LSTM Language Models

You guys probably very excited about ChatGPT. In today class, we will be implementing a very simple language model, which is basically what ChatGPT is, but with a simple LSTM. You will be surprised that it is not so difficult at all.

Paper that we base on is *Regularizing and Optimizing LSTM Language Models*, https://arxiv.org/abs/1708.02182

In [17]:
```
import torch
import torch.nn as nn
import torch.optim as optim

import torchtext, datasets, math
from tqdm import tqdm
```

In [18]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "mps" if torch.backends.mps.is_available() else "cpu")
print(device)
```

```
mps
```

In [19]:
```
SEED = 1234
torch.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
```

1. Load data - Wiki Text

We will be using wikitext which contains a large corpus of text, perfect for language modeling task. This time, we will use the `datasets` library from HuggingFace to load.

In [20]:
```
# dataset = datasets.load_dataset("wikitext", "wikitext-2-raw-v1")
# dataset = datasets.load_dataset("monology/pile-uncopyrighted")
# Load a small subset of the dataset
# dataset = datasets.load_dataset("monology/pile-uncopyrighted", split="train[:1%]")

# Split the dataset into train, validation, and test sets
# train_test_split = dataset.train_test_split(test_size=0.2, seed=SEED)
# test_valid_split = train_test_split["test"].train_test_split(test_size=0.5, seed=SEED)

# dataset = {"train": train_test_split["train"], "validation": test_valid_split["train"], "test": test_valid_split["test"]}
```

In [21]:
```
# https://huggingface.co/datasets/allenai/c4
dataset = datasets.load_dataset("allenai/c4", "realnewslike")
print(dataset)
```

```
Resolving data files:   0%|          | 0/1024 [00:00<?, ?it/s]
Resolving data files:   0%|          | 0/512 [00:00<?, ?it/s]
Loading dataset shards:   0%|          | 0/76 [00:00<?, ?it/s]
DatasetDict({
    train: Dataset({
        features: ['text', 'timestamp', 'url'],
        num_rows: 13799838
    })
    validation: Dataset({
        features: ['text', 'timestamp', 'url'],
        num_rows: 13863
    })
})
```

In [22]:
```
train_test_split = dataset["train"].train_test_split(test_size=0.2, seed=SEED)
train_val_split = train_test_split["train"].train_test_split(test_size=0.2, seed=SEED)
dataset["train"] = train_val_split["train"]
dataset["validation"] = train_val_split["test"]
dataset["test"] = train_test_split["test"]
print(dataset)
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'timestamp', 'url'],
        num_rows: 8831896
    })
    validation: Dataset({
        features: ['text', 'timestamp', 'url'],
        num_rows: 2207974
    })
    test: Dataset({
        features: ['text', 'timestamp', 'url'],
        num_rows: 2759968
    })
})
```

In [23]:
```
# Reduce the sample size of train, test, and validation sets
dataset["train"] = dataset["train"].select(range(20000))
dataset["validation"] = dataset["validation"].select(range(2000))
dataset["test"] = dataset["test"].select(range(2000))
```

In [24]:
```
print(f'{dataset["train"].shape=}')
print(f"{dataset['validation'].shape=}")
print(f"{dataset['test'].shape=}")
```

```
dataset["train"].shape=(20000, 3)
dataset['validation'].shape=(2000, 3)
dataset['test'].shape=(2000, 3)
```

2. Preprocessing

Tokenizing

Simply tokenize the given text to tokens.

In [25]:
```
# tokenizer = torchtext.data.utils.get_tokenizer("basic_english")

# tokenize_data = lambda example, tokenizer: {"tokens": tokenizer(example["text"])}

# tokenized_dataset = dataset.map(tokenize_data, remove_columns=["text"], fn_kwargs={"tokenizer": tokenizer})
```

In [26]:
```
tokenizer = torchtext.data.utils.get_tokenizer("basic_english")
```

In [27]:
```
def tokenize_dataset(dataset, tokenizer):
    def tokenize_data(example):
        return {"tokens": tokenizer(example["text"])}

    tokenized_dataset = {split: data.map(tokenize_data, remove_columns=["text"]) for split, data in dataset.items()}
    return tokenized_dataset


tokenized_dataset = tokenize_dataset(dataset, tokenizer)
```

```
Map:   0%|          | 0/20000 [00:00<?, ? examples/s]
Map:   0%|          | 0/2000 [00:00<?, ? examples/s]
Map:   0%|          | 0/2000 [00:00<?, ? examples/s]
```

In [28]:
```
print(tokenized_dataset)
```

