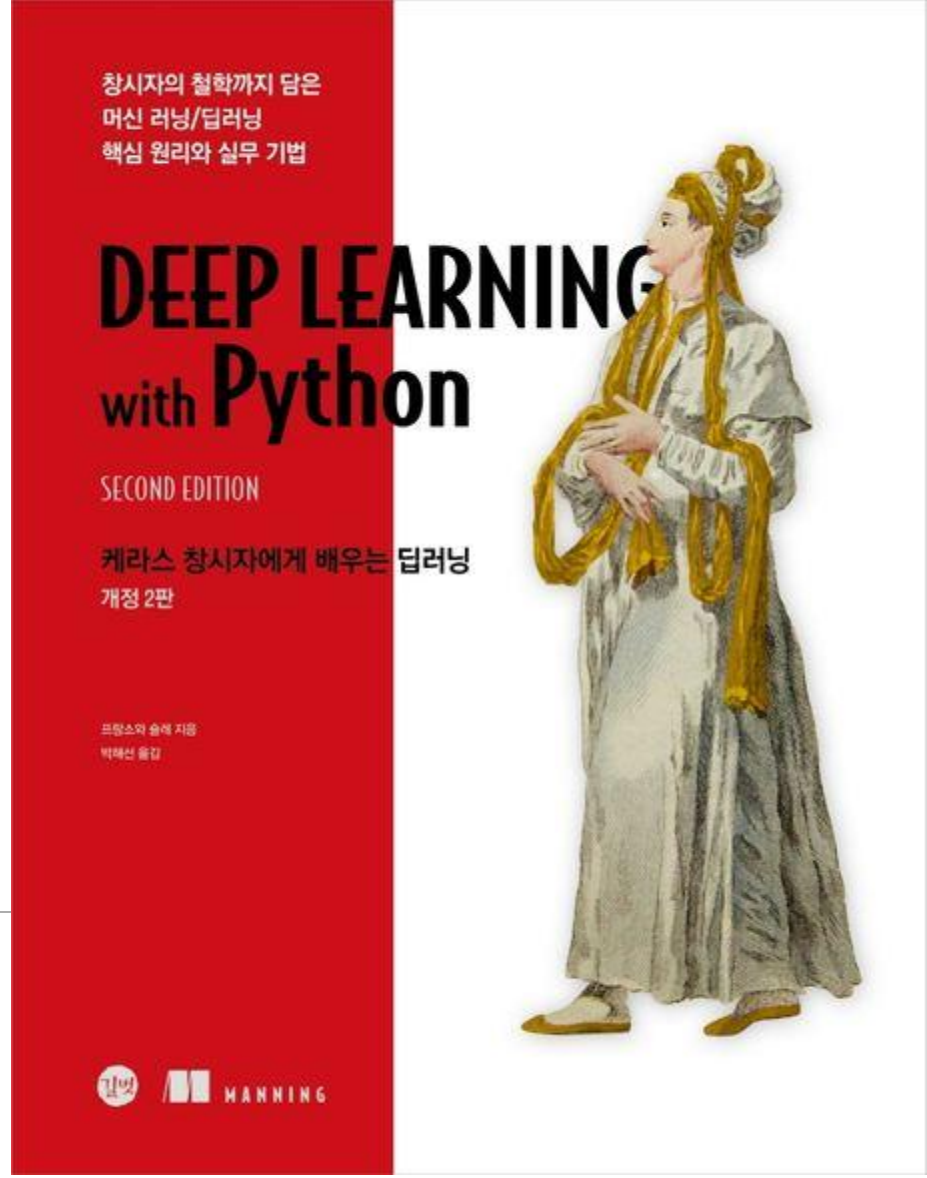
딥러닝

대구가톨릭대학교 AI빅데이터공학과

이 승 민



10章 面向时间序列的深度学习

10.1 各类时间序列任务

10.1 各类时间序列任务

各种类型的时间序列任务

- **时间序列（timeseries）数据**是指以固定间隔进行测量所获得的所有数据。
  - 例如：股票的每日价格、城市的每小时电力消耗量、商店的每周销售额等。
- 时间序列既可以来自**自然现象**（如地震活动、河中鱼类数量变化、某地区天气），也可以反映**人类的活动模式**（如网站访问量、国家GDP、信用卡交易等）。

- 与之前的数据不同，要处理时间序列，必须理解**系统的动态特性（dynamics）**。
  - 例如：周期性、时间趋势、规律性模式与突发性增长等。

10.1 各类时间序列任务

各种类型的时间序列任务

- **最常见的时间序列任务是预测（forecasting）**
- 即根据**当前时间点的时间序列数据**，预测**未来将要发生的事情**。
  - 预测几个小时后的电力消耗量 → 可用于估算电力需求
  - 预测几个月后的收益 → 可用于预算规划
  - 预测几天后的天气 → 可用于日程安排
- **本章的重点：预测（Forecasting）**
- **基于时间序列可进行的任务非常多样**
  - **分类（Classification）**
    - 给时间序列分配一个或多个**类别标签（categorical label）**
    - 例如：给出网站访问者行为的时间序列，判断该访问者是**机器人（bot）还是真实用户（human）**

10.1 各类时间序列任务

各类时间序列任务

- **事件检测（Event Detection）**
  - 在连续的数据流中识别预计会发生的特定事件
  - 特别有用的应用是“热词（hotword）检测”
  - 模型监测音频流，当检测到“OK Google”或“你好 Alexa”这样的唤醒词时触发响应

- **异常检测（Anomaly Detection）**
  - 在连续的数据流中检测异常现象
  - 如果公司网络中出现异常活动，可能意味着外部攻击
  - 如果生产线上测得异常值，需要人工前往检查
  - 异常检测通常使用无监督学习（unsupervised learning）进行
  - 由于往往事先不知道要寻找的异常类型，因此难以用具体的异常样本来训练模型

10.1各类时间序列任务

各类时间序列任务

- 在处理时间序列时，可以看到许多领域特化的数据表示方法。
- 例如，你可能听说过傅里叶变换（Fourier transform），它将时间序列值分解为多个不同的频率成分。
- 傅里叶变换对具有周期性和振动特征的数据（如声音、高层建筑的振动、脑电波等）非常有用。
- 在深度学习中，可以使用傅里叶分析或梅尔频率（Mel-frequency）分析等特定领域的特征表示形式，
- 作为特征工程的一部分，在模型训练前对数据进行预处理，从而使模型更容易处理任务。
- 本节不涉及此类技术，
- 而是将重点放在建模部分

10.2 温度预测问题

10.2 温度预测问题

温度预测问题

- 本章中的所有代码示例都围绕一个问题展开。
- 已知来自建筑屋顶传感器的时间序列数据（例如气压、湿度等每小时的测量值），预测24小时后的温度。
- 接下来你会发现，这是一个相当困难的问题！
- 通过这个温度预测任务，将强调时间序列数据与之前所见的数据集在本质上不同。
- 我们将看到，全连接网络和卷积网络并不适合处理这类数据集，
- 而“循环神经网络（RNN）”在解决此类问题上表现更为出色。

10.2 温度预测问题

温度预测问题

- 该数据收集自德国耶拿（Jena）市的“马克斯·普朗克生物地球化学研究所（Max Planck Institute for Biogeochemistry）”的气象观测站。
- 数据集中包含温度、气压、湿度、风向等 14 个观测值，每隔 10 分钟记录一次。
- 原始数据从 2003 年开始记录，但在本示例中仅使用 2009 年至 2016 年之间的数据。
- 首先需要下载并加载数据。

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

10.2 温度预测问题

温度预测问题

- 分析一下数据

코드 10-1 예나 날씨 데이터셋 조사하기

```
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))
```

10.2 温度预测问题

温度预测问题

- 输出的行数为 **420,551 行**（每一行代表一个时间步，包含日期和 14 个天气信息的记录）
- 表头如下

```
["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)"]
```

10.2 温度预测问题

温度预测问题

- 将全部 420,551 条数据 转换为 Numpy 数组
- 将 温度（摄氏度） 作为一个单独的数组，其余数据组成另一个数组
- 第二个数组包含用于预测未来温度的特征
- 排除 ‘Date Time’ 列

코드 10-2 데이터 파싱

```
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] -------- 두 번째 열을 'temperature' 배열에 저장합니다.
    raw_data[i, :] = values[:] -------- (온도를 포함하여) 모든 열을 'raw_data' 배열에 저장합니다.
```

```
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]     # 将第二列存入 'temperature' 数组
    raw_data[i, :] = values[:]     # 将包含温度在内的所有列存入 'raw_data' 数组
```

10.2 温度预测问题

温度预测问题

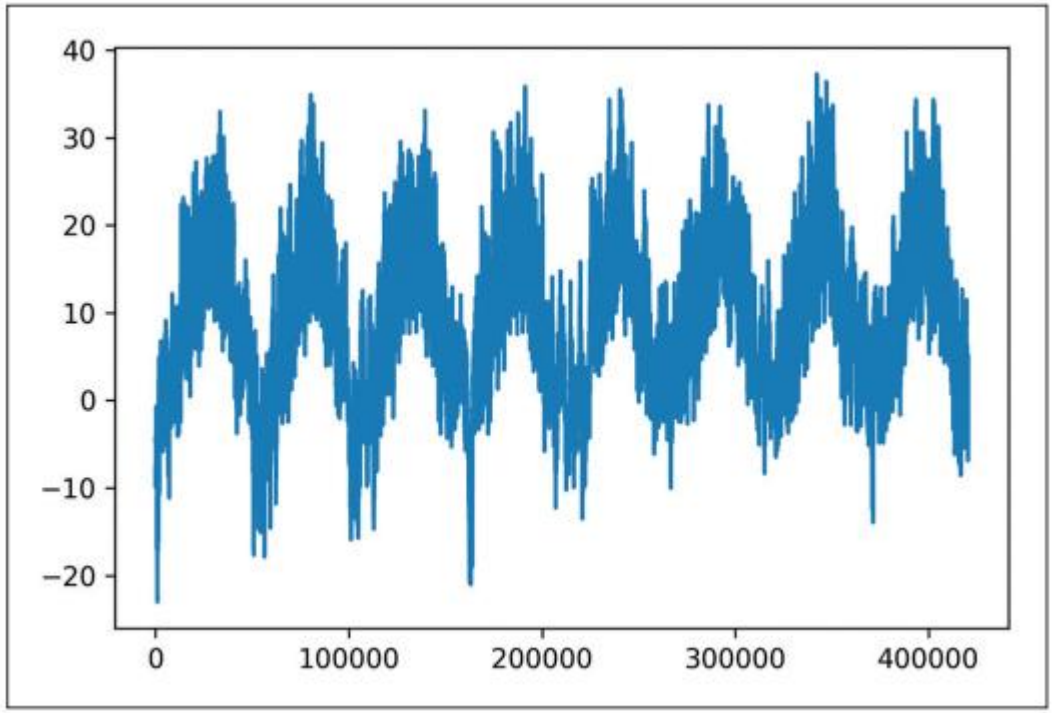
- 图 10-1 是一个显示**随时间变化的温度（摄氏度）**的图表。
- 从该图可以清楚看到，每年的温度变化具有周期性特征（本数据的时间跨度为 8 年）

