最近两年，注意力模型（Attention Model）被广泛使用在自然语言处理、图像识别及语音识别等各种不同类型的深度学习任务中，是深度学习技术中最值得关注与深入了解的核心技术之一。

本文以机器翻译为例，深入浅出地介绍了深度学习中注意力机制的原理及关键计算机制，同时也抽象出其本质思想，并介绍了注意力模型在图像及语音等领域的典型应用场景。

注意力模型最近几年在深度学习各个领域被广泛使用，无论是图像处理、语音识别还是自然语言处理的各种不同类型的任务中，都很容易遇到注意力模型的身影。所以，了解注意力机制的工作原理对于关注深度学习技术发展的技术人员来说有很大的必要。

人类的视觉注意力

从注意力模型的命名方式看，很明显其借鉴了人类的注意力机制，因此，我们首先简单介绍人类视觉的选择性注意力机制。

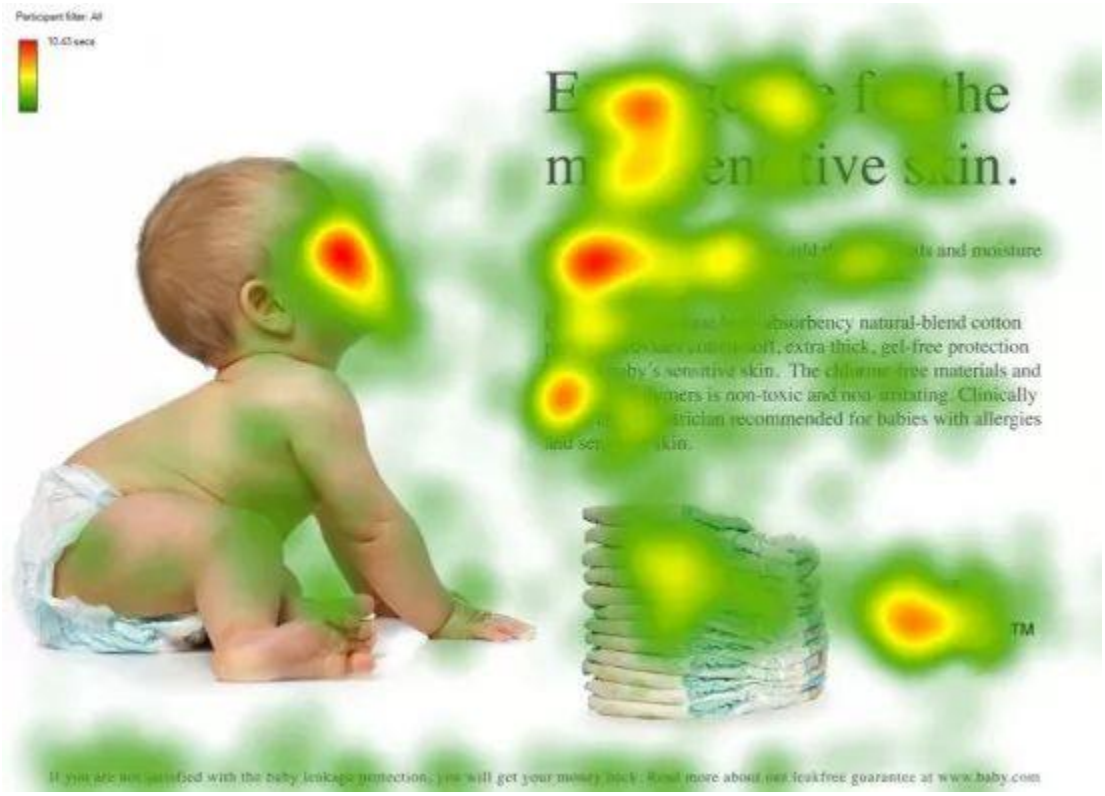

图 1 人类的视觉注意力

视觉注意力机制是人类视觉所特有的大脑信号处理机制。人类视觉通过快速扫描全局图像，获得需要重点关注的目标区域，也就是一般所说的注意力焦点，而后对这一区域投入更多注意力资源，以获取更多所需要关注目标的细节信息，而抑制其他无用信息。

这是人类利用有限的注意力资源从大量信息中快速筛选出高价值信息的手段，是人类在长期进化中形成的一种生存机制，人类视觉注意力机制极大地提高了视觉信息处理的效率与准确性。

图 1 形象化展示了人类在看到一副图像时是如何高效分配有限的注意力资源的，其中红色区域表明视觉系统更关注的目标，很明显对于图 1 所示的场景，人们会把注意力更多投入到人的脸部，文本的标题以及文章首句等位置。

深度学习中的注意力机制从本质上讲和人类的选择性视觉注意力机制类似，核心目标也是从众多信息中选择出对当前任务目标更关键的信息。

Encoder-Decoder 框架

要了解深度学习中的注意力模型，就不得不先谈 Encoder-Decoder 框架，因为目前大多数注意力模型附着在 Encoder-Decoder 框架下，当然，其实注意力模型可以看作一种通用的思想，本身并不依赖于特定框架，这点需要注意。

Encoder-Decoder 框架可以看作是一种深度学习领域的研究模式，应用场景异常广泛。图 2 是文本处理领域里常用的 Encoder-Decoder 框架最抽象的一种表示。

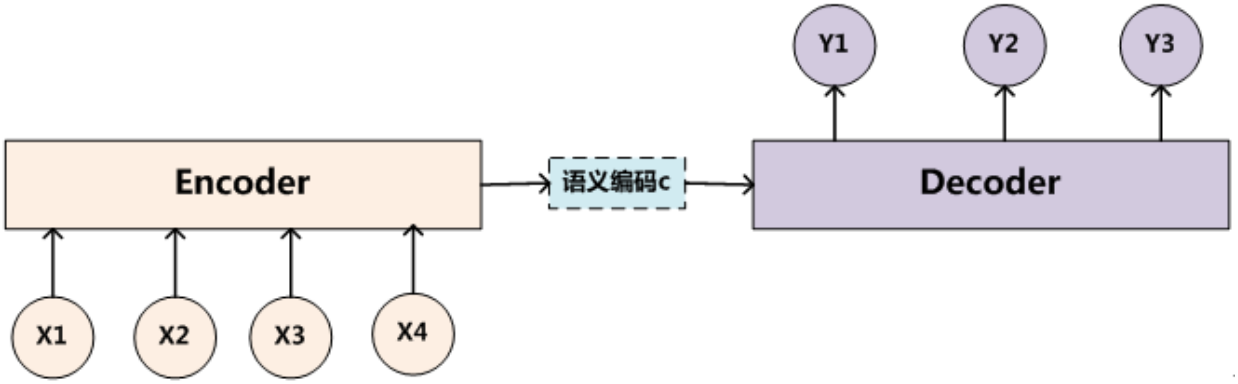

图 2 抽象的文本处理领域的 Encoder-Decoder 框架

文本处理领域的 Encoder-Decoder 框架可以这么直观地去理解：可以把它看作适合处理由一个句子（或篇章）生成另外一个句子（或篇章）的通用处理模型。对于句子对<Source,Target>，我们的目标是给定输入句子 Source，期待通过 Encoder-Decoder 框架来生成目标句子 Target。Source 和 Target 可以是同一种语言，也可以是两种不同的语言。而 Source 和 Target 分别由各自的单词序列构成：

$$\mathbf{Source} = \langle x_1, x_2 \ldots x_m \rangle$$
$$\mathbf{Target} = \langle y_1, y_2 \ldots y_n \rangle$$

