算法分析大作业报告-22009200443-曾舒蕾

第二题 独自完成

Time-series Generative Adversarial Networks 论文背景

任务背景

时间序列数据在许多领域中（如金融、医疗、生物信号、智能电网等）具有重要的应用价值。这些数据通常包含复杂的时间依赖关系和动态特征，因此如何生成高质量的时间序列数据成为了一个重要的研究问题。

该论文发表时的主要挑战是：时间序列数据具有独特的时间相关性，生成的序列不仅需要在每个时间点上匹配特征分布，还需要捕捉变量之间在时间维度上的复杂动态关系。

在此之前的方法无法同时满足以下两点：

1. 在生成任务中有效捕捉时间序列的复杂动态依赖关系。
2. 提供一种灵活的生成机制，使生成的时间序列不仅真实且具备预测能力。

因此，本文的主要任务是提出一种新型的时间序列生成框架，既能生成高质量的时间序列数据，也能在生成过程中保留时间动态特性，为多种领域的实际应用（如合成数据生成和预测建模）提供支持。

动机

本论文的动机是因为在此之前的时间序列生成方法存在以下不足：

1. **动态捕捉不足**：基于GAN的方法忽视了时间序列的时间依赖性，难以生成具有真实动态特性的序列。
2. **生成控制有限**：自回归模型过于确定性，无法随机生成多样化的时间序列样本。
因此，研究需要一种既能生成高质量时间序列，又能捕捉其时间动态特性的新方法。

贡献

- **提出TimeGAN框架**：结合生成对抗网络（GAN）的灵活性和监督学习对时间动态的控制能力，首次实现同时生成真实和动态一致的时间序列。
- **引入监督损失**：通过监督损失显式学习时间序列的条件分布，提升生成数据的时间动态一致性。
- **嵌入网络设计**：利用嵌入网络，将高维时间序列映射到低维潜在空间，简化建模难度并提高生成效率。
- **广泛实验验证**：在真实和合成数据上，TimeGAN在生成质量（分布匹配）和预测能力（动态一致性）上显著优于现有方法。

方法

本文提出了一种生成高质量时间序列数据的框架 **TimeGAN**，通过结合生成对抗网络（GAN）的无监督学习和监督学习的时间动态控制能力，解决现有方法在时间序列生成中的不足。以下按照论文中的结构详细阐述。

1. Problem Formulation

时间序列数据由静态特征 S 和时间特征 $X_{1:T}$ 组成，目标是学习一个生成模型 $\hat{p}(S, X_{1:T})$ 以逼近真实分布 $p(S, X_{1:T})$。

- 静态特征：$S \in \mathcal{S}$，描述不随时间变化的属性（如性别）。
- 时间特征：$X_{1:T} = (x_1, x_2, \ldots, x_T)$，描述时间序列的动态变化（如生命体征）。

生成目标包括：

1. **全局分布匹配**：生成的序列分布 $\hat{p}(S, X_{1:T})$ 应接近真实分布 $p(S, X_{1:T})$。
   $\min_{\hat{p}} D(p(S, X_{1:T}) \| \hat{p}(S, X_{1:T}))$
   其中 D 是分布间的距离度量。
2. **条件分布匹配**：生成序列的每一时间步条件分布 $\hat{p}(x_t|S, X_{1:t-1})$ 应接近真实条件分布 $p(x_t|S, X_{1:t-1})$。
   $\min_{\hat{p}} \sum_{t=1}^{T} D(p(x_t|S, X_{1:t-1}) \| \hat{p}(x_t|S, X_{1:t-1}))$

2. Proposed Model: Time-series GAN (TimeGAN)

TimeGAN 通过引入以下四个网络模块解决时间序列生成问题：

1. **嵌入网络（Embedding Network）**：将时间序列和静态特征映射到潜在空间。
2. **恢复网络（Recovery Network）**：将潜在表示映射回原始空间。
3. **生成器（Generator）**：从随机噪声生成潜在表示。
4. **判别器（Discriminator）**：区分真实和生成的潜在表示。

3. Embedding and Recovery Functions

嵌入网络和恢复网络帮助模型在潜在空间中高效学习时间动态关系。

(1) 嵌入网络

嵌入网络将原始时间序列 $(S, X_{1:T})$ 映射到潜在空间 $(h_S, h_{1:T})$，定义为：

- 静态特征嵌入：
  $h_S = e_S(s)$
- 时间特征嵌入（递归形式）：
  $h_t = e_X(h_S, h_{t-1}, x_t)$
  其中，e_S 和 e_X 是嵌入网络的参数化函数。

(2) 恢复网络

恢复网络将潜在表示 $(h_S, h_{1:T})$ 映射回原始空间 $(\tilde{s}, \tilde{x}_{1:T})$：

- 恢复静态特征：
  $\tilde{s} = r_S(h_S)$
- 恢复时间特征：
  $\tilde{x}_t = r_X(h_t)$
  其中，r_S 和 r_X 是恢复网络的参数化函数。

(3) 重构损失

嵌入-恢复的目标是保证潜在空间的表示与原始数据一致，定义为重构损失：
$L_R = \mathbb{E}_{(s, x_{1:T}) \sim p} \left[\|s - \tilde{s}\|^2 + \sum_{t=1}^{T} \|x_t - \tilde{x}_t\|^2 \right]$

