Introduction to Artificial Intelligence - Final Project

group: 27 member: 張詠哲、彭星樺、洪德輝

- **Topic :**

CoachAI Badminton Challenge 2023 in conjunction with IJCAI 2023
-Track 2: Forecasting Future Turn-Based Strokes in Badminton Rallies

- **Introduction**

Badminton necessitates a combination of physical prowess and tactical acumen to execute strategies and emerge victorious. In addition to maintaining stamina and employing sophisticated tactics, accurately predicting opponents' strokes is a crucial skill for badminton players seeking to secure victory. By forecasting future strokes based on past ones, coaches can guide players effectively and devise winning strategies. This predictive analysis simulates players' tactics, providing insights into frequently returned shot types and targeted areas, which inform decision-making. Moreover, stroke forecasting contributes to engaging storytelling within the badminton community by assessing probability distributions of returns during matches. However, stroke forecasting remains a challenging task due to the dynamic nature of the game.

Hence, The goal of the competition consists of two tasks to stimulate effective solutions for badminton analytics:

- **Track 1** : focuses on the automatic annotation of technical data for badminton match videos. Participants are tasked with designing solutions employing computer vision techniques to automatically annotate shot-by-shot data from match videos.

- **Track 2 :** centers around the forecasting of future turn-based strokes in badminton rallies. Participants are required to develop predictive models capable of forecasting upcoming strokes, including shot types and locations, based on past stroke sequences.

Since our final project only needs to participation in Track 2. So, there are some specific details. The objective of Track 2 is to forecast future strokes, encompassing shot types and locations, given a sequence of past strokes. This process is commonly referred to as stroke forecasting. For each singles rally, participants must predict future strokes, including shot types and corresponding area coordinates, for the subsequent n steps. The value of n varies based on the length of the rally.

● Literature Review / Related Works

There are many paper about the badminton forecast. Including: predicting the future trajectory、forecasting upcoming strokes (including shot types and locations)、analyze opponents' strengths and weaknesses and style、predicting types of returning strokes、players next move...etc. Following are some related work about our task:

[1] Modeling Turn-Based Sequences for Player Tactic Applications in Badminton Matches

The main purpose of this paper is to explore how to apply deep learning techniques to address technical challenges in badminton analytics and propose a unified badminton language to describe the process of the shot.The paper proposes three essential tasks, including measuring the win probability of each shot, forecasting the tactics of the players, and explaining the model behavior, and proposes a deep learning model that captures both short-term and long-term dependencies in a badminton rally to measure shot influences. They also introduces a framework that uses two Transformer-based extractors and a position-aware gated fusion network to fuse rally contexts and contexts of two players with information weights and position weights to forecast the possible tactics of players. By accomplishing these tasks, badminton players and coaches can analyze opponents' strengths and weaknesses and style, and the third task can explain the model's behavior. Additionally, the paper proposes a framework that uses two encoder-decoder extractors and a position-aware fusion network to forecast the possible tactics of players. These contributions can help the research and badminton communities better understand and apply deep learning techniques to address challenges in badminton analytics. Because the stroke is also determined by the player style. Therefore we think this paper is related to our work

[2] Prediction of Shuttle Trajectory in Badminton Using Player's Position

The approach proposed in this paper is to use a time-sequence model that takes into account both the shuttle and player position information to predict the future shuttle trajectory in badminton matches. The authors suggest using recurrent neural networks (RNNs) as the sequential model to input previous shuttle trajectories and output the future shuttle trajectories. The model also takes into account the players' position information to predict the future shuttle trajectory. The proposed method outperformed baseline methods that use only the shuttle position information as the input and other methods that use time-sequence models. We think the shuttle trajectory may related to the stroke. The differenent between our approach and this paper is that our approach is base on transformers to modify and this paper is use RNN.

[3] Where Will Players Move Next? Dynamic Graphs and Hierarchical Fusion for Movement Forecasting in Badminton

The main purpose of this paper is to address the challenge of movement forecasting in badminton, specifically predicting the types of returning strokes and the locations players will move to based on previous strokes. The goal is to develop a model that can accurately forecast the movement patterns and shot types of players in badminton rallies. The paper introduces the Player Movements (PM) graph as a representation of the badminton rally, capturing the players' locations and their relationships over time. They propose the DyMF model, which utilizes an encoder-decoder architecture and incorporates interaction style extractors and hierarchical fusion modules. This paper is alse a predicting the types of returning strokes, which is very similar to our task. But it can also predict the player location.

