Nikolay Vorontsov

LLMs and GenAI for NLP, 2024
Report on the Exercises in Labs 1 – 6
GitHub repository: **nicksnlp**

This is what I have done:

Week 1

What are tokenisers?

Tokenisers are essential for the implementation of neural networks, each model exist with a tokeniser that was used when training that network.

To tokenise the text means to break it into pieces, whether into words or to smaller parts such as suffixes and stems, characters. There are different tokenisers available, each is best suited to a particular task and/or language.

Why are they important for language modelling and LLMs?

They are crucial for the further processing of texts. Tokenisation simplifies the text, breaks it into logical units, reduces the vocabulary size, as well as enable models to handle new and rare words. This decreases the training time for the models, and improves their generalisation capabilities.

What different tokenisation algorithms there are and which ones are the most popular ones and why?

Text can be tokenised in different ways: into sentences (e.g. in NLTK `sent_piece`), words (e.g.`split()` in python) or on some smaller elements: morphemes (rule-based tokenisation), or into parts of the words, selected on other principles, into characters or bytes.

Some of the well known tokenisers include BPE (Byte Pair Encoding) and Sentence-Piece:

BPE
BPE initially splits the texts of the given sample into characters, but then learning from the co-occurence, merges some of the characters into larger units, this is done recursively. The result of this tokenisation is usually a vocabulary of some *subword-units*, not necesserily morphemes. BPE is good for treating rare or unknown words, among other things. BPE's simple algorithm is provided in [2]:

```
import re
import collections

def get_stats(vocab):
    pairs = collections.defaultdict(int)
    for word, freq in vocab.items():
        symbols = word.split()
        for i in range(len(symbols) - 1):
            pairs[symbols[i], symbols[i + 1]] += freq
```

```
    return pairs

def merge_vocab(pair, v_in):
    v_out = {}
    bigram = re.escape(' '.join(pair))
    p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
    for word in v_in:
        w_out = p.sub(''.join(pair), word)
        v_out[w_out] = v_in[word]
    return v_out

vocab = {
    'l o w </w>': 5,
    'l o w e r </w>': 2,
    'n e w e s t </w>': 6,
    'w i d e s t </w>': 3
}

num_merges = 10

for i in range(num_merges):
    pairs = get_stats(vocab)
    if not pairs:
        break
    best = max(pairs, key=pairs.get)
    vocab = merge_vocab(best, vocab)
    print(vocab)
```

Sentence-Piece
Sentence-Piece is another language-independent subword tokeniser, based on unigram model in combination with BPE. It works well with non-phonemic symbols, such as Japanese and Chinese, and is used in such models as T5, XML-R or mBERT.

The traditional BERT model uses WordPiece. GPT-2 and GPT-3 models uses BPE on a byte-level, which is effective for treating special characters, for example.

The combination of strategies for tokenisation may be useful to fit training for a particular domain.

An important feature of every tokeniser is also the ability to decode back the tokens into the readable texts, without losses! For example in neural machine translation, a translated text is evaluated on the decoded examples in comparison to the validation counterparts.

But why not to tokenise everything into bytes, or at least characters?
Well, it will lead to very long sequences that the neural network have to process, the vectors will become too long to compute efficiently, and some semantic information that comes from co-occurrence may be lost, or at least require much more computational power to be captured by transformers or other models during training.

References:

1. “Why are tokenisers important for language modelling and LLMs?”, and further discussion. ChatGPT, OpenAI, 31 Oct. 2024
2. “tokenisers”, and further discussion. ChatGPT, OpenAI, 31 Oct. 2024
3. Neural Machine Translation of Rare Words with Subword Units: https://arxiv.org/abs/1508.07909
4. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing: https://arxiv.org/abs/1808.06226
5. Notes on BERT tokenizer and model

Week 2

References:

Week 3

References:

Week4

Fine-tuning a model with LLMs, PEFT, LoRA

supervised_finetuning.ipynb

I've created accounts on HuggingFace, Weights&Biases, I will use my regualar Google account. I use access tokens in *Secrets* on Colab.

I am running notebook on Colab Pay-As-You-Go, T4 GPU.

