VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

Transformer Models

Dr. David Raj M

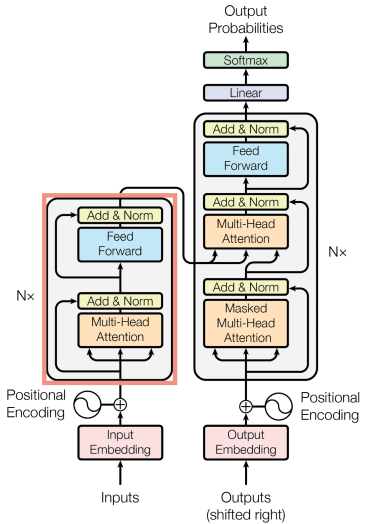
Assistant Professor
Department of Mathematics
School of Advanced Sciences
Vellore Institute of Technology, Chennai

Feb 12, 2026

This lecture is delivered at the FDP in Dayananda Sagar Engineering College.

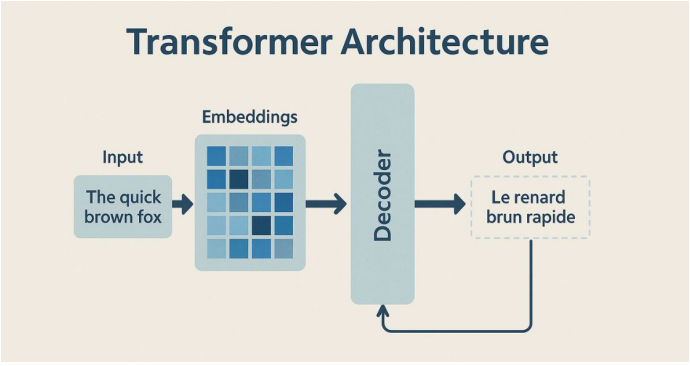
Contents

Transformer Models



Transformer Models: Overview

- Type of neural network architecture excelling at sequential data processing.
- Most prominent in large language models (LLMs).
- Elite performance in AI fields: computer vision, speech recognition, time series forecasting.



Origins of the Transformer

Seminal Paper

"Attention is All You Need" (2017) by Vaswani et al.
Watershed moment in deep learning.

Evolution from RNN-based sequence-to-sequence models for machine translation.

- Cutting-edge advancements across nearly every ML discipline.

Primary Applications in NLP

Despite versatility, transformers are most discussed in NLP:

- Chatbots
- Text generation
- Summarization
- Question answering
- Sentiment analysis

Key Milestones

BERT (2019, Google)

- Bidirectional Encoder Representations from Transformers
- Encoder-only model
- Basis for modern word embeddings
- Powers vector databases, Google search

GPT-3 (OpenAI)

- Generative Pre-trained Transformer
- Autoregressive decoder-only LLM
- Powered ChatGPT launch
- Catalyzed generative AI era

Multimodal Capabilities

Transformers excel at discerning correlations within data sequences:

- **Vision Transformers (ViTs)** often exceed CNNs in:
  - Image segmentation
  - Object detection
- Power diffusion models for image generation
- Enable multimodal TTS and vision language models (VLMs)

Consider the following pair of sentences:

The lion chased the deer.

The deer chased the lion.

- The two sentences are clearly very different (and the second one is unusual).
- However, the bag-of-words representation would consider them identical.
- Hence, this type of representation works well for simpler applications (such as classification), but a greater degree of linguistic intelligence is required for more sophisticated applications such as sentiment analysis or machine translation.

- Therefore, it is important to somehow encode information about the word ordering more directly within the architecture of the network.
- The goal of such an approach would be to reduce the parameter requirements with increasing sequence length; recurrent neural networks provide an excellent example of (parameter-wise) frugal architectural design with the help of domain-specific insights.
- Thus, the two primary requirements for sequence processing are:
  - the capability to receive and handle inputs in the order they appear in the sequence, and
  - the consistent treatment of inputs at each time point, considering the previous input history.

A key challenge is that we somehow need to construct a neural network with a fixed number of parameters, but with the ability to process a variable number of inputs.

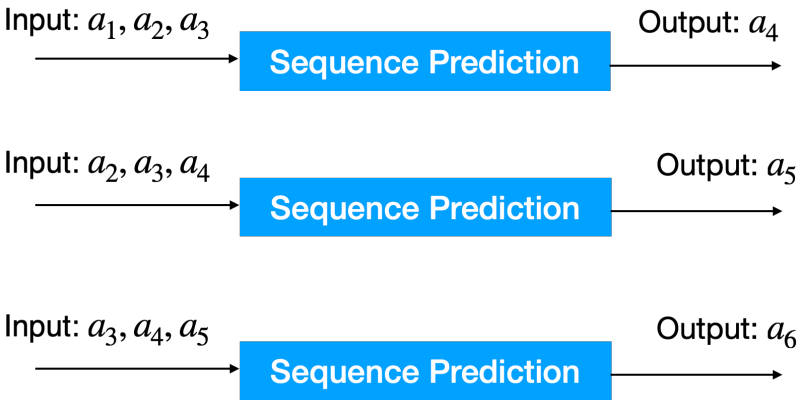
Types of Tasks in Sequence

Sequence Learning

- Sequence Prediction
- Sequence Classification
- Sequence Generation
- Sequence to Sequence

