SuperAGI

December 1, 2023

SuperAGI - Assignment - UMESH KUMAR - IIT DELHI - 2020EE10563

Q1. Explanation

In logistic regression, the model learns weights for each feature based on how strongly they correlate with the target variable. When we duplicate a feature in our dataset, we essentially create two identical features which carry the same information.

Here's what typically happens in such a scenario:

Weights of Non-Duplicated Features (W_new_0, W_new_1, ..., W_new_n-1): The weights for all other features (not duplicated) will likely remain similar to their original values (W_0, W_1, ..., W_n-1), assuming that the duplicated feature does not significantly change the overall dynamics of the dataset. However, minor adjustments may occur due to the re-training process and potential interactions with the duplicated feature.

Weights of Duplicated Features (W_new_n and W_new_n+1): The weights for the duplicated features (W_new_n and W_new_n+1) will be an interesting aspect to observe. In an ideal scenario, if the logistic regression model perfectly recognizes the redundancy, it might assign each of these weights approximately half of the original weight of the duplicated feature. This division happens because the model understands that the predictive power of the original feature is now shared between two features.

Specifically, if W_n was the original weight for the feature before duplication, then W_new_n and W_new_n+1 might both be close to W_n / 2. This is under the assumption that the logistic regression model adequately distributes the influence of the now duplicated feature across both instances.

Potential Variability Due to Regularization and Data Characteristics: The exact behavior can vary depending on factors like the regularization used in the model and the characteristics of the data. For instance, if strong regularization is applied, the model might more aggressively penalize the weights, leading to smaller weights for both duplicated features.

While the exact values of the weights will depend on various factors including the data, the training process, and regularization, a common expectation is that the duplicated features will share the importance (weight) that was originally assigned to the single feature before duplication.

Q2. Explanation:

In order to answer this question, you would normally compare the click-through rates (CTRs) of each template to the control template (Template A) with a 95% confidence level using a statistical test such a percentage z-test or chi-squared test. Nonetheless, we can draw certain conclusions based only on the provided data and without actually doing any calculations. Given the data:

Template A (Control) CTR = 10% Template B CTR = 7% Template C CTR = 8.5% Template D CTR = 12% Template E CTR = 14%

And the goal is to find which conclusion is true with 95% confidence.

a. "We have too little data to conclude that A is better or worse than any other template with 95% confidence." - This statement seems incorrect because, with 1000 emails sent for each template, we typically have enough data to reach a statistical conclusion.

b. "E is better than A with over 95% confidence, B is worse than A with over 95% confidence. You need to run the test for longer to tell where C and D compare to A with 95% confidence." - This statement is likely true for the comparison between A and E, as well as A and B, since the differences in their CTRs are substantial. However, without running statistical tests, we cannot be certain whether we need more data for C and D. This statement is partially true based on the given CTRs but lacks the backing of a statistical test for full validation.

c. "Both D and E are better than A with 95% confidence. Both B and C are worse than A with over 95% confidence." - This statement might be true for E being better than A, but claiming the same for D and stating that B and C are worse than A with over 95% confidence requires a statistical test to verify.

Based on the options provided and without performing the statistical tests, the most likely correct statement seems to be: b. "E is better than A with over 95% confidence, B is worse than A with over 95% confidence. You need to run the test for longer to tell where C and D compare to A with 95% confidence."

This is because the CTR for E is significantly higher than A, and for B it is significantly lower. For C and D, while the CTRs are different from A, we cannot determine the confidence level of these differences without further statistical analysis. Therefore, more data and testing might be necessary to make a statistically significant conclusion for C and D.

Q3. Explanation :

For logistic regression with m training examples and n features, where each feature vector is sparse and has an average of k non-zero entries (with kn), the computational cost of each gradient descent iteration can be approximated as follows:

In well-optimized packages, the computation will primarily involve operations on the non-zero entries of the feature vectors. The key operations during each gradient descent iteration are the computation of the hypothesis function and the gradient update for each parameter.

Hypothesis Computation: For logistic regression, the hypothesis for a single example is computed

as the sigmoid function of the weighted sum of the features. Since there are k non-zero features on average, the cost of computing the hypothesis for one example is O(k).

Gradient Computation: The gradient of the cost function with respect to each parameter is the average over all training examples of the product of the prediction error and the corresponding feature. Due to sparsity, each parameter update is only affected by non-zero features. Thus, the average number of operations per parameter per training example is proportional to k.

Parameter Updates: There are n parameters to update, but for each example, only k updates are needed due to sparsity. Therefore, the total cost for updating all parameters across all examples is O(mk).

Given that modern, well-written packages utilize sparse matrix optimizations, the approximate computational cost of each gradient descent iteration for the entire training set is O(mk). This is significantly less than the O(mn) cost you would expect with dense feature vectors.

The approximate computational cost of each gradient descent iteration of logistic regression in modern well-written packages, given the sparsity of the features, is O(mk) , which is linear with respect to the number of training examples and the average number of non-zero features per example.

Q4. Explanation:

The different methods for generating additional training data and their likely impact on the accuracy of the version V2 of a text classifier, let's consider each approach:

a. Using the V1 classifier on 1 million random stories and selecting the 10k where the V1 classifier's output is closest to the decision boundary: This method is likely to provide examples where V1 is least certain. The new classifier V2 trained on these examples would improve on cases where V1 was ambivalent. This method helps in refining the decision boundary of the classifier but might not expose the classifier to a wide variety of harder examples.

b. Getting 10k random labeled stories from the 1000 news sources: This approach introduces randomness and variety into the training set. It might not specifically target the weaknesses of V1, but it ensures that V2 has a broader understanding of the data distribution. This can potentially increase generalizability but may not significantly improve upon the specific errors made by V1.

c. Picking a random sample of 1 million stories, labeling them, and then selecting 10k stories where V1's output is both wrong and farthest from the decision boundary: This approach focuses on the most erroneous predictions of V1. By training V2 on these examples, the new classifier is directly addressing the areas where V1 performs poorly. This method is likely to yield the most significant improvements in accuracy since it focuses on correcting the most substantial mistakes of V1. Ignoring the differences in costs and efforts, here's how the methods might rank in terms of improving accuracy:

Method c: Most likely to improve accuracy significantly since it focuses on the mistakes of V1.