코드 10-3 전체 온도를 그래프로 그리기

```
from matplotlib import pyplot as plt

plt.plot(range(len(temperature)), temperature)
plt.show()
```

10.2 温度预测问题

▼ 图 10-1 数据集中整个期间的温度（°C）



10.2 温度预测问题

温度预测问题

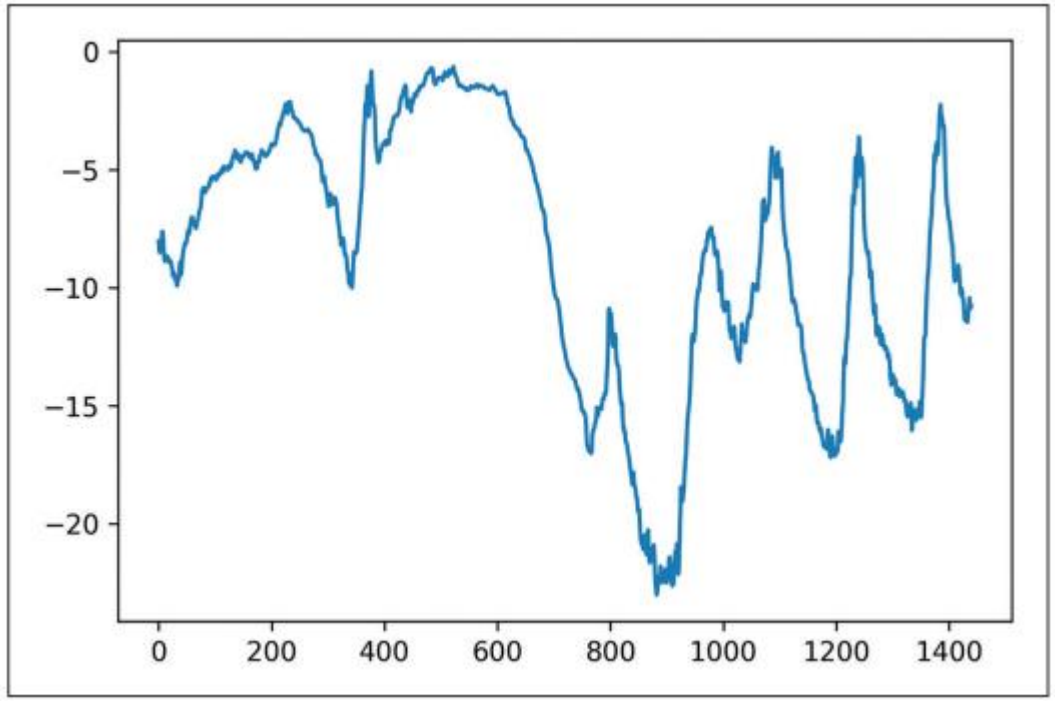
- **图 10-2** 缩短时间范围，展示了最初 10 天的温度数据图。
- 由于每 10 分钟记录一次数据，因此一天共有 **144 个数据点**。

코드 10-4 처음 10일간의 온도를 그래프로 그리기

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

10.2 温度预测问题

▼ 图 10-2 数据集的前 10 天温度（°C）



10.2 温度预测问题

温度预测问题

- 从该图中可以看到每日周期性；
- 尤其是最后4天，这种规律更加明显；
- 这10天的时间对应于非常寒冷的冬季。

10.2 温度预测问题

温度预测问题

- **务必在数据中寻找周期性。**
- 不同时间范围内的周期性是时序数据中重要且普遍的特征。
- 在天气、商场停车位、网站流量、超市销售量、健身追踪器（fitness tracker）中的步数等数据中，都可以看到日周期性和年周期性。（由人类生成的数据往往还具有每周周期性。）
- 因此，在探索数据时，要尝试识别此类周期性模式。

10.2 温度预测问题

温度预测问题

- 在这个数据集中，使用过去几个月的数据来预测下个月的平均气温是相对容易的问题，
- 因为年度周期性较为稳定。
- 但如果观察逐日数据，会发现气温变化非常不稳定。
- 那么，是否可以预测按日级别的时序数据呢？
- 让我们来直接验证一下。

10.2温度预测问题

温度预测问题

- 在所有示例中，前 50% 的数据用于训练，接下来的 25% 用于验证，最后 25% 用于测试。
- 在处理时间序列数据时，验证数据和测试数据必须比训练数据更新，
- 因为时间序列预测并不是从未来预测过去，而是基于过去来预测未来。
- 因此，必须按照这一原则划分验证集和测试集。
- 如果将时间轴反转，某些问题可能会变得非常容易解决，但那是不现实的！

코드 10-5 각 분할에 사용할 샘플 개수 계산하기

```
>>> 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)
num_train_samples: 210225
num_val_samples: 105112
num_test_samples: 105114
```

10.2 温度预测问题

数据准备

- 这道题的准确定义如下：给定以每小时为间隔采样的5天数据，能否预测24小时后的温度？
- 接下来我们先将数据预处理成可输入神经网络的形式。这个步骤很简单，因为数据已经是数值型，所以不需要额外的向量化。
- 数据中的各个时间序列量纲不同（例如：气压以 mbar 为单位，约为 1000；而 H_2O 浓度以 mmol/mol 为单位，约为 3），因此需要对每个时间序列分别归一化，使其具有相似范围的小数值。
- 在此示例中，将使用前 210,225 个时间步（timestep）作为训练数据，
- 并在该范围内计算均值和标准差用于标准化。

10.2 温度预测问题

数据准备

코드 10-6 데이터 정규화

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

10.2 温度预测问题

数据准备

- 现在我们来创建一个返回过去 5 天数据与 24 小时后目标温度批次的 Dataset 对象
- 该数据集中样本间有大量重复（样本 N 与样本 N+1 的大部分时间步都重叠）
- 若将所有样本加载进内存，内存开销会很大
- 因此，仅将 raw_data 和 temperature 数组保存在内存中，按需生成样本
- 虽可通过 Python 生成器实现，但 Keras 内置了数据集工具函数
- (timeseries_dataset_from_array())
- 使用该函数可减少工作量
- 该函数适用于各种时间序列预测任务

10.2 温度预测问题

数据准备

- timeseries_dataset_from_array() 理解
- 为了理解 timeseries_dataset_from_array() 函数的工作方式，我们先来看一个简单示例
- 当你将时间序列数据数组传入 data 参数时，
- timeseries_dataset_from_array() 函数会从原始时间序列中提取窗口（称为“序列 sequence”）
- 例如：若 data = [0 1 2 3 4 5 6] 且 sequence_length=3，
- 传入 timeseries_dataset_from_array() 函数后会生成以下样本：
- [0 1 2], [1 2 3], [2 3 4], [3 4 5], [4 5 6]
- 你可以通过 targets 参数向函数传入目标数组
- targets 数组的第一个元素应对应于从 data 数组生成的第一个序列的目标
- 若执行时间序列预测任务，targets 通常与 data 数组相同，但会有时间上的滞后差异

10.2 温度预测问题

数据准备

- 例如，当 data = [0 1 2 3 4 5 6 ...] 且 sequence_length=3 时，可将 targets = [3 4 5 6 ...] 传入，用于构建一个从该时间序列中预测下一个时间步的数据集
- 我们来测试一下

```
import numpy as np
    from tensorflow import keras

    int_sequence = np.arange(10)                创建一个从 0 到 9 的有序整数数组
    dummy_dataset = keras.utils.timeseries_dataset_from_array(
            data=int_sequence[:-3],             生成的序列将从 [0 1 2 3 4 5 6] 中抽样
            targets=int_sequence[3:],           以 data[N] 开始的序列，其目标值为 data[N + 3]
            sequence_length=3,                  序列的长度为 3 个时间步
            batch_size=2,                       序列的批量大小为 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]))
```

10.2 温度预测问题

数据准备

- 这个代码输出如下

[0, 1, 2] 3
[1, 2, 3] 4
[2, 3, 4] 5
[3, 4, 5] 6
[4, 5, 6] 7

10.2 温度预测问题

数据准备

- 使用 **timeseries_dataset_from_array()** 来创建三个数据集：训练集、验证集和测试集。
- 使用以下参数值
  - sampling_rate = 6
    - 每小时采样一个数据点
    - 即每 6 个数据点中只使用 1 个
  - sequence_length = 120
    - 使用前 5 天（120 小时）的数据
  - delay = sampling_rate * (sequence_length + 24 - 1)
    - 序列的目标是序列末尾 **24 小时后** 的温度

10.2 温度预测问题

数据准备

- 创建训练数据集时，只使用最初 50% 的数据：设置 start_index = 0，end_index = num_train_samples
- 创建验证数据集时，使用接下来的 25% 数据：设置 start_index = num_train_samples，end_index = num_train_samples + num_val_samples
- 创建测试数据集时，使用剩余的样本：设置 start_index = num_train_samples + num_val_samples

코드 10-7 훈련, 검증, 테스트 데이터셋 만들기

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

10.2 温度预测问题

数据准备

```
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)
```

10.2 温度预测问题

数据准备

```
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)
```

10.2 温度预测问题

数据准备

- 校正输入和输出的类型和张量维

```
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)
```

10.2 温度预测问题

数据准备

- 每个数据集返回一个形状为 (samples, targets) 的元组
- samples：由 256 个样本组成的批次
- 每个样本包含连续 120 小时的输入数据
- targets：包含 256 个目标温度的数组
- 由于样本是随机打乱的，因此批次中相邻的两个样本（例如 samples[0] 与 samples[1]）并不一定在时间上接近

코드 10-8 훈련 데이터셋의 배치 크기 확인하기

```
>>> for samples, targets in train_dataset:
>>>     print("샘플 크기:", samples.shape)
>>>     print("타깃 크기:", targets.shape)
>>>     break
샘플 크기: (256, 120, 14)
타깃 크기: (256,)
```

10.2 温度预测问题

常识水平的基准点

- 这用于确认问题是否为正常问题，如果是高水平的机器学习模型，则应当超越此基准
- 这种常识水平的解法在需要处理没有已知解决方案的新问题时非常有用
- 经典的例子是某个类别占绝大多数的不平衡分类任务
- 例如：若数据集中类别 A 的样本占 90%，类别 B 的样本占 10%，
- 那么对此分类问题的“常识水平”方法是：始终预测新样本属于类别 A
- 这种分类器的准确率大约是 90%
- 对于基于机器学习的模型而言，
- 若其性能无法超过 90%，则不能视为有用
- 有时，想要超越这种最基本的基准点是非常困难的

10.2 温度预测问题

常识水平的基准点

- 在这种情况下，时间序列数据具有连续性，并且可以假设具有按日的周期性（今天的温度很可能与明天的温度相似）
- 因此，常识性的解决方案是假设从现在起24小时后的温度与现在相同
- 我们将使用如下定义的平均绝对误差（MAE）来评估该方法

```
np.mean(np.abs(preds - targets))
```

10.2 温度预测问题

常识水平的基准点

- 下面代码10-9的评价

코드 10-9 상식 수준 모델의 MAE 계산하기

```
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"검증 MAE: {evaluate_naive_method(val_dataset):.2f}")
print(f"테스트 MAE: {evaluate_naive_method(test_dataset):.2f}")
```

