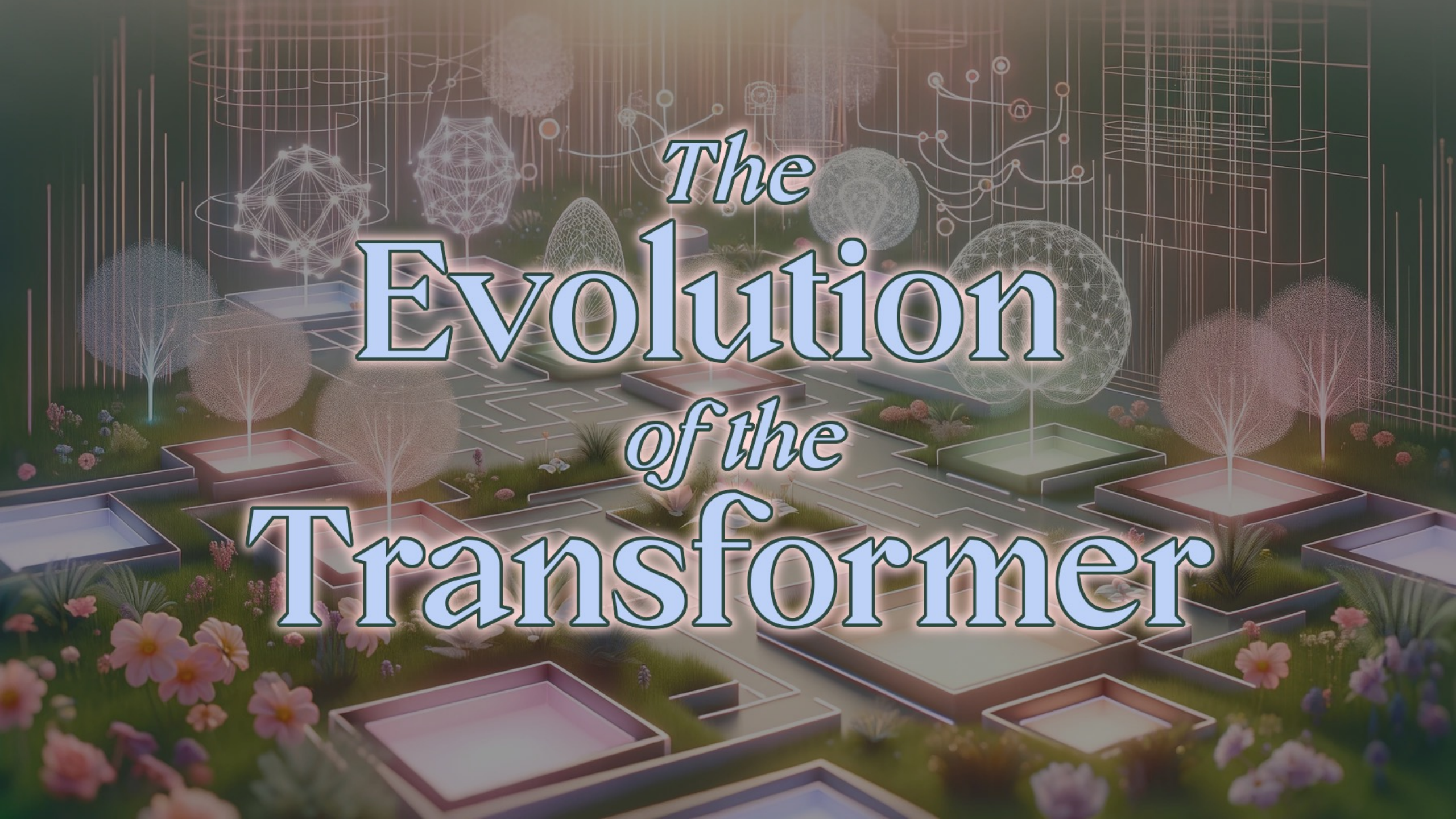
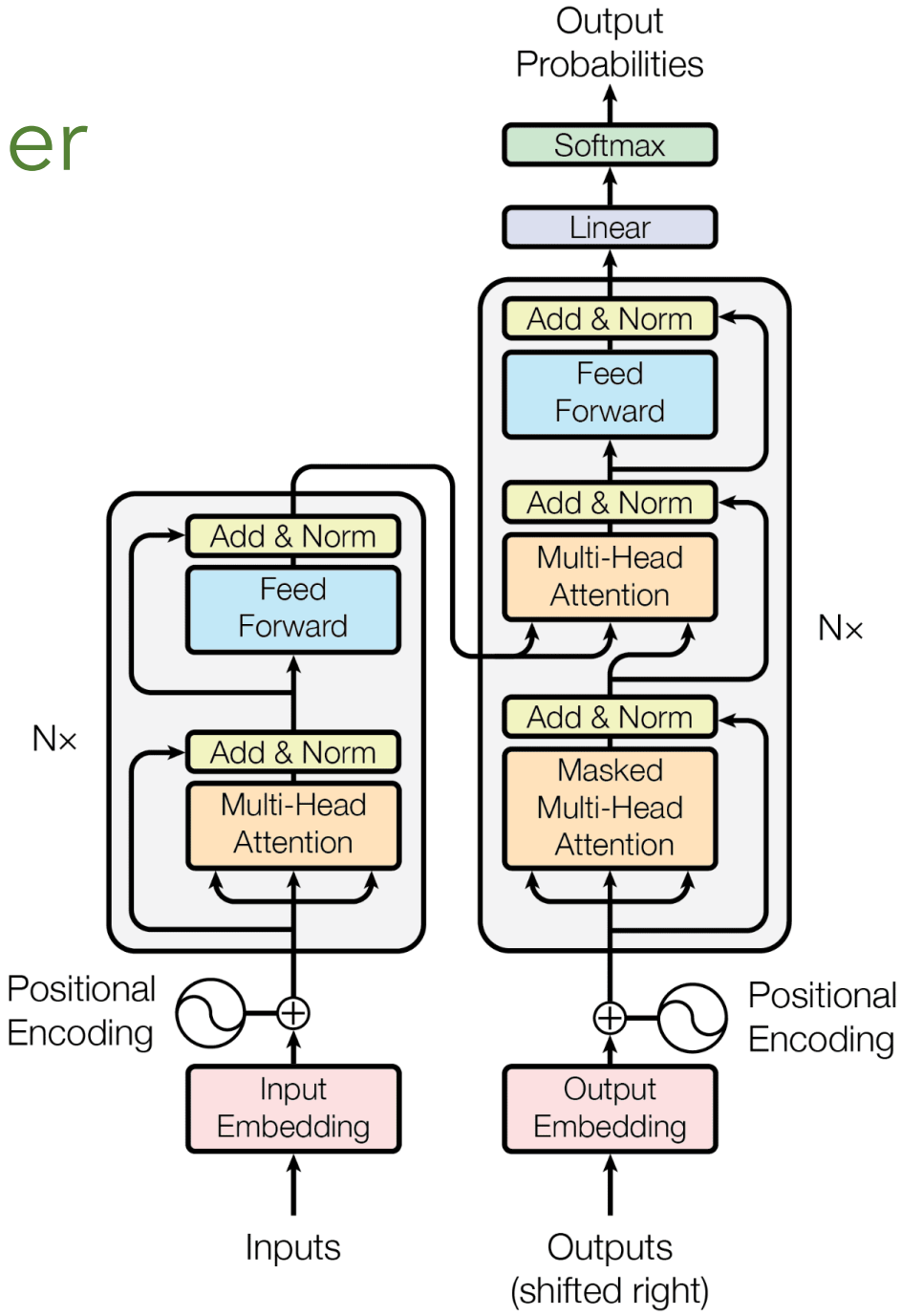
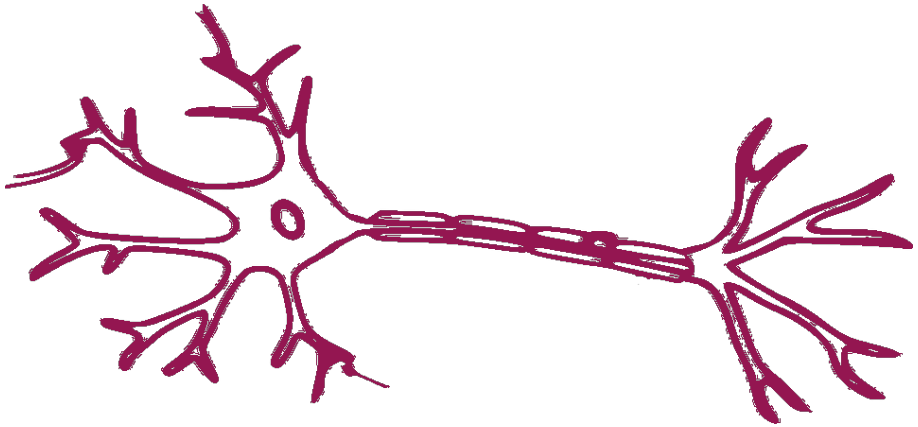

The
Evolution
of the
Transformer

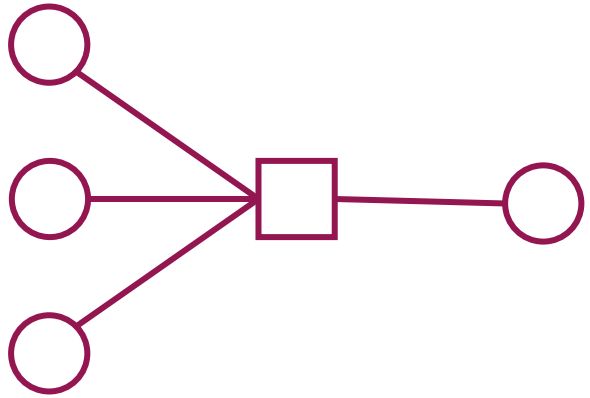
The Transformer



1943: The "Artificial Neuron"

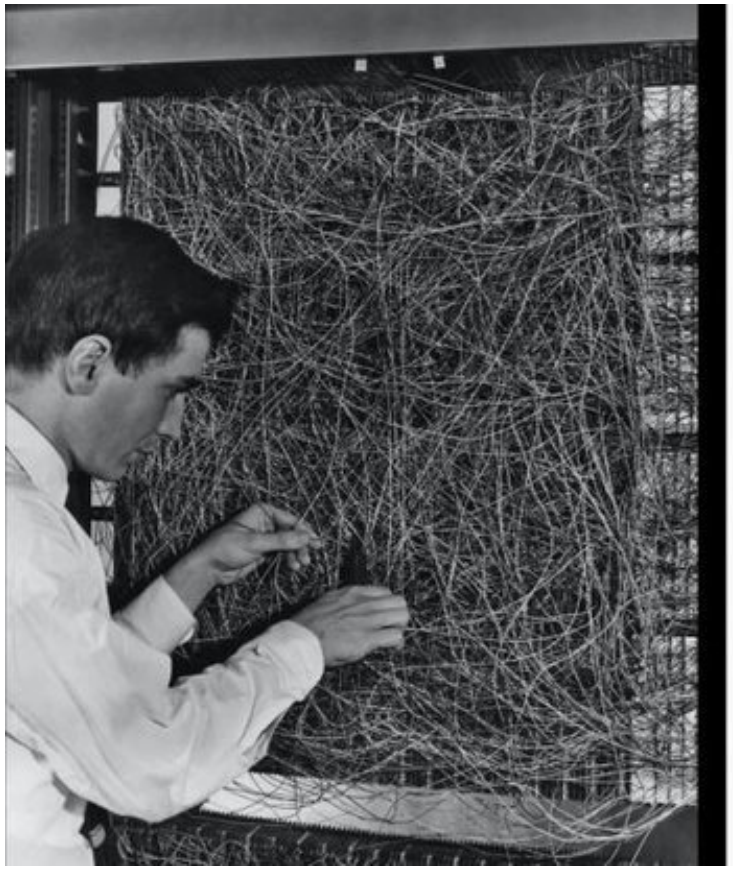
Warren McCulloch and Walter Pitts





1957: The Perceptron

Frank Rosenblatt



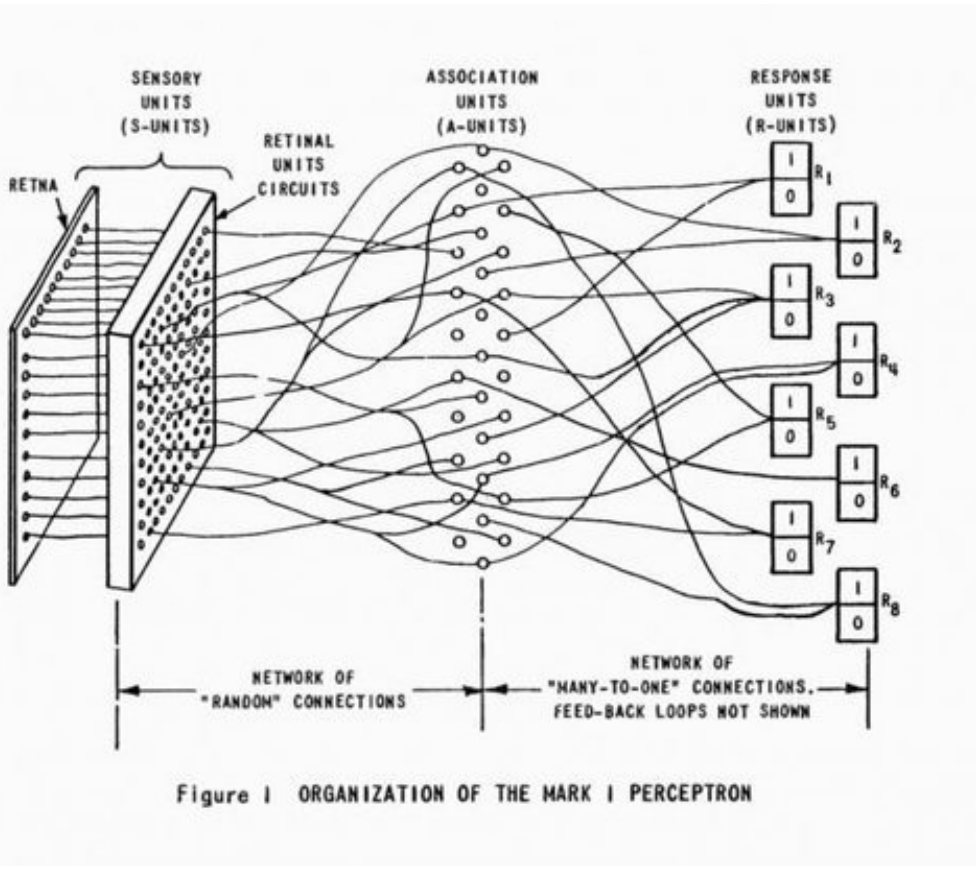
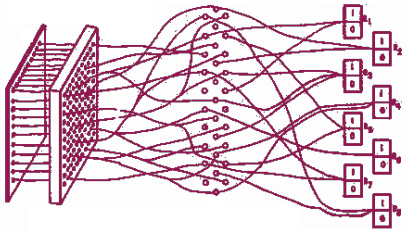


Figure 1 ORGANIZATION OF THE MARK I PERCEPTRON

1943:
The “Artificial Neuron”

1957: The ”Perceptron”



1943:
The “Artificial Neuron”

1969: Perceptrons

Marvin Minsky and Seymour Papert

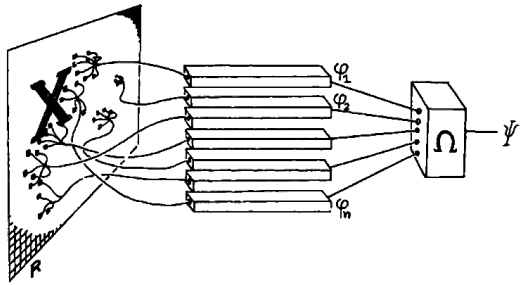


Figure 0.1

1970: Backpropagation

Seppo Linnainmaa

TAYLOR EXPANSION OF THE ACCUMULATED ROUNDING ERROR

*T*1. [Initialize.] $p \leftarrow n$, $C[i] \leftarrow 0$ for $i = 1, \ldots, m$, $i \neq N$, $C[N] \leftarrow 1$.
*T*2. [Read.] $i \leftarrow I[p]$, $j \leftarrow J[p]$, $k \leftarrow K[p]$, $dj \leftarrow Dj[p]$, $dk \leftarrow Dk[p]$.
*T*3. [Coefficient $c_{n,p}$ completed.] coef $\leftarrow C[i]$, $C[i] \leftarrow 0$. The contents of coef is now equal to the value of $c_{n,p}$ and can be utilized.
*T*4. [Coefficient zero?] If coef $= 0$, go to step *T*6.
*T*5. [Update table *C*.] If $dj \neq 0$ then $C[j] \leftarrow C[j] + \text{coef} \times dj$. If $dk \neq 0$ then $C[k] \leftarrow C[k] + \text{coef} \times dk$.
*T*6. [Decrease *p*.] Decrease p by 1. If $p > 0$, return to step *T*2, otherwise the algorithm terminates.

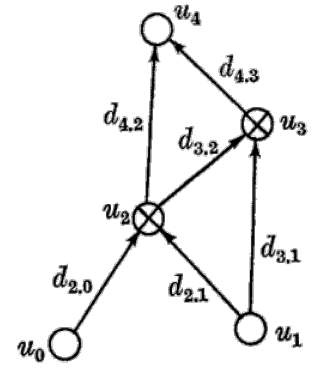
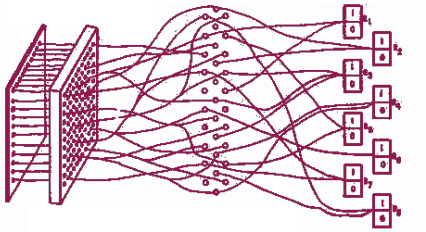
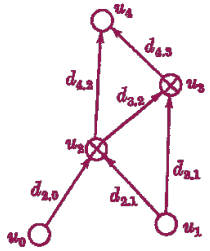


Figure 1. A computing process as a graph.

1957: The ”Perceptron”

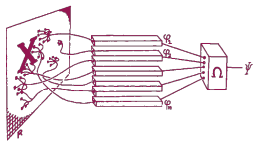


1970:
Backprop
proposed



1943:
The “Artificial Neuron”

1969:
Perceptrons

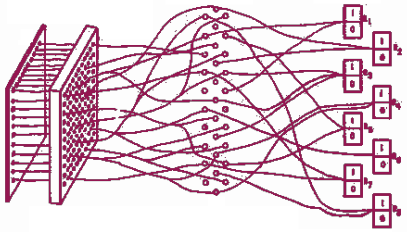


1970s & 1980s:
“AI Winter”

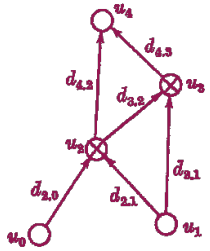
1986: Applied backprop

D Rumelhart, G Hinton, R Williams

1957: The ”Perceptron”

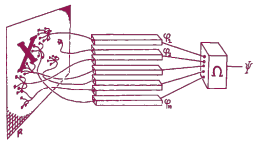


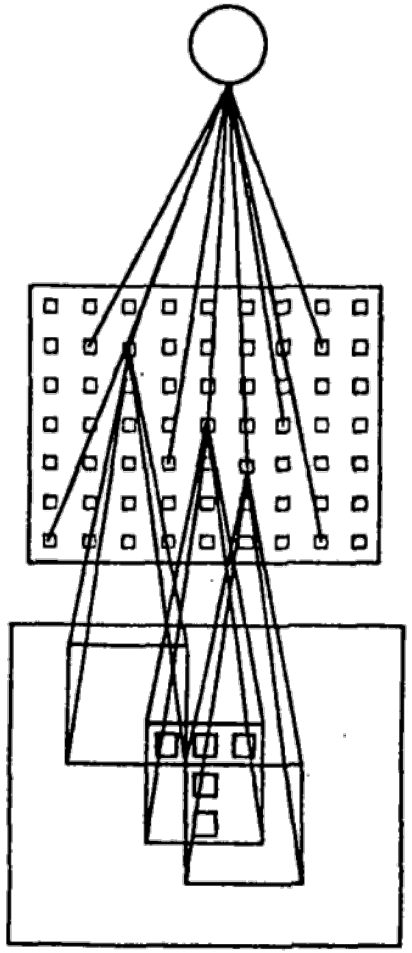
1970: Backprop proposed



1943: The “Artificial Neuron”

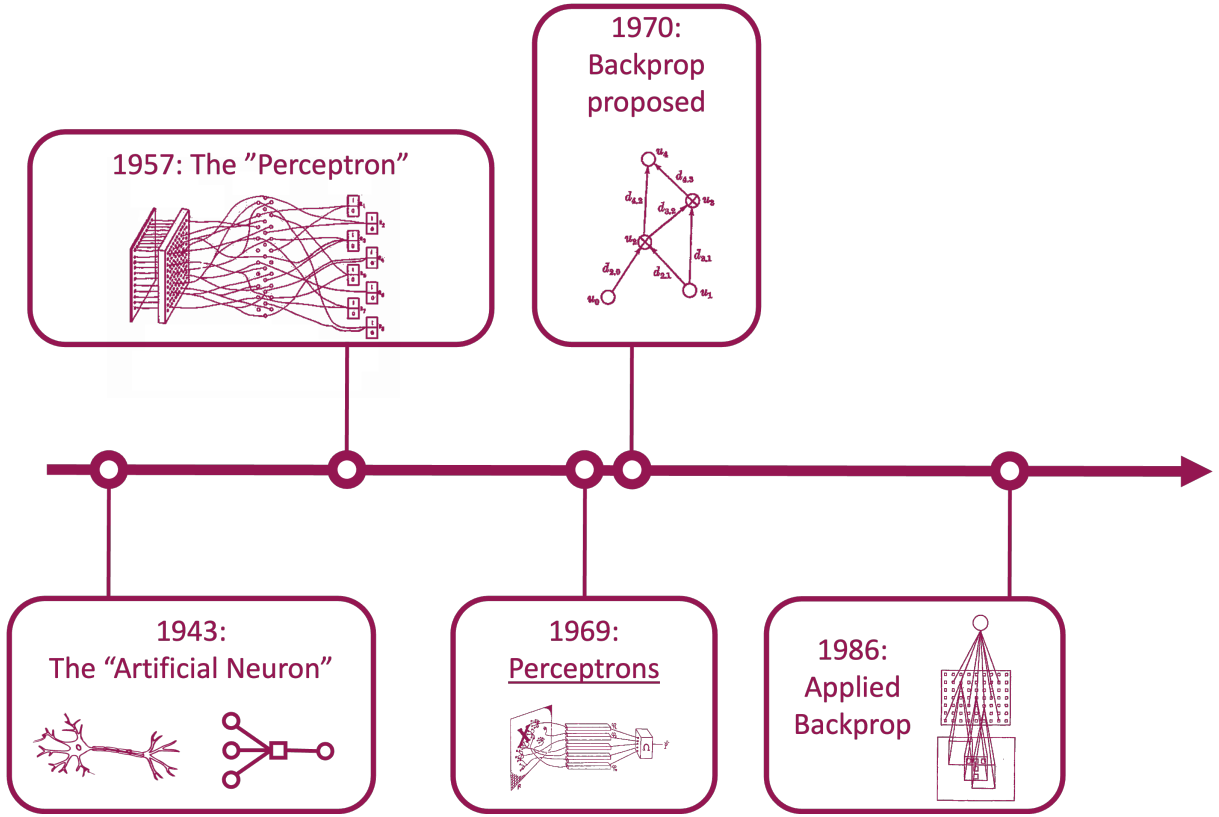
1969: Perceptrons





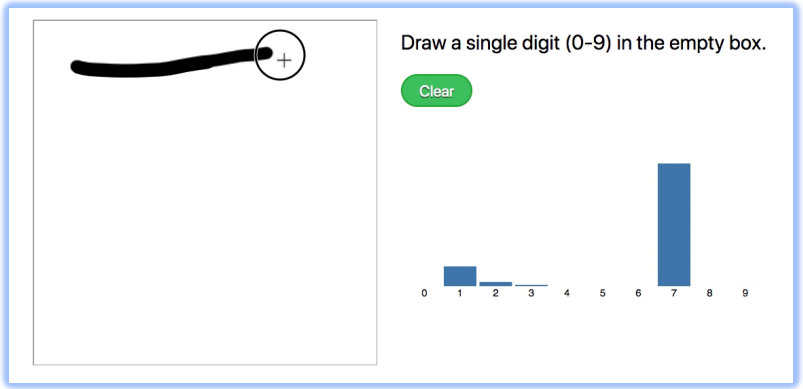
Neural Networking in the 20th Century

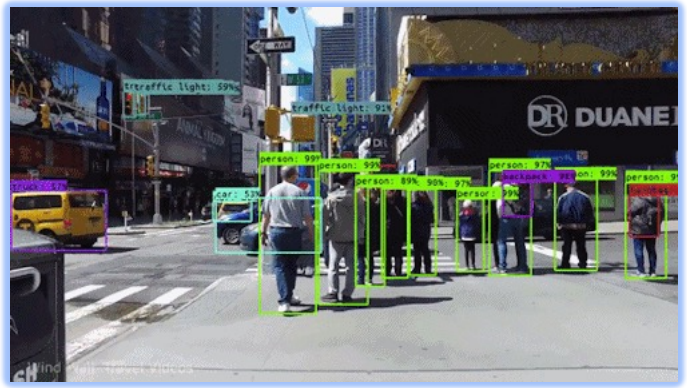
- **Linear & Nonlinear**: Composition of layers for computation
- **Trainable**: backpropagation

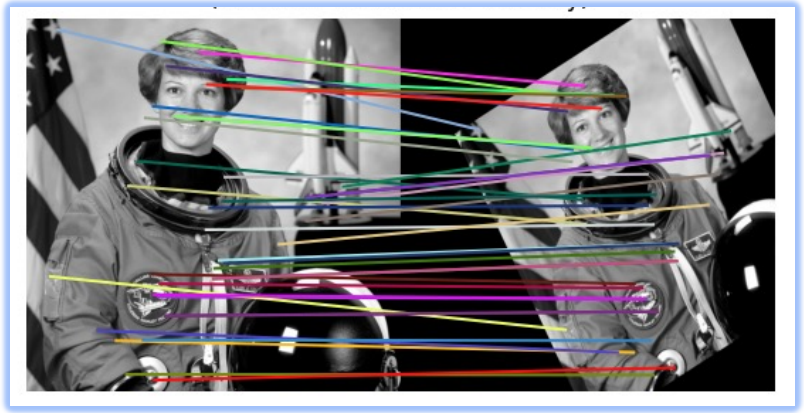


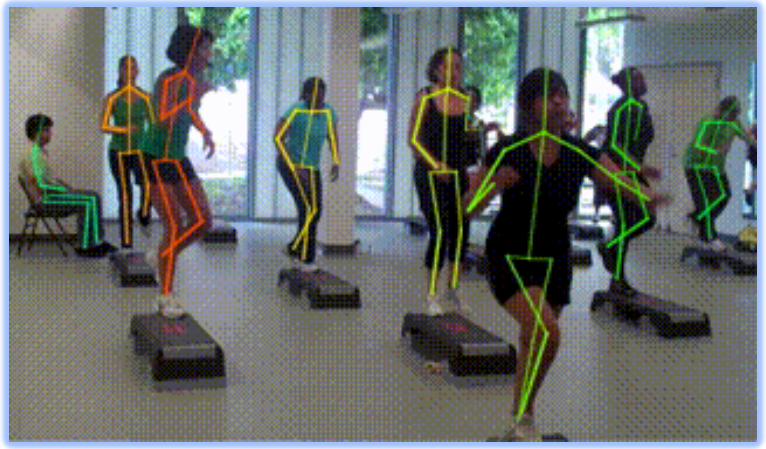
Computer Vision

A multitude of interesting challenges







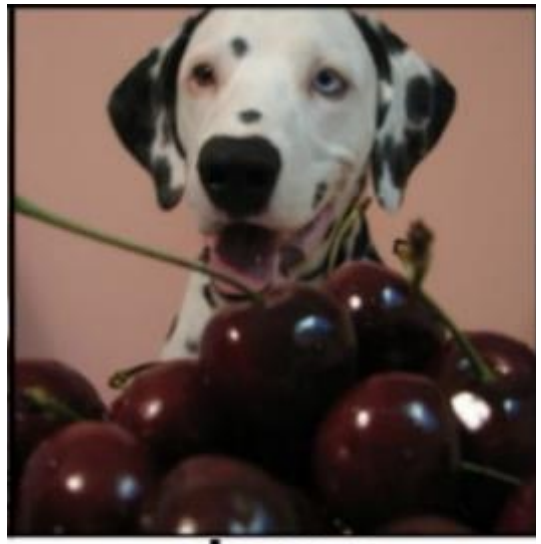


ImageNet & ILSVRC (2009)

1,000 image categories

1.2 million images for training

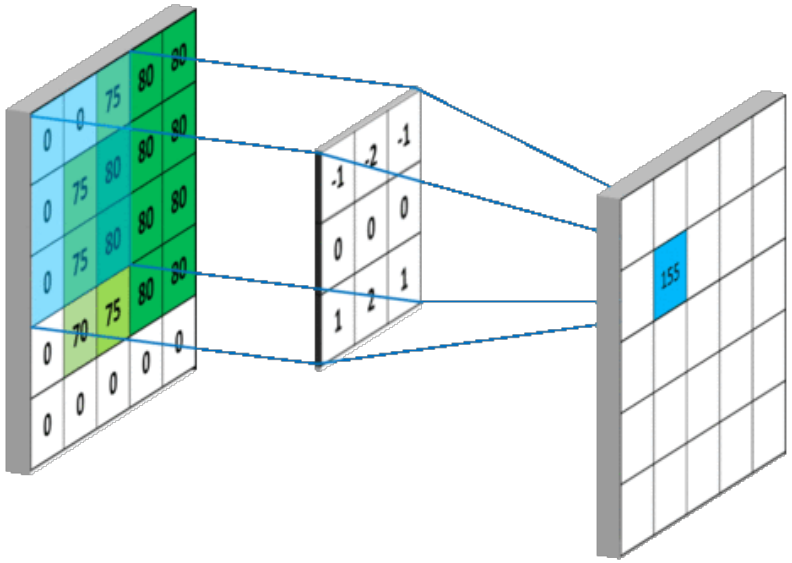
100k images for evaluation

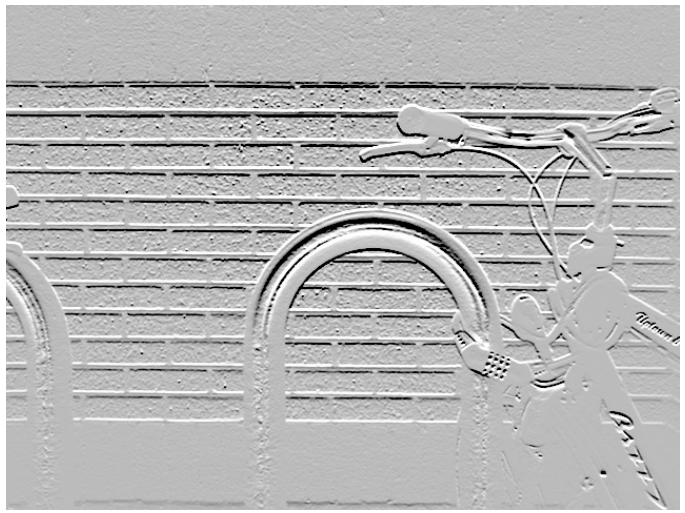


Filters

1	2	1
0	0	0
-1	-2	-1

Horizontal
Edge Detector

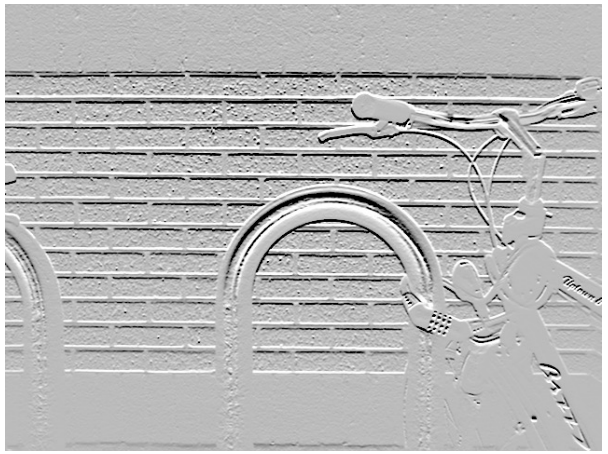




Filters

1	2	1
0	0	0
-1	-2	-1

Horizontal
Edge Detector

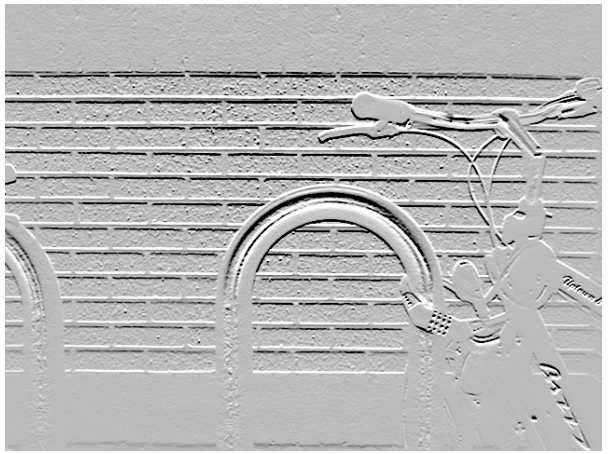


```
Gy = np.array([[1, 2, 1],
               [0, 0, 0],
               [-1, -2, -1]])
rows, columns = np.shape(grayscale_image)
sobel_filtered_image = np.zeros(shape=(rows, columns))

for i in range(rows - 2):
    for j in range(columns - 2):
        gy = np.sum(np.multiply(Gy, grayscale_image[i:i+3, j:j+3]))
        sobel_filtered_image[i + 1, j + 1] = gy
```