Method a: Could improve accuracy in a more nuanced way by targeting ambiguous cases.

Method b: Likely to have the least impact on accuracy improvement as it doesn't target specific weaknesses of V1.

If the goal is to maximize the pure accuracy of classifier V2, focusing on the previous mistakes of V1 (as in method c) is probably the most effective strategy.

This approach allows the model to learn from the most critical failures, which could lead to a more robust classifier. However, it's also essential for a classifier to be exposed to a variety of data, as in method b, to prevent overfitting to specific types of errors. Therefore, a combination of methods might be ideal in practice.

Q.5 Explanation:

Considering each case,

a. Maximum Likelihood Estimate (MLE) The MLE for a binomial distribution is simply the ratio of observed successes (heads) to the total number of trials (tosses):

pMLE = k/n

b. Bayesian Estimate Under the Bayesian framework with a uniform prior, the posterior distribution for p after observing k heads in n tosses is a Beta distribution with parameters α=k+1 and β=nk+1. The expected value of the posterior distribution, which is the mean of the Beta distribution, is given by:

pBayesian=$\alpha/(\alpha+\beta)$ = (k+1)/(n+2)

c. Maximum a Posteriori (MAP) Estimate The MAP estimate for a Beta distribution with a uniform prior is the mode of the posterior distribution.

For a Beta distribution, the mode is (α1)/($\alpha+\beta$2), provided $\alpha>1$ and $\beta>1$. If k=0 or k=n, the mode would be at the boundaries, 0 or 1, respectively. Otherwise, the MAP estimate is:

pMAP = (k1+1)/(nk1+1+k1+1) = k/n

In cases where k=0 or k=n, the MAP estimate should be adjusted to avoid undefined behavior (division by zero or undefined mode for the Beta distribution). For k=0, the mode (MAP estimate) is 0, and for k=n, the mode is 1.

However, with a uniform prior, the MAP estimate coincides with the MLE for interior values of k, because the prior is constant and therefore does not affect the location of the peak of the posterior distribution.

Coding Assignment: Implementation and Optimization of GPT-2 Model

Steps are mentioned below for each function to be implemented in all Tasks

Import Necessary Libraries

Need PyTorch, a popular deep learning library. Used its various modules for building the GPT-2 architecture

[52]:
```
!pip install torch
```

```
Requirement already satisfied: torch in
/Applications/anaconda3/lib/python3.11/site-packages (2.1.1)
Requirement already satisfied: filelock in
/Applications/anaconda3/lib/python3.11/site-packages (from torch) (3.9.0)
Requirement already satisfied: typing-extensions in
/Applications/anaconda3/lib/python3.11/site-packages (from torch) (4.7.1)
Requirement already satisfied: sympy in
/Applications/anaconda3/lib/python3.11/site-packages (from torch) (1.11.1)
Requirement already satisfied: networkx in
/Applications/anaconda3/lib/python3.11/site-packages (from torch) (3.1)
Requirement already satisfied: jinja2 in
/Applications/anaconda3/lib/python3.11/site-packages (from torch) (3.1.2)
Requirement already satisfied: fsspec in
/Applications/anaconda3/lib/python3.11/site-packages (from torch) (2023.4.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/Applications/anaconda3/lib/python3.11/site-packages (from jinja2->torch)
(2.1.1)
Requirement already satisfied: mpmath>=0.19 in
/Applications/anaconda3/lib/python3.11/site-packages (from sympy->torch) (1.3.0)
```

[53]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

Positional and Token Embeddings

GPT-2 uses embeddings to convert tokens (words) into vectors and adds positional encodings to maintain the order of words.

[54]:
```
class GPT2Embeddings(nn.Module):
    def __init__(self, vocab_size, max_seq_len, embed_dim):
        super().__init__()
        self.token_embeddings = nn.Embedding(vocab_size, embed_dim)
        self.position_embeddings = nn.Embedding(max_seq_len, embed_dim)
    def forward(self, input_ids):
        position_ids = torch.arange(0, input_ids.size(1)).unsqueeze(0)
        return self.token_embeddings(input_ids) + self.
 ↪position_embeddings(position_ids)
```

Multi-Head Self-Attention

The self-attention mechanism allows the model to weigh the importance of different words in a sentence.

[55]:
```
class MultiHeadSelfAttention(nn.Module):
    def __init__(self, embed_dim, num_heads):
        super().__init__()
        self.embed_dim = embed_dim
        self.num_heads = num_heads
        self.head_dim = embed_dim // num_heads

        self.query = nn.Linear(embed_dim, embed_dim)
        self.key = nn.Linear(embed_dim, embed_dim)
        self.value = nn.Linear(embed_dim, embed_dim)
        self.out = nn.Linear(embed_dim, embed_dim)

    def forward(self, x):
        batch_size = x.size(0)
        Q = self.query(x)
        K = self.key(x)
        V = self.value(x)
        Q = Q.view(batch_size, -1, self.num_heads, self.head_dim).transpose(1, 2)
        K = K.view(batch_size, -1, self.num_heads, self.head_dim).transpose(1, 2)
        V = V.view(batch_size, -1, self.num_heads, self.head_dim).transpose(1, 2)
        attention_scores = torch.matmul(Q, K.transpose(-2, -1)) / torch.
↪sqrt(torch.tensor(self.head_dim, dtype=torch.float32))
        attention_probs = F.softmax(attention_scores, dim=-1)
        attention_output = torch.matmul(attention_probs, V)
        attention_output = attention_output.transpose(1, 2).contiguous().
↪view(batch_size, -1, self.embed_dim)
        return self.out(attention_output)
```

Point-Wise Feed-Forward Network

This network consists of two linear transformations with a ReLU activation in between.

[56]:
```
class FeedForward(nn.Module):
    def __init__(self, embed_dim, ff_dim):
        super().__init__()
        self.fc1 = nn.Linear(embed_dim, ff_dim)
        self.fc2 = nn.Linear(ff_dim, embed_dim)

    def forward(self, x):
        return self.fc2(F.relu(self.fc1(x)))
```

Transformer Block

Each transformer block combines the attention and feed-forward networks.

