Image Processing and Recognition

Dr. Călin-Adrian POPA

Lecture 9

December 6th, 2022

5 Attention mechanisms

- the optic nerve of our visual system receives massive sensory input, far exceeding what the brain can fully process
- fortunately, not all stimuli are equal
- focalization and concentration of consciousness have enabled us to direct attention to objects of interest, such as preys and predators, in the complex visual environment
- the ability of paying attention to only a small fraction of the information has evolutionary significance, allowing human beings to live and succeed

5.1 Attention cues

- information in our environment is not scarce, attention is
- when inspecting a visual scene, our optic nerve receives information at the order of 10^8 bits per second, far exceeding what our brain can fully process
- fortunately, we learned from experience (also known as data) that *not all sensory inputs are equal*
- throughout human history, the capability of directing attention to only a fraction of information of interest has enabled our brain to allocate resources more smartly to survive, to grow, and to socialize, such as detecting predators, preys, and mates

5.1 Attention cues

- to explain how our attention is deployed in the visual world, a two-component framework has emerged
- in this framework, subjects selectively direct the spotlight of attention using both the *nonvolitional cue* and the *volitional cue*
- the nonvolitional cue is based on the *saliency* of objects in the environment
- imagine there are five objects in front of us: a newspaper, a research paper, a cup of coffee, a notebook, and a book, such as in Figure 1
- while all the paper products are printed in black and white, the coffee cup is red
- in other words, this coffee is intrinsically salient in this visual environment, automatically and involuntarily drawing attention
- so, we bring the fovea (the center of the macula, where visual acuity is highest) onto the coffee, as shown in Figure 1

5.1 Attention cues

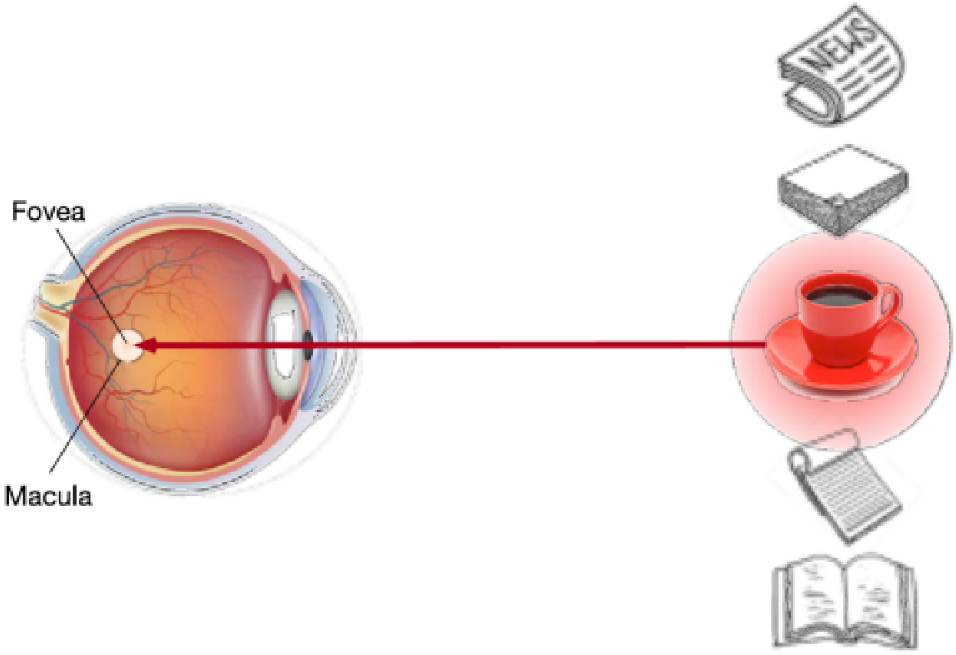

Figure 1: Using the nonvolitional cue based on saliency (red cup, non-paper), attention is involuntarily directed to the coffee.

