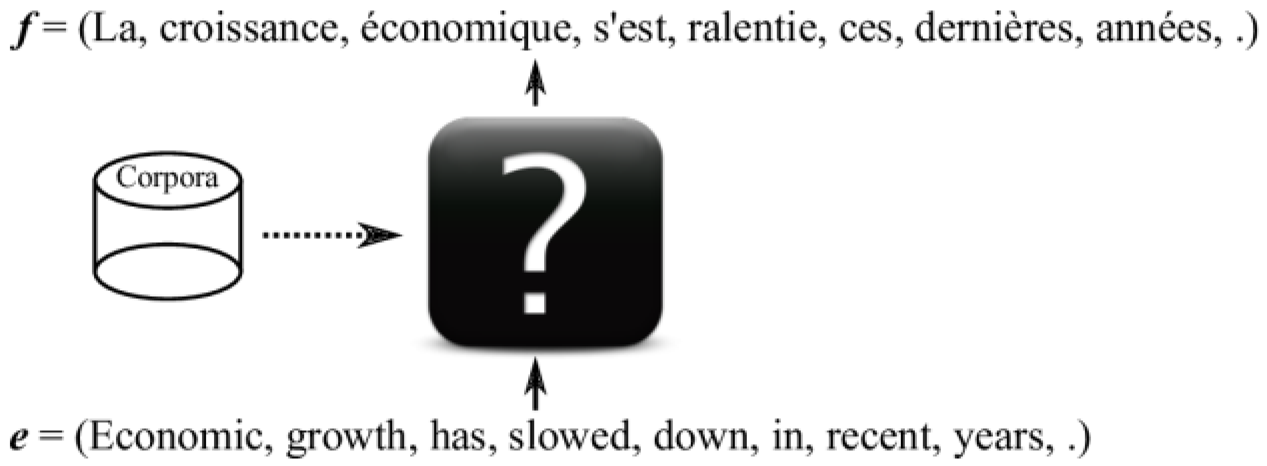
STATISTICAL MACHINE TRANSLATION

Machine translation

- Ways to build a machine that can translate languages
  - Rule-based approach
    - We can ask a bilingual speaker to give us a set of rules transforming a source sentence into a correct translation
    - We don't even know **the set of rules** underlying a single language, not to mention the rules underlying a pair of languages
  - Statistical machine translation
    - We let the machine learn from data how to translate rather than design a set of rules for the machine

Statistical Machine Translation


f = (La, croissance, économique, s'est, ralentie, ces, dernières, années, .)
Corpora
?
e = (Economic, growth, has, slowed, down, in, recent, years, .)

Characteristics of machine translation

- One-to-many in the sense that one source sentence can be translated into many possible translations

- We model the translation function not as a deterministic function but as **a conditional probability $\boldsymbol{p(y|x)}$** of a target sentence (translation) y given x.
  - The conditional probability may apply an equally high probability to more than one well-separated configurations/sentences, leading to a one-to-many relationship between source and target sentences.

Formulation

- To collect pairs of source sentences and their corresponding translations (a pair of source and corresponding translation, respectively)

$$D = \{(x^n, y^n)\}_{n=1}^{N}$$

- To score a model by looking at how well the model works on the training data *D*

Formulation

- Score
  - the log-likelihood of the model on each pair is simply **how high a log-probability the model assigns to the pair**

$$L(\theta, D) = \sum_{(x^n, y^n) \in D} \log p(y^n | x^n, \theta)$$

- Training/Learning
  - If the log-likelihood $L(\cdot,\cdot)$ is low, the model is not giving enough probability mass to the correctly translated pairs.
  - We want to find a configuration of the model, or the values of the parameters θ that maximizes this log-likelihood, or score.

Modeling – Neural Machine Translation

- The goal of NMT
  - to design a fully trainable model of which every component is tuned based on training corpora to maximize its translation performance

- Recurrent Neural Networks
  - One important property of machine translation, or any task based on natural languages, is that we deal with variable-length input $X = (x_1, x_2, \dots, x_T)$ and output $Y = (y_1, y_2, \dots, y_{T'})$. In other words, T and T' are not fixed

RNN AS A PROBABILISTIC SEQUENCE MODEL

- $p(X) = p(x_1, x_2, \dots, x_T)$ into

$$p(x_1, x_2, \dots, x_T) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)\cdots p(x_T|x_1, \dots, x_{T-1})$$
$$= p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)\cdots p(x_T|x_1, \dots, x_{T-1}) = \prod_{t=1}^{T} p(x_t|x_{<t})$$

- We let an RNN model $p(x_t|x_{<t})$ at each time t by

$$p(x_t|x_{<t}) = g_\theta(h_{t-1})$$
$$h_{t-1} = \phi_\theta(x_{t-1}, h_{t-2})$$

- g_θ outputs a probability distribution conditioned on the whole history up to the $(t-1)$ –th symbol via h_{t-1}. In other words, at each time step, the RNN tries to predict the next symbol given the history of the input symbols.

RNN

- Probabilistically model a sequence

$$
\begin{aligned}
p(x_1, x_2, \ldots, x_T) &= p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)\cdots p(x_T|x_1, \ldots, x_{T-1}) \\
&= \textstyle\prod_{t=1}^{T} p(x_t|x_{<t})
\end{aligned}
$$

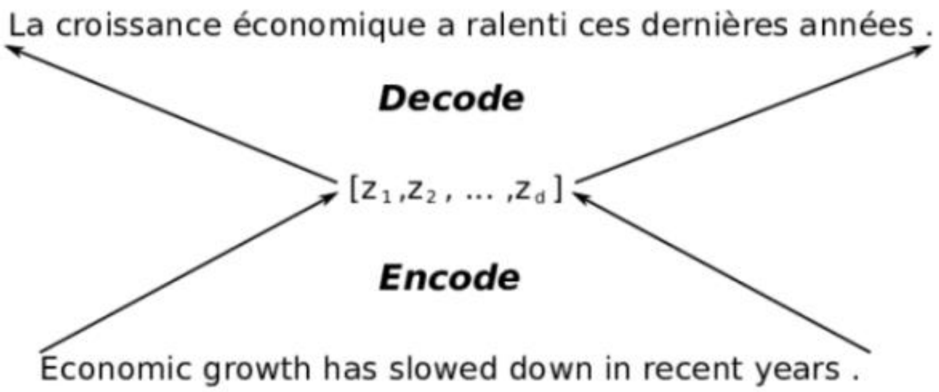
- Summarize a sequence
  - to compress a sequence of input symbols into a fixed-dimensional vector by using recursion

ENCODER–DECODER ARCHITECTURE FOR MACHINE TRANSLATION

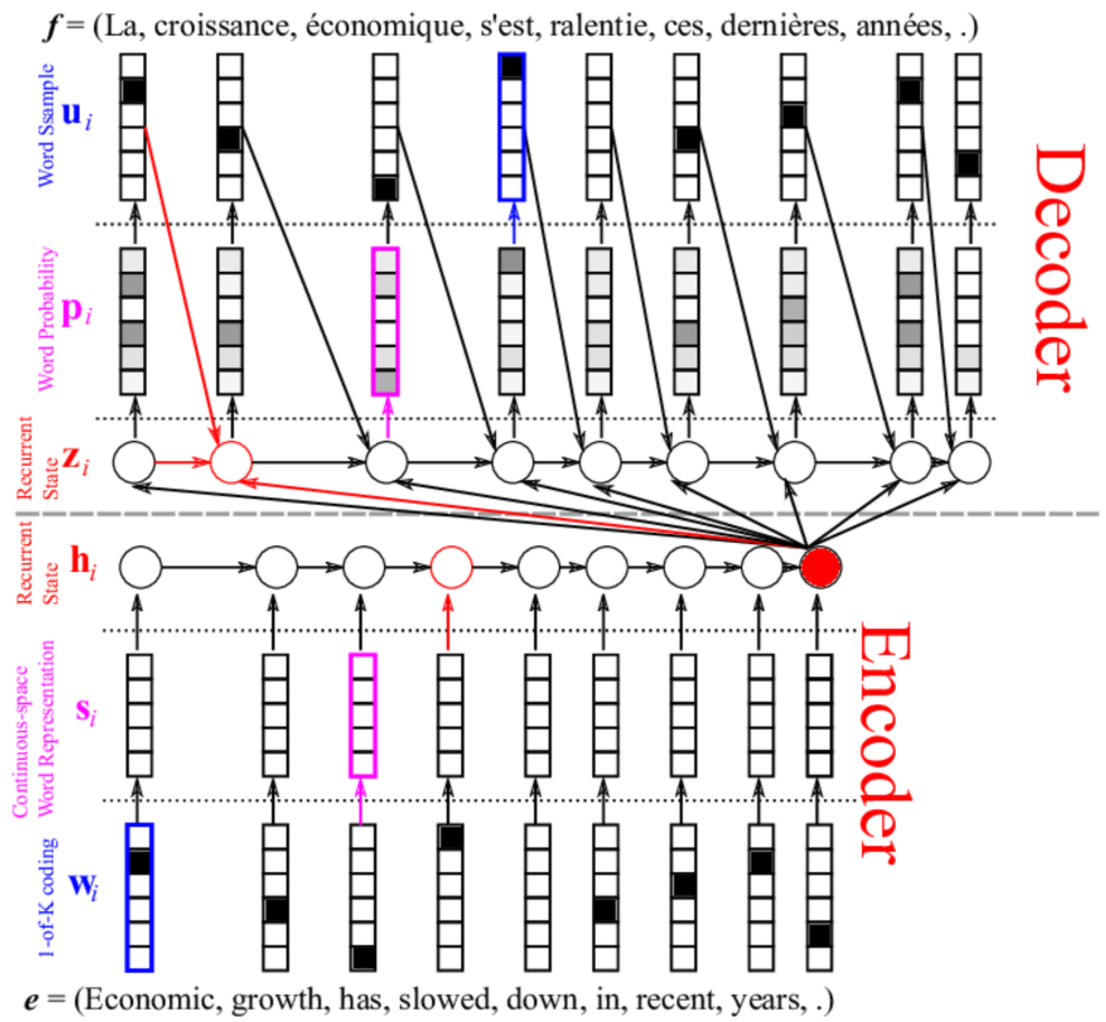
https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-2/