[57]:
```
class TransformerBlock(nn.Module):
    def __init__(self, embed_dim, num_heads, ff_dim):
        super().__init__()
        self.attention = MultiHeadSelfAttention(embed_dim, num_heads)
        self.ffn = FeedForward(embed_dim, ff_dim)
        self.layernorm1 = nn.LayerNorm(embed_dim)
        self.layernorm2 = nn.LayerNorm(embed_dim)

    def forward(self, x):
        attention_output = self.attention(x)
        x = self.layernorm1(x + attention_output)
        ffn_output = self.ffn(x)
        x = self.layernorm2(x + ffn_output)
        return x
```

Assembling GPT-2

Now, you can assemble the full GPT-2 model using the components you've created.

[58]:
```
class GPT2(nn.Module):
    def __init__(self, vocab_size, max_seq_len, embed_dim, num_heads, ff_dim,␣
↪num_layers):
        super().__init__()
        self.embeddings = GPT2Embeddings(vocab_size, max_seq_len, embed_dim)
        self.transformer_blocks = nn.ModuleList([TransformerBlock(embed_dim,␣
↪num_heads, ff_dim) for _ in range(num_layers)])
        self.lm_head = nn.Linear(embed_dim, vocab_size, bias=False)

    def forward(self, input_ids):
        x = self.embeddings(input_ids)
        for block in self.transformer_blocks:
            x = block(x)
        logits = self.lm_head(x)
        return logits
```

Load Pre-Trained GPT-Neo 125M Model

First, you need to install the Transformers library if you haven't already. You can do this via pip Then, you can load the GPT-Neo 125M model

[59]:
```
!pip install transformers
```

```
Requirement already satisfied: transformers in
/Applications/anaconda3/lib/python3.11/site-packages (4.32.1)
Requirement already satisfied: filelock in
/Applications/anaconda3/lib/python3.11/site-packages (from transformers) (3.9.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.15.1 in
/Applications/anaconda3/lib/python3.11/site-packages (from transformers)
(0.15.1)
Requirement already satisfied: numpy>=1.17 in
/Applications/anaconda3/lib/python3.11/site-packages (from transformers)
(1.24.3)
Requirement already satisfied: packaging>=20.0 in
/Applications/anaconda3/lib/python3.11/site-packages (from transformers) (23.1)
Requirement already satisfied: pyyaml>=5.1 in
/Applications/anaconda3/lib/python3.11/site-packages (from transformers) (6.0)
Requirement already satisfied: regex!=2019.12.17 in
/Applications/anaconda3/lib/python3.11/site-packages (from transformers)
(2022.7.9)
Requirement already satisfied: requests in
/Applications/anaconda3/lib/python3.11/site-packages (from transformers)
(2.31.0)
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in
/Applications/anaconda3/lib/python3.11/site-packages (from transformers)
(0.13.2)
Requirement already satisfied: safetensors>=0.3.1 in
/Applications/anaconda3/lib/python3.11/site-packages (from transformers) (0.3.2)
Requirement already satisfied: tqdm>=4.27 in
/Applications/anaconda3/lib/python3.11/site-packages (from transformers)
(4.65.0)
Requirement already satisfied: fsspec in
/Applications/anaconda3/lib/python3.11/site-packages (from huggingface-
hub<1.0,>=0.15.1->transformers) (2023.4.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/Applications/anaconda3/lib/python3.11/site-packages (from huggingface-
hub<1.0,>=0.15.1->transformers) (4.7.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/Applications/anaconda3/lib/python3.11/site-packages (from
requests->transformers) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in
/Applications/anaconda3/lib/python3.11/site-packages (from
requests->transformers) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/Applications/anaconda3/lib/python3.11/site-packages (from
requests->transformers) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in
/Applications/anaconda3/lib/python3.11/site-packages (from
requests->transformers) (2023.7.22)
```

[60]:
```
from transformers import GPTNeoForCausalLM, GPT2Tokenizer

model_name = 'EleutherAI/gpt-neo-125M'
model = GPTNeoForCausalLM.from_pretrained(model_name)
tokenizer = GPT2Tokenizer.from_pretrained(model_name)
```

Running a Sample Prediction

With the model loaded, you can now generate text. Here's an example of how to do it:

[61]:
```
prompt = "Describe Lord Krishna?"

inputs = tokenizer(prompt, return_tensors="pt")
outputs = model.generate(
    **inputs,
    max_length=200,          # Increase the maximum length
    num_return_sequences=1,
    temperature=0.9,         # Adjust for randomness
    top_k=50,                # Limits the next word selection to top-k words
    top_p=0.95,              # Nucleus sampling: restricts sampling pool to top p%
    repetition_penalty=1.2, # Penalize repetition
    no_repeat_ngram_size=2, # Prevent repeating n-grams
    pad_token_id=tokenizer.eos_token_id
)

generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)

sentences = generated_text.split('.')
output_text = '.'.join(sentences[:-1]) + '.'

print("Generated Text:\n")
print(output_text)
```

```
/Applications/anaconda3/lib/python3.11/site-
packages/transformers/generation/configuration_utils.py:362: UserWarning:
`do_sample` is set to `False`. However, `temperature` is set to `0.9` -- this
flag is only used in sample-based generation modes. You should set
`do_sample=True` or unset `temperature`.
  warnings.warn(
/Applications/anaconda3/lib/python3.11/site-
packages/transformers/generation/configuration_utils.py:367: UserWarning:
`do_sample` is set to `False`. However, `top_p` is set to `0.95` -- this flag is
only used in sample-based generation modes. You should set `do_sample=True` or
unset `top_p`.
  warnings.warn(

Generated Text:
```

```
Describe Lord Krishna?

Lord Krishna is a Hindu deity, the creator of the universe. He is the god of all
the gods, and the most important of them all. Krishna was the first to be
created by God, but he was also the founder of many other gods.
He is also known as the "father of Krishna" (the "Father of All"), and is
believed to have been the father of Shiva. Shiva is said to possess the power of
a great god, Shiva the Great, who is called the God of Love. The god is known to
exist in the form of an animal, which is referred to as a "goddess". Shiva was
said by the ancient Greeks to represent the divine essence of God. In the Vedic
tradition, it is thought that Shiva represents the essence or essence-of-God. It
is possible that the origin of this deity is unknown. However, there is evidence
that it was a divine being.
```

This script takes a prompt, tokenizes it, generates a sequence of text, and then decodes it back into human-readable form.