- after drinking coffee, we become caffeinated, and want to read a book
- so we turn our head, refocus our eyes, and look at the book, as depicted in Figure 2
- different from the case in Figure 1, where the coffee biases us towards selecting based on saliency, in this task-dependent case, we select the book under cognitive and volitional control
- using the volitional cue based on variable selection criteria, this form of attention is more deliberate
- it is also more powerful, with the subject's voluntary effort

5.1 Attention cues

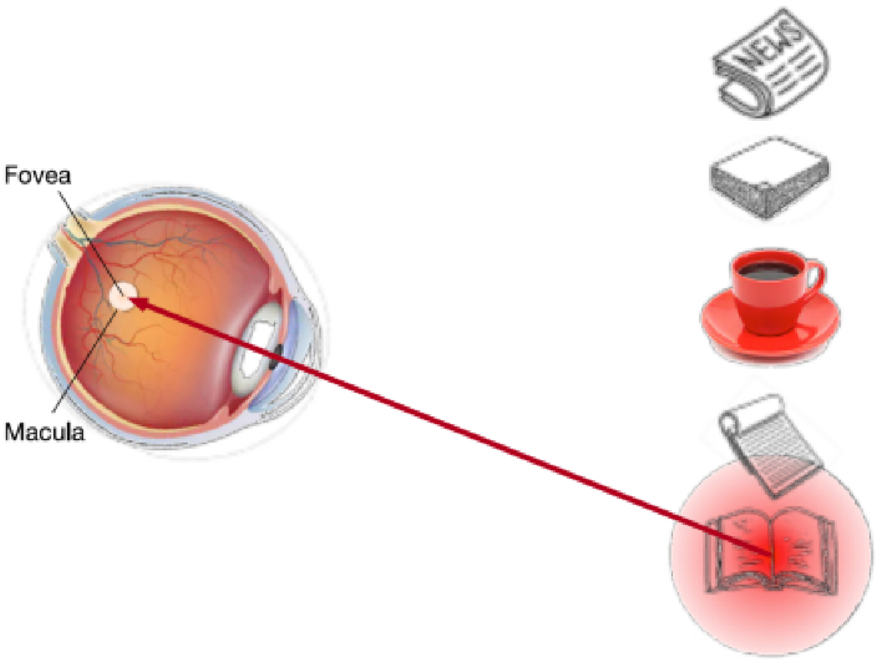

Figure 2: Using the volitional cue (want to read a book) that is task-dependent, attention is directed to the book under volitional control.

5.1 Attention cues

- inspired by the nonvolitional and volitional attention cues that explain the attentional deployment, in the following we will describe a framework for designing attention mechanisms by incorporating these two attention cues
- to begin with, consider the simpler case where only nonvolitional cues are available
- to bias selection over sensory inputs, we can simply use a parameterized fully-connected layer or even a non-parameterized max or average pooling layer

- therefore, what sets attention mechanisms apart from those fully-connected layers or pooling layers is the inclusion of the volitional cues
- in the context of attention mechanisms, we refer to volitional cues as *queries*
- given any query, attention mechanisms bias selection over sensory inputs (e.g., intermediate feature representations) via *attention pooling*
- these sensory inputs are called *values*, in the context of attention mechanisms
- more generally, every value is paired with a *key*, which can be thought of as the nonvolitional cue of that sensory input
- as shown in Figure 3, we can design attention pooling so that the given query (volitional cue) can interact with keys (nonvolitional cues), which guides bias selection over values (sensory inputs)