Encoder–Decoder for Machine Translation

- When translating a short sentence in English to Korean, a brain *encodes* the English sentence into a set of neuronal activations, and from those activations, the corresponding Korean sentence is generated.
- In other words, the process of (human) translation involves the *encoder* which turns a sequence of words into a set of neuronal activations (or spikes, or whatever's going on inside a biological brain) and the *decoder* which generates a sequence of words in another language, from the set of activations

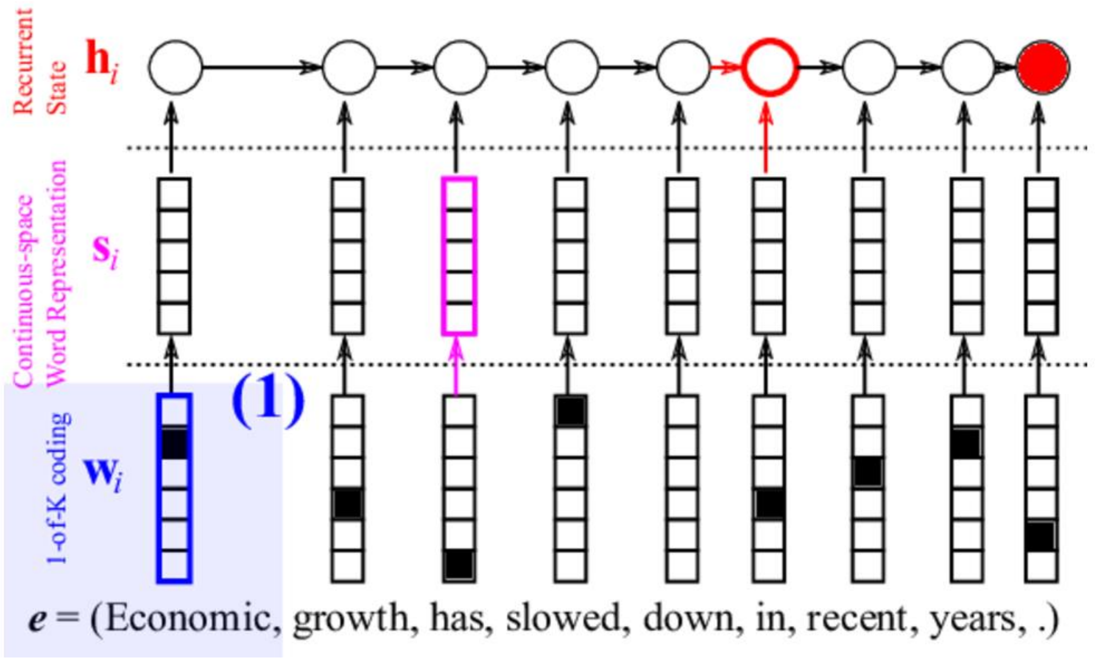

La croissance économique a ralenti ces dernières années .
Decode
$[z_1, z_2, \ldots, z_d]$
Encode
Economic growth has slowed down in recent years .

Big picture of the whole system


f = (La, croissance, économique, s'est, ralentie, ces, dernières, années, .)
Word Ssample $\mathbf{u}_i$
Word Probability $\mathbf{p}_i$
Recurrent State $\mathbf{z}_i$
Decoder
Recurrent State $\mathbf{h}_i$
Continuous-space Word Representation $\mathbf{s}_i$
1-of-K coding $\mathbf{w}_i$
Encoder
e = (Economic, growth, has, slowed, down, in, recent, years, .)

Encoder

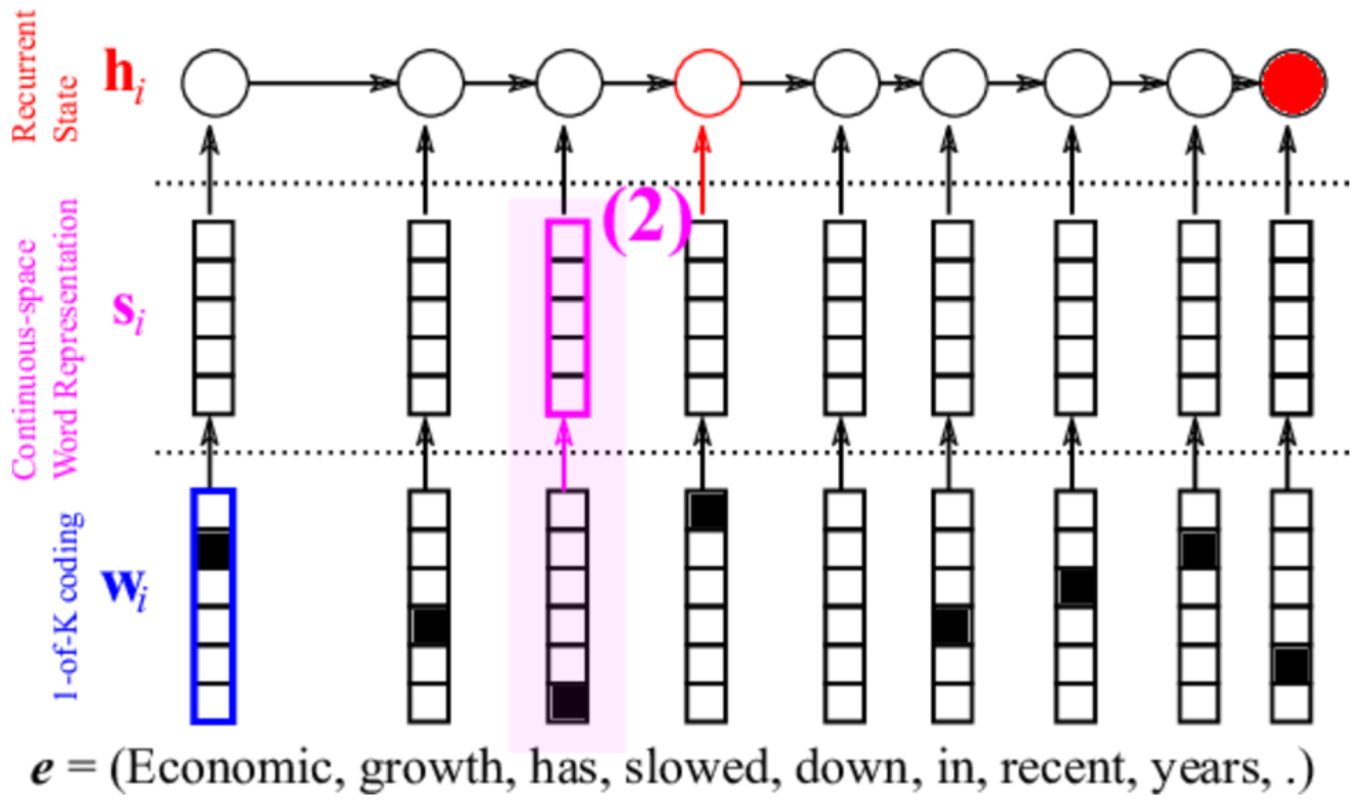
- A straightforward application of a recurrent neural network, based on its property of sequence summarization
  - RNN h_T is the summary of the whole input sentence.


Recurrent State
$\mathbf{h}_i$
Continuous-space Word Representation
$\mathbf{s}_i$
1-of-K coding
$\mathbf{w}_i$
(1)
e = (Economic, growth, has, slowed, down, in, recent, years, .)

A word to a one-hot vector

Encoder

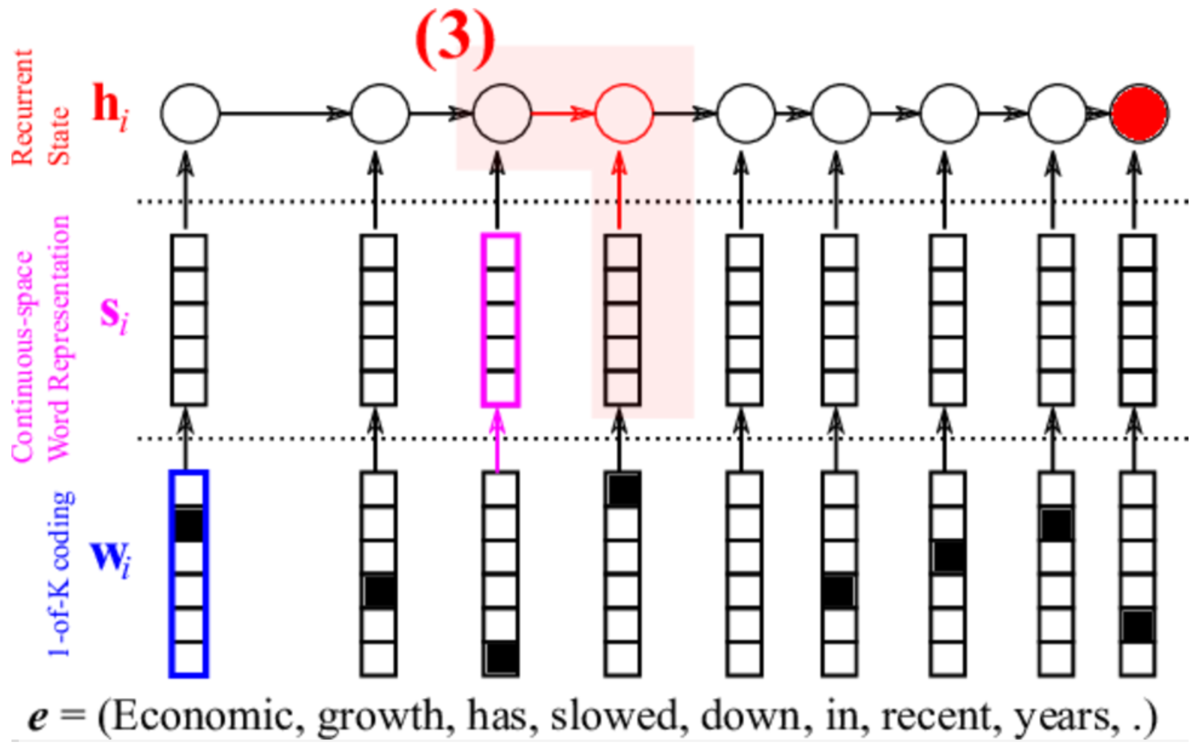
- A one-hot vector to a continuous-space representation



