSSL setup with the variant discussed in Section 5.3, where the pretext head parameters are not assumed fixed during the IJ approximation. Results are documented across the VOC07, DTD, Pets, and Flowers datasets. No significant average differences are present.

BiSSL + FT	VOC07	DTD	Pets	Flowers
Original	71.4 ± 0.1	63.8 ± 0.3	77.7 ± 0.5	84.2 ± 0.3
Non-fixed $\boldsymbol{\phi}_P$ During IJ Approx.	71.3 ± 0.1	63.6 ± 0.3	78.4 ± 0.3	84.1 ± 0.1
Avg Diff	-0.1	-0.2	$+0.7$	-0.1

Table 6.4 compares the original BiSSL implementation with this modified approach, and the results show no significant difference in downstream accuracy between them. Further discussion of these findings is deferred to Chapter 8.

7 | Experiments and Results: NLP

This chapter presents experiments evaluating the impact of applying BiSSL to NLP tasks. We apply BiSSL on the GPT pretext task introduced in Section 2.3 of Chapter 2, where considerations regarding such implementation were addressed in Section 5.4 of Chapter 5. We begin by outlining relevant implementation details, followed by documenting downstream performance on a partition of the GLUE benchmark [50].

7.1 Implementation Details

We follow the original GPT implementation [30] as closely as possible. Any modifications needed to make it work within the BiSSL framework are highlighted explicitly. Broader general implementation details regarding BiSSL that are not specific to the NLP/GPT setup have already been covered in Chapter 6 and are not repeated here. Where relevant, we point the reader to the appropriate sections of that chapter for background.

7.1.1 Datasets Overview

All datasets used in this chapter are publicly available and are accessed via the Hugging Face *datasets* library [73].

Tokenization

We reuse the same tokenizer as the original GPT model. Specifically, we adopt the preconditioned `OpenAIGPTTokenizer` class provided by the Hugging Face `transformers` library [74], which employs both the tokenizer logic and the pretrained vocabulary of $V = 40{,}478$ tokens used by the original GPT model.

Pretext Task Data

The BooksCorpus dataset [75] will be used for self-supervised pretraining. It consists of unlabeled text from over 7,000 unique unpublished books, spanning various literature genres. As it contains long stretches of text, it is well-suited in this context as this allows the pretraining procedure to learn long-range dependencies. A context window of $k = 512$ is used, hence sentences from the pretraining corpus are divided into chunks of exactly 512 tokens.

Downstream Task Data

We train and evaluate models on a subset of the General Language Understanding Evaluation (GLUE) benchmark [50], which includes the following 7 datasets spanning three task types:

- *Classification:* The Corpus of Linguistic Acceptability (CoLA) [76] and Stanford Sentiment Treebank-2 (SST2) [77]. Both are binary classification tasks. CoLA involves classifying grammatical correctness, while the task of SST2 is to classify (positive or negative) sentiment of sentences.
- *Natural Language Inference:* Recognizing Textual Entailment (RTE) [78] and Question NLI (QNLI) [79]. Both datasets consist of sentence pairs containing a premise and a hypothesis. RTE involves binary classification of whether the premise entails the hypothesis, contradicts the hypothesis, or neither. QNLI consists of pairs of questions and context sentences, whereas the task is to assess whether the context sentence contains the answer to the question.
- *Sentence Similarity:* Microsoft Research Paraphrase Corpus (MRPC) [80], Quora Question Pairs (QQP) [81], and Semantic Textual Similarity Benchmark (STSB) [82]. These tasks assess semantic equivalence between sentence pairs. MRPC and QQP are binary classification tasks, while STSB is a regression task with similarity scores ranging from 1 to 5.

Each dataset is evaluated using its associated task-specific metrics, which implementations are provided by the Hugging Face `evaluate` library [83]. CoLA is evaluated using the Matthews correlation, STSB the Pearson correlation, MRPC and QQP the F1-score, and the rest the top-1 classification accuracy. For details on these metrics, we refer to the original GLUE source outlined above.

Although GLUE provides dedicated training, validation, and test splits, the test labels are not publicly accessible. We therefore reconfigure the original validation sets as test sets. For training and validation, we partition the first 80% for training, and the remaining 20% for validation in the ordering provided by Hugging Face. As the ordered datasets all appear to be pre-shuffled, we assume this split is effectively IID.

The GLUE benchmark also includes the MultiNLI Matched (MNLI-m), MultiNLI Mismatched (MNLI-mm) [84] and Winograd NLI (WNLI) [85] datasets, but these are excluded from experimentation in this project. MNLI-m and MNLI-mm share the same training partition but differ in their validation and test splits, making evaluation on MNLI-mm unreliable given the project's validation setup. Due to their relatively large size, they were also omitted to conserve computational resources, where we argue that the large-scale QQP dataset already serves to illustrate performance on a higher resource-demanding task. WNLI was excluded after preliminary testing showed that fine-tuning performance consistently degraded below random chance, indicating that the task is too difficult for this model class. In line with the original GPT paper, we do not report results on this dataset.

Downstream Input Formatting Tokenized inputs are constructed differently depending on task type. Let s, d, and e denote special start, delimiter, and end tokens, respectively. These special tokens expand the vocabulary beyond the $V = 40{,}478$ tokens to $V + V_D = 40{,}480$ for classification, and to $V + V_D = 40{,}481$ for entailment/similarity.

For classification tasks, a single sentence is wrapped with start and end tokens. Given a tokenized input $x_1, \ldots, x_N$, the model input is then $\mathbf{x} = [s, x_1, \ldots, x_N, e]^T$.

Entailment tasks require the input to capture the sentence ordering (a premise followed by a hypothesis). Given a tokenized premise $x_1^p, \ldots, x_{N_p}^p$ and hypothesis $x_1^h, \ldots, x_{N_h}^h$, the input is $\mathbf{x} = [s, x_1^p, \ldots, x_{N_p}^p, d, x_1^h, \ldots, x_{N_h}^h, e]^T$.

For similarity tasks, sentence order is not of importance. To ensure this is captured by the model, each input sequence does, in this case, yield two model inputs with reversed sentence orders. Given the tokenized sequences $x_1^a, \ldots, x_{N_a}^a$ and $x_1^b, \ldots, x_{N_b}^b$, the inputs are then $\mathbf{x}_1 = [s, x_1^a, \ldots, x_{N_a}^a, d, x_1^b, \ldots, x_{N_b}^b, e]^T$ and $\mathbf{x}_2 = [s, x_1^b, \ldots, x_{N_b}^b, d, x_1^a, \ldots, x_{N_a}^a, e]^T$. The model processes these inputs slightly differently, which is outlined in the "Downstream Fine-Tuning" subsection below.

To ensure compatibility with the pretrained GPT model, input sequences longer than the context window of $k = 512$ are discarded (which for each dataset is a negligible small fraction), and shorter sequences are padded.

7.1.2 Baseline Setup

Self-Supervised Pretraining

We reuse the publicly available GPT model implementation and pretrained weights provided by the `OpenAIGPTLMHeadModel` class of the Hugging Face `transformers` library [74], whose weights are achieved in a way that mirrors the original GPT pretraining setup. This section outlines key aspects of the pretraining process to support subsequent experimental configurations for BiSSL.

The model has approximately 117 million trainable parameters, with architecture-specific hyperparameters listed in Table B.1 of Appendix B. Training is performed using the Adam optimizer [86] with a batch size of 64 over a total of 100 epochs. The learning rate increases linearly from 0 over the first 2000 update steps, peaking at $2.5 \cdot 10^{-4}$, and then decays following a cosine schedule without restarts [71]. A weight decay of 0.01 is applied to all parameters except for biases and scale parameters in the layer normalizations. The pretext task training objective is as described back in Section 2.3.4.

Downstream Fine-Tuning

We fine-tune the pretrained model ourselves on all GLUE datasets to ensure that the resulting baselines are directly comparable with those obtained using BiSSL. Most aspects of the original fine-tuning procedure are mirrored in the implementation of this project, and we document all relevant details here.

Unless otherwise noted, we reuse the same hyperparameters as for self-supervised pretraining, which also includes using the Adam optimizer. We apply dropout [87] with probability 0.1 to the transformer outputs before the output layer(s). We use the mean square error training objective on the STSB dataset, and the cross-entropy on the rest. The original paper specifies a learning rate of $6.25 \cdot 10^{-5}$, batch size of 32, and three training epochs for “most tasks”. However, it does not clarify which tasks deviate from this setup or how. To ensure that the configuration can be extended consistently to BiSSL, we instead perform a grid search over learning rates and weight decays. This involves a total of 32 different combinations of logarithmically uniformly spaced learning rates

$[10^{-6}, 1.93 \cdot 10^{-6}, 3.72 \cdot 10^{-6}, 7.2 \cdot 10^{-6}, 1.39 \cdot 10^{-5}, 2.68 \cdot 10^{-5}, 5.18 \cdot 10^{-5}, 10^{-4}]$ and weight decays $[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}]$. The learning rate follows a cosine decaying schedule with a warm-up period from 0 covering the first 0.2% of total training steps.

We use the downstream objective described by Equation (2.3) and Section 2.3.5 and set the auxiliary language modeling objective scaling to $\nu = 0.5$. As outlined previously, sentence similarity tasks involve processing of two input sequences $\mathbf{x}_1$ and $\mathbf{x}_2$. These are processed independently through the transformer blocks, resulting in outputs $H_L(\mathbf{x}_1)$ and $H_L(\boldsymbol{x}_2)$. These are then summed element-wise and passed through the task-specific output layer, while the language modeling heads produce separate outputs for each sequence, which are averaged before computing the auxiliary loss. We provide a modified version in the BiSSL implementation of the `OpenAIGPTLMHeadModel`-class from the `transformers` library to account for this altered processing.

We evaluate validation performance ten times per epoch at evenly spaced intervals. The final evaluation always coincides with the last update step. As for the experiments in Chapter 6, we save the model whenever validation performance exceeds all previous evaluations. For each downstream task, we train ten models using the considered best configuration from the grid search, using different random seeds. We report the mean and standard deviation across these runs.

7.1.3 BiSSL Setup

We adopt the original BiSSL framework described in Chapter 4, following the training pipeline shown in Figure 4.1.

Downstream Head Parameters In light of the considerations from Section 5.4.1 of Chapter 5, we define the downstream head to include both the task-specific output layer as well as the downstream-specific rows in the embedding matrix, without the need to make any modifications to the theoretical setup. These rows correspond to the special start, end, and delimiter tokens. However, since BiSSL requires the option to insert an external backbone model in place of the downstream backbone (the first term in (4.1)), we have modified the implementation to allow such substitution of specific rows of the embedding matrix, enabling the replacement of solely the rows corresponding to those considered part of the backbone model.

Self-Supervised Pretraining We reuse the pretrained model described in Section 7.1.2.

Downstream Head Warm-up We conduct the downstream head warm-up for varying durations, conditioned on the downstream dataset size. For the SST2, QNLI, and QQP datasets, it is conducted for 10% of an epoch, and otherwise 50% of an epoch for the remaining datasets. The learning rate and weight decay identified as optimal in the baseline fine-tuning setup are reused as a starting point, with minor occasional adjustments. Except for the backbone being frozen and that the learning rate stays constant, the same setup used for downstream fine-tuning is otherwise applied here.

BiSSL The BiSSL optimization procedure largely follows the setup in Section 6.1.3, where mainly specific hyperparameters are adjusted for these experiments. For general

Table 7.1: Comparison of classification accuracies between the conventional SSL pipeline and the BiSSL pipeline. Accuracies significantly higher than their counterparts are marked in bold.

	CoLA	SST2	RTE	QNLI	MRPC	QQP	STSB
Only FT	47.0 ± 1.8	92.2 ± 0.3	60.3 ± 4.1	85.7 ± 0.3	85.2 ± 1.0	87.0 ± 0.2	86.3 ± 0.6
BiSSL + FT	47.0 ± 1.6	$\mathbf{93.0 \pm 0.3}$	64.4 ± 1.1	85.9 ± 0.4	85.2 ± 0.3	87.1 ± 0.1	$\mathbf{87.5 \pm 0.3}$
Avg Diff	0.0	**+0.8**	+4.1	+0.2	0.0	+0.1	**+1.2**

details regarding gradient approximation and batch handling, we refer to that section. We reuse the exact optimizer configurations specified for baseline SSL pretraining and downstream fine-tuning in Section 7.1.2, except that the lower-level peak learning rate is reduced to $5 \cdot 10^{-6}$.

