About TimeMAE

File Structure

- `dataset.py`
  定义数据集的类，定义了该数据集的一些内置方法
- `datautils.py`
  数据集分类和处理
- `model`
  - `layers.py`
    `transformer` 的 layers 层
  - `TimeMAE.py`
    TimeMAE 模型
- `args.py`
  根据 `datautils` 中处理数据集的方法设置参数。
  并设置其他参数，比如 cuda 之类
- `classification.py`
  分类器，只有两个方法
- `loss.py`
  计算 loss
- `process.py`
  定义训练类和训练函数
- `visualize.py`
  可视化结果
- `main.py`
  运行整个流程

- `run.sh`
  脚本文件，用于运行 `main.py` 和设置超参数
  运行三次，猜测可能是计算误差。

Dataset

仅介绍 HAR 数据集：使用 load_HAR 函数，可以发现：TRAIN_DATA_ALL, TRAIN_DATA, TEST_DATA 分别用于预训练、微调和测试。其中各个数据集的规模如下：

```
TRAIN torch.Size([7352, 128, 9]) torch.Size([7352])
VAL torch.Size([1471, 128, 9]) torch.Size([1471])
TEAT torch.Size([2947, 128, 9]) torch.Size([2947])
ALL_TRAIN torch.Size([8823, 128, 9]) torch.Size([8823])
```

其中 TRAIN_DATA_ALL 是 TRAIN 和 VAL 的拼接。

重点：
在设定了 batch-size 之后，进入 encoder 的 tensor 结构是：(batch_size, sequence_length, dimension) =（128， 7， 64）

Model and Encoder

在模型中，使用自定义多层的 transformer 模型 TransformerBlock 做 Encoder，如下：

```
class Encoder(nn.Module):
    def __init__(self, args):
        super(Encoder, self).__init__()
        d_model = args.d_model
        attn_heads = args.attn_heads
        d_ffn = 4 * d_model
        layers = args.layers
        dropout = args.dropout
        enable_res_parameter = args.enable_res_parameter
        # TRMs
        self.TRMs = nn.ModuleList(
```

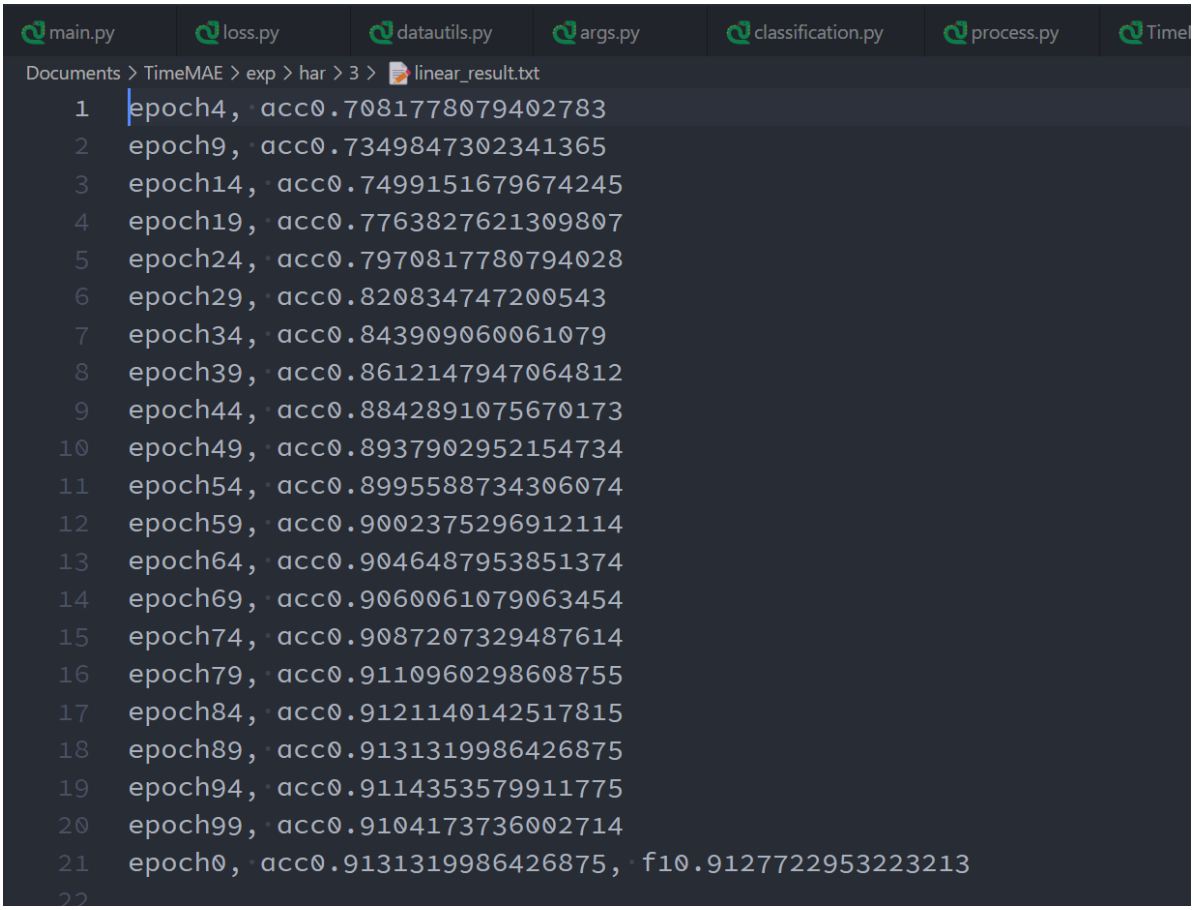
```
            [TransformerBlock(d_model, attn_heads, d_ffn,
enable_res_parameter, dropout) for i in range(layers)])

    def forward(self, x):
        for TRM in self.TRMs:
            x = TRM(x, mask=None)
        return x
```

