Neurocomputing

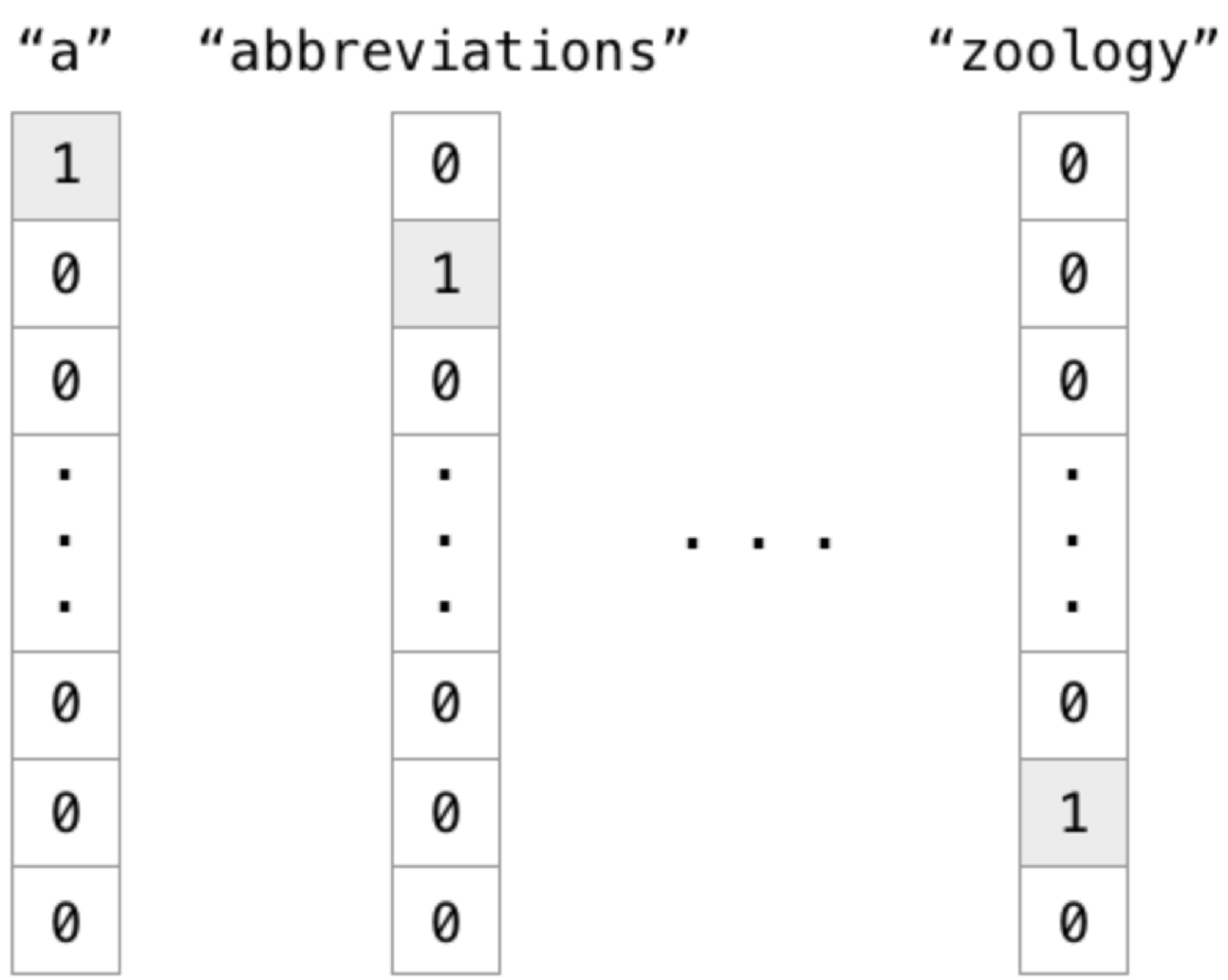
Natural Language Processing

Julien Vitay
Professur für Künstliche Intelligenz - Fakultät für Informatik

1 - word2vec

Representing words

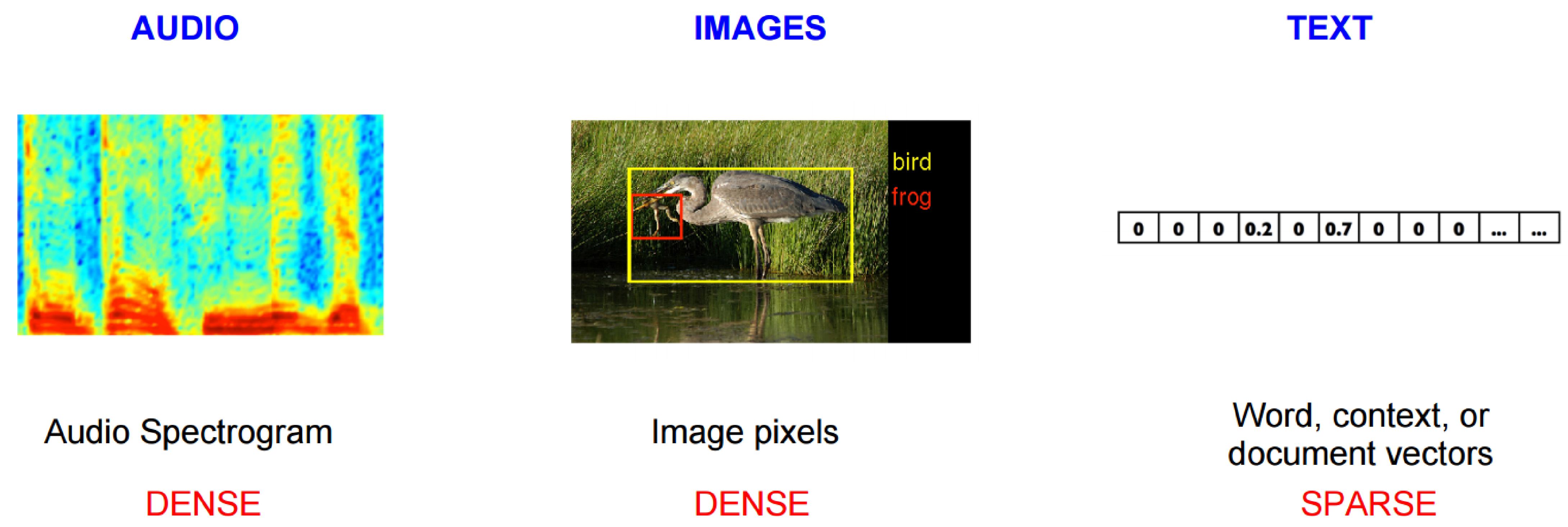
- The most famous application of RNNs is **Natural Language Processing** (NLP): text understanding, translation, etc…
- Each word of a sentence has to be represented as a vector $\mathbf{x}_t$ in order to be fed to a LSTM.
- Which representation should we use?
- The naive solution is to use **one-hot encoding**, one element of the vector corresponding to one word of the dictionary.



