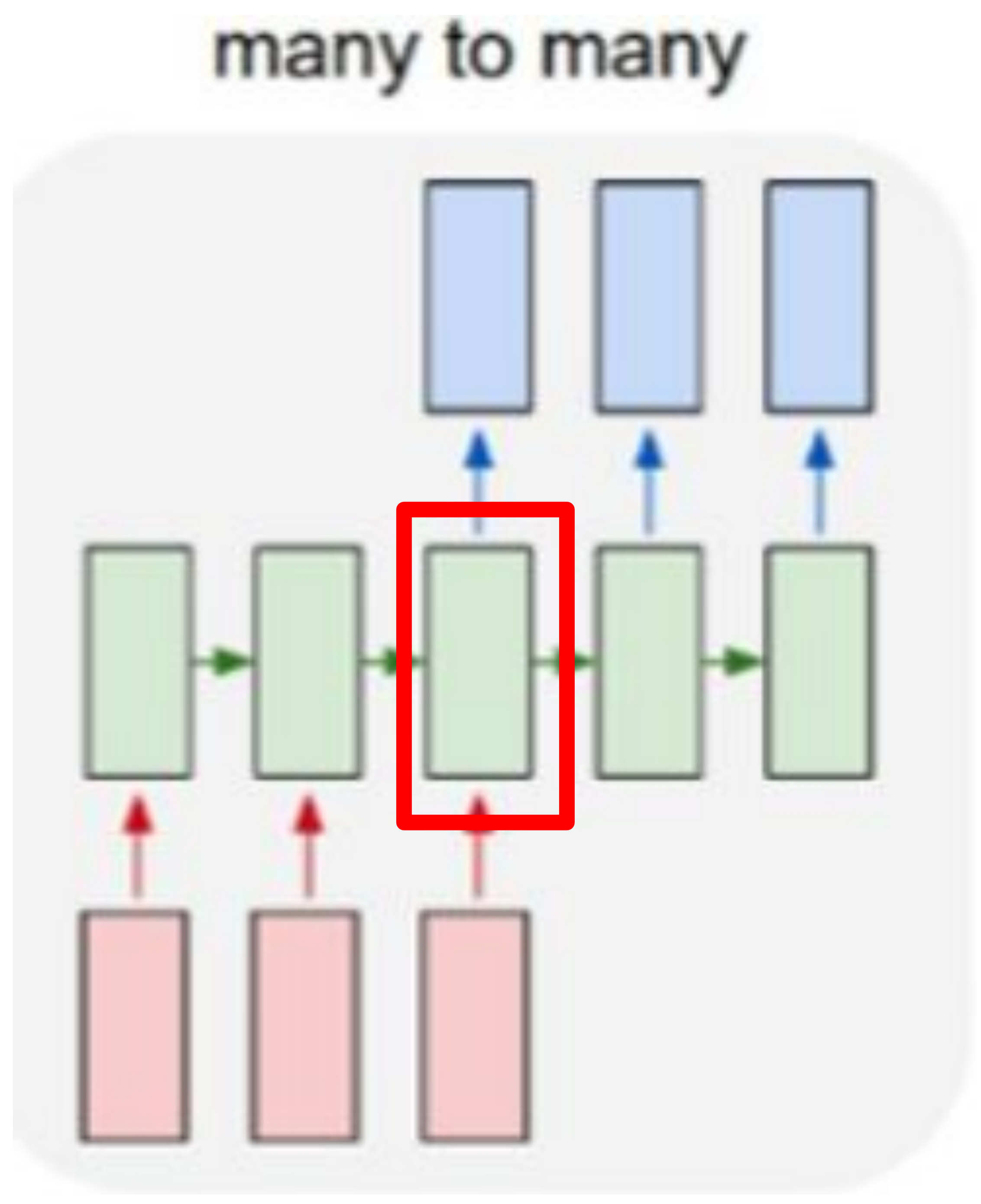
IF3270 Pembelajaran Mesin

Attention & Transformer

Tim Pengajar IF3270

Review: Sequence Model Architecture



fixed-sized input vector xt	RNN/ LSTM state st	fixed-sized output vector ot

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

- Generate **a hidden vector** as representation of input sequence, then use it to generate output.
- For encoder-decoder (Seq2Seq: Sequence to Sequence) architecture.

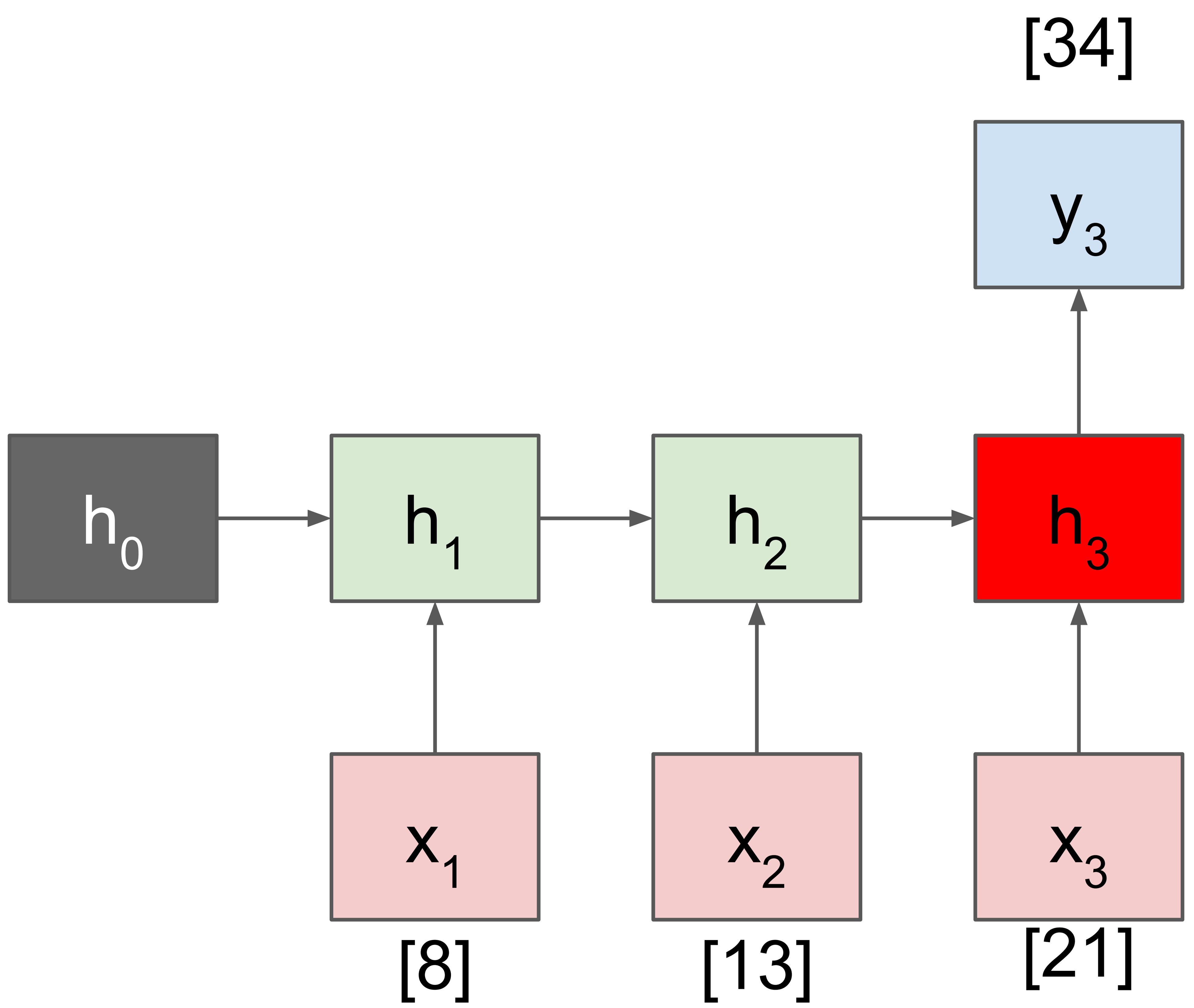
  Traditional Seq2Seq model discard all the intermediate states of the encoder and use only its final states (vector) to initialize the decoder

Illustration: Many to One (Simple Model)