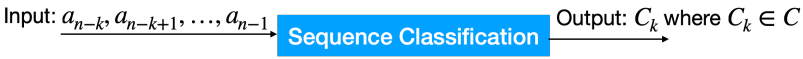
  ⋮

Sequence Prediction



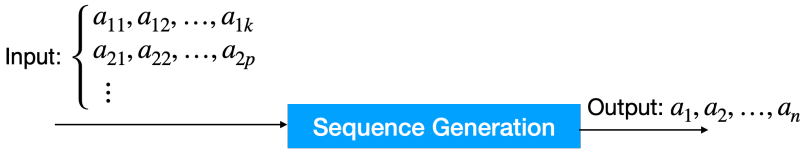
Sequence Prediction

Sequence Classification



Example:
Sentiment analysis where a pice of text is given as input and model has to label or classify the text as positive, negative or neutral. Here C $= \{$Positive, Negative or Neutral$\}$

Sequence Generation

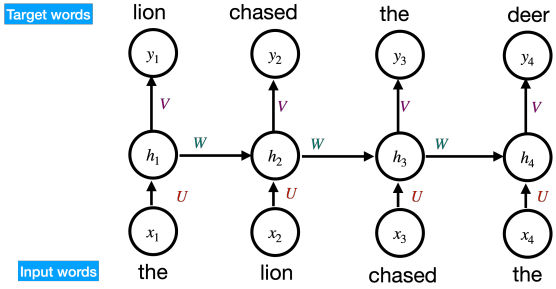


Example:
When a model is trained with the music of some genre the model will generate new music with the same genre.

Sequence to Sequence

Example:
Translation where is sequence of sentence is saying in English and this needs to be converted to French.

Architecture of RNN

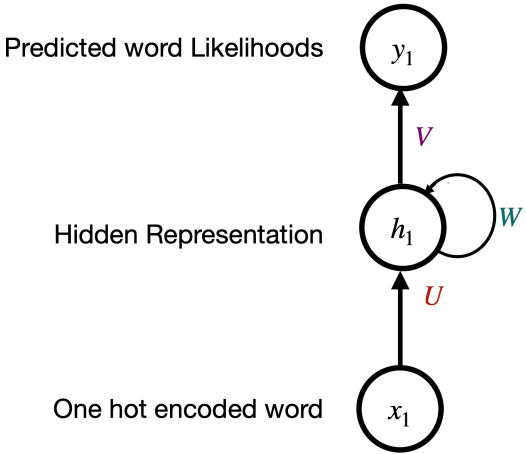


The weight matrices in different temporal layers are shared to ensure that the same function is used at each time stamp.

Architecture of RNN

In general,

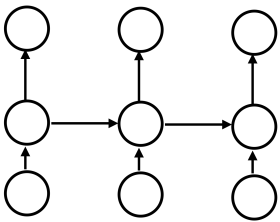
- the input vector at time t (e.g., one-hot encoded vector of the t^{th} word) is x_t
- the hidden state at time t is h_t
- the output vector at time t (e.g., predicted probabilities of the $(t+1)^{\text{th}}$ word) is y_t.



- Note that each time stamp had an input, output and hidden unit.
- In practice, it is possible for either the input or the output units to be missing at any particular time-stamp.

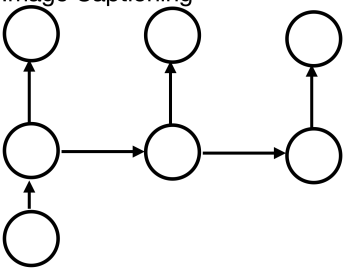
No missing inputs or outputs
Example:
Forecasting, Language Models

(many to many)

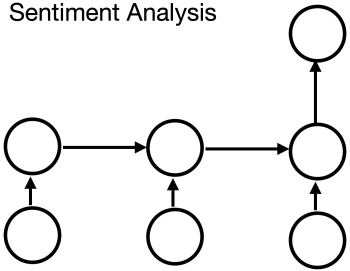


(one to one)

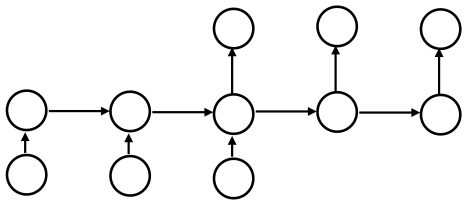
Missing inputs
(one to many)
Example:
Image Captioning



Missing outputs
(many to one)
Example:
Sentiment Analysis



Missing inputs and outputs.
Example:
Translation



Variations of RNN

- Both x_t and y_t are d-dimensional for a lexicon of size d.
- The hidden vector h_t is p-dimensional, where$p$ regulates the complexity of the embedding.
- In many applications like classification, the output is not produced at each time unit but is only triggered at the last time-stamp in the end of the sentence.