Source: https://cdn-images-1.medium.com/max/1600/1*ULfyiWPKgWceCqyZeDTl0g.png

Representing words

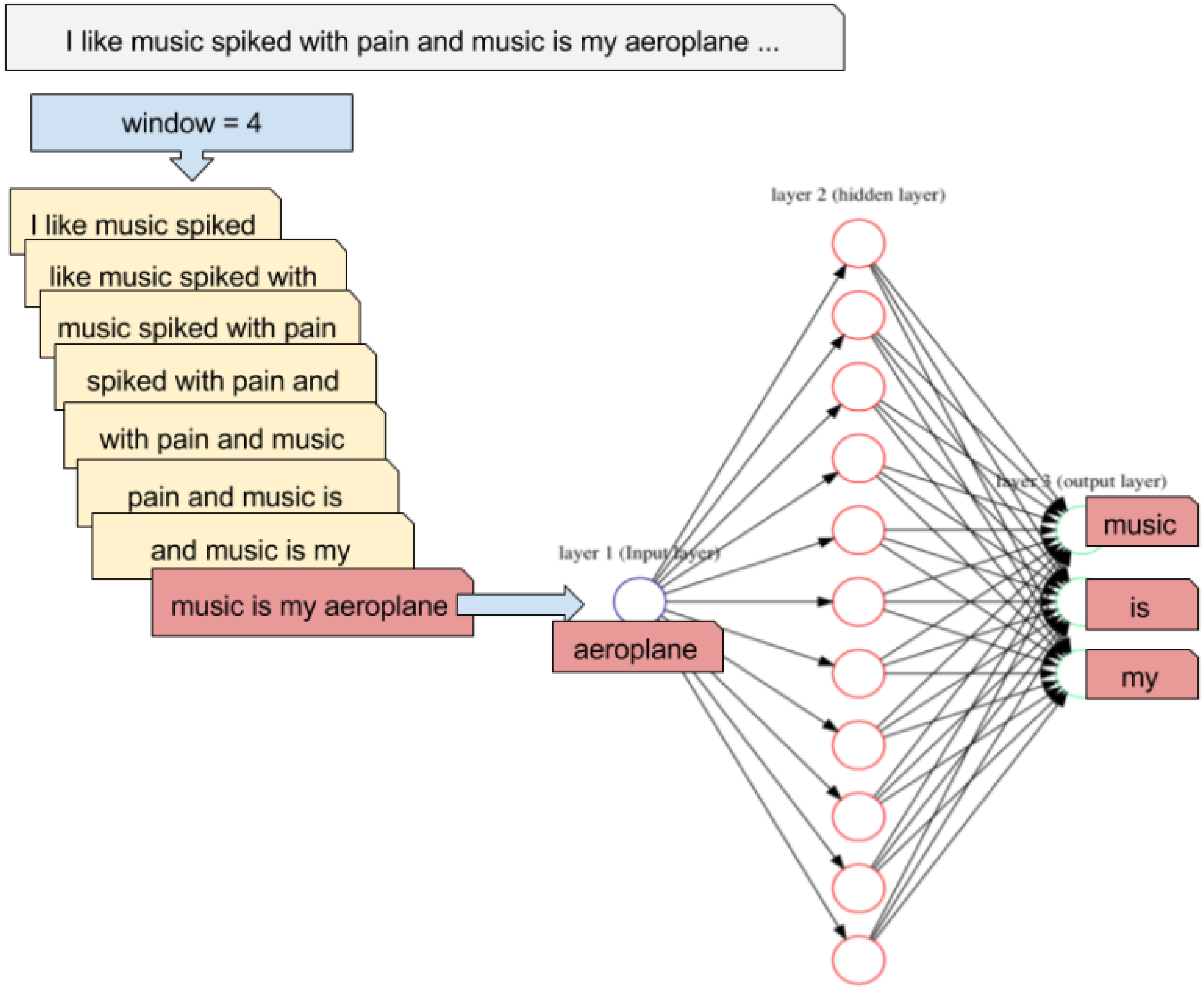
- One-hot encoding is not a good representation for words:
    - The vector size will depend on the number of words of the language:
        - English: 171,476 (Oxford English Dictionary), 470,000 (Merriam-Webster)... 20,000 in practice.
        - French: 270,000 (TILF).
        - German: 200,000 (Duden).
        - Chinese: 370,000 (Hanyu Da Cidian).
        - Korean: 1,100,373 (Woori Mal Saem)
    - Semantically related words have completely different representations (“endure” and “tolerate”).