Filters

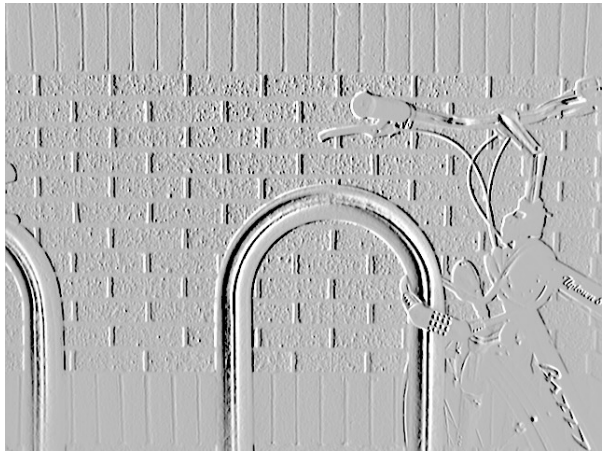
1	2	1
0	0	0
-1	-2	-1

Horizontal
Edge Detector

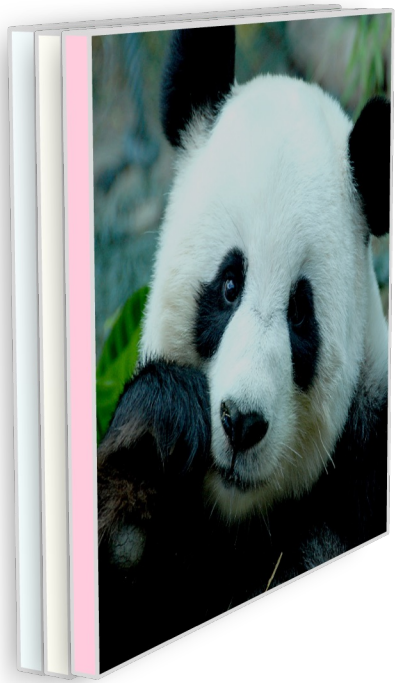


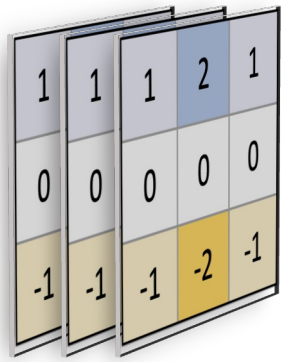
1	0	-1
2	0	-2
1	0	-1

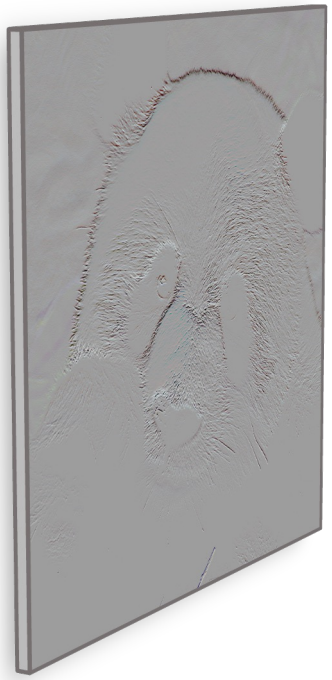
Vertical
Edge Detector



Filters

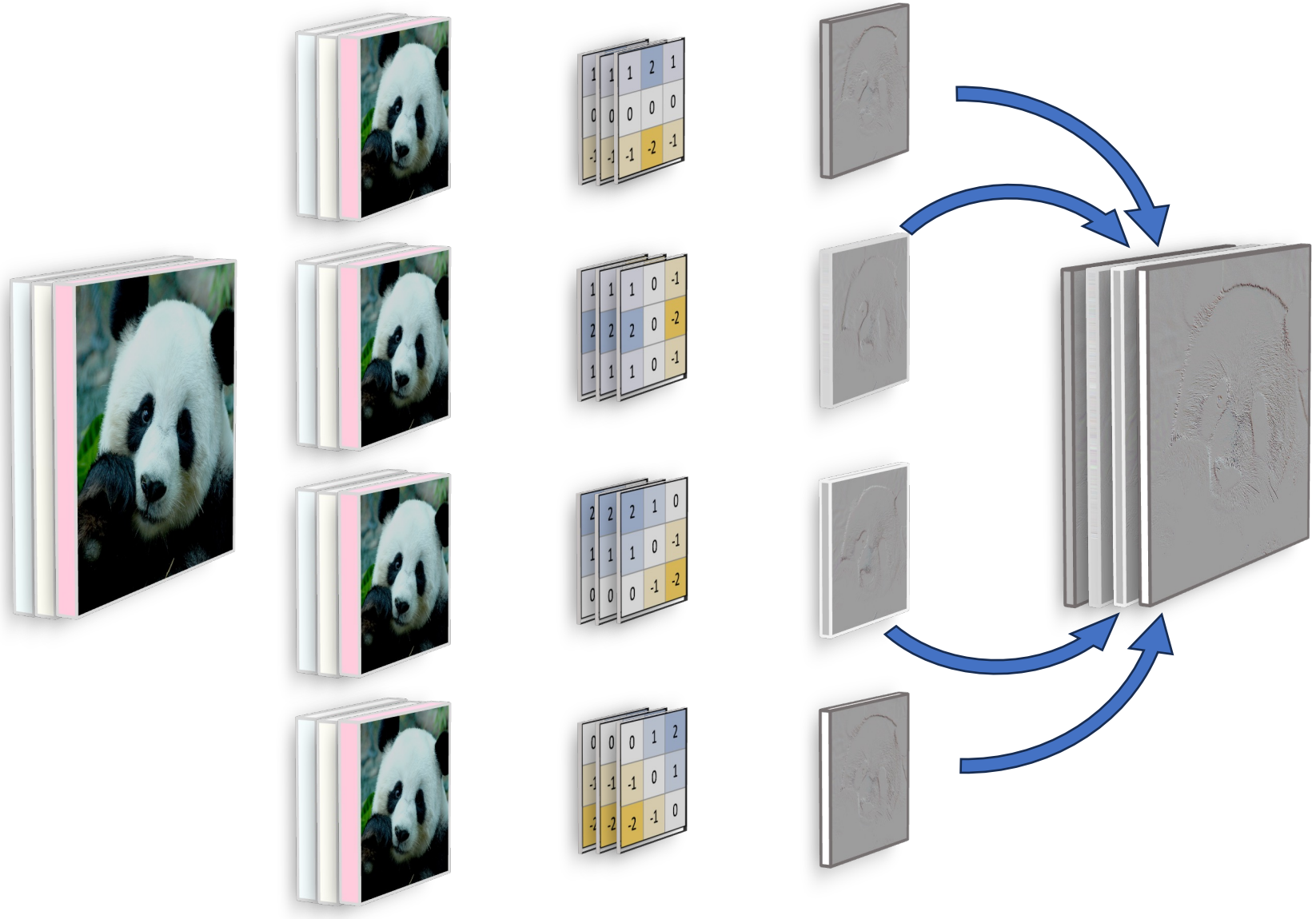




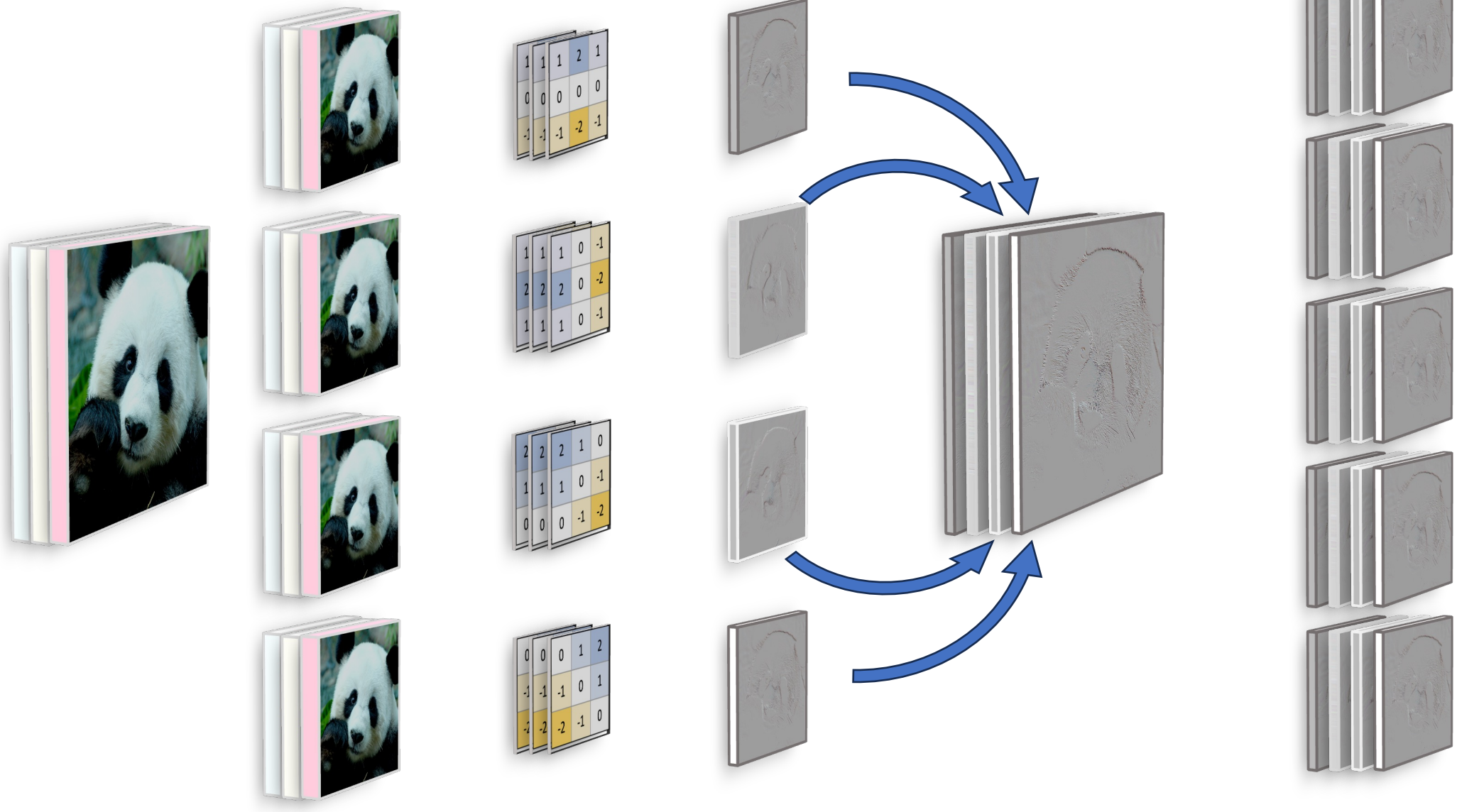


Filters

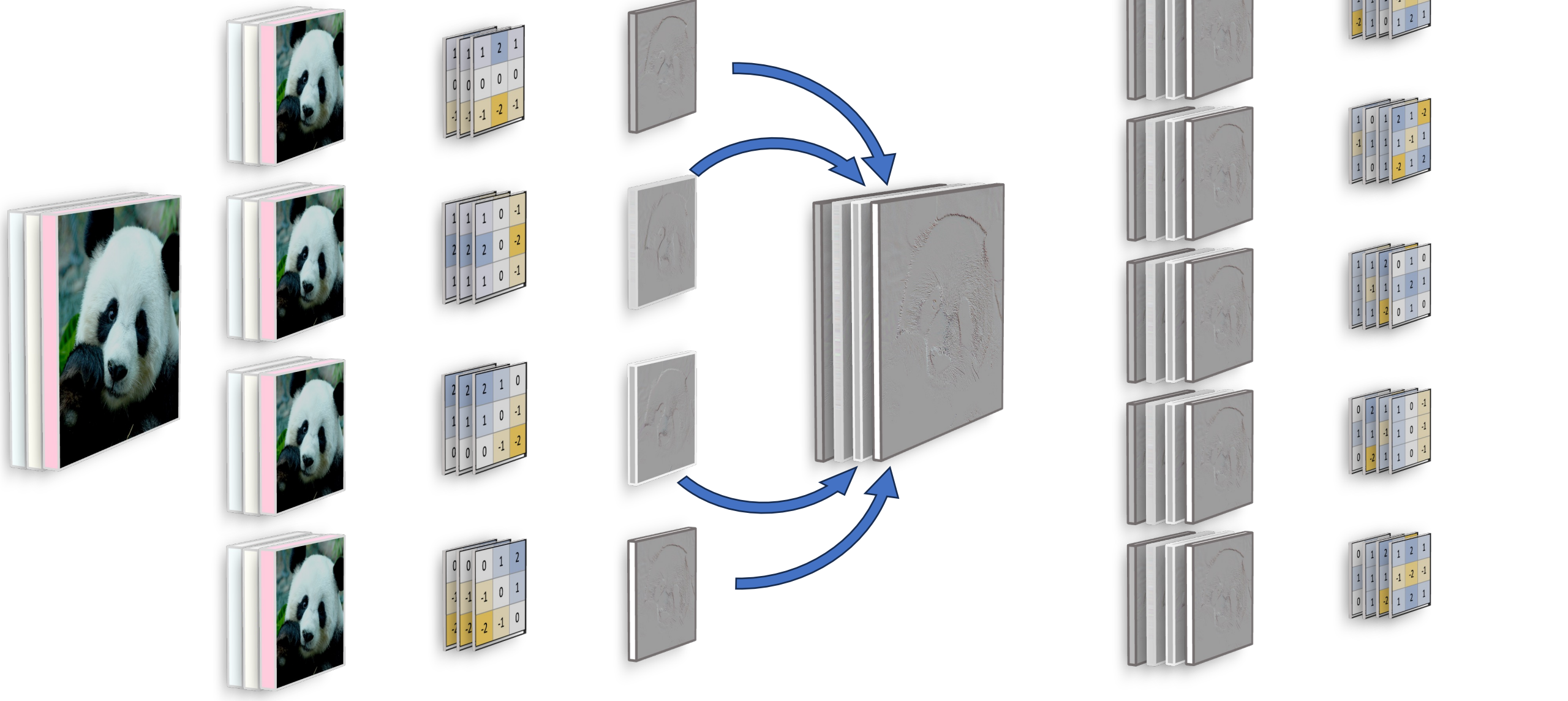
Filters



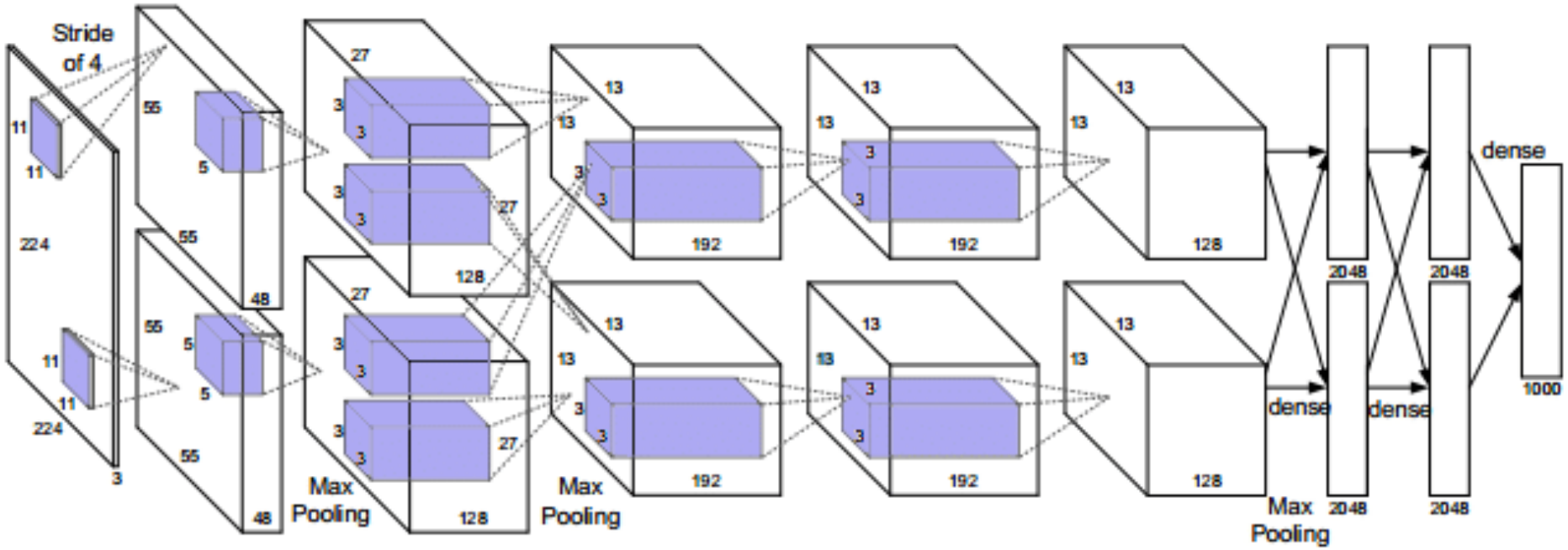
Filters



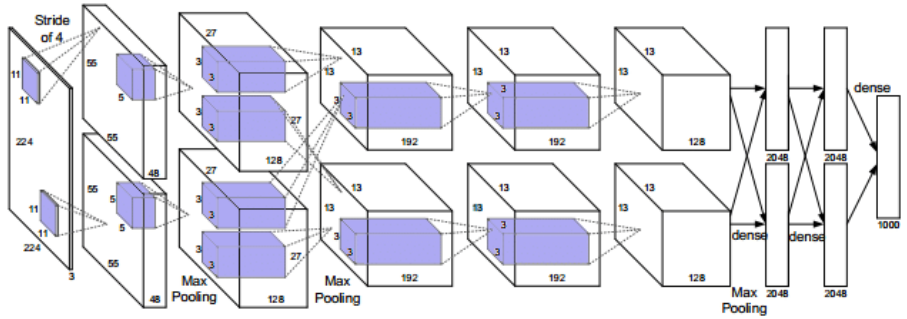
Filters



AlexNet (2012)

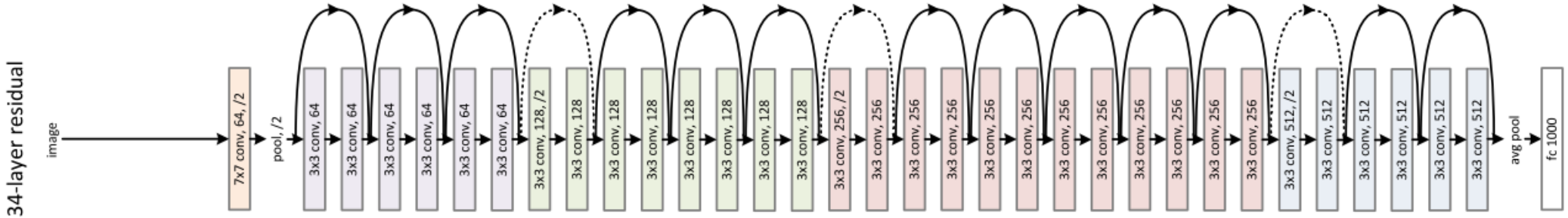


AlexNet (2012)

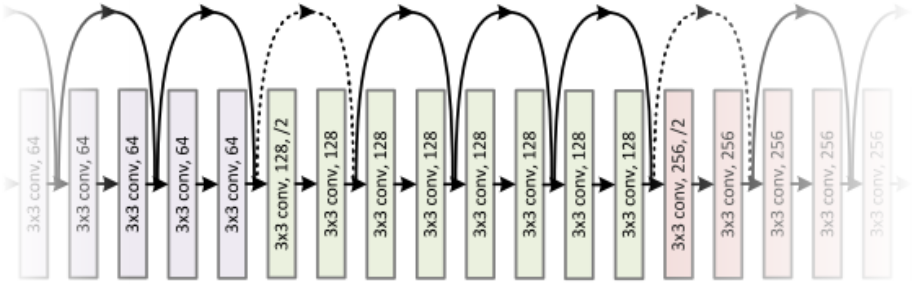



```python
import torch.nn as nn

class AlexNet(nn.Module):
    def __init__(self, num_classes: int = 1000, dropout: float = 0.5) -> None:
        super().__init__()
        _log_api_usage_once(self)
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(p=dropout),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(p=dropout),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

ResNet (2015)

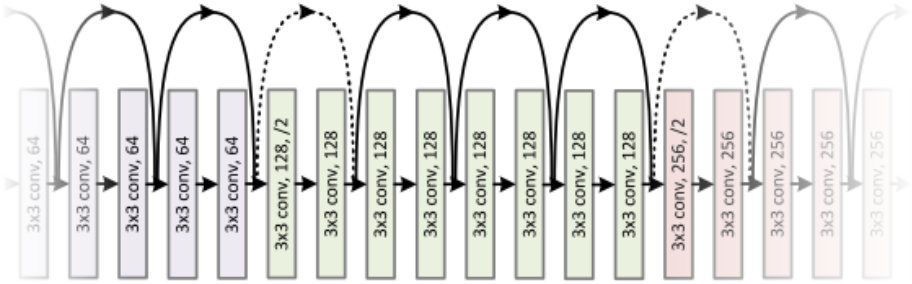

34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 256, /2
3x3 conv, 512, /2
avg pool
fc 1000

ResNet (2015)




```python
class Bottleneck(nn.Module):
    def __init__(self, *args, **kwargs) -> None:
        self.conv1 = conv1x1(inplanes, width)
        self.bn1 = norm_layer(width)
        self.conv2 = conv3x3(width, width, stride, groups, dilation)
        self.bn2 = norm_layer(width)
        self.conv3 = conv1x1(width, planes * self.expansion)
        self.bn3 = norm_layer(planes * self.expansion)
        self.relu = nn.ReLU(inplace=True)
        self.downsample = downsample
        self.stride = stride

    def forward(self, x: Tensor) -> Tensor:
        identity = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)
        out = self.relu(out)

        out = self.conv3(out)
        out = self.bn3(out)

        out += identity
        out = self.relu(out)

        return out
```

ResNet (2015)



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

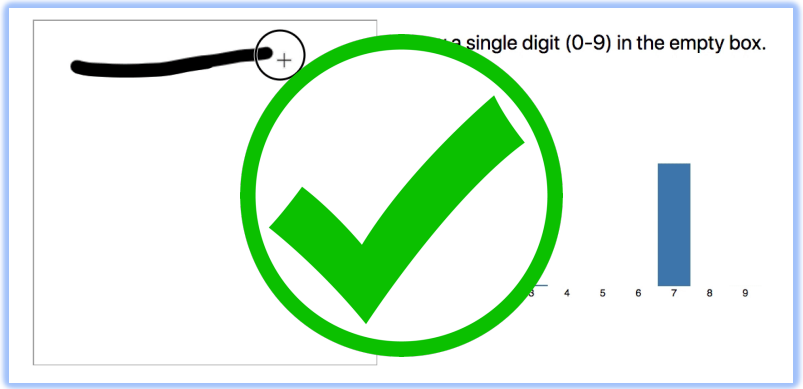
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

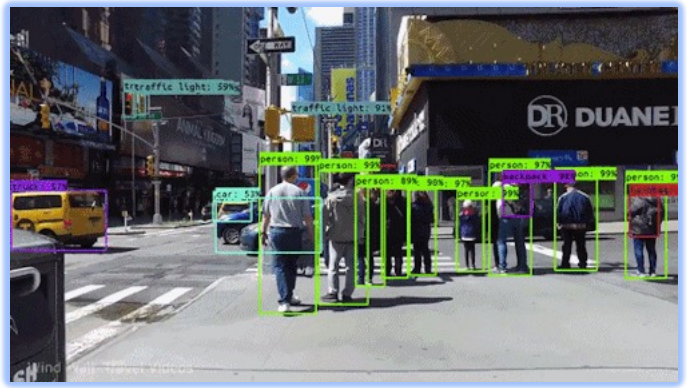
$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

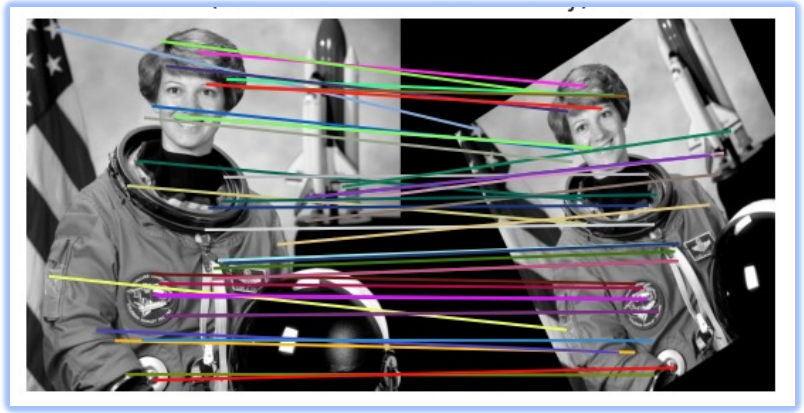
```python
class Bottleneck(nn.Module):
    def __init__(self, *args, **kwargs) -> None:
        self.conv1 = conv1x1(inplanes, width)
        self.bn1 = norm_layer(width)
        self.conv2 = conv3x3(width, width, stride, groups, dilation)
        self.bn2 = norm_layer(width)
        self.conv3 = conv1x1(width, planes * self.expansion)
        self.bn3 = norm_layer(planes * self.expansion)
        self.relu = nn.ReLU(inplace=True)
        self.downsample = downsample
        self.stride = stride

    def forward(self, x: Tensor) -> Tensor:
        identity = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)
        out = self.relu(out)

        out = self.conv3(out)
        out = self.bn3(out)

        out += identity
        out = self.relu(out)