Although output and input units may be present only at a subset of the time-stamps, we examine the simple case in which they are present in all time-stamps.

- The hidden state at time t is given by a function of the input vector at time t and the hidden vector at time $(t-1)$:

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

- This function is defined with the use of weight matrices and activation functions and the same weights are used at each time-stamp.
- Therefore, even though the hidden state evolves over time, the weights and the underlying function $f(\cdot, \cdot)$ remain fixed over all time-stamps (i.e., sequential elements) after the neural network has been trained.
- A separate function $y_t = g(h_t)$ is used to learn the output probabilities from the hidden states.

More precisely, at any time point t, we have

$$h_t = f(Ux_t + Wh_{t-1})$$
$$y_t = g(Vh_t)$$

where, f, g are the activation functions in the hidden and output layers, respectively.

Remark

- Although the hidden states change at each time stamp, the weight matrices stay fixed over the various time-stamps.
- Note that the output vector y_t is a set of continuous values with the same dimensionality as the lexicon.
- A softmax layer is applied on top of y_t so that the results can be interpreted as probabilities.

Applications

Most of the applications of RNNs fall into one of two categories:

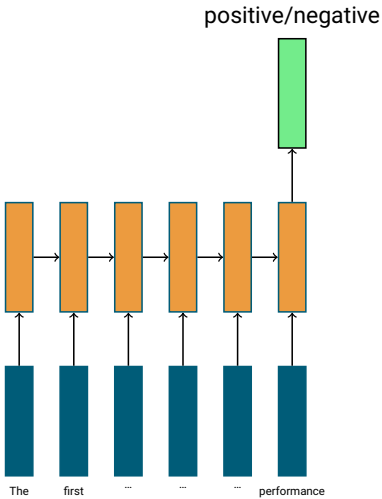
- Conditional language modeling
- Leveraging token specific outputs

Long Short Term Memory(LSTM)

Consider the task of predicting the sentiment (positive/negative) of a review:

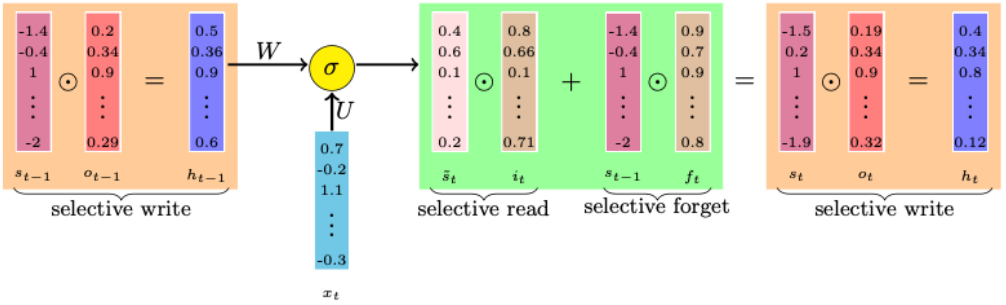
The first half of the movie was dry but the second half really picked up pace. The lead actor delivered an amazing performance

By the time we reach the end of the document the information obtained from the first few words is completely lost



Ideally we want to

- forget the information added by stop words (a, the, etc.)
- selectively read the information added by previous sentiment bearing words (awesome, amazing, etc.)
- selectively write new information from the current word to the state



Gates:

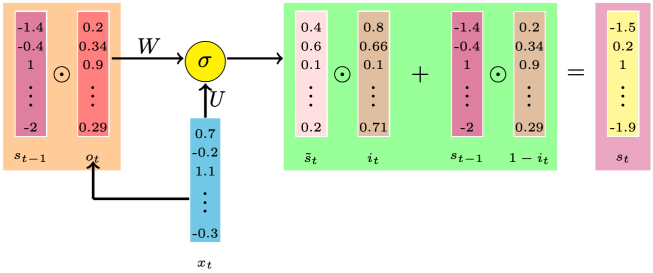
$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$
$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$
$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

States:

$$\tilde{s_t} = \sigma(W h_{t-1} + U x_t + b)$$
$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s_t}$$
$$h_t = o_t \odot \sigma(s_t) \text{ and } rnn_{out} \quad = h_t$$

- LSTM has many variants which include different number of gates and also different arrangement of gates
- The one which we just saw is one of the most popular variants of LSTM
- Another equally popular variant of LSTM is Gated Recurrent Unit which we will see next

GRU



The full set of equations for GRUs

Gates:

$$o_t = \sigma(W_o s_{t-1} + U_o x_t + b_o)$$
$$i_t = \sigma(W_i s_{t-1} + U_i x_t + b_i)$$

States:

$$\tilde{s_t} = \sigma(W(o_t \odot s_{t-1}) + U x_t + b)$$
$$s_t = (1 - i_t) \odot s_{t-1} + i_t \odot \tilde{s_t}$$

- No explicit forget gate (the forget gate and input gates are tied)
- The gates depend directly on s_{t-1} and not the intermediate h_{t-1} as in the case of LSTMs

How LSTMs avoid the problem of vanishing gradients

- During forward propagation the gates control the flow of information
- They prevent any irrelevant information from being written to the state
- Similarly during backward propagation they control the flow of gradients
- It is easy to see that during backward pass the gradients will get multiplied by the gate

- If the state at time $t-1$ did not contribute much to the state at time t then during backpropagation the gradients flowing into s_{t-1} will vanish.
- But this kind of a vanishing gradient is fine (since s_{t-1} did not contribute to st we don't want to hold it responsible for the crimes of s_t)
- The key difference from vanilla RNNs is that the flow of information and gradients is controlled by the gates which ensure that the gradients vanish only when they should (i.e., when s_{t-1} didn't contribute much to s_t) [Refer[1] for more details]

[1] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *30th International Conference on Machine Learning, ICML 2013*. 2013.

