# Extending minBERT in a Multi-Task Setting

Stanford CS224N Default Project under TA Mentorship from **Jingwen Wu**

**Jean-Philippe Lemay** jplemay@stanford.edu
**Veljko Skarich** vskarich@stanford.edu
**Joe Wang** jwang02@stanford.edu

## Abstract

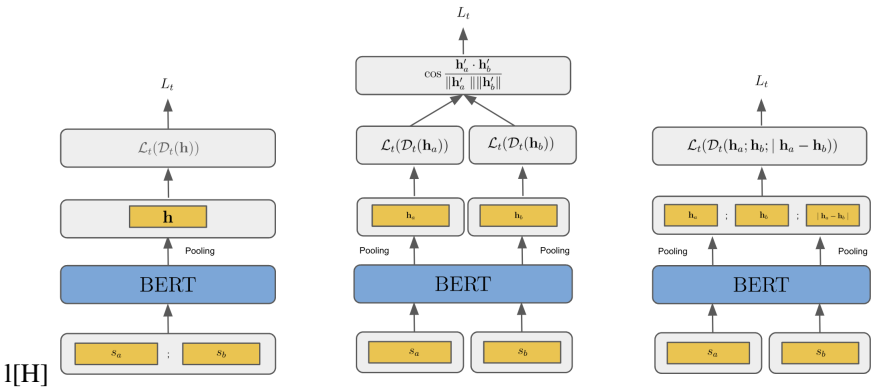
Our project focuses on extending the BERT model Sun et al. (2019) to optimize its performance across a NLU multi-task challenge involving (a) Sentiment Analysis (SA), (b) Paraphrase Detection (PD) and (c) Semantic Textual Similarity (STS). After having fine-tuned minBERT independently for each downstream tasks, we further explore improvements along these key model design dimensions: 1) We implement two alternate sentence-level embedding approaches inspired by the Sentence-BERT (SBERT) model: one with Cosine-Similarity and one with Concatenation, 2) we seek to augment BERT's multi-task representation capacity by adding a parallel Projected Attention Layer (PAL) structure and 3) we investigate key aspects of multi-task learning design choices including task sampling methodologies, optimizer choice as well as Gradient Surgery (GS). We find that both PAL and GS independently provide a performance boost over our multi-task baseline of approximately 1.0%. Within each model, annealed sampling generally perform better than other approaches in a multi-task setting and, finally, that combining PAL and GS introduces cross-purpose conflict leading to sub-optimal multi-task performance.

## 1 Introduction

Our project consisted of two phases: 1) We first completed the implementation and fine-tuning of BERT introduced in Devlin et al. (2018) and trained its **base** version independently on the three downstream tasks establishing single-task baselines and 2) implementing key extensions across different design axes focused on achieving top performance in this specific multi-task context. In our second phase, we explore (a) two different usage of the BERT model for sentence-pair comparison tasks (PD and STS) using the Sentence-BERT siamese structure described in Reimers and Gurevych (2019), (b) an augmented representation of the BERT model adding a task-specific multi-head self-attention block across each BERT layers along Stickland and Murray (2019) named PAL, (c) different task sampling approaches in the context of multi-task training, and (d) the improvement of the gradient computation within our multi-task learning implementing gradient surgery described in Yu et al. (2020). Furthermore, we experimented changing the aggregation methodology for the sentence representation in the BERT model as well as the learning optimization algorithm for our multi-task baseline. We describe in details these extensions in the following section as well as presenting our baselines.

## 2 Related Work

Sentence-BERT (SBERT) introduced by Reimers and Gurevych (2019) builds on various advances in NLP, particularly leveraging the BERT architecture for sentence embeddings. SBERT introduces a Siamese and triplet network structure, enabling efficient computation of sentence similarities. This method significantly reduces the computational complexity compared to traditional BERT models. By focusing on sentence-level tasks such as semantic textual similarity and information retrieval, SBERT provides robust and meaningful sentence embeddings. Stickland and Murray (2019) explore multi-task learning by sharing a single BERT model with minimal task-specific parameters. Traditional multi-task learning approaches often require separate models for each task,




l[H]

(a) Base BERT (b) SBERT with Cosine-Similarity (c) SBERT with Concatenation

Figure 1: Illustration of the different sentence embedding architectures. Of note, all hidden vectors $\mathbf{h}$ are assumed to be indexed by the specific single-task $t$ at hand which we avoid for lighter notation.