5.1 Attention cues

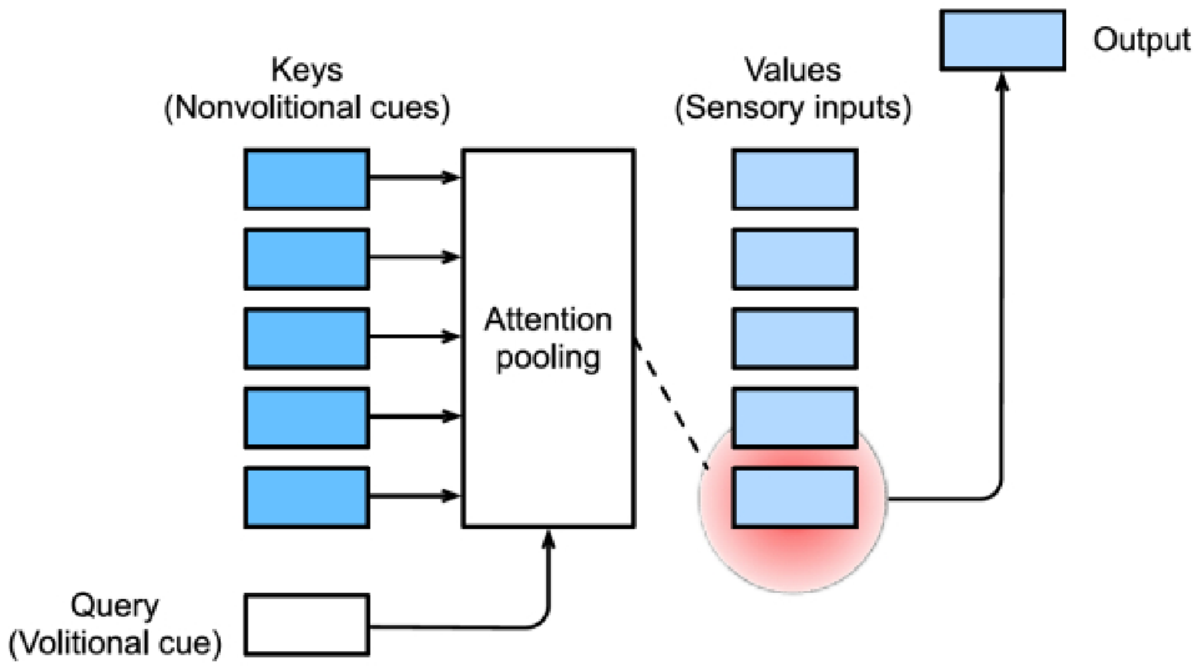

Figure 3: Attention mechanisms bias selection over values (sensory inputs) via attention pooling, which incorporates queries (volitional cues) and keys (nonvolitional cues).

5.2 Attention pooling: Nadaraya-Watson kernel regression

- now we know the major components of attention mechanisms under the framework in Figure 3
- to recapitulate, the interactions between queries (volitional cues) and keys (nonvolitional cues) result in *attention pooling*
- the attention pooling selectively aggregates values (sensory inputs) to produce the output
- in this section, we will describe attention pooling in greater detail, to give a high-level view of how attention mechanisms work in practice
- specifically, the Nadaraya-Watson kernel regression model, proposed in 1964, is a simple yet complete example for demonstrating machine learning with attention mechanisms

5.2 Attention pooling: Nadaraya-Watson kernel regression

- to keep things simple, let us consider the following regression problem: given a dataset of input-output pairs $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, how to learn f to predict the output $\hat{y} = f(x)$ for any new input x?
- we begin with perhaps the world's "dumbest" estimator for this regression problem: using average pooling to average over all the training outputs:

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

- obviously, average pooling omits the inputs x_i
- a better idea was proposed by Nadaraya and Watson, namely to weigh the outputs y_i according to their input locations:

$$f(x) = \sum_{i=1}^{n} \frac{K(x - x_i)}{\sum_{j=1}^{n} K(x - x_j)} y_i, \tag{2}$$

  where K is a *kernel*
- the estimator in (2) is called *Nadaraya-Watson kernel regression*; here, we will not dive into details of kernels

5.2 Attention pooling: Nadaraya-Watson kernel regression

- recall the framework of attention mechanisms in Figure 3; from the perspective of attention, we can rewrite (2) in a more generalized form of *attention pooling*:

$$f(x) = \sum_{i=1}^{n} \alpha(x, x_i) y_i, \tag{3}$$

  where x is the *query* and (x_i, y_i) is the *key-value* pair
- comparing (3) and (1), the attention pooling here is a weighted average of values y_i
- the *attention weight* $\alpha(x, x_i)$ in (3) is assigned to the corresponding value y_i based on the interaction between the query x and the key x_i, modeled by α
- for any query, its attention weights over all the key-value pairs are a valid probability distribution: they are non-negative and sum up to one

5.2 Attention pooling: Nadaraya-Watson kernel regression

- to gain intuitions of attention pooling, we can consider a Gaussian kernel defined as:

$$K(u) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u^2}{2}\right).$$

- plugging the Gaussian kernel into (3) and (2) gives:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} \alpha(x, x_i) y_i \\ &= \sum_{i=1}^{n} \frac{\exp\left(-\frac{1}{2}(x - x_i)^2\right)}{\sum_{j=1}^{n} \exp\left(-\frac{1}{2}(x - x_j)^2\right)} y_i \\ &= \sum_{i=1}^{n} \operatorname{softmax}\left(-\frac{1}{2}(x - x_i)^2\right) y_i. \end{aligned} \tag{4}$$

