Attention

Attention

1. Motivation
2. Innerworkings
3. Application

Motivation

1. Recurrent Neural Network (RNN)
2. Translation Bottleneck

Recurrent Neural Network

First, we have to understand the problem of Language Modeling. It is the task of predicting the next words:

Example:

What could be the word?

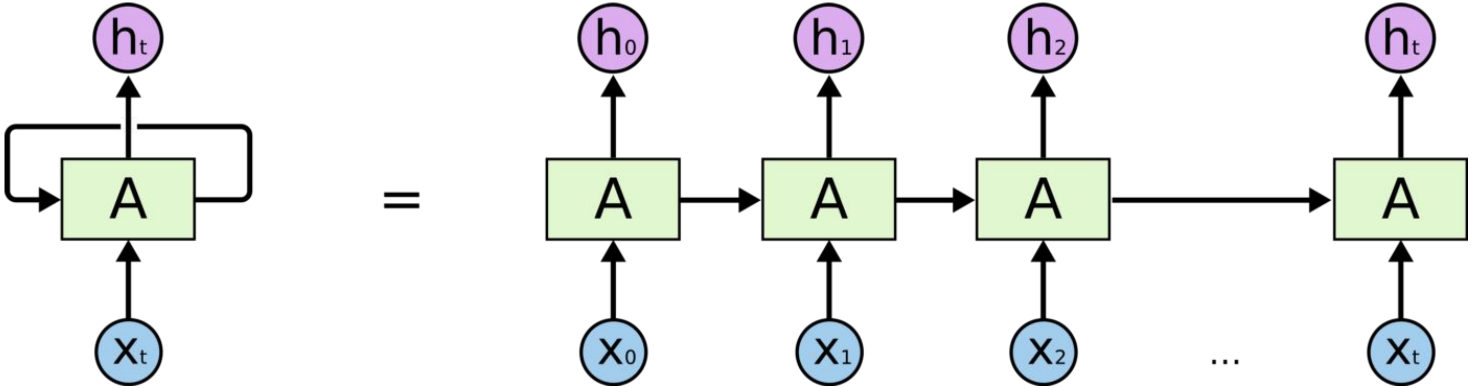
Students open their ____

Recurrent Neural Network

To predict the word, we have to use the context around it. However, what if the context is way far before it? How can we define a model that can detect the context in general case?

Recurrent Neural Network

Recurrent Neural Network



Long short-term memory (LSTM)

There is still a problem as in some long sentence, if we multiply the matrix A many times, then the information from hidden states that are far away will be vanished.

Vanishing Gradient

LSTM is a modified version of RNN that allows us to combat against vanishing gradient.

RNN Application: **Machine Translation**

The task of translating a sentence from one language to a sentence in another language.