    - The representation is extremely **sparse** (a lot of useless zeros).



Source: https://www.tensorflow.org/tutorials/representation/word2vec

word2vec

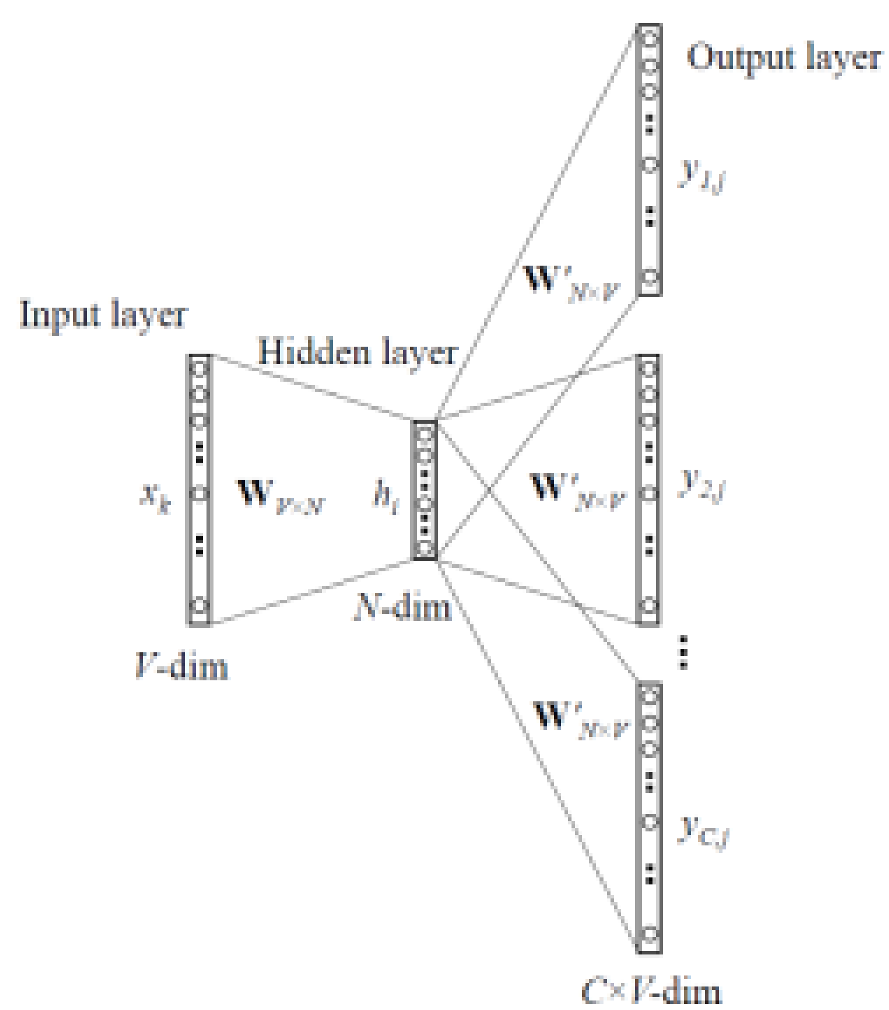
- **word2vec** learns word **embeddings** by trying to predict the current word based on the context (CBOW, continuous bag-of-words) or the context based on the current word (skip-gram).
- It uses a three-layer autoencoder-like NN, where the hidden layer (latent space) will learn to represent the one-hot encoded words in a dense manner.