Encoder 顾名思义就是对输入句子 Source 进行编码，将输入句子通过非线性变换转化为中间语义表示 C：

$$C = \mathcal{F}(x_1, x_2 \ldots x_m)$$

对于解码器 Decoder 来说，其任务是根据句子 Source 的中间语义表示 C 和之前已经生成的历史信息$y_1,y_2……y_{i-1}$来生成 i 时刻要生成的单词y_i：

$$y_i = G(C, y_1, y_2 \cdots y_{i-1})$$

每个 y_i 都依次这么产生，那么看起来就是整个系统根据输入句子 Source 生成了目标句子 Target。如果 Source 是中文句子，Target 是英文句子，那么这就是解决机器翻译问题的 Encoder-Decoder 框架；如果 Source 是一篇文章，Target 是概括性的几句描述语句，那么这是文本摘要的 Encoder-Decoder 框架；如果 Source 是一句问句，Target 是一句回答，那么这是问答系统或者对话机器人的 Encoder-Decoder 框架。由此可见，在文本处理领域，Encoder-Decoder 的应用领域相当广泛。

Encoder-Decoder 框架不仅仅在文本领域广泛使用，在语音识别、图像处理等领域也经常使用。比如对于语音识别来说，图 2 所示的框架完全适用，区别无非是 Encoder 部分的输入是语音流，输出是对应的文本信息；而对于“图像描述”任务来说，Encoder 部分的输入是一副图片，Decoder 的输出是能够描述图片语义内容的一句描述语。一般而言，文本处理和语音识别的 Encoder 部分通常采用 RNN 模型，图像处理的 Encoder 一般采用 CNN 模型。

Attention 模型

本节先以机器翻译作为例子讲解最常见的 Soft Attention 模型的基本原理，之后抛离 Encoder-Decoder 框架抽象出了注意力机制的本质思想，然后简单介绍最近广为使用的 Self Attention 的基本思路。

Soft Attention 模型

图 2 中展示的 Encoder-Decoder 框架是没有体现出“注意力模型”的，所以可以把它看作是注意力不集中的分心模型。为什么说它注意力不集中呢？请观察下目标句子 Target 中每个单词的生成过程如下：

$$y_1 = f(C)$$
$$y_2 = f(C, y_1)$$
$$y_3 = f(C, y_1, y_2)$$

其中 f 是 Decoder 的非线性变换函数。从这里可以看出，在生成目标句子的单词时，不论生成哪个单词，它们使用的输入句子 Source 的语义编码 C 都是一样的，没有任何区别。

而语义编码 C 是由句子 Source 的每个单词经过 Encoder 编码产生的，这意味着不论是生成哪个单词，y_1，y_2还是y_3，其实句子 Source 中任意单词对生成某个目标单词 y_i 来说影响力都是相同的，这是为何说这个模型没有体现出注意力的缘由。这类似于人类看到眼前的画面，但是眼中却没有注意焦点一样。

如果拿机器翻译来解释这个分心模型的 Encoder-Decoder 框架更好理解，比如输入的是英文句子：Tom chase Jerry，Encoder-Decoder 框架逐步生成中文单词：“汤姆”，“追逐”，“杰瑞”。

在翻译“杰瑞”这个中文单词的时候，分心模型里面的每个英文单词对于翻译目标单词“杰瑞”贡献是相同的，很明显这里不太合理，显然“Jerry”对于翻译成“杰瑞”更重要，但是分心模型是无法体现这一点的，这就是为何说它没有引入注意力的原因。

没有引入注意力的模型在输入句子比较短的时候问题不大，但是如果输入句子比较长，此时所有语义完全通过一个中间语义向量来表示，单词自身的信息已经消失，可想而知会丢失很多细节信息，这也是为何要引入注意力模型的重要原因。

上面的例子中，如果引入 Attention 模型的话，应该在翻译“杰瑞”的时候，体现出英文单词对于翻译当前中文单词不同的影响程度，比如给出类似下面一个概率分布值：

（Tom,0.3）(Chase,0.2) (Jerry,0.5)

每个英文单词的概率代表了翻译当前单词“杰瑞”时，注意力分配模型分配给不同英文单词的注意力大小。这对于正确翻译目标语单词肯定是有帮助的，因为引入了新的信息。

同理，目标句子中的每个单词都应该学会其对应的源语句子中单词的注意力分配概率信息。这意味着在生成每个单词y_i的时候，原先都是相同的中间语义表示 C 会被替换成根据当前生成单词而不断变化的C_i。理解 Attention 模型的关键就是这里，即由固定的中间语义表示 C 换成了根据当前输出单词来调整成加入注意力模型的变化的C_i。增加了注意力模型的 Encoder-Decoder 框架理解起来如图 3 所示。

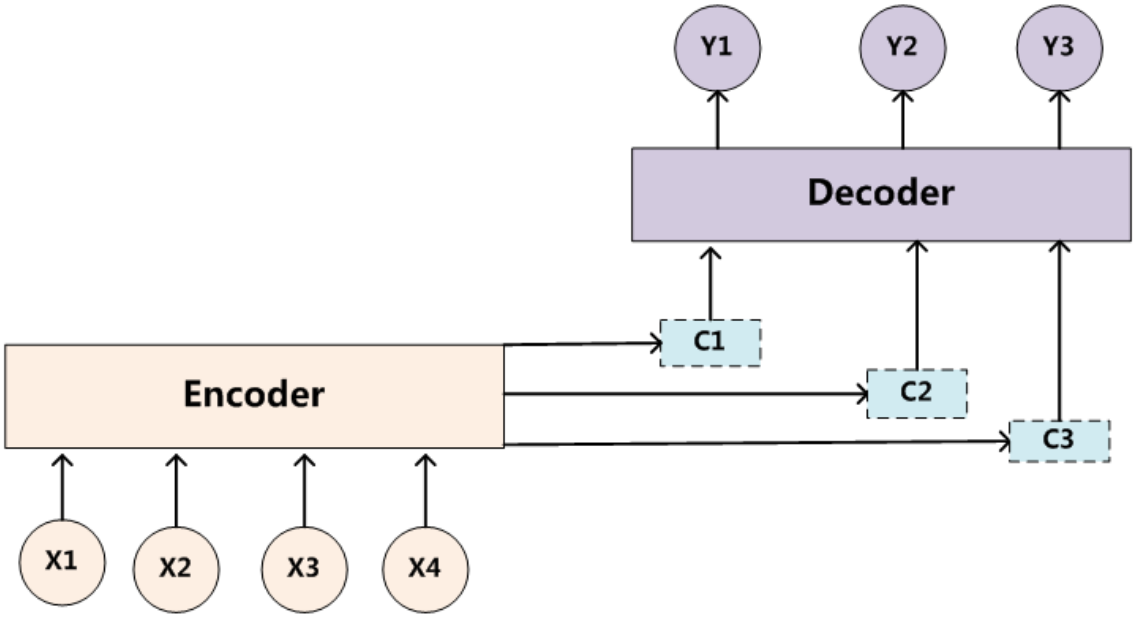

图 3 引入注意力模型的 Encoder-Decoder 框架

即生成目标句子单词的过程成了下面的形式：

$$y_1 = f1(C_1)$$
$$y_2 = f1(C_2, y_1)$$
$$y_3 = f1(C_3, y_1, y_2)$$

而每个C_i可能对应着不同的源语句子单词的注意力分配概率分布，比如对于上面的英汉翻译来说，其对应的信息可能如下：

$$C_{汤姆} = g(0.6 * f2("Tom"), 0.2 * f2(Chase), 0.2 * f2("Jerry"))$$

$$C_{追逐} = g(0.2 * f2("Tom"), 0.7 * f2(Chase), 0.1 * f2("Jerry"))$$

$$C_{杰瑞} = g(0.3 * f2("Tom"), 0.2 * f2(Chase), 0.5 * f2("Jerry"))$$

其中，f2 函数代表 Encoder 对输入英文单词的某种变换函数，比如如果 Encoder 是用的 RNN 模型的话，这个 f2 函数的结果往往是某个时刻输入 x_i 后隐层节点的状态值；g 代表 Encoder 根据单词的中间表示合成整个句子中间语义表示的变换函数，一般的做法中，g 函数就是对构成元素加权求和，即下列公式：

