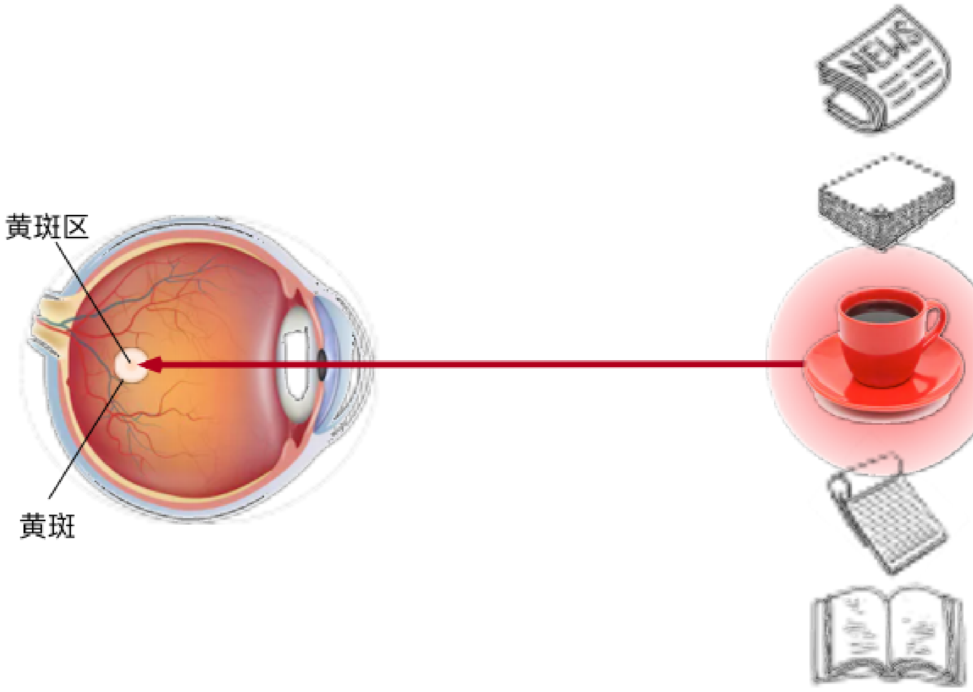
1.注意力机制引入

- 经济学中对于注意力的看法：
  人们正处在“注意力经济”时代，即人类的注意力被视为可以交换的、有限的、有价值的且稀缺的商品。许多商业模式也被开发出来去利用这一点：在音乐或视频流媒体服务上，人们要么消耗注意力在广告上，要么付钱来隐藏广告；为了在网络游戏世界的成长，人们要么消耗注意力在游戏战斗中，从而帮助吸引新的玩家，要么付钱立即变得强大。总之，注意力不是免费的。
- 生物学中的注意力提示：
  - 非自主性的注意力提示：
    非自主性提示是基于环境中物体的突出性和易见性。想象一下，假如我们面前有五个物品：一份报纸、一篇研究论文、一杯咖啡、一本笔记本和一本书。

    所有纸制品都是黑白印刷的，但咖啡杯是红色的。换句话说，这个咖啡杯在这种视觉环境中是突出和显眼的，不由自主地引起人们的注意。

    所以我们会把视力最敏锐的地方放到咖啡上，正如上图所示。
  - 自主性的注意力提示：
    喝咖啡后，我们会变得兴奋并想读书，所以转过头，重新聚焦眼睛，然后看看书，此时选择书是受到了认知和意识的控制，因此注意力在基于自主性提示去辅助选择时将更为谨慎。受试者的主观意愿推动，选择的力量也就更强大。
  - 上面的**自主性和非自住性注意力提示**解释了 人类的注意力的方式 。

