HuggingFace vs TorchText Tokenizer

Jonathan Stokes

August 1, 2024

1 Introduction

I have been doing model training and inference using a 4GB NVIDIA GeForce GTX 1650 GPU and a 20 core Intel Core i5-13600KF Processor. This has allowed me to generate 10,000 GOPS game traces in about 30 minutes. This is fast enough for initial exploration but if I want to explore training the transformer model against a variety of training agents it would be nice if I could generate game traces faster.

When I was still working for Intel I decided to buy a 8GB Intel Arc A750 GPU. I made this decision because the Arc A750 was reasonably priced, might be interesting, and might improves Intel GPU's if I found issues in using the Arc A750 for ML. Unfortunately Intel decided to lay me off shortly after I purchased the Arc A750 and in attempting to use the GPU for ML I ran into a number of bugs which may not be problems with with the Arc A750 per se but none the less prevented me from using it. Specifically I ran into a compatibility between the MKL library which is part of Intel's oneAPI, intel-extension-for-pytorch (ipex), and Pytorch; see github bug. As initially there was no clear solution to the issue I decided to train the transformer model on the NVIDIA card.

Using the NVIDIA card I trained the GOPS transformer model and assessed its performance verse a random agent. However the Pytorch transformer model tutorial I used as the basis for the GOPS transformer model uses TorchText for tokenizing sentences. And when I returned to the github thread on the Pytorch compatibility issue and saw that someone had discovered a work around, I found that the work around was not compatible with TorchText.

The upshot of this is that I procrastinated on getting the transformer model to run on the Arc A750 GPU. Doing this required rewriting the transformer model to use HuggingFace's Tokenizer library instead of TorchText as the models tokenizer. In using the Arc A750 GPU I hoped I could reduce model training or inference time. The following gives the training and inference results for the transformer model using the HuggingFace and TorchText tokenizers on a 20 core Intel Core i5-13600KF Processor (CPU), NVIDIA GeForce GTX 1650 GPU (CUDA), and 8GB Intel Arc A750 GPU (XPU). I found that the HuggingFace tokenizer slowed down model training but sped up inference speed. I also found that the performance of the transformer model is GOPS was not the same using the HuggingFace and TorchText tokenizer.

Note that these results are using the vanilla version of HuggingFace's tokenizer library, specifically the WordLevel tokenizer model. There is also have a fast version implemented in Rust which I have yet to try. Also this is not a perfect apples to apples comparison, for example I improved the data batching when rewriting the transformer model to use the HuggingFace tokenizer; but I still think the results below are still interesting.

2 Performance Results

The following subsections look at the training and inference time and performance results for the HuggingFace and TorchText tokenizers on various devices.

2.1 Training Results

First lets look at the epoch training time for the devices and tokenizers. Here the models were trained on 100k game traces previously generated under the Unique regime, see notes "Training Results" for details. In Figure 1 we see that using the HuggingFace tokenizer the Arc GPU (XPU) is faster than training with the NVIDIA GPU (CUDA) or the CPU. However training with the NVIDIA GPU (CUDA) or the CPU using the TorchText tokenizer is significantly faster than training with the HuggingFace tokenizer. This difference in performance may be because TorchText is implemented in C++ while the vanilla implementation of the HuggingFace tokenizer is in Python, but I am not certain of this.

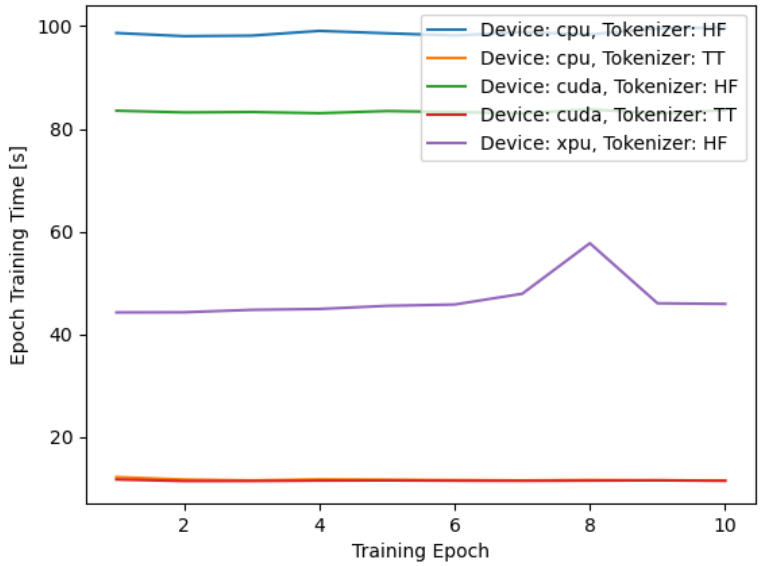

Figure 1: Epoch training time [s]. HuggingFace (HF), TorchText (TT).