$$C_i = \sum_{j=1}^{L_x} a_{ij} h_j$$

其中，L_x 代表输入句子 Source 的长度，a_{ij} 代表在 Target 输出第 i 个单词时 Source 输入句子中第 j 个单词的注意力分配系数，而 h_j 则是 Source 输入句子中第 j 个单词的语义编码。假设 C_i 下标 i 就是上面例子所说的“ 汤姆” ，那么 L_x 就是 3，h1=f(“Tom”)，h2=f(“Chase”),h3=f(“Jerry”)分别是输入句子每个单词的语义编码，对应的注意力模型权值则分别是 0.6,0.2,0.2，所以 g 函数本质上就是个加权求和函数。如果形象表示的话，翻译中文单词“汤姆”的时候，数学公式对应的中间语义表示 C_i 的形成过程类似图 4。

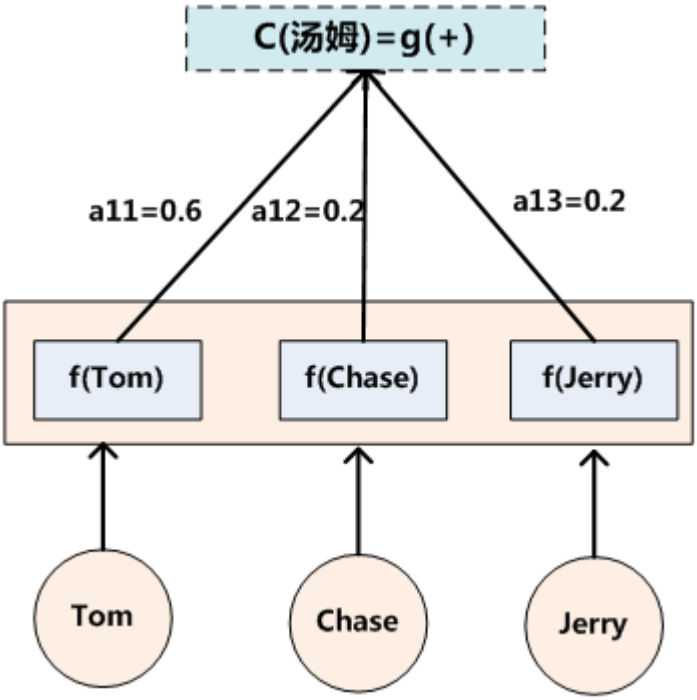

图 4 Attention 的形成过程

这里还有一个问题：生成目标句子某个单词，比如“汤姆”的时候，如何知道 Attention 模型所需要的输入句子单词注意力分配概率分布值呢？就是说“汤姆”对应的输入句子 Source 中各个单词的概率分布：(Tom,0.6)(Chase,0.2) (Jerry,0.2) 是如何得到的呢？

为了便于说明，我们假设对图 2 的非 Attention 模型的 Encoder-Decoder 框架进行细化，Encoder 采用 RNN 模型，Decoder 也采用 RNN 模型，这是比较常见的一种模型配置，则图 2 的框架转换为图 5。

