ED5310

ALGORITHMS IN COMPUTATIONAL GEOMETRY

END-TERM PROJECT REPORT

Submitted By:
Yegneswaran RV - ME21B220
Bhagvat Girish-ME21B043
Submitted To: Dr Ramanathan M

Direct PointerNet Implementation

Our 1st code implements a **Pointer Network (PointerNet)** to solve the **Convex Hull problem**. A Convex Hull is the smallest shape that can enclose a set of points in 2D space, and the model learns to predict the sequence of points forming this shape. Here's a breakdown of each class and its role:

1. EncoderBase

- **Purpose**: A template for encoders, providing a structure for creating custom encoders.
- **Structure**:
    - Acts as a base class for other encoders like `RNNEncoder`.
    - Contains a `forward` method, which must be defined in child classes.

2. RNNEncoder

- **Purpose**: Processes the input sequence (2D points) using an LSTM.
- **Structure**:
    - **Attributes**:
        - An RNN (e.g., LSTM) processes the sequence.
        - Supports variable-length sequences using a feature called `packing`.
    - **Methods**:
        - `__init__`: Sets up the RNN with parameters like number of layers, bidirectionality, etc.
        - `forward`: Processes input sequences and outputs a compressed representation (`memory_bank`) and final RNN states.

3. Attention

- **Purpose**: Helps the decoder focus on specific parts of the input sequence.
- **Structure**:
    - **Attributes**:
        - Type of attention (dot-product or a more general approach). Here dot product was used.
        - Layers to transform inputs and outputs.
    - **Methods**:
        - `score`: Calculates similarity between input points and decoder state.
        - `forward`: Computes attention weights and creates a context vector by focusing on relevant input points.

4. RNNDecoderBase

- **Purpose**: A template for creating RNN-based decoders.
- **Structure**:
    - **Attributes**: Defines an RNN structure for decoding.
    - **Methods**:
        - `__init__`: Sets up the RNN for decoding.
        - `forward`: Abstract method for processing decoder inputs (must be implemented in derived classes).

5. PointerNetRNNDecoder

- **Purpose**: A decoder that uses attention to predict the next point in the Convex Hull sequence.
- **Structure**:
    - **Attributes**:
        - An attention module to generate context vectors during decoding.
    - **Methods**:
        - `forward`: Uses the RNN and attention to predict the order of points step by step.

6. CHDataset

- **Purpose**: Loads and processes the Convex Hull dataset.
- **Structure**:
    - **Attributes**:
        - Stores maximum input/output sequence lengths.
        - Special markers for the start and end of sequences.
        - Data in preprocessed format.
    - **Methods**:
        - `__init__`: Loads the dataset and prepares input/output sequences with padding.
        - `_load_data`: Reads and formats the dataset from a file.
        - `__len__`: Returns the total number of examples in the dataset.
        - `__getitem__`: Retrieves a specific example (input/output sequences).

7. PointerNet

- **Purpose**: Combines all components (encoder, decoder, and attention) into a complete Pointer Network model.
- **Structure**:
    - **Attributes**:
        - An `RNNEncoder` for processing input points.
        - A `PointerNetRNNDecoder` for predicting the Convex Hull sequence.

- **Methods**:
  - `__init__`: Initializes the encoder and decoder.
  - `forward`: Processes the input through the encoder and generates predictions with the decoder.

8. PointerNetLoss

- **Purpose**: Defines how to calculate the loss during training.
- **Structure**:
  - **Methods**:
    - `__init__`: Sets up the loss calculation.
    - `forward`: Compares the model's predictions with the actual sequence and calculates the loss. Handles variable-length sequences with a mask.

Dataset Format:

Each line in the dataset has:

x1 y1 x2 y2 ... xn yn output i1 i2 ... im

- **xk yk**: Coordinates of input points.
- **i1 i2 ... im**: Indices of points forming the Convex Hull.

