Exploring Multi-Task Learning for Robust Language Encoding with BERT

Stanford CS224N Default Project

Alejandro Lozano
Department of Biomedical Data Science
Stanford University
lozanoe@stanford.edu

Laura Bravo
Department of Biomedical Data Science
Stanford University
lmbravo@stanford.edu

Abstract

Transformer-based Large Language Models (LLMs) have revolutionized Natural Language Processing (NLP). By analyzing large amounts of text data, LLMs are capable of identifying relationships between words and phrases, as well as their context, resulting in a more nuanced language understanding. LLMs are transferable, allowing them to be pre-trained on large data sets and later fine-tuned on smaller downstream-specific tasks. However, fine-tuning can lead to catastrophic forgetting, where previously learned information is lost. In this work we propose a BERT-based architecture that promotes representation generalization by training on multiple tasks: Sentiment Analysis (SA), Paraphrase Detection (PD), and Semantic Textual Similarity (STS). Our experiments suggest that even when accounting for task interference, a Multi-task Learning (MTL) framework is only effective when it can leverage related tasks.

1 Key Information to include

- Mentor: Hans Hanley

2 Introduction

In recent years, Natural Language Processing (NLP) has been revolutionized by transformer-based Large Language Models (LLMs) [1]. By analyzing vast amounts of text data, LLMs can identify subtle relationships between words and phrases, as well as the context in which they are used, enabling them to better capture the nuances of human language, compared to previous approaches. Additionally, LLMs are transferable, meaning that they can be pre-trained on large amounts of data and then fine-tuned on smaller task-specific datasets. Thus enabling the development of more efficient and effective models for specific NLP tasks, without the need for extensive training data. For instance, BERT [2] (one of the first LLMs) led to state-of-the-art performance in multiple downstream tasks, out-competing specialized systems at the time [3].

However, the fine-tuning process can often lead to catastrophic forgetting, that is, when a neural network forgets previously learned information when trained on new and unrelated data. Thus, leading to a loss of generalizability in the learned representations [4]. This phenomenon has motivated research avenues that improve the robustness of learned representations for downstream tasks. One such approach is Multi-task learning (MTL) which promotes representation generalization by training on multiple related tasks simultaneously. In MTL, models learn to share information across tasks via a common feature extraction backbone, while capturing relevant features for each task with task-specific components [5].

Nevertheless, the nature of MTL involves facing challenges that remain open-ended research questions, for example task interference, which occurs when tasks negatively impact each other's performance [6]. This problem is particularly relevant when tasks have different requirements or are not well-aligned. In this work, we study the problem of developing robust embeddings with LLMs for NLP downstream tasks by proposing a MTL BERT-based architecture and involving strategies to

handle task interference. In particular, we apply our approach to the tasks of Sentiment Analysis (SA), Paraphrase Detection (PD) and Semantic Textual Similarity (STS). We validate our approach across all tasks by comparing it to the single-task fine-tuned paradigm and find that only when the tasks are "related" the MTL approach provides a significant boost in performance (e.g. increased accuracy).

3 Related Work

3.1 Transformer-based Models for Natural Language Understanding

Since the introduction of BERT [2], Transformer based pre-trained language models have dominated the field of Natural Language Understanding. Such architectures have provided unprecedented improvement of accuracy on various tasks compared to traditional models (e.g LSTM) at the cost of increasing the number of parameters, making them computationally expensive and unreliable due to memory limitations of available hardware [7]. The latter hinders their adoption for applications such as sentence-pair regression tasks e.g. large-scale semantic textual similarity, clustering, paraphrasing detection, and information retrieval via semantic search. Hence, several works have addressed this issue with architectonic solutions. Poly-encoders proposed by Humeau *et al.* [8] addresses the run-time overhead from BERT by computing candidate embeddings using attention. However, the score function used by this approach is not symmetric and the computational overhead is still large for some applications (e.g clustering). In [9] a Siamese triplet network architecture (Sentence-BERT) was proposed as a computationally efficient replacement for BERT. As an example, on a modern V100 GPU, hierarchical clustering of 10,000 sentences on standard BERT requires 65 hours (since 50 Million sentence combinations must be compared), while it only takes 5 seconds with Sentence-BERT. Another successful approach was ALBERT [10], where two parameter reduction techniques to lower memory consumption and increase the training speed of BERT were introduced, showing comprehensive empirical evidence that such methods lead to models that scale much better compared to the original BERT.

