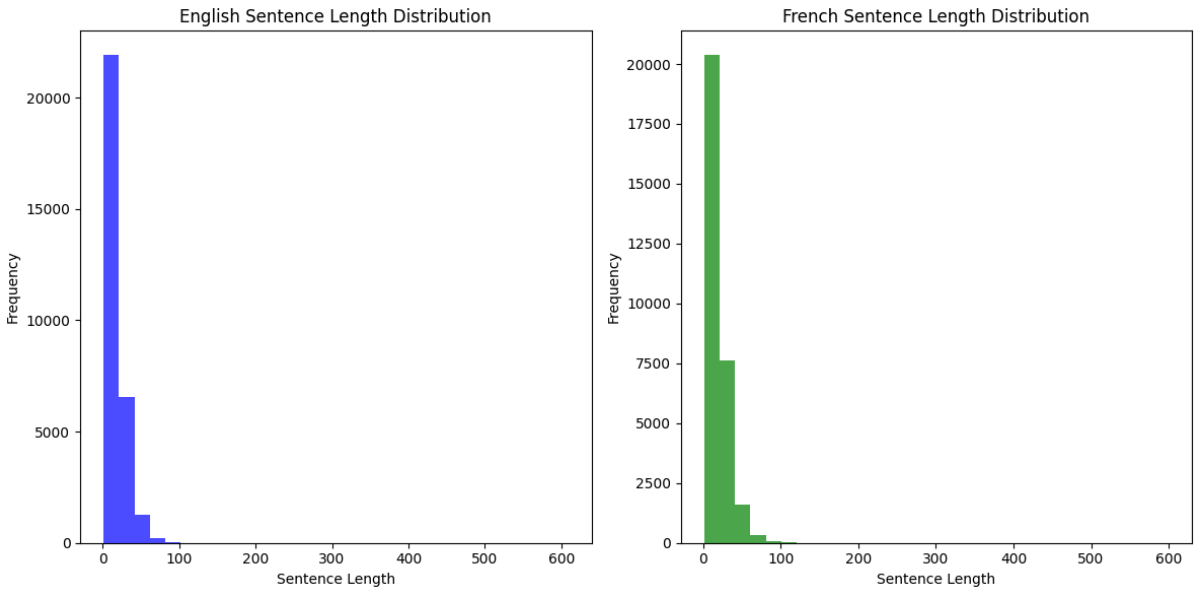
Report on Assignment 2: Transformers from Scratch

1. Sentence Length Distribution Analysis

- **Dataset Overview**:
  - `train.en`: 30,000 lines
  - `train.fr`: 30,000 lines
  - `dev.en`: 887 lines
  - `dev.fr`: 887 lines
  - `test.en`: 1,305 lines
  - `test.fr`: 1,305 lines
- **Length Statistics**:
  - **English**: Longest Sentence = 609, 95th Percentile Length = 42
  - **French**: Longest Sentence = 600, 95th Percentile Length = 44
  - **Chosen Max Sequence Length**: 48
- **Sentence Length Distribution**: To effectively train the model, sentence length distributions for both English and French were analyzed. A cutoff max sequence length of 48 was selected based on the percentile lengths to ensure that most sentences fall within this limit while not exceeding it significantly.
- **Distributions**:



---

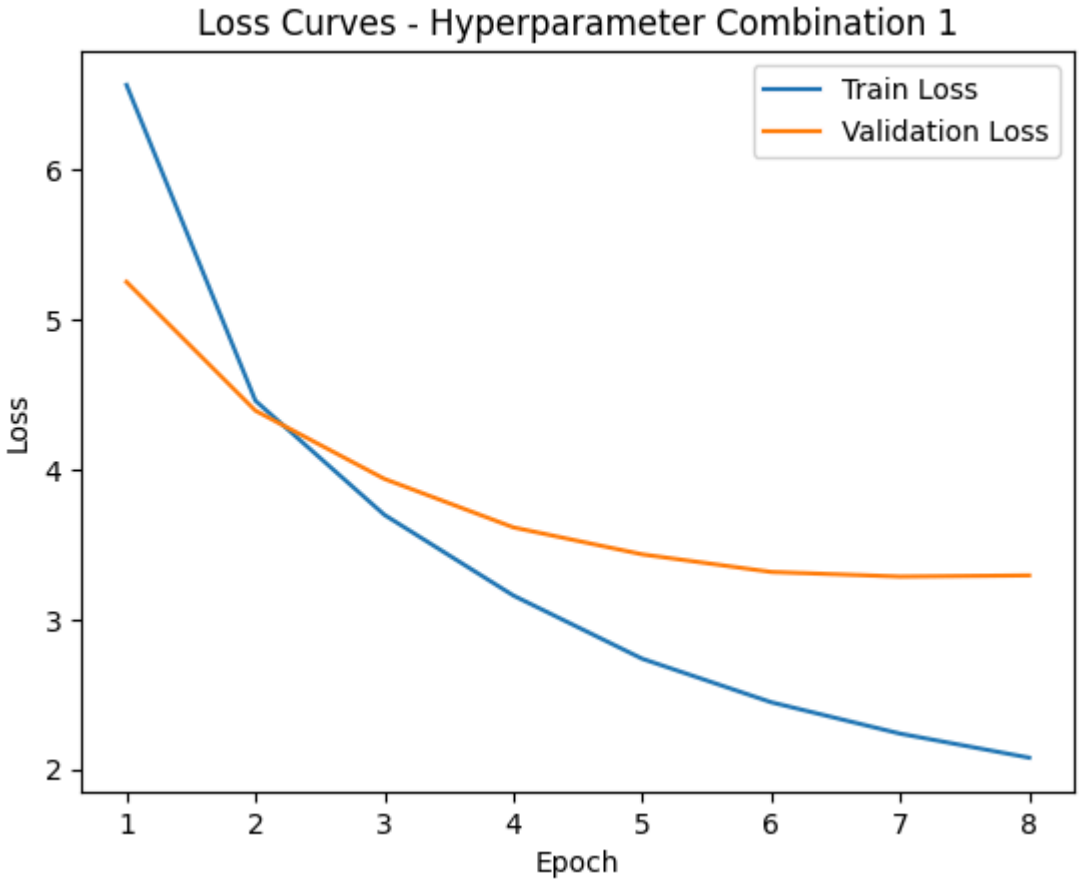
2. Model Performance Analysis

Hyperparameter Configurations and Their Results

Each configuration is evaluated in terms of training loss, validation loss, test loss, and BLEU scores across the train, validation, and test datasets.

2.1 Hyperparameter Combination 1

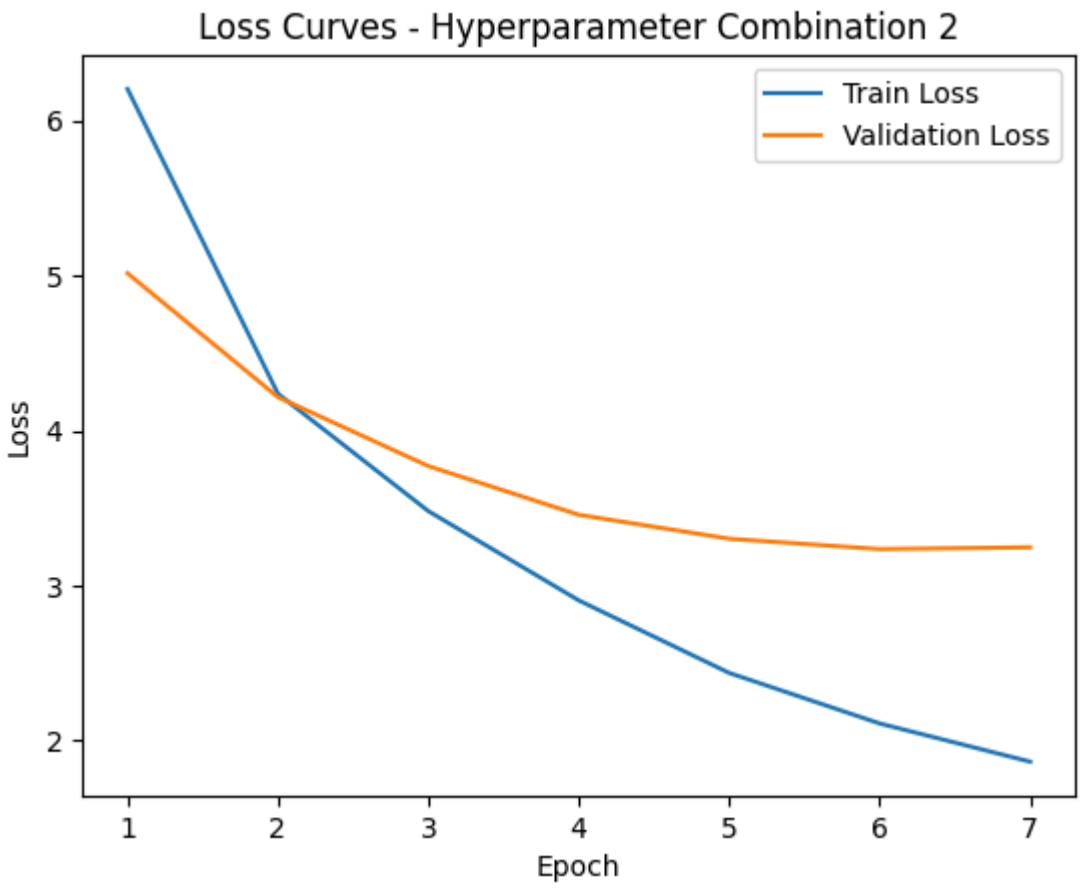
- **Parameters**: `{'d_embed': 256, 'd_ff': 512, 'h': 4, 'N_encoder': 2, 'N_decoder': 2, 'dropout': 0.1}`
- **Model Size**: 9,839,376 parameters
- **Performance**:
  - **Train Loss**: 2.2383
  - **Validation Loss**: 3.2868
  - **Test Loss**: 2.8929
  - **BLEU Scores**: `Train = 29.32`, `Validation = 12.81`, `Test = 16.26`



- **Analysis**: This model uses a small embedding size (256) and hidden dimension (512), resulting in a relatively small model size. While the train BLEU is decent, there is a significant gap in validation and test BLEU, indicating some overfitting or a model that lacks capacity for the task.

2.2 Hyperparameter Combination 2

- **Parameters**: `{'d_embed': 512, 'd_ff': 512, 'h': 8, 'N_encoder': 2, 'N_decoder': 2, 'dropout': 0.1}`
- **Model Size**: 22,812,432 parameters
- **Performance**:
  - **Train Loss**: 2.1100
  - **Validation Loss**: 3.2351
  - **Test Loss**: 2.8725
  - **BLEU Scores**: `Train = 30.72`, `Validation = 13.24`, `Test = 16.54`

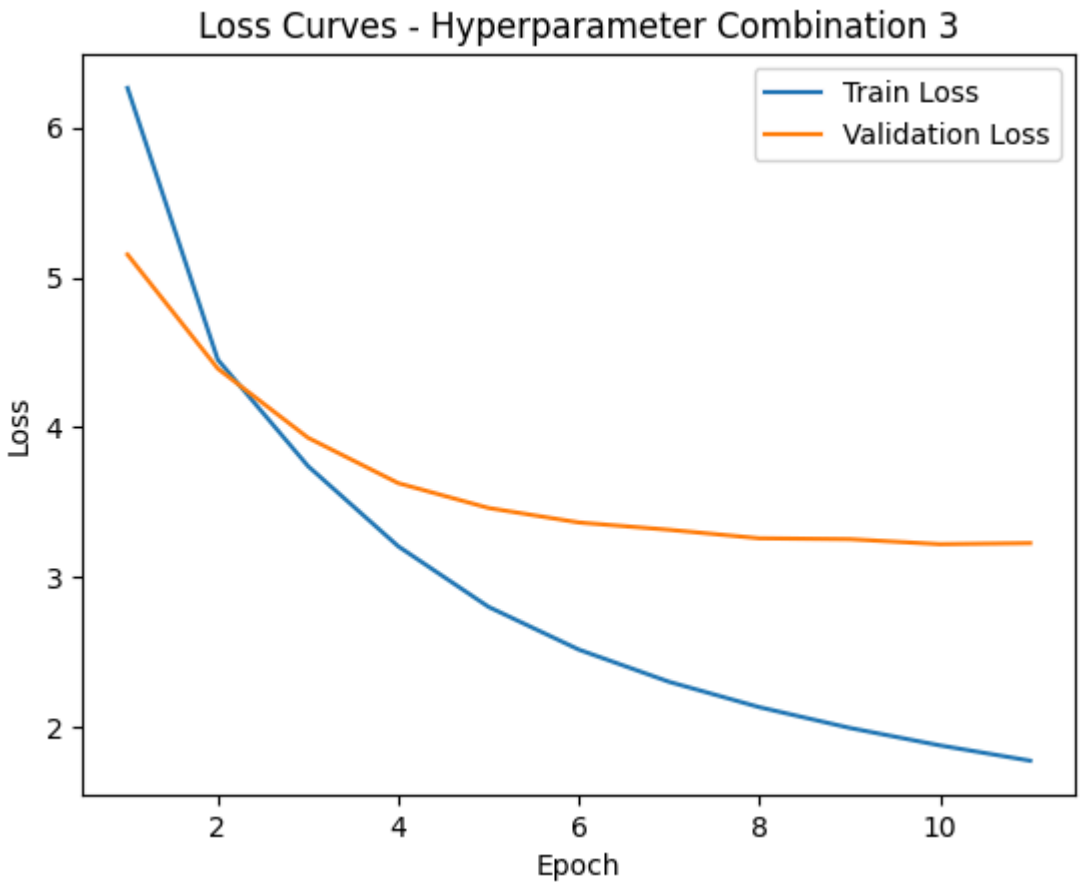


- **Analysis**: Increasing the embedding dimension (`d_embed`) to 512 and the number of attention heads (`h`) to 8 improves the BLEU scores slightly. The model has more

capacity to learn patterns in the data, reflected in better BLEU scores and lower losses compared to Combination 1.

2.3 Hyperparameter Combination 3

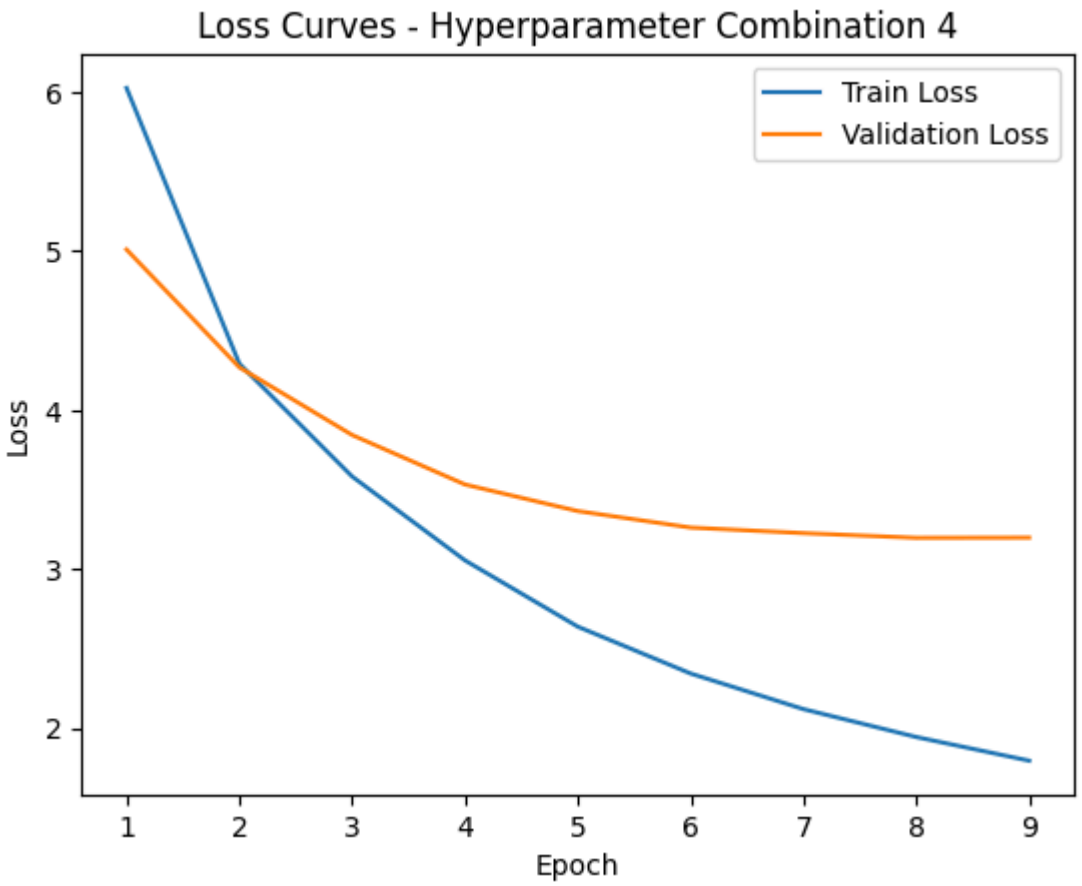
- **Parameters**: `{'d_embed': 512, 'd_ff': 1024, 'h': 8, 'N_encoder': 3, 'N_decoder': 3, 'dropout': 0.2}`
- **Model Size**: 30,168,848 parameters
- **Performance**:
  - **Train Loss**: 1.8754
  - **Validation Loss**: 3.2187
  - **Test Loss**: 2.8402
  - **BLEU Scores**: `Train = 31.04`, `Validation = 14.22`, `Test = 17.91`



- **Analysis**: Adding more layers to both encoder and decoder (3 each) and increasing the feed-forward dimension (`d_ff`) to 1024 shows further improvement. The higher dropout of 0.2 helps in regularization, leading to better generalization (higher validation and test BLEU).

2.4 Hyperparameter Combination 4

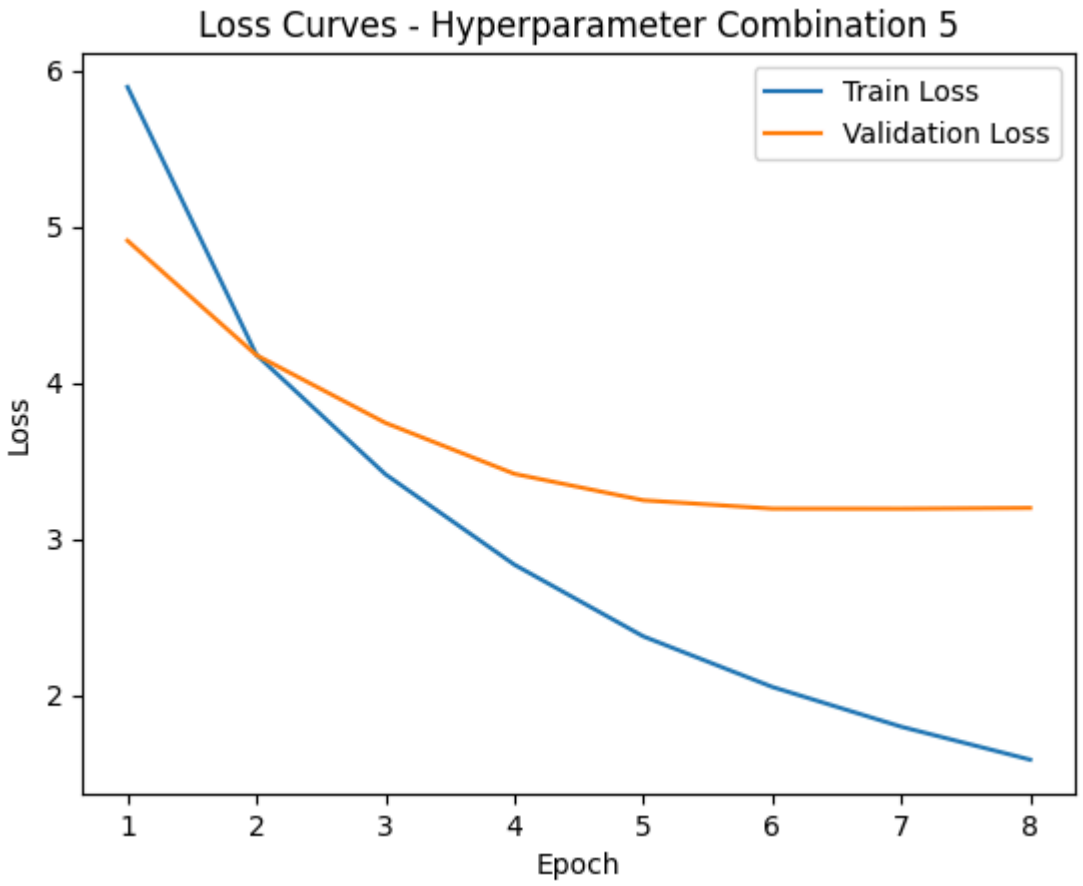
- **Parameters**: `{'d_embed': 768, 'd_ff': 2048, 'h': 8, 'N_encoder': 3, 'N_decoder': 3, 'dropout': 0.2}`
- **Model Size**: 61,766,416 parameters
- **Performance**:
  - **Train Loss**: 1.9440
  - **Validation Loss**: 3.1965
  - **Test Loss**: 2.7984
  - **BLEU Scores**: `Train = 32.35`, `Validation = 14.11`, `Test = 17.74`



- **Analysis**: Increasing the embedding size and feed-forward dimension significantly boosts the model size but also shows diminishing returns on BLEU scores. The model still improves but only marginally compared to Combination 3, suggesting it may be approaching optimal capacity for this task.

2.5 Hyperparameter Combination 5

- **Parameters**: `{'d_embed': 768, 'd_ff': 2048, 'h': 8, 'N_encoder': 4, 'N_decoder': 4, 'dropout': 0.1}`
- **Model Size**: 75,158,288 parameters
- **Performance**:
  - **Train Loss**: 1.7961
  - **Validation Loss**: 3.1928
  - **Test Loss**: 2.8042
  - **BLEU Scores**: `Train = 34.78`, `Validation = 14.33`, `Test = 17.78`



- **Analysis**: Adding more encoder and decoder layers provides a very slight increase in BLEU scores. However, the high model size does not significantly outperform Combination 4, indicating possible overfitting.

2.6 Best Hyperparameter Combination

- **Parameters**: `{'d_embed': 768, 'd_ff': 2048, 'h': 8, 'N_encoder': 3, 'N_decoder': 3, 'dropout': 0.1}`
- **Model Size**: 61,581,976 parameters
- **Performance**:

- **Train Loss**: 1.7276
- **Validation Loss**: 3.1818
- **Test Loss**: 2.8663
- **BLEU Scores**: `Train = 40.15`, `Validation = 13.99`, `Test = 18.45`

```
model_size: 61581976, train_set_size: 30016
train loss=4.766, lr=0.00011: 100%|          | 938/938 [02:14<00:00,  6.96it/s]
ep: 0: train_loss=5.89587, valid_loss=4.89752
train loss=3.694, lr=0.00022: 100%|          | 938/938 [02:19<00:00,  6.73it/s]
ep: 1: train_loss=4.10683, valid_loss=4.10351
train loss=3.412, lr=0.00034: 100%|          | 938/938 [02:21<00:00,  6.65it/s]
ep: 2: train_loss=3.35000, valid_loss=3.67579
train loss=2.525, lr=0.00030: 100%|          | 938/938 [02:21<00:00,  6.65it/s]
ep: 3: train_loss=2.76996, valid_loss=3.38056
train loss=2.625, lr=0.00026: 100%|          | 938/938 [02:21<00:00,  6.64it/s]
ep: 4: train_loss=2.30997, valid_loss=3.25373
train loss=1.892, lr=0.00024: 100%|          | 938/938 [02:21<00:00,  6.64it/s]
ep: 5: train_loss=1.98135, valid_loss=3.18586
train loss=1.789, lr=0.00022: 100%|          | 938/938 [02:21<00:00,  6.63it/s]
ep: 6: train_loss=1.72758, valid_loss=3.18177
train loss=1.543, lr=0.00021: 100%|          | 938/938 [02:21<00:00,  6.64it/s]
ep: 7: train_loss=1.51626, valid_loss=3.21269
train loss=1.362, lr=0.00020: 100%|          | 938/938 [02:21<00:00,  6.63it/s]
ep: 8: train_loss=1.33117, valid_loss=3.27250
```

```
test set examples:
4
src:  It also means tomorrow, after a day and a night.
trg:  Ça veut aussi dire demain, après un jour et une nuit.
pred: Cela signifie aussi demain, après une nuit et une nuit.
src:  The sun is coming up above the horizon. Sunrise.
trg:  Le soleil se lève au-dessus de l'horizon. L'aube.
pred: Le soleil sort au horizonz.
src:  A door. Put a plank inside the door, it's a door bolt.
trg:  Une porte. Une planche dans la porte, c'est un verrou.
pred: Un porte. Metraz une planche à l'intérieur, c'est une porte-ole.
train_loss: 1.7276, valid_loss: 3.1818, test_loss: 2.8663
test_bleu: 18.4489, valid_bleu: 13.9961 train_bleu: 40.1484
```

- **Analysis**: This configuration balances the model size and complexity, providing the best overall test BLEU score. The lower dropout (0.1) allows the model to fit better, and the embedding dimensions and feed-forward layers provide enough capacity for the model to learn the necessary features without overfitting.

---

4. Conclusion and Observations

- **Performance Across Hyperparameters**: As the model size and capacity increased (with larger `d_embed`, `d_ff`, more layers), the BLEU scores generally improved. However, beyond a certain point, the additional capacity led to diminishing returns.
- **Balancing Model Complexity and Regularization**: The best performance was achieved with a balanced approach, where the model had enough capacity to learn patterns but also maintained regularization to avoid overfitting.
- **Loss Curves and Overfitting**: Loss curves indicate that as the model becomes more complex, training loss decreases more rapidly, but validation loss does not always improve proportionately, suggesting overfitting in larger models.

5. Theory Questions

Question 1: Purpose of Self-Attention and Capturing Dependencies in Sequences

Purpose of Self-Attention:

Capturing Dependencies Across Tokens in a Sequence: The purpose of self-attention is to allow a model to consider the entire sequence of input tokens at each position when making predictions. This enables the model to capture relationships between tokens regardless of their distance from one another, addressing the limitations of older architectures like RNNs that struggle with long-range dependencies.

Parallelization of Computation: Unlike Recurrent Neural Networks (RNNs), which process tokens sequentially, self-attention allows for **parallel processing** of all tokens in a sequence. This makes transformers more computationally efficient and faster to train.

How Self-Attention Captures Dependencies:

1. **Contextual Relationships:** At each token position, self-attention computes a weighted sum of all other tokens in the sequence, allowing the model to consider context from all positions. The computed weights, called "attention scores," reflect the importance of each token in the sequence relative to the current token being processed.
2. **Parallel Processing and Global Context:** Unlike RNNs, which process tokens sequentially, self-attention allows parallel processing of tokens. By doing so, self-attention captures dependencies across the entire sequence at once, making it easier to understand both short-term and long-term dependencies.
3. **Weighting Mechanism:** Self-attention uses queries, keys, and values derived from the input embeddings. The "query" for a specific token is used to measure its compatibility with all other tokens' "keys," resulting in a set of attention scores. These scores are then used to compute a weighted average of the "values," providing a context-aware representation for each token. Thus, self-attention effectively allows

the model to "attend" to the most relevant parts of the sequence when producing an output for each token.

