Applied Deep Learning for NLP

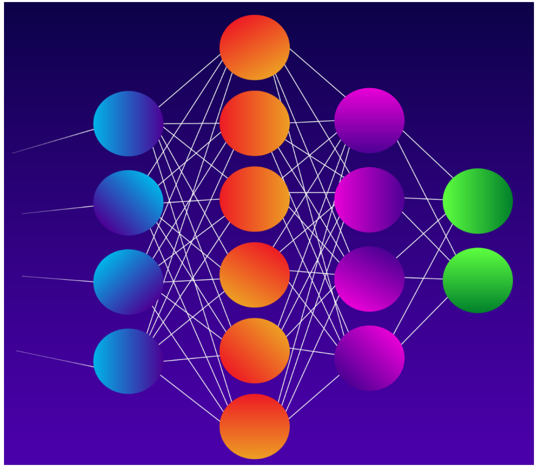
Week 5 - Seq2Seq and Attention

Juan Carlos Medina Serrano

Technische Universität München
Hochschule für Politik
Political Data Science

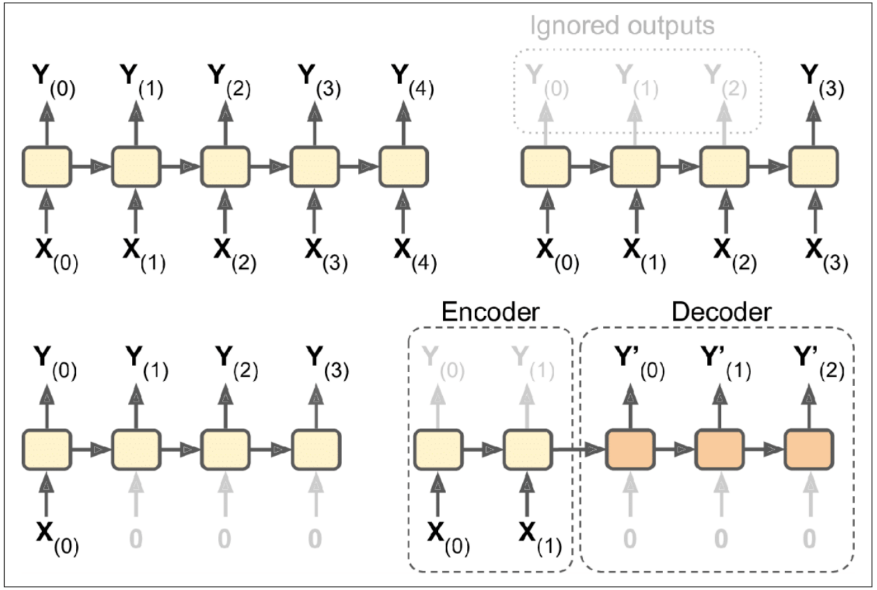
Munich, 19. November 2020

political data science



Seq2Seq

The seq2seq model is also called the Encoder-Decoder. The network is composed of an encoder and a decoder part, both RNNbased, and capable of consuming and returning sequences of outputs corresponding to multiple time steps.



Applications: Text summarization, machine translation, parsing

Task: Machine Translation

The task of translating a sentence x from one language (the source language) to a sentence y in another language (the target language).

Ich trinke Milch ——> Je bois du lait

Task: Machine Translation

How do we evaluate a translation model?

