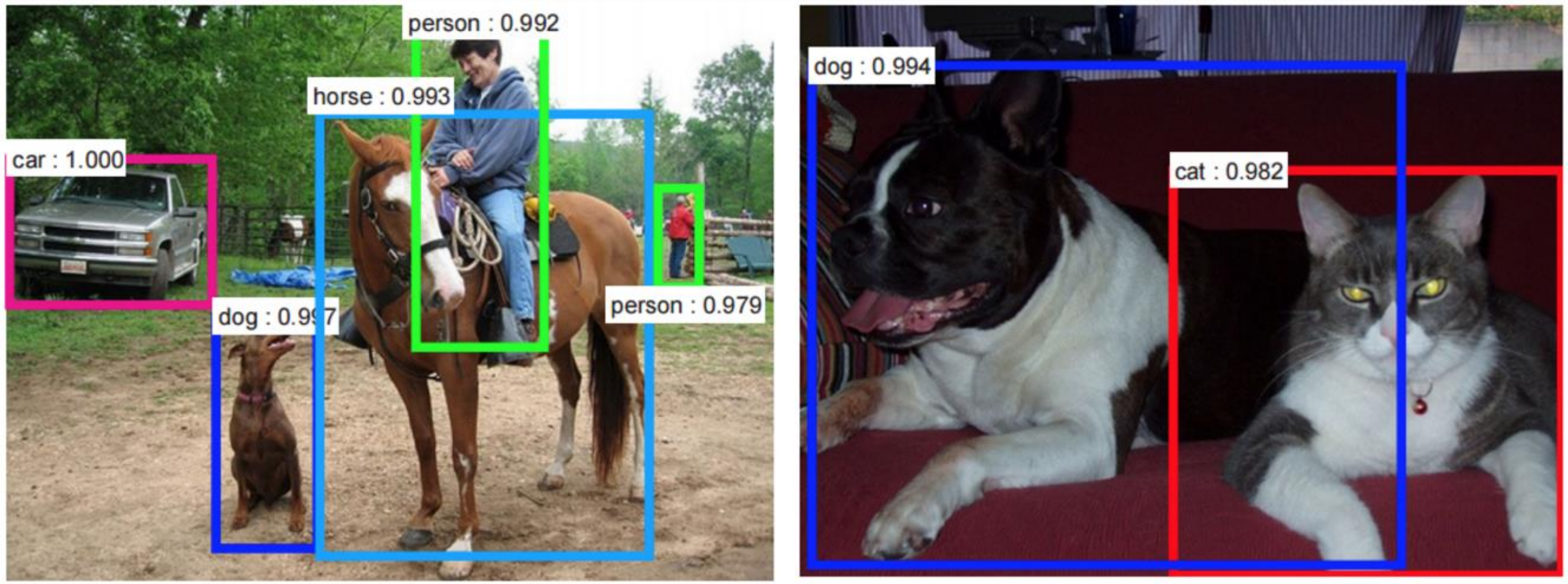
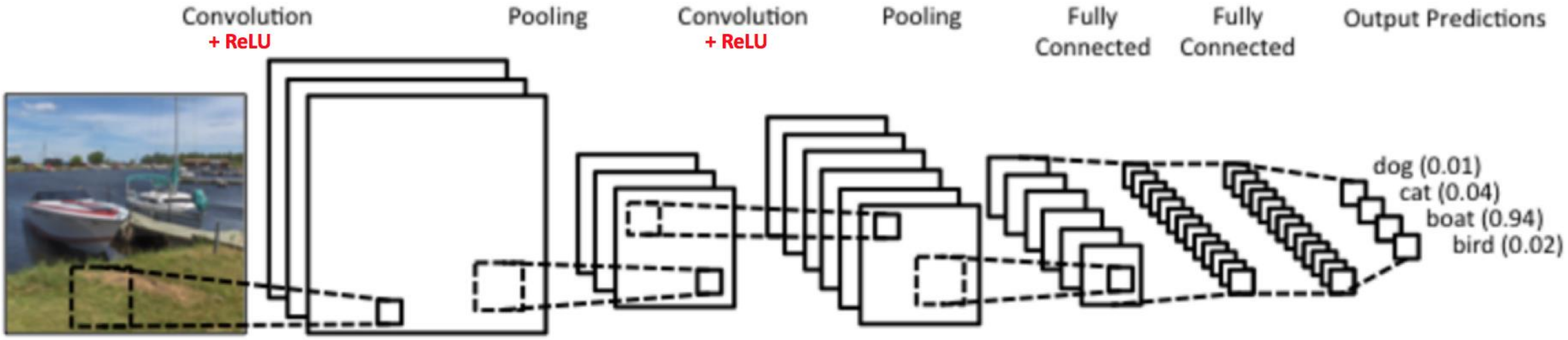
Convolutional Neural Networks

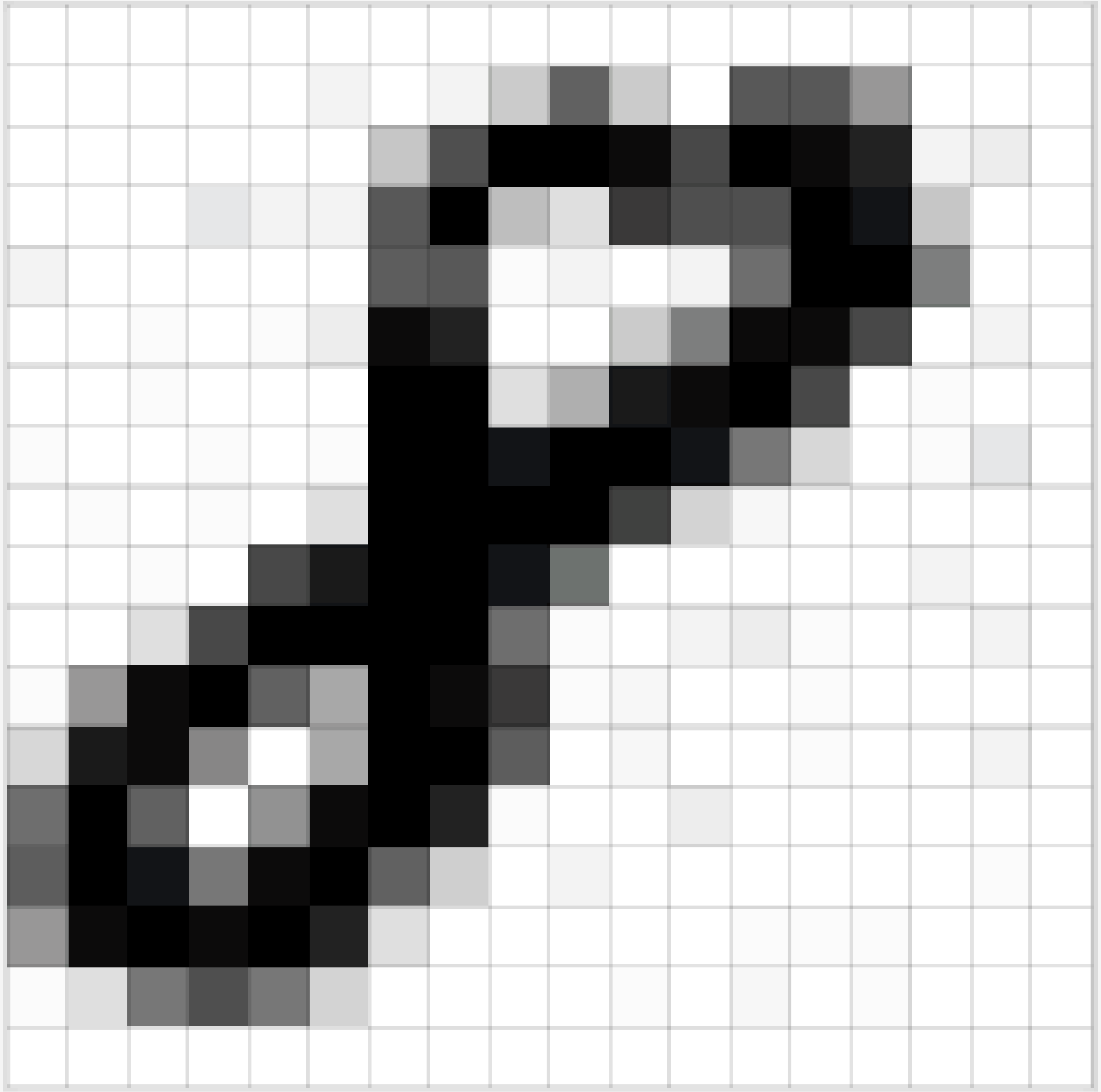


Convolution Layer



An Image is a matrix of pixel values

Essentially, every image can be represented as a matrix of pixel values.



The Convolution Step

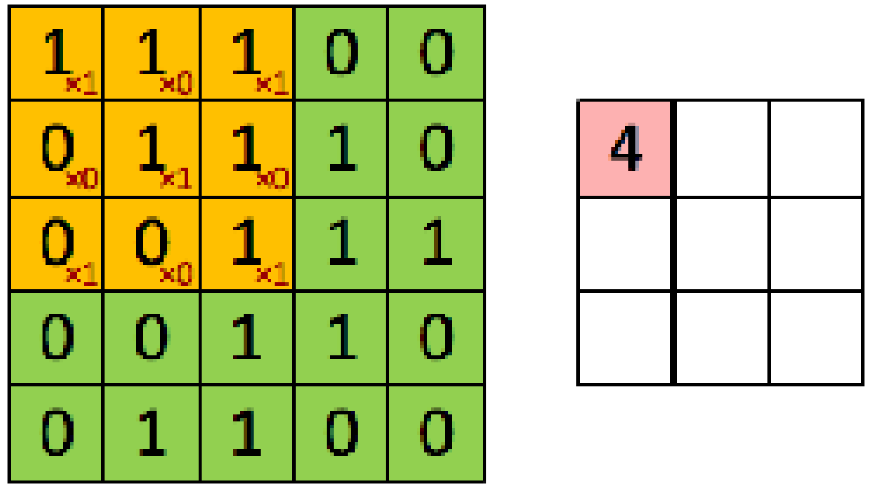
ConvNets derive their name from the ”convolution operator”. The primary purpose of Convolution in case of a ConvNet is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data. We will not go into the mathematical details of Convolution here, but will try to understand how it works over images.

Consider a 5 x 5 image whose pixel values are only 0 and 1 (note that for a grayscale image, pixel values range from 0 to 255, the green matrix below is a special case where pixel values are only 0 and 1):

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Also, consider another 3 x 3 matrix as shown below:

1	0	1
0	1	0
1	0	1

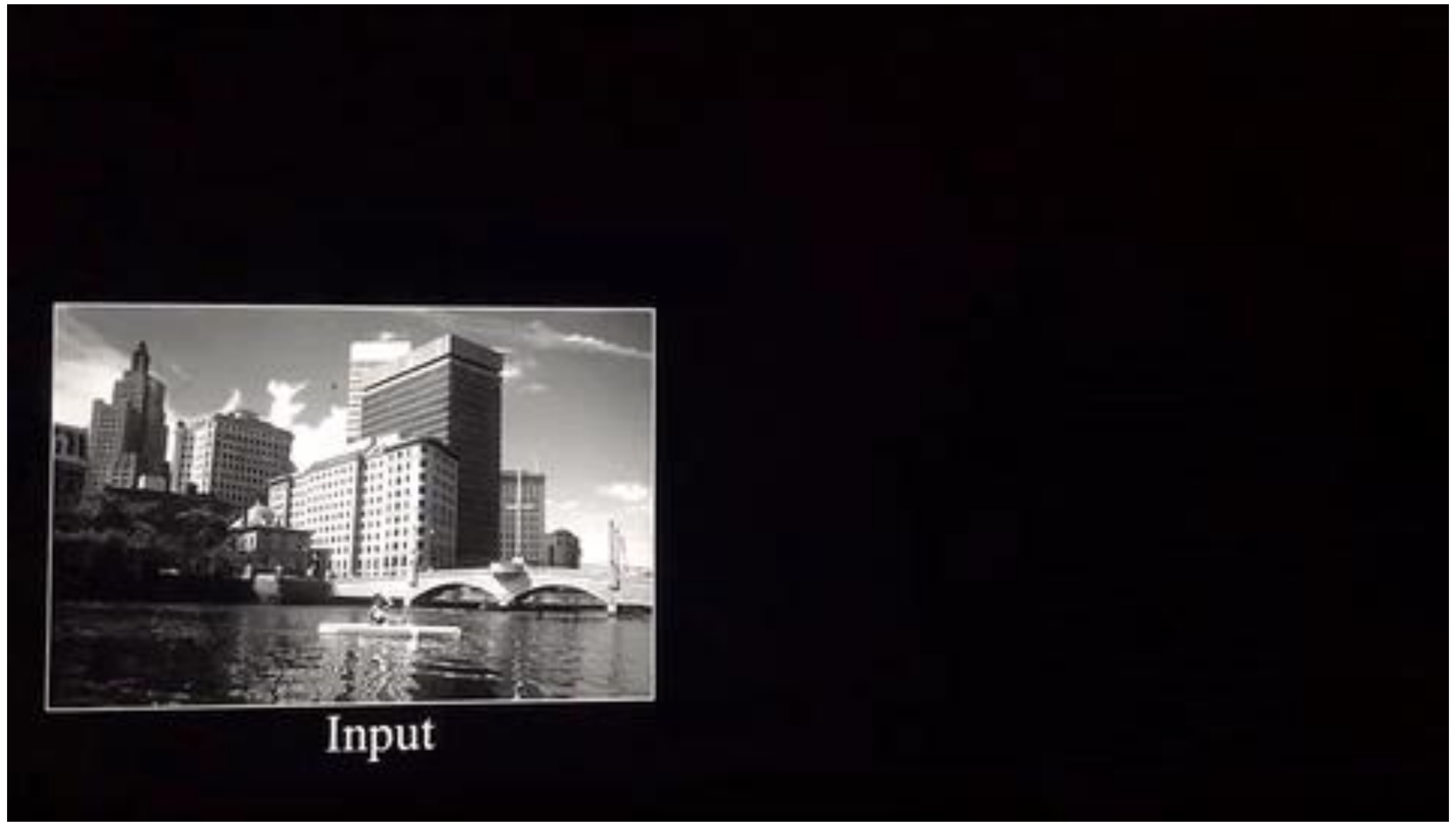


Image

Convolved Feature

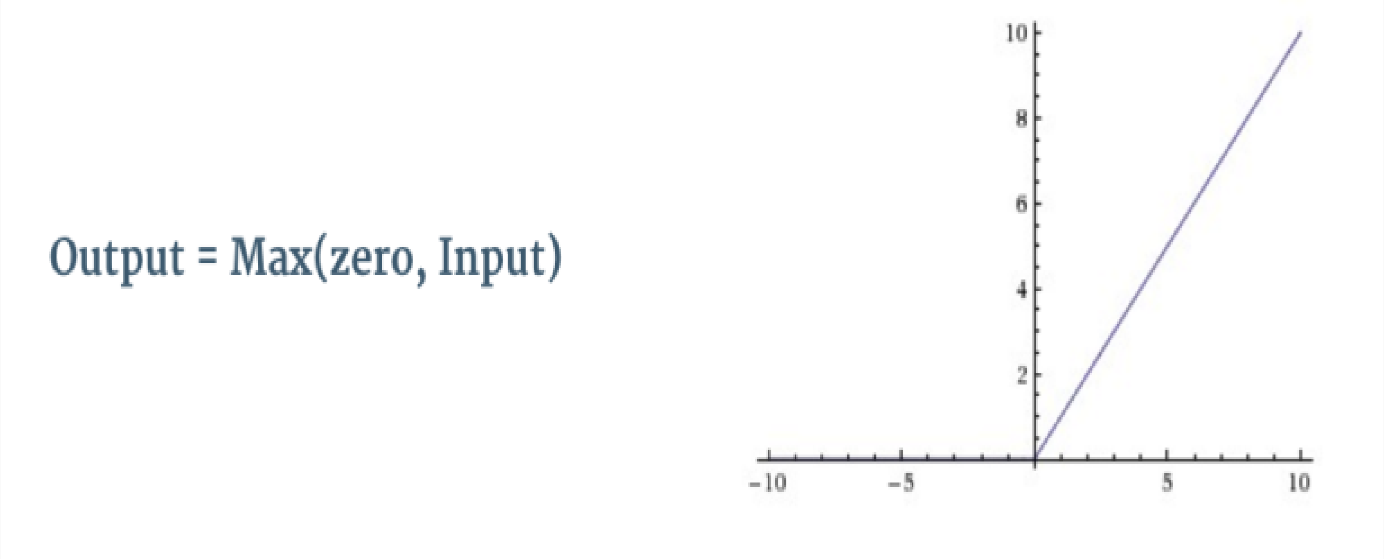
In the table below, we can see the effects of convolution of the above image with different filters. As shown, we can perform operations such as Edge Detection, Sharpen and Blur just by changing the numeric values of our filter matrix before the convolution operation

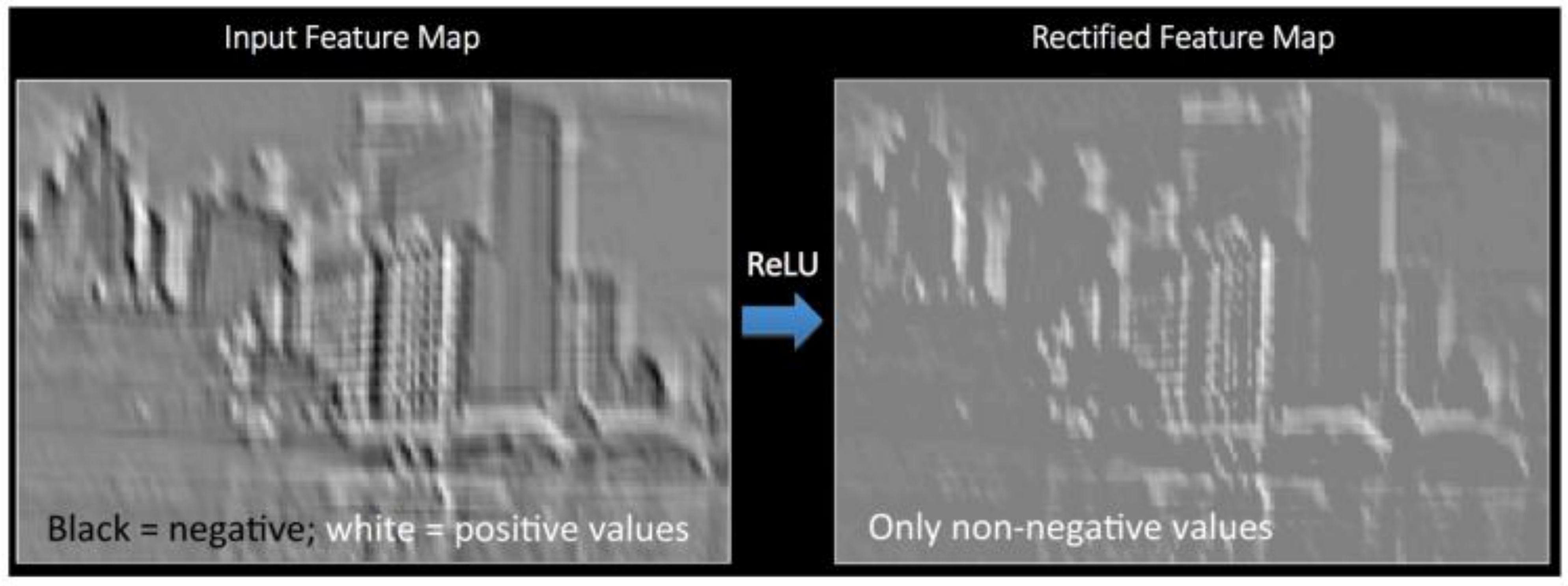
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	



Introducing Non Linearity (ReLU)

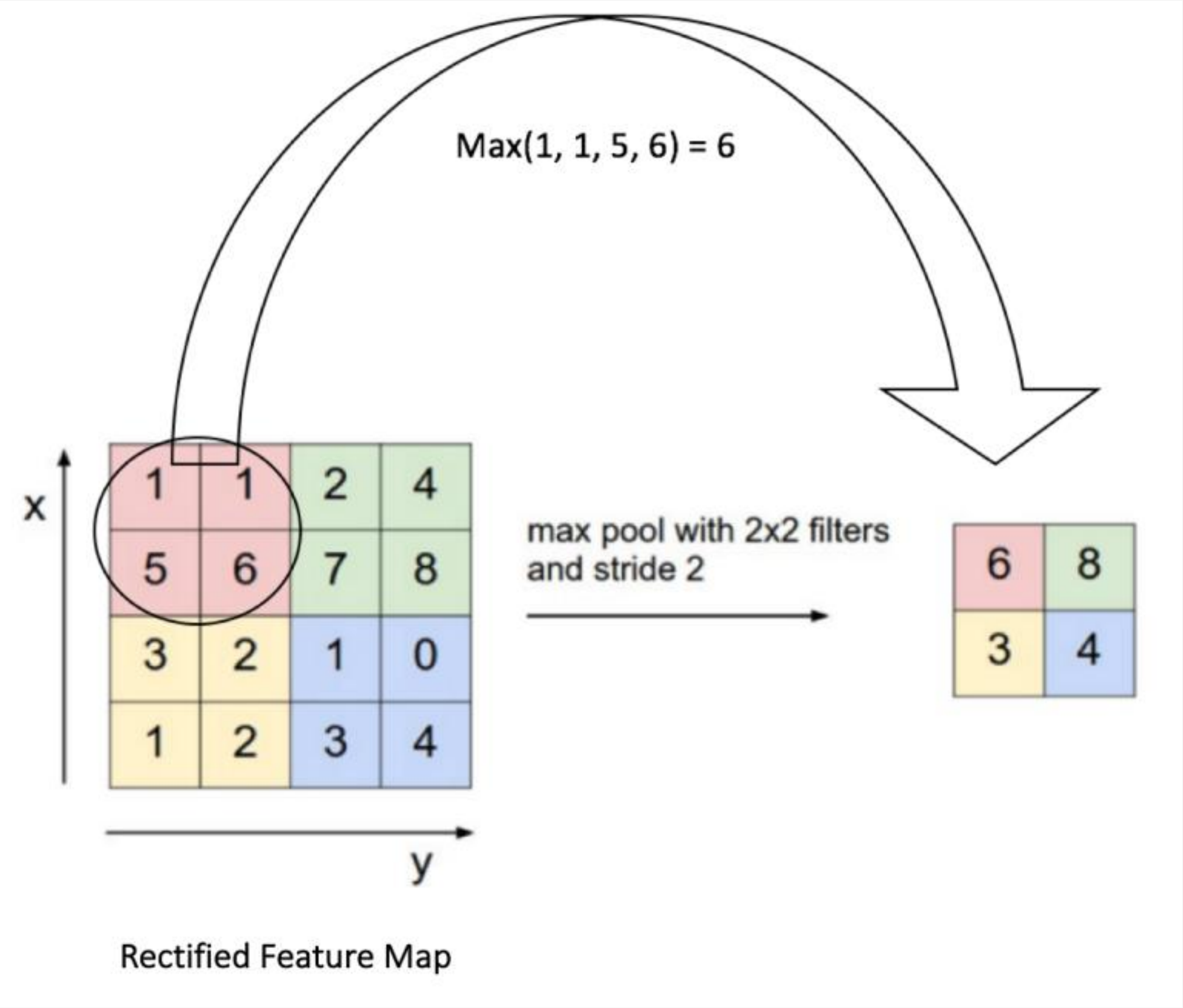
An additional operation called ReLU has been used after every Convolution operation