        return out
```

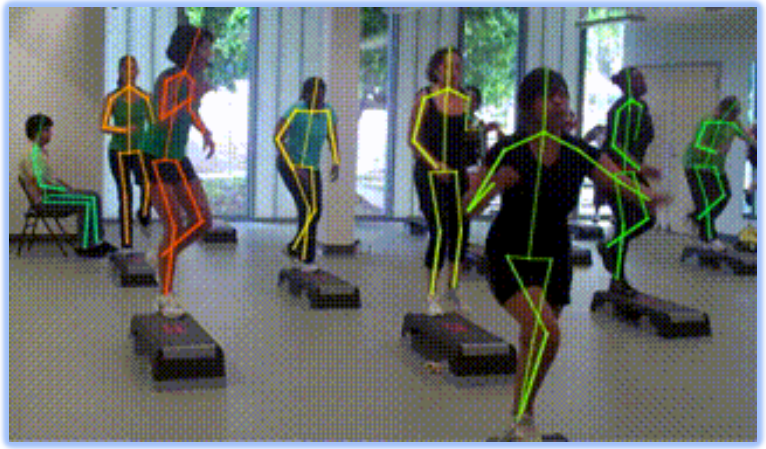
Computer Vision

A multitude of interesting challenges



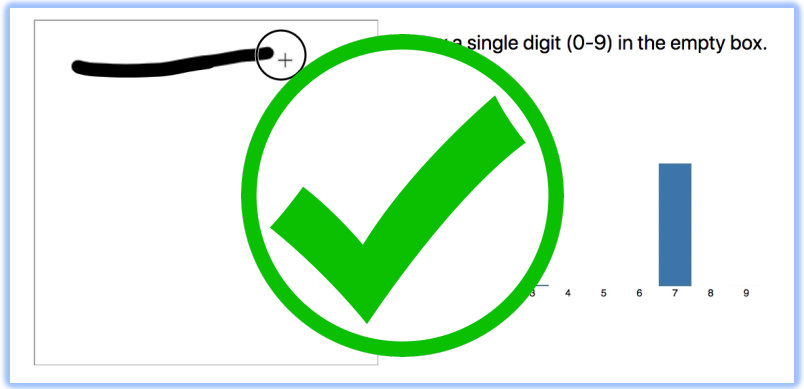


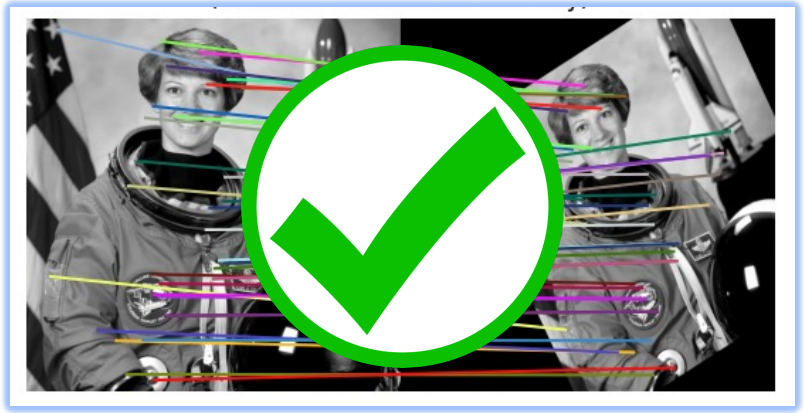




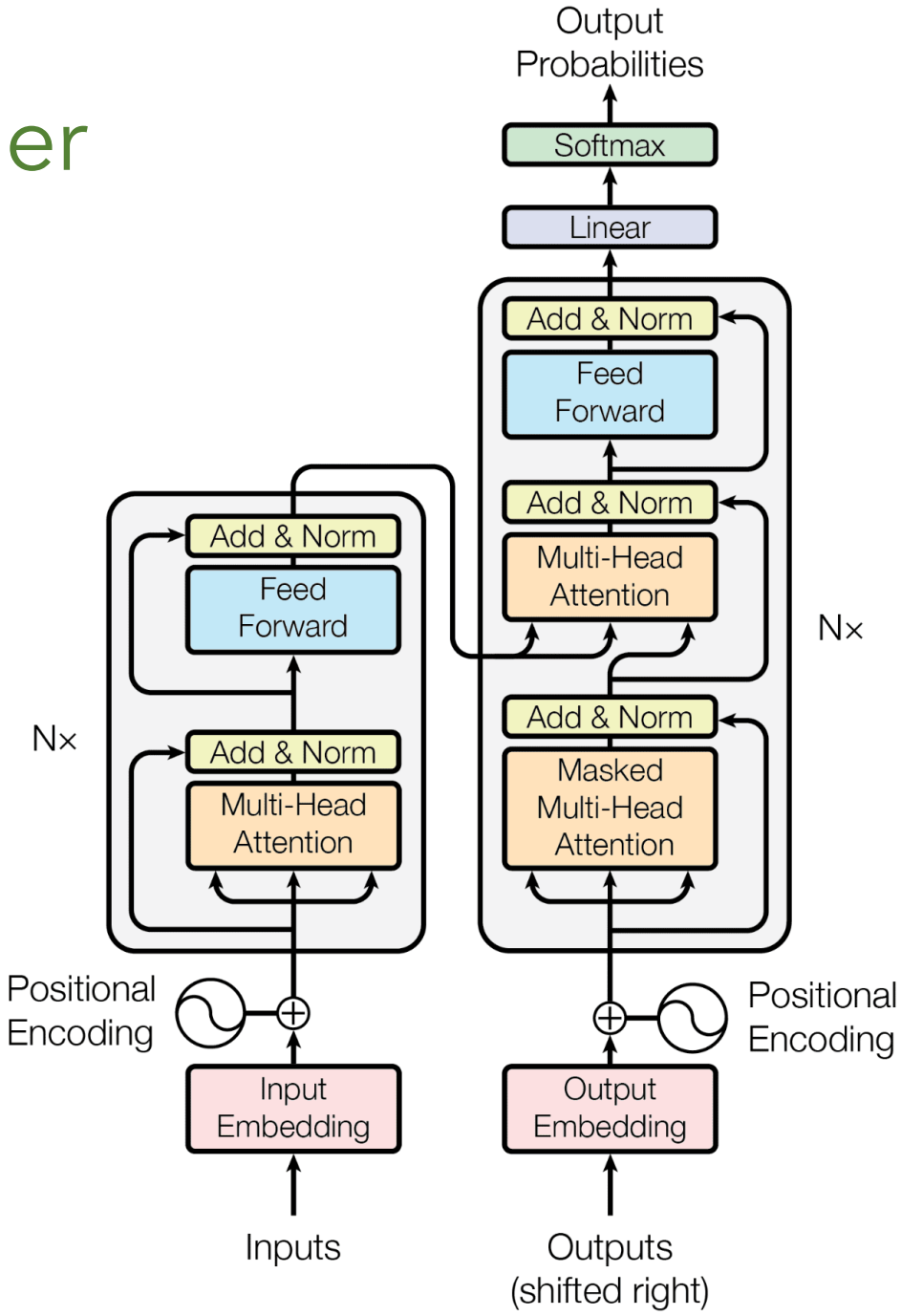
Computer Vision

A multitude of interesting challenges

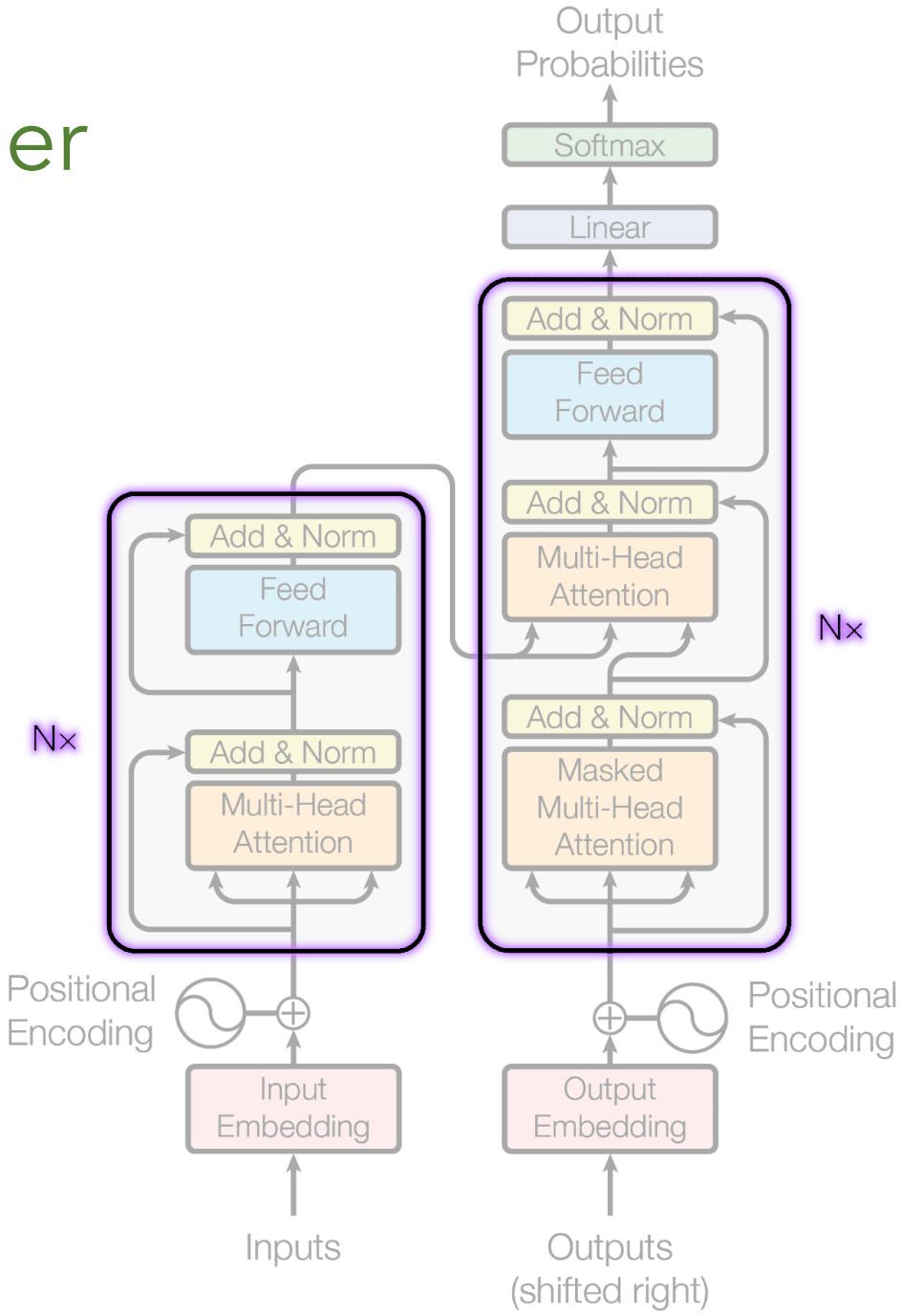




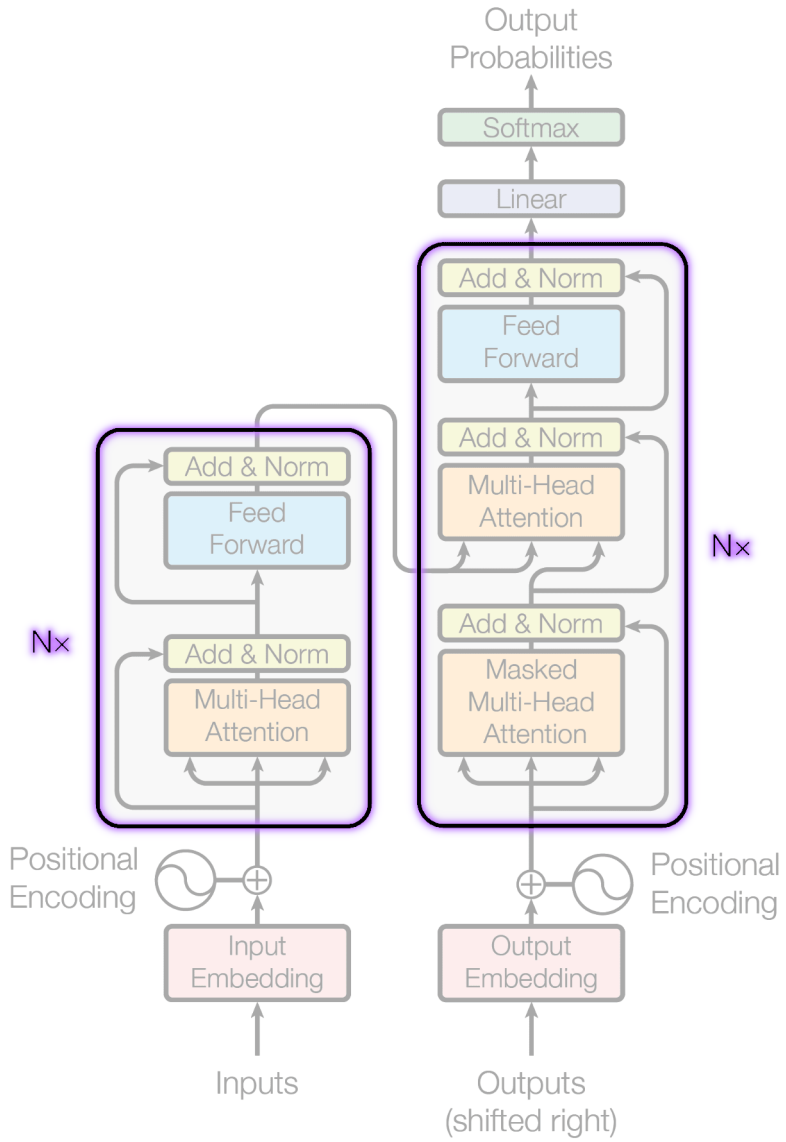
The Transformer

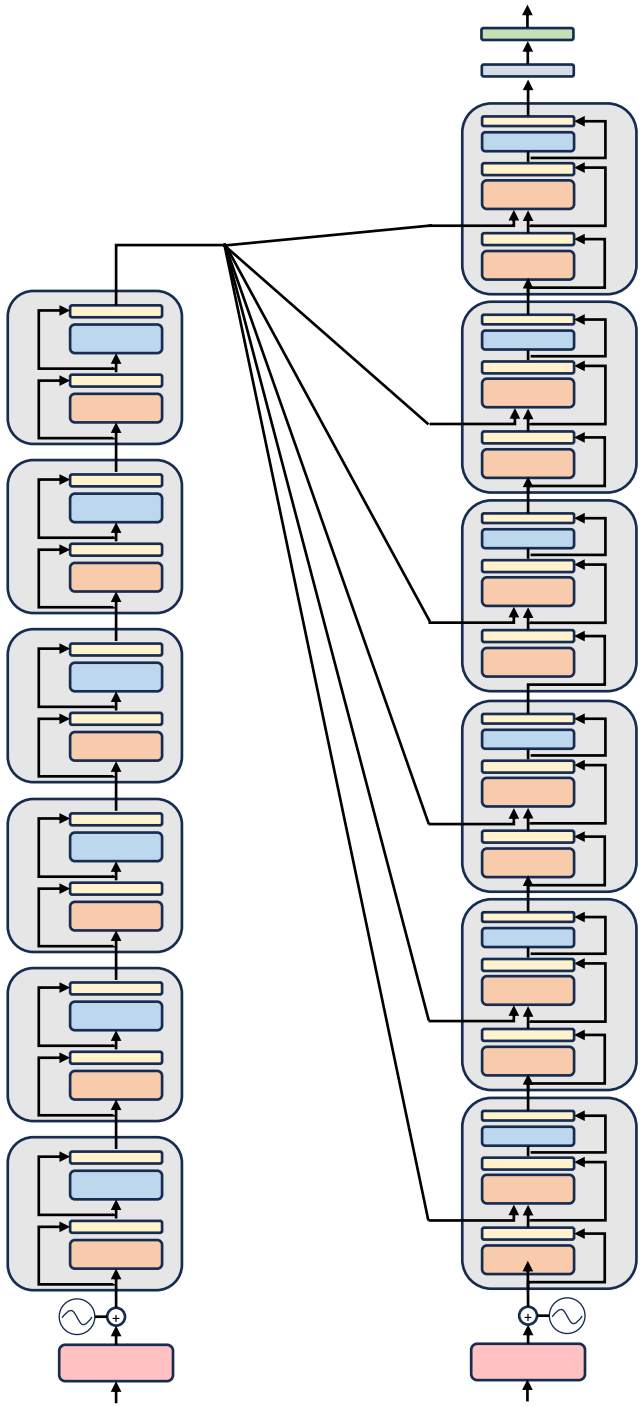


The Transformer

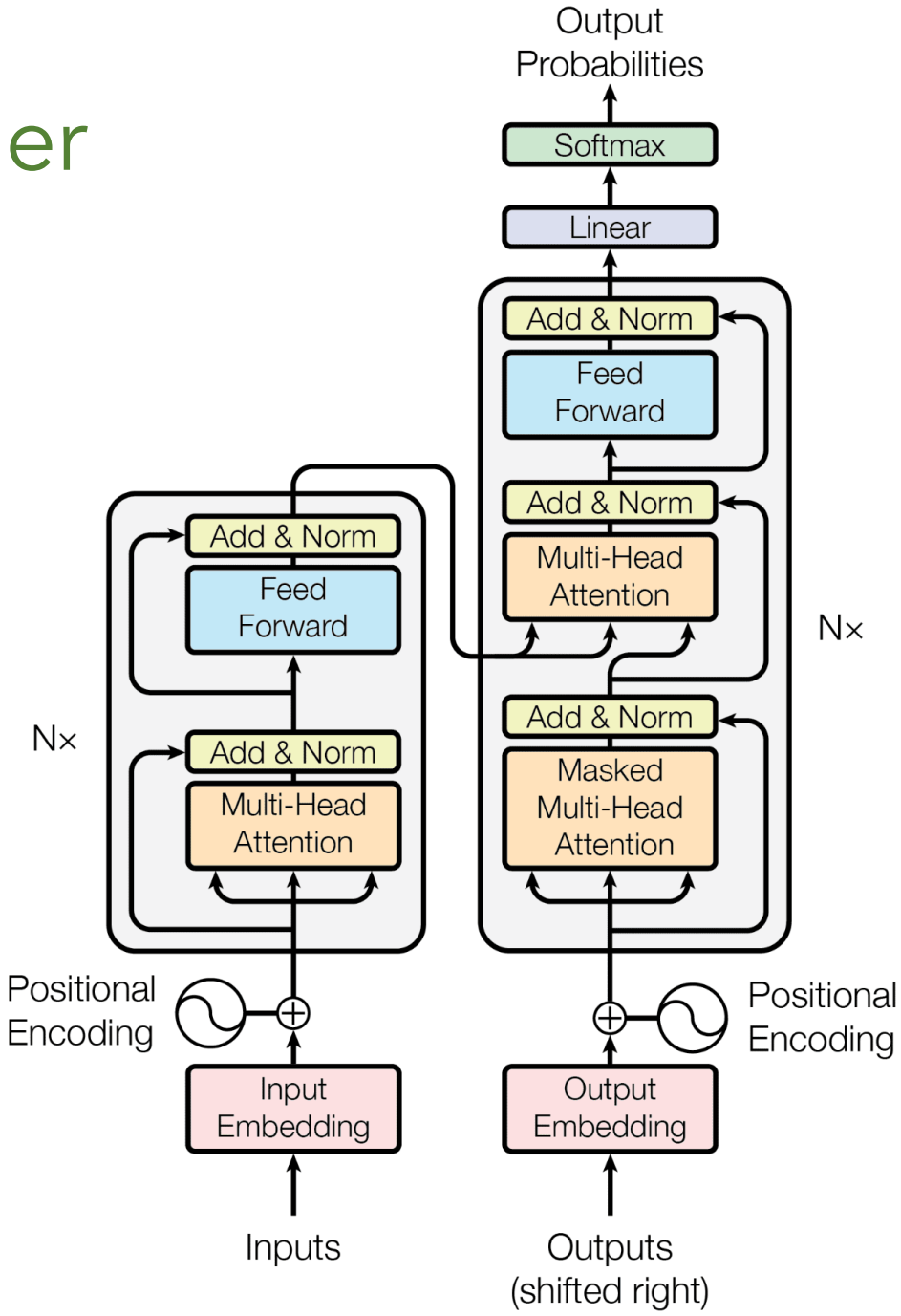


The Transformer

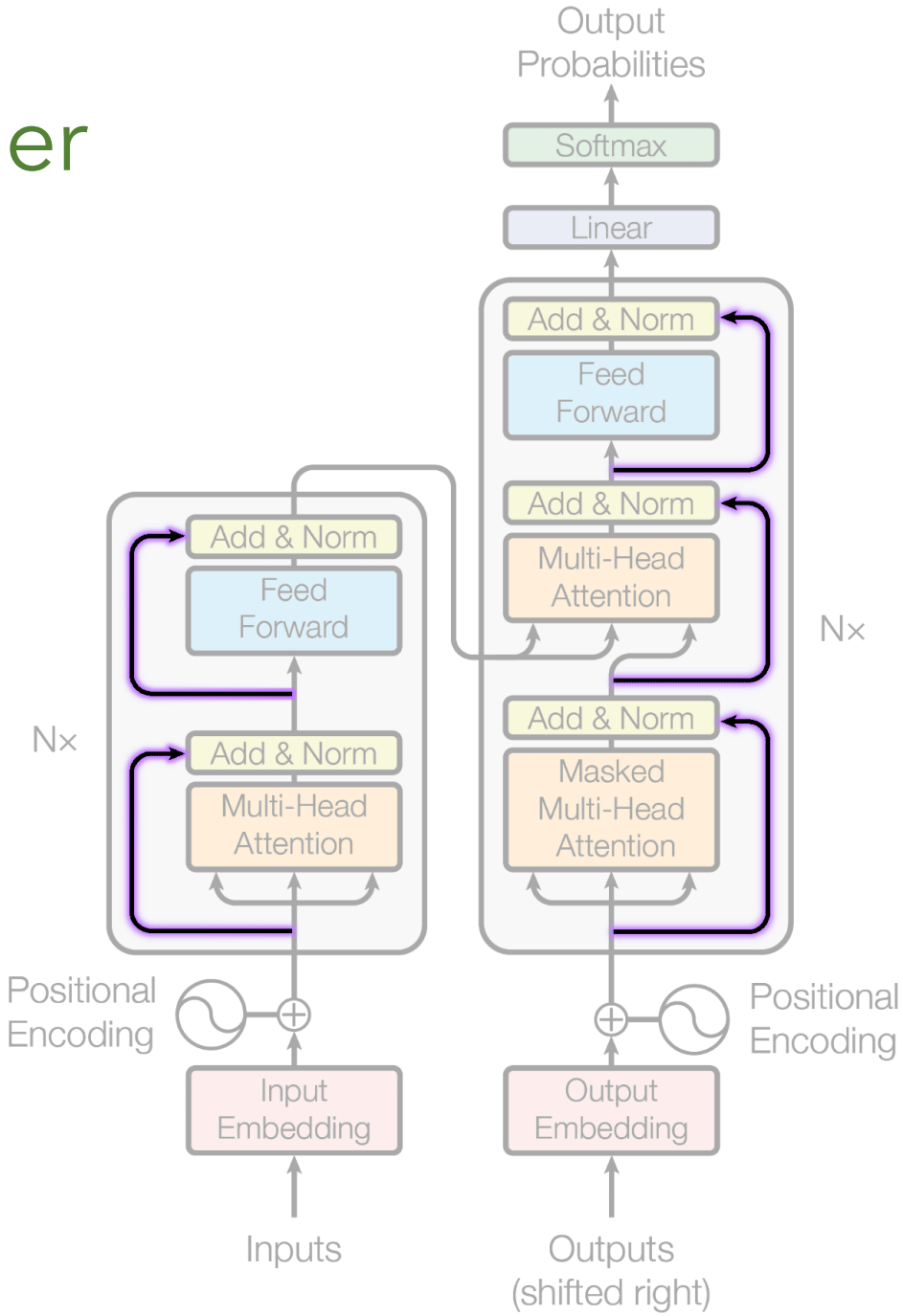




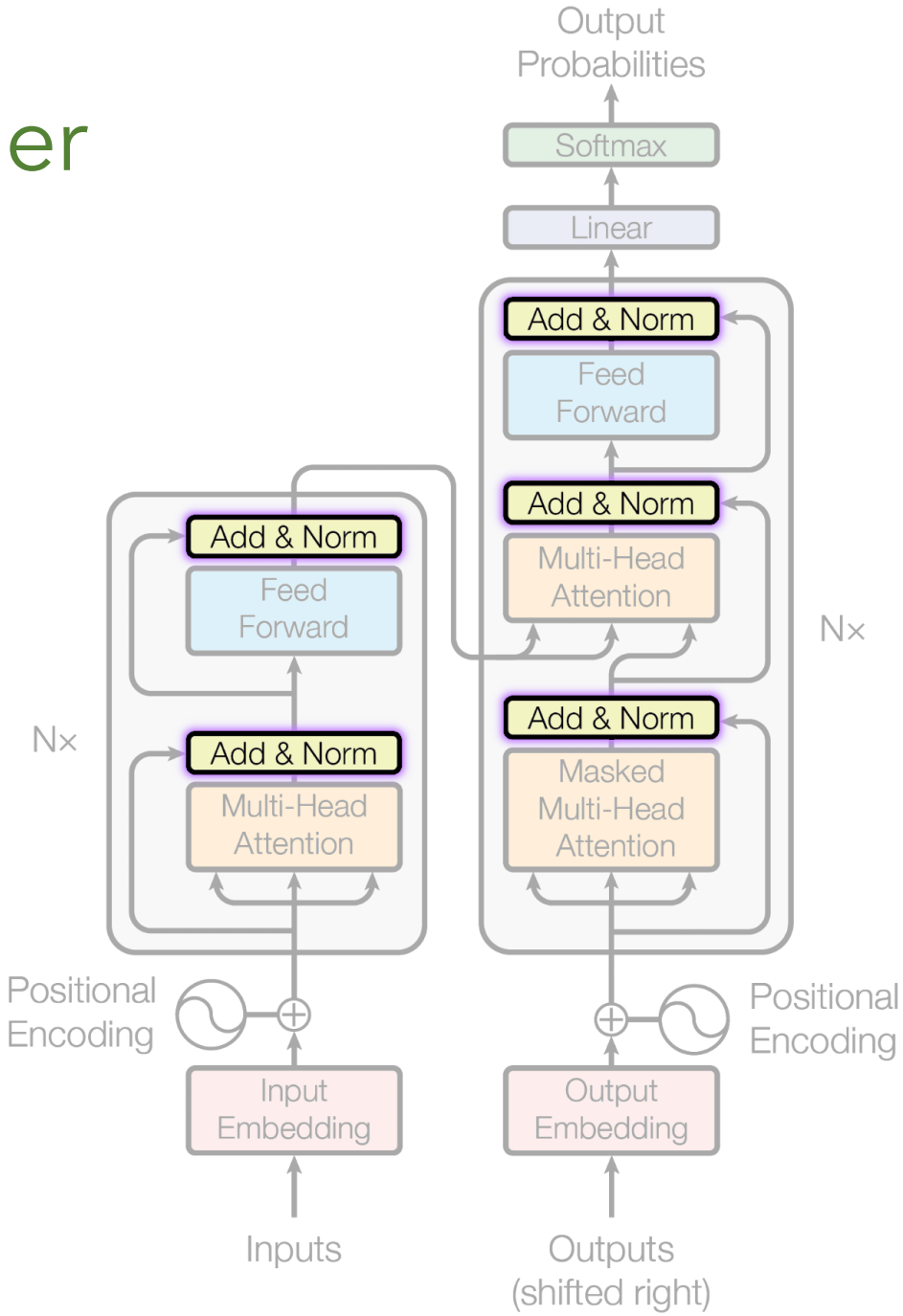
The Transformer



The Transformer



The Transformer



Natural Language Processing

Manipulating Human Language with Computers

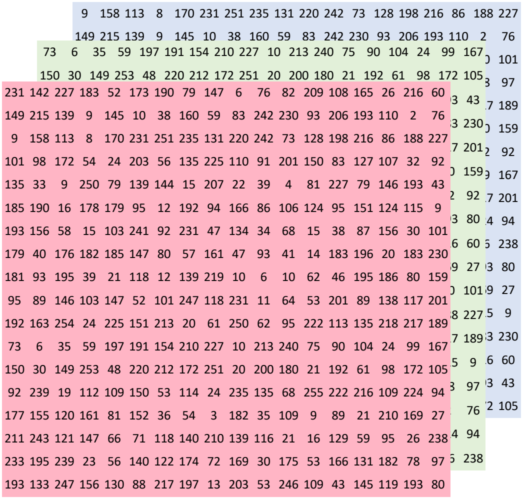
Are there many interesting problems in this space?

Sequence to Sequence Modeling

E.g., Translation

Como un cordero al matadero	→	Like a lamb to the slaughter
Donde hay humo, hay fuego	→	Where there's smoke, there's fire
Nunca es tarde para aprender	→	It's never too late to learn
Que sera, sera	→	Whatever will be, will be

Representation: CV vs NLP

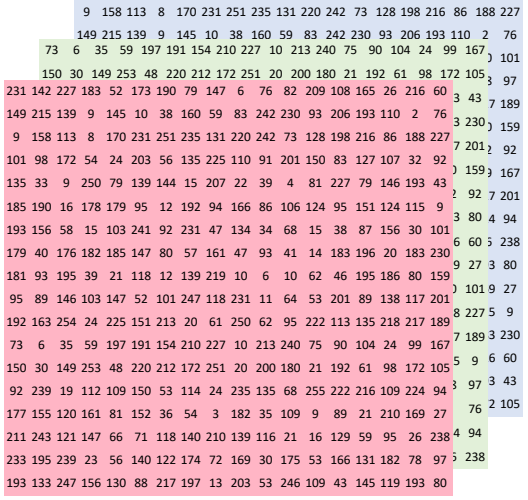


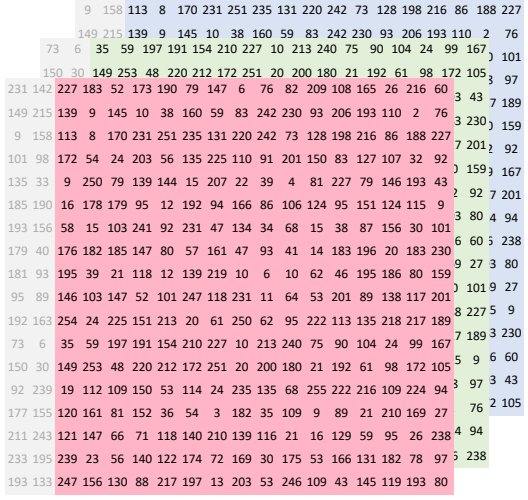
Representation: CV vs NLP

Representation: CV vs NLP

slaughter

73	6C	61	75	67	68	74	65	72





slaughter

73	6C	61	75	67	68	74	65	72

Embedding

Encoding Words into Vectors

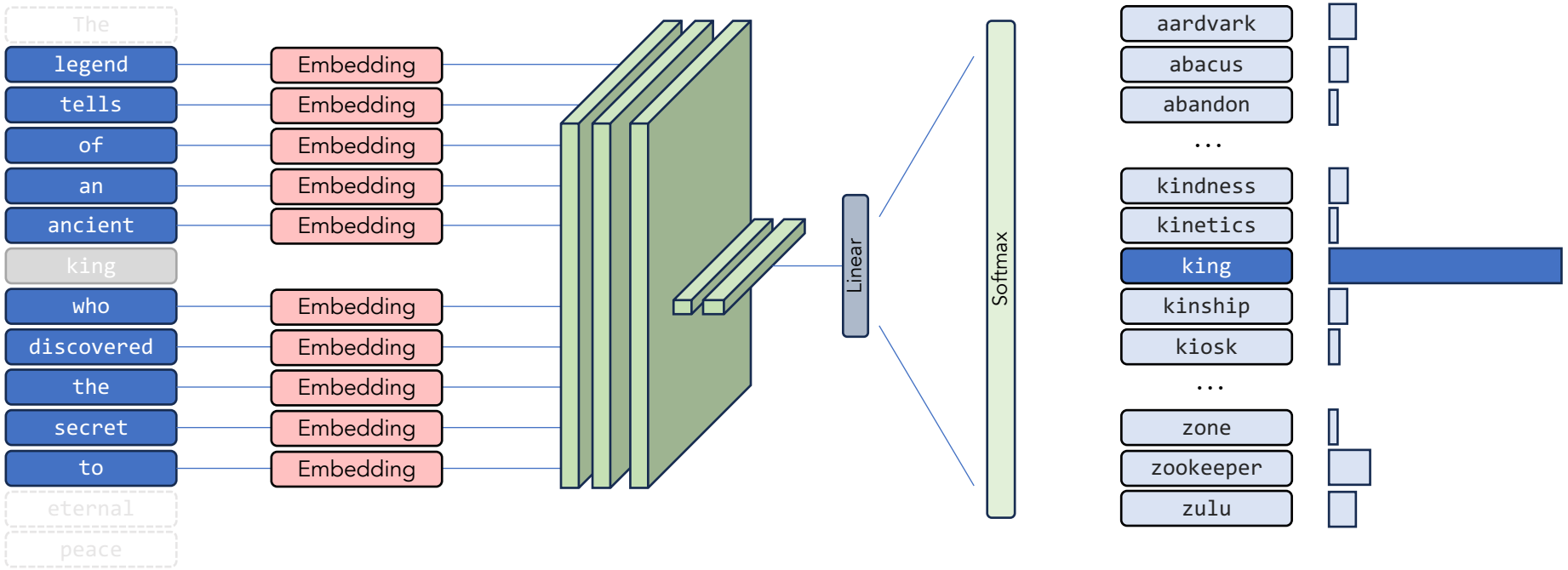
king → `np.array([ 0.603  0.198 -0.585 … -0.074])`

Embedding

Encoding Words into Vectors

king → `np.array([ 0.603  0.198 -0.585 … -0.074])`

"The legend tells of an ancient king who discovered the secret to eternal peace."



Embedding

Encoding Words into Vectors

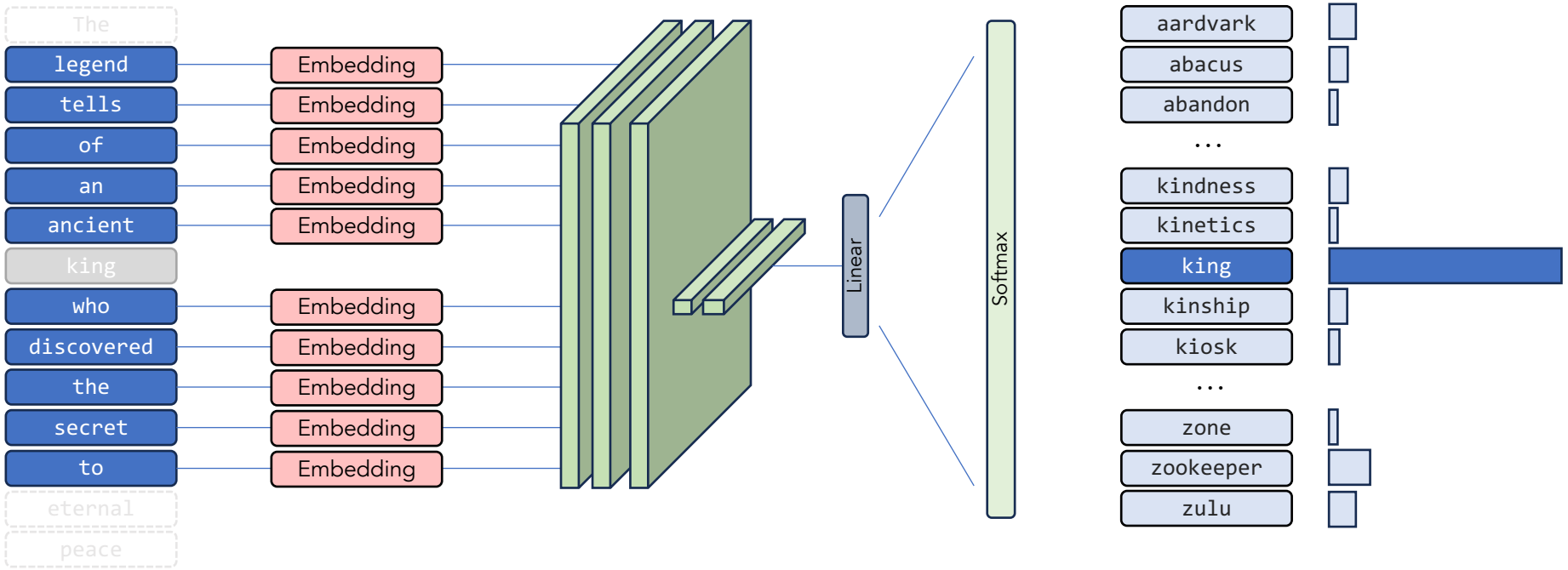
king → `np.array([ 0.603  0.198 -0.585 … -0.074])`

queen → `np.array([ 1.298  1.422 -1.122 …  0.244])`

man → `np.array([ 0.252 -0.836 -0.294 …  0.455])`

woman → `np.array([ 0.947  0.388 -0.831 …  0.773])`

"The legend tells of an ancient king who discovered the secret to eternal peace."



Embedding

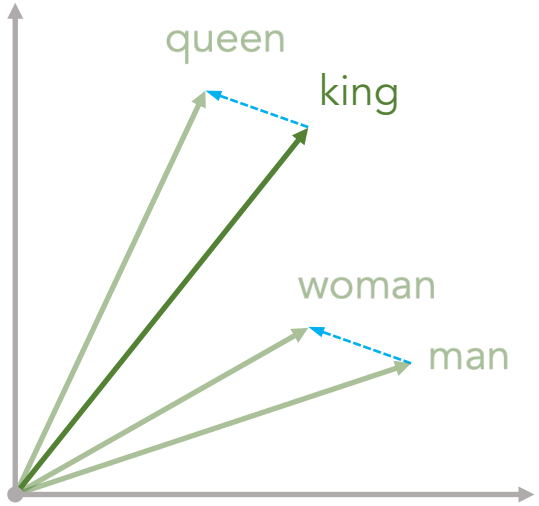
Encoding Words into Vectors

king → `np.array([ 0.603  0.198 -0.585 … -0.074])`

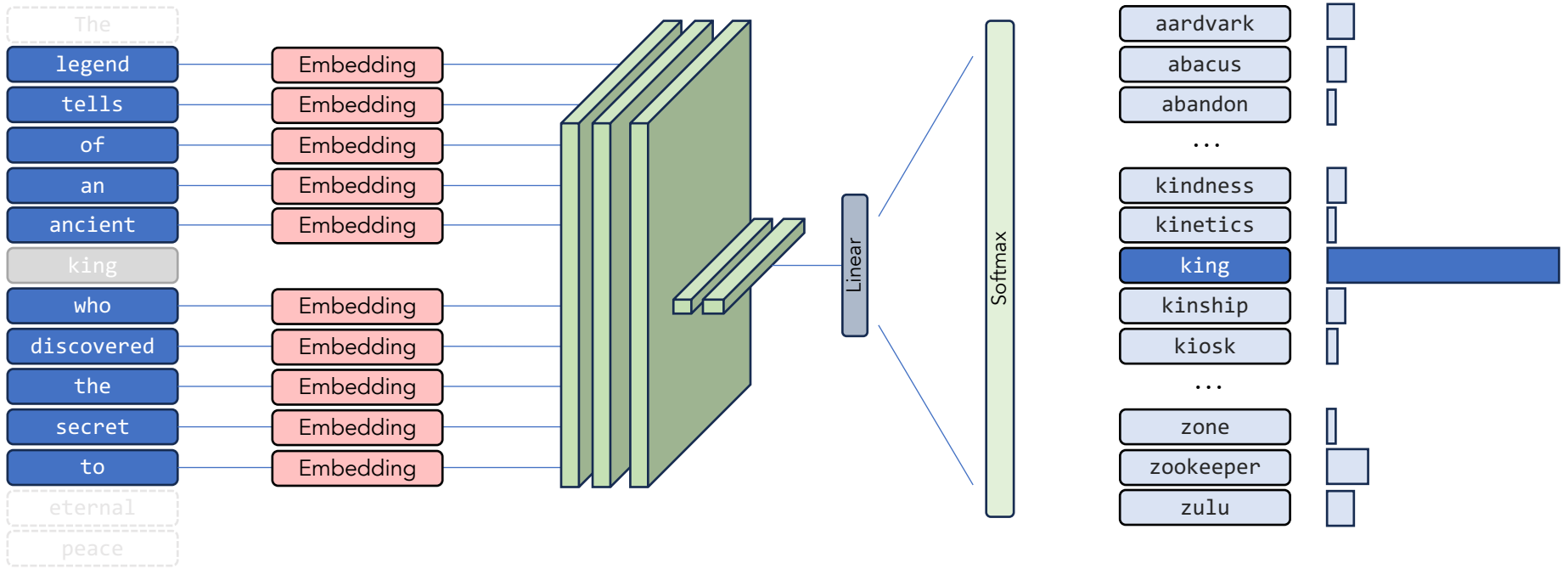
queen → `np.array([ 1.298  1.422 -1.122 …  0.244])`

man → `np.array([ 0.252 -0.836 -0.294 …  0.455])`