leading to high parameter usage. The study introduces Projected Attention Layers (PALs) to reduce parameters while maintaining performance. PALs allow for efficient adaptation across multiple tasks, achieving state-of-the-art results on the GLUE benchmark with approximately seven times fewer parameters, particularly excelling in the Recognizing Textual Entailment task. Gradient surgery ( Yu et al. (2020)) addresses the challenges of conflicting gradients in multi-task learning scenarios. Traditional approaches often suffer from gradient conflicts, where beneficial updates for one task may hinder another. GS mitigates these conflicts by projecting each task's gradient onto the normal plane of the gradients from other tasks, ensuring more harmonious updates and improving overall performance. This technique leads to more efficient training and better handling of multiple tasks simultaneously. Our work focuses on implementing these key NLP advancements in multi-task setting and understand their interactions, i.e. are they improving performance in isolation and by combining them together. We also want to experiment the practical challenges of combining different dataset sizes and how these extensions interact together in that context.

## 3 Approach

In this section, we present in details each of the extensions we explored across sentence-level embeddings, multi-ask representation using the PAL representation augmentation and learning algorithm adaptations.

### 3.1 Sentence-Level Embeddings

This exploration is relevant for sentence pairs $(s_a, s_b)$ based tasks $t$, i.e. $t \in \{\text{PD}, \text{STS}\}$. In the classic implementation of dual sentence tasks within the BERT model, the token embeddings of $s_a$ and $s_b$ are concatenated together to create one sequence representation that is fed through the BERT architecture. The BERT model outputs a unique hidden state vectors representation $\mathbf{h} \in \mathbb{R}^{2d_L \times d}$ for the sentence pair on which we apply a task-specific $t$ dropout $\mathcal{D}_t$ and linear $\mathcal{L}_t$ layer before applying the classifier loss function $L_t$ (here $d_L$ is the maximum sequence length and $d$ is the size of the hidden state vector). The full pipeline of steps for the classic BERT architecture is illustrated above in Figure 1 (a).

Alternatively, the SBERT architecture proposes to independently extract BERT hidden states for each sentence sequence using the same BERT network thereby forcing the model to learn sentence-level representations. These sentence-level representations are then used to compare sentences between them and, as such, a distance metric is utilized to perform sentence-pair comparison tasks. We explore two comparison designs. The first one uses cosine-similarity between the two sentence-level representations (Figure 1 (b)) and the second one concatenates two sentence-level representation together with the element-wise absolute difference and apply a dropout-linear transformation on this representation towards the classification task (Figure 1 (c)).

The BERT model outputs hidden vector embeddings for each token of the sequence fed into it. However, only one hidden state vector is passed upward in the task-specific classification pipeline to represent the full sequence. In Figure 1, this operation is illustrated by the *pooling* label. The usual pooling method is to choose the hidden state embedding of the `[CLS]` token. Alternatively, Reimers and Gurevych (2019) proposes to explore other pooling methods such as taking the element-wise mean or maximum amongst all the sequence token hidden state vectors. In our experiments, we are testing SBERT with the mean pooling method.

### 3.2 Multi-Task Representation: Projected Attention Layer (PAL)

One of our key extension is our implementation of the PAL augmented BERT architecture. Before introducing the PAL extension, Figure 2 illustrate our baseline BERT model architecture in a multi-task setting. We follow the same base BERT model structure as in Stickland and Murray (2019) with 12 BERT layers of 12 head self-attention blocks feeding through each other. The question we are now asking is whether or not this model's representation capacity is sufficient to tackle all 3 downstream tasks jointly. The approach we took to explore this question was to implement an augmented multi-task representation of the BERT model introduced in Stickland and Murray (2019) where the architecture of minBERT is augmented by the addition of a parallel task-specific PAL. Specifically, Stickland and Murray (2019) proposes to modify each BERT layer as follows: $\text{BL}_t^{\text{PAL}}(\mathbf{h}_l) = \text{LN}(\underbrace{\mathbf{h}'_l + \text{S}(\mathbf{h}_l)}_{=\text{BL}_l} \oplus \underbrace{\text{TS}_t(\mathbf{h}_l)}_{\text{PAL}})$, where $\text{TS}_t(\mathbf{h}_l)$ is the additional task-specific $t \in \{\text{SA}, \text{PD}, \text{STS}\}$ parallel PAL term for each BERT layer $l$, $\text{BL}_l$ is the standard BERT layer comprised of the multi-head self-attention block $\text{S}(\mathbf{h}_l)$ and its residual connection $\mathbf{h}'_l$, and LN is the layer norm operator. Here, $\mathbf{h}'_l = \text{LN}(\mathbf{h}_l + \text{MH}(\mathbf{h}_l))$ where $\text{MH}(\mathbf{h}_l)$ is the multi-head attention block. Furthermore, $\text{S}(\mathbf{h}_l) = \text{FFN}(\mathbf{h}')$ where the feed-forward network here involves two linear transformations and a GELU ( Hendrycks and Gimpel (2016)) activation transformation. We refer the reader to the original transformer paper Vaswani et al. (2017) for a description of the multi-head self-attention structure and to Devlin et al. (2018) for the original BERT model application. Task-specific attention is defined as $\text{TS}_t(\mathbf{h}_l) = V_t^D \cdot g_t(V_t^E \cdot \mathbf{h}_l)$, where $V_t^E$ and $V_t^D$ are the projection (encoding and decoding respectively) matrices and the function $g_t(\cdot)$ is itself a multi-head self-attention block. We index the matrices $V_t^E$, $V_t^D$ and $g_t(\cdot)$ with the subscript $t$ to highlight that these parameters are defined for each task but are shared across the BERT layers. The architecture is illustrated in Figure 3. The operator $\oplus$ is used to map the visuals of Figure 3 to highlight the additive nature of the extension (but it is a regular element-wise vector addition). Furthermore, the sequential BERT-PAL layers are chained up in the forward pass where $\mathbf{h}_{l+1} = \text{BL}_t^{\text{PAL}}(\mathbf{h}_l)$. A few important observations:

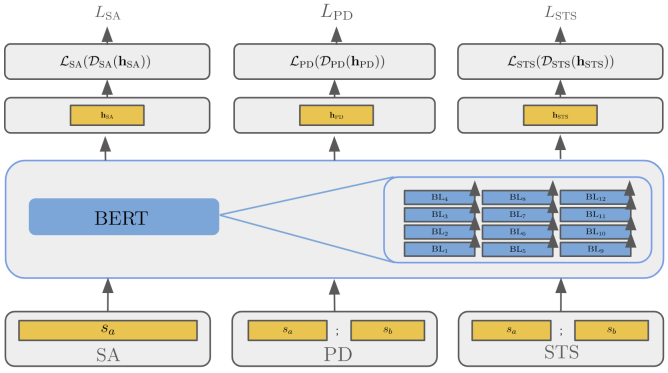



Figure 2: Multi-Task BERT Architecture.

- $g_t$ is a 12-head self-attention transformation which corresponds to the number of heads in the self-attention layer of the base BERT model;
- The hidden dimension size of $g_t$ is chosen to be $d_p = 204$ as suggested in Stickland and Murray (2019). Hence, the encoding matrices $V_t^E$ have dimension $d \times d_p$ and the decoding matrices $V_t^E$ have dimension $d_s \times d$ where $d = 768$ (i.e. the size of the BERT model embeddings);
- We note that the BERT+PAL structure adds additional representation capability by adding parameters $V_t^E$, $V_t^D$ and $g_t$ for each task $t$ and represent 5.46M additional parameters to be learned (from 109.49M for base BERT to 114.95M for base BERT + PAL). In the context of multi-task training, the task sampling method (which we cover in the next section) alternates

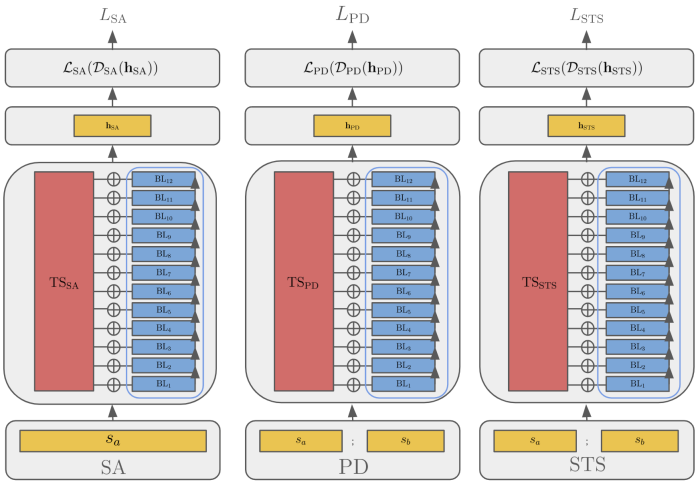



Figure 3: BERT and PAL Network Architecture for Multi-Task Training.

between each task at each gradient update step and the task-specific self-attention block $\text{TS}_t$ enables additional capacity in addition to the task-specific top layers $\mathcal{L}_t$;

- Finally, we note that removing the PAL blocks of Figure 3 yields a model architecture that is equivalent to the one illustrated in Figure 1 (a) where the blue BERT block corresponds to the 12 layers BERT blue block appearing in Figure 3. One can observe the same fact by setting $\text{TS}_t(\mathbf{h}_l) = 0$ in the equation for $\text{BL}_t^{\text{PAL}}(\mathbf{h}_l)$.