[4] ShuttleNet: Position-aware Fusion of Rally Progress and Player Styles for Stroke Forecasting in Badminton

The main purpose of this paper is to propose a novel neural network model called ShuttleNet for stroke forecasting in badminton games. The ShuttleNet model is based on an encoder-decoder architecture and incorporates rally information and player information with two extractors. In addition, a position-aware gated fusion network is proposed leveraging information dependency and position weights to decide the importance of rally contexts and contexts of the players for returning each stroke. Our main approach is mainly base on this paper to modify

● Datasets

(Due to the absence of a publicly available stroke event dataset, we curated a collection of real-world badminton singles matches from openly accessible sources. To ensure the dataset's accuracy, domain experts were enlisted to manually label the matches. Our dataset comprises 75 high-ranking matches played between 2018 and 2021, involving 31 players from both men's and women's singles categories. After eliminating flawed data, such as replay highlights, we obtained a dataset consisting of 180 sets, 4,325 rallies, and 43,191 strokes. On average, the rallies lasted for 10 shots. To differentiate between strokes, domain experts established ten shot types, including net shot, clear, push/rush, smash, defensive shot, drive, lob, drop, short service, and long service.
For the stroke forecasting task, each stroke in our dataset includes the rally ID, stroke order within a rally, player responsible for the stroke, shot type, and the precise area coordinates

where the shuttlecock was returned.)

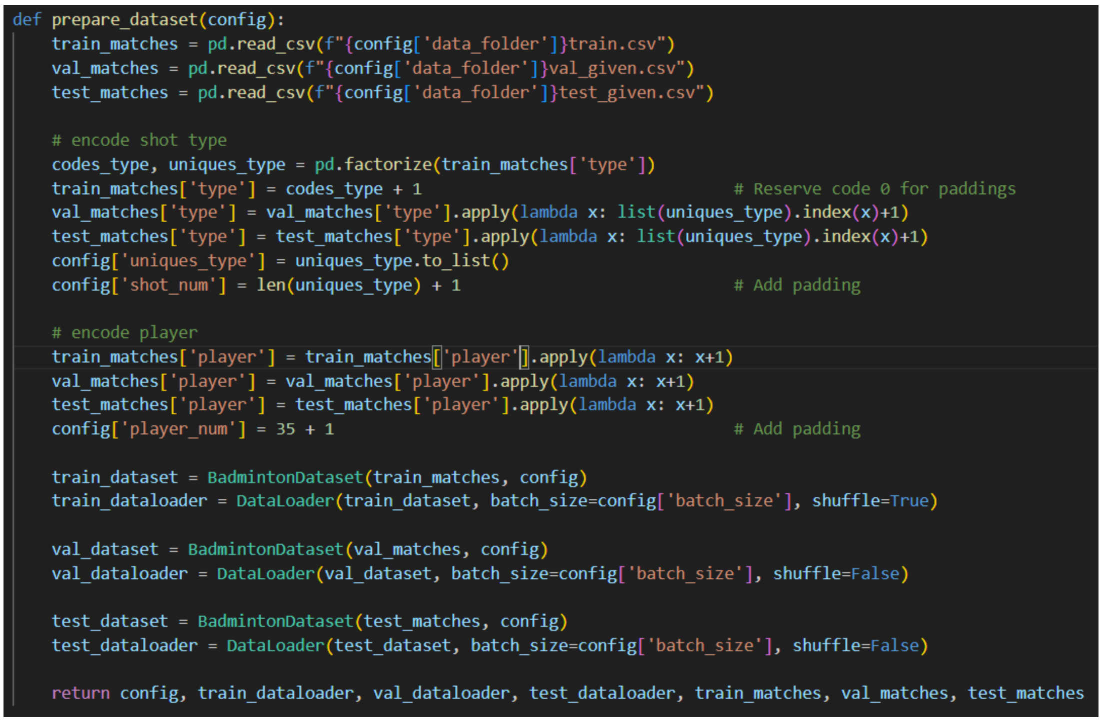
The data set is provided by the competition official. Here is some brief introduction:

- **train.csv,val.csv:**
  record the information about the rallies. Such as time、player、getpoint player、landing_x、landing_y、backhand……etc

- **match_metadata.csv**
  record the informatioin about the match. Such as match id、winner、loser……etc

The approach that we preprocessing the data set is base the code which provide by the ShuttleNet paper. Here are some explain:

Group the data by rally IDs and stores the relevant information in the sequences dictionary. Also implemente a method to retrieve a specific sequence from the dataset, applying padding if needed to ensure uniform length. The prepare_dataset function (**figure 1**) takes care of additional data preprocessing steps. It encodes the shot types and players into integer values using factorization. The encoded values are stored in the respective columns of the dataset. Then return the preprocessed dataset for training, validation, and testing.



```python
def prepare_dataset(config):
    train_matches = pd.read_csv(f"{config['data_folder']}train.csv")
    val_matches = pd.read_csv(f"{config['data_folder']}val_given.csv")
    test_matches = pd.read_csv(f"{config['data_folder']}test_given.csv")

    # encode shot type
    codes_type, uniques_type = pd.factorize(train_matches['type'])
    train_matches['type'] = codes_type + 1                                # Reserve code 0 for paddings
    val_matches['type'] = val_matches['type'].apply(lambda x: list(uniques_type).index(x)+1)
    test_matches['type'] = test_matches['type'].apply(lambda x: list(uniques_type).index(x)+1)
    config['uniques_type'] = uniques_type.to_list()
    config['shot_num'] = len(uniques_type) + 1                            # Add padding

    # encode player
    train_matches['player'] = train_matches['player'].apply(lambda x: x+1)
    val_matches['player'] = val_matches['player'].apply(lambda x: x+1)
    test_matches['player'] = test_matches['player'].apply(lambda x: x+1)
    config['player_num'] = 35 + 1                                         # Add padding

    train_dataset = BadmintonDataset(train_matches, config)
    train_dataloader = DataLoader(train_dataset, batch_size=config['batch_size'], shuffle=True)

    val_dataset = BadmintonDataset(val_matches, config)
    val_dataloader = DataLoader(val_dataset, batch_size=config['batch_size'], shuffle=False)

    test_dataset = BadmintonDataset(test_matches, config)
    test_dataloader = DataLoader(test_dataset, batch_size=config['batch_size'], shuffle=False)

    return config, train_dataloader, val_dataloader, test_dataloader, train_matches, val_matches, test_matches
```

Figure 1. prepare dataset function

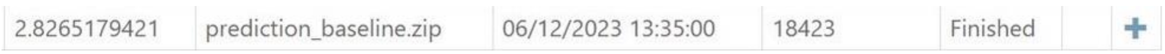
● Baseline

The baseline we use is the ShuttleNet model which is provide by paper code. Here is some brief description about ShuttleNet model:

The model (ShuttleNet) is based on an encoder-decoder architecture and incorporates rally information and player information with two extractors. The first extractor is a rally progress extractor (TRE) that captures the progress of the rally, and the second extractor (TPE) is a player style extractor that models the characteristics of the players. A position-aware gated fusion network (PGFN) is also proposed to integrate rally contexts and contexts of the players by conditioning on information dependency and different positions. It uses to fuse player and game situation information to better predict the outcome of the next shot.

The same embedding layer, prediction layer , hyperparameters were used for all the baselines. Moreover, since all the baselines take single inputs, we concatenated shot types and area and projected them to same dimension of these baselines instead.

