Deep Learning for Vision & Language

Natural Language Processing I: RNNs and Transformers

RICE UNIVERSITY

Second Assignment

- Due Next Monday and third and final assignment to follow soon.

- Submit your project proposal – think about the amount of work it would take to a) Create an assignment 4, b) Solve assignment 4. Often in research and entrepreneurship asking a good question/finding the right problem is more important than giving a great answer/solution.

Recurrent Neural Networks

- These are models for handling sequences of things.

- Each input is not a vector but a sequence of input vectors.

- e.g. Each input can be a “word embedding” or any “word” representation – we will use in our first examples one-hot encoded tokens but in practice continuous dense word embeddings are used.

The Embedding Layer nn.Embedding

EMBEDDING

CLASS torch.nn.Embedding(*num_embeddings*, *embedding_dim*, *padding_idx=None*, *max_norm=None*, *norm_type=2.0*, *scale_grad_by_freq=False*, *sparse=False*, *_weight=None*, *device=None*, *dtype=None*) [SOURCE]

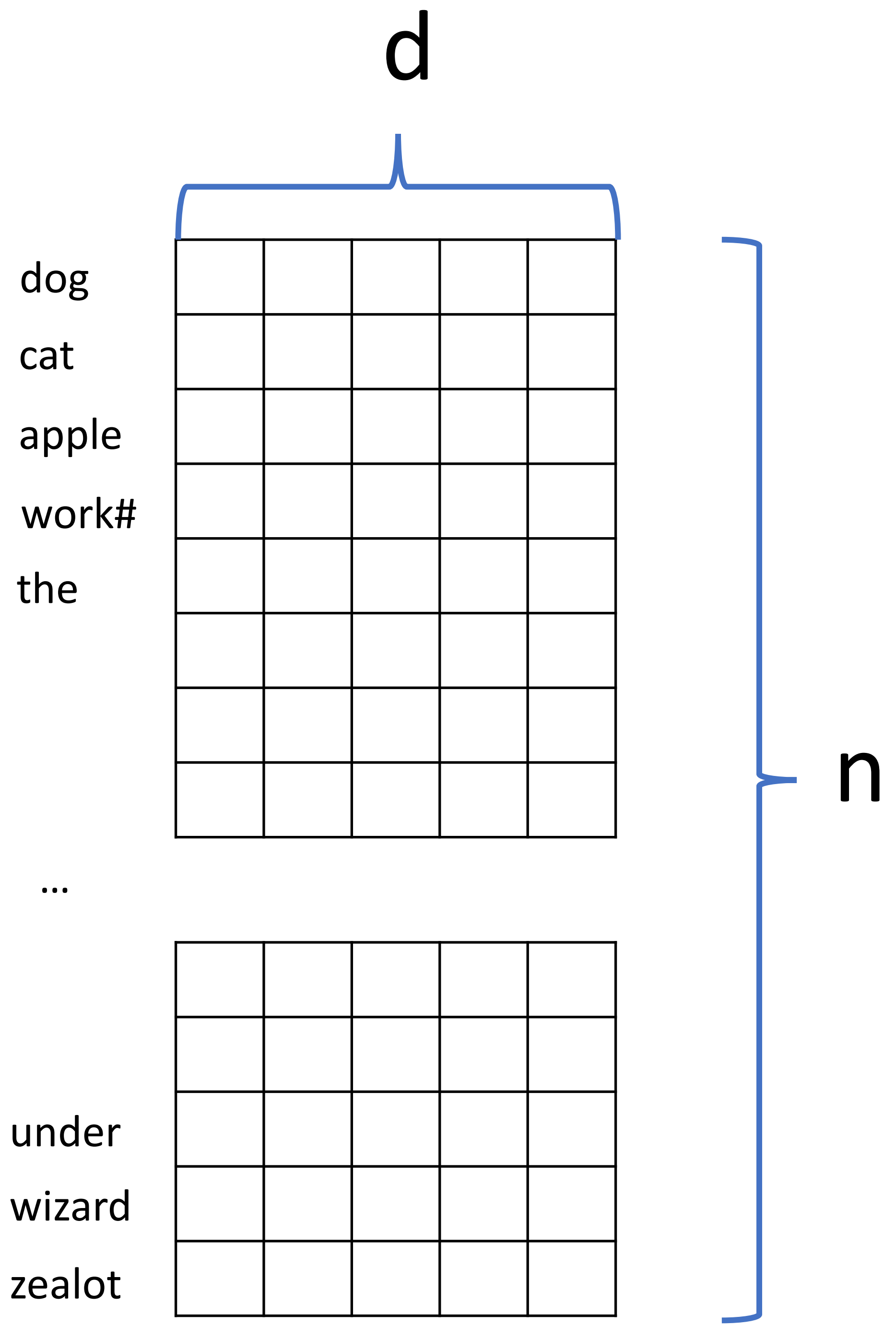
A simple lookup table that stores embeddings of a fixed dictionary and size.

This module is often used to store word embeddings and retrieve them using indices. The input to the module is a list of indices, and the output is the corresponding word embeddings.

Parameters:

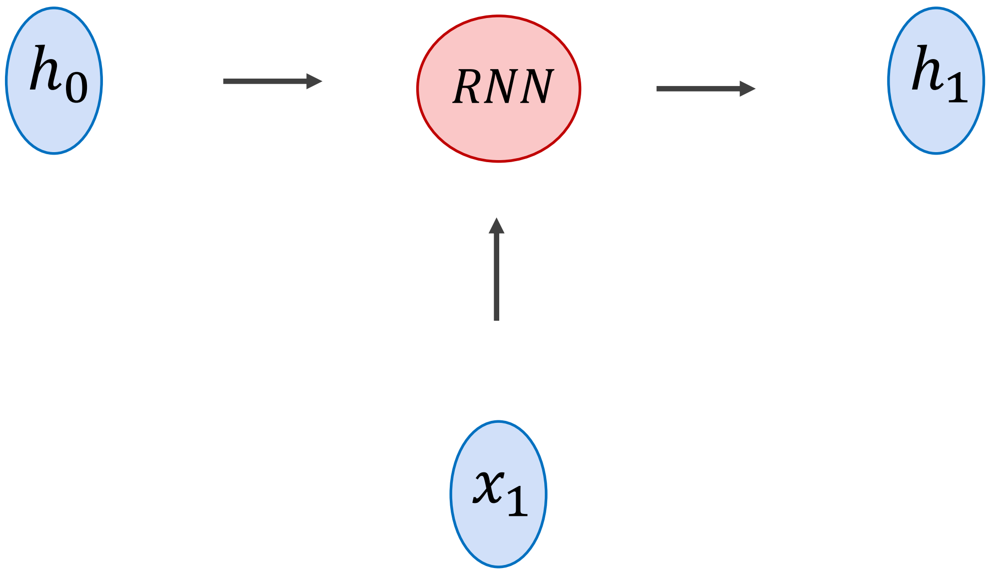
- **num_embeddings** (*int*) – size of the dictionary of embeddings
- **embedding_dim** (*int*) – the size of each embedding vector
- **padding_idx** (*int*, *optional*) – If specified, the entries at `padding_idx` do not contribute to the gradient; therefore, the embedding vector at `padding_idx` is not updated during training, i.e. it remains as a fixed “pad”. For a newly constructed Embedding, the embedding vector at `padding_idx` will default to all zeros, but can be updated to another value to be used as the padding vector.

The Embedding Layer
nn.Embedding



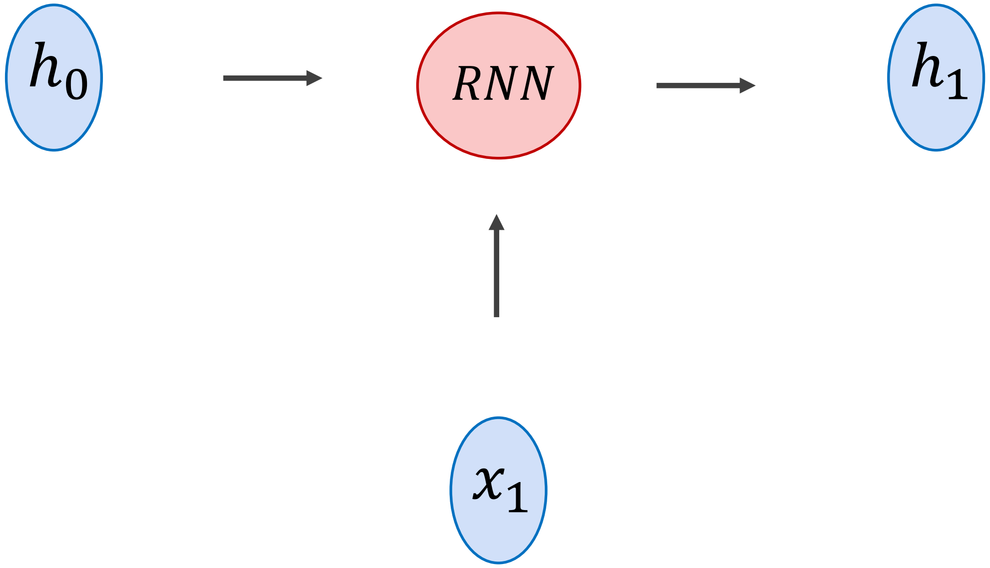
nn.Embedding(n, d)

Recurrent Neural Network Cell

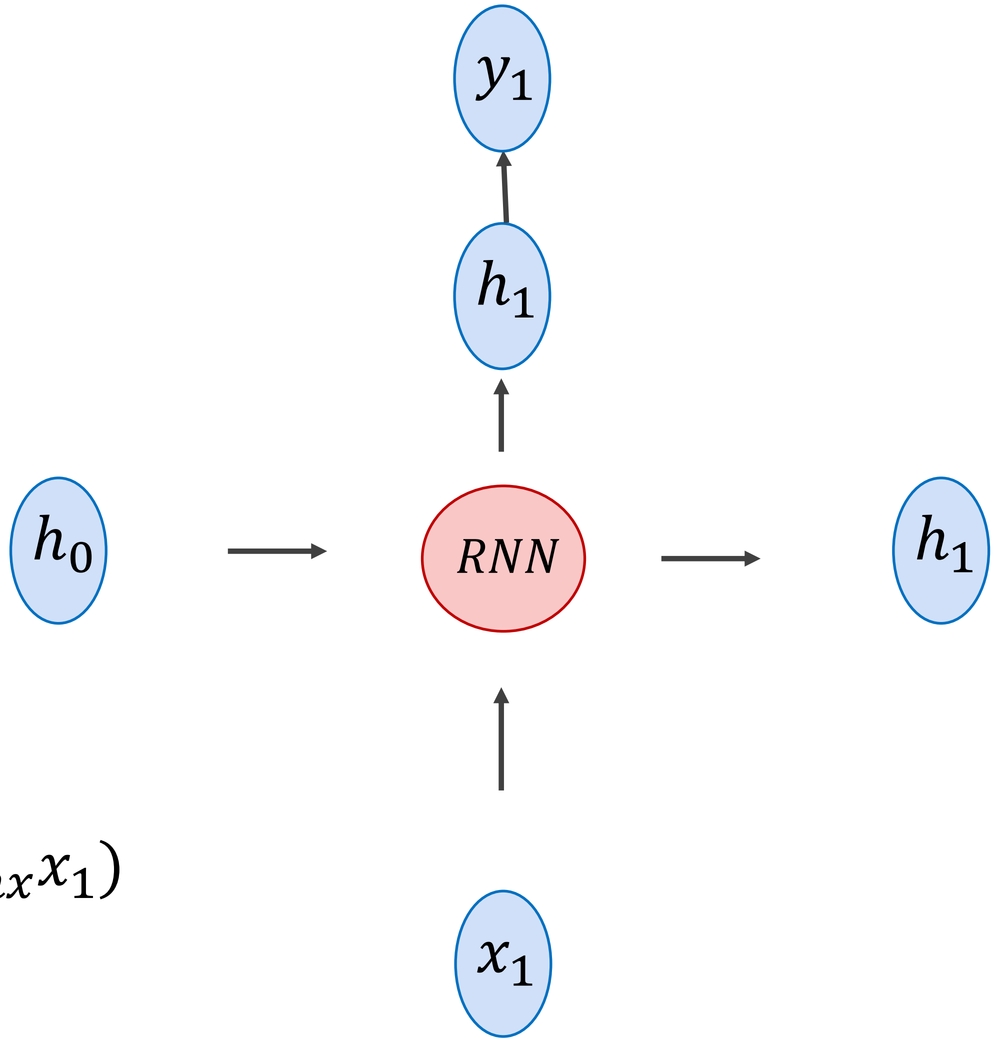


Recurrent Neural Network Cell

$$h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)$$



Recurrent Neural Network Cell



$$h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)$$

$$y_1 = \mathrm{softmax}(W_{hy}h_1)$$

Recurrent Neural Network Cell

$$y_1 = [0.1, 0.05, 0.05, 0.1, 0.7]$$

$$h_1 = [0.1 \quad 0.2 \; 0 \; -0.3 \; -0.1\,]$$

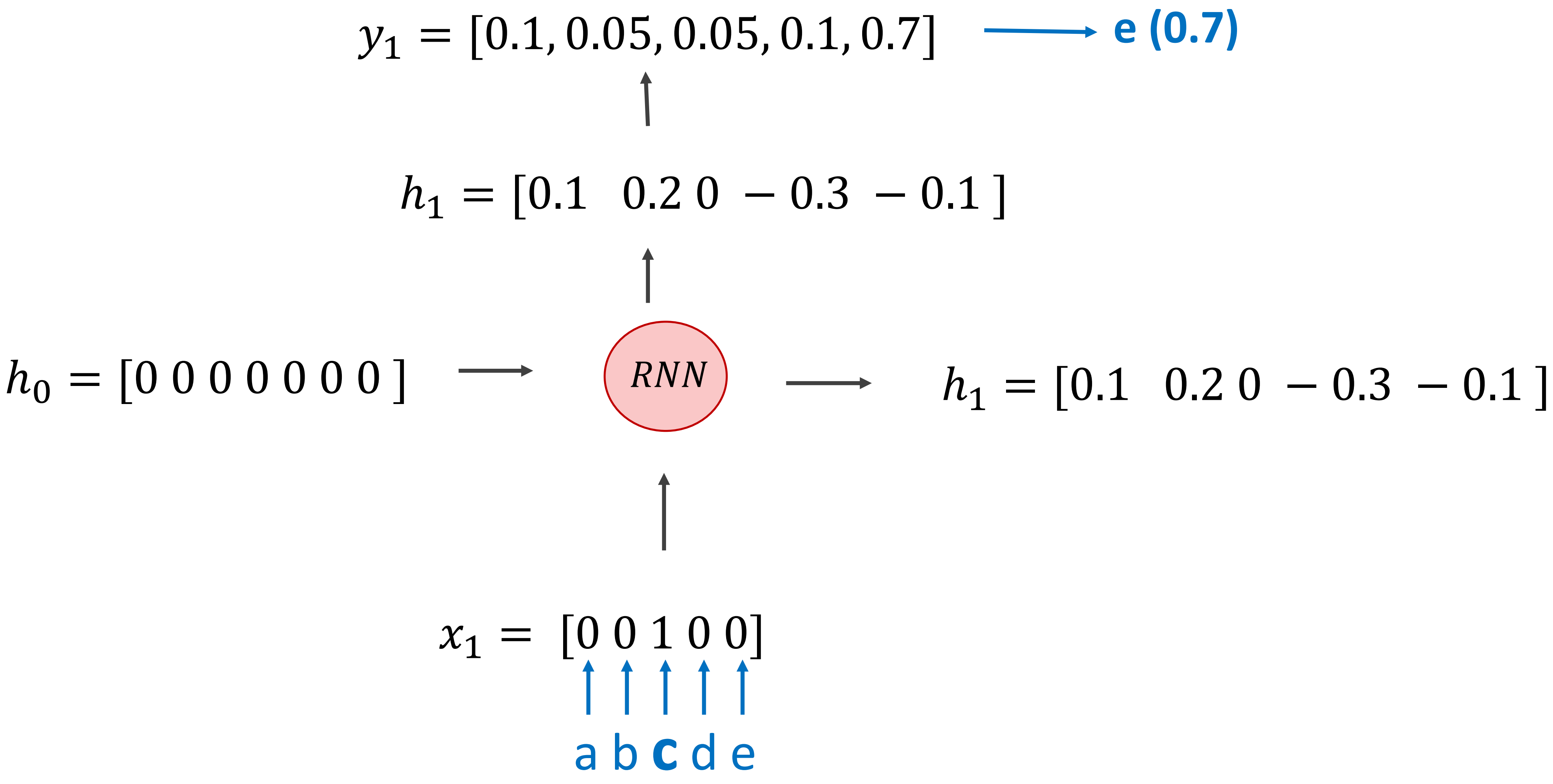
$$h_0 = [0\;0\;0\;0\;0\;0\;0\,] \longrightarrow RNN \longrightarrow h_1 = [0.1 \quad 0.2 \; 0 \; -0.3 \; -0.1\,]$$

$$x_1 = \; [0\;0\;1\;0\;0]$$

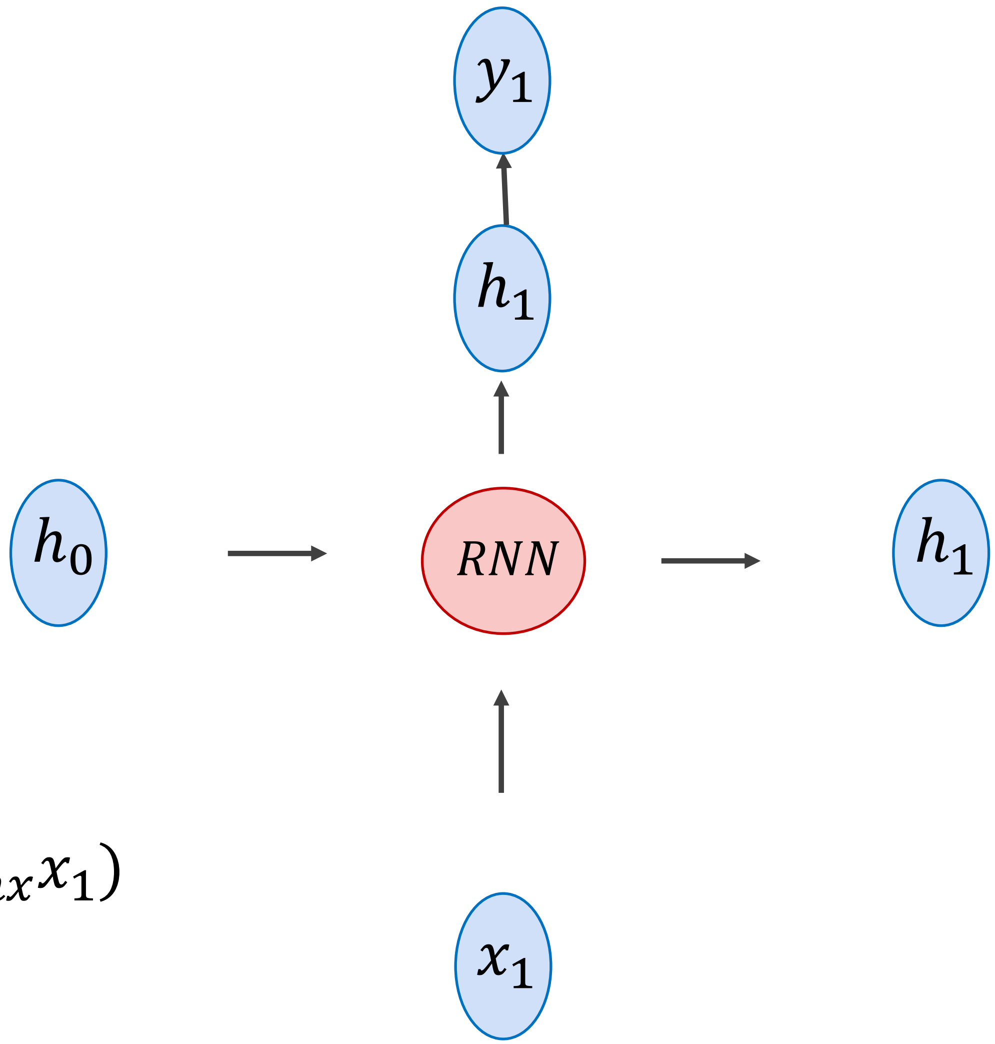
$$h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)$$

