시계열을 위한 딥러닝

面向时间序列的深度学习

다양한 종류의 시계열 작업

各种类型的时间序列任务

기온 예측 문제

气温预测问题

```
!wget https://s3.amazonaws.com/keras-datasets/jena_climate_2009_2016.csv.zip
!unzip jena_climate_2009_2016.csv.zip
```

예나 날씨 데이터셋 조사하기

调查耶拿（德国城市名）天气数据集

```
import os
fname = os.path.join("jena_climate_2009_2016.csv")

with open(fname) as f:
    data = f.read()

lines = data.split("\n")
header = lines[0].split(",")
lines = lines[1:]
print(header)
print(len(lines))
```

Date Time	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H2OC (mmol/mol)	rho (g/m**3)	wv (m/s)	max. wv (m/s)	wd (deg)
01.01.2009 00:10:00	996.52	-8.02	265.4	-8.9	93.3	3.33	3.11	0.22	1.94	3.12	1307.75	1.03	1.75	152.3
01.01.2009 00:20:00	996.57	-8.41	265.01	-9.28	93.4	3.23	3.02	0.21	1.89	3.03	1309.8	0.72	1.5	136.1
01.01.2009 00:30:00	996.53	-8.51	264.91	-9.31	93.9	3.21	3.01	0.2	1.88	3.02	1310.24	0.19	0.63	171.6
01.01.2009 00:40:00	996.51	-8.31	265.12	-9.07	94.2	3.26	3.07	0.19	1.92	3.08	1309.19	0.34	0.5	198
01.01.2009 00:50:00	996.51	-8.27	265.15	-9.04	94.1	3.27	3.08	0.19	1.92	3.09	1309	0.32	0.63	214.3

데이터 파싱

数据解析

```
import numpy as np
temperature = np.zeros((len(lines),))
raw_data = np.zeros((len(lines), len(header) - 1))
for i, line in enumerate(lines):
    values = [float(x) for x in line.split(",")[1:]]
    temperature[i] = values[1]
    raw_data[i, :] = values[:]
```

전체 기온을 그래프로 그리기

将整体气温绘制成图表

```
from matplotlib import pyplot as plt
plt.plot(range(len(temperature)), temperature)
plt.show()
```

처음 10일간의 기온을 그래프로 그리기

将前10天的气温绘制成图表

```
plt.plot(range(1440), temperature[:1440])
plt.show()
```

각 분할에 사용할 샘플 수 계산하기

计算各划分所使用的样本数量

```
num_train_samples = int(0.5 * len(raw_data))
num_val_samples = int(0.25 * len(raw_data))
num_test_samples = len(raw_data) - num_train_samples - num_val_samples
print("num_train_samples:", num_train_samples)
print("num_val_samples:", num_val_samples)
print("num_test_samples:", num_test_samples)
```

데이터 준비

数据准备

데이터 정규화

数据归一化

```
mean = raw_data[:num_train_samples].mean(axis=0)
raw_data -= mean
std = raw_data[:num_train_samples].std(axis=0)
raw_data /= std
```

```
import numpy as np
from tensorflow import keras
int_sequence = np.arange(10)
dummy_dataset = keras.utils.timeseries_dataset_from_array(
    data=int_sequence[:-3],
    targets=int_sequence[3:],
    sequence_length=3,
    batch_size=2,
)

for inputs, targets in dummy_dataset:
    for i in range(inputs.shape[0]):
        print([int(x) for x in inputs[i]], int(targets[i]))
```

훈련, 검증, 테스트 데이터셋 만들기

构建训练、验证和测试数据集

```
sampling_rate = 6
sequence_length = 120
delay = sampling_rate * (sequence_length + 24 - 1)
batch_size = 256

train_dataset = keras.utils.timeseries_dataset_from_array(
    raw_data[:-delay],
    targets=temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=0,
    end_index=num_train_samples)

val_dataset = keras.utils.timeseries_dataset_from_array(
    raw_data[:-delay],
    targets=temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=num_train_samples,
    end_index=num_train_samples + num_val_samples)
```

```
test_dataset = keras.utils.timeseries_dataset_from_array(
    raw_data[:-delay],
    targets=temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=num_train_samples + num_val_samples)
```

훈련 데이터셋의 배치 크기 확인하기

查看训练数据集的批量大小

```
for samples, targets in train_dataset:
    print("Sample size:", samples.shape)
    print("Target size:", targets.shape)
    break
```

```
import tensorflow as tf

def fix_shapes(x, y):
    x = tf.cast(x, tf.float32)
    y = tf.cast(y, tf.float32)
    y = tf.reshape(y, (-1, 1))
    x = tf.ensure_shape(x, [None, sequence_length, raw_data.shape[-1]])
    y = tf.ensure_shape(y, [None, 1])
    return x, y

train_dataset = train_dataset.map(fix_shapes).prefetch(tf.data.AUTOTUNE)
val_dataset   = val_dataset.map(fix_shapes).prefetch(tf.data.AUTOTUNE)
test_dataset  = test_dataset.map(fix_shapes).prefetch(tf.data.AUTOTUNE)

print(train_dataset.element_spec)
```

```
for samples, targets in train_dataset:
    print("Sample size:", samples.shape)
    print("Target size:", targets.shape)
    break
```

