MULTI-TASK TRAINING WITH HUGGING FACE TRANSFORMERS

A recipe for multi-task training with Transformers' Trainer and NLP datasets

Hugging Face has been building a lot of exciting new NLP functionality lately. The newly released NLP provides a wide coverage of task data sets and metrics, as well as a simple interface for processing and caching the inputs extremely efficiently. They have also recently introduced a Trainer class to the Transformers library that handles all of the training and validation logic.

1. Library setup:

First up, we will install the *NLP* and *Transformers* libraries.

2. Fetching our data:

convert our data from xml files to Data Frames then convert it to dictionary that contains the key is: the name of each task and the value is: the data that related with it.

3. Clean and Prepare Data:

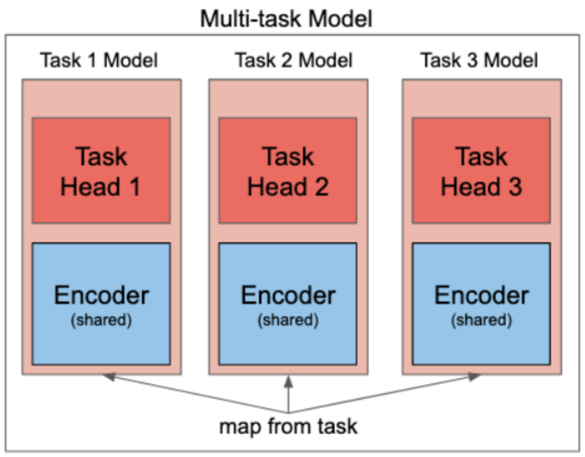
with sentiment analysis task we split each sentence into many depending on their targets so after that we can train one by one with simple way.

then we are cleaning data with many approach to removing unnecessary characters then apply Arabic stemming.

also we convert the target in each task to numeric data to can deal with models.

4. Creating a Multi-task Model:

We are going to create separate models for each task, but we are going make them share the same encoder.



the MultitaskModel class consists of only two components - the shared "encoder", a dictionary to the individual task models. Now, we can simply create the corresponding task models by supplying the invidual model classes and

.

model configs. We will use Transformers' AutoModels to further automate the choice of model class given a model architecture (in our case, let's use arabet,marbert, qarib).

5. Processing our task data:

We have created a dictionary of NLP datasets above, but we need to do a little more work to convert the respective task data into model inputs.

We'll start by first getting the tokenizer corresponding to our model.

Next, we'll convert raw text to tokenized inputs so we will convert them with the corresponding special tokens. (The tokenizer's batch_encode_plus method handles this for us).

Now we can use dataset.map method available in the NLP library to apply the functions over our entire datasets. The NLP library that handles the mapping efficiently and caches the features.

The tokenizer returns a dictionary with three important items that will be the Features in MultitaskTrainer as a train dataset which continue :
input_ids: are the indices corresponding to each token in the sentence.
attention_mask: indicates whether a token should be attended to or not.
token_type_ids: identifies which sequence a token belongs to when there is more than one sequence.
Next up, we need to: Set up our data loading ,Set up our Trainer,Start training!

6. Preparing a multi-task data loader and Trainer:

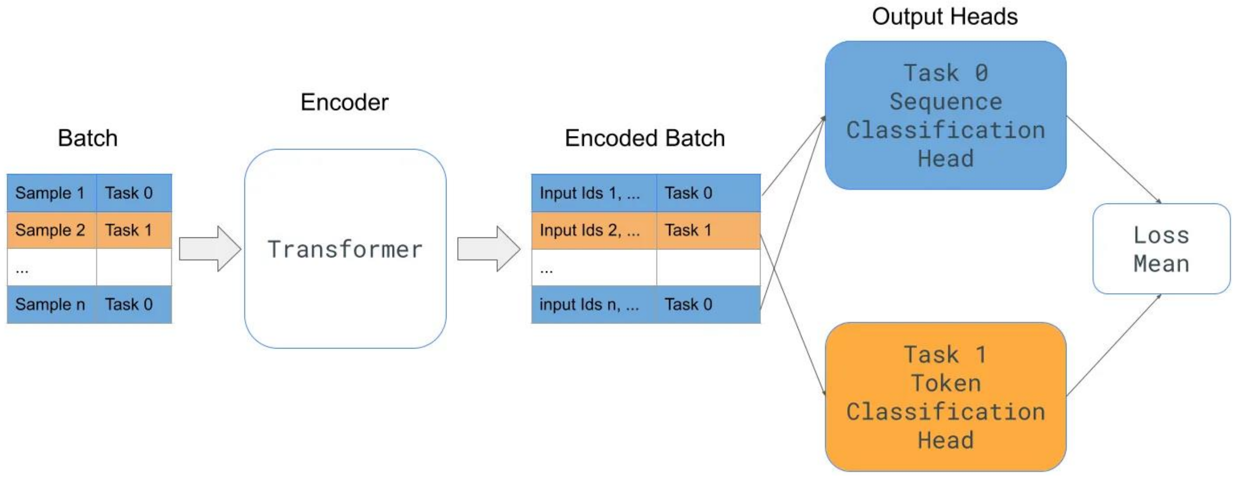
Setting up a multi-task data loader should be simple in principle - we simply need to sample from multiple single-task data loaders with some probability, and feed each batch to the multi-task model above. Of course, along with each batch, we also need to tell the model what task it is for, so MultitaskModel knows to use the right corresponding task-model.

However, because we want to use the built-in Trainer class in Transformers, this gets a little tricky, since the Trainer expects a single data loader, and expects a very specific format of per-batch data. This slice of code is somewhat of a hack around that constraint. (This can become a lot more streamlined with some tweaks to the Trainer code from the Hugging Face folks).

We need to define a MultitaskDataloader that combines several data loaders into a single "data loader". This MultitaskDataloader should do what we described: sample from different single-task data loaders, and yield a task batch and the corresponding task name (we're going to add the task_name to the batch data).

7. Time to train!

we have done all the hard work. We can now simply create our MultitaskTrainer, and start training!



(This takes about ~90 minutes for me on Colab, but it will depend on the GPU you are allocated.)

8. Evalute our multi-task model:

Now, we can evaluate our multi-task model on all three tasks. In this case, we can simply use single-task data loaders, since we are evaluating each task individually.

We will use the (private) prediction_loop method from the Trainer.

🙜 The results:

	model_name	sentimenet_valid_accuracy	sentiment_test_accuracy	ner_test_accuracy
0	arabert	0.847211	0.849907	0.732296
1	marbert	0.845594	0.838789	0.735117
2	qarib	0.854487	0.843731	0.724035