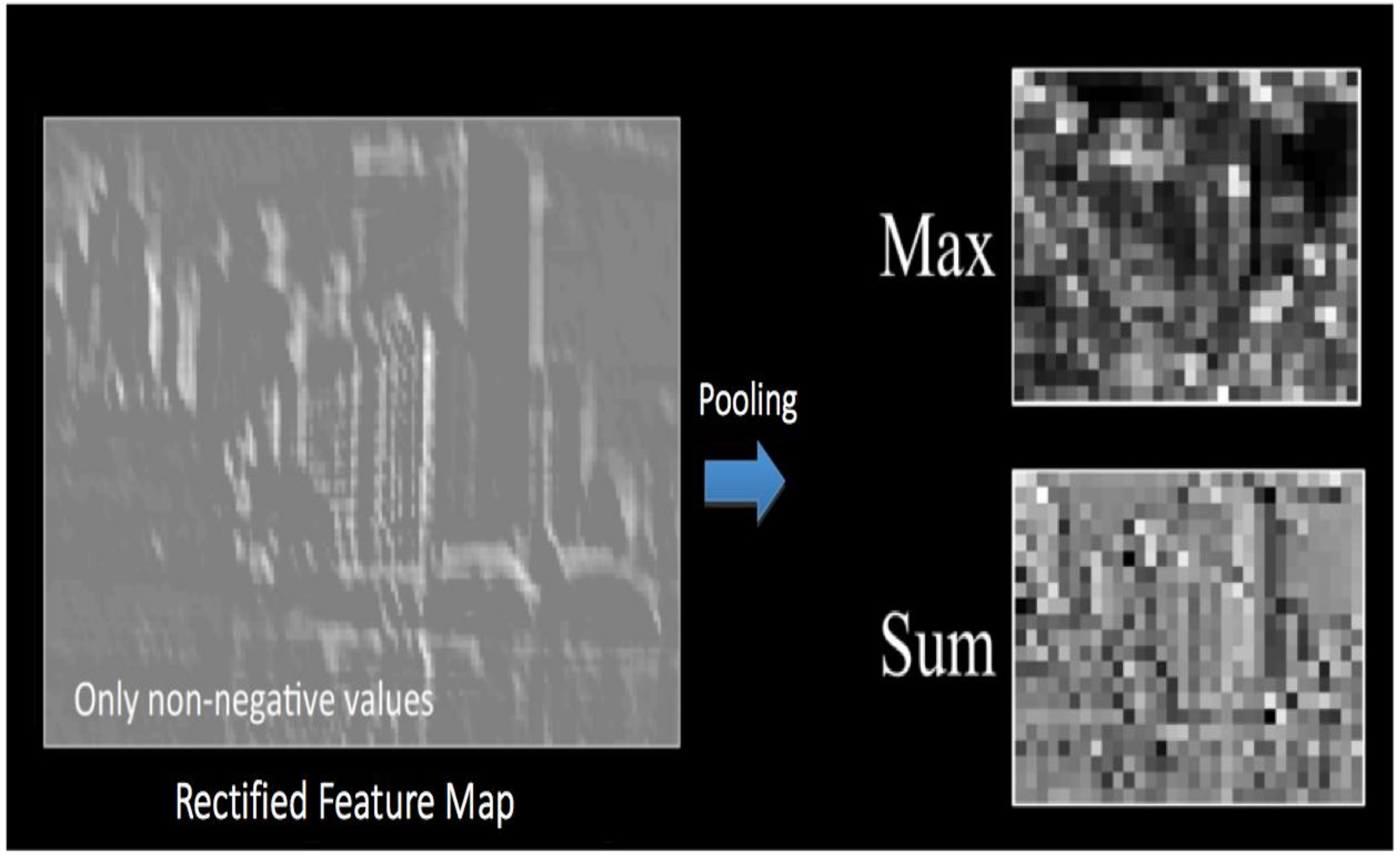




The Pooling Step

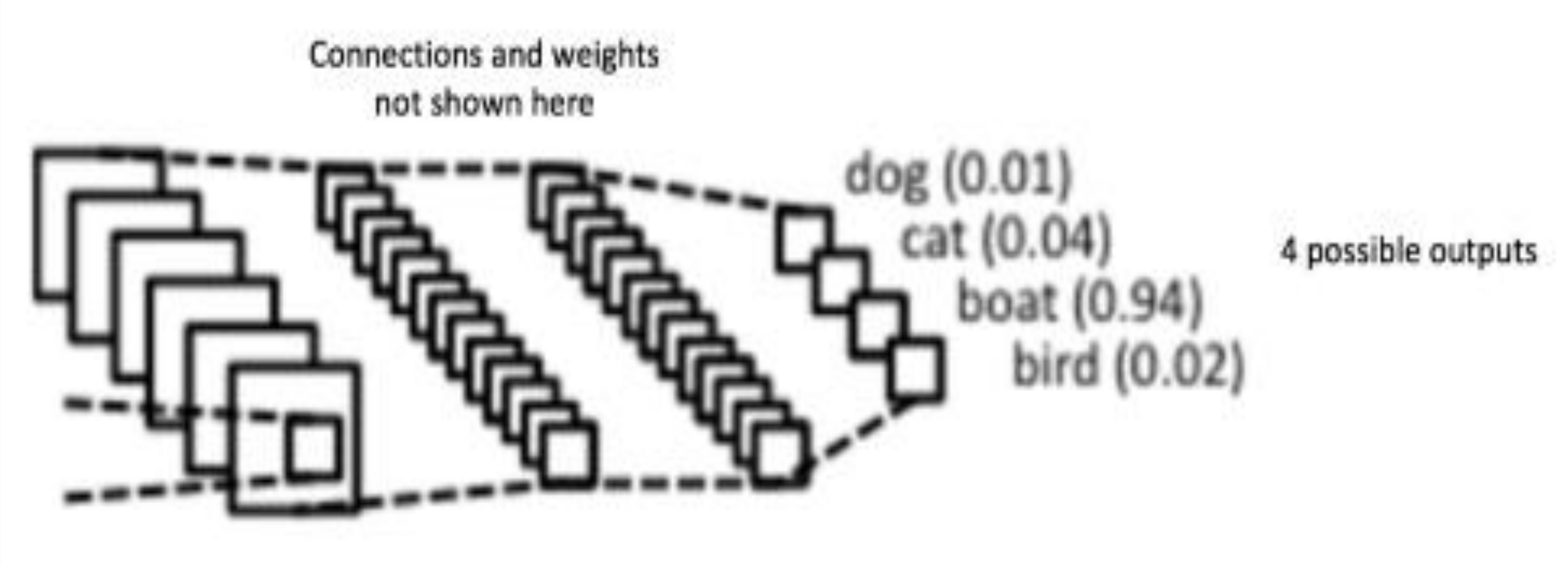
Spatial Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc.





Fully Connected Layer

The Fully Connected layer is a traditional Multi Layer Perceptron that uses a softmax activation function in the output layer (other classifiers like SVM can also be used, but will stick to softmax in this post). The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer.

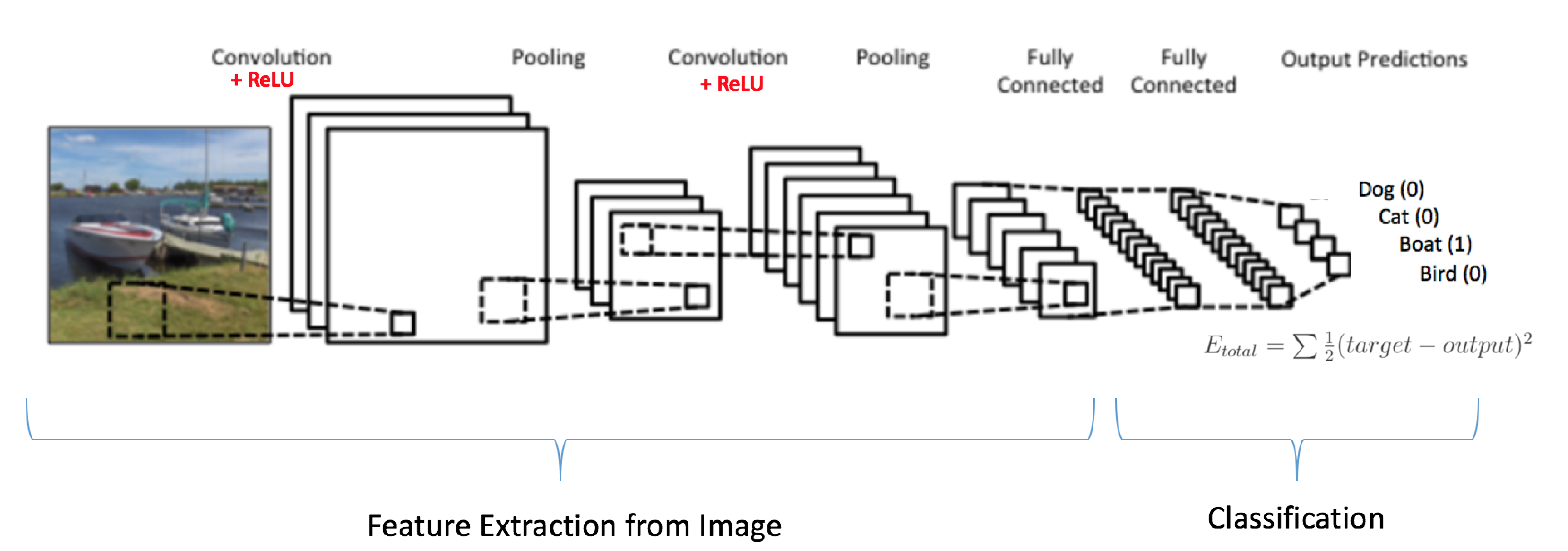


Putting it all together – Training using Backpropagation

As discussed above, the Convolution + Pooling layers act as Feature Extractors from the input image while Fully Connected layer acts as a classifier.

Note that below, since the input image is a boat, the target probability is 1 for Boat class and 0 for other three classes, i.e.

- Input Image = Boat
- Target Vector = [0, 0, 1, 0]



Recurrent neural networks

- Dates back to (Rumelhart *et al.*, 1986)
- A family of neural networks for handling sequential data, which involves variable length inputs or outputs

- Especially, for natural language processing (NLP)

Sequential data

- Each data point: A sequence of vectors $x^{(t)}$, for $1 \leq t \leq \tau$
- Batch data: many sequences with different lengths τ
- Label: can be a scalar, a vector, or even a sequence

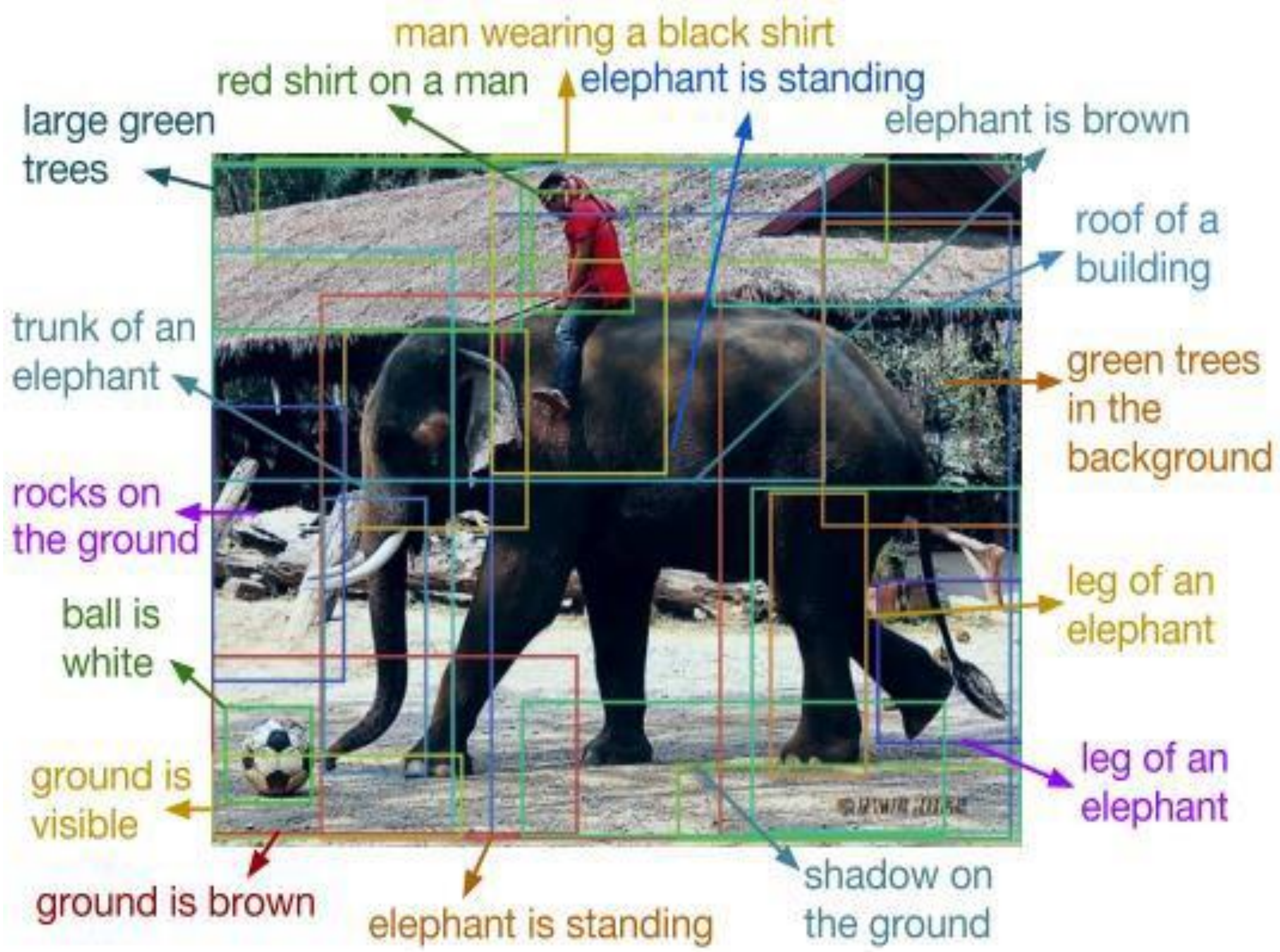
- Example
  - Sentiment analysis
  - Machine translation

More complicated sequential data

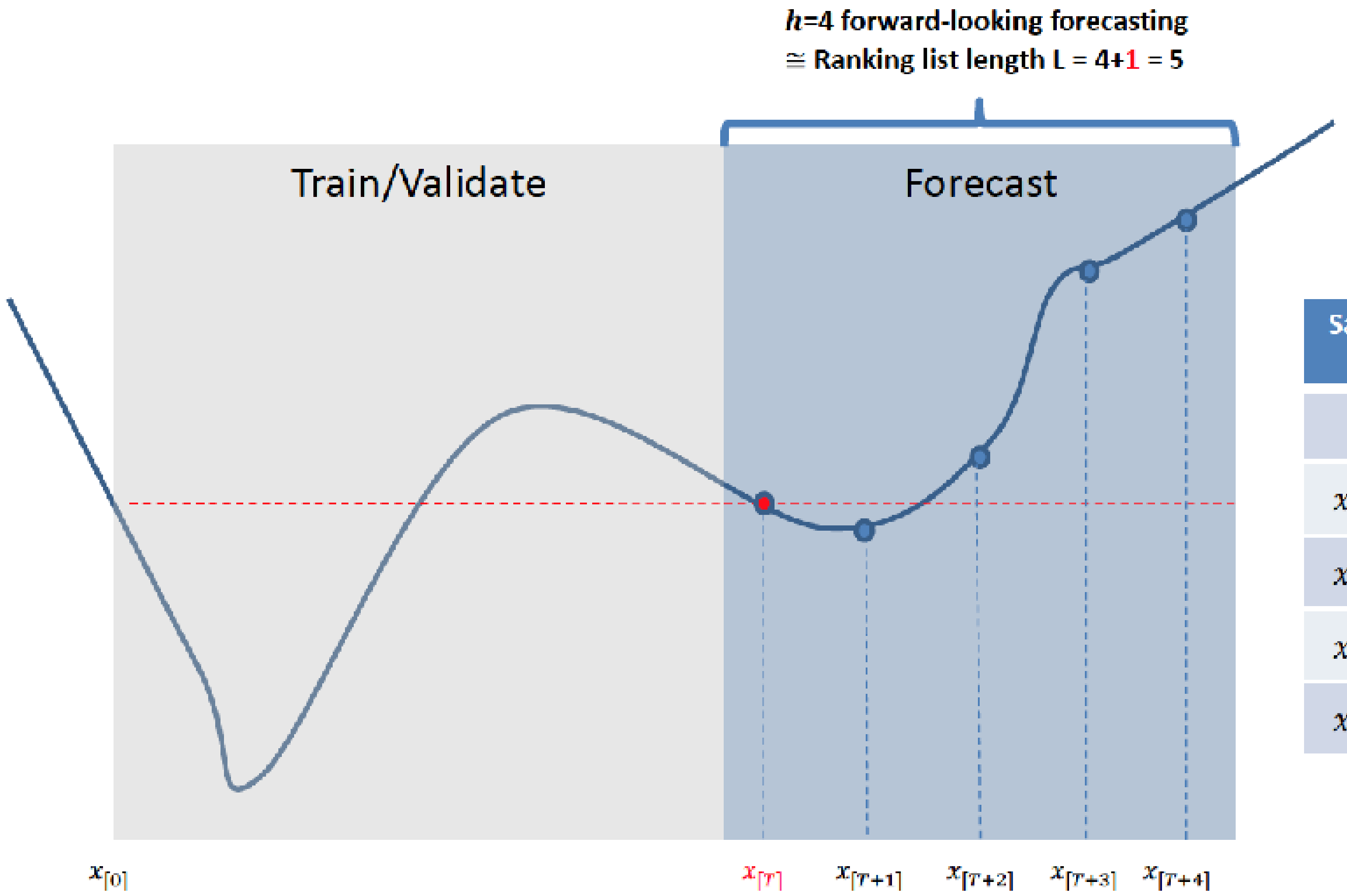
- Data point: two dimensional sequences like images
- Label: different type of sequences like text sentences