设置注意力机制的框架：

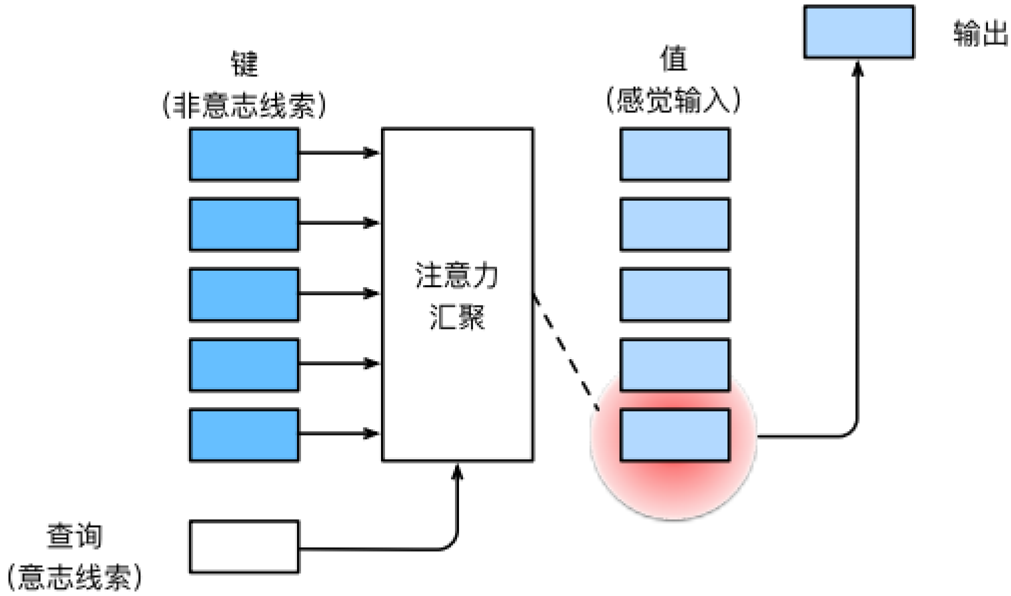
> 下面来看看如何通过这两种注意力提示，用神经网络来设计注意力机制的框架。

首先，考虑一个相对简单的状况，即只使用非自主性提示。

要想将选择偏向于感官输入，则可以简单地使用参数化的全连接层，甚至是非参数化的最大汇聚层或平均汇聚层。因此，“是否包含自主性提示”将注意力机制与全连接层或汇聚层区别开来。在注意力机制的背景下，自主性提示被称为**查询**（query）。

给定任何查询，注意力机制通过**注意力汇聚**（attention pooling）将选择引导至**感官输入**（sensory inputs，例如中间特征表示）。在注意力机制中，这些感官输入被称为**值**（value）。更通俗的解释，每个值都与一个**键**（key）配对，

这可以想象为感官输入的非自主提示。



如 :numref: `fig_qkv` 所示，可以通过设计注意力汇聚的方式，

便于给定的查询（自主性提示）与键（非自主性提示）进行匹配，

这将引导得出最匹配的值（感官输入）。

- 在没有任何提示的情况下，人本身可能会产生很多的$key \rightarrow value$。
  此时key就是 `非自主性提示` ，value是对应的 `感官输入` 。
  通俗的理解，就是遇到一张图片，仅仅观看图片，我能得到很多$key \rightarrow value$信息。这些东西并没有很强烈的外界引导(比如说我一定要观察里面的动物、或者一定关注书本里面的内容)。此时的所有的$key \rightarrow value$都是 `非自主性的` 。
  自主性的提示，就是一个 `query` ，询问。比如，现在为了解决问题：图片里面有几只猫。
  此时，我们就会在看图片的时候，有自主性的关注。在这种情况下，对于之前所有的$key \rightarrow value$，每一个键值对，都会有一个关注程度。
  很明显key和猫有关系的，会明显比key和狗有关系的，更能吸引我们。
  所以也就引入了后文中，对于相似性的计算。

> - **用自己的语言再总结一下这个框架**：
>   首先，没有任何提示(并没有强调我要去更加关注图片里的人、或者更加关注猫)，只关注一张图片的时候。会有很多的$key \rightarrow value$产生。此时key就是 `非自主性提示` ，value是对应的 `感官输入` 。
>   此时所有的$key \rightarrow value$都是 `非自主性的` 。
>
>   `自主性提示` ，理解为一个 `query` 。比如现在有一个query是图片里面有几只猫。
>   正式的说：给定任何查询，注意力机制通过 `注意力汇聚（attention pooling）` 将选择引导至对应的 `感官输入` （也就是value）。
>   通俗的讲：在这种情况下，对于之前所有的$key \rightarrow value$，每一个键值对，都会有一个关注程度。很明显key和猫有关系的，会明显比key和狗有关系的，更能吸引我们。我们也就会被

> 引导至 `与猫有关的key` 对应的value（感官输入）。
> 上面的 `对于不同键值对的关注程度` ，也就是后文中，对于相似性的计算。

权重的解释：

在注意力机制中，对于某一个特定的 `query` ，每一个$key \rightarrow value$会被赋予一个新的权重，通过新的权重和value才能够计算出来最后的结果。
权重的作用：可以在每一个query下，模型更加聚焦于有用的信息上。

- 此时的工作模式为：
  加权平均的总和值：与“平均汇聚”不同，注意力机制通过“加权”来调整各个输入对结果的贡献。具体来说，不同的输入会有不同的权重，权重是通过计算查询（Query）和键（Key）之间的关系得出的。这就是注意力机制的核心思想：根据输入的重要性来动态地调整每个部分的权重。
  查询（Query）：是我们要查询的内容。
  键（Key）：是我们用来与查询进行比较的其他信息。
  值（Value）：是我们最终会用来做汇总的输入信息。

  权重计算：通过计算查询和键之间的关系（通常是相似度或匹配度），得到每个输入的权重。然后，这些权重会用来调整输入的值，最后加权平均得到一个最终的聚合结果。
- 进行权重的可视化。

> ！！！！ 11权重的可视化。
> 这里其实就只是简单的 进行 权重的显示。
> 在ipynb里面 和 pdf里面的解释都挺好的。
>
> 如果后面有用到，再加一些总结。
> 这里之后再进行记录。
> 对于代码的解释，需要的时候再问chatgpt.