Based on preliminary testing from the validation accuracies on the RTE dataset, we use $T = 200$ training stage alternations with $N_L = 20$ and $N_U = 4$ lower- and upper-level iterations respectively (see Algorithm 2). This setup appeared to benefit significantly from stronger lower-level regularization, so we increased $\lambda = 0.1$ throughout, in contrast to the choice of 0.001 used in our original implementation [49]. With a lower-level batch size of 64, the total number of available batches is 32227 (while we cannot confirm that this is the exact number of available batches used to train the original model, we reasonably assume it is within the proximity). This means that the lower-level in this configuration trains for what corresponds to roughly $\frac{TN_L}{32227} = \frac{200 \cdot 20}{32227} \approx 0.12$ pretraining epoch, which is a negligible overhead.

Downstream Fine-Tuning The downstream fine-tuning procedure follows the exact same procedure as the baseline downstream fine-tuning, except that the backbone is instead initialized with the lower-level backbone achieved from the previous BiSSL stage.

7.2 Downstream Task Performance

Table 7.1 reports the means and standard deviations of the downstream task accuracies, comparing the standard self-supervised learning pipeline (Only FT) with the BiSSL-augmented pipeline (BiSSL+FT). The results indicate that BiSSL occasionally yields significant performance improvements, while otherwise matching the baseline accuracy. No clear trend emerges regarding task type, as the gains appear across tasks, including SST2 (classification), RTE (entailment), and STSB (sentence similarity). While the average improvements of +4.4% on RTE do not exceed the baseline's standard deviation range, the drastic average increase along the reduction in variance (from 4.1 to 1.1) still suggests an improvement from BiSSL. CoLA and MRPC show no significant gains, though reduced accuracy variance is still observed, similar to the results from Chapter 6 and [49]. QNLI and QQP exhibit slight improvements, though likely not significant. Crucially, BiSSL never underperforms relative to the baseline, supporting the conclusion that its benefits extend to NLP tasks without risk of performance degradation.

8 | Discussion

This chapter reflects on key aspects of the revisited areas of BiSSL along with the outcome of the corresponding experiments in Chapter 6 and 7.

Randomness of Experiments Most of the experiments in Chapter 6 used $H = 25$ randomly sampled learning rates and weight decays for the fine-tuning hyperparameter grid search. While this in most cases provided adequate hyperparameters, it does far from guarantee that any of the 25 sampled configurations are close to optimal.

The BiSSL pipeline adds further stochasticity through its sequence of interdependent training stages. A poorly initialized seed during BiSSL can propagate and degrade final performance, and due to computational constraints, we run only a single BiSSL training per dataset and hyperparameter setup. As we currently have no established strategy that reliably predicts downstream performance during BiSSL, the only alternative would be to conduct the costly process of repeating the full BiSSL process with multiple seeds and perform fine-tuning for each. This is impractical and likely infeasible in most scenarios.

The main takeaway is that results should be understood in terms of general trends. Outliers and unexpected drops are likely artifacts of randomness and should not be overemphasized. Examples where this may be the case are seen in Figure 6.2 for $T = 800$ and the lower-right plot of the baseline in Figure 6.5 at 300 fine-tuning epochs.

8.1 Hyperparameter Sensitivity

BiSSL generally appeared robust to most hyperparameter changes. The most notable exceptions were for the number of upper-level iterations N_U and the number of training stage alternations T, as shown in Figures 6.3 and 6.2. When either was set too low, BiSSL was executed for an insufficient duration, which unsurprisingly led to less enhanced downstream performance. However, performance gains plateaued beyond a certain threshold, suggesting that BiSSL can find optimal model initializations in a feasible time span.

Surprisingly, varying other hyperparameters such as the number of lower-level iterations N_L and the regularization weight λ had limited effect on downstream accuracy. One possible reason for the insensitivity to λ is the fixed dampening factor $\lambda_{\text{damp}} = 10$ used in the upper-level gradient approximation (see Algorithm 3). This dampening factor reduces the influence of the specific value of λ on the final update, at least for the dataset used here. Still, results from Section 6.4 suggest that poor choices of λ can lead to significant performance drops, implying that regularization still does play an important role in the composite problem. Benchmarking BiSSL, where both λ and λ_{damp} are varied concurrently, would provide a more complete picture of how the lower-level regularization impacts the

composite problem.

The variation in fine-tuning duration illustrated in Figure 6.5, underpins the robustness of models trained via BiSSL. As seen, the BiSSL-pretrained models performed consistently across different numbers of fine-tuning epochs, whereas standard baselines exhibited accuracy fluctuations and in some cases showed signs of overfitting. This pattern suggests that BiSSL provides a more reliable initialization, making downstream performance less sensitive to the specific fine-tuning setup.

That said, any general conclusion from these experiments remains potentially tentative, as most experiments were conducted on a single dataset. Broader experimentation is needed to fully assess if the insensitivity of BiSSL is retained across tasks. Nonetheless, the findings presented here provide a strong initial indication that BiSSL may be more robust than previously assumed.

8.2 Impact of Adaptable λ

As discussed earlier, varying λ had a limited impact on downstream performance. The results in Section 6.4, where λ was allowed to vary during training, were therefore somewhat surprising, as most strategies adversarially affected downstream performance.

8.2.1 Scheduling

One possible explanation is that scheduling allowed λ to get too small, potentially violating the convexity assumption of the lower-level objective. This is important because the derivation of the upper-level gradient (via the comments following Proposition 4.2) assumes that the regularization objective r convexifies the lower-level problem. When λ drops near the $\lambda_{\min} = 10^{-8}$ used in the experiments, this assumption may break, leading to unstable gradients and poor updates during BiSSL. While the scheduling results were weak, it may still be worth rerunning experiments with a larger $\lambda_{\min}$ to keep the optimization stable.

It's also relevant to mention that the scheduling experiments used a small-scale setup with reduced model capacity. Such models may be less able to balance the competing pretext and downstream losses in BiSSL, making them more sensitive to the value of λ. This may explain why BiSSL outperformed or matched the "only FT" baseline in the small-scale setup, but underperformed in the full-scale configuration (Table 6.3). Similarly, $\lambda = 0.1$ was used for the NLP experiments in Chapter 7, as preliminary testing found smaller values less effective. Together, these results suggest that the optimal choice of λ is tightly linked to architectural characteristics.

8.2.2 Learnable Variant

Section 6.4.2 explored BiSSL with a learnable λ, and found no gains in downstream performance. The upper-level gradient with respect to λ (from Proposition 5.2) may help to explain why.

The gradient involves the dot product between the lower-level backbone shift induced by the regularization term $\boldsymbol{\theta}_D - \boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D, \lambda)$ and the first term of the upper-level gradient $\frac{\mathrm{d}\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda)}{\mathrm{d}\boldsymbol{\theta}_D} \nabla_{\boldsymbol{\theta}} \mathcal{L}^D(\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D,\lambda), \boldsymbol{\phi}_D)$. Hence, when the shift aligns with the upper-level gradient, the upper-level optimization encourages a larger value of λ. Conversely, when they are

anti-aligned or near orthogonal, λ either decreases or stays near constant. In other words, the optimizer increases λ when regularization helps the upper-level downstream task during BiSSL, and conversely decreases λ when regularization disrupts downstream performance.

This may explain why the learned λ values tend to converge toward the relatively large value of 1 in practice (Figure F.1 and F.2). As seen in the scheduling experiments results of Figure 5.1, such high values of λ lead to models with downstream performance close to those obtained from conventional fine-tuning. The influence of the pretext task is diminished, and the benefits of BiSSL vanish.

Adding a more forward-looking objective to the upper-level that depends on λ and reflects more directly how it impacts subsequent fine-tuning performance could help, in theory. But designing such a term is nontrivial and would potentially further increase computational complexity. More fundamentally, the results across experiments suggest that downstream performance is not sensitive to λ nor beneficial by making it adaptive. Hence, while λ influences the balance between pretext and downstream loss, it does not appear to be a key driver of the overall benefits imposed by BiSSL. Future work may be better directed at other components.

8.3 Inclusion of Pretext Head in IJ Approximation

Including the pretext head in the IJ approximation did not consistently improve downstream performance, as evidenced by the results in Section 6.5. One possibility for why that is would be that including additional parameters from the pretext head increases the size and complexity of the approximated IJ matrix, making it harder to estimate accurately. Since we rely on the conjugate gradient (CG) method for approximation, larger matrices may amplify numerical instability or noise. Performance may benefit from this more general formulation if a more robust or efficient solver were available, but in its current form, the added complexity may do more harm than good.

Even if the pretext head could be effectively incorporated, doing so still introduces non-trivial computational overhead. While the projection head in our setup (based on SimCLR) is relatively small, many modern contrastive-based pretext tasks use large projection heads, sometimes even larger than the backbone itself [23, 88]. In such settings, including the pretext head would add substantial cost to each backward pass through the IJ approximation.

An alternative explanation is that the pretext head simply does not contribute meaningful information to the BiSSL optimization in its current form. If its parameters have little influence on the downstream loss via the shared backbone, then including them in the upper-level gradient brings little benefit. In that case, the fixed pretext head assumption used in the original BiSSL framework may already strike a very good balance between both theoretical rigor and practical efficiency.

Overall, while it may be theoretically appealing to incorporate the pretext head for a more complete gradient, the practical benefits appear limited given the added computational burden. Unless future methods find a way to exploit the pretext head more efficiently, the fixed-head approximation of the original BiSSL setup remains a now further justified and sensible design choice.

8.4 Adaptation to NLP Tasks

As shown in Table 7.1, BiSSL never underperforms relative to the baseline and occasionally yields significant performance gains. However, these improvements are generally less pronounced than those reported in image-based settings (see [49] and Table 6.1). While this may partly be attributed to general domain-specific factors, we will here address potential aspects of the BiSSL framework that may impact its effectiveness in the NLP setting.

In the image domain, pixel-level variation can lead to substantial divergence between pretext and downstream inputs. In contrast, text inputs are discrete and drawn from a fixed-size vocabulary. Although token embeddings introduce some continuity, each token still maps to a fixed embedding vector. As a result, the inputs seen during pretraining and fine-tuning are nearly identical, with variation primarily being in the sequence ordering. This may imply that the distribution shift BiSSL is designed to address is less prominent in NLP, weakening its utility.

The embedding matrix defines the continuous input structure to the attention mechanism and is further reused in the output layer of the language modeling head. Applying the lower-level regularization r to this matrix may therefore impose overly rigid constraints, potentially restricting the lower-level's ability to effectively optimize the full model. Relaxing this constraint on the embedding matrix specifically could allow the lower-level more flexibility. Since the attention layers remain regularized by the upper-level parameters, the lower-level would still need to find embeddings that are implicitly compatible with both levels concurrently.

Another important consideration is the inclusion of the language modeling loss in the downstream objective. As it aligns the two tasks more closely, it may actually interfere with BiSSL, reducing its ability to prioritize the actual downstream objective. This could potentially lead to poor task adaptation. The observation that a larger value of λ was optimal in the NLP setting ($\lambda = 0.1$ for NLP tasks versus $\lambda = 0.001$ for vision tasks) may support this interpretation, as it suggests the upper-level needed stronger influence on the bilevel problem. Future experiments that exclude the LM loss from the downstream objective may provide a cleaner testbed for BiSSL.

Finally, a trade-off to consider is computational cost. GPT fine-tunes relatively quickly, so the overhead introduced by BiSSL becomes proportionally larger compared to standard fine-tuning. In some applications, this may diminish the relative appeal of using BiSSL in NLP unless it can deliver substantial gains.

9 | Conclusion and Future Work

This project advanced the study of the BiSSL framework through extended empirical evaluation across hyperparameter variations, design modifications, and adaptation to new data domains.

Across variations in most core hyperparameters of BiSSL, downstream performance remained stable over a wide range of values, provided that a sufficient number of total upper-level gradient steps were conducted. Attempts to improve BiSSL by scheduling the regularization parameter λ were largely unsuccessful. Performance degraded in many cases, suggesting that such scheduling introduces instability without providing any clear benefits.

An extension of BiSSL was proposed in which λ is treated as a learnable parameter, and an efficient gradient expression was derived for this case. This variant also failed to improve downstream performance, reinforcing the implication that λ is best kept fixed during training. Another modification of BiSSL relaxed the assumption that the pretext head parameters were fixed during approximation of the upper-level gradient, which led to the derivation of a more general expression for the implicit Jacobian. Despite this more complete formulation, downstream performance remained unaltered. This may indicate that the pretext head adds little useful information to the upper-level optimization, or that its contribution is not effectively captured by the current approximation method.

Lastly, BiSSL was applied to a range of downstream natural language processing tasks using the pretext task introduced by the generative pretrained transformer (GPT) framework. In this setting, BiSSL occasionally achieved significant accuracy gains, while it otherwise matched the baseline performance. Although aspects of the current BiSSL design may limit its effectiveness in NLP, the results nonetheless demonstrate that it can be applied without degrading downstream performance.

