CS 449 Final Project Proposal

Due: November 26, 2024 at 11:59pm

1 Names

Murat Sankaya, Qingyaun (Jason) Yao

2 Abstract

Our final project seeks to recreate the Transformer model detailed in the "Attention is All You Need" paper using PyTorch from scratch. After completing the implementation, we will use our model on one of the machine translation tasks outlined in the paper (either English-to-German or English-to-French).

3 Big Changes

We initially hoped to develop the Transformer model as described in the paper and then extend the model with encoder-only and decoder-only architectures to test them on aligned problems like classification and generation, respectively. However, since development took longer than expected, we are now planning to just apply our model to a machine translation task.

We also hope to try a decoder-only model architecture (like GPT), mainly due to personal interest, but due to our inexperience, we cannot predict how difficult this will be. We will likely have to continue that work on our own after the quarter ends.

4 Data

4.1 Describe your dataset(s)

We currently have two datasets we are certain we will try the model on:

1. A small **synthetic dataset**, where the model learns to count `1` and `2` and determines which one occurs more frequently in a sequence of tokens: `0`, `1`, and `2`.
2. The **WMT 2014 English-to-German Dataset**. While this dataset contains around 4.5 million examples, we may train on a subset of it.

We are also considering testing on a dataset that is more complicated than the counting dataset but less complex than the machine translation dataset, to further evaluate our model's performance.

4.2 Load your dataset(s)

```
!pip install datasets

from datasets import load_dataset

dataset = load_dataset("wmt14", "de-en")  # WMT-14 English-German
train_data = dataset["train"]
valid_data = dataset["validation"]
test_data = dataset["test"]

train_data = dataset["train"].select(range(10))  # First 10 samples
```

4.3 Small dataset

```
def make_onehot(arr, vocab_size):
   # Convert to one-hot representation
   # https://en.wikipedia.org/wiki/One-hot

   n, max_length = arr.shape
   onehot_data = np.zeros([n, max_length, vocab_size])
   for v in range(vocab_size):
       onehot_row = np.zeros([vocab_size])
       onehot_row[v] = 1
       onehot_data[arr == v] = onehot_row

   return onehot_data


class CountingDataset(torch.utils.data.Dataset):
   def __init__(self, n, max_length=8, vocab_size=8):

       assert vocab_size > 2
       self.n = n
       self.vocab_size = vocab_size
       seq_lengths = np.random.randint(max_length // 2, max_length, n)
       data = np.random.randint(0, vocab_size, [n, max_length])
```

```
        # Replace elements past the sequence length with -1
        for i in range(n):
            data[i, slice(seq_lengths[i] + 1, None)] = -1

        onehot_data = make_onehot(data, vocab_size)

        # Label is whether ones outnumber twos in the sequence
        num_ones = (data == 1).sum(axis=1, keepdims=True)
        num_twos = (data == 2).sum(axis=1, keepdims=True)
        label = (num_ones > num_twos).astype(int)

        self.data = torch.tensor(onehot_data).float()
        self.label = torch.tensor(label).long()

    def __len__(self):
        return self.n

    def __getitem__(self, item_index):
        """
        Allow us to select items with `dataset[0]`
        Returns (x, y)
            x: the data tensor
            y: the label tensor
        """
        return self.data[item_index], self.label[item_index]

d = CountingDataset(3, max_length=8, vocab_size=3)
d[0]
```

5 Methods

For the German-English machine translation task, we will use the Transformer model that we developed, along with the cross-entropy loss function to calculate the information loss in the system. Since the model outputs a probability distribution over the vocabulary, we want it to more confidently predict the next token that matches the ground truth in the training data.

We will try to align our training setup as closely as possible with the setup described in the paper. However, given that we likely won't have the same level of computational resources, we will make some adjustments to optimize for our case.

6 Old Deliverables

6.1 Essential Goals

- **EG-Goal 1: Develop the code for transformer model introduced in Vaswani et al. (2017)**
  - We are close to finishing this. We will be including the embedding module from `torch.nn`. Developing the entire model without relying on external resources was more challenging than we initially thought. Some nuances only became apparent during implementation.
  - We also need to run more experiments to ensure all parameters are visible within PyTorch's gradient graph, that backpropagation works correctly, and that the optimizer updates everything as expected. Additionally, we need to verify there are no inefficiencies, since we plan to train on a significant dataset.
- **EG-Goal 2: Test the transformer model on at least one nontrivial(high complexity) dataset.**
  - Once we are confident in our model and have validated it on at least one toy problem, we aim to start this within a week.

6.2 Desired Goals

- **DG-Goal 1: Submit a well-documented Jupyter notebook that could serve as a tutorial for someone with limited knowledge.**
  - After finishing the essential goals, we believe this will be straightforward. By then, we should already have a solid understanding of the Transformer's inner workings.

- **DG-Goal 2: Include code for visualizations and demos in the Jupyter notebook, such as visualizing how the attention mechanism works on a sequence of text.**
  - As with DG-Goal 1, this should be easier after the essential goals are completed. While we might not create a super detailed notebook, we plan to include at least a few visuals and documentation

6.3 Stretch Goals

- **SG-Goal 1: Extend the code to different architectures.**
- **SG-Goal 1.1: Implement a decoder-only version (Radford et al., 2019) and test it on a relevant dataset, the WikiText-2.**
- **SG-Goal 1.2: Implement an encoder-only version (Devlin et al., 2018) and test it on a relevantdataset, the GLUE Benchmark**
- **SG-Goal 1.3: Explore the application of the transformer model on different types of data beyond sequenced text, if time permits.**

        - We are unfortunately abandoning these goals. Although all of them are highly interesting and excite us, we don’t think we will have enough time to get to them.

7 Results So Far

We unfortunately do not have any empirical data on a trained model. We created 28 unit tests that are all currently passing.

They can be run the same way the tests on out HWs as:

```
python -m pytest -s -k  test_<test prefix>

python -m pytest # runs them all
```

8 New Deliverables

8.1 Stretch Goals

1. We will use our model to perform inference. While it is difficult to predict how well it will produce translations, we will give it a try.
2. Depending on the computational resources provided to us, we may attempt to train on the full English-to-German dataset.

9 Hopes and Concerns

So far, we are proud to have developed the entire Transformer architecture from scratch using PyTorch. We are especially happy we managed this without referring to external resources when stuck.

However, there are some concerns:

1. **Correctness:** We cannot yet be certain that our implementation is completely correct.
2. **Performance:** We are unsure of its efficiency or optimality.

10 References

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *30*, 5998–6008.

11 Guidance on Feedback

We would appreciate feedback on the following:

1. **MultiHead Attention Implementation:**
    - Does the implementation of MultiHead attention look correct? This was the trickiest part to implement.
2. **Use of PyTorch Patterns:**
    - Are we making proper use of PyTorch's systematic patterns? For example, are all weights visible to the autograd system?
3. **Bottlenecks:**
    - Are there any observable bottlenecks in our implementation in terms of acceleration or performance?
4. **Custom Functions:**
    - We have written some functions outside of `torch.nn.Module`. We believe these are not critical to any core PyTorch features, such as autograd. Are our assumptions here correct?