Surprisingly the training performance metrics using the two tokenizer libraries also differ. In Figure 2 we see that while the validation loss reaches a minimum on training epoch 5 using both the TorchText and HuggingFace tokenizers that the the validation loss is much lower using the HuggingFace tokenizer.

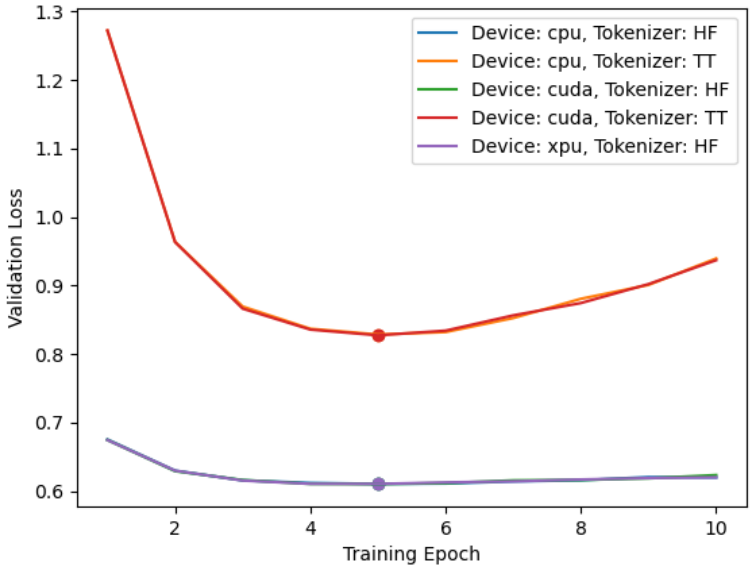

Figure 2: Validation loss, dots mark the minima. HuggingFace (HF), TorchText (TT).

In Figure 3 we see something similar with a proxy for model performance, the fraction of illegal cards selected by the transformer model. In Figure 3 we see that the fraction of illegal cards selected using the HuggingFace tokenizer is less than the fraction selected using the TorchText tokenizer.

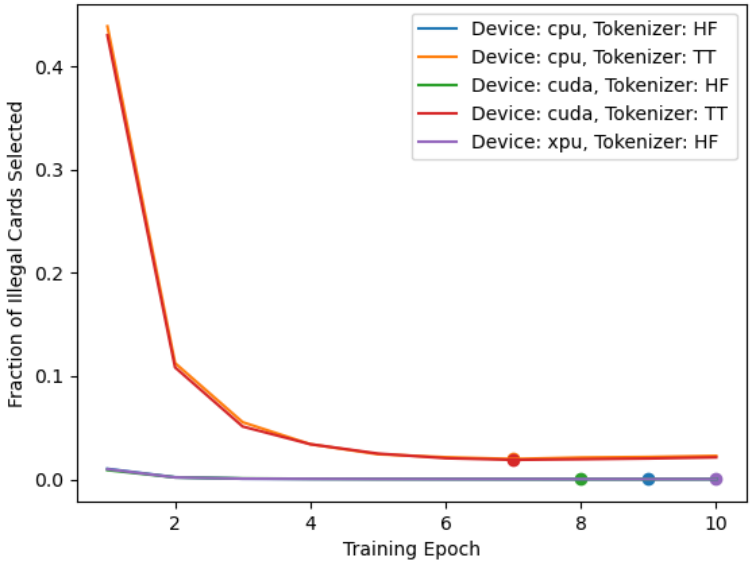

Figure 3: Fraction of illegal cards selected, dots mark the minima. HuggingFace (HF), TorchText (TT).

2.2 Inference Results

The motivation for switching from TorchText to the HuggingFace tokenizer was to reduce the time required to generate game traces for simulated games of GOPS. To measure this I timed how long it took to generate 100 game traces for the various tokenizer and device configurations. These results are for a single trial and so do not measure the variance in the time required to generate game traces, nonetheless I think the results in Table 1 serve as decent ball park numbers.

Device	Processes	Tokenizer	Gen. Time [s]
CPU	20	HuggingFace	447.93
CPU	20	TorchText	462.65
CUDA	12	HuggingFace	4.30
CUDA	12	TorchText	17.43
XPU	6	HuggingFace	8.44

Table 1: Time to generate 100 game traces [s].