### 3.3 Multi-Task Learning

**Task Sampling** One of the key challenges in our multi-task context is the significant difference in the size of the datasets between tasks. This means that during multi-task training, the allocation of passes through each task during each step & epoch is a key design feature. Various approaches have been proposed and in our analysis we focus on the following:

- **Round Robin (RR)** At each epoch, the number of steps is determined in such a way that all datasets are exhausted within each epoch. This approach is computationally intensive and will be heavily weighted towards the task with the largest relative dataset size (in our case, this is Quora for PD). Each epoch will be extensively trained and the task weight will be constant across epochs.
- **Proportional (Prop) & Uniform (U)** In these cases, we endogenously fix the number of steps per epoch and at each time step, we sample the task that will be used for the gradient update with a categorical distribution with probabilities proportional (Prop) to their relative dataset size (i.e. $p_t \propto N_t$) or picked from a uniform distribution (i.e. $p_t = \frac{1}{3}$ the probability of picking task $t$). This approach also induces constant task weights across epochs and will also mean that depending on the number of steps utilized, the smaller datasets will be covered potentially many times.
- **Annealed Sampling (SA)** The AS method proposed in Stickland and Murray (2019) determines a categorical distribution for sampling across tasks at each time step that is proportional to a power of the training dataset size, i.e. $p_t \propto N_t^{\gamma}$ where $\gamma = 1 - 0.8\frac{e-1}{E-1}$ for task $t$ at epoch $e$ of a total number of epochs $E$. This approach skews the sampling distribution to the bigger dataset (Quora in our case) early in the training and gradually becomes more evenly distributed towards the end of the training as we are more concerned with the interference between tasks.
- **RR & AS** We also tested a combination of RR and SA for our gradient surgery implementation where we are first ensuring the first three example in the minibatch dealt with each task and the remainder of the examples were determined with SA.

**Gradient Surgery (GS)** One of the main difficulties with multi-task learning is that the gradients for different tasks often conflict. This is because the global minima for different tasks might not be

the same, and they might have different optimal hidden state representations. But ideally, the goal would be to ensure gradient updates don't conflict. Gradient surgery ( Yu et al. (2020)) addresses this by projecting a task's gradient onto the normal plane of any other task's conflicting gradient. This technique corrects conflicting gradients in two areas: magnitude differences and conflicting directions. Additionally, gradient surgery helps traverse areas of the loss surface with pathological curvature by mitigating oscillations from inter-task conflicts.

Within each gradient update step, GS applies a transformation to the gradients of the **shared** parameters of the network architecture as follows: Let $\mathbf{g}_t$ be the gradient of the task $t$ and $\mathbf{g}_s$ be the gradient of the task $s$. The algorithm proceeds as follows:

- If the cosine-similarity between $\mathbf{g}_t$ and $\mathbf{g}_s$ is negative indicating conflicting gradients (i.e., if the dot product $\mathbf{g}_t \cdot \mathbf{g}_s < 0$), we replace $\mathbf{g}_t$ by its projection onto the normal plane of $\mathbf{g}_s$, i.e., $\mathbf{g}_t = \mathbf{g}_t - \frac{\mathbf{g}_t \cdot \mathbf{g}_s}{||\mathbf{g}_s||^2}\mathbf{g}_s$;
- Otherwise (if no conflicting gradients), keep $\mathbf{g}_t$ intact.

This procedure is applied to all tasks other than task $t$ in random order within the current step's batch, and the transformed gradients are summed by task. For BERT with PAL, task-specific layers are excluded from this transformation, so only **shared** parameters are affected by GS. We used the implementation from Tseng (2020). The challenge was deciding the dataset proportions for each gradient update, as the Quora dataset is much larger than the others. Proper gradient projection required gradients from at least two, ideally three, datasets. We accumulated gradients for 8 batches before projection: one batch from each dataset, with the remaining 5 batches using annealed sampling (as described in the Task Sampling section, our RR + SA case).