Notations

$$s_t = \sigma(U\ x_t + W s_{t-1} + b)$$

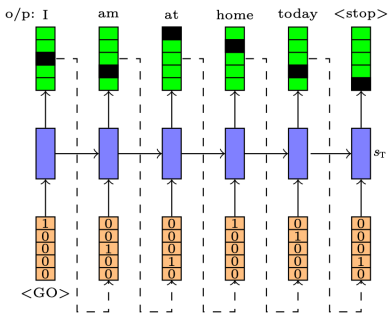
$$\tilde{s}_t = \sigma(W(o_t \odot\ s_{t-1}) + U x_t + b)$$
$$s_t = i_t \odot\ s_{t-1} + (1 - i_t) \odot \tilde{s}_t$$

$$\tilde{s}_t = \sigma(W\ h_{t-1} + U x_t + b)$$
$$s_t = f_t \odot\ s_{t-1} + i_t \odot \tilde{s}_t$$
$$h_t = o_t \odot \sigma(s_t)$$

$$s_t = \text{RNN}(\ s_{t-1}, x_t)$$

$$s_t = \text{GRU}(\ s_{t-1}, x_t)$$

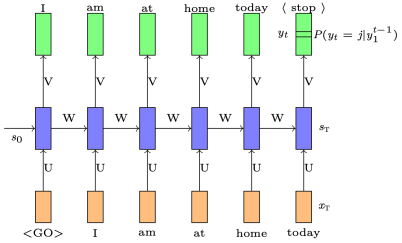
$$h_t, s_t = \text{LSTM}(\ h_{t-1}, s_{t-1}, x_t)$$



- So far, our RNN was able to take a sequence and return a sequence.
- The input at each step is the word that we predicted at the previous time step.
  In general,
  $$s_t = RNN(s_{t-1}, x_t)$$
- Let j be the index of the word which has been assigned the max probability at time step $t-1$

  $$x_t = e(v_j),$$

  where $e(v_j)$ is a one-hot vector representing jth word in the vocabulary. (In practice, instead of one hot representation we use a pre-trained word embedding of the jth word)



Model:

$$s_t = \sigma\left(Ws_{t-1} + Ux_t + b\right)$$

$$P\left(y_t = j \middle| y_1, \ldots, y_{t-1}\right) = softmax(Vs_t + c)_j$$

Parameters: U, V, W, b, c

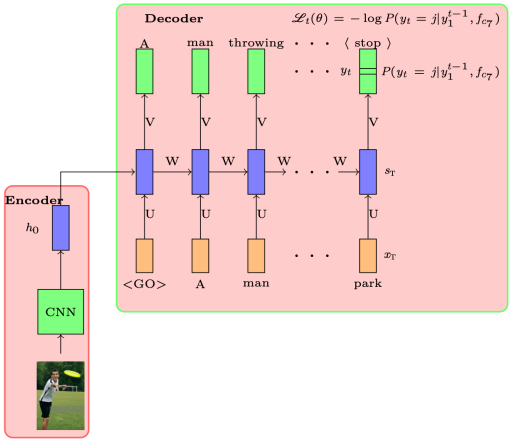
Loss:

$$L(U, V, W, b, c) = \sum_{t=1}^{T} L_t(U, V, W, b, c)$$

$$L_t(U, V, W, b, c) = -\log P(y_t = l_t | y_1, \ldots, y_{t-1})$$

where l_t is the true word at time step t.

Encoder-Decoder Architecture



- **Task:** Image captioning
- **Data:** $\{x_i = image_i,\ y_i = caption_i\}_{i=1}^{N}$
- **Model:**
  - **Encoder:**

$$s_0 = CNN(x_i)$$

  - **Decoder:**

$$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$

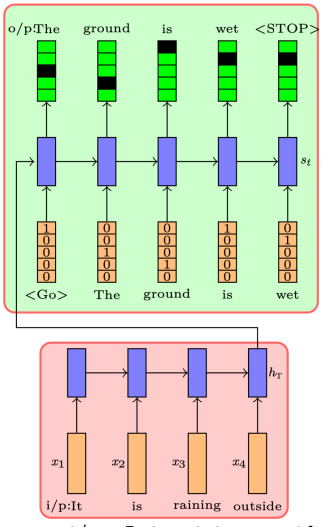
$$P(y_t|y_1^{t-1}, I) = softmax(Vs_t + b)$$