Overall, the findings suggest that the original design of BiSSL is robust and effective. Its benefits are retained across a wide range of hyperparameter choices, and it appears adaptable across architectures and input domains.

9.1 Future Work

While this project extended the empirical and theoretical understanding of BiSSL, several open directions remain for future exploration:

- Along the original BiSSL paper [49], this project supports the claims of improved downstream alignment primarily through classification accuracies and selected feature visualizations. Utility of other analytical tools could yield deeper insights into this claim, such as estimation of the effective dimension [89] or using information plane

analysis [90] on the backbone latent space.

- BiSSL uses the conjugate gradient method to approximate the upper-level gradient. Future work could evaluate alternatives such as Neumann series [91] or M-FAC [92] approximations, which may provide better estimates and/or reduce computational cost.
- Given that including the pretext head parameters in the implicit Jacobian calculation did not improve performance, future work could venture in the opposite direction by instead imposing further simplifying assumptions. For example, prior work [93, 94] shows that fine-tuning only the batch normalization (BN) layers in CNNs can nearly match training all model parameters. Similarly, for ResNet-based backbones, assuming all parameters except BN layers are fixed during Jacobian approximation could significantly reduce computational cost while possibly sustaining the downstream performance gains.
- Continuing in the focus on the pretext head, explicitly incorporating the pretext head parameters into the upper-level may enable the upper-level to actually advantageously leverage the pretext head parameters. This could be approached in numerous ways, where one such approach is proposed in Appendix G for inspiration.
- A simple fine-tuning extension worth exploring is to train for a fixed number of epochs and then apply early stopping as soon as the validation accuracy or loss begins to degrade. As BiSSL-preconditioned models tend to converge quickly, this strategy may reduce computation while preserving downstream performance gains.
- A challenge lies in the fact that BiSSL currently lacks a way to monitor how well the lower-level backbone will transfer to the downstream task during BiSSL. Developing unsupervised metrics tailored to this setting could help speed up the preliminary testing phases by being able to rule out configurations that cause degenerate backbone initializations early on. For instance, [95] proposed a mutual information-based criterion to evaluate the quality of latent representation during self-supervised training, which may be adaptable for this purpose.

Bibliography

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li *et al.*, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[3] X. Zhai, A. Oliver, A. Kolesnikov, and L. Beyer, "S4l: Self-supervised semi-supervised learning," *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1476–1485, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:167209887

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

[5] Z. He, "Deep learning in image classification: A survey report," in *2020 2nd International Conference on Information Technology and Computer Application (ITCA)*, 2020, pp. 174–177.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[7] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz *et al.*, "Image segmentation using deep learning: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 7, pp. 3523–3542, 2021.

[8] J. Bharadiya, "A comprehensive survey of deep learning techniques natural language processing," *European Journal of Technology*, vol. 7, pp. 58–66, 05 2023.

[9] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 2, pp. 604–624, 2021.

[10] H. Purwins, B. Li, T. Virtanen, J. Schlüter, S.-Y. Chang *et al.*, "Deep learning for audio signal processing," *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 2, pp. 206–219, 2019.

[11] D. Wang and J. Chen, "Supervised speech separation based on deep learning: An overview," *IEEE/ACM transactions on audio, speech, and language processing*, vol. 26, no. 10, pp. 1702–1726, 2018.

[12] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang *et al.*, "Self-supervised learning: Generative or contrastive," *IEEE transactions on knowledge and data engineering*, vol. 35, no. 1, pp. 857–876, 2021.

[13] K. Wiggers, "Yann lecun and yoshua bengio: Self-supervised learning is the key to human-level intelligence," *VentureBeat*. [Online]. Available: https://venturebeat.com/ai/yann-lecun-and-yoshua-bengio-self-supervised-learning-is-the-key-to-human-level-intelligence/

[14] E. Orhan, V. Gupta, and B. M. Lake, "Self-supervised learning through the eyes of a child," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 9960–9971. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/7183145a2a3e0ce2b68cd3735186b1d5-Paper.pdf

[15] J. Gui, T. Chen, J. Zhang, Q. Cao, Z. Sun *et al.*, "A survey on self-supervised learning: Algorithms, applications, and future trends," *IEEE transactions on pattern analysis and machine intelligence*, vol. PP, 06 2024.

[16] J. Anton, L. Castelli, M. F. Chan, M. Outters, W. H. Tang *et al.*, "How well do self-supervised models transfer to medical imaging?" *Journal of Imaging*, vol. 8, no. 12, 2022. [Online]. Available: https://www.mdpi.com/2313-433X/8/12/320

[17] Z.-H. Tan, "Self-supervised learning for multimodel data: From models to loss functions," University Lecture, 2023.

[18] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[19] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607.

[20] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 9729–9738.

[21] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond *et al.*, "Bootstrap your own latent-a new approach to self-supervised learning," *Advances in neural information processing systems*, vol. 33, pp. 21 271–21 284, 2020.

[22] X. Chen and K. He, "Exploring simple siamese representation learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 15 750–15 758.

[23] A. Bardes, J. Ponce, and Y. LeCun, "VICReg: Variance-invariance-covariance regularization for self-supervised learning," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=xm6YD62D1Ub

[24] Y.-A. Chung, W.-N. Hsu, H. Tang, and J. Glass, "An Unsupervised Autoregressive Model for Speech Representation Learning," in *Proc. Interspeech 2019*, 2019, pp. 146–150.

[25] R. C. Staudemeyer and E. R. Morris, "Understanding lstm–a tutorial into long short-term memory recurrent neural networks," *arXiv preprint arXiv:1909.09586*, 2019.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[27] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *North American Chapter of the Association for Computational Linguistics*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:52967399

[28] C. Doersch, "Tutorial on variational autoencoders," *arXiv preprint arXiv:1606.05908*, 2016.

[29] K. He, X. Chen, S. Xie, Y. Li, P. Dollár *et al.*, "Masked autoencoders are scalable vision learners," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 16 000–16 009.

[30] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[31] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei *et al.*, "Language models are unsupervised multitask learners," 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:160025533

[32] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.

[33] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad *et al.*, "Gpt-4 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2303.08774

[34] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. rahman Mohamed *et al.*, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Annual Meeting of the Association for Computational Linguistics*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:204960716

[35] M. Caron, H. Touvron, I. Misra, H. Jegou, J. Mairal *et al.*, "Emerging properties in self-supervised vision transformers," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 9630–9640.

[36] A. Baevski, W.-N. Hsu, Q. Xu, A. Babu, J. Gu *et al.*, "Data2vec: A general framework for self-supervised learning in speech, vision and language," in *International Conference on Machine Learning*. PMLR, 2022, pp. 1298–1312.

[37] Y. Dubois, T. Hashimoto, S. Ermon, and P. Liang, "Improving self-supervised learning by characterizing idealized representations," *ArXiv*, vol. abs/2209.06235, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:252222375

[38] S. Zaiem, T. Parcollet, and S. Essid, "Less forgetting for better generalization: Exploring continual-learning fine-tuning methods for speech self-supervised representations," *arXiv preprint arXiv:2407.00756*, 2024.

[39] L. Wang, X. Zhang, H. Su, and J. Zhu, "A comprehensive survey of continual learning: Theory, method and application," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[40] M. Boschini, L. Bonicelli, A. Porrello, G. Bellitto, M. Pennisi *et al.*, "Transfer without forgetting," in *European Conference on Computer Vision*. Springer, 2022, pp. 692–709.

[41] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1127647

[42] Y. Zhang, P. Khanduri, I. C. Tsaknakis, Y. Yao, M.-F. Hong *et al.*, "An introduction to bi-level optimization: Foundations and applications in signal processing and machine learning," *ArXiv*, vol. abs/2308.00788, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:260378880

[43] Y. Zhang, Y. Yao, P. Ram, P. Zhao, T. Chen *et al.*, "Advancing model pruning via bi-level optimization," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho *et al.*, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 18 309–18 326.

[44] M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-Paz, "Invariant risk minimization," *ArXiv*, vol. abs/1907.02893, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:195820364

[45] Y. Zhang, P. Sharma, P. Ram, M. Hong, K. R. Varshney *et al.*, "What is missing in IRM training and evaluation? challenges and solutions," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=MjsDeTcDEy

[46] A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine, "Meta-learning with implicit gradients," in *Neural Information Processing Systems*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:202542766

[47] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning*, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:6719686

[48] Y. Zhang, G. Zhang, P. Khanduri, M.-F. Hong, S. Chang *et al.*, "Revisiting and advancing fast adversarial training through the lens of bi-level optimization," in *International Conference on Machine Learning*, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:245424850

[49] G. W. Zakarias, L. K. Hansen, and Z.-H. Tan, "Bissl: Enhancing the alignment between self-supervised pretraining and downstream fine-tuning via bilevel optimization," *arXiv preprint arXiv:2410.02387*, 2025.

[50] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy *et al.*, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," 2019, in the Proceedings of ICLR.

[51] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, 1st ed. Springer, 2007.

[52] A. Kanezaki, Y. Matsushita, and Y. Nishida, "Rotationnet for joint object categorization and unsupervised pose estimation from multi-view images," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 43, no. 1, pp. 269–283, 2021.

[53] Y. Ouali, C. Hudelot, and M. Tami, "An overview of deep semi-supervised learning," *ArXiv*, vol. abs/2006.05278, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:219558952

[54] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar, "Cats and dogs," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[55] S. Haykin, *Neural networks: a comprehensive foundation.* Prentice Hall PTR, 1994.

[56] T. Wang and P. Isola, "Understanding contrastive representation learning through alignment and uniformity on the hypersphere," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 9929–9939.

[57] T. Gao, X. Yao, and D. Chen, "SimCSE: Simple contrastive learning of sentence embeddings," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, Eds. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 6894–6910. [Online]. Available: https://aclanthology.org/2021.emnlp-main.552/

[58] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang *et al.*, "A survey on evaluation of large language models," *ACM Trans. Intell. Syst. Technol.*, vol. 15, no. 3, 2024.

[59] M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing," 2017, to appear.

[60] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi *et al.*, "Generating wikipedia by summarizing long sequences," in *ICLR*, 2018.

[61] A. Dontchev and R. Rockafellar, *Implicit Functions and Solution Mappings: A View from Variational Analysis*, ser. Springer Series in Operations Research and Financial Engineering. Springer New York, 2014. [Online]. Available: https://books.google.dk/books?id=LnAgBAAAQBAJ

[62] N. Zucchet and J. Sacramento, "Beyond backpropagation: bilevel optimization through implicit differentiation and equilibrium propagation," *Neural Computation*, vol. 34, no. 12, pp. 2309–2346, 2022.

[63] J. L. Nazareth, "Conjugate gradient method," *WIREs Computational Statistics*, vol. 1, no. 3, pp. 348–353, 2009. [Online]. Available: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wics.13

[64] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," USA, Tech. Rep., 1994.

[65] H. Zheng, Z. Yang, W. Liu, J. Liang, and Y. Li, "Improving deep neural networks using softplus units," in *2015 International Joint Conference on Neural Networks (IJCNN)*, 2015, pp. 1–4.

[66] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, 11–13 Apr 2011, pp. 215–223. [Online]. Available: https://proceedings.mlr.press/v15/coates11a.html

[67] M. Everingham, L. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge." *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, 2010.

[68] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, *et al.*, "Describing textures in the wild," in *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[69] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.

[70] Y. You, I. Gitman, and B. Ginsburg, "Large batch training of convolutional networks," *arXiv: Computer Vision and Pattern Recognition*, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:46294020

[71] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=Skq89Scxx

[72] S. Ruder, "An overview of gradient descent optimization algorithms," *ArXiv*, vol. abs/1609.04747, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:17485266

[73] Q. Lhoest, A. Villanova del Moral, Y. Jernite, A. Thakur, P. von Platen *et al.*, "Datasets: A community library for natural language processing," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, H. Adel and S. Shi, Eds. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 175–184.

[74] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Q. Liu and D. Schlangen, Eds. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45.

[75] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun *et al.*, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 19–27.

[76] A. Warstadt, A. Singh, and S. R. Bowman, "Neural network acceptability judgments," *arXiv preprint 1805.12471*, 2018.

[77] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning *et al.*, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of EMNLP*, 2013, pp. 1631–1642.

[78] L. Bentivogli, I. Dagan, H. T. Dang, D. Giampiccolo, and B. Magnini, "The fifth PASCAL recognizing textual entailment challenge," 2009.

[79] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "SQuAD: 100,000+ questions for machine comprehension of text," in *Proceedings of EMNLP*. Association for Computational Linguistics, 2016, pp. 2383–2392.