### 3.4 Baselines

The first phase of our project completed the implementation of BERT and enabled single-task training across the three downstream tasks of our context (see Table 3 in the Appendix for single-task baselines; also shown in the top left corner of Table 2). Our multi-task baseline was obtained by running a round robin task sampling method and adding the task-specific loss of the sampled task of each gradient update step (look for Multi-Task Baseline in Table 2). We define$\alpha_{\text{SA}}$ as the classification accuracy for SA, $\alpha_{\text{PD}}$ as the classification accuracy for PD, $\rho_{\text{STS}}$ as the Pearson correlation coefficient used for STS accuracy and $\psi = \frac{\alpha_{\text{SA}}+\alpha_{\text{PD}}+0.5+0.5\times\rho_{\text{STS}}}{3}$ is the aggregate multi-task accuracy[1].

## 4 Experiments

### 4.1 Data, Evaluation Method and Experimental Details

The data used for each task is described in the project handout, i.e. SST for SA, Quora (QQP) for PD and SemEval STS Benchmark for STS. Please refer to Table 4 in the Appendix for details. Our evaluation metrics have been described in section 3.3. Throughout our experiments, we used the following setup:

- In each of our training scenarios, we fine-tuned the full model starting from the pre-trained BERT weights setup in the code base infrastructure of the DFP and, as such, we used a learning rate of $lr = 1\text{E}^{-5}$ throughout;
- We used AdamW optimizer (with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1\text{E}^{-6}$) throughout except in the optimizer choice experiment where the specifics are provided;
- We trained for 10 epochs except for the optimizer choice where we trained for 7 epochs. For task sampling Prop, SA and U, we fixed the number of steps per epoch at 2400 as per Stickland and Murray (2019). We tested different batch sizes to adapt with available computing resources. For single-task SBERT experiment, we used batch size of 64. For multi-task scenarios, we used 32 batch size for non GS runs and 16 for GS related runs. This means that for Prop, SA and U, we trained on the smaller datasets for SA and STS approx. 3X to 5X over whereas for the massive dataset PD we used approx. only 10% of the dataset;
- We trained all our scenarios on A100 GPU processor with 40GB of RAM.

[1]The transformation on the Pearson coefficient ensures the accuracy value for task STS lies in $[0, 1]$ for comparability with accuracy for SA and PD.

### 4.2 Results

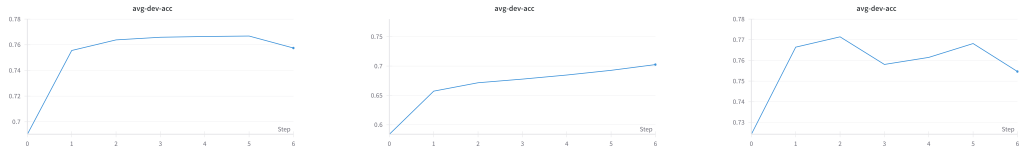
**Sentence-Level Embedddings** We have trained our model under both SBERT with Cosine-Similarity and SBERT with Concatenation for the two sentence-pair tasks PD and STS.

| Framework → | Baseline BERT Model | SBERT with Cosine-Similarity | SBERT with Concatenation |
|---|---|---|---|
| $\alpha_{\text{PD}}$ | 0.907 | 0.801 | 0.882 |
| $\rho_{\text{STS}}$ | 0.867 | 0.369 | 0.364 |

Table 1: Single-Task Sentence-Level Embeddings Experiment.

The standout result in Table 1 is the poor performance of SBERT with the STS task vs Baseline. We attribute this to two main reasons: 1) the size of the training dataset is much smaller for STS (8.6K) vs PD (283K training examples) for a sentence pair task thus limiting generalization ability in sentence-level comparisons and 2) the fact that the labels are real numbers within the range $[0, 5]$ whereas PD is a binary label for PD (PD has more data with an easier classification task). Furthermore, the SBERT architecture rely solely on added architecture on top of the BERT model to express similarity between the two input sentence embeddings. Indeed, with the baseline BERT model, all the 12 layers of multi-head self-attention blocks are relied upon to learn dependency characteristics between the two input sentences.

**Multi-Task Learning: Optimizer Choice** Because gradient updates between the three tasks can conflict, we examined the impact of different optimizers. Conflicting updates can cause oscillations, where one task's update counters another's. This issue is worse in multitask training compared to single-task training. Figure 4 shows average accuracy $\psi$ training curves for our multi-task baseline with RR task sampling under different optimizers over 7 epochs. RMSProp ( Kingma and Ba (2014)) reaches peak accuracy earlier than AdamW and SGD, and performs best overall. As expected, SGD shows slower convergence due to its constant learning rate. Although RMSProp has the highest accuracy, it is more unstable. AdamW peaks within the first 7 epochs.




