Attention

注意力和体力一样是一种消耗品

人为了生存不消耗太多大脑资源通常会集中注意力干一件事

像健身一样，你不能一天把胸腿肩全练到位，不然明天怎么生存?

神经网络怎么集中注意力

Query：你吃面包噎到了，开冰箱只会想找水，不会去找菜。因为你的Q（需求）是噎住了，需要水。

Key: 冰箱里面的各种东西，每一样东西都有一个K（关键）：水、面包、可乐、蔬菜。K用于与Q匹配计算相似度。

Value：物品特性（值）：水（V：缓解干燥）面包（V：主食）可乐（V：饮料）蔬菜（V：营养食物）

重要度计算（相似度计算）

$$Q，K = k_1, k_2 \ldots\ldots k_n$$

$$将Q和K内的每一个k点乘计算相似度得到S = s_1, s_2 \ldots\ldots s_n$$

$$做一层Softmax(S)就可以得到权重A = a_1, a_2 \ldots\ldots a_n$$

$$再用A点乘V得到V', 用V'代替V进行下一层计算$$

$$一般K = V(Transformer)，有时候不一样，但一定有关系$$

Q、K、V来源

在模型训练开始时，权重矩阵会被随机初始化。通常以小的随机值（零均值小方差的正态分布）初始化。

这种随机初始化有助于打破对称性，从而确保每个神经元在训练初期有不同的学习路径。

权重矩阵是在模型训练过程中，通过对输入数据的多次迭代优化而学习到的。

这些矩阵在初始化时是随机的，随后通过梯度下降算法逐步调整，以优化模型的性能。

最终用于有效地提取输入数据中的重要特征。

Attention Code

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

# Attention Mechanism
# Nadaraya-Watson kernel

# Generate data
n_train = 50
x_train, _ = torch.sort(torch.rand(n_train) * 5) # ,_ 抛弃第二个返回值即索引

# Target function
def f(x):
    return 2 * torch.sin(x) + x ** 0.8
```

```python
# 添加噪声
y_train = f(x_train) + torch.normal(0.0, 0.5, (n_train,))

# Test data
x_test = torch.arange(0, 5, 0.1)
y_truth = f(x_test)
x_test_len = len(x_test)
print(x_test_len)

def plot_kernel_reg(y_hat):
    d2l.plot(x_test, [y_truth, y_hat], 'x', 'y', legend=['Truth','Pred'],
             xlim=[0,5], ylim=[-1,5])
    d2l.plt.plot(x_train, y_train, 'o', alpha=0.5)

# 平均预测结果
# maan计算平均值
y_hat = torch.repeat_interleave(y_train.mean(), x_test_len)
plot_kernel_reg(y_hat)
d2l.plt.show()

# 非参数注意力汇聚，沐神将Pooling不叫池化叫汇聚
X_test_repeat = x_test.repeat_interleave(n_train).reshape((-1, n_train))
attention_weights = nn.functional.softmax(-(X_test_repeat - x_train) ** 2 / 2, dim=1)
y_hat = torch.matmul(attention_weights, y_train)
plot_kernel_reg(y_hat)
d2l.plt.show()

# 可视化注意力权重
d2l.show_heatmaps(attention_weights.unsqueeze(0).unsqueeze(0),
                  xlabel='Sorted training inputs', ylabel='Sorted test inputs')
d2l.plt.show()


weights = torch.ones((2,10)) * 0.1
values = torch.arange(20.0).reshape((2,10))
print(torch.bmm(weights.unsqueeze(1), values.unsqueeze(-1)))

# 带参数的注意力汇聚
class NWKernelRegression(nn.Module):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.w = nn.Parameter(torch.rand((1,),requires_grad=True))

    def forward(self, queries, keys, values):
        queries = queries.repeat_interleave(keys.shape[1]).reshape(-1,keys.shape[1])
        self.attention_weights = nn.functional.softmax(-((queries - keys) *
self.w)**2/2,dim=1)
        return
torch.bmm(self.attention_weights.unsqueeze(1),values.unsqueeze(-1)).reshape(-1)
X_tile = x_train.repeat((n_train, 1))
Y_tile = y_train.repeat((n_train, 1))
keys = X_tile[(1 - torch.eye(n_train)).type(torch.bool)].reshape((n_train,-1))
values = Y_tile[(1 - torch.eye(n_train)).type(torch.bool)].reshape(n_train, -1)

net = NWKernelRegression()
loss = nn.MSELoss(reduction='none')
trainer = torch.optim.SGD(net.parameters(), lr=0.5)
```

```
animator = d2l.Animator(xlabel='epoch',ylabel='loss',xlim=[1,5])

for epoch in range(5):
    trainer.zero_grad()
    l = loss(net(x_train, keys, values), y_train) / 2
    l.sum().backward()
    trainer.step()
    print(f'epoch {epoch+1}, loss {float(l.sum()):.6f}')
    animator.add(epoch+1, float(l.sum()))
d2l.plt.show()

keys = x_train.repeat((x_test_len, 1))
values = y_train.repeat((x_test_len, 1))
y_hat = net(x_test, keys, values).unsqueeze(1).detach()
plot_kernel_reg(y_hat)
d2l.plt.show()
d2l.show_heatmaps(net.attention_weights.unsqueeze(0).unsqueeze(0),
                  xlabel='Sorted training inputs', ylabel='Sorted testing inputs')
d2l.plt.show()
```