3.2 General Robust Embeddings

Since Contextualized representations retrieved from pre-trained LLM's are central to achieve high performance on downstream NLP tasks, the search for an optimal sentence embedding scheme remains an active research area in computational linguistics [11]. Although BERT-based models employ the [CLS] token vector as a "reasonable" sentence embedding (common starting point in several works), it has been proven that such embeddings yield worse results than GloVe embeddings [12] for several applications such as sarcasm detection [13], semantic co-occurrence [14], and STS [9]. Thus exhaustive analysis comparing different strategies to retrieve embeddings from BERT (and other Encoders such as ALBERT [10]) have been explored. For example, it has been observed that averaging the BERT tokens (a type token pooling strategy along max pooling) for every word in a sentence provides a better result than the [CLS] token [9]. Furthermore applying a CNN after the pooling operation might slightly boost performance for some tasks, but it also has a negative impact for other tasks [11]. Lastly Merchant *et al.* [15] suggest that the linguistic features are not always incorporated into the final prediction layer (nor in earlier layers) showing that the layers closer to the final BERT embedding (such as the second to last) might provide a boost in performance for downstream task. A possible reason for this observation might be that the last layer provides embeddings that capture features necessary for BERT's pre-training pretext task (MASK LM and Next Sentence Prediction)

3.3 Multi-task-based Robust Embeddings

In contrast to the selection of a pooling strategy or layer representation selection, other works have focused on training strategies to obtain robust embeddings. Hence Representation Learning, Meta-learning, and Multi-Task Learning frameworks have been exhaustively studied as plausible solutions [16, 17, 18]. Under the MTL paradigm, models are jointly trained from multiple related tasks, using shared representations to learn generalized features from a collection of tasks, integrating knowledge across domains [19, 20]. At its most general form, the MTL paradigm only requires tailored architecture design [21, 22]. Most common approaches are parallel, hierarchical, modular, and generative adversarial architectures. In the parallel architecture, a model is shared among

multiple tasks while each task has its own output layer. Hierarchical architectures explicitly model the interaction between tasks, while modular architectures decompose a given model into shared components and task-specific (learning task-invariant and task-specific features respectively) [23, 24]. As a common extension, several MTL workflows aim to improve the robustness among tasks via custom optimization processes such as dynamically tuning gradient magnitudes [25], employing meta objectives[26], and hybrid balance methods [27]. However, these types of approaches simply neglect the possibility of getting feedback from conflicting gradients, which might improve performance, but waste the potential conjoined learning space of an MTL setting. Thus Yu *et al.* proposed Gradient Surgery [28], an algorithm to project a conflicting tasks's gradient onto the normal planes, which has shown state-of-the-art performance in several applications [29, 30, 31, 32].

Theoretically, shared embedding increases data efficiency while making a representation more robust for related downstream tasks (providing attention to relevant features), but in practice, it may lead to degraded performance (this is especially true when tasks compete for model capacity) [22, 33]. Hence, recent observations on MTL have questioned the utility of gradiant surgery, meta objectives, and dynamic gradient tuning paradigms. Xin *et al.*[34] suggest that such MTL strategies might not provide better performance than simply optimizing a weighted average of the task losses (obtained via hyper-parameter search). Other studies have showed that a possible reason for such observation may not be a combination of several factors since, unlike transfer task affinities, multi-task affinities are highly sensitive to a number of components external to conjoined loss functions such as dataset size and network capacity [35].

4 Approach

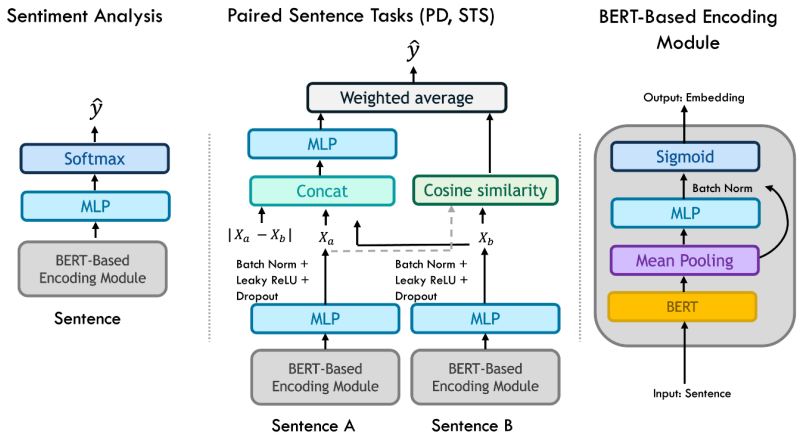

Figure 1: Final architecture for Multi-Task Learning approach. All tasks share the BERT-Based Encoding Module for embedding the input sentence.

To (theoretically) improve the robustness of the learned embeddings we experimented with the MLT paradigm. Figure 1 depicts our proposed final architecture (after exploring different combinations). We then compare the performance of MTL against single-task training using the same model (this serves as our baseline).

Shared BERT Backbone. To promote shared learning all tasks share a BERT-Based Encoding Module h, which takes a sentence s as input and outputs an encoded representation $e = h(s)$ which is then used for the task-specific heads. We process the sentence by first tokenizing the words, adding positional embeddings, and encoding them with the BERT model.