(a) AdamW (Baseline); $\psi = 0.767$ (b) SGD $\mu = 0.9$; $\psi = 0.703$ (c) RMSProp $\alpha = 0.99$, $\epsilon = 1\text{E}^{-8}$; $\psi = 0.771$

Figure 4: Average dev accuracy $\psi$ for the baseline multi-task architecture under different optimizer.

**Multi-Task Performance Across Different Extension Combinations** We present in Table 2 our Multi-Task Performance Matrix illustrating the multi-task performance for our different frameworks. Our key findings are:

- **Effective PAL Contribution (+0.9%)** PAL provides a significant improvement over baseline. Indeed, adding PAL layers increases performance over baseline by 0.9% under RR. This boost is due to its task-specific parameters. In the standard BERT model, all parameters are shared except for the final linear layer. This forces the shared attention layers in the transformer blocks to simultaneously adapt to the conflicting gradient updates of different tasks, splitting attention across tasks and diluting inference accuracy for any one specific task. Against base BERT, PAl increases performance across sampling approaches except under uniform task sampling.
- **Task Sampling Importance (+1.0%)** The choice of task sampling strategy has a important impact on average accuracy. AS with PAL results in a +1.0% boost over PAL with RR, similar to the boost seen using RR between baseline and PAL. We theorize this is because the PD task has so much more data than the other two tasks, and its gradient updates are so dominant. This can be seen in the table, where for our best multitask result (in blue) the accuracy gains are coming from SA and STS, while PD accuracy actually goes down

from PAL and round robin (0.899 → 0.895). This is because annealed sampling increases the proportion of the SA and STS data compared to PD. Furthermore, we observe that AS generally is the better performing sampling approach across the model architectures.

- **Gradient Surgery Over Baseline (+1.2%)** In isolation, GS has a similar performance improvement than PAL over baseline, i.e. an increase of +1.2% under RR. However, this is not the case for the other sampling approaches indicating that GS needs significantly more gradient update steps within each epoch to be effective. GS tries to preserve the gradient update power of the dominant dataset while sheltering the gradients from the tasks with the smaller datasets and hence a more balanced task weight impair its ability to apply this mechanism.
- **Cross-Purpose Conflict PAL+GS (-1.5%)** Another prominent theme is that while PAL and GS perform much better than baseline in isolation, when combining them, mean accuracy decreases by 1.5% even with AS. We theorize that both optimizations together actually work at cross-purposes: while having dedicated layers per task with PAL of course boosts results, this boost is dependent on gradient update algorithm finding an optimal hidden state for the shared parameters, which is not the same optimal hidden state that GS favors. This can be seen in the table in the final section, where for example SA accuracy is much lower for GS + PAL with RR vs. baseline accuracy with RR: 0.492 vs 0.476. While at first we might assume that we simply need more data for STS to take full advantage of dedicated task layers and gradient modification with GS, a closer look at the table reveals that this is probably not the case: even the results for PD with PAL + GS are lower across all sampling strategies compared to baseline, indicating that GS and PAL require distinct shared parameter states to work most effectively.

| **Model** | **Task Sampling** | $\psi$ | $\alpha_{\text{SA}}$ (dev) | $\alpha_{\text{PD}}$ (dev) | $\rho_{\text{STS}}$ (dev) | **Model** | **Task Sampling** | $\psi$ | $\alpha_{\text{SA}}$ (dev) | $\alpha_{\text{PD}}$ (dev) | $\rho_{\text{STS}}$ (dev) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **BERT** | **Single Baseline** | - | **0.530** | **0.907** | **0.867** | - | - | - | - | - | - |
| BERT | **RR** | **0.767** | **0.492** | **0.893** | **0.831** | BERT | **RR** | **0.776** | **0.510** | **0.899** | **0.837** |
| | Prop | 0.768 | 0.49 | 0.877 | 0.873 | +PAL | Prop | 0.774 | 0.509 | 0.876 | 0.873 |
| | AS | 0.783 | 0.512 | 0.899 | 0.873 | | **AS** | **0.786** | **0.524** | **0.895** | **0.878** |
| | U | 0.774 | 0.508 | 0.874 | 0.878 | | U | 0.769 | 0.514 | 0.859 | 0.869 |
| BERT | **RR** | **0.779** | **0.505** | **0.896** | **0.873** | BERT | RR | 0.765 | 0.476 | 0.884 | 0.868 |
| +GS | Prop | 0.768 | 0.49 | 0.877 | 0.873 | +GS | Prop | 0.772 | 0.52 | 0.858 | 0.875 |
| | AS | 0.775 | 0.517 | 0.874 | 0.87 | +PAL | **AS** | **0.771** | **0.503** | **0.875** | **0.872** |
| | U | 0.772 | 0.507 | 0.873 | 0.87 | | U | 0.773 | 0.51 | 0.871 | 0.874 |
| | - | - | - | - | - | | SA + RR | 0.781 | 0.51 | 0.893 | 0.880 |

Table 2: Our **Multi-Task Baseline** in red and our **Best Multi-Task Model** in blue. **For our Best Multi-Task Model, here are our test set results:** $\psi = 0.779$**,** $\alpha_{\text{SA}} = 0.504$**,** $\alpha_{\text{PD}} = 0.895$ **and** $\rho_{\text{STS}} = 0.876$.