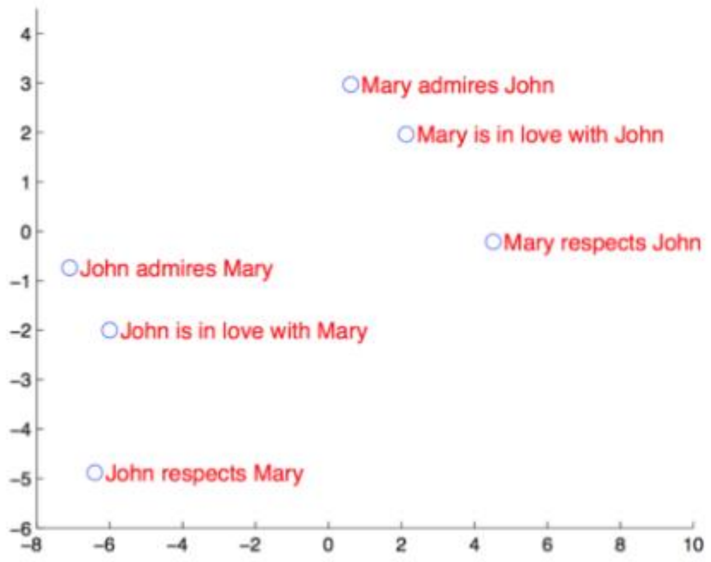
Recurrent State
$\mathbf{h}_i$
Continuous-space Word Representation
$\mathbf{s}_i$
(2)
1-of-K coding
$\mathbf{w}_i$
e = (Economic, growth, has, slowed, down, in, recent, years, .)

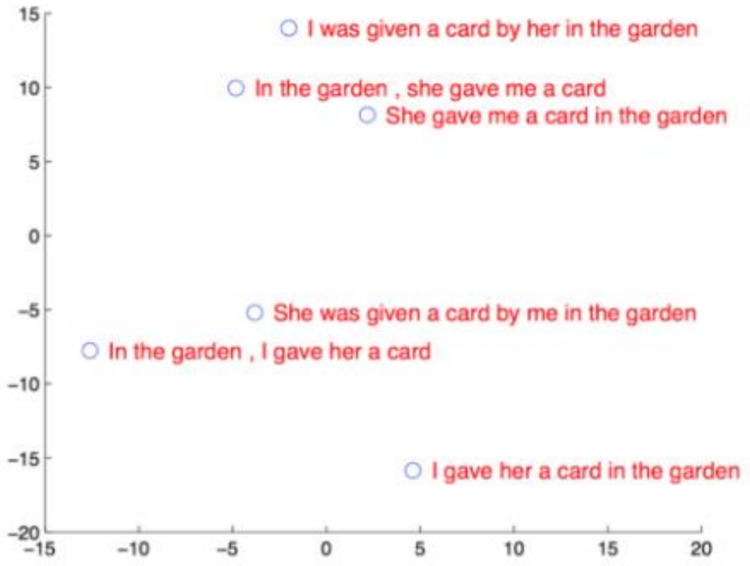
Encoder

- Sequence summarization by a recurrent neural network
  - RNN's internal state h_T represents a summary of the whole source sentence.


(3)
Recurrent State
$\mathbf{h}_i$
Continuous-space Word Representation
$\mathbf{s}_i$
1-of-K coding
$\mathbf{w}_i$
e = (Economic, growth, has, slowed, down, in, recent, years, .)

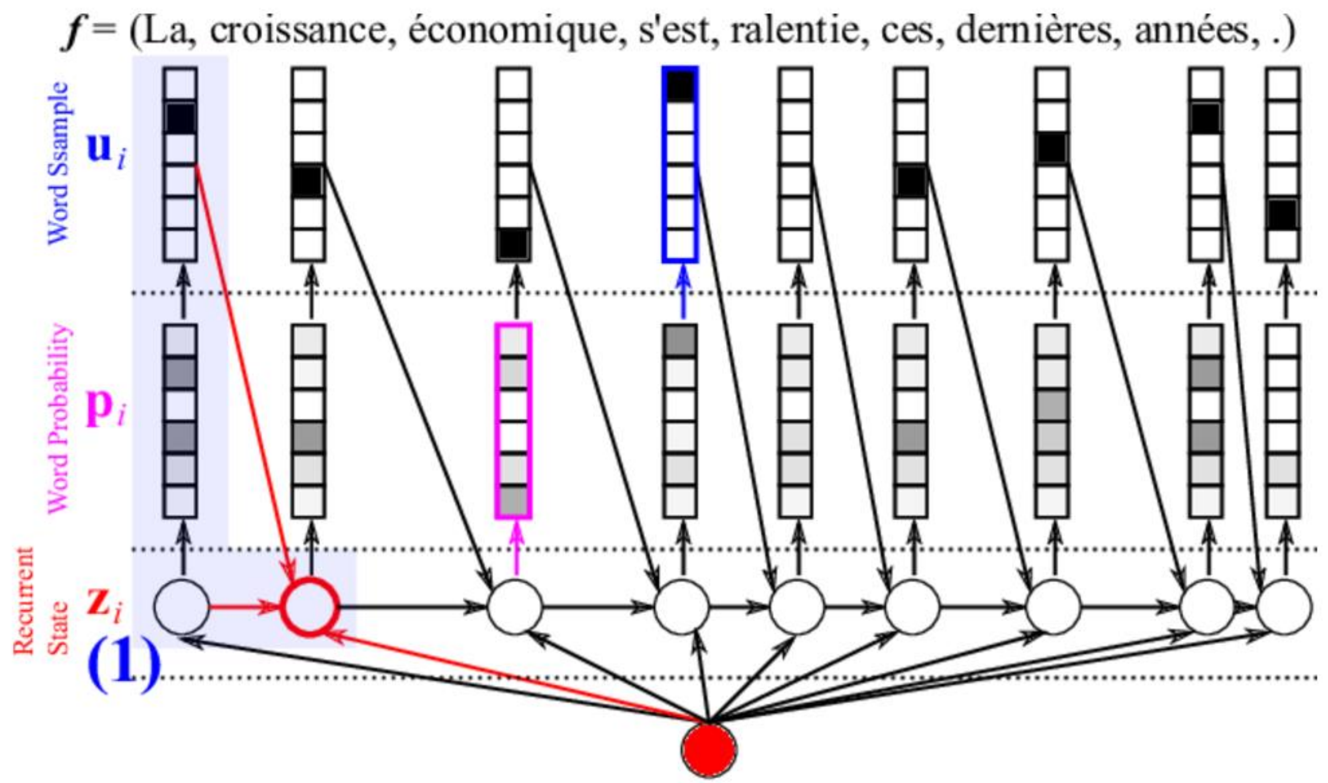
WHAT DOES THE SUMMARY VECTOR LOOK LIKE?



