Documentation for Assignment 4

**main.py – Training and Evaluation Script for Transformer-Based Air Quality Forecasting**

**Purpose:**  
This script implements the full workflow for training, validating, and evaluating a transformer-based time series forecasting model on the Air Quality UCI dataset. It includes dataset preparation, model training, loss monitoring, evaluation using MSE and MAE, and visualization of results.

**Functionality Overview:**

This script performs the following tasks:

1. Loads and prepares the time series dataset using a custom preprocessing utility
2. Defines and initializes a TimeSeriesTransformer model
3. Trains the model using Mean Squared Error loss
4. Evaluates model performance on a validation set
5. Visualizes training progress and final predictions
6. Computes final regression metrics (MSE and MAE)

**Key Components and Flow:**

**1. Hyperparameters:**

BATCH\_SIZE = 32

EPOCHS = 30

LEARNING\_RATE = 1e-3

SEQ\_LEN = 24

TARGET = "CO(GT)"

INPUT\_COLS = ["CO(GT)"]

* Defines model training parameters.
* This configuration performs univariate forecasting of carbon monoxide concentration one step ahead based on the previous 24 time steps.

**2. Data Preparation:**

X\_train, y\_train, X\_val, y\_val, scaler = prepare\_data(...)

* Calls prepare\_data() from utils.preprocessing to:
  + Clean and normalize the dataset
  + Create input/output sequences
  + Split data into training and validation sets
  + Returns PyTorch tensors and the fitted scaler

**3. DataLoaders:**

train\_loader = DataLoader(...)

val\_loader = DataLoader(...)

* Wraps training and validation datasets into PyTorch DataLoader for batch processing

**4. Model Initialization:**

model = TimeSeriesTransformer(...)

* Instantiates a lightweight transformer encoder model from models/transformer\_model.py
* Model architecture:
  + 1 transformer encoder layer
  + 4 attention heads
  + Model dimension: 64
  + Dropout: 0.1

**5. Training Loop:**

for epoch in range(EPOCHS):

...

* Performs forward and backward passes on the training set
* Calculates validation loss after each epoch
* Tracks and prints epoch-level training and validation MSE

**6. Evaluation and Plotting:**

plot\_loss(train\_losses, val\_losses)

plot\_predictions(y\_val.numpy(), predictions.numpy())

* Evaluates the model on the validation set without gradient tracking
* Uses utility functions to:
  + Plot loss curves
  + Plot predicted vs actual values for visual inspection

**7. Metrics:**

mean\_squared\_error(), mean\_absolute\_error()

* Computes and prints:
  + Mean Squared Error (MSE)
  + Mean Absolute Error (MAE)

**Expected Inputs and Outputs**

**Inputs:**

* AirQualityUCI.csv: Dataset of air quality readings with hourly time stamps
* Model hyperparameters and architecture choices (defined in script)

**Outputs:**

* Console logs of training/validation loss and metrics
* Visualization:
  + Loss curve plot
  + Predicted vs actual value plot
* Final evaluation:

Final Test MSE: 0.0034

Final Test MAE: 0.0451

**Dependencies:**

* torch, torch.nn, sklearn, matplotlib, numpy
* Local modules:
  + models.transformer\_model.TimeSeriesTransformer
  + utils.preprocessing.prepare\_data
  + utils.plotting.plot\_loss, plot\_predictions

**preprocessing.py – Data Cleaning and Preparation for Transformer-Based Time Series Forecasting**

**Purpose:**  
This module provides utility functions to load, clean, normalize, and structure time series data for model training. Specifically designed for the Air Quality UCI dataset, it prepares sequences of fixed length for use in a PyTorch-based transformer model.

**Functions Overview**

**load\_and\_clean\_data(filepath, target\_column="CO(GT)", input\_columns=None)**

**Purpose:**  
Loads the Air Quality dataset from CSV format and performs necessary preprocessing.

**Steps:**

* Reads data using semicolon delimiter and European decimal format (comma).
* Drops the last two unnamed columns.
* Replaces placeholder values (-200) with NaNs.
* Removes rows with missing target values.
* Interpolates remaining missing values linearly.
* Drops non-numeric columns (Date and Time).
* Optionally filters to user-specified input columns and target.

**Parameters:**

* filepath (str): Path to the CSV dataset.
* target\_column (str): Column to predict.
* input\_columns (list or None): Optional list of input features to retain.

**Returns:**  
Cleaned pandas.DataFrame.

**scale\_data(df)**

**Purpose:**  
Scales all numeric features to the [0, 1] range using MinMaxScaler.

**Parameters:**

* df (DataFrame): Input dataframe.

**Returns:**

* scaled (ndarray): Scaled data.
* scaler (MinMaxScaler): Fitted scaler object.

**create\_sequences(data, seq\_length=24)**

**Purpose:**  
Transforms continuous data into supervised learning sequences.

**Parameters:**

* data (ndarray): Scaled array with shape (samples, features + 1), where the last column is the target.
* seq\_length (int): Number of past time steps used for prediction.