Key Components Summary:

1. **Encoders (EncoderBase, RNNEncoder)**: Convert input points into compact representations.
2. **Attention**: Helps focus on the most relevant input points for each output.
3. **Decoders (RNNDecoderBase, PointerNetRNNDecoder)**: Predict the sequence of Convex Hull points.
4. **Dataset Loader (CHDataset)**: Prepares training/testing data.
5. **PointerNet**: The full model combining encoder, decoder, and attention.
6. **PointerNetLoss**: Calculates how well the model performs.

This structure enables the Pointer Network to learn the sequence of points forming the Convex Hull for any given set of 2D points.

Drawback:

While predicting, the model's output has repeating indices. We tried to resolve this. To overcome this problem, we have decided on another approach.

MODIFIED APPROACH

Approach

- Input is a file with the coordinates of all points. The first neural Net called simple NN is used to predict whether a point is a part of the convex hull. The input is all points and the point under consideration as the first point. It tries to see if the point can be linearly separated from the other points, meaning that the point will indeed form a vertex of the convex hull. The model predicts 1 if the point is a hull vertex and 0 otherwise ideally and probability ranges from 0 to 1.
- Once the hull vertices are chosen from the given bunch of vertices, the points need to be displayed as sorted angularly with respect to an internal point like a centroid. A pointer network is trained for the same. The model looks at the input points and "predicts" the same number of points which are ordered and are approximately equal to the input points. Using a prediction model eliminates the chance of repetition of points in the sequences and the use of masks to prevent that, the issue we faced in the 1st approach.
- We have considered 5 points as the maximum limit while training and testing the network. The primary challenge we faced with higher point dataset was the high loading time for each epoch. Nevertheless the approach works for higher point dataset because instead of padding with 0 value arrays, we can pad with repeated vertices in the input. Eg: if the limit was 10 and we input 6 points, the rest 4 can be repetitions. While inputting in simple NN, we can choose the 4 from 5 of the vertices except the one under consideration and the model will still predict the linear separability of the 1st point.
- Below are the key classes and their corresponding main functions.

1. `CustomDataset` and `CustomDatasetTest`

These classes define custom datasets for training and testing, respectively.

`CustomDataset`:

- **Purpose**: Prepares the dataset by reading a file where each line contains input points and output indices. The dataset is a .txt file with input as coordinates(2D) separated by a space and the output hull sequence is a sequence of indices of the corresponding points(starting from 10
- **Functions**:
    - `_load_data`: Parses the file into input features (reordered 2D points) and labels (binary: 1 if an index exists in the output, 0 otherwise).

`CustomDatasetTest`:

- **Purpose**: Similar to `CustomDataset`, but creates multiple "rotated" versions of the input points for training. Basically, 10,000 sets of 5 points are given. For training we took 10000 points by labelling the 1st point as 1 if it's there in the hull and 0 if not.

- **Functions:**
    - `_load_data`: For each input-output pair, rotate the points by considering different starting orders.

2. `SimpleNN`

- **Purpose**: A simple feedforward neural network for binary classification (outputs probability using sigmoid). Probability is closer to 1 if the point forms a vertex of the convex hull(this point is linearly separable from the remaining points)
- **Functions**:
    - `fc1`: Fully connected layer (input → hidden).
    - `fc2`: Fully connected layer (hidden → smaller hidden).
    - `fc3`: Fully connected layer (hidden → output).
    - `sigmoid`: Converts the final output into a probability.
- **`forward` Method**: Processes the input sequentially through `fc1`, `fc2`, `fc3`, and applies `sigmoid`.