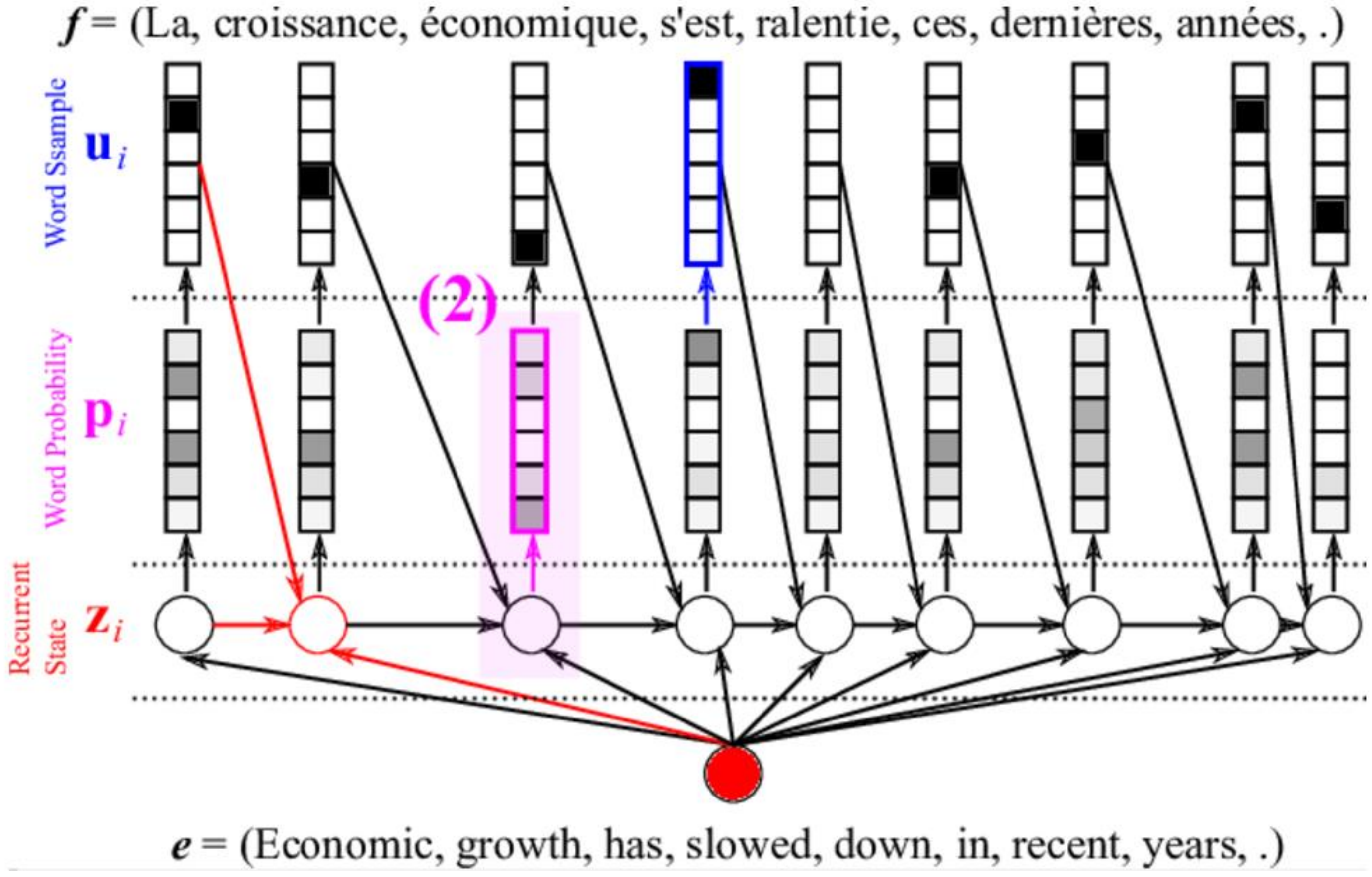


Decoder

- Now we have a nice fixed-size representation of a source sentence

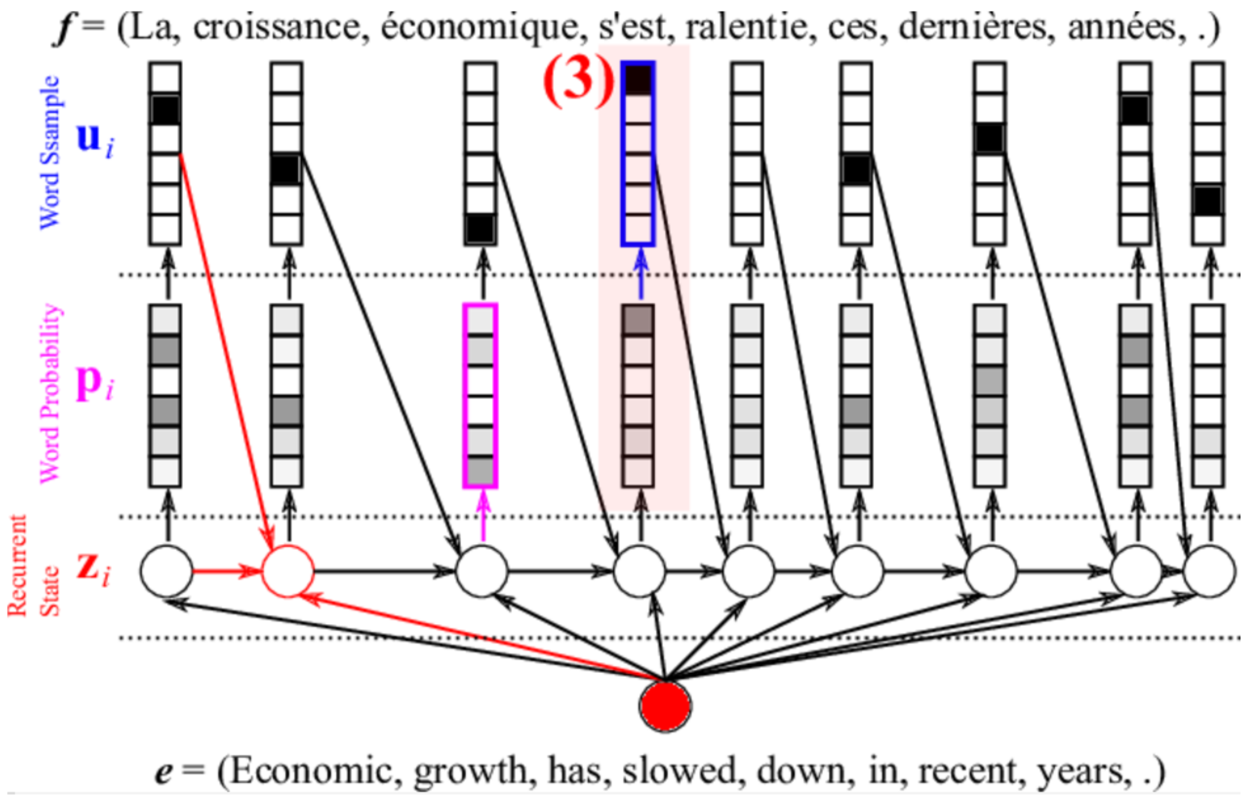


Next word probability


f = (La, croissance, économique, s'est, ralentie, ces, dernières, années, .)
Word Ssample
$\mathbf{u}_i$
Word Probability
$\mathbf{p}_i$
(2)
Recurrent State
$\mathbf{z}_i$
e = (Economic, growth, has, slowed, down, in, recent, years, .)

Sampling

- We have a probability distribution over the target words, which we can use to select a word by sampling the distribution



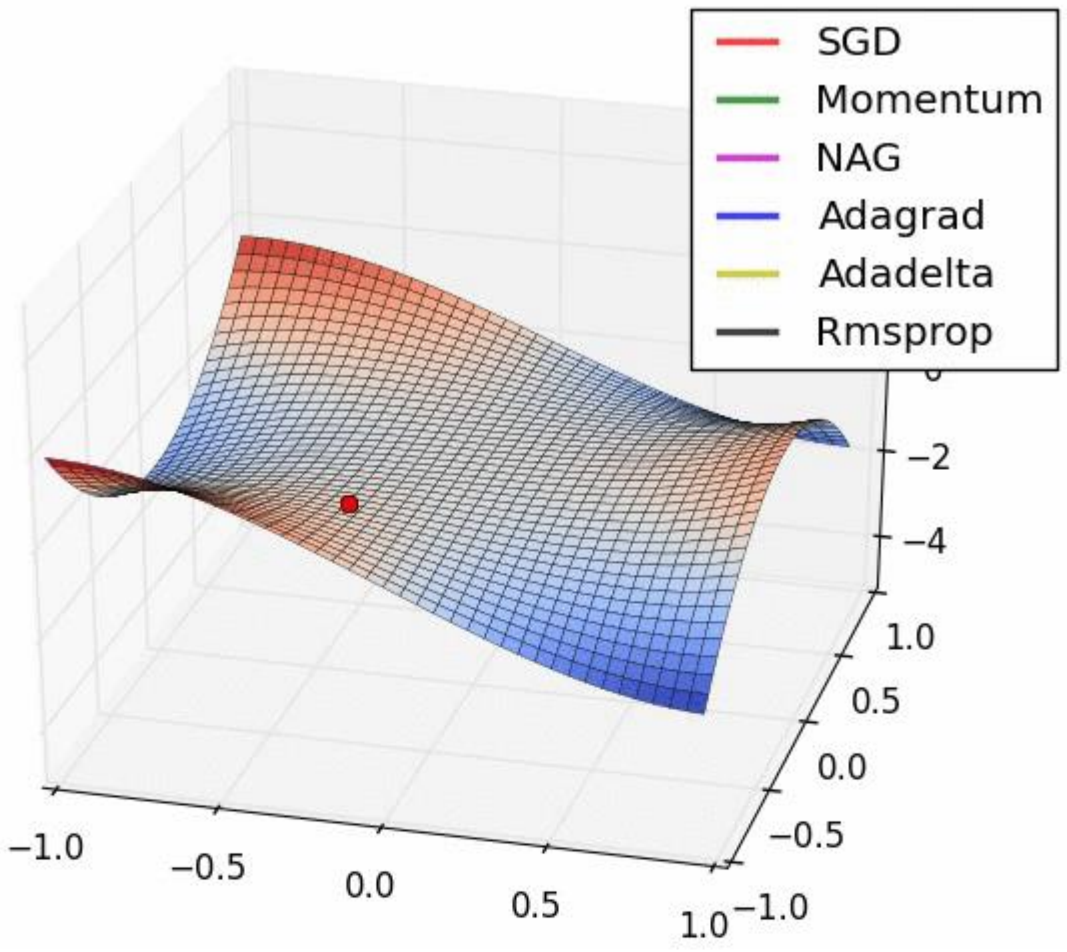
f = (La, croissance, économique, s'est, ralentie, ces, dernières, années, .)
(3)
Word Ssample
u_i
Word Probability
p_i
Recurrent State
z_i
e = (Economic, growth, has, slowed, down, in, recent, years, .)