상식 수준의 기준점

常识水平的基准点

상식적인 기준 모델의 MAE 계산하기

计算常识性基线模型的 MAE

```
def evaluate_naive_method(dataset):
    total_abs_err = 0.
    samples_seen = 0
    for samples, targets in dataset:
        preds = samples[:, -1, 1] * std[1] + mean[1]
        total_abs_err += np.sum(np.abs(preds - targets))
        samples_seen += samples.shape[0]
    return total_abs_err / samples_seen

print(f"Validation MAE: {evaluate_naive_method(val_dataset):.2f}")
print(f"Test MAE: {evaluate_naive_method(test_dataset):.2f}")
```

기본적인 머신 러닝 모델 시도해 보기

尝试基本的机器学习模型

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.Flatten()(inputs)
```

```
x = layers.Dense(16, activation="relu")(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_dense.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
```

```
print(train_dataset.element_spec)
```

```
history = model.fit(train_dataset,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)

model = keras.models.load_model("jena_dense.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

밀집 연결 모델 훈련하고 평가하기

训练**并**评**估全**连**接模型**

결과 그래프 그리기

绘**制**结**果**图**表**

```
import matplotlib.pyplot as plt
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, "bo", label="Training MAE")
plt.plot(epochs, val_loss, "b", label="Validation MAE")
plt.title("Training and validation MAE")
plt.legend()
plt.show()
```

1D 합성곱 모델 시도해 보기

尝试一维卷积模型

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.Conv1D(8, 24, activation="relu")(inputs)
x = layers.MaxPooling1D(2)(x)
x = layers.Conv1D(8, 12, activation="relu")(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Conv1D(8, 6, activation="relu")(x)
x = layers.GlobalAveragePooling1D()(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_conv.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)

model = keras.models.load_model("jena_conv.keras")
print(f"테스트 MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

```
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
```

```
plt.plot(epochs, loss, "bo", label="Training MAE")
plt.plot(epochs, val_loss, "b", label="Validation MAE")
plt.title("Training and validation MAE")
plt.legend()
plt.show()
```

첫 번째 순환 신경망

第一个循环神经网络

간단한 LSTM 기반 모델

简单的基于 LSTM 的模型

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(16)(inputs)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_lstm.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)

model = keras.models.load_model("jena_lstm.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

```
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs[1:], loss[1:], "bo", label="Training MAE")
plt.plot(epochs[1:], val_loss[1:], "b", label="Validation MAE")
plt.title("Training and validation MAE")
plt.legend()
plt.show()
```

순환 신경망 이해하기

理解循环神经网络

넘파이로 구현한 간단한 RNN

用 NumPy 实现的简单 RNN

```
import numpy as np
timesteps = 100
input_features = 32
output_features = 64
inputs = np.random.random((timesteps, input_features))
state_t = np.zeros((output_features,))
W = np.random.random((output_features, input_features))
U = np.random.random((output_features, output_features))
b = np.random.random((output_features,))
successive_outputs = []
for input_t in inputs:
    output_t = np.tanh(np.dot(W, input_t) + np.dot(U, state_t) + b)
    successive_outputs.append(output_t)
    state_t = output_t
final_output_sequence = np.stack(successive_outputs, axis=0)
```

케라스의 순환 층

Keras 的循环层

어떤 길이의 시퀀스도 처리할 수 있는 RNN 층

임의의 길이의 시퀀스를 처리할 수 있는 RNN 계층

```
num_features = 14
inputs = keras.Input(shape=(None, num_features))
outputs = layers.SimpleRNN(16)(inputs)
```

마지막 출력 스텝만 반환하는 RNN 층

仅**返回最后一步**输**出的 RNN** 层

```
num_features = 14
steps = 120
inputs = keras.Input(shape=(steps, num_features))
outputs = layers.SimpleRNN(16, return_sequences=False)(inputs)
print(outputs.shape)
```

전체 출력 시퀀스를 반환하는 RNN 층

返回完整输**出序列的 RNN** 层

```
num_features = 14
steps = 120
inputs = keras.Input(shape=(steps, num_features))
outputs = layers.SimpleRNN(16, return_sequences=True)(inputs)
print(outputs.shape)
```

스태킹(stacking) RNN 층

堆叠 RNN 层

```
inputs = keras.Input(shape=(steps, num_features))
x = layers.SimpleRNN(16, return_sequences=True)(inputs)
x = layers.SimpleRNN(16, return_sequences=True)(x)
outputs = layers.SimpleRNN(16)(x)
```

