How Hugging Face helps making Language Models applicable

Thomas van der Meer

Power to the Transformers

As a big proponent of open-source, my heart breaks a little when I see new models like DALLE-2, IMAGEN and GPT-3 not released to the public. On the contrary, other companies, research groups and the community show enormous resilience by training their own models and open-sourcing them, like Stable-diffusion and BLOOM. Where access to sufficient compute can be a limiting factor nowadays, in my opinion sharing and democratizing these models are key to a bright future of using deep learning models. These models will boost innovation, let people build applications with them and inspire a generation of data scientists, software engineers, creators and everyone in between to use the power of deep learning models.

A key player in this community to make deep learning models more accessible is Hugging Face. What started out as a fun chat bot, later evolved to maintaining a Python package called Transformers and is now an open-source platform where everyone can share their neural network models. With all kinds of tools at your disposal, centralized on a so-called “hub”, training, testing, deploying and sharing your models has never been easier.

With nothing like Hugging Face in the market, this ecosystem is scaling-up to be an integral part for most that are working with transformer models. Additionally, big tech companies such as Facebook and Microsoft are on board and the Transformers architecture is expanding into different modalities like vision and reinforcement learning. Therefore, a gentle introduction to using tools from Hugging Face is justified.

Simple as that

When I started out working with transformer models in 2019, Hugging Face consisted of only the Transformers package. This allowed me to have a state-of-the-art language model at my fingertips in just in a matter of seconds.

```
1  from transformers import AutoTokenizer, AutoModelForMaskedLM
2
3  tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
4  model = AutoModelForMaskedLM.from_pretrained("bert-base-uncased")
```

load_bert_model.py hosted with ❤ by GitHub

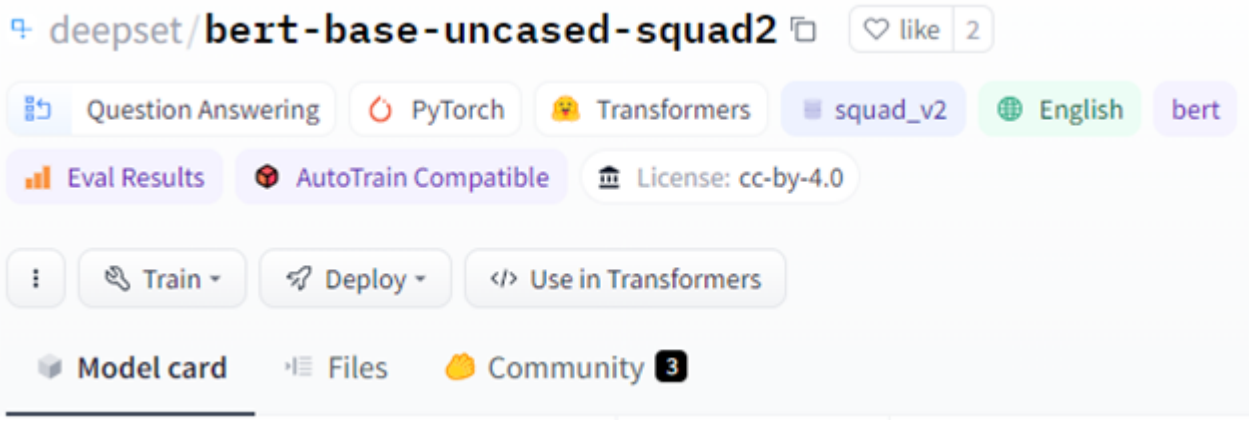
sogeti Part of Capgemini

In the code snippet of the previous page, the model loads the BERT tokenizer and model. BERT (Devlin et al., 2019) is the first large language model to use the transformers architecture introduced by the paper “Attention Is All You Need” (Vaswani et al., 2017). This introduced a systemic shift in the scientific field of NLP.

Back to loading models. The beauty of using the transformers package is the ability to use models out of the box. The catch here however is the type of model we downloaded. For an application, the model needs to be fine-tuned for a downstream task. The base model is the pre-trained version but needs a fine-tuned classifier on top to do the actual decision making. To use an example, we can do a reading comprehension task based on the Stanford Question Answering Dataset (SQuAD).

Fine-tuning vs out-of-the-box

We are at a crossroads now and have two options. Fine-tuning our own model or take a look at the Hugging Face model hub to see if there is a model that is fine-tuned for us already. With our downstream task in the back of our head (Question Answering), we take a look at the model hub and find the following.