- Example: image captioning

Image captioning



Time- series forecasting



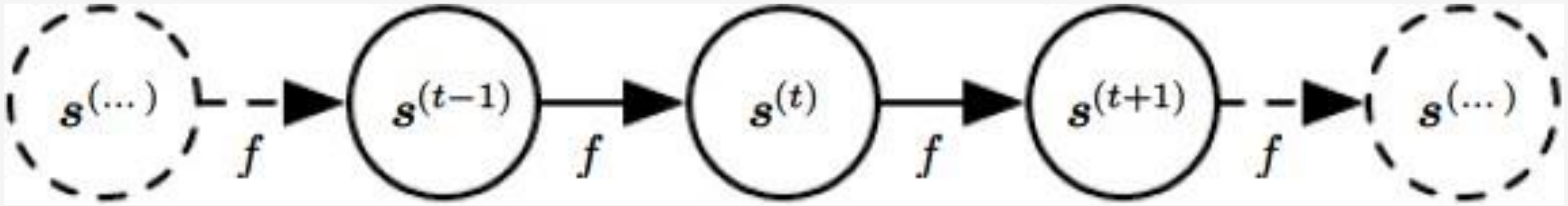
The forecasted ranking list $\pi_{[T,T+4]}$ at t=T

Samples	Ground Truth	Ranking Forecast
$x_{[T]}$	$r_{[T]} = 4$	$\hat{r}_{[T]}$
$x_{[T+1]}$	$r_{[T+1]} = 5$	$\hat{r}_{[T+1]}$
$x_{[T+2]}$	$r_{[T+2]} = 3$	$\hat{r}_{[T+2]}$
$x_{[T+3]}$	$r_{[T+3]} = 2$	$\hat{r}_{[T+3]}$
$x_{[T+4]}$	$r_{[T+4]} = 1$	$\hat{r}_{[T+4]}$

Values of Interest

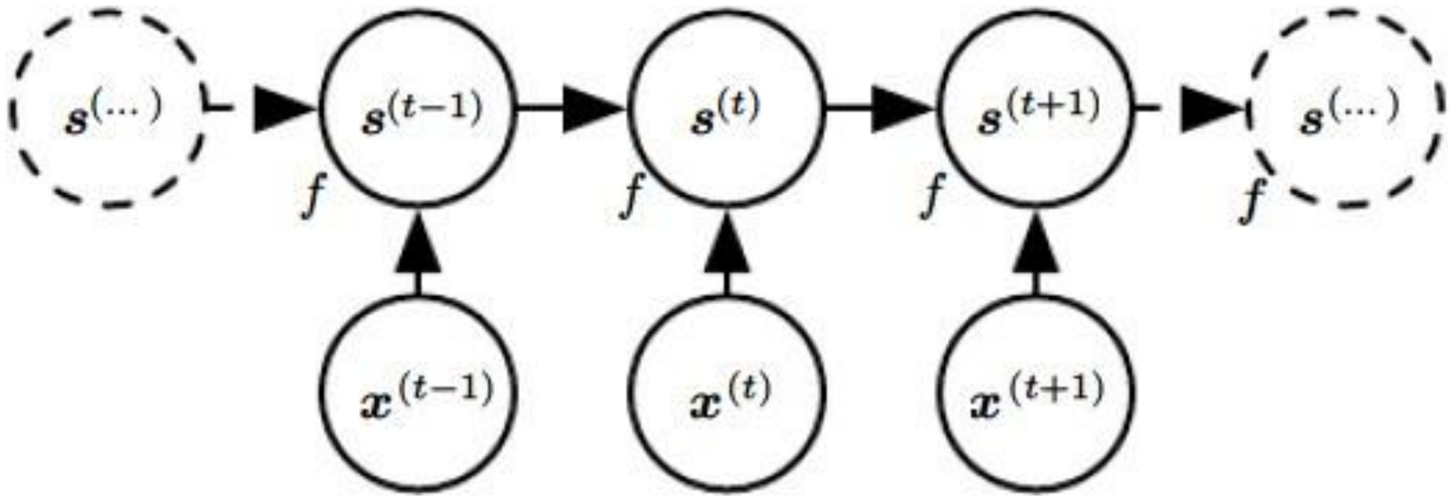
FIGURE 1. Illustration of the multi-step time-series forecasting scheme. There are in total five values of interest including the $\hat{r}_{[t]}$, whose ground truth of $x_{[t]}$ is known and highlighted in red color.

