(Multi-gate) Mixture of ExBerts

Stanford CS224N Default Project

O Sub Kwon
School of Finance
Nankai University
kwonosub@nankai.edu.cn; kwonosub@stanford.edu

Abstract

The objective of this project is to investigate the effectiveness of Mixture of Experts (MoE) [1] and Multi-gate Mixture of Experts (MMoE) architectures [2] in fine-tuning the BERT model [3] for three downstream tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. Our findings indicate that incorporating either MoE or MMoE architecture only marginally improves the model compared to using a single expert. This is primarily due to the emergence of one dominant expert, which outperforms the others across all tasks and thus receives the majority of the weights, diminishing the utility of having multiple experts. Moreover, our analysis reveals that MMoE outperforms MoE when non-dominant experts exhibit performance better than random guessing in at least one task, enabling them to positively contribute to predictions. Incidentally, our top-performing model secures 10th place on both the test and evaluation set leaderboards.

1 Key Information to include

TA mentor: Chaofei Fan. No external collaborators. No external mentor. Not sharing project.

2 Introduction

Transfer learning [4], a method where a pre-trained language model is applied to different NLP tasks, has proven effective. However, when faced with multiple objectives, simply fine-tuning the model for each task separately may not be optimal as it does not fully exploit the advantages of multi-task learning. Thus, our project aims to use a multi-task learning approach, leveraging pre-trained BERT embeddings, to create a model that performs well across three tasks: sentiment analysis, paraphrase detection, and semantic textual similarity [5]. Specifically, we experiment with a novel combination of the BERT model [3] with the Multi-gate Mixture of Experts (MMoE) architecture [2] to understand to what extent this architecture can improve the predictions by exploiting the differences and similarities among these tasks in a transfer learning setting.

We first establish a baseline using a shared bottom architecture, where inputs from all tasks pass through the same BERT layers before being processed by task-specific towers. Techniques like annealed sampling [6] and SMART regularization [7] are applied to enhance the baseline's performance.

Subsequently, we explore the Multi-gate Mixture of Experts (MMoE) architecture and compare its performance with the baseline. MMoE [2] involves inputs passing through multiple copies of BERT layers (i.e., experts) in parallel, with outputs combined using learned softmax weights from a gating network before being processed by task-specific tower networks. MMoE allows for a different weighting for each task by having a separate gating network for each task. MoE [1], a variant of MMoE, uses the same gating network for all tasks.

Success in this architecture hinges on differentiating each expert so they learn unique parameters and contribute differently to predictions. We experiment with two differentiation methods: Xavier

differentiation (XD) and pretraining differentiation (PD). XD is to simply initialize one expert at the original parameters of the BERT model and initialize all other experts with the same parameters plus Xavier normal noise. PD is to pretrain an individually finetuned BERT network and use the resulting parameters for initialization.

We find that both MMoE and MoE architectures outperform the shared bottom network, albeit marginally. However, regardless of the differentiation method used, both architectures tend to produce a dominant expert that is better at all tasks than other experts and is therefore given most weights, diminishing the utility of having multiple experts. Furthermore, the relative performance between MMoE and MoE depends on the differentiation method employed. Under XD, MoE slightly outperforms MMoE, while under PD, MMoE significantly outperforms MoE. This is because under PD, non-dominant experts can still contribute positively to predictions as they perform better than random guessing in at least one task. MMoE's ability to use different softmax functions for each task allows it to leverage the knowledge of non-dominant experts, whereas MoE tends to assign most weight to the dominant expert regardless of the task, thus unable to take advantage of the knowledge of other experts.

3 Related Work

The pretrain-finetune paradigm in NLP has roots tracing back to Howard and Ruder's work [8] on fine-tuning general-purpose language models, primarily consisting of stacked layers of LSTMs. Recently, this paradigm has been dominated by transformer-based large language models like GPT-2 [9], BERT [3], and their extensions. However, typical pretrain-finetune approaches require a new set of weights for each task. Simply training on multiple tasks sequentially can lead to catastrophic forgetting [10], where weights from the previous task are forgotten, diminishing the old task's performance.