[80] W. B. Dolan and C. Brockett, "Automatically constructing a corpus of sentential paraphrases," in *Proceedings of the International Workshop on Paraphrasing*, 2005.

[81] N. D. Shankar Iyer and K. Csernai, "Quora question pairs," 2018. [Online]. Available: https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs

[82] D. Cer, M. Diab, E. Agirre, I. Lopez-Gazpio, and L. Specia, "SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, S. Bethard, M. Carpuat, M. Apidianaki, S. M. Mohammad, D. Cer *et al.*, Eds. Vancouver, Canada: Association for Computational Linguistics, Aug. 2017, pp. 1–14.

[83] L. Von Werra, L. Tunstall, A. Thakur, S. Luccioni, T. Thrush *et al.*, "Evaluate & evaluation on the hub: Better best practices for data and model measurements," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, W. Che and E. Shutova, Eds. Abu Dhabi, UAE: Association for Computational Linguistics, Dec. 2022, pp. 128–136.

[84] A. Williams, N. Nangia, and S. R. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," in *Proceedings of NAACL-HLT*, 2018.

[85] H. J. Levesque, E. Davis, and L. Morgenstern, "The Winograd schema challenge," in *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, vol. 46, 2011, p. 47.

[86] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:6628106

[87] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[88] T. Chen, S. Kornblith, K. Swersky, M. Norouzi, and G. E. Hinton, "Big self-supervised models are strong semi-supervised learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 22 243–22 255.

[89] J. Gawlikowski, C. R. N. Tassi, M. Ali, J. Lee, M. Humt *et al.*, "A survey of uncertainty in deep neural networks," vol. 56, no. Suppl 1, p. 1513–1589, Jul. 2023. [Online]. Available: https://doi.org/10.1007/s10462-023-10562-9

[90] B. C. Geiger, "On information plane analyses of neural network classifiers—a review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 7039–7051, 2022.

[91] J. Lorraine, P. Vicol, and D. Duvenaud, "Optimizing millions of hyperparameters by implicit differentiation," in *International conference on artificial intelligence and statistics*. PMLR, 2020, pp. 1540–1552.

[92] E. Frantar, E. Kurtic, and D. Alistarh, "M-fac: Efficient matrix-free approximations of second-order information," 2021.

[93] F. Kanavati and M. Tsuneki, "Partial transfusion: on the expressive influence of trainable batch norm parameters for transfer learning," in *Proceedings of the Fourth Conference on Medical Imaging with Deep Learning*, ser. Proceedings of Machine Learning Research, M. Heinrich, Q. Dou, M. de Bruijne, J. Lellmann, A. Schläfer *et al.*, Eds., vol. 143. PMLR, 07–09 Jul 2021, pp. 338–353. [Online]. Available: https://proceedings.mlr.press/v143/kanavati21a.html

[94] S. Lange, K. Helfrich, and Q. Ye, "Batch normalization preconditioning for neural network training," *J. Mach. Learn. Res.*, vol. 23, no. 1, Jan. 2022.

[95] A. H. Liu, S.-L. Yeh, and J. R. Glass, "Revisiting self-supervised learning of speech representation from a mutual information perspective," in *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2024, pp. 12 051–12 055.

[96] X. Chen, H. Fan, R. B. Girshick, and K. He, "Improved baselines with momentum contrastive learning," *CoRR*, vol. abs/2003.04297, 2020. [Online]. Available: https://arxiv.org/abs/2003.04297

[97] J. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *ArXiv*, vol. abs/1607.06450, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:8236317

[98] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv: Learning*, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:125617073

[99] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

Appendices

A | SimCLR: Related Pretext Tasks

Several alternative contrastive learning methods related to SimCLR [19] (see Section 2.2 in Chapter 2) have been proposed, each pursuing the general goal of learning meaningful representations by aligning different views of data. While they share the core principles with SimCLR, they differ in architecture, training objectives, and sampling strategies. Below, we briefly outline a selection of influential approaches. Figure A.1 provides an overview of the methods discussed.

Momentum Contrast Momentum Contrast (MoCo) [20] addresses the large batch size requirement in SimCLR by maintaining a dynamic dictionary of negative samples. This dictionary is populated by a momentum encoder, a copy of the main encoder updated via exponential moving average. During training, query representations are produced by the online encoder, while keys (used as negatives) are generated from the momentum encoder. MoCo originally omitted a projection head, but later variants (e.g., MoCo v2 [96]) included one for improved performance.

Bootstrap Your Own Latent Bootstrap Your Own Latent (BYOL) [21] departs from contrastive learning by eliminating the use of negative samples altogether. It trains an online network and a target network on different augmentations of the same input. The online network learns to predict the target network's output, while the target network is updated as an exponential moving average of the online network parameters. This asymmetric structure promotes alignment without explicit negative samples, enabling representation learning through prediction alone.

Variance-Invariance-Covariance Regulation Variance-Invariance-Covariance Regularization (VICReg) [23] further distances itself from contrastive approaches by using only positive pairs and a non-contrastive loss. It introduces a three-term loss: an invariance term aligning representations of augmented views, a variance term enforcing feature diversity across samples, and a covariance term that decorrelates feature dimensions. Together, these terms prevent collapse and encourage the learning of non-trivial, stable representations without relying on negatives or momentum targets.

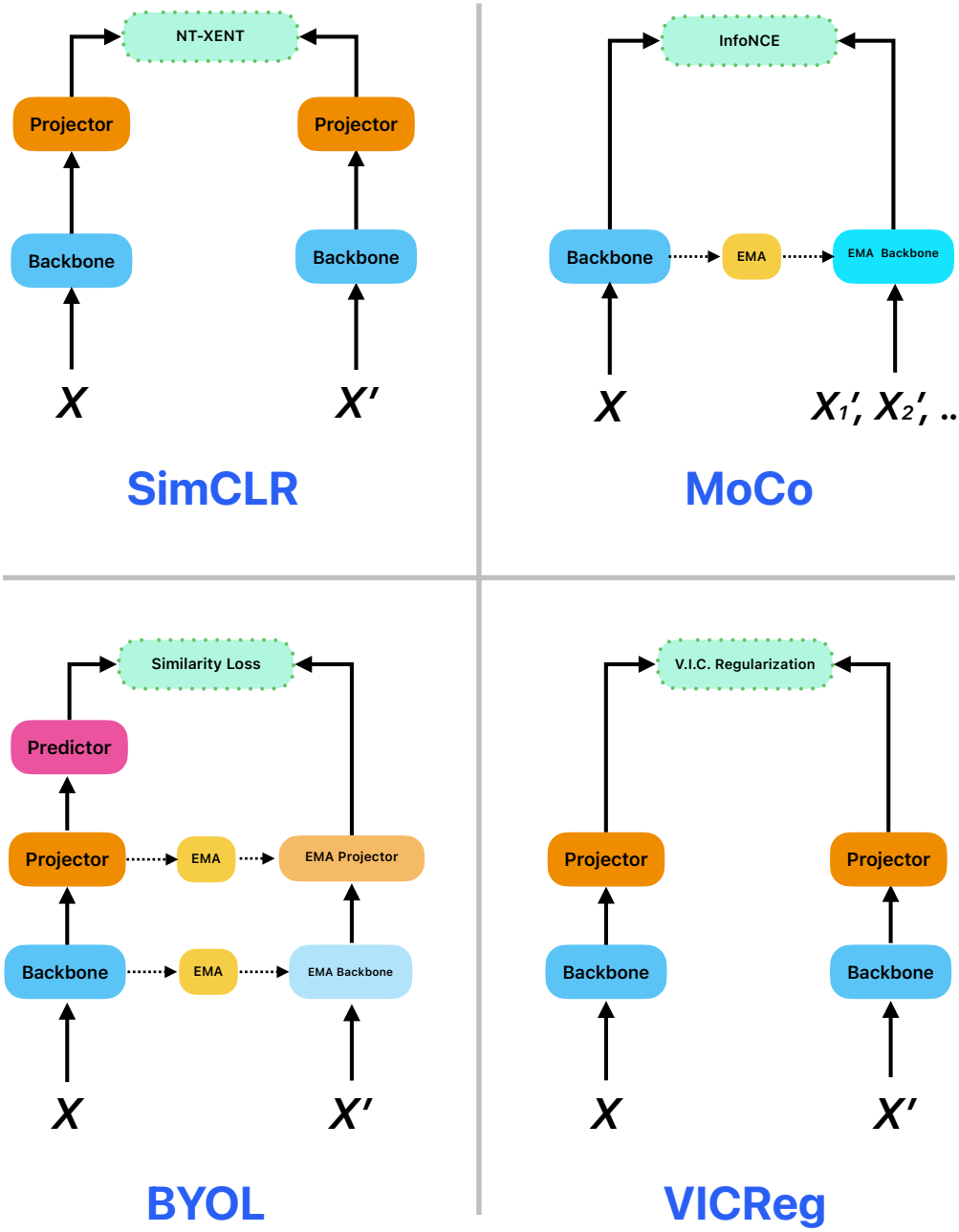

Figure A.1: Illustration of the overall information processing pipelines of the SimCLR, MoCo, BYOL, and VICReg pretext tasks.

B | The Transformer Blocks of GPT

This appendix, based on [26], details the inner workings of the architecture underlying the transformer blocks of the GPT model [30], described in Section 2.3. The heart of the GPT architecture is built from a stack of identical *transformer decoder blocks* [26], each of which computes a context-aware representation of the input tokens while enforcing a strict autoregressive constraint, i.e., each token may only attend to earlier or equal positions in the sequence. This is achieved using *masked multi-headed self-attention*.

Let $k \in \mathbb{N}$ be the context window size, $d \in \mathbb{N}$ the embedding dimension, and $h \in \mathbb{N}$ the number of *attention heads*. At each layer $l \in \{1, \dots, L\}$, and for each attention head $i \in \{1, \dots, h\}$, the model first computes *queries*, *keys*, and *values* via learned linear projections:

$$\begin{aligned} Q_{l,i} &= H_{l-1} W_{l,i}^{Q}, \\ K_{l,i} &= H_{l-1} W_{l,i}^{K}, \\ V_{l,i} &= H_{l-1} W_{l,i}^{V}, \end{aligned}$$

where $H_{l-1} \in \mathbb{R}^{k \times d}$ is the output of the previous layer, and $W_{l,i}^{Q}, W_{l,i}^{K}, W_{l,i}^{V} \in \mathbb{R}^{d \times d_h}$ are learned projection matrices with $d_h = d/h$ being the dimensionality of each head. The scaled dot-product attention is then computed for each head as:

$$\begin{aligned} \text{head}_i &= \text{Attention}(Q_{l,i}, K_{l,i}, V_{l,i}), \\ \text{Attention}(Q, K, V) &= \text{softmax}\left(\frac{QK^{\top}}{\sqrt{d_h}} + M\right) V, \end{aligned} \tag{B.1}$$

where $i \in \{1, \dots, h\}$ and the mask $M \in \mathbb{R}^{k \times k}$ contains $-\infty$ in positions where future tokens should be masked, i.e., its i,j'th entry equals $-\infty$ if $j > i$, and 0 otherwise. In this instance, the softmax operator is defined as:

$$\text{softmax}(A)_{i,j} = \frac{e^{A_{i,j}}}{\sum_{m=1}^{k} e^{A_{i,m}}}, \quad A \in \mathbb{R}^{k \times k}, \tag{B.2}$$

i.e., it applies the softmax operation row-wise. The scaled dot-product attention in (B.1) can be interpreted as follows: the query vector Q encodes what the current token is looking for; the key vector K represents what each other token offers; and the value V contains the actual content to aggregate. The softmax output determines how much each position contributes to the current token's updated representation.

The outputs from all h attention heads are then concatenated and linearly projected, which is what defines the multi-head attention mechanism:

$$\text{MHA}(H_{l-1}) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) W_l^{O},$$

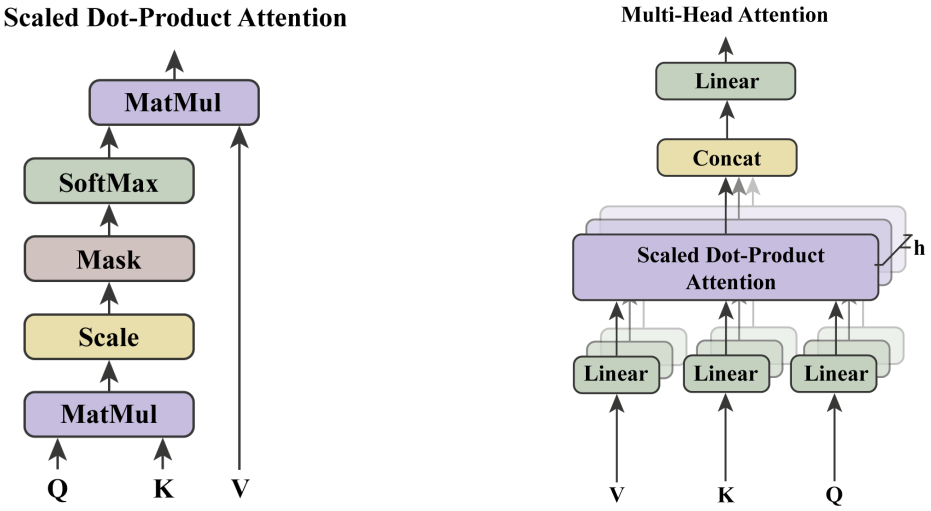

Figure B.1: Information processing pipelines of scaled dot-product attention (left) and multi-head attention (right).

