Trajectory Analysis and Visualization

Piotr Ryszko

November 17, 2024

Contents

1 Goal

The goal of the task is to learn an ML or DNN model that is responsible for multi-class classification of different types of diffusion based on particle 2D trajectories.

2 Data

2.1 Data provided

I was provided with 5 numpy arrays:

- X_train,
- Y_train,
- X_val,
- Y_val,
- X_test.

Unfortunately, I was not provided with a set of labels for the test dataset. This meant that X-test dataset was pointless in my situation. I had to merge training and validation datasets into one, so I can divide it by myself and create needed test dataset.

2.2 Final data used

After merging I got one dataset made of **70 000** records. I decided to split them to create datasets I needed. I decided that train dataset would be **70%** of all records, validation dataset - **15%** and the remaining **15%** would belong to the test dataset. The important part of dividing the data was to use stratification, which refers to the process of ensuring that data is split or sampled in such a way that the proportions of different classes or groups in the dataset are preserved.

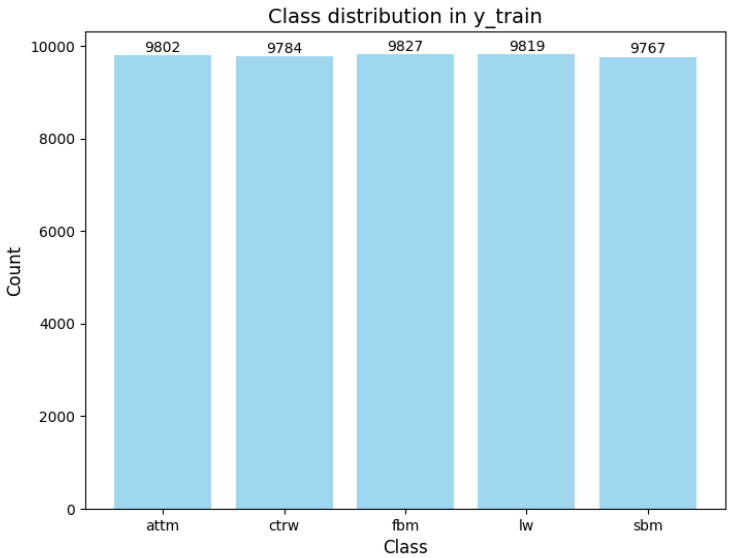


Figure 1: Class distribution in y_train

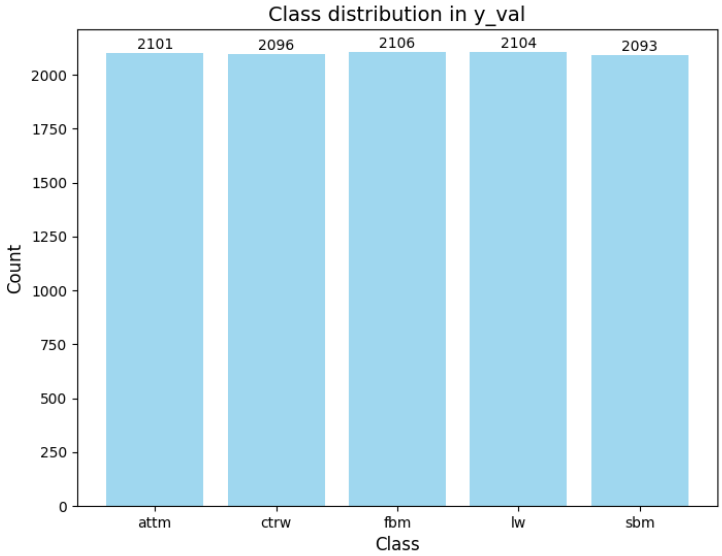


Figure 2: Class distribution in y_val

Figure 3: Class distribution in y_test

2.3 Data format

The data used in this task consists of three datasets, where each dataset X contains N records of size $(L, 2)$. Here:

- L denotes the length of the sequence, which is fixed at 300 for all datasets.
- The second dimension (2) represents the (x, y) coordinates of a particle at each moment in the sequence.