**Logic:**

* For each sample:
  + X[i] = data[i:i+seq\_length, :-1]
  + y[i] = data[i+seq\_length, -1] (target at t+1)

**Returns:**

* X (ndarray): 3D array of shape (n\_samples, seq\_length, n\_features)
* y (ndarray): 1D array of targets

**train\_val\_split(X, y, test\_size=0.2)**

**Purpose:**  
Splits data into training and validation sets in time order (no shuffling).

**Returns:**

* X\_train, X\_val, y\_train, y\_val

**prepare\_data(filepath, target\_column="CO(GT)", input\_columns=None)**

**Purpose:**  
Master function that integrates all preprocessing steps.

**Workflow:**

1. Cleans data using load\_and\_clean\_data()
2. Normalizes it with scale\_data()
3. Converts to sequences with create\_sequences()
4. Splits into train/val sets
5. Converts to PyTorch tensors

**Returns:**

* X\_train (Tensor): Input training sequences of shape (n\_train, seq\_len, n\_features)
* y\_train (Tensor): Target values for training, shape (n\_train, 1)
* X\_val (Tensor): Input validation sequences
* y\_val (Tensor): Validation targets
* scaler (MinMaxScaler): Fitted scaler for inverse transformation or future scaling

**Dependencies:**

* pandas
* numpy
* sklearn.preprocessing.MinMaxScaler
* sklearn.model\_selection.train\_test\_split
* torch

**transformer\_model.py – Transformer-Based Model for Time Series Forecasting**

**Purpose:**  
Defines a PyTorch-based Transformer encoder architecture tailored for time series regression. This model is designed to learn temporal dependencies from multivariate or univariate input sequences and predict a continuous target value (e.g., air quality level) for the next time step.

**Class: TimeSeriesTransformer**

class TimeSeriesTransformer(nn.Module):

This class implements a lightweight transformer encoder model with positional encoding for sequence modeling. It is suitable for multistep time series forecasting tasks where a single prediction is made based on a fixed-length historical window.

**Constructor**

def \_\_init\_\_(self, input\_size=1, d\_model=64, nhead=4, num\_layers=1, dropout=0.1):

**Parameters:**

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| input\_size | Number of input features per time step (e.g., 1 for univariate input) |
| d\_model | Dimensionality of the model’s internal representation |
| nhead | Number of attention heads in the multi-head attention mechanism |
| num\_layers | Number of stacked TransformerEncoderLayer blocks |
| dropout | Dropout rate used in the transformer layers to prevent overfitting |

**Components:**

* input\_proj: A linear layer that projects input features to d\_model dimensions.
* positional\_encoding: A learnable parameter tensor that introduces positional context into the sequence.
* transformer\_encoder: A stack of transformer encoder layers for processing the time series input.
* decoder: A linear layer that maps the final encoder output to a scalar prediction.

**Method: forward**

def forward(self, x):

**Arguments:**

* x (Tensor): Input tensor of shape (batch\_size, sequence\_length, input\_size)

**Returns:**

* Output tensor of shape (batch\_size, 1), representing the prediction for the next time step.

**Processing Steps:**

1. **Input Projection**:  
   Projects raw input features to the model's hidden dimension:

x → (batch\_size, seq\_len, d\_model)

1. **Positional Encoding**:  
   Adds learned positional embeddings to each time step to introduce sequence order information.
2. **Transformer Encoder**:  
   Processes the entire input sequence in parallel using self-attention mechanisms to model dependencies.
3. **Output Extraction**:  
   Uses the output corresponding to the last time step as the representation for the full sequence:

x[:, -1, :] → decoder → (batch\_size, 1)

**Example Input/Output Shapes**

|  |  |
| --- | --- |
| **Tensor** | **Shape** |
| x (input) | (batch\_size, 24, input\_size) |
| output | (batch\_size, 1) |

**Notes:**

* The model is a simplified Transformer intended for short univariate or multivariate sequences.
* The positional encoding is initialized as a learnable tensor rather than using sinusoidal encoding.
* Only the final time step’s encoded representation is used for forecasting.

**Dependencies:**

* torch
* torch.nn

**plotting.py – Visualization Utilities for Model Training and Evaluation**

**Purpose:**  
This module provides plotting utilities to visualize training progress and model predictions in time series forecasting. It supports plotting of training/validation loss curves and actual vs. predicted output sequences.

**Functions**

**plot\_loss(train\_losses, val\_losses)**

**Purpose:**  
Plots the training and validation loss curves across epochs to visualize model learning dynamics.

**Parameters:**

* train\_losses (list or array-like): List of mean training losses for each epoch.
* val\_losses (list or array-like): List of mean validation losses for each epoch.

**Behavior:**

* Produces a line plot with two curves: training loss and validation loss.
* Y-axis represents MSE loss.
* X-axis represents training epoch number.

**Visualization Output:**

* Helps in identifying overfitting, underfitting, or convergence issues during training.

**Plot Title:**  
*“Training and Validation Loss”*

**plot\_predictions(y\_true, y\_pred, num\_samples=100)**

**Purpose:**  
Plots the actual vs predicted output values for a fixed number of time steps to assess model accuracy visually.

**Parameters:**

* y\_true (array-like): Ground truth target values from the validation set.
* y\_pred (array-like): Predicted values generated by the model.
* num\_samples (int): Number of data points to plot (default = 100).

**Behavior:**

* Plots two lines:
  + Actual CO concentration values
  + Predicted CO values by the model
* X-axis represents time step
* Y-axis represents the target variable: CO(GT)

**Plot Title:**  
*“Predicted vs Actual Values”*

**Dependencies:**

* matplotlib.pyplot

**Usage Context:**

These plots are typically called after training is completed, for the final evaluation of model performance:

* plot\_loss() is invoked after the training loop
* plot\_predictions() is used to compare predictions with actual outcomes on the test/validation set