Task-specific heads. For SA we use a straightforward head composed by a MLP and Softmax function to predict the degree of similarity $\hat{y}_{SA} \in [0, 1, 2, 3, 5]$. For PD and STS, as they are based on comparing two sentences our design is inspired by the work from Reimers *et al.* [36]. We employ a siamese network approach to process two input sentences A and B and obtain their representations X_a and X_b. The final output (for our best performing model) is predicted by using a weighted

average between cosine similarity and a mapping function f over these representations to predict $\hat{y}_{STS} \in \mathbb{R} \in [0, 5]$ and $\hat{y}_{PD} \in [0, 1]$. In particular

$$\hat{y}_{task} = \alpha f(concat(X_a, X_b, |X_a - X_b|)) + (\alpha - 1)cosine - similarity(X_a, X_b) \quad (1)$$

Note that for STS the cosine-similarity function is multiplied by 5 to get $\hat{y}_{STS} \in \mathbb{R} \in [0, 5]$

Task-specific head selection We proposed four different architectures and compared their performance in every task (decoupled). For every architecture version, the most significant variations are applied to the Siamese neural network used for STS and PD (As the exploration for an SA architecture was done in the first part of the project).

1. Architecture V_1 (shown in figure Figure 2 located on the Appendix) is a simplified version of V_3 (depicted in Figure 1). Every sentence is encoded into the CLS token and then fed directly into an MLP (different for every task), then the resulting embeddings are fed to a cosine similarity function for PD and STS and soft-max for SA .
2. For V_2 we introduced the BERT-Base Encoding Module of architecture V_3 and left everything else equal to $V1$
3. Architecture V_3a and V_3b (shown in Figure 1) essentially the same, but with different values of α_{task} (shown in equation 1), which reflects a trade-off between an MLP and cosine-similarity function. Hence the two difference between this and the previous architectures is the addition of a weighted average between an MLP and $Cosine - similarity$ function as a final layer and the fact that the MLP applied after to the BERT-Based Encoding Modules is shared among all tasks.
   (a) For V_3a we selected the $\alpha_{PD} = 0.99$ (adding more weight to the MLP layer), $\alpha_{STS} = 0.01$ (adding more weight to the Cosine-similarity layer)
   (b) for V_3b $\alpha_{PD} = 0.01$ (adding more weight to the Cosine-similarity layer) and $\alpha_{STS} = 0.99$(adding more weight to the MLP layer)

BERT Embeddings. We note that as the sentence representation four our final model V_3, instead of using the hidden state of the $< CLS >$ token of the last BERT Layer, we use a mean pooling across all tokens of a given word from the second to last BERT Layer and multiplied this representation by its respective attention weight, as shown in Figure 1. Following [11] we added an optional 1D-convolutional layer on top of these embeddings (The selection of the best performing BERT Embedding is explained in the next section).

Loss functions. For joint training we define a MTL loss function $\mathcal{L}_{overall}$ composed by adding the individual task loss functions $\mathcal{L}_{task}$. $\mathcal{L}_{SA}$ is defined by a standard cross-entropy loss. PD is formulated as a binary classification problem, thus we used a binary-cross entropy, cosine embedding loss, and triplet loss to define $\mathcal{L}_{PD}$ inorder to ensure that the embeddings are expressive, meaning that the model does not trivially learn to increment the norm of some word embeddings to capture similarity as denoted in Bordes *et al.* [37]. Finally, STS is a regression problem, therefore we define its loss ($\mathcal{L}_{STS}$) as the weighted average between the mean-square-error and Lasso losses.

$$\begin{aligned}
\mathcal{L}_{overall} &= c_1\mathcal{L}_{SA} + c_2\mathcal{L}_{PD} + c_3\mathcal{L}_{STS} \\
\mathcal{L}_{SA} &= L_{ce} \\
\mathcal{L}_{PD} &= L_{bce} + L_{cosine-sim-emb} + L_{tripplet-margin-loss} \\
\mathcal{L}_{STS} &= (\gamma)L_{mse} + (\gamma - 1)L_{lasso}
\end{aligned}$$

where the c_i are the scaling constants to project all $\mathcal{L}_{task}$ to the same magnitude.

Time-efficient cosine triplet loss functions for Paraphrase Detection The triplet margin loss function requires an anchor, positive, and negative examples respectively. We compute the cosine triplet loss functions when the label is equal to one, setting the two encoded sentences as an anchor and positive example and a randomly sampled sentence as the negative example. A naive approach to obtain such negative representations would be to create a second data loader (shuffled differently) and sample from two loaders at the same time in order to calculate the BERT embeddings for

positive negative, and anchor examples. To avoid this implementation (which could potentially add more hardware constraints), we proposed a time-efficient implementation where we store the BERT representations of a given batch to be used as the negative samples for the next batch. Though this idea was implemented from scratch, it is a popular approach in Computer Vision (Contrastive Learning) [38, 39] and Knowledge Graph embedding algorithms [40].