Training

- Corpus D must be prepared.
  - Each sample in the corpus is a pair (X^n, Y^n) of source and target sentences.
  - Each sentence is a sequence of integer indices corresponding to words, which is equivalent to a sequence of one-hot vectors.
  - Given any pair from the corpus, the NMT model can compute the conditional log-probability of Y^n given X^n: $\log P(Y^n | X^n, \theta)$, and we write the log-likelihood of the whole training corpus as

$$\mathcal{L}(D, \theta) = \frac{1}{N} \sum_{n=1}^{N} \log P(Y^n | X^n, \theta)$$

Stochastic gradient descent



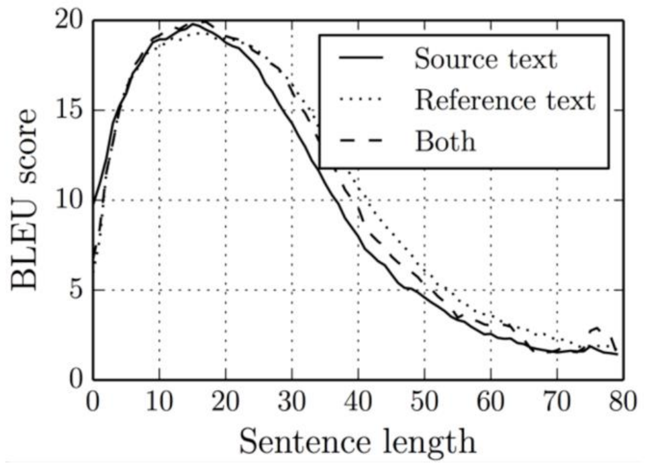
SOFT ATTENTION MECHANISM FOR NEURAL MACHINE TRANSLATION

https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-3/

Simple Encoder–Decoder Architectures

- In the encoder-decoder architecture, the encoder compresses the input sequence as **a fixed-size vector** from which the decoder needs to generate a full translation.

- Translation quality dramatically degrades as the length of the source sentence increases when the encoder-decoder model size is small



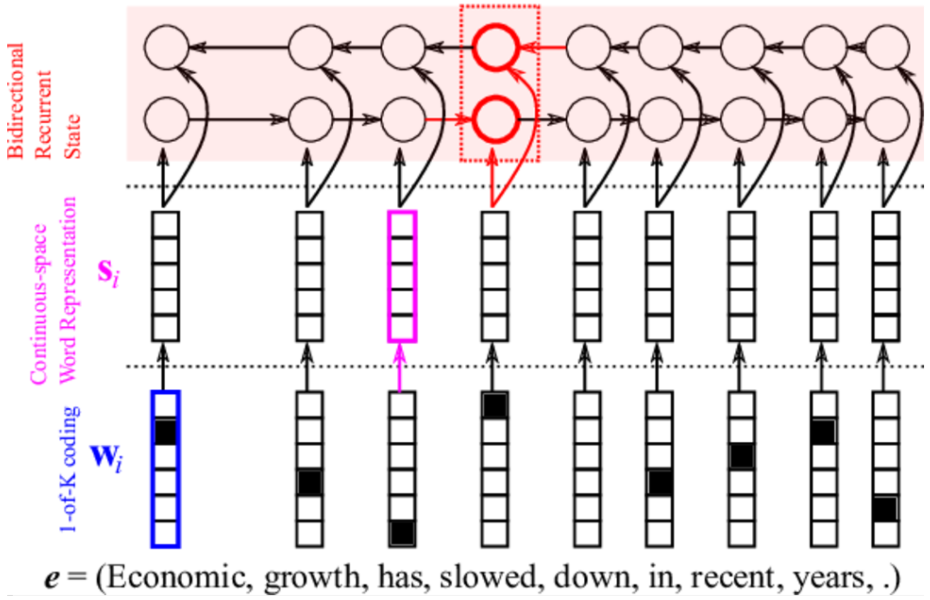
Bidirectional RNN

- The biggest issue with the simple encoder-decoder architecture is that a sentence of any length needs to be compressed into a fixed-size vector

- Bidirectional recurrent neural network (BiRNN) which consists of a forward recurrent neural network (RNN) and a separate *backward* RNN.

Bidirectional RNN

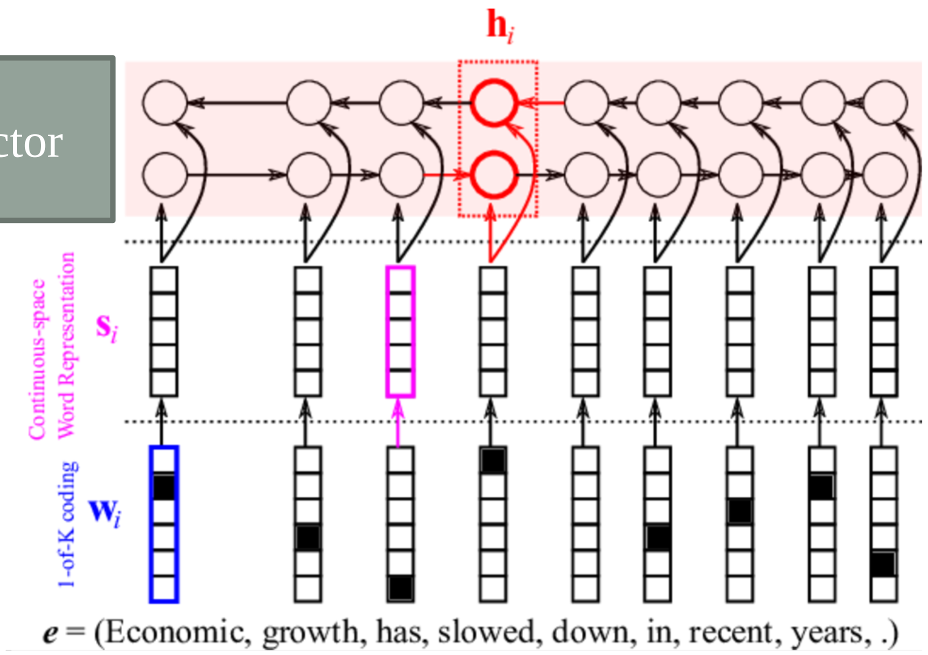
- $h_{\rightarrow j}$ of the forward RNN summarizes the source sentence up to the j-th word beginning from the first word
- $h_{j\leftarrow}$ of the backward RNN up to the j-th word beginning from the last word


Bidirectional Recurrent State
Continuous-space Word Representation
$\mathbf{s}_i$
1-of-K coding
$\mathbf{w}_i$
e = (Economic, growth, has, slowed, down, in, recent, years, .)

Annotation vectors

- Annotation vector as a *context-dependent word representation*
- We can consider this set of context-dependent word representations as a mechanism by which we store the source sentence as a variable-length representation

Annotation vector



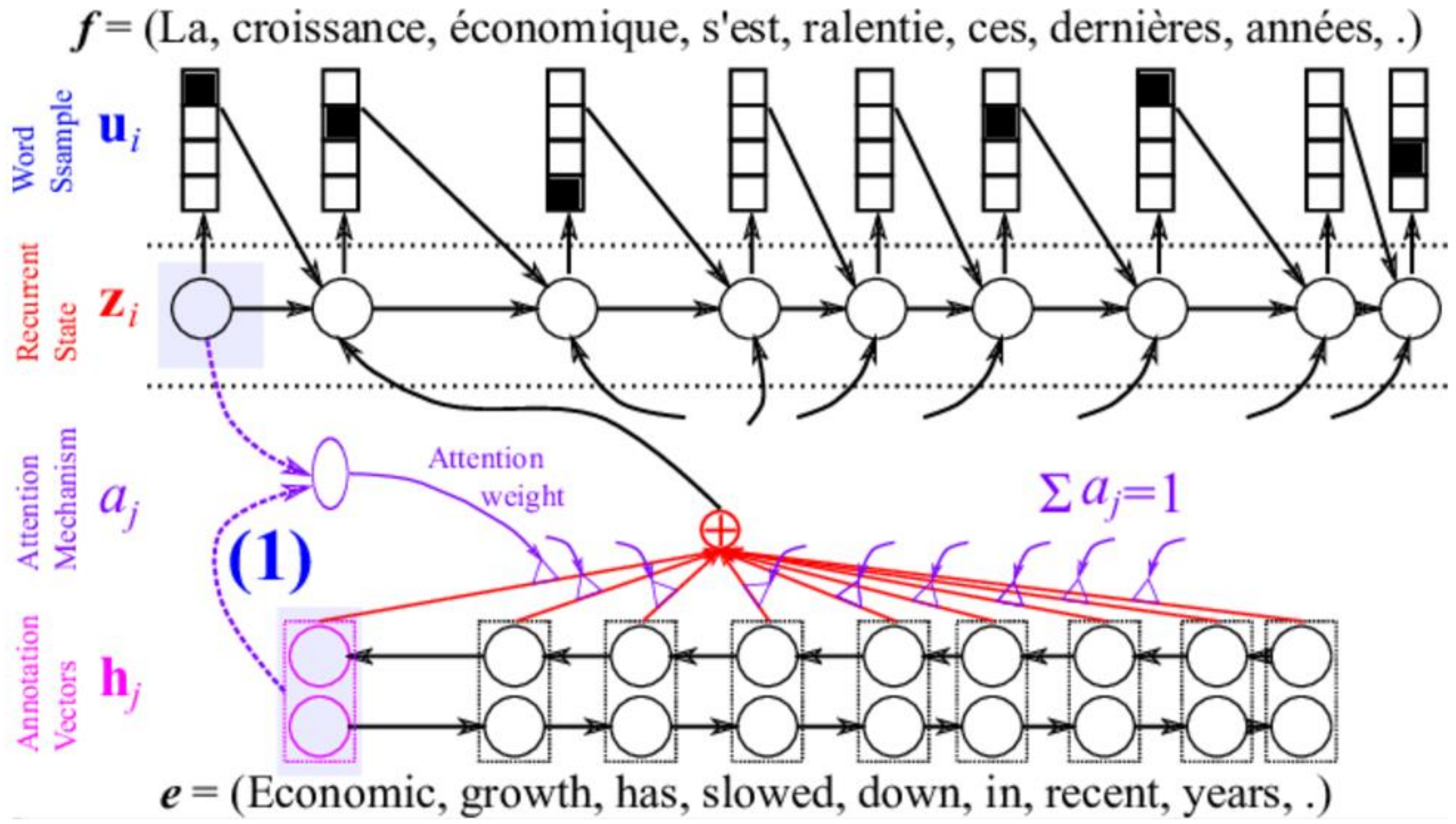
$\mathbf{h}_i$
Continuous-space Word Representation
$\mathbf{s}_i$
1-of-K coding
$\mathbf{w}_i$
e = (Economic, growth, has, slowed, down, in, recent, years, .)