4. Sequence Generator and Discriminator

生成器和判别器在潜在空间中进行对抗学习，生成高质量的潜在时间序列表示。

(1) 生成器

生成器从随机噪声 $(z_S, z_{1:T})$ 中生成潜在表示 $(\hat{h}_S, \hat{h}_{1:T})$：

- 静态特征生成：
  $\hat{h}_S = g_S(z_S)$
- 时间特征生成（递归形式）：
  $\hat{h}_t = g_X(\hat{h}_S, \hat{h}_{t-1}, z_t)$
  其中，z_S 和 z_t 是随机噪声，g_S 和 g_X 是生成器的参数化函数。

(2) 判别器

判别器用于区分真实潜在表示 $(h_S, h_{1:T})$ 和生成潜在表示 $(\hat{h}_S, \hat{h}_{1:T})$：

- 静态特征判别：
  $y_S = d_S(h_S)$
- 时间特征判别：
  $y_t = d_X(h_t)$
  其中，d_S 和 d_X 是判别器的参数化函数。

(3) 对抗损失

对抗损失通过生成器和判别器的对抗训练定义：

- 判别器损失：
  $L_D = \mathbb{E}_{h \sim p} [\log(D(h))] + \mathbb{E}_{z \sim p_z} [\log(1 - D(G(z)))]$
- 生成器损失：
  $L_G = \mathbb{E}_{z \sim p_z} [\log(1 - D(G(z)))]$

5. Jointly Learning to Encode, Generate, and Iterate

为了捕捉时间序列的条件动态分布，TimeGAN 引入监督损失 L_S，对生成器进行显式指导：
$L_S = \mathbb{E}_{(h_S, h_{1:T}) \sim p} \left[\sum_{t=1}^{T} \|\hat{h}_t - h_t\|^2 \right]$
其中，$\hat{h}_t$ 是生成器生成的潜在表示，h_t 是嵌入网络生成的真实潜在表示。

6. Optimization

TimeGAN 的总损失函数结合了三部分：

1. **重构损失** L_R：保证潜在空间与原始空间一致性。
2. **监督损失** L_S：显式捕捉时间序列动态。
3. **对抗损失** L_G 和 L_D：生成逼真的序列。

总损失函数为：
$L = \lambda_R L_R + \lambda_S L_S + \lambda_G L_G$
其中，$\lambda_R, \lambda_S, \lambda_G$ 是超参数，用于平衡各损失的权重。

训练步骤

1. **阶段 1**：训练嵌入和恢复网络，优化重构损失 L_R。
2. **阶段 2**：训练生成器，优化监督损失 L_S。
3. **阶段 3**：通过对抗损失 L_G 和 L_D，优化生成器和判别器。

代码复现

设备信息

实验设备信息：CPU 为 Intel(R) Xeon(R) Gold 5320，26 核 × 2，总线程数 104；内存总容量为 377 GB，可用容量为 353 GB；GPU 配置为 8 块 NVIDIA GeForce RTX 3090，驱动版本为 535.146.02，CUDA 版本为 12.2。

代码与复现结果

复现结果

```
Start Embedding Network Training
Epoch: 599, Loss: 0.0011: 100%|██████████| 600/600 [02:15<00:00,  4.43it/s]

Start Training with Supervised Loss Only
Epoch: 599, Loss: 0.0080: 100%|██████████| 600/600 [01:49<00:00,  5.50it/s]

Start Joint Training
Epoch: 599, E: 0.0971, G: 1.8348, D: 1.8551: 100%|██████████| 600/600 [18:44<00:00,  1.87s/it]

Saved at path: /h3cstore_ns/jcxie/zsl/timegan-pytorch-main/output/test

Generating Data...
Generated data preview:
[[[-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]]

 [[-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]]]

Model Runtime: 23.00885624885559 mins
```

```
Running feature prediction using original data...
Epoch: 19, Loss: 0.2464: 100%|██████████| 20/20 [00:03<00:00,  6.28it/s]
Epoch: 19, Loss: 2.6676: 100%|██████████| 20/20 [00:03<00:00,  6.21it/s]
Running feature prediction using generated data...
Epoch: 19, Loss: 0.0241: 100%|██████████| 20/20 [00:03<00:00,  6.28it/s]
Epoch: 19, Loss: 0.0206: 100%|██████████| 20/20 [00:03<00:00,  6.23it/s]
Feature prediction results:
(1) Ori: [0.2955 0.239 ]
(2) New: [0.1357 0.1448]

Running one step ahead prediction using original data...
Epoch: 19, Loss: 0.4252: 100%|██████████| 20/20 [00:03<00:00,  6.05it/s]
Running one step ahead prediction using generated data...
Epoch: 19, Loss: 0.3763: 100%|██████████| 20/20 [00:03<00:00,  6.13it/s]
One step ahead prediction results:
(1) Ori: 0.3261
(2) New: 0.3078

Total Runtime: 23.721916536490124 mins
```