Multi-task training strategy. We handle task interference by following the work from Yu *et al.* [28], which identified conflicting gradients (gradients for different tasks pointing away from one another) as the primary optimization issue in MTL. Though averaging over the task gradients could provide (under specific assumptions) a correct solution, there are key scenarios in which this may lead to degraded performance. Hence, we implement Gradient Surgery (GS) [28], a method for handling conflicting gradients across tasks, to jointly train our MTL model. Formally, given a collection of tasks $\mathbb{T}_i \in \mathbb{T}$ for $i = \{1, 2, ..n\}$. The gradient for task $\mathbb{T}_j$ is denoted as g_j, while the gradient for task $\mathbb{T}_i$ $i \neq j$ is given by g_i. If a gradient conflict is encountered (a negative cosine similarity) between g_i and g_j, g_i is replaced by its projection onto the normal plane of g_j: $g_i = gi - \frac{g_i \cdot g_j}{||g_j||^2} g_j$. Otherwise, when the cosine similarity is non-negative (non-conflict gradients), the original gradient g_i is unaltered. This process is repeated across all of the other tasks, sampled randomly.

5 Experiments

5.1 Data

For each task we use the datasets from the project instructions including the proposed data splits. For SA: Stanford Sentiment Treebank [41]. For PD: Quora question pairs dataset [42], and for STS: SemEval STS Benchmark dataset [43].

5.2 Evaluation method

We use accuracy and F1 to evaluate the performance for the SA and PD tasks as they are formulated as classification tasks. For the STS task we calculate the Pearson correlation and EM between the predicted and true similarity values.

5.3 Experimental details

For all experiments, we use an NVIDIA A10 GPU. We perform our experiments from a pre-trained checkpoint (provided for the project), thus every subsequent experiment starts (fine-tunes) from this model. In some experiments, we freeze a previously fine-tuned model to further fine-tuned for a single task (fine-tune of the fine-tune), we will refer to this as fine-tuning (and use the word training when we fine-tune the original checkpoint)

Single Task fine-tuning We perform this experiment twice, with two different gaols

1. **Architecture Selection .** To select the best architecture we train every model for 3 epochs with learning rate $1 \times 10e^{-5}$ batch size 16 and dropout probability rate of 0.7 (This was done due to resource constraints).
2. **Base line.** To compare The performance against MTL we train the best performing model (from the previous experiment) on a single task using the following parameters: epochs 15 (with early stopping), learning rate $5 \times 10e^{-6}$ batch size 32, dropout probability 0.7.

Embedding Selection After selecting the best architecture, we explore three BERT embeddings strategies (CLS token, mean pooling, mean pooling + 1D-CNN), due to resource constraints we only train the models for 3 epochs with a dropout probability of 0.7, batch size of 8, a learning rate of $5 \times 10e^{-6}$, and a gradient surgery workflow,

MTL GS training For MTL we train our best-performing architecture from table 2 (V_3) using a dropout probability of 0.65, batch size of 8 with 4 gradient accumulation steps, a learning rate of $8 \times 10e^{-6}$, 15 epochs (01:23 hours per epoch), a gradient surgery workflow, and average pooling BERT embeddings.

Table 1: Test Set Results for all tasks

(a) Sentiment Analysis

Model	F1
Final	52.35

(b) Paraphrase Detection

Model	F1
Final	76.37

(c) Semantic Textual Similarity

Model	EM
Final	61.08

MTL to Single Task fine-tuning After training the model with a MTL paradigm we fine-tuned the resulting checkpoint for every task using the following parameters: epochs 10, batch size 32, dropout probability 0.7, learning rate $6 \times 10e^{-7}$

Ablation studies

1. We train MTL-GS with "similar" tasks (removing SA and leaving PD and STS) with the following parameters. Epochs 10, batch size 16, hidden dropout probability 0.7, learning rate $2 \times 10e^{-6}$ with equal weight for both loss functions (unscaled) $c_2 = c_3 = 1$ and scaled loss functions $c_2 = 1c_3 = 4$. For both scenarios, every epoch took 00:37:15 minutes on average.
2. To see the impact of the loss we remove $L_{cosine-sim-emb}$ and $L_{tripplet-loss}$ from L_{PD}, this experiemnt is denoted as PD_{ce} (since it only uses cross-entropy) . We also train this experiment with equal weight for both loss functions (unscaled) $c_2 = c_3 = 1$ and scaled loss functions $c_2 = 1c_3 = 4$. All the other parameters are left equal to the first ablation study (described above). For both scenarios, every epoch took 00:30:23 minutes on average.

5.4 Results and Analysis

Test set results. Table 1 shows the results of our MTL approach on the test set (obtained fom the lead board). As expected, the results are lower than those in the Dev set. We also note that the relative performance ordering of the tasks is maintained. We obtain position 87 on the lead-board with a global score of 63.27.

Dev Set Results: Architecture Selection. Table 2 shows the results of experimenting with variations of our proposed MTL architecture $V_3(final)$ (see Figure1). Note that these results were obtained by training the model for every single task for 3 epochs. From this, we can derive four conclusions.

