different network

- show validation and test acc.

Introduction

本次模型的建模流程主要分成三大步骤进行：第一部分进行神经网络结构的搭建与超参数的调优，这部分我们主要采用了三种不同的神经网络结构，分别是：MLP、LSTM、GRU。

最后通过对模型的评估，本次分析最终选择了GRU作为最终模型。

模型评估指标选择

本次分析中，我们采用了三种指标来评估模型的表现：**RMSE**、**MAPE**、**R-squared**。

- 在**RMSE**中，我们采用了**Keras**的**mean_squared_error**作为损失函数，并在**Keras**的**metrics**参数中加入了**mean_absolute_percentage_error**和**r_square**，用于在模型训练过程中实时监控模型的表现。

```
# 1. 导入所需库
import numpy as np, pandas as pd, matplotlib as mpl, matplotlib.pyplot as plt, os, sys, time, datetime
%matplotlib inline
%load_ext autotime
import tensorflow as tf
from tensorflow.keras import models, layers, callbacks, optimizers, metrics, losses, utils, backend as K
tf.keras.backend.set_floatx('float32')
print("Using Keras version", tf.keras.__version__, "with tensorflow-",tf.__version__)

# 2. 定义工具函数
def mape(y_true, y_pred):
    y_true = tf.convert_to_tensor(y_true, dtype=tf.float32)
    y_pred = tf.convert_to_tensor(y_pred, dtype=y_true.dtype)
    diff = tf.abs((y_true - y_pred) / tf.clip_by_value(tf.abs(y_true), 1e-7, tf.reduce_max(tf.abs(y_true))))
    return 100. * tf.reduce_mean(diff)

def r_square(y_true, y_pred):
    from tensorflow.keras.metrics import mean_squared_error as mse
    from tensorflow.keras.metrics import mean_absolute_error as mae
    from tensorflow.keras.metrics import mean_absolute_percentage_error as mape
    from tensorflow.keras.metrics import cosine_similarity as cs
    SS_res = K.sum(K.square(y_true - y_pred))
    SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
    return (1 - SS_res/(SS_tot + K.epsilon()))

# 3. 定义时间序列数据集类
class TSDataset:
    def __init__(self,
                 filepath,
                 predicts,
                 exogenous=None,
                 test_ratio=0.1,
                 ):
        assert len(predicts) > 0, "target variable cannot be empty."
        assert len(set(predicts) - set(pd.read_csv(filepath, nrows=0).columns)) == 0, "predicts must be in dataset."
        assert len(set(exogenous) - set(pd.read_csv(filepath, nrows=0).columns)) == len(exogenous), "exogenous must be in dataset."
        assert test_ratio * len(pd.read_csv(filepath)) > len(predicts), "test_ratio too small."
        self.df = pd.read_csv(filepath, encoding='utf-8')
        self.predicts = predicts
        self.exogenous = exogenous if exogenous else []
        self.test_ratio = test_ratio

    @property
    def n_predicts(self):
        return len(self.predicts)

    @property
    def n_exogenous(self):
        return len(self.exogenous)

    @property
    def n_features(self):
        return self.n_predicts + self.n_exogenous

    @property
    def n_samples(self):
        return len(self.df) - self.n_features

    @property
    def n_test(self):
        return int(self.n_samples * self.test_ratio)

    @property
    def n_train(self):
        return self.n_samples - self.n_test

    @property
    def n_lookbacks(self):
        return self.n_train - self.n_features

    @property
    def n_folds(self):
        return self.n_lookbacks - self.n_features

    @property
    def n_sliders(self):
        return self.n_lookbacks + self.n_features

    @property
    def n_rows(self):
        return self.n_sliders + self.n_features

    @property
    def n_cols(self):
        return self.n_features

    @property
    def n_inputs(self):
        return self.n_lookbacks, self.n_features

    @property
    def n_outputs(self):
        return self.n_predicts

    @property
    def n_steps(self):
        return self.n_lookbacks

    @property
    def n_seqs(self):
        return self.n_folds

    @property
    def n_batches(self):
        return self.n_folds

    @property
    def n_epochs(self):
        return self.n_folds

    @property
    def n_steps_per_epoch(self):
        return self.n_folds

    @property
    def n_steps_per_batch
```