```
{'train': Dataset({
    features: ['timestamp', 'url', 'tokens'],
    num_rows: 20000
}), 'validation': Dataset({
    features: ['timestamp', 'url', 'tokens'],
    num_rows: 2000
}), 'test': Dataset({
    features: ['timestamp', 'url', 'tokens'],
    num_rows: 2000
})}
```

In [29]:
```
print(tokenized_dataset["train"][100]["tokens"])
```

```
['several', 'signals', 'of', 'progress', 'in', 'washington', 'pushed', 'stocks', 'higher', 'on', 'monday', '.', 'there', 'are', 'just', 'three', 'days', 'before', 'the', 'us', 'hits', 'the', 'debt', 'ceiling', 'and', 'investors', 'are', 'betting', 'that', 'a', 'deal', 'will', 'soon', 'be', 'reached', '.', 'netflix', '(', 'nflx', ')', 'was', 'trending', 'today', 'on', 'word', 'that', 'the', 'company', 'may', 'be', 'negotiating', 'a', 'cable', 'deal', 'with', 'major', 'operators', 'in', 'the', 'us', '.', 'macy', "'", 's', '(', 'm', ')', 'announced', 'that', 'they', 'will', 'be', 'opening', 'on', 'thanksgiving', 'for', 'the', 'first', 'time', '.', 'expedia', '(', 'expe', ')', 'fell', 'almost', '7%', 'on', 'a', 'recent', 'stock', 'downgrade', '.', 'many', 'are', 'looking', 'ahead', 'to', 'quarterly', 'reports', 'from', 'citigroup', '(', 'c', ')', ',', 'coca-cola', '(', 'ko', ')', 'and', 'intel', '(', 'intc', ')', ',', 'all', 'scheduled', 'for', 'tuesday', '.', 'the', 'chinese', 'yuan', 'hit', 'a', 'new', 'record', 'high', 'against', 'the', 'dollar', 'while', 'the', 'us', 'currency', 'reached', 'an', 'intraday', 'high', 'against', 'the', 'japanese', 'yen', '.', 'progress', 'possible', 'in', 'senate', ',', 'leaders', 'optimistic', 'that', 'a', 'debt', 'deal', 'may', 'be', 'reached', '.', 'shares', 'of', 'netflix', '(', 'nflx', ')', 'jump', '5%', 'on', 'talks', 'of', 'us', 'cable', 'options', '.', 'chinese', 'inflation', 'reaches', 'a', 'seven-month', 'high', 'on', 'higher', 'food', 'prices', 'and', 'drop', 'in', 'exports', '.', 'apple', '(', 'aapl', ')', 'notes', 'that', 'iphone', '5c', 'inventory', 'is', 'moving', ',', 'and', '5s', 'supply', 'is', 'tight', '.', 'eurozone', 'industrial', 'output', 'rebounds', 'and', 'spurs', 'hope', 'for', 'higher', 'economic', 'growth', '.', 'gilead', 'sciences', '(', 'gild', ')', 'rallies', 'after', 'ending', 'a', 'late-stage', 'trial', 'ahead', 'of', 'schedule', '.', 'the', 'government', 'shutdown', '-', 'is', 'there', 'a', 'simple', 'solution', 'that', "'", 's', 'being', 'overlooked', '?', 'why', 'this', 'finance', 'course', 'should', 'be', '(', 'and', 'now', 'is', ')', 'available', 'to', 'the', 'masses', '.', 'budget', 'deal', 'hopes', 'push', 'markets', 'higher', 'on', 'monday', '(', 'nflx', ',', 'aapl', ')', '.', 'investorguide', '.', 'com', '.', 'webfinance', ',', 'inc', '.', 'http', '//www', '.', 'investorguide', '.', 'com/article/14086/budget-deal-hopes-push-markets-higher-on-monday-nflx-aapl-igd/', '(', 'access', 'april', '24', ',', '2019', ')', '.']
```

Numericalizing

We will tell torchtext to add any word that has occurred at least three times in the dataset to the vocabulary because otherwise it would be too big. Also we shall make sure to add `unk` and `eos`.

In [30]:
```
vocab = torchtext.vocab.build_vocab_from_iterator(tokenized_dataset["train"]["tokens"], min_freq=3)
vocab.insert_token("<unk>", 0)
vocab.insert_token("<eos>", 1)
vocab.set_default_index(vocab["<unk>"])
```

In [31]:
```
print(len(vocab))
```

