Assignment 4

NATURAL LANGUAGE PROCESSING

1. Study the transformer code provided in the module Links to an external site.. Identify the points where code is different from the proposed architecture in Google's patent (provided in the module)

Ans: **Key elements and concepts outlined in the patent and comparison to the provided code.**

Google's Patent Overview (US10452978)

Encoder-Decoder Structure: The patent discusses a neural network system that includes an encoder neural network for processing input sequences into encoded representations and a decoder neural network for generating output sequences from those encoded representations.

Self-Attention Mechanism: It emphasizes the use of self-attention mechanisms in both the encoder and decoder parts of the neural network, which allows the model to weigh the importance of different parts of the input sequence when processing a particular element.

Positional Encoding: The document mentions the inclusion of positional embeddings to retain the order of the sequence, as self-attention mechanisms do not inherently consider the sequence order.

Parallel Processing: One of the advantages of the architecture described in the patent is the ability for parallel processing, as opposed to sequential processing found in RNNs, which significantly improves efficiency.

Provided Code Overview

The provided code describes a Transformer-based language model (TransformerBlockLM) with the following characteristics:

Multi-Head Attention: Both the encoder (MultiHeadAttention) and decoder sections implement multi-head attention, allowing the model to capture different aspects of the input data from different representation subspaces.

Positional Embeddings: The model includes positional embeddings to ensure the model accounts for the order of tokens in the input sequence, aligning with the patent's description.

Feedforward Networks: After the attention steps, the model employs position-wise feedforward networks (MLP), which is consistent with the Transformer architecture described in the patent.

Layer Normalization: The model applies layer normalization, which is a common practice in Transformer models to stabilize the training process.

Residual Connections: Residual connections are utilized around each sub-layer (including attention and feedforward networks), allowing for deeper models by mitigating the vanishing gradient problem.

Key Differences

- **Explicit Decoder Auto-regressive Nature**: The provided code explicitly details the auto-regressive nature of the decoder when generating text, which is an essential aspect for language models but not uniquely detailed in the patent description.
- **Custom Layers and Functions**: The code outlines specific implementations of transformer blocks (TransformerBlock), multi-head attention (MultiHeadAttention), and MLPs that are designed to fit within the broader language model structure. While the patent discusses these concepts at a high level, the code provides concrete implementations.
- **Training and Generation Procedures**: The code includes methods for training (fit) and generating text (generate), showcasing the practical application of the Transformer model for language tasks. These aspects go beyond the architectural design described in the patent and focus on operationalizing the model.

Overall, the provided code aligns closely with the architecture described in Google's patent, implementing a Transformer model with self-attention mechanisms, positional embeddings, and parallel processing capabilities. The key differences lie in the specific implementations, operational details, and explicit handling of the auto-regressive nature of the decoder for text generation.

2. Redesign the code to make it more intuitive. Give arguments why do you think your code is better.

Redesigned Code Snippet:

```
import torch

from torch import nn
import torch.nn.functional as F

class SelfAttentionHead(nn.Module):
    def __init__(self, embed_size, head_size):
        super().__init__()
        self.query = nn.Linear(embed_size, head_size, bias=False)
        self.key = nn.Linear(embed_size, head_size, bias=False)
        self.value = nn.Linear(embed_size, head_size, bias=False)

    def forward(self, x):
        Q = self.query(x)
        K = self.key(x).transpose(-2, -1)
        V = self.value(x)
        attention_weights = F.softmax(Q @ K / (K.size(-1) ** 0.5), dim=-1)
        return attention_weights @ V
```

```
class MultiHeadAttention(nn.Module):
    def __init__(self, embed_size, num_heads):
        super().__init__()
        head_size = embed_size // num_heads
        self.heads = nn.ModuleList([SelfAttentionHead(embed_size, head_size) for
_ in range(num_heads)])
        self.linear = nn.Linear(embed_size, embed_size)

    def forward(self, x):
        concatenated_heads = torch.cat([head(x) for head in self.heads], dim=-1)
        return self.linear(concatenated_heads)

class TransformerBlock(nn.Module):
    def __init__(self, embed_size, num_heads, forward_expansion):
        super().__init__()
        self.attention = MultiHeadAttention(embed_size, num_heads)
        self.norm1 = nn.LayerNorm(embed_size)
        self.mlp = nn.Sequential(
            nn.Linear(embed_size, forward_expansion * embed_size),
            nn.ReLU(),
            nn.Linear(forward_expansion * embed_size, embed_size)
        )
        self.norm2 = nn.LayerNorm(embed_size)

    def forward(self, x):
        attention = self.attention(x)
        x = self.norm1(x + attention)
        forward = self.mlp(x)
        return self.norm2(x + forward)

class TransformerLM(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.embed = nn.Embedding(config.vocab_size, config.embed_size)
        self.positional_encoding = nn.Parameter(torch.zeros(1, config.max_len,
config.embed_size))
        self.layers = nn.ModuleList([TransformerBlock(config.embed_size,
config.num_heads, config.forward_expansion) for _ in range(config.num_layers)])
        self.to_logits = nn.Linear(config.embed_size, config.vocab_size)

    def forward(self, x):
        x = self.embed(x) + self.positional_encoding[:, :x.size(1)]
        for layer in self.layers:
            x = layer(x)
```

```
        return self.to_logits(x)

# Configuration object
config = {
    "vocab_size": 10000,
    "embed_size": 512,
    "num_heads": 8,
    "forward_expansion": 4,
    "num_layers": 6,
    "max_len": 100,
}

model = TransformerLM(config)
```