- in (4), a key x_i that is closer to the given query x will get *more attention* via a *larger attention weight* assigned to the key's corresponding value y_i

5.2 Attention pooling: Nadaraya-Watson kernel regression

- notably, Nadaraya-Watson kernel regression is a nonparametric model; thus (4) is an example of *nonparametric attention pooling*
- nonparametric Nadaraya-Watson kernel regression enjoys the *consistency* benefit: given enough data, this model converges to the optimal solution
- nonetheless, we can easily integrate learnable parameters into attention pooling
- as an example, slightly different from (4), in the following, the distance between the query x and the key x_i is multiplied by a learnable parameter w:

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

5.3 Attention scoring functions

- in the previous section, we used a Gaussian kernel to model interactions between queries and keys
- treating the exponent of the Gaussian kernel in (4) as an *attention scoring function* (or *scoring function*, for short), the results of this function were essentially fed into a softmax operation
- as a result, we obtained a probability distribution (attention weights) over values that are paired with keys
- in the end, the output of the attention pooling is simply a weighted sum of the values based on these attention weights

- at a high level, we can use the above algorithm to instantiate the framework of attention mechanisms in Figure 3
- denoting an attention scoring function by a, Figure 4 illustrates how the output of attention pooling can be computed as a weighted sum of values
- since attention weights are a probability distribution, the weighted sum is essentially a weighted average

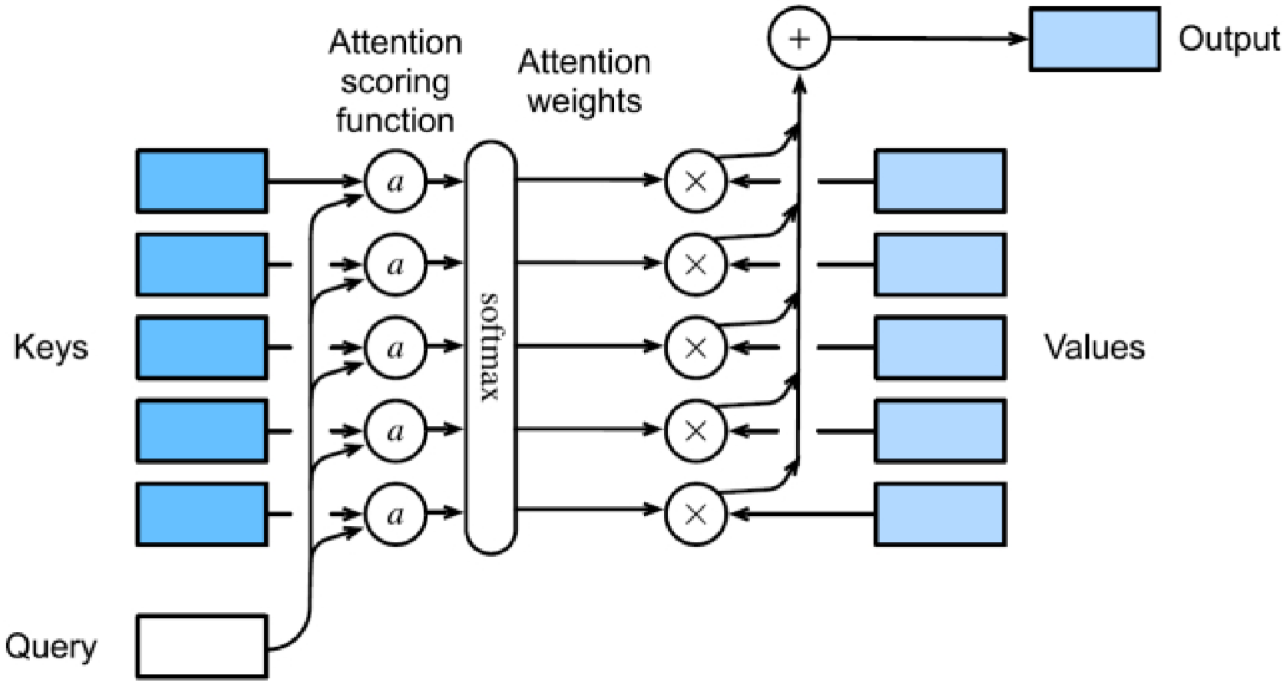

Figure 4: Computing the output of attention pooling as a weighted average of values.

