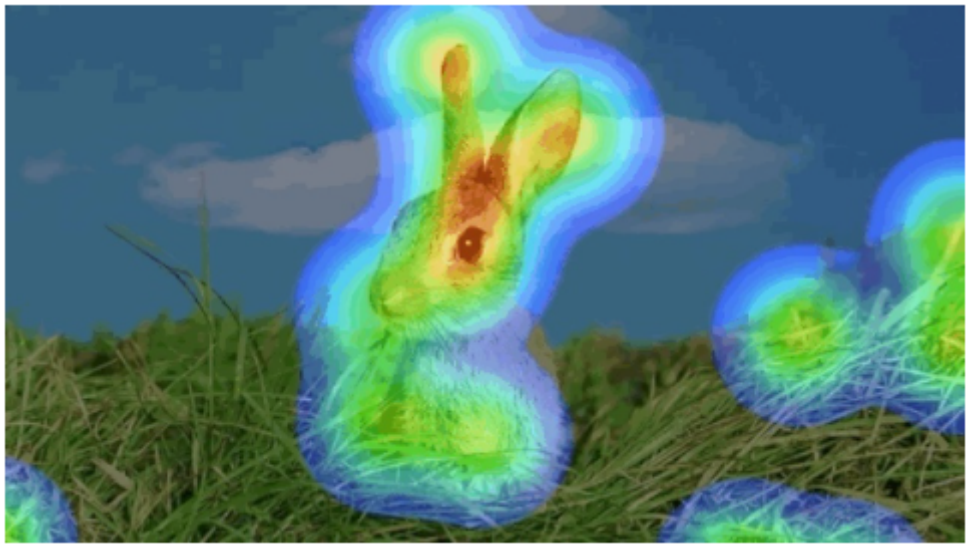
注意力机制（Attention）

- 首先通过一张图来了解注意力机制。



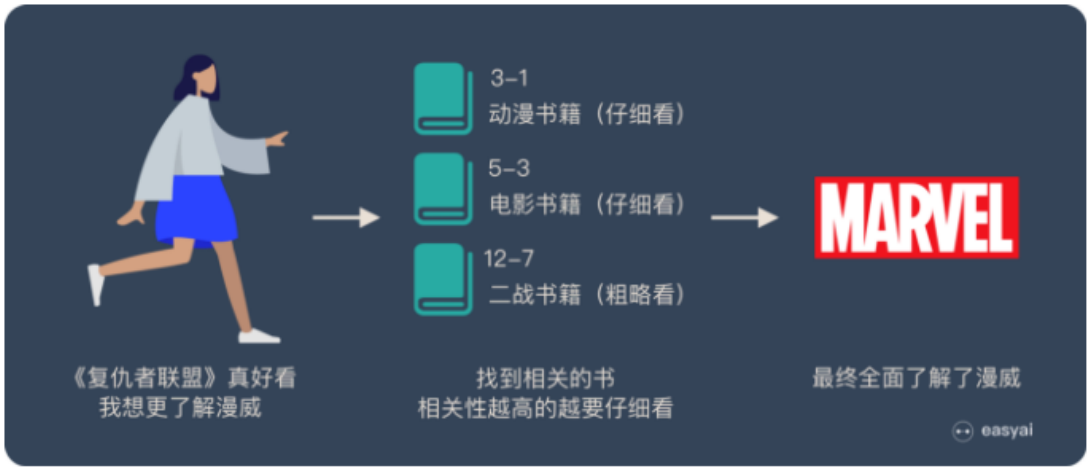
我们对于任何一张图，首先关注的就是关键信息，例如上图中的兔子。

Attention的3大优点

- 参数少
  - 模型复杂度跟CNN、RNN相比，复杂度更小，参数也更少。所以对算力的要求更小。
- 速度快
  - Attention解决了RNN不能并行计算的问题。Attention机制每一步计算不依赖于上一步的计算结果，因此可以和CNN一样并行处理。
- 效果好
  - 在Attention机制引入之前，由于长距离的信息会被弱化，记忆力差。Attention是挑重点，就算文本比较长，也能从中间抓住重点，不丢失重要的信息。