Arguments on how re-designed code is better than the provided code:

1. **Introduction of a Configuration Object**

Original Code: Hyperparameters are directly passed as arguments to the model's constructor. This method scatters hyperparameter definitions throughout the code, adjusting cumbersome and error prone.

Redesigned Code: Uses a single configuration object (config) that centralizes all hyperparameters. This approach makes it much easier to modify, read, and reuse hyperparameter values, facilitating experimentation and configuration management.

2. Modularity and Class Structure:

Original Code: Contains deeply nested classes within the TransformerBlockLM class. While structurally sound, this nesting can make it difficult to navigate the code and reuse individual components in other contexts.

Redesigned Code: Decomposes the model into separate, top-level classes (SelfAttentionHead, MultiHeadAttention, TransformerBlock, and TransformerLM). This separation enhances clarity, promotes code reuse, and adheres to the principle of single responsibility.

3. Self-Attention and Multi-Head Attention Mechanism:

Original Code: Implements self-attention and multi-head attention mechanisms within nested classes. The logic, especially for the attention mechanism, is somewhat convoluted, making it hard to follow.

Redesigned Code: Provides a clear and concise implementation of self-attention in the SelfAttentionHead class and multi-head attention in the MultiHeadAttention class. This simplification makes the mechanism more understandable and easier to debug or extend.

4. Positional Encoding:

Original Code: The use of positional encoding is mentioned but not explicitly shown in the provided code. It's unclear how it's integrated into the model's architecture.

Redesigned Code: Introduces positional encoding as a learnable parameter (self.positional_encoding), clearly adding it to the input embeddings. This explicit approach clarifies the role of positional encoding in the model.

5. Simplified Forward Pass:

Original Code: The forward pass, especially the logic involving the combination of outputs from the multi-head attention and feedforward network, can be hard to trace due to the nesting of operations and classes.

Redesigned Code: Streamlines the forward pass in both the TransformerBlock and TransformerLM classes. Each step of the process is clearly delineated, improving readability and making the flow of data through the model more apparent.

6. PyTorch Best Practices:

Original Code: Does not clearly separate model definition and data handling and could better leverage PyTorch functionalities like ModuleList.

Redesigned Code: Makes use of nn.ModuleList for the layers in the transformer model, adhering to PyTorch best practices for dynamic lists of layers. It also paves the way for more efficient memory usage and parallel computation.

2**. Apply Transformer code provided in the module to train a language models that generates financial discourse in Warren Buffet's style. Train you model using Warren Buffets's Annual Letters to shareholders Download Warren Buffets's Annual Letters to shareholders.**

Evaluate performance in terms of model perplexity:

These are the results and output text generated after training the model.

```
params 1115739
iter 0: train 5.933777332305908 val 5.9342451095581055
iter 1000: train 1.5287889242172241 val 1.6506363153457642
iter 2000: train 1.3243008852005005 val 1.5202888250350952
iter 3000: train 1.2009848356246948 val 1.497991681098938


 we well, I believe this affed in oursuan the meetings
at least three. At these time of our best like. If put hist I way to first acquisition of senses as well. Misantly, we paniblac with 19 end $32 in 2001. We was grot 4

Finance as the Custock is Invelormance. Our flexactors have relibutor of goeshing second companies. Winston both majs core of leading man achier's grown and 7% of the
that was will have likely find from
our job Shace should have a versuspperator of option self-thh prospects on positions,
two then paid has depel generally for our over the period place. A surprise a; outstanding on urstagbles to Europe-valut on this debabled.
Thoughts and cash. Our should he avought
specialls this panic completted conscmentracts, and a lyst manch star.

Naturally, our Getributed the insure we wouldn't row intentiable to emotch- equal, loss will gold public exist, Charlie Mourch $16 billion. In
```

The perplexity P can be calculated using the formula:

$$P = e^{\text{(Average Negative Log-Likelihood)}}$$

The Loss final_val_loss = 1.497991681098938

```
import math

final_val_loss = 1.497991681098938
perplexity = math.exp(final_val_loss)
print(f'Perplexity of the model: {perplexity}')
```

```
Perplexity of the model: 4.472697441520669
```

The perplexity of 4.472697441520669 indicates that, on average, the model is as confused as if it had to choose uniformly and independently among 4.47 possibilities for the next word in the sequence. In other words, when predicting the next word, the model performs almost as well as if it had 4.47 options to choose from at each step, given the true distribution.