Rotary Positional Embedding

Rotary Positional Embedding (RoPE) integrates positional information by applying a rotation to the embedding space.

[62]:
```python
class RotaryPositionalEmbedding(nn.Module):
    def __init__(self, dim, max_seq_len):
        super().__init__()
        self.dim = dim
        inv_freq = 1. / (10000 ** (torch.arange(0, dim, 2).float() / dim))
        t = torch.arange(max_seq_len).type_as(inv_freq)
        sinusoid_inp = torch.einsum("i , j -> i j", t, inv_freq)
        self.register_buffer("sinusoid_inp", sinusoid_inp)

    def forward(self, x):
        sinusoid_inp = self.sinusoid_inp[:x.size(1), :].unsqueeze(0)
        return torch.cat((sinusoid_inp.sin(), sinusoid_inp.cos()), dim=-1)
```

Group Query Attention

Group Query Attention (GQA) groups queries into different sets and applies attention within these sets.

[63]:
```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class GroupQueryAttention(nn.Module):
```

```
    def __init__(self, embed_dim, num_heads, num_groups):
        super().__init__()
        self.num_heads = num_heads
        self.num_groups = num_groups
        self.group_size = num_heads // num_groups
        self.head_dim = embed_dim // num_heads
        assert self.head_dim * num_heads == embed_dim, "embed_dim must be␣
↪divisible by num_heads"
        assert num_heads % num_groups == 0, "num_heads must be divisible by␣
↪num_groups"

        self.query = nn.Linear(embed_dim, embed_dim)
        self.key = nn.Linear(embed_dim, embed_dim)
        self.value = nn.Linear(embed_dim, embed_dim)
        self.out = nn.Linear(embed_dim, embed_dim)

    def forward(self, x):
        B, T, C = x.size()
        Q = self.query(x).view(B, T, self.num_groups, self.group_size, self.
↪head_dim)
        K = self.key(x).view(B, T, self.num_groups, self.group_size, self.
↪head_dim)
        V = self.value(x).view(B, T, self.num_groups, self.group_size, self.
↪head_dim)

        Q = Q.permute(0, 2, 3, 1, 4)
        K = K.permute(0, 2, 3, 1, 4)
        V = V.permute(0, 2, 3, 1, 4)

        Q = Q * (self.head_dim ** -0.5)
        attn = torch.matmul(Q, K.transpose(-2, -1))  # (B, num_groups,␣
↪group_size, T, T)
        attn = F.softmax(attn, dim=-1)
        attention_output = torch.matmul(attn, V)  # (B, num_groups, group_size,␣
↪T, head_dim)
        attention_output = attention_output.permute(0, 3, 1, 2, 4).contiguous()
        attention_output = attention_output.view(B, T, C)

        return self.out(attention_output)
```

Sliding Window Attention

Sliding Window Attention applies attention within a fixed-size window.

[64]:
```
import torch
import torch.nn as nn
```

```
import torch.nn.functional as F

class SlidingWindowAttention(nn.Module):
    def __init__(self, embed_dim, num_heads, window_size):
        super().__init__()
        self.window_size = window_size
        self.num_heads = num_heads
        self.head_dim = embed_dim // num_heads
        assert self.head_dim * num_heads == embed_dim, "embed_dim must be␣
↪divisible by num_heads"

        self.query = nn.Linear(embed_dim, embed_dim)
        self.key = nn.Linear(embed_dim, embed_dim)
        self.value = nn.Linear(embed_dim, embed_dim)
        self.out = nn.Linear(embed_dim, embed_dim)

    def forward(self, x):
        B, T, C = x.size()
        Q = self.query(x).view(B, T, self.num_heads, self.head_dim)
        K = self.key(x).view(B, T, self.num_heads, self.head_dim)
        V = self.value(x).view(B, T, self.num_heads, self.head_dim)

        # Sliding window attention calculation
        # Initialize a zero matrix for the attention scores
        attn_scores = torch.zeros(B, self.num_heads, T, T, device=x.device)

        # Calculate attention scores for each head, considering the window size
        for head in range(self.num_heads):
            for i in range(T):
                window_start = max(0, i - self.window_size)
                window_end = min(T, i + self.window_size + 1)
                attn_scores[:, head, i, window_start:window_end] = torch.matmul(
                    Q[:, i, head].unsqueeze(1),
                    K[:, window_start:window_end, head].transpose(-2, -1)
                ).squeeze(1)

        # Scale the attention scores
        attn_scores = attn_scores / (self.head_dim ** 0.5)

        # Apply softmax to get attention probabilities
        attn_probs = F.softmax(attn_scores, dim=-1)

        # Compute the weighted sum of values
        attention_output = torch.matmul(attn_probs, V)
        attention_output = attention_output.transpose(1, 2).contiguous().view(B,␣
↪T, C)
        return self.out(attention_output)
```

Integration and Testing Integrate these modules into your GPT-2 architecture, replacing the corresponding parts. Ensure the data flow and tensor shapes are compatible throughout the model. After integration, test the model rigorously to ensure it functions as expected and compare its performance to the baseline GPT-2 model.

Single GPU Training Loop

This is the basic training setup where the model, data, and computations are all on a single GPU.

[65]:
```
import torch
import torch.nn as nn
import torch.optim as optim

def train_single_gpu(model, train_loader, epochs, learning_rate):
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    model = model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    model.train()
    for epoch in range(epochs):
        for batch_idx, (data, targets) in enumerate(train_loader):
            data, targets = data.to(device), targets.to(device)

            # Forward pass
            outputs = model(data)
            loss = criterion(outputs, targets)

            # Backward and optimize
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            print(f"Epoch [{epoch+1}/{epochs}], Step [{batch_idx+1}/
 ↪{len(train_loader)}], Loss: {loss.item():.4f}")
```

Distributed Data Parallel (DDP) Training Loop

DDP is used for multi-GPU training where each GPU processes a subset of the data.

