

Fancy NLP

Max Callaghan

2022-11-23

Introduction and Objectives

Assignment 2

Thanks to all who submitted assignment 2. You can expect your grades by the end of this week week.

Fancy NLP

Today we arrive at the promised “fancy NLP”. We’ll look at

1. **Transformers** and BERT-based models for machine learning with texts

Fancy NLP

Today we arrive at the promised “fancy NLP”. We’ll look at

1. **Transformers** and BERT-based models for machine learning with texts
2. **Spacy**: which is a “swiss-army knife” for doing various tasks with texts

Fancy NLP

Today we arrive at the promised “fancy NLP”. We’ll look at

1. **Transformers** and BERT-based models for machine learning with texts
2. **Spacy**: which is a “swiss-army knife” for doing various tasks with texts

We are working in python because that is what most of the machine learning and CS people use, and that is where these resources are available

Fancy NLP

Today we arrive at the promised “fancy NLP”. We’ll look at

1. **Transformers** and BERT-based models for machine learning with texts
2. **Spacy**: which is a “swiss-army knife” for doing various tasks with texts

We are working in python because that is what most of the machine learning and CS people use, and that is where these resources are available

-> For those new to python you should become slightly more familiar with how we run code in notebooks, what a “package” is, and what the code looks like.

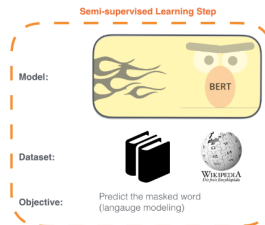
BERT

BERT

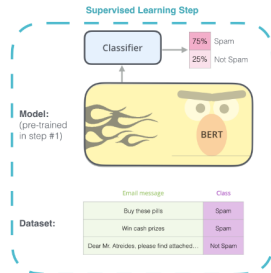
- BERT - (Bidirectional Encoder Representations from Transformers) is a **language representation model** which changed how we do NLP.

1 - **Semi-supervised** training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.



2 - **Supervised** training on a specific task with a labeled dataset.



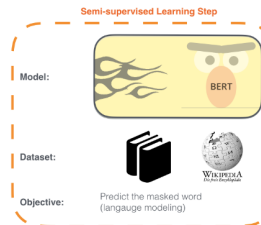
The two steps of how BERT is developed. You can download the model pre-trained in step 1 (trained on un-annotated data), and only worry about fine-tuning it for step 2. [\[Source for book icon\]](#).

BERT

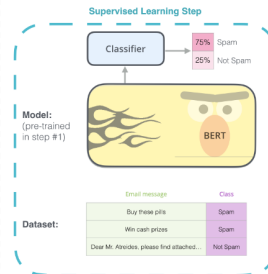
- BERT - (Bidirectional Encoder Representations from Transformers) is a **language representation model** which changed how we do NLP.
- BERT is **pre-trained** (on a masked language model task cf. WordVectors and a next sentence prediction task) on huge text corpora.

1 - **Semi-supervised** training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.



2 - **Supervised** training on a specific task with a labeled dataset.



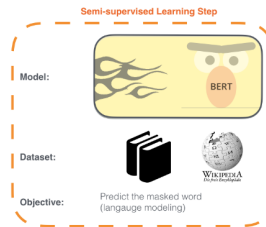
The two steps of how BERT is developed. You can download the model pre-trained in step 1 (trained on un-annotated data), and only worry about fine-tuning it for step 2. [\[Source for book icon\]](#).

BERT

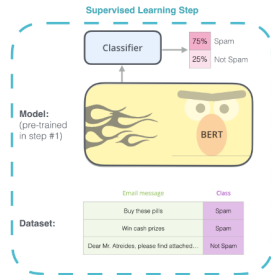
- BERT - (Bidirectional Encoder Representations from Transformers) is a **language representation model** which changed how we do NLP.
- BERT is **pre-trained** (on a masked language model task cf. WordVectors and a next sentence prediction task) on huge text corpora.
- It can be **fine-tuned** on a multitude of different tasks using new data.

1 - **Semi-supervised** training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.



2 - **Supervised** training on a specific task with a labeled dataset.



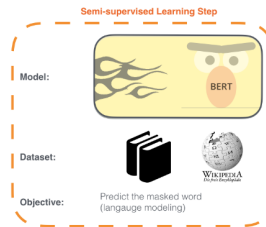
The two steps of how BERT is developed. You can download the model pre-trained in step 1 (trained on un-annotated data), and only worry about fine-tuning it for step 2. [\[Source for book icon\]](#).