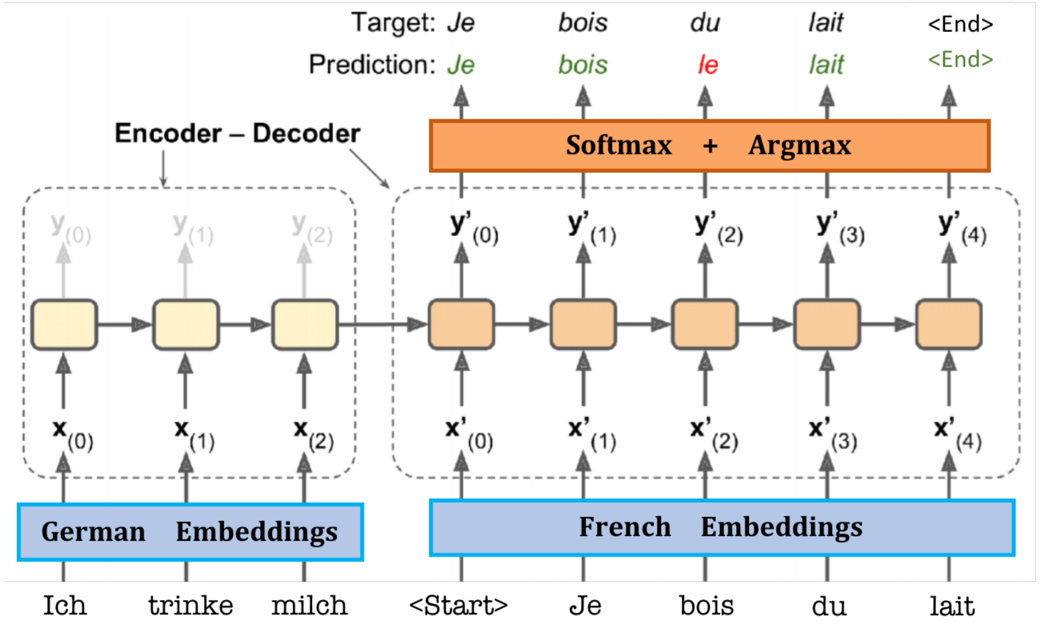
BLEU (Bilingual Evaluation Understudy)) compares the machine-written translation to one or several human-written translation(s), and computes a similarity score based on the **n-gram** precision with a brevity penalty factor added.

Le professeur est arrivé en retard à cause de la circulation.	(Source Original)
The teacher arrived late because of the traffic.	(Reference Translation)
The professor was delayed due to the congestion .	#1 Very low BLEU score
Congestion was responsible for the teacher being late	#2 Slightly higher but low BLEU
The teacher was late due to the traffic.	#3 Higher BLEU than #1 and #2
The professor arrived late because of circulation .	#4 Higher BLEU than #3
The teacher arrived late because of the traffic .	#5 ***Best BLEU Score***

BLEU is useful but imperfect

Neural Machine Translation

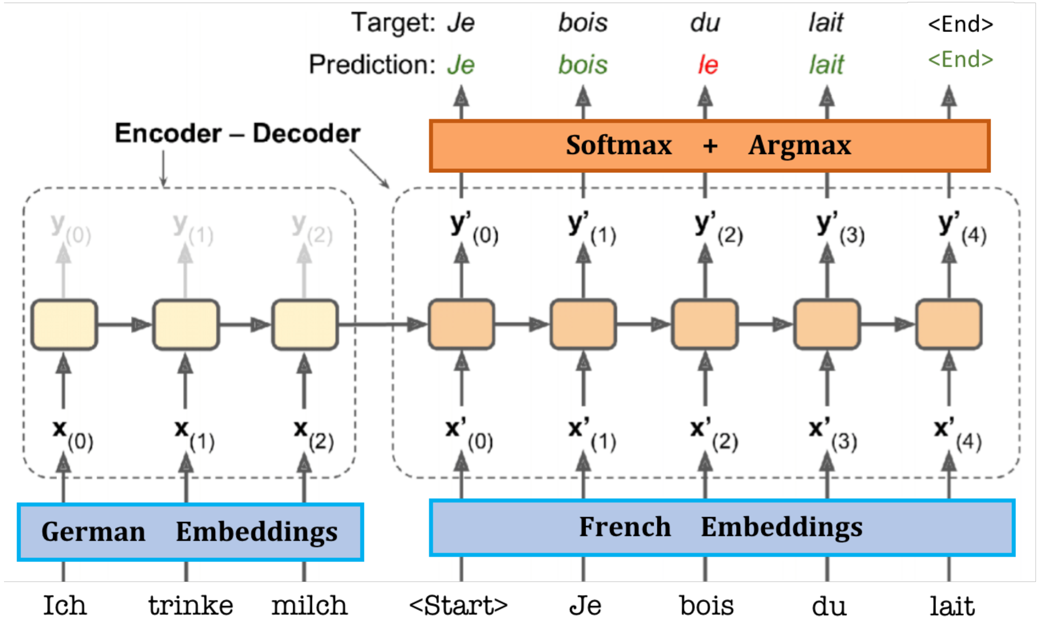
Using a big parallel corpus between two languages:



Seq2Seq and Attention

Neural Machine Translation

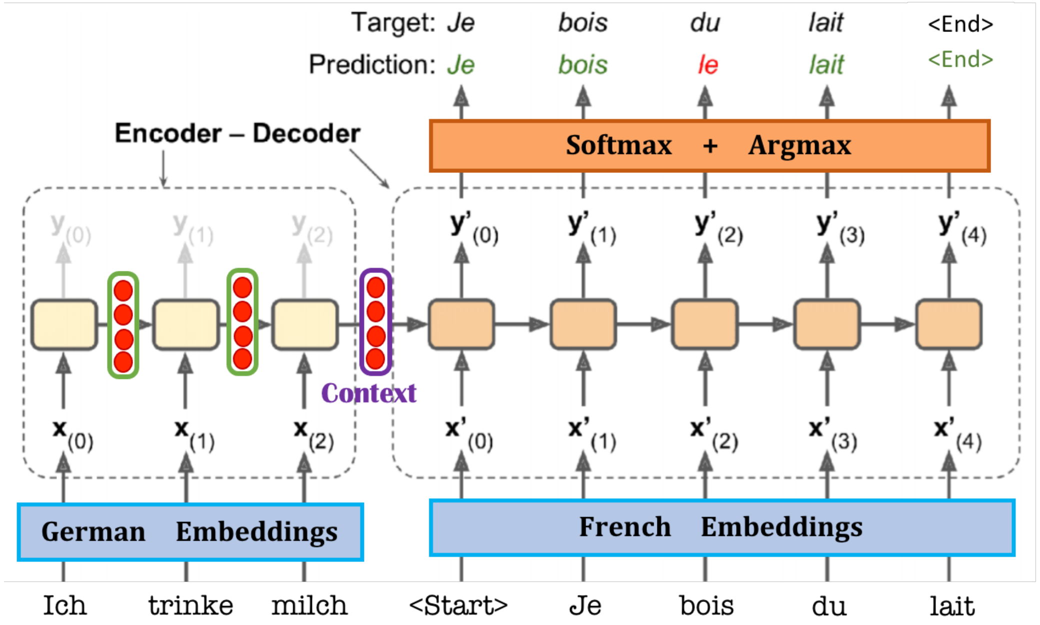
Using a big parallel corpus between two languages:



Teacher forcing: No matter what the Decoder outputs at each timestep, the teacher (us) provides the correct response as input to the Decoder in the next timestep

