**Premise:**

Our goal is to compare training times and BLEU scores after using mixed precision, distributed training, and gradient accumulation during fine tuning of a bilingual sentence translation. We use the HuggingFace Seq2SeqTrainer to easily apply optimizations on the SMaLL-100 and T5-small transformer models as well as compare speedups and accuracy within each model after optimizations.

**Technical Challenges:**

-GPU memory limits. Batch sizes were limited to accommodate the models themselves. With our models, most batch sizes caused CUDA malloc errors in DDP.

-GPU compute limits. No access to v100s or A100s for accelerated computation and evaluation.

-Due to computational and memory limitations, we weren’t able to look at finetuning on multiple language pairs, consideration of back translation, and multilingual translation.

-Lack of smaller LLMs available on HuggingFace makes experimentation less generalized. Most models are in 200+ million parameters in size.

**Approach:**

-Fine tune pre-trained language models SMaLL-100 and T5-small on the task of translating from English to Spanish using HuggingFace’s Seq2SeqTrainer.

-Hyperparameters: AdamW Optimizer with weight decay=0.01, learning rate=2e-5, beta1=0.9, beta2=0.999 used as a baseline to train the models for 1-2 epochs.

-Apply mixed precision, DDP, batch-size variation to T5-small.

-Apply mixed precision, DDP, gradient accumulation, and gradient checkpointing to SMaLL-100.

-Compare performance based on runtimes and accuracy (calculated with SacreBleu metric).

-Both models are run on GCP using 4 T4 GPUs and Distributed Data Parallel.

-T5-Small is experimented on batch sizes varying between 32 and 64, with and without fp16 mixed precision.

-SMaLL-100 was experimented on batch sizes varying between 10 and 25 per device, fp16 mixed precision enabled, varying gradient accumulation, and gradient checkpointing enabled.

-Use the opus\_books dataset with “en-es” config for the dataset with english and spanish sentence pairs for forward translation. Both experiments were trained on 101668 sentence pairs out of 127085.

**Directory** **Structure**:

* requirements.txt
* Small-100/
  + SmallTranslation.py (small100model)
  + tokenization\_small100.py (tokenizer)
* t5\_translation.py (t5 model)

We manually updated parameters within each respective file to run the following experiments:

Commands:

python3 -m torch.distributed.launch –nproc\_per\_node=4 t5\_translation.py

python3 -m torch.distributed.launch –nproc\_per\_node=4 Small-100/SmallTranslation.py

**T5\_results**:

A black screen with white text

Description automatically generated with low confidence

A black screen with white text

Description automatically generated with low confidence

A black screen with white text

Description automatically generated with low confidence

A black screen with white text

Description automatically generated with low confidence

A picture containing text, number, screenshot, line

Description automatically generated

We noticed that increasing batch size often led to CUDA out of memory issues. Additionally the combination of multiple optimizations, as expected, led to faster training than one on its own. Batch size increase resulted in the greatest loss of accuracy and moderate speedup. It seems the best combination to maintain relatively high accuracy while benefiting from faster training time would be to run the model with both distributed data parallel and mixed precision.

**SMaLL-100 Results**:

A picture containing text, screenshot, number, font

Description automatically generated

As the effective batch size increases, there is a speedup in training time along with an increase in throughput.

There is a 1.16x increase in sentence pair throughput between the worst and best cases.

There is an increase in memory footprint on the GPU as effective batch size increases even though gradient accumulation and gradient checkpointing should increase train time due in favor of memory accommodation.

BLEU scores were computed on a very limited testing set due to large evaluation time overhead. Nonetheless, there isn’t large variation in the scores across runs suggesting lack of improvement (use better optimizers and scale learning rates).