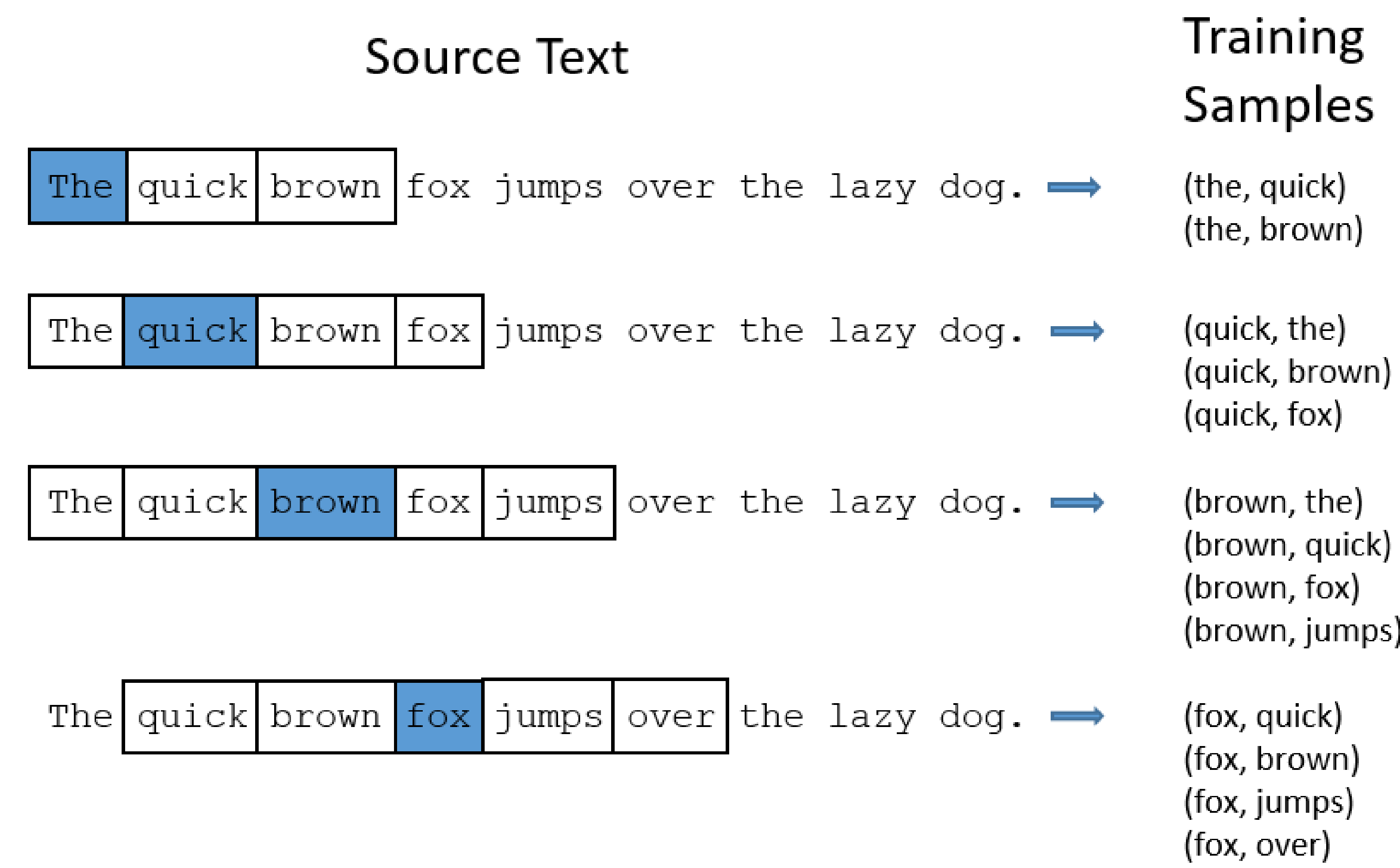


Source: https://jaxenter.com/deep-learning-search-word2vec-147782.html

word2vec

- **word2vec** has three parameters:
  - the **vocabulary size**: number of words in the dictionary.
  - the **embedding size**: number of neurons in the hidden layer.
  - the **context size**: number of surrounding words to predict.
- It is trained on huge datasets of sentences (e.g. Wikipedia).