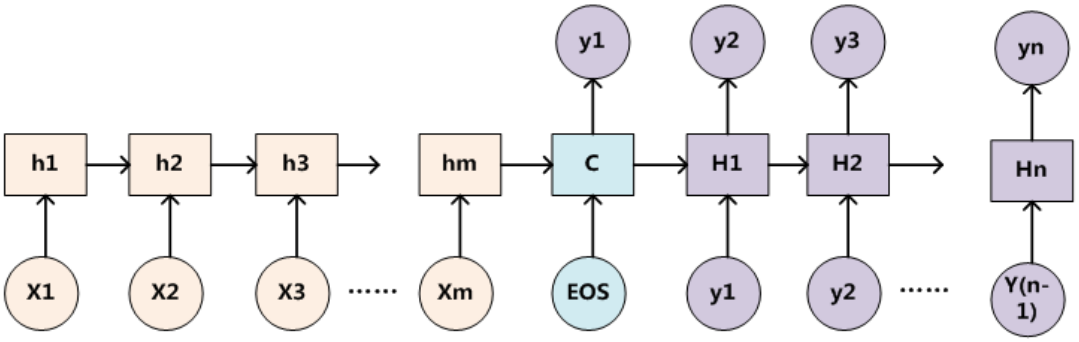

图 5 RNN 作为具体模型的 Encoder-Decoder 框架

那么用图 6 可以较为便捷地说明注意力分配概率分布值的通用计算过程。

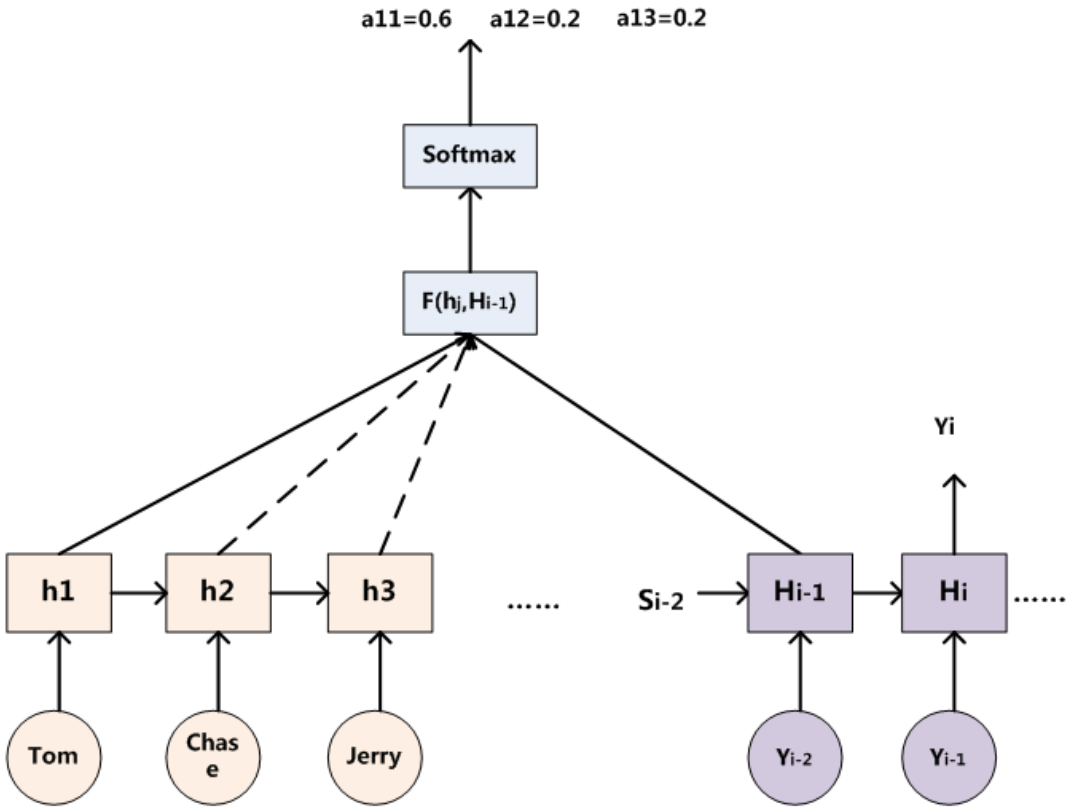

图 6 注意力分配概率计算

对于采用 RNN 的 Decoder 来说，在时刻 i，如果要生成 yi 单词，我们是可以知道 Target 在生成y_i之前的时刻 i-1 时，隐层节点 i-1 时刻的输出值H_{i-1}的，而我们的目的是要计算生成y_i时输入句子中的单词“Tom”、“Chase”、“Jerry”对y_i来说的注意力分配概率分布，那么可以用 Target 输出句子 i-1 时刻的隐层节点状态H_{i-1}去一一和输入句子 Source 中每个单词对应的 RNN 隐层节点状态 hj 进行对比，即通过函数 F(h_j, H_{i-1})来获得目标单词y_i和每个输入单词对应的对齐可能性，这个 F 函数在不同论文里可能会采取不同的方法，然后函数 F 的输出经过 Softmax 进行归一化就得到了符合概率分布取值区间的注意力分配概率分布数值。

绝大多数 Attention 模型都是采取上述的计算框架来计算注意力分配概率分布信息，区别只是在 F 的定义上可能有所不同。图 7 可视化地展示了在英语-德语翻译系统中加入 Attention 机制后，Source 和 Target 两个句子每个单词对应的注意力分配概率分布。

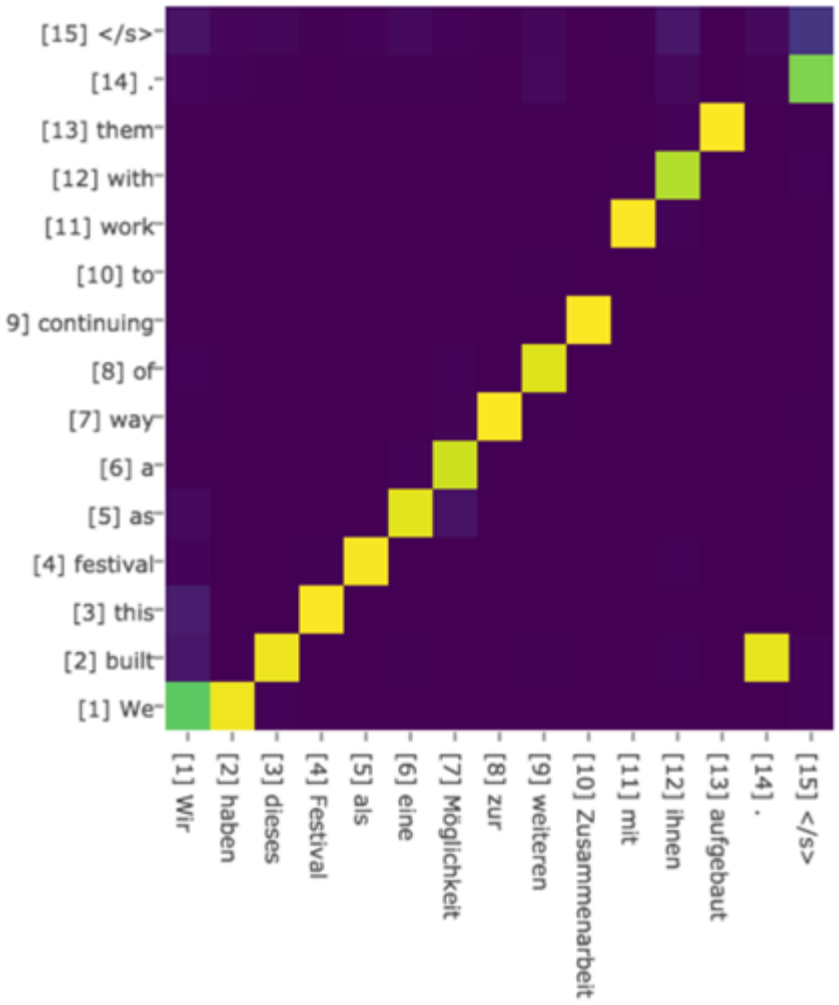

图 7 英语-德语翻译的注意力概率分布

上述内容就是经典的 Soft Attention 模型的基本思想，那么怎么理解 Attention 模型的物理含义呢？一般在自然语言处理应用里会把 Attention 模型看作是输出 Target 句子中某个单词和输入 Source 句子每个单词的对齐模型，这是非常有道理的。

目标句子生成的每个单词对应输入句子单词的概率分布可以理解为输入句子单词和这个目标生成单词的对齐概率，这在机器翻译语境下是非常直观的：传统的统计机器翻译一般在做的过程中会专门有一个短语对齐的步骤，而注意力模型其实起的是相同的作用。

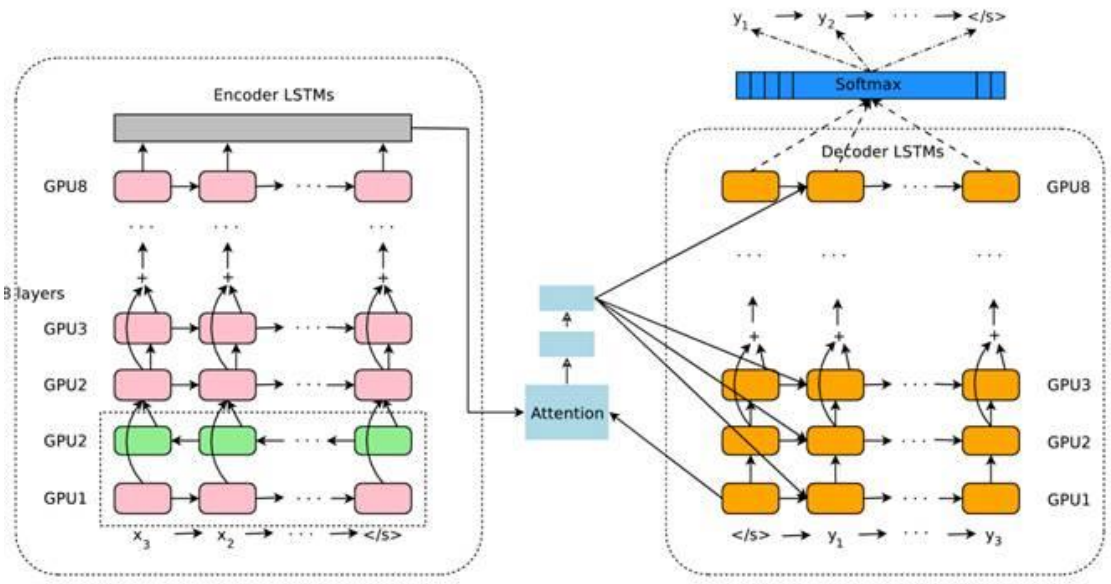

图 8 Google 神经网络机器翻译系统结构图

图 8 所示即为 Google 于 2016 年部署到线上的基于神经网络的机器翻译系统，相对传统模型翻译效果有大幅提升，翻译错误率降低了 60%，其架构就是上文所述的加上 Attention 机制的 Encoder-Decoder 框架，主要区别无非是其 Encoder 和 Decoder 使用了 8 层叠加的 LSTM 模型。