This is the prediction score from the baseline :



2.8265179421	prediction_baseline.zip	06/12/2023 13:35:00	18423	Finished	✚

(the hyperparameter in this part is used by defult they set: seed value = 42 , max ball round = 70, encode length = 4 , batch size = 32 , lr = 1e-4 , epochs = 150, shot_dim = 32 、 area_dim = 32, player_dim = 32, encode_dim = 32 , area num = 5)

● Main Approach

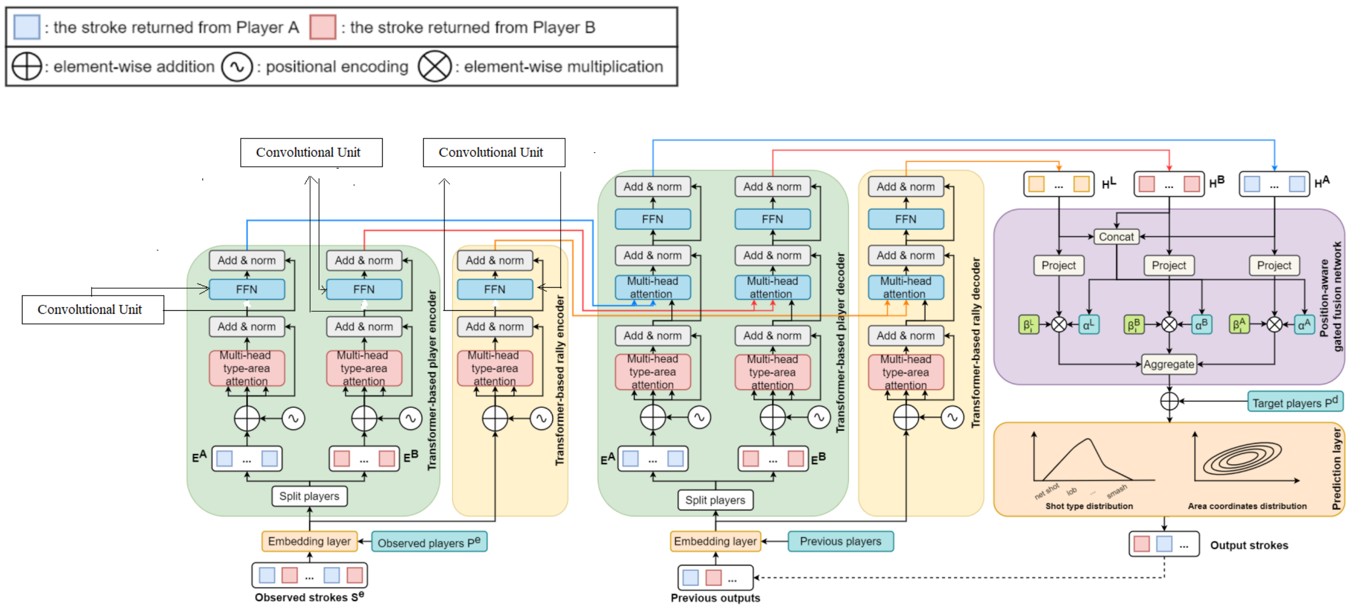
The main approach we implement is modification the ShuttleNet. We add 2 things : convolutional unit & bayesian optimization

1. Improved Transformer Architecture for Sequence to Sequence Translation

They developed the Convolutional Multi-headed attentive Encoder Transformer or COMET architecture to combine the best features from convolutional and self-attention architectures. The convolutional architectures are able to capture positional dependencies and other high level features, but this information is blurred by multiple layers of convolution. On the other hand, the attention mechanisms in the transformer allow immediate access to any location within the input, at the cost of sacrificing wider context. In COMET, the encoder uses self-attention to isolate specific points of interest in the input, while convolutional structures provide a high-level picture of the input.

We use this implementation of convolutional unit into our model.

We add convolutional unit into the EncoderLayer class, Bayesian Optimization as the Optimization Algorithm.



5.1 Convolutional Structures

In the model we employ a combination of dilated convolution and gated activation. Dilated convolutions are a way to increase the receptive field of a kernel without increasing the kernel size by introducing spaces within the kernel. We use linearly increasing dilation to increase the receptive field without significantly increasing computational cost.