$$y_1 = \mathrm{softmax}(W_{hy}h_1)$$

Recurrent Neural Network Cell



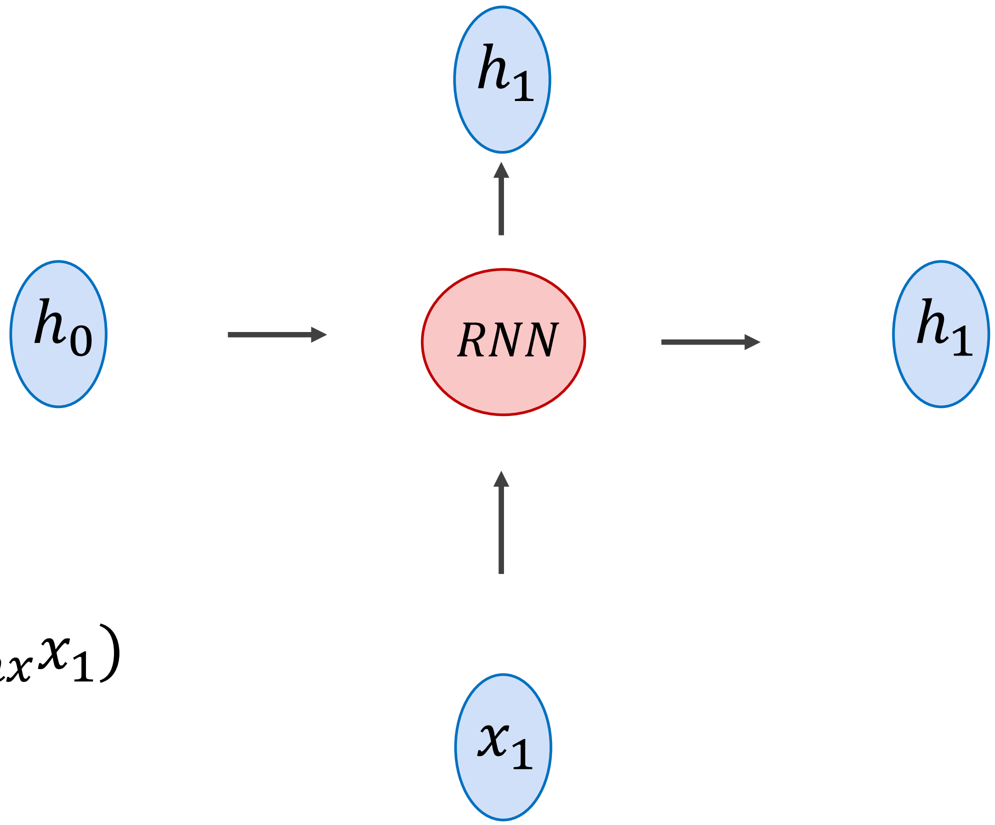
Recurrent Neural Network Cell



$h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)$

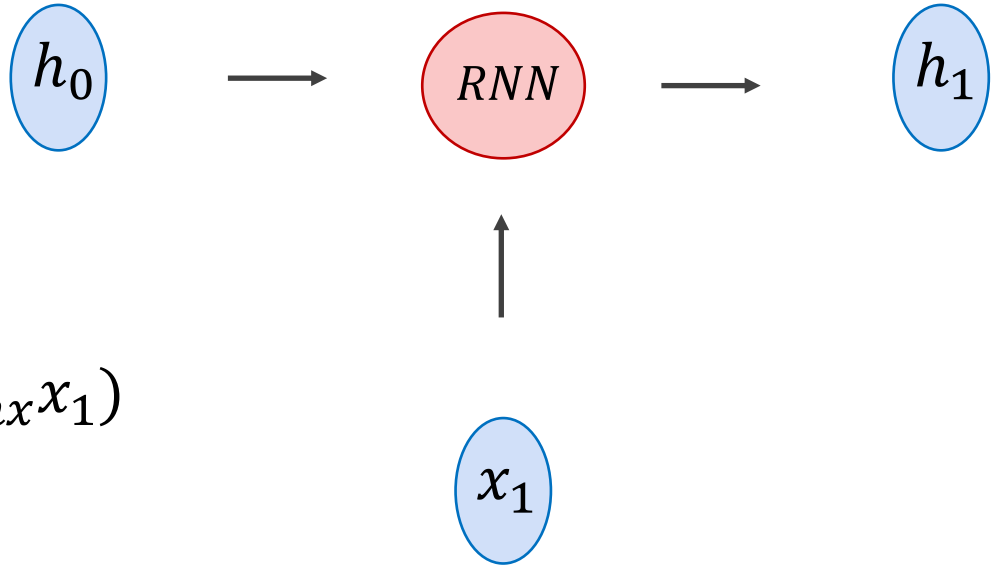
$y_1 = \text{softmax}(W_{hy}h_1)$

Recurrent Neural Network Cell



$$h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)$$

Recurrent Neural Network Cell



$$h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)$$

RNN

CLASS torch.nn.RNN(*self*, *input_size*, *hidden_size*, *num_layers=1*, *nonlinearity='tanh'*, *bias=True*, *batch_first=False*, *dropout=0.0*, *bidirectional=False*, *device=None*, *dtype=None*) [SOURCE]

Apply a multi-layer Elman RNN with $\tanh$ or ReLU non-linearity to an input sequence. For each element in the input sequence, each layer computes the following function:

$$h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$

where h_t is the hidden state at time *t*, x_t is the input at time *t*, and $h_{(t-1)}$ is the hidden state of the previous layer at time *t-1* or the initial hidden state at time *0*. If `nonlinearity` is `'relu'`, then ReLU is used instead of $\tanh$.

Parameters

- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two RNNs together to form a *stacked RNN*, with the second RNN taking in outputs of the first RNN and computing the final results. Default: 1
- **nonlinearity** – The non-linearity to use. Can be either `'tanh'` or `'relu'`. Default: `'tanh'`
- **bias** – If `False`, then the layer does not use bias weights *b_ih* and *b_hh*. Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as (*batch, seq, feature*) instead of (*seq, batch, feature*). Note that this does not apply to hidden or cell states. See the Inputs/Outputs sections below for details. Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each RNN layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional RNN. Default: `False`

Inputs: input, h_0

- **input**: tensor of shape (L, H_{in}) for unbatched input, (L, N, H_{in}) when `batch_first=False` or (N, L, H_{in}) when `batch_first=True` containing the features of the input sequence. The input can also be a packed variable length sequence. See `torch.nn.utils.rnn.pack_padded_sequence()` or `torch.nn.utils.rnn.pack_sequence()` for details.
- **h_0**: tensor of shape $(D * \text{num\_layers}, H_{out})$ for unbatched input or $(D * \text{num\_layers}, N, H_{out})$ containing the initial hidden state for the input sequence batch. Defaults to zeros if not provided.

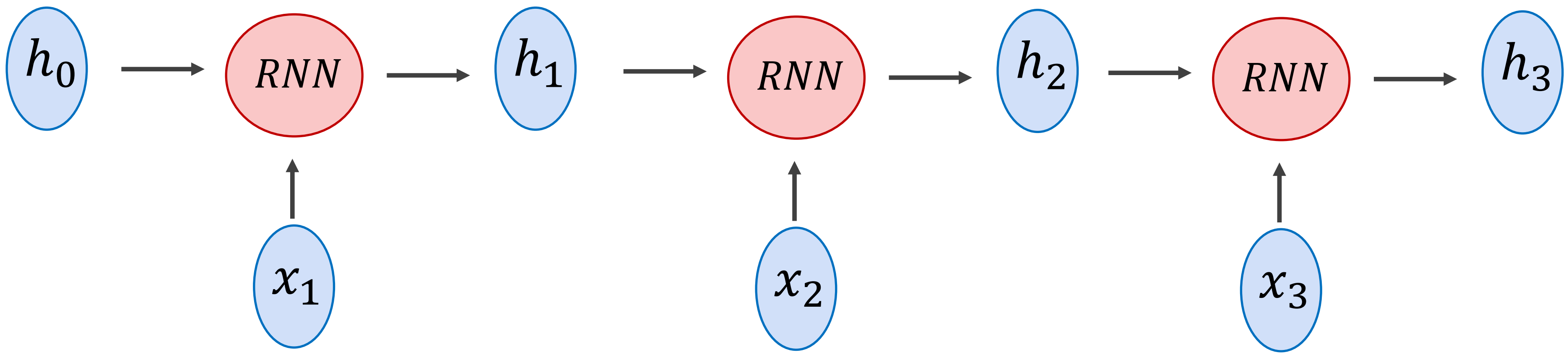
where:

$$\begin{aligned} N &= \text{batch size} \\ L &= \text{sequence length} \\ D &= 2 \text{ if bidirectional=True otherwise } 1 \\ H_{in} &= \text{input\_size} \\ H_{out} &= \text{hidden\_size} \end{aligned}$$

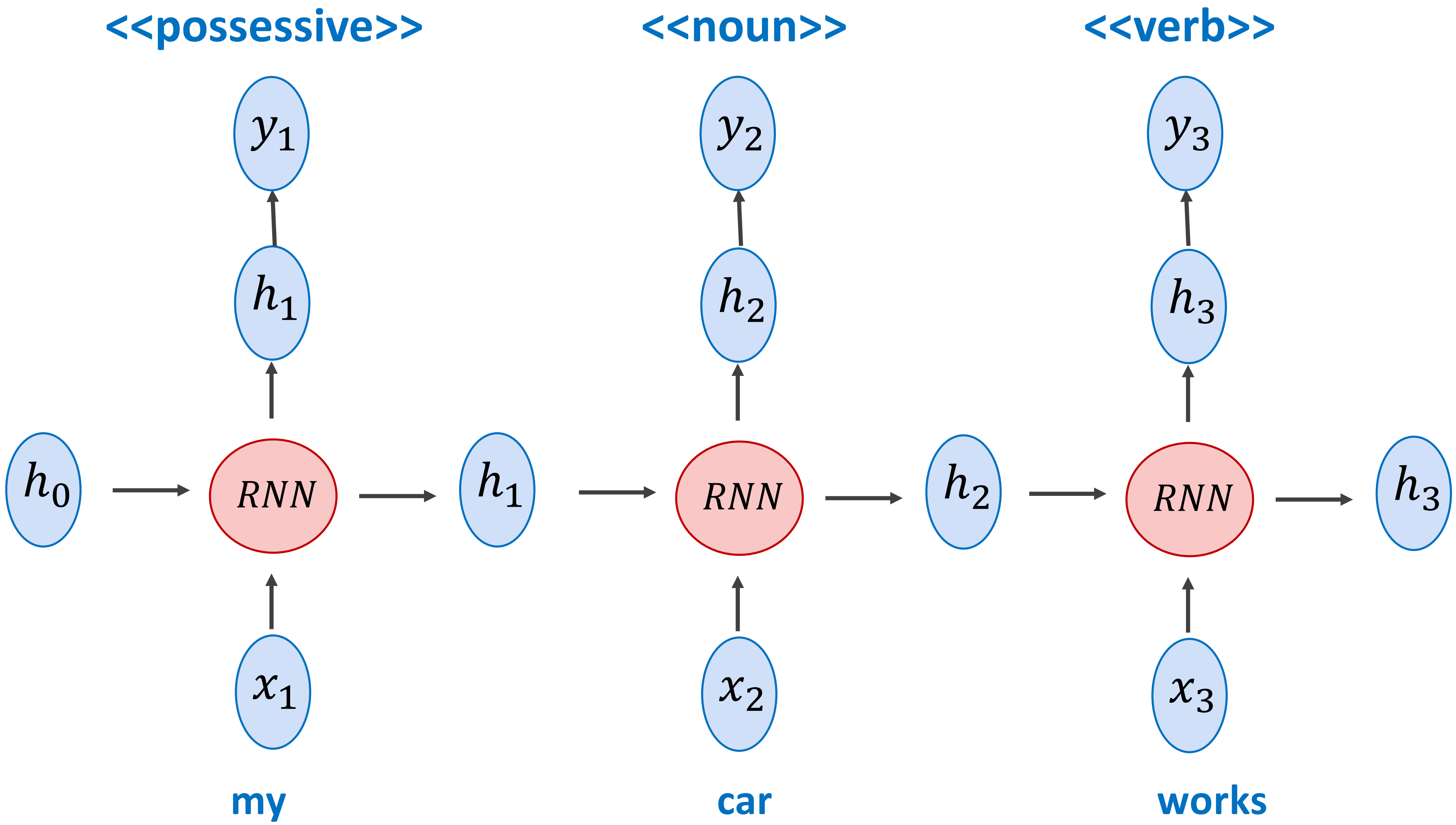
Outputs: output, h_n

- **output**: tensor of shape $(L, D * H_{out})$ for unbatched input, $(L, N, D * H_{out})$ when `batch_first=False` or $(N, L, D * H_{out})$ when `batch_first=True` containing the output features *(h_t)* from the last layer of the RNN, for each *t*. If a `torch.nn.utils.rnn.PackedSequence` has been given as the input, the output will also be a packed sequence.
- **h_n**: tensor of shape $(D * \text{num\_layers}, H_{out})$ for unbatched input or $(D * \text{num\_layers}, N, H_{out})$ containing the final hidden state for each element in the batch.

(Unrolled) Recurrent Neural Network



How can it be used? – e.g. Tagging a Text Sequence
One-to-one Sequence Mapping Problems



How can it be used? – e.g. Tagging a Text Sequence
One-to-one Sequence Mapping Problems

Training examples don’t need to be the same length!

input	**output**
my car works	**<<possessive>> <<noun>> <<verb>>**
my dog ate the assignment	**<<possessive>> <<noun>> <<verb>> <<pronoun>> <<noun>>**
my mother saved the day	**<<possessive>> <<noun>> <<verb>> <<pronoun>> <<noun>>**
the smart kid solved the problem	**<<pronoun>> <<qualifier>> <<noun>> <<verb>> <<pronoun>> <<noun>>**

How can it be used? – e.g. Tagging a Text Sequence
One-to-one Sequence Mapping Problems

Training examples don't need to be the same length!

input	output
L(my car works) = 3	**L (<<possessive>> <<noun>> <<verb>>) = 3**
L(my dog ate the assignment) = 5	**L (<<possessive>> <<noun>> <<verb>> <<pronoun>> <<noun>>) = 5**
L(my mother saved the day) = 5	**L (<<possessive>> <<noun>> <<verb>> <<pronoun>> <<noun>>) = 5**
L(the smart kid solved the problem) = 6	**L (<<pronoun>> <<qualifier>> <<noun>> <<verb>> <<pronoun>> <<noun>>) = 6**

How can it be used? – e.g. Tagging a Text Sequence
One-to-one Sequence Mapping Problems

Training examples don't need to be the same length!

If we assume a vocabulary of a 1000 possible words and 20 possible output tags

input	output
T: 1000 x 3	**T: 20 x 3**
T: 1000 x 5	**T: 20 x 5**
T: 1000 x 5	**T: 20 x 5**
T: 1000 x 6	**T: 20 x 6**

How can it be used? – e.g. Tagging a Text Sequence
One-to-one Sequence Mapping Problems

Training examples don't need to be the same length!

If we assume a vocabulary of a 1000 possible words and 20 possible output tags

input	output
T: 1000 x 3	T: 20 x 3
T: 1000 x 5	T: 20 x 5
T: 1000 x 5	T: 20 x 5
T: 1000 x 6	T: 20 x 6

How do we create batches if inputs and outputs have different shapes?

How can it be used? – e.g. Tagging a Text Sequence
One-to-one Sequence Mapping Problems

Training examples don't need to be the same length!

If we assume a vocabulary of a 1000 possible words and 20 possible output tags

input	output
T: 1000 x 3	T: 20 x 3
T: 1000 x 5	T: 20 x 5
T: 1000 x 5	T: 20 x 5
T: 1000 x 6	T: 20 x 6

How do we create batches if inputs and outputs have different shapes?

Solution 1: Forget about batches, just process things one by one.

How can it be used? – e.g. Tagging a Text Sequence
One-to-one Sequence Mapping Problems

Training examples don't need to be the same length!

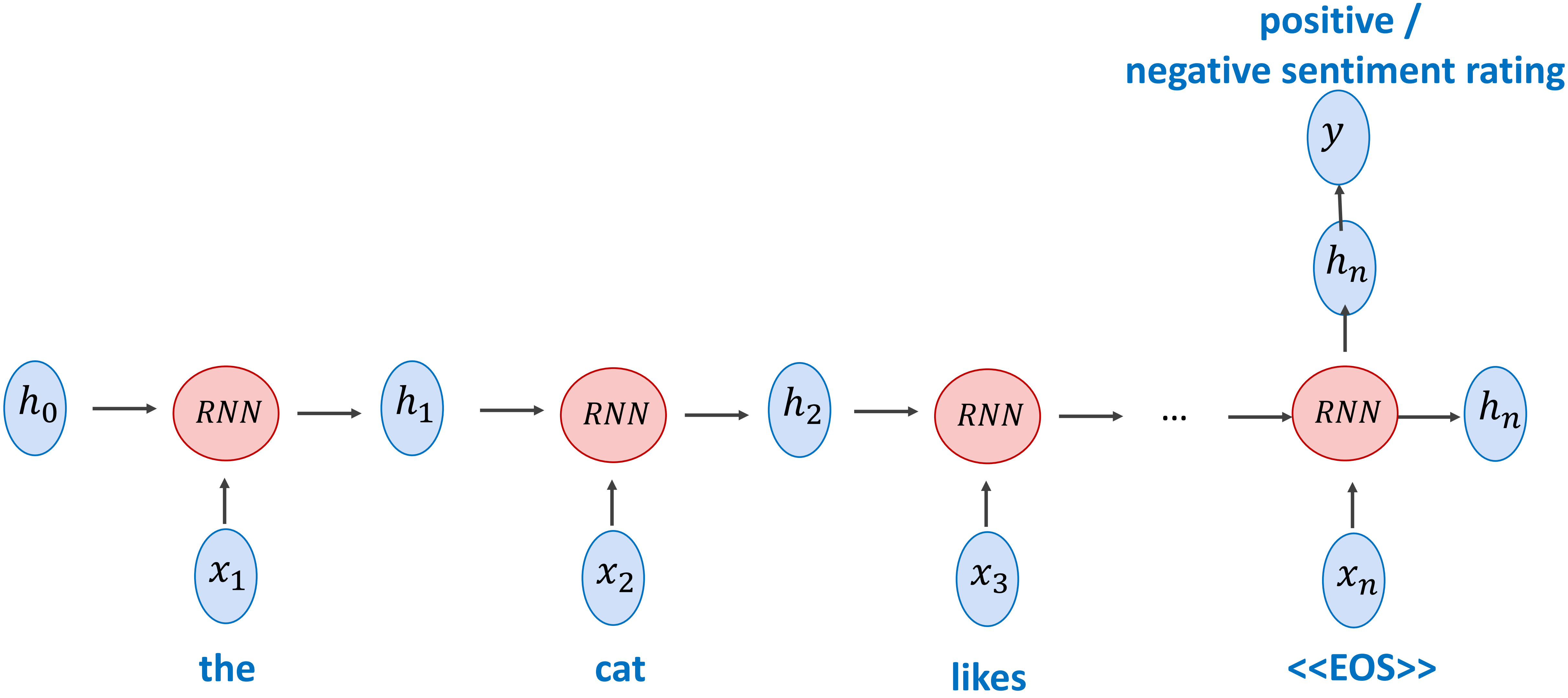
If we assume a vocabulary of a 1000 possible words and 20 possible output tags

input	output
T: 1000 x 3	**T: 20 x 3**
T: 1000 x 5	**T: 20 x 5**
T: 1000 x 5	**T: 20 x 5**
T: 1000 x 6	**T: 20 x 6**

How do we create batches if inputs and outputs have different shapes?