Dataset Fibonacci: 1,2,3, …. f_{1200}

Target f: $x_1\ x_2\ \dots\ x_{ts} \rightarrow x_{ts+1}$

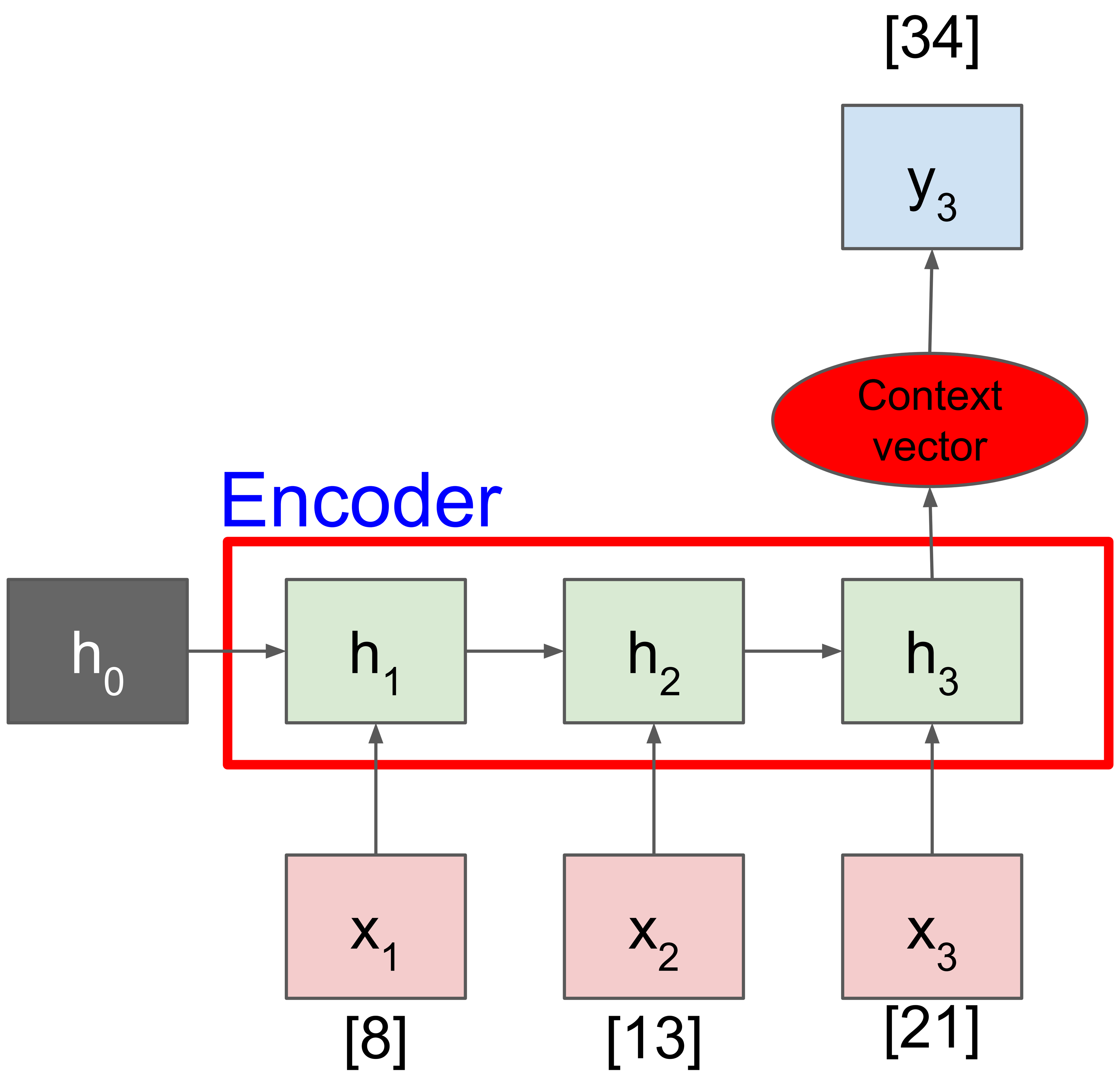
```
modelRNN = Sequential()
modelRNN.add(SimpleRNN(hidden_units, input_shape=(ts,1),
activation='tanh'))
modelRNN.add(Dense(units=dense_units, activation='linear'))
modelRNN.compile(loss='mse', optimizer='adam')
modelRNN.fit(trainX, trainY, epochs=epochs)
```




```
Model: "sequential_49"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_52 (SimpleRNN)   (None, 2)                 8

 dense_62 (Dense)            (None, 1)                 3

=================================================================
Total params: 11 (44.00 Byte)
Trainable params: 11 (44.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
Hidden_units =2
Dense_units = 1

#Par = (1+2+1)*2+(2+1)*1=11
```

Predict Next Element in Series



Input: a sequence of vectors

$$\mathbf{x} = (x_1, \cdots, x_{T_x})$$

Hidden state at time t:

$$h_t = f(x_t, h_{t-1})$$

Context vector: a vector generated from the sequence of the hidden states.

$$c = q(\{h_1, \cdots, h_{T_x}\})$$

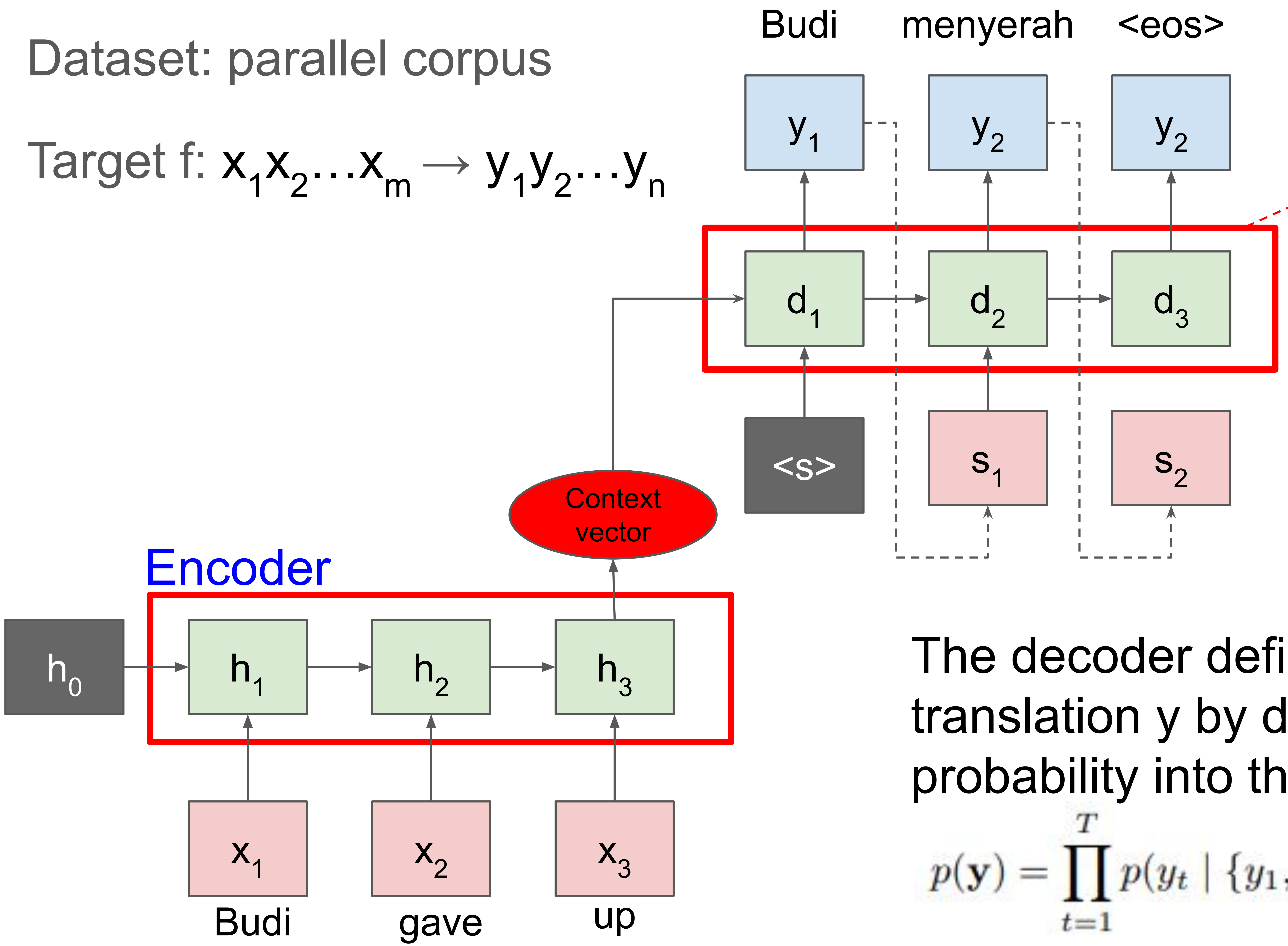
$$q(\{h_1, \cdots, h_T\}) = h_T$$

Bahdanau, D., Cho, K. H., & Bengio, Y. (2015, January). Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015*.

Encoder - Decoder: Machine Translation

Dataset: parallel corpus

Target f: $x_1 x_2 \ldots x_m \rightarrow y_1 y_2 \ldots y_n$



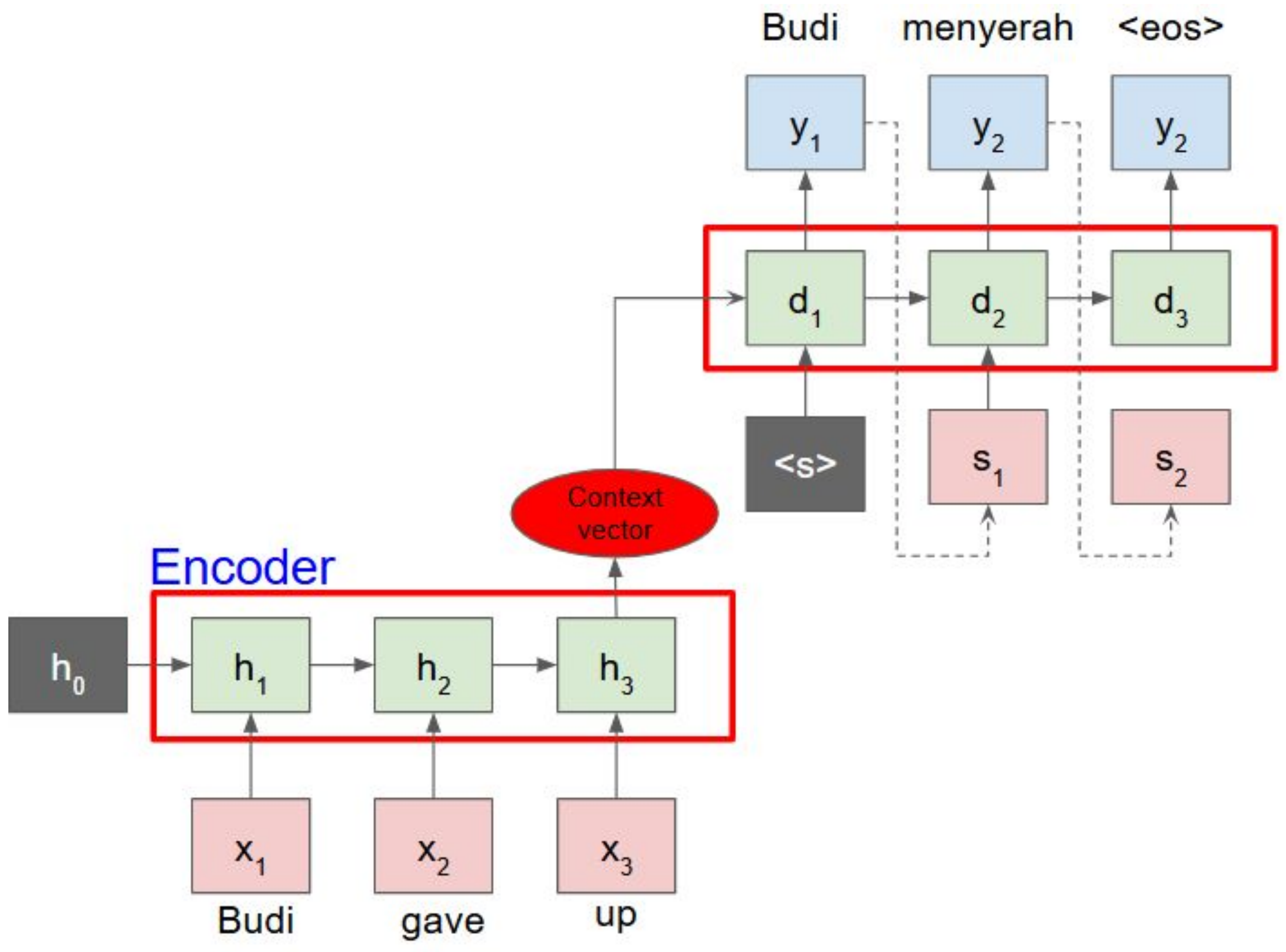
Decoder is trained to predict the next word $y_{t'}$ given the context vector c and all the previously predicted words $\{y_1, \ldots, y_{t'-1}\}$.