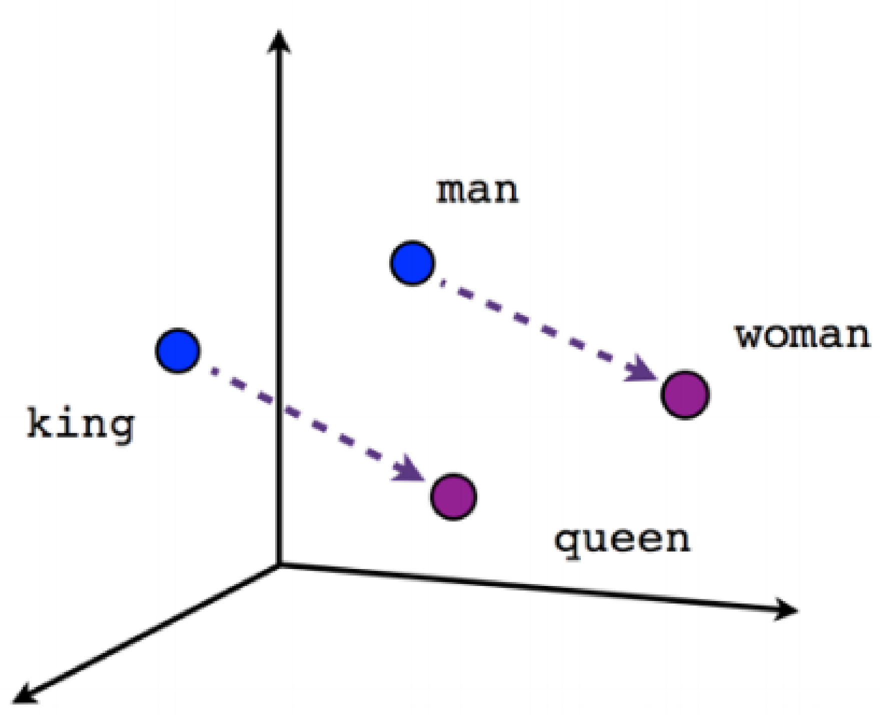


Source: https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

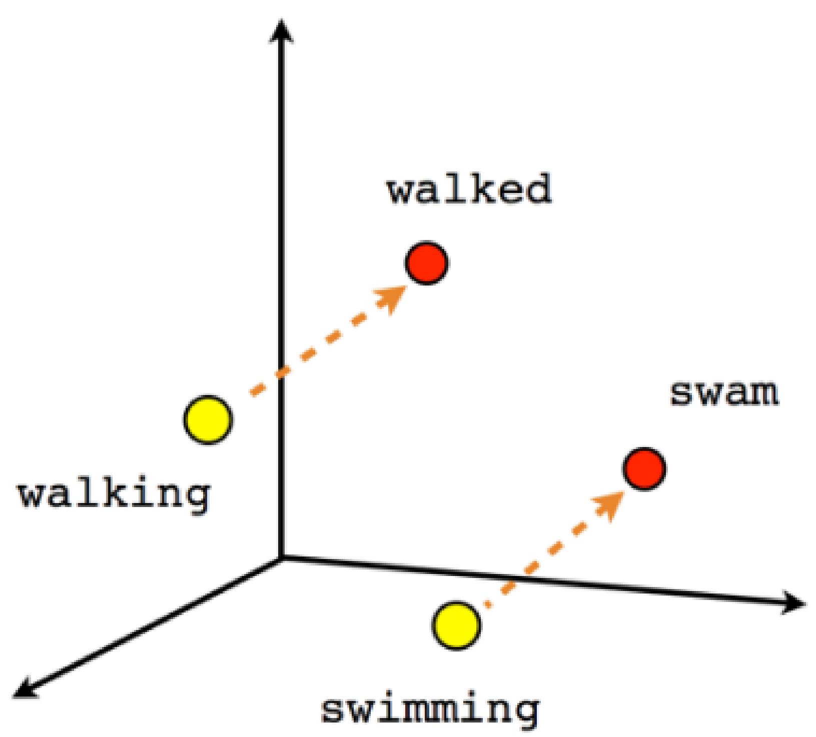
word2vec

- After learning, the hidden layer represents an **embedding vector**, which is a dense and compressed representation of each possible word (dimensionality reduction).
- Semantically close words (“endure” and “tolerate”) tend to appear in similar contexts, so their embedded representations will be close (Euclidian distance).
- One can even perform arithmetic operations on these vectors!