The dimensions of the datasets are as follows:

$$X_{\text{train}} : (48999, 300, 2)$$
$$X_{\text{val}} : (10500, 300, 2)$$
$$X_{\text{test}} : (10501, 300, 2)$$

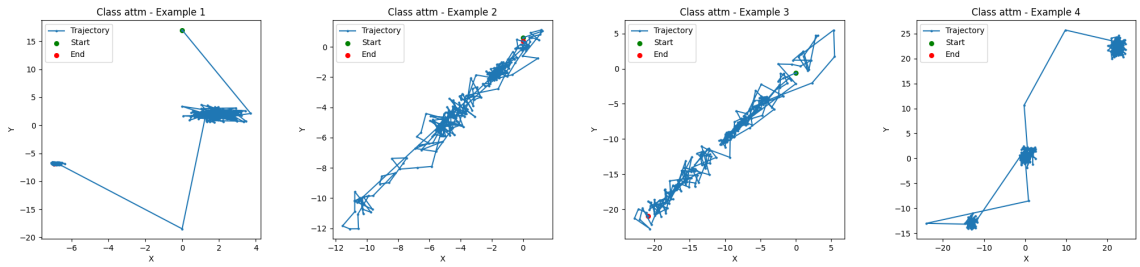
The labels y are encoded using the one-hot encoding technique. Each dataset of labels has N records, where each record represents the class of the sequence in a one-hot encoded format. The dimensions of the label datasets are:

$$y_{\text{train}} : (48999, 5)$$
$$y_{\text{val}} : (10500, 5)$$
$$y_{\text{test}} : (10501, 5)$$

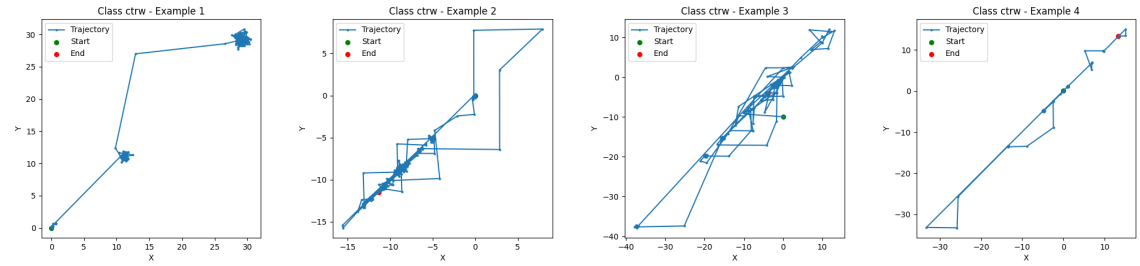
2.4 Classes

The task involves a classification problem with 5 diffusion types (classes). The classes are defined as follows:

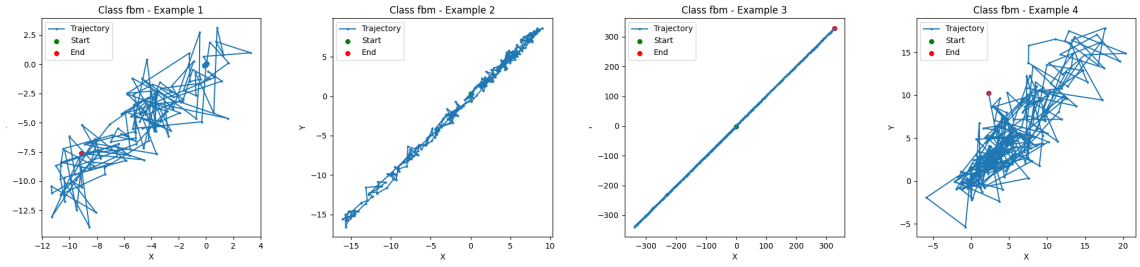
Class 0: attm - annealed transit time



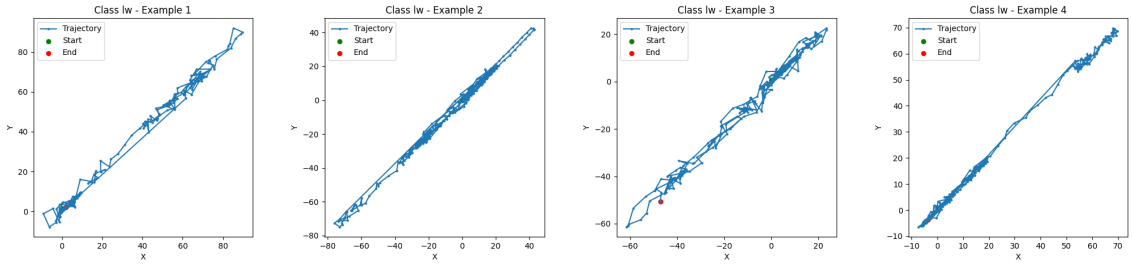
Class 1: ctrw - continuous time random walks