온도 특성은 칼럼 인덱스 1에 있습니다. 따라서 samples[:, -1, 1]이 입력 시퀀스에 있는 마지막 온도 측정값입니다. 특성을 정규화했기 때문에 온도를 섭씨로 바꾸려면 표준 편차를 곱하고 평균을 더해야 합니다.[7]

```
def evaluate_naive_method(dataset):
    total_abs_err = 0.0      # 绝对误差总和
    samples_seen = 0         # 已计算的样本数

    for samples, targets in dataset:
        # 温度特征在列索引1处。
        # 因此 samples[:, -1, 1] 表示输入序列中最后一个时间步的温度测量值。
        # 由于数据已标准化，因此要将温度还原为原始单位，需要乘以标准差 std[1]，再加上均值 mean[1]。
        preds = samples[:, -1, 1] * std[1] + mean[1]

        # 计算预测值与目标值之间的绝对误差并累加
        total_abs_err += np.sum(np.abs(preds - targets))

        # 更新样本计数
        samples_seen += samples.shape[0]

    # 返回平均绝对误差 MAE
    return total_abs_err / samples_seen
```

10.2 温度预测问题

常识水平的基准点

- 常识水平的模型在验证集上取得了 2.44°C 的 MAE，在测试集上取得了 2.62°C 的 MAE
- 如果始终预测 “24小时后的温度与当前温度相同”，平均会产生约 2.5°C 的误差
- 虽然这个结果并不算太差，但你显然不会基于这样的简单规则来启动一个天气预报服务

10.2 温度预测问题

尝试建立基础的机器学习模型

- 在尝试机器学习模型之前，先设定好了“常识水平”的基准点
- 与其直接上手像 RNN 这样复杂且计算成本高的模型，不如先从简单且易于实现的机器学习模型开始（例如一个小规模的全连接网络）
- 以此为基础再引入更复杂的方法，不仅能提供改进的依据，也能获得更实际的收益

10.2 温度预测问题

尝试建立基础的机器学习模型

- 下面的代码 10-10 展示了一个全连接网络（Fully Connected Network），它将数据展开后传入两个 Dense 层
- 由于这是一个典型的回归问题，所以在最后一个 Dense 层中未使用激活函数
- 使用“均方误差（MSE）”作为损失函数，而不是 MAE
- 因为 MSE 在原点处可微分，非常适合用于梯度下降法
- 在 compile() 方法中，将 MAE 添加为监测指标

코드 10-10 밀집 연결 모델 훈련하고 평가하기

```
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)
```

10.2 温度预测问题

尝试建立基础的机器学习模型

```
callbacks = [
    keras.callbacks.ModelCheckpoint("jena_dense.keras",  ····· 콜백을 사용해서 최상의 모델을 저장합니다.
                                    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_dense.keras")
print(f"테스트 MAE: {model.evaluate(test_dataset)[1]:.2f}")  최상의 모델을 다시 로드하고 테스트 데이터에서 평가합니다.
```

```
callbacks = [
   keras.callbacks.ModelCheckpoint("jena_dense.keras",  # 使用回调保存最佳模型
                                   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_dense.keras") # 重新加载最佳模型并在测试集上评估
print(f"测试集 MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

10.2 温度预测问题

尝试建立基础的机器学习模型

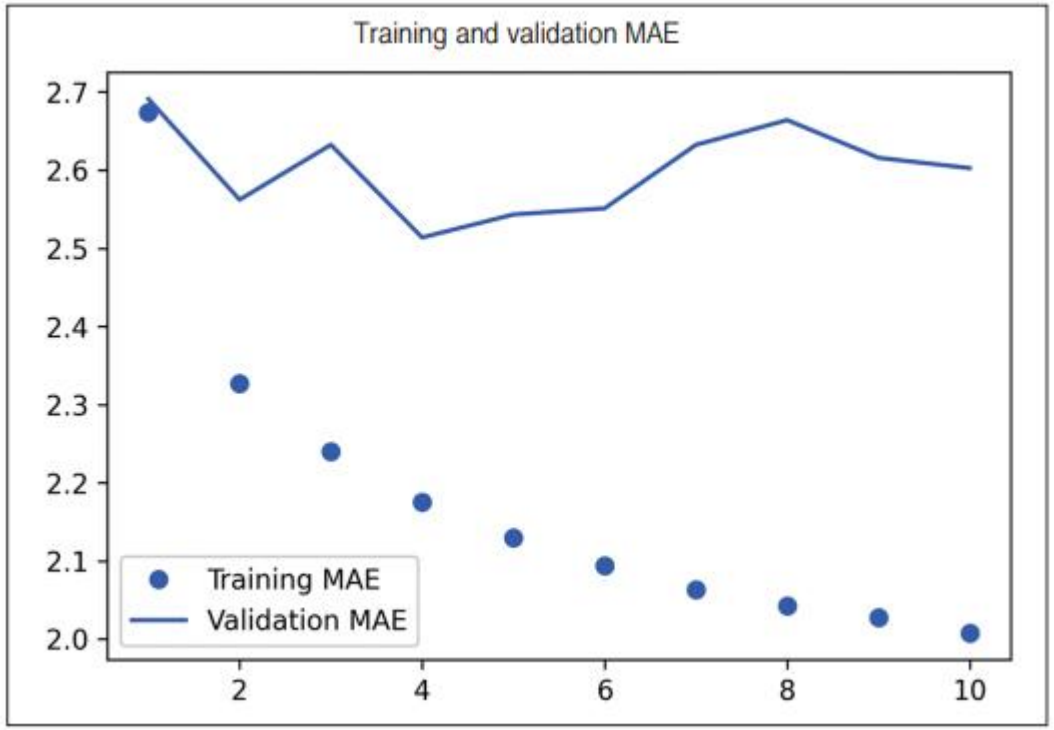
- 绘制训练与验证损失曲线（图 10-3）

코드 10-11 결과 그래프 그리기

```
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()
```

10.2 温度预测问题

▼ 图 10-3 在耶拿温度预测任务中，简单全连接网络的训练与验证曲线MAE



10.2 温度预测问题

尝试建立基础的机器学习模型

- 部分验证损失接近未使用学习的基准点，但不够稳定
- 先建立基准模型是有帮助的
- 这个问题并不容易超越基准模型的性能
- 我们采用的常识中包含了机器学习模型无法捕捉到的大量关键信息

10.2 温度预测问题

尝试基本的机器学习模型

- 如果一个“简单且表现良好的模型（如常识性基准模型）”可以映射输入与输出，那么为什么经过训练的模型却找不到这种映射、表现反而更差？
- 为了解决问题我们探索的模型空间（即假设空间），是由我们设置的网络结构（例如两层神经网络）的所有可能参数组合构成的。
- 常识性模型只是这个假设空间中能被表示的数百万种组合中的一种。
- 这就像在沙滩上找针。
- 从技术上说，假设空间中存在好的解，并不意味着梯度下降法一定能找到它。
- 这是机器学习中普遍存在的一个重大局限性。
- 如果算法没有被硬编码为倾向于寻找某种类型的简单模型，那么即使是简单问题，也可能无法找到简洁的解决方案。
- 因此，良好的特征工程以及与问题相关的模型结构设计非常重要。
- 换句话说，必须明确告诉模型该去寻找什么样的解。

10.2 温度预测问题

尝试 1D 卷积模型

- 从正确利用架构结构的角度来看，由于输入序列具有按日周期性，可以应用卷积模型。
- 对时间轴进行卷积，可以重用不同日期中相同的特征表示。
- 这就像空间方向的卷积在图像中重用不同位置上的相同特征一样。
- 我们已经学习过 Conv2D 和 SeparableConv2D 层。
- 这些层通过小窗口在二维网格上滑动，观察输入数据。
- 此外，还有 1D 甚至 3D 卷积。Conv1D、SeparableConv1D、Conv3D
- Conv1D 层：使用一维窗口在输入序列上进行滑动。
- Conv3D 层：使用立方体窗口在输入体积（Volume）上进行滑动。

10.2 温度预测问题

尝试 1D 卷积模型

- 可以构建与 2D 卷积网络非常相似的 1D 卷积网络
- 对于遵循平移不变性假设的任意序列数据都适用（即，当在序列上滑动窗口时，窗口内的内容无论位置如何都具有相同的性质）
- 让我们将其应用于温度预测问题
- 初始窗口长度设为 24
- 每次（一个周期）观察24 小时的数据
- 由于通过 MaxPooling1D 层 对序列进行下采样，因此将相应地缩小窗口大小

```
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)
```

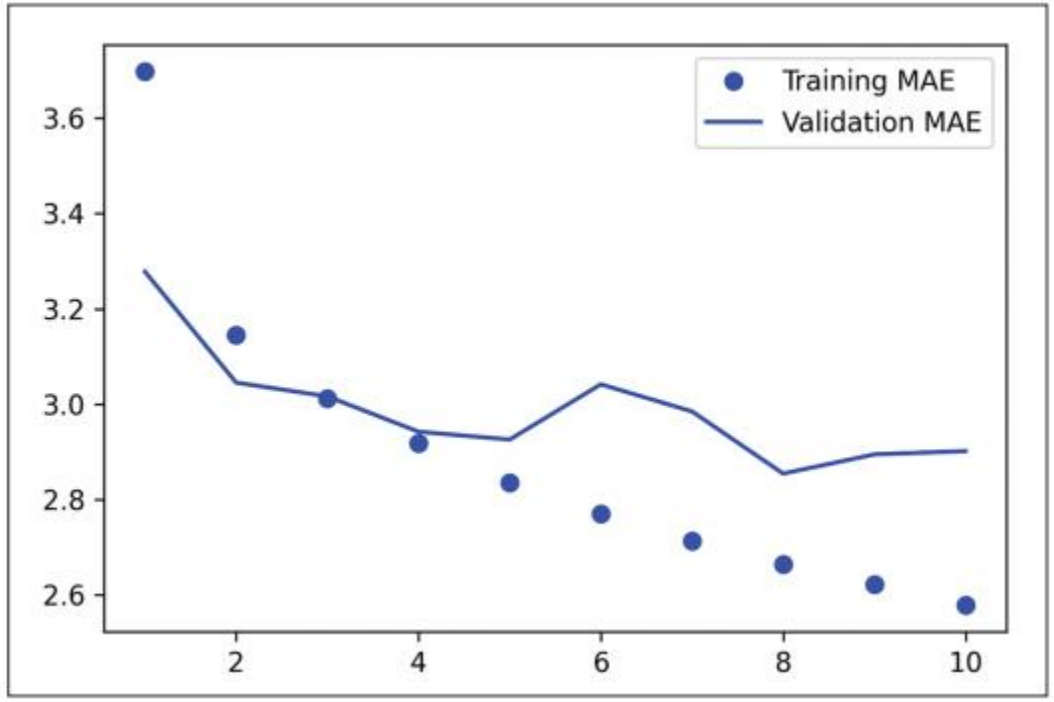
10.2 温度预测问题

尝试 1D 卷积模型