run.ipynb

step1 导入库

导入需要的库

```python
import argparse
import logging
import os
import pickle
import random
import shutil
import time

# 3rd-Party Modules
import numpy as np
import torch
import joblib
from sklearn.model_selection import train_test_split

# Self-Written Modules
from data.data_preprocess import data_preprocess
from metrics.metric_utils import (
    feature_prediction, one_step_ahead_prediction, reidentify_score
)

from models.timegan import TimeGAN
from models.utils import timegan_trainer, timegan_generator
```

step2 设置参数

```python
class Config:
    def __init__(self):
        self.device = 'cuda'
        self.exp = 'test'
        self.is_train = True
        self.seed = 0
        self.feat_pred_no = 2
        self.max_seq_len = 100
        self.train_rate = 0.5
        self.emb_epochs = 600
        self.sup_epochs = 600
        self.gan_epochs = 600
        self.batch_size = 128
        self.hidden_dim = 20
        self.num_layers = 3
        self.dis_thresh = 0.15
        self.optimizer = 'adam'
        self.learning_rate = 1e-3

args = Config()

def str2bool(v):
    if isinstance(v, bool):
       return v
    if v.lower() in ('yes', 'true', 't', 'y', '1'):
        return True
    elif v.lower() in ('no', 'false', 'f', 'n', '0'):
        return False
    else:
        raise argparse.ArgumentTypeError('Boolean value expected.')
```

step4 数据集与模型加载

```python
code_dir = os.path.abspath(".")
if not os.path.exists(code_dir):
    raise ValueError(f"Code directory not found at {code_dir}.")

## Data directory
data_path = os.path.abspath("./data")
if not os.path.exists(data_path):
    raise ValueError(f"Data file not found at {data_path}.")
data_dir = os.path.dirname(data_path)
data_file_name = os.path.basename(data_path)

## Output directories
args.model_path = os.path.abspath(f"./output/{args.exp}/")
out_dir = os.path.abspath(args.model_path)
if not os.path.exists(out_dir):
    os.makedirs(out_dir, exist_ok=True)

# TensorBoard directory
tensorboard_path = os.path.abspath("./tensorboard")
```

```python
if not os.path.exists(tensorboard_path):
    os.makedirs(tensorboard_path, exist_ok=True)

print(f"\nCode directory:\t\t\t{code_dir}")
print(f"Data directory:\t\t\t{data_path}")
print(f"Output directory:\t\t{out_dir}")
print(f"TensorBoard directory:\t\t{tensorboard_path}\n")


os.environ['PYTHONHASHSEED'] = str(args.seed)
random.seed(args.seed)
np.random.seed(args.seed)
torch.manual_seed(args.seed)

if args.device == "cuda" and torch.cuda.is_available():
    print("Using CUDA\n")
    args.device = torch.device("cuda:0")
    # torch.cuda.manual_seed_all(args.seed)
    torch.cuda.manual_seed(args.seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
else:
    print("Using CPU\n")
    args.device = torch.device("cpu")


data_path = "data/stock.csv"
X, T, _, args.max_seq_len, args.padding_value = data_preprocess(
    data_path, args.max_seq_len
)

print(f"Processed data: {X.shape} (Idx x MaxSeqLen x Features)\n")
print(f"Original data preview:\n{X[:2, :10, :2]}\n")

args.feature_dim = X.shape[-1]
args.Z_dim = X.shape[-1]


train_data, test_data, train_time, test_time = train_test_split(
    X, T, test_size=args.train_rate, random_state=args.seed
)
```

运行结果

```
Code directory:                    /h3cstore_ns/
Data directory:                    /h3cstore_ns/
Output directory:                  /h3cstore_ns/
TensorBoard directory:             /h3cstore_ns/

Using CUDA

Loading data...

Dropped 504 rows (outliers)

100%|██████████| 3676/3676 [00:06<00:00, 593.
Processed data: (3676, 100, 6) (Idx x MaxSeqL

Original data preview:
[[[ 0.19376718  0.19446839]
  [ 0.19232369  0.19224311]
  [ 0.19594256  0.19481357]
  [ 0.20078938  0.20019403]
  [ 0.19906535  0.20037676]
  [ 0.19672326  0.19752207]
  [ 0.19728439  0.19644191]
  [-1.         -1.        ]
  [-1.         -1.        ]
  [-1.         -1.        ]]

 [[ 0.4860957   0.49640034]
  [ 0.48522808  0.48878844]
  [ 0.48351736  0.48673669]
  [ 0.48463053  0.48547787]
  [ 0.49108043  0.4905124 ]
  [ 0.48256791  0.48940151]
  [ 0.47696925  0.48430077]
  [-1.         -1.        ]
  [-1.         -1.        ]
  [-1.         -1.        ]]]
```

step5 训练代码

```
start = time.time()

model = TimeGAN(args)
if args.is_train == True:
    timegan_trainer(model, train_data, train_time, args)
generated_data = timegan_generator(model, train_time, args)
generated_time = train_time


end = time.time()

print(f"Generated data preview:\n{generated_data[:2, -10:, :2]}\n")
print(f"Model Runtime: {(end - start)/60} mins\n")
```