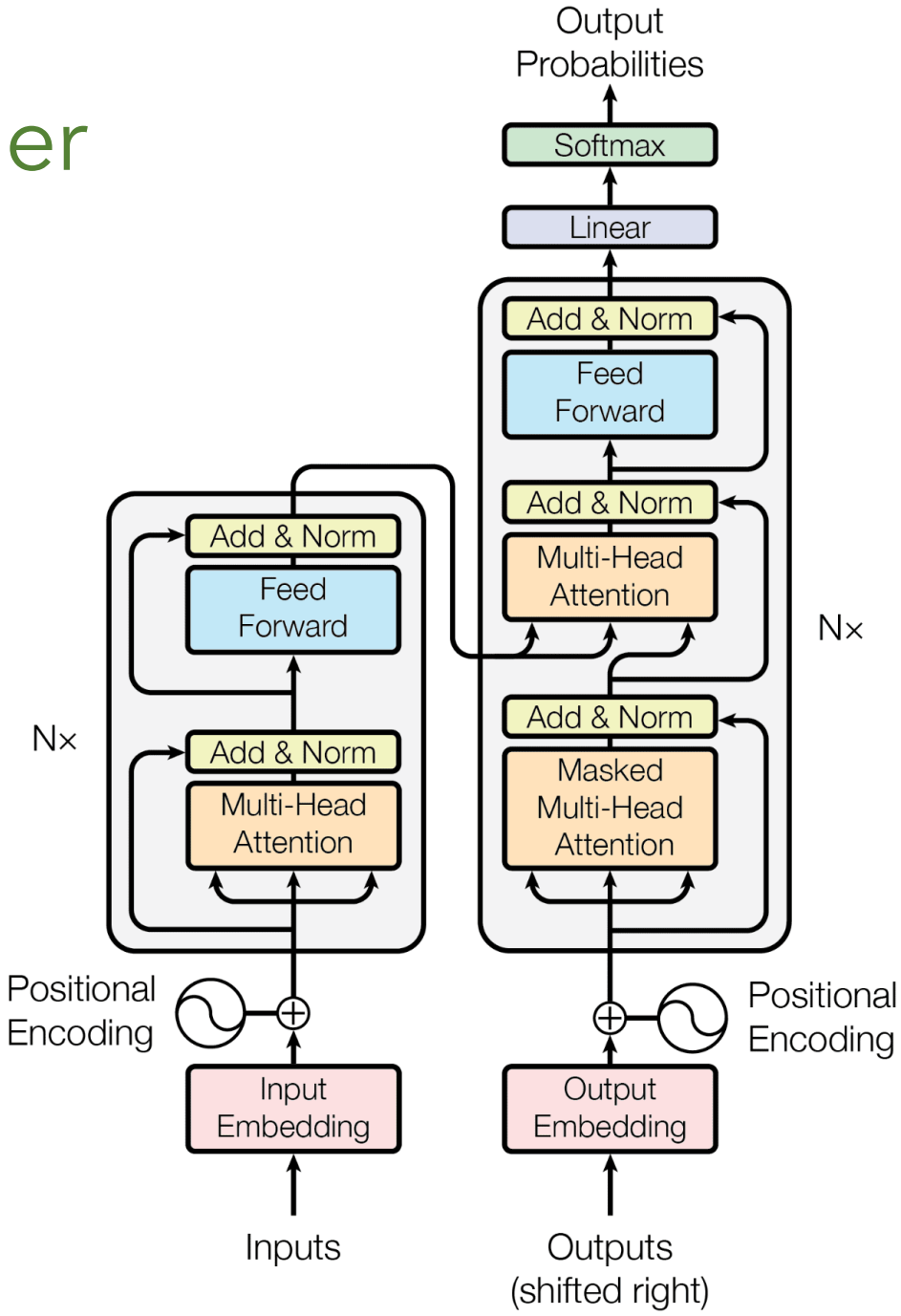
woman → `np.array([ 0.947  0.388 -0.831 …  0.773])`



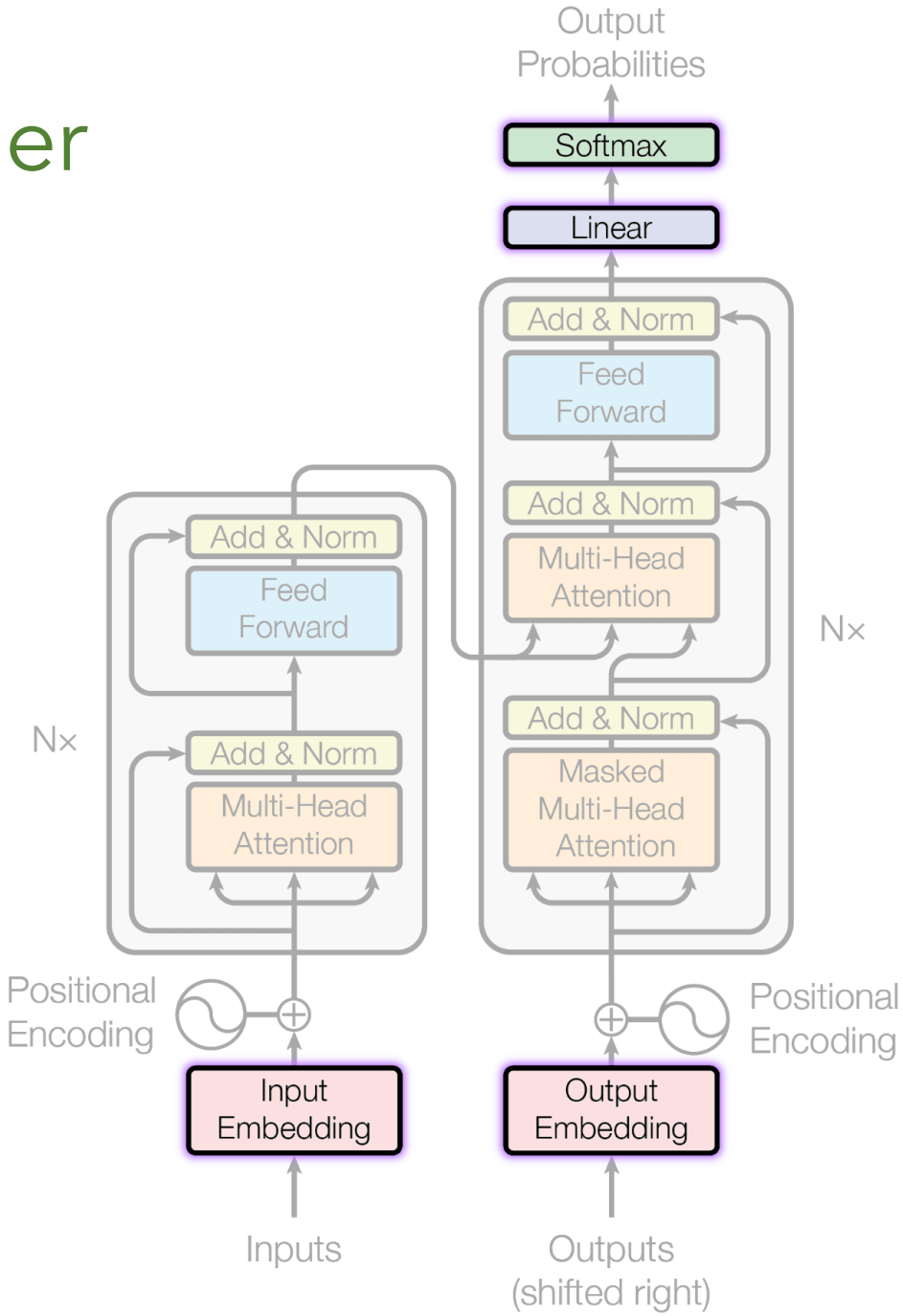
"The legend tells of an ancient king who discovered the secret to eternal peace."



The Transformer



The Transformer



Sequence to Sequence Modeling

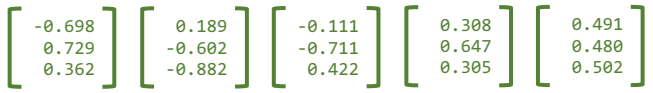
E.g., Translation

Como un cordero al matadero	→	Like a lamb to the slaughter
Donde hay humo, hay fuego	→	Where there's smoke, there's fire
Nunca es tarde para aprender	→	It's never too late to learn
Que sera, sera	→	Whatever will be, will be

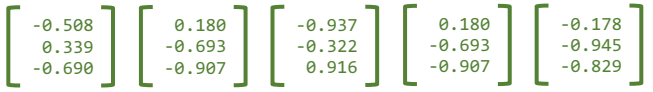
Sequence to Sequence Modeling

E.g., Translation

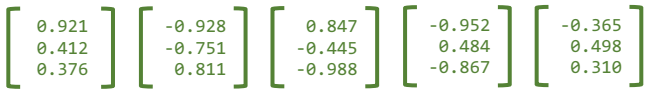
Como un cordero al matadero → Like a lamb to the slaughter



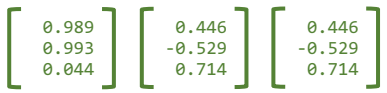
Donde hay humo, hay fuego → Where there's smoke, there's fire



Nunca es tarde para aprender → It's never too late to learn

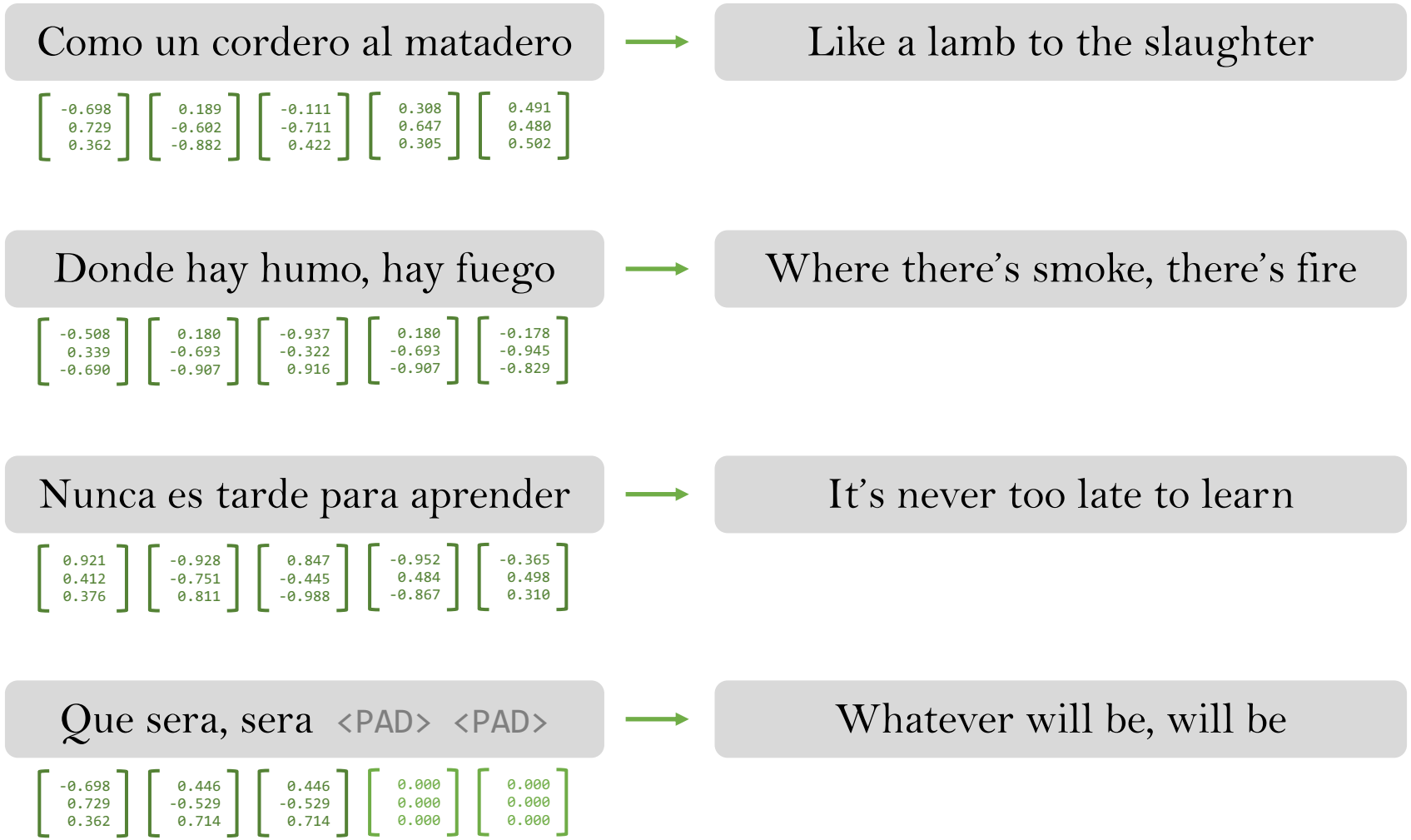


Que sera, sera → Whatever will be, will be



Sequence to Sequence Modeling

E.g., Translation

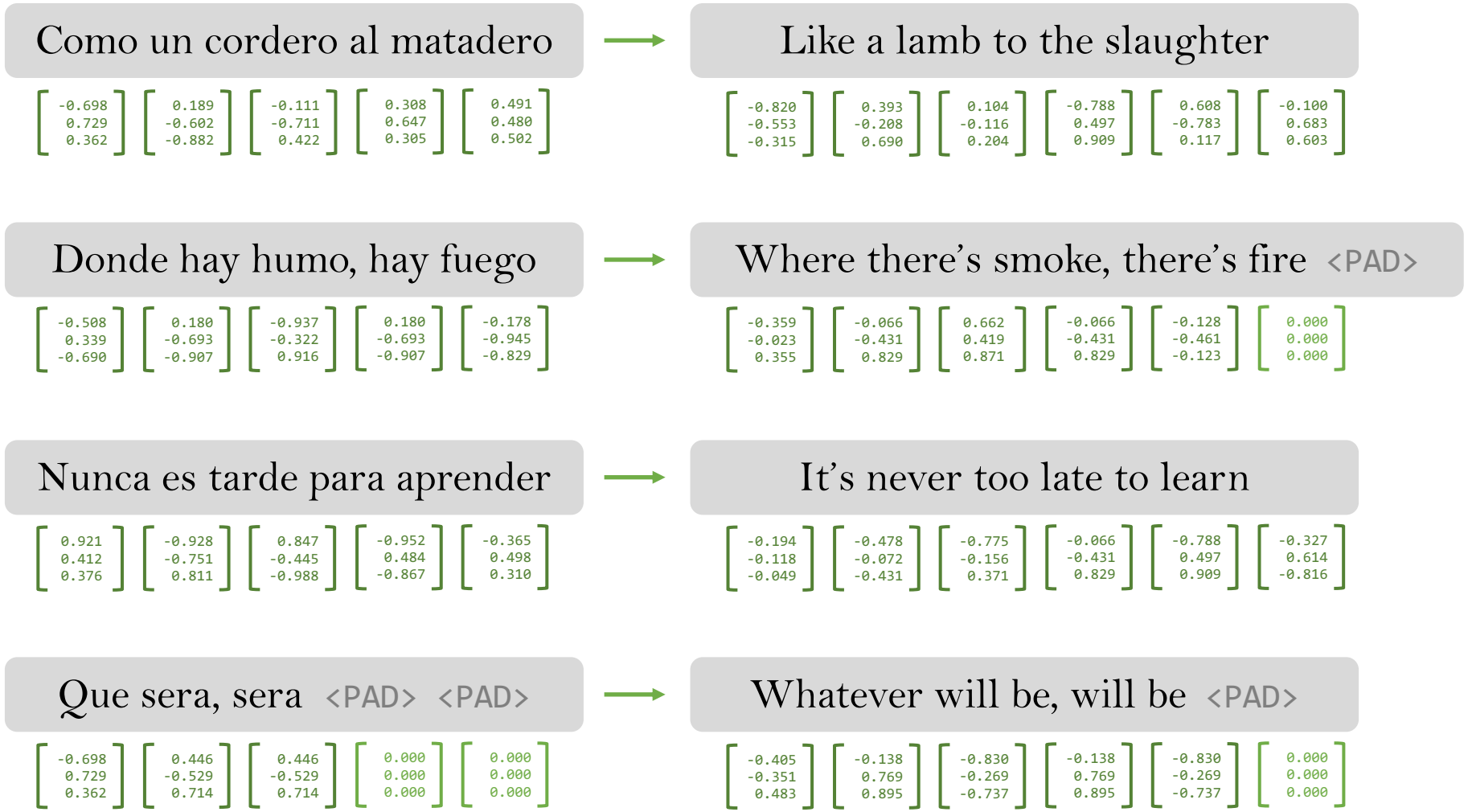


Sequence to Sequence Modeling

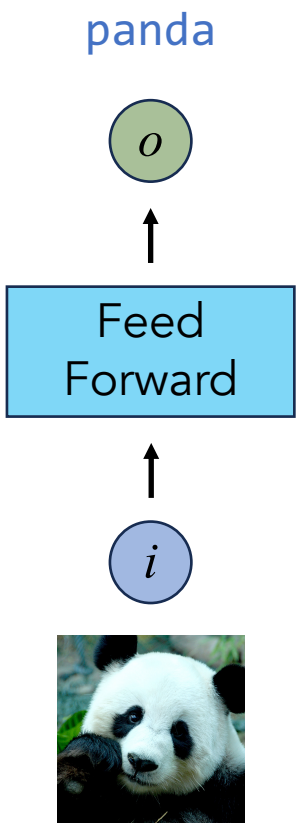
E.g., Translation

Sequence to Sequence Modeling

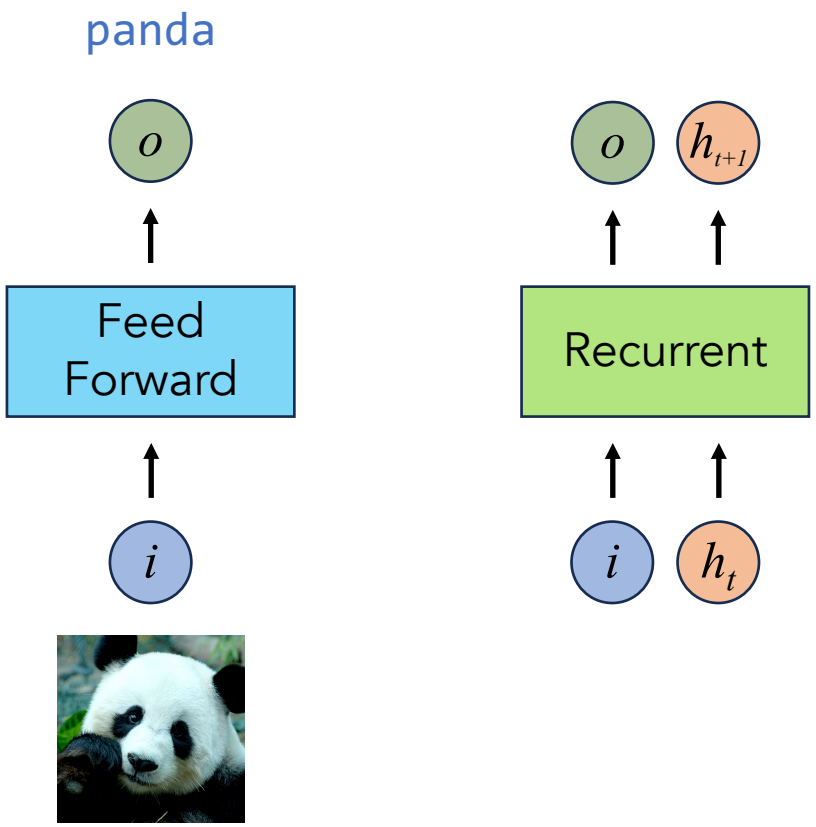
E.g., Translation



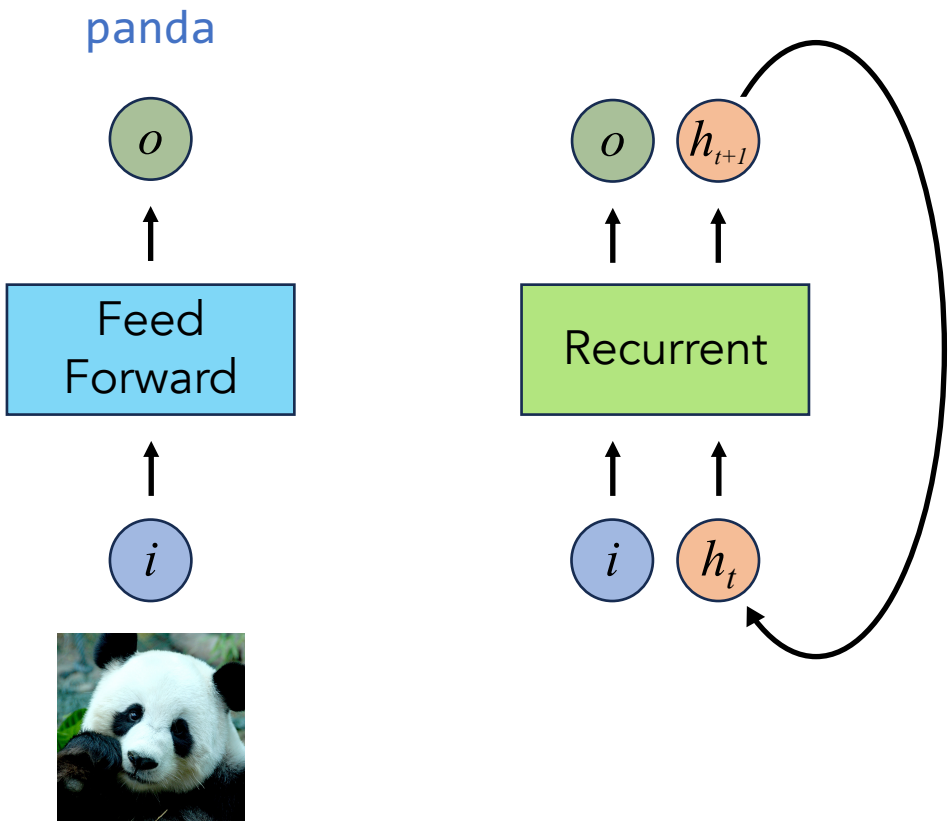
An Alternative Neural Network



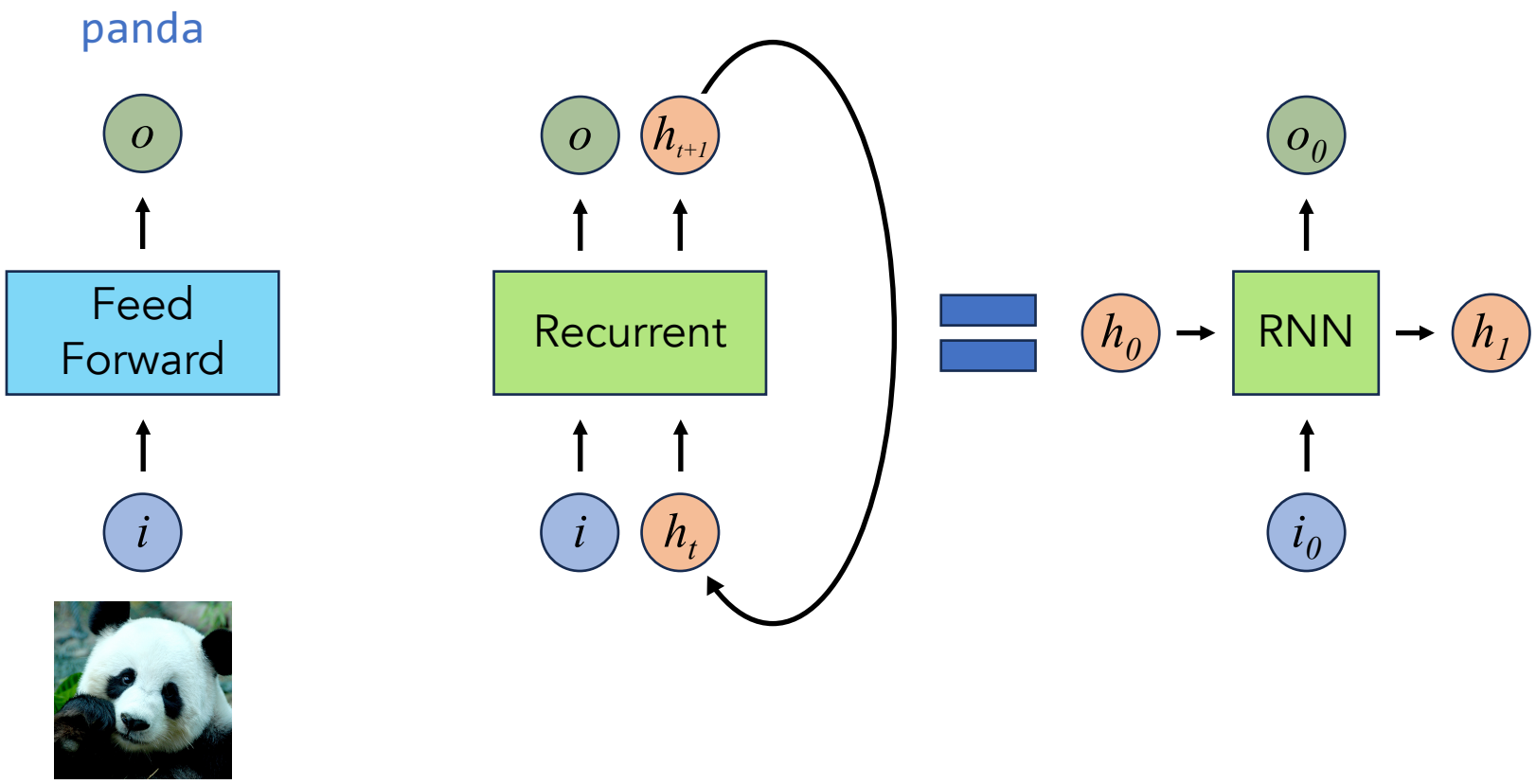
An Alternative Neural Network



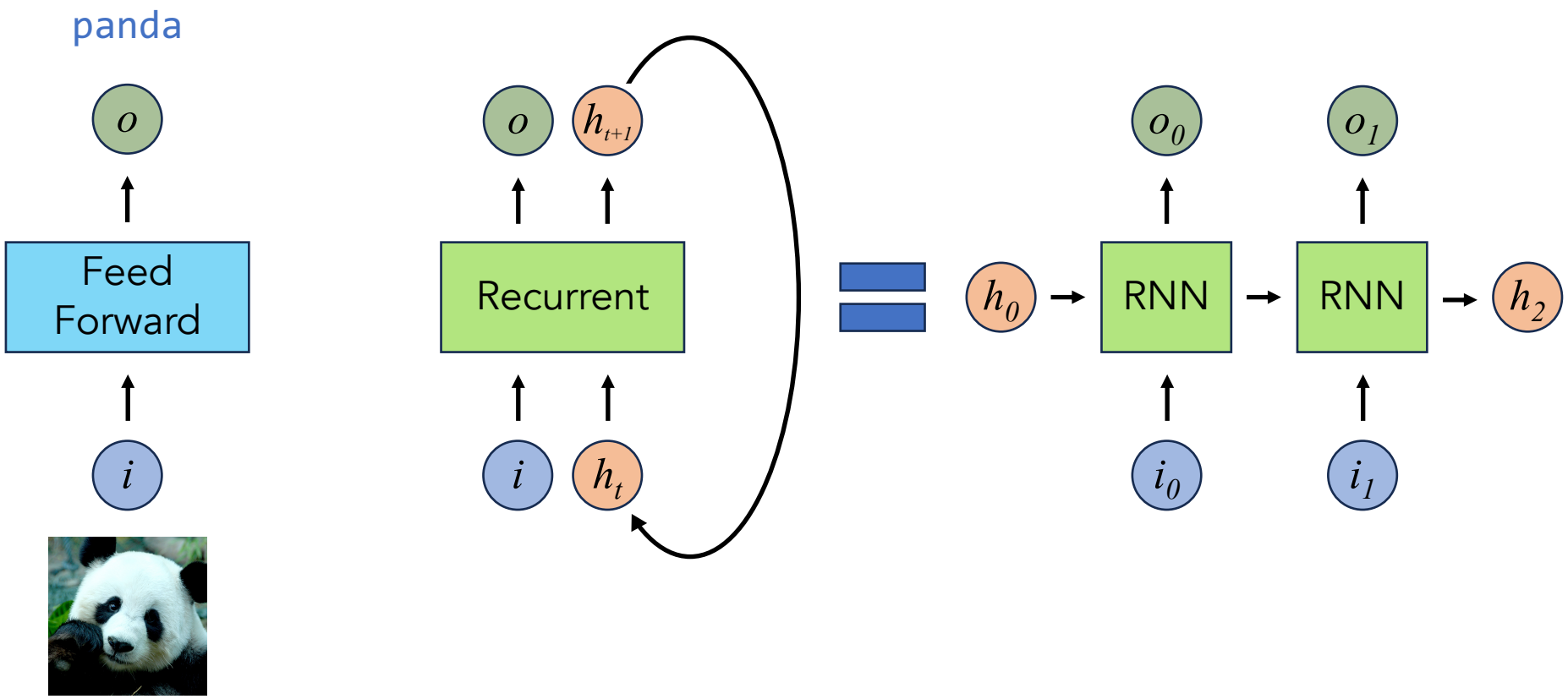
An Alternative Neural Network



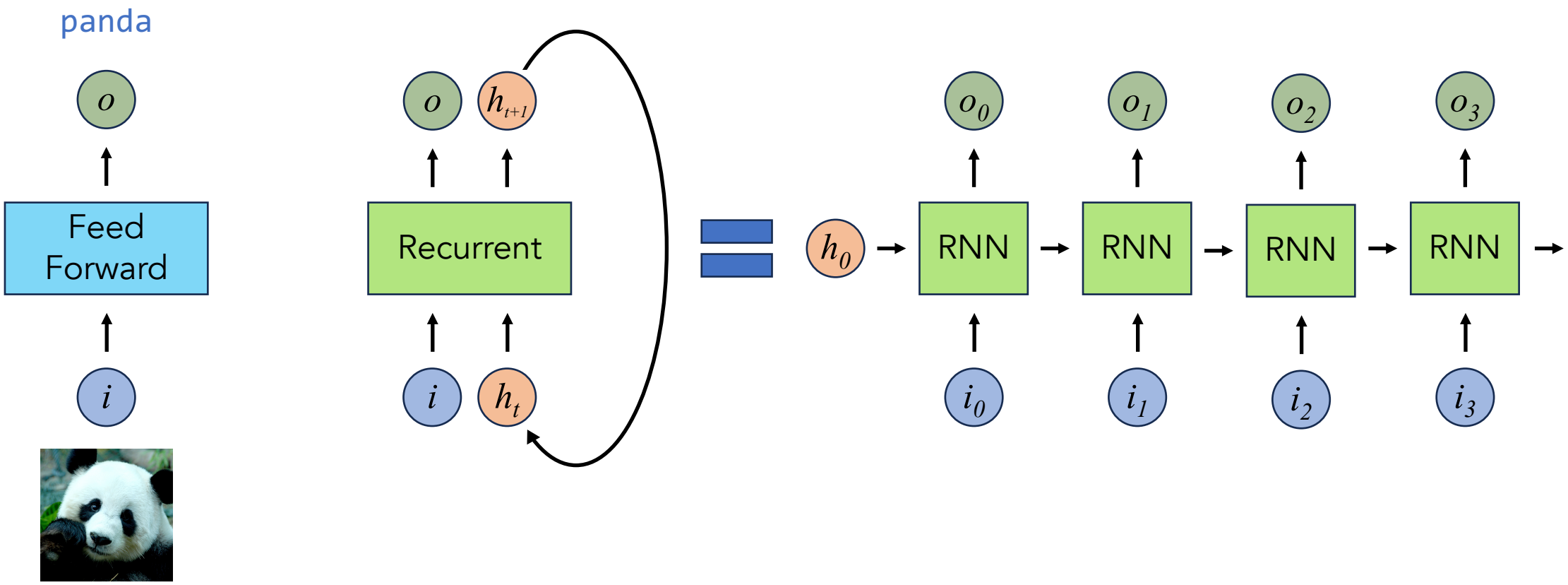
An Alternative Neural Network



An Alternative Neural Network



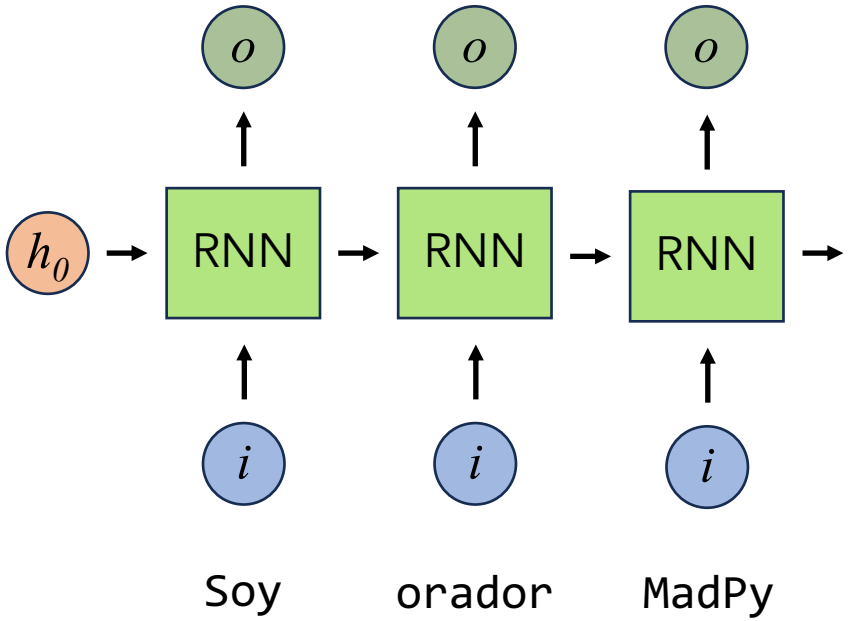
An Alternative Neural Network



RNNs for NLP

Soy orador MadPy

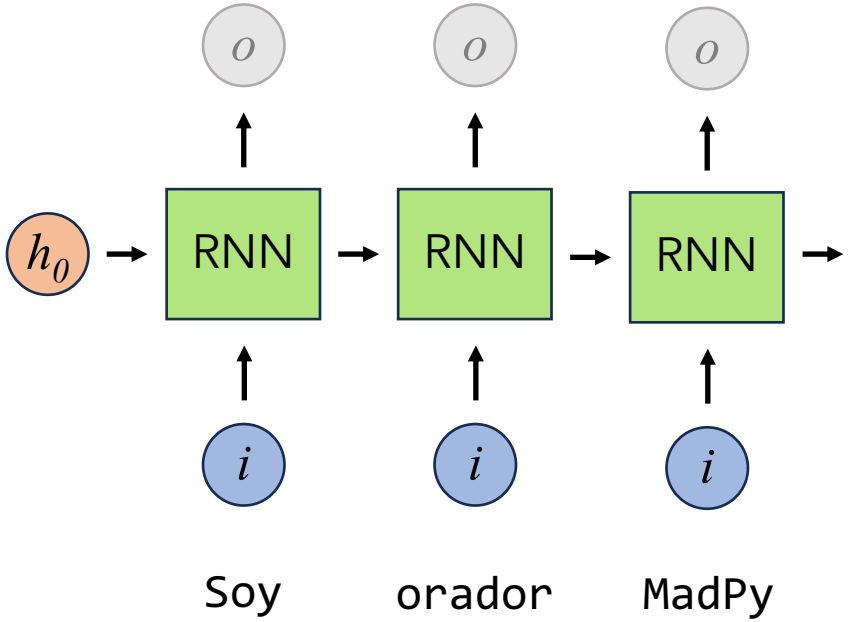
I am a MadPy speaker



RNNs for NLP

Soy orador MadPy

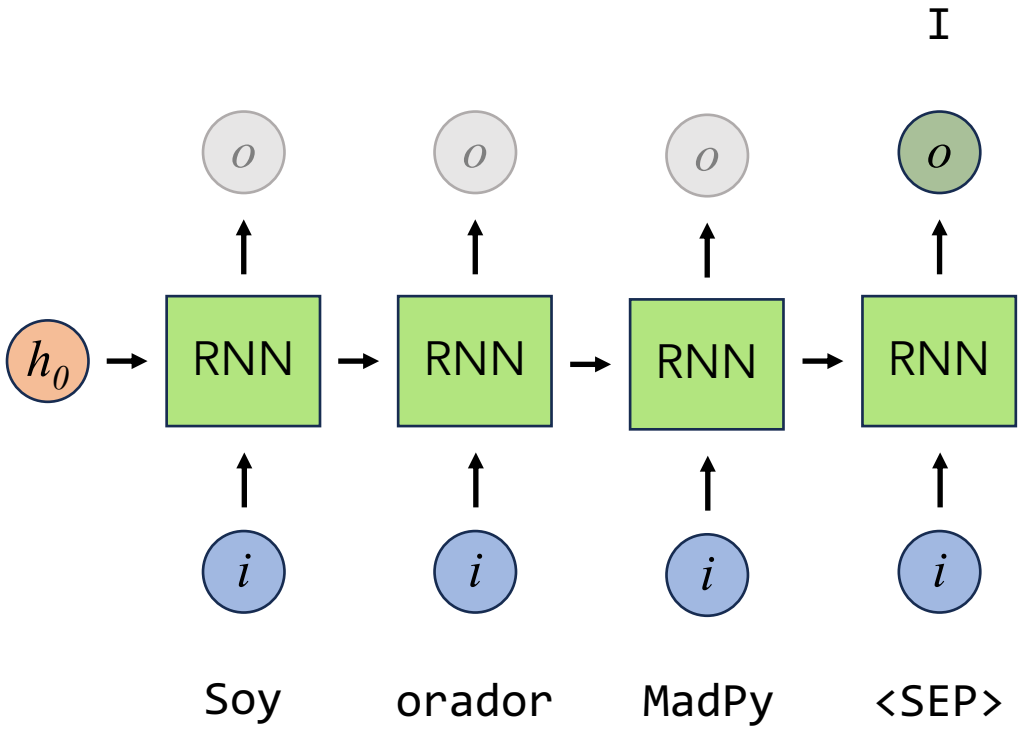
I am a MadPy speaker



RNNs for NLP

Soy orador MadPy

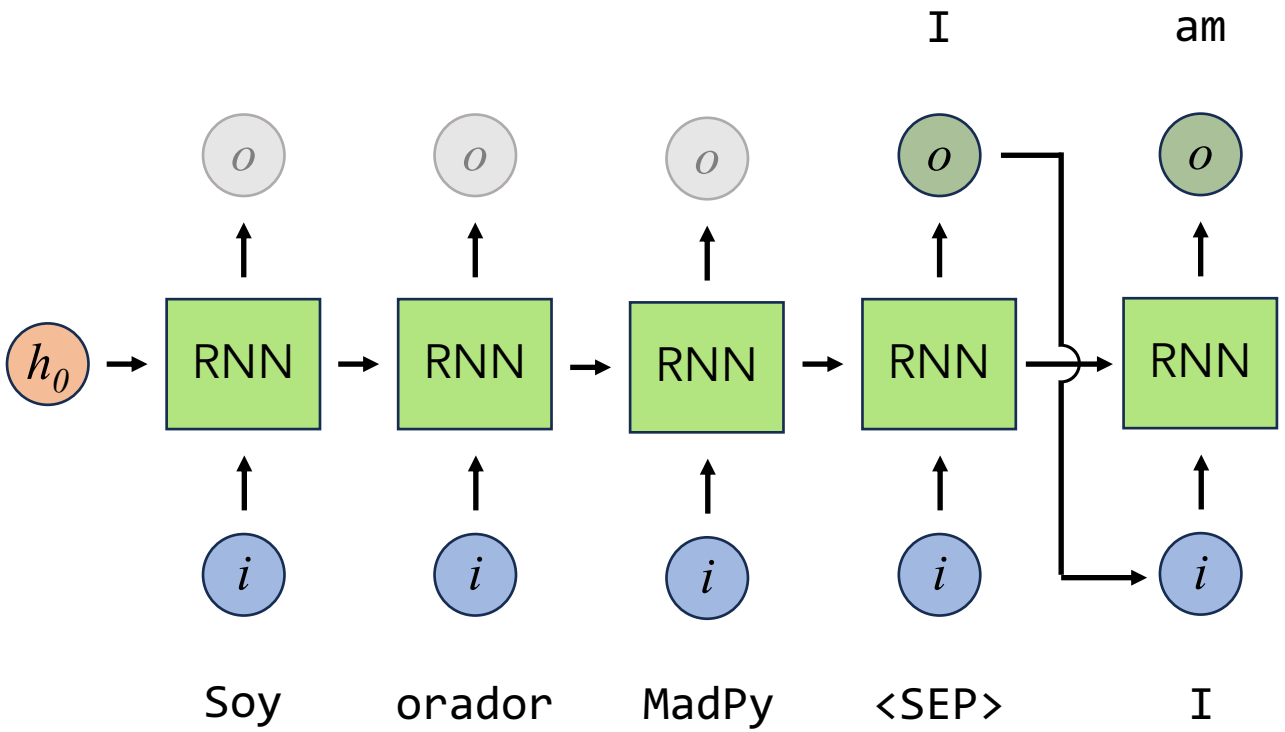
I am a MadPy speaker



RNNs for NLP

Soy orador MadPy

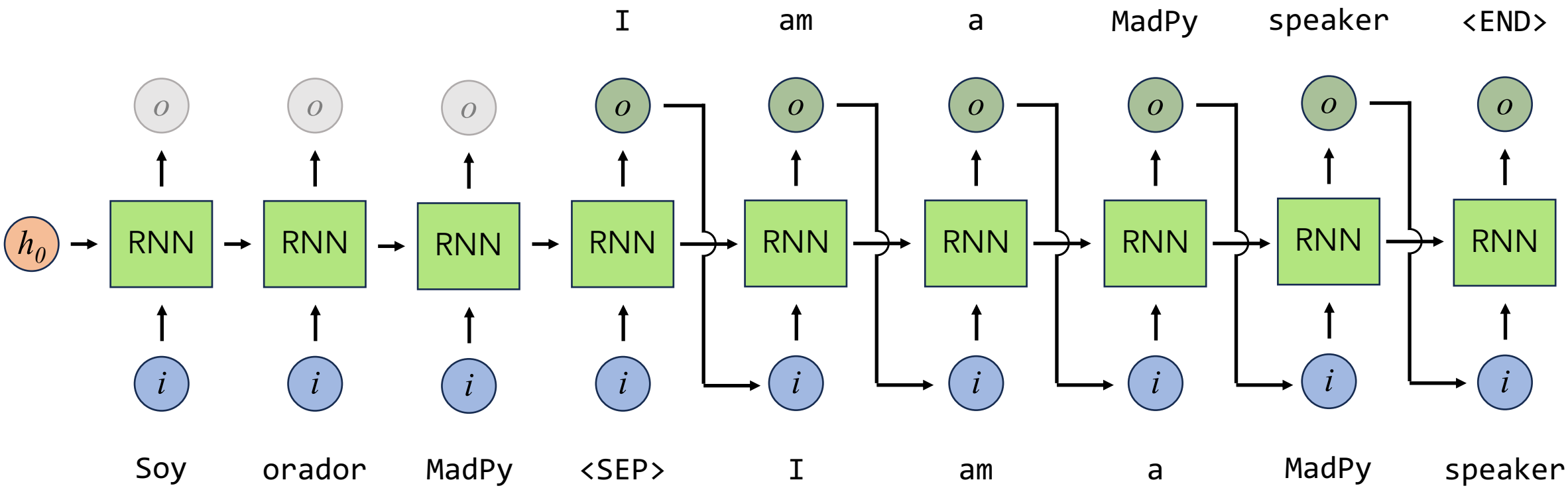
I am a MadPy speaker



RNNs for NLP

Soy orador MadPy

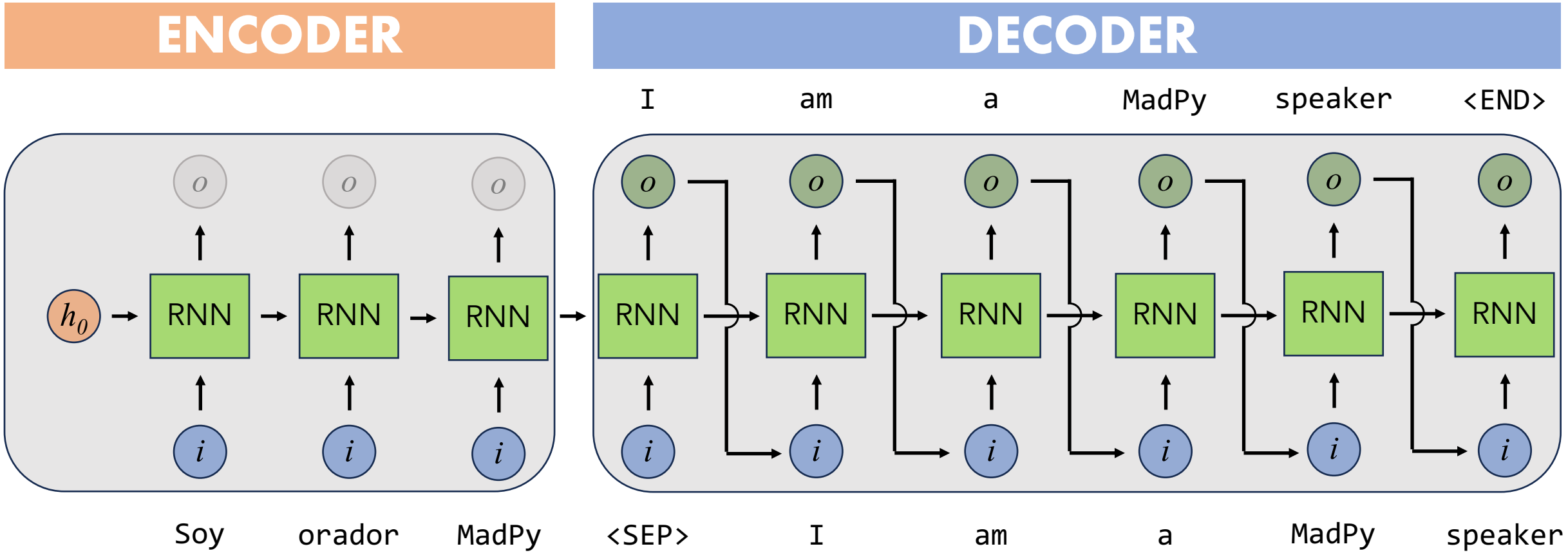
I am a MadPy speaker



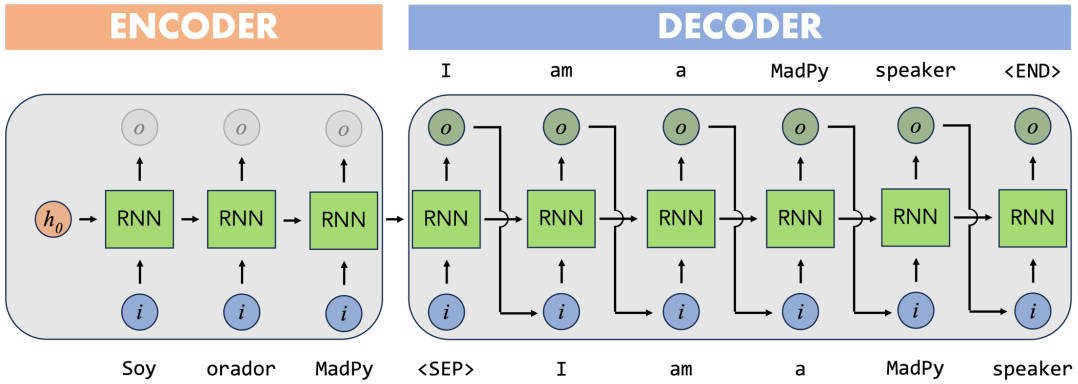
RNNs for NLP

Soy orador MadPy

I am a MadPy speaker



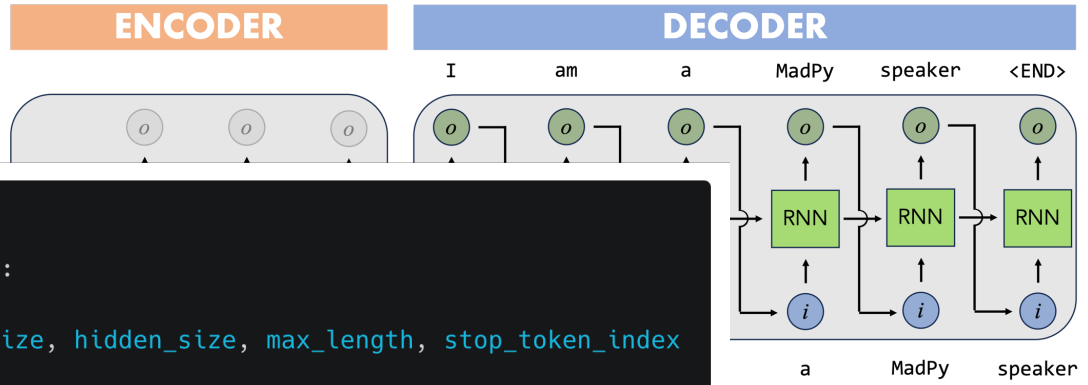
RNNs for NLP



ENCODER