$$p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c)$$

The decoder defines a probability over the translation y by decomposing the joint probability into the ordered conditionals:

$$p(\mathbf{y}) = \prod_{t=1}^{T} p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c)$$

Encoder - Decoder Architecture Example




```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_3 (InputLayer)           [(None, None, 74)]   0           []

 input_4 (InputLayer)           [(None, None, 78)]   0           []

 lstm_2 (LSTM)                  [(None, 256),        338944      ['input_3[0][0]']
                                 (None, 256),
                                 (None, 256)]

 lstm_3 (LSTM)                  [(None, None, 256),  343040      ['input_4[0][0]',
                                 (None, 256),                     'lstm_2[0][1]',
                                 (None, 256)]                     'lstm_2[0][2]']

 dense_1 (Dense)                (None, None, 78)     20046       ['lstm_3[0][0]']

==================================================================================================
Total params: 702030 (2.68 MB)
Trainable params: 702030 (2.68 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html

Encoder - Decoder: Weakness & Solution

- The final hidden state of the encoder creates an information bottleneck. It has to capture the meaning of the whole input sequence because this is all the decoder has access to when generating the output.
    - especially challenging for long sequences
- Alternative solution: allowing the decoder to have access to all of the encoder's hidden states.
    - The general mechanism for this is called **attention** and is a key component in many modern neural network architectures.

Central idea behind Attention is not to throw away those intermediate encoder states but to utilize all the states in order to **construct the context vectors** required by the decoder to generate the output sequence

Bahdanau, D., Cho, K. H., & Bengio, Y. (2015, January). Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015*.

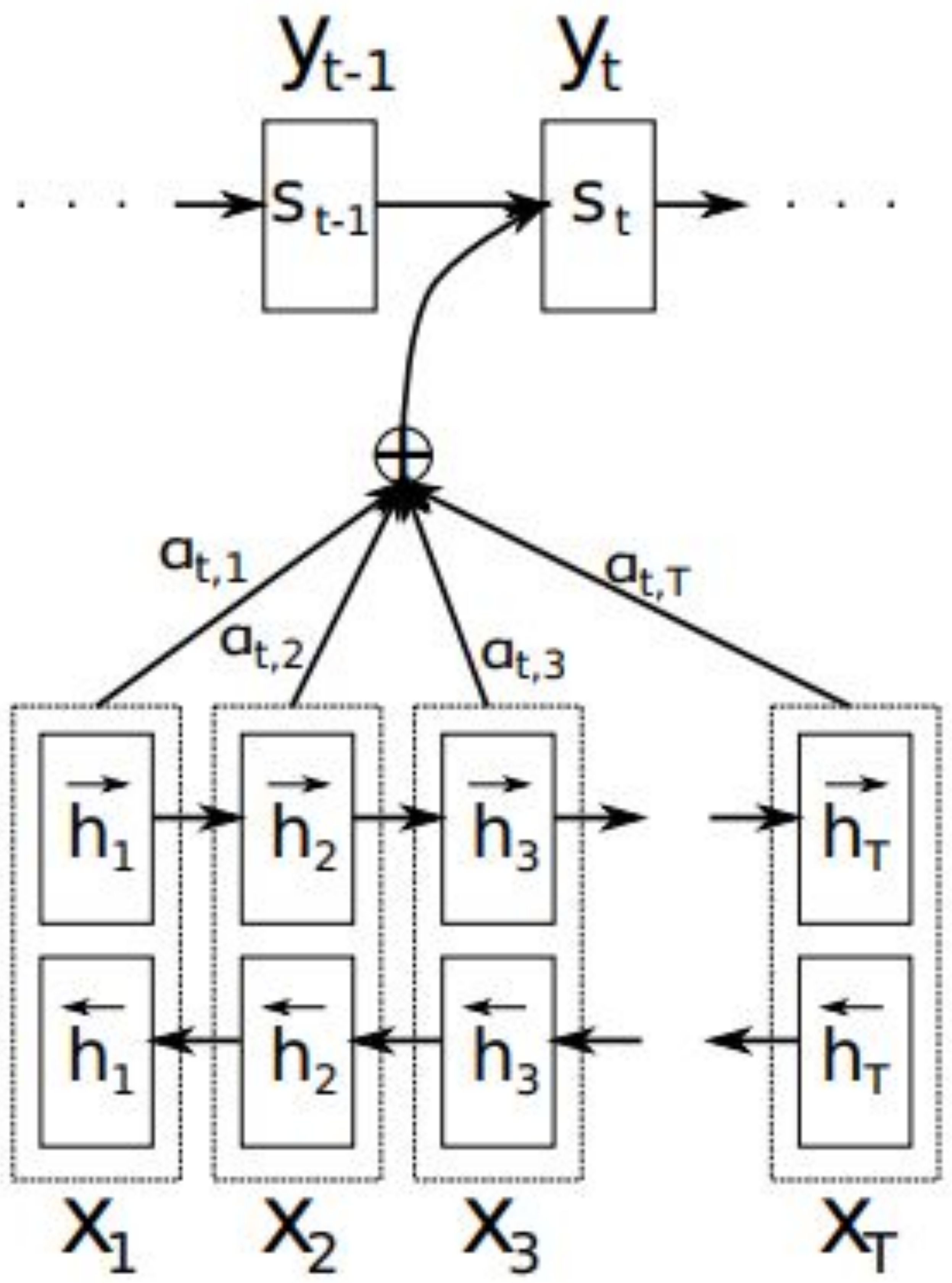
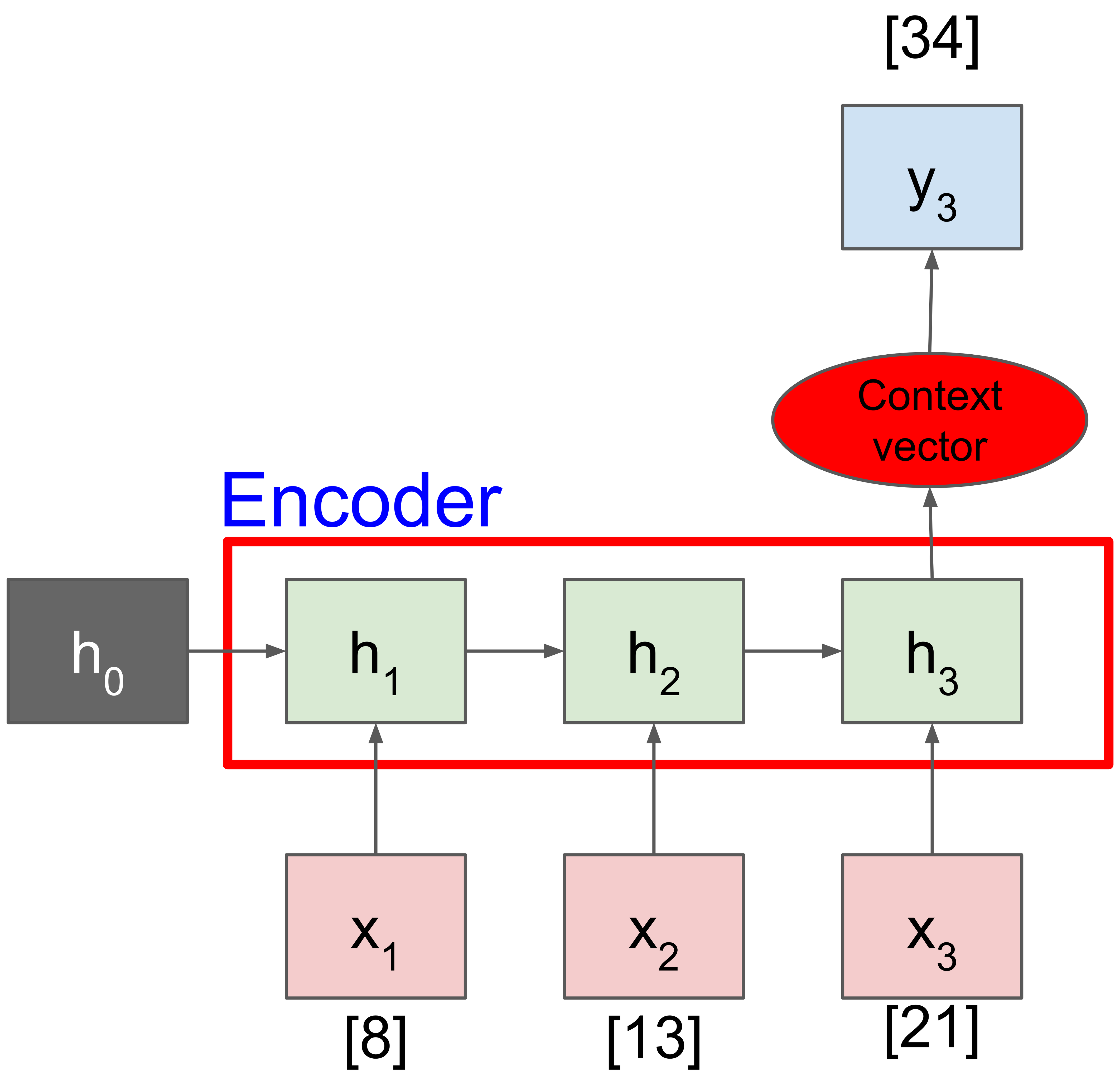
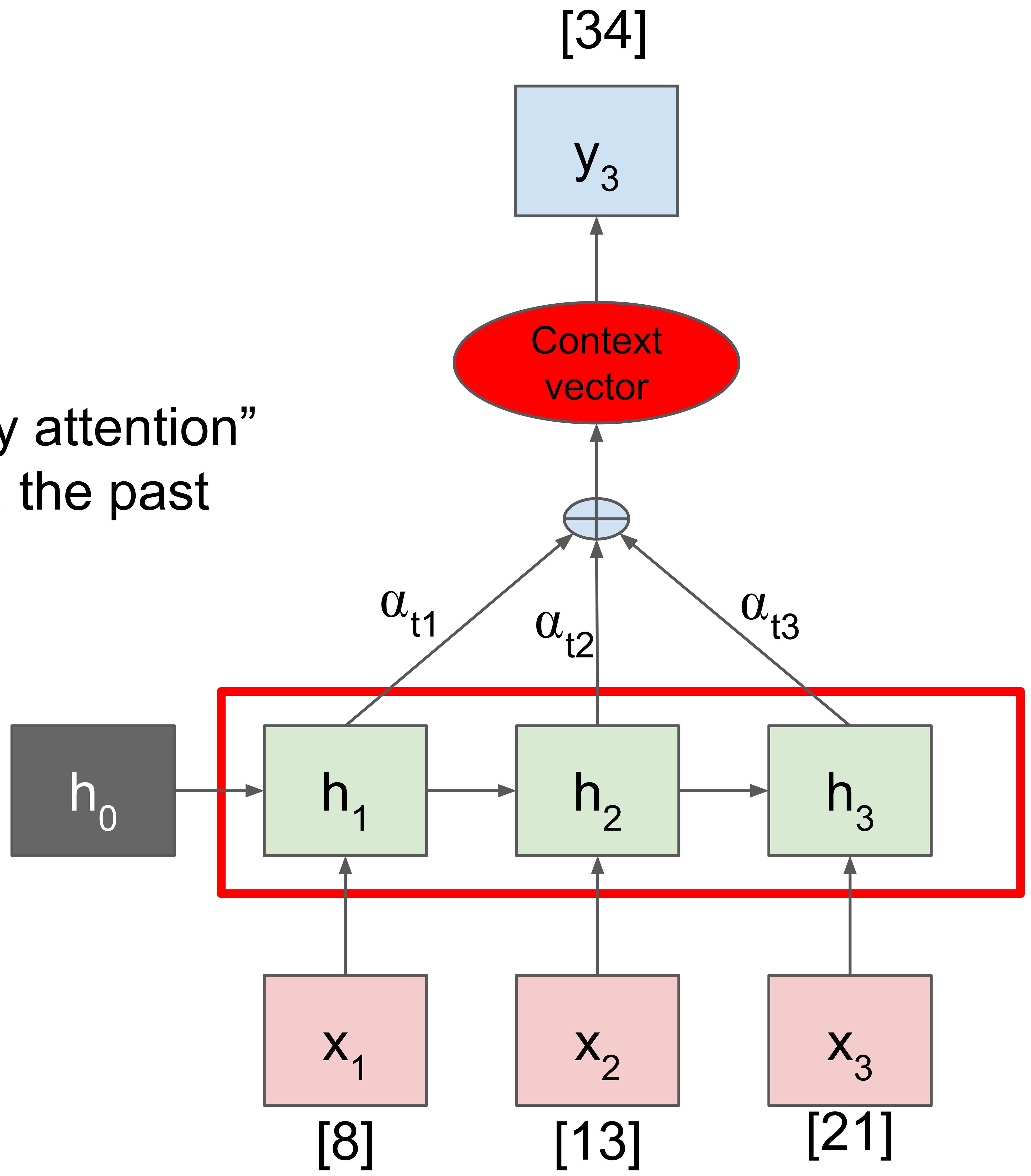


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Many to One with Attention



Assign a weight or "pay attention" to the specific states in the past



https://machinelearningmastery.com/adding-a-custom-attention-layer-to-recurrent-neural-network-in-keras/

Many to One with Attention

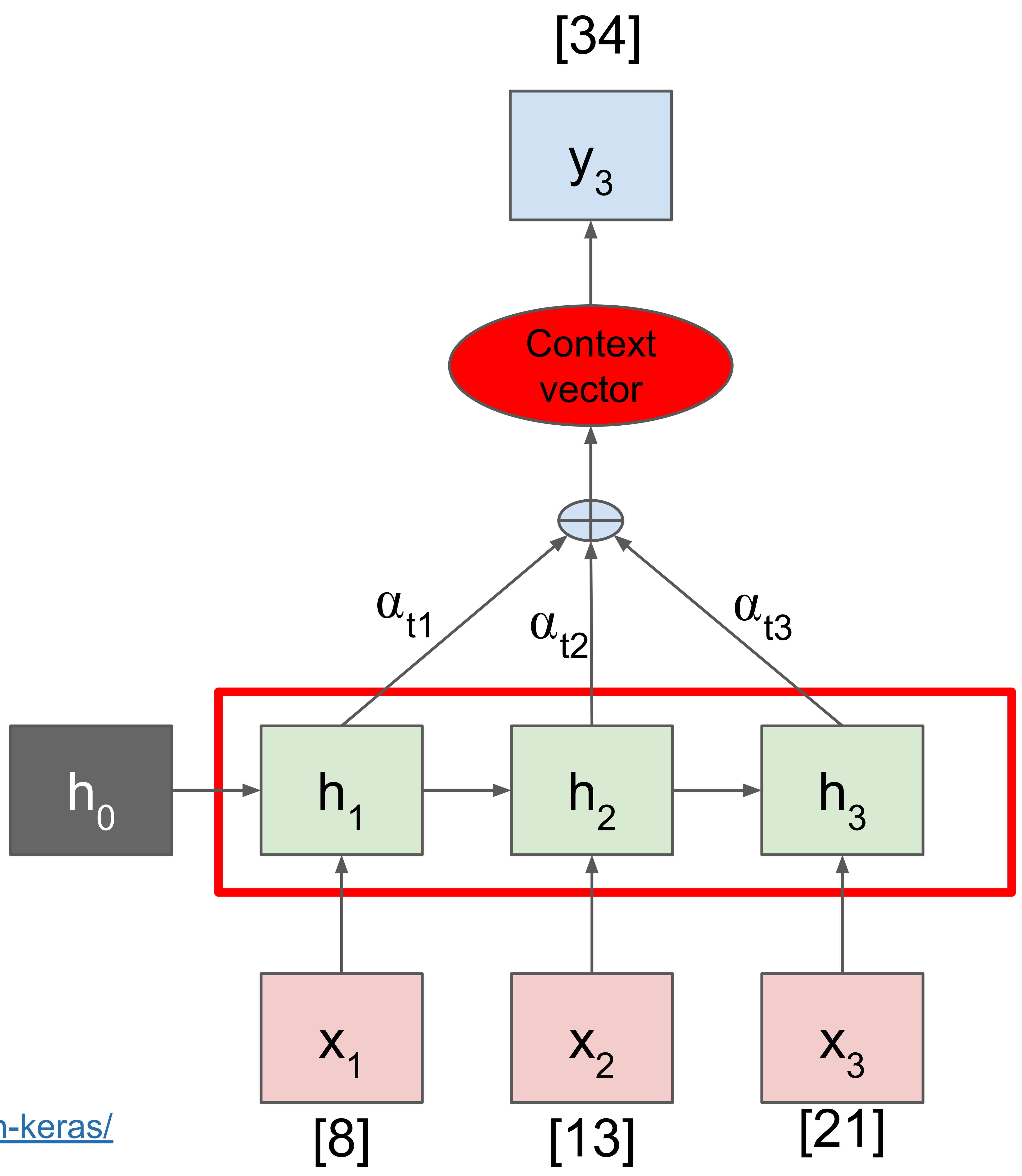
Input: $\mathbf{x} = (x_1, \cdots, x_{T_x})$
Hidden state at time t: $h_t = f(x_t, h_{t-1})$

Context vector c_i for each target y_i:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j.$$

For Fibonacci series, the mean square error on the test set is lower with the attention layer.

https://machinelearningmastery.com/adding-a-custom-attention-layer-to-recurrent-neural-network-in-keras/



Many to One with Attention Architecture Example

```
Model: "sequential_49"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_52 (SimpleRNN)   (None, 2)                 8

 dense_62 (Dense)            (None, 1)                 3

=================================================================
Total params: 11 (44.00 Byte)
Trainable params: 11 (44.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