```
74619
```

In [32]:
```
print(vocab.get_itos()[:10])
```

```
['<unk>', '<eos>', 'the', '.', ',', 'to', 'and', 'of', 'a', 'in']
```

3. Prepare the batch loader

Prepare data

Given "Chaky loves eating at AIT", and "I really love deep learning", and given batch size = 3, we will get three batches of data "Chaky loves eating at", "AIT `<eos>` I really", "love deep learning `<eos>`".

In [33]:
```
def get_data(dataset, vocab, batch_size):
    data = []
    for example in dataset:
        if example["tokens"]:
            tokens = example["tokens"].append("<eos>")
            tokens = [vocab[token] for token in example["tokens"]]
            data.extend(tokens)
    data = torch.LongTensor(data)
    num_batches = data.shape[0] // batch_size
    data = data[: num_batches * batch_size]
    data = data.view(batch_size, num_batches)  # view vs. reshape (whether data is contiguous)
    return data  # [batch size, seq len]
```

In [34]:
```
batch_size = 64
train_data = get_data(tokenized_dataset["train"], vocab, batch_size)
valid_data = get_data(tokenized_dataset["validation"], vocab, batch_size)
test_data = get_data(tokenized_dataset["test"], vocab, batch_size)
```

In [35]:
```
print(f"{train_data.shape=}")
print(f"{valid_data.shape=}")
print(f"{test_data.shape=}")
```

```
train_data.shape=torch.Size([64, 154421])
valid_data.shape=torch.Size([64, 15844])
test_data.shape=torch.Size([64, 15723])
```

4. Modeling

In [36]:
```
class LSTMLanguageModel(nn.Module):
    def __init__(self, vocab_size, emb_dim, hid_dim, num_layers, dropout_rate):
        super().__init__()
        self.num_layers = num_layers
        self.hid_dim = hid_dim
        self.emb_dim = emb_dim

        self.embedding = nn.Embedding(vocab_size, emb_dim)
        self.lstm = nn.LSTM(emb_dim, hid_dim, num_layers=num_layers, dropout=dropout_rate, batch_first=True)
        self.dropout = nn.Dropout(dropout_rate)
        self.fc = nn.Linear(hid_dim, vocab_size)

        self.init_weights()

    def init_weights(self):
        init_range_emb = 0.1
        init_range_other = 1 / math.sqrt(self.hid_dim)
        self.embedding.weight.data.uniform_(-init_range_emb, init_range_other)
        self.fc.weight.data.uniform_(-init_range_other, init_range_other)
        self.fc.bias.data.zero_()
        for i in range(self.num_layers):
            self.lstm.all_weights[i][0] = torch.FloatTensor(self.emb_dim, self.hid_dim).uniform_(-init_range_other, init_range_other)  # We
            self.lstm.all_weights[i][1] = torch.FloatTensor(self.hid_dim, self.hid_dim).uniform_(-init_range_other, init_range_other)  # Wh

    def init_hidden(self, batch_size, device):
        hidden = torch.zeros(self.num_layers, batch_size, self.hid_dim).to(device)
        cell = torch.zeros(self.num_layers, batch_size, self.hid_dim).to(device)
        return hidden, cell

    def detach_hidden(self, hidden):
        hidden, cell = hidden
        hidden = hidden.detach()  # not to be used for gradient computation
        cell = cell.detach()
        return hidden, cell

    def forward(self, src, hidden):
        # src: [batch_size, seq len]
        embedding = self.dropout(self.embedding(src))  # harry potter is
        # embedding: [batch-size, seq len, emb dim]
        output, hidden = self.lstm(embedding, hidden)
        # ouput: [batch size, seq len, hid dim]
        # hidden: [num_layers * direction, seq len, hid_dim]
        output = self.dropout(output)
        prediction = self.fc(output)
        # prediction: [batch_size, seq_len, vocab_size]
        return prediction, hidden
```

5. Training

Follows very basic procedure. One note is that some of the sequences that will be fed to the model may involve parts from different sequences in the original dataset or be a subset of one (depending on the decoding length). For this reason we will reset the hidden state every epoch, this is like assuming that the next batch of sequences is probably always a follow up on the previous in the original dataset.

In [37]:
```
vocab_size = len(vocab)
emb_dim = 1024  # 400 in the paper
hid_dim = 1024  # 1150 in the paper
num_layers = 2  # 3 in the paper
dropout_rate = 0.65
lr = 1e-3
```

In [38]:
```
model = LSTMLanguageModel(vocab_size, emb_dim, hid_dim, num_layers, dropout_rate).to(device)
optimizer = optim.Adam(model.parameters(), lr=lr)
criterion = nn.CrossEntropyLoss()
num_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
print(f"The model has {num_params:,} trainable parameters")
```