5.3 Attention scoring functions

- mathematically, suppose that we have a query $\boldsymbol{q} \in \mathbb{R}^q$ and m key-value pairs $(\boldsymbol{k}_1, \boldsymbol{v}_1), \ldots, (\boldsymbol{k}_m, \boldsymbol{v}_m)$, where any $\boldsymbol{k}_i \in \mathbb{R}^k$ and any $\boldsymbol{v}_i \in \mathbb{R}^v$
- the attention pooling f is instantiated as a weighted sum of the values:

$$f(\boldsymbol{q}, (\boldsymbol{k}_1, \boldsymbol{v}_1), \ldots, (\boldsymbol{k}_m, \boldsymbol{v}_m)) = \sum_{i=1}^{m} \alpha(\boldsymbol{q}, \boldsymbol{k}_i)\boldsymbol{v}_i \in \mathbb{R}^v, \tag{5}$$

  where the attention weight (scalar) for the query $\boldsymbol{q}$ and key $\boldsymbol{k}_i$ is computed by the softmax operation of an attention scoring function a that maps two vectors to a scalar:

$$\alpha(\boldsymbol{q}, \boldsymbol{k}_i) = \text{softmax}(a(\boldsymbol{q}, \boldsymbol{k}_i)) = \frac{\exp(a(\boldsymbol{q}, \boldsymbol{k}_i))}{\sum_{j=1}^{m} \exp(a(\boldsymbol{q}, \boldsymbol{k}_j))} \in \mathbb{R}. \tag{6}$$

- as we can see, different choices of the attention scoring function a lead to different behaviors of attention pooling
- in this section, we introduce two popular scoring functions that we will use to develop more sophisticated attention mechanisms later

5.3 Attention scoring functions

- in general, when queries and keys are vectors of different lengths, we can use additive attention as the scoring function
- given a query $\boldsymbol{q} \in \mathbb{R}^q$ and a key $\boldsymbol{k} \in \mathbb{R}^k$, the *additive attention* scoring function is defined as:

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{w}_v^\top \tanh(\boldsymbol{W}_q \boldsymbol{q} + \boldsymbol{W}_k \boldsymbol{k}) \in \mathbb{R}, \tag{7}$$

  where the parameters $\boldsymbol{W}_q \in \mathbb{R}^{h \times q}$, $\boldsymbol{W}_k \in \mathbb{R}^{h \times k}$, and $\boldsymbol{w}_v \in \mathbb{R}^h$ are learnable
- equivalent to (7), the query and the key are concatenated and fed into an MLP with a single hidden layer whose number of hidden units is h, a hyperparameter, and is implemented by using tanh as the activation function and disabling bias terms

- a more computationally efficient design for the scoring function can be simply the *dot product*
- however, the dot product operation requires that both the query and the key have the same vector length, say d
- assume that all the elements of the query and the key are independent random variables with zero mean and unit variance
- the dot product of both vectors has zero mean and a variance of d

- to ensure that the variance of the dot product still remains one, regardless of vector length, the *scaled dot-product attention* scoring function:

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^\top \boldsymbol{k} / \sqrt{d}$$

  divides the dot product by $\sqrt{d}$
- in practice, we often think in mini-batches for efficiency, such as computing attention for n queries and m key-value pairs, where queries and keys are of length d, and values are of length v
- the scaled dot-product attention of queries $\boldsymbol{Q} \in \mathbb{R}^{n \times d}$, keys $\boldsymbol{K} \in \mathbb{R}^{m \times d}$, and values $\boldsymbol{V} \in \mathbb{R}^{m \times v}$ is:

$$\mathrm{softmax}\left(\frac{\boldsymbol{Q}^\top \boldsymbol{K}}{\sqrt{d}}\right) \boldsymbol{V} \in \mathbb{R}^{n \times v}. \tag{8}$$