```
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}")
```

10.2 温度预测问题

▼ 图10-4 在耶拿温度预测任务中应用的 1D 卷积网络 的训练与验证 MAE



10.2 温度预测问题

尝试 1D 卷积模型

- 从结果看，这个模型的表现比全连接模型更差。
- 验证集 MAE 约 2.9°C，与常识基线模型差距很大。
- 问题出在哪？
- 有两点：
  - **天气数据并不太满足平移不变性假设**
    - 虽然存在日周期性，但早晨的数据与傍晚/深夜的数据性质不同。
    - 天气数据只在非常特定的时间范围内才近似平移不变。
  - **该数据对时间顺序非常敏感**
    - 最近的数据比5天前的数据对预测明天温度更有用。
    - 1D 卷积网络无法很好地利用这一事实，
    - 尤其由于最大池化和全局平均池化层，顺序信息被大量丢失

10.2温度预测问题

第一个循环神经网络（RNN）

- 虽然全连接模型和卷积模型的效果都不理想，但这并不意味着该问题不适合使用机器学习。
- 全连接模型由于展开了时间序列数据，因此在输入数据中丢失了时间的概念。
- 卷积模型则是以相似的方式处理数据的所有部分，并通过池化（Pooling） 操作而丢失了顺序信息。
- 因此，我们将尝试使用具有因果关系和顺序意义的序列数据，并保持其原有结构来进行建模。

10.2 温度预测问题

第一个循环神经网络（RNN）

- 为了解决这类问题，专门设计了“循环神经网络（RNN）”结构。
- 其中，“LSTM（Long Short-Term Memory，长短期记忆网络）”层长期以来一直非常受欢迎。
- 接下来，我们将先尝试应用LSTM层来解决问题。

코드 10-12 간단한 LSTM 기반 모델

```
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)
]
```

10.2 温度预测问题

第一个循环神经网络（RNN）

```
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"테스트 MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

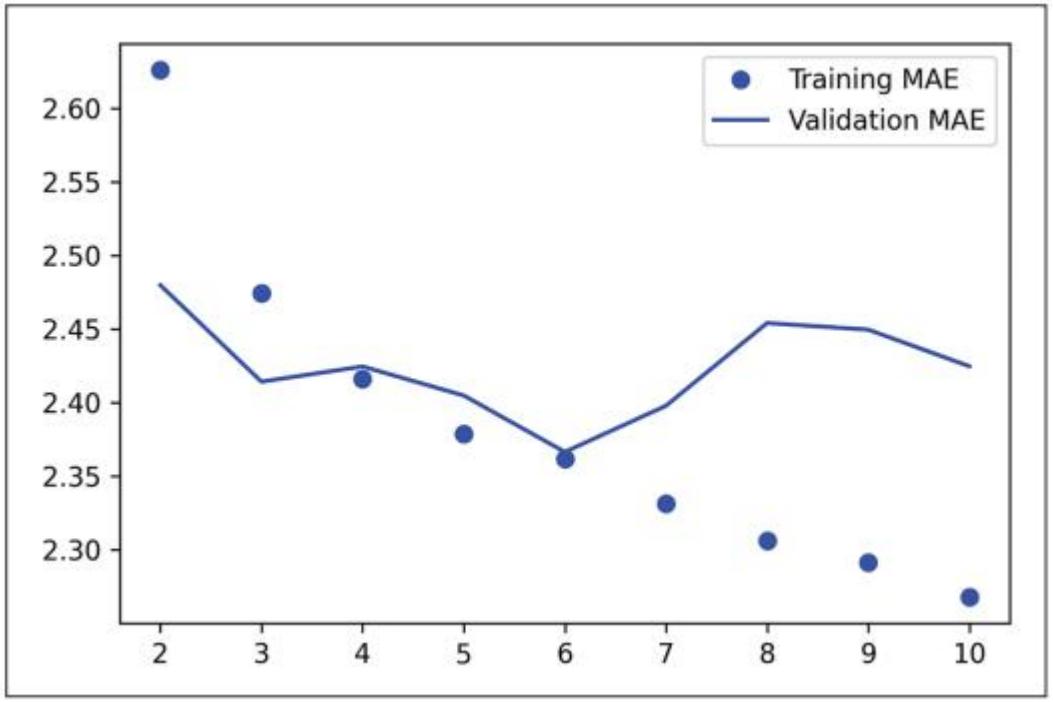
10.2 温度预测问题

第一个循环神经网络（RNN）

- 最低验证 MAE 为 2.36°C，测试 MAE 为 2.55°C。
- 基于 LSTM 的模型终于略微超越了常识水平模型（虽然差距仍然很小），展示了机器学习在该任务中的价值。

10.2 温度预测问题

▼ 在“耶拿温度预测任务”中应用 LSTM 基础模型的训练与验证 MAE由于第 1 个 epoch 的训练 MAE（7.75）过高，会压缩图表比例，因此在图中被排除。



10.3理解循环神经网络（RNN）

10.3 理解循环神经网络（RNN）

理解循环神经网络（RNN)

- 像全连接网络或卷积网络这类我们目前见到的所有神经网络的共同特征是：没有记忆（memory）
- 输入到网络中的数据是分别独立处理的，输入之间不会保留状态信息
- 因此，如果要用这种网络处理序列或时间序列数据点，必须将整个序列一次性输入网络
- 也就是说，必须将整个序列转换为一个数据点
- 例如，在全连接模型中，我们将 5 天的数据展开为一个大的向量再进行处理
- 这样的网络称为前馈网络（Feedforward Network）

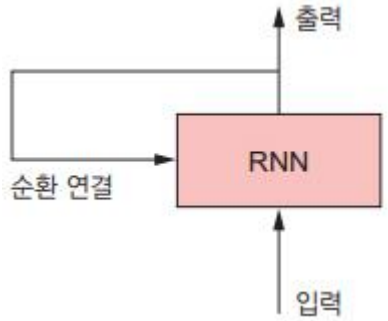
10.3理解循环神经网络（RNN）

理解循环神经网络（RNN)

- 相反，人类在阅读句子时，会记住之前出现的内容，并逐词或按一眼能看到的范围进行处理
- 这有助于自然地表达句子中的含义
- 生物智能会维持一个用于信息处理的内部模型，逐步（递进地）处理信息
- 这种模型利用过去的信息进行构建，并在获取新信息时持续更新
- 虽然是极度简化的版本，但**循环神经网络（RNN，Recurrent Neural Network）**基于相同的原理
- RNN 会在遍历序列元素的同时，将已处理的信息保存在状态（state）中
- 实际上，RNN 是一种内部包含循环（loop）的神经网络（见图 10-6)

10.3理解循环神经网络（RNN）

▼ 그림 10-6 循环神经网络：带有循环的网络



10.3理解循环神经网络（RNN）

理解循环神经网络（RNN)

- RNN 的状态在处理不同序列（例如批次中包含的两个不同样本）之间会被 重置
- 一个序列依然可以被视为一个数据点，即：输入到网络中的一个整体输入
- 不同之处在于：这个数据点 并不会一次性处理完
- 相反，网络会在内部 依次遍历序列中的各个元素

10.3 순환 신경망 이해하기

理解循环神经网络（RNN)

- 为了更清晰地理解 循环 (loop) 和 状态 (state) 的概念，我们实现一个简单的 RNN 前向计算过程。
- 这个 RNN 接收一个形状为 (timesteps, input_features) 的 rank-2 张量（即一个由多个向量组成的序列）作为输入。
- 模型会沿时间步（timestep）进行循环，在每个时间步 t：- 将当前状态（维度为 (state_size,)）与当前输入（维度为 (input_features,)）拼接　- 计算当前时间步的输出（output）。
- 然后，将该输出用作下一时间步的状态。在第一个时间步，由于没有先前的输出，因此没有“当前状态”。此时，RNN 使用网络的初始状态 (initial state) —— 通常是全 0 向量 来初始化状态。

10.3 순환 신경망 이해하기

理解循环神经网络（RNN)

- 如果用伪代码 (pseudocode) 来表示 RNN，其结构如下：

코드 10-13 의사 코드로 표현한 RNN

```
state_t = 0 -------- 타임스텝 t의 상태입니다.
for input_t in input_sequence: -------- 시퀀스의 원소를 반복합니다.
    output_t = f(input_t, state_t)
    state_t = output_t -------- 출력은 다음 반복을 위한 상태가 됩니다.
```

```
state_t = 0                  # 时间步 t 的状态
for input_t in input_sequence:  # 逐个遍历序列中的元素
    output_t = f(input_t, state_t)  # 根据当前输入与状态计算输出
    state_t = output_t          # 当前输出作为下一步的状态
```

10.3 순환 신경망 이해하기

理解循环神经网络（RNN）

- f 函数 将输入与状态转换为输出
- 可以将其改写为使用两个矩阵 W 与 U 以及偏置向量的变换形式
- 这种变换与前馈网络（Feedforward Network）中全连接层执行的变换相似

코드 10-14 좀 더 자세한 의사 코드로 표현한 RNN

```
state_t = 0
for input_t in input_sequence:
    output_t = activation(dot(W, input_t) + dot(U, state_t) + b)
    state_t = output_t
```

10.3 순환 신경망 이해하기

理解循环神经网络（RNN）

- 要完全理解这一过程，我们来用 **NumPy** 实现一个简单 **RNN** 的前向计算。

```
# 代码 10-15 使用 NumPy实现的简单 RNN

import numpy as np

# ----------------------------------
# 1. 定义超参数
# ----------------------------------
timesteps = 100       # 输入序列中的时间步数（time steps）
input_features = 32    # 每个输入的特征维度（input features）
output_features = 64   # 输出特征维度（output features）

# ----------------------------------
# 2. 生成输入与初始状态
# ----------------------------------
inputs = np.random.random((timesteps, input_features))
# 输入数据：一个随机序列，形状为 (100, 32)

state_t = np.zeros((output_features,))
# 初始状态：长度为64的零向量

# ----------------------------------
# 3. 定义权重参数
# ----------------------------------
W = np.random.random((output_features, input_features))
# 输入到输出的权重矩阵(64, 32)

U = np.random.random((output_features, output_features))
# 前一状态到当前状态的权重矩阵(64, 64)

b = np.random.random((output_features,))
# 偏置向量 (64,)

# ----------------------------------
# 4. 执行循环：逐时间步计算输出
# ----------------------------------
successive_outputs = [] # 用于保存每个时间步的输出

for input_t in inputs:
  # input_t: 当前时间步的输入，形状为 (32,)

  # 当前输出 = tanh(W·input_t + U·state_t + b)
  output_t = np.tanh(np.dot(W, input_t) + np.dot(U, state_t) + b)

  # 将输出保存到列表中
  successive_outputs.append(output_t)

  # 更新当前状态为output_t（RNN 特性：当前状态 = 当前输出）
  state_t = output_t

# ----------------------------------
# 5. 堆叠所有时间步的输出
# ----------------------------------
final_output_sequence = np.stack(successive_outputs, axis=0)

# 最终输出：形状为(timesteps, output_features)，即 (100, 64)
print("Final output sequence shape:", final_output_sequence.shape)
print("Example output at last timestep:", final_output_sequence[-1])
```