```
The model has 169,687,931 trainable parameters
```

In [39]:
```
def get_batch(data, seq_len, idx):
    # data #[batch size, bunch of tokens]
    src = data[:, idx : idx + seq_len]
    target = data[:, idx + 1 : idx + seq_len + 1]  # target simply is ahead of src by 1
    return src, target
```

In [40]:
```
def train(model, data, optimizer, criterion, batch_size, seq_len, clip, device):
    epoch_loss = 0
    model.train()
    # drop all batches that are not a multiple of seq_len
    # data #[batch size, seq len]
    num_batches = data.shape[-1]
    data = data[:, : num_batches - (num_batches - 1) % seq_len]  # we need to -1 because we start at 0
    num_batches = data.shape[-1]

    # reset the hidden every epoch
    hidden = model.init_hidden(batch_size, device)

    for idx in tqdm(range(0, num_batches - 1, seq_len), desc="Training: ", leave=True):
        optimizer.zero_grad()

        # hidden does not need to be in the computational graph for efficiency
        hidden = model.detach_hidden(hidden)

        src, target = get_batch(data, seq_len, idx)  # src, target: [batch size, seq len]
        src, target = src.to(device), target.to(device)
        batch_size = src.shape[0]
        prediction, hidden = model(src, hidden)

        # need to reshape because criterion expects pred to be 2d and target to be 1d
        prediction = prediction.reshape(batch_size * seq_len, -1)  # prediction: [batch size * seq len, vocab size]
        target = target.reshape(-1)
        loss = criterion(prediction, target)

        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), clip)
        optimizer.step()
        epoch_loss += loss.item() * seq_len
    return epoch_loss / num_batches
```

In [41]:
```
def evaluate(model, data, criterion, batch_size, seq_len, device):
    epoch_loss = 0
    model.eval()
    num_batches = data.shape[-1]
    data = data[:, : num_batches - (num_batches - 1) % seq_len]
    num_batches = data.shape[-1]

    hidden = model.init_hidden(batch_size, device)

    with torch.no_grad():
        for idx in range(0, num_batches - 1, seq_len):
            hidden = model.detach_hidden(hidden)
            src, target = get_batch(data, seq_len, idx)
            src, target = src.to(device), target.to(device)
            batch_size = src.shape[0]

            prediction, hidden = model(src, hidden)
            prediction = prediction.reshape(batch_size * seq_len, -1)
            target = target.reshape(-1)

            loss = criterion(prediction, target)
            epoch_loss += loss.item() * seq_len
    return epoch_loss / num_batches
```

Here we will be using a `ReduceLROnPlateau` learning scheduler which decreases the learning rate by a factor, if the loss don't improve by a certain epoch.

In [42]:
```
n_epochs = 1
seq_len = 50  # <----decoding length
clip = 0.25

lr_scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, factor=0.5, patience=0)

best_valid_loss = float("inf")

for epoch in range(n_epochs):
    train_loss = train(model, train_data, optimizer, criterion, batch_size, seq_len, clip, device)
    valid_loss = evaluate(model, valid_data, criterion, batch_size, seq_len, device)

    lr_scheduler.step(valid_loss)

    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        # torch.save(model.state_dict(), "best-val-lstm_lm.pt")
        torch.save(
            {
                "model_state_dict": model.state_dict(),
                "vocab_size": vocab_size,
                "emb_dim": emb_dim,
                "hid_dim": hid_dim,
                "num_layers": num_layers,
                "dropout_rate": dropout_rate,
            },
            "best-val-lstm_lm.pt",
        )

    print(f"\tTrain Perplexity: {math.exp(train_loss):.3f}")
    print(f"\tValid Perplexity: {math.exp(valid_loss):.3f}")

    torch.save(
        {
            "model_state_dict": model.state_dict(),
            "vocab": vocab,
            "vocab_size": vocab_size,
            "emb_dim": emb_dim,
            "hid_dim": hid_dim,
            "num_layers": num_layers,
            "dropout_rate": dropout_rate,
        },
```