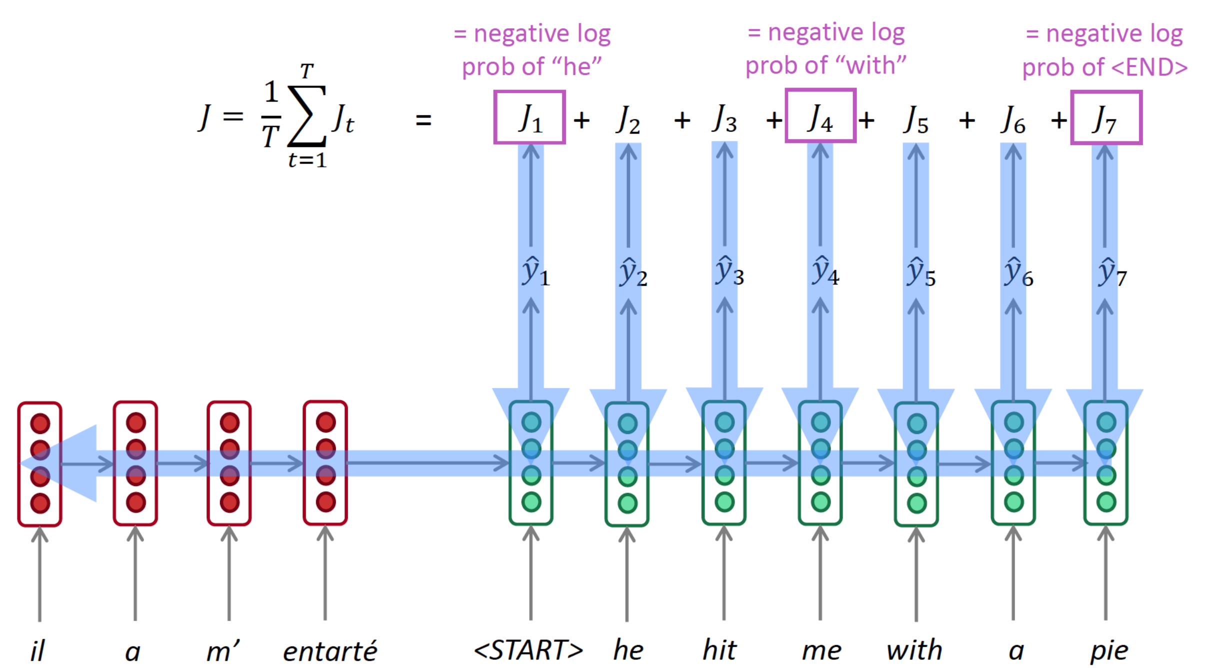
Neural Machine Translation

The Encoder sends the last hidden state to the decoder as input. This is called the context.



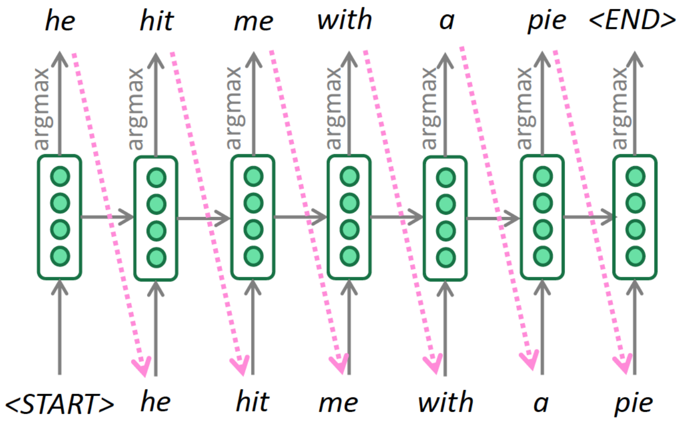
Training the Model

Seq2seq is an **end-to-end** system as backpropagation operates from end translation to the beginning sentence



Generating a Translation

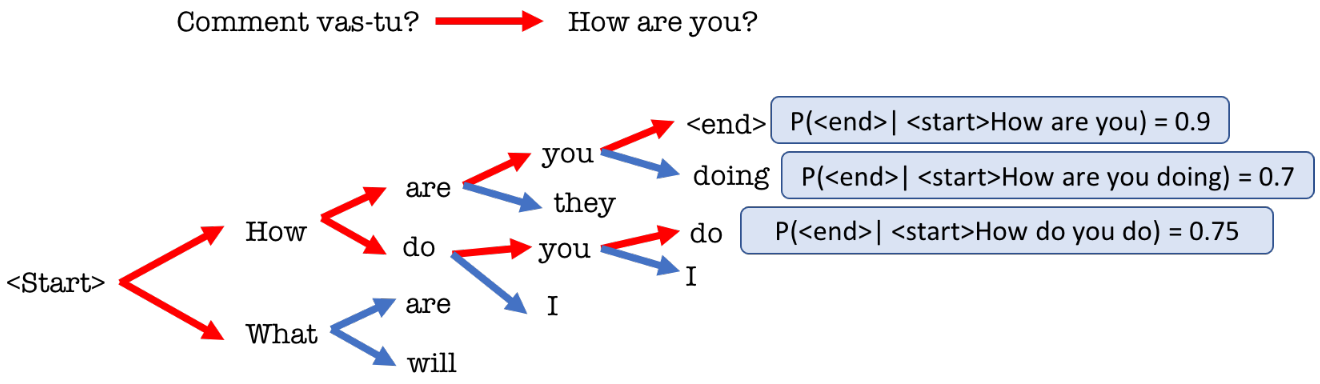
GREEDY translation: We take the word with the highest probability and feed it as input in the next timestep.