BERT

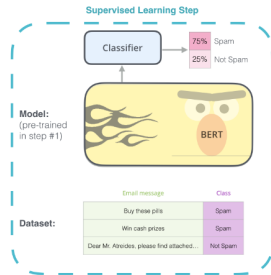
- BERT - (Bidirectional Encoder Representations from Transformers) is a **language representation model** which changed how we do NLP.
- BERT is **pre-trained** (on a masked language model task cf. WordVectors and a next sentence prediction task) on huge text corpora.
- It can be **fine-tuned** on a multitude of different tasks using new data.
- A fine-tuned BERT model offered a step-change increase in performance.

1 - **Semi-supervised** training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.



2 - **Supervised** training on a specific task with a labeled dataset.



The two steps of how BERT is developed. You can download the model pre-trained in step 1 (trained on un-annotated data), and only worry about fine-tuning it for step 2. [\[Source for book icon\]](#).

What's new with BERT

With BERT, the vector representation for each token is *dependent on its context* (using self-attention).

So, the representation of “cooler” will be different in the two sentences:

- Thursday will see **cooler** temperatures in the South-East, as a cold front moves in''
- The new remake of the film is way **cooler** than the original.


This takes us far beyond bag-of-words. BERT has learnt representations of texts that are not reliant on such unrealistic assumptions.

What's wrong with BERT

However, BERT, and large language models in general have some drawbacks (see [Stochastic Parrots](#) among others 🦜):


- Fine-tuning BERT requires more resources

What's wrong with BERT

However, BERT, and large language models in general have some drawbacks (see [Stochastic Parrots](#) among others ):

- Fine-tuning BERT requires more resources
- Understanding results and where they come from is harder

What's wrong with BERT

However, BERT, and large language models in general have some drawbacks (see [Stochastic Parrots](#) among others ):


- Fine-tuning BERT requires more resources
- Understanding results and where they come from is harder
- Training these models requires enormous resources, with significant environmental costs

What's wrong with BERT

However, BERT, and large language models in general have some drawbacks (see [Stochastic Parrots](#) among others 🦜):


- Fine-tuning BERT requires more resources
- Understanding results and where they come from is harder
- Training these models requires enormous resources, with significant environmental costs
- Models keep getting (much) bigger and (marginally) better, concentrating power in the hands of those with big compute budgets

What's wrong with BERT

However, BERT, and large language models in general have some drawbacks (see [Stochastic Parrots](#) among others ):

- Fine-tuning BERT requires more resources
- Understanding results and where they come from is harder
- Training these models requires enormous resources, with significant environmental costs
- Models keep getting (much) bigger and (marginally) better, concentrating power in the hands of those with big compute budgets
- The same problems with bias, but bigger, and less transparent!

What's wrong with BERT

However, BERT, and large language models in general have some drawbacks (see [Stochastic Parrots](#) among others ):

- Fine-tuning BERT requires more resources
- Understanding results and where they come from is harder
- Training these models requires enormous resources, with significant environmental costs
- Models keep getting (much) bigger and (marginally) better, concentrating power in the hands of those with big compute budgets
- The same problems with bias, but bigger, and less transparent!
- ...

Hugging-Face

The Hugging-Face Ecosystem

Huggingface is an ecosystem of thousands of models and datasets for NLP tasks, but also audio and computer vision tasks.

Natural Language Processing



Conversational

1,634 models



Fill-Mask

4,840 models



Question Answering

2,632 models



Sentence Similarity

931 models



Summarization

618 models



Table Question Answering

37 models



Text Classification

10,000 models



Text Generation

6,748 models



Text2Text Generation

2,010 models



Token Classification

5,500 models



Translation

1,784 models



Zero-Shot Classification

25 models

Contribute 🍌

Contribute 🍌

Pipelines

We can run many of these tasks, with any model we can find on huggingface, with just a few lines of code using **Pipelines**.

```
from transformers import pipeline
from rich.pretty import pprint
pipe = pipeline("sentiment-analysis")
```

```
## No model was supplied, defaulted to distilbert-base-uncased-finetuned-sst-2-english and revision af0f99b (https:
## Using a pipeline without specifying a model name and revision in production is not recommended.
```

```
res = pipe(["This movie was really bad", "I loved watching this movie"])
pprint(res)
```

```
## [
##   {'label': 'NEGATIVE', 'score': 0.9998040795326233},
##   {'label': 'POSITIVE', 'score': 0.999700665473938}
## ]
```

Pipelines - Text classification

If you look through the models on huggingface and [filter by task](#), you will find a variety of pre-trained models for text classification. Using one of these is simple.

```
pipe = pipeline("text-classification", model="nbroad/ESG-BERT")
res = pipe("The Hertie School is committed to embedding and mainstreaming diversity, equity and inclusion into all")
pprint(res)
```

```
## [
##   {
##     'label': 'Employee_Engagement_Inclusion_And_Diversity',
##     'score': 0.9726636409759521
##   }
## ]
```

Pipelines - Text Generation (here be dragons!)