Multi-task learning [11] presents a compelling solution by allowing models to tackle multiple tasks simultaneously, thereby improving both efficiency and quality. One popular approach is the shared-bottom model [12], where lower layers are shared across tasks to prevent overfitting. However, this method may suffer from suboptimal performance due to differences between tasks. Previous studies have proposed various strategies to address these task differences in multi-task learning settings. For instance, Stickland and Murray [6] recommend annealed sampling when datasets vary in size across tasks. Yu et al. [13] introduce Gradient Surgery, a technique that manages gradient conflicts between tasks by removing the conflicting part of the gradients. Kendall et al. [14] propose a method that balances multiple loss functions based on the homoscedastic uncertainty of each task. In contrast, MMoE [2] is an architecture that introduces multiple bottom networks and explicitly models task relationships through multiple gating networks. This approach enables tasks to extensively share layers and parameters while remaining robust to task differences.

More broadly, the MMoE architecture represents a novel approach to ensemble methods, known to enhance model performance in deep neural networks. It extends the MoE architecture [1], which consists of multiple subnetworks (i.e., experts) selected based on softmax weights produced by a gating network. This not only reduces computational costs but also improves modeling capability. However, ensuring each expert learns different parameters, especially in a transfer learning setting where they leverage parameters of the same large language model, poses a challenge. Our project addresses this by experimenting with different methods of differentiating the expert networks.

4 Approach

4.1 Baseline: Shared Bottom Network

Since our goal is to assess how well the MMoE architecture performs in a transfer learning scenario using BERT, a logical comparison would involve a shared bottom model with similar hyperparameters.

A shared bottom network consists of one bottom network, represented as function f, that is shared by all tasks, and K tower networks h_k for each of the K tasks, where $k = 1, ..., K$. Given input x from task k, the output of the model can then be simply represented as

$$y_k = h_k(f(x)).$$

To give this architecture the best chance of success, we use **annealed sampling** [6] to address the discrepancies in data volumes among the three tasks. Because, as will be mentioned later, there are substantial variations in dataset sizes, simply training the model by cycling through them in a fixed order would lead to overfitting on the smaller datasets and underfitting on the larger one. Annealed sampling balances the concern of using all datasets to their full extent and training on all datasets evenly. In this approach, each epoch comprises m training steps. At each training step, we select a batch of examples from task i with probability $p_i \propto N_i^\alpha$, where N_i denotes the dataset size and

$$\alpha = 1 - 0.8\frac{e-1}{E-1},$$

with e being the current epoch and E the total number of epochs. Essentially, this method requires us to train more evenly across all tasks, particularly towards the end of training. We implemented this method ourselves.

We also experimented with **SMART regularization** [7] to further boost the performance of the shared-bottom network. However, somewhat surprisingly, it did not improve the performance of the model. See Appendix for the description of SMART regularization and the experimental results.

4.2 Multi-gate Mixture of Experts (MMoE) Architecture

The main framework of this project is a novel combination of the BERT model [3] with the Multi-gate Mixture of Experts (MMoE) architecture [2], comprising the following four main components (see Figure 1): embedding layer, experts, gating networks, tower network.

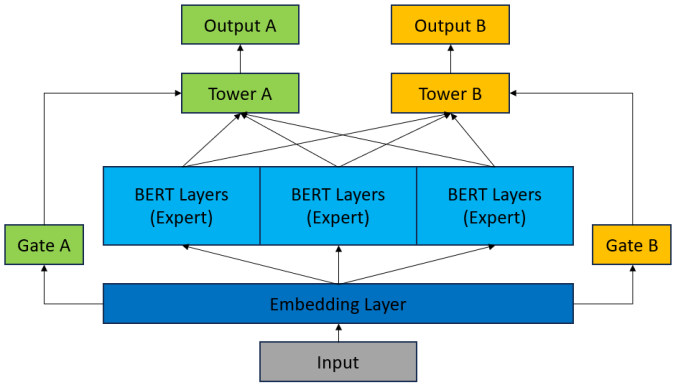


Figure 1: MMoE Architecture

4.2.1 Embedding Layer

Similar to the original BERT model, we have an embedding layer that takes in the sentences and output their embedding. The embedding layer is shared by all experts and gating networks. For notational convenience, we use x to denote the output of this embedding layer instead of the original input.

