INNOCENT KISOKA

January 20, 2025

1 Dataset (20 pts)

1.1 Dummy dataset loading

I Loaded the dummy dataset and analyze a single data item:

- **Type:** The data item is a tuple consisting of a graph G and its optimal tour.
- G is an instance of `networkx.Graph`, while the optimal tour is a Python list.

1.2 Description of Edge and Node Attributes

Edge Attributes:

- **Tour:** Represents the optimal sequence of edges forming the shortest path.
- **Weight:** Corresponds to the Euclidean distance between connected nodes.

Node Attribute:

- **Position** (x, y)**:** The x and y coordinates of each node.

1.3 Dataset Class Implementation

I implemented the dataset class and it returned:

- **X** : A tensor of size $n \times 2$ containing the coordinates of the nodes.
- **Y** : The optimal tour starting and ending at node 0.

1.4 Dataset and DataLoader Creation

Separate `Dataset` objects were created for training, validation, and testing, with corresponding `DataLoader` instances for efficient data management.

2 Model (35 pts)

2.1 Model Description

- **Encoder:** Processes the input graph nodes' coordinates.
- **Decoder:** Outputs the node sequences forming the tour.
- **Positional Encoding:** Used in the decoder to capture sequential dependencies.
- **Causal Masking:** Ensures the decoder output is conditioned only on past nodes.

2.2 Causal Masking

The causal mask ensures that the decoder's output depends only on the past nodes.

2.3 Explanations

2.3.1 Masking

Where: Causal masking is applied in the decoder to ensure that predictions depend only on past outputs and not on future nodes.

Why: This is essential to maintain the autoregressive nature of the decoder. Without masking, the model could access future nodes during training, leading to data leakage.

2.3.2 Positional Encoding in the Encoder

Omission: Positional encoding can be omitted in the encoder.

Why: In the TSP task, node coordinates inherently contain positional information. Adding positional encodings would be redundant and could introduce unnecessary noise.

2.3.3 Positional Encoding in the Decoder

Necessity: Positional encoding is required in the decoder.

Why: Unlike the encoder, the decoder operates on an abstract sequence of node indices, which lack inherent positional information. Positional encodings provide the sequential structure needed for meaningful predictions.

Training (25 pts)

3 Standard Training

3.1 Training Procedure

The standard training pipeline follows these steps:

1. I Loaded training and validation datasets using DataLoaders with a batch size of 32.

2. Initialize the transformer model (`TSPTransformer`), optimizer (AdamW), and loss function (CrossEntropyLoss).

3. Perform forward and backward passes for each batch in the training dataset.

4. Update model weights after each batch using the optimizer.

The loss function is computed by shifting the target sequence `y` as shown in the model diagram to account for teacher forcing.

3.2 Training Results

The training and validation losses were monitored across 10 epochs. The results are as shown in Figure 1.

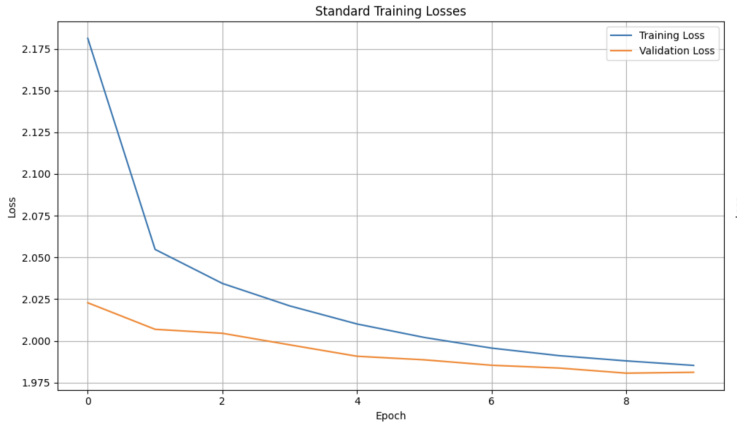

Figure 1: Training and validation losses for standard training.

The validation loss closely follows the training loss, indicating no significant overfitting.

4 Training with Gradient Accumulation

4.1 Training Procedure

Gradient accumulation is used to simulate a larger batch size without increasing memory usage. The procedure involves:

1. Accumulating gradients for a fixed number of 4 steps.

2. Scaling the loss by the number of accumulation steps to ensure proper gradient updates.

3. Updating model weights only after the specified accumulation steps.

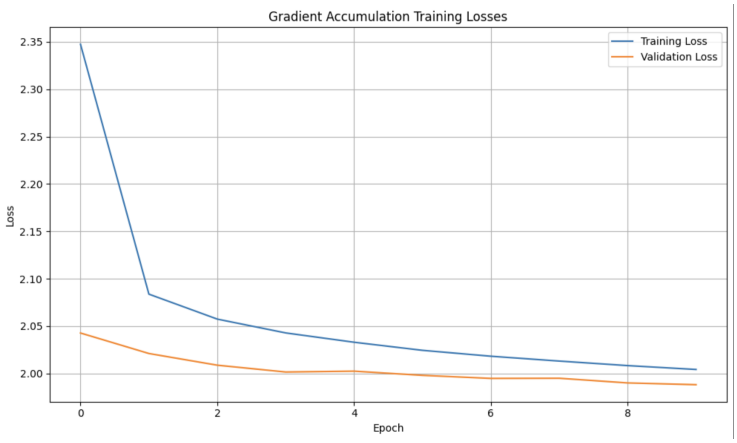

Figure 2: Training and validation losses for gradient accumulation.