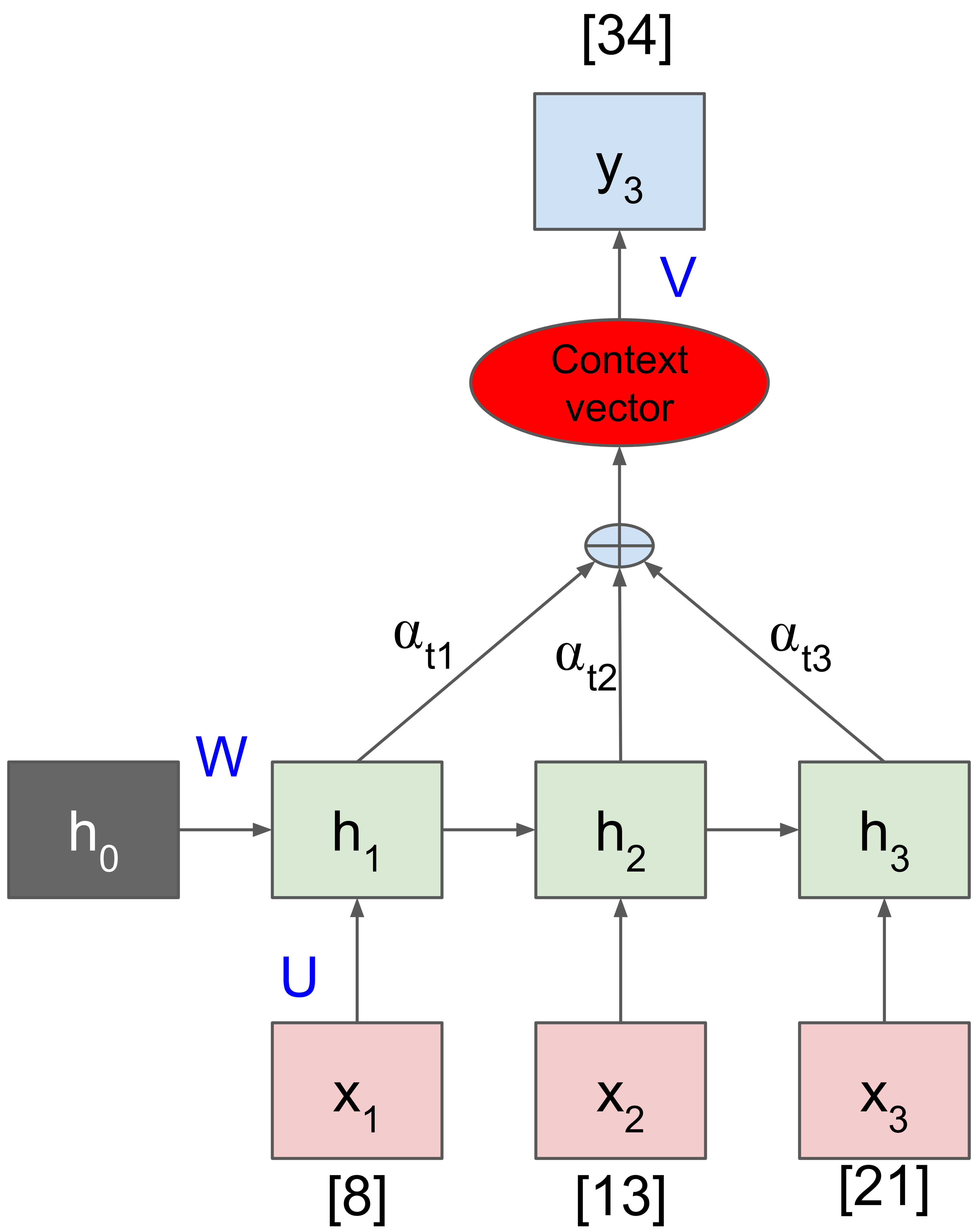
```
Hidden_units =2
Dense_units = 1
Time_steps = 20

#Params = (1+2+1)*2+(2+1)*1=11
```

```
Model: "sequential_63"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_66 (SimpleRNN)   (None, 20, 2)             8

 attention_11 (attention)    (None, 2)                 22

 dense_76 (Dense)            (None, 1)                 3

=================================================================
Total params: 33 (132.00 Byte)
Trainable params: 33 (132.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Hidden_units =2
Dense_units = 1
Time_steps = 20
Attention_units=1

- Attention weight: 20 $\alpha_{t1}..\alpha_{t\_ts}$
  Shape: (time_steps, attention_units)
- Attention bias: 2
  Shape: (hidden_units, attention_units)

Many to One with Attention: Forward Propagation



$y_t = \tanh(Vh_t + b_{hy})$

$c = \alpha_t h_t + b_\alpha$

$$h_t = tanh(Ux_t + (Wh_{t-1} + b_{xh}))$$

RNN with Attention: Context Vector

timestep	Atribut 1	alpha_t	ht_neuron1	ht_neuron2
1	0.0000E+00	-0.6759	0.009	0.404
2	2.2664E-251	-0.6923	0.063	0.087
3	4.5327E-251	-0.7468	0.083	0.270
4	9.0654E-251	-0.7456	0.124	0.102
5	1.5865E-250	-0.8025	0.146	0.186