Note: This is different to text generation, where we sampled from the probability distribution (with or without temperature). Here we don't want creativity, we want the highest probability!!
Still not perfect. Why?

Generating a Translation

BEAM SEARCH translation: Keeps track of a short list of the k-most promising sentences. At each decoder step the sentences are extended by one word, keeping only the k most likely sentences (highest **conditional probabilities)**.
k is called the beam width (normally between 5 to 10). Example with k=2:



Different sentences may produce <END> tokens on different timesteps => Divide by number of timesteps to normalize (first take the logarithm of the probability and then divide)

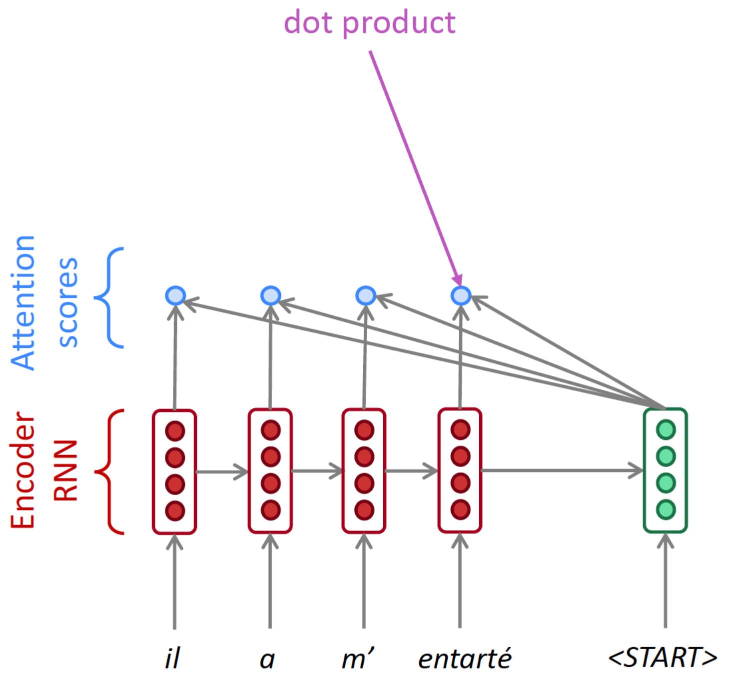
Attention

The context vector is the bottleneck of the Seq2Seq architecture. It has to capture the whole information of the sentence. Can we do better?