```
        "best-val-lstm_lm.pt",
    )
```

```
Training: 100%|██████████| 3088/3088 [26:11<00:00,  1.96it/s]
        Train Perplexity: 630.561
        Valid Perplexity: 344.386
```

6. Testing

In [43]:
```
checkpoint = torch.load("best-val-lstm_lm.pt", weights_only=False, map_location=device)
model = LSTMLanguageModel(
    checkpoint["vocab_size"], checkpoint["emb_dim"], checkpoint["hid_dim"], checkpoint["num_layers"], checkpoint["dropout_rate"]
)
model.load_state_dict(checkpoint["model_state_dict"])
model.to(device)
model.eval()

vocab = checkpoint["vocab"]
print(vocab.get_itos()[:10])
```

```
['<unk>', '<eos>', 'the', '.', ',', 'to', 'and', 'of', 'a', 'in']
```

In [44]:
```
test_loss = evaluate(model, test_data, criterion, batch_size, seq_len, device)
print(f"Test Perplexity: {math.exp(test_loss):.3f}")
```

```
Test Perplexity: 339.976
```

7. Real-world inference

Here we take the prompt, tokenize, encode and feed it into the model to get the predictions. We then apply softmax while specifying that we want the output due to the last word in the sequence which represents the prediction for the next word. We divide the logits by a temperature value to alter the model's confidence by adjusting the softmax probability distribution.

Once we have the Softmax distribution, we randomly sample it to make our prediction on the next word. If we get then we give that another try. Once we get we stop predicting.

We decode the prediction back to strings last lines.

In [45]:
```
def generate(prompt, max_seq_len, temperature, model, tokenizer, vocab, device, seed=None):
    if seed is not None:
        torch.manual_seed(seed)
    model.eval()
    tokens = tokenizer(prompt)
    indices = [vocab[t] for t in tokens]
    batch_size = 1
    hidden = model.init_hidden(batch_size, device)
    with torch.no_grad():
        for i in range(max_seq_len):
            src = torch.LongTensor([indices]).to(device)
            prediction, hidden = model(src, hidden)

            # prediction: [batch size, seq len, vocab size]
            # prediction[:, -1]: [batch size, vocab size] #probability of last vocab

            probs = torch.softmax(prediction[:, -1] / temperature, dim=-1)
            prediction = torch.multinomial(probs, num_samples=1).item()

            while prediction == vocab["<unk>"]:  # if it is unk, we sample again
                prediction = torch.multinomial(probs, num_samples=1).item()

            if prediction == vocab["<eos>"]:  # if it is eos, we stop
                break

            indices.append(prediction)  # autoregressive, thus output becomes input

    itos = vocab.get_itos()
    tokens = [itos[i] for i in indices]
    return tokens
```

In [47]:
```
prompt = "Donald Trump has"
max_seq_len = 30
seed = 0

# smaller the temperature, more diverse tokens but comes
# with a tradeoff of less-make-sense sentence
temperatures = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
for temperature in temperatures:
    generation = generate(prompt, max_seq_len, temperature, model, tokenizer, vocab, device, seed)
    print(str(temperature) + "\n" + " ".join(generation) + "\n")
```

```
0.1
donald trump has been able to be a lot of people who have been able to be able to be a lot of people who have been able to be able to be

0.2
donald trump has been able to be a little of the city . the company said the company was a good , and the same , and the company ' s brother ,

0.3
donald trump has been able to be a little of the city . in the statement , the department of the state ' s most important thing , the end of the jury

0.4
donald trump has been able to be a little of the city . in the statement , the department of the bower was the first and had been able to have a nod

0.5
donald trump has been an investment for the time of the world . the architect of the united states would be the written one of the press of the life-threatening funds . now

0.6
donald trump has been not a nicky challenging time for them . i ' ve heard . i ' ve been the written one of the press song , she said . now

0.7
donald trump has been not reflected by the adaptive headwind of part . all to have been included by these and the bower f . jacob and song flipped the end . now

0.8
donald trump has been not reflected by the adaptive headwind of part . all to have been included by these and births written by liquidation county and song premiers life-threatening with
ties with

0.9
donald trump has been canceled by nicky segmentation . the battery congregation . all refers to art included adam carl and minn bower f . jacob 2-3 ' s life-threatening explores ties with

1.0
donald trump has been canceled phone nicky segmentation quietly headwind williams congregation . all refers to art included adam “ms and minn bower f . partick 2-3 song premiers life-thre
atening harvests ties with
```