4. **Multi-Head Attention for Richer Contextual Understanding:** Self-attention is often applied in a multi-headed fashion, where multiple sets of queries, keys, and values are learned. Each "head" attends to different aspects of the sequence, allowing the model to capture richer contextual relationships.

Benefits of Self-Attention

- **Modeling Long-Range Dependencies**: By allowing each token to attend to every other token in the sequence, self-attention efficiently captures both local and global dependencies in the sequence, regardless of distance.
- **Contextual Representation**: Each token's representation is dynamically updated based on its context in the sequence, allowing the model to understand nuanced meanings (e.g. resolving ambiguities, handling synonyms).

Question 2: Why Use Positional Encodings in Transformers and Recent Advances

Why Use Positional Encodings in Transformers?

- **No Intrinsic Positional Information in Self-Attention:** Unlike RNNs, which inherently process sequences token by token, transformers operate on a set of input tokens without any inherent sense of order. The self-attention mechanism treats all tokens equally and does not have any mechanism to distinguish one token's position from another.
- **Need for Sequence Order Information:** To capture the order of the input sequence (e.g. the difference between "the cat sat" and "sat the cat"), transformers introduce positional encodings. These encodings provide the model with positional information about where each token is located in the sequence.

How Positional Encodings Are Incorporated in Transformers:

In the standard transformer architecture, positional encodings are added to the input token embeddings. Given an embedding vector `E_t` for token `t` and a positional encoding `PE_t`, the final embedding for that token is: `E'_t = E_t + PE_t`

- This augmented embedding is then passed through the transformer layers. The positional encodings, therefore, allow the model to distinguish tokens based on their position in the sequence.

Traditional Sinusoidal Positional Encodings

The original transformer paper uses fixed sinusoidal functions to generate positional encodings:

- **Sine and Cosine Functions**: Each position in the sequence is assigned a vector where even indices of the vector use sine functions and odd indices use cosine functions.
- These fixed encodings provide the transformer with a consistent way to represent positional information without learning any parameters.

Advantages of Sinusoidal Positional Encodings:

- **Fixed and Deterministic**: They are not learned parameters, which means they do not change during training and provide a consistent pattern across different sequences.
- **Smoothness and Generalization**: The sinusoidal pattern allows the model to extrapolate positions beyond the maximum sequence length seen during training.

Recent Advances in Positional Encodings:

1. **Learnable Positional Embeddings**:
    - Instead of using fixed sinusoidal functions, some transformer models learn positional embeddings from scratch.
    - **Difference from Sinusoidal Encodings:** These embeddings are directly learned through backpropagation and optimized for the task. This approach provides flexibility but may lack the generalizability of sinusoidal encodings to sequences longer than those seen during training.
2. **Relative Positional Encodings**:
    - Models like Transformer-XL and T5 use relative positional encodings, where instead of encoding the absolute position of tokens, the model encodes the relative distance between tokens.
    - **Benefit:** This approach captures the relative ordering between tokens better and is more suitable for tasks where context and relative positioning are more important than absolute positions. It can also generalize better to sequences longer than those seen during training.
3. **Rotary Positional Encodings (RoPE)**:
    - Introduced in models like GPT-NeoX, RoPE encodes positional information by rotating token embeddings in the complex plane.
    - **Benefit:** This approach allows for both absolute and relative positional information to be captured in a way that is more efficient and enables better generalization across different sequence lengths.
4. **ALiBi (Attention with Linear Biases)**:
    - ALiBi modifies the self-attention mechanism to incorporate a linear bias based on token positions, without adding explicit positional encodings.
    - **Benefit:** It provides an inductive bias to help the model understand position, improving performance on longer sequences and reducing computational overhead.
5. **Performer (Fourier Positional Encodings)**:
    - The Performer model uses Fourier-based features to encode positional information. These are derived from the Fourier Transform, offering a new way to encode positional data efficiently.

- **Benefit:** The Fourier encodings are computationally efficient and can handle long sequences effectively, which is a key advantage for tasks involving very large input sequences.