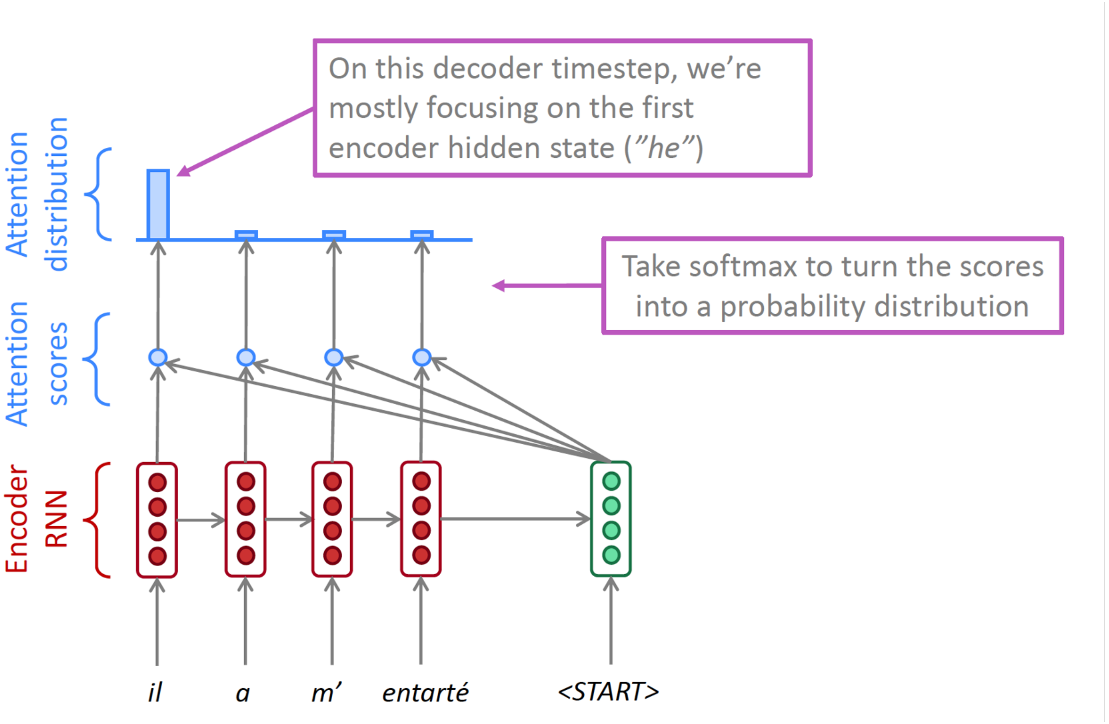
ATTENTION: On each step of the decoder, use **direct connection to the encoder** to focus on a particular part of the input sentence.

This simple idea has revolutionized Deep Learning!