[66]:
```
import torch.distributed as dist
from torch.nn.parallel import DistributedDataParallel as DDP

def setup(rank, world_size):
    dist.init_process_group("nccl", rank=rank, world_size=world_size)

def cleanup():
```

```
    dist.destroy_process_group()

def train_ddp(rank, world_size, model, train_loader, epochs, learning_rate):
    setup(rank, world_size)

    device = torch.device(f"cuda:{rank}")
    model = model.to(device)
    model = DDP(model, device_ids=[rank])

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    model.train()
    for epoch in range(epochs):
        for batch_idx, (data, targets) in enumerate(train_loader):
            data, targets = data.to(device), targets.to(device)

            # Forward pass
            outputs = model(data)
            loss = criterion(outputs, targets)

            # Backward and optimize
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            if rank == 0:
                print(f"Epoch [{epoch+1}/{epochs}], Step [{batch_idx+1}/
↪{len(train_loader)}], Loss: {loss.item():.4f}")

    cleanup()
```

Fully Sharded Data Parallel (FSDP) Training Loop

FSDP shards the model parameters, gradients, and optimizer state across all GPUs.

[73]:
```
from torch.distributed.fsdp import FullyShardedDataParallel as FSDP

def train_fsdp(rank, world_size, model, train_loader, epochs, learning_rate):
    setup(rank, world_size)

    device = torch.device(f"cuda:{rank}")
    model = model.to(device)
    model = FSDP(model)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

```
    model.train()
    for epoch in range(epochs):
        for batch_idx, (data, targets) in enumerate(train_loader):
            data, targets = data.to(device), targets.to(device)

            # Forward pass
            outputs = model(data)
            loss = criterion(outputs, targets)

            # Backward and optimize
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            if rank == 0:
                print(f"Epoch [{epoch+1}/{epochs}], Step [{batch_idx+1}/
↪{len(train_loader)}], Loss: {loss.item():.4f}")
    cleanup()
```

Usage Single GPU: Directly call train_single_gpu. DDP: Call train_ddp for each process. Use torch.multiprocessing.spawn to spawn multiple processes. FSDP: Similar to DDP, but call train_fsdp instead.

[74]:
```
!pip install transformers
```

```
Requirement already satisfied: transformers in
/Applications/anaconda3/lib/python3.11/site-packages (4.32.1)
Requirement already satisfied: filelock in
/Applications/anaconda3/lib/python3.11/site-packages (from transformers) (3.9.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.15.1 in
/Applications/anaconda3/lib/python3.11/site-packages (from transformers)
(0.15.1)
Requirement already satisfied: numpy>=1.17 in
/Applications/anaconda3/lib/python3.11/site-packages (from transformers)
(1.24.3)
Requirement already satisfied: packaging>=20.0 in
/Applications/anaconda3/lib/python3.11/site-packages (from transformers) (23.1)
Requirement already satisfied: pyyaml>=5.1 in
/Applications/anaconda3/lib/python3.11/site-packages (from transformers) (6.0)
Requirement already satisfied: regex!=2019.12.17 in
/Applications/anaconda3/lib/python3.11/site-packages (from transformers)
(2022.7.9)
Requirement already satisfied: requests in
/Applications/anaconda3/lib/python3.11/site-packages (from transformers)
(2.31.0)
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in
/Applications/anaconda3/lib/python3.11/site-packages (from transformers)
```

```
(0.13.2)
Requirement already satisfied: safetensors>=0.3.1 in
/Applications/anaconda3/lib/python3.11/site-packages (from transformers) (0.3.2)
Requirement already satisfied: tqdm>=4.27 in
/Applications/anaconda3/lib/python3.11/site-packages (from transformers)
(4.65.0)
Requirement already satisfied: fsspec in
/Applications/anaconda3/lib/python3.11/site-packages (from huggingface-
hub<1.0,>=0.15.1->transformers) (2023.4.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/Applications/anaconda3/lib/python3.11/site-packages (from huggingface-
hub<1.0,>=0.15.1->transformers) (4.7.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/Applications/anaconda3/lib/python3.11/site-packages (from
requests->transformers) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in
/Applications/anaconda3/lib/python3.11/site-packages (from
requests->transformers) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/Applications/anaconda3/lib/python3.11/site-packages (from
requests->transformers) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in
/Applications/anaconda3/lib/python3.11/site-packages (from
requests->transformers) (2023.7.22)
```

Model Initialization:

Import and initialize the GPTNeoForCausalLM model from Transformers. This model is a pre-trained GPT-Neo model, which is similar to GPT-2 but created by EleutherAI.

[75]:
```
from transformers import GPTNeoForCausalLM, GPT2Tokenizer

model_name = 'EleutherAI/gpt-neo-125M'
model = GPTNeoForCausalLM.from_pretrained(model_name)
tokenizer = GPT2Tokenizer.from_pretrained(model_name)
```

Handling Vocabulary Mismatch

If you choose to map your synthetic dataset's tokens to the model's existing vocabulary, you need to modify how you create your synthetic dataset.

[76]:
```
import random

def generate_synthetic_data(tokenizer, num_sentences=1000, max_length=50):
    vocab_list = list(tokenizer.get_vocab().keys())
    data = []
```