3. `PointerNet`

- **Purpose**: Implements a Pointer Network for predicting ordered sequences of points (e.g., Convex Hull problem).
- **Architecture**:
    - **Input Projection**: Maps input points to a higher-dimensional space. Hidden layer is of size 128.
    - **Encoder**: Encodes input points using an LSTM.
    - **Decoder**: Predicts the next point in the sequence based on attention over encoded points.
    - **Attention**: Computes the importance of each encoded point for predicting the current output.
    - **Output Layer**: Maps the context vector to 2D coordinates.
- **`forward` Method**:
    - Encodes the input.
    - Iteratively decodes the output using attention and updates the decoder's input with the last predicted coordinate.

4. `PointerNetLoss`

- **Purpose**: Computes loss for Pointer Network training.

- **Implementation**:
    - Uses Mean Squared Error (MSE) between predicted and target coordinates.
    - Ensures the predicted sequence matches the ground truth.
    - Note that the Point Net doesn't identify sequences, it rather predicts a float value. This avoids the case of repeated points in the sequences.

5. `train_model`

- **Purpose**: Trains a model using the provided dataset and optimizer(Adam).
- **Steps**:
    1. Loops through the dataset for a fixed number of epochs. Here 2 models are trained. One: the pointer net for predicting sorted sequence and the other the simple NN to identify the point.
    2. For each batch:
        - Converts data and labels to the correct data types.
        - Computes the model's predictions and loss.
        - Backpropagates and updates weights.
    3. Prints the loss after each epoch.

6. `save_model`

- **Purpose**: Saves the model's parameters to a file using `torch.save`.

7. `generate_data`

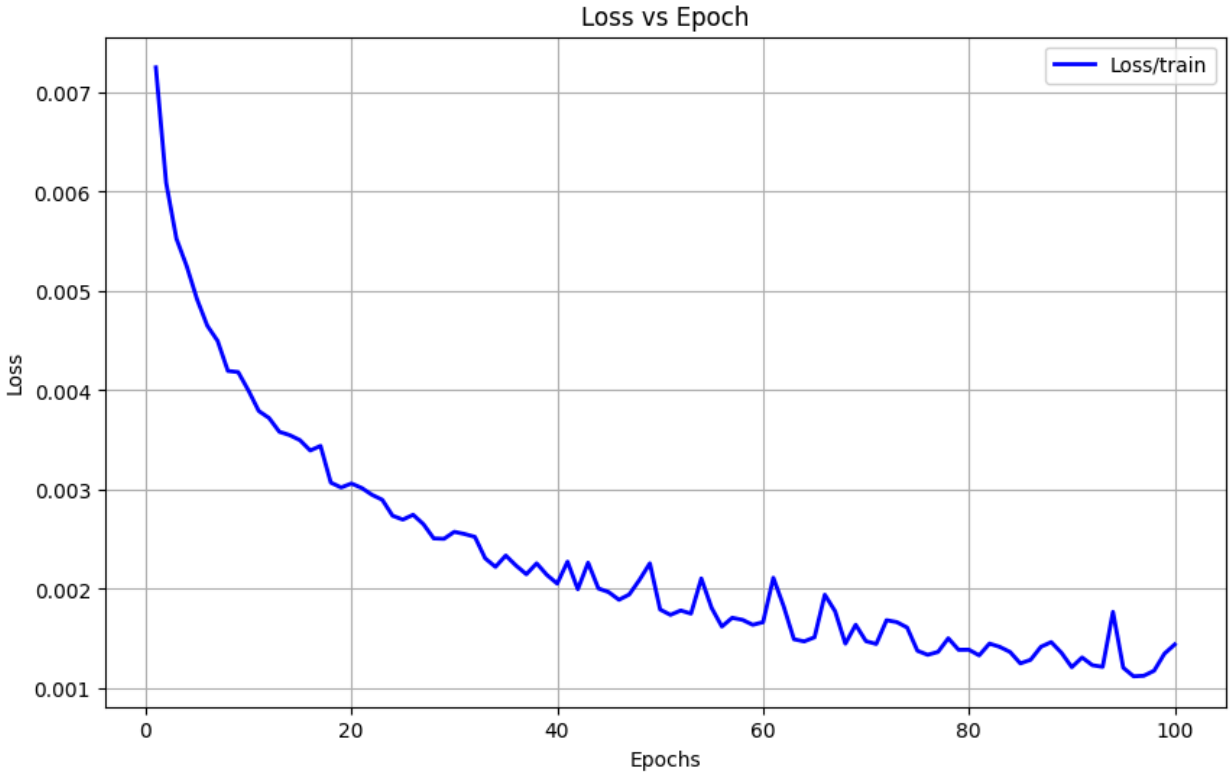
- **Purpose**: Generates synthetic datasets for Convex Hull problems.
- **Steps**:
    - Randomly generates 2D points.
    - Computes the centroid and sorts points by their angles relative to the centroid