Discuss the most impressive text your model generated. What are the high impact design choices behind the generated text.

This is the output text generated using 2000 iterations:

```
params 1115739
iter 0: train 6.013772010803223 val 6.033511161804199
iter 1000: train 1.528861165046692 val 1.642070293426513 7

of, ever attail credited from this by arget
use every gear. His day far
per-interest in in equal reexception. Each News, NFM an
inteasts is mort, tha acquisition and as different instated.")
Re such same that the changes income. State we walls * po

Carried to this hamp., and Accountfries (perty twenty


Berkshire cauly beinaped. "Derge did said be, a bears loses of
$69,839

Nebred, I both Crite again 2, etrons 2, MidAmerican Energy 130, 300

Income textile 1999 229


(222) Remple


36

Rooks. Figures they liabilities. centials as lotely pre-tax earnearned both price again from 20.3% at 9,619

22,186, spermite pricis, our Operate 13 63

(Netwozes provite zero less time. Bith other largest matter the shares.
We by soon that will Berkshire was do to wide very extate reauly by is a given year we week acquired it timportance and important. GAAP Markets 1989 1969 1970s, $41,111, Completing
Rets 2006 42

We burget Other likely need to mature that rate date sareed the are imp
```

This is the output text generated using 6000 iterations:

```
params 1115739
iter 0: train 5.38703727722168 val 5.39325475692749
iter 1000: train 1.5383687019348145 val 1.654220461845398
iter 2000: train 1.336533784866333 val 1.5249453783035278
iter 3000: train 1.2117353677749634 val 1.48215651512146
iter 4000: train 1.1198979616165161 val 1.4908063411712646
iter 5000: train 1.0273840427398682 val 1.510171175003051 8


1984 18.6 31.6 16.2

1982 44.4 31.7 18.2 31.6 16.7

1981 32.3 32.3 7.7 12.8

1993 14.3 10.2 (2.7)

1967 11.2 21.9 0.2 10.9

1993 14.3 11.0 30.9 (19.7)

39.36

1973 11.0 30.3 (2.7)

Credit largely dividends Relative


Berkshire on May 5 to $16 billion
with us to use.

Summed problems that needed to be limit.


2%


Almaha chance, Charlie and I will
celdit, card we hold aptly to though (Berkshire as a joint in stelerge the property-casualth
which hemptically receive from their products is far from its fisked by reputation,
always be that some shareholders of Berkshire has been
profitable. more than - company proceeds to go bet
surfacteries. Last year's a vooce
Berkshire's are goodwill - to strick say of which we would like there shringed the funds of the last moment this day quickly ressight has had been because of the page on —9. At Berkshire, blicket with get come to let us fo
the in Aside. However, then leadership of a pro
```

This is the output text generated using 4000 iterations:

```
params 1115739
iter 0: train 5.933777332305908 val 5.9342451095581055
iter 1000: train 1.5287889242172241 val 1.6506363153457642
iter 2000: train 1.3243008852005005 val 1.5202888250350952
iter 3000: train 1.2009848356246948 val 1.497991681098938


 we well, I believe this affed in oursuan the meetings
at least three. At these time of our best like. If put hist I way to first acquisition of senses as well. Misantly, we paniblac with 19 end $32 in 2001. We was grot 4

Finance as the Custock is Invelormance. Our flexactors have relibutor of goeshing second companies. Winston both majs core of leading man achier's grown and 7% of the
that was will have likely find from
our job Shace should have a versuspperator of option self-thh prospects on positions,
two then paid has depel generally for our over the period place. A surprise a; outstanding on urstagbles to Europe-valut on this debabled.
Thoughts and cash. Our should he avought
specialls this panic completted conscmentracts, and a lyst manch star.

Naturally, our Getributed the insure we wouldn't row intentiable to emotch- equal, loss will gold public exist, Charlie Mourch $16 billion. In
```

The most impressive text that is generated is with 4000 iterations which begins with "we well, I believe this affed in oursuan the meetings." This excerpt attempts to mimic the tone and topics Buffett might discuss, such as acquisitions, shareholder interests, and company performance.

The high-impact design choices that likely contributed to the generated text include:

1. **Self-Attention Mechanism**: This is key to understanding the relationships and relevance of different parts of the input, helping the model generate contextually relevant text.
2. **Multi-Head Attention**: It allows the model to capture different "views" of the data, which is essential for nuanced understanding and generation of complex financial discourse.
3. **Layer Normalization and Residual Connections**: These aid in stabilizing training and enable deeper models, which are necessary for capturing the long-range dependencies in financial reports.
4. **Vocabulary and Tokenization:** The selection of vocabulary and the strategy for tokenizing the input text are crucial. For instance, using a subword tokenizer could help the model better handle rare words and financial jargon.
5. **Training Data:** The choice and quality of the training data are paramount. In this case, using Warren Buffett's letters, which have a distinct style and rich financial lexicon, sets the stage for the model to learn to generate similar text.