x: *L'homme est né libre, et partout il est dans les fers*

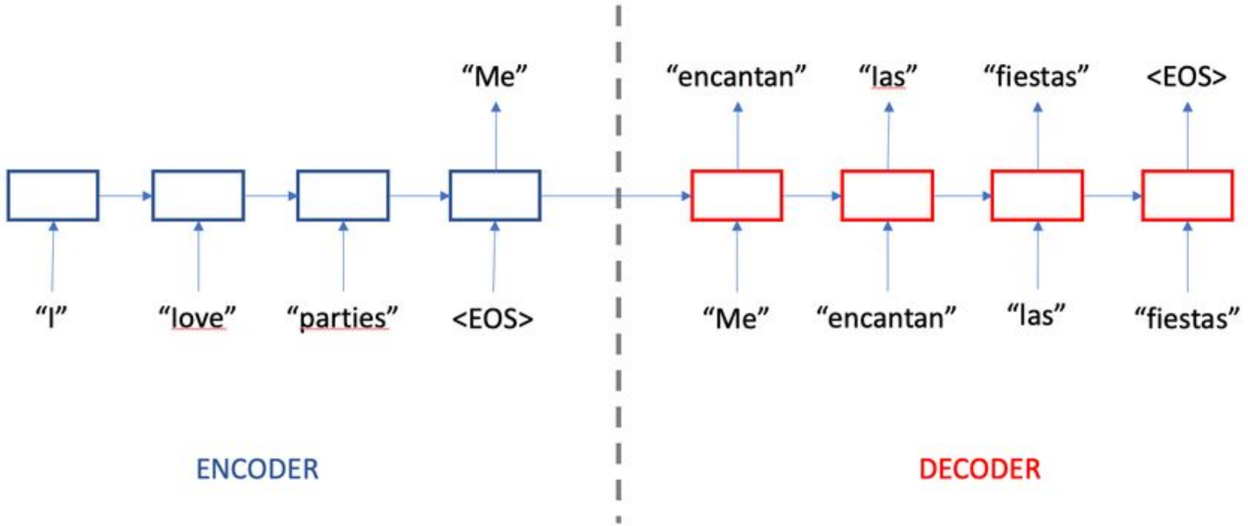
↓

y: *Man is born free, but everywhere he is in chains*

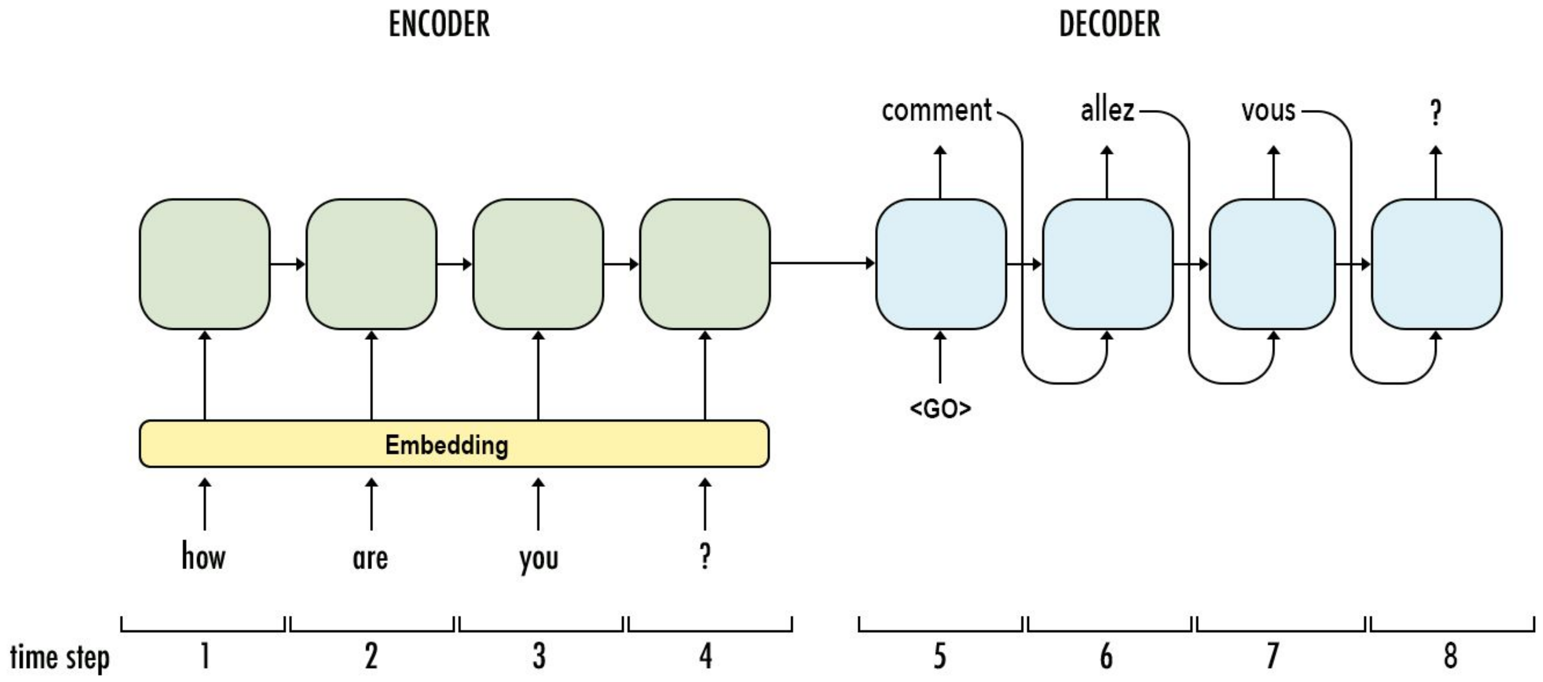
\- Rousseau

Neural Machine Translation (NMT)

- A single neural network
- Architecture: Seq2Seq
- 2 RNNs:
  - Encoder
  - Decoder