Solution 2: Zero padding. We can put the above vectors in **T: 4 x 1000 x 6**

How can it be used? – e.g. Scoring the Sentiment of a Text Sequence
Many-to-one Sequence to score problems



How can it be used? – e.g. Sentiment Scoring
Many to one Mapping Problems

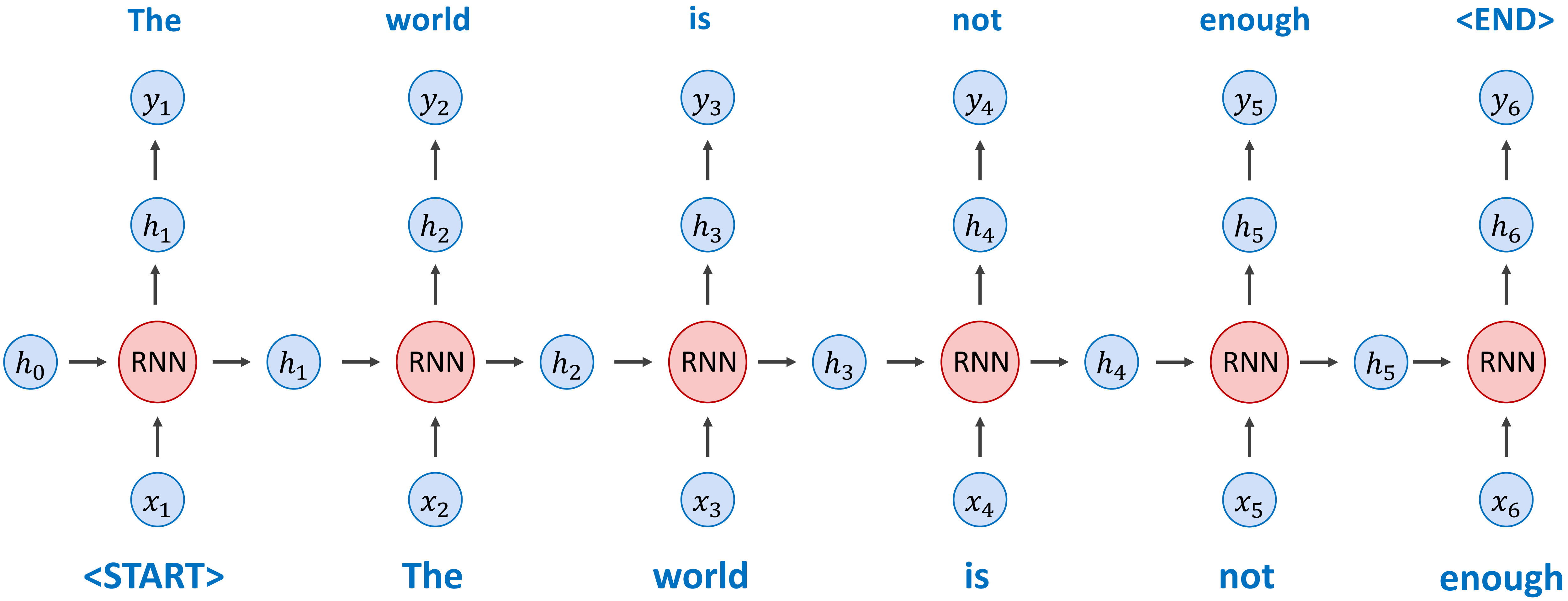
Input training examples don't need to be the same length!
In this case outputs can be.

input	output
this restaurant has good food	Positive
this restaurant is bad	Negative
this restaurant is the worst	Negative
this restaurant is well recommended	Positive

How can it be used? – e.g. Text Generation

Auto-regressive model – Sequence to Sequence during Training, Auto-regressive during test

DURING TRAINING



How can it be used? – e.g. Text Generation
Auto-regressive Models

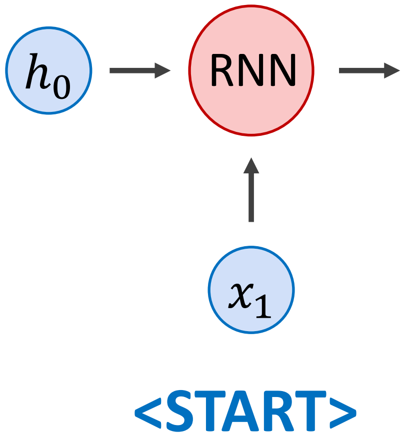
Input training examples don't need to be the same length!
In this case outputs can be.

input	output
<START> this restaurant has good food	this restaurant has good food <END>
<START> this restaurant is bad	this restaurant is bad <END>
<START> this restaurant is the worst	this restaurant is the worst <END>
<START> this restaurant is well recommended	this restaurant is well recommended <END>

How can it be used? – e.g. Text Generation

Auto-regressive model – Sequence to Sequence during Training, Auto-regressive during test

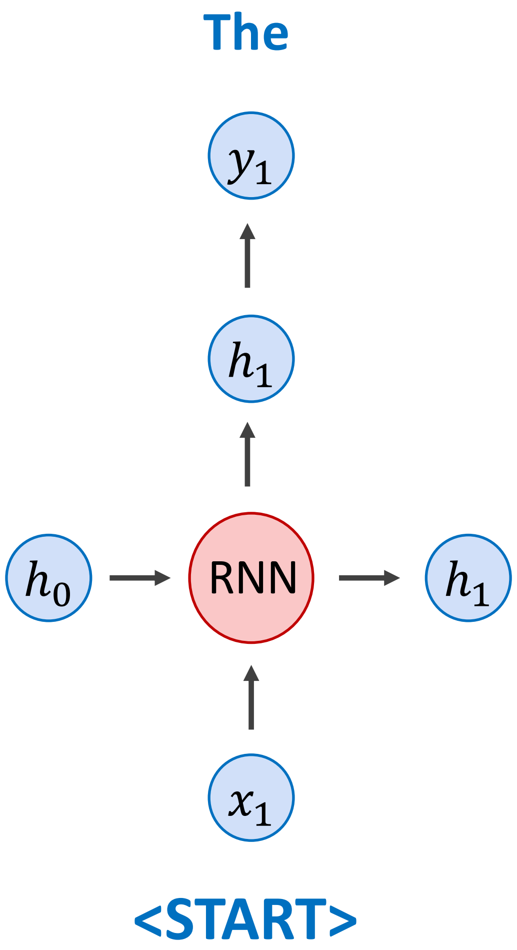
DURING TESTING



How can it be used? – e.g. Text Generation

Auto-regressive model – Sequence to Sequence during Training, Auto-regressive during test

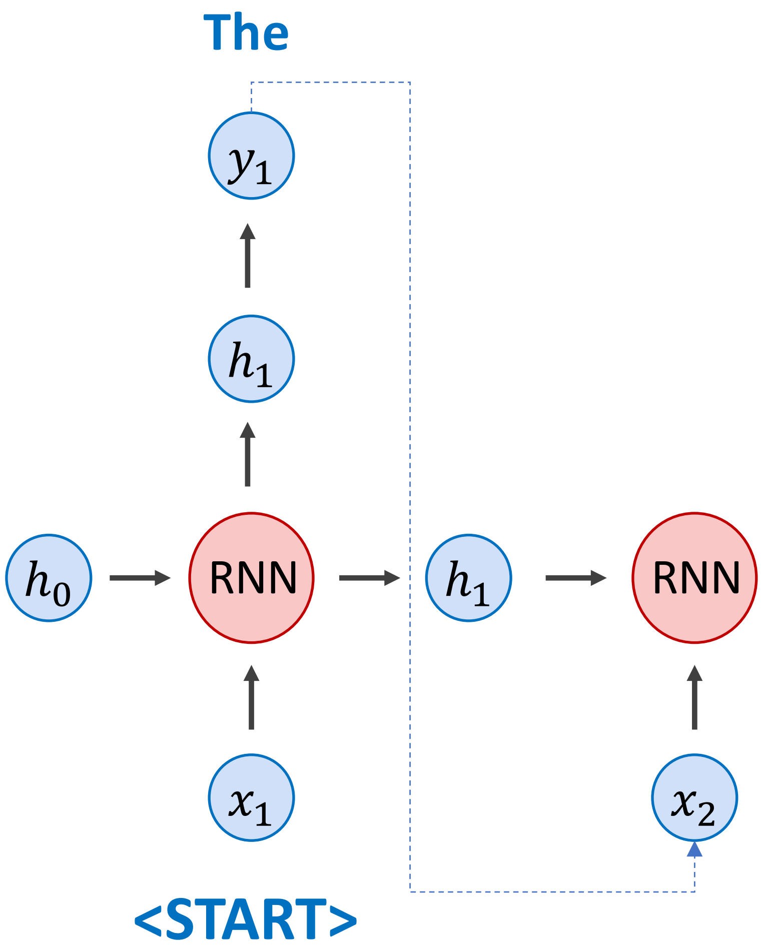
DURING TESTING



How can it be used? – e.g. Text Generation

Auto-regressive model – Sequence to Sequence during Training, Auto-regressive during test

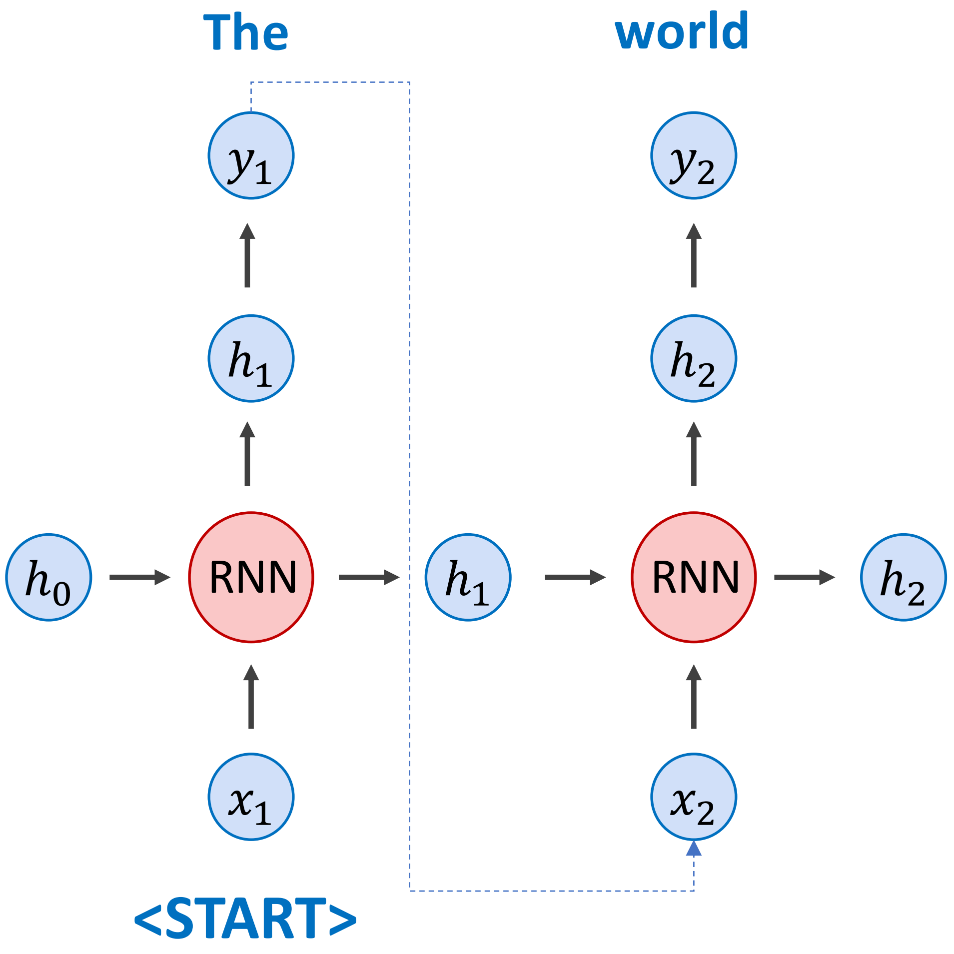
DURING TESTING



How can it be used? – e.g. Text Generation

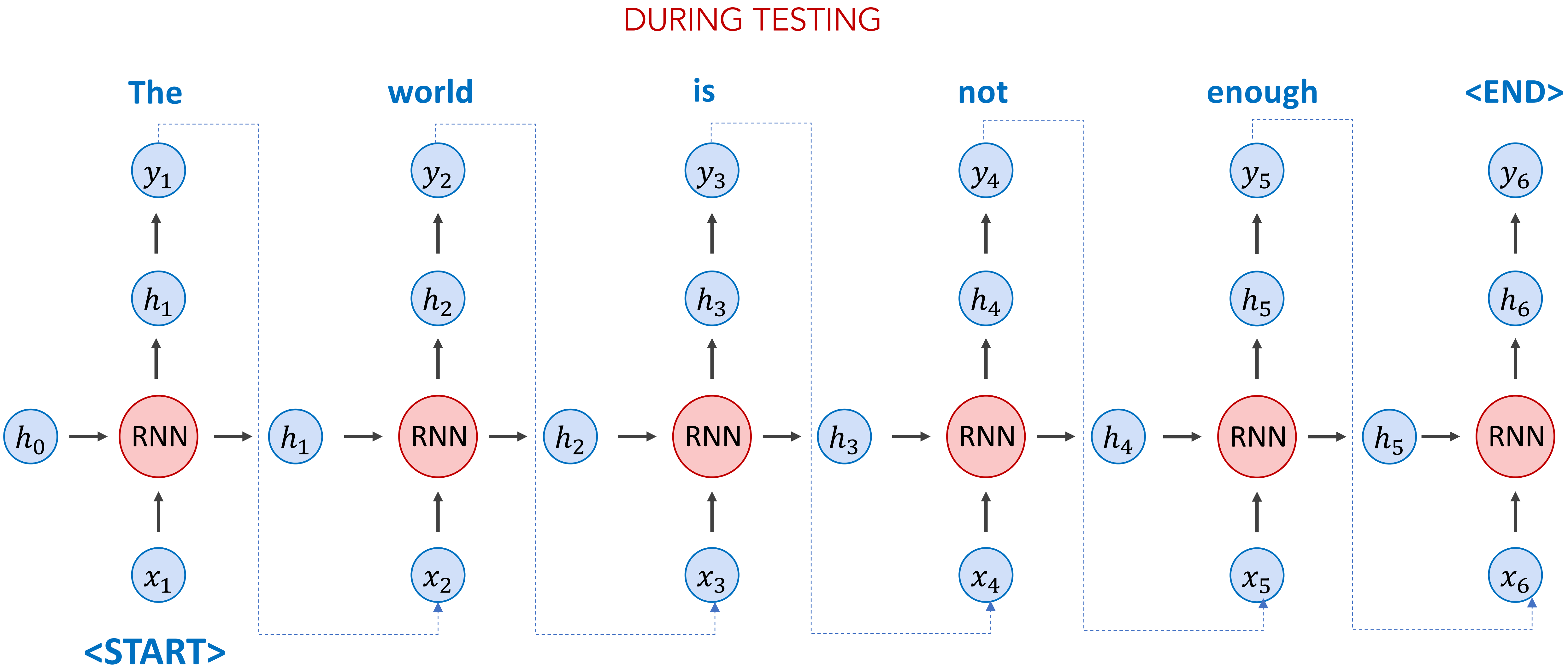
Auto-regressive model – Sequence to Sequence during Training, Auto-regressive during test

DURING TESTING



How can it be used? – e.g. Text Generation

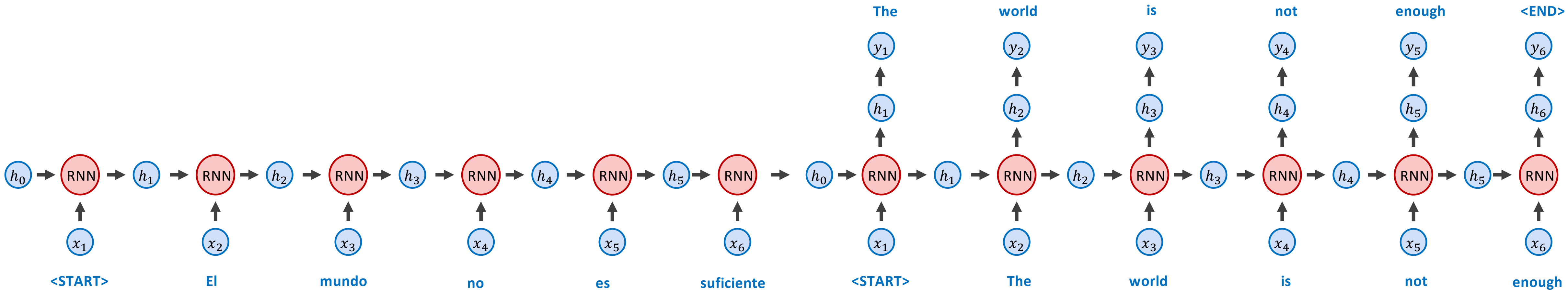
Auto-regressive model – Sequence to Sequence during Training, Auto-regressive during test



How can it be used? – e.g. Machine Translation

Sequence to Sequence – Encoding – Decoding – Many to Many mapping

DURING TRAINING



How can it be used? – e.g. Machine Translation
Sequence to Sequence Models

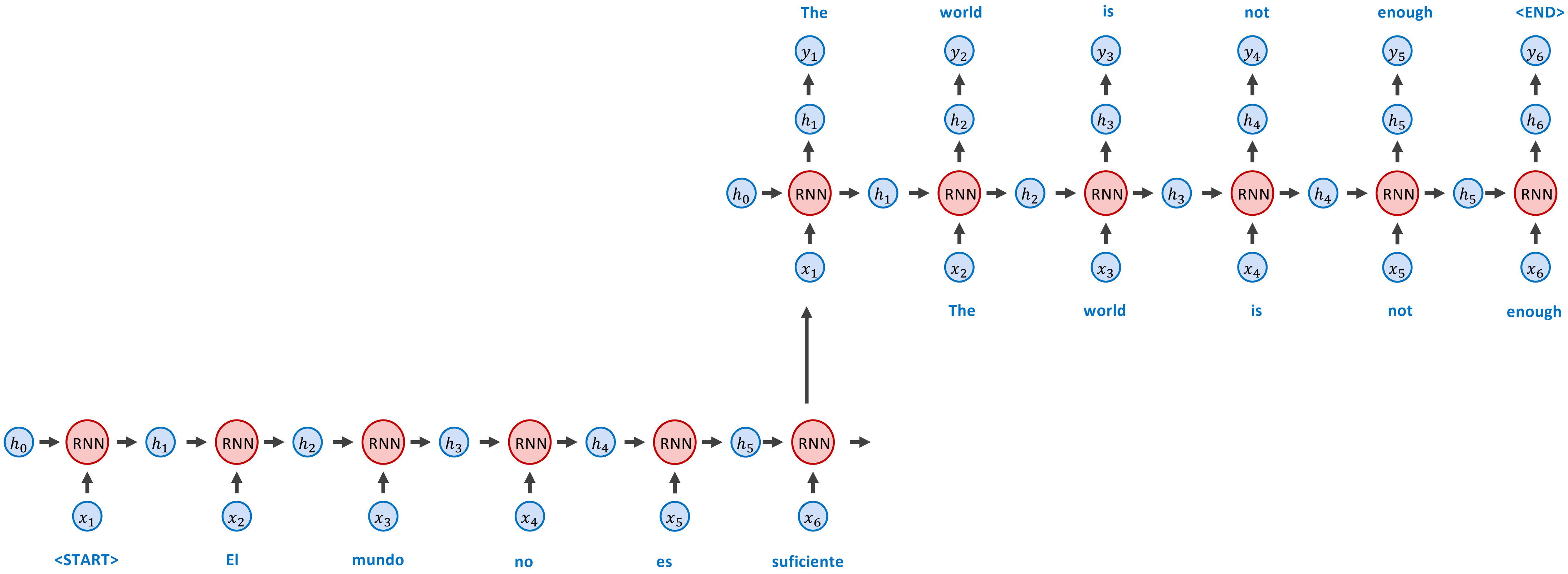
Input training examples don't need to be the same length!
In this case outputs can be.