- **Parameters:** $U_{dec},\ V,\ W_{dec},\ W_{conv},\ b$
- **Loss:**

$$\mathscr{L}(\theta) = \sum_{t=1}^{T} \mathscr{L}_t(\theta) \quad = -\sum_{t=1}^{T} \log P(y_t = \ell_t|y_1^{t-1}, I)$$

- **Algorithm:** Gradient descent with backpropagation

o/p : The ground is wet



i/p : It is raining outside

- **Task:** Textual entailment
- **Data:** $\{x_i = premise_i,\ y_i = hypothesis_i\}_{i=1}^{N}$
- **Model (Option 1):**
  - **Encoder:**

$$h_t = RNN(h_{t-1}, x_{it})$$

  - **Decoder:**

$$s_0 = h_T \quad (T \text{ is length of input})$$

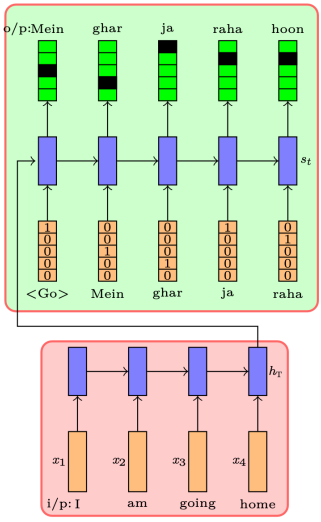
$$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$

$$P(y_t | y_1^{t-1}, x) = softmax(Vs_t + b)$$

- **Parameters:** $U_{dec},\ V,\ W_{dec},\ U_{enc},\ W_{enc}, b$
- **Loss:**

$$\mathscr{L}(\theta) = \sum_{i=1}^{T} \mathscr{L}_t(\theta) = -\sum_{t=1}^{T} \log P(y_t = \ell_t | y_1^{t-1}, x)$$

o/p : Mein ghar ja raha hoon



i/p : I am going home

- **Task:** Machine translation
- **Data:** $\{x_i = source_i,\ y_i = target_i\}_{i=1}^{N}$
- **Model (Option 1):**
  - **Encoder:**

$$h_t = RNN(h_{t-1}, x_{it})$$

  - **Decoder:**

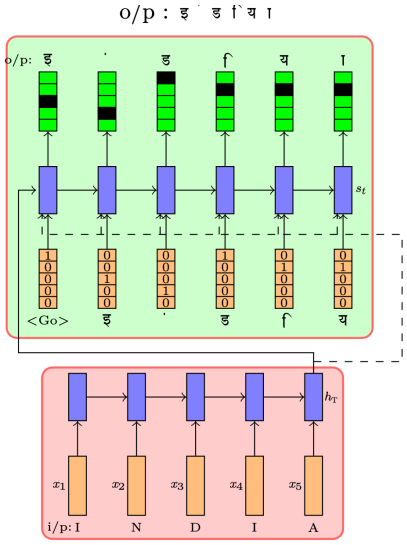
$$s_0 = h_T \quad (T \text{ is length of input})$$

$$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$

$$P(y_t | y_1^{t-1}, x) = softmax(Vs_t + b)$$

- **Parameters:** $U_{dec},\ V,\ W_{dec},\ U_{enc},\ W_{enc},\ b$
- **Loss:**

$$\mathscr{L}(\theta) = \sum_{i=1}^{T} \mathscr{L}_t(\theta) = -\sum_{t=1}^{T} \log P(y_t = \ell_t | y_1^{t-1}, x)$$



- **Task:** Transliteration
- **Data:** $\{x_i = srcword_i,\ y_i = tgtword_i\}_{i=1}^{N}$
- **Model (Option 2):**
  - **Encoder:**

$$h_t = RNN(h_{t-1}, x_{it})$$

  - **Decoder:**

$$s_0 = h_T \quad (T \text{ is length of input})$$

$$s_t = RNN(s_{t-1}, [e(\hat{y}_{t-1}), h_T])$$

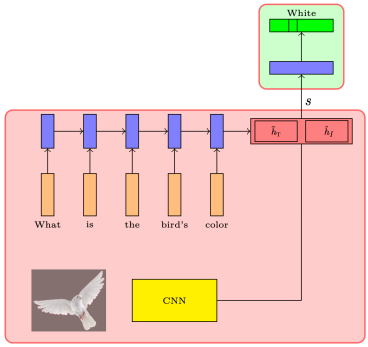
$$P(y_t | y_1^{t-1}, x) = softmax(Vs_t + b)$$

- **Parameters:** $U_{dec},\ V,\ W_{dec},\ U_{enc},\ W_{enc}, b$
- **Loss:**

$$\mathscr{L}(\theta) = \sum_{i=1}^{T} \mathscr{L}_t(\theta) = -\sum_{t=1}^{T} \log P(y_t = \ell_t | y_1^{t-1}, x)$$