- ☑ Loaded dataset, here is an example:

```
Below is an instruction that describes a task. Write a response that
appropriately completes the request.

### Instruction:
Give three tips for staying healthy

### Response:
1. Eat a balanced and nutritious diet: Make sure your meals are inclusive
of a variety of fruits and vegetables, lean protein, whole grains, and
healthy fats. This helps to provide your body with the essential nutrients
to function at its best and can help prevent chronic diseases.

2. Engage in regular physical activity: Exercise is crucial for
maintaining strong bones, muscles, and cardiovascular health. Aim for at
least 150 minutes of moderate aerobic exercise or 75 minutes of vigorous
exercise each week.
```

```
3. Get enough sleep: Getting enough quality sleep is crucial for physical
and mental well-being. It helps to regulate mood, improve cognitive
function, and supports healthy growth and immune function. Aim for 7-9
hours of sleep each night.
```

- ✅ Created config object
- ✅ Downloaded the model
- ✅ Downloaded tokeniser

Applied for an Academic account at **W&B**. Now I can hopefully visualise and save models easier.

I have tried to set-up training parametres, but keep getting errors for different arguments (*max_seq_length*, *dataset_text_field*, *packing*) of their incompatibility with SFTTrainer:

For example:

```
TypeError: SFTTrainer.__init__() got an unexpected keyword argument
'max_seq_length'
```

Solution:

I have changed `tokenizer` into `proccessing_class` in `SDTTrainer`.

Commenting out `max_seq_length=None` etc. from the `SFTTrainer` arguments seems also to work.

For now I will follow the pre-existed setup, with no truncation, padding, max_length. But there is an option to add the following into the code:

```
# Define the maximum sequence length (optional)
max_length = 512  # Set a reasonable length for your model

# Function to process the dataset by tokenizing and padding/truncating
def tokenize_function(batch):
    # Tokenize the 'text' field
    return tokenizer(batch['text'], padding="max_length", truncation=True,
max_length=max_length)

# Apply the function to the entire dataset
dataset = dataset.map(tokenize_function, batched=True)
```

Since I have got this warning:

```
/usr/local/lib/python3.10/dist-packages/trl/trainer/sft_trainer.py:300:
UserWarning: You passed a processing_class with `padding_side` not equal
to `right` to the SFTTrainer. This might lead to some unexpected behaviour
due to overflow issues when training a model in half-precision. You might
```

```
consider adding `processing_class.padding_side = 'right'` to your code.
  warnings.warn(
```

I have added the following line into the code, before defining the `trainer`:

```
tokenizer.padding_side = "right"
```

However, I am not completely sure now if `right` was the correct option for padding, or whether I could get away with no padding.
Here they used the `right`:

- Fine-Tuning Mistral

While for generation the `left` padding side is suggested:

- Generation with LLMs

A note:
If I was to run training several times, I should consider adding specific names (`name="small_run_1K"`) for training runs for better management in W&B into `wand.init(...)`, as well as:

```
training_arguments = TrainingArguments(
    output_dir="./results",
    run_name="unique_run_name",  # Add a custom name here
...
```

- [x] Send `trainer.train()` to run...
  Estimated time needed for training (1 epoch): ~ 8 hours

  UPDATE: Unfortunately, I have been cut off from colab, after it was almost done.
  [561/625 7:03:31 < 48:29, 0.02 it/s, Epoch 0.90/1]

  Verdict: Running this notebook with the available resources, without saving checkpoints outside of Colab was not a good idea... The data is lost and the time too...

  I will change the subset into 1K to check the pipeline, and retrain, I will call the model *shrimp*

