Training report

Task 3A

Preprocessing

All audios are truncated to 16 seconds to avoid CUDA_OUT_OF_MEMORY issues. I do not truncate the reference transcription, because of the time limitations of the task, but this is not ideal, as it may introduce mismatches between the audio and the transcription. In the more ideal case, the transcription should be force-aligned to the audio using an external force-aligner. Then the transcription should be truncated to follow the audio truncation.
The transcriptions are mapped to upper case to match the tokeniser.

Tokeniser

The same tokeniser that is contained within the wav2vec2-large-960h processor is also used for common-voice fine-tuning. This is to avoid needing to change the output layer of the model, which may exacerbate catastrophic forgetting. However, this causes the limitations of the tokeniser design from wav2vec2-large-960h to also be apparent during common-voice fine-tuning. The tokeniser is graphemic. A graphemic tokeniser is less related to the acoustics than a phonetic tokeniser and less related to the semantics than a sentence piece tokeniser. With a less direct relationship, more training data tends to be needed to learn the more complicated relationship patterns. If given more time, it may be more desirable to switch to either a phonetic or sentence piece tokeniser. A tokeniser change necessitates initialising a new output layer. If the tokeniser used for common-voice fine-tuning is different from that of the original seed wav2vec2-large-960h model, then extra care needs to be taken to avoid catastrophic forgetting. For example, first the new output layer could be trained with the rest of the model frozen, and then the entire model could be jointly trained after that.

Feature extractor

The same feature extractor as wav2vec2-large-960h is used. When starting from a pre-trained seed model, the extracted features for fine-tuning must match with what the model is pre-trained to expect. No data augmentation is used, because of lack of time for implementation. If given more time, the model's generalisability can be improved by audio augmentation methods such as specaug, adding noise and reverberation, vocal tract length perturbation, speed augmentation, and adding overlapped speech.

Pipeline hyper-parameters

The following hyper-parameter settings were used:

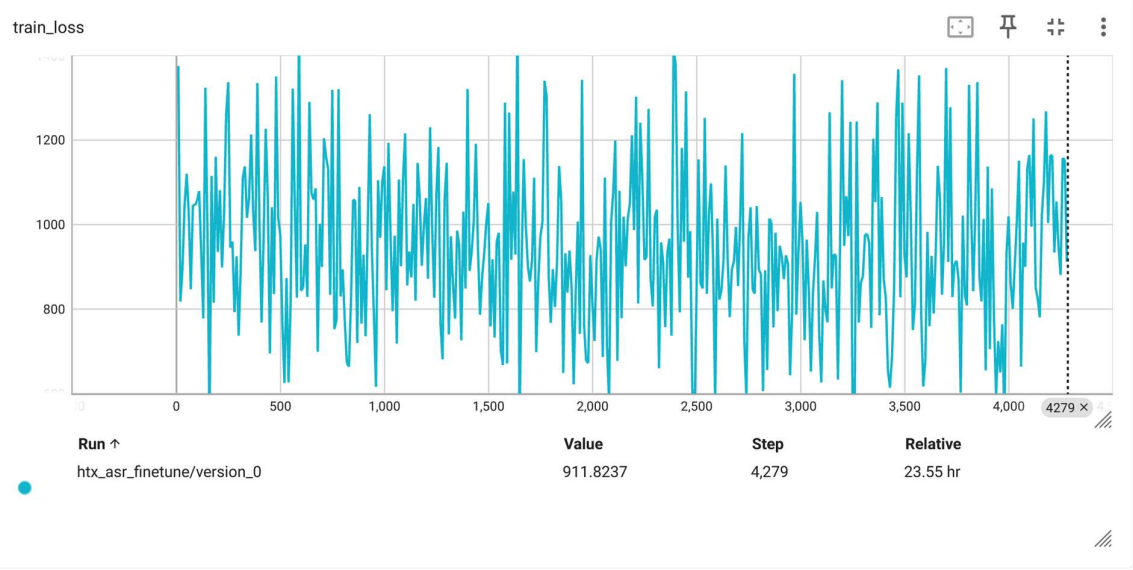
- **Gradient accumulation**
  Use accumulate_grad_batches to aggregate the gradient over more training batches before updating the parameters. This simulates a larger mini-batch, reduces gradient noise, and improves training stability.
- **Gradient clipping**
  Use gradient_clip_val to prevent single large update steps from destabilising the training. This may occur if a sample in the mini-batch is significantly mismatched with what the model is used to.
- **Learning rate schedule**
  A fixed base learning rate of 0.00001 is used. This value is set to be small enough, such that the validation loss and validation WER do not diverge at the initial training steps. A

