lstm

January 24, 2025

1 Task 1 Dataset Acquisition

2 Dataset Description: Harry Potter LSTM Dataset

Dataset Name: Harry Potter LSTM Dataset
Source: Hugging Face Datasets
Creator: Kaung Htet Cho

2.1 Description

The Harry Potter LSTM dataset is a text dataset derived from the Harry Potter book series. It consists of unstructured text data prepared for natural language processing (NLP) tasks, specifically language modeling. This dataset is designed for sequence-to-sequence learning and is commonly used to train models such as LSTMs and Transformers to generate text in the style of the Harry Potter universe.

2.2 Purpose

The dataset is suitable for various NLP tasks, including: - Text generation - Language modeling - Sequence-to-sequence learning

2.3 Source Attribution

The dataset is hosted on the Hugging Face Datasets platform and was created by Kaung Htet Cho. Proper credit has been provided to the dataset's source and creator.

2.4 License

Refer to the Hugging Face page for licensing details.

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

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

[ ]:
```
device = torch.device("cuda:1" if torch.cuda.is_available() else "cpu")
print(device)
```

```
SEED = 42
torch.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
```

```
cuda:1
```

[ ]:
```
import datasets
# dataset = datasets.load_dataset('microsoft/LCC_python') # not enough gpu␣
 ↪memory
# dataset = datasets.load_dataset('codeparrot/github-jupyter-text-code-pairs')␣
 ↪# not enough gpu
# dataset = datasets.load_dataset('codeparrot/github-jupyter-code-to-text') #␣
 ↪not enough gpu
# will proceed with harry potter

dataset = datasets.load_dataset('KaungHtetCho/Harry_Potter_LSTM')
dataset
```

```
README.md:   0%|          | 0.00/21.0 [00:00<?, ?B/s]

(…)in/Harry Potter 1 - Sorcerer's Stone.txt:   0%|          | 0.00/443k [00:00<?
 ↪, ?B/s]

(…)/Harry Potter 2 - Chamber of Secrets.txt:   0%|          | 0.00/490k [00:00<?
 ↪, ?B/s]

(…)y Potter 3 - The Prisoner Of Azkaban.txt:   0%|          | 0.00/615k [00:00<?
 ↪, ?B/s]

(…)/Harry Potter 4 - The Goblet Of Fire.txt:   0%|          | 0.00/1.11M [00:00<?
 ↪, ?B/s]

(…)arry Potter 5 - Order of the Phoenix.txt:   0%|          | 0.00/1.49M [00:00<?
 ↪, ?B/s]

(…)rry Potter 6 - The Half Blood Prince.txt:   0%|          | 0.00/986k [00:00<?
 ↪, ?B/s]

(…)est/Harry Potter 7 - Deathly Hollows.txt:   0%|          | 0.00/1.14M [00:00<?
 ↪, ?B/s]

Generating train split:   0%|          | 0/57435 [00:00<?, ? examples/s]

Generating validation split:   0%|          | 0/5897 [00:00<?, ? examples/s]

Generating test split:   0%|          | 0/6589 [00:00<?, ? examples/s]
```

[ ]:
```
DatasetDict({
    train: Dataset({
        features: ['text'],
        num_rows: 57435
```

```
    })
    validation: Dataset({
        features: ['text'],
        num_rows: 5897
    })
    test: Dataset({
        features: ['text'],
        num_rows: 6589
    })
})
```

[ ]:
```
dataset['train'][14]['text']
```

[ ]:
```
'None of them noticed a large, tawny owl flutter past the window. '
```

3 Task 2 Model Training

3.0.1 Steps for Preprocessing Text Data

1. **Dataset Loading**
   The dataset was loaded using the Hugging Face `datasets` library to ensure ease of access and compatibility with NLP frameworks.

2. **Text Tokenization**
   A `basic_english` tokenizer was used to split text into lowercased word tokens, removing punctuation and unnecessary formatting.

3. **Removing Empty Text Entries**
   Entries with empty or whitespace-only `text` fields were filtered out to retain only meaningful text data.

4. **Tokenizing the Dataset**
   Each text entry was tokenized to create a list of tokens, preparing the dataset for language modeling tasks.

5. **Removing Empty Token Lists**
   Any entries that resulted in empty token lists after tokenization were filtered out to ensure a clean dataset.

6. **Building the Vocabulary**
   A vocabulary was built from the tokenized dataset with a minimum frequency threshold. Special tokens such as `<unk>` and `<eos>` were added to handle unknown words and mark the end of sentences.