Gated activation is a way to control the relevance of an output by multiplying each element in the output by a scaling factor ∈ (0,1). We use two convolutions followed by a gated activation, which we will refer to as a gated convolution

5.2.Convolutional Unit

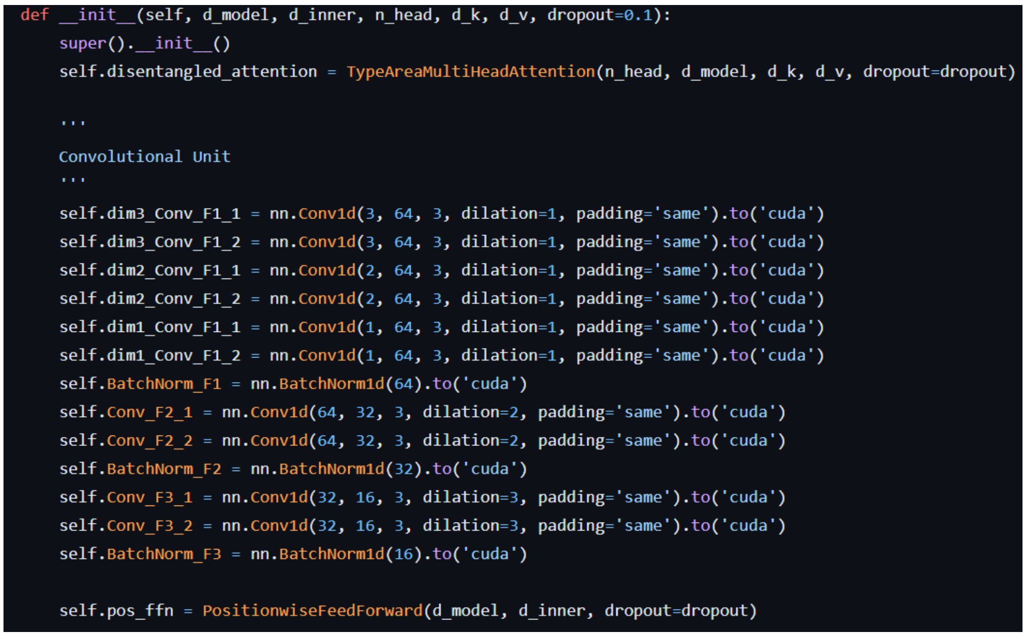
```
ConvUnit($x$)
  input  : Tensor $x \in \mathbb{R}^{d_m \times sl}$
  output : Tensor $y \in \mathbb{R}^{d_m \times sl}$
      $F_1, F_2, F_3 = \texttt{MultiLayerConv}(x)$
      $x' = \texttt{concat}(F_1, F_2, F_3, x)$
      $y = \texttt{FFN}_1(x')$
  return $y$
```

By encoding the raw features of a badminton rally, such as player positions, velocities, and shot characteristics, into higher-level representations, convolutional units enable the model to capture the relevant features that contribute to predicting future turn-based strokes.

A single convolutional layer consists of a 1-dimensional gated convolution along the *sl* dimension of the input tensor. Each layer has kernel size 3, input dimension d_{in}, dilation rate *l* and and number of output features n_f . We pad both ends of the input tensor with *l* zeros in the *sl* dimension to maintain the dimensionality invariant and generate an output tensor with size *nf* ×*sl*.

Let $g_{d_{in},n_f,l} : \mathbb{R}^{d_{in}\times sl} \to \mathbb{R}^{n_f \times sl}$, denote the gated convolution as described. To capture interactions between distant elements in the input sequence, we do a multi-layer convolution to generate three sets of feature maps with 64, 32, and 16 features respectively. To aid training speed we employ a batch normalization on the output of all gated convolutions

```
MultiLayerConv($x$)
  input  : Tensor $x \in \mathbb{R}^{d_m \times sl}$
  output : Feature maps $F_1 \in \mathbb{R}^{64\times sl}, F_2 \in \mathbb{R}^{32\times sl}, F_3 \in \mathbb{R}^{16\times sl}$
      $F_1$ =BatchNorm($g_{d_m,64,1}(x)$)
      $F_2$ =BatchNorm($g_{64,32,2}(F_1)$)
      $F_3$ =BatchNorm($g_{32,16,3}(F_2)$)
  return $F_1, F_2, F_3$
```