4.2.2 Experts

We introduce S copies of the original BERT modules, referred to as "experts," which are shared among all tasks, where S is not necessarily equal to the number of tasks K. Each expert s, represented as function f_s, receives input x from the embedding layer and produces expert embeddings $f_s(x)$.

4.2.3 Gating Networks

We also introduce a dedicated "gating network" g_k for each task k. Given input x from task k, the corresponding gating network produce softmax weights $g_k(x)$, whose dimension is equal to the

number of experts S. These weights serve to mix the expert embeddings produced by the experts embeddings $f_s(x)$ to produced mixed embedding

$$z = \sum_{s}^{S} (g_k(x))_s f_s(x)$$

The architecture of each gating network we use is as follows: first, the embedding x goes through a BERT layer (initialized at the parameters of the first BERT layer) and then a feed forward neural network with one hidden layer of same dimension as the word embedding and ReLU activation.

4.2.4 Tower Network

Finally, the mixed embedding z is sent to a task-specific "tower network" h_k that produces the final output

$$y_k = h_k(z)$$

For each tower network, we use a feed forward neural network with one hidden layer of same dimension as the word embedding, ReLU activation, and a dropout probability of 0.3.

4.2.5 Expert Differentiation

In order for each expert to contribute differently to the predictions, it is crucial that each expert learn different parameters. We experiment with two natural ways to initialize each expert differently. One technique, which we call Xavier differentiation (XD), is to initialize one expert at the original BERT parameters and to initialize all other experts at the same parameters plus a Xavier normal random noise. In this way, we can initialize each expert differently although the experts with the random noise are arguably inferior copies of the one without the random noise.

Another technique, which we call pretraining differentiation (PD), is to "pretrain" a network that consists of an embedding layer, an expert network, and a tower network using only the training data of a single task and use the resulting parameters of the expert network and the tower network for initialization. To use this technique, it must be that the number of experts equals to the number of tasks. For pretraining, we only use 3 epochs because our goal is only to have different initialization for different expert and aggressive pretraining may result in overfitting.

It is worth mentioning that, unlike the expert networks, we do not have to use any differentiation technique for the gating networks because each gating network is trained on a different training set.

4.2.6 MoE Architecture

Note that this architecture allows a special case, called Mixture of Experts (MoE) architecture [1], where we only have one gating network g for all tasks. That is, regardless of the task k, we have mixed embedding

$$z = \sum_{s}^{S} (g(x))_s f_s(x).$$

This special case is useful in that it helps us isolate the effect of having multiple gating network from that of having multiple experts.

5 Experiments

5.1 Data

We utilize three datasets: the Stanford Sentiment Treebank (SST), Quora (QQP), and SemEval STS Benchmark (STS).

- The SST dataset consists of single-sentence movie reviews with the label of negative, somewhat negative, neutral, somewhat positive, or positive. We have 8,544 training examples, 1,101 evaluation examples, and 2,210 test examples in this dataset.

- The Quora dataset comprises pairs of questions labeled with binary values indicating whether particular instances are paraphrases of one another. We have 282,010 training examples, 40,429 evaluation examples, and 80,859 test examples in this dataset.
- The STS dataset consists of sentence pairs with integer scores ranging from 0 to 5 that indicate to what extent the sentences are simialar to each other. We have We 6,040 training examples, 863 evaluation examples, and 1,725 test examples in this dataset.

5.2 Evaluation method

We evaluate the models' performance by aggregating three metrics: the accuracy on the SST task (a_{SST}), the accuracy on the QQP task (a_{QQP}), and the Pearson correlation coefficient for the STS task (ρ_{STS}). We aggregate these metrics to obtain an overall score through the following equation:

$$\text{Score} = \frac{1}{3}a_{SST} + \frac{1}{3}a_{QQP} + \frac{1}{6}(\rho_{STS} + 1).$$

5.3 Experimental details

Unless noted otherwise, all models were trained with a learning rate of 1e-5 for 10 epochs (2000 training steps per epoch) using annealed sampling. Due to the constraint of GPU memory, we used gradient accumulation for 2 batches of size 8 in each training step. All models used the AdamW optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon =$ 1e-6, and a weight decay of 1e-4.