```python
class EncoderRNN(nn.Module):
    def __init__(self, input_size, embedding_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(input_size, embedding_size)
        self.rnn = nn.RNN(embedding_size, hidden_size)

    def forward(self, input, hidden=None):
        embedded = self.embedding(input)
        output, hidden = self.rnn(embedded, hidden)
        return output, hidden
```

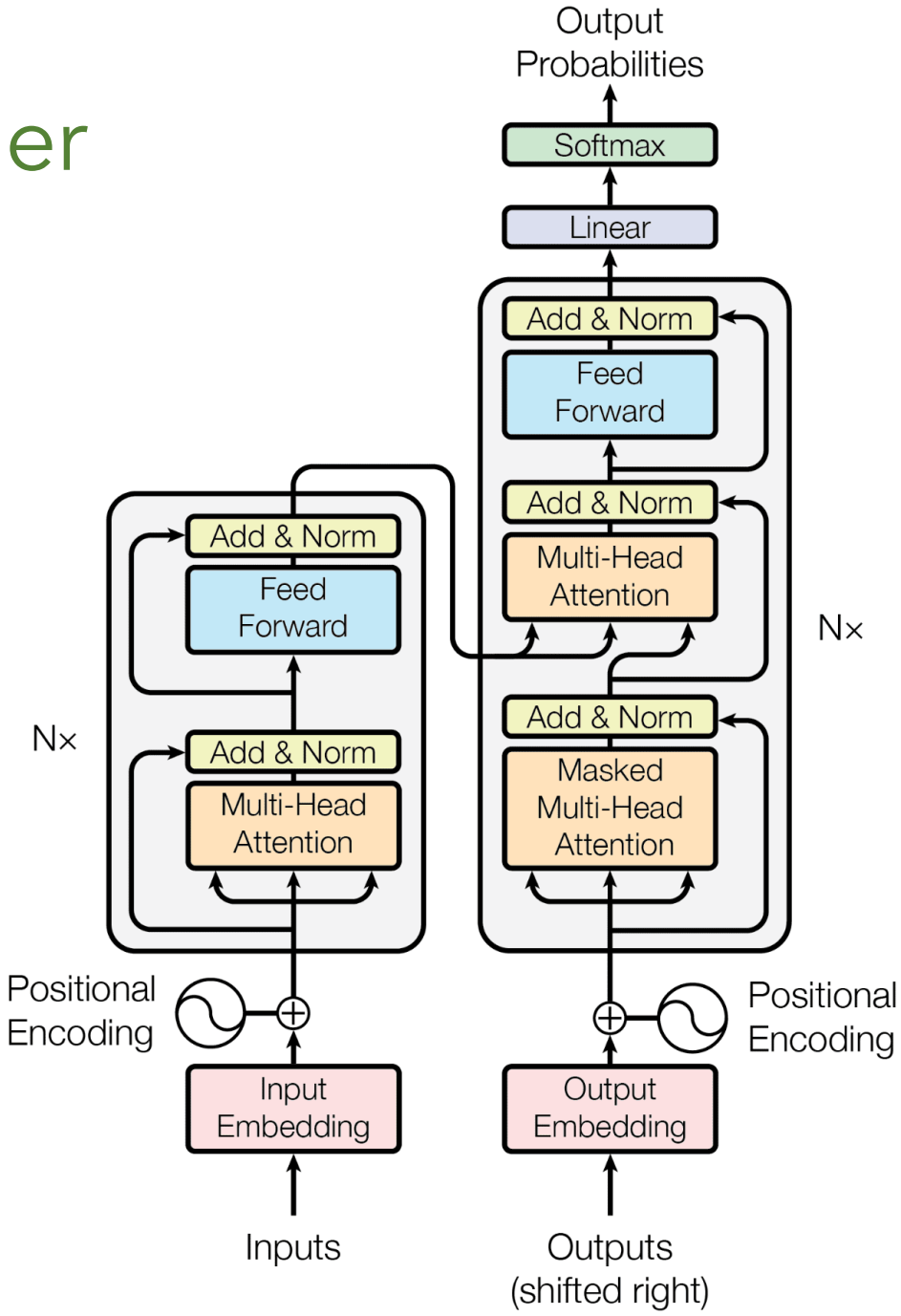
RNNs for NLP



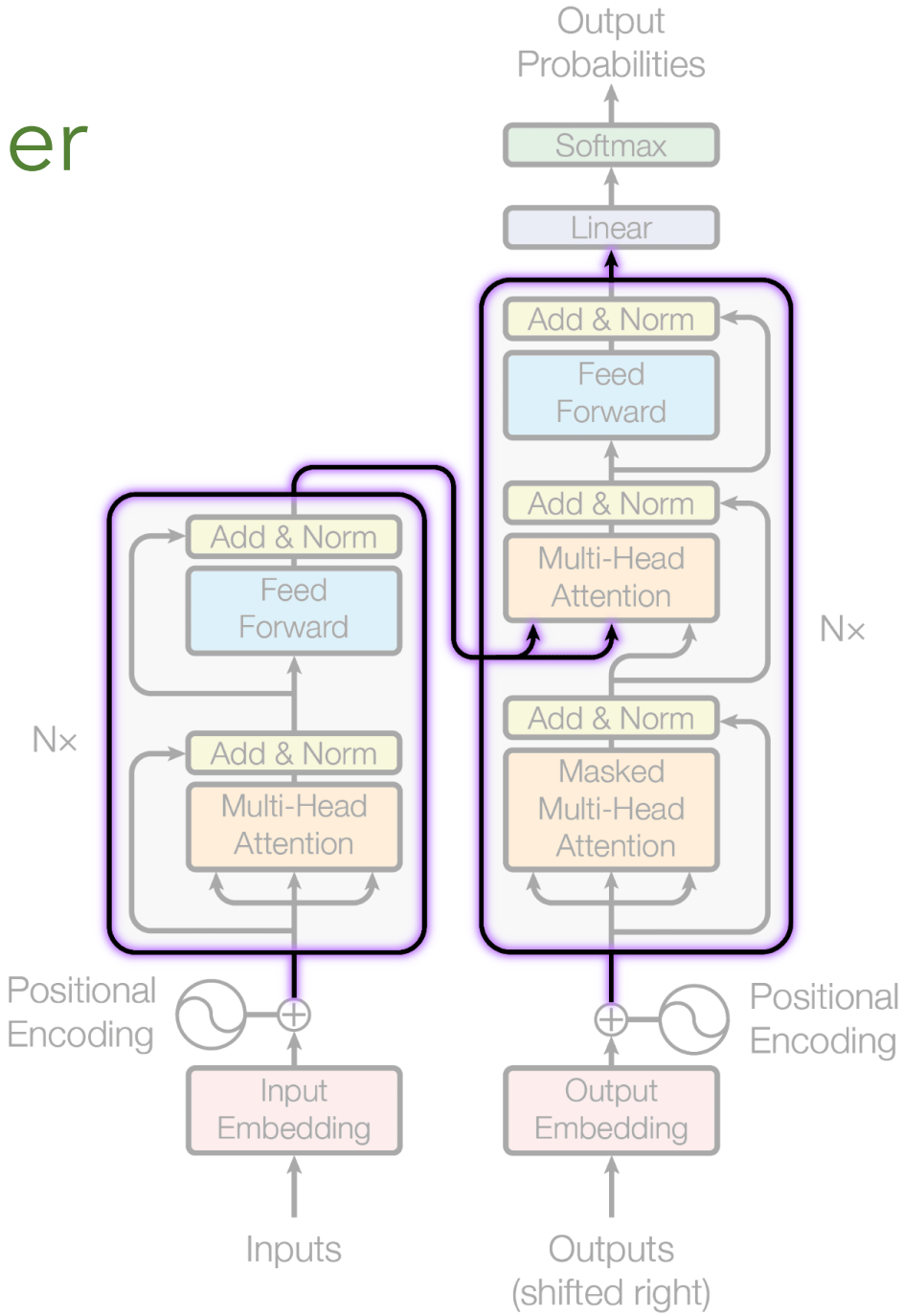
DECODER