Firstly, a model card that describes the properties of the language model. Additionally, the model is accompanied by a webpage where the owner can provide information about their model. Just like the ‘bert-base model’, you can download this model in two lines.

```
1  from transformers import AutoTokenizer, AutoModelForMaskedLM
2
3  tokenizer = AutoTokenizer.from_pretrained("deepset/bert-base-uncased-squad2")
4  model = AutoModelForMaskedLM.from_pretrained("deepset/bert-base-uncased-squad2")
```

load_model.py hosted with ❤ by GitHub view raw

If we would like to train the model, we can do that ourselves. You can load the dataset through the Datasets package and with little preprocessing, the data is ready to go. I've created a simple Google Colab notebook where you can train your own model by using a language model from the model hub. Training on Google Colab should take around 4 hours (for the 'distilbert-base-uncased' model, a smaller but competitive model) while doing 3 epochs. The number of epochs is to your own liking. I've used three, since the BERT paper uses 3 epochs for an older version of SQuAD but the RoBERTa (Liu et al., 2019) paper does two epochs (due to early-stopping).

Assessing your model

When finding a model in the model hub, the webpage shows if there are any metrics on this model available. With this information, you can quantitively decide on what model to use for your application. However, not all models have these metrics posted. Running the metrics yourself can be done through Hugging Face Evaluate. A package where you can load a model, a dataset and a (set of) metric(s) that returns the results.

```python
from transformers import AutoTokenizer, AutoModelForQuestionAnswering
from datasets import load_dataset
from evaluate import evaluator

tokenizer = AutoTokenizer.from_pretrained("navteca/roberta-large-squad2")
model = AutoModelForQuestionAnswering.from_pretrained("navteca/roberta-large-squad2")
dataset = load_dataset("squad_v2")['validation']
task_evaluator = evaluator('question-answering')
eval_results = task_evaluator.compute(
    model_or_pipeline = model,
    tokenizer = tokenizer,
    data = dataset,
    squad_v2_format = True,
    metric = "squad_v2"
)
```

run_metrics_squad.py hosted with ❤ by GitHub view raw

To go through the code snippet above step by step; First the tokenizer and then the model and dataset are loaded. The task evaluator is instantiated with a standard setting of 'question-answering'. (See this link to website for other presets). Finally, we run the evaluator by passing the model, tokenizer and data, plus some extra information about the dataset.

This results in the following output:

Search this file...

Metric	Score
exact	84.97
f1	88.31

metrics.csv hosted with ❤ by GitHub view raw

Use it

Now that we have some metrics on our model and we are satisfied with the outcome, let's see if we can get some qualitative results. Therefore, instantiating an inference pipeline to easily ingest text and give the desired output back to the user is the way to go.

```python
from transformers import pipeline, AutoTokenizer, AutoModelForQuestionAnswering

tokenizer = AutoTokenizer.from_pretrained("navteca/roberta-large-squad2")
model = AutoModelForQuestionAnswering.from_pretrained("navteca/roberta-large-squad2")

qa_pipeline = pipeline(task='question-answering', model=model, tokenizer=tokenizer)

context = r"""Eldrick Tont "Tiger" Woods (born December 30, 1975) is an American professional golf
He is tied for first in PGA Tour wins, ranks second in men's major championships, and holds numero
Woods is widely regarded as one of the greatest golfers of all time and is one of the most famous
He is an inductee of the World Golf Hall of Fame.
Following an outstanding junior, college, and amateur golf career, Woods turned professional in 19
By the end of April 1997, he had won three PGA Tour events in addition to his first major, the 199
"""

question = "when did Tiger Woods turn professional?"

result = qa_pipeline(question=question, context=context)
f"Answer: '{result['answer']}', score: {round(result['score'], 4)}"
```

pipeline_inference.py hosted with ❤ by GitHub 7 view raw

"Answer: '1996', score: 0.9396"

As stated above, the model has the ability to comprehend the text and question that is given to finally react with an answer. While testing, a simple question where the answer can be derived directly from the text is almost always spot on. Combining multiple facts from the text becomes harder and shows that there is still room for improvement in the models' reasoning abilities.

Another area of improvement is the CPU inference. This can take a while, especially if there is a lot of text to be analyzed by the model. Therefore, a GPU is most of the time a necessity. If you do not want to run this through a Colab notebook or don't have a GPU available, an option is to make use of the Hugging Face inference API.

Recap

The goal of this blog is to show a simple way to load, train, test and use language models through the Hugging Face ecosystem. I am convinced that this is a step in the right direction to democratize the use of NLP in modern applications, by making deep learning accessible to everyone. Hopefully this blog shows that you don't have to be very tech-savvy and/or a deep learning expert to use language models. By standing on the shoulders of literal tech giants, everyone can take advantage of the latest progress in the field of NLP. To get an idea what the possibilities are right now, visit the spaces page, where users show off their applications powered by transformer models. If you want to go through the steps yourself, visit this notebook to play with Transformers.