11711 ANLP Assignment 1 Report

Yiming Fu (Andrew ID: yimingfu)

Summary

In this assignment, I have chosen the second provided option for A+ improvement, which is

- Enable zero-shot prompting using a more principled inference algorithm than our current implementation. For example, we did not include an attention mask despite right-padding all inputs (to enable batch prediction); this could be improved.

Specifically, I have realized **padding mask** and **causal mask** for our Llama model, and achieved significant accuracy improvements in the **zero-shot prompting** setting.

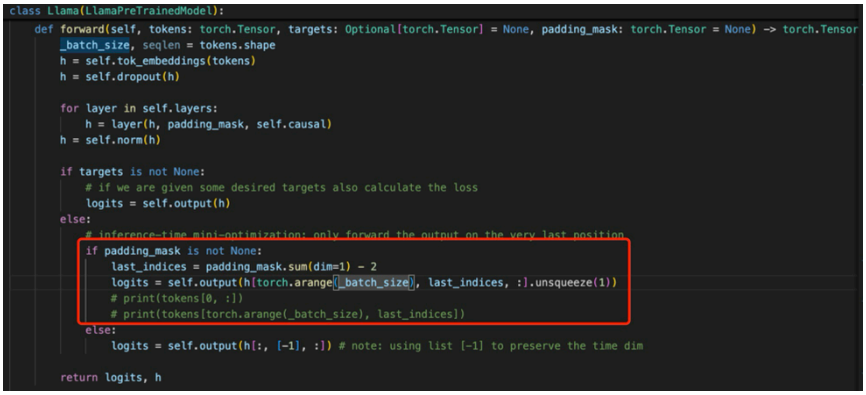
Note: The padding mask and causal mask mechanism is added to the Llama model's forward function. But there is no training process in the zero-shot prompting setting, so the newly-added mechanism only affects the inference process.

Code Explanation

In `llama.py`, we add the padding mask and causal mask for "compute_query_key_value_scores" function in the "Attention" class, which includes these two masks in attention calculation.

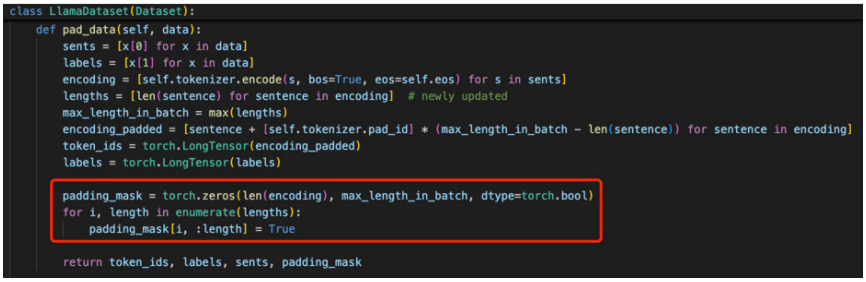
```python
class Attention(nn.Module):
    def compute_query_key_value_scores(self,
                                       query: torch.Tensor,
                                       key: torch.Tensor,
                                       value: torch.Tensor,
                                       padding_mask: torch.Tensor = None,
                                       causal: bool = False) -> torch.Tensor:
        '''
        Jointly compute Scaled Dot Product Attention (see Section 3.2.1 in
        https://arxiv.org/abs/1706.03762 for details). The query, key, and
        value tensors each have shape (bs, n_local_heads, seqlen, head_dim).
        An optimal implemention will jointly computing attention for multiple
        heads (n_local_heads of them) at once using matrix/tensor operations.

        Make sure to use attention_dropout (self.attn_dropout) on the computed
        attention matrix before applying it to the value tensor.
        '''
        # todo
        # raise NotImplementedError
        scores = torch.matmul(query, key.transpose(-2, -1)) / math.sqrt(self.head_dim)

        if causal:
            seq_len = query.shape[2]
            causal_mask = torch.tril(torch.ones(seq_len, seq_len, device=query.device))
            causal_mask = causal_mask[None, None, :, :]
            scores = scores.masked_fill(causal_mask == 0, float('-inf'))

        if padding_mask is not None:
            padding_mask = get_extended_attention_mask(padding_mask, query.dtype)
            scores = scores + padding_mask

        attn_weight = F.softmax(scores, dim=-1)
        attn_output = torch.matmul(self.attn_dropout(attn_weight), value)  # (bs, n_local_heads, seqlen, head_dim)
        return attn_output
```

Then in the "Llama" class, we modify the logic to fetch the last **effective** token, which is neither padding nor null. (The token before the first padding is null, so we select the token before it.)



```python
class Llama(LlamaPreTrainedModel):
    def forward(self, tokens: torch.Tensor, targets: Optional[torch.Tensor] = None, padding_mask: torch.Tensor = None) -> torch.Tensor
        _batch_size, seqlen = tokens.shape
        h = self.tok_embeddings(tokens)
        h = self.dropout(h)

        for layer in self.layers:
            h = layer(h, padding_mask, self.causal)
        h = self.norm(h)

        if targets is not None:
            # if we are given some desired targets also calculate the loss
            logits = self.output(h)
        else:
            # inference-time mini-optimization: only forward the output on the very last position
            if padding_mask is not None:
                last_indices = padding_mask.sum(dim=1) - 2
                logits = self.output(h[torch.arange(_batch_size), last_indices, :].unsqueeze(1))
                # print(tokens[0, :])
                # print(tokens[torch.arange(_batch_size), last_indices])
            else:
                logits = self.output(h[:, [-1], :]) # note: using list [-1] to preserve the time dim

        return logits, h
```