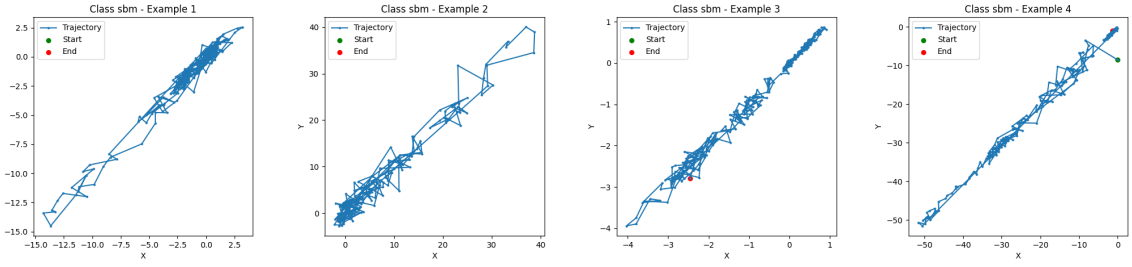
Class 2: fbm - fractional Brownian motion



Class 3: lw - Lévy walks



Class 4: sbm - scaled Brownian motion



2.5 First impressions

Class 3 - Lévy walks appear to be the most distinguishable among the diffusion types, as the trajectories within this class exhibit high similarity and consistent patterns. This suggests that they are likely to be the easiest class for a model to classify accurately. Conversely, Class 0 - annealed transit time poses a greater challenge. The provided examples demonstrate considerable overlap with characteristics observed in trajectories from Classes 1 (continuous time random walks) and 3 (Lévy walks), potentially leading to false classification.

To validate these observations, we will proceed to develop a classification model and evaluate its performance. This will allow us to empirically confirm whether the hypothesized ease of classification for Class 3 and difficulty for Class 0 align with the model's results.

3 Model Architecture Justification

The model architecture chosen for this diffusion classification task is specifically designed to effectively capture the temporal and sequential dependencies present in particle trajectories. The selection of various neural network components is motivated by the following considerations:

3.1 Bidirectional Long Short-Term Memory (LSTM) Layers

The use of Bidirectional LSTM layers is motivated by the need to capture both forward and backward dependencies in the sequence data. LSTM networks are particularly well-suited for time-series or sequence-based data, as they are designed to retain long-term dependencies, which is essential when dealing with particle trajectories over time. The bidirectional nature of these LSTMs enables the model to process the data in both directions, allowing it to fully utilize information from both the past and future states of the system. This approach helps the model learn complex patterns in the diffusion process, which may not be fully captured by a unidirectional LSTM.

```
# Bidirectional LSTM layer to capture both past and future dependencies
x = Bidirectional(LSTM(64, return_sequences=True))(input_layer)
x = Dropout(0.3)(x)
```

3.2 Dropout for Regularization

Dropout layers are employed after each LSTM layer to prevent overfitting, which can occur due to the complexity of the model and the relatively limited size of the dataset. By randomly setting a fraction of the input units to zero during training, dropout helps ensure that the model generalizes better to unseen data. The dropout rate of 0.3 is a standard choice that helps strike a balance between regularization and retaining sufficient model capacity.

```
# Dropout layer to prevent overfitting
x = Dropout(0.3)(x)
```

3.3 Attention Mechanism

The inclusion of an Attention mechanism allows the model to focus on the most relevant parts of the input sequence when making predictions. In the context of diffusion classification, not all time steps in the particle trajectory are equally important for determining the diffusion type. The Attention layer enables the model to assign varying levels of importance to different time steps, improving its ability to learn the most relevant features. This mechanism is especially valuable in tasks where certain parts of the sequence contain critical information, while others may be less informative.

```
# Attention mechanism to focus on relevant parts of the sequence
query = x  # LSTM output serves as both query and value
value = x
key = x  # Self-attention mechanism works with these 3 inputs
attention_output = Attention()([query, value, key])

attention_output = LayerNormalization()(attention_output)
```

3.4 Layer Normalization

Layer normalization is applied after the Attention mechanism to stabilize and accelerate training. It helps ensure that the inputs to the LSTM layers are well-scaled and reduces the risk of vanishing or exploding gradients. By normalizing the outputs of the Attention mechanism, layer normalization facilitates smoother learning and enables the model to converge faster during training. This is particularly important in deep architectures like the one used here, where each additional layer adds complexity and increases the potential for instability during training.

```
# Apply Layer Normalization to stabilize training
attention_output = LayerNormalization()(attention_output)
```