You can also use a model to generate text, but be warned this is likely to cause convincing-sounding “hallucinations” [examples and discussion](#).

```
from textwrap import wrap
run_galactica = False
if run_galactica:
    pipe = pipeline("text-generation", model="facebook/galactica-1.3b")
else:
    pipe = pipeline("text-generation")
```

```
## No model was supplied, defaulted to gpt2 and revision 6c0e608 (https://huggingface.co/gpt2).
## Using a pipeline without specifying a model name and revision in production is not recommended.
```

```
res = pipe("Large language models can be useful. However,")
```

```
## Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```

```
## /home/max/software/py39/lib/python3.9/site-packages/transformers/generation_utils.py:1202: UserWarning: Neither
## warnings.warn(
```

```
pprint(wrap(res[0]["generated_text"]))
```

```
## [
##   'Large language models can be useful. However, more and more languages',
##   'can contain different language features, and the lack of',
##   'representation of such features means it's not clear how different',
##   'their representation is from each other. As it stands, an example of a',
```


Exercise

Generate a response using any pipeline that shows the potential of language models to cause *harm*.

Describe a real world application in which it would do so.

Fine-tuning

We can also fine-tune any pre-trained model for a classification problem for which we have labelled examples. This requires a few steps, some not inconsiderable computational power, and some patience.

Using GPUs speeds things up considerably.

We will run a small example on our own machines, with a small set of texts.

```
# Let's take our texts and our labels again
texts, y = zip(
    *[
        ("Climate change is impacting human systems", 1),
        ("Climate change is caused by fossil fuels", 0),
        ("Agricultural yields are affected by climate change", 1),
        ("System change not climate change", 0),
        ("higher temperatures are impacting human health", 1),
        ("Forest fires are becoming more frequent due to climate change", 1),
        ("Machine learning can read texts", 0),
        ("AI can help solve climate change!", 0),
        ("We need to save gas this winter", 0),
        ("More frequent droughts are impacting crop yields", 1),
        ("Many communities are affected by rising sea levels", 1),
        ("Global emissions continue to rise", 0),
        ("Ecosystems are increasingly impacted by rising temperatures", 1),
        ("Emissions from fossil fuels need to decline", 0),
```

Tokenization

First we need to tokenize our texts. It's easier if we put everything in a huggingface dataset object and tokenize this

```
from datasets import Dataset
from transformers import AutoTokenizer
dataset = Dataset.from_dict({"text": texts, "label": y})
model_name = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
def tokenize_function(examples):
    return tokenizer(examples["text"], padding="longest", truncation=True)
tokenized_dataset = dataset.map(tokenize_function, batched=True)
```

```
## 0%|          | 0/1 [00:00<?, ?ba/s]100%|#####| 1/1 [00:00<00:00, 90.43ba/s]
```

```
tokenized_dataset
```

```
## Dataset({
##   features: ['text', 'label', 'input_ids', 'token_type_ids', 'attention_mask'],
##   num_rows: 15
## })
```

Tokenization

To make this even simpler, we can create a function that turns a list of texts (and list of labels) into a tokenized dataset, given a tokenizer.

```
def datasetify(x, tokenizer, y=None):
    data_dict = {"text": x}
    if y is not None:
        data_dict["label"] = y
    dataset = Dataset.from_dict(data_dict)

    def tokenize_function(examples):
        return tokenizer(examples["text"], padding="longest", truncation=True)

    return dataset.map(tokenize_function, batched=True)

train_data = datasetify(texts, tokenizer, y)
```

```
## 0%|          | 0/1 [00:00<?, ?ba/s]100%|#####| 1/1 [00:00<00:00, 262.31ba/s]
```

Training a model

We can load a model and train it using an instance of the Trainer class.

