ECE 157B/272B Homework 4: Text Classification

Part 2: Pre-trained Language Models

Due date: Friday, March 11th, 2022, 11:59PM

Introduction

In the last several lectures, we have seen how convolutional neural networks (CNN) can be applied to various image-based machine learning tasks. If we view the CNN as the fundamental building block for image-related applications, one question naturally comes into our mind-what is the fundamental building block for solving text-based problems?

This homework will also be done through Google Colab. It is split into 2 major parts (separate submissions).

Part 2

Part 2 will explore some more modern advances in the realm of text (or speech) processing. We will look at the transformer architecture, and take advantage of pre-trained networks to complete the same task that we trained from scratch in part 1. With this new approach, we also introduce the HuggingFace libraries Transformers and Datasets, which are regularly used platforms for transformer-based NLP today. They are built on PyTorch, so you will have to adapt away from Tensorflow a little bit.

Data Set

We will use Keras utility function: *get_file* to download the dataset from this URL and cache it on the file system allocated by the Colab session.

The dataset contains 16000 programming questions from Stack Overflow. Each question (E.g., "How do I sort a dictionary by value?") is labeled with exactly one tag (Python, CSharp, JavaScript, or Java).

Here's the code snippet for loading the dataset from URL:

```python
from tensorflow.keras import utils

data_url = \
    'https://storage.googleapis.com/download.tensorflow.org/data/stack_overflow_16k.tar.gz'
dataset = utils.get_file(
    'stack_overflow_16k.tar.gz',
    data_url,
    untar=True,
    cache_dir='',
    cache_subdir='/PATH/TO/DIRECTORY/ON/COLAB/FILESYSTEM') # Specify download directory
```

Finally, since we are using HuggingFace's interface, we will need to change to a DataSet object. This will be provided for you, since the exact method is confusing and out of scope. In english: huggingface uses lists for text input so we need to convert to a list. Tensorflow also loads the dataset as binary strings, so we need to decode them (built in for str in python).

```
def tfds_to_hugds(tfds, split):
    x, y = [], np.array([], dtype=int)
    for batchx, batchy in tfds:
        x.extend([t.numpy().decode() for t in batchx])
        y = np.append(y, batchy.numpy())
    return datasets.Dataset.from_dict({'text': x, 'labels': y}, split=split)

huggingface_data = datasets.DatasetDict(
    train=tfds_to_hugds(data_train, 'train'),
    validation=tfds_to_hugds(data_val, 'validation'),
    test=tfds_to_hugds(data_test, 'test')
)
```

Part 2: Transformers and BERT

All Students (157B/272B)

1. (3 pts) Prepare data
   - (a) (1 pt) Download dataset. This is the same code as last week.
   - (b) (1 pt) Split into train, validation, and test. This is the same code as last week.
   - (c) (1 pt) Convert to `datasets.DatasetDict` format using the given conversion function (see above).
2. (5 pts) Vectorize the text. Last week we wrote a standardization function and a TextVectorization layer from Keras. This week, we will use a Tokenizer, which was used by BERT.
   - (a) (1 pts) Use `AutoTokenizer` from `transformers` to load a BERT tokenizer. We will use `distilbert-base-uncased`.
   - (b) (2 pts) Take a training sample text, and pass it through the Tokenizer. Compare it to passing it through your TextVectorization layer. (Quickly load/adapt a TextVectorization layer, this code is the same as last week.) Describe the differences in the output. You will have to explore a bit here using built-ins like `type()` or `.keys()`.
   - (c) (2 pts) Call `tokenizer.decode()` and `tokenizer.convert_ids_to_tokens()` on `input_ids` from the tokenizer output above. What do you notice? In the "convert" output, what does “##” seem to mean?

     You may have to do `output.input_ids[0]` to unbatch it so that these functions can handle the input ids.
   - (d) (2 pts) Generate tokenized datasets from the text dataset by defining a function and using DataSet.map(). The keys here are to include truncation in case sentences are too long, and padding so that all inputs are the same length.

3. (12 pts) Conceptual break: Read this blog on transformers and self-attention. You may also read the original paper, linked in the Grad section below, if that is more helpful.