Attention 机制的本质思想

如果把 Attention 机制从上文讲述例子中的 Encoder-Decoder 框架中剥离，并进一步做抽象，可以更容易看懂 Attention 机制的本质思想。

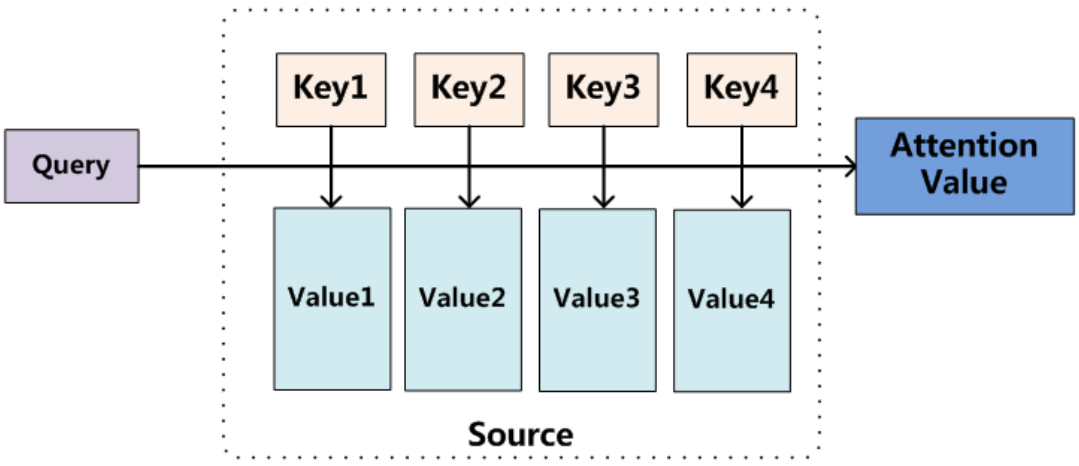

图 9 Attention 机制的本质思想

我们可以这样来看待 Attention 机制（参考图 9）：将 Source 中的构成元素想象成是由一系列的<Key,Value>数据对构成，此时给定 Target 中的某个元素 Query，通过计算 Query 和各个 Key 的相似性或者相关性，得到每个 Key 对应 Value 的权重系数，然后对 Value 进行加权求和，即得到了最终的 Attention 数值。所以本质上 Attention 机制是对 Source 中元素的 Value 值进行加权求和，而 Query 和 Key 用来计算对应 Value 的权重系数。即可以将其本质思想改写为如下公式：

$$\mathbf{Attention(Query, Source)} = \sum_{i=1}^{L_x} \boldsymbol{Similarity(Query, Key_i)} * \boldsymbol{Value_i}$$

其中，L_x=||Source||代表 Source 的长度，公式含义即如上所述。上文所举的机器翻译的例子里，因为在计算 Attention 的过程中，Source 中的 Key 和 Value 合二为一，指向的是同一个东西，也即输入句子中每个单词对应的语义编码，所以可能不容易看出这种能够体现本质思想的结构。

当然，从概念上理解，把 Attention 仍然理解为从大量信息中有选择地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，这种思路仍然成立。聚焦的过程体现在权重系数的计算上，权重越大越聚焦于其对应的 Value 值上，即权重代表了信息的重要性，而 Value 是其对应的信息。

从图 9 可以引出另外一种理解，也可以将 Attention 机制看作一种软寻址（Soft Addressing）:Source 可以看作存储器内存储的内容，元素由地址 Key 和值 Value 组成，当前有个 Key=Query 的查询，目的是取出存储器中对应的 Value 值，即 Attention 数值。通过 Query 和存储器内元素 Key 的地址进行相似性比较来寻址，之所以说是软寻址，指的不像一般寻址只从存储内容里面找出一条内容，而是可能从每个 Key 地址都会取出内容，取出内容的重要性根据 Query 和 Key 的相似性来决定，之后对 Value 进行加权求和，这样就可以取出最终的 Value 值，也即 Attention 值。所以不少研究人员将 Attention 机制看作软寻址的一种特例，这也是非常有道理的。

至于 Attention 机制的具体计算过程，如果对目前大多数方法进行抽象的话，可以将其归纳为两个过程：第一个过程是根据 Query 和 Key 计算权重系数，第二个过程根据权重系数对 Value 进行加权求和。而第一个过程又可以细分为两个阶段：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 10 展示的三个阶段。

