Homework 3 - Language Modeling

Mudit Arora

1 Introduction

The goal of this homework is to design a language model and train it on the Penn Treebank (text-only) dataset. This problem is an unsupervised task, which means it is a type of machine learning that learns from data without human supervision.

1.1 Problem Formulation

Given a sequence of tokens $x_1, x_2, ..., x_t$, the model aims to predict the probability distribution of the next token x_{t+1}. Formally:

$$P(x_{t+1}|x_1, x_2, ..., x_t; \theta)$$

where θ represents the model parameters.

1.2 Dataset

The Penn Treebank dataset [Cod24] is a widely-used corpus in natural language processing, particularly for benchmarking language models. The dataset characteristics include:

- Training set: Approximately 912,344 tokens
- Validation set: 131,768 tokens
- Test set: 129,654 tokens

The dataset has a dictionary with three splits—train, validation, and test, and each example in each split is a dictionary with one key: sentence.

Example: ptb['train'][2]['sentence'] will print mr. <unk> is chairman of <unk> n.v. the dutch publishing group

2 Models

For this particular Homework problem, we'll be using a decoder-only transformer, which is a type of recurrent neural network (RNN) with Attention that focuses exclusively on generating output based on an input sequence.

ID	ppl
2	2.314
5	5.23
6	1.12

Table 1: Submission Example

2.1 Decoder-Only Architecture

The architecture follows the standard transformer decoder design [Wol23] with the following specifications:

- Multi-head self-attention layers (8 heads)
- Position-wise feed-forward networks
- Layer normalization and residual connections
- Dropout applied to attention and feed-forward layers

Key hyperparameters:

- Number of layers: 6
- Hidden dimension: 512
- Feed-forward dimension: 2048
- Attention heads: 8
- Maximum sequence length: 50
- Dropout rate: 0.0001
- Batch Size: 32

2.2 Architectural Advantages

The decoder-only transformer architecture offers several specific advantages for the PTB language modeling task:

1. **Masked Self-Attention[Vas+17]**: The causal masking in decoder blocks naturally enforces the autoregressive property required for language modeling, preventing information leakage from future tokens.

2. **Shared Parameters**: The architecture enables parameter sharing across layers while maintaining distinct positional information through:
    - Position-aware self-attention mechanisms
    - Learned positional embeddings
    - Residual connections preserving position information
3. **Efficient Training**: The architecture facilitates efficient training through:
    - Parallel processing of sequences
    - Effective gradient flow through residual connections
    - Stable optimization due to layer normalization

2.3 Embedding Method

The embedding method processes the Penn Treebank dataset by treating each sentence as a sequence of tokens. We use special tokens to mark the start (<s>) and end (</s>) of sentences, handle unknown words with <unk>, and use <pad> for padding shorter sequences to a fixed length. Each token is mapped to a numerical index using the PTB vocabulary, creating input-target pairs where the input is the sequence of tokens and the target is the same sequence shifted by one position. This creates a natural setup for next-token prediction, where the model learns to predict each word based on all previous words in the sentence.

2.4 Implementation

These are the steps that were taken to build this transformer[Al-23]:

- Multi-Head Attention: Computes the attention between each pair of positions in a sequence. It calculates attention scores, reshapes the input tensor into multiple heads, and combines the attention outputs from all heads. The forward method computes the multi-head self-attention, allowing the model to focus on some different aspects of the input sequence.

- Position-wise Feed-Forward Networks: The class initializes with two linear transformation layers and a ReLU activation function. The forward method applies these transformations and activation function sequentially to compute the output. This process enables the model to consider the position of input elements while making predictions.

- Positional Encoding: The class calculates sine and cosine values for even and odd indices, respectively, based on the scaling factor. The forward method computes the positional encoding by adding the stored positional encoding values to the input tensor, allowing the model to capture the position information of the input sequence.

- Decoder Layer: The Decoder Layer initializes with input parameters and components such as Multi Head Attention modules for masked self-attention and cross-attention, a Position Wise Feed Forward module, three layer normalization modules, and a dropout layer.

- Transformer Model: The Transformer class combines the previously defined modules to create a complete Transformer model. During initialization, the Transformer module sets up input parameters and initializes various components, including embedding layers for source and target sequences, a Positional Encoding module, and Decoder Layer modules to create stacked layers, a linear layer for projecting decoder output, and a dropout layer.

3 Experiments

Splitting and Data Imbalance:

- The dataset comes with predefined training, validation, and test sets. No additional data splitting was required as we used these established partitions:

- No special handling for data imbalance was implemented as this is a next-token prediction task where imbalance is inherent to natural language

Hyperparameter Selection:

- We experimented with different values for only epochs to find the best performing configuration.

- We also experimented with different models (Encoder-only Transformer[Kum23] and Encoder-Decoder Transformer)

Evaluation Methods:

- We calculated the training loss and validation perplexity

4 Results

The model with the following configuration performed the best:

- Epoch: 5
- Decoder-only Transformer
- Same hyper-parameters as mentioned before

4.1 Comparison of Different Hyper-parameters and Models

These are few of the comparisons, where we changed the Epoch, and different Models:

Epoch	Loss	Val ppl
1	3.6879	3.6363

Table 2: Epoch: 1 (Encoder-Decoder Transformer)

Epoch	Loss	Val ppl
1	5.7630	199.8328
2	5.1175	151.7837
3	4.8103	130.5019
4	4.5813	119.2175
5	4.3861	113.7173

Table 3: Best Model - Epoch: 5 (Decoder-only Transformer)

Epoch	Loss	Val ppl
1	5.7619	197.4525
2	5.1105	150.7355
3	4.8065	130.5559
4	4.5777	119.2927
5	4.3832	112.7127

Table 4: Epoch: 5 (Encoder-only Transformer)

Epoch	Loss	Val ppl
1	5.7637	198.1887
2	5.1108	150.5324
3	4.8102	130.0424
4	4.5817	119.0964
5	4.3881	112.1893
6	4.2139	110.8099
7	4.0486	109.5005
8	3.8898	111.6921
9	3.7344	112.7112
10	3.5819	118.5763

Table 5: Epoch: 10 (Decoder-only Transformer)

References

[Al-23] Mohammed Al-Araimi. *Build Your Own Transformer From Scratch Using PyTorch*. https://towardsdatascience.com/build-your-own-transformer-from-scratch-using-pytorch-84c850470dcb. Accessed: November 2024. 2023.

[Cod24] Papers with Code. *Papers with Code - Penn Treebank Dataset*. https://paperswithcode.com/dataset/penn-treebank. Accessed: 2024. 2024.

[Kum23] Saurabh Kumar. *Understanding Transformers: Encoder*. https://medium.com/@mr.sk12112002/understanding-transformers-encoder-1f269b1cc943. Accessed: November 2024. 2023.

[Vas+17] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[Wol23] Cameron R. Wolfe. *Decoder-Only Transformers: The Workhorse of LLMs*. https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse. Accessed: November 2024. 2023.