Table B.1: Original hyperparameter values used in GPT [30].

Parameter	Value
V (Tokenizer Vocabulary Size)	40478
k (Context Window Size)	512
d (Token Embedding Dimension)	768
h (Number of Attention Heads)	12
d_h (Head Embedding Dimension)	$d/h = 64$
L (Number of Transformer Layers)	12
d_{ff} (FFN Embedding Dimension)	3072

where $W_l^O \in \mathbb{R}^{d\times d}$. Figure B.1 illustrates the flow of information through a single masked multi-headed attention block. The output of the attention mechanism is then added to the input via a residual connection, followed by layer normalization [97]

$$A_l = \text{LayerNorm}(H_{l-1} + \text{MHA}(H_{l-1})),$$

which is passed through a feedforward network, finally followed by another layer normalization, yielding the transformer block output:

$$\begin{aligned} H_l &= \text{LayerNorm}(A_l + \text{FFN}(A_l)), \\ \text{FFN}(Y) &= \text{GELU}(YW_1 + B_1)W_2 + B_2, \end{aligned}$$

where GELU is the Gaussian Error Linear Unit [98], $W_1 \in \mathbb{R}^{d\times d_{\text{ff}}}$, $W_2 \in \mathbb{R}^{d_{\text{ff}}\times d}$, $B_1 = [\mathbf{b}_1^\top, \ldots, \mathbf{b}_1^\top]^\top \in \mathbb{R}^{k\times d_{ff}}$, $B_2 = [\mathbf{b}_2^\top, \ldots, \mathbf{b}_2^\top]^\top \in \mathbb{R}^{k\times d}$ with $\mathbf{b}_1 \in \mathbb{R}^{d_{ff}}$, $\mathbf{b}_2 \in \mathbb{R}^d$.

This process repeats across all L transformer blocks, yielding increasingly abstract contextual representations. Table B.1 lists the originally used parameters, which will be adopted in Chapter 7. The resulting model has approximately 117 million parameters.

C | Theorems and Proofs

Theorem C.1 (Implicit Function Theorem [61, 62])
Let f be continuously differentiable and $(\bar{\mathbf{x}},\bar{\mathbf{y}})$ be given such that $f(\bar{\mathbf{x}},\bar{\mathbf{y}}) = \mathbf{0}$. Further assume that the Jacobian matrix $\partial_{\mathbf{x}} f(\bar{\mathbf{x}},\bar{\mathbf{y}})$ is invertible. Then there exists a unique and differentiable implicit function $\mathbf{y} \mapsto \mathbf{x}^*_{\mathbf{y}}$ defined in a neighborhood of $\bar{\mathbf{y}}$ such that $\mathbf{x}^*_{\bar{\mathbf{y}}} = \bar{\mathbf{x}}$, which verifies $f(\mathbf{x}^*_{\mathbf{y}},\mathbf{y}) = \mathbf{0}$ for all $\mathbf{y}$ in that neighborhood.

Proposition C.2 (Restated Proposition 5.5)
Let the bilevel optimization problem in Definition 5.4 be given. Assuming the lower-level (5.11) satisfies the conditions of Theorem 3.2, then the following IJ is given by

$$\frac{\mathrm{d}\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)}{\mathrm{d}\boldsymbol{\theta}_D} = M_{\boldsymbol{\theta}_D,\boldsymbol{\omega}}(\boldsymbol{\theta}_D) M_{\boldsymbol{\omega}}(\boldsymbol{\theta}_D), \quad \boldsymbol{\theta} \in \mathcal{G}_0$$

where $\mathcal{G}_0$ is the lower-level stationary set as given in Definition 3.1, and

$$M_{\boldsymbol{\theta}_D,\boldsymbol{\omega}}(\boldsymbol{\theta}_D) = \begin{bmatrix} \nabla^2_{\boldsymbol{\theta}_D\boldsymbol{\theta}_P} r(\boldsymbol{\theta}_D,\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) & \mathbf{O}_{L\times P_P} \end{bmatrix},$$

$$M_{\boldsymbol{\omega}}(\boldsymbol{\theta}_D) = \left[\frac{1}{\lambda} \frac{\mathrm{d}^2}{\mathrm{d}^2\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)} \mathcal{L}^P(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) + \begin{bmatrix} \nabla^2_{\boldsymbol{\theta}_P} r(\boldsymbol{\theta}_D,\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) & \mathbf{O}_{L\times P_P} \\ \mathbf{O}_{P_P\times L} & \mathbf{O}_{P_P\times P_P} \end{bmatrix} \right]^{-1}.$$

Proof (Proposition 5.5/ C.2): From Theorem 3.2, we get the Implicit Jacobian (IJ) of the lower-level solution in Definition 5.4 to be:

$$\frac{\mathrm{d}\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)}{\mathrm{d}\boldsymbol{\theta}_D} = -\nabla^2_{\boldsymbol{\theta}_D\boldsymbol{\omega}} G(\boldsymbol{\theta}_D,\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)) \left[\nabla^2_{\boldsymbol{\omega}} G(\boldsymbol{\theta}_D,\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))\right]^{-1} \in \mathbb{R}^{L\times L+P_P}, \tag{C.1}$$

for $\boldsymbol{\theta} \in \mathcal{G}_0$. To make the derivation of (C.1) tractable, we first establish a simple but crucial result regarding the effect of sub-parameter mappings on differentiation, which shows that working with the composite variable $\boldsymbol{\omega}$ does not require re-deriving the gradient expressions from scratch. It is formalized in Lemma C.3 and states that differentiation of a function $f(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}))$ with respect to $\boldsymbol{\omega}$ can be expressed in terms of the concatednated vector $\left[\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_P,\boldsymbol{\phi}_P)^\top, \nabla_{\boldsymbol{\phi}} f(\boldsymbol{\theta}_P,\boldsymbol{\phi})^\top\right]^\top$. Hence, the sub-parameter mappings let us *recover the original gradients with respect to their original arguments.*

We now individually compute the two Hessian matrices in the IJ of (C.1). First, consider the cross-Hessian

$$\nabla^2_{\boldsymbol{\theta}_D\boldsymbol{\omega}} G(\boldsymbol{\theta}_D,\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)) = \nabla_{\boldsymbol{\theta}_D} \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\omega}} \Big(\mathcal{L}^P(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) + \lambda r(\boldsymbol{\theta}_D,\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) \Big)$$

$$= \nabla_{\boldsymbol{\theta}_D} \begin{pmatrix} \nabla_{\boldsymbol{\theta}_P} \Big(\mathcal{L}^P(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) + \lambda r(\boldsymbol{\theta}_D,\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) \Big) \\ \nabla_{\boldsymbol{\phi}_P} \mathcal{L}^P(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) \end{pmatrix}$$

$$= -\lambda \Big[\nabla^2_{\boldsymbol{\theta}_D\boldsymbol{\theta}_P} r(\boldsymbol{\theta}_D,\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) \quad \mathbf{O}_{L\times P_P} \Big],$$

where the last and second last equalities are obtained through the use of Lemma C.3. Using this lemma again, the second Hessian is derived as follows:

$$\begin{aligned} \nabla^2_{\boldsymbol{\omega}} G(\boldsymbol{\theta}_D,\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)) &= \frac{\mathrm{d}^2}{\mathrm{d}^2\boldsymbol{\omega}} \Big(\mathcal{L}^P(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) + \lambda r(\boldsymbol{\theta}_D,\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) \Big) \\ &= \frac{\mathrm{d}^2}{\mathrm{d}^2\boldsymbol{\omega}} \mathcal{L}^P(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)) + \lambda \begin{bmatrix} \nabla^2_{\boldsymbol{\theta}_P} r(\boldsymbol{\theta}_D,\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) & \mathbf{O}_{L\times P_P} \\ \mathbf{O}_{P_P\times L} & \mathbf{O}_{P_P\times P_P} \end{bmatrix}. \end{aligned}$$

Substituting the above expression into (C.1) finally yields

$$\begin{aligned} &\frac{\mathrm{d}\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)}{\mathrm{d}\boldsymbol{\theta}_D} \\ &= -\nabla^2_{\boldsymbol{\theta}_D\boldsymbol{\omega}} G(\boldsymbol{\theta}_D,\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)) \big[\nabla^2_{\boldsymbol{\omega}} G(\boldsymbol{\theta}_D,\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)) \big]^{-1} \\ &= \lambda \big[\nabla^2_{\boldsymbol{\theta}_D\boldsymbol{\theta}_P} r(\boldsymbol{\theta}_D,\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) \quad \mathbf{O}_{L\times P_P} \big] \\ &\quad \cdot \left[\frac{\mathrm{d}^2}{\mathrm{d}^2\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)} \mathcal{L}^P(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) + \lambda \begin{bmatrix} \nabla^2_{\boldsymbol{\theta}_P} r(\boldsymbol{\theta}_D,\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) & \mathbf{O}_{L\times P_P} \\ \mathbf{O}_{P_P\times L} & \mathbf{O}_{P_P\times P_P} \end{bmatrix} \right]^{-1} \\ &= \big[\nabla^2_{\boldsymbol{\theta}_D\boldsymbol{\theta}_P} r(\boldsymbol{\theta}_D,\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) \quad \mathbf{O}_{L\times P_P} \big] \\ &\quad \cdot \left[\frac{1}{\lambda} \frac{\mathrm{d}^2}{\mathrm{d}^2\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)} \mathcal{L}^P(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) + \begin{bmatrix} \nabla^2_{\boldsymbol{\theta}_P} r(\boldsymbol{\theta}_D,\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) & \mathbf{O}_{L\times P_P} \\ \mathbf{O}_{P_P\times L} & \mathbf{O}_{P_P\times P_P} \end{bmatrix} \right]^{-1} \\ &= M_{\boldsymbol{\theta}_D,\boldsymbol{\omega}}(\boldsymbol{\theta}_D) M_{\boldsymbol{\omega}}(\boldsymbol{\theta}_D). \end{aligned}$$

■

Lemma C.3 (Derivatives under Sub-Parameter Mappings)
Let $f : \mathbb{R}^L \times \mathbb{R}^{P_P} \to \mathbb{R}$ be twice differentiable, and let $\boldsymbol{\omega} \in \mathbb{R}^{L+P_P}$ with corresponding sub-parameter mappings $\gamma_{\boldsymbol{\theta}}$ and $\gamma_{\boldsymbol{\phi}_P}$ be given as in Definition 5.3. Then

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\omega}} f(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega})) = \begin{pmatrix} \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta},\boldsymbol{\phi}_P)|_{\boldsymbol{\theta}=\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega})} \\ \nabla_{\boldsymbol{\phi}} f(\boldsymbol{\theta}_P,\boldsymbol{\phi})|_{\boldsymbol{\phi}=\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega})} \end{pmatrix},$$

and

$$\frac{\mathrm{d}^2}{\mathrm{d}^2\boldsymbol{\omega}} f(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega})) = \begin{bmatrix} \nabla^2_{\boldsymbol{\theta}} f(\boldsymbol{\theta},\boldsymbol{\phi}_P)|_{\boldsymbol{\theta}=\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega})} & \nabla^2_{\boldsymbol{\theta}\boldsymbol{\phi}} f(\boldsymbol{\theta},\boldsymbol{\phi})|_{\boldsymbol{\theta}=\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}),\boldsymbol{\phi}=\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega})} \\ \nabla^2_{\boldsymbol{\phi}\boldsymbol{\theta}} f(\boldsymbol{\theta},\boldsymbol{\phi})|_{\boldsymbol{\phi}=\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}),\boldsymbol{\theta}=\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega})} & \nabla^2_{\boldsymbol{\phi}} f(\boldsymbol{\theta}_P,\boldsymbol{\phi})|_{\boldsymbol{\phi}=\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega})} \end{bmatrix}.$$