- **Algorithm:** Gradient descent with backpropagation

O/p: White



Question: What is the bird's color

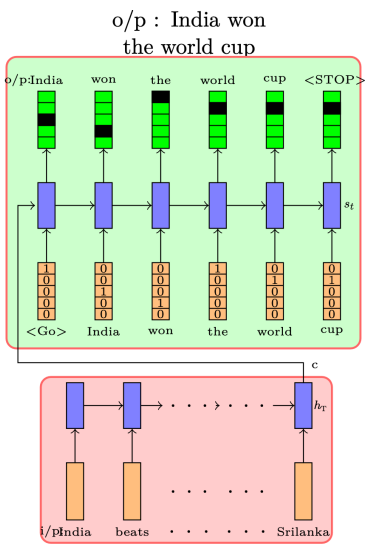
- **Task:** Image Question Answering
- **Data:** $\{x_i = \{I, q\}_i,\ y_i = Answer_i\}_{i=1}^{N}$
- **Model:**
  - **Encoder:**

    $$\hat{h}_I = CNN(I),\ \tilde{h}_t = RNN(\tilde{h}_{t-1}, q_{it})$$

    $$s = [\tilde{h}_T; \hat{h}_I]$$
  - **Decoder:**

    $$P(y|q, I) = softmax(Vs + b)$$
- **Parameters:** $V,\ b,\ U_q,\ W_q,\ W_{conv}, b$
- **Loss:**

  $$\mathscr{L}(\theta) = -\log P(y = \ell | I, q)$$
- **Algorithm:** Gradient descent with backpropagation



i/p : India beats Srilanka to win ICC WC 2011. Dhoni and Gambhir's half centuries help beat SL

- **Task:** Document Summarization
- **Data:** $\{x_i = Document_i, \; y_i = Summary_i\}_{i=1}^{N}$
- **Model:**
  - **Encoder:**

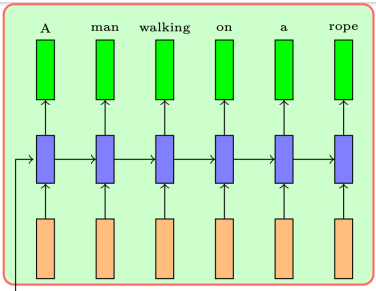
$$h_t = RNN(h_{t-1}, x_{it})$$

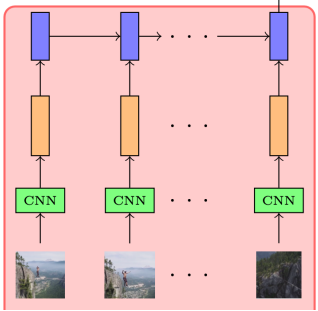
  - **Decoder:**

$$s_0 = h_T$$
$$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$
$$P(y_t | y_1^{t-1}, x) = softmax(Vs_t + b)$$

- **Parameters:** $U_{dec}, \; V, \; W_{dec}, \; U_{enc}, \; W_{enc}, b$
- **Loss:**

$$\mathscr{L}(\theta) = \sum_{i=1}^{T} \mathscr{L}_t(\theta) = -\sum_{t=1}^{T} \log P(y_t = \ell_t | y_1^{t-1}, x)$$





- **Task:** Video Captioning
- **Data:** $\{x_i = video_i,\ y_i = desc_i\}_{i=1}^{N}$
- **Model:**
  - **Encoder:**

$$h_t = RNN(h_{t-1}, CNN(x_{it}))$$

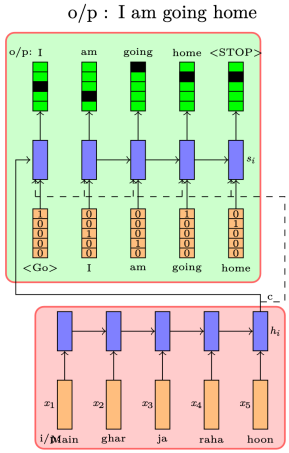
  - **Decoder:**

$$s_0 = h_T$$
$$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$
$$P(y_t|y_1^{t-1}, x) = softmax(Vs_t + b)$$

- **Parameters:** U_{dec}, W_{dec}, V, b, W_{conv}, U_{enc}, W_{enc}, b
- **Loss:**

$$\mathscr{L}(\theta) = \sum_{i=1}^{T} \mathscr{L}_t(\theta) = -\sum_{t=1}^{T} \log P(y_t = \ell_t|y_1^{t-1}, x)$$

Attention Mechanism



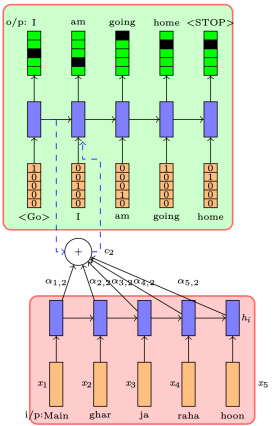
Note that, the input from the encoder is passed for every timestep in the decoder.

While this may be helpful, this can also become too much information for the decorder. In other words, the decoder may not understand for which portion of the sentence the importance should be given to decode the current timestep output.