运行结果

```
Start Embedding Network Training
Epoch: 599, Loss: 0.0011: 100%|██████████| 600/600 [02:15<00:00,  4.43it/s]

Start Training with Supervised Loss Only
Epoch: 599, Loss: 0.0080: 100%|██████████| 600/600 [01:49<00:00,  5.50it/s]

Start Joint Training
Epoch: 599, E: 0.0971, G: 1.8348, D: 1.8551: 100%|██████████| 600/600 [18:44<00:00,  1.87s/it]

Saved at path: /h3cstore_ns/jcxie/zsl/timegan-pytorch-main/output/test

Generating Data...
Generated data preview:
[[[-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]]

 [[-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]
  [-1.0003532 -1.0001502]]]

Model Runtime: 23.00885624885559 mins
```

step6 结果测试与保存

```
with open(f"{args.model_path}/train_data.pickle", "wb") as fb:
    pickle.dump(train_data, fb)
with open(f"{args.model_path}/train_time.pickle", "wb") as fb:
    pickle.dump(train_time, fb)
with open(f"{args.model_path}/test_data.pickle", "wb") as fb:
    pickle.dump(test_data, fb)
with open(f"{args.model_path}/test_time.pickle", "wb") as fb:
    pickle.dump(test_time, fb)
with open(f"{args.model_path}/fake_data.pickle", "wb") as fb:
    pickle.dump(generated_data, fb)
with open(f"{args.model_path}/fake_time.pickle", "wb") as fb:
    pickle.dump(generated_time, fb)



# Define enlarge data and its labels
enlarge_data = np.concatenate((train_data, test_data), axis=0)
enlarge_time = np.concatenate((train_time, test_time), axis=0)
enlarge_data_label = np.concatenate((np.ones([train_data.shape[0], 1]),
np.zeros([test_data.shape[0], 1])), axis=0)

# Mix the order
```

```python
idx = np.random.permutation(enlarge_data.shape[0])
enlarge_data = enlarge_data[idx]
enlarge_data_label = enlarge_data_label[idx]



# 1. Feature prediction
feat_idx = np.random.permutation(train_data.shape[2])[:args.feat_pred_no]
print("Running feature prediction using original data...")
ori_feat_pred_perf = feature_prediction(
    (train_data, train_time),
    (test_data, test_time),
    feat_idx
)
print("Running feature prediction using generated data...")
new_feat_pred_perf = feature_prediction(
    (generated_data, generated_time),
    (test_data, test_time),
    feat_idx
)

feat_pred = [ori_feat_pred_perf, new_feat_pred_perf]

print('Feature prediction results:\n' +
        f'(1) Ori: {str(np.round(ori_feat_pred_perf, 4))}\n' +
        f'(2) New: {str(np.round(new_feat_pred_perf, 4))}\n')

# 2. One step ahead prediction
print("Running one step ahead prediction using original data...")
ori_step_ahead_pred_perf = one_step_ahead_prediction(
    (train_data, train_time),
    (test_data, test_time)
)
print("Running one step ahead prediction using generated data...")
new_step_ahead_pred_perf = one_step_ahead_prediction(
    (generated_data, generated_time),
    (test_data, test_time)
)

step_ahead_pred = [ori_step_ahead_pred_perf, new_step_ahead_pred_perf]

print('One step ahead prediction results:\n' +
        f'(1) Ori: {str(np.round(ori_step_ahead_pred_perf, 4))}\n' +
        f'(2) New: {str(np.round(new_step_ahead_pred_perf, 4))}\n')

print(f"Total Runtime: {(time.time() - start)/60} mins\n")
```

运行结果

```
Running feature prediction using original data...
Epoch: 19, Loss: 0.2464: 100%|██████████| 20/20 [00:03<00:00,  6.28it/s]
Epoch: 19, Loss: 2.6676: 100%|██████████| 20/20 [00:03<00:00,  6.21it/s]
Running feature prediction using generated data...
Epoch: 19, Loss: 0.0241: 100%|██████████| 20/20 [00:03<00:00,  6.28it/s]
Epoch: 19, Loss: 0.0206: 100%|██████████| 20/20 [00:03<00:00,  6.23it/s]
Feature prediction results:
(1) Ori: [0.2955 0.239 ]
(2) New: [0.1357 0.1448]

Running one step ahead prediction using original data...
Epoch: 19, Loss: 0.4252: 100%|██████████| 20/20 [00:03<00:00,  6.05it/s]
Running one step ahead prediction using generated data...
Epoch: 19, Loss: 0.3763: 100%|██████████| 20/20 [00:03<00:00,  6.13it/s]
One step ahead prediction results:
(1) Ori: 0.3261
(2) New: 0.3078

Total Runtime: 23.721916536490124 mins
```