input	output
<START> este restaurante tiene buena comida	**this restaurant has good food <END>**
<START> this restaurant has good food	
<START> el mundo no es suficiente	**the world is not enough <END>**
<START> the world is not enough	

How can it be used? – e.g. Machine Translation

Sequence to Sequence – Encoding – Decoding – Many to Many mapping

DURING TRAINING – (Alternative)



Problems

- Long Sequences lead to vanishing

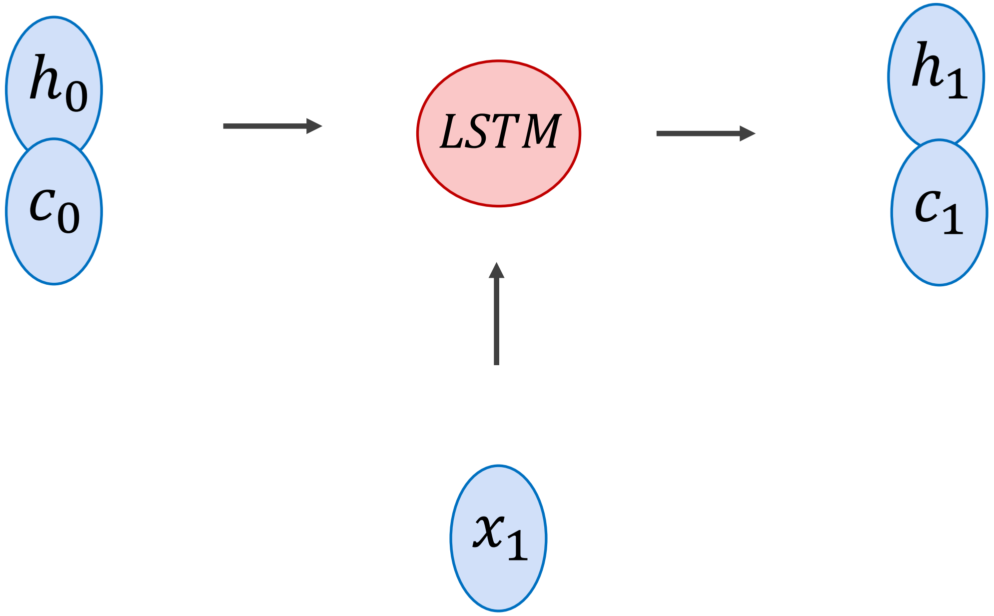
- Hidden states can not carry information in a long sequence (Telephone Game problem)

Solutions Proposed

- Use another hidden state variable and experiment with more complex transition functions than $h = \tanh(W_1 h + W_2 x)$.
  - Read about LSTMs, GRUs, etc

LSTM Cell (Long Short-Term Memory)

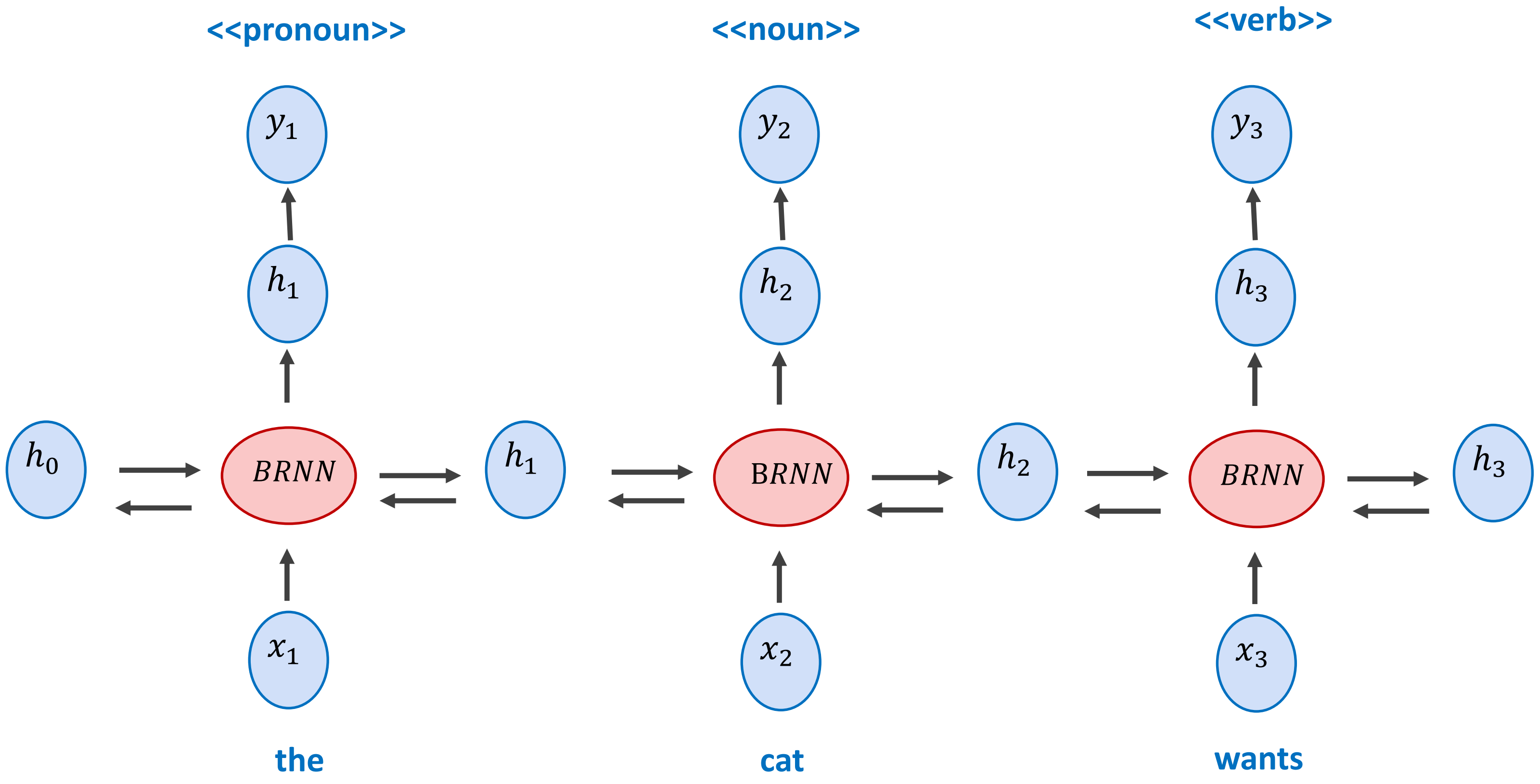
$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right) \quad (7)$$
$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right) \quad (8)$$
$$c_t = f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right) \quad (9)$$
$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right) \quad (10)$$
$$h_t = o_t \tanh(c_t) \quad (11)$$



Solutions Proposed

- Use another hidden state variable and experiment with more complex transition functions than h = tanh(W_1h + W_2x).
  - Read about LSTMs, GRUs, etc
- Encode the sentences both from left-to-right and right-to-left using two RNNs and combine the final hidden states from each direction.
  - Read about Bidirectional RNNs (BiRNNs), BiLSTMs, BiGRUs

Bidirectional Recurrent Neural Network



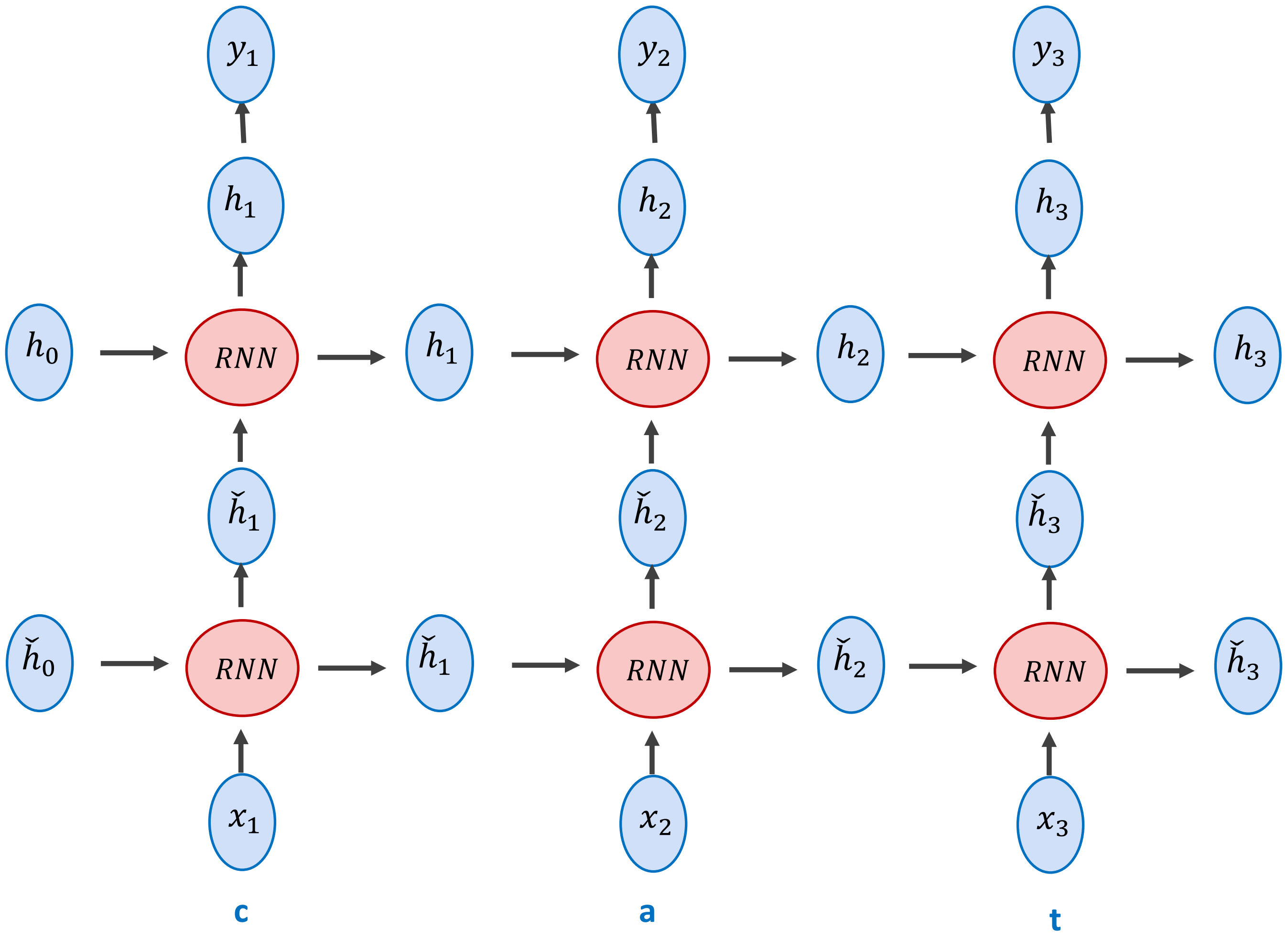
Solutions Proposed

- Use another hidden state variable and experiment with more complex transition functions than $h = \tanh(W_1 h + W_2 x)$.
  - Read about LSTMs, GRUs, etc

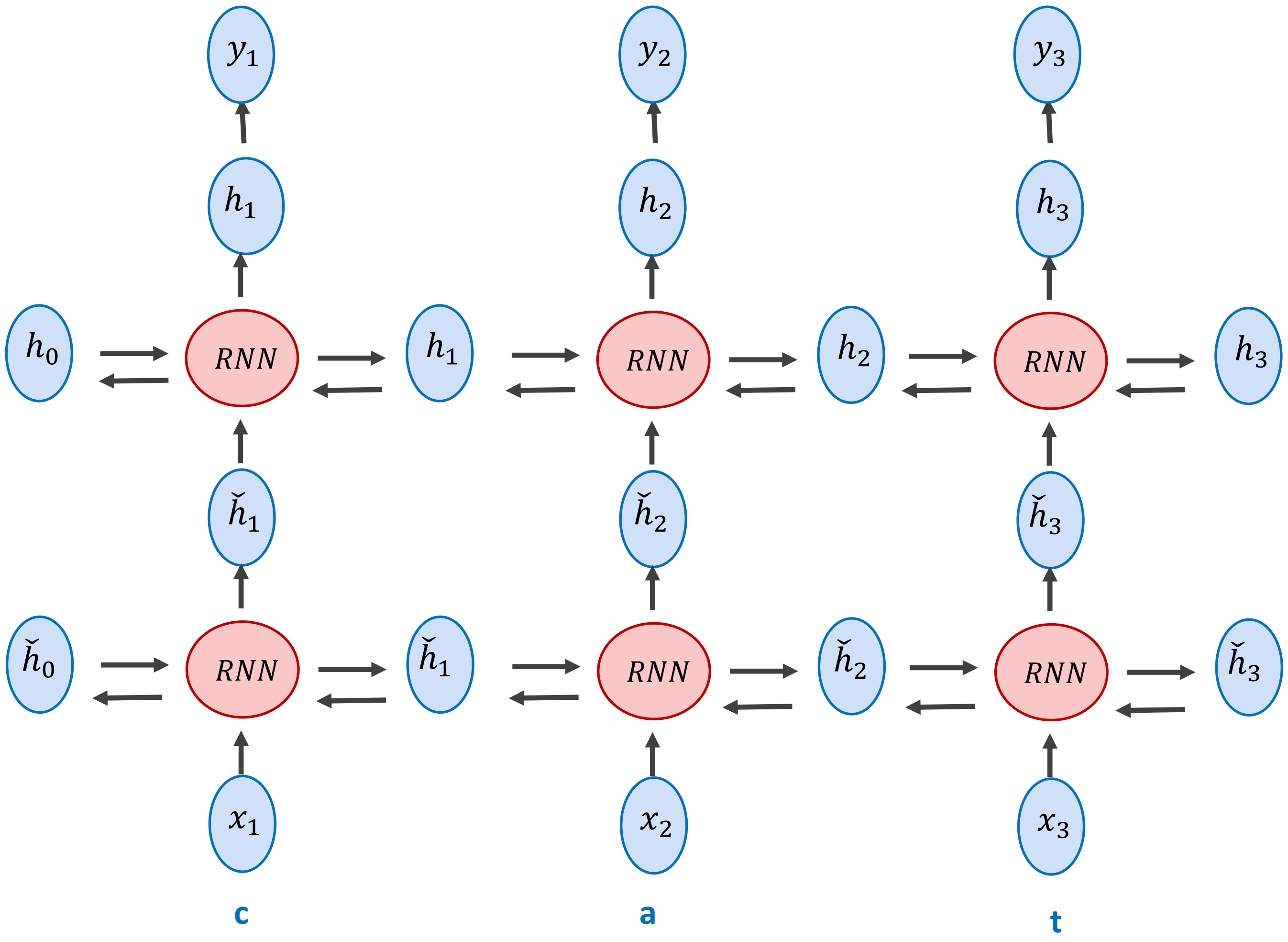
- Encode the sentences both from left-to-right and right-to-left using two RNNs and combine the final hidden states from each direction.
  - Read about Bidirectional RNNs (BiRNNs), BiLSTMs, BiGRUs

- Stack RNNs, use an RNN that feeds its output states to another RNN and this second RNN outputs the final output states.
  - Stacked RNNs, or Deep RNNs.