   (a) (2 pts) Explain 2 disadvantages of RNNs (LSTM) in NLP tasks. You will find this information from lecture/sources outside the blog.

   (b) Reprinted for convenience is the self-attention mechanism. The letters correspond to “Query”, “Key” and “Value” matrices. d_k is the hidden dimension size, but is not super important for us to look at right now.

$$A = softmax(\frac{QK^T}{d_k})V \tag{1}$$

   - (2 pts) Explain, in your own words , the purpose of Query/Key/Value vectors in self attention.
   - (2 pts) Explain how self-attention (eqn 1) fixes the 2 main disadvantages of LSTM.

   (c) A LSTM’s structure inherently means that current cell will only have information about past words (from the hidden state passed along). This means we get both (1) information in a causal manner, and (2) positional information by the order in which the words are processed.

   - (2 pts) Explain how transformers add position information to the input words. See **Beast #3** in the blog.
   - (3 pts) Explain how transformers make sure the decoder stack acts in a causal manner as well (We don’t want it to "look into the future" at words that haven’t been generated yet). See **Beast #4** in the blog.
   - (1 pts) BERT models are actually just the encoder portion of transformers. What do you expect their attention mask to be?

4. (6 pts) Load the BERT model and prepare for training

   (a) (2 pts) Load `distilbert-base-uncased` using `AutoModelForSequenceClassification`. Make sure to indicate how many classes!

   (b) (3 pts) Explain the difference between `AutoModelForSequenceClassification` and `AutoModel` in view of "foundation models".

   (c) (1 pts) Set `TrainingArguments`. (Some) Initial arguments are given for you, which should be adequate, just set your epochs and batch size. Be sure to adjust the epochs once you start training and see the expected time!

   (d) (2 pts) Load a Hugginface `Trainer` to use the tokenized train and validation data.

5. (5 pts) Train ("fine-tune") the BERT model!

   (a) (1 pts) Train it - `trainer.train()`. That’s it!

   (b) (2 pts) Comment on the training time compared to the LSTM. Any thoughts on why?

   (c) (2 pts) HuggingFace’s interface makes the training a one line call with no arguments instead of TF’s (define loss) –> (compile) –> (fit). Where in our process would we have set all the hyper-parameters?

6. (6 pts) Evaluate the performance.

   (a) (2 pts) Report the test set accuracy. You can get the predicted logits from `trainer.predict(...)`. How does this compare to your best LSTM accuracy?

   (b) (2 pts) Take the following questions as input, what are the predicted tags?

      i. "how do I extract keys from a dict into a list?"

      ii. "debug public static void main(string[] args) ..."

   Do you agree? A function is provided to correctly get the logits for you, since after training some of the the model will be on the gpu, but the text you make will be on the cpu.

   (c) (2 pts) What is the most annoying aspect about using/training/evaluating BERT? For our problem, is this annoyance worth the performance?

Grad/Extra Credit (272B)

1. (11 Points) Understanding the how and why of Transformers.

   (a) Read the original transformers paper.

   (b) (3 pts) Explain the main computational limitation of transformers/BERT, especially in comparison with LSTM. Be thorough.

   (c) (2 pts) Explain the purpose of $\sqrt{d_k}$ term in equation 1.

   (d) (4 pts) Instead of pure self-attention, the authors used **multi-head-attention**. Explain the purpose of this mechanism i) intuitively, and ii) in terms of computational complexity.

   (e) (2 pts) It's not the easiest to see from the paper, but their diagram shows a "add and norm" layer within each transformer layer. Explain the "add" component to the best of your understanding. (Hint - is something here similar to Recurrent networks? Residual networks?). If you are unsure, that is OK. Try to find some intuition and defend your stance.

What to Turn In

Save your Google Colab notebook as ***.ipybn*** file and submit it to Gradescope: `https://www.gradescope.com/courses/351582`.

The Colab notebook should contain the following parts for grading:

- Answers to all the questions and diagrams/plots listed in the above questions
- Code for answering those questions that can be run to reproduce those results if needed

Note: Do not close the cell outputs after running the code cells, so that all logging and plots will be saved in the notebook for grading.

Good Luck!
ECE 157B TAs