- d_model 表示 Transformer 编码器的隐藏层维度。
- attn_heads 表示注意力头的数量。
- d_ffn 是 FeedForward 层的隐藏层维度，这里设置为隐藏层维度的 4 倍。
- layers 表示 Transformer 编码器的层数。
- dropout 是 dropout 概率。
- enable_res_parameter 是一个布尔值，表示是否启用残差连接中的可学习参数。
- self.TRMs 是一个由多个 TransformerBlock 组成的列表，构建了 Transformer 编码器。

Run

在服务器上搭建环境，运行之后的结果为（仅 HAR 数据集）：



与论文中对比，可见结果非常符合。

Metrics Datasets	HAR
FineZero	68.02±0.84
FineZero+	66.53±0.70
TST	87.39±0.27
TNC	87.82±0.18
TS-TCC	77.63±0.20
TS2Vec	78.16±0.80
TimeMAE	**91.31±0.10**
FineZero	66.39±1.04
FineZero+	64.65±0.83
TST	87.18±0.29
TNC	87.63±0.19
TS-TCC	77.09±0.14
TS2Vec	77.43±0.91
TimeMAE	**91.25±0.06**

Modify

要求用 BERT 的网络架构和网络参数初始化模型，并替换 TimeMAE 的Encoder。这里做了一些尝试，修改了 TimeMAE 的 Encoder 部分：

```
from transformers import BertModel, BertConfig

class Encoder(nn.Module):
    def __init__(self, args):
        super(Encoder, self).__init__()
        bert_config = BertConfig(
            hidden_size=args.d_model,
            num_hidden_layers=args.layers,
```

```
            num_attention_heads=args.attn_heads,
            intermediate_size=4 * args.d_model,
            hidden_dropout_prob=args.dropout,
            attention_probs_dropout_prob=args.dropout
        )
        self.bert_model = BertModel(config=bert_config)

    def forward(self, x):
        x = self.bert_model(x)
        x = x.last_hidden_state
        return x
```

但是这里遇到了一系列问题：BertModel 接收的参数是一个规模为 （batch_size, sequence_length）的 整形 tensor，而这里的时间序列数据是一个规模为 (batch_size, sequence_length, dimension) =（128， 7， 64）的浮点型 tensor。使用了一些降维方式，但是仍然无效。

TODO

研究如何将数据映射为 BertModel 可接受的输入方式。

> 不知是否是我方向有点错误？