3.5 Dense Layers and Output Layer

After the sequence processing layers (LSTM and Attention), a Dense layer with ReLU activation is used to further refine the model's representations before the final classification layer. The output layer consists of 5 units with a softmax activation function, which is appropriate for multiclass classification. This allows the model to output a probability distribution over the five possible classes, representing the different types of diffusion.

```
# Dense layer for further refinement
x = Dense(64, activation='relu')(x)

# Output layer for 5 classes (types of diffusion)
output_layer = Dense(5, activation='softmax')(x)  # 5 diffusion types as classes
```

3.6 Why This Architecture?

The combination of Bidirectional LSTM layers, Attention mechanisms, and Dropout regularization is well-suited for time-series classification tasks like diffusion type classification. The bidirectional LSTM

allows the model to capture the full context of particle trajectories, while Attention enables it to focus on the most relevant information at each time step. Layer normalization helps stabilize the training process, and the final dense layers are tailored for multiclass classification. Together, these components create a robust model capable of accurately classifying different types of diffusion based on particle trajectories.

This architecture provides a balance between complexity and generalization, making it well-suited for the task at hand while ensuring that the model does not overfit to the training data.

3.7 Model summary

This is the return string of keras **model.summary()** method.

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 300, 2)	0	-
bidirectional (Bidirectional)	(None, 300, 128)	34,304	input_layer[0][0]
dropout (Dropout)	(None, 300, 128)	0	bidirectional[0][0]
lstm_1 (LSTM)	(None, 300, 64)	49,408	dropout[0][0]
dropout_1 (Dropout)	(None, 300, 64)	0	lstm_1[0][0]
attention (Attention)	(None, 300, 64)	0	dropout_1[0][0], dropout_1[0][0], dropout_1[0][0]
layer_normalization (LayerNormalizatio…	(None, 300, 64)	128	attention[0][0]
lstm_2 (LSTM)	(None, 32)	12,416	layer_normalization…
dropout_2 (Dropout)	(None, 32)	0	lstm_2[0][0]
dense (Dense)	(None, 64)	2,112	dropout_2[0][0]
dense_1 (Dense)	(None, 5)	325	dense[0][0]

Total params: 98,693 (385.52 KB)

Figure 9: Summary

4 Results

4.1 Before we begin

Unfortunately I could train my model only a few times (+/- 10), because each time it took around **40 minutes** to execute. I tried using Google Collab or Kaggle, but the program was executing even longer.

4.2 Training

The best result was achieved using **10 epochs** with **batch_size of 10**.

```
Epoch 1/10
490/490 ━━━━━━━━━━━━━━━━━━━━ 226s 453ms/step - accuracy: 0.3389 - loss: 1.4587 - val_accuracy: 0.4580 - val_loss: 1.2142
Epoch 2/10
490/490 ━━━━━━━━━━━━━━━━━━━━ 219s 447ms/step - accuracy: 0.4810 - loss: 1.1733 - val_accuracy: 0.5465 - val_loss: 1.0161
Epoch 3/10
490/490 ━━━━━━━━━━━━━━━━━━━━ 220s 449ms/step - accuracy: 0.5559 - loss: 1.0058 - val_accuracy: 0.5930 - val_loss: 0.9183
Epoch 4/10
490/490 ━━━━━━━━━━━━━━━━━━━━ 215s 438ms/step - accuracy: 0.5931 - loss: 0.9184 - val_accuracy: 0.6142 - val_loss: 0.8477
Epoch 5/10
490/490 ━━━━━━━━━━━━━━━━━━━━ 209s 426ms/step - accuracy: 0.6153 - loss: 0.8637 - val_accuracy: 0.6106 - val_loss: 0.8581
Epoch 6/10
490/490 ━━━━━━━━━━━━━━━━━━━━ 206s 421ms/step - accuracy: 0.6277 - loss: 0.8302 - val_accuracy: 0.6345 - val_loss: 0.8186
Epoch 7/10
490/490 ━━━━━━━━━━━━━━━━━━━━ 207s 422ms/step - accuracy: 0.6378 - loss: 0.8053 - val_accuracy: 0.6549 - val_loss: 0.7660
Epoch 8/10
490/490 ━━━━━━━━━━━━━━━━━━━━ 213s 435ms/step - accuracy: 0.6518 - loss: 0.7729 - val_accuracy: 0.6695 - val_loss: 0.7360
Epoch 9/10
490/490 ━━━━━━━━━━━━━━━━━━━━ 213s 434ms/step - accuracy: 0.6663 - loss: 0.7432 - val_accuracy: 0.6703 - val_loss: 0.7314
Epoch 10/10
490/490 ━━━━━━━━━━━━━━━━━━━━ 207s 422ms/step - accuracy: 0.6767 - loss: 0.7229 - val_accuracy: 0.6784 - val_loss: 0.7106
329/329 ━━━━━━━━━━━━━━━━━━━━ 17s 51ms/step
```