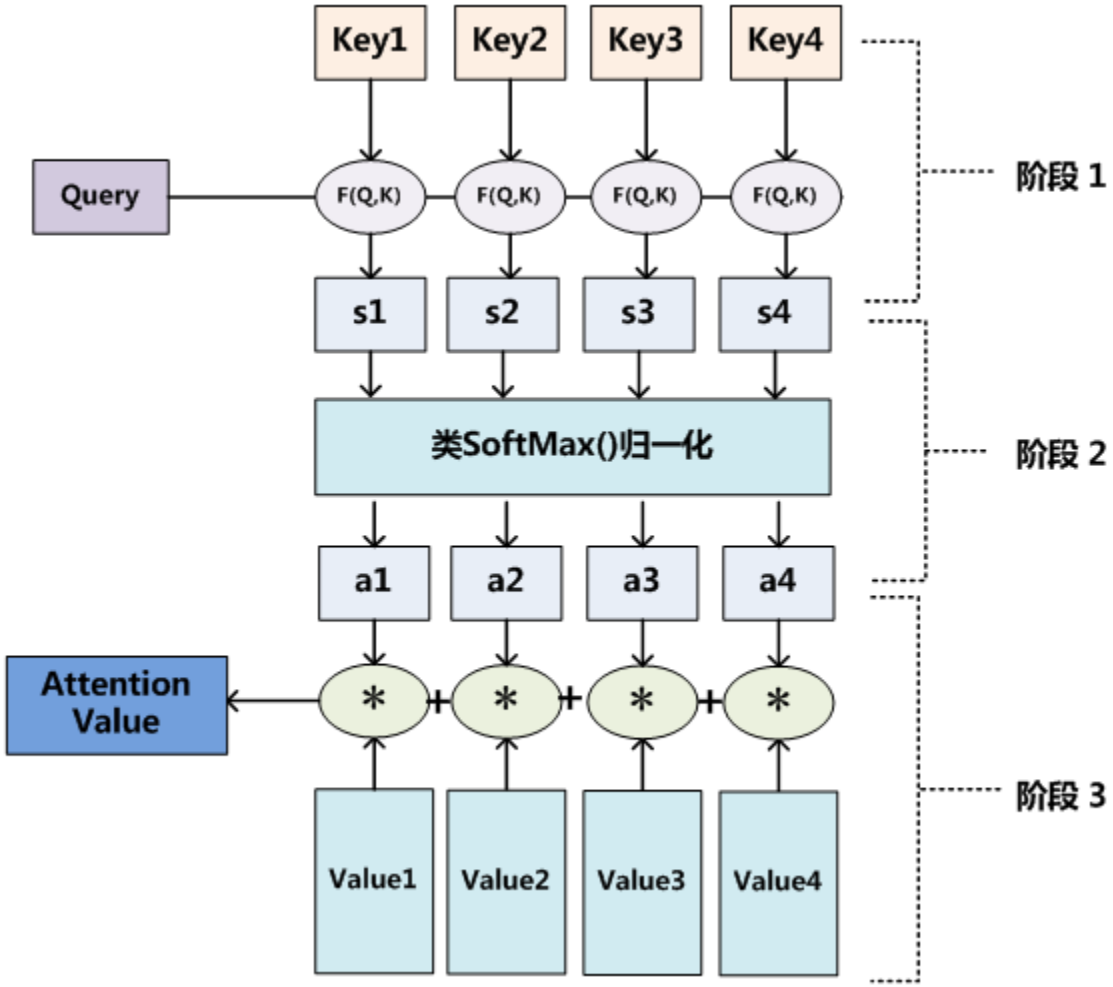

图 10 三阶段计算 Attention 过程

在第一个阶段，可以引入不同的函数和计算机制，根据 Query 和某个Key_i，计算两者的相似性或者相关性，最常见的方法包括：求两者的向量点积、求两者的向量 Cosine 相似性或者通过再引入额外的神经网络来求值，即如下方式：

点积： $\mathbf{Similarity}(\mathbf{Query}, \boldsymbol{Key_i}) = \mathbf{Query} \cdot \boldsymbol{Key_i}$

Cosine 相似性： $\mathbf{Similarity}(\mathbf{Query}, \boldsymbol{Key_i}) = \frac{\mathbf{Query} \cdot \boldsymbol{Key_i}}{\|\boldsymbol{Query}\| \cdot \|\boldsymbol{Key_i}\|}$

MLP 网络： $\mathbf{Similarity}(\mathbf{Query}, \boldsymbol{Key_i}) = \mathbf{MLP}(\mathbf{Query}, \boldsymbol{Key_i})$

第一阶段产生的分值根据具体产生的方法不同其数值取值范围也不一样，第二阶段引入类似 SoftMax 的计算方式对第一阶段的得分进行数值转换，一方面可以进行归一化，将原始计算分值整理成所有元素权重之和为 1 的概率分布；另一方面也可以通过 SoftMax 的内在机制更加突出重要元素的权重。即一般采用如下公式计算：

$$a_i = Softmax(Sim_i) = \frac{e^{Sim_i}}{\sum_{j=1}^{L_x} e^{Sim_j}}$$

第二阶段的计算结果 a_i 即为 $Value_i$ 对应的权重系数，然后进行加权求和即可得到 Attention 数值：

$$\mathbf{Attention(Query, Source)} = \sum_{i=1}^{L_x} a_i \cdot Value_i$$

通过如上三个阶段的计算，即可求出针对 Query 的 Attention 数值，目前绝大多数具体的注意力机制计算方法都符合上述的三阶段抽象计算过程。

Self Attention 模型

通过上述对 Attention 本质思想的梳理，我们可以更容易理解本节介绍的 Self Attention 模型。Self Attention 也经常被称为 intra Attention（内部 Attention），最近一年也获得了比较广泛的使用，比如 Google 最新的机器翻译模型内部大量采用了 Self Attention 模型。

在一般任务的 Encoder-Decoder 框架中，输入 Source 和输出 Target 内容是不一样的，比如对于英-中机器翻译来说，Source 是英文句子，Target 是对应的翻译出的中文句子，Attention 机制发生在 Target 的元素 Query 和 Source 中的所有元素之间。而 Self Attention 顾名思义，指的不是 Target 和 Source 之间的 Attention 机制，而是 Source 内部元素之间或者 Target 内部元素之间发生的 Attention 机制，也可以理解为 Target=Source 这种特殊情况下的注意

力计算机制。其具体计算过程是一样的，只是计算对象发生了变化而已，所以此处不再赘述其计算过程细节。

如果是常规的 Target 不等于 Source 情形下的注意力计算，其物理含义正如上文所讲，比如对于机器翻译来说，本质上是目标语单词和源语单词之间的一种单词对齐机制。那么如果是 Self Attention 机制，一个很自然的问题是：通过 Self Attention 到底学到了哪些规律或者抽取出了哪些特征呢？或者说引入 Self Attention 有什么增益或者好处呢？我们仍然以机器翻译中的 Self Attention 来说明，图 11 和图 12 是可视化地表示 Self Attention 在同一个英语句子内单词间产生的联系。