- 一句话总结注意力机制的过程：
  `查询` （自主提示）和 `键` （非自主提示）之间交互形成 `注意力汇聚` ，注意力汇聚有选择地聚合 `值` （感官输入）。

2.Nadaraya-Watson核回归模型

简单但是完整的例子。

2.1引入问题

- 问题：（通过实际的问题的解决过程来学习对应的模型）
  回归问题：给定的成对的“输入 - 输出”数据集$\{(x_1, y_1), \ldots, (x_n, y_n)\}$。
  如何学习f来预测任意新输入x的输出$\hat{y} = f(x)$?

- 根据下面的非线性函数生成一个人工数据集，其中加入的噪声项为ϵ：
  $y_i = 2\sin(x_i) + x_i^{0.8} + \epsilon,$
  其中ϵ服从均值为0和标准差为0.5的正态分布。在这里生成了50个训练样本和50个测试样本。

2.2创建数据集

- 生成50哥训练样本和50个测试样本：

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

```
n_train = 50   # 训练样本数
x_train, _ = torch.sort(torch.rand(n_train) * 5)     # 排序后的训练样本
#_返回的是排序之后的索引 不过这个索引之后并不会用到。
#进行排序是为了之后再可视化的时候，方便观看效果。
#训练样本x的范围是(0,5)
```

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

y_train = f(x_train) + torch.normal(0.0, 0.5, (n_train,))    #计算训练样本的输出
x_test = torch.arange(0, 5, 0.1)    # 构建测试样本
y_truth_test = f(x_test)    # 测试样本的真实输出
n_test = len(x_test)    # 测试样本数
n_test
```

- x_train是训练数据的x，y_train是训练数据的y.
  x_test是测试数据的x，y_truth_test是测试数据的真实输出。

2.3绘制图像来可视化最后的预测结果 & 仅仅使用所有训练数据输出的平均值作为结果进行预测

- 下面的函数将绘制所有的训练样本（样本由圆圈表示），不带噪声项的真实数据生成函数f（标记为“Truth”），以及学习得到的预测函数（标记为“Pred”）。

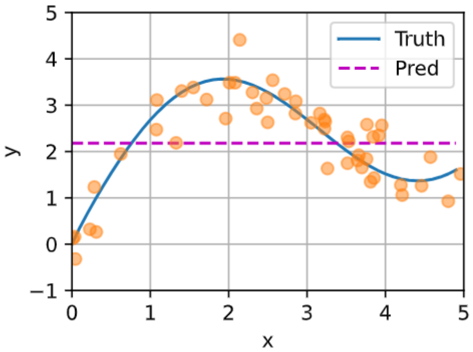
```
def plot_kernel_reg(y_hat): #y_hat 为学习到的最后的预测结果。
      d2l.plot(x_test, [y_truth_test, y_hat], 'x', 'y', legend=['Truth', 'Pred'],
                  xlim=[0, 5], ylim=[-1, 5])
      d2l.plt.plot(x_train, y_train, 'o', alpha=0.5);
```

- 仅仅使用：

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} y_i, \tag{1}$$