```python
class AutoRegressiveDecoder(nn.Module):
    def __init__(
        self, output_size, embedding_size, hidden_size, max_length, stop_token_index
    ):
        super(AutoRegressiveDecoder, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(output_size, embedding_size)
        self.rnn = nn.RNN(embedding_size, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)
        self.max_length = max_length
        self.stop_token_index = stop_token_index

    def forward(self, hidden, start_token_index):
        device = hidden.device
        input = torch.tensor([[start_token_index]], device=device)
        outputs = []
        logits_list = []
        for _ in range(self.max_length):
            embedded = self.embedding(input)
            rnn_output, hidden = self.rnn(embedded, hidden)
            logits = self.out(rnn_output[0])
            logits_list.append(logits)
            top1 = logits.argmax(1)
            outputs.append(top1.item())
            if top1.item() == self.stop_token_index:
                break
            input = top1.unsqueeze(0)
        return outputs, torch.stack(logits_list, dim=0)
```

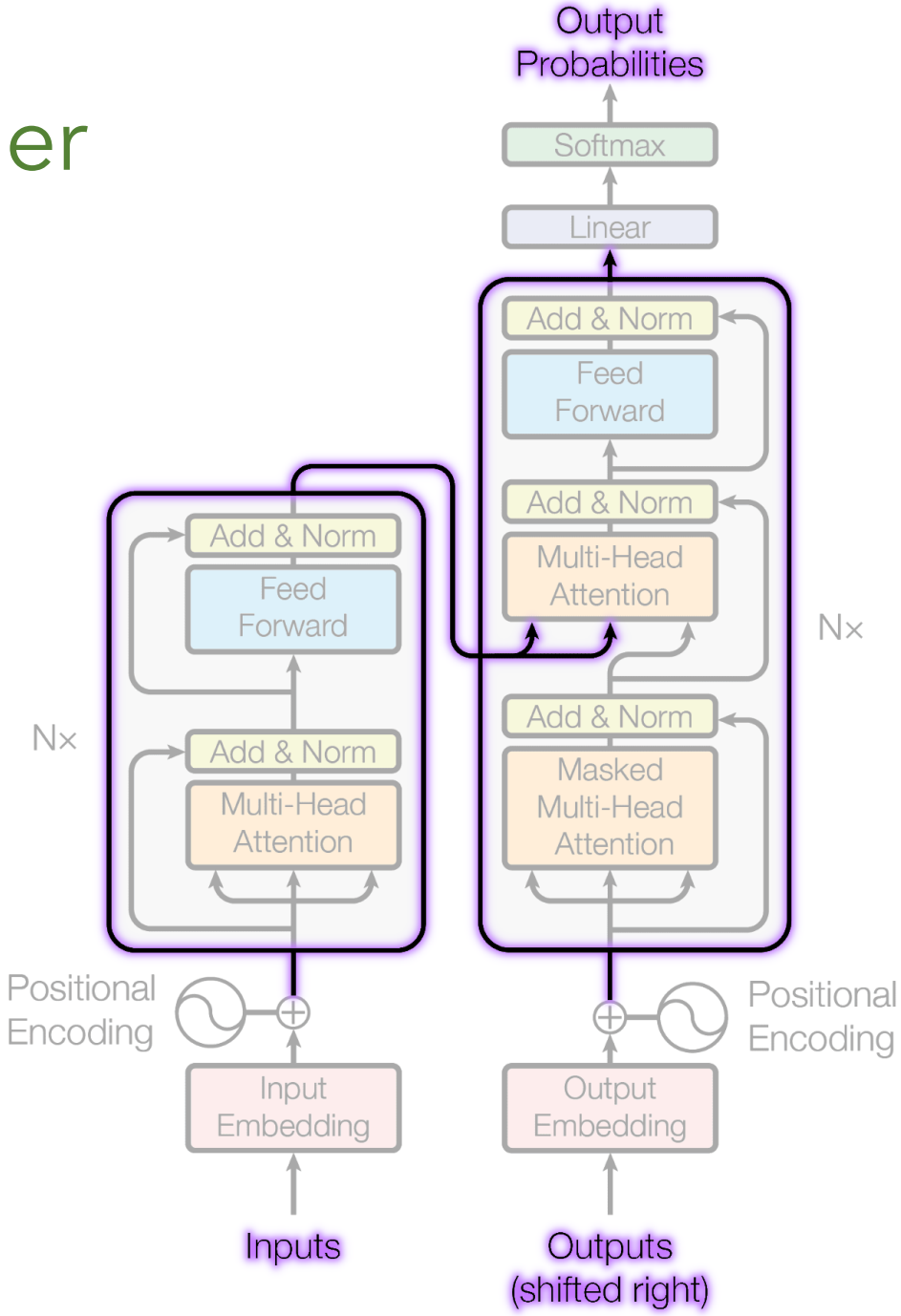
The Transformer



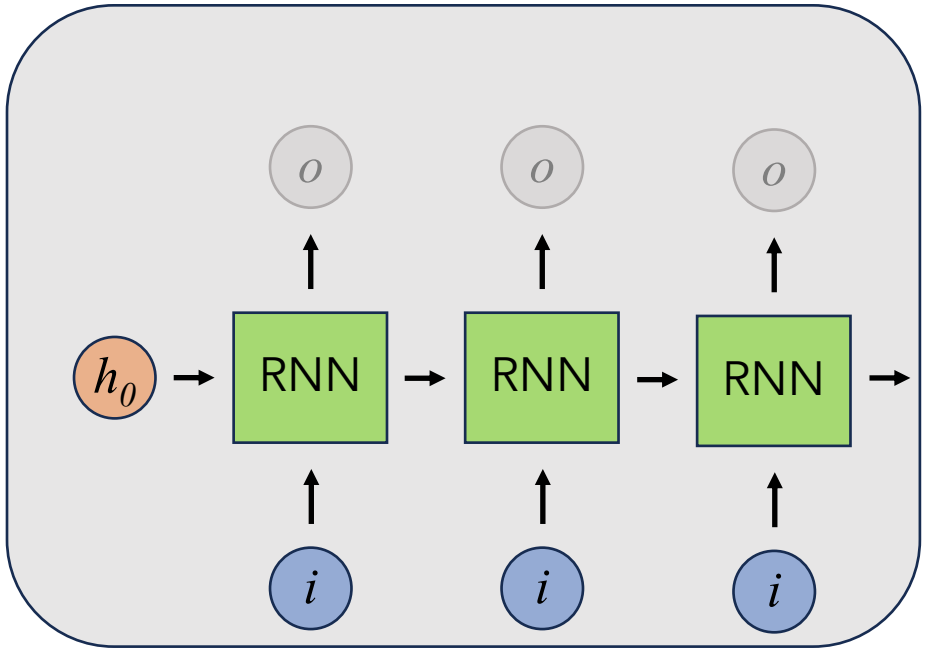
The Transformer



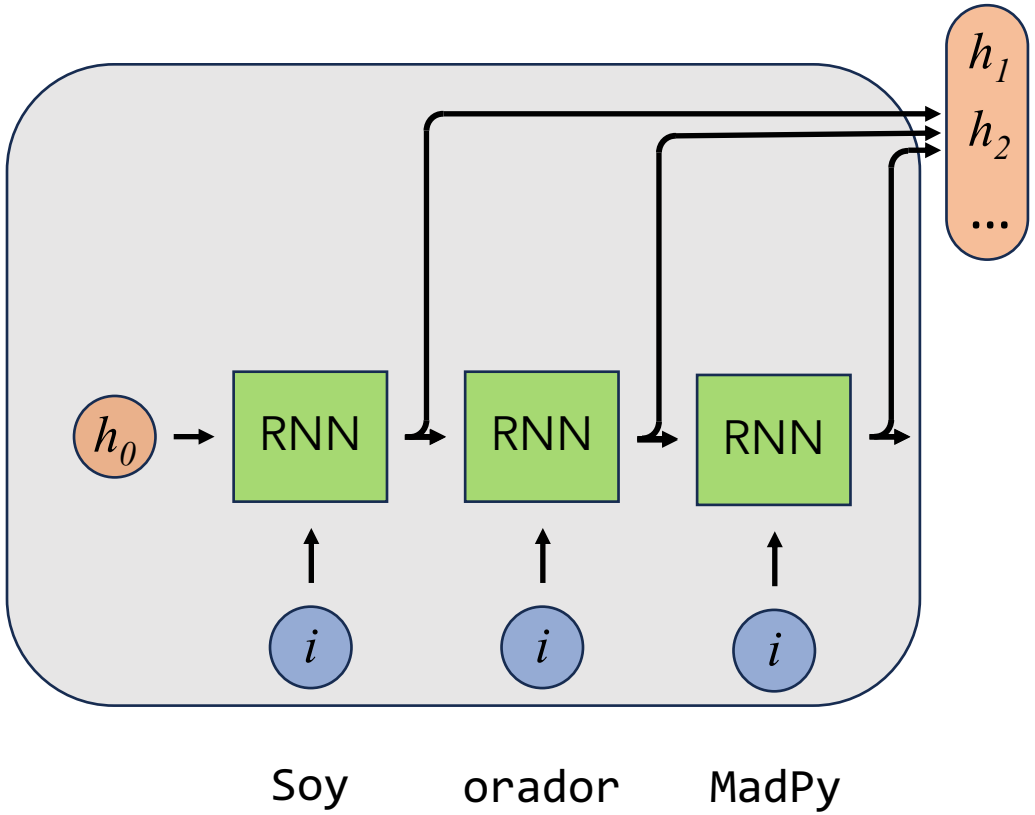
The Transformer



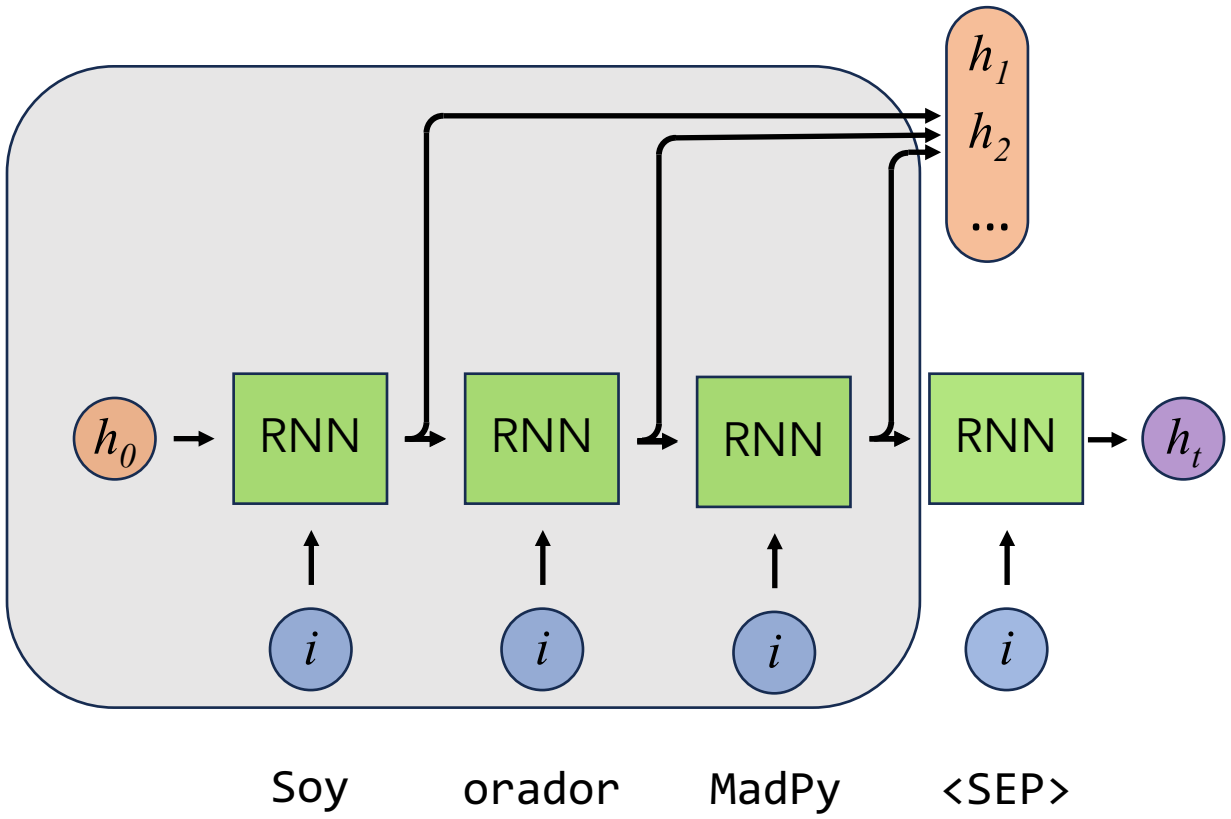
RNNsearch



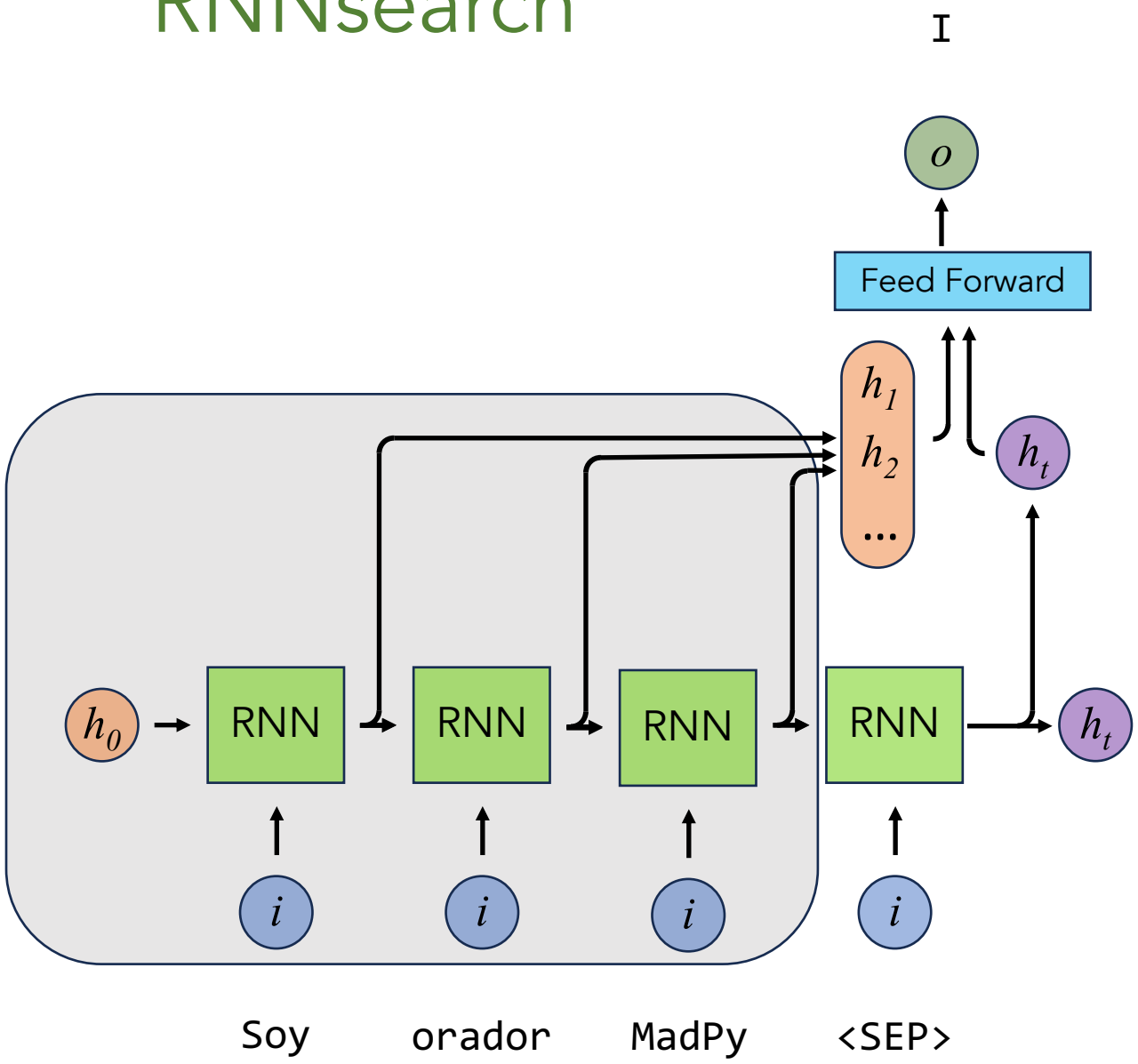
RNNsearch



RNNsearch



RNNsearch

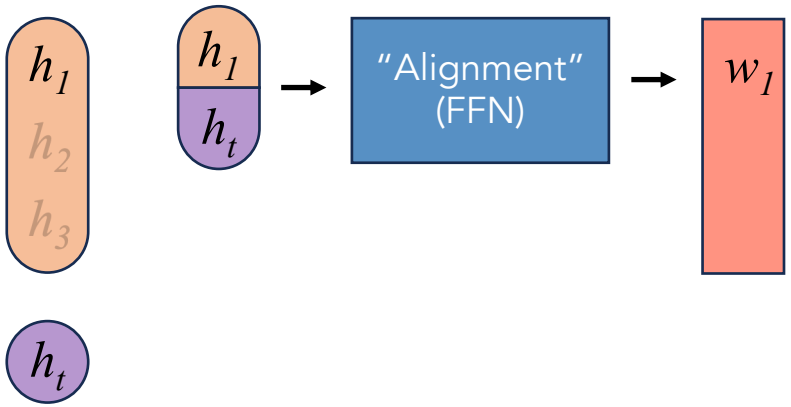

I
o
Feed Forward
h_1
h_2
...
h_t
h_0
RNN
RNN
RNN
RNN
h_t
i
i
i
i
Soy
orador
MadPy
<SEP>

RNNsearch

The “Attention” Mechanism

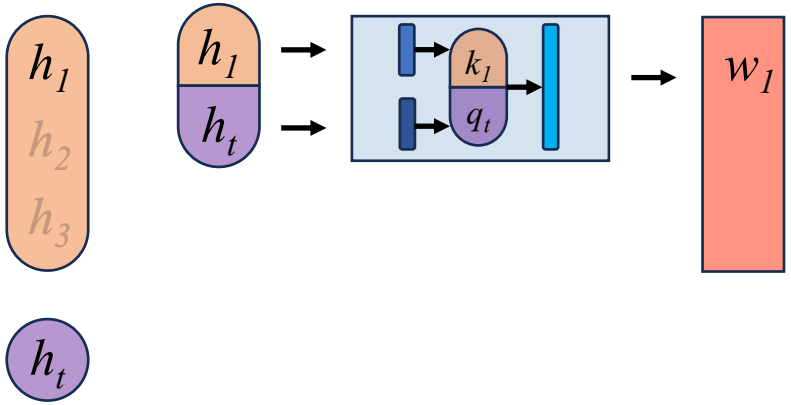
RNNsearch

The "Attention" Mechanism



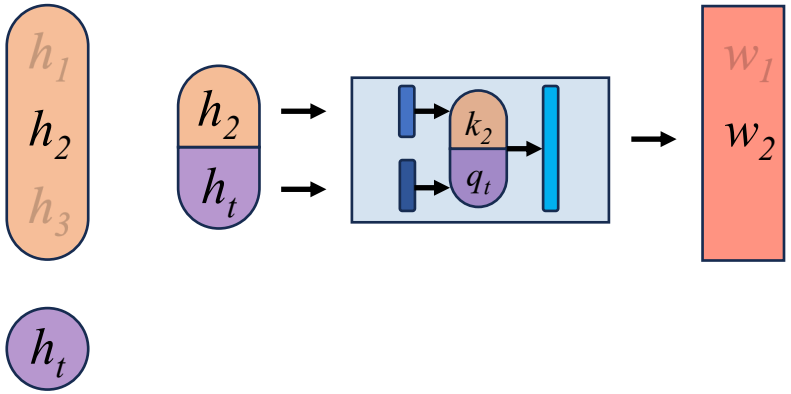
RNNsearch

The “Attention” Mechanism



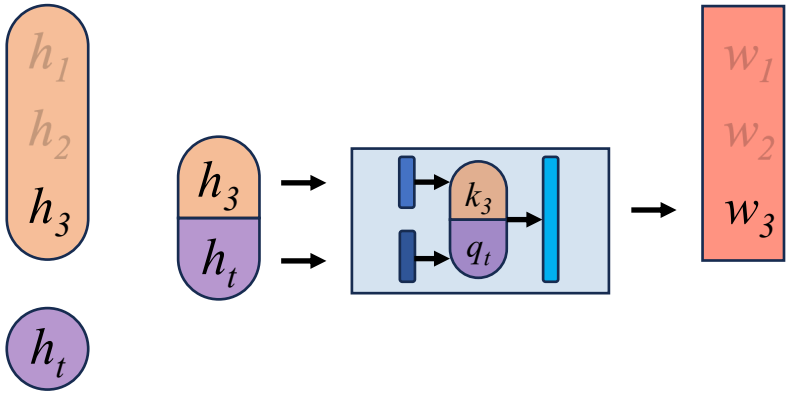
RNNsearch