4.2 Training Results

The training and validation losses for gradient accumulation are shown in Figure 2.

Gradient accumulation results in slightly smoother training loss curves compared to standard training, as shown in Figure 2.

4.3 Comparison of Training Strategies

The primary differences between standard training and gradient accumulation are:

- **Batch Size Simulation:** Gradient accumulation allows for effective training with a simulated larger batch size, which stabilizes gradient updates.
- **Training Stability:** Loss curves for gradient accumulation exhibit reduced fluctuations compared to standard training.
- **Memory Efficiency:** Gradient accumulation reduces memory usage, making it suitable for large models.

4.4 Overfitting Analysis

In both training methodologies, the validation loss closely follows the training loss, indicating that overfitting was avoided. The model achieved steady improvements in performance across epochs without significant divergence between training and validation losses.

5 Testing (15 pts)

Explanation of Transformer TSP Function (10 pts)

The `transformer_tsp` function uses a trained transformer model to solve the Traveling Salesperson Problem (TSP) by predicting the shortest tour. It encodes

TSP instances as graphs, where nodes represent cities and edges represent distances, leveraging self-attention layers to capture spatial relationships. At each step, the model predicts the next node to visit based on the current state of the tour, following a greedy strategy to complete the route.

Comparison with NLP Greedy Sampling

Both methods rely on step-by-step predictions for efficiency but differ in context:

- **NLP**: Operates on sequential data (e.g., sentences), where predictions depend on the prior token sequence.
- **TSP**: Operates on graph data, predicting nodes based on spatial relationships in the graph structure.

Boxplot Analysis of Optimality Gaps (5 pts)

The testing phase involved evaluating the following methods on the test dataset: The evaluation results are as shown in the Figure3

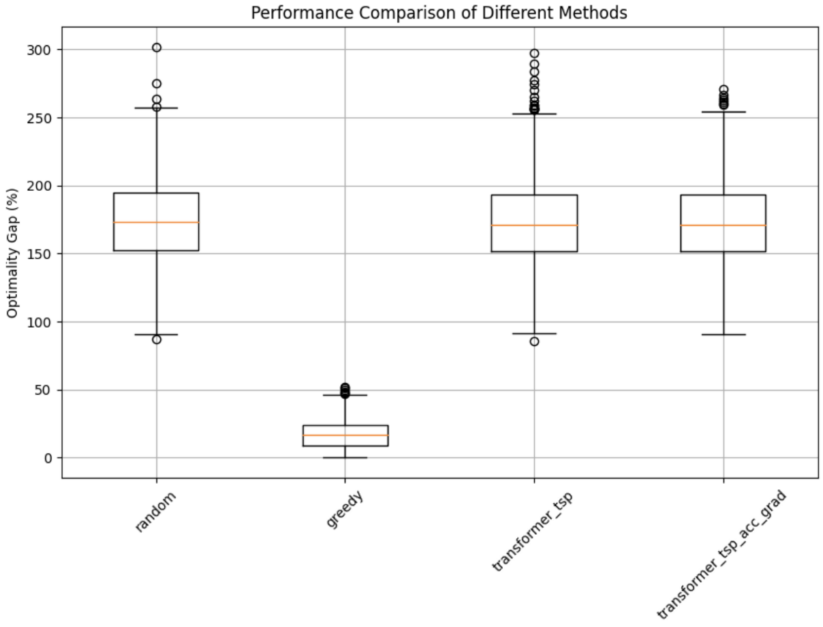

Figure 3: Boxplot Analysis Results

Critique (5 pts)

Model Architecture

- **Strengths:** Transformers can capture long-range dependencies and learn global patterns in the graph, potentially outperforming heuristics like the greedy algorithm.

- **Limitations:** Transformers are complex and computationally expensive, with a risk of overfitting and low interpretability, making it challenging to understand the model's decision-making.

Dataset Size and Diversity

- **Strengths:** A large and diverse dataset allows the model to generalize better.
- **Limitations:** A small or homogeneous dataset can lead to poor generalization, and a dataset without real-world diversity may hurt the model's performance on real-world graphs.

Generalizability to Larger Graphs

- **Strengths:** The transformer can theoretically handle larger graphs.
- **Limitations:** As graph size grows, the model faces significant computational challenges, including increased memory usage and slower processing due to the quadratic time complexity of transformers.

Generalizability to Graphs Without Coordinates

- **Strengths:** The model can work with abstract graph structures, making it flexible for graphs without spatial coordinates.
- **Limitations:** The absence of node coordinates may reduce the model's ability to make informed decisions, especially for spatially related problems.

Model Size

- **Strengths:** A larger model can capture more complex patterns.
- **Limitations:** Larger models risk overfitting and come with increased computational demands, especially for larger datasets or graphs.

Hyperparameter Choices

- **Strengths:** Hyperparameter tuning can improve model performance.
- **Limitations:** Hyperparameter optimization is time-consuming and sensitive to initial conditions, making it challenging to find the best configuration.

Other Factors

- **Training Time and Resources:** Transformers require significant training resources, which can be a limitation for large graphs.
- **Evaluation Metrics:** Focusing on the optimality gap is useful, but additional metrics like computation time and memory usage could offer a more complete assessment.