models.timegan.py

```
# -*- coding: UTF-8 -*-
import torch
import numpy as np

class EmbeddingNetwork(torch.nn.Module):
    """The embedding network (encoder) for TimeGAN
    """
    def __init__(self, args):
        super(EmbeddingNetwork, self).__init__()
        self.feature_dim = args.feature_dim
        self.hidden_dim = args.hidden_dim
        self.num_layers = args.num_layers
        self.padding_value = args.padding_value
        self.max_seq_len = args.max_seq_len

        # Embedder Architecture
        self.emb_rnn = torch.nn.GRU(
            input_size=self.feature_dim,
            hidden_size=self.hidden_dim,
            num_layers=self.num_layers,
            batch_first=True
        )
        self.emb_linear = torch.nn.Linear(self.hidden_dim, self.hidden_dim)
        self.emb_sigmoid = torch.nn.Sigmoid()

        # Init weights
        # Default weights of TensorFlow is Xavier Uniform for W and 1 or 0 for b
        # Reference:
        # - https://www.tensorflow.org/api_docs/python/tf/compat/v1/get_variable
        # -
https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/laye
rs/legacy_rnn/rnn_cell_impl.py#L484-L614
        with torch.no_grad():
            for name, param in self.emb_rnn.named_parameters():
```

```
                if 'weight_ih' in name:
                    torch.nn.init.xavier_uniform_(param.data)
                elif 'weight_hh' in name:
                    torch.nn.init.xavier_uniform_(param.data)
                elif 'bias_ih' in name:
                    param.data.fill_(1)
                elif 'bias_hh' in name:
                    param.data.fill_(0)
            for name, param in self.emb_linear.named_parameters():
                if 'weight' in name:
                    torch.nn.init.xavier_uniform_(param)
                elif 'bias' in name:
                    param.data.fill_(0)

    def forward(self, X, T):
        """Forward pass for embedding features from original space into latent
space
        Args:
            - X: input time-series features (B x S x F)
            - T: input temporal information (B)
        Returns:
            - H: latent space embeddings (B x S x H)
        """
        # Dynamic RNN input for ignoring paddings
        X_packed = torch.nn.utils.rnn.pack_padded_sequence(
            input=X,
            lengths=T,
            batch_first=True,
            enforce_sorted=False
        )

        # 128 x 100 x 71
        H_o, H_t = self.emb_rnn(X_packed)

        # Pad RNN output back to sequence length
        H_o, T = torch.nn.utils.rnn.pad_packed_sequence(
            sequence=H_o,
            batch_first=True,
            padding_value=self.padding_value,
            total_length=self.max_seq_len
        )

        # 128 x 100 x 10
        logits = self.emb_linear(H_o)
        # 128 x 100 x 10
        H = self.emb_sigmoid(logits)
        return H

class RecoveryNetwork(torch.nn.Module):
    """The recovery network (decoder) for TimeGAN
    """
    def __init__(self, args):
        super(RecoveryNetwork, self).__init__()
        self.hidden_dim = args.hidden_dim
        self.feature_dim = args.feature_dim
        self.num_layers = args.num_layers
```

```
        self.padding_value = args.padding_value
        self.max_seq_len = args.max_seq_len

        # Recovery Architecture
        self.rec_rnn = torch.nn.GRU(
            input_size=self.hidden_dim,
            hidden_size=self.hidden_dim,
            num_layers=self.num_layers,
            batch_first=True
        )
        self.rec_linear = torch.nn.Linear(self.hidden_dim, self.feature_dim)

        # Init weights
        # Default weights of TensorFlow is Xavier Uniform for W and 1 or 0 for b
        # Reference:
        # - https://www.tensorflow.org/api_docs/python/tf/compat/v1/get_variable
        # -
https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/laye
rs/legacy_rnn/rnn_cell_impl.py#L484-L614
        with torch.no_grad():
            for name, param in self.rec_rnn.named_parameters():
                if 'weight_ih' in name:
                    torch.nn.init.xavier_uniform_(param.data)
                elif 'weight_hh' in name:
                    torch.nn.init.xavier_uniform_(param.data)
                elif 'bias_ih' in name:
                    param.data.fill_(1)
                elif 'bias_hh' in name:
                    param.data.fill_(0)
            for name, param in self.rec_linear.named_parameters():
                if 'weight' in name:
                    torch.nn.init.xavier_uniform_(param)
                elif 'bias' in name:
                    param.data.fill_(0)

    def forward(self, H, T):
        """Forward pass for the recovering features from latent space to original
space
        Args:
            - H: latent representation (B x S x E)
            - T: input temporal information (B)
        Returns:
            - X_tilde: recovered data (B x S x F)
        """
        # Dynamic RNN input for ignoring paddings
        H_packed = torch.nn.utils.rnn.pack_padded_sequence(
            input=H,
            lengths=T,
            batch_first=True,
            enforce_sorted=False
        )

        # 128 x 100 x 10
        H_o, H_t = self.rec_rnn(H_packed)

        # Pad RNN output back to sequence length
```

```
        H_o, T = torch.nn.utils.rnn.pad_packed_sequence(
            sequence=H_o,
            batch_first=True,
            padding_value=self.padding_value,
            total_length=self.max_seq_len
        )

        # 128 x 100 x 71
        X_tilde = self.rec_linear(H_o)