The “Attention” Mechanism



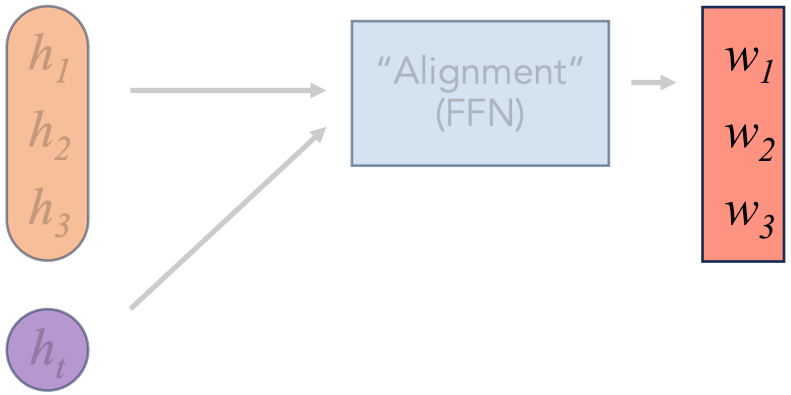
RNNsearch

The “Attention” Mechanism



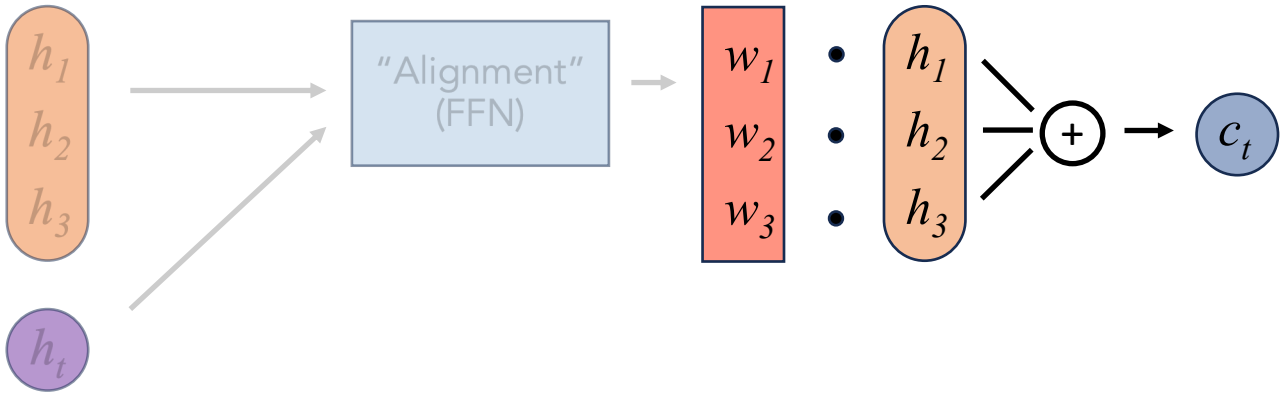
RNNsearch

The “Attention” Mechanism



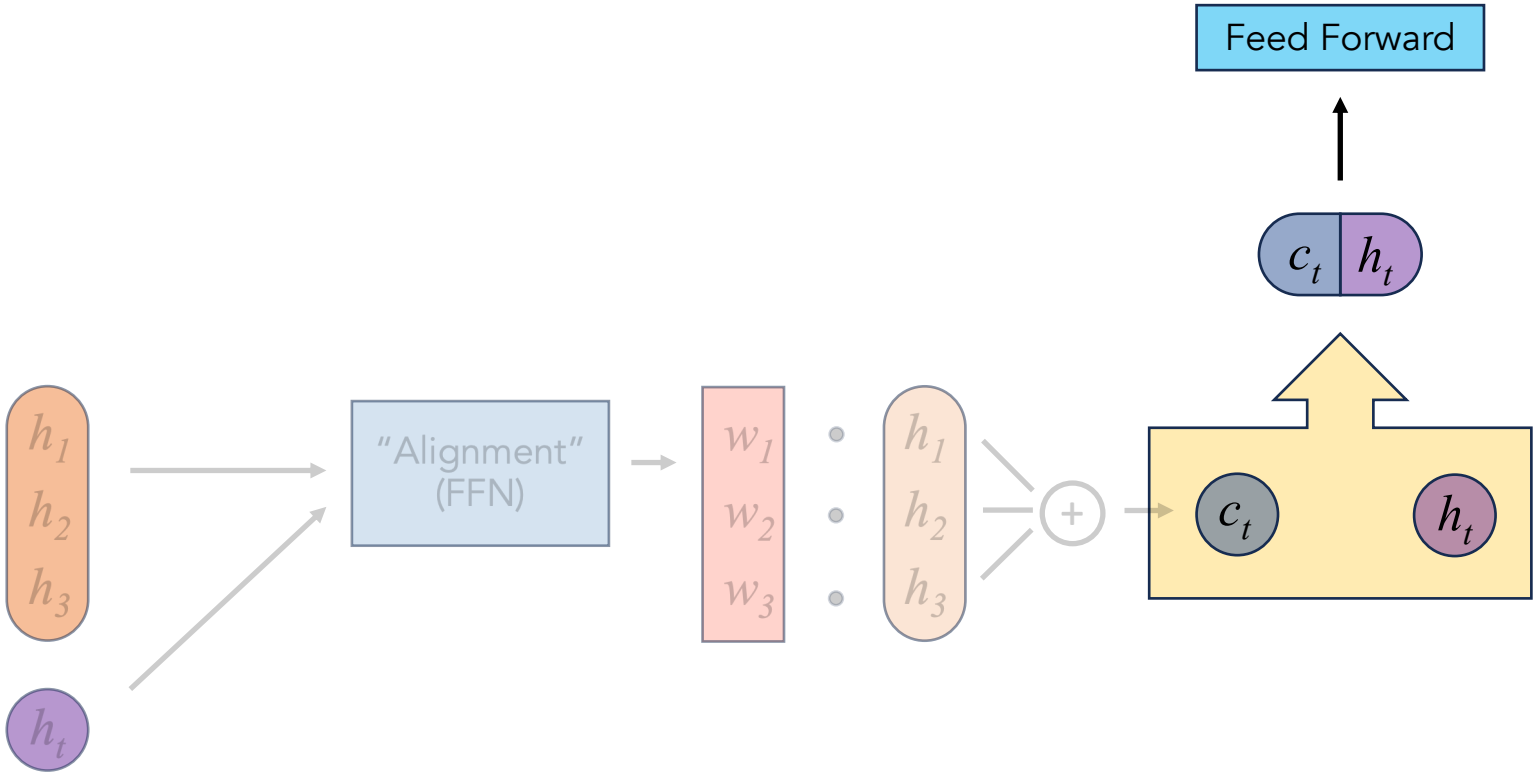
RNNsearch

The "Attention" Mechanism



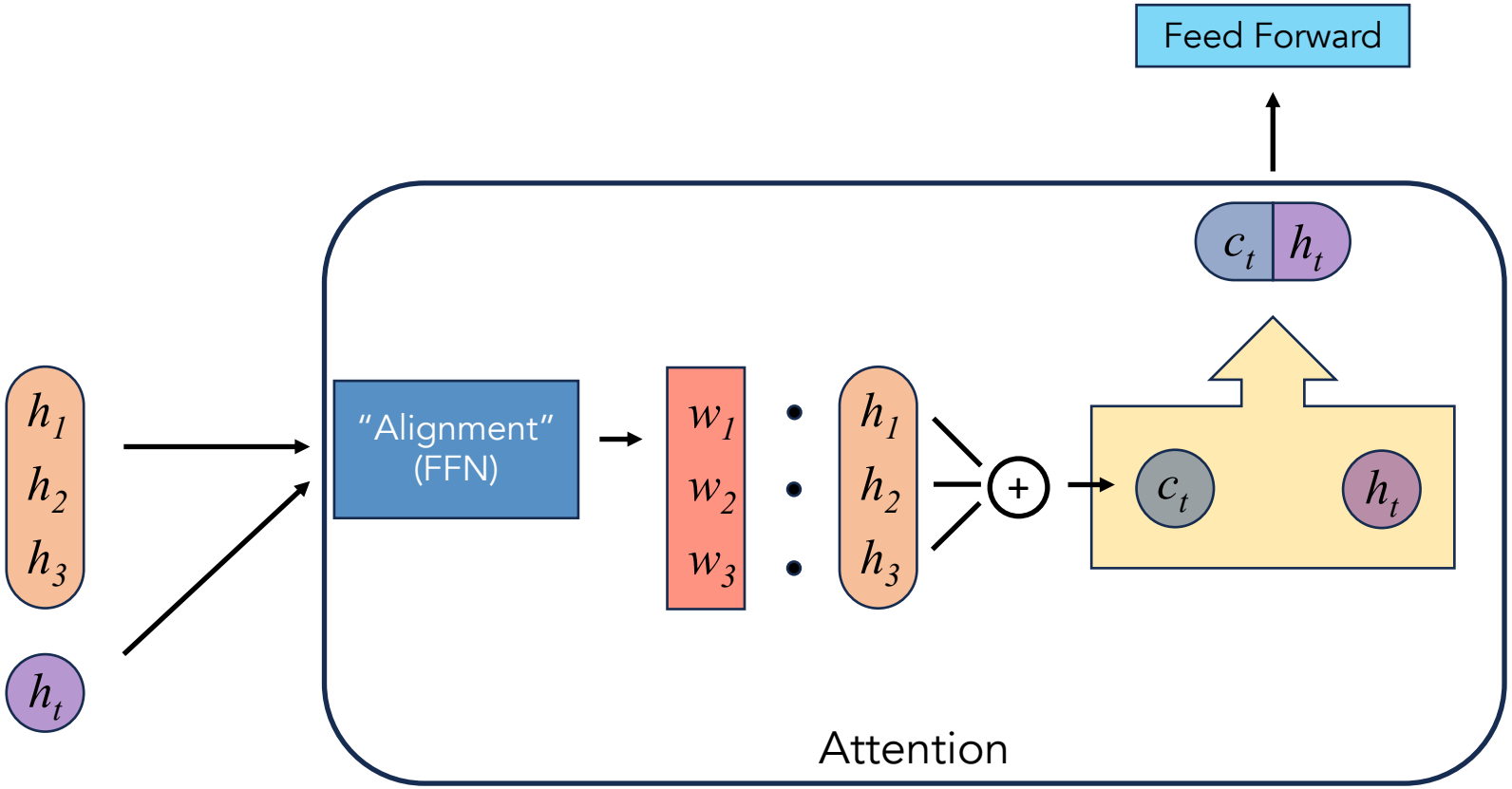
RNNsearch

The "Attention" Mechanism

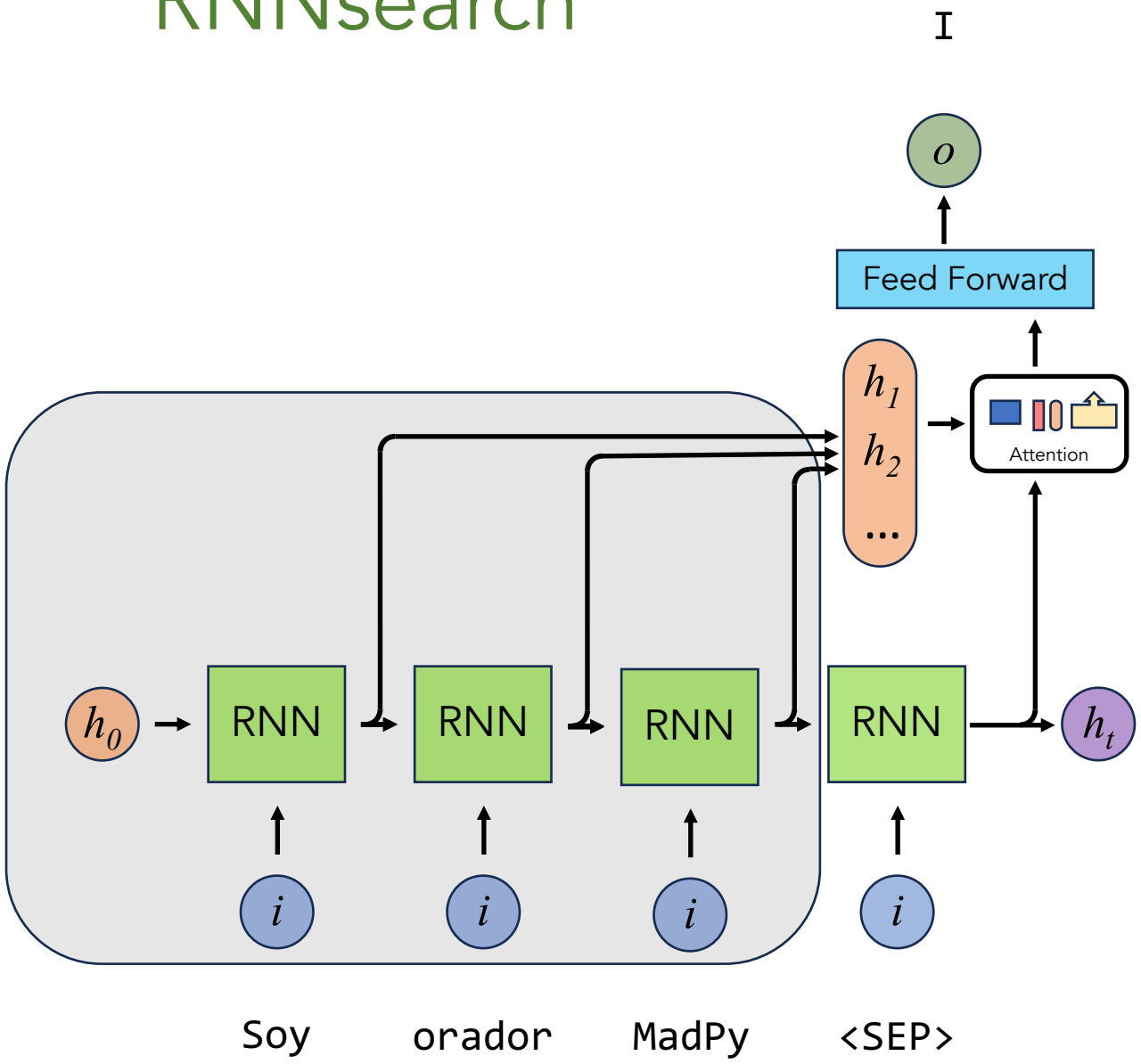


RNNsearch

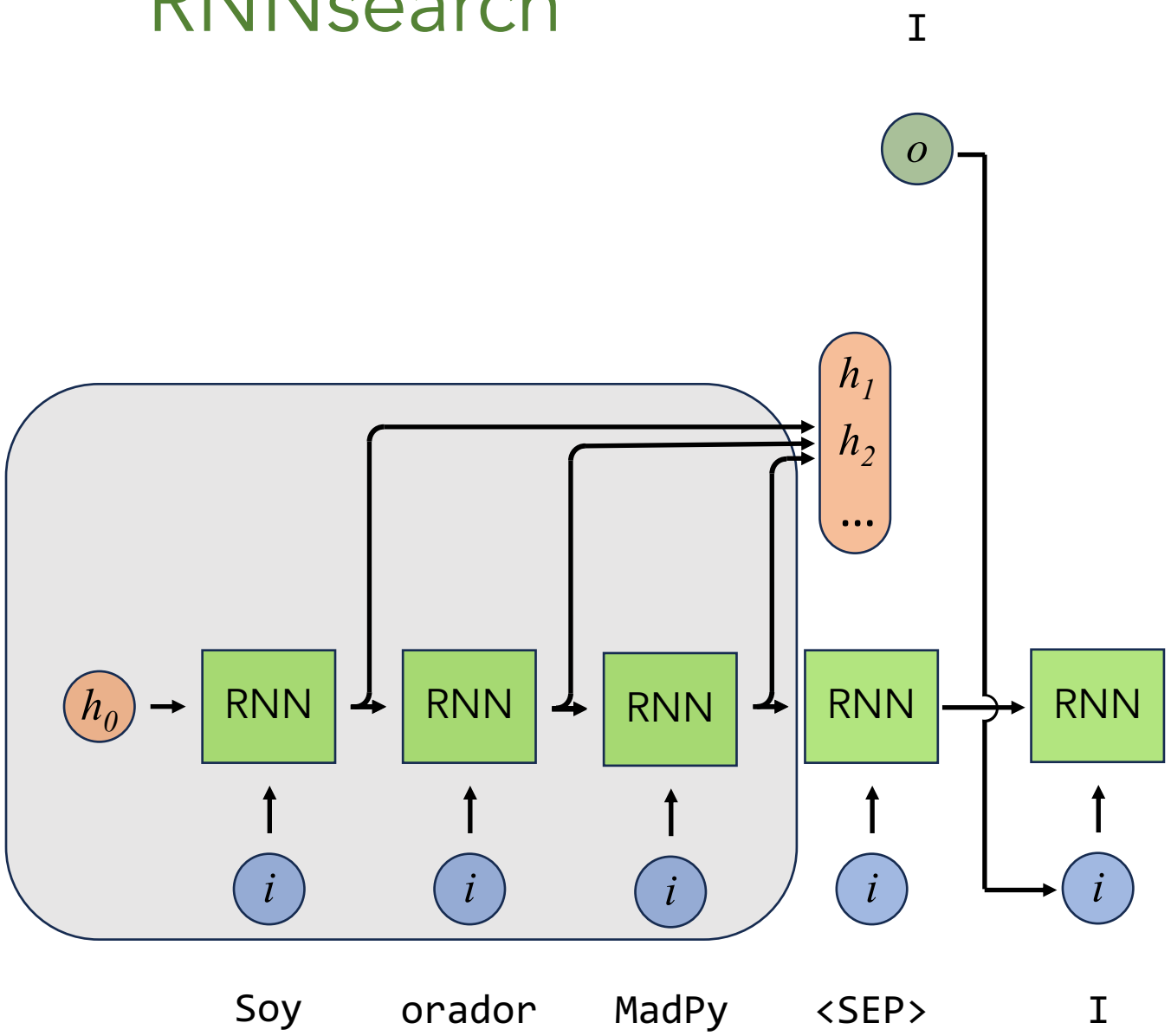
The "Attention" Mechanism



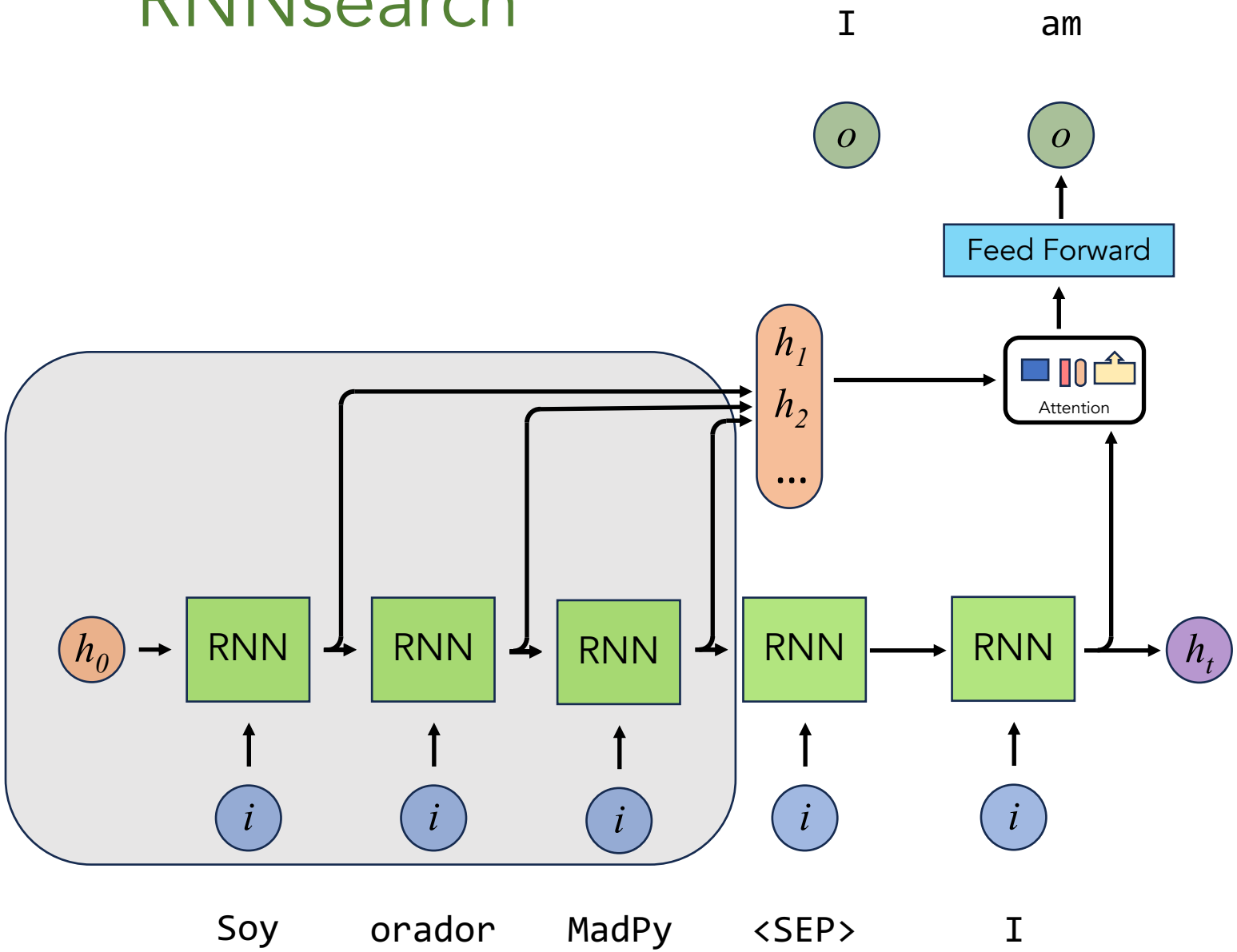
RNNsearch


I
o
Feed Forward
h1
h2
...
Attention
h0
RNN
RNN
RNN
RNN
ht
i
i
i
i
Soy
orador
MadPy
<SEP>

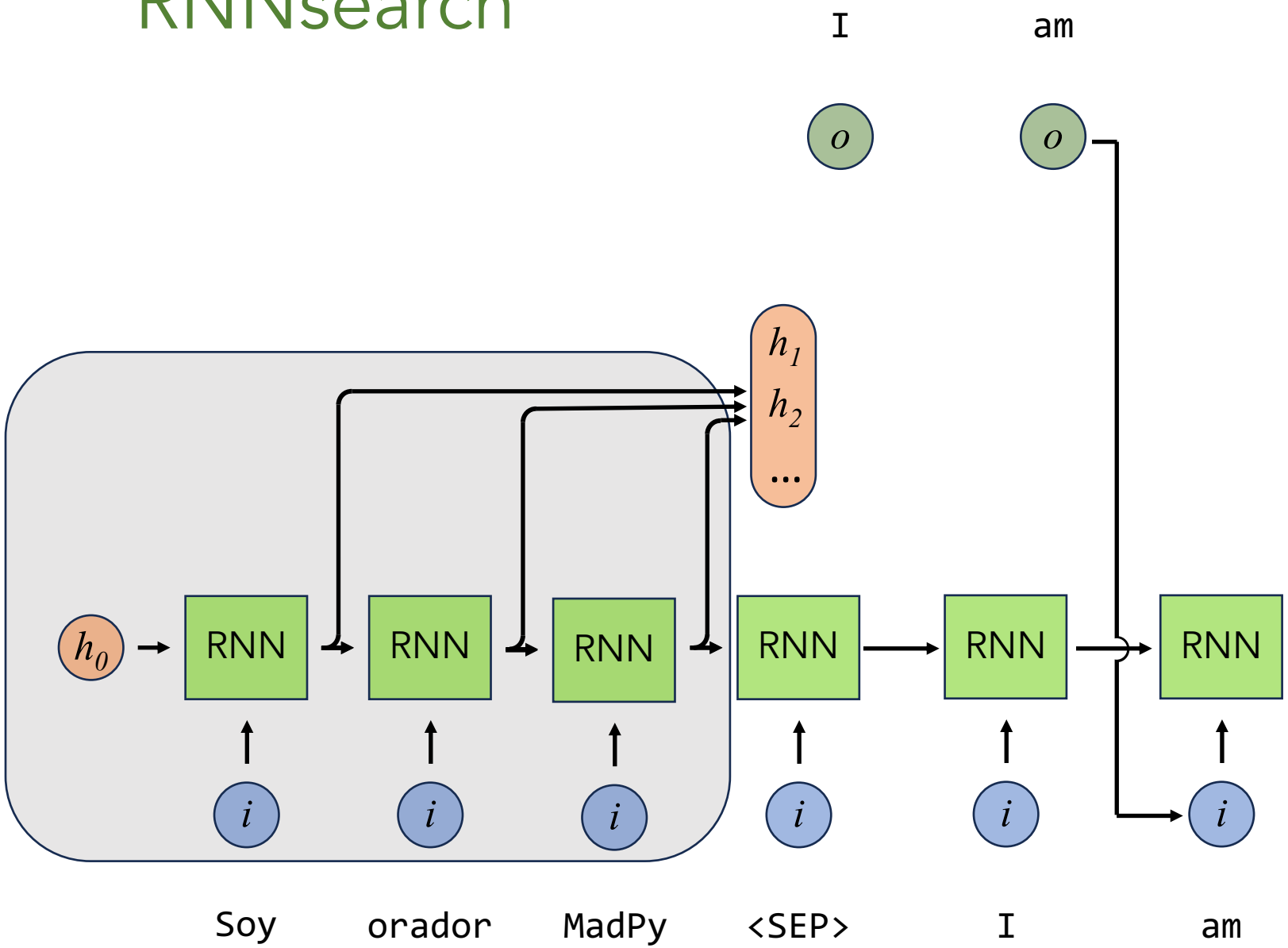
RNNsearch



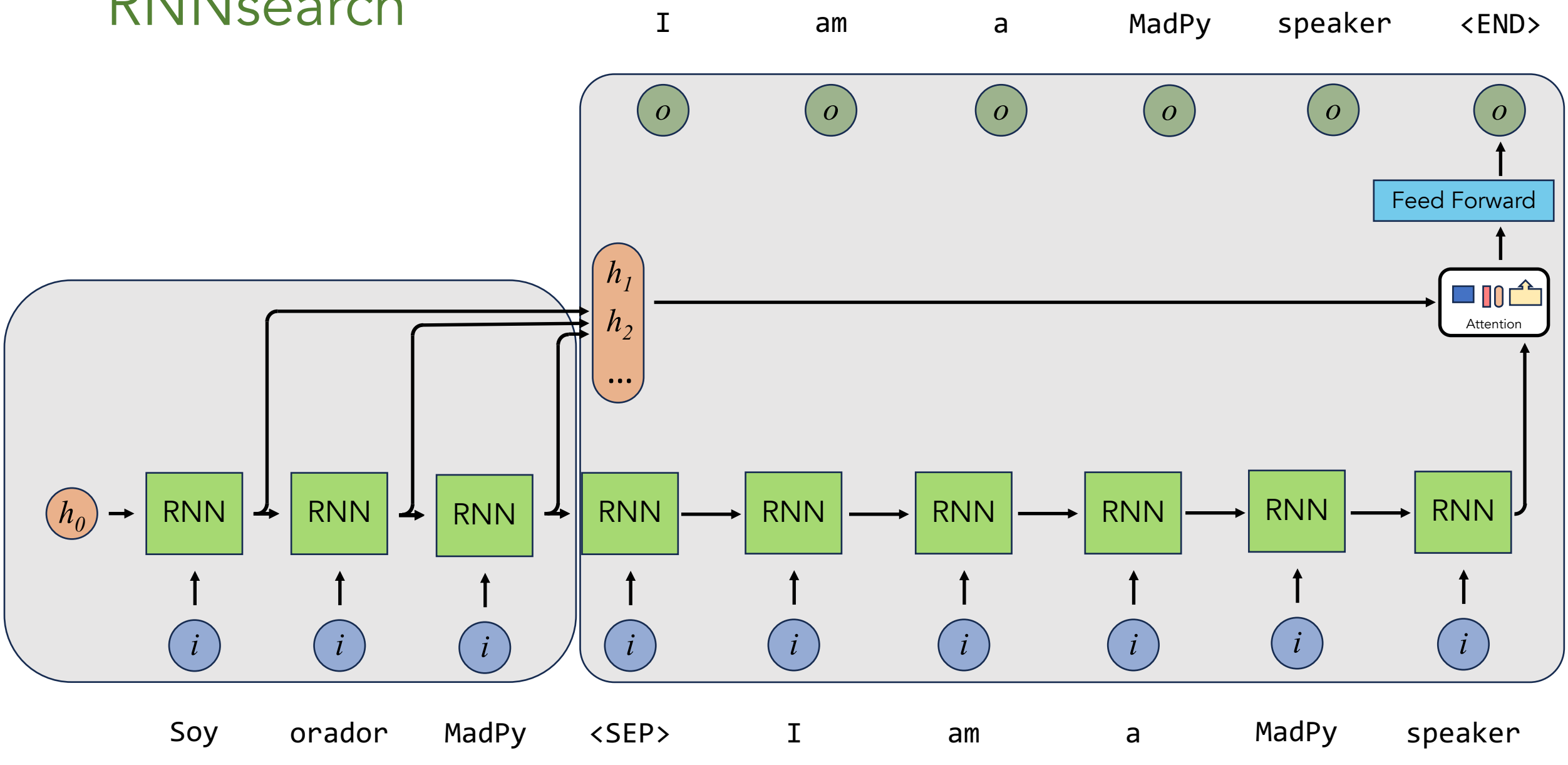
RNNsearch


I
am
o
o
Feed Forward
h_1
h_2
...
Attention
h_0
RNN
RNN
RNN
RNN
RNN
h_t
i
i
i
i
i
Soy
orador
MadPy
<SEP>
I

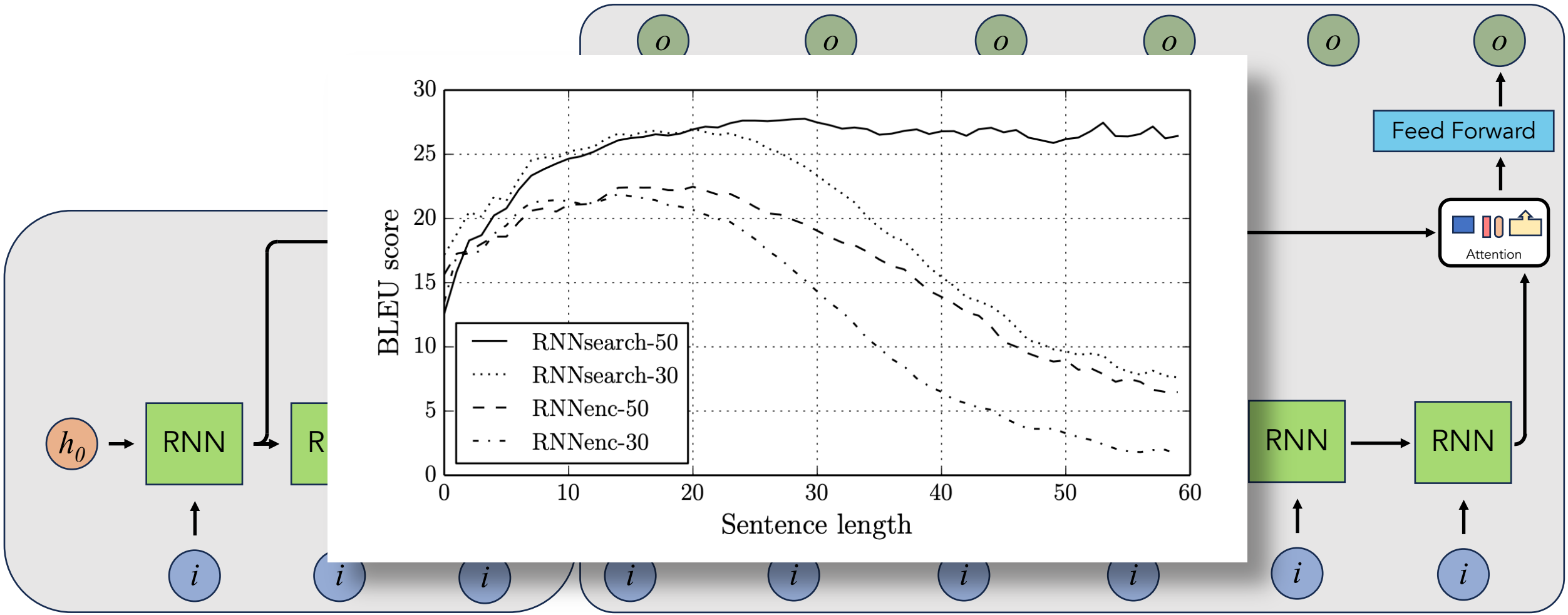
RNNsearch


I
am
o
o
h_1
h_2
...
h_0
RNN
RNN
RNN
RNN
RNN
RNN
i
i
i
i
i
i
Soy
orador
MadPy
<SEP>
I
am

RNNsearch

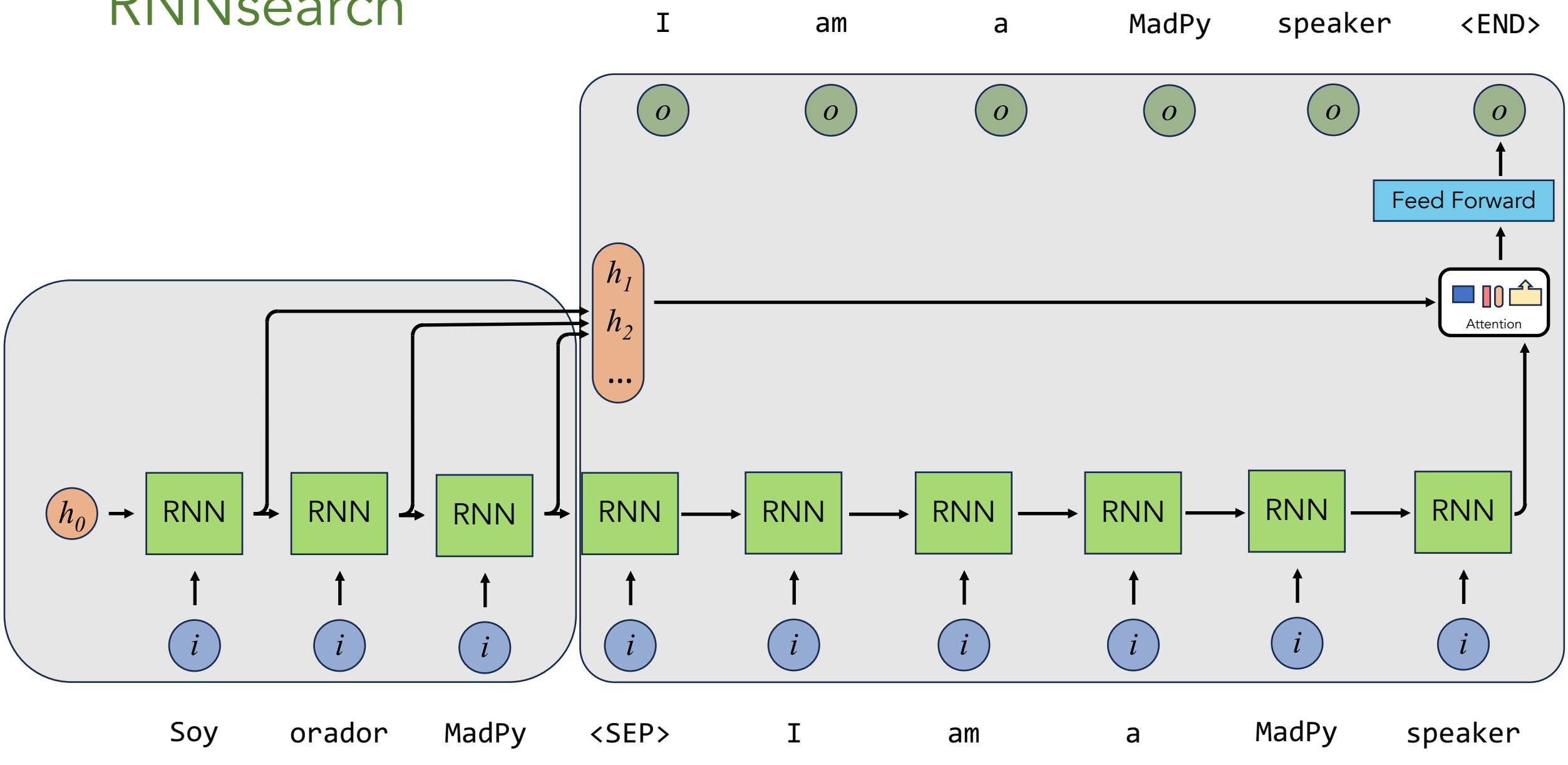

I
am
a
MadPy
speaker
<END>
o
Feed Forward
Attention
h_1
h_2
...
h_0
RNN
i
Soy
orador
MadPy
<SEP>
I
am
a
MadPy
speaker

RNNsearch


I
am
a
MadPy
speaker
<END>
Feed Forward
Attention
RNN
RNN
RNN
Soy
orador
MadPy
<SEP>
I
am
a
MadPy
speaker

BLEU score
Sentence length
RNNsearch-50
RNNsearch-30
RNNenc-50
RNNenc-30
0
5
10
15
20
25
30
0
10
20
30
40
50
60

RNNsearch


I
am
a
MadPy
speaker
<END>
o
Feed Forward
Attention
h_1
h_2
...
h_0
RNN
i
Soy
orador
MadPy
<SEP>
I
am
a
MadPy
speaker

Attention Is All You Need

Ashish Vaswani[*]
Google Brain
avaswani@google.com

Noam Shazeer[*]
Google Brain
noam@google.com

Niki Parmar[*]
Google Research
nikip@google.com

Jakob Uszkoreit[*]
Google Research
usz@google.com

Llion Jones[*]
Google Research
llion@google.com

Aidan N. Gomez[*†]
University of Toronto
aidan@cs.toronto.edu