1. Adding a layer on top of the BERT embeddings (Before feeding them to task-specific MLPs) further increases the performance of architecture. The latter can be seen by comparing V_1 against V_2. Our hypothesis is that this approach works since we are adding an extra layer to learn semantic features important for a downstream task, which helps the model "diverge" from the feature learned for BERT's pre-train strategy.
2. Comparing the different variations of V_3a we can conclude that Mean-Pooling tokens provide more expressive embeddings than CLS tokens. This result aligns with Wang *et al.* [44] and Reimers *et al.* [9] (an example of initial comparison between Mean-Pooling and CLS token is provided in the Appendix)
3. Seemingly similar tasks require different architecture designs. As shown V_3a vs V_3b, PD greatly benefitted from having an MLP as a final layer while STS performed the best when a scaled cosine-similarity layer was used. This aligns with previous observations from Reimers *et al.* [9] and Viji *et al.* [45] that using cosine similarity for STS yields the best results compared to other methods such as imputing two sentences to BERT (separated with the [SEP] token) and applying an MLP head on top.
4. Sharing an MLP among across tasks benefits* the performance of a model. This observation can be seen by comparing V_2 vs V_3a and V_3b (only using the results of the cosine embedding to set all things equal). However, it is important to clarify (as pointed out in the next subsection) that this only benefits tasks if they are similar in nature (e.g. PD and STS). This insight is supported by recent progress in MTL, where researchers are trying to exploit the complex relationships across different tasks by sharing layers across different task heads [46, 47, 48].

Dev Set Results: MTL-GS Does it really work? able 3 shows the comparison of our final method (V_3 architecture with MTL and GS) with the baselines of training a single task by decoupling our proposed architecture's task-specific heads. For clarity Table 3 also notes how the model was fine-tuned. we hilight the following:

1. Version V_3b shows how the BERT-Based Encoding Module improves the results for STS, but does not perform better than V_1 for SA and PD (which uses CLS tokens). We hypothesize that for our proposed sentence representation to be effective, it needs to be paired with an effective architecture design (especially at the final layer)in order to benefit all tasks.
2. As it has been reported previously, jointly training all tasks with (GS-MTL) provided robust embeddings (best average overall tasks), but it leads to degraded performance [22, 33], meaning that jointly training **all** tasks does not yield the best result possible since single-task training for SA and PD outperforms MTL-GS (only STS benefits).
3. However, from previous experiments we observed an improvement when including GS to MTL optimization. Thus, we hypothesize that there was task interference (even after projecting the gradients) which could be caused by dissimilar tasks interacting. It is possible that by using all tasks, the additional examples lead to the creation of an even sparser embedding space to the detriment of the representation of individual tasks. Following this idea, we perform an ablation study to see if training "similar" tasks would increase performance. As shown by all the variations of GS-MTL (only PD + STS) and GS-MTL, our hypothesis is correct, since this lead to the best-performing models for both PD and STS.
4. For PD and STS, the Best Performing model we obtained was obtained via scaled weights shown by MTL-GS (only PD +STS). This means that GS benefits greatly when the loss functions of different tasks have similar magnitudes.
5. Comparing our loss function ablation study we conclude that adding a triplet loss function benefited the overall performance of MTL-GS for PD and STS. Furthermore, our implementation only adds 7 extra minutes per epoch
6. As a final highlight, we also notice that even when not directly training with MTL tasks some tasks can benefit from the other tasks in a low data regime. For example, there is a marked improvement for SA considering that the model has never been trained for SA (see rows 2 and 3).

Table 2: Results of Architecture Variations with Multi-task Learning on the Dev Set.

Architecture	Embedding Type	SA Acc	PD Acc	STS PC
V_1	CLS Token	45.4	45.6	9.9
V_2	Mean-Pooling	-	66.6	13.6
$V_3a(Final)$	Mean-Pooling	**46.2**	**70.1**	**40.2**
V_3a	Mean-Pooling + CNN	-	67.9	36.9
V_3a	CLS Token	-	66.8	38.9
V_3b	Mean-Pooling	-	37.5	12.3

6 Conclusion

Our analysis shows that multitask learning with gradient surgery using all tasks provides the most robust embeddings based on the average performance. However, it does not provide the best results possible for all individual tasks (compared to single-task fine-tuning) unless the tasks are similar in nature (e.g PD and STS). This finding partially aligns with current views on MTL published by Standly *et al.* [35]. Furthermore, MTL-GS significantly boosts performance when conflicting tasks are excluded and the losses are scaled to a similar magnitude. This shows that even after using MTL-GS it is necessary to use a workflow to regularize the loss functions of all tasks in an MTL paradigm. This opens up future work on how to systematically define similar tasks and how to scale them jointly.