7. **Setting Default Index for Unknown Tokens**
   The `<unk>` token was set as the default index to handle out-of-vocabulary words during model training.

[ ]:
```
tokenizer = torchtext.data.utils.get_tokenizer('basic_english')
```

```
filtered_dataset = dataset.filter(lambda example: example['text'].strip() != '')

tokenize_data = lambda example, tokenizer: {'tokens':␣
 ↪tokenizer(example['text'])}

tokenized_dataset = dataset.map(tokenize_data, remove_columns=[],␣
 ↪fn_kwargs={'tokenizer': tokenizer})
filtered_tokenized_dataset = tokenized_dataset.filter(lambda example:␣
 ↪len(example['tokens']) > 0)
filtered_tokenized_dataset['train'][212]['tokens']
```

[ ]:
```
['the',
 'boa',
 'constrictor',
 'jabbed',
 'its',
 'tail',
 'at',
 'the',
 'sign',
 'again',
 'and',
 'harry',
 'read',
 'on',
 'this',
 'specimen',
 'was',
 'bred',
 'in',
 'the',
 'zoo',
 '.',
 'oh',
 ',',
 'i',
 'see',
 '--',
 'so',
 'you',
 "'",
 've',
 'never',
 'been',
 'to',
 'brazil',
 '?']
```

```
[ ]: vocab = torchtext.vocab.build_vocab_from_iterator(
         tokenized_dataset['train']['tokens'],
         min_freq=3,
         specials=['<unk>', '<eos>'],
         special_first=True
     )
     vocab.set_default_index(vocab["<unk>"])
     len(vocab), vocab.get_itos()[:10]
```

```
[ ]: (9803, ['<unk>', '<eos>', '.', ',', 'the', 'and', 'to', "'", 'of', 'a'])
```

```
[ ]: 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)
         return data
```

```
[ ]: batch_size = 128
     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)
```

3.0.2 Model Architecture

The model used for training is an **LSTM-based Language Model** designed to predict the next word in a sequence. Below are the key components of the architecture:

1. **Embedding Layer**
   - Converts input tokens (word indices) into dense vector representations of a fixed size (`emb_dim`).
   - Initialized uniformly within a small range to ensure stable training.
2. **LSTM Layers**
   - A multi-layer Long Short-Term Memory (LSTM) network with `num_layers` layers and a hidden dimension of `hid_dim`.
   - Incorporates dropout regularization (`dropout_rate`) to reduce overfitting.
   - Processes the sequential input and learns temporal dependencies in the data.
3. **Dropout Layer**
   - Adds dropout after the embedding and LSTM layers to further regularize the model and improve generalization.
4. **Fully Connected Layer**
   - Maps the output of the LSTM to the vocabulary size (`vocab_size`) to produce logits

for each token in the vocabulary.
- The weights and biases are initialized with uniform distribution for better convergence.

5. **Initialization**
   - Custom initialization of weights for the embedding, fully connected, and LSTM layers to ensure stable and efficient training.
6. **Hidden State Management**
   - The `init_hidden` method initializes the LSTM's hidden and cell states with zeros.
   - The `detach_hidden` method detaches hidden states from the computation graph during training to prevent backpropagation through time from exceeding the current batch.

[ ]:
```
class LSTMLanguageModel(nn.Module):
    def __init__(self, vocab_size, emb_dim, hid_dim, num_layers, dropout_rate):
        super(LSTMLanguageModel, self).__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_emb)
        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)
            self.lstm.all_weights[i][1] = torch.FloatTensor(self.emb_dim, self.
↪hid_dim).uniform_(-init_range_other, init_range_other)

    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, cells):
        hidden, cell = cells
        hidden = hidden.detach()
        cell = cell.detach()

        return hidden, cell

    def forward(self, src, hidden):
        embedding = self.dropout(self.embedding(src)) # [batch_size, seq_len]
        output, hidden = self.lstm(embedding, hidden) # [batch_size, seq_len,␣
 ↪hid_dim]

        output     = self.dropout(output)
        prediction = self.fc(output)

        return prediction, hidden
```

[ ]:
```
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
```

[ ]:
```
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

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 36,879,947 trainable parameters
```