…	…	…	…	…
19	1.5330E-247	0.7771	0.341	-0.002
20	2.4805E-247	0.7768	0.344	-0.005

$c=\alpha_t h_t+b_\alpha$

$c_1=\alpha_t h_{t\_n1}+b_{\alpha\_n1}$

$c_2=\alpha_t h_{t\_n2}+b_{\alpha\_n2}$

c=[c1,c2]

Encoder Decoder without vs with Attention

Encoder:

$$h_t = f\left(x_t, h_{t-1}\right)$$

$$h_t = f\left(x_t, h_{t-1}\right)$$

Context Vector:

$$c = q\left(\{h_1, \cdots, h_{T_x}\}\right)$$
$$q\left(\{h_1, \cdots, h_T\}\right) = h_T$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j.$$

$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{T_x} \exp\left(e_{ik}\right)}$$

$$e_{ij} = a(s_{i-1}, h_j)$$

the process of “paying attention” can be learned during training

Decoder:

$$p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c)$$

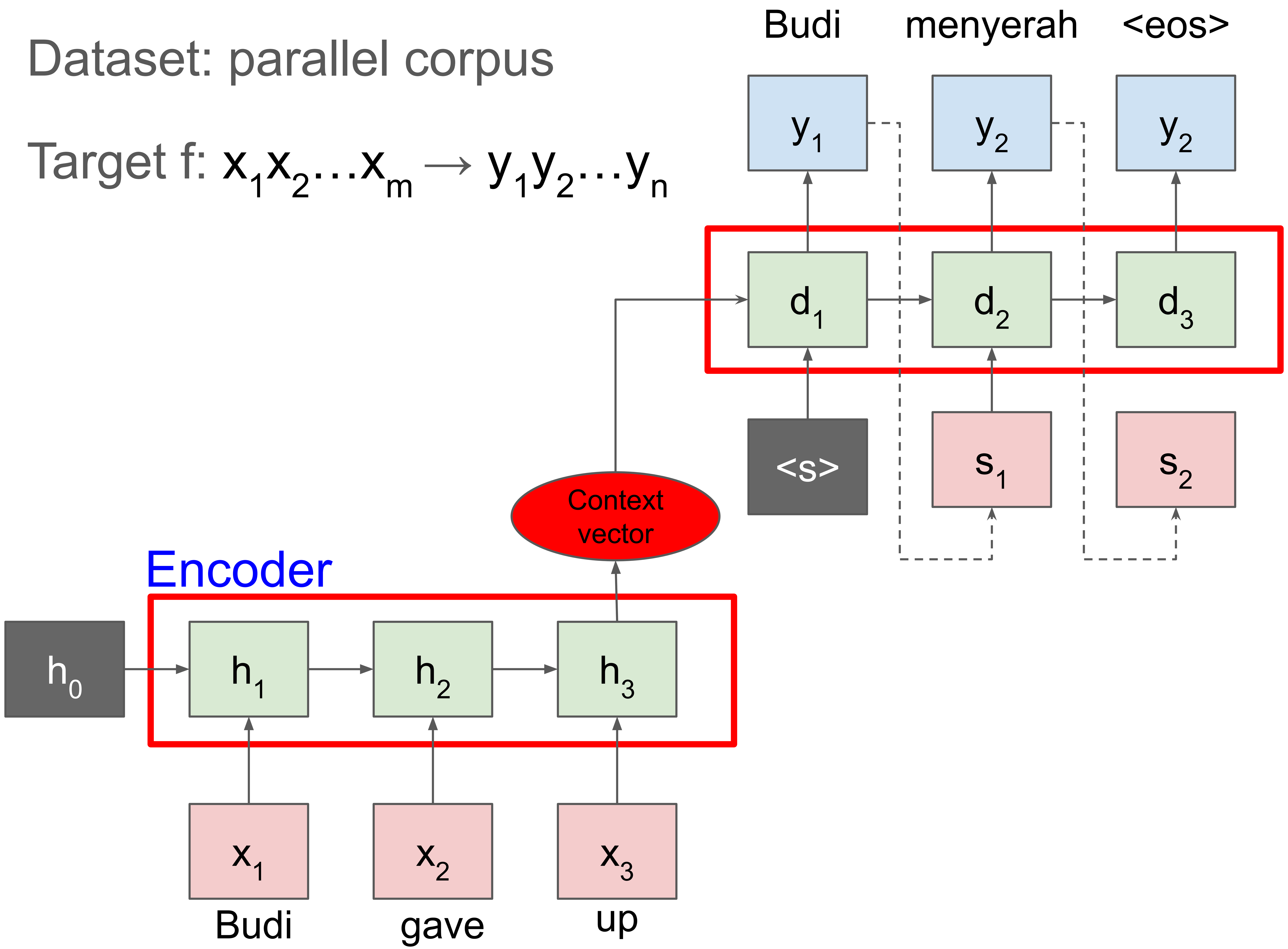
$$p(y_i|y_1, \ldots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i)$$
$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Encoder - Decoder with Attention

Dataset: parallel corpus

Target f: $x_1x_2\ldots x_m \rightarrow y_1y_2\ldots y_n$



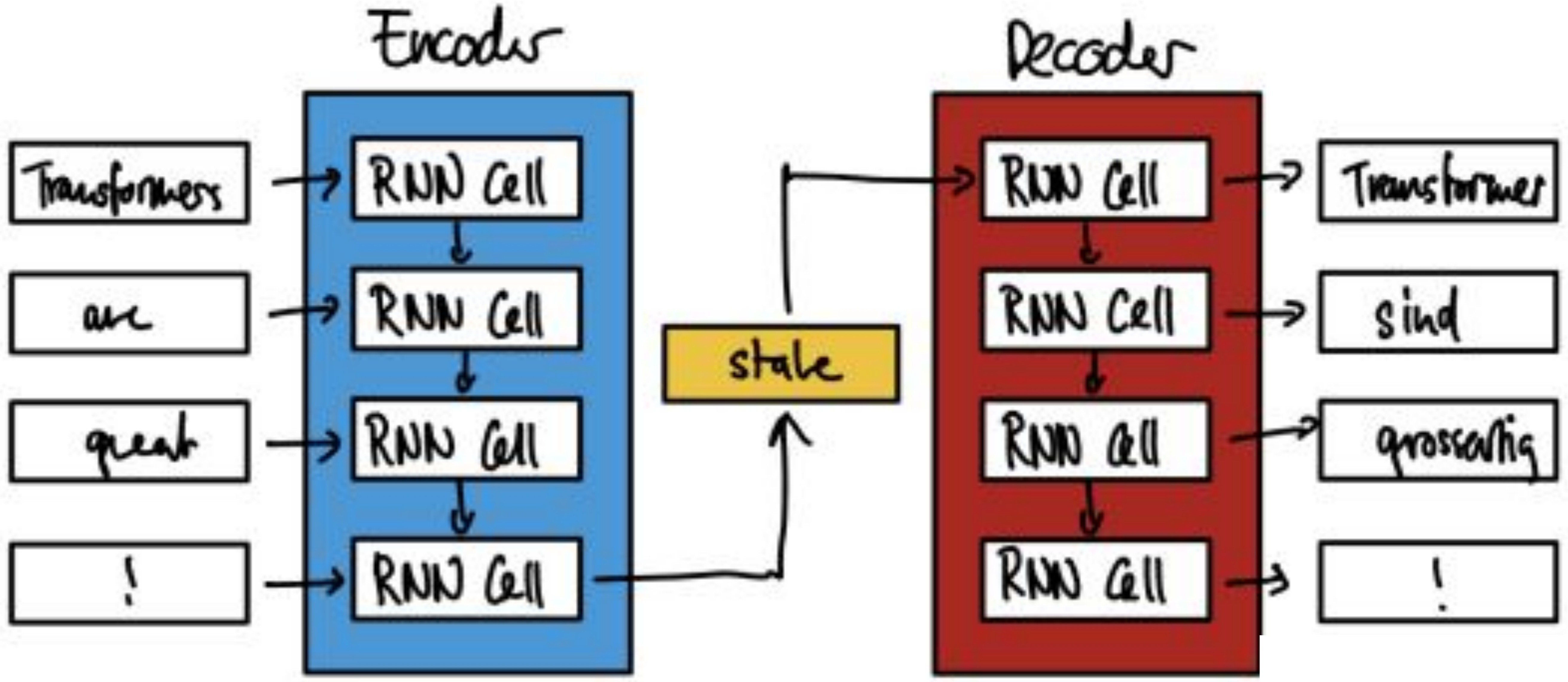
Decoder is trained to predict the next word $y_{t'}$ given the context vector c and all the previously predicted words $\{y_1,\ldots, y_{t'-1}\}$.