A typical dynamic system



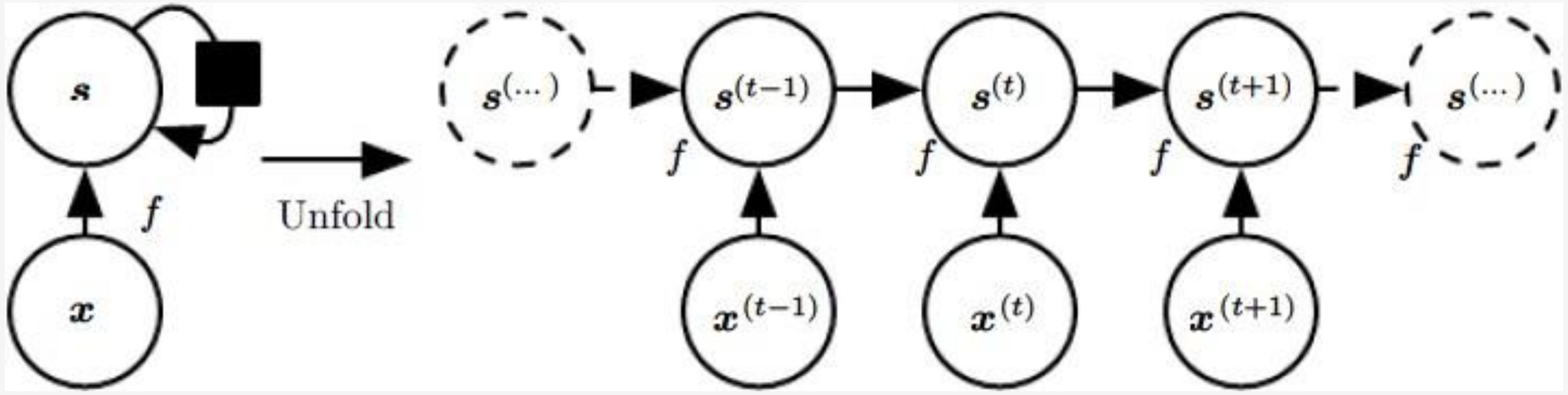
$$s^{(t+1)} = f(s^{(t)})$$

A system driven by external data



$$s^{(t+1)} = f(s^{(t)}, x^{(t+1)})$$

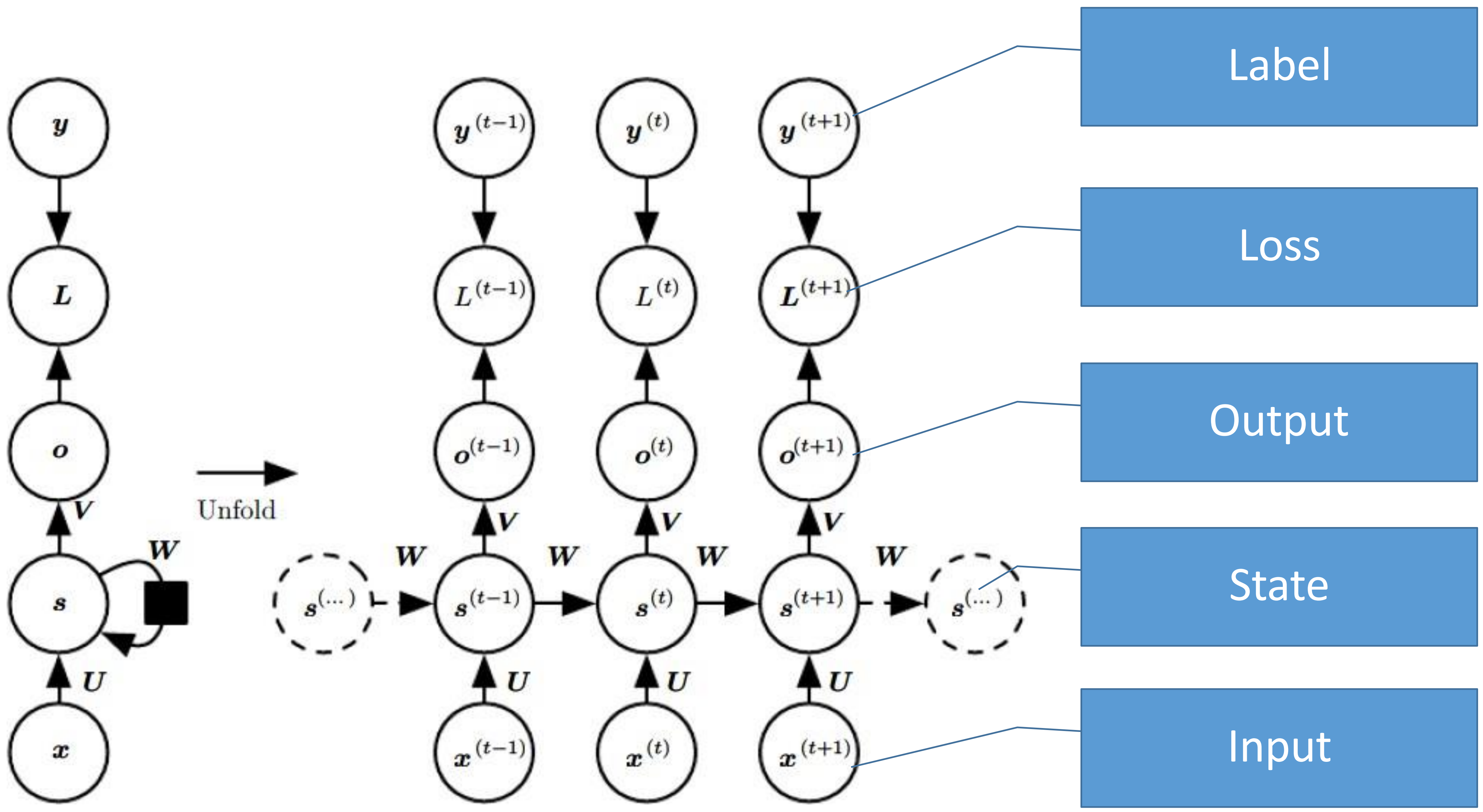
Compact view



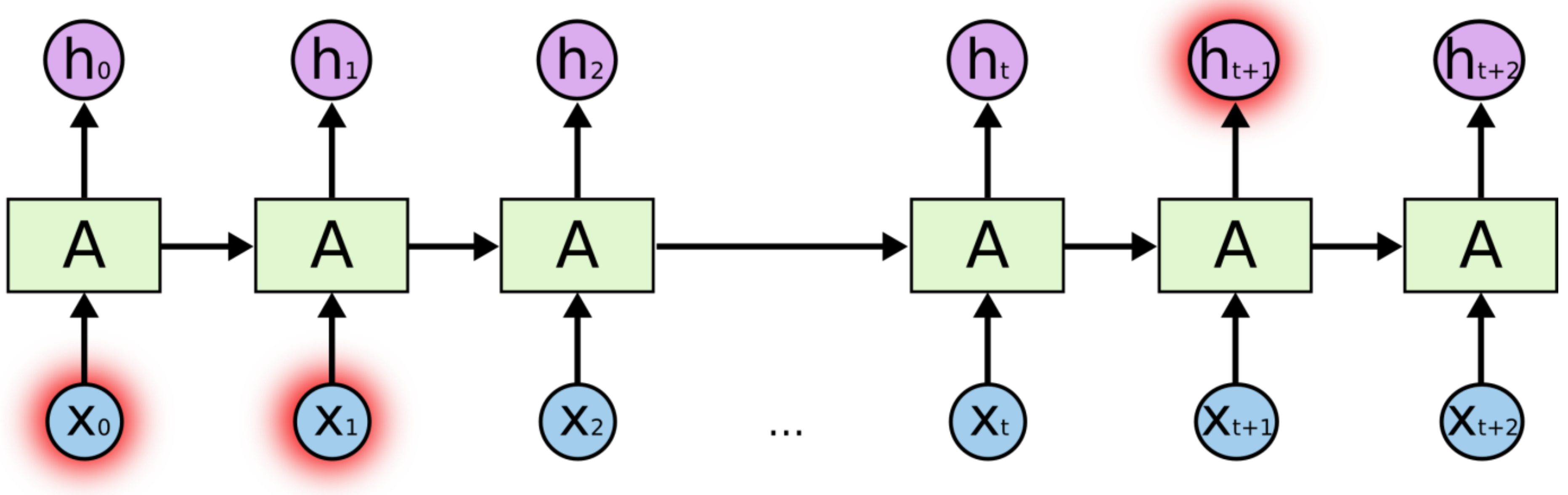
$$s^{(t+1)} = f(s^{t}, x^{(t+1)})$$

Recurrent neural networks

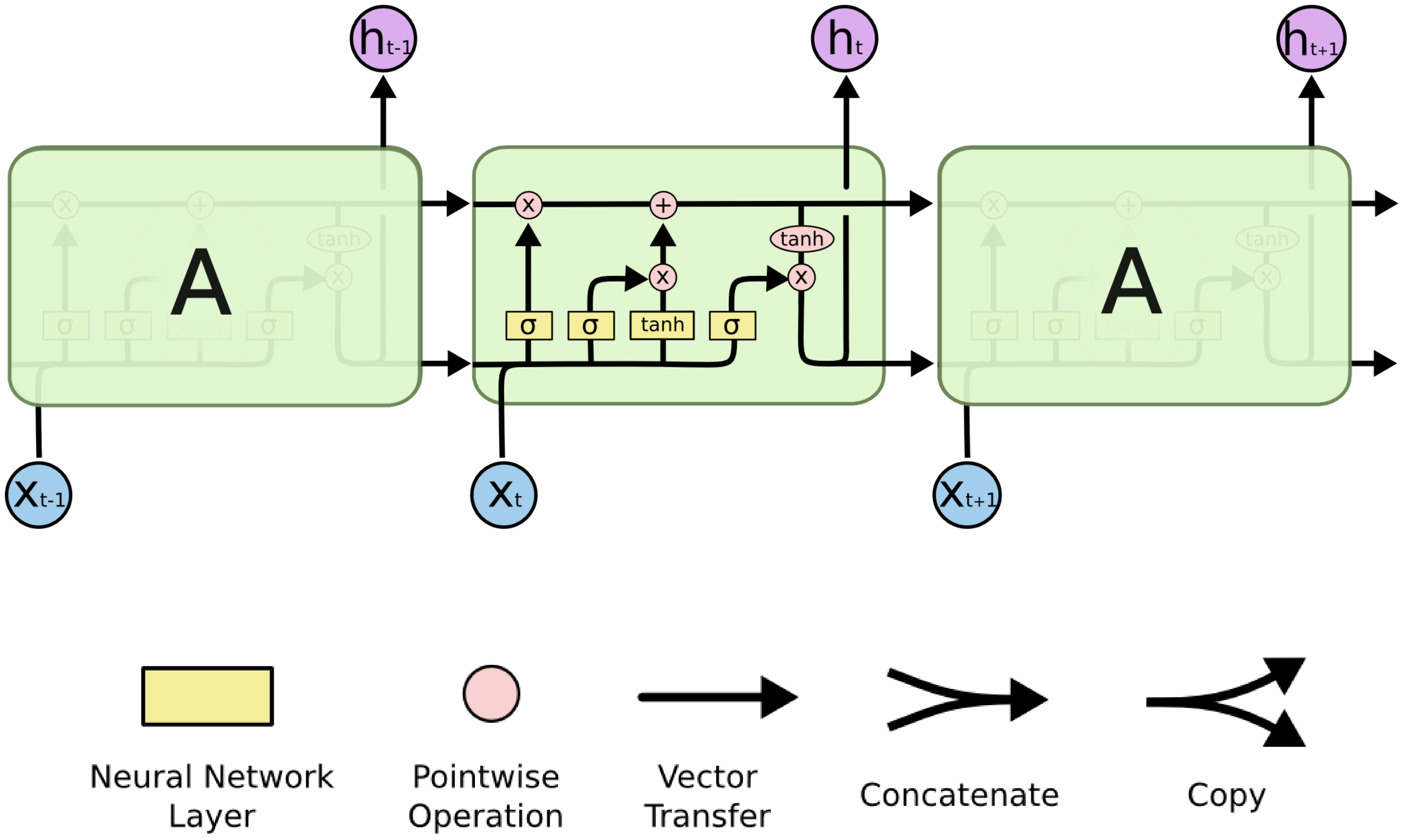
- Use the same computational function and parameters across different time steps of the sequence
- Each time step: takes the input entry and the previous hidden state to compute the output entry
- Loss: typically computed every time step


Unfold
Label
Loss
Output
State
Input

- It is very difficult to train RNNs to retain information over many time steps
- This make is very difficult to learn RNNs that handle long-distance dependencies, such as subject-verb agreement.



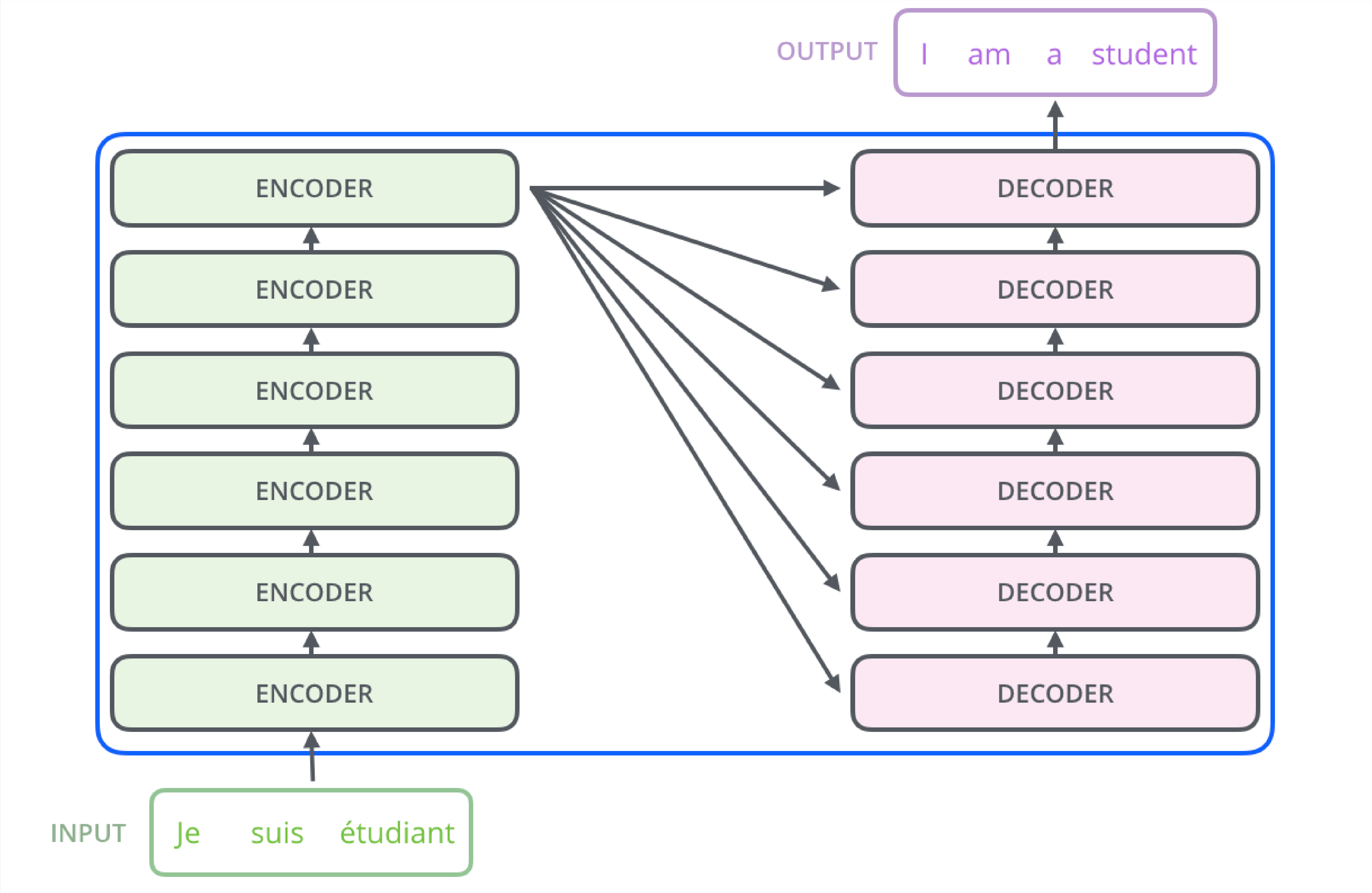
LSTM Network Architecture



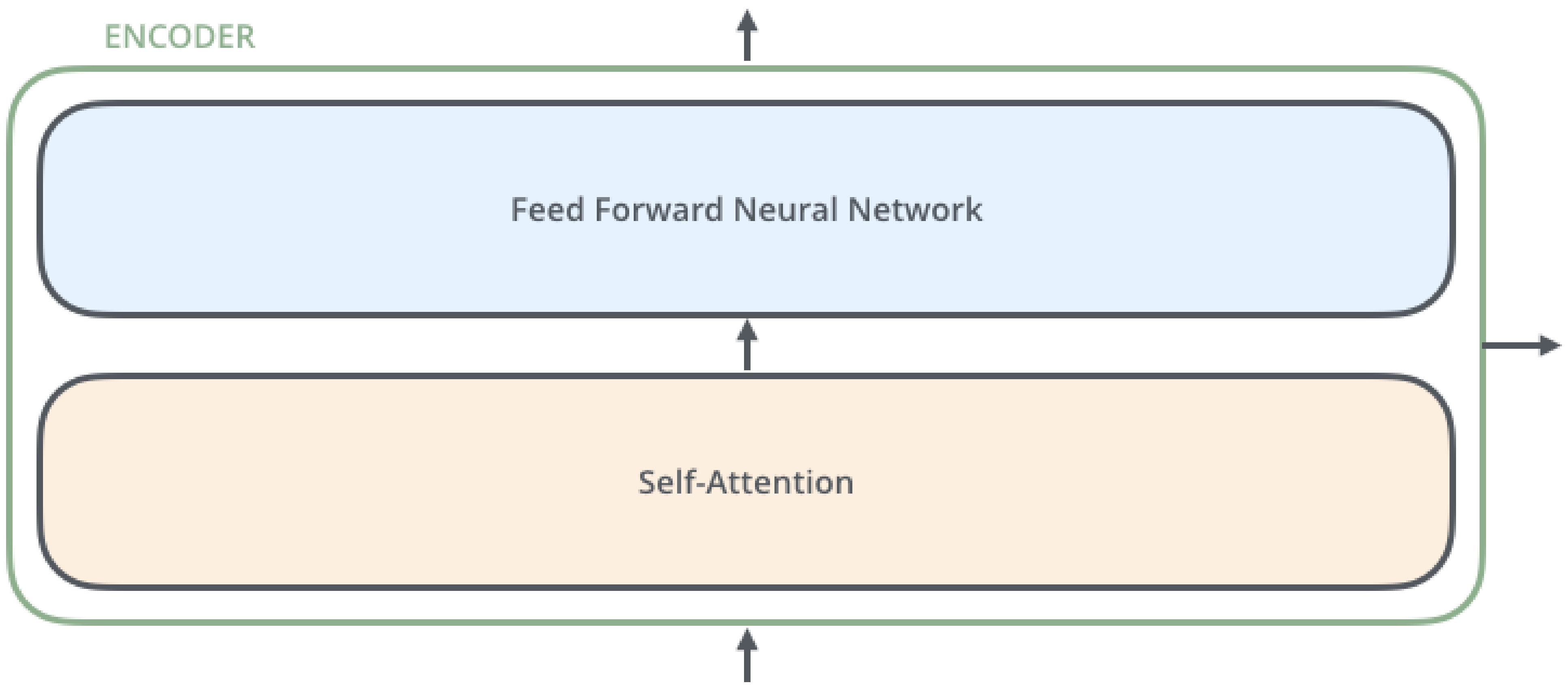
Advantage

- Hidden state: a lossy summary of the past
- Shared functions and parameters: greatly reduce the capacity and good for generalization in learning
- Explicitly use the prior knowledge that the sequential data can be processed by in the same way at different time step (e.g., NLP)