## 5 Analysis

**Confusion Matrices** We present a confusion matrix comparative analysis of our Baseline vs our Best Multi-Task Model across downstream tasks in Figure5 where the last column is the difference $\Delta$ between best and baseline matrices. A few key observations:

- **SA** ($\alpha_{\text{SA}}$**: 0.492 → 0.524**): We observe that the best model reduces error occurrences in the lower and higher diagonal of the matrices, i.e. the confusion matrix becomes more concentrated on its diagonal indicating better prediction accuracy. We also note that prediction errors are more important for the neutral, negative and positive classes and that the confusions seems to be more concentrated around neutral vs somewhat (positive or negative), i.e. the more ambiguous classes;
- **PD** ($\alpha_{\text{PD}}$**:0.893 → 0.895**): For PD, it is hard to distinguish a strong improvement as the accuracy changed marginally between the two;
- **STS** ($\rho_{\text{STS}}$**: 0.831 → 0.878**): Note that we discretized the continuous labels in 10 categories (every 0.5 between 0 and 5) for analysis purposes. We see even more clearly the best model

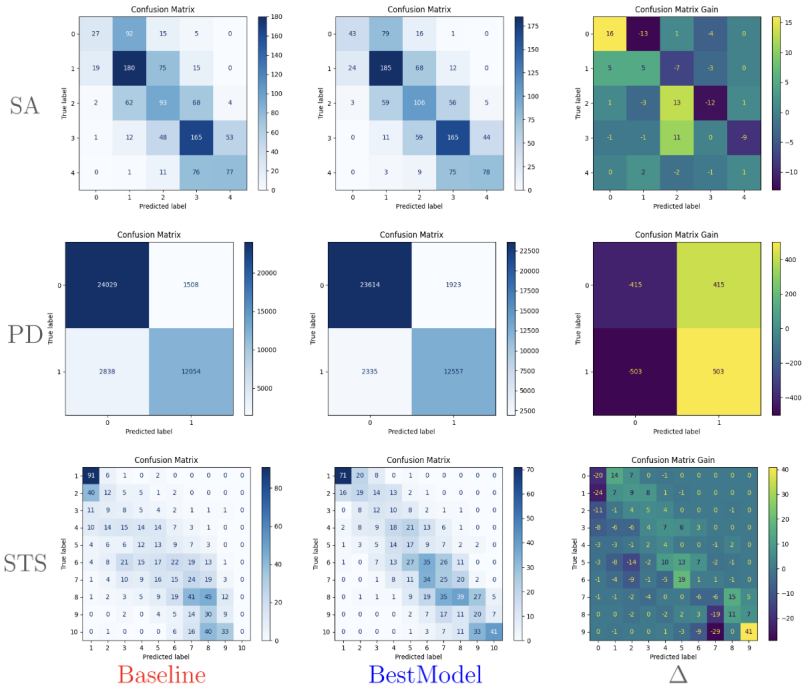



Figure 5: Confusion matrix analysis for our best model against baseline across downstream tasks.

concentrating on its diagonal and both models seem to be more confused amongst the more less similar pairs than for the more similar pairs.

**Example prediction** We analyzed specific examples to observe best model improvements or deteriorations vs baseline. On the SST task, our prediction for *"Still , as a visual treat, the film is almost unsurpassed"* went from accurate class 4 to inaccurate class 2 probably due to the unusual syntactic structure of the sentence. On the PD task however, the comparison $s_a$ = *"How could a laptop with built in internet be created?"* vs $s_b$ = *"Is it possible to have a laptop with built in internet access?* is correctly classified as paraphrase in our best model. Lastly, the comparison $s_a$ = *"It's also a matter of taste."* vs $s_b$ = *"It's definitely just a matter of preference."* on STS yields better results with our best model with prediction of 2.9 vs baseline prediction of 0.8 but still very distant from the ground truth of 5.