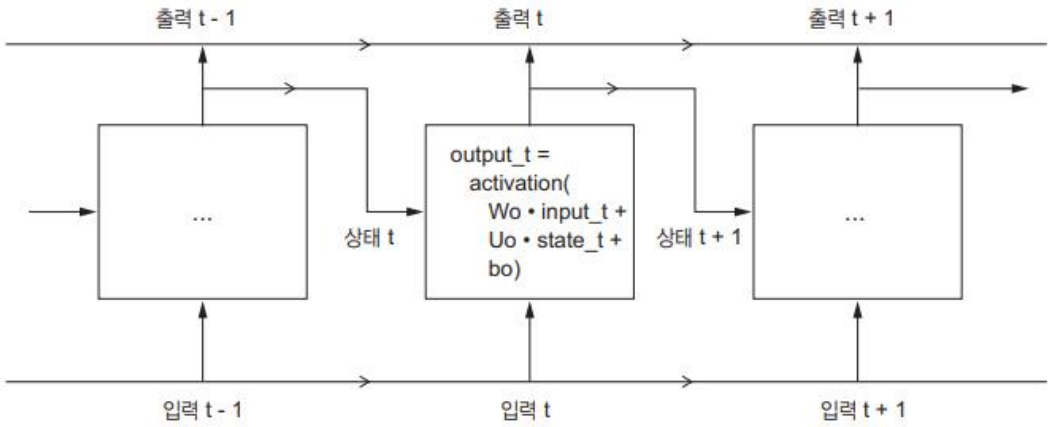
10.3 理解循环神经网络（RNN）

理解循环神经网络（RNN）

- 总结来说，RNN 在循环时，只是一个 for 循环，会重复使用之前计算得到的信息。
- 当然，符合此定义的 RNN 类型有很多种。
- 本例展示的是 最简单的 RNN 形式。
- RNN 是由一个 step（步骤）函数 来定义的。
- 在本例中，其形式如下（见图 10-7）。

```
output_t = np.tanh(np.dot(W, input_t) + np.dot(U, state_t) + b)
```

▼ 图 10-7 随时间展开的简单 RNN



10.3理解循环神经网络（RNN）

理解循环神经网络（RNN）

- 在这个例子中，最终输出是形状为 (timesteps, output_features) 的二阶张量。
- 每个时间步（time step）表示时间 t 时刻的输出。
- 输出张量中每个时间步 t 的输出，都包含了输入序列中从时间步 0 到 t 的所有历史信息。
- 因此，在许多情况下并不需要整个输出序列。
- 因为最后一个输出（循环中的最后一次 output_t）已经包含了整个序列的信息，所以只需要最后一个输出即可。

10.3理解循环神经网络（RNN）

Keras 的循环层

- 用 Numpy 简单实现的过程，实际上对应于 Keras 的 SimpleRNN 层。
- SimpleRNN 的一个不同点在于，它不像 Numpy 示例中那样只处理一个序列，而是像 Keras 的其他层一样，能够处理序列的 批次（ batch ）。
- 即，它接收的输入形状不是 (timesteps, input_features)，而是 (batch_size, timesteps, input_features)。
- 在开始时，可以在 Input() 函数的 shape 参数中，将 timesteps 项设为 None。
- 这样就可以处理长度可变的序列。

코드 10-16 어떤 길이의 시퀀스도 처리할 수 있는 RNN 층

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

10.3理解循环神经网络（RNN）

Keras 的循环层

- 当模型需要处理 可变长度序列 时，尤其有用。
- 如果所有序列长度都相同，最好指定一个固定的输入长度。
- model.summary() 可以提供输出长度的信息（这些信息始终有用），并可用于部分性能优化。

10.3理解循环神经网络（RNN）

Keras 的循环层

- Keras 中的所有循环层（ SimpleRNN, LSTM, GRU ）都可以以两种模式运行。
- 可以返回：- 每个时间步的输出所组成的整个序列（ 形状为 (batch_size, timesteps, output_features) 的三维张量），- 或者仅返回输入序列的最后一个输出（ 形状为 (batch_size, output_features) 的二维张量）。
- 这两种模式可以通过构造函数中的参数 return_sequences 来控制。
- 下面我们以 SimpleRNN 为例，来看一个仅返回最后时间步输出的示例

코드 10-17 마지막 출력 스텝만 반환하는 RNN 층

```
>>> num_features = 14
>>> steps = 120
>>> inputs = keras.Input(shape=(steps, num_features))
>>> outputs = layers.SimpleRNN(16, return_sequences=False)(inputs)
>>> print(outputs.shape)
(None, 16)
```

return_sequences=False가 기본값입니다.

10.3理解循环神经网络（RNN）

Keras 的循环层

- 以下示例返回整个状态序列

코드 10-18 전체 출력 시퀀스를 반환하는 RNN 층

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

10.3理解循环神经网络（RNN）

Keras 的循环层

- 为了增强网络的表达能力，有时堆叠多个循环层是有用的。
- 在这种设置下，中间层应设置为返回整个输出序列。

코드 10-19 스태킹(stacking) RNN 층

```
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)
```

10.3理解循环神经网络（RNN）

Keras 的循环层

- 实际上几乎不会使用 SimpleRNN 层
- 一般来说，SimpleRNN 过于简单，不适合实际应用
- 理论上，SimpleRNN 可以在时间 t 保留所有先前时间步的信息
- 但在实际中，无法学习长时间跨度的依赖关系
- 这是因为出现了类似于层数较多的普通网络（前馈网络）中所见的梯度消失（vanishing gradient）问题
- 也就是说，就像在前馈网络中增加层数会使训练变得困难一样
- 1990 年代初，Hochreiter、Schmidhuber 和 Bengio 等人对这种现象的理论原因进行了研究

10.3 理解循环神经网络（RNN）

Keras 的循环层

- 幸运的是，SimpleRNN 并不是 Keras 中唯一的循环层
- 为了解决这一问题，提出了改进的 LSTM 和 GRU 层
- 我们来看看 LSTM 层
- 长短期记忆（Long Short-Term Memory, LSTM）算法由 Hochreiter 和 Schmidhuber 于 1997 年开发
- 该算法是对梯度消失问题研究的结晶

10.3 理解循环神经网络（RNN）

Keras 的循环层

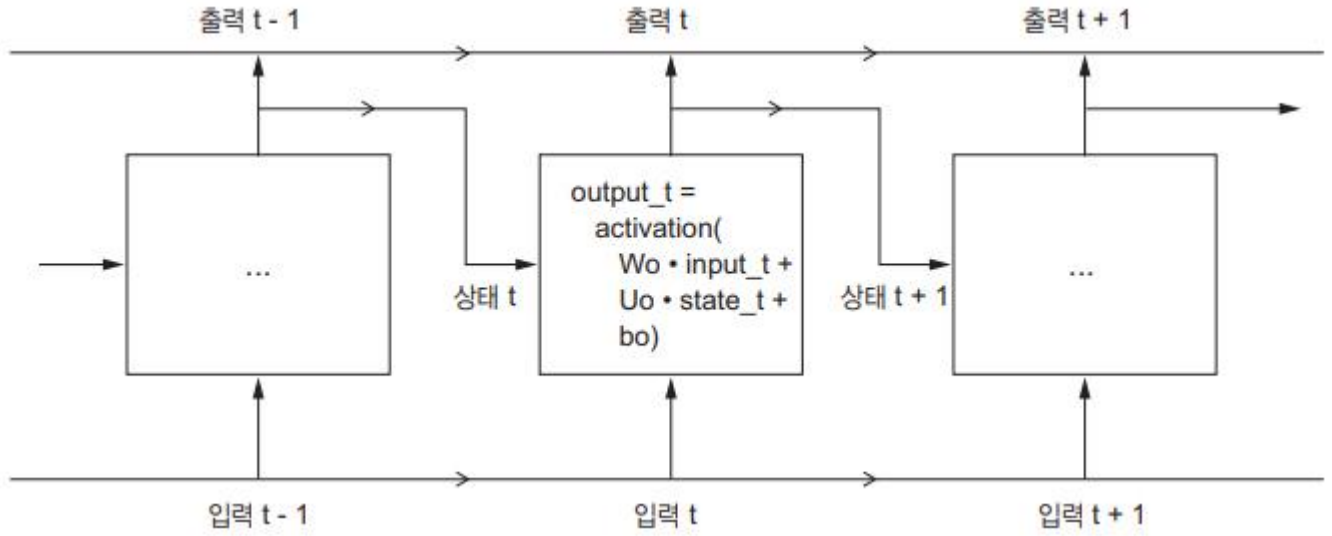
- 这一层是之前看到的 SimpleRNN 的一种变体，增加了跨越多个时间步传递信息的方法。
- 可以想象一个与要处理的序列并行运行的传送带。
- 在序列的某个时刻提取的信息被放到传送带上，移动到需要该信息的时间步后取下使用。
- 这就是 LSTM 的工作原理。
- 通过存储信息以备后用，可以防止在处理过程中旧信号逐渐消失。
- 这让人联想到第 9 章学到的残差连接（Residual Connection）。
- 这两个想法非常相似。

10.3 理解循环神经网络（RNN）

Keras 的循环层

- 为了更详细地理解这一点，我们先从 SimpleRNN 的单元（cell） 画起（图 10-8）。
- 由于会出现多组权重矩阵，我们用表示**输出（output）**的字母 o 来标记该单元中的 W 和 U 矩阵，即写作 W_o 和 U_o。

▼ 图 10-8：LSTM 层的起点——SimpleRNN 单元结构



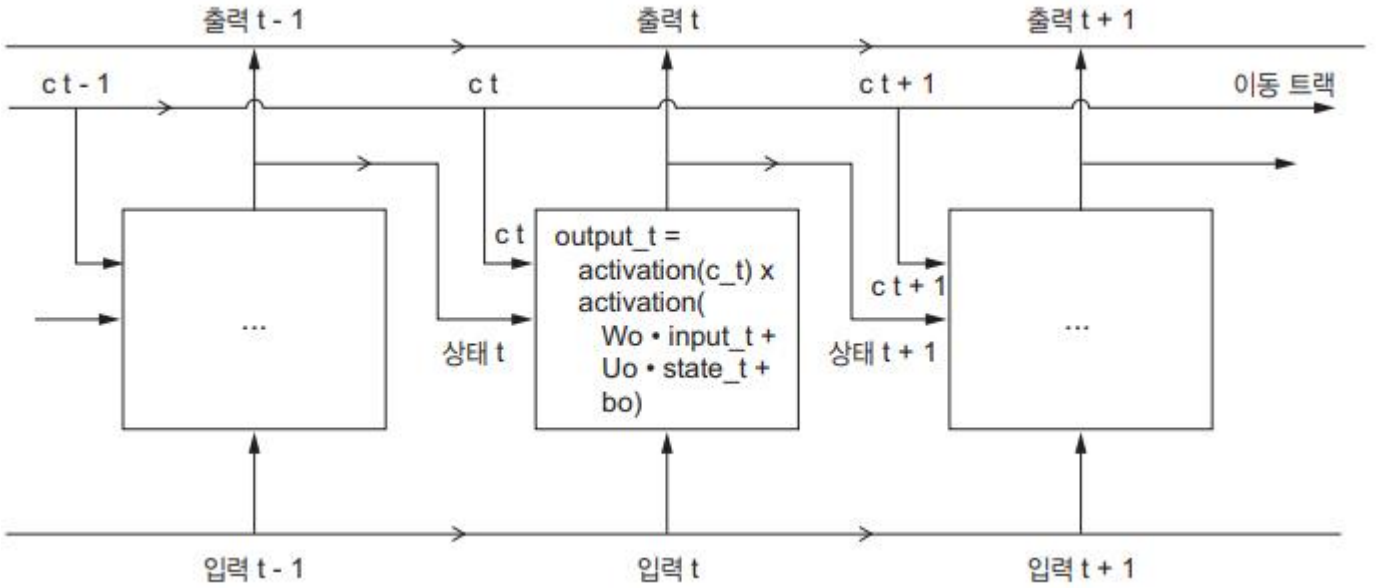
10.3 理解循环神经网络（RNN）

Keras 的循环层

- 让我们在图 10-8 中添加一个跨越时间步传递信息的数据流。
- 在时间步 n，我们将此值称为传递状态 ctc_tct。
- 其中，c 表示“传递（carry）”。
- 利用这一信息，单元结构会发生如下变化：
- 传递状态与输入连接和循环连接（即状态）相连（通过与权重矩阵做点积、加上偏置并施加激活函数的全连接层变换实现）。
- 随后，传递状态会影响传递到下一时间步的状态（通过激活函数和乘法操作实现）。
- 从概念上看，这个负责“传递数据”的流，会调节下一个输出与状态（见图 10-9）。
- 到目前为止，内容都比较简单。