        return X_tilde

class SupervisorNetwork(torch.nn.Module):
    """The Supervisor network (decoder) for TimeGAN
    """
    def __init__(self, args):
        super(SupervisorNetwork, self).__init__()
        self.hidden_dim = args.hidden_dim
        self.num_layers = args.num_layers
        self.padding_value = args.padding_value
        self.max_seq_len = args.max_seq_len

        # Supervisor Architecture
        self.sup_rnn = torch.nn.GRU(
            input_size=self.hidden_dim,
            hidden_size=self.hidden_dim,
            num_layers=self.num_layers-1,
            batch_first=True
        )
        self.sup_linear = torch.nn.Linear(self.hidden_dim, self.hidden_dim)
        self.sup_sigmoid = torch.nn.Sigmoid()

        # Init weights
        # Default weights of TensorFlow is Xavier Uniform for W and 1 or 0 for b
        # Reference:
        # - https://www.tensorflow.org/api_docs/python/tf/compat/v1/get_variable
        # -
https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/laye
rs/legacy_rnn/rnn_cell_impl.py#L484-L614
        with torch.no_grad():
            for name, param in self.sup_rnn.named_parameters():
                if 'weight_ih' in name:
                    torch.nn.init.xavier_uniform_(param.data)
                elif 'weight_hh' in name:
                    torch.nn.init.xavier_uniform_(param.data)
                elif 'bias_ih' in name:
                    param.data.fill_(1)
                elif 'bias_hh' in name:
                    param.data.fill_(0)
            for name, param in self.sup_linear.named_parameters():
                if 'weight' in name:
                    torch.nn.init.xavier_uniform_(param)
                elif 'bias' in name:
                    param.data.fill_(0)

    def forward(self, H, T):
        """Forward pass for the supervisor for predicting next step
```

```
        Args:
            - H: latent representation (B x S x E)
            - T: input temporal information (B)
        Returns:
            - H_hat: predicted next step data (B x S x E)
        """
        # Dynamic RNN input for ignoring paddings
        H_packed = torch.nn.utils.rnn.pack_padded_sequence(
            input=H,
            lengths=T,
            batch_first=True,
            enforce_sorted=False
        )

        # 128 x 100 x 10
        H_o, H_t = self.sup_rnn(H_packed)

        # Pad RNN output back to sequence length
        H_o, T = torch.nn.utils.rnn.pad_packed_sequence(
            sequence=H_o,
            batch_first=True,
            padding_value=self.padding_value,
            total_length=self.max_seq_len
        )

        # 128 x 100 x 10
        logits = self.sup_linear(H_o)
        # 128 x 100 x 10
        H_hat = self.sup_sigmoid(logits)
        return H_hat

class GeneratorNetwork(torch.nn.Module):
    """The generator network (encoder) for TimeGAN
    """
    def __init__(self, args):
        super(GeneratorNetwork, self).__init__()
        self.Z_dim = args.Z_dim
        self.hidden_dim = args.hidden_dim
        self.num_layers = args.num_layers
        self.padding_value = args.padding_value
        self.max_seq_len = args.max_seq_len

        # Generator Architecture
        self.gen_rnn = torch.nn.GRU(
            input_size=self.Z_dim,
            hidden_size=self.hidden_dim,
            num_layers=self.num_layers,
            batch_first=True
        )
        self.gen_linear = torch.nn.Linear(self.hidden_dim, self.hidden_dim)
        self.gen_sigmoid = torch.nn.Sigmoid()

        # Init weights
        # Default weights of TensorFlow is Xavier Uniform for W and 1 or 0 for b
        # Reference:
        # - https://www.tensorflow.org/api_docs/python/tf/compat/v1/get_variable
```

```
        # -
https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/laye
rs/legacy_rnn/rnn_cell_impl.py#L484-L614
        with torch.no_grad():
            for name, param in self.gen_rnn.named_parameters():
                if 'weight_ih' in name:
                    torch.nn.init.xavier_uniform_(param.data)
                elif 'weight_hh' in name:
                    torch.nn.init.xavier_uniform_(param.data)
                elif 'bias_ih' in name:
                    param.data.fill_(1)
                elif 'bias_hh' in name:
                    param.data.fill_(0)
            for name, param in self.gen_linear.named_parameters():
                if 'weight' in name:
                    torch.nn.init.xavier_uniform_(param)
                elif 'bias' in name:
                    param.data.fill_(0)

    def forward(self, Z, T):
        """Takes in random noise (features) and generates synthetic features
within the latent space
        Args:
            - Z: input random noise (B x S x Z)
            - T: input temporal information
        Returns:
            - H: embeddings (B x S x E)
        """
        # Dynamic RNN input for ignoring paddings
        Z_packed = torch.nn.utils.rnn.pack_padded_sequence(
            input=Z,
            lengths=T,
            batch_first=True,
            enforce_sorted=False
        )

        # 128 x 100 x 71
        H_o, H_t = self.gen_rnn(Z_packed)

        # Pad RNN output back to sequence length
        H_o, T = torch.nn.utils.rnn.pad_packed_sequence(
            sequence=H_o,
            batch_first=True,
            padding_value=self.padding_value,
            total_length=self.max_seq_len
        )

        # 128 x 100 x 10
        logits = self.gen_linear(H_o)
        # B x S
        H = self.gen_sigmoid(logits)
        return H