Attention的原理

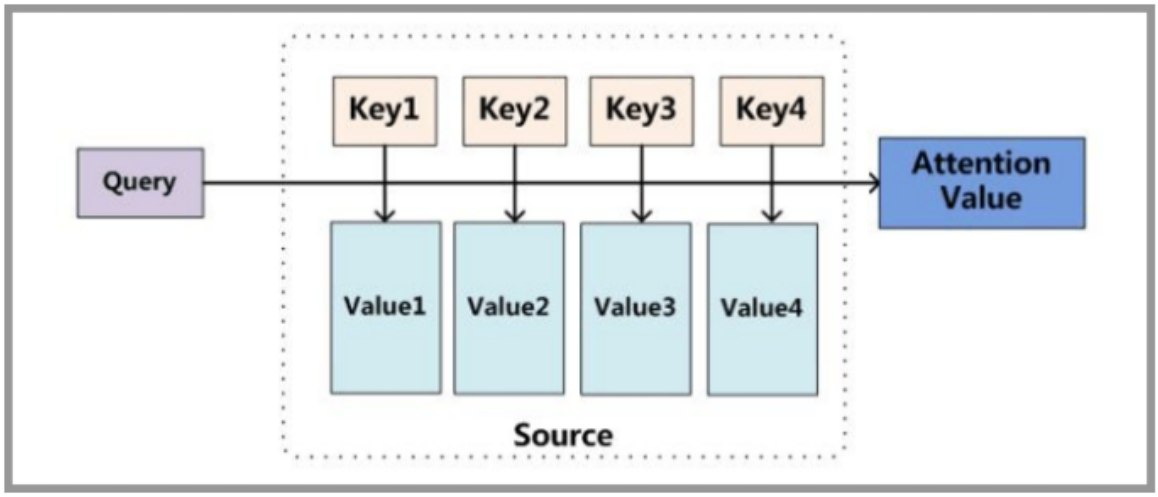
- 用这个例子来解释attention的原理：



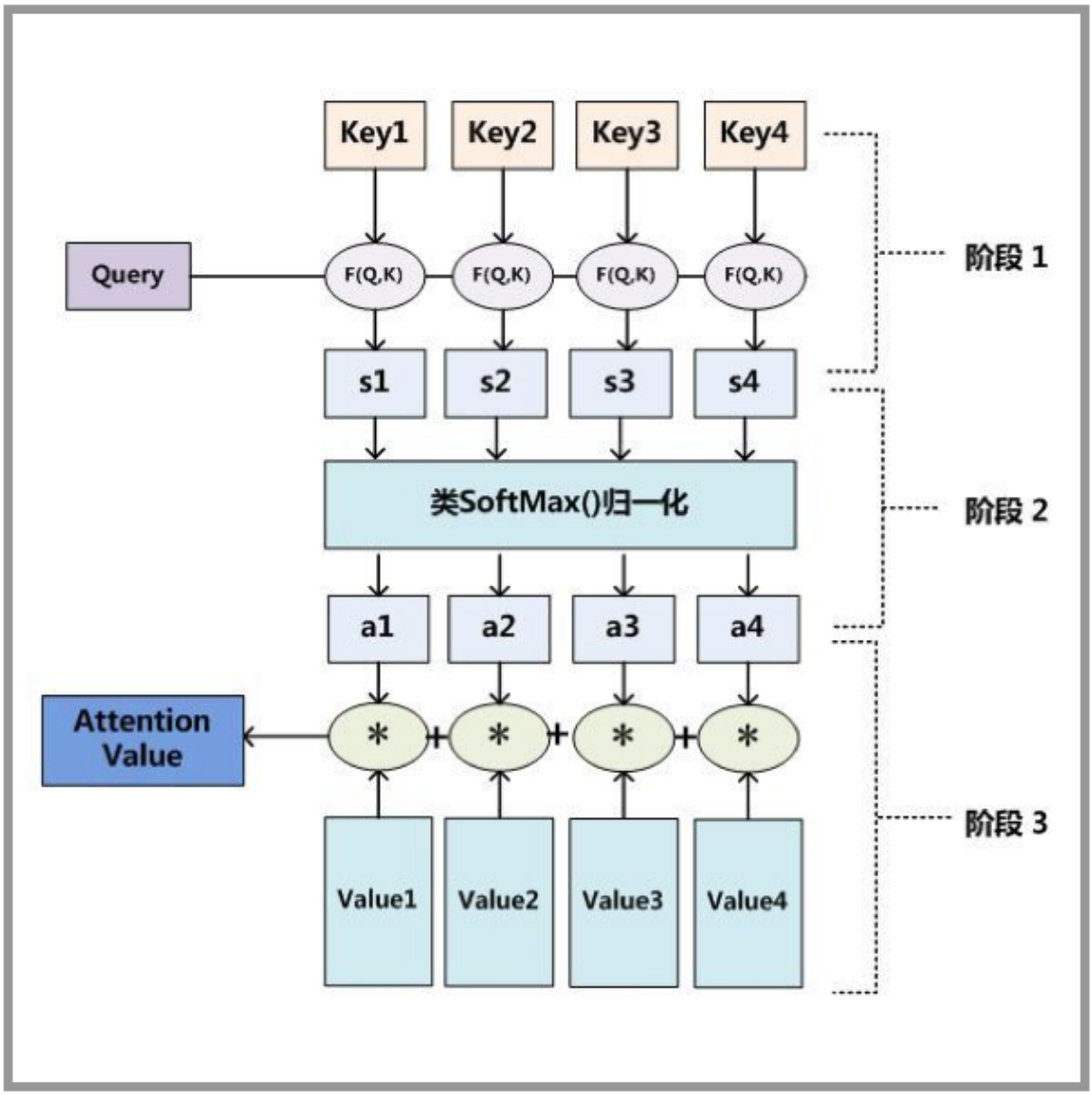
图书馆（source）里有很多书（value），为了方便查找，我们给书做了编号（key）。当我们想要了解漫威（query）的时候，我们就可以看看那些动漫、电影、甚至二战相关的书籍。