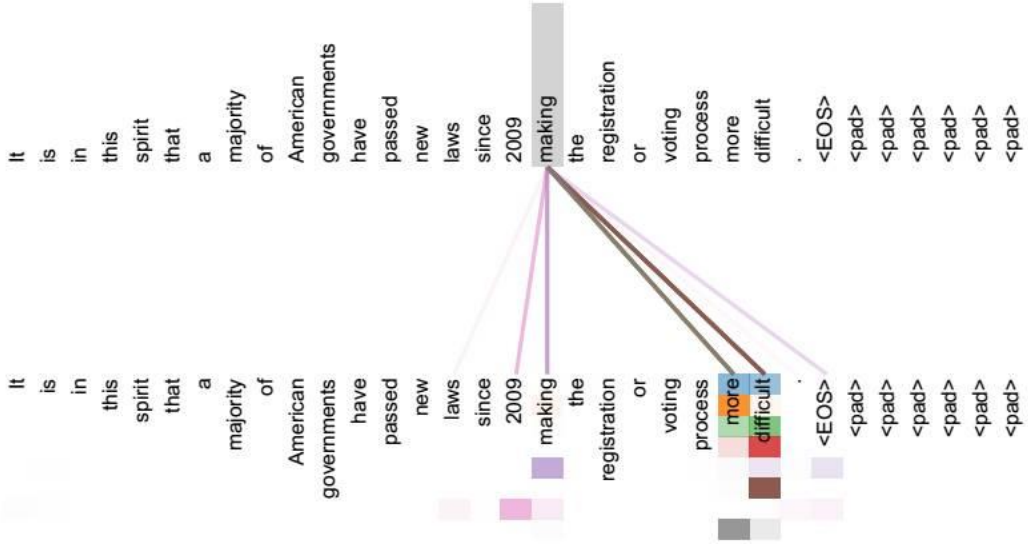

图 11 可视化 Self Attention 实例

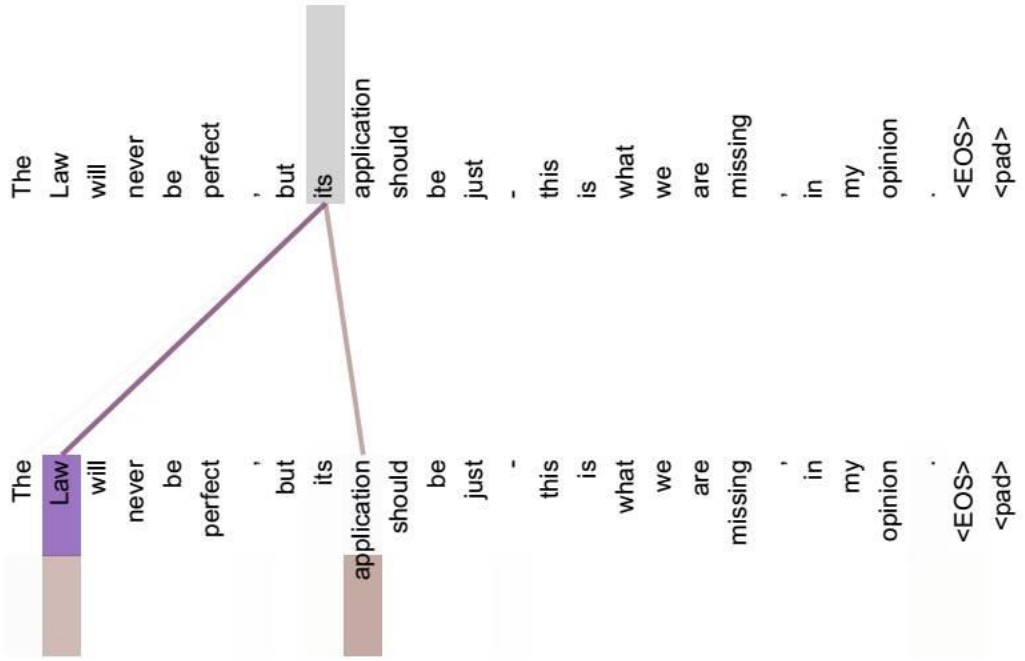

图 12 可视化 Self Attention 实例

从两张图（图 11、图 12）可以看出，Self Attention 可以捕获同一个句子中单词之间的一些句法特征（比如图 11 展示的有一定距离的短语结构）或者语义特征（比如图 12 展示的 its 的指代对象 Law）。

很明显，引入 Self Attention 后会更容易捕获句子中长距离的相互依赖的特征，因为如果是 RNN 或者 LSTM，需要依次序序列计算，对于远距离的相互依赖的特征，要经过若干时间步步骤的信息累积才能将两者联系起来，而距离越远，有效捕获的可能性越小。

但是 Self Attention 在计算过程中会直接将句子中任意两个单词的联系通过一个计算步骤直接联系起来，所以远距离依赖特征之间的距离被极大缩短，有利于有效地利用这些特征。除此外，Self Attention 对于增加计算的并行性也有直接帮助作用。这是为何 Self Attention 逐渐被广泛使用的主要原因。

Attention 机制的应用

前文有述，Attention 机制在深度学习的各种应用领域都有广泛的使用场景。上文在介绍过程中我们主要以自然语言处理中的机器翻译任务作为例子，下面分别再从图像处理领域和语音识别选择典型应用实例来对其应用做简单说明。

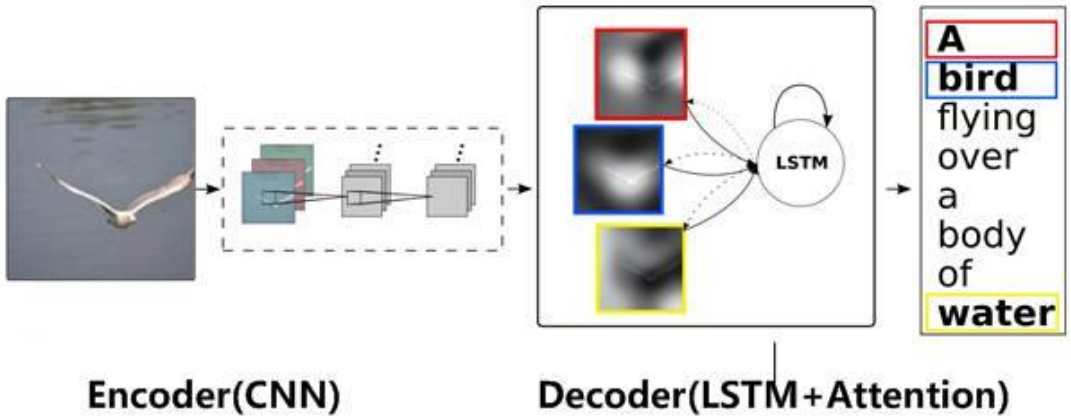

图 13 图片-描述任务的 Encoder-Decoder 框架

图片描述（Image-Caption）是一种典型的图文结合的深度学习应用，输入一张图片，人工智能系统输出一句描述句子，语义等价地描述图片所示内容。很明显这种应用场景也可以使用 Encoder-Decoder 框架来解决任务目标，此时 Encoder 输入部分是一张图片，一般会用 CNN 来对图片进行特征抽取，Decoder 部分使用 RNN 或者 LSTM 来输出自然语言句子（参考图 13）。

此时如果加入 Attention 机制能够明显改善系统输出效果，Attention 模型在这里起到了类似人类视觉选择性注意的机制，在输出某个实体单词的时候会将注意力焦点聚焦在图片中相应的区域上。图 14 给出了根据给定图片生成句子“A person is standing on a beach with a surfboard.”过程时每个单词对应图片中的注意力聚焦区域。

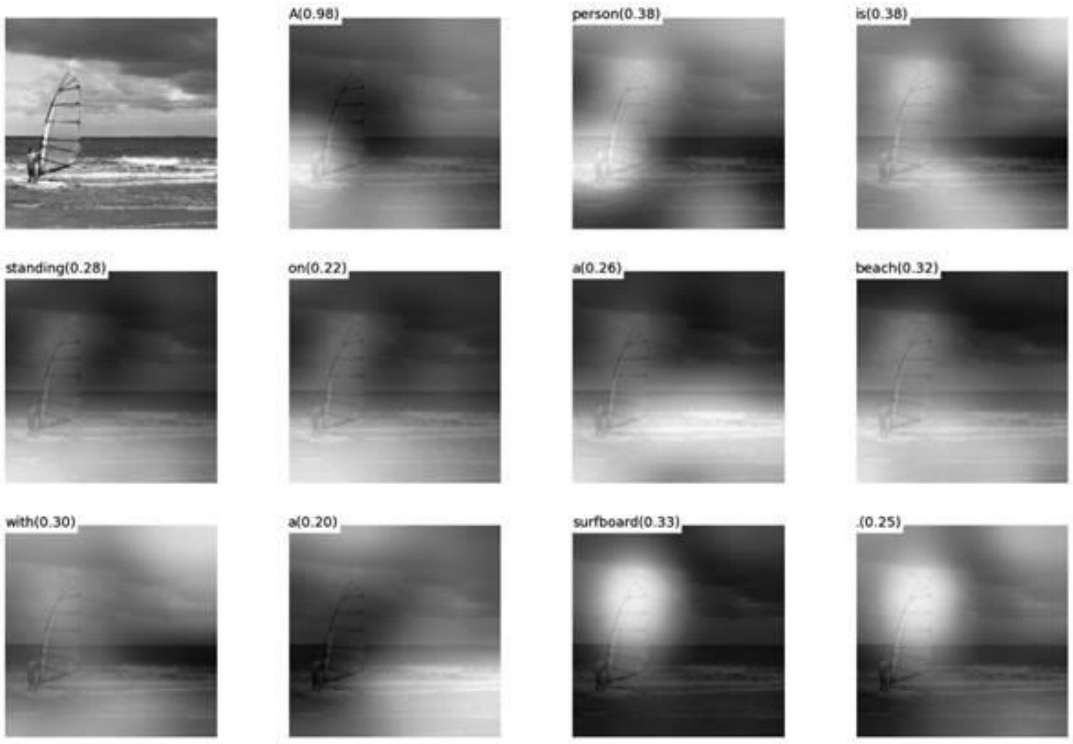

图 14 图片生成句子中每个单词时的注意力聚焦区域

图 15 给出了另外四个例子形象地展示了这种过程，每个例子上方左侧是输入的原图，下方句子是人工智能系统自动产生的描述语句，上方右侧图展示了当 AI 系统产生语句中划横线单词的时候，对应图片中聚焦的位置区域。比如当输出单词 dog 的时候，AI 系统会将注意力更多地分配给图片中小狗对应的位置。