Figure 10: Progress through each epoch

4.3 Evaluation

In this section, I performed evaluation of the classification model designed to identify types of diffusion based on particle trajectories. The evaluation is carried out using several key classification metrics, such as accuracy, precision, recall, and F1-score. These metrics allow for a comprehensive analysis of the model's performance in various aspects, providing insight into its ability to correctly classify instances and distinguish between different types of diffusion.

```
Accuracy: 0.6749833349204838
Precision: 0.6817241616721798
Recall: 0.6749833349204838
F1-score: 0.6640850353457624
```

Figure 11: Metrics

Below, we present the confusion matrix for the model's performance on the test set, which helps in understanding the distribution of predictions across the various diffusion types.

```
Confusion Matrix:
 [[ 690  393   84    9  925]
 [ 449 1585   43    2   18]
 [  21   17 1192  358  518]
 [   3    1  185 1840   75]
 [ 137    2  125   48 1781]]
```

Figure 12: Confusion matrix

4.4 ROC Curve

In the case of diffusion type classification, I used the one-vs-rest approach to generate the ROC curve for each class, allowing us to assess the model's ability to correctly identify each diffusion type while minimizing false positives.

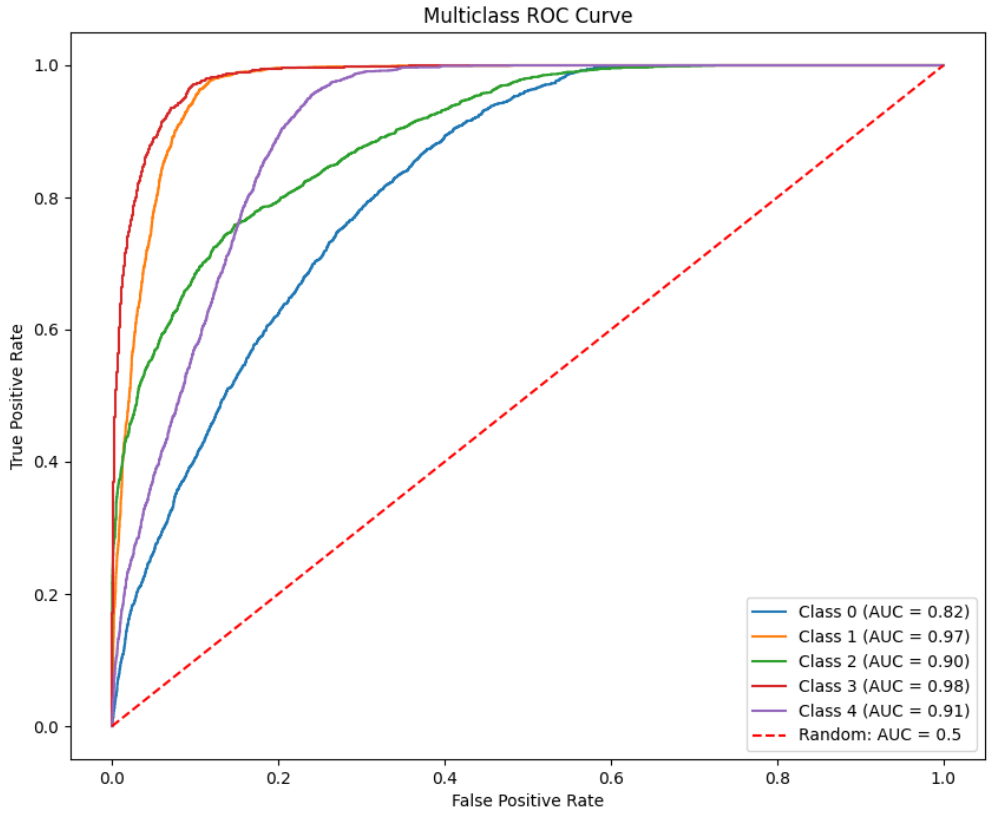


Figure 13: Confusion matrix

5 Summary

5.1 Final words

The F1 score, **exceeding 0.66**, was particularly gratifying, especially considering that this was my first experience with utilizing LSTM layers. Given the challenges of the project, I consider it a success, as it has significantly expanded my understanding and skills in deep learning. The ROC Curve further supports the hypothesis we proposed earlier, showing that class 3 was the easiest to classify, while class 0 posed the greatest challenge. This experience has strengthened my passion for neural networks, and I look forward to contributing to more projects in this domain. Thank you for your time.

5.2 Github

Full code is availble on my ***GitHub Repository***.