Encoder and Decoder RNNs

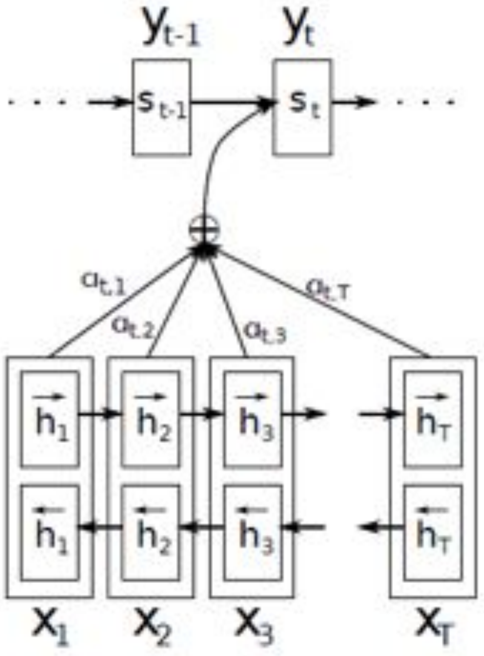


Translation Bottleneck

- No connection between the lengths of the input and output.
- Ineffectiveness on very long inputs.
- Different levels of significance of words.

Solution: **Attention**

- The decoder network looks at the **entire input** sequence at every decoding step.
- Decide what input words are **important.**



Why Attention?

- Improves NMT performance.
- Solves the bottleneck problem.

Inner Workings of Attention Model

1. Goals
2. Process
3. Additive and Multiplicative Attention
4. Implementation with Pytorch

The goal of Attention

The goal of Attention is to focus on certain words in the encoder to output the correct word in the decoder.

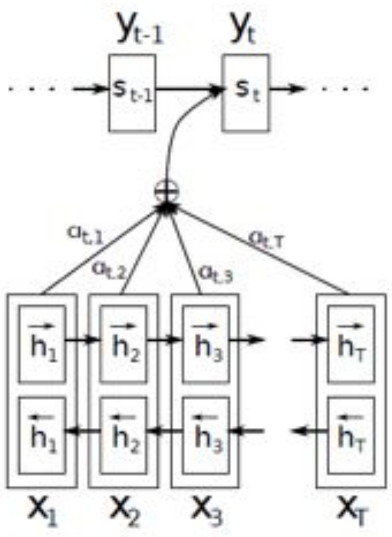
French: il a m'entarté ⇒ English: He hit me with a pie.

The Process of Attention Model

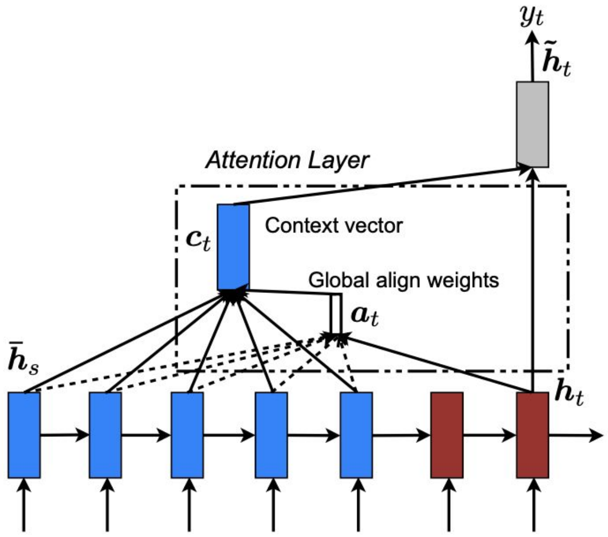
There are **3 main steps** of Attention Model:

- Calculate the context vector (based on the **<u>score</u>** of hidden state h_j of the encoder) of the current state s_i of the decoder.
- Update the current hidden state of the decoder based on the context vector, previous state, etc.
- Return a word based on the state s_i.

Attention Visualization



Additive Attention



Multiplicative Attention

Variations of Attention

Additive Attention

$$\boldsymbol{e}_i = \boldsymbol{v}^T \tanh(\boldsymbol{W}_1 \boldsymbol{h}_i + \boldsymbol{W}_2 \boldsymbol{s})$$

Multiplicative Attention

$$\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{W} \boldsymbol{h}_i$$

Additive Attention Implementation with Pytorch

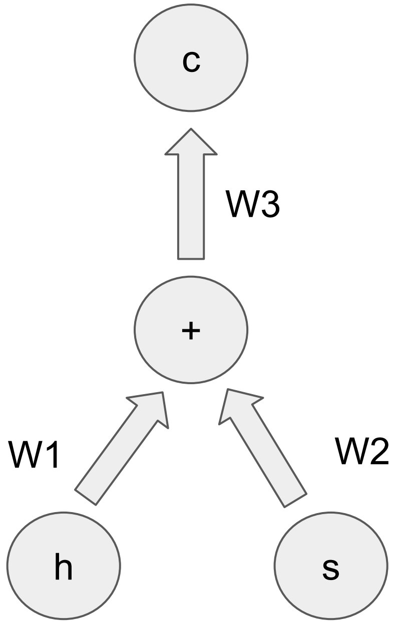
class Attention(nn.Module):