The results in Table 1 show that using the HuggingFace tokenizer game trace generation is nearly twice as fast on the NVIDIA GPU (CUDA) compared to the Arc A750 (XPU) and 4 times faster than the TorchText tokenizer running on the NVIDIA GPU (CUDA). To understand these results I have to explain how game traces are generated. Each game trace is a record of a simulated game and each simulated game requires a Python multiprocessing process to run. So my 20 core CPU can run 20 game simulations simultaneously, the 4GB NVIDIA GPU (CUDA) can run 12 processes before running out of memory, and the 8GB Arc A750 GPU (XPU) can only run 6 processes before the multiprocessing pool fails to close. Now this limitation of the 8GB Arc A750 GPU (XPU) to 6 processes could be an issue with my code or it could be that the GPU runs out of memory and kills a processes without alerting the multiprocessing library. I have not been able to root cause this issue especially as I could not find a way to monitor the Arc A750 GPU (XPU)'s realtime memory usage on Linux.

As suggested by the difference validation loss and fraction of illegal cards selected in model training for the two tokenizers there is also a difference in the performance of the transformer model against a random agent for the two tokenizer libraries. First recall that the models were trained on 100k game traces previously generated under the Unique regime, see notes "Training Results" for details. In those results we saw a model achieve a win rate of 56% in 1000 simulated games. In the results below we see that in 10k simulated games the HuggingFace transformer model achieves a win rate of 58.9% and the TorchText transformer model achieves a win rate of 53.8%. For details see Table 2,

	Random Agent Wins	Transformer Model Wins	Ties	Unique States
HuggingFace	4096	5893	11	111,324
TorchText	4601	5383	16	111,229

Table 2: Win results for 10k games using HuggingFace and TorchText tokenizer.

This difference in performance using the HuggingFace and TorchText tokenizer is reflected in a difference in the the number of times a card is played as a function of the prize value. Figure 4 shows this for all moves played in the simulated games including the random agents and Figure 5 is the same data filtered for moves the led to a win. In both sets a plots we see a difference in the HuggingFace plots on the left which have more distinct "peeks" than the TorchText transformer plots on the right.

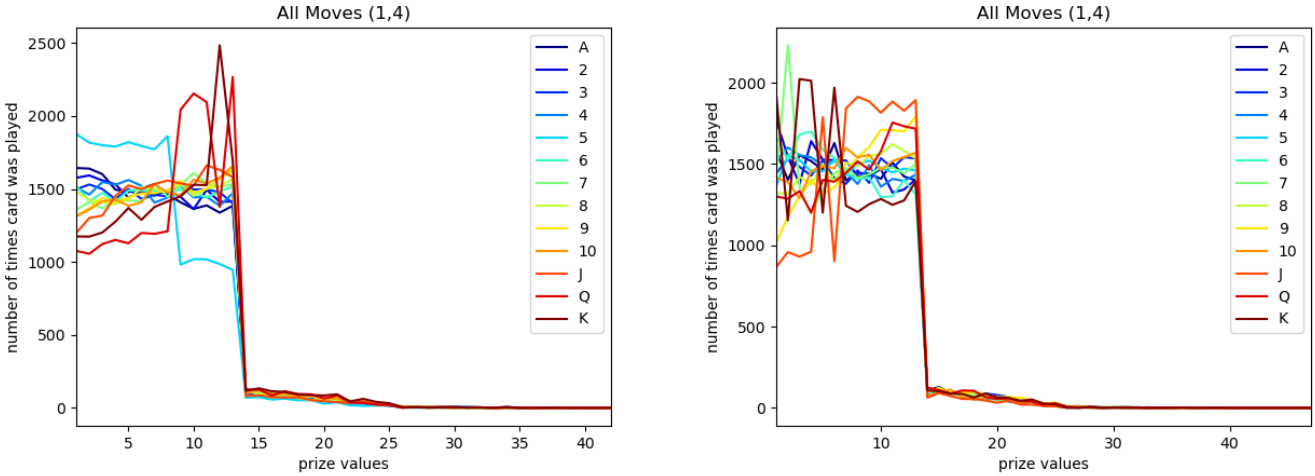

Figure 4: All cards played: **(Left)** HuggingFace tokenizer, **(Right)** TorchText tokenizer.