为了提高效率，并不是所有的书都会仔细看，针对漫威来说，动漫，电影相关的会看的仔细一些（权重高），但是二战的就只需简单扫一下即可（权值低）。

注意力机制模型



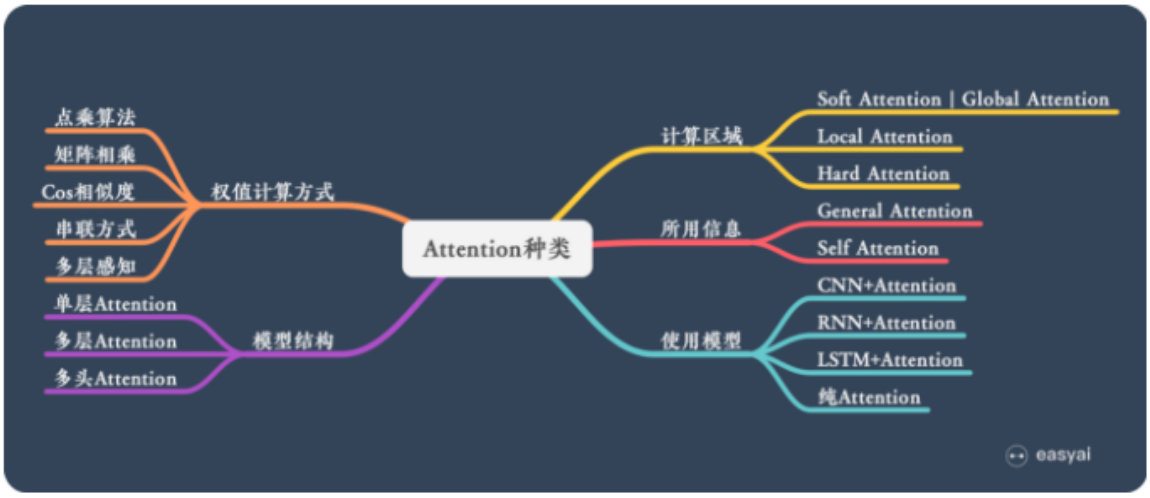
Attention原理的3步分解：



- 第一步：query和key进行相似度计算，得到权值。
- 第二步：将权值进行归一化，得到直接可用的权值。
- 第三步：将权值和value进行加权求和。

- Attention是从大量信息中筛选出少量重要信息，并聚焦到这些重要信息上，忽略大多不重要的信息。权重越大越聚焦于其对应的Value值上，即权值代表了信息的重要性，而Value是其对应的信息。
- Attention计算过程可以归纳为两个过程：第一个过程是根据Query和Key计算权重系数，第二个过程根据权值系数对Value进行加权求和。第一个过程又可以分为两个阶段：第一个阶段根据Query和Key计算两者的相似性或者相关性；第二个阶段对第一阶段的原始分值进行归一化处理；这样，可以将Attention的计算过程抽象为如图展示的三个阶段。