10.3 理解循环神经网络（RNN）

▼ 图 10-9 SimpleRNN 到 LSTM 的转变：增加“传递路径”



10.3 理解循环神经网络（RNN）

Keras 的循环层

- 现在复杂的部分是数据流中**下一个传递状态 (c_{t+1})** 的计算方式
- 这里涉及到 3 种不同的变换，这三种变换都与 SimpleRNN 形式相同。
- 每个变换都有其**独立的权重矩阵**，分别用 **i, f, k** 表示。

```
y = activation(dot(state_t, U) + dot(input_t, W) + b)
```

- 通过结合 $\mathbf{i_t}$, $\mathbf{f_t}$, $\mathbf{k_t}$ 来计算新的传递状态 **c_{t+1}**

코드 10-20 LSTM 구조의 의사 코드(1/2)[17]

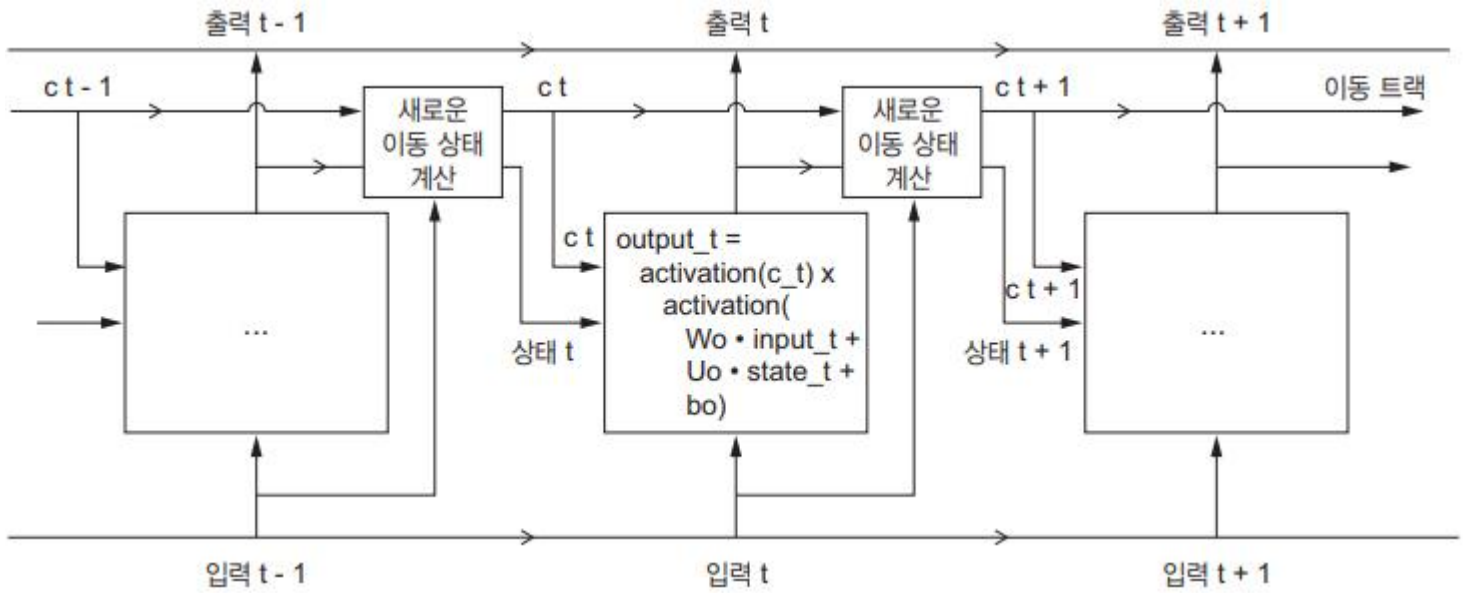
```
output_t = activation(c_t) * activation(dot(input_t, Wo) + dot(state_t, Uo) + bo)
i_t = activation(dot(state_t, Ui) + dot(input_t, Wi) + bi)
f_t = activation(dot(state_t, Uf) + dot(input_t, Wf) + bf)
k_t = activation(dot(state_t, Uk) + dot(input_t, Wk) + bk)
```

코드 10-21 LSTM 구조의 의사 코드(2/2)

```
c_t+1 = i_t * k_t + c_t * f_t
```

10.3 理解循环神经网络（RNN）

▼ 图 10-10 LSTM 构成



该图展示了 **LSTM 在每个时间步 (t-1, t, t+1)** 如何：
1.接收输入（입력）与前一状态（상태 t-1）；
2.计算新的携带状态（새로운 이동 상태 계산）；
3.结合携带状态 c_t 与当前状态 $state_t$ 计算输出 $output_t$；
4.将信息传递到下一个时间步。

10.3 理解循环神经网络（RNN）

Keras 的循环层

- 通过分析这些运算的作用，可以对各个符号的含义获得一定的洞察。
- 例如，ctc_tct 与 ftf_tft 的乘积可以理解为在传递数据流中有意删除与当前移动无关的信息。
- 另一方面，iti_tit 和 ktk_tkt 则提供当前的相关信息，并用新信息更新传递轨道。
- 不过，总体而言，这样的解释并没有太大的实际意义。

10.3 理解循环神经网络（RNN）

Keras 的循环层

- 这些运算实际上执行的操作，取决于与运算相关的权重矩阵。
- 这些权重通过端到端（End-to-End）方式进行学习。
- 在每次训练迭代中，这一过程都会重新开始，因此无法为这些运算赋予特定目的。
- RNN 单元的结构决定了假设空间。
- 在训练过程中，模型会在该空间中找到较优的参数。
- 单元的结构并不决定单元的功能，
- 它取决于单元的权重。
- 即使是相同结构的单元，若权重不同，也会执行完全不同的任务。
- 因此，RNN 单元中的运算组合，应被理解为假设空间的约束条件，而非工程化设计

10.3 理解循环神经网络（RNN）

Keras 的循环层

- 的确，将 RNN 单元的实现方式 等约束条件的选择，交由优化算法（如遗传算法、强化学习算法等）而非人工工程师，可能是更好的选择。
- 未来的网络结构可能也会以这样的方式构建。
- 总结来说，完全没必要去理解 LSTM 单元的具体结构。
- 这也不是我们需要做的事情。
- 只需记住 LSTM 单元的作用即可。
- 即：通过将过去的信息重新注入未来，来解决梯度消失问题。

10.4 循环神经网络的高级用法

10.4 循环神经网络的高级用法

循环神经网络的高级用法

- 到目前为止，我们学习了以下内容：
  - 什么是 RNN 以及 RNN 的工作方式
  - 什么是 LSTM，以及它为什么比简单的 RNN 更擅长处理长序列
  - 如何使用 Keras 的 RNN 层 来处理序列数据

10.4 循环神经网络的高级用法

循环神经网络的高级用法

- 接下来我们将学习 **RNN 的多种高级功能**。
- 这些内容将有助于你**最大限度地利用深度学习的序列模型**
- 在本节结束后，你将掌握在 **Keras 中使用循环神经网络（RNN）** 所需的大部分核心知识。
- 我们将讨论以下主题
  - 循环 Dropout（ Recurrent Dropout ）
    - 一种特殊的 Dropout，用于循环层中防止过拟合
  - 堆叠循环层（ Stacking Recurrent Layer ）
    - 提高模型的**表达能力（ representational power ）**，但计算成本也会增加。
  - **双向循环层（ Bidirectional Recurrent Layer ）**
    - 让相同信息从不同方向输入循环网络，从而**提高准确度并延长记忆保持时间**。

10.4 循环神经网络的高级用法

为减少过拟合，使用**循环 Dropout（Recurrent Dropout）**

- 我们将重新使用在 10.2.5 节 中超过常规基准的 LSTM 模型。
- 通过观察训练损失与验证损失曲线，可以判断模型是否出现过拟合。
- 经过几轮 epoch 后，若训练损失和验证损失之间出现显著差距，说明过拟合开始发生。
- 为解决这种现象，我们已学习过常规 Dropout 技术。
- Dropout 的原理是：在训练时随机关闭输入层中的部分单元（units），打破数据中偶然的相关性。
- 然而，在“循环神经网络（RNN）”中正确地应用 Dropout 并不简单。

10.4 循环神经网络的高级用法

为减少过拟合，使用**循环 Dropout（Recurrent Dropout）**

- 早期研究表明：在循环层之前应用 Dropout 并不能有效起到正则化作用，反而会妨碍模型学习。
- 直到 2016 年，Yarin Gal 在其关于 贝叶斯深度学习（Bayesian Deep Learning） 的博士论文中，提出了在循环神经网络中正确使用 Dropout 的方法。
- 关键在于：不应在每个 time step（时间步） 随机改变 Dropout 掩码；而应使用固定的 Dropout 掩码（相同模式的单元置零），并将其应用于所有时间步。
- 对于 GRU 和 LSTM 等门控循环单元，要对循环层内部计算中的激活函数应用同样的固定 Dropout 掩码（称为 Recurrent Dropout Mask）。
- 在所有时间步使用相同的 Dropout 掩码，网络才能让误差信号在时间维度上正确传播。
- 若每个时间步使用不同的随机掩码，则会干扰误差信号的传播，从而破坏训练过程。

10.4 循环神经网络的高级用法

为减少过拟合，使用**循环 Dropout（Recurrent Dropout）**

- Yarin Gal 使用 Keras 进行研究，并帮助在 Keras 的循环层中实现了这一功能。
- Keras 中的所有循环层都有 两个 dropout 参数。
- dropout：用于指定层输入的 dropout 比例，是一个 浮点数值。
- recurrent_dropout：用于指定循环状态的 dropout 比例。
- 接下来将对第一个 LSTM 示例中的 LSTM 层应用 循环 dropout，观察它对过拟合的影响。