class DiscriminatorNetwork(torch.nn.Module):
    """The Discriminator network (decoder) for TimeGAN
    """
```

```python
    def __init__(self, args):
        super(DiscriminatorNetwork, self).__init__()
        self.hidden_dim = args.hidden_dim
        self.num_layers = args.num_layers
        self.padding_value = args.padding_value
        self.max_seq_len = args.max_seq_len

        # Discriminator Architecture
        self.dis_rnn = torch.nn.GRU(
            input_size=self.hidden_dim,
            hidden_size=self.hidden_dim,
            num_layers=self.num_layers,
            batch_first=True
        )
        self.dis_linear = torch.nn.Linear(self.hidden_dim, 1)

        # Init weights
        # Default weights of TensorFlow is Xavier Uniform for W and 1 or 0 for b
        # Reference:
        # - https://www.tensorflow.org/api_docs/python/tf/compat/v1/get_variable
        # -
https://github.com/tensorflow/tensorflow/blob/v2.3.1/tensorflow/python/keras/laye
rs/legacy_rnn/rnn_cell_impl.py#L484-L614
        with torch.no_grad():
            for name, param in self.dis_rnn.named_parameters():
                if 'weight_ih' in name:
                    torch.nn.init.xavier_uniform_(param.data)
                elif 'weight_hh' in name:
                    torch.nn.init.xavier_uniform_(param.data)
                elif 'bias_ih' in name:
                    param.data.fill_(1)
                elif 'bias_hh' in name:
                    param.data.fill_(0)
            for name, param in self.dis_linear.named_parameters():
                if 'weight' in name:
                    torch.nn.init.xavier_uniform_(param)
                elif 'bias' in name:
                    param.data.fill_(0)

    def forward(self, H, T):
        """Forward pass for predicting if the data is real or synthetic
        Args:
            - H: latent representation (B x S x E)
            - T: input temporal information
        Returns:
            - logits: predicted logits (B x S x 1)
        """
        # Dynamic RNN input for ignoring paddings
        H_packed = torch.nn.utils.rnn.pack_padded_sequence(
            input=H,
            lengths=T,
            batch_first=True,
            enforce_sorted=False
        )

        # 128 x 100 x 10
```

```python
        H_o, H_t = self.dis_rnn(H_packed)

        # Pad RNN output back to sequence length
        H_o, T = torch.nn.utils.rnn.pad_packed_sequence(
            sequence=H_o,
            batch_first=True,
            padding_value=self.padding_value,
            total_length=self.max_seq_len
        )

        # 128 x 100
        logits = self.dis_linear(H_o).squeeze(-1)
        return logits

class TimeGAN(torch.nn.Module):
    """Implementation of TimeGAN (Yoon et al., 2019) using PyTorch
    Reference:
    - https://papers.nips.cc/paper/2019/hash/c9efe5f26cd17ba6216bbe2a7d26d490-
Abstract.html
    - https://github.com/jsyoon0823/TimeGAN
    """
    def __init__(self, args):
        super(TimeGAN, self).__init__()
        self.device = args.device
        self.feature_dim = args.feature_dim
        self.Z_dim = args.Z_dim
        self.hidden_dim = args.hidden_dim
        self.max_seq_len = args.max_seq_len
        self.batch_size = args.batch_size

        self.embedder = EmbeddingNetwork(args)
        self.recovery = RecoveryNetwork(args)
        self.generator = GeneratorNetwork(args)
        self.supervisor = SupervisorNetwork(args)
        self.discriminator = DiscriminatorNetwork(args)

    def _recovery_forward(self, X, T):
        """The embedding network forward pass and the embedder network loss
        Args:
            - X: the original input features
            - T: the temporal information
        Returns:
            - E_loss: the reconstruction loss
            - X_tilde: the reconstructed features
        """
        # Forward Pass
        H = self.embedder(X, T)
        X_tilde = self.recovery(H, T)

        # For Joint training
        H_hat_supervise = self.supervisor(H, T)
        G_loss_S = torch.nn.functional.mse_loss(
            H_hat_supervise[:,:-1,:],
            H[:,1:,:]
        ) # Teacher forcing next output
```

```
        # Reconstruction Loss
        E_loss_T0 = torch.nn.functional.mse_loss(X_tilde, X)
        E_loss0 = 10 * torch.sqrt(E_loss_T0)
        E_loss = E_loss0 + 0.1 * G_loss_S
        return E_loss, E_loss0, E_loss_T0

    def _supervisor_forward(self, X, T):
        """The supervisor training forward pass
        Args:
            - X: the original feature input
        Returns:
            - S_loss: the supervisor's loss
        """
        # Supervision Forward Pass
        H = self.embedder(X, T)
        H_hat_supervise = self.supervisor(H, T)

        # Supervised loss
        S_loss = torch.nn.functional.mse_loss(H_hat_supervise[:,:-1,:],
H[:,1:,:])        # Teacher forcing next output
        return S_loss

    def _discriminator_forward(self, X, T, Z, gamma=1):
        """The discriminator forward pass and adversarial loss
        Args:
            - X: the input features
            - T: the temporal information
            - Z: the input noise
        Returns:
            - D_loss: the adversarial loss
        """
        # Real
        H = self.embedder(X, T).detach()

        # Generator
        E_hat = self.generator(Z, T).detach()
        H_hat = self.supervisor(E_hat, T).detach()

        # Forward Pass
        Y_real = self.discriminator(H, T)            # Encoded original data
        Y_fake = self.discriminator(H_hat, T)        # Output of generator +
