AdaptNLP and fastai: Finding and Filling Gaps within Transformers

Zachary Mueller
Machine Learning Engineer
zmueller@novetta.com

What is AdaptNLP

- Beginning in early February of 2020, Andrew Chang saw a need for a simpler API when it comes to providing inference with the HuggingFace library
- At the time, the Pipeline API wasn't flexible enough, and we needed something that could be used in deployment quickly
- This became the Easy* modules, designed to quickly allow for inference with multiple models in memory and with easy to use interfaces
- As a company, Novetta saw benefit in making the library open-source and there has been community engagement and use with the library since

- Documentation: https://novetta.github.io/adaptnlp
- GitHub: https://github.com/novetta/adaptnlp

AdaptNLP and fastai

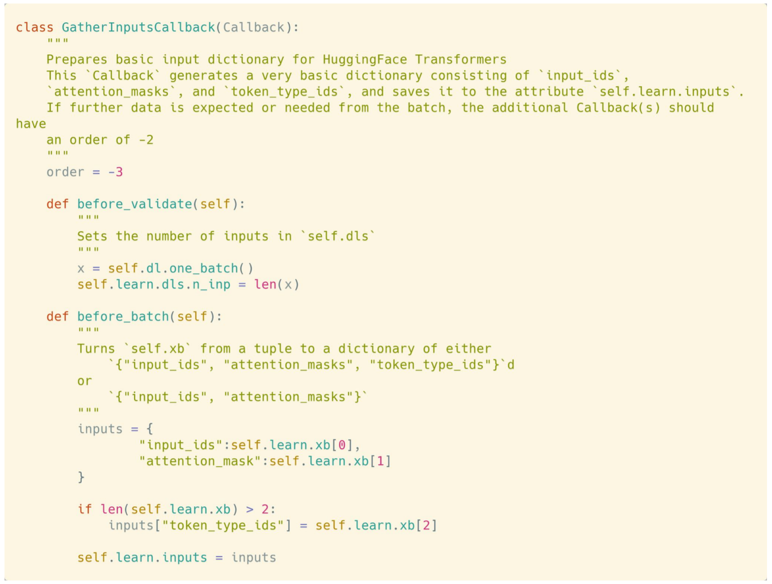
- About six months ago I was placed on the project, and soon after I was the lead developer on it
- I wanted to integrate the fast.ai suite, as it could help us **modularize** our code and help us with our long-term goal: A **centralized fine-tuning API**

AdaptNLP and fastai - Baby Steps

- It started simple, rewriting our inference API to integrate fastai through Callbacks and converting the library to nbdev with lib2nbdev
- From there I started building a new fine-tuning API to let us make use of fastai's best practices and tools such as the learning rate finder and various fit functions
- But **why** use fastai? **Why** integrate if we have working foundations in Pytorch?

Why Callbacks

- Understanding the Callback system is the **single most important aspect** of learning to use fastai and trying to integrate Pytorch
- It allows you **full access** to the fastai training loop,
- Helps make your code modularized and simple
- Lets your code become easier to work with and refactor in the future
- As a result, we could reduce our boilerplate code by at least 80%, while keeping the system clean and extendable



```
class GatherInputsCallback(Callback):
    """
    Prepares basic input dictionary for HuggingFace Transformers
    This `Callback` generates a very basic dictionary consisting of `input_ids`,
    `attention_masks`, and `token_type_ids`, and saves it to the attribute `self.learn.inputs`.
    If further data is expected or needed from the batch, the additional Callback(s) should
have
    an order of -2
    """
    order = -3

    def before_validate(self):
        """
        Sets the number of inputs in `self.dls`
        """
        x = self.dl.one_batch()
        self.learn.dls.n_inp = len(x)

    def before_batch(self):
        """
        Turns `self.xb` from a tuple to a dictionary of either
            `{"input_ids", "attention_masks", "token_type_ids"}`d
        or
            `{"input_ids", "attention_masks"}`
        """
        inputs = {
                "input_ids":self.learn.xb[0],
                "attention_mask":self.learn.xb[1]
        }

        if len(self.learn.xb) > 2:
            inputs["token_type_ids"] = self.learn.xb[2]

        self.learn.inputs = inputs
```

