Computing Shared Structure in Language With Multitask Learning

William Huang [1] Kapil Iyer [1] Karthik Seetharaman [1]

[1]Computer Science, Stanford

Introduction

Multitask learning [1] is a machine learning approach based on the idea that multiple related tasks can be learned jointly, allowing a single model to leverage shared structure as an inductive bias to achieve strong performance on each individual task. Models that learn multiple tasks simultaneously are theoretically more generalizable and more computationally efficient [2].

While multitask learning has many potential benefits, there are some issues with it as well, chief of which is that of *conflicting gradients*. In particular, when learning multiple tasks jointly, the model can encounter situations where gradients for different tasks conflict with or dominate one another, which can lead to sub-optimal solutions or convergence failure. **This project aims to compare the performance of various methods that have been proposed to combat the conflicting gradients problem.**

Background: Proposed Solutions

This project investigates three methods of combating the conflicting gradients problem:

1. **Gradient Normalization** (Grad Norm) [3]. The GradNorm algorithm learns the weights w_i for a loss function $L = \sum w_i L_i$, where L_i are the individual loss functions for each task. Defining $G_W^{(i)}(t)$ as the L_2 norm of the gradient of $w_i(t)L_i(t)$ at time t and $\overline{G}_W(t)$ as the average gradient norm across all tasks at time t, we aim to weight the gradient norm of each task as the average times the inverse training rate of task i, which we denote $r_i(t)$. Then, the desired gradient norm for task i is $G_W^{(i)}(t) := \overline{G}_W(t)[r_i(t)]^\alpha$, where α is a hyperparameter. Then, the GradNorm gradient descent algorithm is implemented as an L_1 loss function between the actual and target gradients:

$$L_{\text{grad}}(t; w_i(t)) := \sum_i \left| G_W^{(i)}(t) - \overline{G}_W(t) r_i(t)^\alpha \right|_1$$

2. **Projecting Conflicting Gradients** (PCGrad) [4]. In PCGrad, if two gradients are conflicting, one is projected onto the normal plane of the other to resolve the conflict. In particular, if two gradients $\mathbf{g}_1$ and $\mathbf{g}_2$ have negative cosine similarity, $\mathbf{g}_1$ is replaced with its projection onto the normal plane of $\mathbf{g}_2$, which is $\mathbf{g}_i := \mathbf{g}_i - \frac{\mathbf{g}_i \cdot \mathbf{g}_j}{||\mathbf{g}_j||^2}\mathbf{g}_j$. This is repeated for all tasks in a random order.

3. **Conflict-Averse Gradient Descent** (CAGrad) [5]. CAGrad works by finding an update vector for the parameters θ at each step that decreases each individual task losses as well as the overall loss. Given learning rate α, we consider the minimum decrease rate across losses using a first-order Taylor approximation ($[T]$ denotes the set of different tasks):

$$R(\theta, d) = \max_{i \in [T]} \left\{ \frac{1}{\alpha}(L_i(\theta - \alpha d) - L_i(\theta)) \right\} \approx - \min_{i \in [T]} \langle g_i, d \rangle.$$

Note that if $R(\theta, d) < 0$, then all losses are decreases for some small α, so $R(\theta, d)$ is a measurement of gradient conflict. On each step, CAGrad solves the optimization

$$\max_{d \in \mathbb{R}^m} \min_{i \in [T]} \langle g_i, d \rangle \text{ s.t. } ||d - g_0|| \leq c||g_0||,$$

where $c \in [0, 1)$ is a hyperparameter that controls convergence rate.

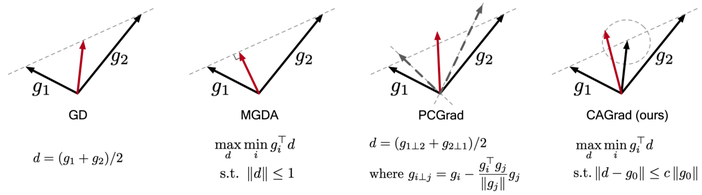


Figure 1. An overview of how gradients are modified in different methods, taken from [5]. Note that this project does not consider multiple gradient descent algorithm (MGDA).

Tasks

We analyze how different multitask-trained models perform on two related NLP tasks: Paraphrase Detection (PARA), a binary classification problem indicating whether two sentences are paraphrases of one another, and Semantic Textual Similarity (STS), a regression problem indicating the extent to which two sentences are semantically similar. We use the Quora Question Pairs and SemEval STS datasets.

Model Architecture

Our models follow a general architecture: we build on top of a shared transformer encoder and use two task-specific heads, one for the PARA task and one for the STS task. The PARA head consists of two linear layers with ReLU activations, a hidden interaction layer that receives the concatenated hidden representations. The binary classification logit is produced via a sigmoid. The STS head also consists of two linear layers with ReLU activations. The 0-5 regression logit is produced via cosine similarity, ReLU activation, then a multiplication by 5, primarily relying on finetuning the transformer to learn similar sentence representations. We incorporate a generous dropout of 50% between each layer for regularization.