$$p(y_i|y_1,\ldots,y_{i-1},\mathbf{x}) = g(y_{i-1}, s_i, c_i)$$

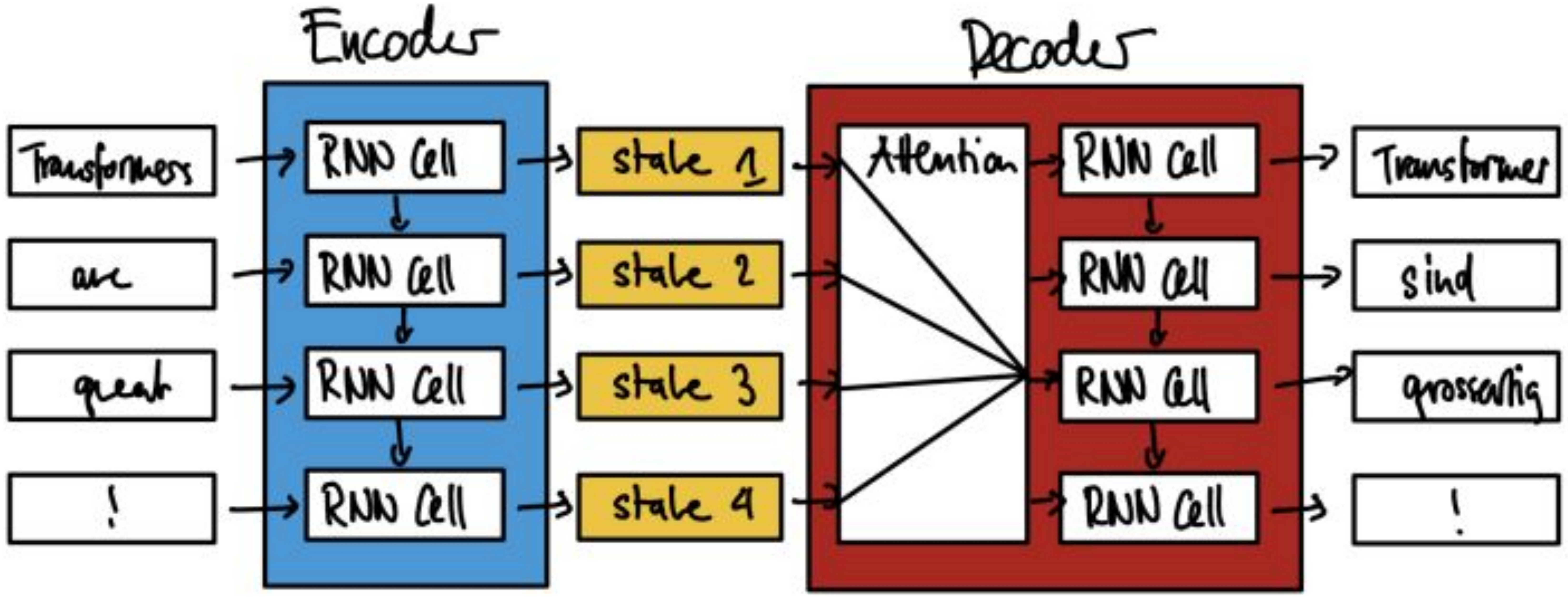
$$p(y_t \mid \{y_1,\cdots,y_{t-1}\}, c) = g(y_{t-1}, s_t, c)$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

Vanilla RNN Encoder - Decoder without vs with Attention

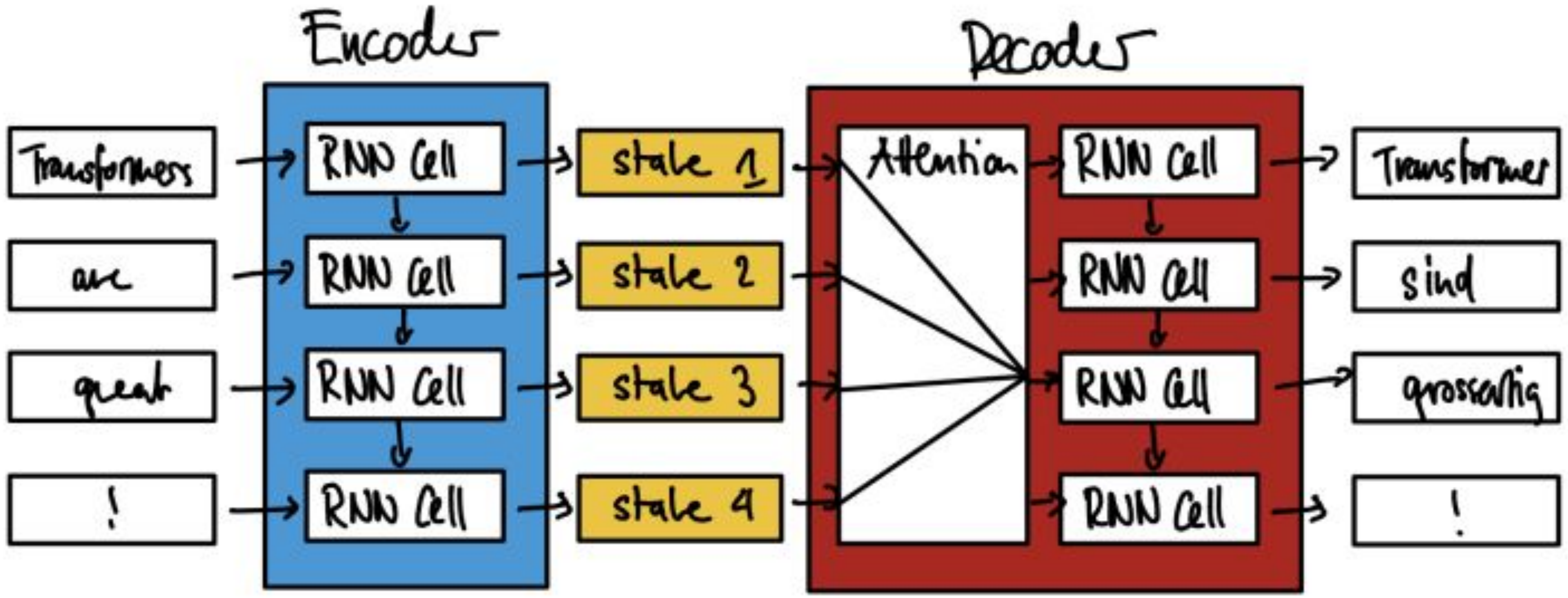


What is the difference between these architectures ?



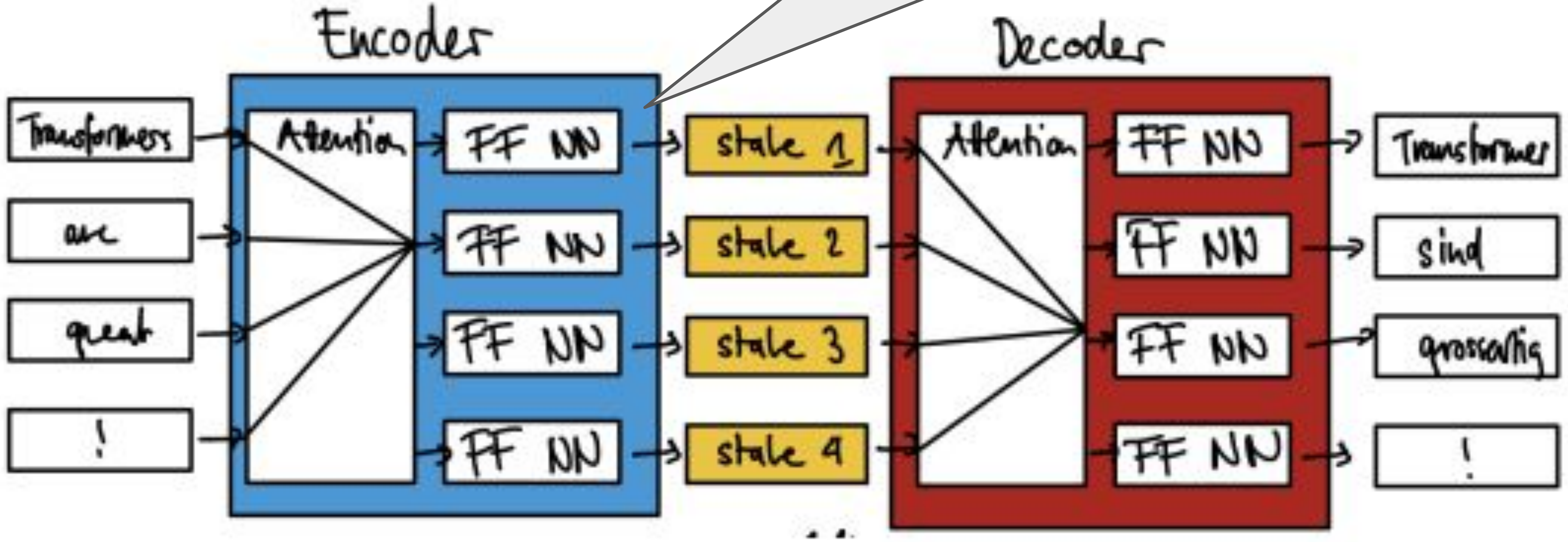
Tunstall, L., Von Werra, L., & Wolf, T. (2022). *Natural language processing with transformers*. " O'Reilly Media, Inc.".

Vanilla RNN Enc-Dec with Attention vs Transformers



The Transformer architecture replaced the recurrent units inside the encoder and decoder entirely with self-attention layers and simple feed-forward networks.

all the tokens are fed sequentially (Vanilla RNN Enc-Dec) vs in parallel through the model (transformers)



Tunstall, L., Von Werra, L., & Wolf, T. (2022). *Natural language processing with transformers*. " O'Reilly Media, Inc.".

Vanilla RNN Enc-Dec vs Transformers

- Moving from a sequential processing to a fully parallel processing unlocked strong computational efficiency gains allowing to train on orders of magnitude larger corpora for the same computational cost.