```
    for _ in range(num_sentences):
        sentence_length = random.randint(10, max_length)
        sentence = random.choices(vocab_list, k=sentence_length)
        data.append(' '.join(sentence))

    return data

synthetic_data = generate_synthetic_data(tokenizer)
```

[77]:
```
synthetic_data = generate_synthetic_data(tokenizer, num_sentences=5,␣
↪max_length=20)
```

Generating a Test Dataset

For testing purposes, I created a simple synthetic dataset. However, for more realistic testing, especially to evaluate the model's language generation capabilities, it's better to use a real-world dataset. Here, created a synthetic one for simplicity:

[78]:
```
def generate_synthetic_dataset(num_samples, max_length):
    import random
    import torch

    data = torch.randint(0, 50257, (num_samples, max_length))
    targets = torch.randint(0, 50257, (num_samples, max_length))
    return data, targets

num_samples = 1000
max_length = 50
data, targets = generate_synthetic_dataset(num_samples, max_length)
```

Setting Up DataLoader for Testing

[79]:
```
from torch.utils.data import TensorDataset, DataLoader

dataset = TensorDataset(data, targets)
train_loader = DataLoader(dataset, batch_size=8, shuffle=True)
```

[80]:
```
def train_single_gpu(model, train_loader, epochs, learning_rate):
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    model = model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    model.train()
    for epoch in range(epochs):
        for batch_idx, (data, targets) in enumerate(train_loader):
```

```
            data, targets = data.to(device), targets.to(device)

            outputs = model(data)
            logits = outputs.logits
            loss = criterion(logits.view(-1, logits.size(-1)), targets.view(-1))

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            print(f"Epoch [{epoch+1}/{epochs}], Step [{batch_idx+1}/
↪{len(train_loader)}], Loss: {loss.item():.4f}")
```

[81]:
```
train_single_gpu(model, train_loader, epochs=2, learning_rate=2e-5)
```

```
Epoch [1/2], Step [1/125], Loss: 13.5120
Epoch [1/2], Step [2/125], Loss: 13.1184
Epoch [1/2], Step [3/125], Loss: 13.1120
Epoch [1/2], Step [4/125], Loss: 12.6890
Epoch [1/2], Step [5/125], Loss: 12.3483
Epoch [1/2], Step [6/125], Loss: 12.3654
Epoch [1/2], Step [7/125], Loss: 12.2126
Epoch [1/2], Step [8/125], Loss: 12.0616
Epoch [1/2], Step [9/125], Loss: 11.9245
Epoch [1/2], Step [10/125], Loss: 11.9683
Epoch [1/2], Step [11/125], Loss: 11.8689
Epoch [1/2], Step [12/125], Loss: 11.8611
Epoch [1/2], Step [13/125], Loss: 11.7912
Epoch [1/2], Step [14/125], Loss: 11.7366
Epoch [1/2], Step [15/125], Loss: 11.7492
Epoch [1/2], Step [16/125], Loss: 11.6288
Epoch [1/2], Step [17/125], Loss: 11.5451
Epoch [1/2], Step [18/125], Loss: 11.7411
Epoch [1/2], Step [19/125], Loss: 11.6695
Epoch [1/2], Step [20/125], Loss: 11.5431
Epoch [1/2], Step [21/125], Loss: 11.5362
Epoch [1/2], Step [22/125], Loss: 11.5725
Epoch [1/2], Step [23/125], Loss: 11.4490
Epoch [1/2], Step [24/125], Loss: 11.5516
Epoch [1/2], Step [25/125], Loss: 11.5040
Epoch [1/2], Step [26/125], Loss: 11.4472
Epoch [1/2], Step [27/125], Loss: 11.3814
Epoch [1/2], Step [28/125], Loss: 11.4155
Epoch [1/2], Step [29/125], Loss: 11.4677
Epoch [1/2], Step [30/125], Loss: 11.3683
Epoch [1/2], Step [31/125], Loss: 11.4020
Epoch [1/2], Step [32/125], Loss: 11.2634
Epoch [1/2], Step [33/125], Loss: 11.3684
```

```
Epoch [1/2], Step [34/125], Loss: 11.3872
Epoch [1/2], Step [35/125], Loss: 11.2706
Epoch [1/2], Step [36/125], Loss: 11.3198
Epoch [1/2], Step [37/125], Loss: 11.3658
Epoch [1/2], Step [38/125], Loss: 11.3267
Epoch [1/2], Step [39/125], Loss: 11.3417
Epoch [1/2], Step [40/125], Loss: 11.4282
Epoch [1/2], Step [41/125], Loss: 11.3650
Epoch [1/2], Step [42/125], Loss: 11.3446
Epoch [1/2], Step [43/125], Loss: 11.2871
Epoch [1/2], Step [44/125], Loss: 11.3496
Epoch [1/2], Step [45/125], Loss: 11.1752
Epoch [1/2], Step [46/125], Loss: 11.3473
Epoch [1/2], Step [47/125], Loss: 11.2720
Epoch [1/2], Step [48/125], Loss: 11.3698
Epoch [1/2], Step [49/125], Loss: 11.2957
Epoch [1/2], Step [50/125], Loss: 11.2481
Epoch [1/2], Step [51/125], Loss: 11.2939
Epoch [1/2], Step [52/125], Loss: 11.3135
Epoch [1/2], Step [53/125], Loss: 11.3558
Epoch [1/2], Step [54/125], Loss: 11.2340
Epoch [1/2], Step [55/125], Loss: 11.2548
Epoch [1/2], Step [56/125], Loss: 11.2934
Epoch [1/2], Step [57/125], Loss: 11.3427
Epoch [1/2], Step [58/125], Loss: 11.2475
Epoch [1/2], Step [59/125], Loss: 11.2084
Epoch [1/2], Step [60/125], Loss: 11.1749
Epoch [1/2], Step [61/125], Loss: 11.2171
Epoch [1/2], Step [62/125], Loss: 11.2315
Epoch [1/2], Step [63/125], Loss: 11.2538
Epoch [1/2], Step [64/125], Loss: 11.2430
Epoch [1/2], Step [65/125], Loss: 11.2465
Epoch [1/2], Step [66/125], Loss: 11.2386
Epoch [1/2], Step [67/125], Loss: 11.1484
Epoch [1/2], Step [68/125], Loss: 11.1878
Epoch [1/2], Step [69/125], Loss: 11.2254
Epoch [1/2], Step [70/125], Loss: 11.2274
Epoch [1/2], Step [71/125], Loss: 11.2364
Epoch [1/2], Step [72/125], Loss: 11.2443
Epoch [1/2], Step [73/125], Loss: 11.2988
Epoch [1/2], Step [74/125], Loss: 11.2502
Epoch [1/2], Step [75/125], Loss: 11.2493
Epoch [1/2], Step [76/125], Loss: 11.2905
Epoch [1/2], Step [77/125], Loss: 11.2069
Epoch [1/2], Step [78/125], Loss: 11.1492
Epoch [1/2], Step [79/125], Loss: 11.2843
Epoch [1/2], Step [80/125], Loss: 11.1914
Epoch [1/2], Step [81/125], Loss: 11.2512
```