Selective focus

- With this variable-length representation of a source sentence, the decoder now needs to be able to ***selectively* focus on one or more of the context-dependent word representations**, or the annotation vectors, for each target word.

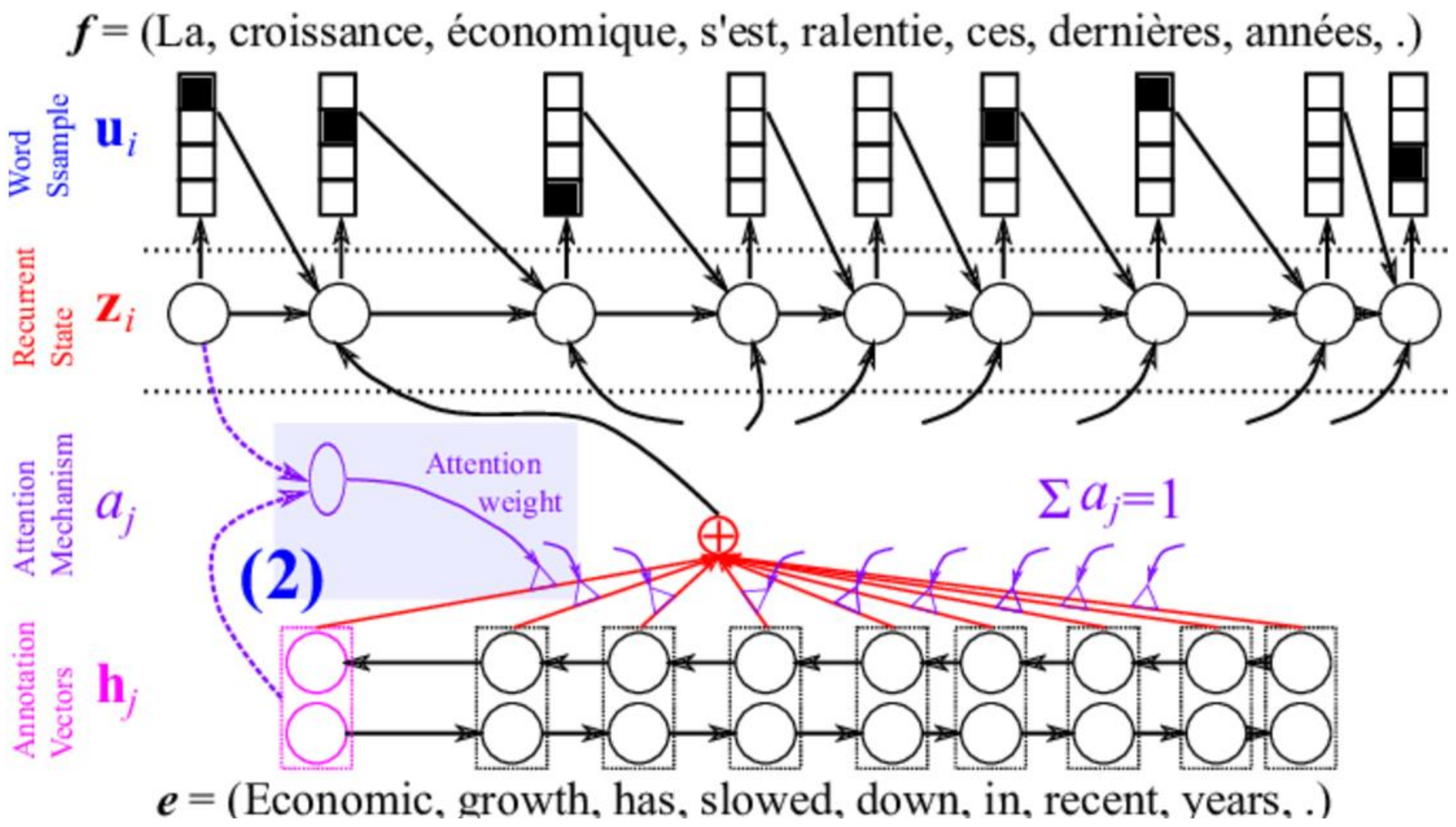
- A typical translator looks at each source word x_j (or its context-dependent representation h_j), considers it together with the already translated words $(y_1, y_2, \dots, y_{t-1})$ and decides how (ir)relevant the source word x_j is for the next target word. It repeats this process for every word in the source sentence.

Attention mechanism

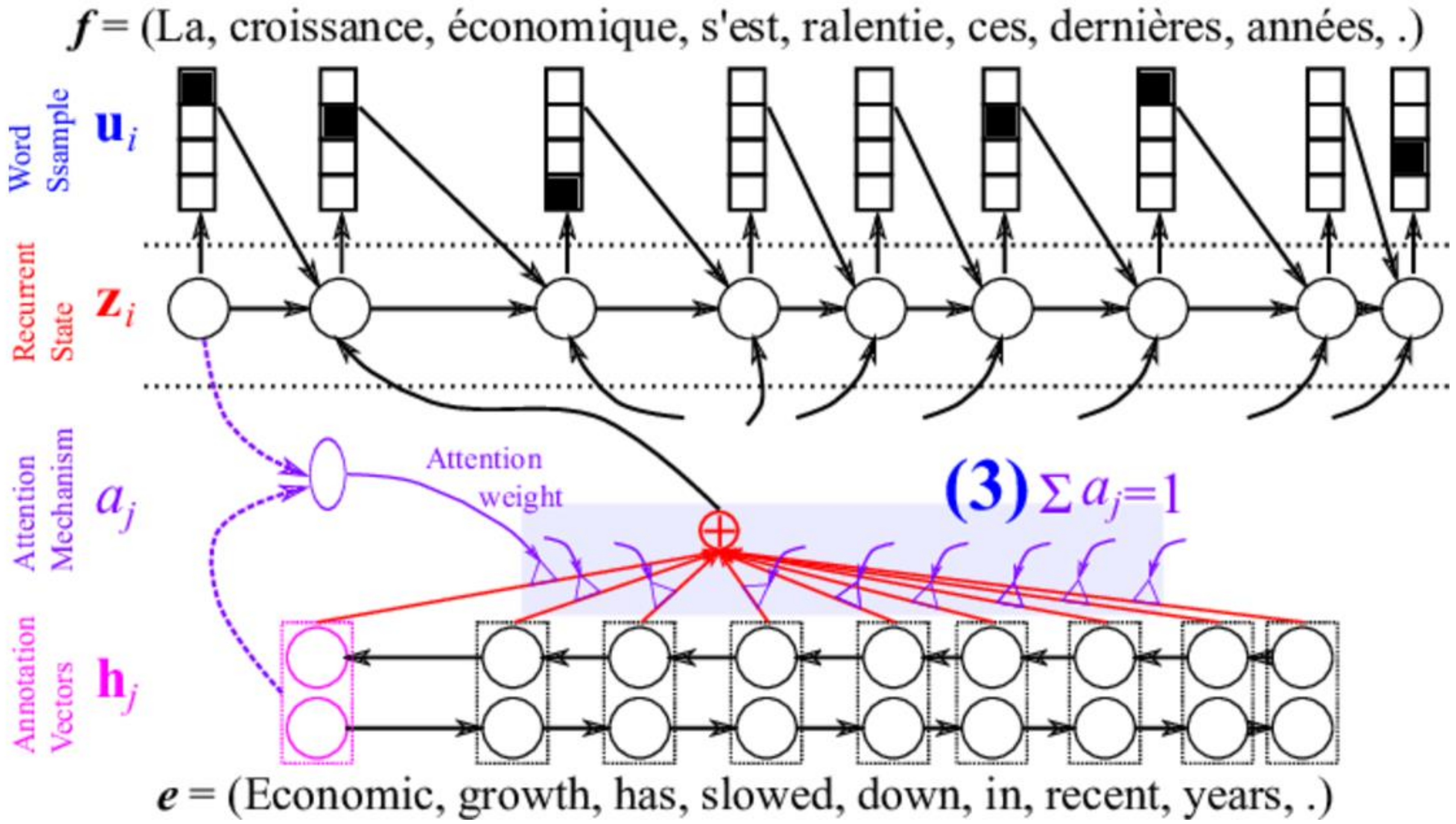


f = (La, croissance, économique, s'est, ralentie, ces, dernières, années, .)
Word Ssample
$\mathbf{u}_i$
Recurrent State
$\mathbf{z}_i$
Attention Mechanism
a_j
Attention weight
$\sum a_j=1$
(1)
Annotation Vectors
$\mathbf{h}_j$
e = (Economic, growth, has, slowed, down, in, recent, years, .)