Proof (Lemma C.3): Using the identity from (5.8), we derive

$$\frac{\mathrm{d}\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega})}{\mathrm{d}\boldsymbol{\omega}} = \frac{\mathrm{d}[I_L \quad \mathbf{O}_{L\times P_P}]\boldsymbol{\omega}}{\mathrm{d}\boldsymbol{\omega}} = \frac{\mathrm{d}\boldsymbol{\omega}}{\mathrm{d}\boldsymbol{\omega}} \begin{bmatrix} I_L \\ \boldsymbol{O}_{P_P\times L} \end{bmatrix} = \begin{bmatrix} I_L \\ \boldsymbol{O}_{P_P\times L} \end{bmatrix},$$

where a similar result is achieved for $\frac{\mathrm{d}\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega})}{\mathrm{d}\boldsymbol{\omega}}$ using (5.9). Then, we express

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\omega}} f(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega})) &= \frac{\mathrm{d}\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega})}{\mathrm{d}\boldsymbol{\omega}} \nabla_{\boldsymbol{\theta}} f(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega})) + \frac{\mathrm{d}\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega})}{\mathrm{d}\boldsymbol{\omega}} \nabla_{\boldsymbol{\phi}} f(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega})) \\ &= \begin{bmatrix} I_L \\ \boldsymbol{O}_{P_P\times L} \end{bmatrix} \nabla_{\boldsymbol{\theta}} f(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega})) + \begin{bmatrix} \boldsymbol{O}_{L\times P_P} \\ I_{P_P} \end{bmatrix} \nabla_{\boldsymbol{\phi}} f(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega})) \\ &= \begin{pmatrix} \nabla_{\boldsymbol{\theta}} f(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega})) \\ \nabla_{\boldsymbol{\phi}} f(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega})) \end{pmatrix}. \end{aligned}$$

The result for $\frac{\mathrm{d}^2}{\mathrm{d}^2\boldsymbol{\omega}} f(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}),\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}))$ is derived by continuing in a similar manner on the expression above, and is therefore omitted. ■

D | Bilevel Training Algorithms - Application Examples

Incorporating a training framework into a bilevel optimization problem necessitates careful consideration of the problem formulation to derive feasible and effective gradients tailored for application-specific training algorithms. This appendix introduces two examples where bilevel optimization has been successfully integrated in meta-learning and model pruning, respectively, providing further insights into how this challenge can be addressed.

D.1 Meta-Learning

This section is based on [46]. Meta-learning, or "learning to learn," is a machine learning approach that enables algorithms to learn new tasks more efficiently by leveraging prior knowledge from previous tasks. Unlike traditional methods, meta-learning focuses on improving the learning process itself, allowing models to quickly adapt to new tasks with minimal data and training.

Assume that a collection of tasks $\{\mathcal{T}_i\}_i^M$, each associated with dataset $\mathcal{D}_i$ separated into partitions $\mathcal{D}_i^{\text{tr}}$ and $\mathcal{D}_i^{\text{test}}$, are given. In this setup, a set of meta-parameters $\boldsymbol{\theta}$ serves as an initialization for training algorithms that solve tasks associated with datasets $\mathcal{D}_i$, resulting in updated parameters $\boldsymbol{\phi}_i = \mathcal{A}lg_i(\boldsymbol{\theta},\mathcal{D}_i^{\text{tr}})$. The meta-learning task is then structured as a bilevel optimization problem through the formulation

$$\begin{aligned} \underset{\boldsymbol{\theta}\in\mathbb{R}^N}{\text{minimize}} \ & \frac{1}{M}\sum_{i=0}^{M} \mathcal{L}_i\Big(\mathcal{A}lg_i^*(\boldsymbol{\theta},\mathcal{D}_i^{\text{tr}}),\mathcal{D}_i^{\text{test}}\Big) \\ \text{s.t.} \quad & \mathcal{A}lg_i^*(\boldsymbol{\theta},\mathcal{D}_i^{\text{tr}}) \in \underset{\boldsymbol{\phi}_i\in\mathbb{R}^N}{\text{argmin}}\ \mathcal{L}_i(\boldsymbol{\phi}_i,\mathcal{D}_i^{\text{tr}}) + \frac{\lambda}{2}\|\boldsymbol{\phi}_i-\boldsymbol{\theta}\|_2^2, \end{aligned}$$

for $i = 1,\dots,M$. Thus, the upper-level objective is tasked with finding a good initialization that suits all the various tasks specified by the lower-level objectives. The additional regularization term $\|\boldsymbol{\phi}_i-\boldsymbol{\theta}\|_2^2$ introduces convexity of the lower-level objective and enforces similarity between the task specific parameters $\boldsymbol{\phi}_i$ and meta-parameters $\boldsymbol{\theta}$ while also being an essential addition in the derivation of a feasible and nontrivial implicit Jacobian (IJ) (see Equation (3.2)). By use of Theorem 3.2, it can be shown that the IJ in this case becomes

$$\frac{\mathrm{d}\mathcal{A}lg_i^*(\boldsymbol{\theta},\mathcal{D}_i^{\text{tr}})}{\mathrm{d}\boldsymbol{\theta}} = \left(I_N + \frac{1}{\lambda}\nabla^2_{\boldsymbol{\phi}_i}\mathcal{L}_i(\boldsymbol{\phi}_i,\mathcal{D}_i^{\text{tr}})|_{\boldsymbol{\phi}_i=\mathcal{A}lg(\boldsymbol{\theta},\mathcal{D}_i^{\text{tr}})}\right)^{-1}, \tag{D.1}$$

where I_N is the $N \times N$-dimensional identity matrix. This leads to the derivative of the upper-level objective being

$$\frac{1}{M}\sum_{i=0}^{M}\left(I_N + \frac{1}{\lambda}\nabla^2_{\boldsymbol{\phi}}\mathcal{L}_i(\boldsymbol{\phi}_i, \mathcal{D}_i^{\text{tr}})|_{\boldsymbol{\phi}=\mathcal{A}lg(\boldsymbol{\theta},\mathcal{D}_i^{\text{tr}})}\right)^{-1}\nabla_{\boldsymbol{\phi}_i}\mathcal{L}_i\Big(\boldsymbol{\phi}_i, \mathcal{D}_i^{\text{test}}\Big)\Big|_{\boldsymbol{\phi}_i=\mathcal{A}lg_i^*(\boldsymbol{\theta})}, \quad \text{(D.2)}$$

This setup outlines a practical training algorithm which recursively finds solutions to all lower-level objectives $\boldsymbol{\phi}_i$, while storing all gradients $\nabla_{\boldsymbol{\phi}}\mathcal{L}_i(\boldsymbol{\phi}_i, \mathcal{D}_i^{\text{tr}})$ that are subsequently used to update the upper-level objective via (D.2) and Algorithm 1. The regularization objective is here crucial for leveraging the benefits of bilevel optimization, as it can be shown that excluding it would cause (D.1) to equal the zero matrix. A similarly structured regularization objective is implemented in BiSSL [49], as detailed in Chapter 4.

D.2 Model Pruning

This section is based on [43]. Model pruning is a technique used in deep learning that eliminates less important weights of a deep neural network. The primary objective of pruning is to create a smaller, more efficient model that maintains most of the original model's performance while requiring fewer resources. This task inherently involves solving two tasks: finding a pruning mask $\mathbf{m} \in \{0,1\}^N$ and model parameters $\boldsymbol{\theta} \in \mathbb{R}^N$ such that that $\|\mathbf{m}\|$ is sufficiently small and the pruned model with parameters $\mathbf{m} \odot \boldsymbol{\theta}$ still performs effectively. Given the intrinsic interdependence between $\mathbf{m}$ and $\boldsymbol{\theta}$, it is appropriate to formulate this problem as a bilevel optimization problem. The model pruning task, with the loss function denoted as ℓ, is formulated as a bilevel optimization problem:

$$\underset{\mathbf{m}\in\{0,1\}^N}{\text{minimize}}\ \ell(\mathbf{m}\odot\boldsymbol{\theta}^*(\mathbf{m})) \quad \text{s.t.} \quad \boldsymbol{\theta}(\mathbf{m}) \in \underset{\boldsymbol{\theta}^*\in\mathbb{R}^N}{\text{argmin}}\ \ell(\mathbf{m}\odot\boldsymbol{\theta}) + \frac{\gamma}{2}\|\boldsymbol{\theta}\|_2^2.$$

In practice, the mask $\mathbf{m}$ is relaxed to lie in $[0,1]^N$ during training, allowing meaningful gradients to be obtained with respect to $\mathbf{m}$. After training, hard thresholding is then applied on $\mathbf{m}$ such that it only consists of binary entries. The term $\|\boldsymbol{\theta}\|_2^2$ is included to enforce approximate convexity of the lower-level objective and, importantly, to ensure an advantageous expression for the upper-level derivative. To see why, using Theorem 3.2, the IJ can in this context be expressed as:

$$\frac{\mathrm{d}\boldsymbol{\theta}^*(\mathbf{m})}{\mathrm{d}\boldsymbol{\theta}} = -\nabla^2_{\mathbf{m}\theta}\ell(\mathbf{m}\odot\boldsymbol{\theta}^*)\Big[\nabla^2_{\boldsymbol{\theta}}\ell(\mathbf{m}\odot\boldsymbol{\theta}^*) + \gamma I_N\Big]^{-1}.$$

Now, using the Hessian-free assumption $\nabla^2_{\boldsymbol{\theta}}\ell(\mathbf{m}\odot\boldsymbol{\theta}^*) = \mathbf{O}$, the expression simplifies to

$$\frac{\mathrm{d}\boldsymbol{\theta}^*(\mathbf{m})}{\mathrm{d}\boldsymbol{\theta}} = -\frac{1}{\gamma}\nabla^2_{\mathbf{m}\theta}\ell(\mathbf{m}\odot\boldsymbol{\theta}^*). \quad \text{(D.3)}$$

This assumption is relatively reasonable when using linear layers with piecewise linear activation functions like the ReLU activation [99]. Although (D.3) appears to involve computation of a Hessian matrix, due to the specific relation between $\mathbf{m}$ and $\boldsymbol{\theta}$ being $\mathbf{m} \odot \boldsymbol{\theta}$, it can be shown that

$$\frac{\mathrm{d}\boldsymbol{\theta}^*(\mathbf{m})}{\mathrm{d}\boldsymbol{\theta}} = -\frac{1}{\gamma}\mathrm{diag}(\nabla_{\mathbf{z}}\ell(\mathbf{z})),$$

which leads to the total derivative of the upper-level objective being

$$\frac{\mathrm{d}\ell(\mathbf{m} \odot \boldsymbol{\theta}^*)}{\mathrm{d}\mathbf{m}} = (\boldsymbol{\theta}^* - \frac{1}{\gamma}\mathbf{m} \odot \nabla_{\mathbf{z}}\ell(\mathbf{z})) \odot \nabla_{\mathbf{z}}\ell(\mathbf{z}),$$

suggesting that the exact upper level gradient (under the Hessian-free assumption) can feasibly be calculated using a first-order optimization method. From here, proposing a training algorithm that alternates between updating the upper and lower-level objectives is relatively straightforward.

To re-emphasize, adding the weight regularization term to the lower-level objective is crucial in this case for obtaining the expression, as otherwise, relying on the Hessian-free assumption would result in the IJ becoming zero. This example also illustrates how the specific structure of the relationship between the upper and lower-level parameters can lead to a more practically feasible expression of the upper-level derivative.

E | Importance of the Second Term of the Upper-Level in BiSSL

Recall the bilevel optimization problem of BiSSL in Definition 4.1. At first glance, the second term in the upper-level objective of (4.1) might appear redundant, since the upper and lower levels are already coupled via $\boldsymbol{\theta}_P^*(\boldsymbol{\theta}_D)$. However, this term plays a crucial role in ensuring that the lower-level problem (4.2) remains non-trivial and meaningfully contributes to the overall bilevel objective. To illustrate this, first consider the standard fine-tuning optimization problem

$$\min_{\boldsymbol{\theta},\boldsymbol{\phi}_D} \mathcal{L}^D(\boldsymbol{\theta},\boldsymbol{\phi}_D), \tag{E.1}$$

and the simplified bilevel optimization problem

$$\min_{\boldsymbol{\theta}_D,\boldsymbol{\phi}_D} \mathcal{L}^D\left(\boldsymbol{\theta}_P^*\left(\boldsymbol{\theta}_D\right),\boldsymbol{\phi}_D\right) \text{ s.t. } \boldsymbol{\theta}_P^*\left(\boldsymbol{\theta}_D\right) \in \operatorname*{argmin}_{\boldsymbol{\theta}_P} \min_{\boldsymbol{\phi}_P} \mathcal{L}^P\left(\boldsymbol{\theta}_P,\boldsymbol{\phi}_P\right) + \lambda\|\boldsymbol{\theta}_D - \boldsymbol{\theta}_P\|_2^2. \tag{E.2}$$

The above problem corresponds to the BiSSL setup in Definition (4.1), but with the second term in the upper-level objective removed and using $r(\boldsymbol{\theta}_D, \boldsymbol{\theta}_P) = \|\boldsymbol{\theta}_D - \boldsymbol{\theta}_P\|_2^2$, reflecting the practical setup to be employed. We now formalize why omitting the second term makes the bilevel structure degenerate in the following lemma.