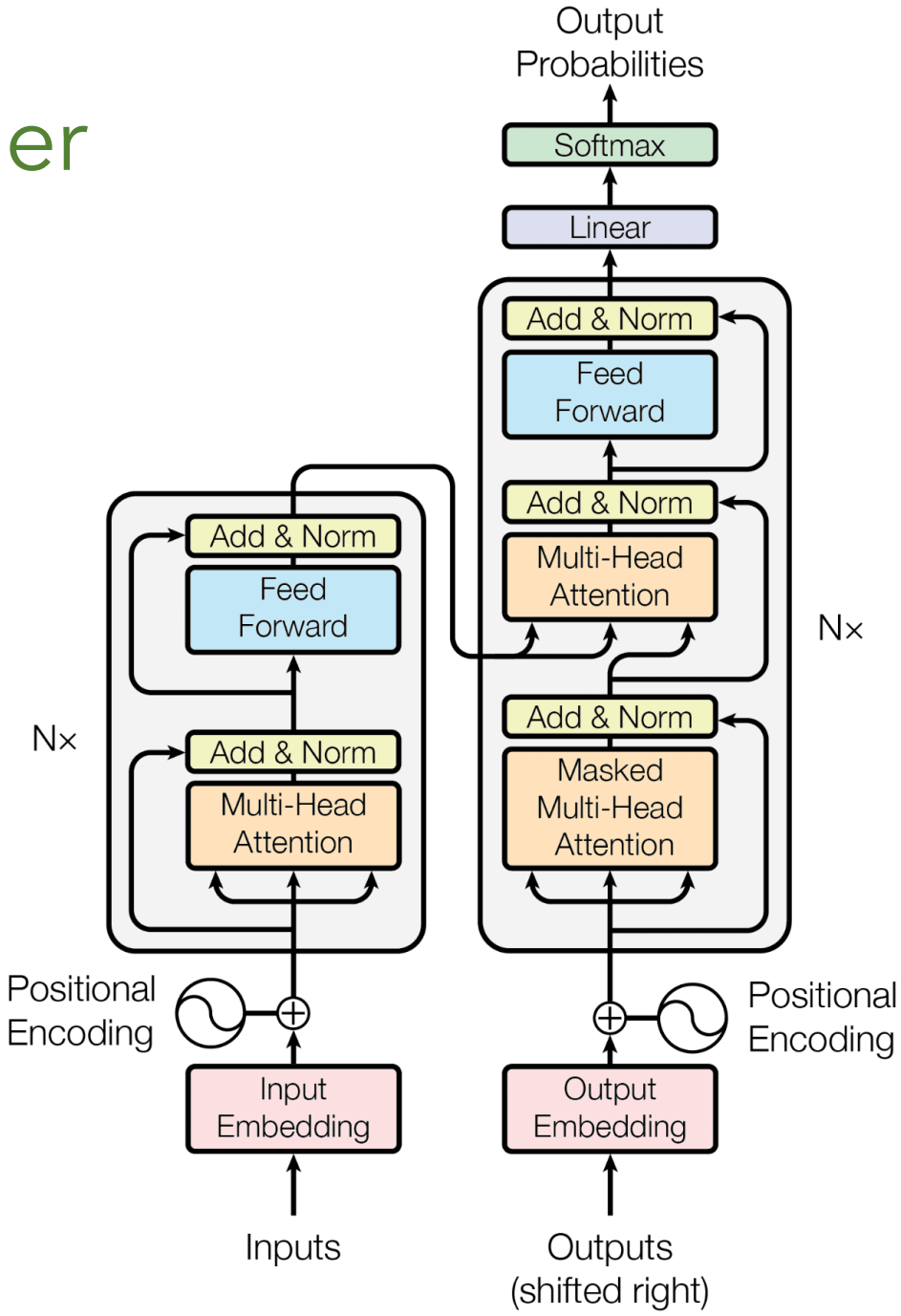
Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

Illia Polosukhin[*]
illia.polosukhin@gmail.com

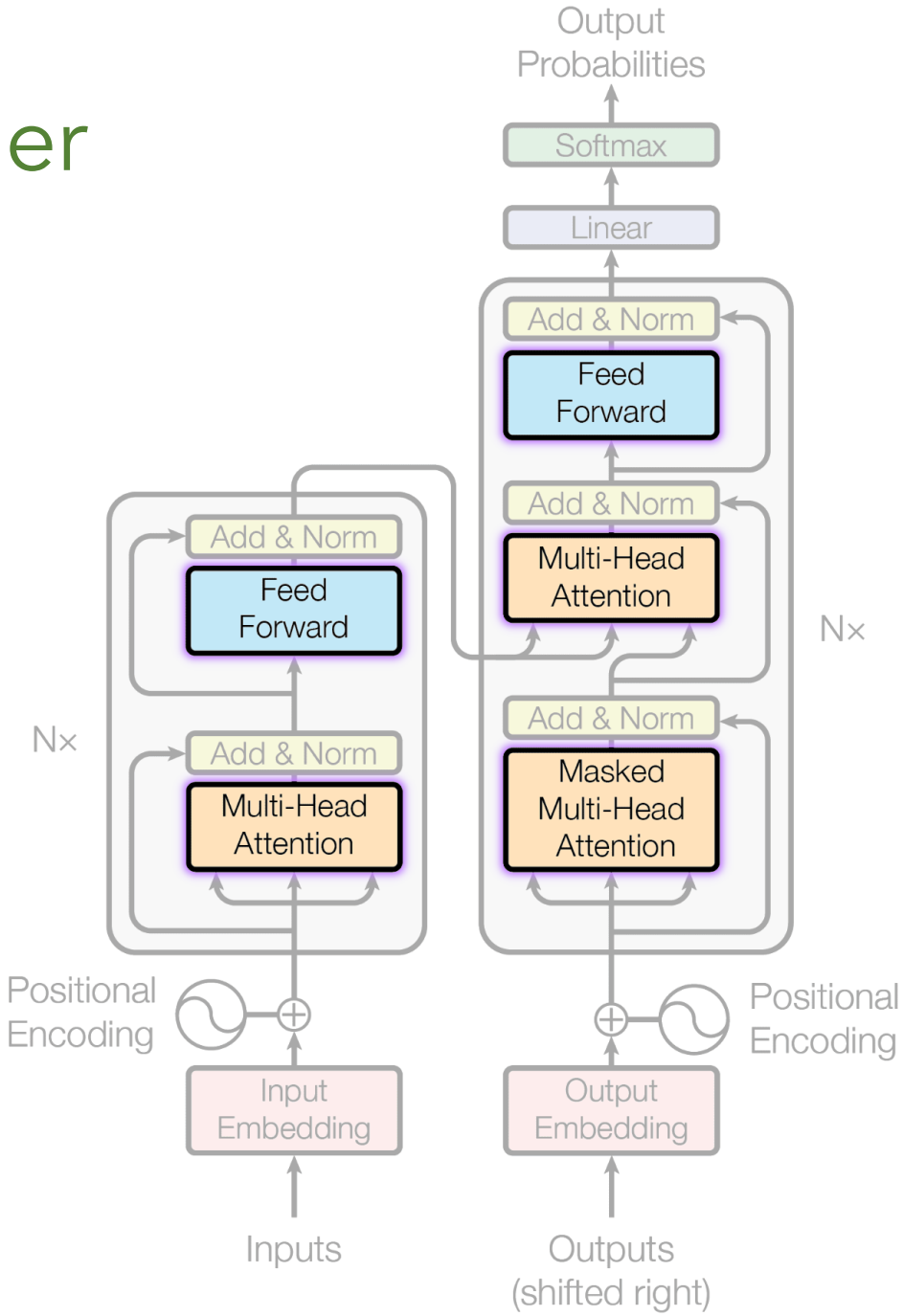
Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder-decoder configuration. The best performing models also connect the encoder and decoder through an attention

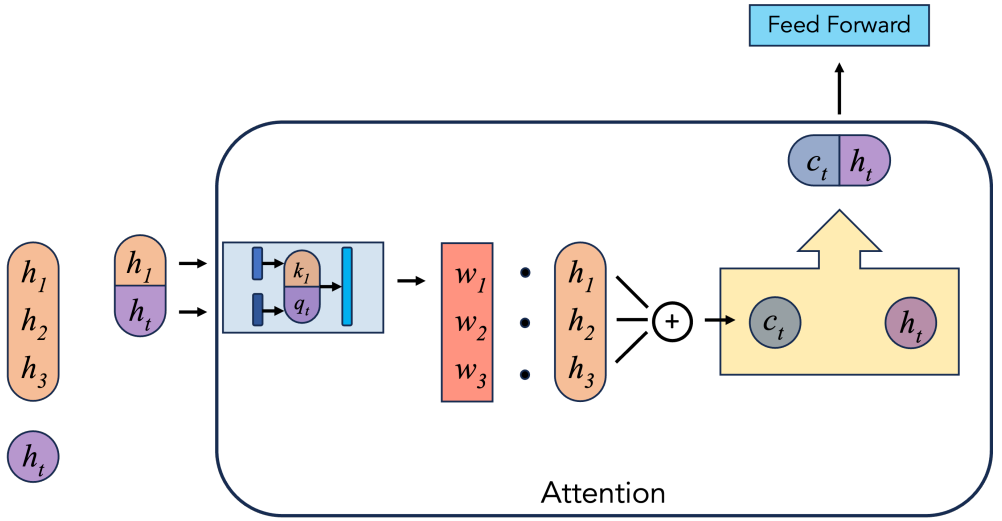
The Transformer



The Transformer



The Evolution of Attention



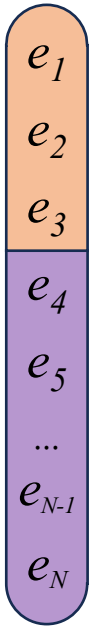
Hidden states generated by RNNs

Alignment NN layers convert:

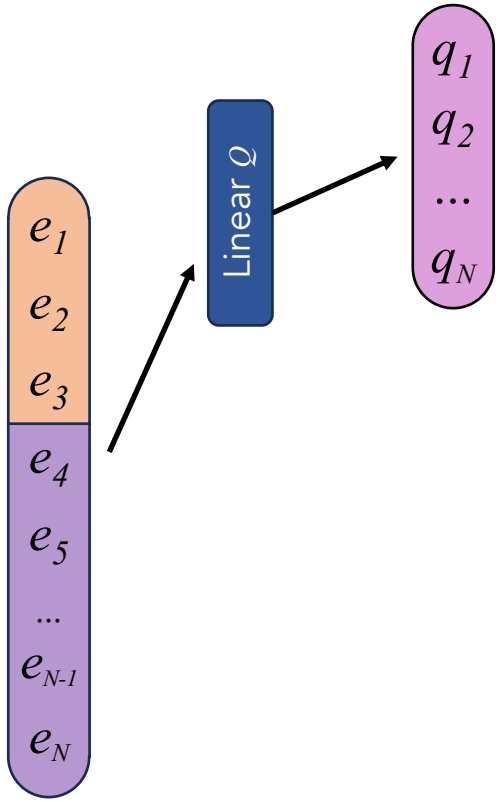
- encoder hidden states to *keys*
- decoder hidden states to *queries*
- *keys* & *queries* to *weights*

weights used to average unmodified encoder hidden states

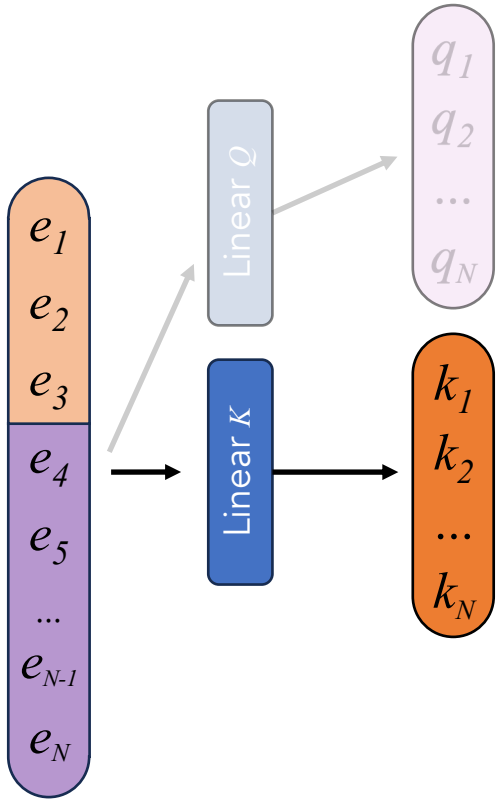
The Evolution of Attention



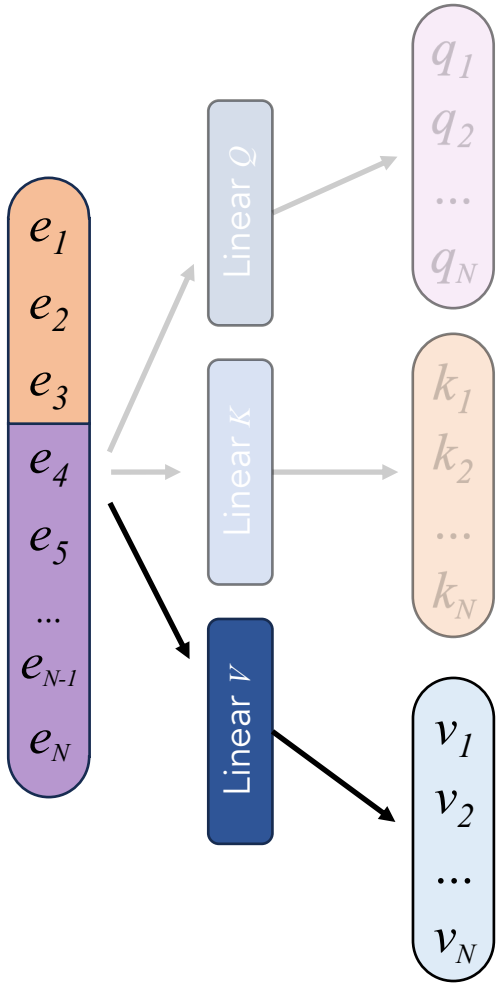
The Evolution of Attention



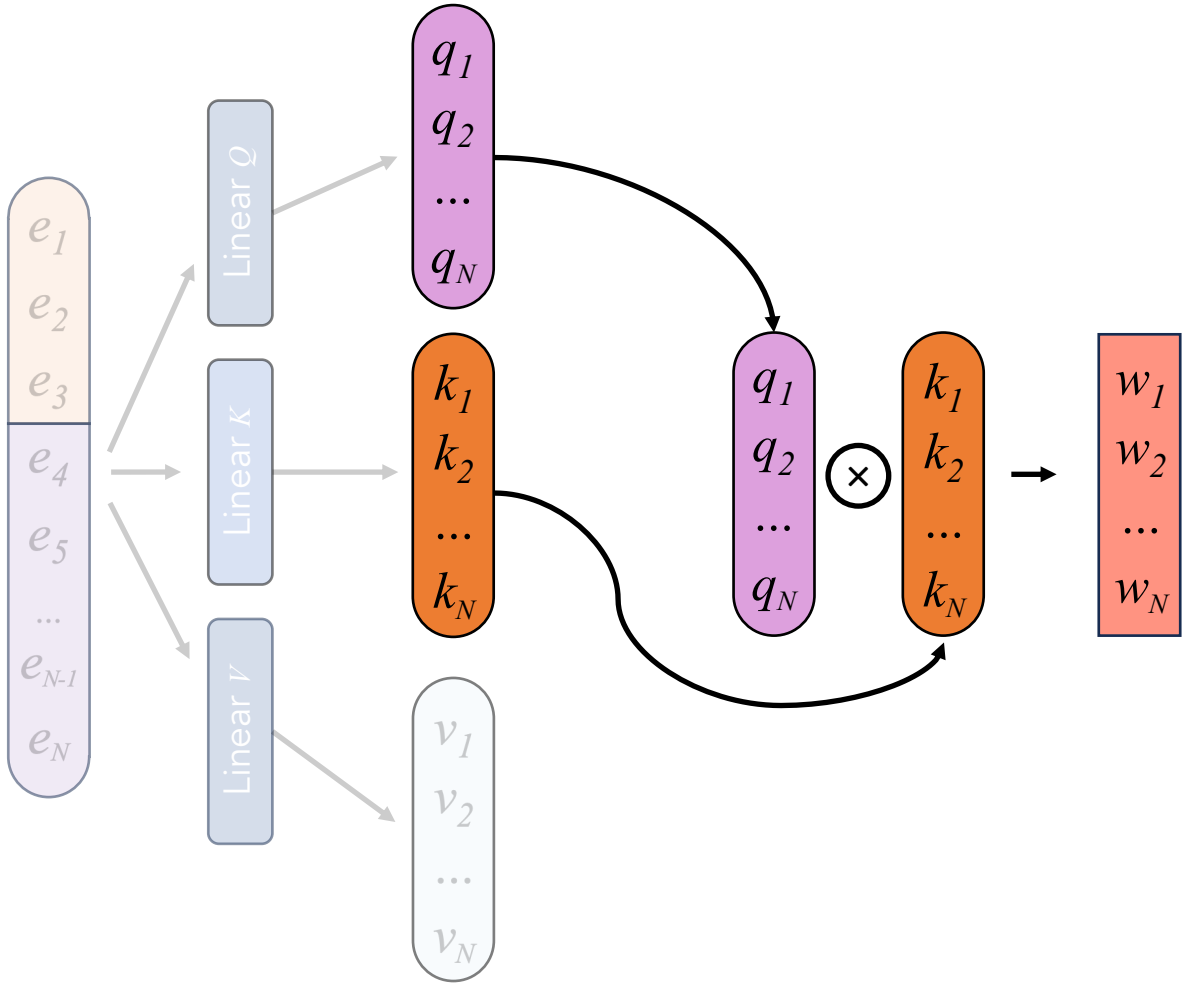
The Evolution of Attention



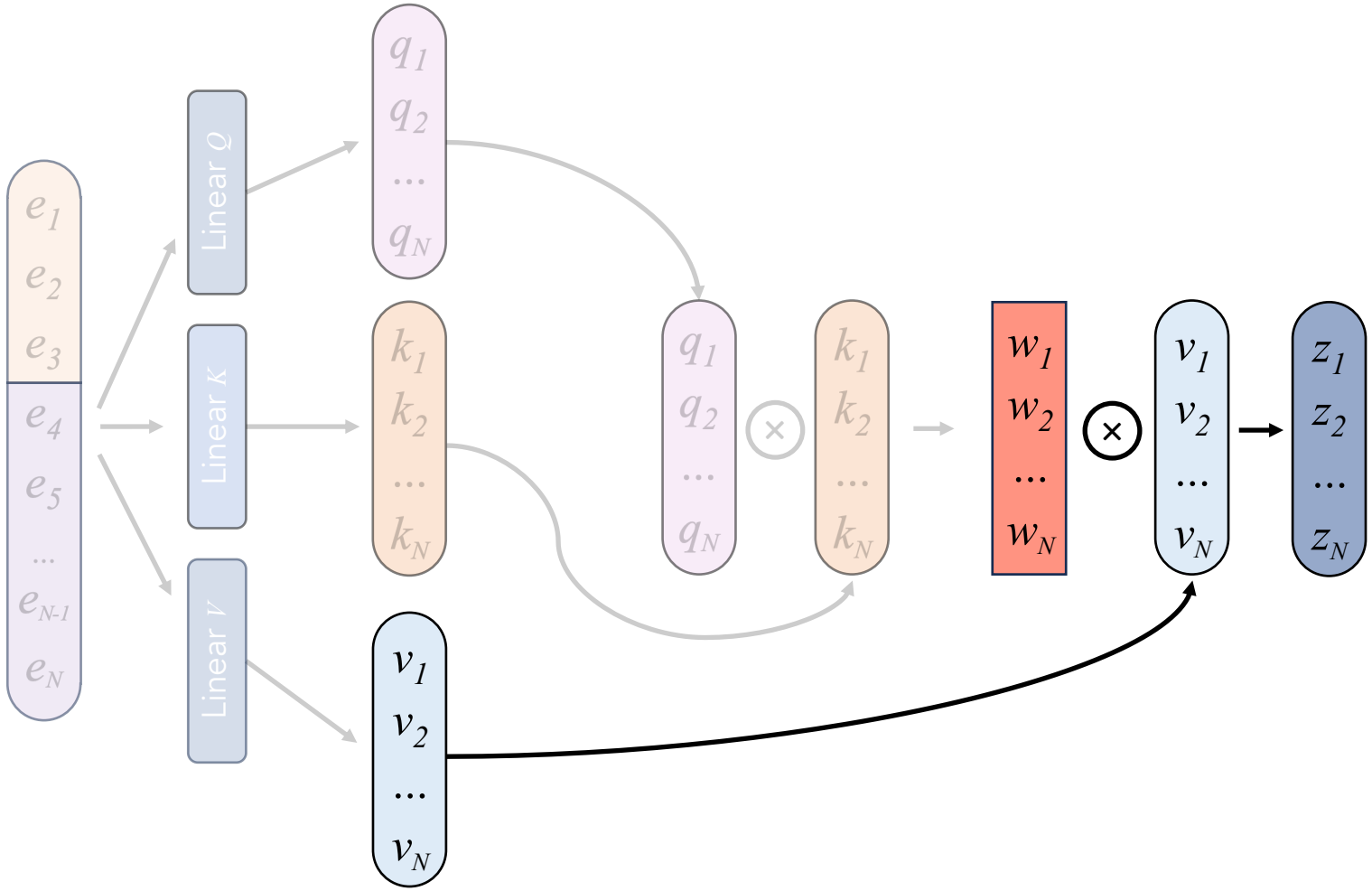
The Evolution of Attention



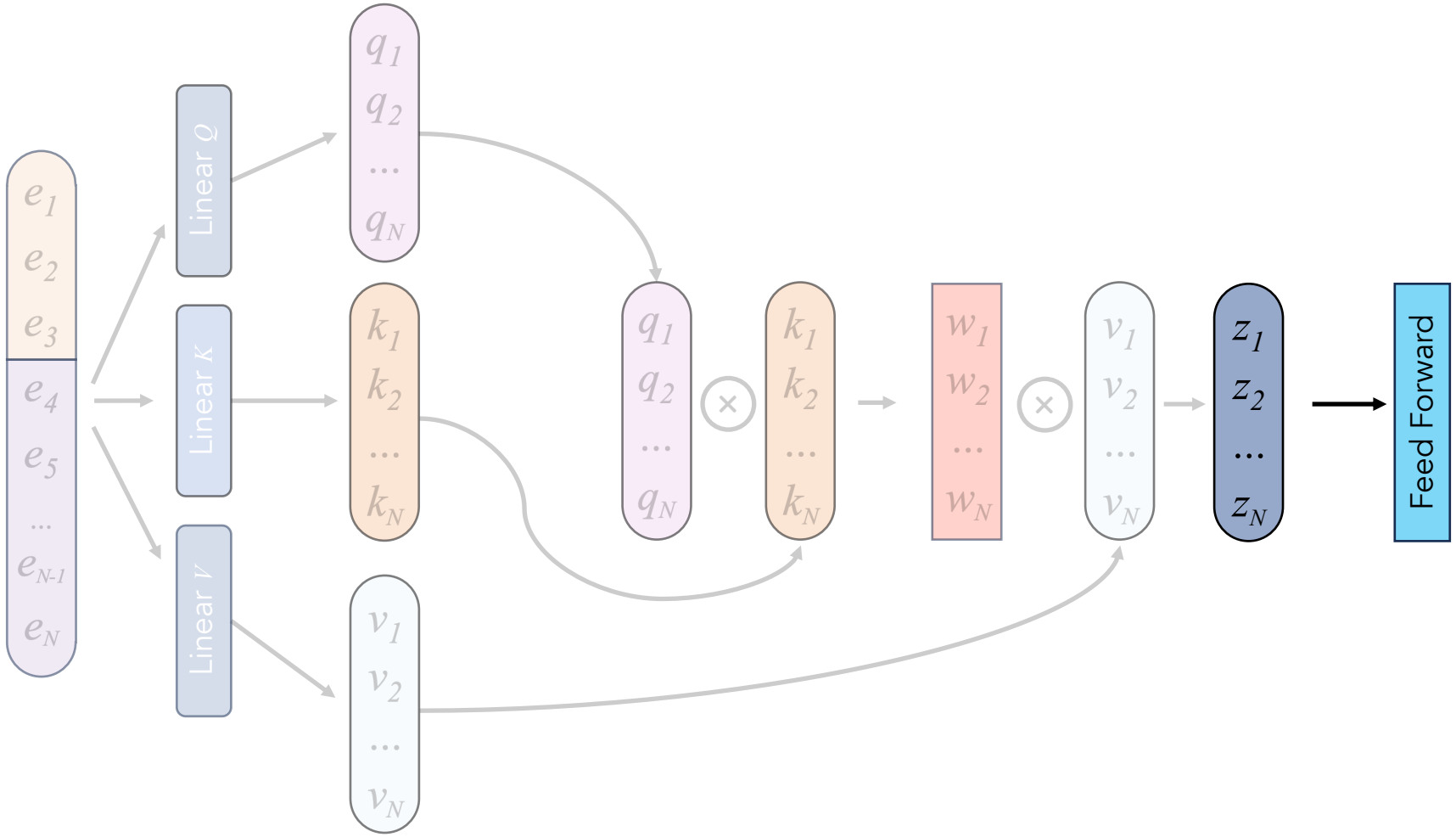
The Evolution of Attention



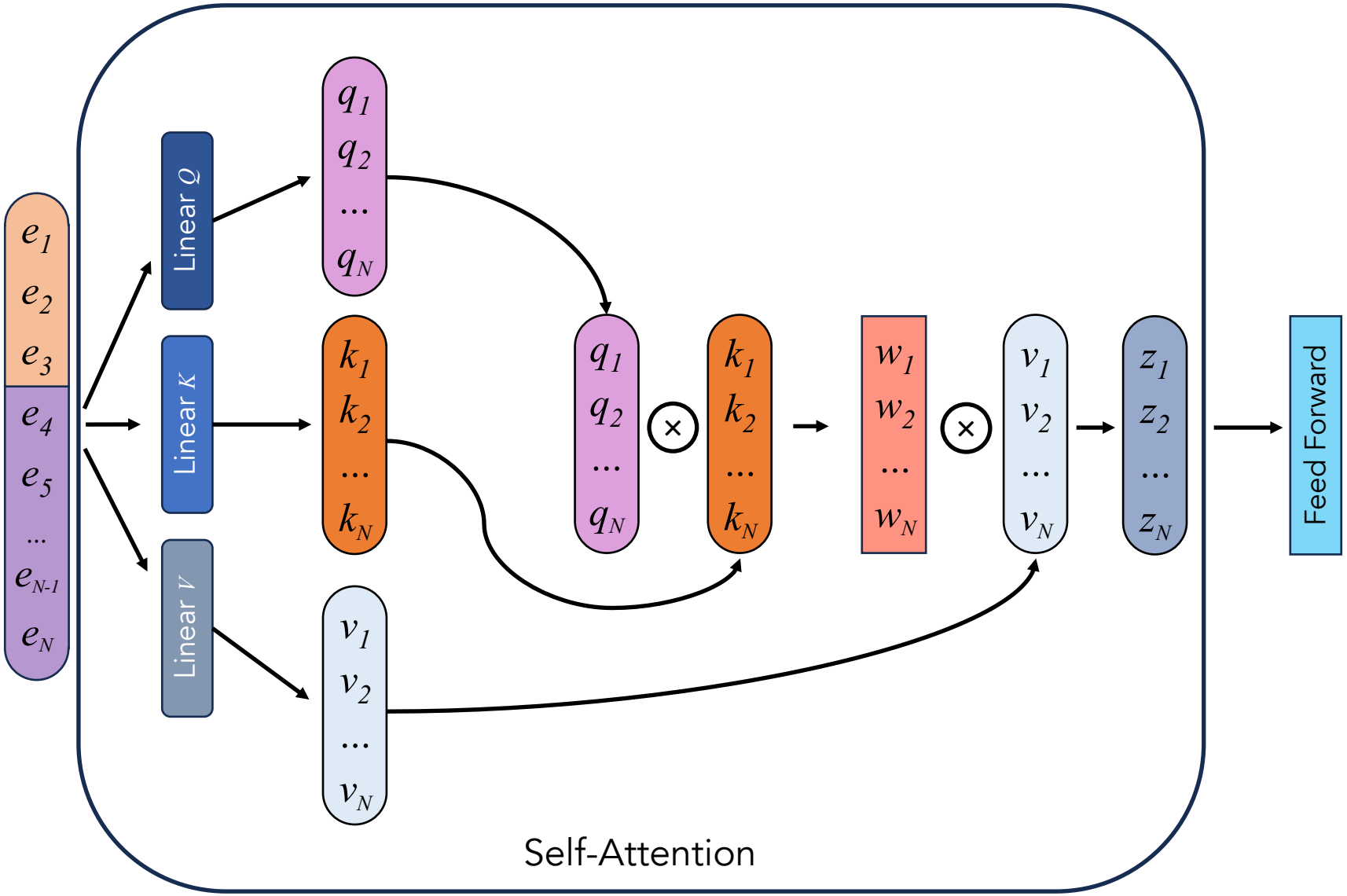
The Evolution of Attention



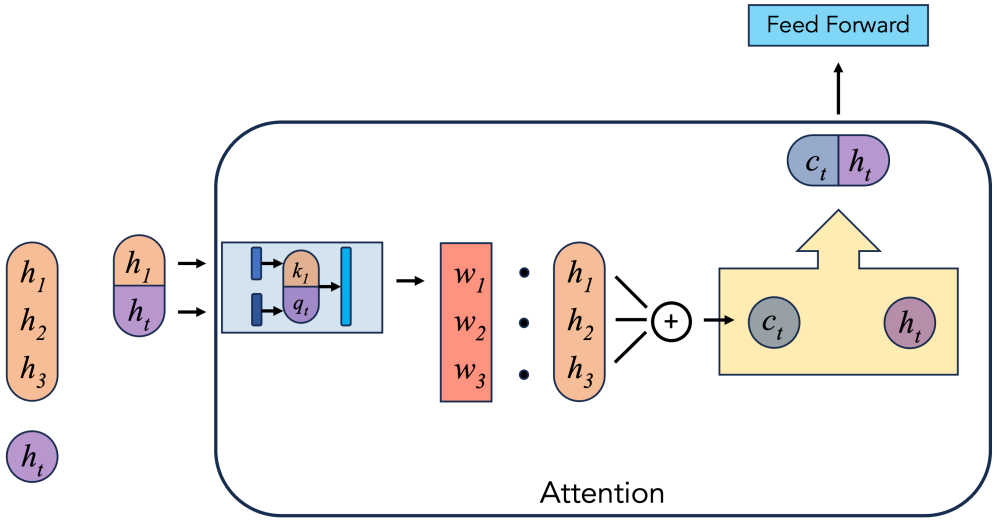
The Evolution of Attention



The Evolution of Attention



The Evolution of Attention

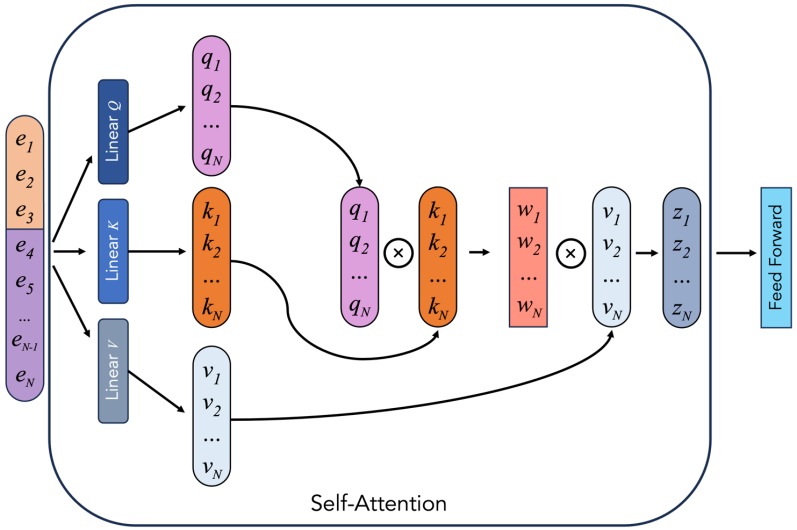


Hidden states generated by RNNs

Alignment NN layers convert:

- encoder hidden states to *keys*
- decoder hidden states to *queries*
- *keys* & *queries* to *weights*

weights used to average unmodified encoder hidden states

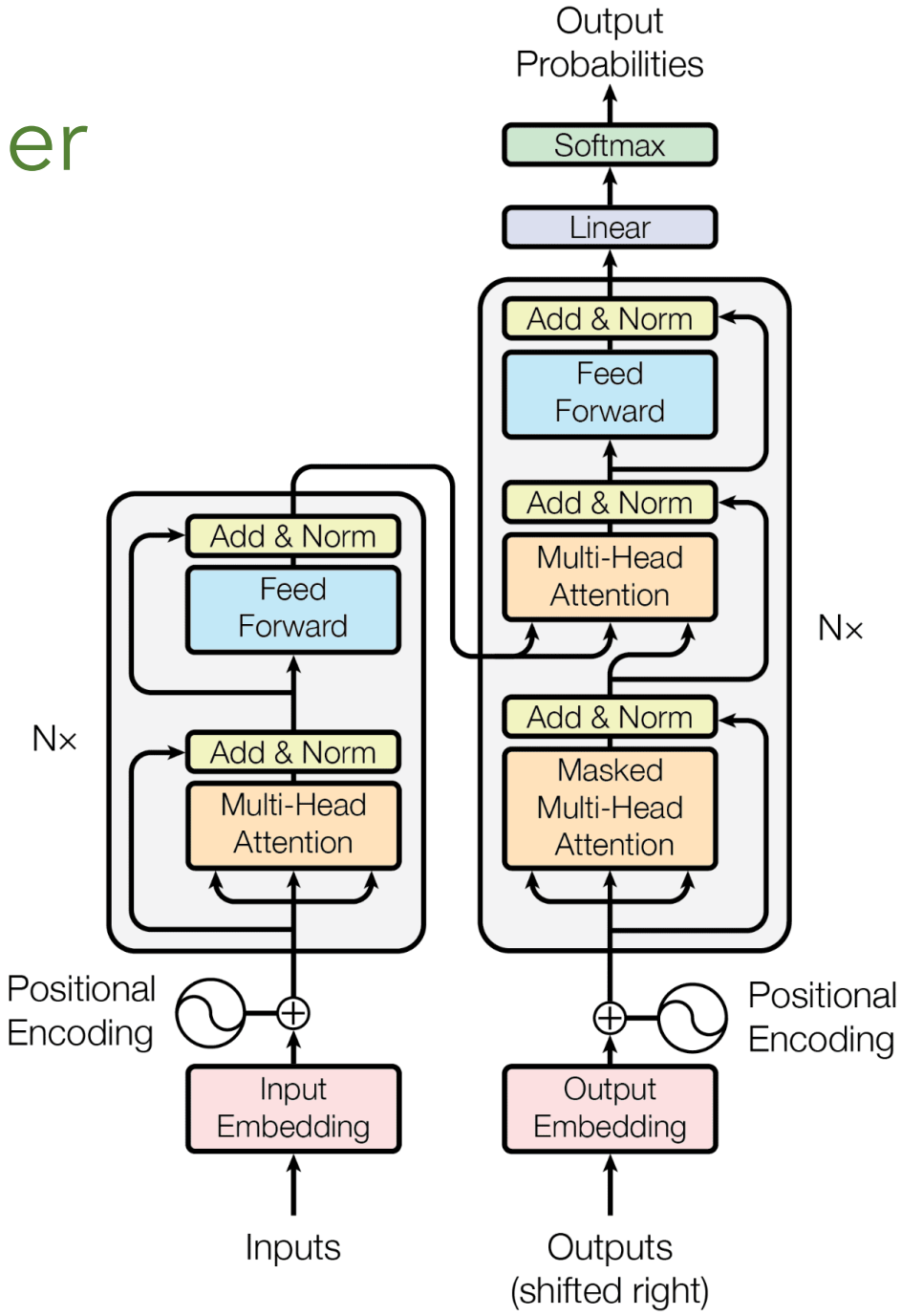


Linear NNs directly create *keys*, *queries*, and *values*

Matrix mult converts *keys* & *queries* to *weights*

weights used to average *values*

The Transformer



Multi-Head Attention