fixed learning rate is not ideal, but is used due to time constraints. A more ideal learning rate schedule is to include an initial ramp up phase, where at initial training steps, the learning rate starts small and is gradually increased up to a point. This ramp up avoids training instabilities that may occur if the initial model parameters do not reside near to a convex region of the training criterion measured on the training set.

Visualisation of training progress

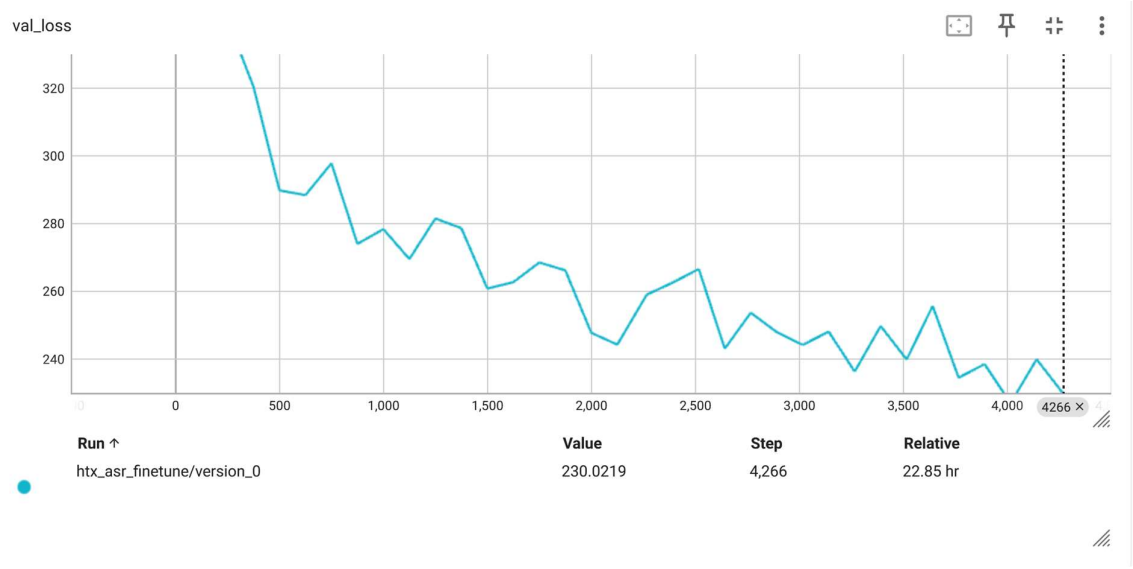
The progress of training is monitored by visualising the following variables, by using the Tensorboard logger.

- **Training loss**



The training loss is expected to be noisy, as it is computed per-mini-batch. However, it is expected to have a gradual improving trend on average, if training is proceeding properly.