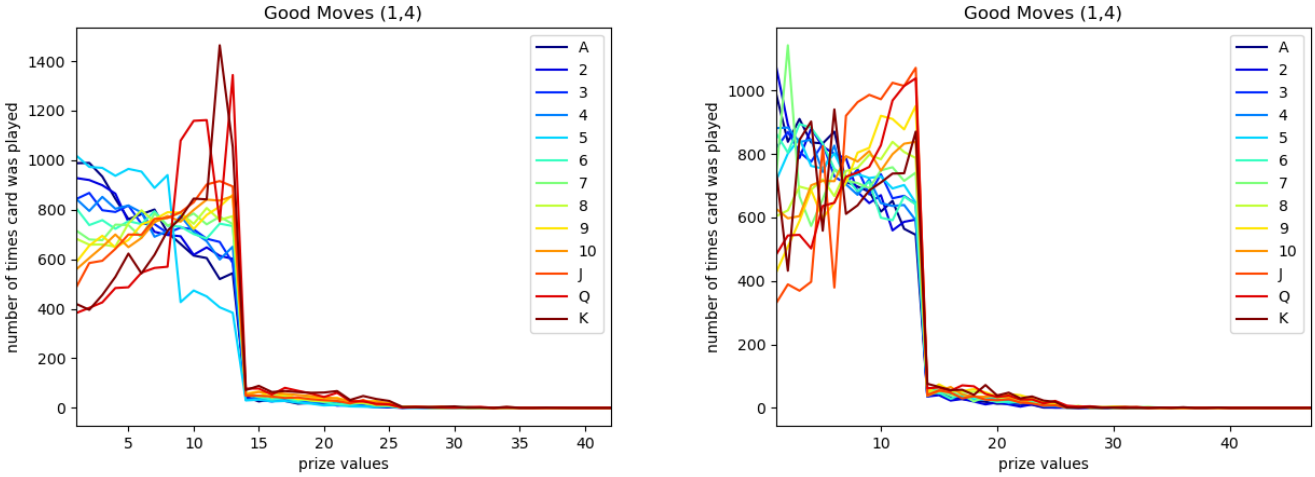

Figure 5: All cards played: **(Left)** HuggingFace tokenizer, **(Right)** TorchText tokenizer.

This difference is even clearer in Figure 6 which only plots the number of times and Ace or King was played. Figure 6 shows that the HuggingFace transformer model learned to play the King more often for high value prizes while the TorchText transformer model did not.

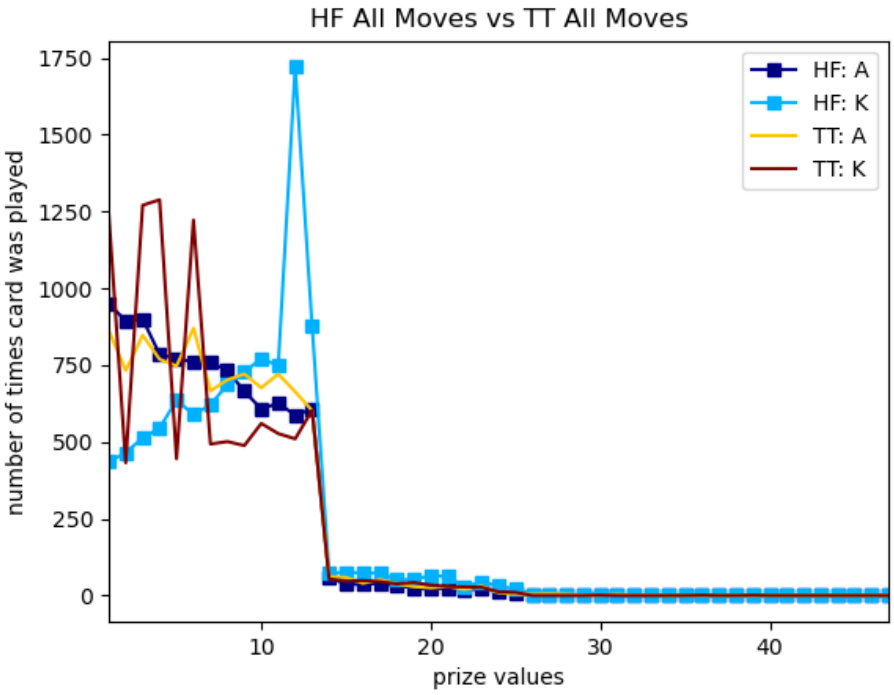

Figure 6: HuggingFace (HF) vs TorchText (TT) for all transformer moves.

3 Conclusion

I am not sure why there is a difference in performance between using the HuggingFace and TorchText transformers. My motivation for making the change was to try to run the transformer model on the Arc A750 GPU not because I thought it would have an effect on the models game performance and I have not looked into exactly how either tokenizer works. I am also not sure why the Arc A750 GPU the game trace generation time is disappointing and given the limited instrumentation Intel made public to monitor its GPU's on Linux it may not be possible to know.