Additionally, fine-tuning on top of MTL-GS with conflicting tasks does not provide a substantial benefit. Contrary to what we expected, it yields lower performance than fine-tuning a single task at a

Table 3: Results of Training Strategies for Multi-task Learning on the Dev Set. GS: Gradient Surgery; MTL: Multi-Task Learning. Ft: Fine tuning.

Model	Training Strategy	SA Ft	PD Ft	STS Ft	SA Acc	PD Acc	STS PC	Average
V_3a	Single task	✓			**51.9**	37.5	43.3	44.23
V_3a	Single task		✓		20.8	71.7	30.7	41.1
V_3a	Single task			✓	37.6	25.5	43.9	35.67
V_3a	MTL				32.6	58.7	52.7	48.0
V_3b	GS-MTL				34.8	37.5	10.2	27.5
V_3a	GS-MTL				46.3	60.1	63.0	56.47
V_3a	GS-MTL	✓			49.9	37.8	45.7	44.47
V_3a	GS-MTL		✓		46.3	65.6	58.9	56.93
V_3a	GS-MTL			✓	45.5	50.7	60.1	52.12
V_3a	GS-MTL (only PD + STS unscaled)				22.3	69.6	**65.2**	52.36
V_3a	GS-MTL (only PD + STS scaled)				23.8	76.3	63.4	54.50
V_3a	GS-MTL (only PD_{ce} + STS unscaled)				21.4	75.3	39.6	45.43
V_3a	GS-MTL (only PD_{ce}+ STS scaled)				21.0	**76.6**	43.3	46.96

time. From our architecture experiments, we found that for a PD task, a final MLP layer performed better than cosine similarity. On the other hand for STS a cosine similarity layer performed better than an MLP. This behavior highlights that fairly similar tasks can have drastic performance differences with the same architecture.

Limitations In our work we aimed to make a fair comparison against MT, MT-GS, and single-task training, meaning that even though we first tried to obtain the best-performing architecture, no hyper-parameter search was done for every training strategy (which typically yields better results). Furthermore, we only compared two MTL, MTL-GS learning strategies, with three tasks, thus we cannot make a generalized statement regarding the performance of modern MTL paradigms against single-task training. However, we highlight recent criticisms of GS, specifically those claiming that a hyper-parameter search to scaled loss functions to a similar magnitude might provide better performance [34]. This claim could be considered unfair since our results show that GS also benefits from scaled loss functions, hence it should not be used as replacement for hyper-parameter search, but in conjunction.

Another limitation of our work is that we lack a metric to define "similar" tasks which is important to ensure that the subdivision of tasks (into similar tasks) is well fundamented . In future work, we propose to explore a pre-evaluation worflow before training, one in which we keep track of the conflict gradients among all tasks and used this metric to asses and select tasks to jointly train together with gradient surgery.

References

[1] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022.

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

[3] Ivano Lauriola, Alberto Lavelli, and Fabio Aiolli. An introduction to deep learning in natural language processing: Models, techniques, and tools. *Neurocomputing*, 470:443–456, 2022.

[4] Ahmet Iscen, Thomas Bird, Mathilde Caron, Alireza Fathi, and Cordelia Schmid. A memory transformer network for incremental learning. *arXiv preprint arXiv:2210.04485*, 2022.

[5] Rahul Manohar Samant, Mrinal Bachute, Shilpa Gite, and Ketan Kotecha. Framework for deep learning-based language models using multi-task learning in natural language understanding: A systematic literature review and future directions. *IEEE Access*, 2022.

[6] Chulun Zhou, Zhihao Wang, Shaojie He, Haiying Zhang, and Jinsong Su. A novel multi-domain machine reading comprehension model with domain interference mitigation. *Neurocomputing*, 500:791–798, 2022.

[7] Young Jin Kim and Hany Hassan Awadalla. Fastformers: Highly efficient transformer models for natural language understanding. *arXiv preprint arXiv:2010.13382*, 2020.

[8] Samuel Humeau, Kurt Shuster, Marie-Anne Lachaux, and Jason Weston. Real-time inference in multi-sentence tasks with deep pretrained transformers. *arXiv preprint arXiv:1905.01969*, 2019.

[9] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[10] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[11] Hyunjin Choi, Judong Kim, Seongho Joe, and Youngjune Gwon. Evaluation of bert and albert sentence embedding performance on downstream nlp tasks. In *2020 25th International conference on pattern recognition (ICPR)*, pages 5482–5487. IEEE, 2021.

[12] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[13] Akshay Khatri et al. Sarcasm detection in tweets with bert and glove embeddings. *arXiv preprint arXiv:2006.11512*, 2020.

[14] Leilei Gan, Zhiyang Teng, Yue Zhang, Linchao Zhu, Fei Wu, and Yi Yang. Semglove: Semantic co-occurrences for glove from bert. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:2696–2704, 2022.

[15] Amil Merchant, Elahe Rahimtoroghi, Ellie Pavlick, and Ian Tenney. What happens to bert embeddings during fine-tuning? *arXiv preprint arXiv:2004.14448*, 2020.