supervisor
        Y_fake_e = self.discriminator(E_hat, T)      # Output of generator

        D_loss_real =
torch.nn.functional.binary_cross_entropy_with_logits(Y_real,
torch.ones_like(Y_real))
        D_loss_fake =
torch.nn.functional.binary_cross_entropy_with_logits(Y_fake,
torch.zeros_like(Y_fake))
        D_loss_fake_e =
torch.nn.functional.binary_cross_entropy_with_logits(Y_fake_e,
torch.zeros_like(Y_fake_e))

        D_loss = D_loss_real + D_loss_fake + gamma * D_loss_fake_e
```

```
        return D_loss

    def _generator_forward(self, X, T, Z, gamma=1):
        """The generator forward pass
        Args:
            - X: the original feature input
            - T: the temporal information
            - Z: the noise for generator input
        Returns:
            - G_loss: the generator's loss
        """
        # Supervisor Forward Pass
        H = self.embedder(X, T)
        H_hat_supervise = self.supervisor(H, T)

        # Generator Forward Pass
        E_hat = self.generator(Z, T)
        H_hat = self.supervisor(E_hat, T)

        # Synthetic data generated
        X_hat = self.recovery(H_hat, T)

        # Generator Loss
        # 1. Adversarial loss
        Y_fake = self.discriminator(H_hat, T)        # Output of supervisor
        Y_fake_e = self.discriminator(E_hat, T)      # Output of generator

        G_loss_U = torch.nn.functional.binary_cross_entropy_with_logits(Y_fake,
torch.ones_like(Y_fake))
        G_loss_U_e =
torch.nn.functional.binary_cross_entropy_with_logits(Y_fake_e,
torch.ones_like(Y_fake_e))

        # 2. Supervised loss
        G_loss_S = torch.nn.functional.mse_loss(H_hat_supervise[:,:-1,:],
H[:,1:,:])        # Teacher forcing next output

        # 3. Two Momments
        G_loss_V1 = torch.mean(torch.abs(torch.sqrt(X_hat.var(dim=0,
unbiased=False) + 1e-6) - torch.sqrt(X.var(dim=0, unbiased=False) + 1e-6)))
        G_loss_V2 = torch.mean(torch.abs((X_hat.mean(dim=0)) - (X.mean(dim=0))))

        G_loss_V = G_loss_V1 + G_loss_V2

        # 4. Summation
        G_loss = G_loss_U + gamma * G_loss_U_e + 100 * torch.sqrt(G_loss_S) + 100
* G_loss_V

        return G_loss

    def _inference(self, Z, T):
        """Inference for generating synthetic data
        Args:
            - Z: the input noise
            - T: the temporal information
        Returns:
```

```
            - X_hat: the generated data
        """
        # Generator Forward Pass
        E_hat = self.generator(Z, T)
        H_hat = self.supervisor(E_hat, T)

        # Synthetic data generated
        X_hat = self.recovery(H_hat, T)
        return X_hat

    def forward(self, X, T, Z, obj, gamma=1):
        """
        Args:
            - X: the input features (B, H, F)
            - T: the temporal information (B)
            - Z: the sampled noise (B, H, Z)
            - obj: the network to be trained (`autoencoder`, `supervisor`,
`generator`, `discriminator`)
            - gamma: loss hyperparameter
        Returns:
            - loss: The loss for the forward pass
            - X_hat: The generated data
        """
        if obj != "inference":
            if X is None:
                raise ValueError("`X` should be given")

            X = torch.FloatTensor(X)
            X = X.to(self.device)

        if Z is not None:
            Z = torch.FloatTensor(Z)
            Z = Z.to(self.device)

        if obj == "autoencoder":
            # Embedder & Recovery
            loss = self._recovery_forward(X, T)

        elif obj == "supervisor":
            # Supervisor
            loss = self._supervisor_forward(X, T)

        elif obj == "generator":
            if Z is None:
                raise ValueError("`Z` is not given")

            # Generator
            loss = self._generator_forward(X, T, Z)

        elif obj == "discriminator":
            if Z is None:
                raise ValueError("`Z` is not given")

            # Discriminator
            loss = self._discriminator_forward(X, T, Z)
```

```
            return loss

        elif obj == "inference":

            X_hat = self._inference(Z, T)
            X_hat = X_hat.cpu().detach()

            return X_hat

        else: raise ValueError("`obj` should be either `autoencoder`,
`supervisor`, `generator`, or `discriminator`")

        return loss
```

dataset.py

```
# -*- coding: UTF-8 -*-
import numpy as np
import torch

class TimeGANDataset(torch.utils.data.Dataset):
    """TimeGAN Dataset for sampling data with their respective time

    Args:
        - data (numpy.ndarray): the padded dataset to be fitted (D x S x F)
        - time (numpy.ndarray): the length of each data (D)
    Parameters:
        - x (torch.FloatTensor): the real value features of the data
        - t (torch.LongTensor): the temporal feature of the data
    """
    def __init__(self, data, time=None, padding_value=None):
        # sanity check
        if len(data) != len(time):
            raise ValueError(
                f"len(data) `{len(data)}` != len(time) {len(time)}"
            )

        if isinstance(time, type(None)):
            time = [len(x) for x in data]

        self.X = torch.FloatTensor(data)
        self.T = torch.LongTensor(time)

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return self.X[idx], self.T[idx]

    def collate_fn(self, batch):
        """Minibatch sampling
        """
        # Pad sequences to max length
        X_mb = [X for X in batch[0]]
```

```
    # The actual length of each data
    T_mb = [T for T in batch[1]]

    return X_mb, T_mb
```