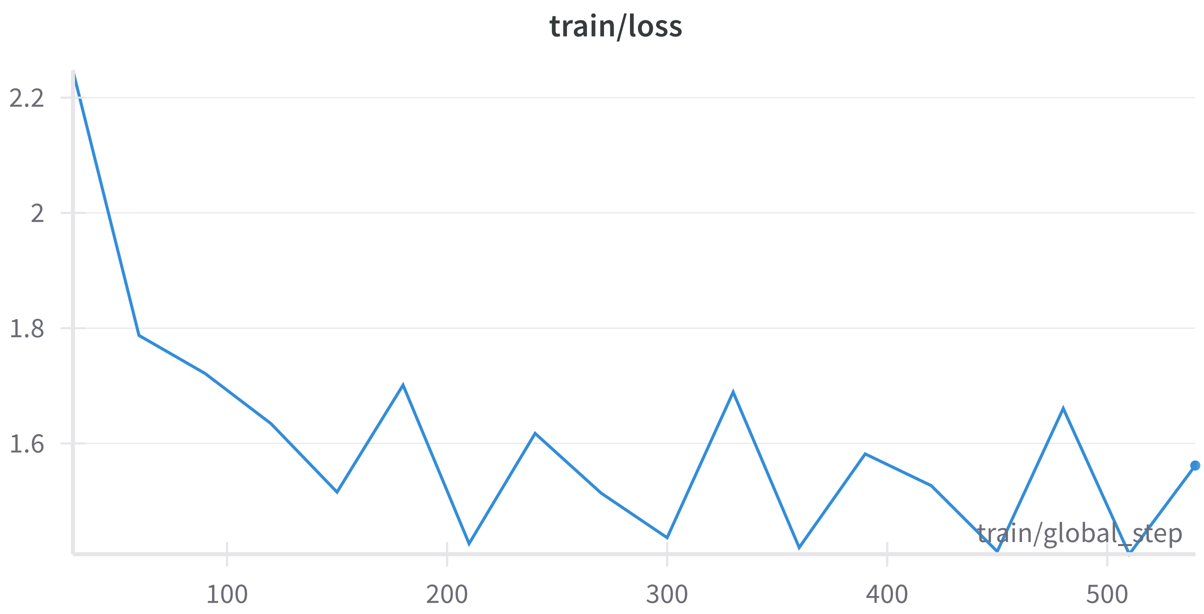
  Also I will mount the Google Drive and to save checkpoints and other data there, so I could use checkpoints to resume training if it fails during the process. If Colab fails, the environments and all the data gets cleared too.

  For that reason I have added `resume_from_checkpoint=True` and `save_total_limit=3` into `TrainingArguments`. For 1K, there should be 63 steps, so I have set up `save_steps=10`, this can be 50 for 10K datapoints (625 steps) training. I have first tested the pipeline with 100 datapoints, and then run it with 1K.

Possible alternative 1: Save the checkpoints and models to W&B, it then needs to be loaded for resuming, with a callback function as an *artifact*...
Possible alternative 2: Do the whole training somewhere outside of Colab with a SLURM script.

- ☑ Evaluate training results and loss with W&B

  For the failed **10K** run went pretty well with the loss function looking as follows.



  The training on **1K datapoints** the loss gained **1.6542** at step 60. This must be lower than in 10K since the warm-up was shorter.

  However, one need to decide what metrics/parameters to use to properly evaluate the model... This stays beyond the scope of this exercise, we somehow evaluate the results with the `stream` function, indeed while in 100 datapoints test-run the results very rather hallucinative, with 1K, although with a lot of repetitive information they are already reasonably good, but what is good depends of course on our needs...

- ☑ Save the model (Where!? Yes, in Colab environment...)

  Saving with the name `new_model` caused issues when later pushing the model, therefore I have saved it with a different path, not `new_model`.

- ☑ Loaded the base model When loading `base_model` I have set up `device_map = {"": 0}`, and implemented quantisation, by adding: `quantization_config=bnb_config` into parameters. `bnb_config` was defined earlier.

- ☑ Merged the `base_model` and `new_model` and pushed into HuggingFace.

  The new model has 3.87B parameters.

- ☑ Created a model card for this model: https://huggingface.co/nicksnlp/shrimp/blob/main/README.md

Not hasslefree, but everything worked at the end. The main issue was in adapting the code so it correctly processes paths when saving to Hugging Face, adding functionality to keep intermediate models and resume training from checkpoints, as well as adding correctly quantisation and padding. I've learned a lot of things about setting things up, and the process!

Selecting another base model and dataset

I wanted to try using T5 Google Model and a labelled QA dataset to adapt it for detecting hallucinations in texts. Unfortunately, even *flan-t5-small* has 77M parameters, so I will save this idea for the future. Ideally, I will try to use W&B and run the training on Puhti with a SLURM script.

An alternative is to use one of the llama family (they seem to be good for hallucination detection), they range from 1B to 70B in size. For example `meta-llama/Llama-2-7b-hf`. Finding a good dataset is problematic, I will need to adapt an existing one. For now I will use a sample dictionary. The code is implemented in the notebook *Fine-tune_llama.ipynb*.

More about the model:

- [Is Bigger and Deeper Always Better? Probing LLaMA Across Scales and Layers]()

I had to write a function to preprocess data. The idea is to have a dataset of a following format:
{"text": "The Eiffel Tower is located in Berlin, Germany.",
"labels": [0, 0, 0, 0, 0, 0, 1, 1]},
where labels `1` will mark hallucinated words, like "Berlin" and "Germany", and the rest will be marked as `0`

The preprocessing marks those labels to the tokenised texts, applies padding, returns an attention mask, and has a compatibility to process the dataset in batches (the one that I still have to find).