10.4 循环神经网络的高级用法

为减少过拟合，使用**循环 Dropout（Recurrent Dropout）**

- 由于使用了 Dropout（随机失活），无需再为正则化而刻意调整网络规模。
- 使用具有 两倍数量单元的 LSTM 层。
- （如果不加正则化，这个网络一开始就会发生过拟合——可以直接测试看看）其表达能力可能会更强。
- 采用 Dropout 正则化的网络通常需要更长时间才能完全收敛。
- 因此，将 训练轮次（epoch）增加两倍 来训练网络。

10.4 循环神经网络的高级用法

为减少过拟合，使用**循环 Dropout（Recurrent Dropout）**

코드 10-22 드롭아웃 규제를 적용한 LSTM 모델 훈련하고 평가하기

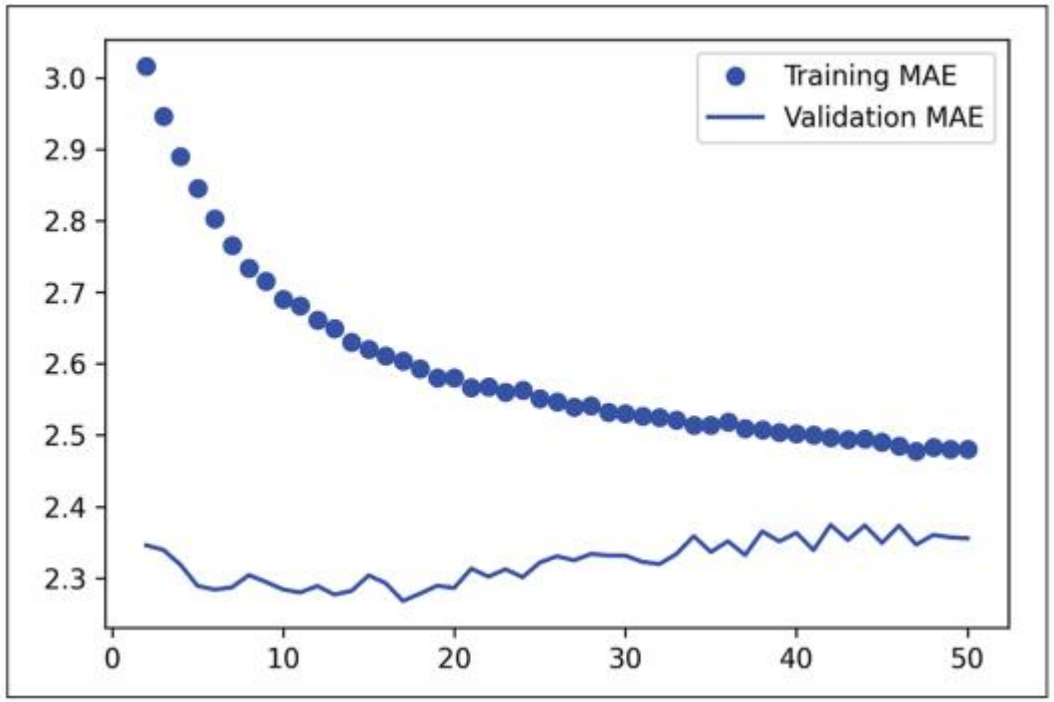
```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(32, recurrent_dropout=0.25)(inputs)
x = layers.Dropout(0.5)(x) -------- Dense 층에 규제를 추가하기 위해 LSTM 층 뒤에도 Dropout 층을 추가합니다.
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

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)
```

10.4 循环神经网络的高级用法

为减少过拟合，使用**循环 Dropout（Recurrent Dropout）**

- 在第 20 个 epoch 之前都没有出现过拟合。
- 达到了 验证集 MAE = 2.27°、测试集 MAE = 2.45° 的最低误差。

▼ 图 10-11 使用 Dropout 正则化的 LSTM 模型在“耶拿温度预测任务”中的训练与验证损失



10.4 循环神经网络的高级用法

为减少过拟合，使用**循环 Dropout（Recurrent Dropout）**

- RNN 运行时性能（runtime performance）
- 像本章中的这种参数数量非常少的循环神经网络，相比 GPU，往往在多核 CPU 上运行更快。
- 这是因为它只包含小规模的矩阵乘法，并且由于存在 for 循环，连续的乘法难以很好地并行化。
- 对于大规模的 RNN，GPU 则可能会提供更大的帮助。

10.4 循环神经网络的高级用法

为减少过拟合，使用**循环 Dropout（Recurrent Dropout）**

- 使用默认参数设置的 Keras LSTM 和 GRU 层在 GPU 上运行时，可以利用 cuDNN 内核。
- cuDNN 是由 NVIDIA 提供的高度优化的底层算法实现。
- 一如既往，cuDNN 内核速度非常快，但灵活性较差。
- 如果尝试执行默认内核不支持的操作，会明显感受到速度下降。
- 因此，使用 NVIDIA 提供的功能时，通常需要遵守一定的限制。
- 例如，LSTM 和 GRU cuDNN 内核不支持循环丢弃（recurrent dropout）。
- 如果在层中添加循环丢弃（虽然计算开销相同），通常会退回到普通的 TensorFlow 实现，其速度比 GPU 模式慢约 2~5 倍。

10.4 循环神经网络的高级用法

为减少过拟合，使用**循环 Dropout（Recurrent Dropout）**

- 无法使用 cuDNN 时，可以通过展开 (unrolling) RNN 层来提高运行速度。
- 展开 for 循环的意思是去掉循环结构，直接将循环内容重复 N 次。
- 对于 RNN 的 for 循环，展开后可以帮助 TensorFlow 更好地优化计算图。
- 但这会显著增加 RNN 的内存使用量。
- 因此，仅适用于较短的序列（100 步或以下）。
- 此外，只有当模型能提前知道数据中的时间步数量时才能使用（即传入 Input() 函数的 shape 参数中没有 None 项时）。

10.4 循环神经网络的高级用法

为减少过拟合，使用**循环 Dropout（Recurrent Dropout）**

- 使用下面这个

sequence_length是 None所以不行

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

通过传递 unroll=True 来展开（unrolling）。

10.4 循环神经网络的高级用法

堆叠循环层（Stacking Recurrent Layers）

- 过拟合已经不再出现，但似乎存在性能瓶颈，因此需要增加网络的容量和表达能力。
- 记住一般的机器学习工作流程。
- （假设已经通过使用 Dropout 等手段减少了过拟合）在出现过拟合之前，逐步增加模型容量是有益的。
- 只要模型没有出现明显的过拟合，就说明还没有达到足够的容量。
- 若要提升网络的容量，一般可通过增加层中单元的数量或添加更多层来实现。
- 堆叠循环层是构建更强大循环网络的经典方法。
- 例如，直到不久前，Google 翻译算法都使用包含 7 个大型 LSTM 层的庞大模型。
- 在 Keras 中，如果要依次堆叠循环层，则所有中间层必须输出整个序列（Rank-3 张量），而不仅仅是最后一个时间步的输出。
- 正如之前所学，需要设置 return_sequences=True。

10.4 循环神经网络的高级用法

堆叠循环层（Stacking Recurrent Layers）

- 在以下示例中，将堆叠两个使用 Dropout 正则化的循环层。
- 我们将进行一些变化，使用 GRU（门控循环单元，Gated Recurrent Unit）替代 LSTM。
- GRU 与 LSTM 非常相似。
- 可以将 GRU 看作是 LSTM 的结构更简单、更精简的版本。
- 当循环神经网络（RNN）开始在小规模研究社区中重新受到关注时，Cho Kyunghyun 等人于 2014 年首次提出了 GRU。

코드 10-23 드롭아웃 규제와 스태킹을 적용한 GRU 모델을 훈련하고 평가하기

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.GRU(32, recurrent_dropout=0.5, return_sequences=True)(inputs)
x = layers.GRU(32, recurrent_dropout=0.5)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
```

10.4 循环神经网络的高级用法

堆叠循环层（Stacking Recurrent Layers）

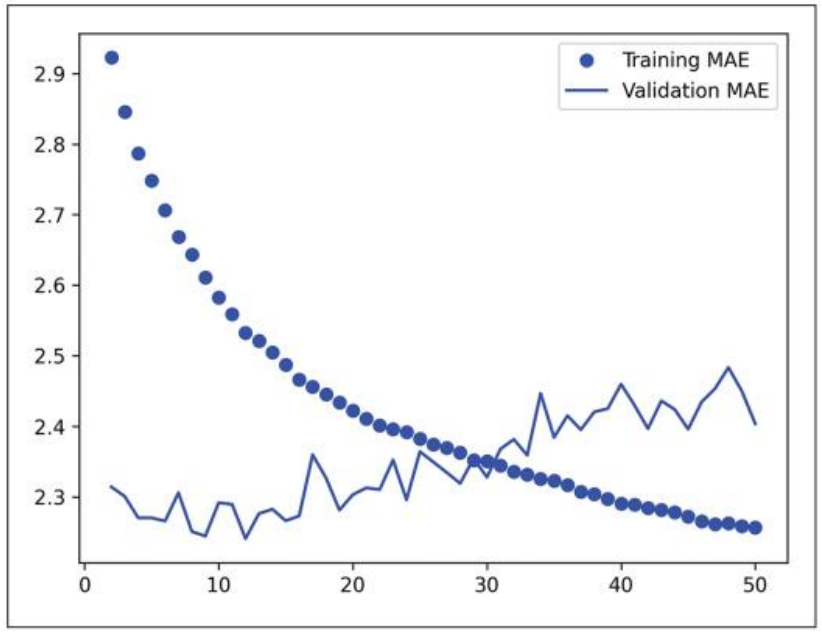
```
callbacks = [
    keras.callbacks.ModelCheckpoint("jena_stacked_gru_dropout.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=50,
                    validation_data=val_dataset,
                    callbacks=callbacks)
model = keras.models.load_model("jena_stacked_gru_dropout.keras")
print(f"테스트 MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

10.4 循环神经网络的高级用法

堆叠循环层（Stacking Recurrent Layers）

- 达到 2.39° 的测试 MAE（相比基准性能 提升了 8.8%）
- 虽然新增的层并未带来显著改进，但性能略有提升
- 在此情况下，可以认为 增加网络容量并不会带来帮助

▼ 图 10-12：应用于耶拿温度预测任务的 堆叠 GRU 网络 的训练与验证



10.4 循环神经网络的高级用法

使用双向 RNN

- 双向 RNN 是 RNN 的一种变体，在特定任务中能比基本 RNN 表现出更优异的性能。
- 在自然语言处理领域，它被誉为“瑞士军刀”般的常用工具。
- RNN 对顺序特别敏感。
- 即，它按照输入序列的时间步顺序进行处理。
- 若将时间步打乱或反转，会彻底改变 RNN 从序列中学习到的表示。
- 这也是为什么对于如温度预测这类顺序具有意义的问题，RNN 特别适合的原因。
- 双向 RNN 利用 RNN 对顺序敏感的特性。
- 它使用两个 RNN（如 GRU 或 LSTM）。
- 每个 RNN 分别按一个方向（时间顺序或反向顺序）处理输入序列，然后将两者的表示结合。
- 因为双向 RNN 能同时从两个方向处理序列，所以它能够捕捉到单向 RNN 容易遗漏的模式。