Attention的N种类型



- Attention有多种不同的类型：**Soft Attention**、**Hard Attention**、**静态Attention**、**动态Attention**、**Self Attention**等等。

- **计算区域**
  - **Soft Attention**，这是比较常见的Attention方式，对所有的key求权重概率，每个key都有一个对应的权值，是一种全局的计算方式。但是计算量可能会比较大一些。
  - **Hard Attention**，这种方式是直接精准定位到某个key，其余key就都不管了，相当于这个key的概率是1，其余key的概率全是0.因此这种对齐方式要求很高，要求一步到位，如果没有正确对齐，会有很大影响。另一方面，因为不可导，一般需要用强化学习的方法进行训练。
  - **Local Attention**，这种方式其实是以上两种方式的折中，对一个窗口区域进行计算，先用Hard方式定位到某个地方，以这个点为中心可以得到一个窗口区域，在这个小区域内用Soft方式来计算Attention。
- **所用信息**

  假设我们要对一段原文计算Attention，那么所用信息包括内部信息和外部信息，内部信息指的是原文本身的信息，外部信息指的是除原文以外的额外信息。
  - **General Attention**，这种方式利用到了外部信息，常用于需要构建两段文本关系的任务，query一般包含了额外信息，根据外部query对原文进行对齐。
  - **Local Attention**，这种方式只是用内部信息，key和value以及query只和输入原文有关，在self-attention中，key=value=query。既然没有外部信息，那么在原文中的每个词可以跟该句子中的所有词进行Attention计算，相当于寻找原文内部的关系。
- **结构层次**
  - **单层Attention**，用一个query对一段原文进行一次attention。
  - **多层Attention**，一般用于文本具有层次关系的模型，假设我们把一个document划分成多个句子，在第一层，我们分别对每个句子使用attention计算出一个句向量（也就是单层attention）；在第二层，我们对所有句向量再做attention计算出一个文档向量（也是一个单层attention），最后再用这个文档向量去做任务。
  - **多头Attention**，用到了多个query对一段原文进行了多次attention，每个query都关注到原文的不同部分，相当于重复做多次单层attention：

    $$head_i = Attention(q_i, K, V)$$

    最后再把这些结果拼接起来：

    $$MultiHead(Q,\ K,\ V) = Concat(head_1, \ldots, head_h)W^O$$
- **模型方面**
  - **CNN+Attention**

CNN的卷积操作可以提取重要特征，但是CNN的卷积感受野是局部的，需要通过叠加多层卷积区去扩大视野。另外，Max Pooling直接提取数值最大的特征，很像Hard attention。

1. 在卷积操作前做attention，比如Attention-Based BCNN-1，这个任务是文本蕴含任务需要处理两段文本，同时对两端输入的序列向量进行attention，计算出特征向量，再拼接到原始向量中，作为卷积层的输入。
2. 在卷积操作后做attention，比如Attention-Based BCNN-2，对两段文本的卷积层的输出做attention，作为pooling层的输入。
3. 在pooling层做attention，代替max pooling。比如Attention pooling，首先我们用LSTM学到一个比较好的句向量，作为query，然后用CNN先学习到一个特征矩阵作为key，再用query对key产生权值，进行attention，得到最后的句向量。

- **LSTM+Attention**

  LSTM内部有Gate机制，其中input gate选择哪些当前信息进行输入，forget gate选择遗忘哪些过去信息，这算是一定程度的Attention了，而且可以解决长期依赖问题，实际上STM需要一步一步去捕捉序列信息，在长文本上表现是会随着step增加而慢慢衰减，难以保留全部的有用信息。

  LSTM通常需要得到一个向量，再去做任务，常用方式有：

  1. 直接使用最后 的hidden state（可能会说你是一定的前文信息，难以表达全文）。
  2. 对所有step下的hidden state进行等权平均。
  3. Attention机制，对所有step的hidden state进行加权，把注意力集中到整段文本比较重要的hidden state信息。

- **纯Attention**

- **相似度计算方式**

  在做attention的时候，需要计算query和某个key的分数（相似度）。

  - 点乘：$s(q,k) = q^T k$
  - 矩阵相乘：$s(q,k) = q^T k$
  - cos相似度：$s(q,k) = \frac{q^T k}{||q|| \cdot ||k||}$
  - 串联方式：把q和k拼接起来：$s(q,k) = W[q;k]$
  - 多层感知机：$s(q,k) = v_a^T tanh(Wq + Uk)$