This is how the tokenised_data should look like:

```
('text', 'The Eiffel Tower is located in Berlin, Germany.') LENGTH:  47
('labels', [-100, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
-100, -100, -100, -100]) LENGTH:  128
('original_word_labels', [0, 0, 0, 0, 0, 0, 1, 1]) LENGTH:  8
('input_ids', [1, 450, 382, 2593, 295, 23615, 338, 5982, 297, 5115, 29892,
9556, 29889, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]) LENGTH:  128
('tokens', ['<s>', '_The', '_E', 'iff', 'el', '_Tower', '_is', '_located',
'_in', '_Berlin', ',', '_Germany', '.', '</s>', '</s>', '</s>', '</s>',
```

```
'</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>',
'</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>',
'</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>',
'</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>',
'</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>',
'</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>',
'</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>',
'</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>',
'</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>',
'</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>',
'</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>',
'</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>', '</s>',
'</s>', '</s>', '</s>']) LENGTH:  128
('attention_mask', [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0]) LENGTH:  128
```

For the rest of the architecture I am using the similar set-up as in the previous exercise, with quantisation.

I have loaded the model and checked its layers. There are 32 layers. I have not chaneged any target layers for low-rank adaptation.

In `peft_config` I have changed the `task_type` to `TOKEN_CLS`, which is the one needed for classification.

In the base model itself there is no layer, responsible for classification. An extra layer is added by `AutoModelForTokenClassification` with `num_labels=2`. As prescribed by Gemini, when assesing my code:

> "The classification layer is added on top of the base model, making it separate. We want to fine-tune the base model to produce good representations which are then projected to the correct number of classes by the newly added classification layer.

Leaving specified `target_modules` raises an error, which with explanation by Gemini, I decide to comment out:

> No `target_modules`: PEFT automatically selects relevant linear layers (typically attention and MLP layers) based on task_type.

But without `target_modules` it is imposible to run SFTTrainer, Trainer does not support peft_config... It is a dead end.

Unfortunately, as it looks Peft is tricky to adapt for classification task... I've found an article, I will dive into it: https://medium.com/@preeti.rana.ai/instruction-tuning-llama-2-7b-for-news-classification-1784e06441c8

Okay, finally, (thanks to Gemini 2.0 Flash). It seemed to work by reducing the target_modules to `["q_proj", "v_proj"]`. Gemini also insisted I should add collator, may be it is what made things work... I will test it later.

The code works, here is an example of Inference:

Input:

> input_text = "Alexanderplatz is located in London City, it has been there since 1966."

Output:

> Hallucinated words: ['▁Alexander', '▁in', ',', '▁has', '▁there', '▁since', '1', '6', '.'] ['Alexander', 'in', ',', 'has', 'there', 'since', '1', '6', '.']

But now I need more data.

The model is saved and pushed to Hugging Face: https://huggingface.co/nicksnlp/llama-7B-hallucination

UPDATE: The problem was in **collator**. The training worked with a larger selection of parameters, but the results are ***different***:
Hallucinated words:
['▁Alexander', 'platz', '▁is', '▁located', '▁in', '▁London', '▁City', ',', '▁it', '▁has', '▁been', '▁since', '▁', '1', '9', '6', '6', '.']
['Alexander', 'platz', 'is', 'located', 'in', 'London', 'City', ',', 'it', 'has', 'been', 'since', '', '1', '9', '6', '6', '.']

Utilising DPO instead of supervised fine-tuning

Fine_tune_a_Mistral_7b_model_with_DPO.ipynb

Since this is a bonus exercise, I will hopefully do it later on...

References:

1. https://chatgpt.com/share/677021c2-0128-800b-957b-511b29768fd4
2. https://chatgpt.com/share/67708817-1a4c-800b-a17a-c99bcdcbd05d
3. https://chatgpt.com/share/67709535-13c0-800b-b07a-2446d28e701a
4. https://chatgpt.com/share/67714f3a-390c-800b-8a6c-2da3d5c5815b
5. https://chatgpt.com/share/6771c958-592c-800b-a846-1a10425d06f0
6. https://chatgpt.com/share/67729fee-da9c-800b-808a-28a722cd3174

Week 5

References:

Week 6

References:
