ANLP - Assignment 2: Transformers

2 - Theory Questions

2.1 Question 1

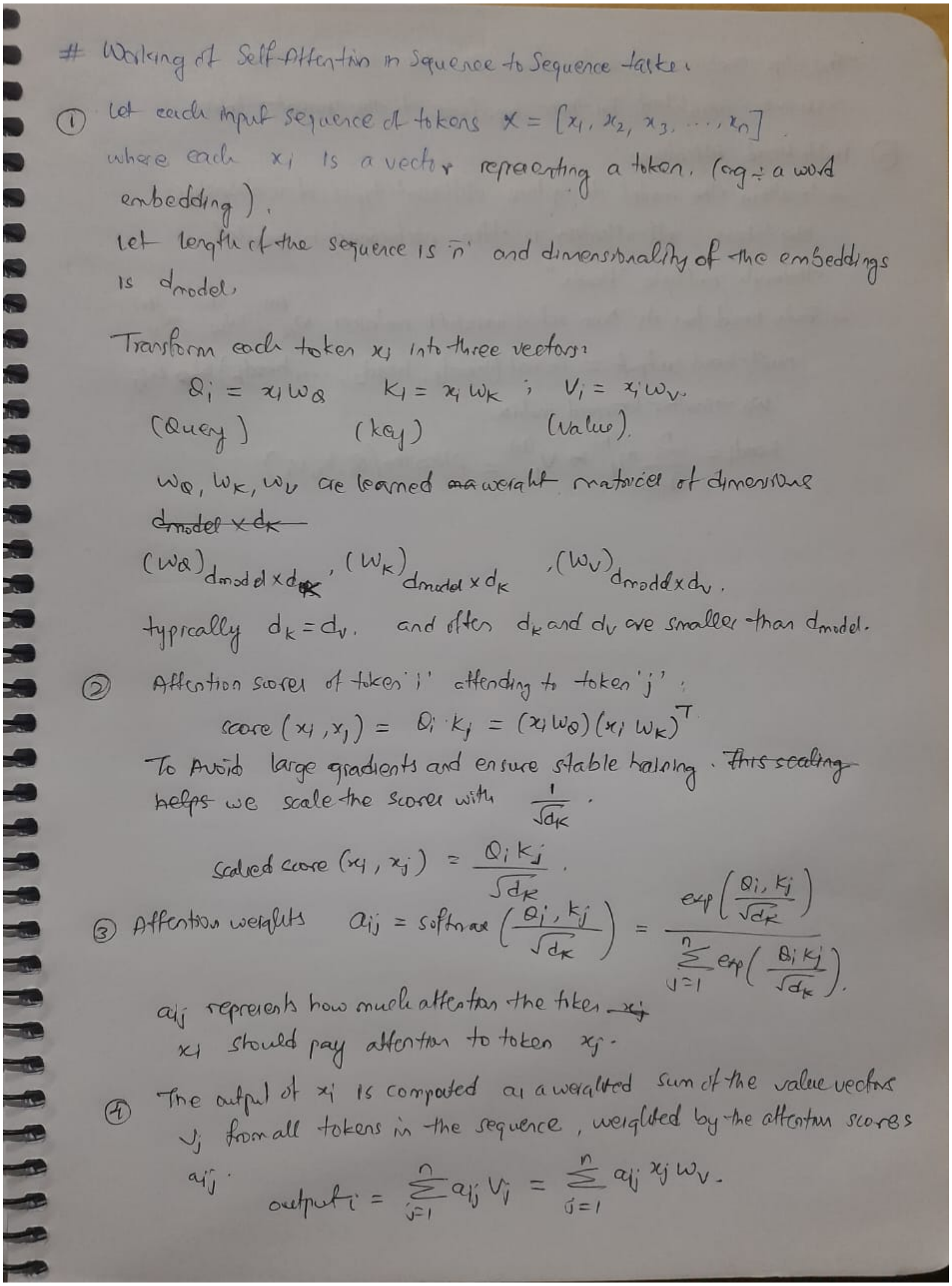
What is the purpose of self-attention, and how does it facilitate capturing dependencies in sequences?

Solution:

Self-Attention:

- Its primary purpose is to capture dependencies between elements in a sequence. Its regardless of the distance from one another, by enabling the model to “attend” to different parts of the sequence when processing each element.
- Main purposes of Self-attention:
    1. Capturing Dependencies:
        i. In Seq2Seq tasks such as Machine Translation, Text generation, each token can highly depend on other tokens in a sentence, even if they are apart.
        ii. Self-attention helps to model this dependencies more effectively than traditional architectures like RNNs and CNNs which suffer from problems such as Vanishing Gradient, and Exploding Gradient.
        iii. Although theoretical improvements were proposed on top of RNNs but practically they fail to capture long-term dependencies.
    2. Parallel Processing:
        i. Unlike RNNs, which process sequences step by step (i.e., sequentially), self-attention allows the model to process entire sequences in parallel.
        ii. This significantly speeds up the training time and inference. And also allows to leverage the performance of GPUs more efficiently.
    3. Context Awareness:
        i. Each element in a sequence is allowed to attend to all other elements in its context. This allows the model to consider the full context.
        ii. This allows model to learn from the full context, which tokens are important in order to generate the correct token.

Working of Self-attention:



Working of Self-Attention in Sequence to Sequence tasks.

① Let each input sequence of tokens $X = [x_1, x_2, x_3, \dots, x_n]$ where each x_i is a vector representing a token. (eg:- a word embedding).

Let length of the sequence is 'n' and dimensionality of the embeddings is d_{model}.

Transform each token x_i into three vectors:

$$Q_i = x_i W_Q \quad K_i = x_i W_K \; ; \quad V_i = x_i W_V.$$

(Query) (key) (Value).

W_Q, W_K, W_V are learned ~~...~~ weight matrices of dimensions ~~$d_{model} \times d_K$~~

$$(W_Q)_{d_{model} \times d_K}, \; (W_K)_{d_{model} \times d_K}, \; (W_V)_{d_{model} \times d_V}.$$

typically $d_K = d_V$. and often d_K and d_V are smaller than d_{model}.

② Attention scores of token 'i' attending to token 'j':

$$score(x_i, x_j) = Q_i \cdot K_j = (x_i W_Q)(x_j W_K)^T$$

To Avoid large gradients and ensure stable training. ~~This scaling helps~~ we scale the scores with $\frac{1}{\sqrt{d_K}}$.

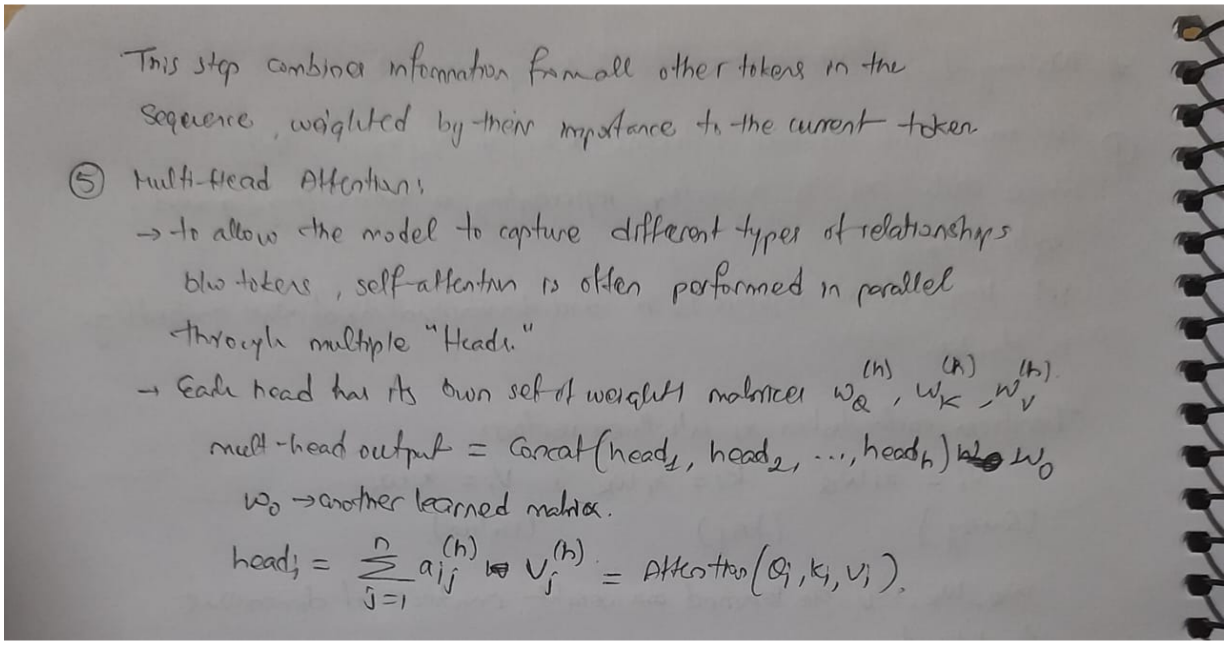
$$\text{Scaled score}(x_i, x_j) = \frac{Q_i K_j}{\sqrt{d_K}}.$$

③ Attention weights $a_{ij} = \text{softmax}\left(\frac{Q_i, K_j}{\sqrt{d_K}}\right) = \dfrac{\exp\left(\frac{Q_i, K_j}{\sqrt{d_K}}\right)}{\sum_{j=1}^{n} \exp\left(\frac{Q_i K_j}{\sqrt{d_K}}\right)}.$

a_{ij} represents how much attention the token ~~x_j~~ x_i should pay attention to token x_j.

④ The output of x_i is computed as a weighted sum of the value vectors V_j from all tokens in the sequence, weighted by the attention scores a_{ij}.

$$\text{output}_i = \sum_{j=1}^{n} a_{ij} V_j = \sum_{j=1}^{n} a_{ij} x_j W_V.$$



This step combines information from all other tokens in the sequence, weighted by their importance to the current token.

⑤ Multi-Head Attention:

→ to allow the model to capture different types of relationships b/w tokens, self-attention is often performed in parallel through multiple "Heads."

→ Each head has its own set of weight matrices $W_Q^{(h)}, W_K^{(h)}, W_V^{(h)}$.

mult-head output $=$ Concat$(head_1, head_2, \dots, head_h)$ W_O

W_O → another learned matrix.

$$head_i = \sum_{j=1}^{n} a_{ij}^{(h)} V_j^{(h)} = \text{Attention}(Q_i, K_i, V_i).$$

2.2 Question 2

Why do transformers use positional encodings in addition to word embeddings? Explain how positional encodings are incorporated into the transformer architecture. Briefly describe recent advances in various types of positional encodings used for transformers and how they differ from traditional sinusoidal positional encodings.

Solution:

Transformers use Positional Encoding in addition to word embeddings:

1. Unlike RNNs, transformers process all words in a sequence simultaneously rather than sequentially. This enables parallel processing, which enables faster training and inference.
2. But without additional information, the model would have a way to know the order of the words in the input sequence.
3. Positional encoding encodes the information of relative positions of each word.

Positional Encoding encodings are incorpoted to the input embeddings:

1. Word embeddings are created for each token in the input sequence.
2. Positional encodings are generated for each position in the sequence.These encodings are generated keeping the below points in mind:
    a. Positional embeddings should be same irrespective of sequence length or content.
    b. Shift in the embeddings space should not be large.
    c. Ability to estimate distance between two tokens should remain.
    d. PE must be deterministic based on some underlying rule.
3. The positional encodings are added element-wise to the word embeddings.
4. The resulting sum is used as the input to the first layer of the transformer.

Traditional Transformers in "Attention is all you need" used sinusodial positional encodings. These are generated using sine and cosine functions of different frequencies.

```
PE(pos, 2i) = sin(pos / 10000^(2i / d_model))
PE(pos, 2i + 1) = cos(pos / 10000^(2i / d_model))
```

pos → position
i → dimension
d_model → embedding dimension

These encodings have some desirable properties, such as allowing the model to extrapolate to sequence lengths longer than those seen during training.

Recent Advances in Positional Encodings:

1. Learned Positional Encodings:
    a. Instead of using predefined functions, the positional encodings are learned during training.
    b. This allows the model to potentially discover more optimal position representations for the specific task.
2. Relative Positional Encodings:
    a. Encoding relative distances between tokens rather than absolute positions.
    b. They have been shown to be particularly effective in tasks requiring understanding of relative positions, such as in music generation or code understanding.
3. Rotary Position Encodings (RoPE);
    a. Applies a rotation to the word embedding based on their position.
    b. Advantage: compatible with linear self-attention variants and has shown good performance in various tasks.
4. ALiBi (Attention with Linear Biases):
    a. Modifies the attention mechanism itself by adding a position-dependent bias term.
    b. It has shown strong performance, especially for extrapolating to longer sequences.
5. T5’s relative attention bias:
    a. uses learned relative attention biases, which are added to the attention scores.
    b. This allows the model to learn position-dependent attention patterns.

3.3 - Hyperparameter Tuning

Set 1:

- **N (number of layers)**: 4
- **h (attention heads)**: 8
- **d_model (embedding dimensions)**: 256
- **dropout**: 0.2
- **d_ff (feed-forward dimension)**: 1024

Reasons:

- **Lightweight and faster to train**: This smaller configuration (4 layers, 8 heads) is well-suited for scenarios where computational resources are limited or if you're working with smaller datasets.
- **Lower dropout**: A 0.2 dropout works well with a smaller model to prevent underfitting, allowing the model to fit the data more precisely.
- **Lower memory footprint**: With smaller `d_model` and `d_ff`, this setup reduces memory usage while still maintaining good performance on less complex tasks.

Set 2:

- **N (number of layers)**: 4
- **h (attention heads)**: 6
- **d_model (embedding dimensions)**: 384
- **dropout**: 0.35
- **d_ff (feed-forward dimension)**: 1536

Reasons?

1. **Fewer layers and attention heads**: A 4-layer architecture with 6 attention heads is sufficient for a small dataset. This configuration keeps the model simple and less prone to overfitting, as deeper models tend to memorize small datasets rather than generalize.

2. **Smaller `d_model`**: With a `d_model` of 384, you reduce the model's capacity, making it more suited to smaller datasets. It balances expressiveness without overloading the model with too many parameters.
3. **Dropout of 0.35**: This higher dropout rate is important for small datasets to avoid overfitting. It introduces regularization to ensure the model generalizes better on unseen data.
4. **Moderate `d_ff`**: The `d_ff` of 1536 provides a reasonable intermediate dimension in the feed-forward network, capturing sufficient feature interactions without being as large as the full 2048 or higher. It keeps training efficient but still powerful.

Set 3:

- **N (number of layers)**: 6
- **h (attention heads)**: 8
- **d_model (embedding dimensions)**: 512
- **dropout**: 0.3
- **d_ff (feed-forward dimension)**: 2048

Reasons:

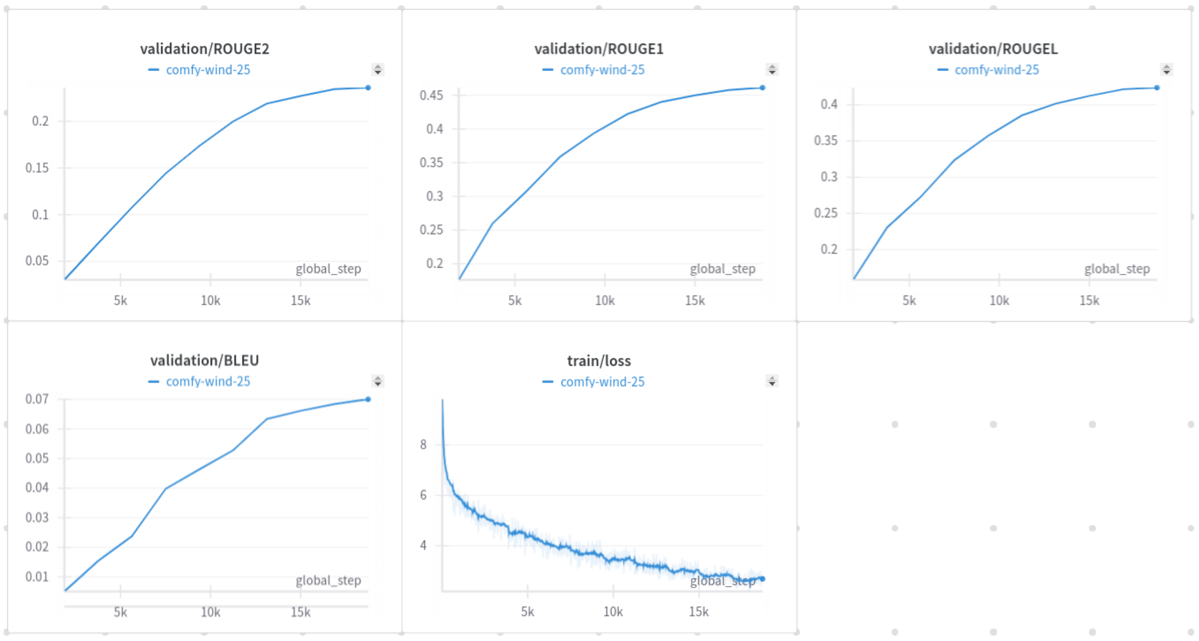
- **Balanced architecture**: A 6-layer Transformer with 8 attention heads and 512-dimensional embeddings has been shown to work well in tasks like machine translation, as seen in models like the original Transformer base configuration.
- **Moderate dropout**: Dropout of 0.3 is a good choice to avoid overfitting while allowing the model to generalize better.
- **Efficient and effective**: This setup offers a good trade-off between complexity and performance without being too computationally intensive.

Results:

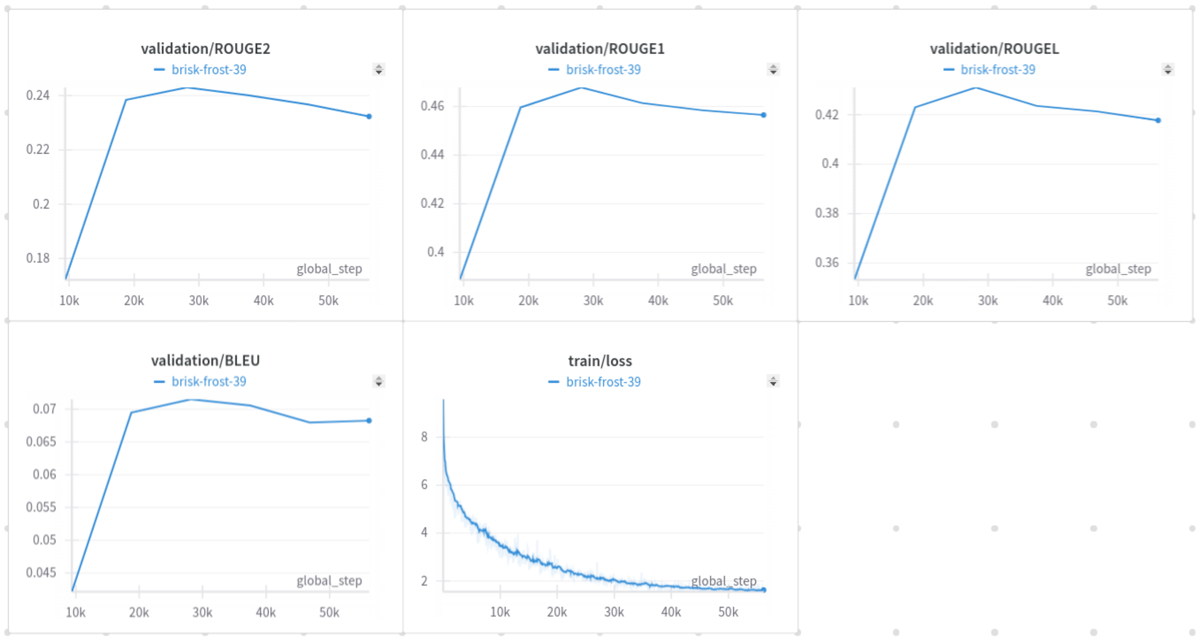
Set	Epochs	N	h	d_model	dropout	d_ff	Final Loss	Val BLEU
base_10	10	6	8	512	0.1	2048	2.65164	0.070075
base_30	30	6	8	512	0.1	2048	1.69787	0.068262 (m 0.0715)
1	10	4	8	256	0.2	1024	3.72393	0.049069
2	10	4	6	384	0.35	1536	3.72307	0.042818
3	10	6	8	512	0.3	2048	3.49993	0.045452

Loss Graphs:

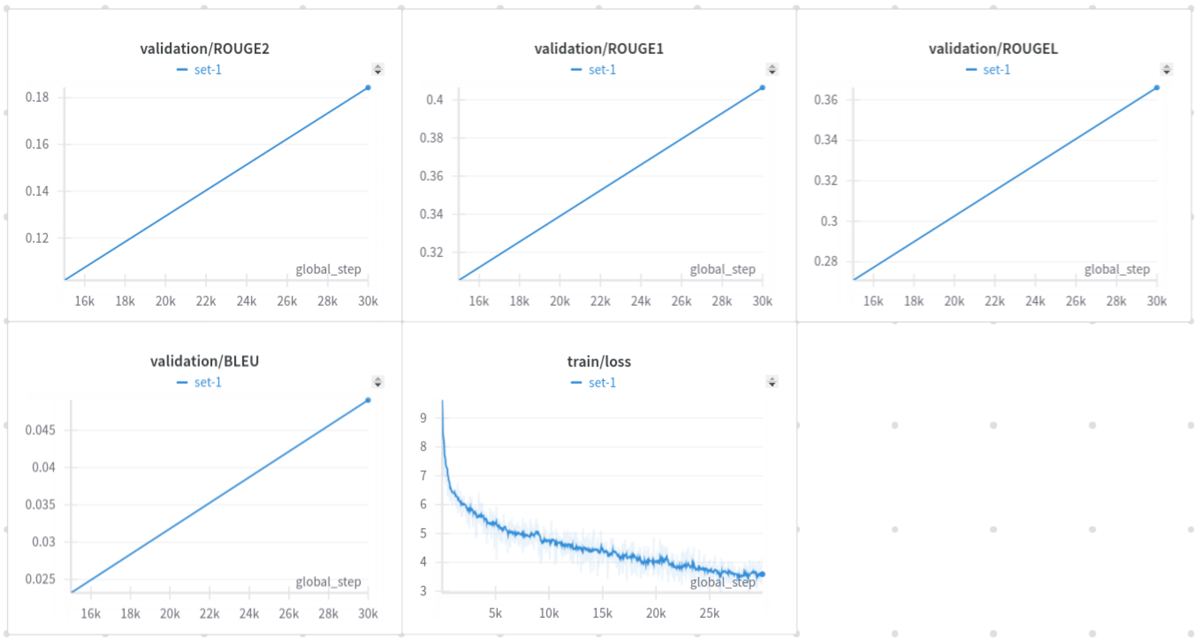
1. `base_10_epochs`:



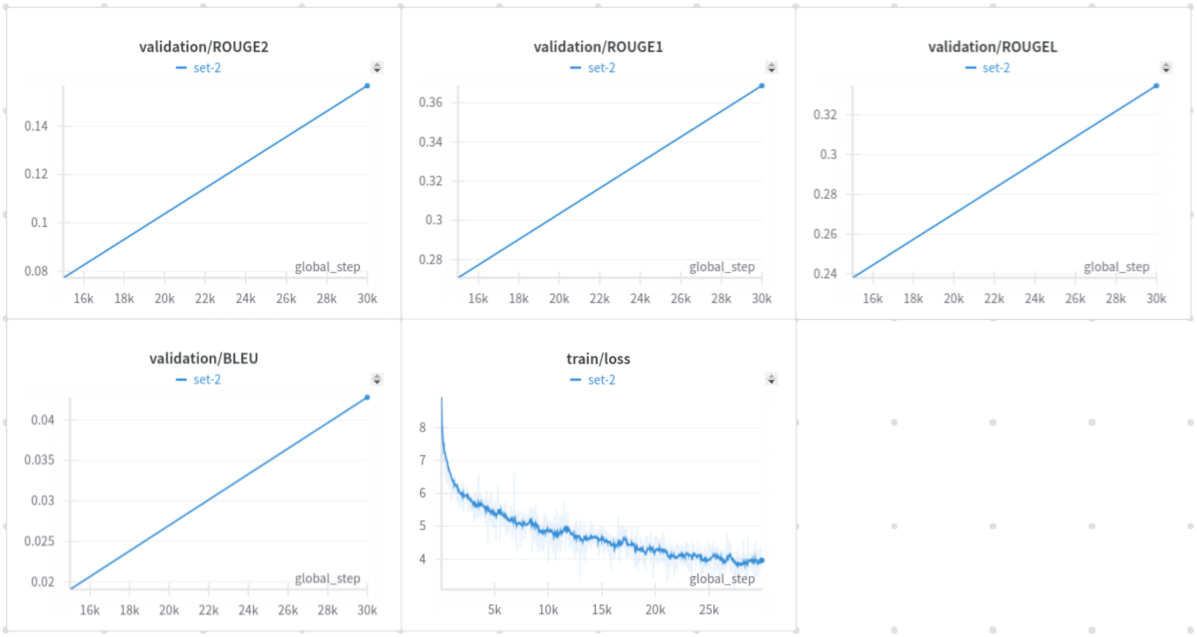
2. `base_30_epochs`:



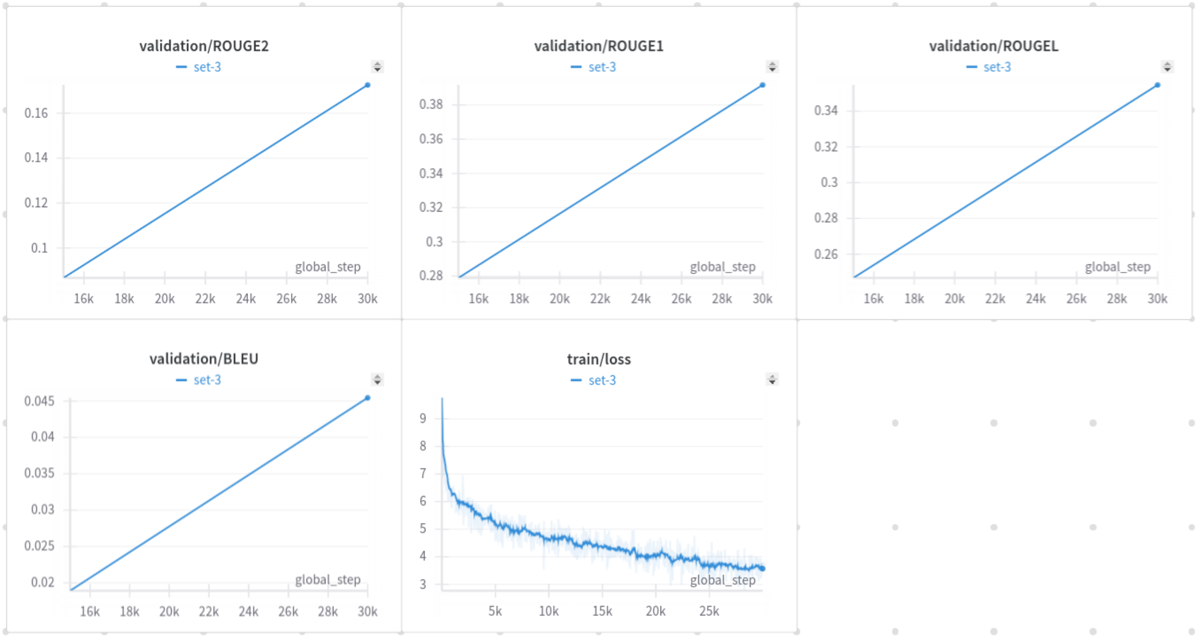
3. `set-1`:



4. `set-2`:

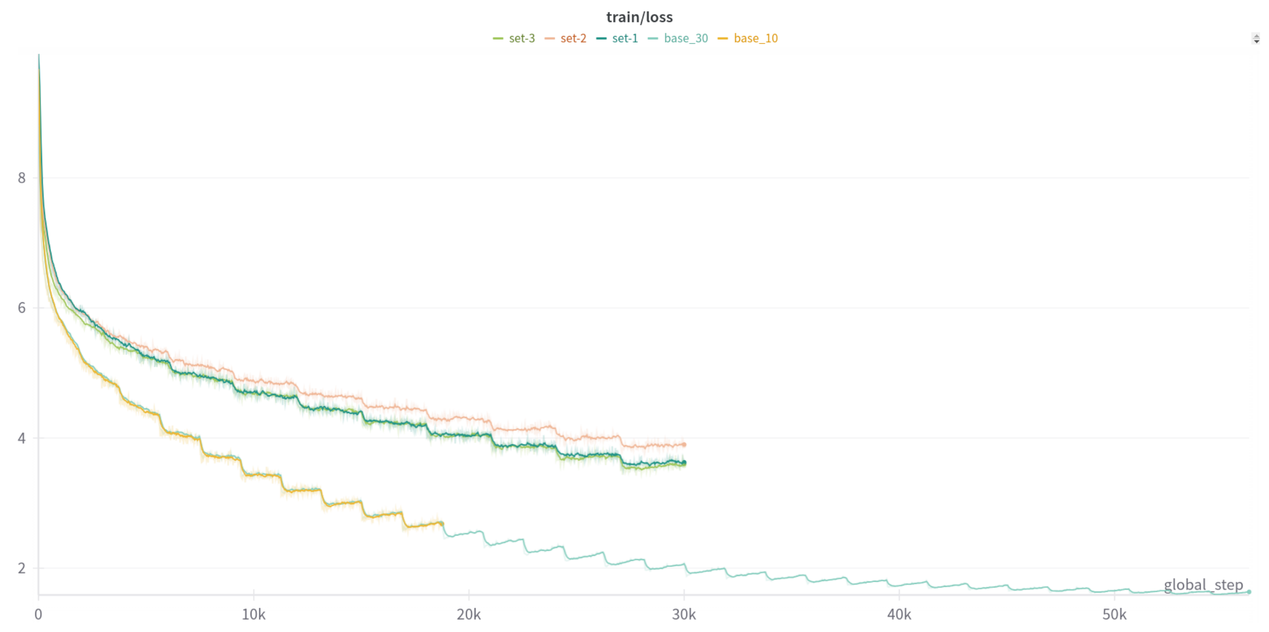


5. `set-3`:



Analysis:

Overall Train Loss trends:



Key Observations

1. **Impact of Training Duration**
   - Base_30 achieved significantly lower final loss (1.69787) compared to Base_10 (2.65164)
   - However, BLEU scores remained similar, which suggest potential overfitting despite lower loss
2. **Model Complexity Trade-offs**
   - Base configurations consistently outperformed Sets 1-3 in terms of BLEU scores
   - Simpler models (Set 1-2) showed higher final loss values despite various architectural changes
3. **Dropout Effects**
   - Higher dropout (0.35 in Set 2) led to slightly worse performance compared to moderate dropout
   - The base configuration with lowest dropout (0.1) achieved the best BLEU scores
4. **Architecture Sizing**
   - Reducing layers from 6 to 4 (Sets 1 & 2) resulted in decreased performance
   - Maintaining 6 layers but increasing heads and dropout (Set 3) didn't improve over base configuration