Lemma E.1 (Solutions Under Removal of the Second Term in Upper-Level) Let $(\bar{\boldsymbol{\theta}},\bar{\boldsymbol{\phi}}_D)$ be stationary points of the conventional fine-tuning optimization problem in (E.1) Define

$$\boldsymbol{\phi}_D^* := \bar{\boldsymbol{\phi}}_D, \quad \boldsymbol{\theta}_P^* := \bar{\boldsymbol{\theta}}, \quad \boldsymbol{\theta}_D^*(\boldsymbol{\phi}_P) := \bar{\boldsymbol{\theta}} + \frac{1}{\lambda}\nabla_{\boldsymbol{\theta}}\mathcal{L}^P(\boldsymbol{\theta},\boldsymbol{\phi}_P)|_{\boldsymbol{\theta}=\bar{\boldsymbol{\theta}}},$$

and let $\boldsymbol{\phi}_P^*$ be a stationary point in the sense that $\nabla_{\boldsymbol{\phi}_P}\mathcal{L}^P(\bar{\boldsymbol{\theta}},\boldsymbol{\phi}_P)|_{\boldsymbol{\phi}_P=\boldsymbol{\phi}_P^*} = \mathbf{0}$. Then the parameters $(\boldsymbol{\theta}_D^*(\boldsymbol{\phi}_P^*),\boldsymbol{\phi}_D^*,\boldsymbol{\theta}_P^*,\boldsymbol{\phi}_P^*)$ satisfies the stationary conditions of the bilevel optimization problem in (E.2). Additionally, the stationary condition

$$\nabla_{\boldsymbol{\theta}} G(\boldsymbol{\theta}_D^*(\boldsymbol{\phi}_P), \boldsymbol{\theta}, \boldsymbol{\phi}_P)|_{\boldsymbol{\theta}=\boldsymbol{\theta}_P^*} = \mathbf{0}$$

holds $\forall \boldsymbol{\phi}_P \in \mathbb{R}^{P_P}$.

The lemma shows that in the absence of the second term in the upper-level objective, the bilevel problem in (E.2) effectively reduces to standard fine-tuning. The regularization term in the lower-level acts only to pull $\boldsymbol{\theta}_P$ towards $\boldsymbol{\theta}_D$ but since the upper-level is

unconstrained in its choice of $\boldsymbol{\theta}_D$, it can simply set it to enforce $\boldsymbol{\theta}_P = \bar{\boldsymbol{\theta}}$. As a result, the bilevel structure becomes vacuous, where the lower-level recovers the exact same solution as achieved by conventional fine-tuning. In contrast, when the second term is included as in (4.1), the upper-level must then balance optimizing $\boldsymbol{\theta}_D$ for downstream performance while also shaping the lower-level solution to be useful for that purpose, providing a more complex interaction between the objectives.

Note: In the original work, we provided a simplified version of the above result, omitting the task-specific heads to enhance focus solely on the backbone dynamics, and claimed that this omission did not alter the outcome. Lemma E.1 makes this claim rigorous, as it shows that the backbone solutions $\boldsymbol{\theta}_P^*$ and $\boldsymbol{\theta}_D^*$ retain the same structure regardless of how the heads are obtained.

Proof (Lemma E.1): We verify that the provided parameters $\boldsymbol{\theta}_P^*$, $\boldsymbol{\phi}_D^*$, $\boldsymbol{\theta}_D^*(\boldsymbol{\phi}_P^*)$ and $\boldsymbol{\phi}_P^*$ are stationary points. Since $(\bar{\boldsymbol{\theta}},\bar{\boldsymbol{\phi}}_D)$ are stationary for (E.1), we immediately have for the upper-level of (E.2) that

$$\nabla_{\boldsymbol{\phi}_D}\mathcal{L}^D(\boldsymbol{\theta}_P^*,\boldsymbol{\phi}_D)|_{\boldsymbol{\phi}_D=\boldsymbol{\phi}_D^*} = \nabla_{\boldsymbol{\phi}_D}\mathcal{L}^D(\bar{\boldsymbol{\theta}},\boldsymbol{\phi}_D)|_{\boldsymbol{\phi}_D=\bar{\boldsymbol{\phi}}_D} = \mathbf{0},$$

and

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}^D(\boldsymbol{\theta},\boldsymbol{\phi}_D^*)|_{\boldsymbol{\theta}=\boldsymbol{\theta}_P^*} = \nabla_{\boldsymbol{\theta}}\mathcal{L}^D(\boldsymbol{\theta},\bar{\boldsymbol{\phi}}_D)|_{\boldsymbol{\theta}=\bar{\boldsymbol{\theta}}} = \mathbf{0}.$$

Since $\bar{\boldsymbol{\theta}} = \boldsymbol{\theta}_P^*$, the lower-level head solution $\boldsymbol{\phi}_P^*$ satisfies its stationarity condition by design: $\nabla_{\boldsymbol{\phi}_P}\mathcal{L}^P(\bar{\boldsymbol{\theta}},\boldsymbol{\phi}_P)|_{\boldsymbol{\phi}_P=\boldsymbol{\phi}_P^*} = \mathbf{0}$. The stationarity of the lower-level with respect to its backbone parameters is verified as follows:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}}G(\boldsymbol{\theta}_D^*(\boldsymbol{\phi}_P^*),\boldsymbol{\theta},\boldsymbol{\phi}_P^*)|_{\boldsymbol{\theta}=\boldsymbol{\theta}_P^*} &= \nabla_{\boldsymbol{\theta}}\mathcal{L}^P(\boldsymbol{\theta},\boldsymbol{\phi}_P^*)|_{\boldsymbol{\theta}=\boldsymbol{\theta}_P^*} + \lambda(\boldsymbol{\theta}_P^* - \boldsymbol{\theta}_D^*(\boldsymbol{\phi}_P^*)) \\
&= \nabla_{\boldsymbol{\theta}}\mathcal{L}^P(\boldsymbol{\theta},\boldsymbol{\phi}_P^*)|_{\theta=\bar{\boldsymbol{\theta}}} + \lambda(\bar{\boldsymbol{\theta}} - \frac{1}{\lambda}\nabla_{\boldsymbol{\theta}}\mathcal{L}^P(\boldsymbol{\theta},\boldsymbol{\phi}_P^*)|_{\theta=\bar{\boldsymbol{\theta}}} - \bar{\boldsymbol{\theta}}) \\
&= \mathbf{0}.
\end{aligned}$$

Finally, we see that substituting any $\boldsymbol{\phi}_P \in \mathbb{R}^{P_P}$ in place of $\boldsymbol{\phi}_P^*$ in the derivations above still satisfies the lower-level stationary condition with respect to the backbone parameters, proving the latter statement of the lemma. ■

F | Additional Results

F.1 Results Tables

This section holds tables with the numerical values depicted in the Figures of the experiments of Chapter 6.

F.1.1 Hyperparameter Influence

Tables F.1-F.5 shows the table entries corresponding to the Figures 6.1- 6.4 that presents the results from the experiments outlined in Section 6.3.

Table F.1: Downstream Top-1 accuracies on the Pets dataset corresponding to the "BiSSL+FT" curve in Figure 6.1, reporting the impact of varying the regularization strength λ in BiSSL.

λ	1.0	0.1	0.01	**0.001**	0.0001
BiSSL + FT	77.4 ± 0.2	77.3 ± 0.2	77.8 ± 0.3	77.7 ± 0.5	77.3 ± 0.1

Table F.2: Downstream Top-1 accuracies on the Pets dataset corresponding to the "BiSSL+FT" curve in Figure 6.2, reporting the impact of varying the number of training stage alternations T in BiSSL.

T	100	200	300	400	**500**	600	800	1000
BiSSL + FT	74.6 ± 0.5	77.8 ± 0.2	78.0 ± 0.3	77.8 ± 0.2	77.7 ± 0.5	77.5 ± 0.3	76.6 ± 0.4	78.5 ± 0.4

Table F.3: Downstream Top-1 accuracies on the Pets dataset corresponding to the "BiSSL+FT" curve in Figure 6.3, reporting the impact of varying the number upper-level iterations N_U in BiSSL.

N_U	1	2	4	**8**	10	12
BiSSL + FT	75.2 ± 0.4	76.2 ± 0.6	78.2 ± 0.3	77.7 ± 0.5	78.3 ± 0.2	78.5 ± 0.3

F.1.2 Adaptive Scaling of λ

Tables F.6- F.8 shows the table entries corresponding to Figures 5.1 6.8 that presents the results from the experiments outlined in Section 6.4.

Table F.4: Downstream Top-1 accuracies on the Pets dataset corresponding to the "BiSSL+FT" curve in Figure 6.4, reporting the impact of varying the number lower-level iterations N_L in BiSSL.

N_L	1	2	5	10	**20**	30	40	50
BiSSL + FT	77.2 ± 0.2	77.7 ± 0.5	77.9 ± 0.4	77.7 ± 0.2	77.7 ± 0.5	77.6 ± 0.3	77.5 ± 0.1	77.6 ± 0.6

Table F.5: Downstream accuracies on the Pets, DTD, VOC07, and Flowers datasets corresponding to the "Only FT" and "BiSSL+FT" curves in Figure 6.5, reporting the impact of varying the number of subsequent fine-tuning epochs.

FT Epochs	10	25	50	100	200	300	**400**
Pets:							
BiSSL + FT	77.8 ± 0.2	77.3 ± 0.4	77.7 ± 0.3	77.7 ± 0.3	77.9 ± 0.2	77.7 ± 0.4	77.7 ± 0.5
Only FT	70.6 ± 0.6	75.7 ± 0.5	75.5 ± 0.5	74.3 ± 0.8	76.0 ± 0.4	74.0 ± 0.5	73.2 ± 0.3
DTD:							
BiSSL + FT	62.9 ± 0.5	63.0 ± 0.7	62.8 ± 0.7	63.6 ± 0.3	63.8 ± 0.5	63.8 ± 0.3	63.8 ± 0.3
Only FT	58.2 ± 0.9	59.7 ± 0.5	59.8 ± 0.6	60.5 ± 0.7	60.3 ± 0.8	60.3 ± 0.7	60.3 ± 0.9
VOC07:							
BiSSL + FT	70.0 ± 0.2	70.2 ± 0.2	71.4 ± 0.1	71.4 ± 0.1	71.4 ± 0.1	71.3 ± 0.1	71.4 ± 0.1
Only FT	67.5 ± 0.4	69.6 ± 0.2	69.4 ± 0.1	69.6 ± 0.1	70.9 ± 0.2	70.9 ± 0.2	71.0 ± 0.1
Flowers:							
BiSSL + FT	51.2 ± 0.7	84.0 ± 0.2	83.9 ± 0.2	84.2 ± 0.2	84.1 ± 0.2	84.3 ± 0.2	84.2 ± 0.3
Only FT	26.9 ± 1.3	78.6 ± 0.4	80.4 ± 0.4	81.2 ± 0.5	82.4 ± 0.4	81.1 ± 0.5	82.6 ± 0.3

Table F.6: Downstream top-1 accuracies on the Pets dataset corresponding to the 'Cosine Increase' and 'Cosine Decrease' curves in Figure 6.6, showing the impact of applying cosine-based λ schedules during BiSSL training, with varying values for the maximal schedule parameter λ_{max}.

λ_{max}	10.0	1.0	0.1	0.01	0.002	0.001	0.0001
Cosine Increase	66.5 ± 0.3	66.5 ± 0.3	59.4 ± 0.5	64.6 ± 0.2	70.3 ± 0.3	69.5 ± 0.3	65.3 ± 0.6
Cosine Decrease	66.2 ± 0.2	66.0 ± 0.3	63.1 ± 0.3	60.3 ± 0.4	70.1 ± 0.3	68.8 ± 0.8	67.1 ± 0.3

Table F.7: Downstream top-1 accuracies on the Pets dataset corresponding to the curves in Figure 6.7, showing the impact of applying an exponential scheduler on λ during BiSSL training, with varying values for the maximal schedule parameter λ_{max}.