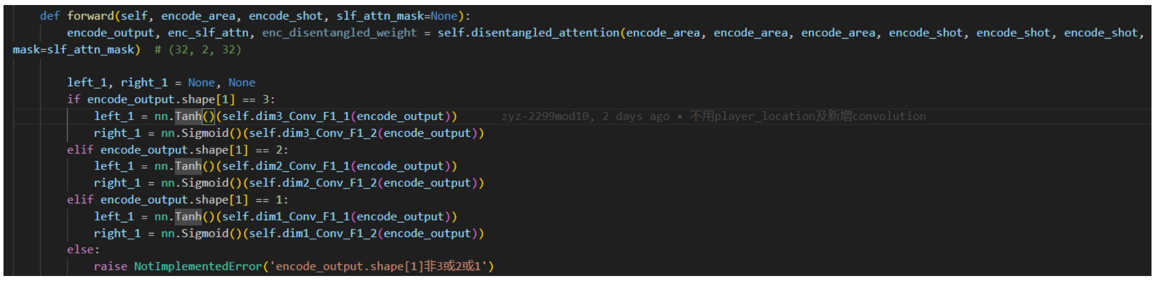

```python
def __init__(self, d_model, d_inner, n_head, d_k, d_v, dropout=0.1):
    super().__init__()
    self.disentangled_attention = TypeAreaMultiHeadAttention(n_head, d_model, d_k, d_v, dropout=dropout)

    '''
    Convolutional Unit
    '''
    self.dim3_Conv_F1_1 = nn.Conv1d(3, 64, 3, dilation=1, padding='same').to('cuda')
    self.dim3_Conv_F1_2 = nn.Conv1d(3, 64, 3, dilation=1, padding='same').to('cuda')
    self.dim2_Conv_F1_1 = nn.Conv1d(2, 64, 3, dilation=1, padding='same').to('cuda')
    self.dim2_Conv_F1_2 = nn.Conv1d(2, 64, 3, dilation=1, padding='same').to('cuda')
    self.dim1_Conv_F1_1 = nn.Conv1d(1, 64, 3, dilation=1, padding='same').to('cuda')
    self.dim1_Conv_F1_2 = nn.Conv1d(1, 64, 3, dilation=1, padding='same').to('cuda')
    self.BatchNorm_F1 = nn.BatchNorm1d(64).to('cuda')
    self.Conv_F2_1 = nn.Conv1d(64, 32, 3, dilation=2, padding='same').to('cuda')
    self.Conv_F2_2 = nn.Conv1d(64, 32, 3, dilation=2, padding='same').to('cuda')
    self.BatchNorm_F2 = nn.BatchNorm1d(32).to('cuda')
    self.Conv_F3_1 = nn.Conv1d(32, 16, 3, dilation=3, padding='same').to('cuda')
    self.Conv_F3_2 = nn.Conv1d(32, 16, 3, dilation=3, padding='same').to('cuda')
    self.BatchNorm_F3 = nn.BatchNorm1d(16).to('cuda')

    self.pos_ffn = PositionwiseFeedForward(d_model, d_inner, dropout=dropout)
```

g represents GatedConv(). Here, the input x corresponds to the first return value (encode_output) of the self.disentangled_attention() function inside the forward() function of the EncoderLayer class.

The dimension of encode_output can have three possibilities: ((32, 1, 32), (32, 2, 32), or (32, 3, 32)). For explanation, I will use the case where the dimension is (32, 2, 32).Therefore, d_m is 2, and sl is 32 (batch size is the first dimension = 32).

The domain and range of g are $g_{d_{in},n_f,l} : \mathbb{R}^{d_{in} \times sl} \to \mathbb{R}^{n_f \times sl}$. So, F_1 has a dimension of (32, n_f, sl) = (32, 64, 32). Similarly, F_2 has a dimension of (32, 32, 32), and F_3 has a dimension of (32, 16, 32). The third value in the subscript of g represents the dilation of the convolution.