```
from transformers import AutoModelForSequenceClassification, Trainer
# We set num_labels to 2 for binary classification, as we have two classes - positive and negative
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)
```

```
## Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertForSequenceClassif
## - This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a model trained
## - This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint of a model that
## Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncase
## You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
trainer = Trainer(model=model, train_dataset=datasetify(texts, tokenizer, y))
# Once this has been instantiated we can apply the train() method
```

```
## 0%|          | 0/1 [00:00<?, ?ba/s]100%|#####| 1/1 [00:00<00:00, 260.27ba/s]
```

```
trainer.train()
```

```
## {'train_runtime': 20.8521, 'train_samples_per_second': 2.158, 'train_steps_per_second': 0.288, 'train_loss': 0.6
## TrainOutput(global_step=6, training_loss=0.6397539774576823, metrics={'train_runtime': 20.8521, 'train_samples_p
##
```

```
## The following columns in the training set don't have a corresponding argument in `BertForSequenceClassification.
## /home/max/software/py39/lib/python3.9/site-packages/transformers/optimization.py:306: FutureWarning: This implem
## warnings.warn(
```

```
## ***** Running training *****
```

```
## Num examples = 15
```

Making predictions

With a trained model, we can make predictions on a set of new texts

```
# To generate predictions, we just need to supply a dataset to the predict method
new_texts = [
    "climate change is impacting terrestrial ecosystems",
    "Machine Learning will solve climate change",
    "Fossil fuels are responsible for rising temperature",
]
new_y = [1,0,0]

pred = trainer.predict(datasetify(new_texts), tokenizer, new_y))
```

```
## 0%|          | 0/1 [00:00<?, ?ba/s]100%|#####| 1/1 [00:00<00:00, 315.65ba/s]
## The following columns in the test set don't have a corresponding argument in `BertForSequenceClassification.forward`
## ***** Running Prediction *****
## Num examples = 3
## Batch size = 8
## 0%|          | 0/1 [00:00<?, ?it/s]100%|#####| 1/1 [00:00<00:00, 3216.49it/s]

pred
```

```
## PredictionOutput(predictions=array([[ -0.49534038, -0.13962194],
## [ 0.10782427, -0.41164818],
## [-0.18241827, -0.51983464]], dtype=float32), label_ids=array([1, 0, 0]), metrics={'test_loss': 0.51213175})
```

Turning logits into probabilities

As we did last week, we need a variation of the **sigmoid** function (**Softmax**) to turn our predictions (which are returned as logits) into probabilities. The softmax activation function ensures the probabilities for each class add up to 1 for each document. Note that these probabilities are not necessarily well calibrated.

```
# To generate predictions, we just need to supply a dataset to the predict method
from torch import tensor
from torch.nn import Sigmoid, Softmax
activation = (Softmax())
activation(tensor(pred.predictions))
```

```
## tensor([[0.4120, 0.5880],
##         [0.6270, 0.3730],
##         [0.5836, 0.4164]])
```

```
##
```

```
## <string>:1: UserWarning: Implicit dimension choice for softmax has been deprecated. Change the call to include d
```

Creating a predict_proba function

If we miss our predict_proba function from sklearn, we can *subclass* Trainer, to provide an additional method

```
from transformers.trainer_utils import PredictionOutput

class ProbTrainer(Trainer):
    def predict_proba(self, test_dataset: Dataset) -> PredictionOutput:
        logits = self.predict(test_dataset).predictions
        activation = Softmax()
        return activation(tensor(logits)).numpy()
```

```
trainer = ProbTrainer(model=model, train_dataset=datasetify(texts, tokenizer, y))
```

```
## 0%|          | 0/1 [00:00<?, ?ba/s]100%|#####| 1/1 [00:00<00:00, 228.55ba/s]
## No `TrainingArguments` passed, using `output_dir=tmp_trainer`.
## PyTorch: setting up devices
## The default value for the training argument `--report_to` will change in v5 (from all installed integrations to
trainer.train()
```

```
## {'train_runtime': 22.1918, 'train_samples_per_second': 2.028, 'train_steps_per_second': 0.27, 'train_loss': 0.30
## TrainOutput(global_step=6, training_loss=0.30997474988301593, metrics={'train_runtime': 22.1918, 'train_samples_
##
## The following columns in the training set don't have a corresponding argument in `BertForSequenceClassification.
## ***** Running training *****
## Num examples = 15
```


Creating a predict_proba function

If we miss our predict_proba function from sklearn, we can *subclass* Trainer, to provide an additional method

```
pred = trainer.predict_proba(datasetify(new_texts, tokenizer))
```

```
## 0%|          | 0/1 [00:00<?, ?ba/s]100%|#####| 1/1 [00:00<00:00, 294.83ba/s]
## The following columns in the test set don't have a corresponding argument in `BertForSequenceClassification.forward`
## ***** Running Prediction *****
##   Num examples = 3
##   Batch size = 8
## 0%|          | 0/1 [00:00<?, ?it/s]100%|#####| 1/1 [00:00<00:00, 3339.41it/s]
## <string>:5: UserWarning: Implicit dimension choice for softmax has been deprecated. Change the call to include d
```

```
pred
```

```
## array([[0.2340269 , 0.7659731 ],
##        [0.7636742 , 0.23632582],
##        [0.84745055, 0.15254942]], dtype=float32)
```

Hyperparameters

The original BERT paper defined a hyperparameter space that should be searched when fine-tuning a BERT model. These are data-dependent and values outside these ranges may make improvements in some contexts.