```
y_hat = torch.repeat_interleave(y_train.mean(), n_test)
plot_kernel_reg(y_hat)
```

结果：



可以明显看到，效果很不好。

2.4非参数注意力汇聚

- 原先模型的公式以及简单解释：

  显然，平均汇聚忽略了输入x_i。于是Nadaraya (Nadaraya, 1964)和 Watson (Watson, 1964)提出了一个更好的想法，根据输入的位置对输出y_i进行加权：

  $$f(x) = \sum_{i=1}^{n} \frac{K(x - x_i)}{\sum_{j=1}^{n} K(x - x_j)} y_i, \tag{10.2.3}$$

  其中K是核（kernel）。公式 (10.2.3)所描述的估计器被称为 *Nadaraya-Watson*核回归（Nadaraya-Watson kernel regression）。这里不会深入讨论核函数的细节，但受此启发，我们可以从 图10.1.3中的注意力机制框架的角度

  我们并不研究这个公式，重点在于，得益于这样的启发，`根据注意力机制框架的角度重写了` 更加通用的**注意力汇聚**公式。

- 通用的**注意力汇聚**公式。

  $$f(x) = \sum_{i=1}^{n} \alpha(x, x_i) y_i \tag{2}$$

  其中x是查询，(x_i, y_i)是键值对。注意力汇聚是y_i的加权平均。
  将查询x和键x_i之间的关系建模为**注意力权重**（attention weight）$\alpha(x, x_i)$。

  这个权重将被分配给每一个对应值y_i。对于任何查询，模型在所有键值对注意力权重都是一个有效的概率分布：它们是非负的，并且总和为1。

- 在这个问题里面，我们借助**高斯核**来实现**注意力权重**的计算。

  高斯核的公式： $K(u) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{u^2}{2}).$

  讲其带入 `原先模型的公式里面`，再带入通用的**注意力汇聚**公式。就得到：

  $$\begin{aligned} f(x) &= \sum_{i=1}^{n} \alpha(x, x_i) y_i \\ &= \sum_{i=1}^{n} \frac{\exp\left(-\frac{1}{2}(x - x_i)^2\right)}{\sum_{j=1}^{n} \exp\left(-\frac{1}{2}(x - x_j)^2\right)} y_i \\ &= \sum_{i=1}^{n} \operatorname{softmax}\left(-\frac{1}{2}(x - x_i)^2\right) y_i. \end{aligned}$$

  基于这个公式，我们就可以进行预测了。
  `观察公式：` 会发现对于当前的查询 `x`，如果越接近一个key `x_i` 。对应的**注意力权重**也就会越大。

- **基于上述公式，计算并且绘制最后的结果**：

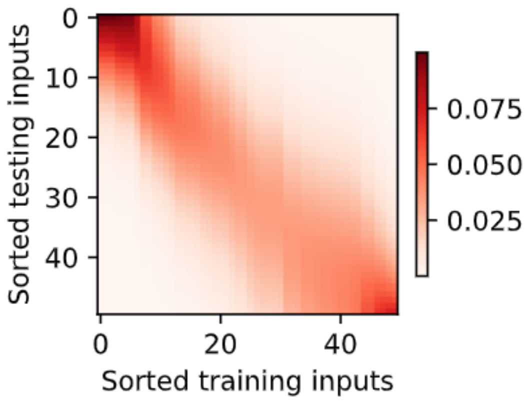
```
#对于每一个需要进行测试的元素，所有的训练集都应该与他计算出来一个相似度，也就是注意力权重。
#所以注意力权重的shapr = (n_test,n_train)。行数等于 待测数据个数，列数等于 训练集样本个数。
X_repeat = x_test.repeat_interleave(n_train).reshape((-1, n_train))
print(X_repeat)

#计算每一个的权重：
attention_weights = nn.functional.softmax(-(X_repeat - x_train)**2 / 2, dim=1)

#得到每一个对应的权重，就可以计算最终结果：
y_hat = torch.matmul(attention_weights, y_train)
plot_kernel_reg(y_hat)
```

- 打印注意力权重的热图：

```
d2l.show_heatmaps(attention_weights.unsqueeze(0).unsqueeze(0),
                                  xlabel='Sorted training inputs',
                                  ylabel='Sorted testing inputs')
```



因为训练数据和测试数据都是排好序的。此时观察可以发现 query - key 越接近，最后的**注意力权重**越高。

2.5带参数的注意力汇聚

将原先的模型修改为：

$$
\begin{aligned}
f(x) &= \sum_{i=1}^{n} \alpha(x, x_i) y_i \\
&= \sum_{i=1}^{n} \frac{\exp\left(-\frac{1}{2}((x - x_i)w)^2\right)}{\sum_{j=1}^{n} \exp\left(-\frac{1}{2}((x - x_j)w)^2\right)} y_i \\
&= \sum_{i=1}^{n} \operatorname{softmax}\left(-\frac{1}{2}((x - x_i)w)^2\right) y_i.
\end{aligned}
$$

接下来我们讨论如何训练模型来学习这个参数w.

> **批量矩阵乘法操作**：
> 假如一个矩阵X形状为$a \times b$,Y的形状为$b \times c$.
> 矩阵乘法：XY 结果的形状为$a \times c$.

批量矩阵乘法：
假设第一个小批量数据包含n个矩阵$\mathbf{X}_1, \ldots, \mathbf{X}_n$，形状为$a \times b$，
第二个小批量包含n个矩阵$\mathbf{Y}_1, \ldots, \mathbf{Y}_n$，形状为$b \times c$。
它们的批量矩阵乘法得到n个矩阵$\mathbf{X}_1\mathbf{Y}_1, \ldots, \mathbf{X}_n\mathbf{Y}_n$，形状为$a \times c$。
因此，[**假定两个张量的形状分别是(n,a,b)和(n,b,c)，它们的批量矩阵乘法输出的形状为(n,a,c)**]。

代码为：

```
X = torch.ones((2, 1, 4))
Y = torch.ones((2, 4, 6))
torch.bmm(X, Y).shape
```