- __init__(self, dim_h, dim_s, dim_c)
- forward(self, hidden_encodes, hidden_decode)

Initialize Model

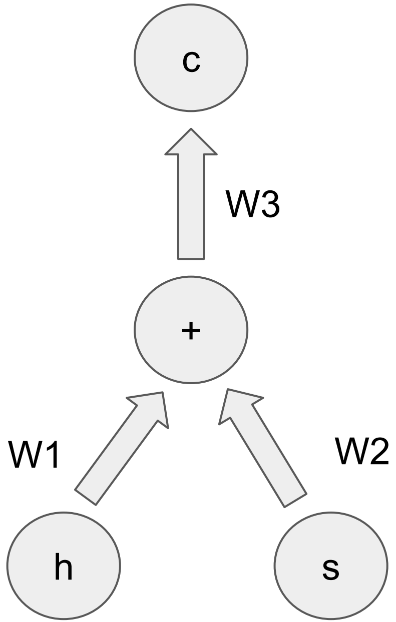
__init__(self, dim_h, dim_s, dim_c):

- W1 = Linear Layer (dim_h, dim_s)
- W2 = Linear Layer (dim_s, dim_c)
- W3 = Linear Layer (dim_c, 1)
- a_ij = Softmax Layer



forward(hidden_encodes, hidden_decode)

- Initialize W1 with hidden_encodes
- Initialize W2 with hidden_decode
- Combine W1 and W2
- Get score values
- Calculate softmax values of scores
- Multiply by the hidden states in the encoder



```python
def __init__(self, dim_h, dim_s, dim_c):
    """
    dim_h: the number of features of each hidden layer of the encoder
    dim_s: the number of features of each hidden layer of the decoder
    dim_c: the number of features of the output from the combination of
           the previous two vectors
    """
    super().__init__()

    self.dim_h = dim_h
    self.dim_s = dim_s
    self.dim_c = dim_c

    # The first layer deals with the matrix correspond to the hidden layers in the encoder
    self.w1 = nn.Linear(dim_h, dim_c, bias=False)

    # The second layer deals with the matrix correspond to the hidden layers in the decoder
    # Note that bias=True means it allows addition.
    self.w2 = nn.Linear(dim_s, dim_c, bias=True)

    # The third layer simply calculates the vector that converts the previous sum in to a vector
    # containing score of each pair of layers
    self.w3 = nn.Linear(dim_c, 1, bias=False)

    # The last layer just convert w3 into softmax values
    self.a_ij = nn.Softmax()
```

```python
def forward(self, hidden_encodes, hidden_decode):
    # Combine the term w1*encoders + w2*decoder
    comb = self.w1(hidden_encodes) + self.w2(hidden_decode)

    # Get the score values
    out = self.w3(comb)

    # Calculate the softmax value and multiply it by the hidden layers in the encoder
    context = torch.matmul(torch.transpose(self.a_ij(out), 0, 1), hidden_encodes)

    return context
```

Attention in Question and Answering (Q&A) Model

Why is attention necessary in Q&A?

Attention in Q&A

Attention is important for Question and Answer problem because it gives the models a way to focus on which part of a paragraph to produce correct answer.

Attention in Q&A

Document: What was supposed to be a fantasy sports car ride at Walt Disney World Speedway turned deadly when a Lamborghini crashed into a guardrail. The crash took place Sunday at the Exotic Driving Experience, which bills itself as a chance to drive your dream car on a racetrack. The Lamborghini's passenger, 36-year-old Gary Terry of Davenport, Florida, died at the scene, Florida Highway Patrol said. The driver of the Lamborghini, 24-year-old Tavon Watson of Kissimmee, Florida, lost control of the vehicle, the Highway Patrol said. (...)
Question: Officials say the driver, 24-year-old Tavon Watson, lost control of a _______
Answer candidates: Tavon Watson, Walt Disney World Speedway, Highway Patrol, Lamborghini, Florida, (...)
Answer: Lamborghini

Our Schedule

1. This week:

   Wednesday + Friday → Implement Attention in Named Entity Recognition

2. Sunday + Next week:

   Self Attention and Transformer