- At the same time, removing the sequential processing bottleneck of information makes the transformer architecture more efficient on several task that requires aggregating information over long time spans.
- The scaling laws of deep learning models: *larger models trained on more data in many cases yield better results*.
    - scaling models comes at the price of requiring large amounts of training data
- Transformer revolution started: transfer learning

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

Vaswani, A. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*, I.

… We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train …

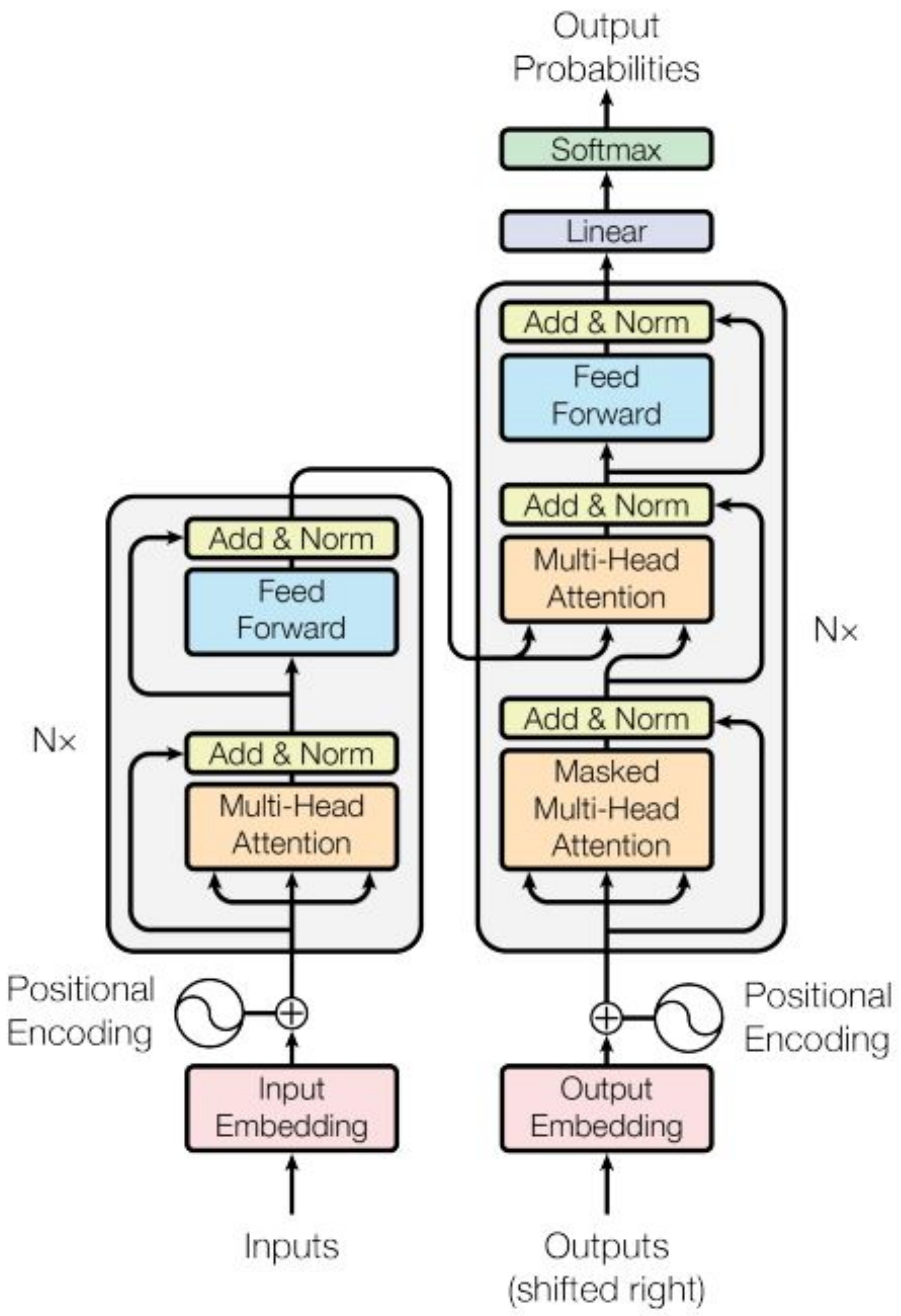
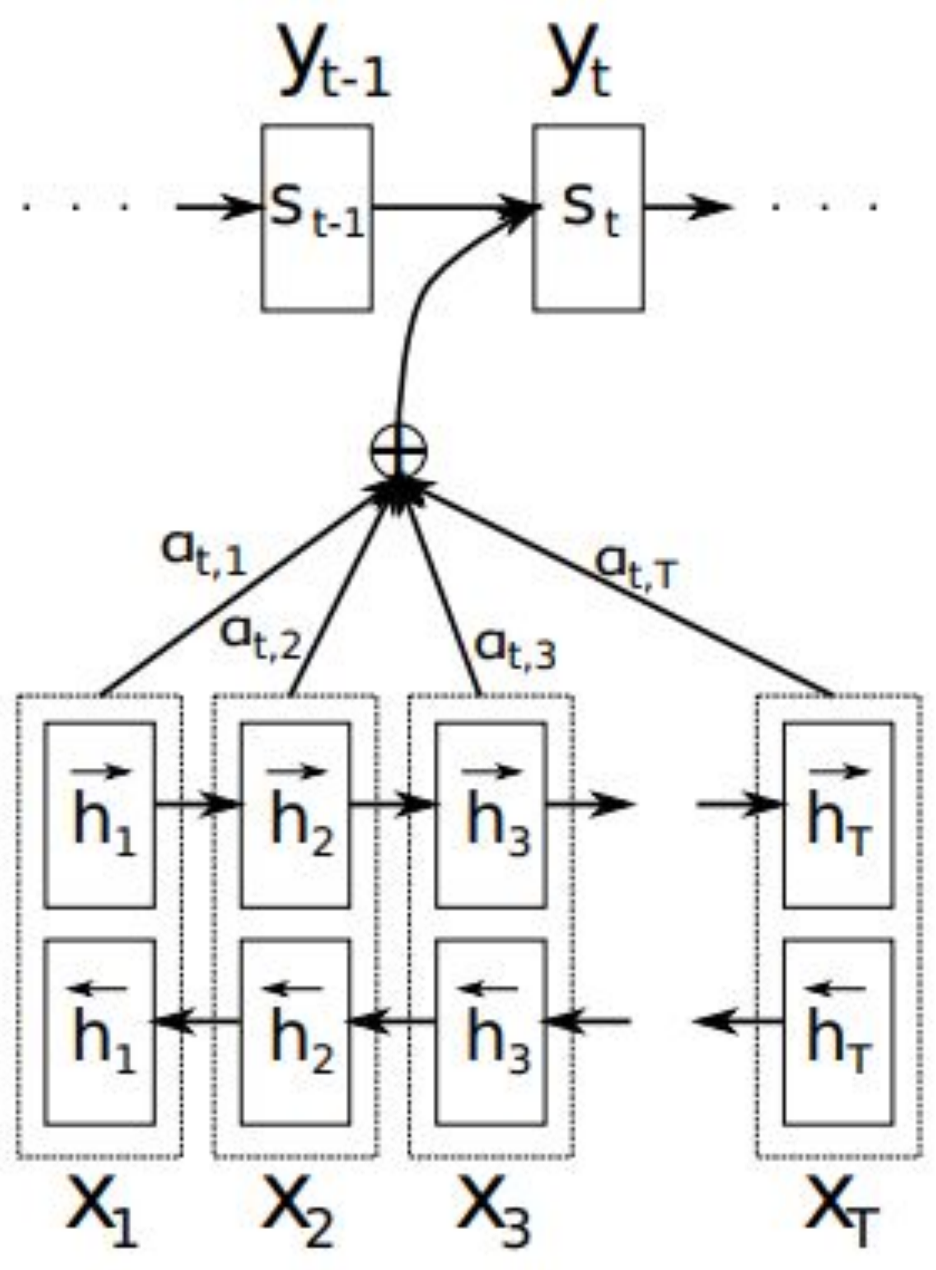


Figure 1: The Transformer - model architecture.

Scientific Breakthrough

Attention for Encoder-Decoder (Seq2Seq)



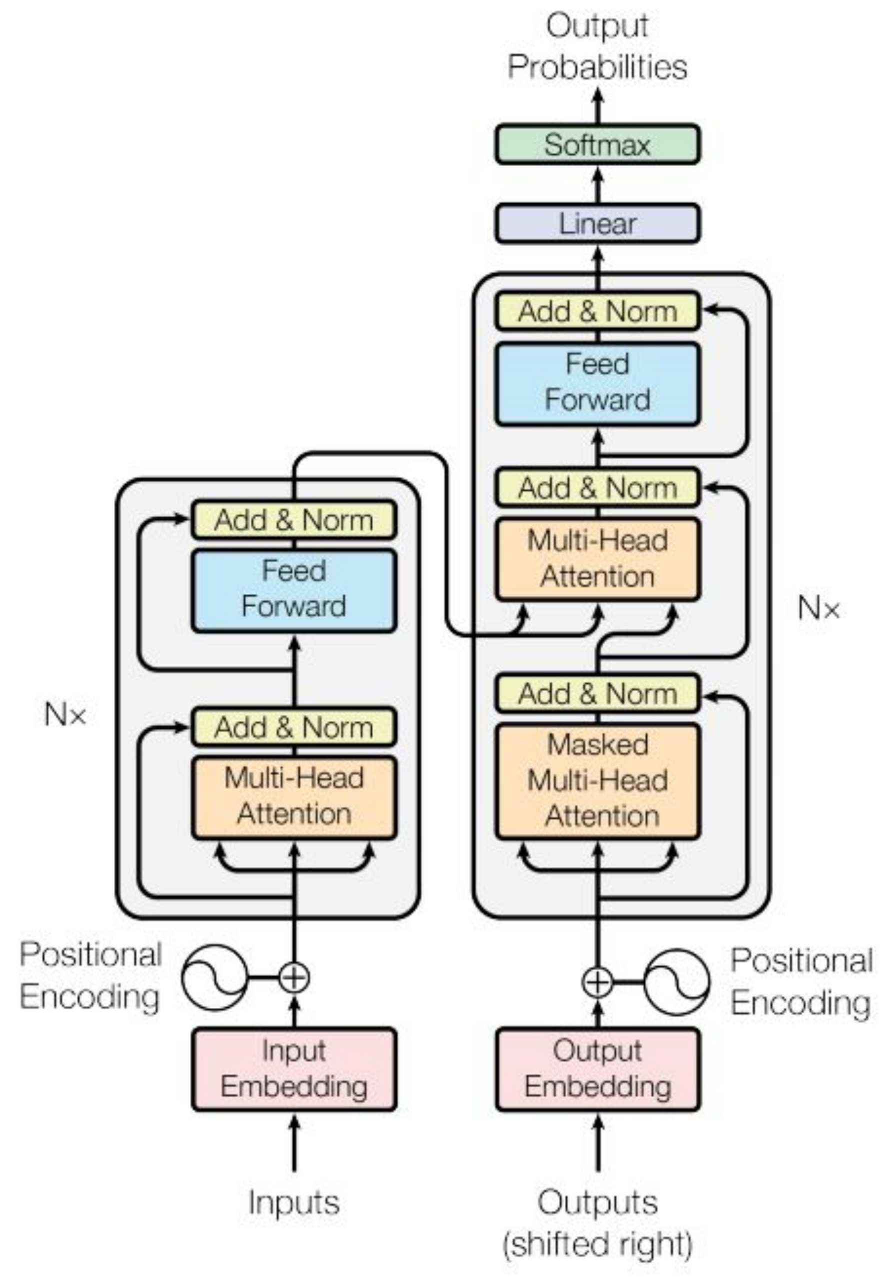
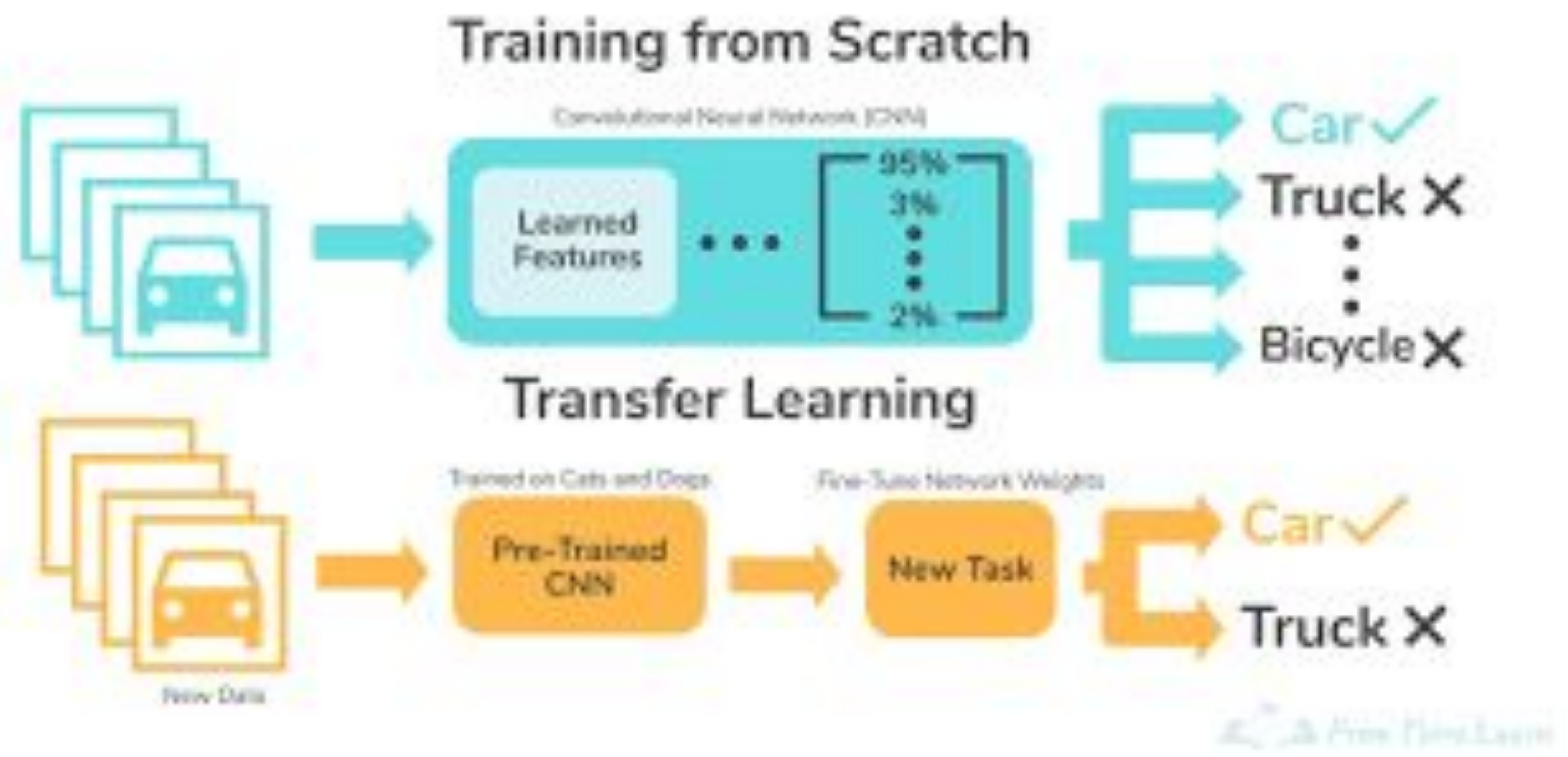


Figure 1: The Transformer - model architecture.

Transfer Learning



Sumber gambar: https://medium.com/@saba99/transfer-learning-bbf6b67deb88

Questions ?