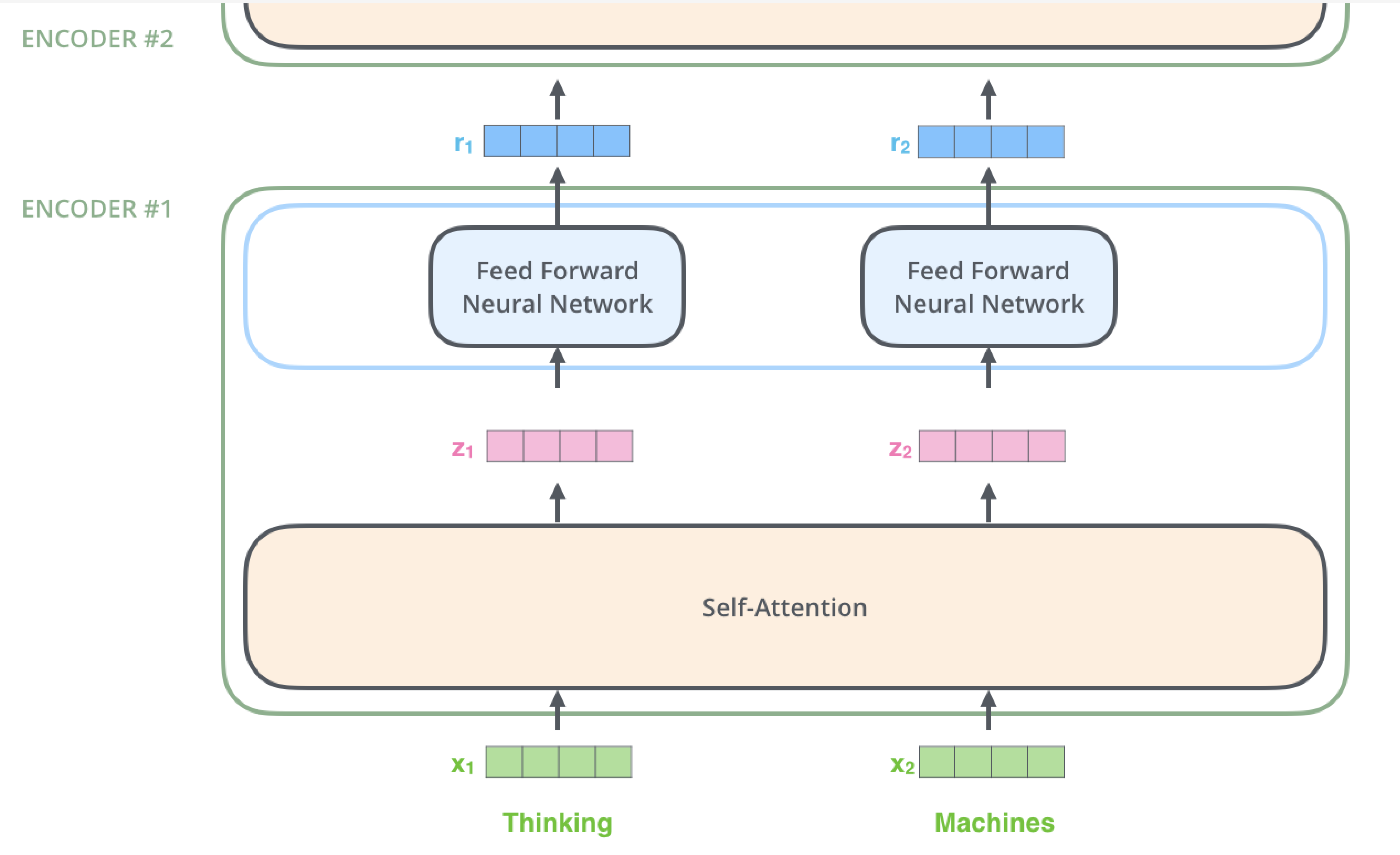
Transformer



An Encoder Block: same structure, different parameters

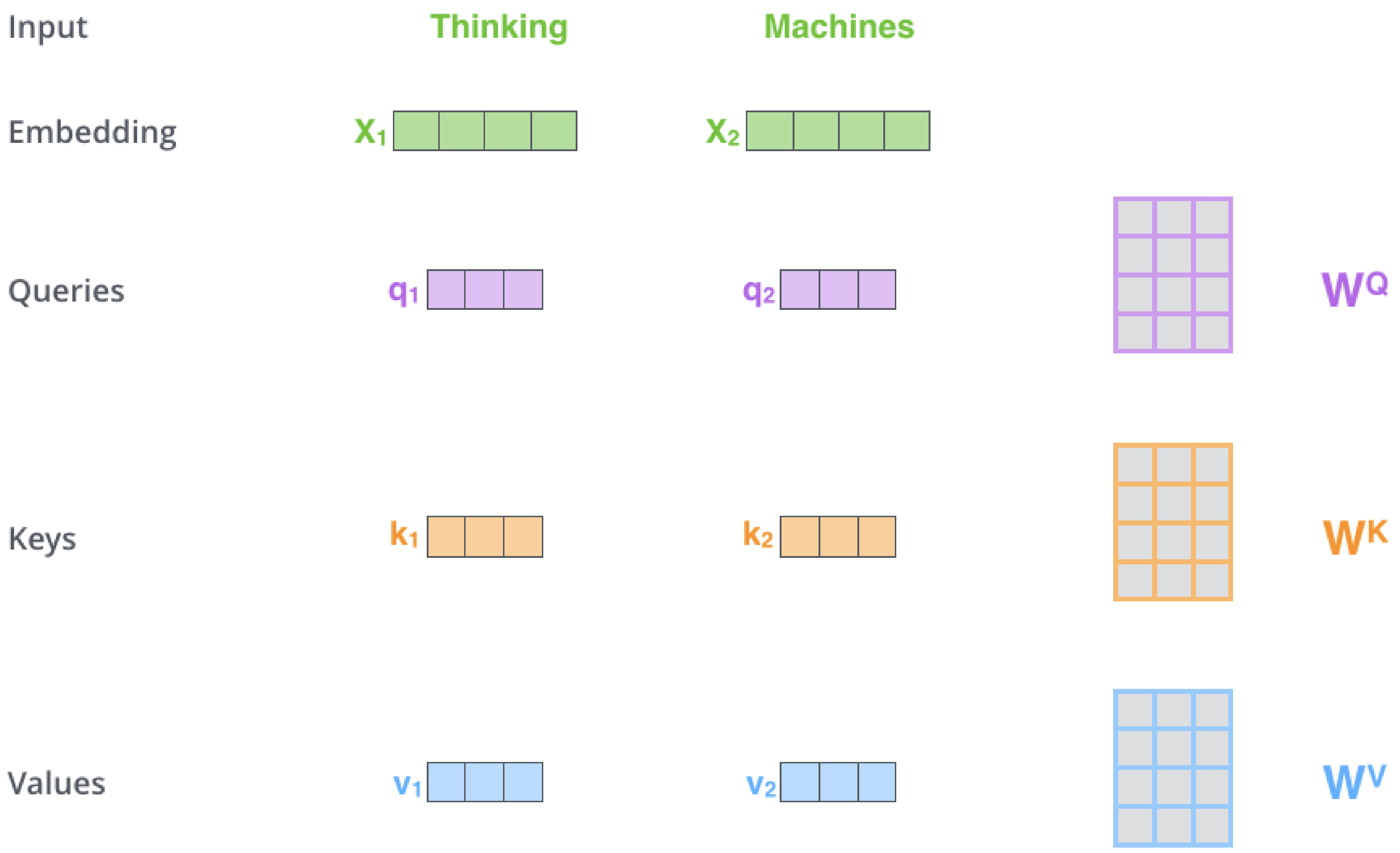


Encoder



Note: The ffnn is independent
for each word.
Hence can be parallelized.

Self Attention

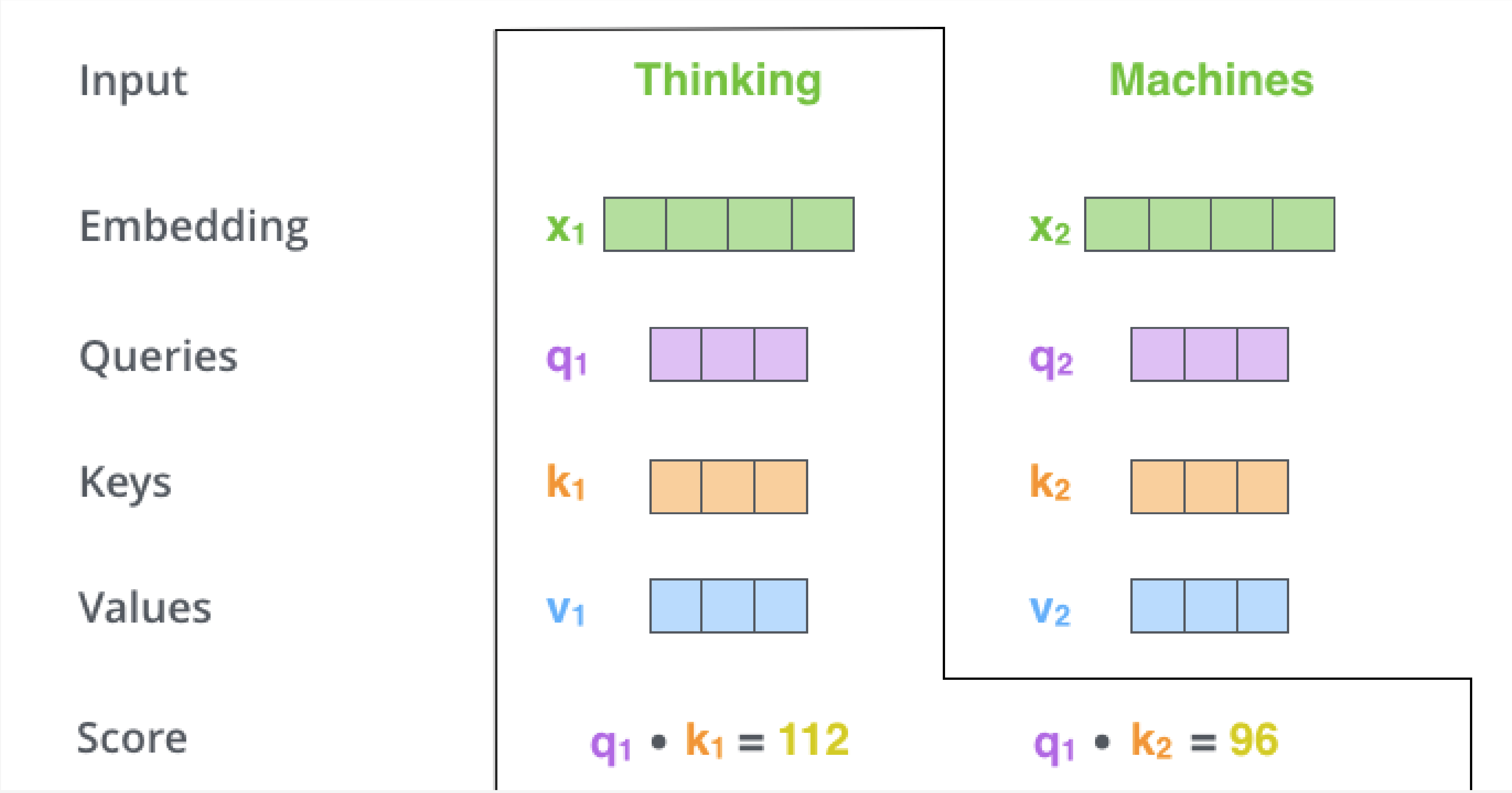


First we create three vectors by multiplying input embedding (1x512) x_i with three matrices (64x512):

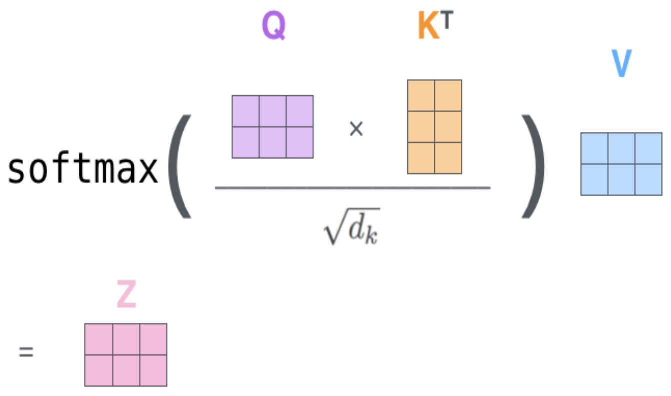
$q_i = x_i W^Q$
$K_i = x_i W^K$
$V_i = x_i W^V$

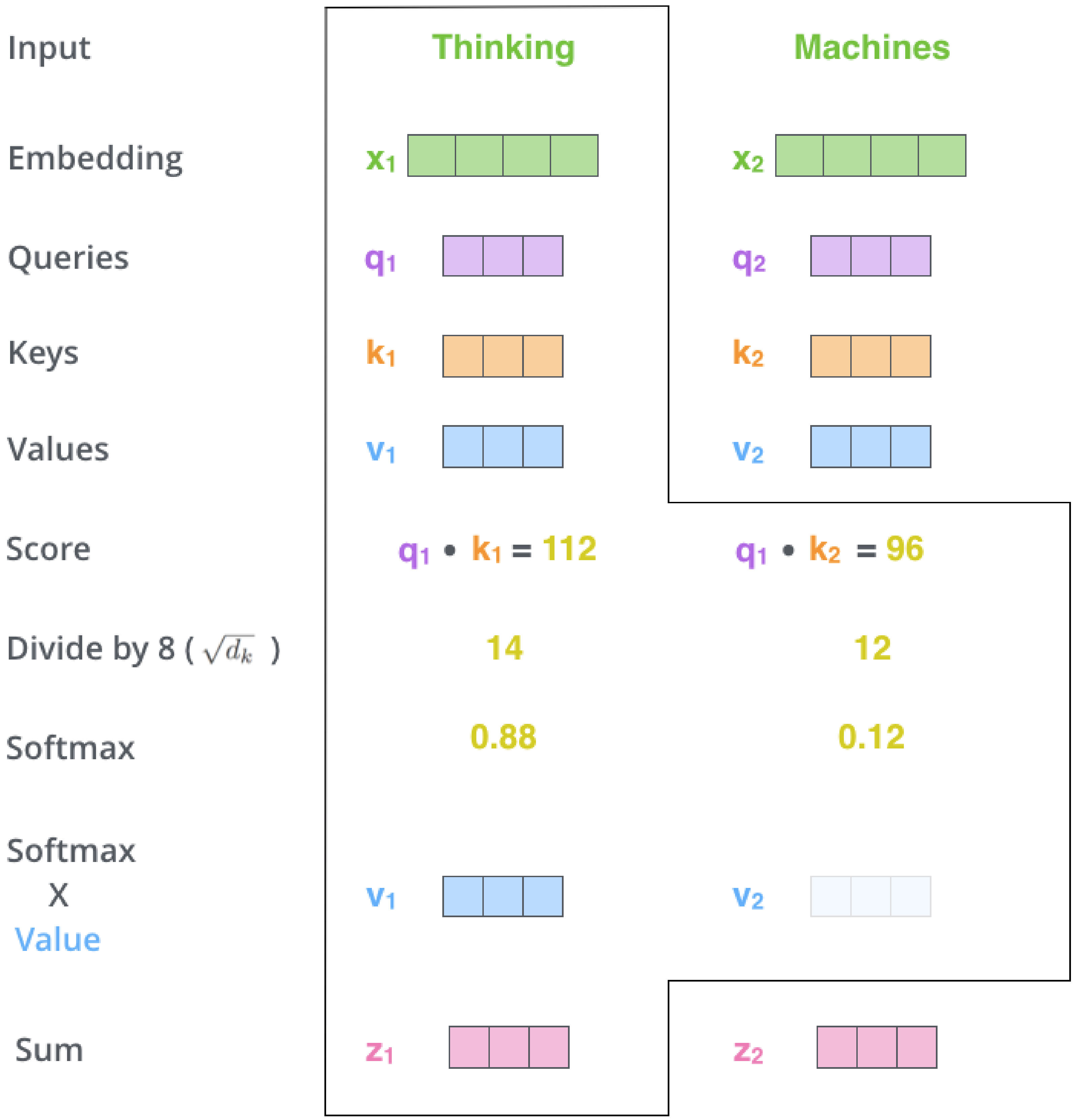
Self Attention

Now we need to calculate a score to determine how much focus to place on other Parts of the input.