순환 신경망의 고급 사용법

循环神经网络的高级用法

과대적합을 감소하기 위해 순환 드롭아웃 사용하기

使用循环式 Dropout 来降低过拟合

드롭아웃 규제를 적용한 LSTM 모델 훈련하고 평가하기

训练**并**评**估**应**用 Dropout 正**则**化的 LSTM 模型**

```
# inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
# # 훈련 속도를 높이기 위해 순환 드롭아웃을 제외합니다.
# # 为了提高训练速度, 省略循环 Dropout。
# #x = layers.LSTM(32, recurrent_dropout=0.25)(inputs)

# x = layers.LSTM(32)(inputs)
# x = layers.Dropout(0.5)(x)
# outputs = layers.Dense(1)(x)
# model = keras.Model(inputs, outputs)

# callbacks = [
#     keras.callbacks.ModelCheckpoint("jena_lstm_dropout.keras",
#                                     save_best_only=True)
# ]
# model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
# history = model.fit(train_dataset,
```

```
#                   epochs=50,
#                   validation_data=val_dataset,
#                   callbacks=callbacks)
```

```
import gdown

url = "https://drive.google.com/uc?id=1ZYA3ljzQE1Hn1BJxFl4KOycRnRjmnpRz"
gdown.download(url, output="jena_lstm_dropout.keras")

url = "https://drive.google.com/uc?id=1vpd014itahNJhsTDO2reTpl4sZgLTYqZ"
gdown.download(url, output="history.json")

from keras.callbacks import History
import json
history = History()

with open("history.json","r") as f:
  history.history = json.load(f)
```

```
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, "bo", label="Training MAE")
plt.plot(epochs, val_loss, "b", label="Validation MAE")
plt.title("Training and validation MAE")
plt.legend()
plt.show()
```

```
inputs = keras.Input(shape=(sequence_length, num_features))
x = layers.LSTM(32, recurrent_dropout=0.2, unroll=True)(inputs)
```

스태킹 순환 층

堆叠循环层

드롭아웃 규제와 스태킹을 적용한 GRU 모델을 훈련하고 평가하기

训练**并**评**估**应**用 Dropout 正**则**化与堆**叠**的 GRU（门控循**环单**元）模型**

```
# inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
# # 훈련 속도를 높이기 위해 순환 드롭아웃을 제외합니다.
# # 为了提高训练速度, 省略循环 Dropout。
# # x = layers.GRU(32, recurrent_dropout=0.5, return_sequences=True)(inputs)
# # x = layers.GRU(32, recurrent_dropout=0.5)(x)
# x = layers.GRU(32, return_sequences=True)(inputs)
# x = layers.GRU(32)(x)
# x = layers.Dropout(0.5)(x)
# outputs = layers.Dense(1)(x)
# model = keras.Model(inputs, outputs)

# callbacks = [
#     keras.callbacks.ModelCheckpoint("jena_stacked_gru_dropout.keras",
#                                     save_best_only=True)
# ]
# model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])

# history = model.fit(train_dataset,
#                   epochs=50,
#                   validation_data=val_dataset,
#                   callbacks=callbacks)
```

```
import gdown

url = "https://drive.google.com/uc?id=1ttyMy8RV5kzLyjboorbjlJXQ8jDjOmkU"
gdown.download(url, output="jena_stacked_gru_dropout.keras")

url = "https://drive.google.com/uc?id=1ezPXdSDF8phlTtmr7lG_KljDRMTS6vbw"
gdown.download(url, output="history.json")
```

```
from keras.callbacks import History
import json
history = History()

with open("history.json","r") as f:
  history.history = json.load(f)
```

```
model = keras.models.load_model("jena_stacked_gru_dropout.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

```
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, "bo", label="Training MAE")
plt.plot(epochs, val_loss, "b", label="Validation MAE")
plt.title("Training and validation MAE")
plt.legend()
plt.show()
```

양방향 RNN 사용하기

使用双向 RNN

시퀀스를 거꾸로 뒤집어 LSTM 모델 훈련하기 (책에는 없음)

通过反转序列来训练 LSTM 模型（书中未收录）

```
def train_generator():
    while True:
        for samples, targets in train_dataset:
            yield samples[:, ::-1, :], targets

def val_generator():
    while True:
        for samples, targets in val_dataset:
            yield samples[:, ::-1, :], targets

train_gen = train_generator()
val_gen = val_generator()
```

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
# 훈련 속도를 높이기 위해 순환 드롭아웃을 제외합니다.
# 为了提高训练速度, 省略循环 Dropout。
# x = layers.LSTM(32, recurrent_dropout=0.25)(inputs)
x = layers.LSTM(32)(inputs)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_gen,
                    epochs=10,
                    steps_per_epoch=819,
                    validation_data=val_gen,
                    validation_steps=410)
```

```
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, "bo", label="Training MAE")
plt.plot(epochs, val_loss, "b", label="Validation MAE")
plt.title("Training and validation MAE")
plt.legend()
plt.show()
```

양방향 LSTM 모델 훈련하고 평가하기

训练并评估双向 LSTM 模型

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.Bidirectional(layers.LSTM(16))(inputs)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=10,
                    validation_data=val_dataset)
```

```
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, "bo", label="Training MAE")
plt.plot(epochs, val_loss, "b", label="Validation MAE")
plt.title("Training and validation MAE")
plt.legend()
plt.show()
```