Since encode_output's dimension can have three possibilities: ((32, 1, 32), (32, 2, 32), or (32, 3, 32)), in the implementation part of the code:



```
    def forward(self, encode_area, encode_shot, slf_attn_mask=None):
        encode_output, enc_slf_attn, enc_disentangled_weight = self.disentangled_attention(encode_area, encode_area, encode_area, encode_shot, encode_shot, encode_shot, 
mask=slf_attn_mask)  # (32, 2, 32)

        left_1, right_1 = None, None
        if encode_output.shape[1] == 3:
            left_1 = nn.Tanh()(self.dim3_Conv_F1_1(encode_output))        zyz-2299mod10, 2 days ago • 不用player_location及新增convolution
            right_1 = nn.Sigmoid()(self.dim3_Conv_F1_2(encode_output))
        elif encode_output.shape[1] == 2:
            left_1 = nn.Tanh()(self.dim2_Conv_F1_1(encode_output))
            right_1 = nn.Sigmoid()(self.dim2_Conv_F1_2(encode_output))
        elif encode_output.shape[1] == 1:
            left_1 = nn.Tanh()(self.dim1_Conv_F1_1(encode_output))
            right_1 = nn.Sigmoid()(self.dim1_Conv_F1_2(encode_output))
        else:
            raise NotImplementedError('encode_output.shape[1]非3或2或1')
```

For the FFN we use the original PositionWiseFeedForward

$$\text{GatedConv}(x) = f(k_f * x) \otimes \sigma(k_\sigma * x)$$

Since PyTorch's Conv1d() doesn't require setting the kernel values and only needs the kernel size, we don't need to worry about k_f and k_σ.

Finally, when performing concatenation in ConvUnit(), the dimensions are as follows: F_1 has a dimension of (32, 64, 32), F_2 has (32, 32, 32), F_3 has (32, 16, 32), and x has (32, 2, 32). Therefore, after concatenation, the resulting dimension is (32, 64 + 32 + 16 + 2, 32) = (32, 114, 32).

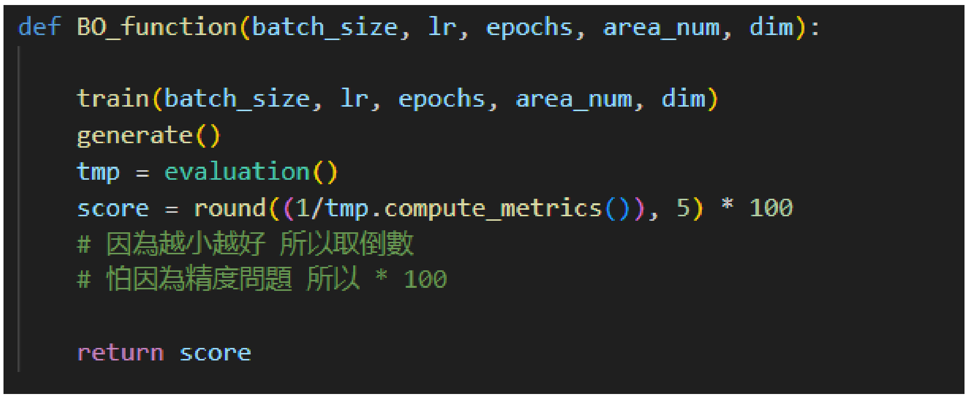
2. practical bayesian optimization of machine learning algorithms

Bayesian optimization is a BlackBox optimization algorithm , works by constructing a posterior distribution of functions (gaussian process) that best describes the function you want to optimize. As the number of observations grows, the posterior distribution improves, and the algorithm becomes more certain of which regions in parameter space are worth exploring and which are not. It build a probabilistic model that incorporates the objective function and use it to select the optimal hyperparameters for evaluating the correct objective function (similar to conditional probability).

The difference from other procedures is that it constructs a probabilistic model for f(x) and then exploits this model to make decisions about where in X to next evaluate the function, while integrating out uncertainty. The essential philosophy is to use all of the information available from previous evaluations of f(x) and not simply rely on local gradient and Hessian approximations. This results in a procedure that can find the minimum of difficult non-convex functions with relatively few evaluations, at the cost of performing more computation to determine the next point to try.

It predicts the probability distribution of the function values at any point based on a set of sampled points. It accomplishes this through Gaussian Process Regression and an Acquisition Function (AF). The AF calculates the level of exploration for each point and determines the next sampled point by finding the extremum of the Acquisition Function. Finally, the extremum of the sampled points is returned as the optimum of the function.