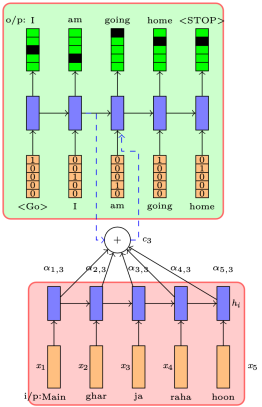
Can we do better?

How about feeding the encoder to the decoder this way?

At $t = 1$

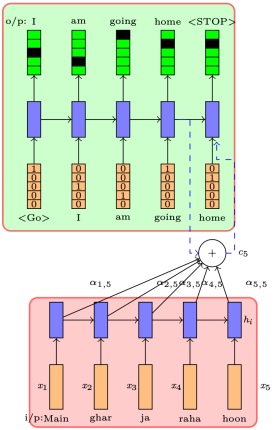


At $t = 2$



...

At $t = 4$



Note that each time, only some fraction of the encoder output is been fed into the decoder

- What we have tried now is called **attention mechanism**.
- Of course, we have to learn those fractions during the model training.
- To enable this, we define

$$e_{jt} = f_{att}(s_{t-1}, c_j)$$

- This quantity captures the importance of the jth input word for decoding the tth output word (we will see the exact form of f_{att} later)
- We can normalize these weights by using the softmax function.
- From now on we will refer to the decoder RNN's state at the t-th timestep as st and the encoder RNN's state at the j-th time step as c_j
- Since, we have to learn the e_{jt}, we write that in parametric form:

$$e_{jt} = V_{att}^T \tanh(U_{att} s_{t-1} + W_{att} c_j)$$

Can we check if the attention model actually learns something meaningful ?

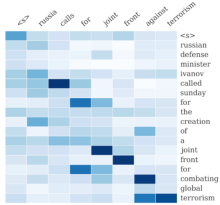


Figure: Example output of attention-based summarization system [Rush et al. 2015.]

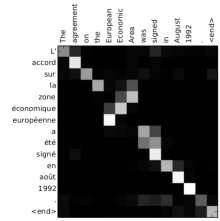
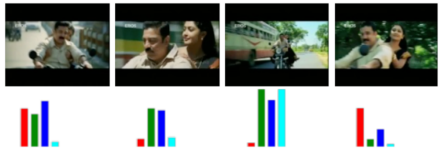


Figure: Example output of attention-based neural machine translation model [Cho et al. 2015].

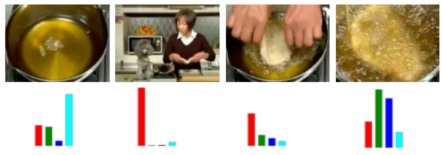
- The heat map shows a soft alignment between the input and the generated output.
- Each cell in the heat mapsssss corresponds to α_{tj} (i.e., the importance of the jth input word for predicting the tth output word as determined by the model)

Example output of attention-based video captioning system [Yao et al. 2015.]



+Local+Global: A man and a woman are talking on the road

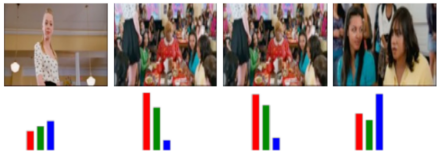
Ref: A man and a woman ride a motorcycle



+Local+Global: Someone is frying a fish in a pot

+Local: Someone is frying something
+Global: The person is cooking
Basic: A man cooking its kitchen

Ref: A woman is frying food



+Local+Global: the girl grins at him

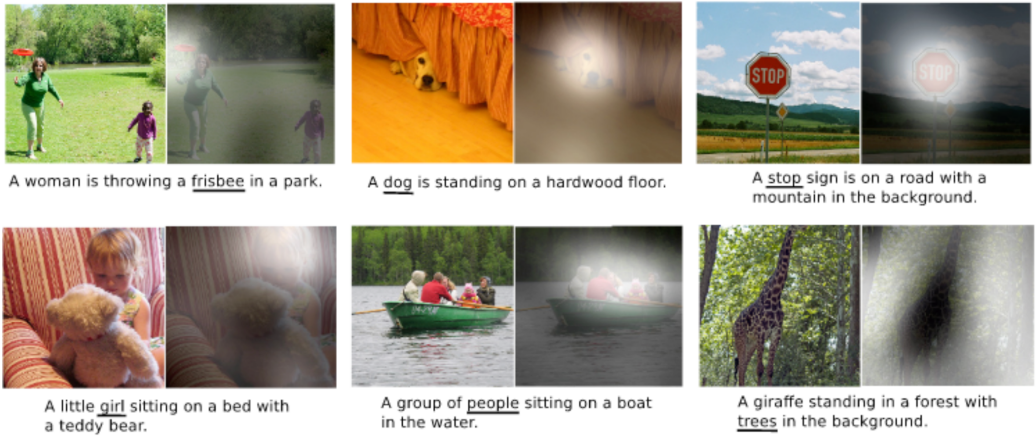
Ref: SOMEONE and SOMEONE swap a look

+Local+Global: as SOMEONE sits on the table, SOMEONE shifts his gaze to SOMEONE

+Local: with a smile SOMEONE arrives
+Global: SOMEONE sits at a table
Basic: now, SOMEONE grins