[16] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169, 2021.

[17] Jane X Wang. Meta-learning in natural and artificial intelligence. *Current Opinion in Behavioral Sciences*, 38:90–95, 2021.

[18] Simon Graham, Quoc Dang Vu, Mostafa Jahanifar, Shan E Ahmed Raza, Fayyaz Minhas, David Snead, and Nasir Rajpoot. One model is all you need: multi-task learning enables simultaneous histology image segmentation and classification. *Medical Image Analysis*, 83:102685, 2023.

[19] Michael Crawshaw. Multi-task learning with deep neural networks: A survey. *arXiv preprint arXiv:2009.09796*, 2020.

[20] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. *Advances in neural information processing systems*, 31, 2018.

[21] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, 2008.

[22] Chris Fifty, Ehsan Amid, Zhe Zhao, Tianhe Yu, Rohan Anil, and Chelsea Finn. Efficiently identifying task groupings for multi-task learning. *Advances in Neural Information Processing Systems*, 34:27503–27516, 2021.

[23] Shijie Chen, Yu Zhang, and Qiang Yang. Multi-task learning in natural language processing: An overview. *arXiv preprint arXiv:2109.09138*, 2021.

[24] Yu Zhang and Qiang Yang. An overview of multi-task learning. *National Science Review*, 5(1):30–43, 2018.

[25] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International conference on machine learning*, pages 794–803. PMLR, 2018.

[26] Xian Li and Hongyu Gong. Robust optimization for multilingual translation with imbalanced data. *Advances in Neural Information Processing Systems*, 34:25086–25099, 2021.

[27] Liyang Liu, Yi Li, Zhanghui Kuang, J Xue, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Towards impartial multi-task learning. iclr, 2021.

[28] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836, 2020.

[29] Lucas Mansilla, Rodrigo Echeveste, Diego H Milone, and Enzo Ferrante. Domain generalization via gradient surgery. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6630–6638, 2021.

[30] Xiaojun Zhou, Yuan Gao, Chaojie Li, and Zhaoke Huang. A multiple gradient descent design for multi-task learning on edge computing: Multi-objective machine learning approach. *IEEE Transactions on Network Science and Engineering*, 9(1):121–133, 2021.

[31] Qiwei Bi, Jian Li, Lifeng Shang, Xin Jiang, Qun Liu, and Hanfang Yang. Mtrec: Multi-task learning over bert for news recommendation. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2663–2669, 2022.

[32] Tao Qi, Fangzhao Wu, Chuhan Wu, Peiru Yang, Yang Yu, Xing Xie, and Yongfeng Huang. Hierec: Hierarchical user interest modeling for personalized news recommendation. *arXiv preprint arXiv:2106.04408*, 2021.

[33] Jaekeol Choi, Euna Jung, Jangwon Suh, and Wonjong Rhee. Improving bi-encoder document ranking models with two rankers and multi-teacher distillation. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2192–2196, 2021.

[34] Derrick Xin, Behrooz Ghorbani, Ankush Garg, Orhan Firat, and Justin Gilmer. Do current multi-task optimization methods in deep learning even help? *arXiv preprint arXiv:2209.11379*, 2022.

[35] Trevor Standley, Amir Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? In *International Conference on Machine Learning*, pages 9120–9132. PMLR, 2020.

[36] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[37] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

[38] Yannis Kalantidis, Mert Bulent Sariyildiz, Noe Pion, Philippe Weinzaepfel, and Diane Larlus. Hard negative mixing for contrastive learning. *Advances in Neural Information Processing Systems*, 33:21798–21809, 2020.

[39] Xiao Wang, Yuhang Huang, Dan Zeng, and Guo-Jun Qi. Caco: Both positive and negative samples are directly learnable via cooperative-adversarial contrastive learning. *arXiv preprint arXiv:2203.14370*, 2022.

[40] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197*, 2019.

[41] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[42] Travis Addair. Duplicate question pair detection with deep learning. *Stanf. Univ. J*, 2017.

[43] Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. * sem 2013 shared task: Semantic textual similarity. In *Second joint conference on lexical and computational semantics (* SEM), volume 1: proceedings of the Main conference and the shared task: semantic textual similarity*, pages 32–43, 2013.

[44] Yile Wang, Leyang Cui, and Yue Zhang. How can bert help lexical semantics tasks? *arXiv preprint arXiv:1911.02929*, 2019.

[45] D Viji and S Revathy. A hybrid approach of weighted fine-tuned bert extraction with deep siamese bi–lstm model for semantic text similarity identification. *Multimedia Tools and Applications*, 81(5):6131–6157, 2022.

[46] Tianxin Wang, Fuzhen Zhuang, Ying Sun, Xiangliang Zhang, Leyu Lin, Feng Xia, Lei He, and Qing He. Adaptively sharing multi-levels of distributed representations in multi-task learning. *Information Sciences*, 591:226–234, 2022.

