Sequence models

处理按照时间排序或其他顺序排列的数据。（得有规律）

例如：文本，语言，时间序列。

常见序列模型。

RNN、LSTM、Transformer。

自回归模型

用过去的数据预测未来的数据。

- 第一种策略：训练到n的时候不必从保留从1到n-1的序列，满足一个跨度为τ（tau）的时间跨度就行，好处是参数永远不变。
- 第二种策略：保留一些对过去预测的总结，同时更新预测和总结。

马尔可夫模型

当前状态只依赖于前一个状态，而与更早的状态无关。

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_{t-1}) \text{ 当 } P(x_1 \mid x_0) = P(x_1).$$

$$\begin{aligned} P(x_{t+1} \mid x_{t-1}) &= \frac{\sum_{x_t} P(x_{t+1}, x_t, x_{t-1})}{P(x_{t-1})} \\ &= \frac{\sum_{x_t} P(x_{t+1} \mid x_t, x_{t-1}) P(x_t, x_{t-1})}{P(x_{t-1})} \\ &= \sum_{x_t} P(x_{t+1} \mid x_t) P(x_t \mid x_{t-1}) \end{aligned}$$

Code

很简单的MLP实现，不需要注释。

```
import torch
import torch.nn as nn
from d2l import torch as d2l

T = 1000
time = torch.arange(1, T + 1, dtype=torch.float32)
x = torch.sin(0.01 * time) + torch.normal(0, 0.2, (T,))
d2l.plot(time, [x], 'time', 'x', xlim=[1, 1000], figsize=(20, 8))

tau = 4  # τ = 4
features = torch.zeros((T - tau, tau))
print(features)
for i in range(tau):
    features[:, i] = x[i:T - tau + i]
labels = x[tau:].reshape(-1, 1)
```

```
batch_size, n_train = 16, 600
train_iter = d2l.load_array((features[:n_train], labels[:n_train]),
                            batch_size, is_train=True)

def init_weights(m):
    if type(m) == nn.Linear:
        nn.init.normal_(m.weight)

def get_net():
    net = nn.Sequential(nn.Linear(tau, 10),
                        nn.ReLU(),
                        nn.Linear(10, 1)
                        )
    net.apply(init_weights)
    return net

loss = nn.MSELoss()
def train(net, train_iter, loss, epochs, lr):
    optimizer = torch.optim.Adam(net.parameters(), lr=lr)
    for epoch in range(epochs):
        for X, y in train_iter:
            optimizer.zero_grad()
            l = loss(net(X), y)
            l.backward()
            optimizer.step()
        print(f'epoch {epoch + 1}, '
              f'loss {d2l.evaluate_loss(net, train_iter, loss):.3f}')

net = get_net()
train(net, train_iter, loss, 100, 0.01)

# 单步预测
onestep_preds = net(features)
d2l.plot([time, time[tau:]],
         [x.detach().numpy(), onestep_preds.detach().numpy()],'time',
         'x', legend=['data', '1-step preds'], xlim=[1, 1000],figsize=(6, 3))
d2l.plt.show()

# 多步预测
multistep_preds = torch.zeros(T)
multistep_preds[:n_train + tau] = x[:n_train + tau]
for i in range(n_train + tau, T):
    multistep_preds[i] = net(multistep_preds[i - tau:i].reshape((1,-1)))
d2l.plot( [time, time[tau:], time[n_train + tau:]],
    [x.detach().numpy(), onestep_preds.detach().numpy(), multistep_preds[n_train +
tau:].detach().numpy()],
    'time',  'x',   legend = ['data', '1-step preds', 'multistep preds'],xlim=
[1,1000],figsize=(6,3) )
d2l.plt.show()

# 变化步数预测
max_steps = 64
features = torch.zeros((T - tau - max_steps + 1, tau + max_steps))
for i in range(tau):
    features[:, i] = x[i:i + T - tau - max_steps + 1]

for i in range(tau, tau + max_steps):
    features[:,i] = net(features[:, i - tau:i]).reshape(-1)
```

```
steps = (1, 4, 16, 64)
d2l.plot([time[tau + i - 1:T - max_steps + i] for i in steps],
         [features[:,(tau + i - 1)].detach().numpy() for i in steps],
         'time', 'x',legend = [f'{i}-step preds' for i in steps], xlim = [5,1000],
         figsize=(6,3) )
d2l.plt.show()
```