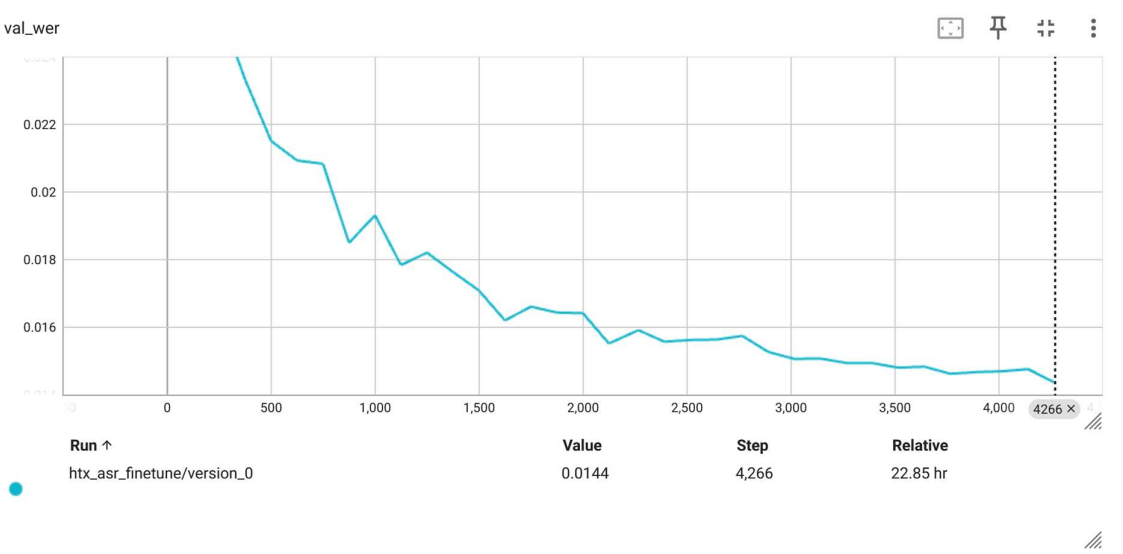
- **Validation loss**



The validation loss is expected to first improve, and then diverge when overfitting to the training data occurs. A model checkpoint near the optimum of the validation loss can

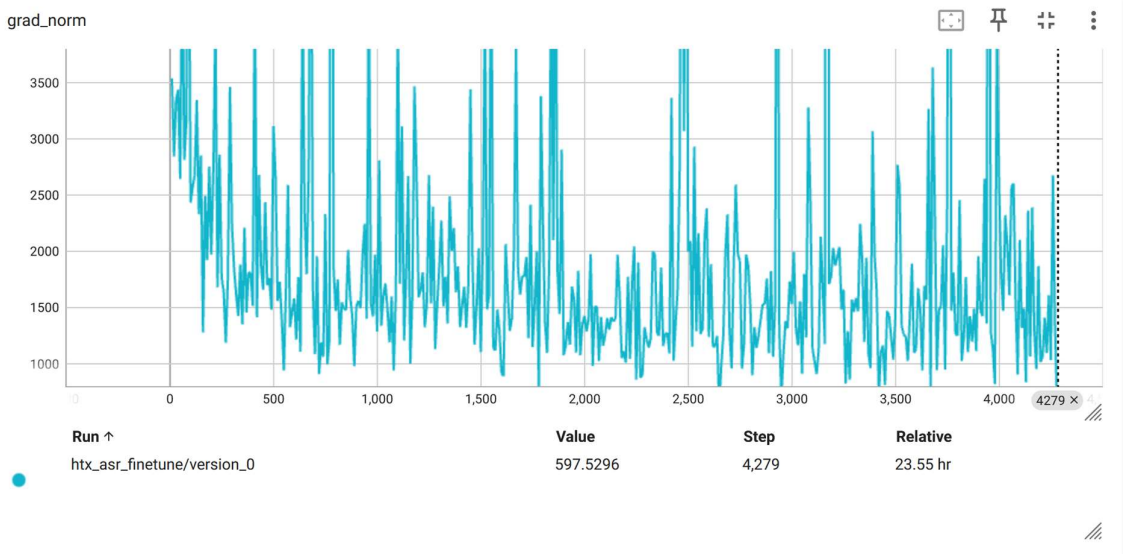
be chosen so that it generalises reasonably well to mismatched data. In the plot, training has only proceeded for two epochs due to time limitations. Thus, the validation loss has not converged yet.