[47] Dripta S Raychaudhuri, Yumin Suh, Samuel Schulter, Xiang Yu, Masoud Faraki, Amit K Roy-Chowdhury, and Manmohan Chandraker. Controllable dynamic multi-task architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10955–10964, 2022.

[48] Akari Asai, Mohammadreza Salehi, Matthew E Peters, and Hannaneh Hajishirzi. Attempt: Parameter-efficient multi-task tuning via attentional mixtures of soft prompts. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6655–6672, 2022.

A Appendix

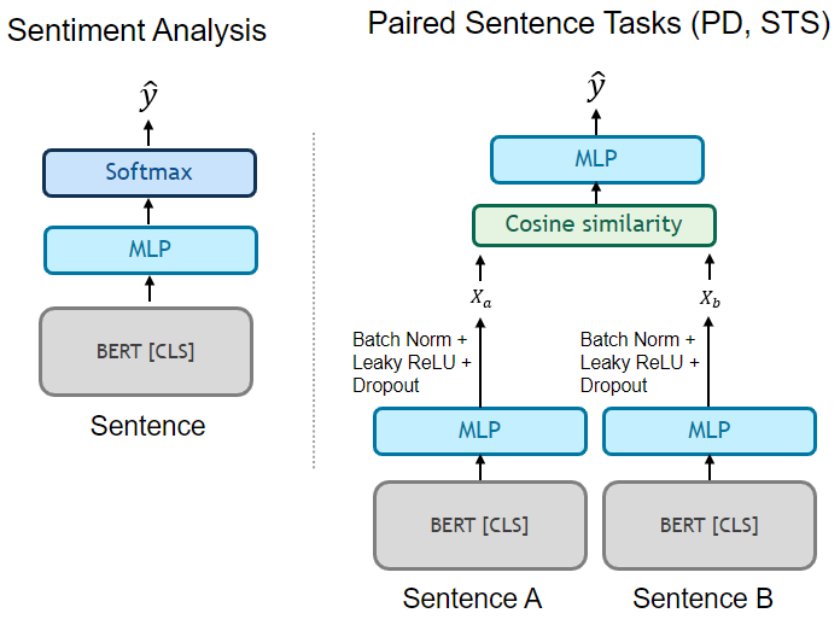

Figure 2: Architecture V_1 for Multi-Task Learning approach.

A.0.1 BERT <CLS> vs BERT average pooling embedding

We performed a toy test with the given sentence:

1. "The dog is dancing on the stage“
2. "The dog is barking on the stage“
3. "The flying car is about to land in Nevada“

Our hypothesis is that even though sentences 1 and 2 are not exactly the same, their embeddings should be closer if compared against sentence 3. For this experiment, we use a pre-trained model of BERT, computed both the <CLS> and mean-attention embeddings, and performed a Singular Value Decomposition (SVD) to visualize them. As seen in figure 3 (a) The mean attention embeddings are able to capture that sentence 1 and 2 are closer than sentence 3. This is not true the <CLS> embeddings as shown in 3 (b)

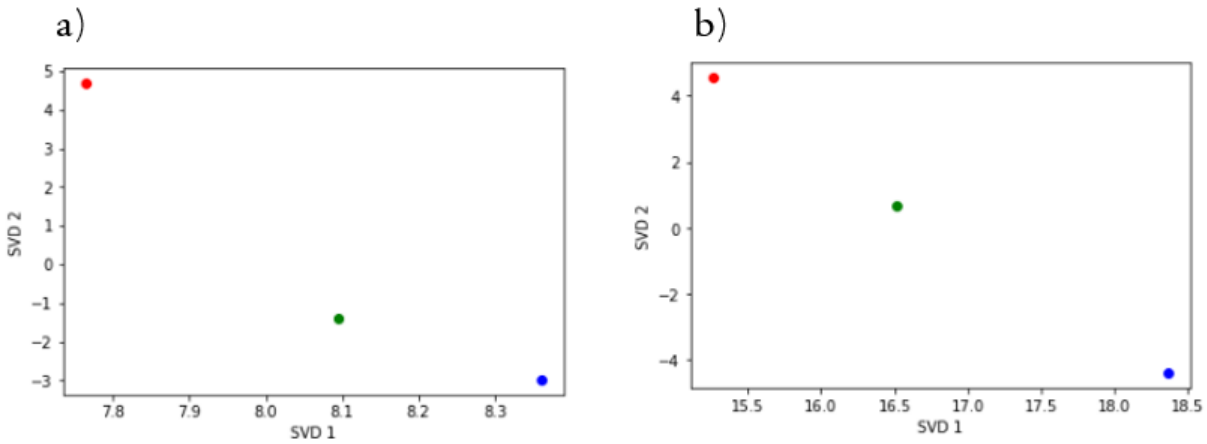

Figure 3: BERT <CLS> (b) vs BERT mean-attention (a) Embeddings: SVD decomposition of BERT embeddings for sentence 1,2, and 3,