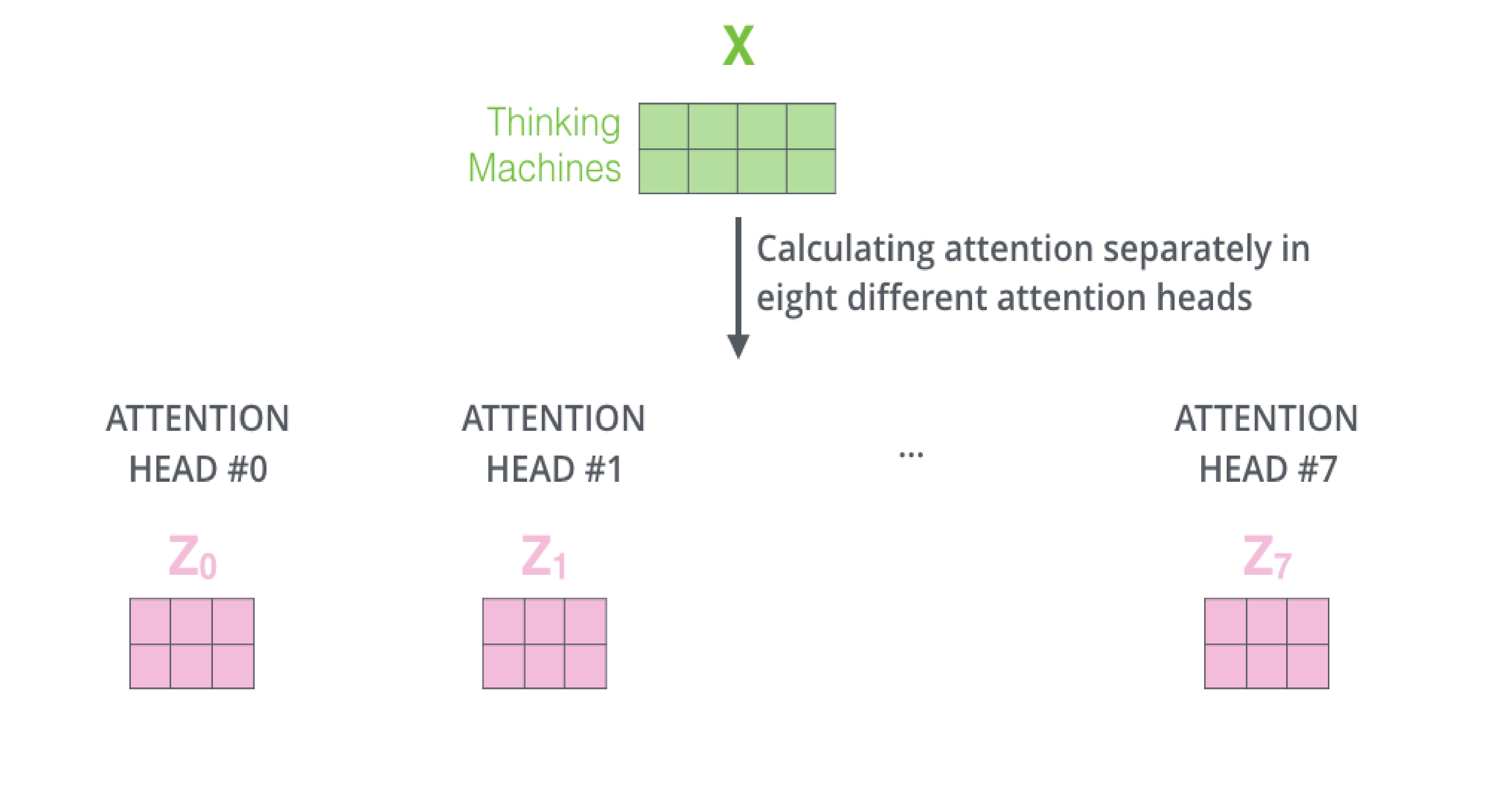
Self Attention





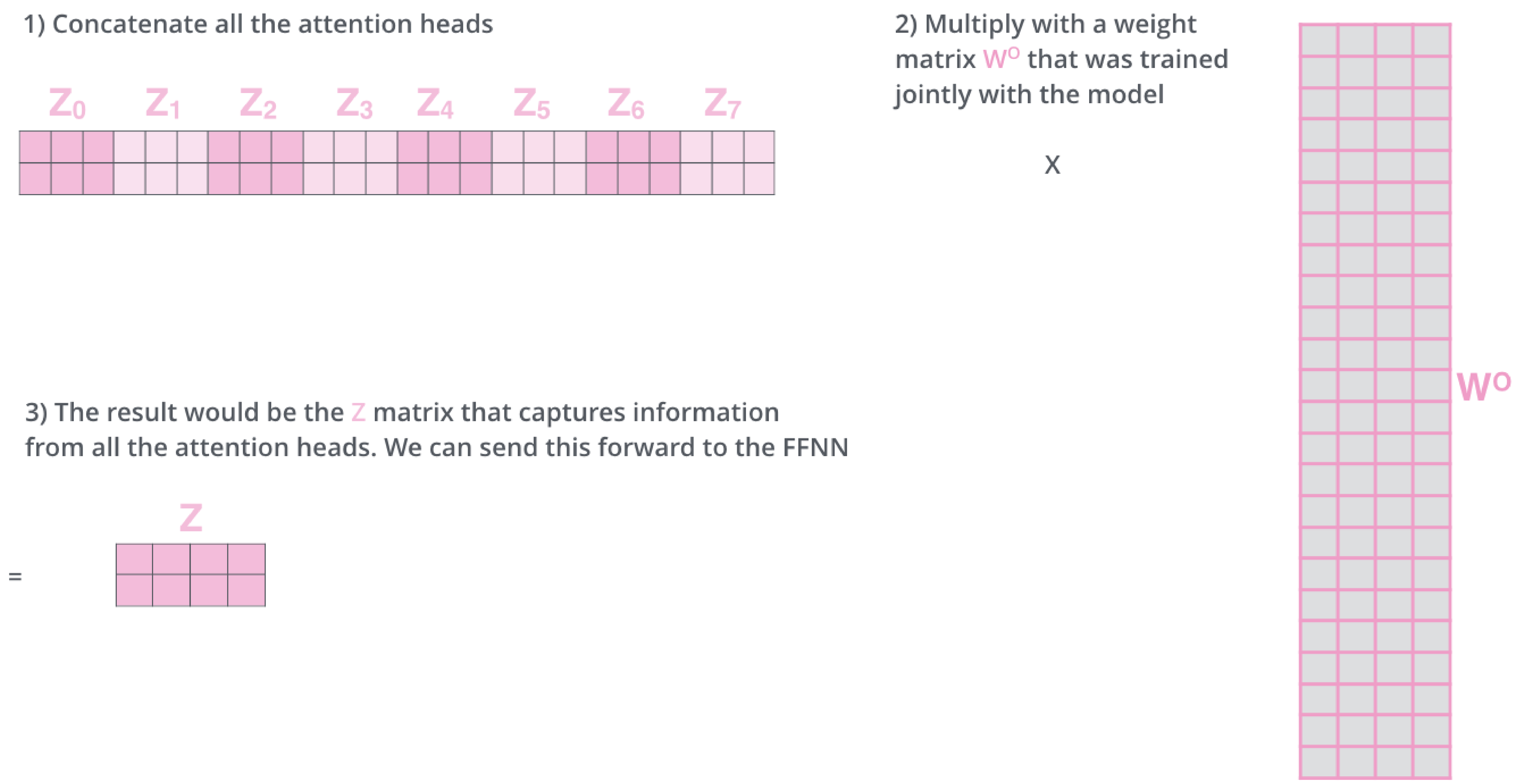
Multiple heads

1. It expands the model's ability to focus on different positions.
2. It gives the attention layer multiple "representation subspaces"



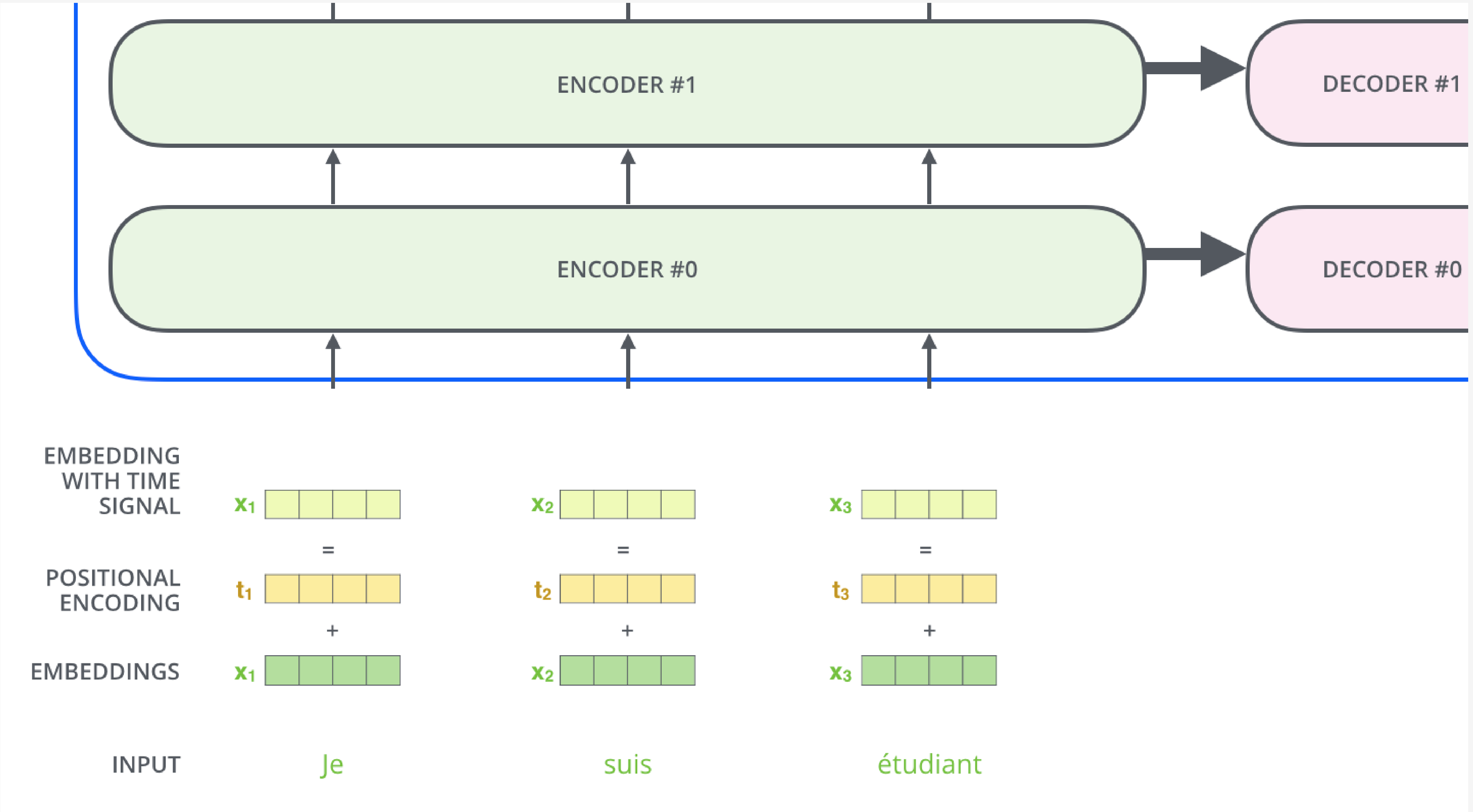
NAC

The output is expecting only a 2x4 matrix, hence,



Representing the input order (positional encoding)

The transformer adds a vector to each input embedding. These vectors follow a specific pattern that the model learns, which helps it determine the position of each word, or the distance between different words in the sequence. The intuition here is that adding these values to the embedings provides meaningful distances between the embedding vectors once they're projected into Q/K/V vectors and during dot-product attention.