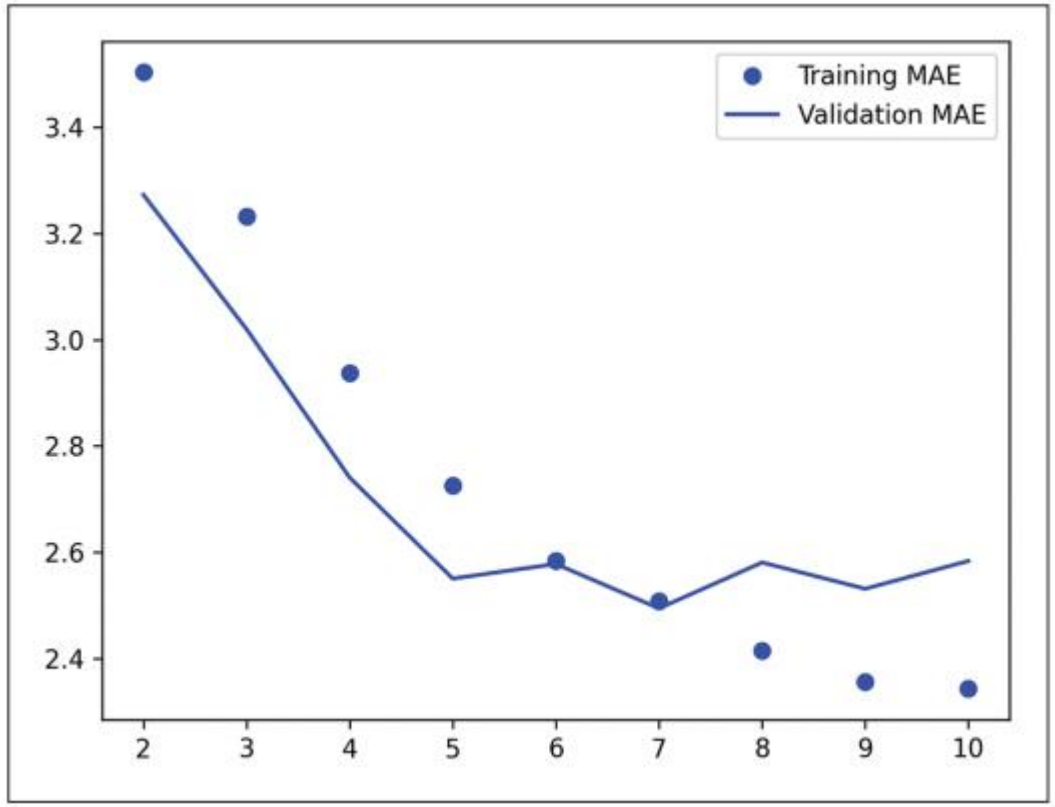
10.4 循环神经网络的高级用法

使用双向 RNN

- 令人惊讶的是，本节中 RNN 层按照时间顺序（旧的时间步先输出）处理序列的方式，其实并没有什么理论依据。
- 至少，之前并没有人好奇这个设定是否最优。
- 如果我们尝试让模型以相反的时间方向（即最近的时间步先输出）处理输入序列，RNN 的性能会如何呢？
- 我们将尝试这种方式并验证结果。
- 所需的只是构建一个沿时间维度反转输入序列的数据生成器（将生成器函数的最后一行改为：yield samples[:, ::-1, :], targets）。
- 然后，与本节第一个 LSTM 示例相同，训练一个基于 LSTM 的模型。

10.4 循环神经网络的高级用法

▼ 图 10-13 为了预测耶拿（Jena）的温度任务，使用反转序列训练的 LSTM 模型的训练损失与验证损失



10.4 循环神经网络的高级用法

使用双向 RNN

- 顺序被反转的 LSTM，其性能甚至低于常识水平的基准线
- 在这种情况下，按照时间顺序处理起着关键作用
- 一般而言，LSTM 层更擅长记住最近的信息，而不是久远的过去
- 此外，越接近当前时间的气象数据点，在预测中越有用（这也是常识性基准表现较强的原因）
- 因此，按照时间顺序处理的网络性能必须优于反向处理的网络

然而，在自然语言处理等许多其他任务中，情况却并非如此：

- 理解一句话时，单词的重要性并不完全由其在句子中的位置决定
- 对于文本数据集，反向处理序列的效果与顺序处理几乎相同
- 人类也可以倒着读文本（可以试试看！）
- 虽然词序对语言理解很重要，但并非决定性因素

10.4 循环神经网络的高级用法

使用双向 RNN

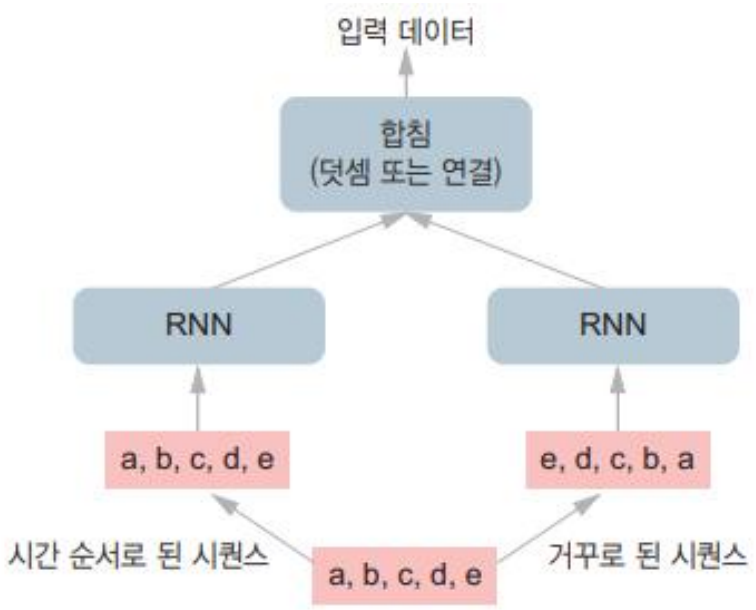
- 类比来说，就像如果我们的人生是从死亡开始、最后在出生中结束，那么我们对世界的感知将会完全不同。
- 在机器学习中，如果一种新的表示方式是有用的，那么它就值得被使用。
- 不同的表示之间的差异越大越好。
- 因为它们能够为数据提供全新的视角，并捕捉到其他方式可能忽略的特征。
- 这种互补的表示有助于提升任务性能。
- 这也正是第 13 章将要探讨的”集成学习（ensemble）”的核心概念。

10.4 循环神经网络的高级用法

使用双向 RNN

- 双向 RNN 利用这一思想，提升了按时间顺序处理的 RNN 的性能。
- 由于能够从输入序列的两个方向进行观察（见图 10-14），它可以获得更丰富的潜在表示，并捕捉到在单向时间处理时可能被忽略的模式。

▼ 图 10-14 双向 RNN 层的工作方式



10.4 循环神经网络的高级用法

使用双向 RNN

- 该类通过第一个参数接收一个循环层（RNN）的实例
- Bidirectional 类会基于接收到的循环层创建第二个新实例
- 一个实例按时间顺序处理输入序列，另一个实例按反向顺序处理输入序列
- 接下来我们将在温度预测任务中应用它

코드 10-24 양방향 LSTM 모델 훈련하고 평가하기

```
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)
```

10.4 循环神经网络的高级用法

使用双向 RNN

- 该模型的性能不如普通的 LSTM 层
- 其原因很容易理解
- 所有预测性能的一半来自按时间顺序处理的网络
- 另一半（按时间反向顺序处理的部分）在此类任务中的表现很差（因为最近信息远比过去信息更重要）
- 同时，由于包含了按时间反向顺序处理的层，网络的容量翻倍，更容易出现过拟合
- 双向 RNN 更适用于文本数据或顺序重要但顺序方向不重要的其他类型数据
- 实际上，在Transformer 结构出现之前（将在下一章学习），2016 年期间双向 LSTM 层曾在许多自然语言处理任务中取得顶级性能

10.4 循环神经网络的高级用法

更进一步

- 为了提升温度预测问题的性能，还有许多可以尝试的方向：
  - 调整堆叠的每个循环层的**单元数量**与**Dropout 比例**
    - 当前设置大多是**随机选择的**，可能还未达到最优
  - 调整 **RMSprop 优化器** 的学习率，或尝试使用**其他优化器**
  - 在循环层上方堆叠**多个 Dense 层**，而不是只使用一个
  - 改进模型输入：
    - 试验**更长或更短的序列**，或调整**采样间隔（sampling_rate）**
    - 或者进行**特征工程**

10.4 循环神经网络的高级用法

更进一步

- 一如既往，深度学习更接近艺术而非科学
- 虽然可以提出一些适用于特定问题的一般性指导方针，但每个数据集都是不同的
- 因此需要基于经验，对多种策略进行评估与尝试
- 目前并不存在能够提前预测哪种方法最优的理论
- 必须多次重复实验与尝试
- 根据作者经验，从非机器学习基准提升约 10% 已是此数据集的最优表现
- 虽称不上“卓越”，但依然是有意义的改进
- 如果拥有更广范围、多个地区的气象数据，则更容易预测近未来的天气
- 但若只有单一地区的观测值，则预测难度极大
- 当前地点的天气变化会随周边地区的天气模式而改变

10.4 循环神经网络的高级用法

更进一步

- **股票市场与机器学习**
- 一些读者可能会尝试将此处介绍的方法用于预测股票市场中的证券价格（或汇率等）。
- 但股票市场与天气模式等自然现象有着完全不同的统计特性。
- 在股票市场中，过去的表现并不是预测未来预期收益的良好指标。
- 这就像是只看后视镜开车一样。
- 因此，机器学习更适合用于天气、用电量、商店客流量等——在这些领域中，过去的数据能够成为预测未来趋势的可靠指标。

10.4 循环神经网络的高级用法

更进一步

- 请永远记住：所有交易的本质都是信息套利（information arbitrage）。
- 投资者通过利用他人忽略的数据或洞察来获得利润。
- 因此，若在股市中仅使用众所周知的机器学习算法和公开数据，最终只会走进死胡同。
- 因为相较于其他人并无信息优势。
- 最终可能一无所获，只是浪费时间和资源。

10.5 总结

10.5总结

总结

- 就像第 5 章中学到的那样，在解决新问题时，最好先在所选指标上设定一个常识性基准。
  - 如果没有基准，就无法判断模型是否真的得到了提升。
- 为了判断额外成本是否合理，应在尝试高计算成本的模型之前先尝试简单模型。
  - 有时，简单模型反而是最佳选择
- 当数据的时间顺序很重要，特别是包含时间序列数据时，循环神经网络（RNN）是合适的选择。
  - 这类模型往往能优于将时间序列“展开”后处理的模型。
  - 在Keras 中，两个核心的 RNN 层是 **LSTM 层** 和 **GRU 层**。

10.5总结

总结

- 如果要在循环网络中使用 Dropout，就必须在时间步之间使用固定的 Dropout 掩码和循环 Dropout 掩码。
  - 这两者都已包含在 Keras 的循环层中。
  - 只需在循环层中使用 recurrent_dropout 参数即可。

- 堆叠式 RNN（Stacked RNN）比单层 RNN 提供更强的表达能力。
  - 由于计算成本较高，因此并不总是值得尝试。
  - 在**机器翻译等复杂问题**中确实有帮助，但在小型、简单问题中未必如此。