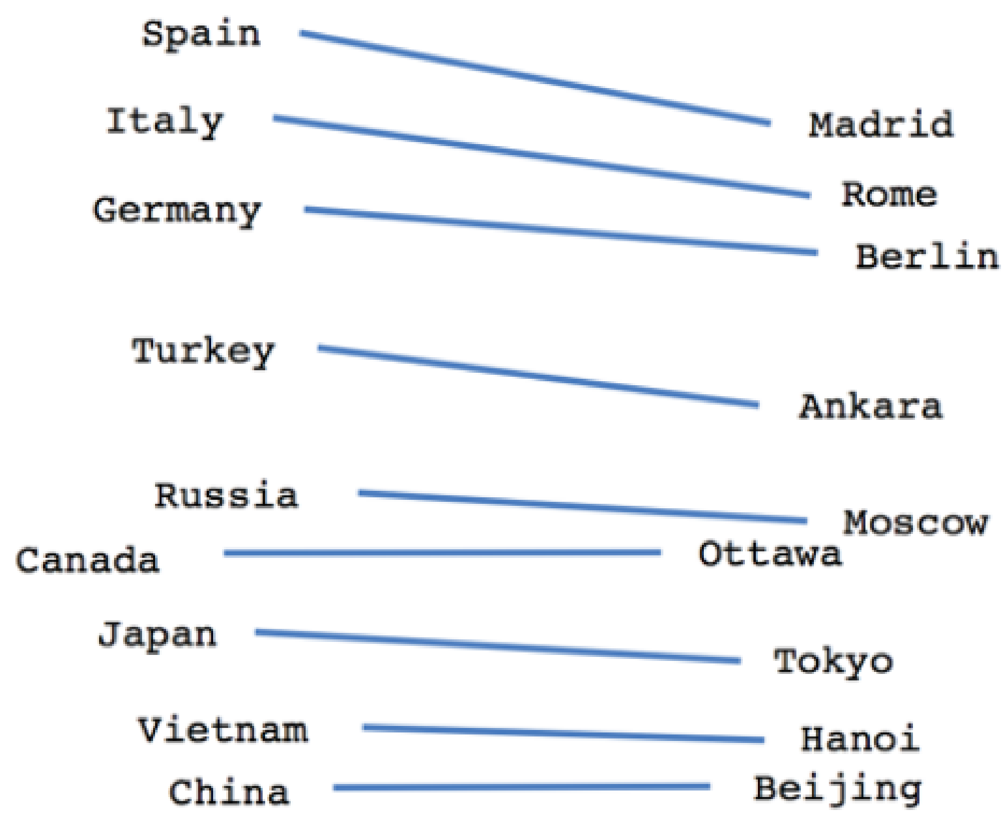
> queen = king + woman - man



Male-Female



Verb tense

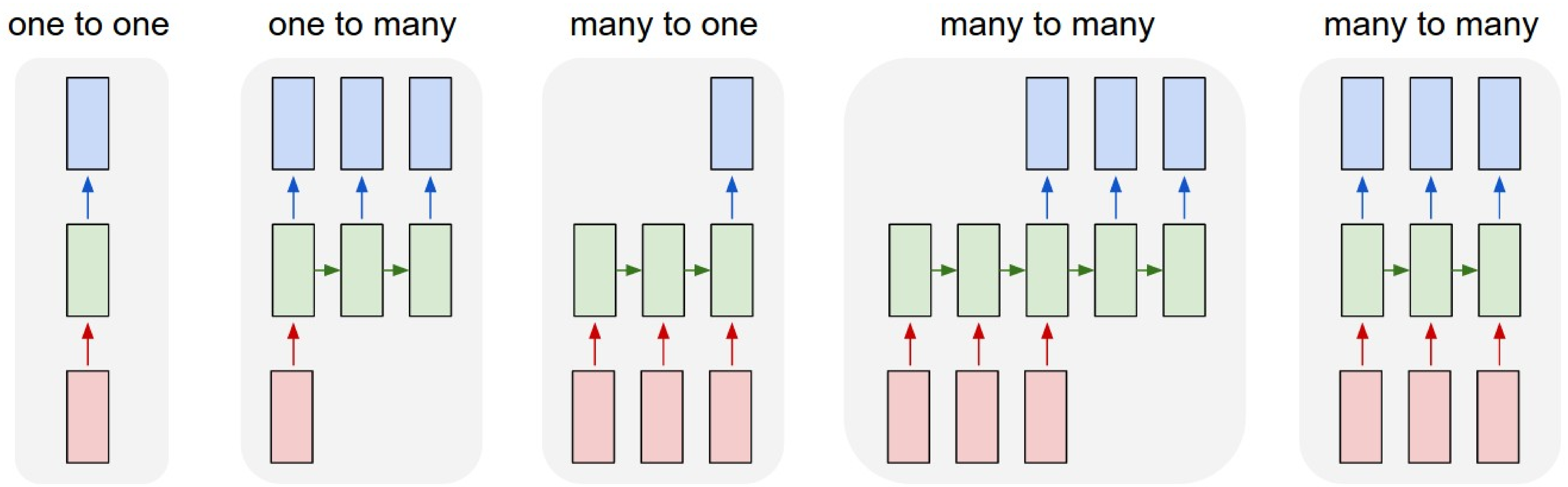


Country-Capital

Source : https://www.tensorflow.org/tutorials/representation/word2vec

2 - Applications of RNNs

Classification of LSTM architectures



Source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