Ref: SOMEONE gaze at SOMEONE

Attention over Images



Examples of the attention-based model attending to the correct object (white indicates the attended regions,underlines indicates the corresponding word) [Kyunghyun Cho et al., 2015]

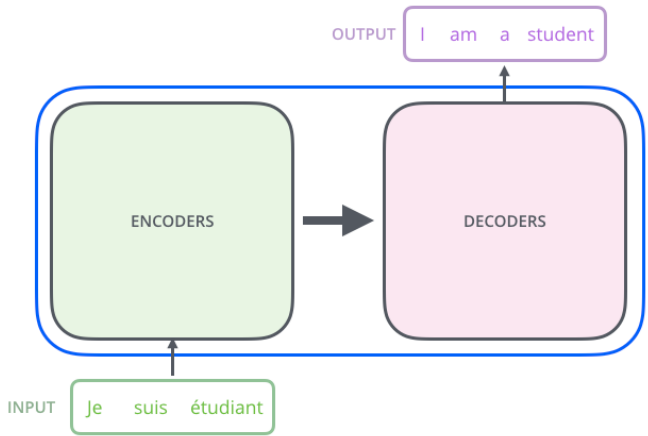
Types of Attention in Transformer

- **Self-Attention (Encoder)**: All Q,K,V from encoder.
- **Self-Attention (Decoder)**: Masked (no future peeking).
- **Encoder-Decoder**: Q from decoder, K,V from encoder.

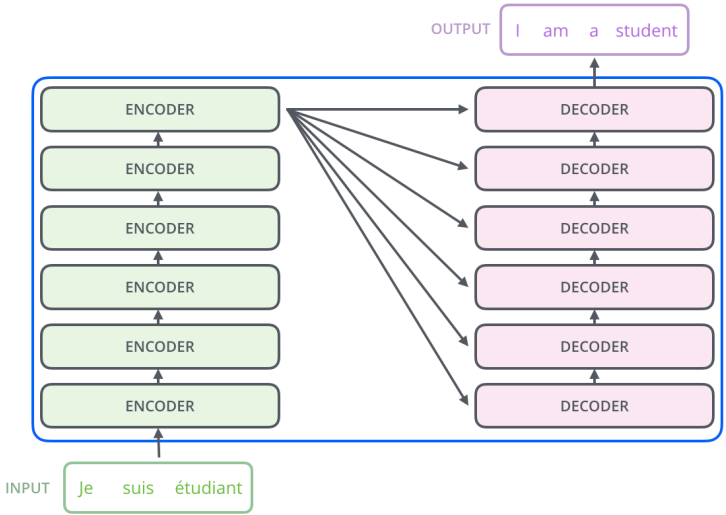
Summary

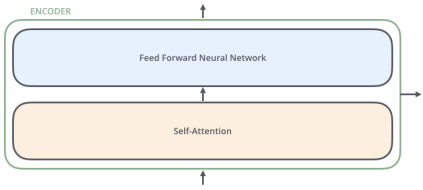

INPUT
Je suis étudiant
THE TRANSFORMER
OUTPUT
I am a student

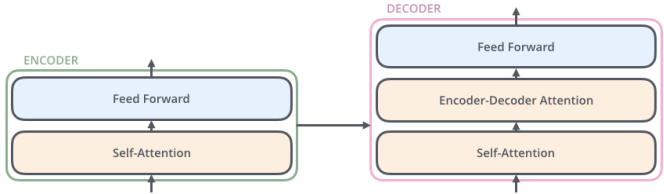
Transformers contains an encoding component, a decoding component, and connections between them.



The encoding component is a stack of encoders. The decoding component is a stack of decoders of the same number.

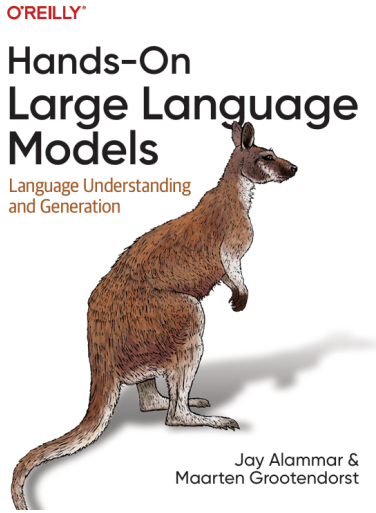



ENCODER
Feed Forward Neural Network
Self-Attention



- The encoder's inputs first flow through a self-attention layer – a layer that helps the encoder look at other words in the input sentence as it encodes a specific word. We'll look closer at self-attention later in the post.
- The outputs of the self-attention layer are fed to a feed-forward neural network. The exact same feed-forward network is independently applied to each position.
- The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence

Read More...


O'REILLY
Hands-On
Large Language
Models
Language Understanding
and Generation
Jay Alammar &
Maarten Grootendorst

References I

[1] Jay Alammar. *The Illustrated Transformer*. 2018. URL: https://jalammar.github.io/illustrated-transformer/ (visited on 02/13/2026).

[2] Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[4] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *30th International Conference on Machine Learning, ICML 2013*. 2013.