Learning Curve Analysis

1. **Base_10**: Shows steady decline in loss (with loss slightly increasing at the end of an epoch)
2. **Base_30**: Exhibits continued improvement beyond 10 epochs
3. **Sets 1-3**: All show higher volatility in training curves

Hyperparameter Impact

1. **Number of Layers (N)**
   - 6 layers consistently outperformed 4 layers
   - Suggests task complexity requires deeper architecture
2. **Attention Heads (h)**
   - Increasing heads from 6 to 8 didn't yield significant improvements
   - Base configuration with 6 heads achieved best results
3. **Embedding Dimensions (d_model)**

   - Reducing d_model to 256 or 384 resulted in worse performance
   - 512 dimensions appear optimal for the task
4. **Dropout**
   - 0.1 dropout (base) outperformed higher dropout rates
   - Suggests dataset might not require strong regularization

Theoretical Analysis of Transformer Hyperparameter Impact

1. Model Depth (Number of Layers)

Observed Trend

- 6 layers (Base, Set 3) outperformed 4 layers (Sets 1 & 2) in terms of Final Loss
- Base_10 BLEU: 0.070075 vs Set 1 BLEU: 0.049069

Theoretical Explanation

1. **Hierarchical Feature Learning**
   - Each transformer layer performs:
     - Self-attention to capture relationships
     - Feed-forward neural network for feature transformation
   - More layers allow for more sophisticated feature hierarchies:
     - Lower layers: local patterns, syntax
     - Middle layers: semantic relationships
     - Higher layers: complex, task-specific features
2. **Refinement of Representations**
   - Each layer refines representations from previous layers
   - 6 layers provide more opportunities for refinement
   - Can be likened to iterative refinement in optimization

2. Attention Heads

Observed Trend

- Base (6 heads) slightly outperformed Set 3 (8 heads)
- No linear improvement with more heads

Possible Explanation

1. **Multi-Aspect Representation**
   - Each head can focus on different aspects:
     - Some may focus on syntactic relationships
     - Others on semantic relationships
     - Still others on positional information
2. **Diminishing Returns**
   - Too many heads can lead to:
     - Redundant attention patterns
     - More complex optimization landscape
   - Theory suggests optimal head count relates to sequence length and embedding dimension

3. Embedding Dimensions (d_model)

Observed Trend

- d_model=512 (Base) outperformed d_model=256 (Set 1) and d_model=384 (Set 2)

Possible Explanation

1. **Representational Capacity**
    - Larger d_model means more dimensions to encode token information
    - Relationship to attention heads:

    ```
    Per-head dimension = d_model / num_heads
    Base: 512/6 ≈ 85 dimensions per head
    Set 1: 256/8 = 32 dimensions per head
    ```

    - Set 1's 32 dimensions per head may be insufficient for rich representations
2. **Impact on Feed-Forward Networks**
    - d_ff is typically 4x d_model
    - Smaller d_model constrains d_ff, limiting transformation capacity

4. Dropout Rate

Observed Trend

- Lower dropout (0.1 in Base) outperformed higher dropout (0.35 in Set 2)

Possible Explanation

1. **Regularization vs Information Flow**
    - Transformer layers rely on residual connections
    - Higher dropout can disrupt information flow between layers
    - Example calculation:

    ```
    Information preserved:
    Base (0.1 dropout): 90% per layer
    After 6 layers: 0.9^6 ≈ 53% of original signal

    Set 2 (0.35 dropout): 65% per layer
    After 4 layers: 0.65^4 ≈ 18% of original signal
    ```

2. **Dataset Size Considerations**
    - Higher dropout often needed for:
        - Small datasets
        - Large models relative to data size
    - Better performance with 0.1 suggests dataset might be sufficiently large

5. Training Duration

Observed Trend

- Base_30 achieved lower loss but similar BLEU to Base_10

Theoretical Explanation

1. **Optimization Landscape**
    - Transformer loss landscape is highly non-convex
    - More epochs allow finding deeper minima
    - But deeper minima don't always generalize better

2. **Token-Level vs Sequence-Level Metrics**
    - Loss is token-level cross-entropy
    - BLEU is sequence-level metric
    - Improving token-level predictions may not improve sequence-level quality