- **One to One**: classical feedforward network.

  Image $\longrightarrow$ Label.
- **One to Many**: single input, many outputs.

  Image $\longrightarrow$ Text.
- **Many to One**: sequence of inputs, single output.

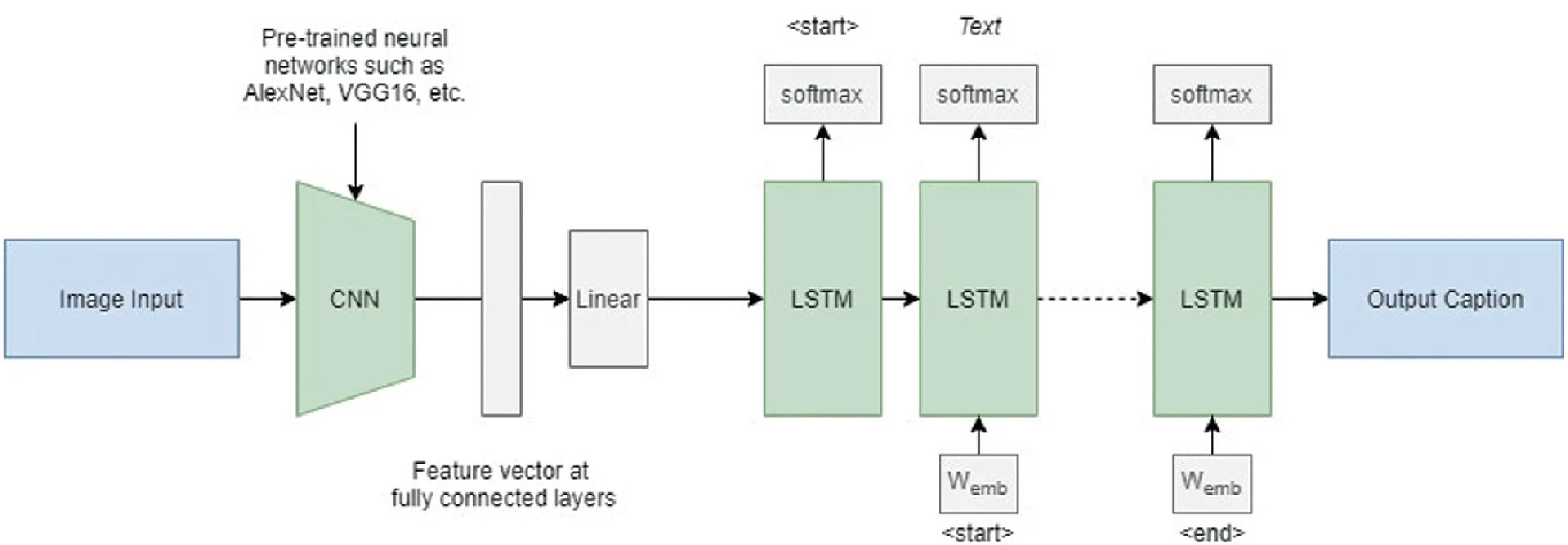
  Video / Text $\longrightarrow$ Label.
- **Many to Many**: sequence to sequence.