- we studied the machine translation problem in Section 4.6, where we designed an encoder-decoder architecture, based on two RNNs, for sequence to sequence learning
- specifically, the RNN encoder transforms a variable-length sequence into a fixed-shape context variable, then the RNN decoder generates the output (target) sequence token by token, based on the generated tokens and the context variable
- however, even though not all the input (source) tokens are useful for decoding a certain token, the *same* context variable that encodes the entire input sequence is still used at each decoding step

5.4 Bahdanau attention

- in a separate but related challenge of handwriting generation for a given text sequence, Graves designed a differentiable attention model to align text characters with the much longer pen trace, where the alignment moves only in one direction
- inspired by the idea of learning to align, Bahdanau et al. proposed a differentiable attention model without the severe unidirectional alignment limitation
- when predicting a token, if not all the input tokens are relevant, the model aligns (or attends) only to parts of the input sequence that are relevant to the current prediction
- this is achieved by treating the context variable as an output of attention pooling

- when describing Bahdanau attention for the RNN encoder-decoder below, we will follow the same notation in Section 4.6
- the new attention-based model is the same as that in Section 4.6, except that the context variable $\boldsymbol{c}$ is replaced by $\boldsymbol{c}_{t'}$ at any decoding time step t'
- suppose that there are T tokens in the input sequence, then the context variable at the decoding time step t' is the output of attention pooling:

$$\boldsymbol{c}_{t'} = \sum_{t=1}^{T} \alpha(\boldsymbol{s}_{t'-1}, \boldsymbol{h}_t)\boldsymbol{h}_t,$$

  where the decoder hidden state $\boldsymbol{s}_{t'-1}$ at time step $t'-1$ is the query, and the encoder hidden states $\boldsymbol{h}_t$ are both the keys and values, and the attention weight α is computed as in (6), using the additive attention scoring function defined by (7)
- slightly different from the vanilla RNN encoder-decoder architecture, the same architecture with Bahdanau attention is depicted in Figure 5

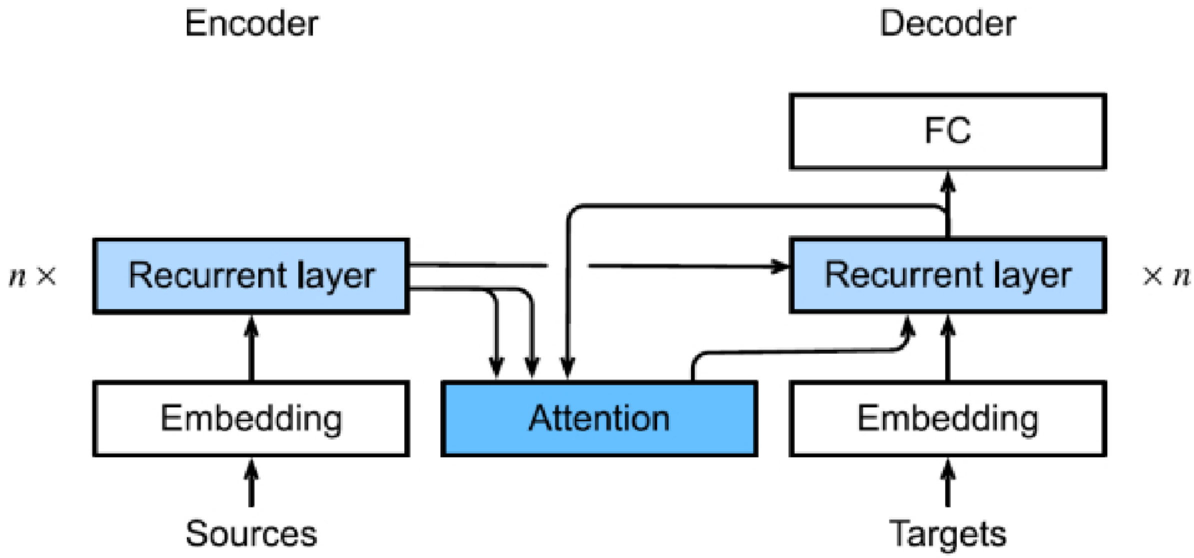

Figure 5: Layers in an RNN encoder-decoder model with Bahdanau attention.

Thank you!