```
params = {
    "batch_size": [16, 32],
    "learning_rate": [5e-5, 3e-5, 2e-5],
    "number of epochs": [2,3,4]
}
import itertools
def product_dict(**kwargs):
    keys = kwargs.keys()
    vals = kwargs.values()
    for instance in itertools.product(*vals):
        yield dict(zip(keys, instance))
param_space = list(product_dict(**params))
len(param_space)
```

Hyperparameters

We can plug these values into a `TrainingArguments` object, which we pass to our trainer. This would take quite some time.

```
from transformers import TrainingArguments
for p in param_space:
    training_args = TrainingArguments(
        num_train_epochs=p["number of epochs"],
        learning_rate=p["learning_rate"],
        per_device_train_batch_size=p["batch_size"],
        output_dir="out"
    )
    trainer = ProbTrainer(model=model, train_dataset=datasetify(texts, tokenizer, y), args=training_args)
    trainer.train()
    # Evaluate our model
```

Pre-training

If we have lots of *unlabelled* data from a specific domain, doing additional *pretraining* of a transformer-based model may increase its performance in classifying *labelled* data.

See [Don't Stop Pretraining: Adapt Language Models to Domains and Tasks](#).

However, this is *much* more resource intensive, and there are now thousands of models available, some of which might have been trained on similar data.

Exercise - Training your own model

In small groups, repeat the classification task in last week's slides with a model of your choosing from transformers.

Did the model perform better or worse than our support vector machine?

Spacy

What is spacy

Spacy provides “industrial-strength” Natural Language Processing.

It is most useful for processing texts, but its latest version also claims support for Transformers and the huggingface model ecosystem.

We'll explore a few examples to get you used to the syntax and documentation

Advanced tokenization

You can process a document into a list of tokens with Spacy, and each token contains many **attributes and methods**

```
import spacy
nlp = spacy.load("en_core_web_md")
text = "Students at the Hertie School in Berlin learn how to use Spacy"
doc = nlp(text)
for token in doc:
    print(token.text, token.lemma_, token.pos_, token.is_stop)
```

```
## Students student NOUN False
## at at ADP True
## the the DET True
## Hertie Hertie PROPN False
## School School PROPN False
## in in ADP True
## Berlin Berlin PROPN False
## learn learn VERB False
## how how SCONJ True
## to to PART True
## use use VERB False
## Spacy Spacy PROPN False
```


Named Entity Recognition

Named entities are real-world objects referred to in a text. Spacy can guess what phrases are what types of real-world objects.

```
nlp = spacy.load("en_core_web_md")
doc = nlp(text)
for ent in doc.ents:
    print(ent.text, ent.label_)
```

```
## the Hertie School ORG
```

```
## Berlin GPE
```

Embeddings in Spacy

Spacy also already puts our documents and tokens into an embedding space

```
nlp = spacy.load("en_core_web_md")
doc = nlp(text)
tok = doc[0]
print(tok.vector.shape)
```

```
## (300,)
```

```
texts = [
    "The acclaimed author penned novels based on her life",
    "Nobel prize-winning writer writes autobiographical fiction"
]
docs = [nlp(t) for t in texts]
docs[0].similarity(docs[1])
```

```
## 0.6186866838865561
```

Exercise

In small groups, retrieve our manifesto dataset.

Extract only the adjectives from the manifestos.

Which adjectives are used most often by each party?

Wrapup

Wrapup

Today we have covered

- How transformer-based models work (at a very high level) and what they do
- How to use pre-trained models for a variety of tasks
- How to fine-tune existing models for our own classification tasks
- How to use spacy

This concludes the input part of the course!

Assignment 3

Next week we'll see 7 student presentations.

Each group will have **8** minutes to present (*strictly enforced*), with 4 minutes for questions (mainly from the audience!).

Present directly from your laptop, which we will connect to the projector via HDMI cable (bring an adaptor if necessary). Try to keep transitions < 1 minute!

Introduction and Objectives
○○○

BERT
○○○○

Hugging-Face
○○○○○○○○○○○○○○○○○○○○

Spacy
○○○○○○

Wrapup
○○○●