5.4 Results

Table 1 displays the combined outcomes of our models on the evaluation set. It is clear that all MoE and MMoE models surpass the baseline shared-bottom network, as expected. However, the performance contrast between MoE and MMoE models depends on how the experts are initialized. Recall that, in the case of XD, one expert begins with the original BERT parameters, while others have these parameters plus random noise. In the case of PD, each expert starts with parameters slightly fine-tuned for each task. We observe that under XD, the MoE architecture slightly outperforms the MMoE architecture, whereas the reverse is true under PD. We will explore this further in the next section.

Table 1: Evaluation Set Results

Model	Dev SST	Dev QPP	Dev STS	Score
Individual Finetune	0.525	**0.905**	0.840	
Shared Bottom	0.493	0.886	0.862	0.770
XD MoE	0.510	0.886	0.872	0.777
XD MMoE	0.503	0.885	**0.876**	0.775
PD MoE	0.497	0.885	0.862	0.771
PD MMoE	**0.526**	0.887	0.860	**0.781**
Ensemble	**0.536**	0.895	**0.883**	**0.791**

Additionally, (M)MoE models outperform the individually fine-tuned models (utilized for pretraining) solely on the STS task. This indicates limited potential for transfer learning across these tasks.

Finally, it is noteworthy that the top-performing model is the ensemble of all five multi-task models listed in the table, comprising one shared bottom network and four (M)MoE models. The ensemble significantly outperforms all the (M)MoE models, despite each (M)MoE model being an ensemble itself. The rationale behind this will be elucidated in the next section.

Table 2: Test Set Results

Model	Dev SST	Dev QPP	Dev STS	Score
Ensemble	0.534	0.895	0.883	0.790

Table 2 presents the test set results of the ensemble model. As anticipated, the ensemble demonstrates consistent performance on the test set comparable to that on the evaluation set. As of the current writing, the ensemble model's performance ranks 10th on the test leaderboard.

6 Analysis

6.1 Expert Specialization

To understand why the relative performance of MoE and MMoE models depends on how the experts are initialized, we first examine the extent to which each expert of an (M)MoE architecture specializes in each task. For each expert of an (M)MoE model, we create a shared bottom network comprising the embedding layer, the tower networks, and that specific expert. This procedure yields three shared bottom networks for each (M)MoE model. Subsequently, we randomly sample 500 examples from the evaluation set of each dataset and assess the constructed shared bottom networks on these datasets. Table 3 provides a summary of their performance.

Table 3: Expert Specialization

(M)MoE Model	Expert Index	Dev SST	Dev QPP	Dev STS
XD MoE	1	0.536	0.860	0.849
	2	0.120	0.364	0.083
	3	0.144	0.364	-0.035
XD MMoE	1	0.512	0.878	0.841
	2	0.144	0.364	-0.053
	3	0.144	0.364	0.067
PD MoE	1	0.518	0.884	0.845
	2	0.278	0.870	-0.423
	3	0.216	0.370	0.753
PD MMoE	1	0.528	0.886	0.837
	2	0.120	0.636	-0.304
	3	0.216	0.364	0.142