Stacked Recurrent Neural Network

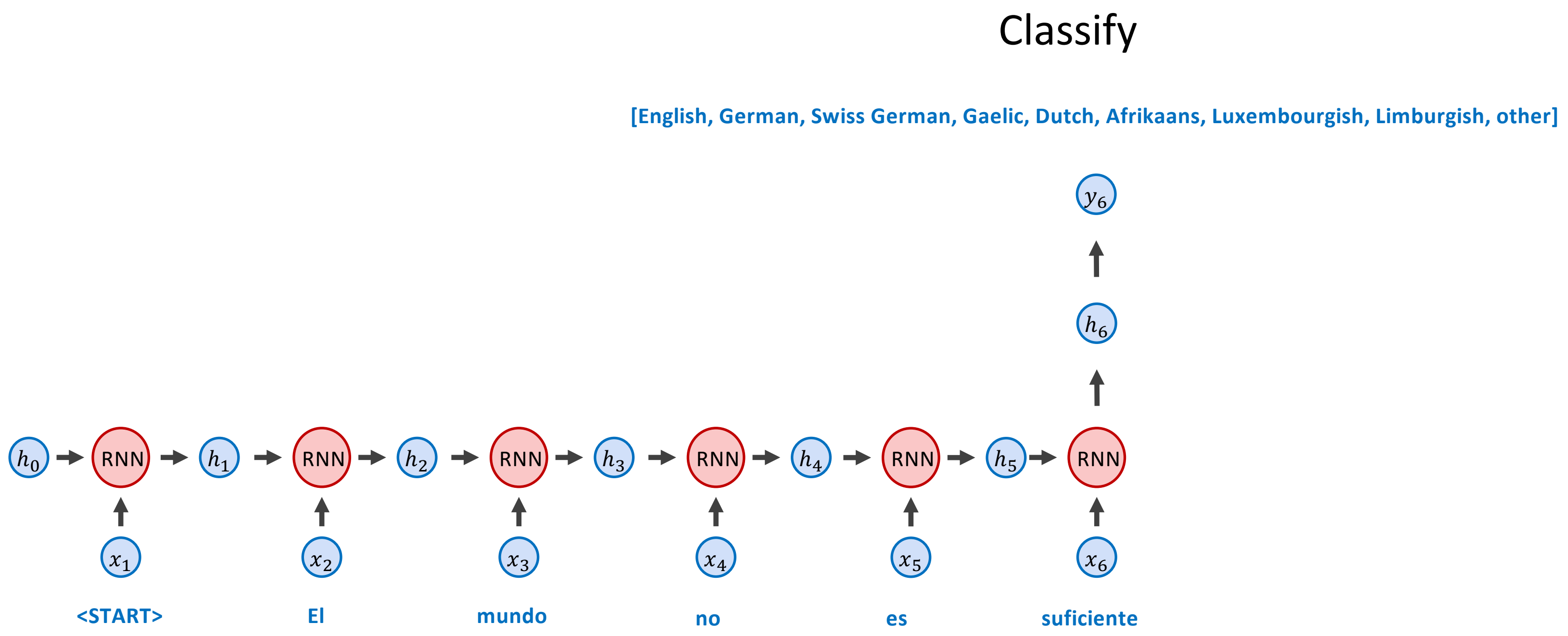


Stacked Bidirectional Recurrent Neural Network

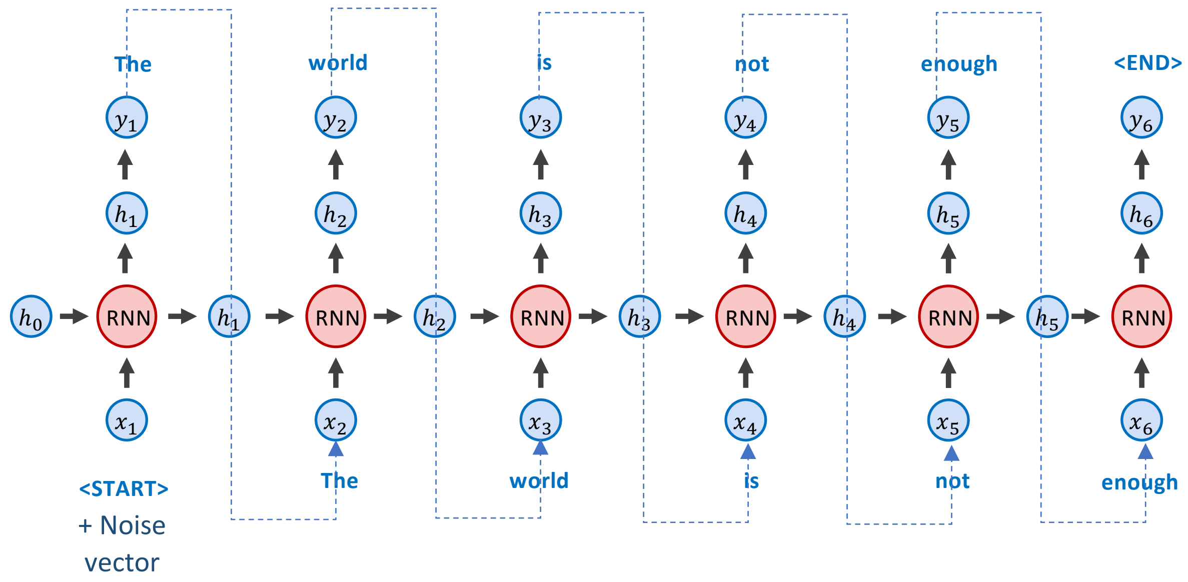


Best Solution: Learning Attention Weights

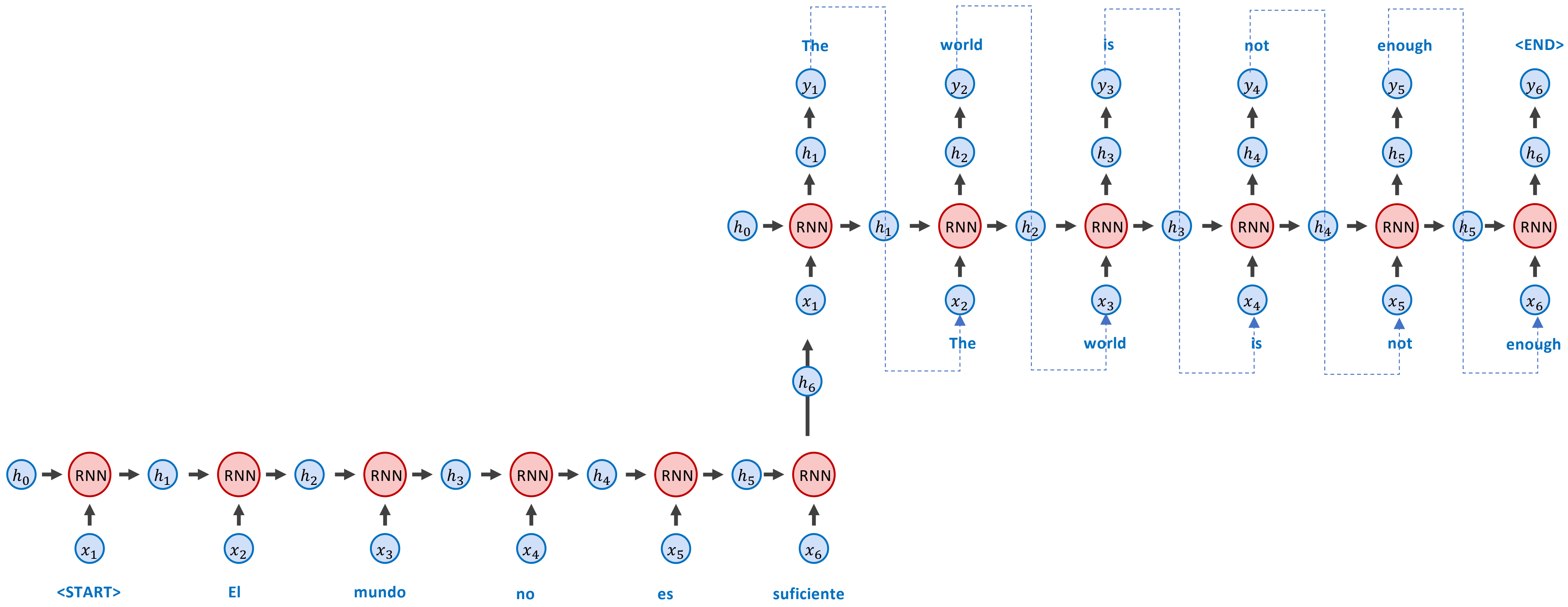
RNNs – Sequence to score prediction



RNNs for Text Generation (Auto-regressive)

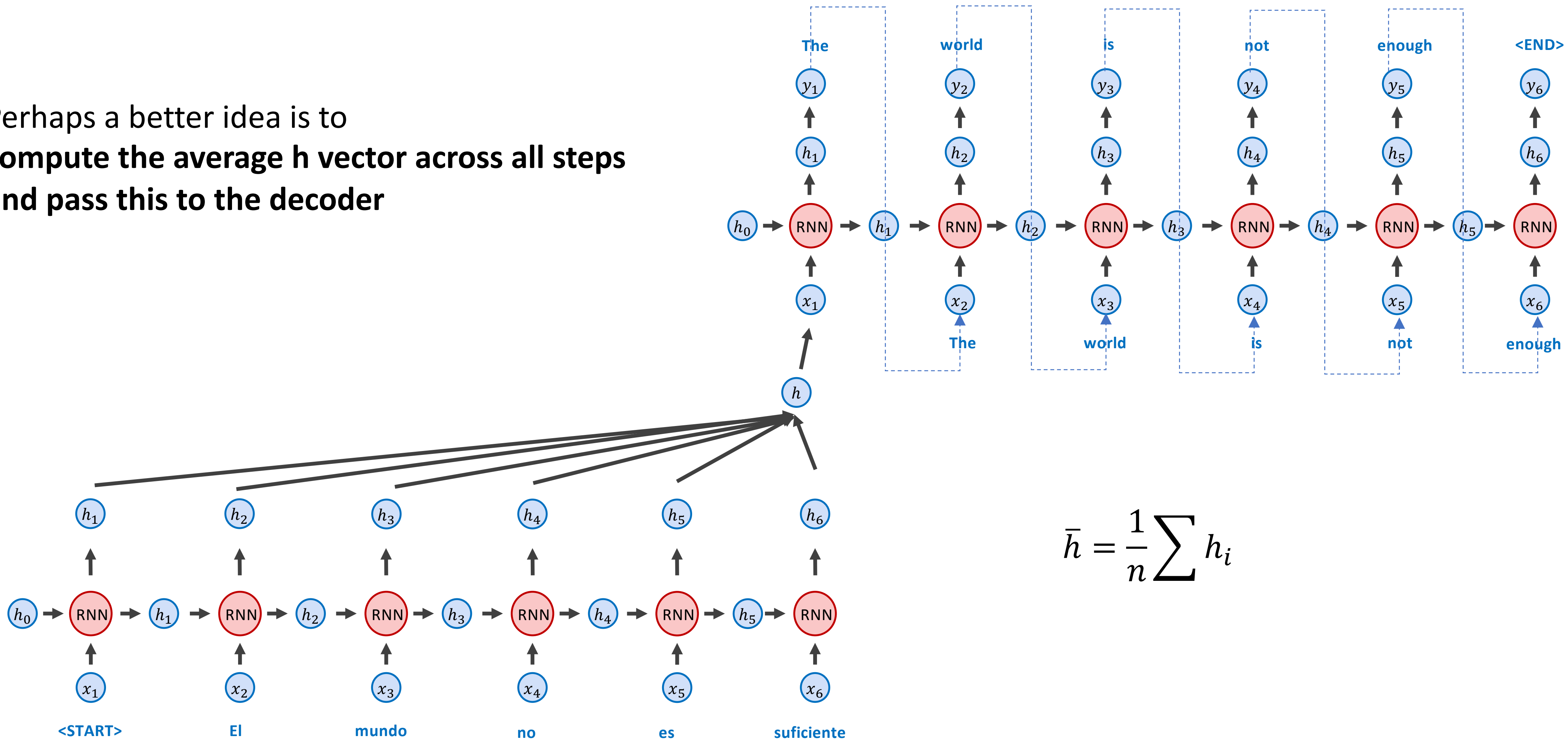


RNNs for Machine Translation Seq-to-Seq



RNNs for Machine Translation Seq-to-Seq

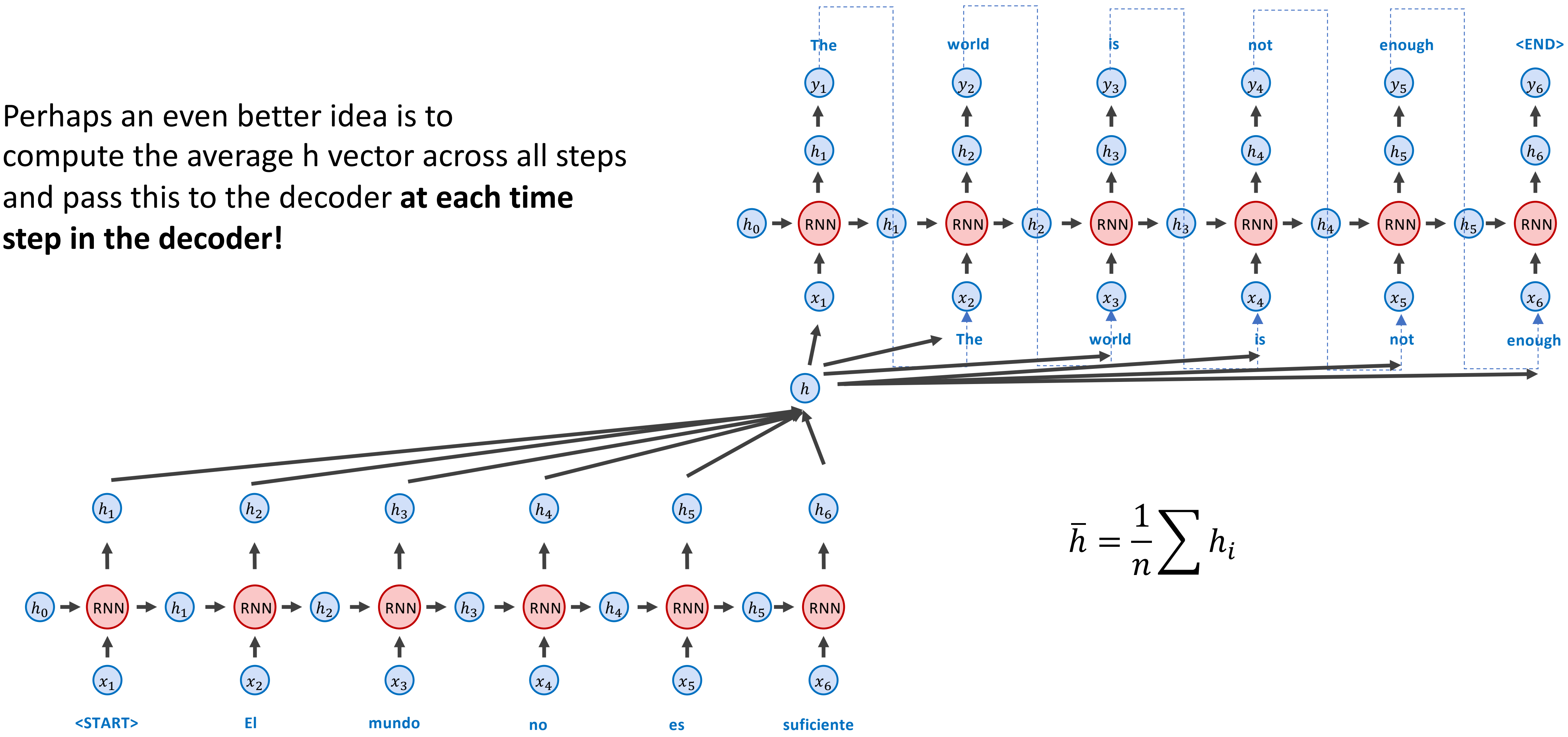
Perhaps a better idea is to
compute the average h vector across all steps and pass this to the decoder



$$\bar{h} = \frac{1}{n}\sum h_i$$

RNNs for Machine Translation Seq-to-Seq

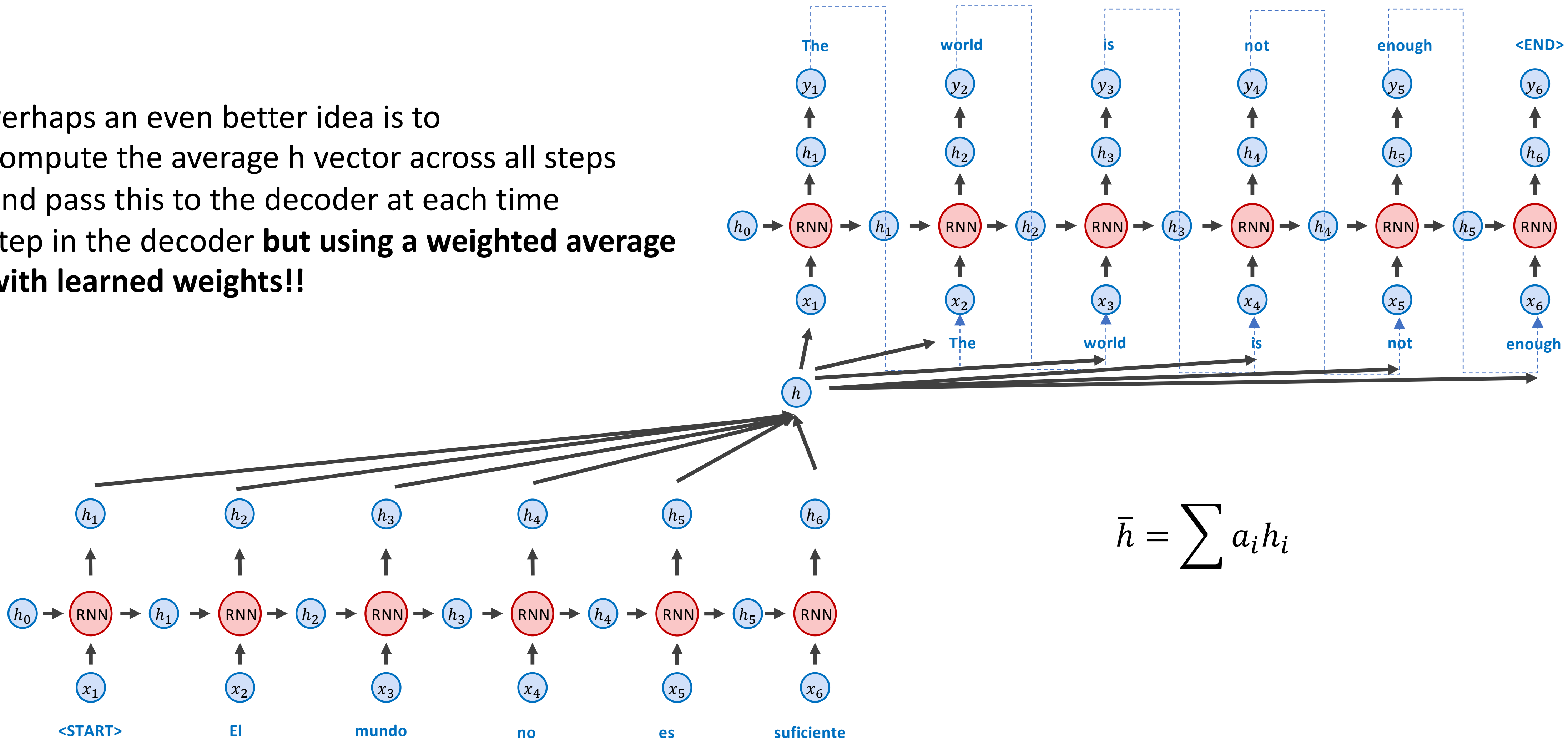
Perhaps an even better idea is to compute the average h vector across all steps and pass this to the decoder **at each time step in the decoder!**



$$\bar{h} = \frac{1}{n}\sum h_i$$

RNNs for Machine Translation Seq-to-Seq

Perhaps an even better idea is to compute the average h vector across all steps and pass this to the decoder at each time step in the decoder **but using a weighted average with learned weights!!**

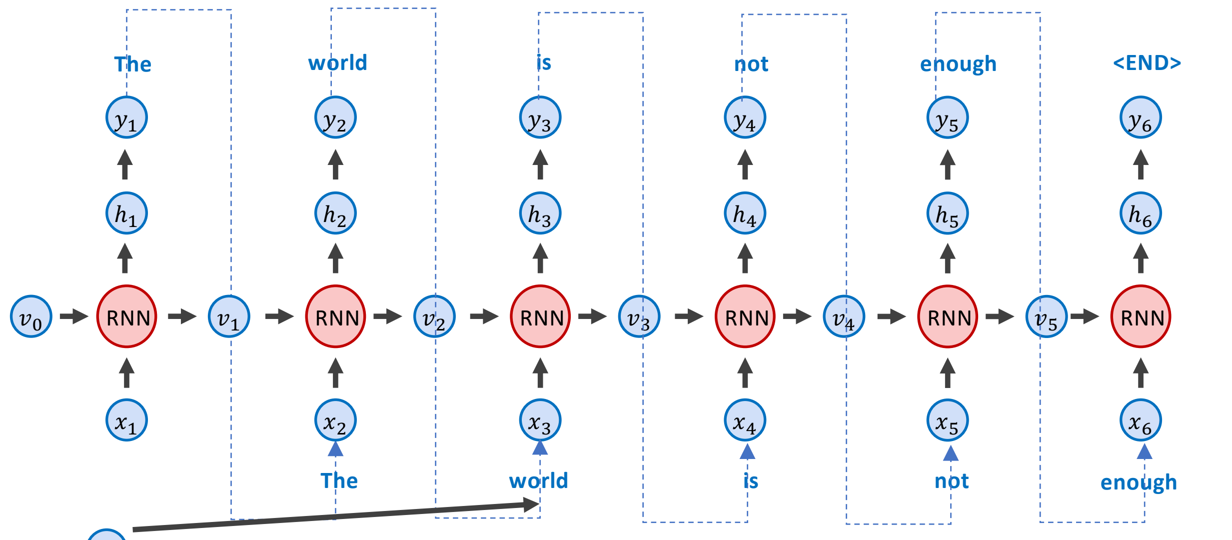


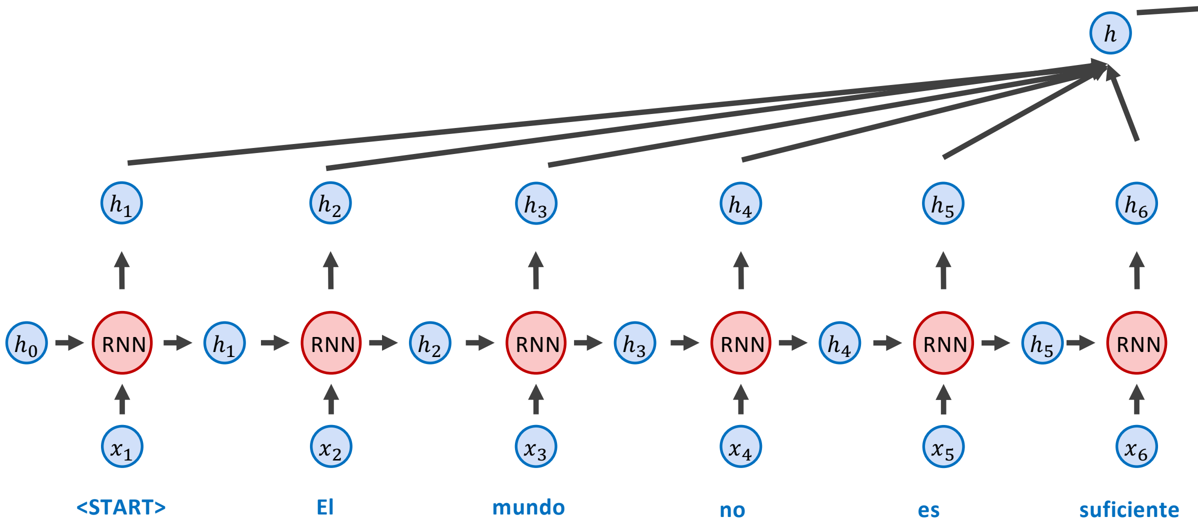
$$\bar{h} = \sum a_i h_i$$

RNNs for Machine Translation Seq-to-Seq

Only showing the third time step encoder-decoder connection

Perhaps an even better idea is to compute the average h vector across all steps and pass this to the decoder at each time step in the decoder but using a weighted average with learned weights, **and the weights are specific for each time step!!!**





$$\bar{h_j} = \sum a_{j,i} h_i$$

such that:

$$a_{j,i} = \frac{\exp(h_j v_{j-1})}{\sum \exp(h_i v_{i-1})}$$

Neural Machine Translation by Jointly Learning to Align and Translate

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal

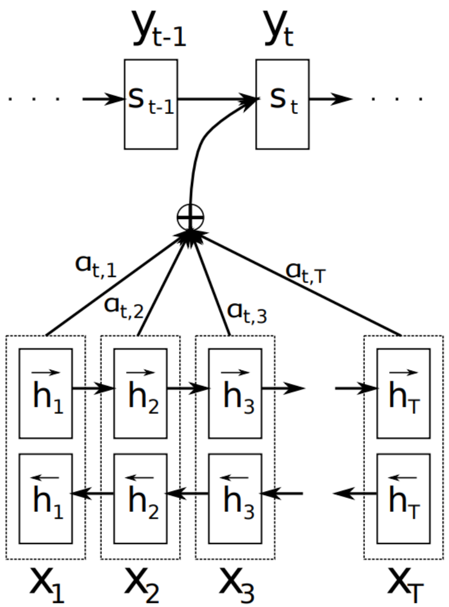


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

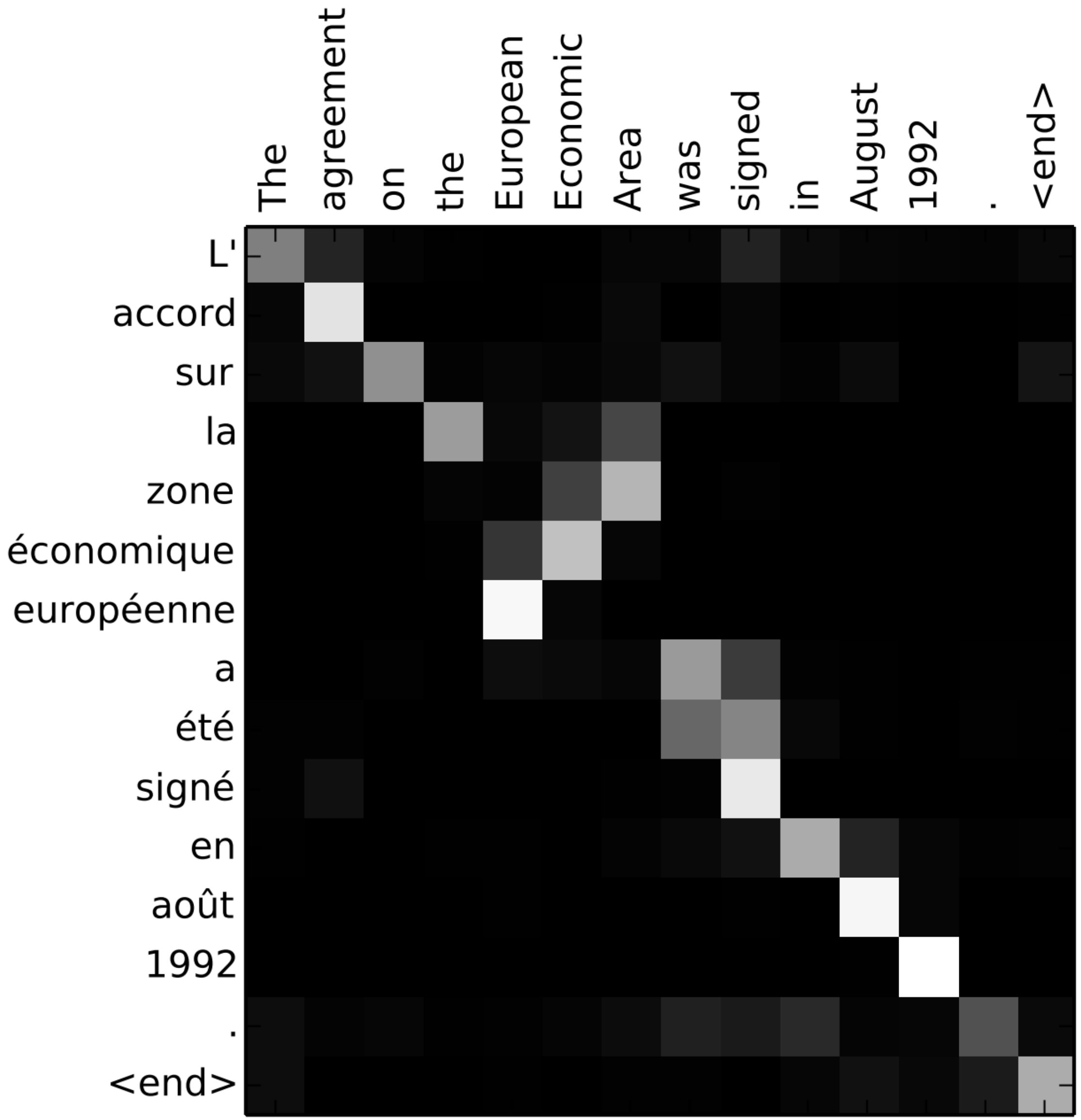
Let's take a look at one of the first papers introducing this idea.

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

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

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

Let’s look at the Attention weights



Transformers: Attention is All You Need

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

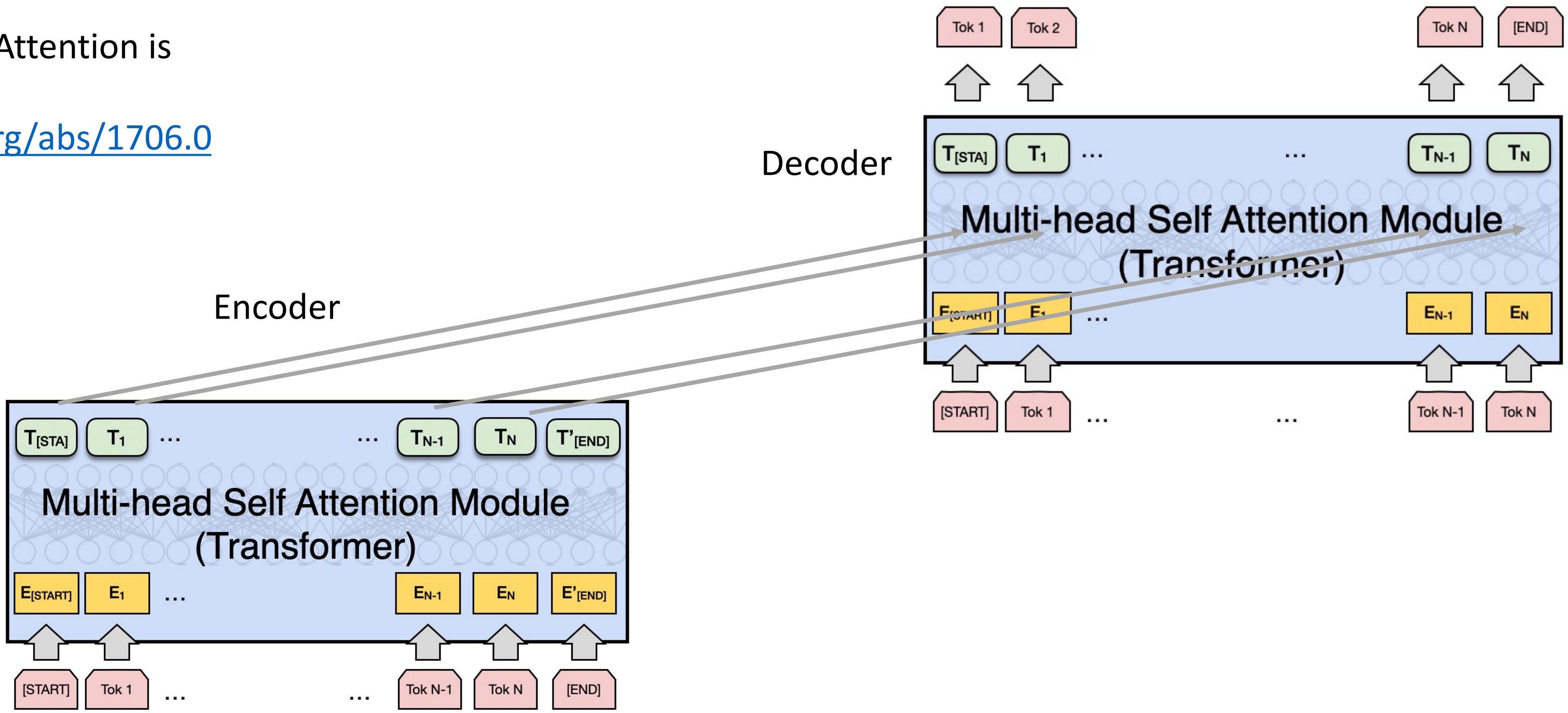
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Attention is All you Need (no RNNs)

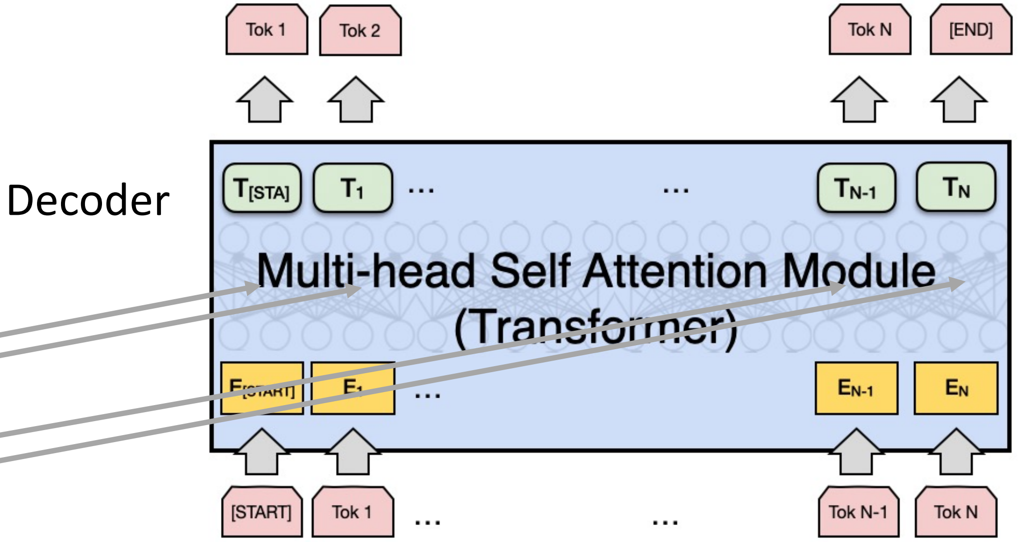
Vaswani et al. Attention is all you need https://arxiv.org/abs/1706.03762

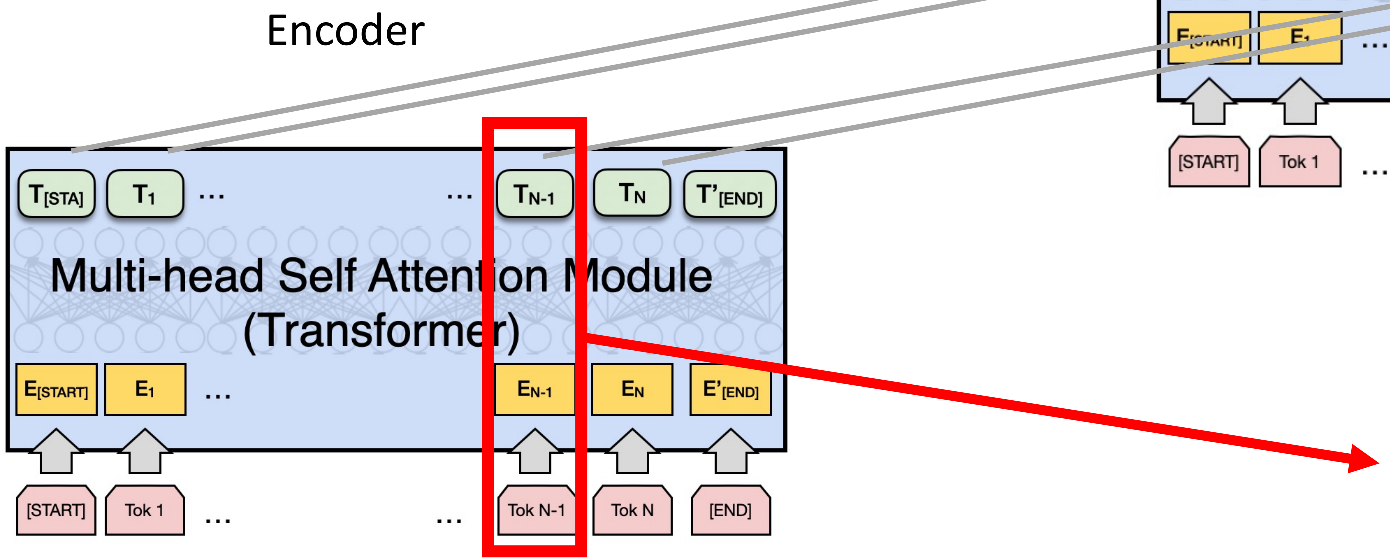


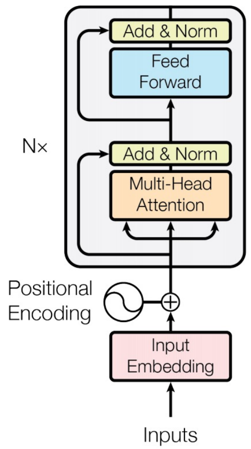
Fixed number of input tokens
[but hey! we can always define a large enough length and add mask tokens]

Attention is All you Need (no RNNs)

Vaswani et al. Attention is all you need https://arxiv.org/abs/1706.03762





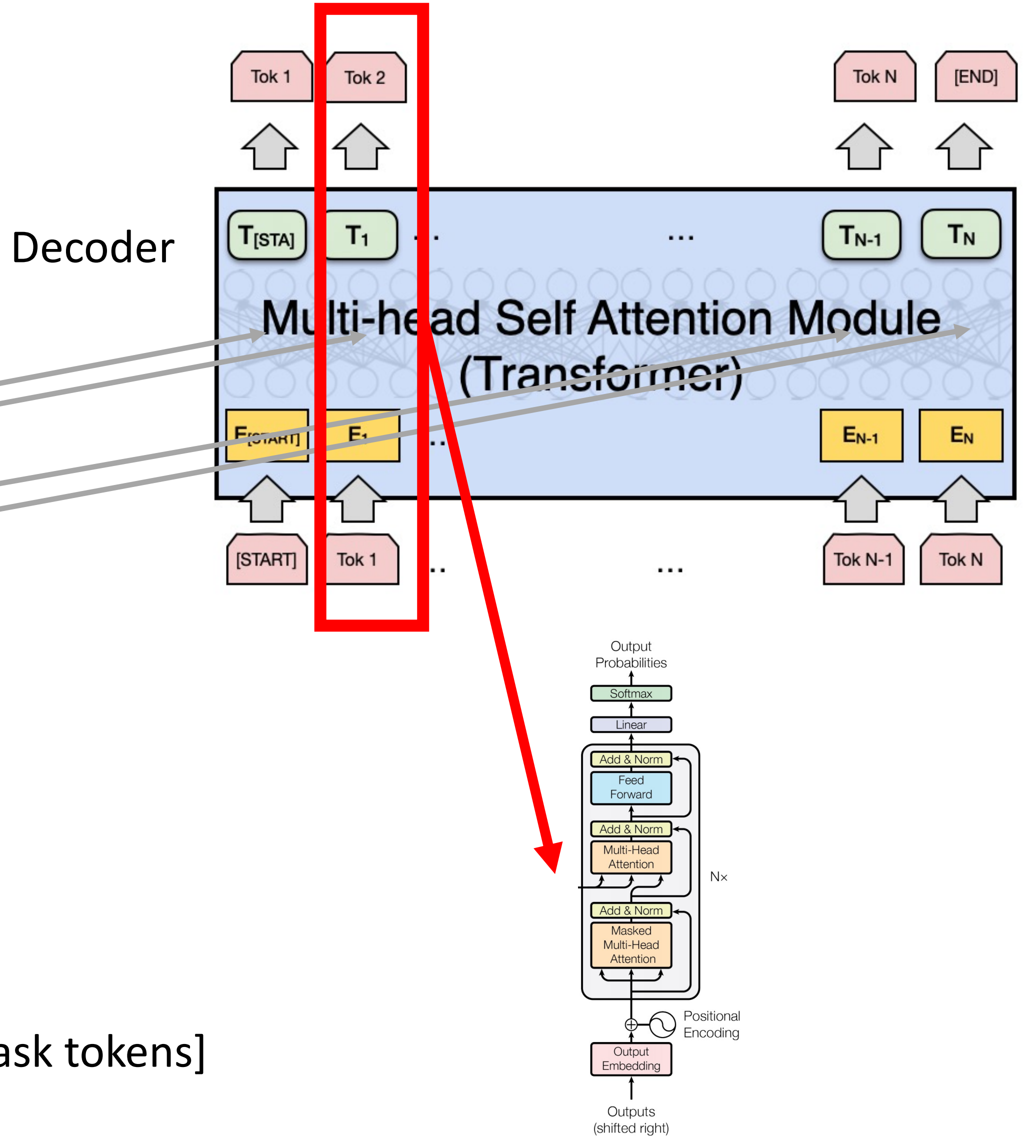


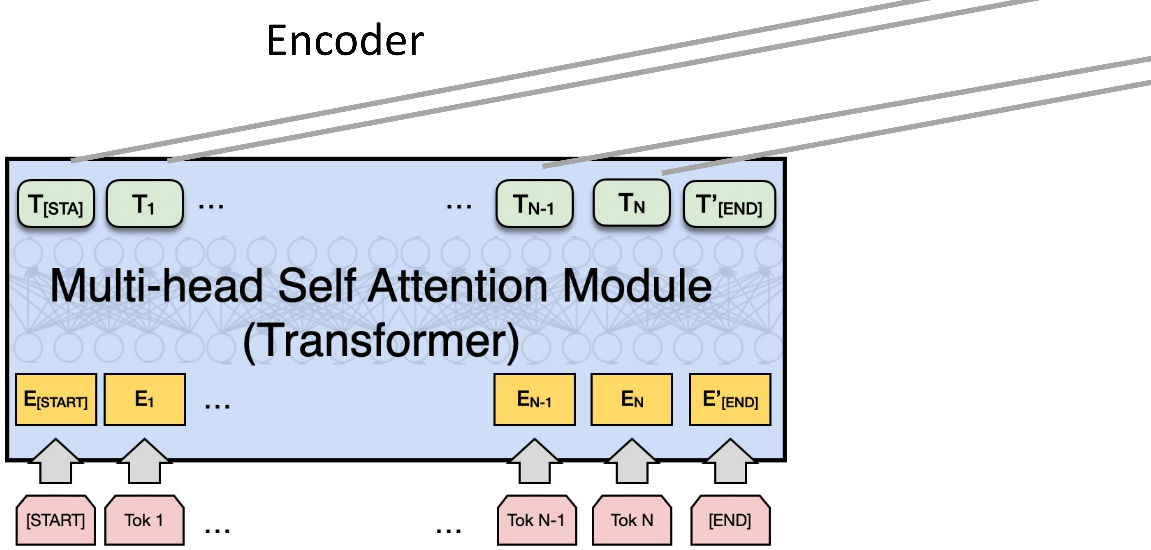
Fixed number of input tokens

[but hey! we can always define a large enough length and add mask tokens]

Attention is All you Need (no RNNs)