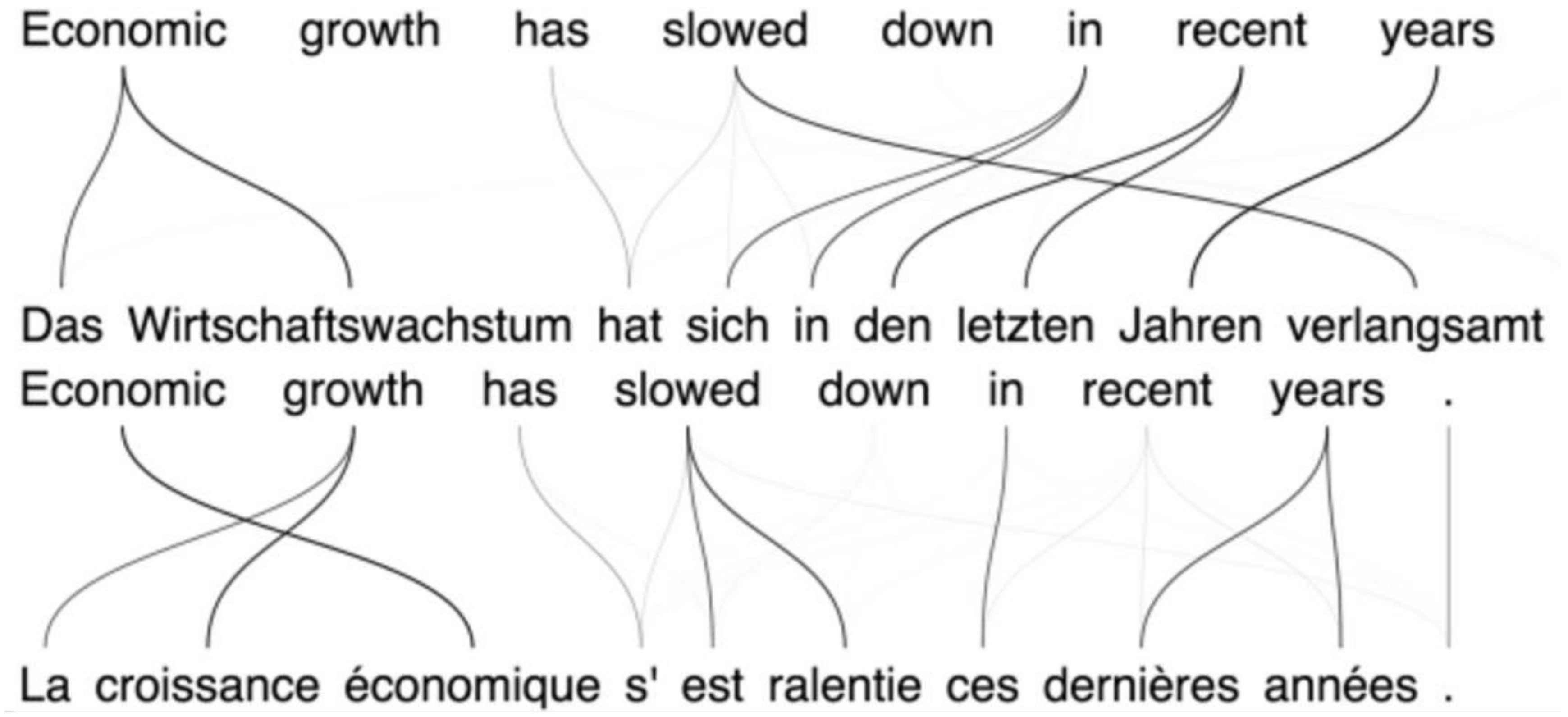
Attention mechanism



Attention mechanism



Example

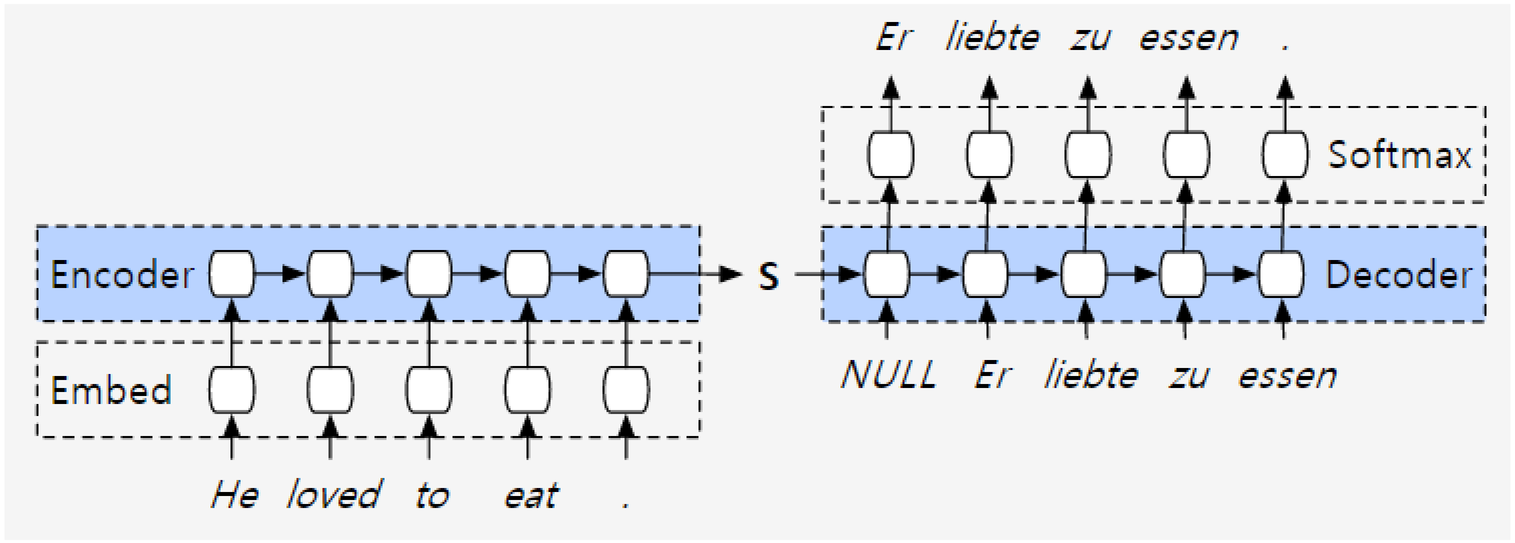

Economic growth has slowed down in recent years
Das Wirtschaftswachstum hat sich in den letzten Jahren verlangsamt
Economic growth has slowed down in recent years .
La croissance économique s' est ralentie ces dernières années .

GOOGLE'S NEURAL MACHINE TRANSLATION

http://smerity.com/articles/2016/google_nmt_arch.html

Encoder–decoder model

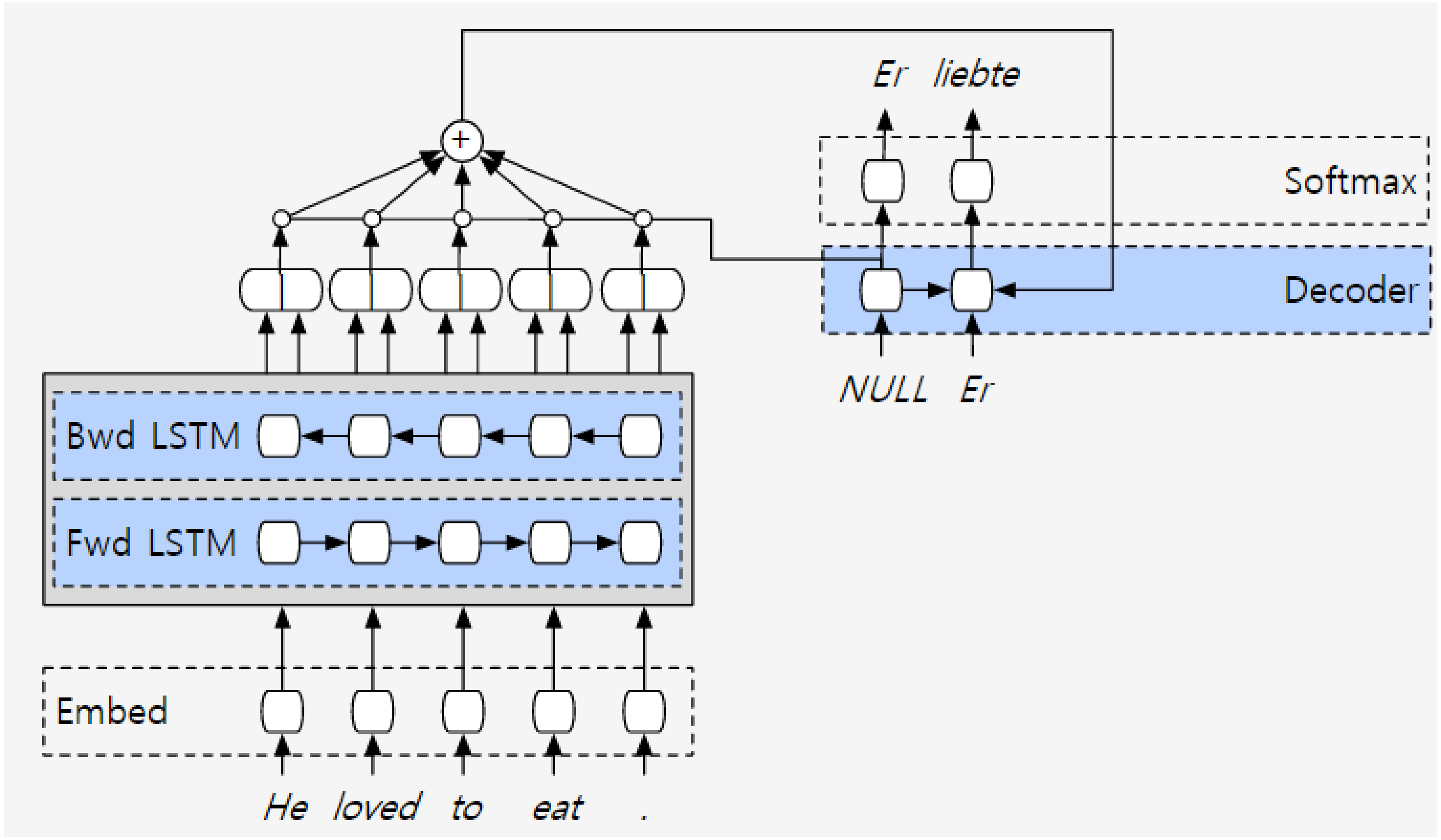
- Take an recurrent neural network (RNN) - usually an LSTM - and encode a sentence written in language A (English).
- The RNN spits out a hidden state, which we refer to as **S**.
- This hidden state hopefully represents all the content of the sentence.
- This hidden state **S** is then supplied to the decoder, which generates the sentence in language B (German) word by word.