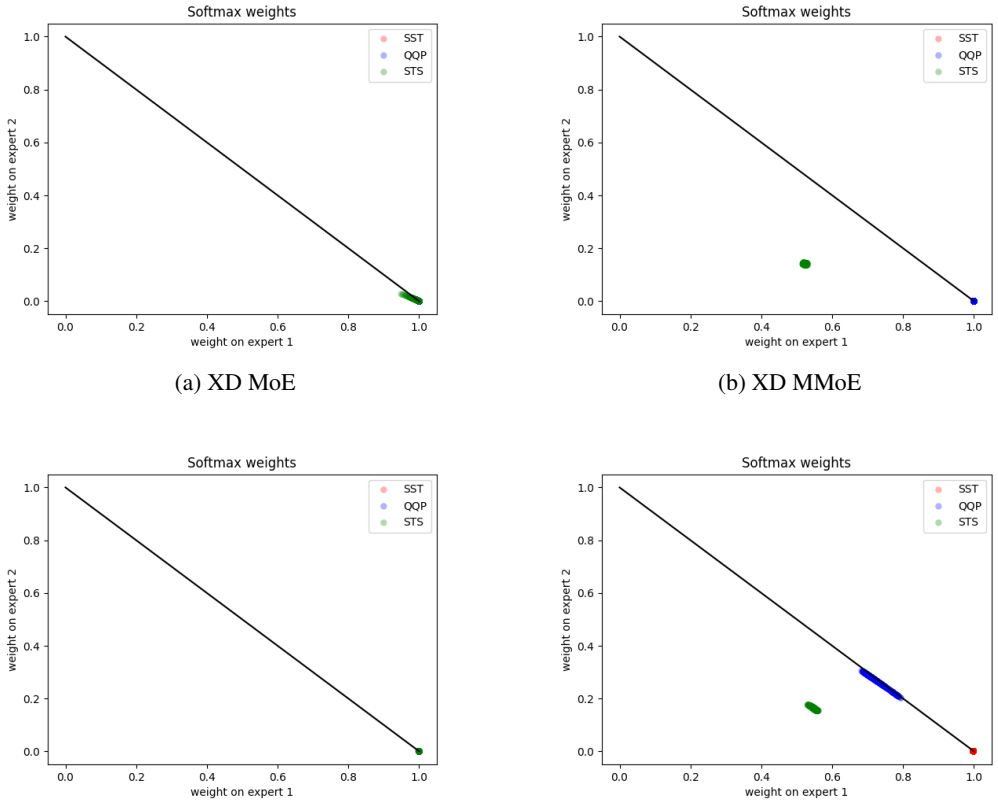
Surprisingly, our analysis unveils a consistent pattern across each (M)MoE architecture: a dominant expert consistently outperforms all others across every task. In the XD case, this dominant expert is the one initialized using the original BERT model parameters. Conversely, in the PD case, the dominant expert emerges from that pretrained using the SST dataset. While the precise reason for this dominance remains unclear, we hypothesize that it may be related to the fact that the task of sentiment analysis is markedly different from other two tasks.

Furthermore, the non-dominant experts initialized via XD perform even worse than random guessing across all tasks. This contrasts with those initialized via PD, which generally exhibit performance surpassing random guessing in the tasks they were pretrained on.

6.2 Expert Weights

Figure 2 illustrates how weights are distributed among experts in different networks for various tasks, using the same datasets as before. Each plot showcases the weight assigned to the dominant expert (expert 1) on the horizontal axis and the weight assigned to expert 2 on the vertical axis. The weight on expert 3 is computed as 1 minus the sum of the weights on experts 1 and 2. For instance, a point at (1, 0) signifies that, for a specific example, expert 1 carries all the weight while the others carry none.

Our analysis reveals that across all tasks, networks predominantly favor the dominant expert. In the MoE architecture, where tasks share a single gating network, all probability mass concentrates on the dominant expert, regardless of initialization. Conversely, the MMoE architecture, with separate gating networks per task, allows flexibility in allocating weights to non-dominant experts if beneficial. However, under XD, the poor performance of non-dominant experts means assigning them weights could harm overall performance. Consequently, for SST and QQP, all weights are directed to the dominant expert.



(a) XD MoE

(b) XD MMoE

(c) PD MoE

(d) PD MMoE

Figure 2: Softmax Weights by Gating Networks

On the contrary, in the PD MMoE model, while all weights still favor expert 1 for SST, other tasks exhibit some weight distribution. For QQP, approximately 20% of the weight is allocated to expert 2, pretrained on this task. Similarly, for STS, around 20% of the weight is allocated to expert 3, pretrained on this task. Thus, the PD MMoE model achieves a more balanced weight distribution because non-dominant experts contribute positively to predictions. Hence, the superior performance of the non-dominant experts under PD, coupled with multiple gating networks, elucidates why MMoE outperforms MoE under PD.

The behavior of these (M)MoE models also elucidates why, in the previous section, the ensemble model significantly outperforms all the (M)MoE networks. Although an (M)MoE network is theoretically an ensemble model, it does not effectively operate as one because all weights are assigned to a single expert.

7 Conclusion

This project explores the potential of a Multi-gate Mixture of Experts (MMoE) architecture to enhance the fine-tuning process of a large language model within a multitask learning framework. Our findings indicate that while MMoE architecture does yield improvements over a shared bottom network, these enhancements are only marginal. This minimal improvement can be attributed to the consistent emergence of a dominant expert across tasks, regardless of how the experts are initialized. Consequently, the gating network tends to allocate all weights to this dominant expert.