- **Validation WER**



The validation WER is expected to reach an optimum and then diverge at an earlier step than the validation loss. The validation loss has only one degree of mismatch, which is in the data, compared to training. On the other hand, the validation WER has an additional degree of mismatch, which is in the metric, compared to training, and can therefore be expected to diverge earlier. A model checkpoint near the optimum of the validation WER can be expected to perform better than one near the optimum of the validation loss, when measuring the WER on mismatched data.
The validation WER can be expected to express greater fluctuation than the validation loss, due to the discretisation in its computation, making it more difficult to interpret. Fortunately, this phenomenon is not apparent in the plots here.

- **Gradient norm**



Spikes in the gradient norm may cause sudden large parameter changes, which may destabilise model training. Furthermore, frequent clipping of the gradient may hinder proper convergence. Thus, monitoring the gradient norm informs about whether these phenomena are happening.
The gradient norm in this plot does not seem to be scaled to correctly match the gradient_clip_val of PyTorch-lightning. I was not able to correct this, due to time constraints.

Task 3C

cv-valid-test WER = 7.07 %

Task 4

cv-valid-dev WER for seed model = 10.82 %
cv-valid-dev WER for fine-tuned model = 7.79 %

The WER of the fine-tuned model is better than that of the seed model, as is expected from the result of fine-tuning on data that is domain-matched with the cv-valid-dev data. However, it is reasonable to expect that the fine-tuned model may not generalise well to data that is not domain-matched with the fine-tuning data, due to the limited diversity of the fine-tuning data.

The most effective way to improve the model is to train on more diverse speech-text data. However, it is often expensive to collect such data.

Besides training on more diverse data, generalisation can also be improved during training by:

1) Using data augmentation to increase the diversity of the fine-tuning data.
2) Regularising the model parameters to not drift too far from the seed model.
3) Including data from the original pre-training of the seed model into fine-tuning if available.
4) Change the training criterion from CTC maximum likelihood to a minimum expected WER criterion, which is more closely matched with the WER evaluation metric.
5) Average the model parameters over several checkpoints, to reduce the dependence on the chosen stopping criterion. However, this requires careful tuning to ensure that the checkpoints in the ensemble are sufficiently similar to yield an averaged model that resides within a well explored region of the parameter space, and yet sufficiently diverse to yield significant combination improvements.

One approach to yield improvements without re-training the model is to modify the decoding approach. The current approach decodes by choosing the most likely grapheme at each frame. This ignores word information and the dependency between neighbouring decoded graphemes. Instead, word information can be considered in two ways. First, the search space can be constrained with a list of valid words. To do this, a graphemic lexicon is created over the list of valid words, a process that can be automated. Then, a weighted finite-state transducer can be created to represent this grapheme to word mapping. Beam search decoding using toolkits like Kaldi can then be used to generate a word lattice from the CTC model outputs. The hypothesis can then be decoded from the word lattice. Second, an external language model can be used during the beam search to incorporate word prior and transition probabilities.

Given a word lattice, it is then possible to decode by choosing the word sequence with the minimum expected edit distance against other word sequences in the word lattice distribution. This has the following advantages over simply choosing the most likely word sequence. First, it introduces the possibility to hypothesise word sequences that are not contained in the word lattice. Second, it considers all lattice paths in the decoding, thereby reducing the sensitivity to specific lattice paths and improving generalisation. Third, there is theoretical justification from Bayes' decision theory, which dictates that the measured WER can be optimised by decoding the word sequence through minimising the expected edit distance, rather than by choosing the most likely word sequence. Fourth, the process of finding such a hypothesis automatically yields word confidence scores, which may be useful to the application. This can be done by first converting the word lattice into a confusion network, using toolkits like Kaldi. Then the hypothesised word sequence can be decoded by choosing the most likely word within each confusion set.

The current model uses a CTC framework. This makes a strong assumption that the current grapheme is conditionally independent of other graphemes along the sequence. The assumption can be relaxed by instead using either a hybrid, RNN-T, or attention encoder-decoder framework. To do this, the pre-trained model can be used as the encoder in the respective framework, which is then fine-tuned for the speech recognition task.