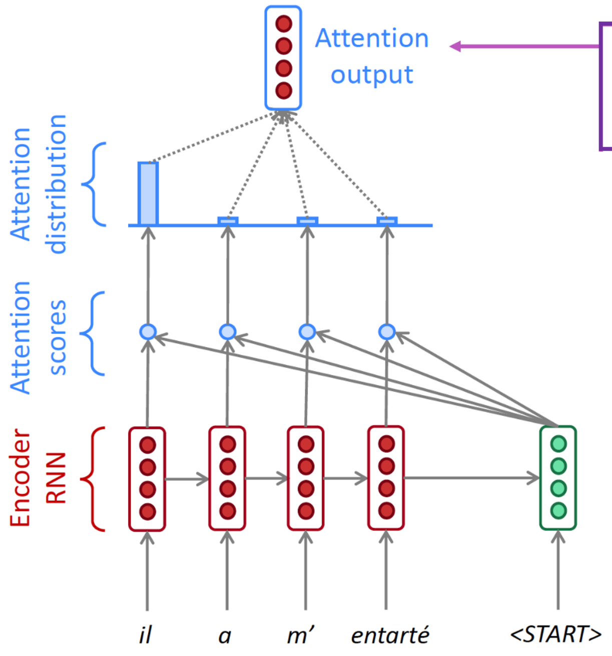
Attention


dot product
Attention scores
Encoder RNN
il
a
m'
entarté
<START>

Attention

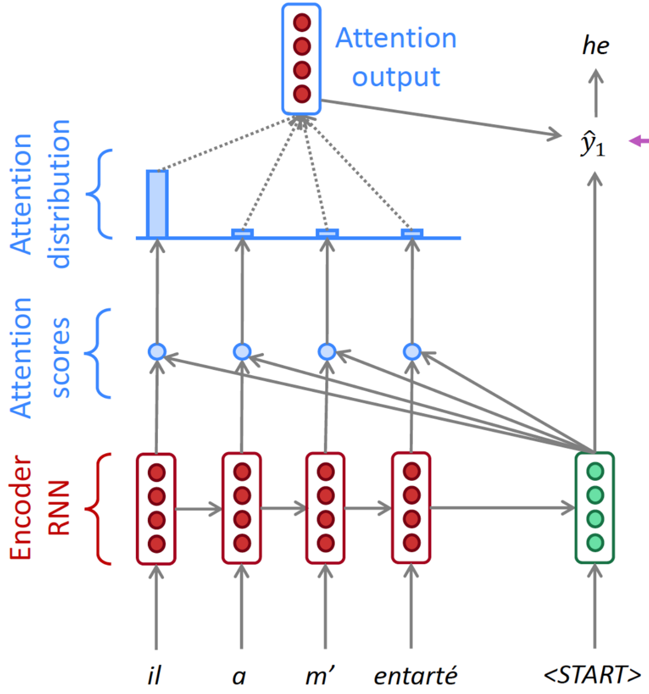

On this decoder timestep, we're mostly focusing on the first encoder hidden state ("*he*")
Take softmax to turn the scores into a probability distribution
Attention distribution
Attention scores
Encoder RNN
il
a
m'
entarté
<START>

Attention



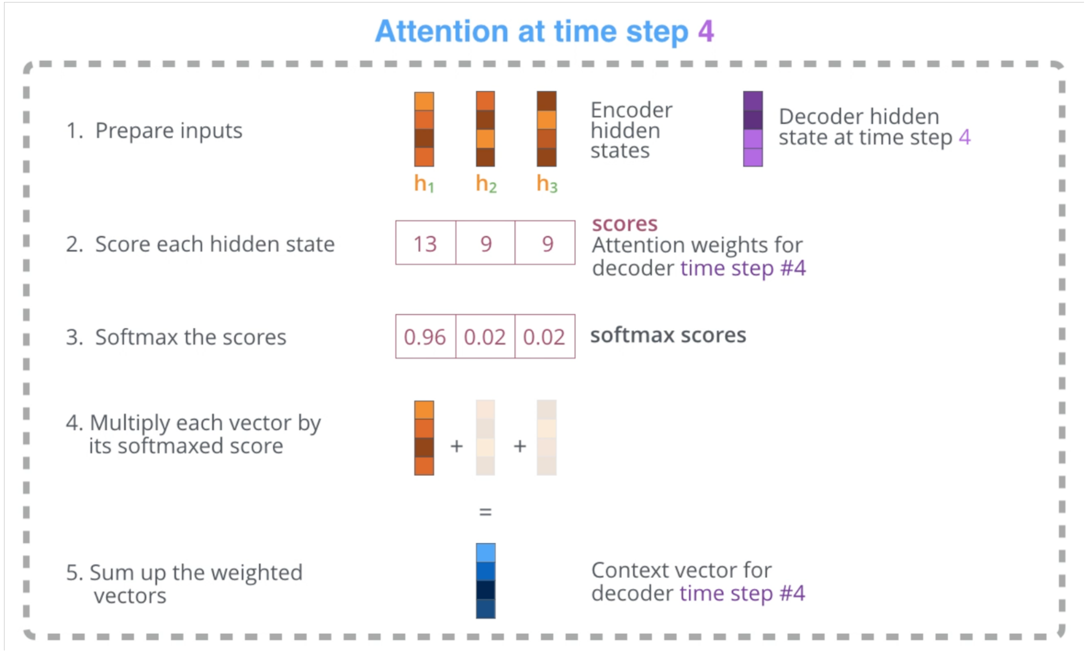
Use the attention distribution to take a **weighted sum** of the encoder hidden states.

Attention



Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_1$ as before

Attention


Attention at time step 4
1. Prepare inputs
Encoder hidden states
Decoder hidden state at time step 4
h1
h2
h3
2. Score each hidden state
13
9
9
scores
Attention weights for decoder time step #4
3. Softmax the scores
0.96
0.02
0.02
softmax scores
4. Multiply each vector by its softmaxed score
+
+
=
5. Sum up the weighted vectors
Context vector for decoder time step #4

Attention

There are different ways of generating the scores for each hidden states:

- Dot product attention

$$score_i = d_i^T * h_i$$

- Multiplicative attention

$$score_i = d_i^T * W * h_i$$

- Additive attention

$$score_i = v^T * tanh(W_1 * d_i + W_2 * h_i)$$

The weight matrices (W) and the weight vector are parameters to be learned from the model

Attention

Attention outputs can be used to visualize **alignment**