[ ]:
```
def get_batch(data, seq_len, idx):
    src    = data[:, idx:idx+seq_len]
    target = data[:, idx+1:idx+seq_len+1]
    return src, target
```

```
[ ]: def train(model, data, optimizer, criterion, batch_size, seq_len, clip, device):
        epoch_loss = 0
        model.train()

        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)

        for idx in tqdm(range(0, num_batches - 1, seq_len), desc='Training',␣
     ↪leave=False):
            optimizer.zero_grad()

            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)

            loss.backward()
            torch.nn.utils.clip_grad_norm_(model.parameters(), clip)
            optimizer.step()

            epoch_loss += loss.item() * seq_len

        return epoch_loss / num_batches


    def valid(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
```

[ ]:
```
n_epochs = 50
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')

import time

epoch_start = time.time()

for epoch in range(0, n_epochs):
    train_loss = train(model, train_data, optimizer, criterion, batch_size,␣
 ↪seq_len, clip, device)
    valid_loss = valid(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(), f"models/best_model_lstm_lm.pt")

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

    print(f'\t[{str(epoch + 1)}/{str(n_epochs)}] epochs progress')

elapsed_epoch = time.time() - epoch_start
print("Train time taken: ", elapsed_epoch)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[52], line 14
     11 epoch_start = time.time()
     13 for epoch in range(0, n_epochs):
---> 14     train_loss = train(model, train_data, optimizer, criterion, batch_size, seq_len, clip, device)
     15     valid_loss = valid(model, valid_data, criterion, batch_size, seq_len, device)
     17     lr_scheduler.step(valid_loss)

Cell In[44], line 30, in train(model, data, optimizer, criterion, batch_size, seq_len, clip, device)
     27     torch.nn.utils.clip_grad_norm_(model.parameters(), clip)
     28     optimizer.step()
---> 30     epoch_loss += loss.item() * seq_len
     32 return epoch_loss / num_batches

KeyboardInterrupt:
```

[ ]:
```
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

model = LSTMLanguageModel(vocab_size, emb_dim, hid_dim, num_layers, dropout_rate).to(device)
model.load_state_dict(torch.load('models/best_model_lstm_lm.pt', map_location=device))

test_loss = valid(model, test_data, criterion, batch_size, seq_len, device)
print(f'Test Perplexity: {math.exp(test_loss):.3f}')
```

```
Test Perplexity: 83.111
```

[ ]:
```
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
```

[ ]:
```
prompt = 'Harry Potter is '
max_seq_len = 100
seed = 0

#smaller the temperature, more diverse tokens but comes
#with a tradeoff of less-make-sense sentence
temperatures = [0.25, 0.3, 0.5, 0.7, 0.75, 0.8, 1.0, 1.8, 1.9, 2.1, 3.1415]
for temperature in temperatures:
    generation = generate(prompt, max_seq_len, temperature, model, tokenizer,
                          vocab, device, seed)
    print(str(temperature)+'\n'+' '.join(generation)+'\n')
```

```
0.25
harry potter is a bit of a

0.3
harry potter is a bit of a joke .

0.5
harry potter is being mistreated ,

0.7
```

harry potter is being mistreated , sir . that was well a prefect .

0.75
harry potter is being mistreated , sir . that was well a prefect , and you would have been cleared with him . . .

0.8
harry potter is being mistreated , sir . that was well a prefect , and you would have been cleared with him . . .

1.0
harry potter is being mistreated , sir . that was well worked . . . .

1.8
harry potter is being sure you gave the new committee ! well should went devils , hurry down it . . give stew back pig ? dead me stupid , pay gifts kill malfoy and twice gurg still . his unpleasantly grunt . as tree issued he placed me nobody never knew viktor been petrified hissed them in fact in gold . now rolled down closer for harrys farewell black over weeks , both acted above voldemort and mostafa helped to gryffindors forward fair .

1.9
harry potter is being sure isn ' tell madam fat opportunity well worked . i ve stumped hurry down it dodge doing mad-eye stew back up ? seemed me caught binky pay safely in malfoy going time gurg weak . makes his end while those as tree issued he have been nobody never knew sorry . harry put himself together in pieces in gold . there rolled connected closer for lifted between riffraff jars like interest since nicolas answering decided and mostafa helped was gryffindors forward fair .

2.1
harry potter is being sure isn ' tell madam fat opportunity well worked . i ve stumped hurry onto it dodge doing mad-eye stew back pig ? seemed me caught feelings pay safely in malfoy going time gurg weak . makes his end while those as tree issued he placed me nobody never knew viktor been petrified hissed them in eighteen crates , and now mcgonagall puts closer for harrys farewell hannah disguise once , both acted above voldemort and mostafa helped to gryffindors forward fair .

3.1415
harry potter is being stumbled piercing tale snapping pouring drawn november well pipes went prefect indignantly hurry