Vaswani et al. Attention is all you need
https://arxiv.org/abs/1706.03762

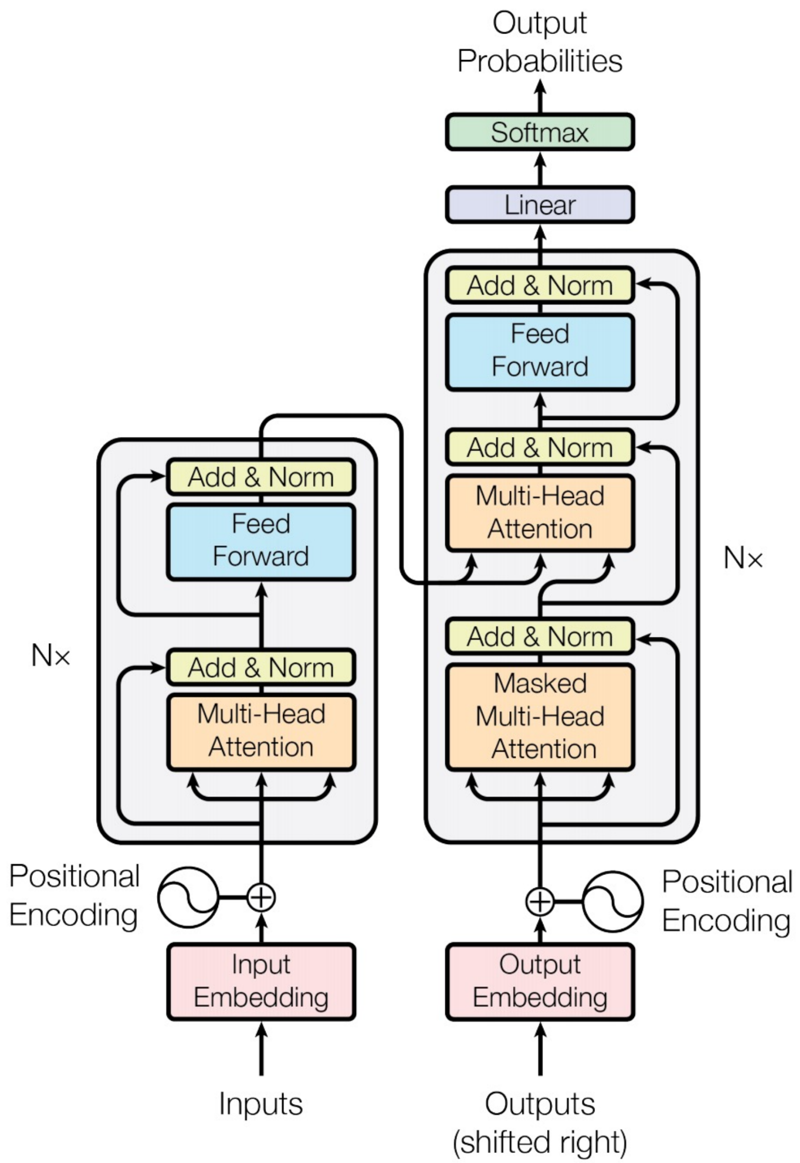




Fixed number of input tokens
[but hey! we can always define a large enough length and add mask tokens]

We can also draw this as in the paper:

Vaswani et al. Attention is all you need https://arxiv.org/abs/1706.03762

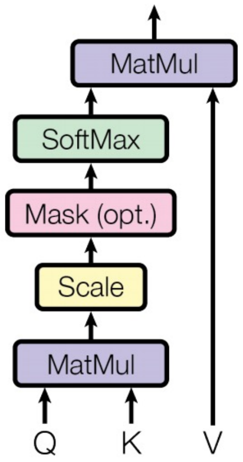


Regular Attention: + Scaling factor

Vaswani et al. Attention is all you need https://arxiv.org/abs/1706.03762

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

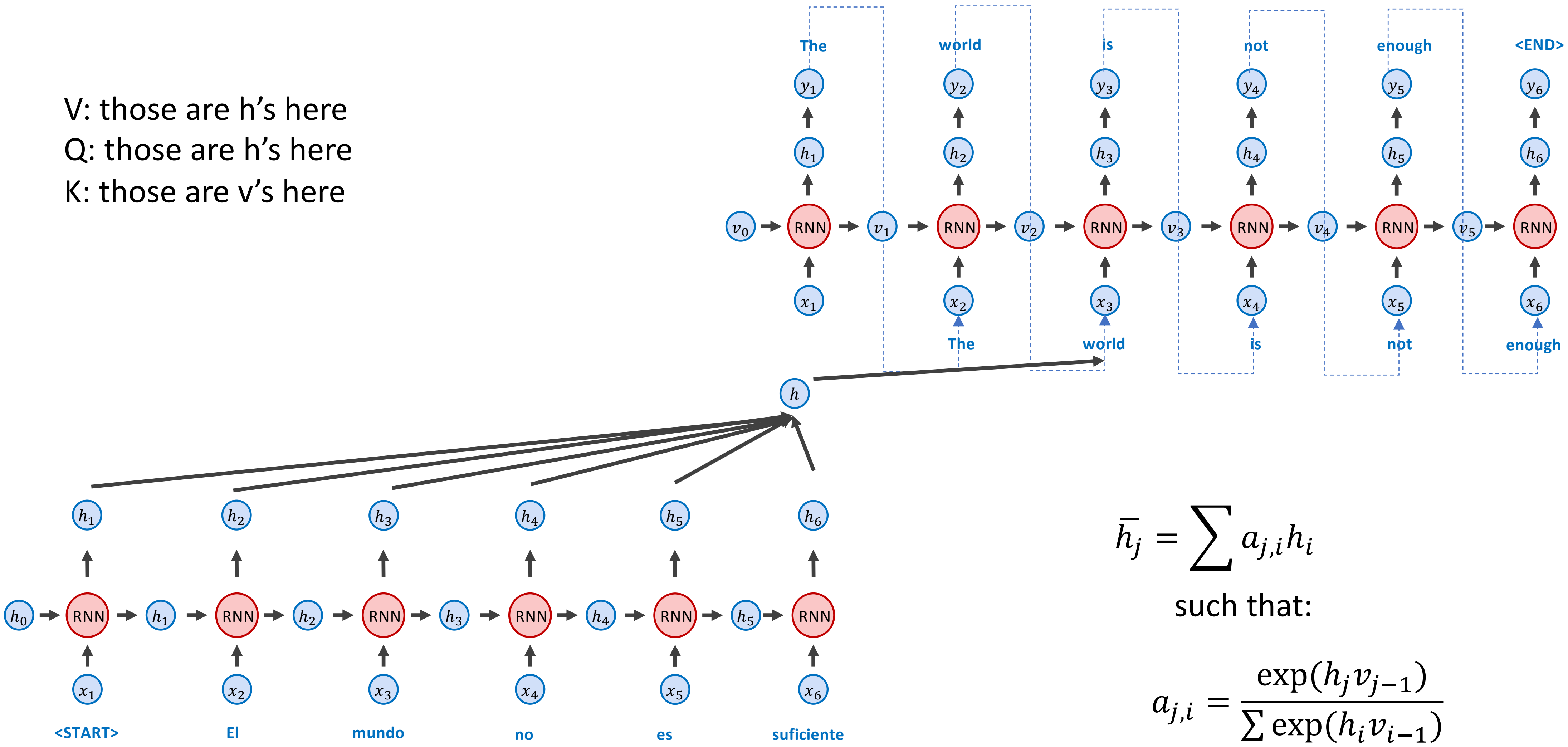
Scaled Dot-Product Attention



This is not unlike what we already used before

Only showing the third time step encoder-decoder connection

V: those are h’s here
Q: those are h’s here
K: those are v’s here



$$\bar{h_j} = \sum a_{j,i} h_i$$

such that:

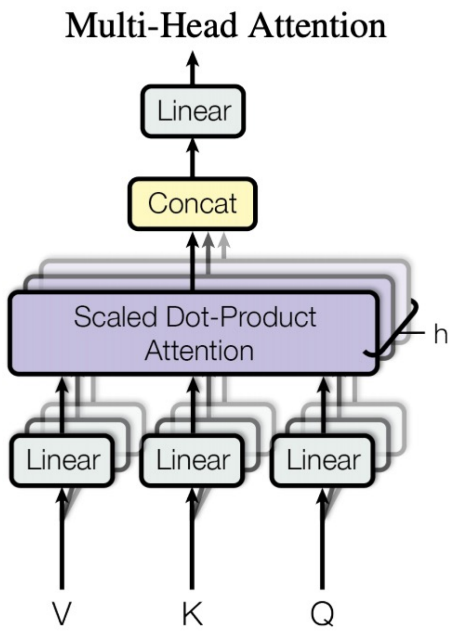
$$a_{j,i} = \frac{\exp(h_j v_{j-1})}{\sum \exp(h_i v_{i-1})}$$

Multi-head Attention: Do not settle for just one set of attention weights.

Vaswani et al. Attention is all you need https://arxiv.org/abs/1706.03762

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

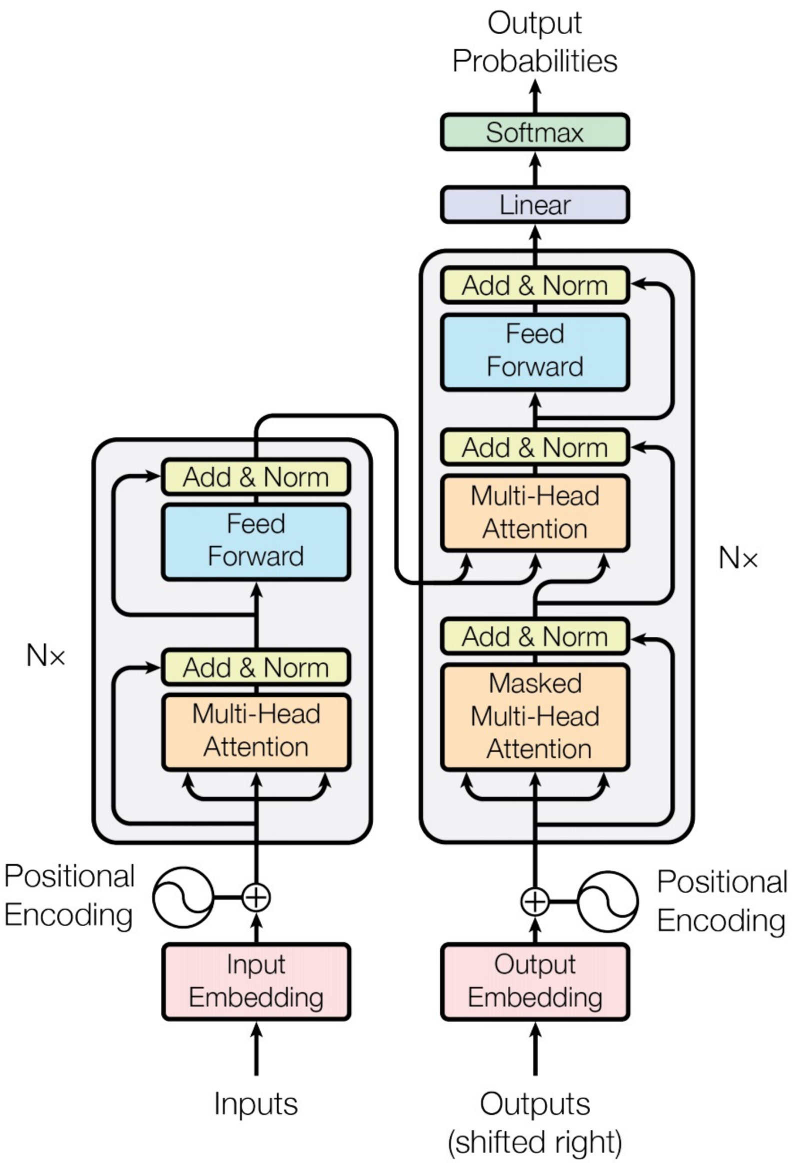
Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.



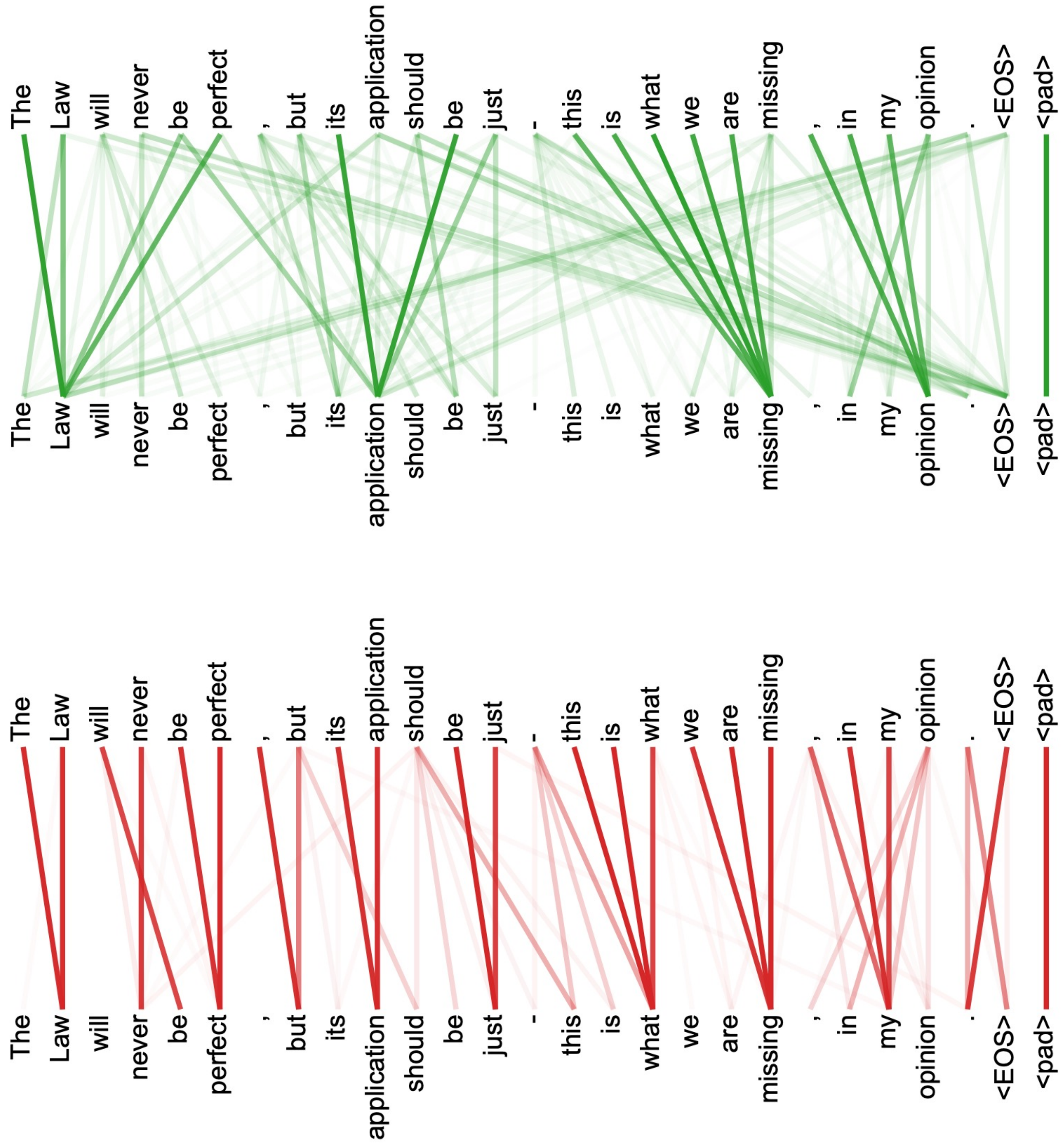
We can lose track of position since we are aggregating across all locations

Vaswani et al. Attention is all you need https://arxiv.org/abs/1706.03762

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



Multi-headed attention weights are harder to interpret obviously

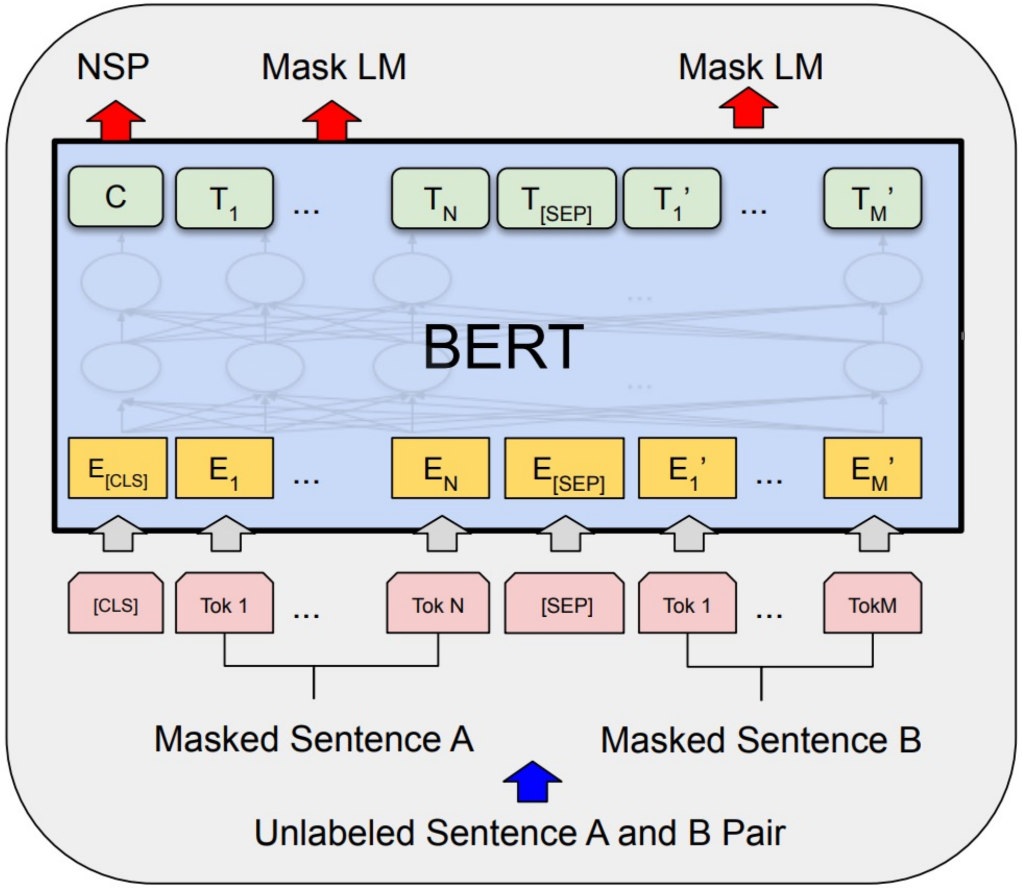


The BERT Encoder Model

Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding . https://arxiv.org/abs/1810.04805

Important things to know

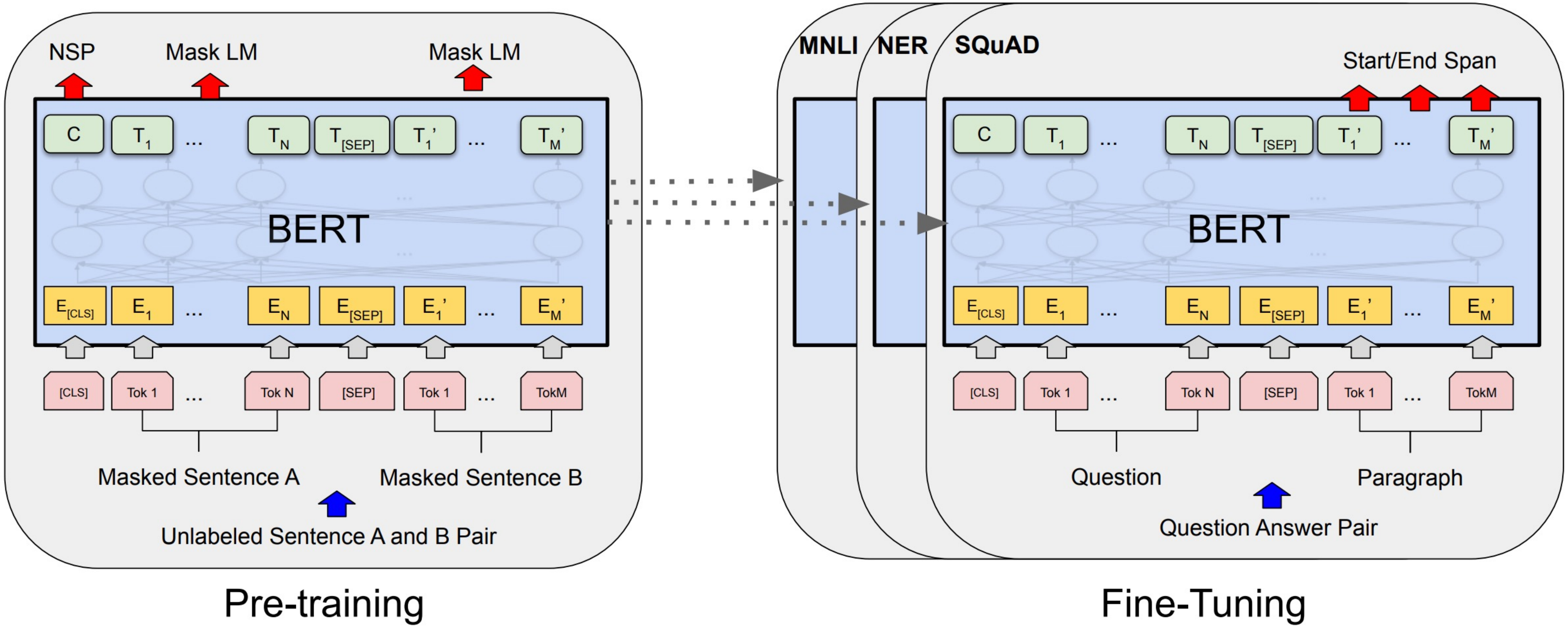
- No decoder
- Train the model to fill-in-the-blank by masking some of the input tokens and trying to recover the full sentence.
- The input is not one sentence but two sentences separated by a [SEP] token.
- Also try to predict whether these two input sentences are consecutive or not.