## 6 Conclusion

To summarize, we first explored using BERT for sentence-level embedding learning under two distinct architecture geared towards sentence-pair downstream tasks and concluded that the classic usage of the BERT model for that objective remains effective performance wise. The central theme of our multi-task context was figuring out how to manage conflicting gradient updates for our three tasks. The problem can be broken down into two distinct spheres: actual conflicting gradients that result from seeking different loss function minima for separate tasks, and the effect of one dataset (PD) being much larger than the other two, thus overwhelming the gradient updates of the other two tasks over time. To mitigate against the first problem, we deployed task-specific layers with PAL (to in effect train slightly separate models for each task) and modified the conflicting gradient updates directly with GS. To address the second problem, we experimented with various task sampling strategies to prevent PD from completely dominating the parameter updates over time. While both mitigation strategies were effective in isolation, we found that combining PAL and GS together were not additive, and actually performed worse than baseline under most task sampling approaches.

## 7 Ethics Statement

One specific area of concern for widespread AI use in society is how it might reinforce harmful prejudices and stereotypes. One important issue in understanding bias in langauge modeling is the role that names specific to a racial, ethnic or gender group might play in reproducing societal bias. For example, many sentiment analysis systems assign more negative sentiment scores to sentences that contain African American sounding names, than they do to sentences with more traditionally white sounding names. This is a result of bias in the training data, but also do to a lack of representation of black names in the pre-training corpus, positive or negative. How can we be sure of this? One way to show that black names are not as represented is to check how many tokens the WordPiece algorithm needs to represent them. This is because the more common a word is in the training corpus, the fewer tokens WordPiece will require to represent it. A simple experiment will show that WordPiece requires many more tokens to represent common black names. For example, the African American name female name Nia requires two tokens, while the white male name Geoffrey requires one.

Another issue with names revolves around gender. Again referencing sentiment analysis, female sounding names tend to be associated more with feelings of warmth and positivity, while male sounding names correlate with negativity and aggression. How can we mitigate this problem? One way would be to try to include more "balanced" training sources, but this soon becomes an almost impossible task—we have to make many subjective choices around the emotional valence of certain names, screen the context for bias, etc. Instead, there is another solution that is actually much simpler: Upon inference time, we can use a named entity recognition filter on the input for names and substitute them with the [UNK] token before making the forward pass. That way, the model can still be trained on whatever flawed corpus we have available, and still mitigate the biasing potential of names. This technique would of course also be effective for the racial name problem discussed in the above paragraph.

## 8 Team Contributions

It is of note that all team members were actively involved in all aspects of our project. These descriptions are reflective of each team member's key contributions: **Jean-Philippe Lemay** - Reports drafting, PAL implementation, task sampling, confusion analysis, poster production and general project management, **Veljko Skarich** - Reports drafting, model interpretation, code structure for extensions and multiple configurations required, gradient surgery and task sampling, and **Joe Wang** - Reports drafting, Sentence-BERT extension, literature review, training runs management and results interpretation.

## References


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

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

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

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

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

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune bert for text classification? In *Chinese computational linguistics: 18th China national conference, CCL 2019, Kunming, China, October 18–20, 2019, proceedings 18*, pages 194–206. Springer.

Wei-Cheng Tseng. 2020. Weichengtseng/pytorch-pcgrad.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

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

# A Appendix

| Task → Training ↓ | SA $\alpha_{\text{SA}}$ (dev) | PD $\alpha_{\text{PD}}$ (dev) | STS $\rho_{\text{STS}}$ (dev) |
|---|---|---|---|
| Last Layer | **0.405** | **0.701** | **0.507** |
| Full Model | **0.530** | **0.907** | **0.867** |

Table 3: Single-Task Baselines.

| **Task** | **Dataset** | **Description** | **# Train** | **# Dev** | **# Test** | **# Size** | **Weight** |
|---|---|---|---|---|---|---|---|
| SA | Stanford Sentiment Treebank (SST) | Single sentence movie reviews with 5 class sentiment label | 8,544 | 1,101 | 2,210 | 11,855 | **2.9%** |
| PD | Quora | Question pairs with paraphrase binary indicator label | 282,010 | 40,429 | 80,859 | 404,298 | **95.1%** |
| STS | SemEval STS Benchmark | Sentence pairs with 5 class similarity score label | 6,040 | 863 | 1,725 | 8,628 | **2.1%** |

Table 4: Description of the data sources and associated downstream task.

| Gradient Update Algo | $\psi$ | Epoch (dev) |
|---|---|---|
| AdamW (Baseline) | 0.767 | 5 |
| SGD | 0.703 | 7 |
| RMSProp | 0.771 | 3 |

Table 5: Optimizer Experiment.