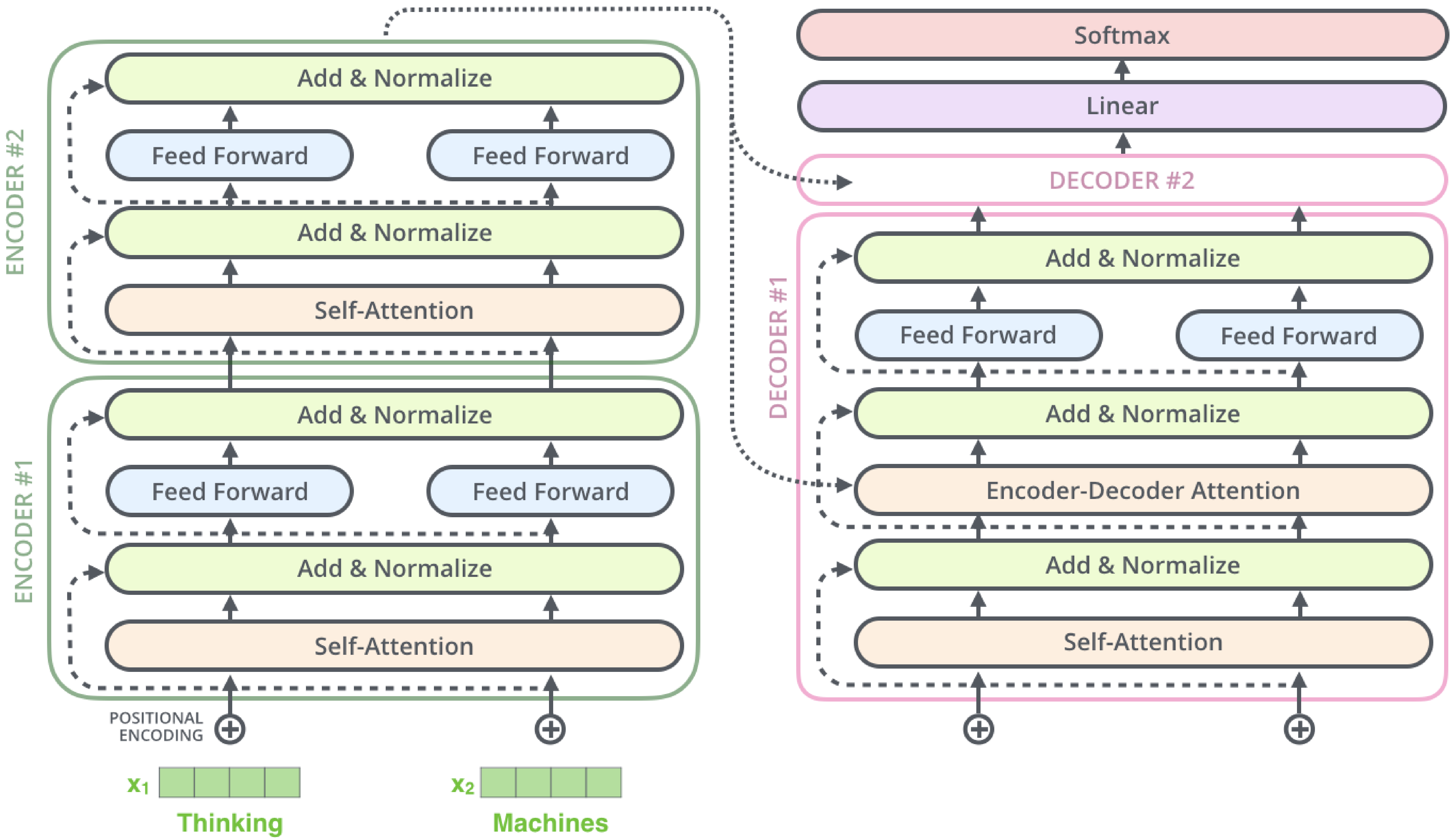


The complete transformer

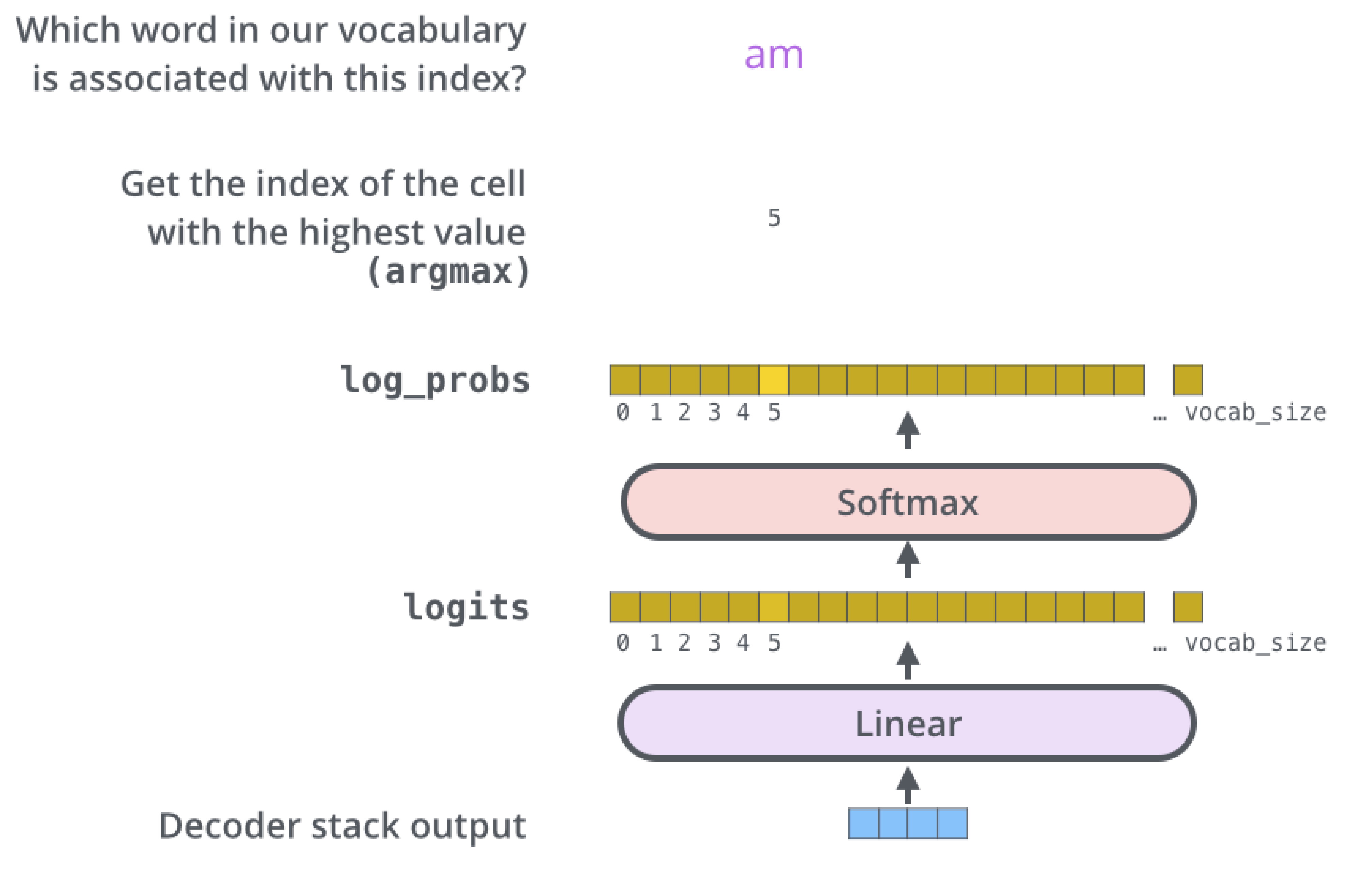
The encoder-decoder attention is just like self attention, except it uses K, V from the top of encoder output, and its own Q

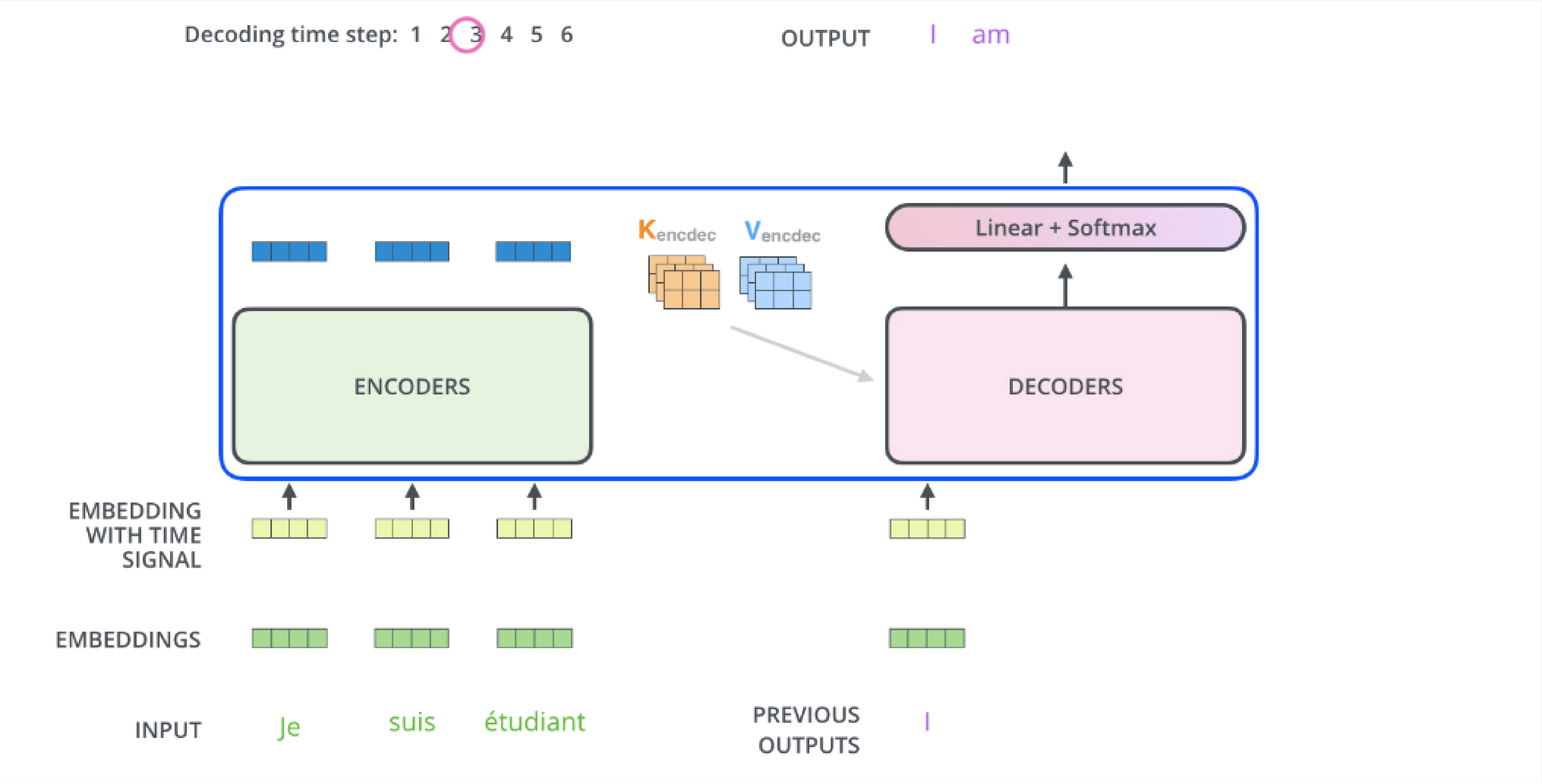
Note: In decoder, the input is "incomplete" when calculating self-attention.

The solution is to set future unknown values with "-inf".

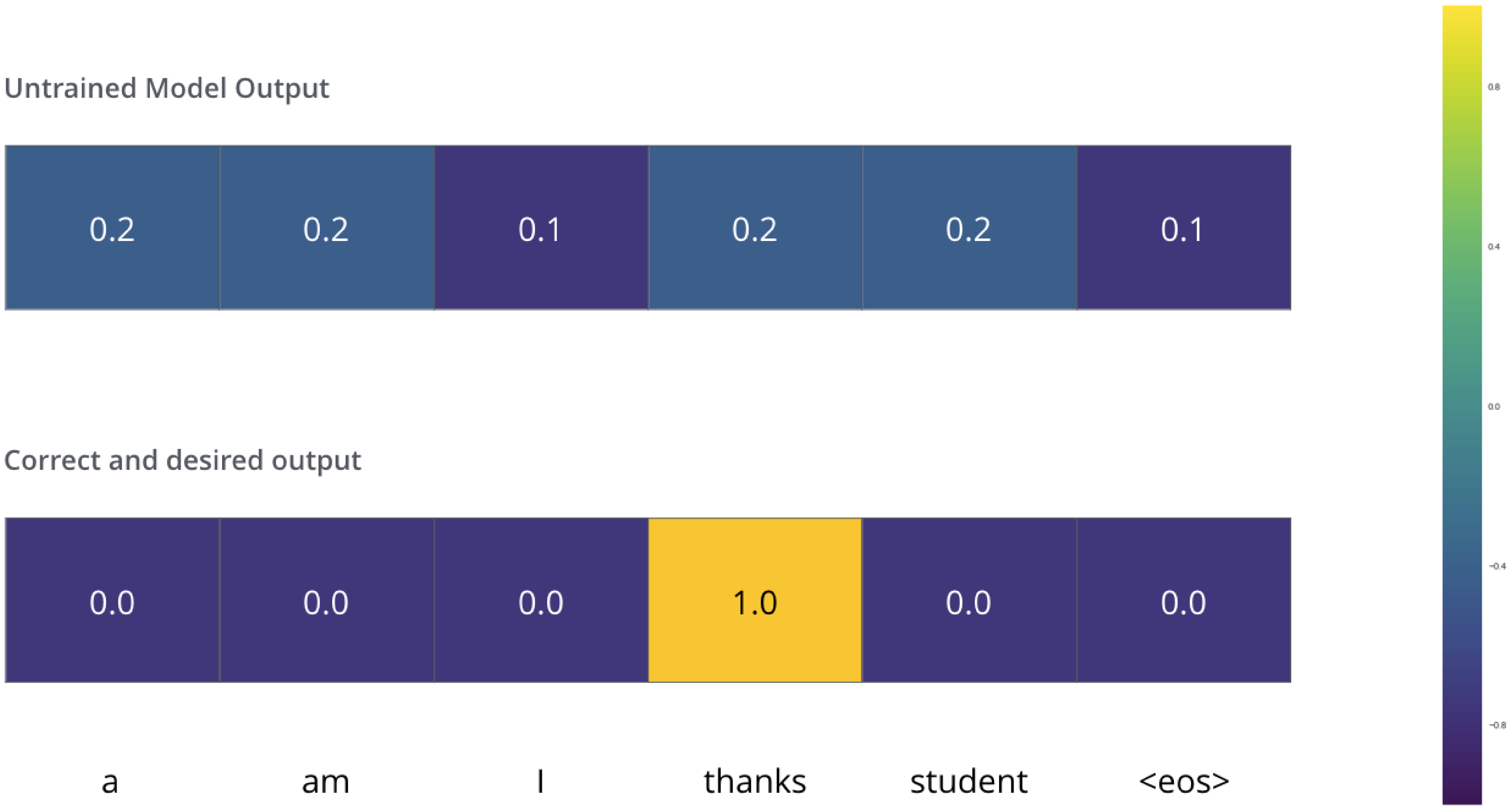


Decoder’s Output Linear Layer




Decoding time step: 1 2 3 4 5 6
OUTPUT
I
am
Linear + Softmax
Kencdec
Vencdec
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS OUTPUTS
I

Training and the Loss Function



We can use cross Entropy.

We can also optimize two words at a time: using BEAM search: keep a few alternatives for the first word.

NAC

NAC

NACC

NAC

NAC

NAC

NAC

NACC

NAC

NAC

NAC

NACC

NAC

NAC

NAC

NAC

NAC

NACC

NAC

NAC

NAC

NACC

NAC

NAC

NACC

NACC