Pre-training

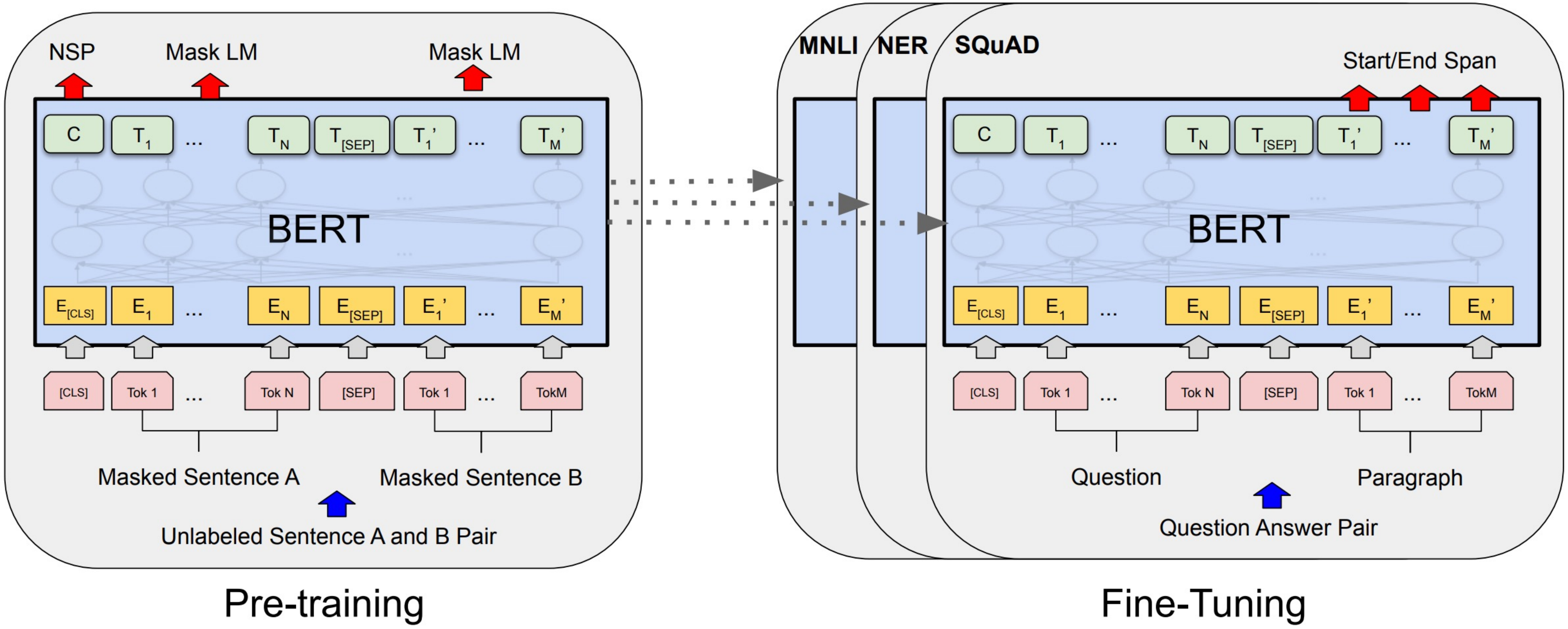
The BERT Encoder Model

Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding . https://arxiv.org/abs/1810.04805

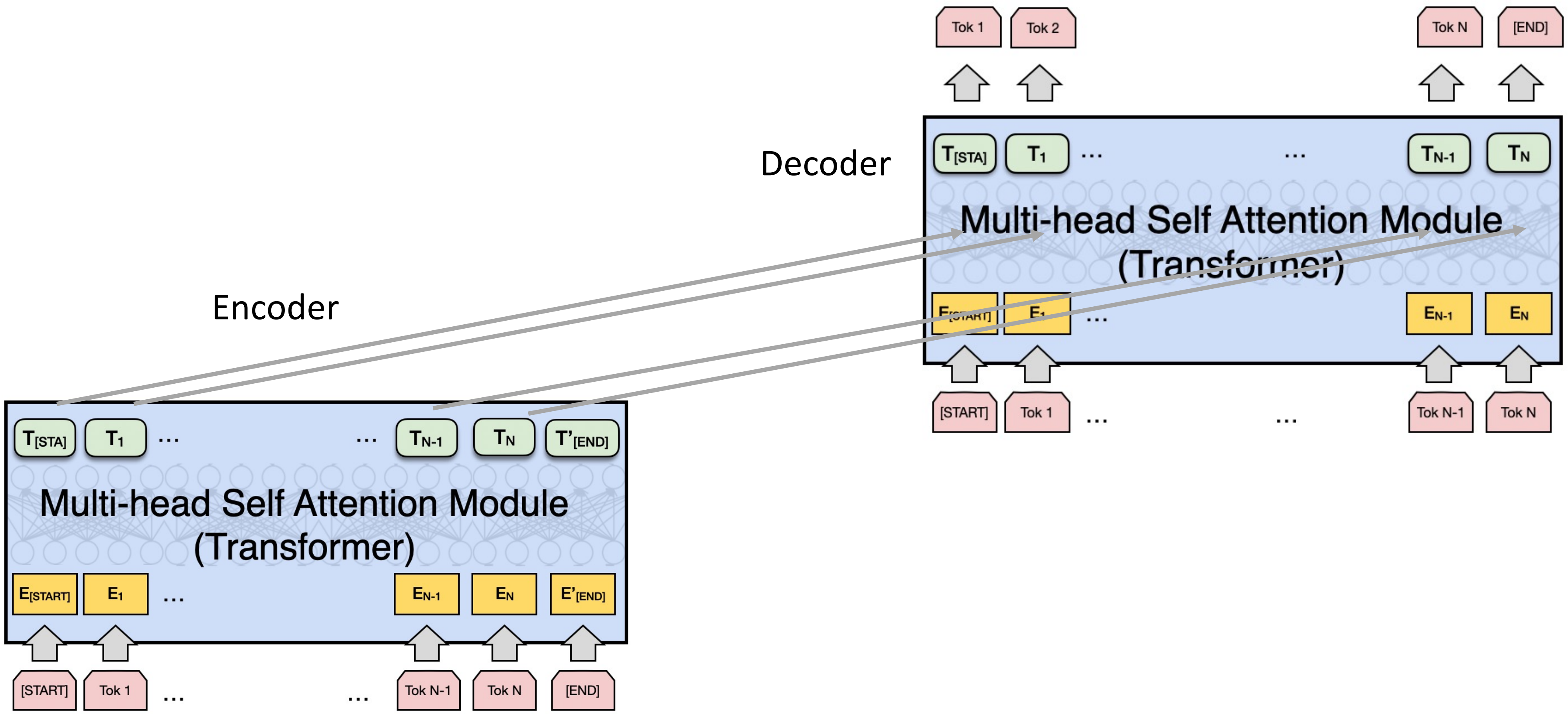


The BERT Encoder-only Model

Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding . https://arxiv.org/abs/1810.04805

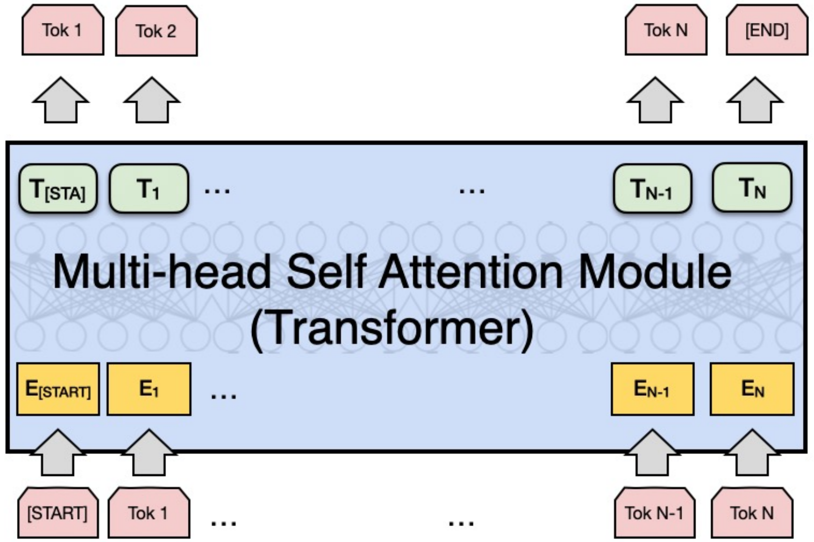


The T5 Encoder-Decoder Model

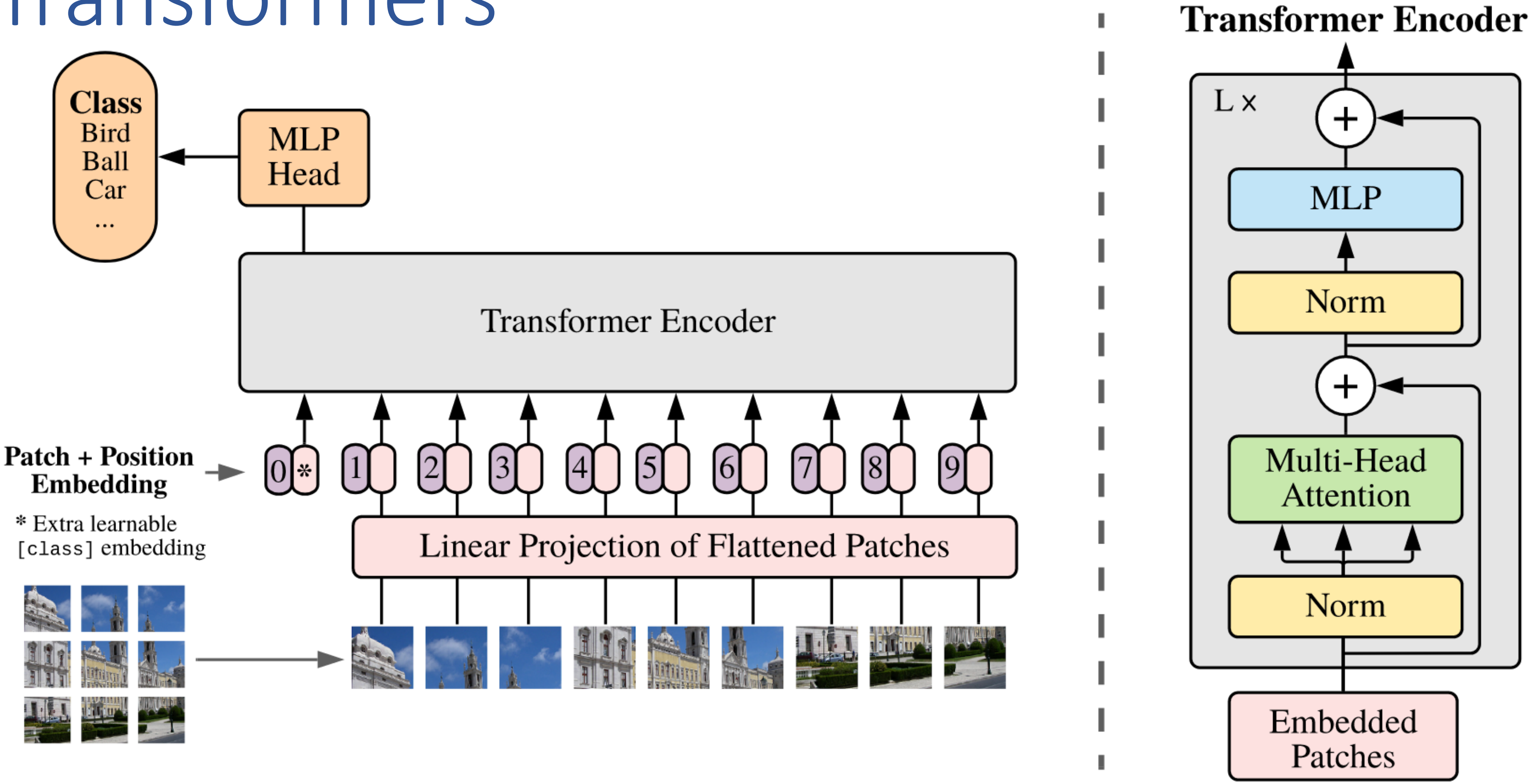


The GPT-2, GPT-3 Decoder-only Model

Decoder



Vision Transformers

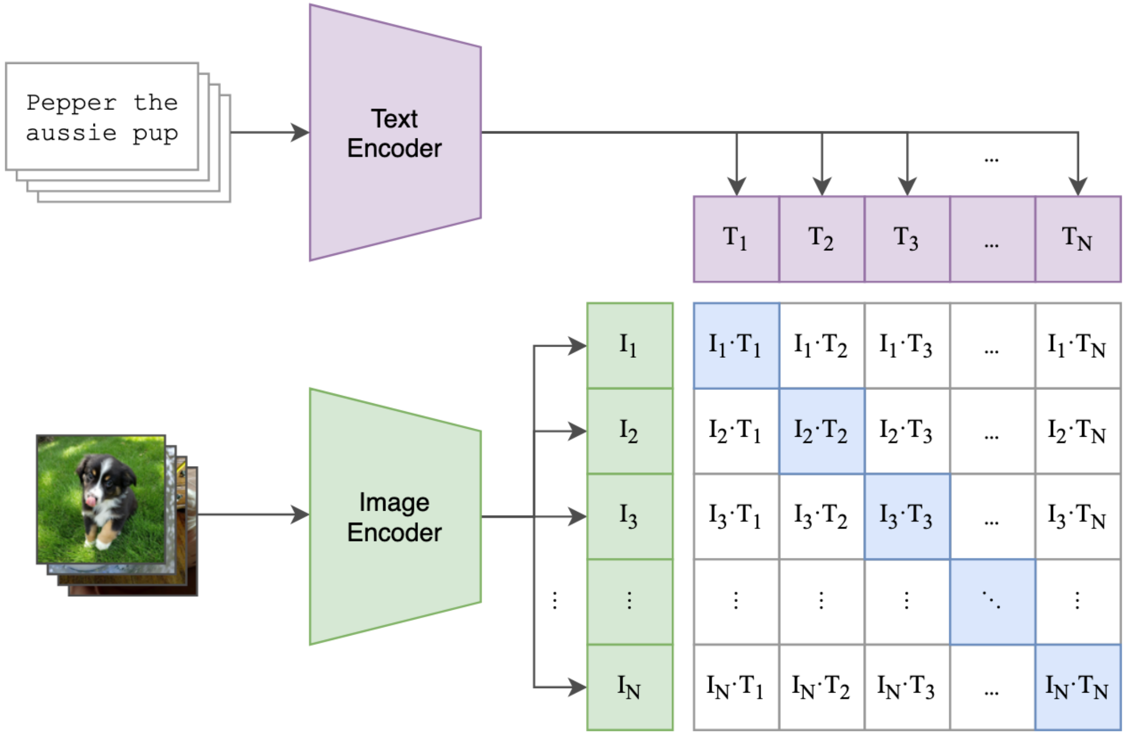


https://arxiv.org/abs/2010.11929

An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby

The CLIP Model



$$L = \sum_k \ell(I_k T_k)$$

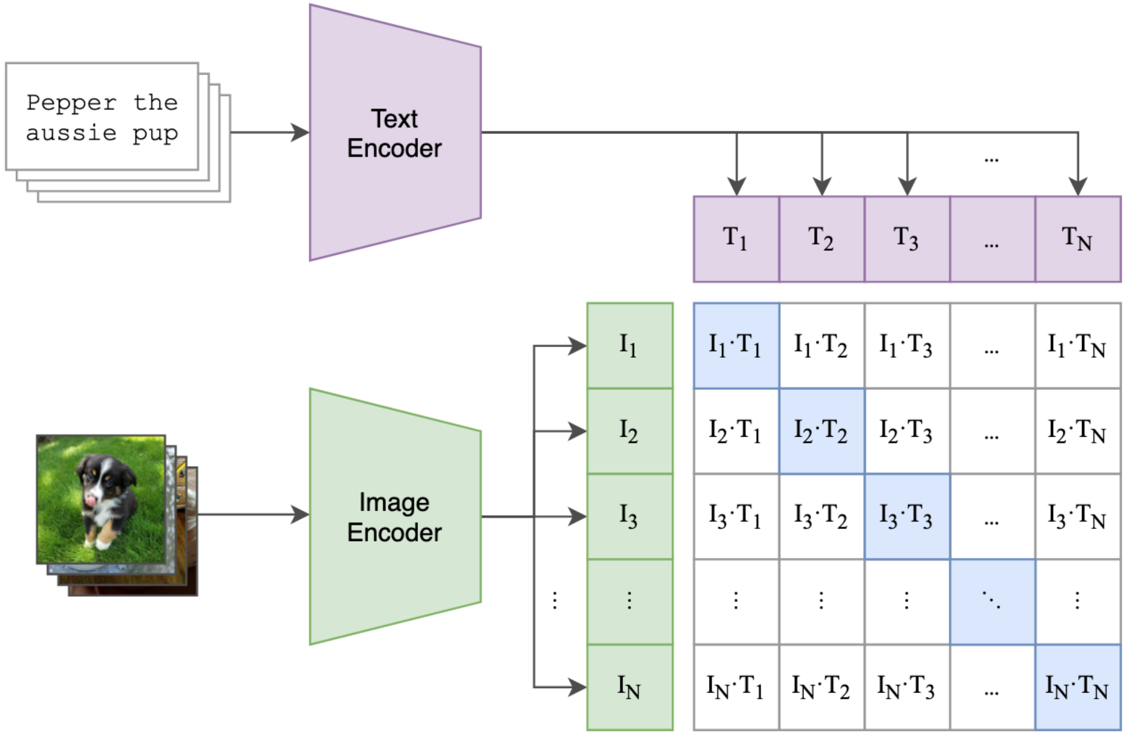
$$\ell(I_k T_k) = -\log\left(\frac{\exp(sim(I_k, T_k))}{\sum_{t=1}^{2N} 1[k \neq i]\exp(sim(I_k, T_t))}\right)$$

https://arxiv.org/abs/2103.00020

Learning Transferable Visual Models From Natural Language Supervision

[Alec Radford](), [Jong Wook Kim](), [Chris Hallacy](), [Aditya Ramesh](), [Gabriel Goh](), [Sandhini Agarwal](), [Girish Sastry](), [Amanda Askell](), [Pamela Mishkin](), [Jack Clark](), [Gretchen Krueger](), [Ilya Sutskever]()

The CLIP Model



$$L = \sum_k \ell_1(I_k T_k) + \ell_2(I_k T_k)$$

$$\ell_1(I_k T_k) = -\log\left(\frac{\exp(sim(I_k, T_k))}{\sum_{t=1}^{2N} 1[k \neq i] \exp(sim(I_k, T_t))}\right)$$

$$\ell_2(I_k T_k) = -\log\left(\frac{\exp(sim(I_k, T_k))}{\sum_{t=1}^{2N} 1[k \neq i] \exp(sim(I_t, T_k))}\right)$$

https://arxiv.org/abs/2103.00020

Learning Transferable Visual Models From Natural Language Supervision
Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, Ilya Sutskever

Questions?