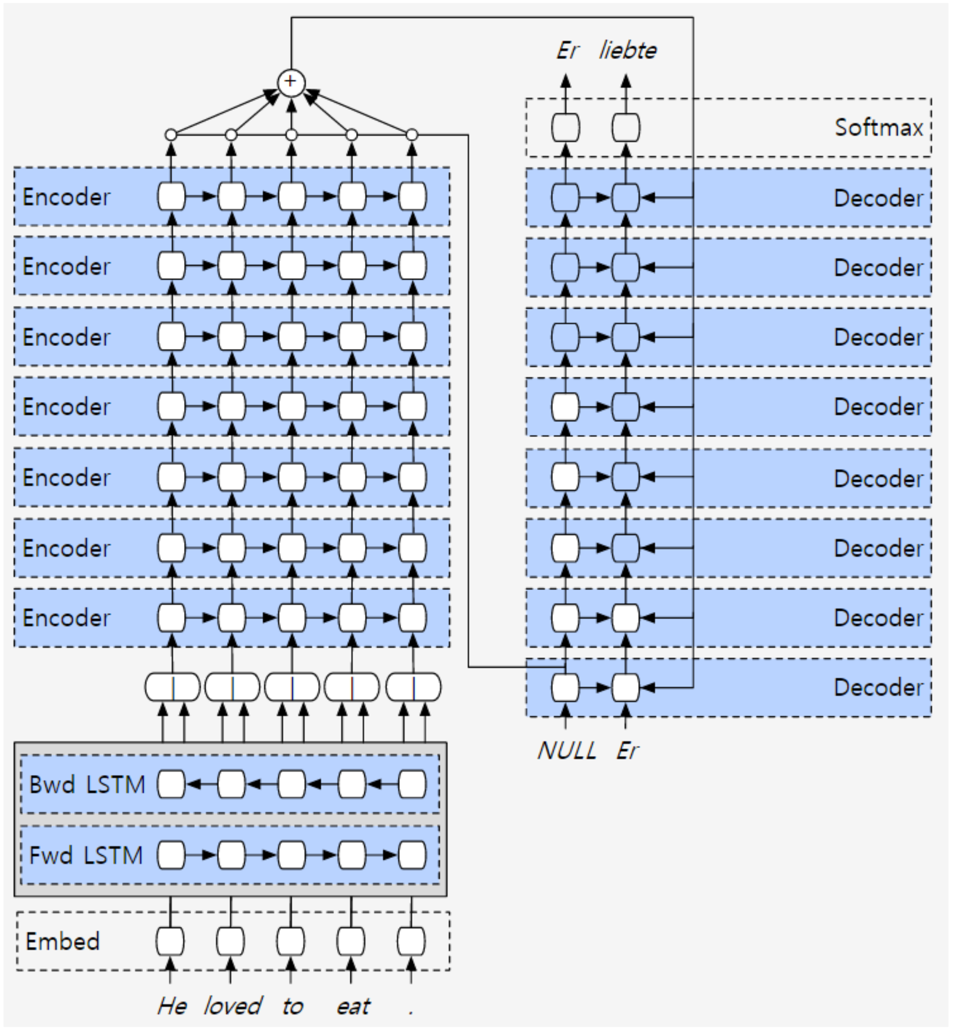
Drawbacks

- First,
  - this architecture has very **limited memory**. That final hidden state of the LSTM, which we call **S**, is where you're trying to cram the entirety of the sentence you have to translate. **S** is usually only a few hundred units (read: floating point numbers) long - **the more you try to force into this fixed dimensionality vector, the more lossy the neural network is forced to be**.
- Second,
  - **the deeper a neural network is, the harder it is to train**. For recurrent neural networks, the longer the sequence is, the deeper the neural network is along the time dimension. This results in vanishing gradients. Even with RNNs specifically made to help prevent vanishing gradients, such as the LSTM, this is still a fundamental problem.

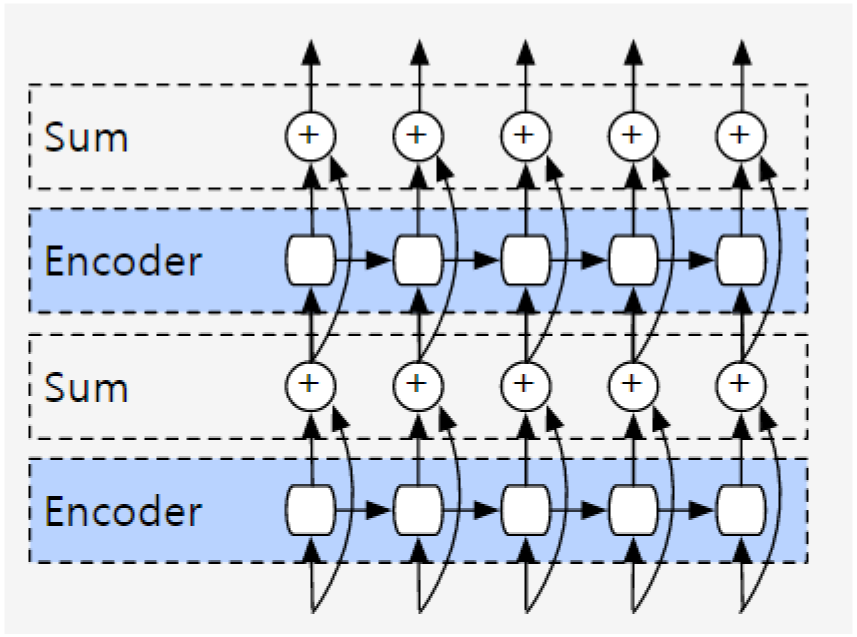
Attention based encoder-decoder



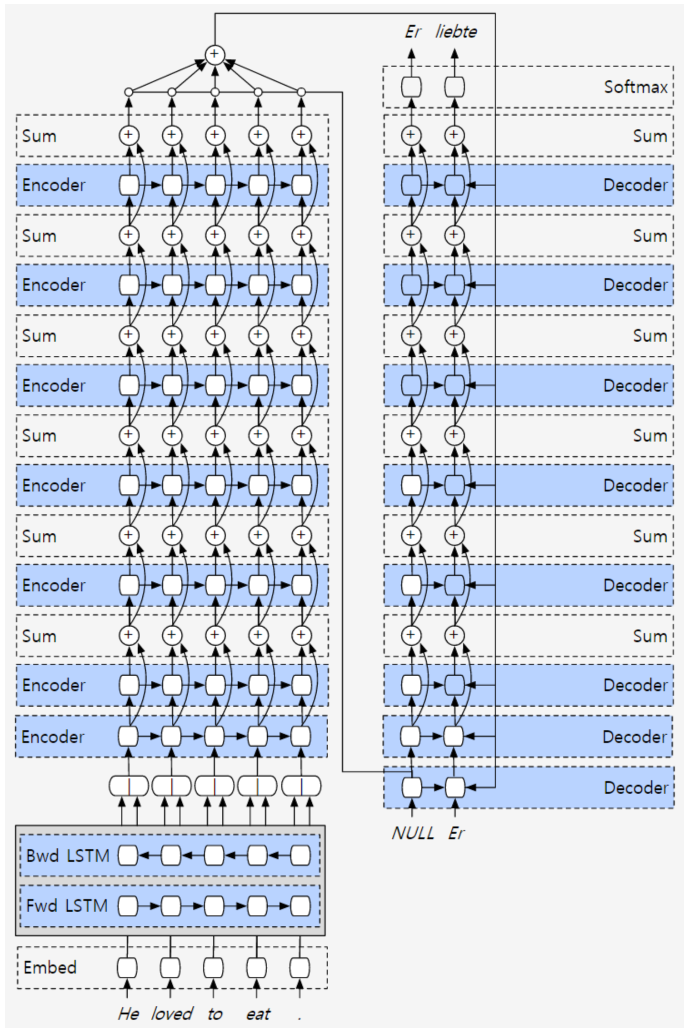
The deep is for deep learning



Residuals are the new hotness



Putting it all together



BACKUPS

Score/probability

- We score each word k given a hidden state z_i such that
$$e(k) = w_k^T z_i + b_k$$
where w_k and b_k are the target word vector and a bias, respectively.

- The dot product between two vectors. The dot product is larger when the target word vector w_k and the decoder's internal state z_i are similar to each other, and smaller otherwise.

$$p(w_i = k | w_1, w_2, \ldots, w_{i-1}, h_T) = \frac{\exp(e(k))}{\sum_j \exp(e(j))}$$