```
Epoch [1/2], Step [82/125], Loss: 11.1688
Epoch [1/2], Step [83/125], Loss: 11.1825
Epoch [1/2], Step [84/125], Loss: 11.2122
Epoch [1/2], Step [85/125], Loss: 11.2772
Epoch [1/2], Step [86/125], Loss: 11.2615
Epoch [1/2], Step [87/125], Loss: 11.2598
Epoch [1/2], Step [88/125], Loss: 11.2404
Epoch [1/2], Step [89/125], Loss: 11.2956
Epoch [1/2], Step [90/125], Loss: 11.3133
Epoch [1/2], Step [91/125], Loss: 11.1919
Epoch [1/2], Step [92/125], Loss: 11.1947
Epoch [1/2], Step [93/125], Loss: 11.2029
Epoch [1/2], Step [94/125], Loss: 11.2093
Epoch [1/2], Step [95/125], Loss: 11.1685
Epoch [1/2], Step [96/125], Loss: 11.1770
Epoch [1/2], Step [97/125], Loss: 11.2199
Epoch [1/2], Step [98/125], Loss: 11.1749
Epoch [1/2], Step [99/125], Loss: 11.3266
Epoch [1/2], Step [100/125], Loss: 11.3156
Epoch [1/2], Step [101/125], Loss: 11.1663
Epoch [1/2], Step [102/125], Loss: 11.1683
Epoch [1/2], Step [103/125], Loss: 11.1136
Epoch [1/2], Step [104/125], Loss: 11.2784
Epoch [1/2], Step [105/125], Loss: 11.1792
Epoch [1/2], Step [106/125], Loss: 11.1856
Epoch [1/2], Step [107/125], Loss: 11.2555
Epoch [1/2], Step [108/125], Loss: 11.1049
Epoch [1/2], Step [109/125], Loss: 11.2841
Epoch [1/2], Step [110/125], Loss: 11.2299
Epoch [1/2], Step [111/125], Loss: 11.0887
Epoch [1/2], Step [112/125], Loss: 11.2555
Epoch [1/2], Step [113/125], Loss: 11.1879
Epoch [1/2], Step [114/125], Loss: 11.1572
Epoch [1/2], Step [115/125], Loss: 11.3010
Epoch [1/2], Step [116/125], Loss: 11.1383
Epoch [1/2], Step [117/125], Loss: 11.1476
Epoch [1/2], Step [118/125], Loss: 11.0961
Epoch [1/2], Step [119/125], Loss: 11.1581
Epoch [1/2], Step [120/125], Loss: 11.1093
Epoch [1/2], Step [121/125], Loss: 11.2513
Epoch [1/2], Step [122/125], Loss: 11.2373
Epoch [1/2], Step [123/125], Loss: 11.2124
Epoch [1/2], Step [124/125], Loss: 11.2389
Epoch [1/2], Step [125/125], Loss: 11.1801
Epoch [2/2], Step [1/125], Loss: 10.7445
Epoch [2/2], Step [2/125], Loss: 10.8582
Epoch [2/2], Step [3/125], Loss: 10.8365
Epoch [2/2], Step [4/125], Loss: 10.8676
```

```
Epoch [2/2], Step [5/125], Loss: 10.8328
Epoch [2/2], Step [6/125], Loss: 10.7977
Epoch [2/2], Step [7/125], Loss: 10.7842
Epoch [2/2], Step [8/125], Loss: 10.7500
Epoch [2/2], Step [9/125], Loss: 10.7775
Epoch [2/2], Step [10/125], Loss: 10.7274
Epoch [2/2], Step [11/125], Loss: 10.7382
Epoch [2/2], Step [12/125], Loss: 10.8177
Epoch [2/2], Step [13/125], Loss: 10.7340
Epoch [2/2], Step [14/125], Loss: 10.7861
Epoch [2/2], Step [15/125], Loss: 10.8514
Epoch [2/2], Step [16/125], Loss: 10.7232
Epoch [2/2], Step [17/125], Loss: 10.7148
Epoch [2/2], Step [18/125], Loss: 10.7753
Epoch [2/2], Step [19/125], Loss: 10.8083
Epoch [2/2], Step [20/125], Loss: 10.7867
Epoch [2/2], Step [21/125], Loss: 10.8254
Epoch [2/2], Step [22/125], Loss: 10.6846
Epoch [2/2], Step [23/125], Loss: 10.7995
Epoch [2/2], Step [24/125], Loss: 10.7813
Epoch [2/2], Step [25/125], Loss: 10.7550
Epoch [2/2], Step [26/125], Loss: 10.6536
Epoch [2/2], Step [27/125], Loss: 10.7785
Epoch [2/2], Step [28/125], Loss: 10.7899
Epoch [2/2], Step [29/125], Loss: 10.7827
Epoch [2/2], Step [30/125], Loss: 10.8224
Epoch [2/2], Step [31/125], Loss: 10.6975
Epoch [2/2], Step [32/125], Loss: 10.8134
Epoch [2/2], Step [33/125], Loss: 10.7919
Epoch [2/2], Step [34/125], Loss: 10.7350
Epoch [2/2], Step [35/125], Loss: 10.7990
Epoch [2/2], Step [36/125], Loss: 10.7659
Epoch [2/2], Step [37/125], Loss: 10.8071
Epoch [2/2], Step [38/125], Loss: 10.7476
Epoch [2/2], Step [39/125], Loss: 10.8150
Epoch [2/2], Step [40/125], Loss: 10.7318
Epoch [2/2], Step [41/125], Loss: 10.6941
Epoch [2/2], Step [42/125], Loss: 10.8114
Epoch [2/2], Step [43/125], Loss: 10.6926
Epoch [2/2], Step [44/125], Loss: 10.7644
Epoch [2/2], Step [45/125], Loss: 10.7038
Epoch [2/2], Step [46/125], Loss: 10.6646
Epoch [2/2], Step [47/125], Loss: 10.7669
Epoch [2/2], Step [48/125], Loss: 10.7524
Epoch [2/2], Step [49/125], Loss: 10.8422
Epoch [2/2], Step [50/125], Loss: 10.6346
Epoch [2/2], Step [51/125], Loss: 10.6743
Epoch [2/2], Step [52/125], Loss: 10.7773
```