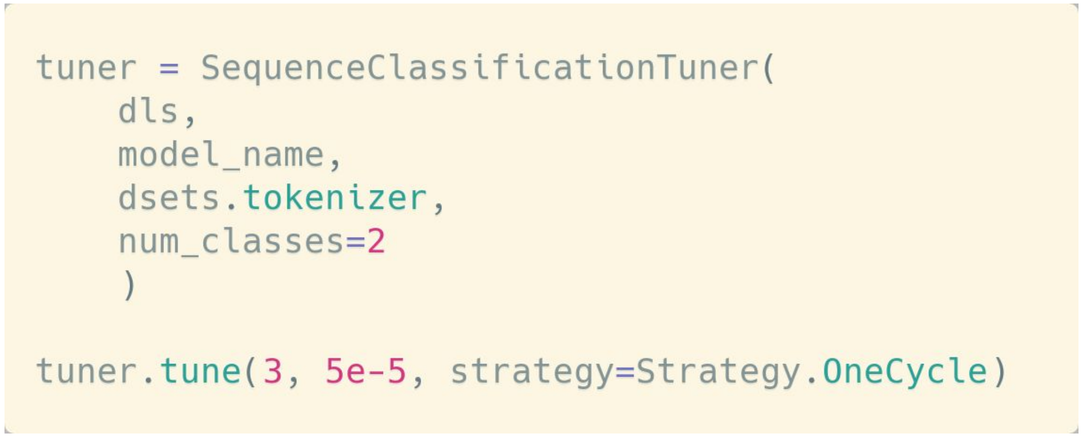
Inspirations from fastai: Data API

- fastai did a wonderful job at keeping its data and training API's be as flexible as possible for the pytorch framework, and I wanted to bring that into AdaptNLP
- This translates to:
  - A high-level API
    - Anyone can approach
    - Does not lose any flexibility
  - A mid-level API
    - Can take HuggingFace datasets
    - Can take raw torch loaders
  - A low-level API
    - Raw PyTorch and Datasets

```python
# Raw HuggingFace
raw_datasets = load_dataset("glue", "mrpc")
model_name = 'bert-base-uncased'

# Tokenize Function
def tok_func(item, tokenizer, tokenize_kwargs):
  return tokenizer(item['sentence1'], item['sentence2'], **tokenize_kwargs)

remove_cols=['sentence1', 'sentence2', 'idx']
tokenize_kwargs = {'max_length':64, 'padding':True}

# Data API
dsets = TaskDatasets(
    raw_datasets['train'], raw_datasets['validation'],
    tokenizer_name = model_name,
    tokenize_kwargs = tokenize_kwargs,
    tokenize_func = tok_func,
    remove_cols = remove_cols
)

dsets.dataloaders(
  batch_size=8,
collate_fn=DataCollatorWithPadding(tokenizer=dsets.tokenizer)
)
```

Inspirations from fastai: Training API

- Currently in development, the new fine-tuning API is more than just “accept some DataLoaders and train”
- We directly integrate with fastai’s **Learner** through two interfaces:
    - A high-level interface
        - Intuitive access to 5 or 6 basic functions
    - A low-level interface
        - Exposes the fastai Learner
        - Granting full access to the entire framework
- These two approaches allow for user flexibility everywhere without having to worry about limiting drawbacks of each approach, and in each raw PyTorch can replace what the other API performs



```
tuner = SequenceClassificationTuner(
    dls,
    model_name,
    dsets.tokenizer,
    num_classes=2
    )

tuner.tune(3, 5e-5, strategy=Strategy.OneCycle)
```

Let’s See A Demo

Questions?