As our baselines, we train a pair of individually trained models, as well as a single vanilla multitask model whose loss function is simply the average of the loss functions for the PARA and STS tasks.

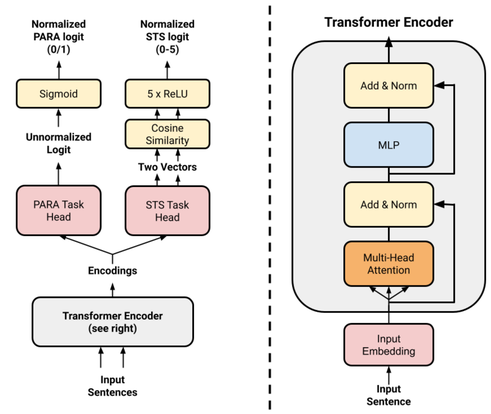


Figure 2. Left: our overall model architecture. Right: transformer encoder architecture.

Methods

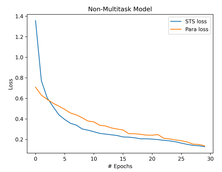
In training our models, we use mean squared error (MSE) loss for the STS regression task, and we use binary cross entropy (BCE) loss for the PARA binary classification task. We perform a grid search in order to optimize the hyperparameters for our models, including the batch size (512), the initial learning rate (1e-2) and the number of epochs (30). We use an LR scheduler to decay our LR as the validation error begins to stop increasing. We use the AdamW optimizer with a weight decay of 0.01 for regularization.

To evaluate performance, we use a standard accuracy metric. Since STS is a regression task, we define for it a custom accuracy metric defined by the labels being within ± 0.5 of the output logits. We then take the average of the PARA and STS accuracy as a model's overall accuracy, which we seek to maximize:

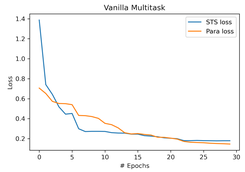
$$\text{Acc} = \frac{1}{2}(\text{Acc}_{STS} + \text{Acc}_{PARA}) = \frac{1}{2}\left(\frac{\sum_{i=1}^n \mathbf{1}[|y_{STSi} - \hat{y}_{STSi}| \leq 0.5]}{n} + \frac{\sum_{i=1}^n \mathbf{1}[y_{PARAi} = \hat{y}_{PARAi}]}{n} \right).$$

Results

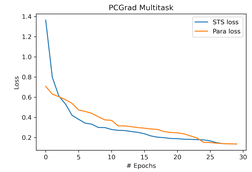
	PARA Acc.			STS Acc.			Average Acc.		
Model	Train	Val	Test	Train	Val	Test	Train	Val	Test
Random	0.500	0.500	0.500	0.200	0.200	0.200	0.350	0.350	0.350
Individual	0.883	0.781	0.779	0.652	0.518	0.522	0.768	0.650	0.651
Vanilla Multitask	0.845	0.730	0.722	0.540	0.392	0.391	0.693	0.561	0.557
PCGrad	0.885	0.782	0.782	0.650	0.520	0.514	0.768	0.651	0.648
GradNorm	0.933	0.799	0.794	0.671	0.529	0.531	0.802	0.664	0.663
CAGrad	0.913	0.789	0.784	0.669	0.524	0.524	0.791	0.657	0.654



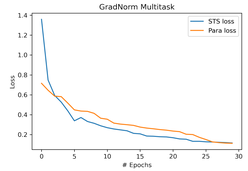
(a) Non-Multitask Train Loss



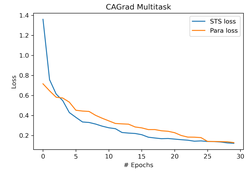
(b) Vanilla Multitask Train Loss



(c) PCGrad Train Loss



(d) GradNorm Train Loss



(e) CAGrad Train Loss

Discussion

The vanilla multitask model performs quite poorly on the STS task when compared to the individually trained models; this is likely due to the fact that the gradients from the PARA model dominated the gradients from the STS model. Also, the loss curves for both the PARA and STS tasks under the vanilla multitask model plateau well before 30 epochs, indicating conflicting gradients preventing the model from improving. This issue of conflicting gradients is somewhat noticeable in the last few epochs of the PCGrad model, but the loss curves do not flatline in both the GradNorm and CAGrad models, showing that these models perform the best.

If the topology of one task's parameter space is more challenging than the others, it can lead to the domination of gradients. For example, the gradients from a complex task can be more volatile, leading to larger updates to the shared parameters. This dominance can occur because the optimization process may spend more time navigating the challenging task's parameter space, focusing on finding the best solutions for that task, while potentially neglecting the other tasks. This likely manifests here due to a difference in complexity between the STS and PARA tasks, where the former is more complex than the latter [4].

References

[1] Rich Caruana. Multitask learning. Machine Learning, 28(1):41–75, 1997.

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

[3] Zhao Chen, et al. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks, 2018.

[4] Tianhe Yu, et al. Gradient surgery for multi-task learning, 2020.

[5] Bo Liu, et al. Conflict-averse gradient descent for multi-task learning, 2021.