Therefore, it can be expressed as follows: Bayesian optimization builds a probabilistic model with the objective function and uses it to select the best hyperparameters for evaluating the objective function (similar to conditional probability). It predicts the probability distribution of function values at unobserved points using Gaussian Process Regression and uses the Acquisition Function to determine the level of exploration for each point. The extremum of the Acquisition Function is then computed to determine the next sampled point. In the end, the extremum of the sampled points is returned as the optimum of the function.



```
def BO_function(batch_size, lr, epochs, area_num, dim):

    train(batch_size, lr, epochs, area_num, dim)
    generate()
    tmp = evaluation()
    score = round((1/tmp.compute_metrics()), 5) * 100
    # 因為越小越好 所以取倒數
    # 怕因為精度問題 所以 * 100

    return score
```

● Evaluation Metric

To predict the score we need to calculate the average of Cross Entropy (CE) and Mean Absolute Error(MAE). For this experiment we are using shot type as the CE and landing x, landing y as the MAE. There is 10 type of shot in CE which the result will equal to 1. Then, we will add it with MAE. We will sample it 6 times and choose the best score. The calculation of score are following:

$$Score = \min(l_1, l_2, \dots l_6)$$

$$l_i = \mathrm{AVG}(\mathrm{CE} + \mathrm{MAE})$$

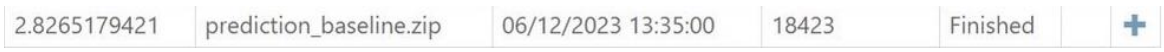
$$= \frac{\sum_{r=1}^{|R|} \sum_{n=\gamma+1}^{|r|} \left[S_n \log \hat{S}_n + (|x_n - \hat{x}_n| + |y_n| - \hat{y}_n) \right]}{|R| \cdot (|r| - \gamma)}$$

● Results & Analysis

Beside use the approach above, we have also experient add other feature like player location x、y. And following are the result we got (screenshot from Codalab, the value from left to right are score, file, date, file size, status):

■ Results

1. Baseline:

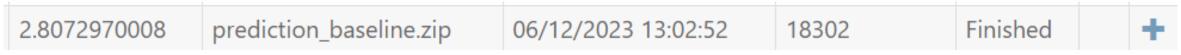


2.8265179421	prediction_baseline.zip	06/12/2023 13:35:00	18423	Finished

2. Add player location x, y

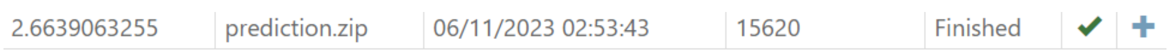
We forgot to screenshot the result in this part, but the score are 2.8429123256

3. Only use convolutional unit (don't add player location):



2.8072970008	prediction_baseline.zip	06/12/2023 13:02:52	18302	Finished

4. use convolutional unit + bayesian optimization (don't add player location)



2.6639063255	prediction.zip	06/11/2023 02:53:43	15620	Finished	✓

■ Analysis

player location x, y is suck.

● Future Work

We can add more features to the model , because we only have two features which is shot type and landing area.Theres still a lot of data we can use inside of the training data.

● Contribution of each Member

- 109350008 張詠哲 35 %

- 109550009　彭星樺 35 %
- 0716106　　洪德輝 30 %

● References

[1] Modeling Turn-Based Sequences for Player Tactic Applications in Badminton Matches
[2] Prediction of Shuttle Trajectory in Badminton Using Player's Position
[3] Where Will Players Move Next? Dynamic Graphs and Hierarchical Fusion for Movement Forecasting in Badminton
[4] ShuttleNet: Position-aware Fusion of Rally Progress and Player Styles for Stroke Forecasting in Badminton
[5] Austin Wang; Adviser: Prof. Karthik Narasimhan . 2019.　Improved Transformer Architecture for Sequence to Sequence Translation.
[6] Jasper Snoek; Hugo Larochelle ; Ryan P. Adams . 2012 . Practical Bayesian Optimazation of Machine Learning Algorithms.