Although our MMoE models did not achieve significantly superior performance compared to a shared bottom model, our study sheds light on the challenges of effectively implementing MMoE architecture using multiple copies of the same large language model. It also suggests avenues for enhancing this architecture. We hypothesize that better performance could be attained by regulating

the learning speed of the dominant expert and expediting the learning of the other non-dominant experts to achieve comparable performance.

At a broader level, this concept bears resemblance to the literature on Generative Adversarial Networks (GANs), which discusses the importance of balancing the learning speed of the discriminator and generator to prevent the latter's learning from stalling [15, 16]. Our model exhibits a similar dynamic: if all weights are assigned to the dominant expert too early, the learning of the non-dominant expert may stagnate. In this sense, the gating network and the non-dominant experts exhibit a similar adversarial relationship to the discriminator and generator in a GAN. Consequently, for future endeavors, we intend to explore techniques from the GAN literature and evaluate their effectiveness in enhancing our model's performance.

8 Ethics Statement

A large language model that is fine-tuned for sentiment analysis, paraphrase detection, and semantic textual similarity could be risky for society. One concern is that it might pose harm to non-native English speakers. Since the model is fine-tuned only with English data, it might not accurately analyze sentiments in non-English texts. This bias could be harmful, especially if important decisions, such as content censorship, rely on the model's results. To reduce this risk, we can fine-tune the model using a wider range of texts, including those in other languages.

Another concern is about privacy. Because the model is outstanding at measuring textual similarities, it could be misused by malicious actors, such as authoritarian governments, to identify individuals, such as human rights activists, by analyzing anonymous social media content. To address this risk, we can augment the fine-tuning data so that the model's predictions are not influenced by specific writing styles.

References

[1] RA Jacobs. Adaptive mixture of local experts. *Neural Computation*, 3:337–345, 1993.

[2] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1930–1939, 2018.

[3] 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.

[4] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3:1–40, 2016.

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

[6] Asa Cooper Stickland and Iain Murray. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *International Conference on Machine Learning*, pages 5986–5995. PMLR, 2019.

[7] Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. *arXiv preprint arXiv:1911.03437*, 2019.

[8] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[9] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[10] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[11] Shijie Chen, Yu Zhang, and Qiang Yang. Multi-task learning in natural language processing: An overview. *ACM Computing Surveys*, 2021.

[12] Rich Caruna. Multitask learning: A knowledge-based source of inductive bias. In *Machine learning: Proceedings of the tenth international conference*, pages 41–48, 1993.

[13] 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.

[14] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7482–7491, 2018.

[15] Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. *Advances in neural information processing systems*, 28, 2015.

[16] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.

A Appendix

A.1 SMART Regularization: Description and Experimental Results

When establishing our benchline, to prevent overfitting during fine-tuning, we consider **SMART regularization** [7], which introduces a smoothness-inducing regularizer to the loss function:

$$\mathcal{R}_s(\theta) = \frac{1}{n}\sum_{i=1}^{n} \max_{||\tilde{x}_i - x_i||_p \leq \varepsilon} l_s(f(\tilde{x}_i;\theta), f(x_i;\theta))$$

where x_i denotes the embedding of the input sentences obtained from the first embedding layer of the language model, $f(\cdot;\theta)$ denotes the model, ε is a tuning parameter, and l_s is chosen as the symmetric KL-divergence loss for classification tasks and mean squared error loss for regression tasks. The optimization is performed using Bregman proximal point optimization. Intuitively, this regularization method penalizes the sensitivity of the model to a small change in the embedding of the input sentences and thereby inducing a smoother prediction in x_i. We implemented this method using the smart-pytorch package.

We find that, as can be seen from Table 4, SMART regularization does not improve the performance of the baseline model.

Table 4: SMART Regularization Results

SMART Weight	Dev SST	Dev QPP	Dev STS	Score
0	0.528	0.883	0.873	0.782
1	0.521	0.872	0.873	0.776
3	0.513	0.850	0.865	0.765
5	0.479	0.831	0.854	0.746