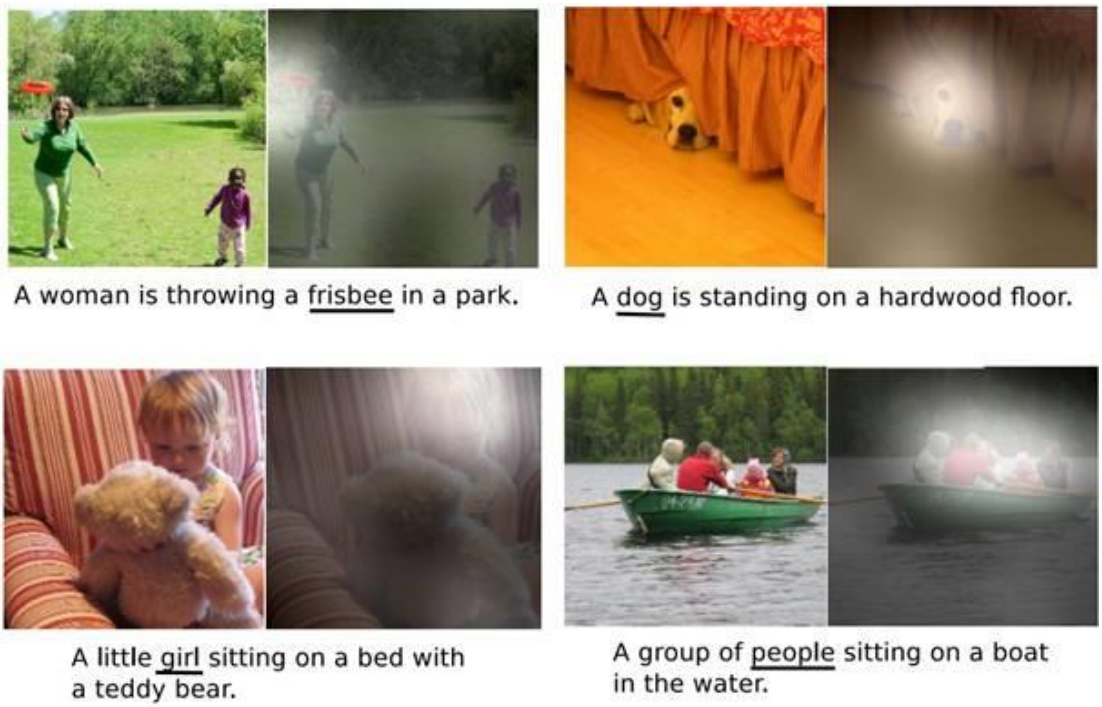

图 15 图像描述任务中 Attention 机制的聚焦作用

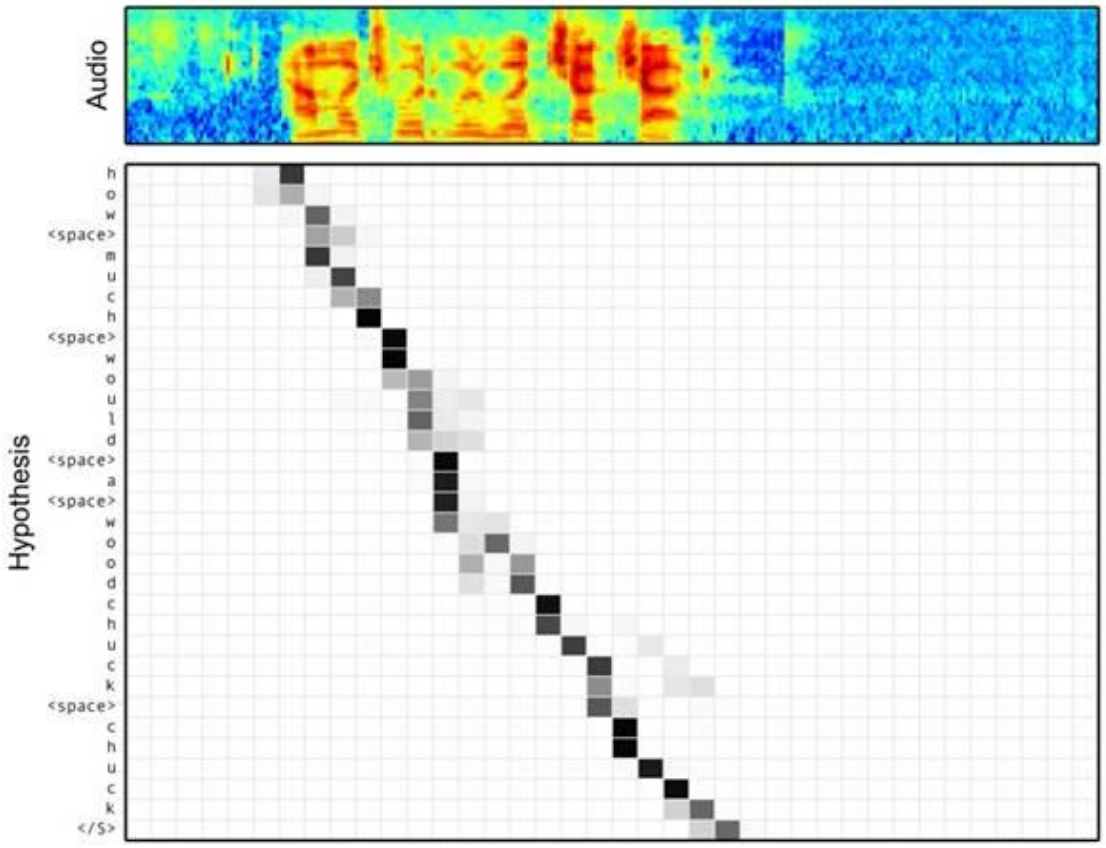

图 16 语音识别中音频序列和输出字符之间的 Attention

语音识别的任务目标是将语音流信号转换成文字，所以也是 Encoder-Decoder 的典型应用场景。Encoder 部分的 Source 输入是语音流信号，Decoder 部分输出语音对应的字符串流。

图 16 可视化地展示了在 Encoder-Decoder 框架中加入 Attention 机制后，当用户用语音说句子 how much would a woodchuck chuck 时，输入部分的声音特征信号和输出字符之间的注意力分配概率分布情况，颜色越深代表分配到的注意力概率越高。从图中可以看出，在这个场景下，Attention 机制起到了将输出字符和输入语音信号进行对齐的功能。

上述内容仅仅选取了不同 AI 领域的几个典型 Attention 机制应用实例，Encoder-Decoder 加 Attention 架构由于其卓越的实际效果，目前在深度学习领域里得到了广泛的使用，了解并熟练使用这一架构对于解决实际问题会有极大帮助。