```
Epoch [2/2], Step [53/125], Loss: 10.6997
Epoch [2/2], Step [54/125], Loss: 10.7581
Epoch [2/2], Step [55/125], Loss: 10.7539
Epoch [2/2], Step [56/125], Loss: 10.8540
Epoch [2/2], Step [57/125], Loss: 10.8517
Epoch [2/2], Step [58/125], Loss: 10.8162
Epoch [2/2], Step [59/125], Loss: 10.6741
Epoch [2/2], Step [60/125], Loss: 10.6128
Epoch [2/2], Step [61/125], Loss: 10.6435
Epoch [2/2], Step [62/125], Loss: 10.7391
Epoch [2/2], Step [63/125], Loss: 10.6853
Epoch [2/2], Step [64/125], Loss: 10.7893
Epoch [2/2], Step [65/125], Loss: 10.6112
Epoch [2/2], Step [66/125], Loss: 10.7046
Epoch [2/2], Step [67/125], Loss: 10.7454
Epoch [2/2], Step [68/125], Loss: 10.6987
Epoch [2/2], Step [69/125], Loss: 10.7797
Epoch [2/2], Step [70/125], Loss: 10.6814
Epoch [2/2], Step [71/125], Loss: 10.6396
Epoch [2/2], Step [72/125], Loss: 10.6336
Epoch [2/2], Step [73/125], Loss: 10.7421
Epoch [2/2], Step [74/125], Loss: 10.7333
Epoch [2/2], Step [75/125], Loss: 10.6771
Epoch [2/2], Step [76/125], Loss: 10.7071
Epoch [2/2], Step [77/125], Loss: 10.7274
Epoch [2/2], Step [78/125], Loss: 10.7268
Epoch [2/2], Step [79/125], Loss: 10.6301
Epoch [2/2], Step [80/125], Loss: 10.6994
Epoch [2/2], Step [81/125], Loss: 10.7052
Epoch [2/2], Step [82/125], Loss: 10.7522
Epoch [2/2], Step [83/125], Loss: 10.7524
Epoch [2/2], Step [84/125], Loss: 10.6951
Epoch [2/2], Step [85/125], Loss: 10.6525
Epoch [2/2], Step [86/125], Loss: 10.7219
Epoch [2/2], Step [87/125], Loss: 10.7220
Epoch [2/2], Step [88/125], Loss: 10.7887
Epoch [2/2], Step [89/125], Loss: 10.7162
Epoch [2/2], Step [90/125], Loss: 10.8428
Epoch [2/2], Step [91/125], Loss: 10.6732
Epoch [2/2], Step [92/125], Loss: 10.7556
Epoch [2/2], Step [93/125], Loss: 10.7036
Epoch [2/2], Step [94/125], Loss: 10.7011
Epoch [2/2], Step [95/125], Loss: 10.6429
Epoch [2/2], Step [96/125], Loss: 10.6809
Epoch [2/2], Step [97/125], Loss: 10.7399
Epoch [2/2], Step [98/125], Loss: 10.6251
Epoch [2/2], Step [99/125], Loss: 10.7206
Epoch [2/2], Step [100/125], Loss: 10.7361
```

```
Epoch [2/2], Step [101/125], Loss: 10.7774
Epoch [2/2], Step [102/125], Loss: 10.6525
Epoch [2/2], Step [103/125], Loss: 10.5635
Epoch [2/2], Step [104/125], Loss: 10.6669
Epoch [2/2], Step [105/125], Loss: 10.6310
Epoch [2/2], Step [106/125], Loss: 10.7037
Epoch [2/2], Step [107/125], Loss: 10.6674
Epoch [2/2], Step [108/125], Loss: 10.6808
Epoch [2/2], Step [109/125], Loss: 10.6894
Epoch [2/2], Step [110/125], Loss: 10.6366
Epoch [2/2], Step [111/125], Loss: 10.6017
Epoch [2/2], Step [112/125], Loss: 10.5954
Epoch [2/2], Step [113/125], Loss: 10.6715
Epoch [2/2], Step [114/125], Loss: 10.6839
Epoch [2/2], Step [115/125], Loss: 10.7835
Epoch [2/2], Step [116/125], Loss: 10.6577
Epoch [2/2], Step [117/125], Loss: 10.6520
Epoch [2/2], Step [118/125], Loss: 10.7136
Epoch [2/2], Step [119/125], Loss: 10.7215
Epoch [2/2], Step [120/125], Loss: 10.6323
Epoch [2/2], Step [121/125], Loss: 10.7197
Epoch [2/2], Step [122/125], Loss: 10.7409
Epoch [2/2], Step [123/125], Loss: 10.7179
Epoch [2/2], Step [124/125], Loss: 10.6961
Epoch [2/2], Step [125/125], Loss: 10.5523
```

Challenges and Solutions:

- Encountered a warning regarding tensor handling in the dataset class, which was resolved by directly using the tensors without re-wrapping them.
- Adjusted tokenization to align with the model's expected input format and to efficiently handle padding.
- Faced limitations in fully testing DDP and FSDP due to the need for a multi-GPU setup.

Conclusion:

The implementation and testing of the GPT-Neo model with the synthetic dataset provided insights into model setup and performance across different training paradigms. The Single GPU setup was successfully tested, while DDP and FSDP require further exploration in a suitable environment. # References: - Hugging Face Transformers Documentation - https://huggingface.co/EleutherAI/gpt-neo-125m - PyTorch Documentation

THANK YOU !!