8. `ConvexHullDataset`

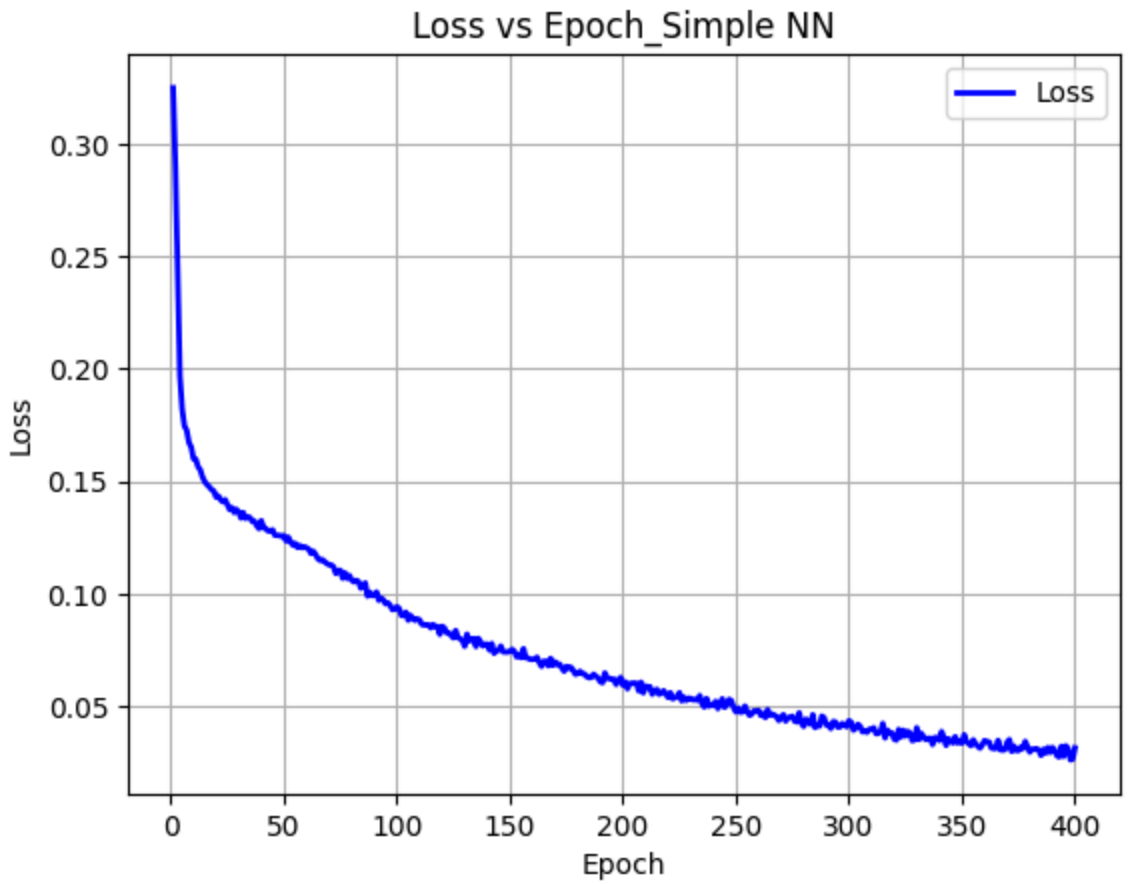
- **Purpose**: Wraps the synthetic data into a PyTorch `Dataset`.
- **Features**:
    - Pads sequences to a fixed length.The max length we considered here was 5, so sequences can be of length 3,4,5. Padding is done for those with lengths 3 and 4