Finally, we create a `run_llama_new.py` file, which is quite similar to `run_llama.py`. Here we need to return padding masks in "LlamaDataset", and modify corresponding train & eval procedures.



```python
class LlamaDataset(Dataset):
    def pad_data(self, data):
        sents = [x[0] for x in data]
        labels = [x[1] for x in data]
        encoding = [self.tokenizer.encode(s, bos=True, eos=self.eos) for s in sents]
        lengths = [len(sentence) for sentence in encoding]  # newly updated
        max_length_in_batch = max(lengths)
        encoding_padded = [sentence + [self.tokenizer.pad_id] * (max_length_in_batch - len(sentence)) for sentence in encoding]
        token_ids = torch.LongTensor(encoding_padded)
        labels = torch.LongTensor(labels)

        padding_mask = torch.zeros(len(encoding), max_length_in_batch, dtype=torch.bool)
        for i, length in enumerate(lengths):
            padding_mask[i, :length] = True

        return token_ids, labels, sents, padding_mask
```

Result Display

SST Zero-Shot Prompting

The advanced result with padding mask is shown below.

```
/content/CMU-11711-HW1# python run_llama_new.py --opti
on prompt --batch_size 10 --train data/sst-train.txt -
-dev data/sst-dev.txt --test data/sst-test.txt --label
-names data/sst-label-mapping.json --dev_out sst-dev-a
dvanced-output.txt --test_out sst-test-advanced-output
.txt --use_gpu
args: {'train': 'data/sst-train.txt', 'dev': 'data/sst
-dev.txt', 'test': 'data/sst-test.txt', 'label_names':
 'data/sst-label-mapping.json', 'pretrained_model_path
': 'stories42M.pt', 'max_sentence_len': None, 'seed':
1337, 'epochs': 5, 'option': 'prompt', 'use_gpu': True
, 'generated_sentence_low_temp_out': 'generated-senten
ce-temp-0.txt', 'generated_sentence_high_temp_out': 'g
enerated-sentence-temp-1.txt', 'dev_out': 'sst-dev-adv
anced-output.txt', 'test_out': 'sst-test-advanced-outp
ut.txt', 'lora_rank': 4, 'lora_alpha': 1.0, 'batch_siz
e': 10, 'hidden_dropout_prob': 0.3, 'lr': 2e-05}
load 8544 data from data/sst-train.txt
load 1101 data from data/sst-dev.txt
load 2210 data from data/sst-test.txt
eval: 100%|██████████| 111/111 [00:02<00:00, 48.06it/s]
eval: 100%|██████████| 221/221 [00:03<00:00, 57.55it/s]
dev acc :: 0.278
test acc :: 0.297
```

Accuracies on both dev and test dataset have increased significantly. (0.237/0.250 before)

CFIMDB Zero-Shot Prompting

The advanced result with padding mask is shown below.

```
/content/CMU-11711-HW1# python run_llama_new.py --opti
on prompt --batch_size 10 --train data/cfimdb-train.tx
t --dev data/cfimdb-dev.txt --test data/cfimdb-test.tx
t --label-names data/cfimdb-label-mapping.json --dev_o
ut cfimdb-dev-advanced-output.txt --test_out cfimdb-te
st-advanced-output.txt --use_gpu
args: {'train': 'data/cfimdb-train.txt', 'dev': 'data/
cfimdb-dev.txt', 'test': 'data/cfimdb-test.txt', 'labe
l_names': 'data/cfimdb-label-mapping.json', 'pretraine
d_model_path': 'stories42M.pt', 'max_sentence_len': No
ne, 'seed': 1337, 'epochs': 5, 'option': 'prompt', 'us
e_gpu': True, 'generated_sentence_low_temp_out': 'gene
rated-sentence-temp-0.txt', 'generated_sentence_high_t
emp_out': 'generated-sentence-temp-1.txt', 'dev_out':
'cfimdb-dev-advanced-output.txt', 'test_out': 'cfimdb-
test-advanced-output.txt', 'lora_rank': 4, 'lora_alpha
': 1.0, 'batch_size': 10, 'hidden_dropout_prob': 0.3,
'lr': 2e-05}
load 1707 data from data/cfimdb-train.txt
load 245 data from data/cfimdb-dev.txt
load 488 data from data/cfimdb-test.txt
eval: 100%|██████████| 25/25 [00:02<00:00,  8.53it/s]
eval: 100%|██████████| 49/49 [00:05<00:00,  9.48it/s]
dev acc :: 0.584
test acc :: 0.779
```

Accuracies on both dev and test dataset have increased significantly. (0.490/0.109 before)

Executing Instructions

For **SST Zero-Shot Prompting**, run the following shell script to execute `run_llama_new.py`.

```
python run_llama_new.py --option prompt --batch_size 10 --train data/sst-train.txt --dev
data/sst-dev.txt --test data/sst-test.txt --label-names data/sst-label-mapping.json --dev_out
sst-dev-advanced-output.txt --test_out sst-test-advanced-output.txt --use_gpu
```

For **CFIMDB Zero-Shot Prompting**, run the following shell script to execute `run_llama_new.py`.

```
python run_llama_new.py --option prompt --batch_size 10 --train data/cfimdb-train.txt --dev
data/cfimdb-dev.txt --test data/cfimdb-test.txt --label-names data/cfimdb-label-mapping.json
--dev_out cfimdb-dev-advanced-output.txt --test_out cfimdb-test-advanced-output.txt --use_gpu
```