λ_{max}	0.1	0.01	0.001	0.0001
Exponential Increase ($\alpha = 1$)	61.3 ± 0.4	57.7 ± 0.7	67.1 ± 0.4	64.8 ± 0.7
Exponential Increase ($\alpha = 10$)	65.4 ± 0.4	57.4 ± 0.6	70.1 ± 0.4	65.4 ± 0.6
Exponential Increase ($\alpha = 100$)	64.6 ± 0.3	57.3 ± 0.7	69.9 ± 0.4	65.8 ± 0.3

Table F.8: Downstream top-1 accuracies on the Pets dataset corresponding to the curves in Figure 6.8, showing the impact of making λ learnable with $\lambda_{\text{damp}} = 0$ and $\lambda_{\text{damp}} = 10$ along varying values for the initial value of $\sigma(\lambda)$.

$\sigma(\lambda)_{\text{init}}$	2.0	1.0	0.1	0.01	0.001	0.0001	0.00001
Learnable λ ($\lambda_{\text{damp}} = 0$)	77.0 ± 0.2	76.4 ± 0.2	76.7 ± 0.2	77.1 ± 0.3	77.2 ± 0.3	76.4 ± 0.3	76.7 ± 0.3
Learnable λ ($\lambda_{\text{damp}} = 10$)	76.6 ± 0.2	77.1 ± 0.1	76.4 ± 0.3	77.5 ± 0.3	77.0 ± 0.4	77.1 ± 0.2	75.5 ± 0.2

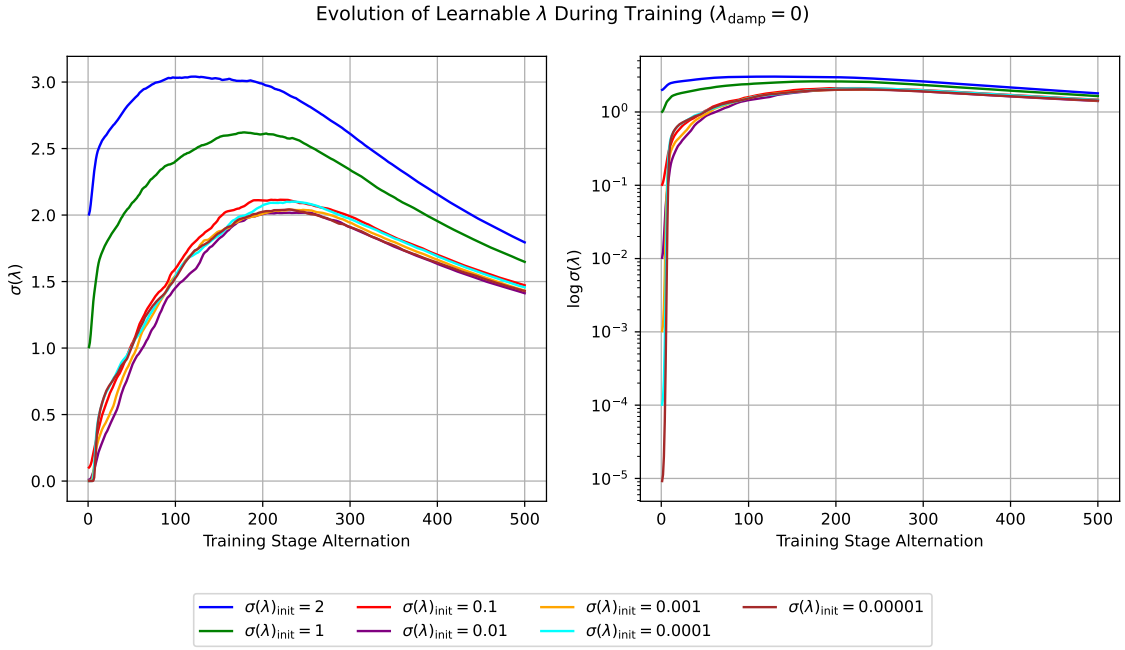

Figure F.1: Evolution of $\sigma(\lambda)$ during BiSSL training with $\lambda_{\text{damp}} = 0$ under various initializations.

F.2 Learnable λ

Figures F.1 and F.2 show how λ evolves during BiSSL training when treated as a learnable parameter, under two configurations: without dampening ($\lambda_{\text{damp}} = 0$) and with dampening ($\lambda_{\text{damp}} = 10$). These trajectories correspond to the experiments introduced in Section 6.4.2.

Without dampening, we observe that λ exhibits a steep initial increase across all runs. However, around the midpoint of training, values begin to decline and eventually converge toward a common value, visually estimated to lie near 1. Notably, all but the runs initialized at 1 and 2 appear to catch the same trajectory early on.

When dampening is applied, the trajectories become more diverse, converge more slowly, and generally appear less noisy. Initial values influence the evolution more strongly, with the runs starting at 1 and 2 decreasing over time, while those with lower initial values increase. Still, the trend suggests a gradual drift toward a common range, though dampening tempers this.

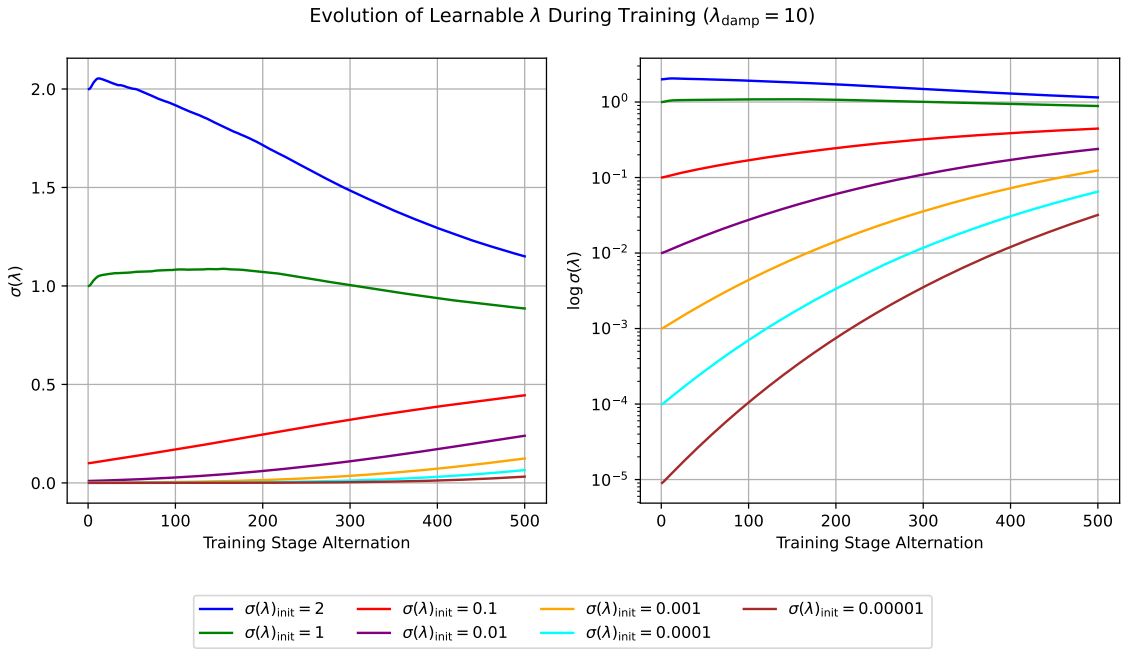

Figure F.2: Evolution of $\sigma(\lambda)$ during BiSSL training with $\lambda_{\text{damp}} = 10$ under various initializations.

G | Future Work: Regularizing the Pretext Head via the Upper-Level

The experiments in Section 6.5 indicate that using the generalized IJ from Proposition 5.5 does not improve downstream performance. One possible explanation, discussed in Chapter 8, is that the pretext head remains insufficiently integrated into the BiSSL bilevel formulation (Definition 4.1). This appendix outlines a proposed extension of BiSSL that embeds the pretext head more directly into the bilevel structure.

We adopt the notation from Section 5.3, working with the alternative BiSSL formulation using a concatenated lower-level parameter vector, given by Definition 5.4. We propose the following modified problem:

$$\begin{aligned} \min_{\boldsymbol{\theta}_D, \boldsymbol{\phi}_D} \quad & \mathcal{L}^D(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)), \boldsymbol{\phi}_D) + \mathcal{L}^D(\boldsymbol{\theta}_D, \boldsymbol{\phi}_D) + \frac{\mu}{2} \|\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))\|_2^2 \\ \text{s.t.} \quad & \boldsymbol{\omega}^*(\boldsymbol{\theta}_D) \in \min_{\boldsymbol{\omega}} \mathcal{L}^P(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}), \gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega})) + \mathcal{L}^P(\boldsymbol{\theta}_D, \gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega})) + \lambda r(\boldsymbol{\theta}_D, \gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega})). \end{aligned}$$

The upper-level now includes a regularization term on the pretext head. This encourages the upper-level to achieve a backbone $\boldsymbol{\theta}_D$ that leads the lower-level to achieve regularized pretext head parameters $\boldsymbol{\phi}_P$. I.e., the upper-level needs now to find a backbone on which the pretext task objective can find a general, likely non-overfitted solution. To ensure this upper-level term is not treated as constant during optimization, we introduce an additional term into the lower-level objective that conditions on $\boldsymbol{\theta}_D$. This creates meaningful gradient paths between the levels through the pretext head. The resulting formulation thus further intertwines the levels through the pretext head, while preserving the structure of the lower-level backbone optimization, aside from the inclusion of additional head gradients.

For future reference, we briefly outline the gradient of the upper-level objective (omitting the second term for simplicity):

$$\begin{aligned} & \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}_D} \left(\mathcal{L}^D(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)), \boldsymbol{\phi}_D) + \frac{\mu}{2} \|\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))\| \right) \\ &= \frac{\mathrm{d}\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))}{\mathrm{d}\boldsymbol{\theta}_D} \nabla_{\boldsymbol{\theta}} \mathcal{L}^D(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)), \boldsymbol{\phi}_D) + \frac{\mathrm{d}\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))}{\mathrm{d}\boldsymbol{\theta}_D} \nabla_{\boldsymbol{\phi}_P} \frac{\mu}{2} \|\gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))\|_2^2 \\ &= \frac{\mathrm{d}\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)}{\mathrm{d}\boldsymbol{\theta}_D} \begin{bmatrix} I_L \\ \boldsymbol{O}_{P_P \times L} \end{bmatrix} \nabla_{\boldsymbol{\theta}} \mathcal{L}^D(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)), \boldsymbol{\phi}_D) + \frac{\mathrm{d}\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)}{\mathrm{d}\boldsymbol{\theta}_D} \begin{bmatrix} \boldsymbol{O}_{L \times P_P} \\ I_{P_P} \end{bmatrix} \mu \gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)) \\ &= \frac{\mathrm{d}\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)}{\mathrm{d}\boldsymbol{\theta}_D} \begin{bmatrix} \nabla_{\boldsymbol{\theta}} \mathcal{L}^D(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)), \boldsymbol{\phi}_D) \\ \mu \gamma_{\boldsymbol{\phi}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)) \end{bmatrix}. \end{aligned}$$

Hence, the above upper-level gradient conveniently still only needs the involvement of a single IJ. Under the same regularity assumptions as imposed in Proposition 4.2, the explicit expression for the IJ is then

$$\frac{\mathrm{d}\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)}{\mathrm{d}\boldsymbol{\theta}_D} = AB,$$

where

$$A = -\Big[\lambda\nabla^2_{\boldsymbol{\theta}_D\boldsymbol{\theta}_P} r(\boldsymbol{\theta}_D,\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) \quad \nabla^2_{\boldsymbol{\theta}_D\phi_P}\mathcal{L}^P(\boldsymbol{\theta}_D,\gamma_{\phi_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)))\Big]$$

and

$$B = \left[\frac{\mathrm{d}^2}{\mathrm{d}^2\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)}\mathcal{L}^P(\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)),\gamma_{\phi_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) + \begin{bmatrix}\lambda\nabla^2_{\boldsymbol{\theta}_P} r(\boldsymbol{\theta}_D,\gamma_{\boldsymbol{\theta}_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D))) & \mathbf{O}_{L\times P_P}\\ \mathbf{O}_{P_P\times L} & \nabla^2_{\phi_P}\mathcal{L}^P(\boldsymbol{\theta}_D,\gamma_{\phi_P}(\boldsymbol{\omega}^*(\boldsymbol{\theta}_D)))\end{bmatrix}\right]^{-1}$$

The derivations of this expression follow closely the structure of the proof of Proposition 5.5 (outlined in Appendix C), and are hence omitted.

This formulation increases computational cost, as the above expression reveals that it requires three Hessian-vector products per gradient step rather than one. Whether this overhead is justified depends on the degree to which it enhances performance or stability in practice.