9. Accuracy Computation

- **Purpose**: Computes binary accuracy for test predictions.
- **Steps**:
    - Compares the predicted outputs with true labels.
    - Counts the number of correct predictions and computes the percentage accuracy. This was used to verify the accuracy of the linear separable determining simple neural network. The same had an accuracy of 94% when computed with 10,000*4=40000 points.



The above is the plot of loss versus epochs(100) for the pointer Net used for predicting the sorted order.



The above is the plot of loss vs epochs(400) for the simple NN used for labelling whether the vertex is a convex hull vertex.

Recent Developments

DeepHullNet: A New Approach to Geometric Problems

DeepHullNet introduces a groundbreaking approach to solving both convex and concave hull problems using advanced deep learning techniques. While Pointer Networks (Ptr-Net) pioneered the use of neural networks for convex hull problems, they struggle with the complexity of concave hull computation. DeepHullNet addresses these challenges by combining the strengths of Pointer Networks with Transformers, a model well-known for its ability to understand complex relationships in data.

Key Innovations in DeepHullNet

The hybrid architecture of DeepHullNet integrates Pointer Networks for sequence modeling and Transformers for capturing spatial relationships. While Ptr-Net's reliance on Long Short-Term Memory (LSTM) excels at sequence generation, it falls short in understanding intricate spatial dependencies. Transformers, with their self-attention layers, enhance this ability, making DeepHullNet adept at identifying the natural boundaries required for concave hulls.

DeepHullNet also leverages learned rules from data to handle the adaptive strategies required for concave hulls. This overcomes the rigid, rule-based approach of classical algorithms and addresses the inefficiencies of Ptr-Net with irregular datasets. The results are impressive: DeepHullNet achieves a 5% improvement in convex hull accuracy and an 8% improvement in concave hull accuracy over Ptr-Net while significantly reducing runtime, making it ideal for real-time applications.

Advantages Over Pointer Networks

Pointer Networks use LSTM-based encoder-decoder architectures that process sequences step-by-step. This sequential nature can lead to slower performance and struggles with long-range dependencies. In contrast, DeepHullNet employs Transformers, which process all sequence elements in parallel, ensuring efficiency and scalability for larger datasets. The self-attention mechanism in Transformers captures both local and global dependencies, enabling better performance in convex and concave hull tasks.

Furthermore, DeepHullNet adopts a hybrid loss function designed to balance precision and computational demands for both hull types. This, combined with its parallelizable architecture, allows it to handle variable-length sequences of points more effectively than Ptr-Net, making it faster and more adaptable for diverse datasets.

Applications and Performance

DeepHullNet generalizes well across datasets with varying densities and distributions, making it suitable for practical applications in geospatial analysis, clustering, and computer graphics. It demonstrates superior performance in capturing essential features of point clouds, enabling accurate boundary computations for complex geometries.

In summary, DeepHullNet replaces the LSTM-based architecture of Pointer Networks with a Transformer-based design, resulting in better accuracy, scalability, and efficiency. Its ability to handle both convex and concave hull problems with precision and speed sets a new benchmark for deep learning in geometric problem-solving.

References

1)Pointer Networks By: Oriol Vinyals, Meire Fortunato, Navdeep Jaitly

2)An implementation of Pointer-Networks with Extensions By: Xiaoxi Wang and Dong Wang

3)Deephullnet: a deep learning approach for solving the convex hull and concave hull problems with transformer By: Haojian Liang, Shaohua Wang, Song Gao