  Text $\longrightarrow$ Text.

  Video $\longrightarrow$ Text.

One to Many: image caption generation

- **Show and Tell** uses the last FC layer of a CNN to feed a LSTM layer and generate words.
- The pretrained CNN (VGG16, ResNet50) is used as a **feature extractor**.



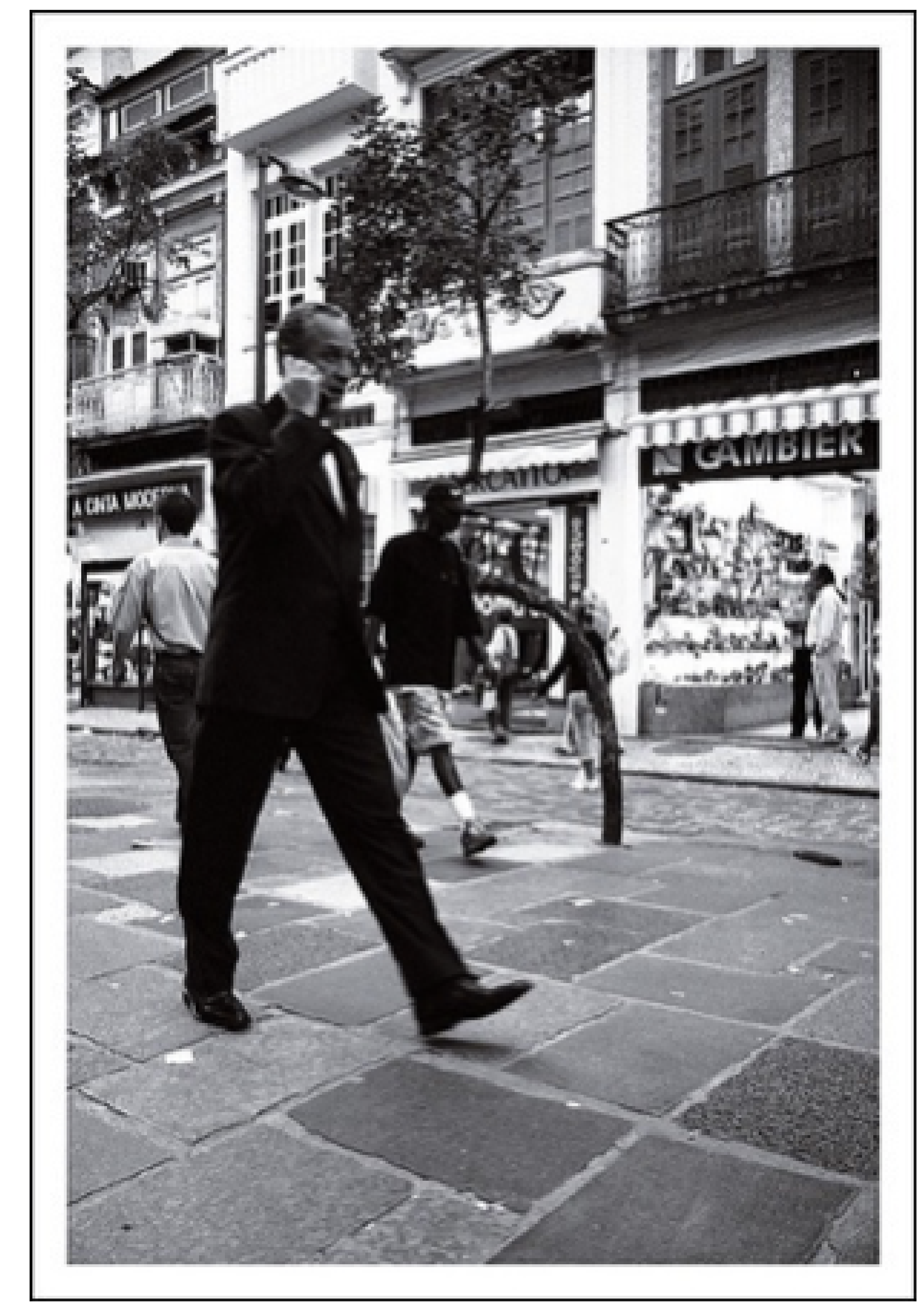
Source: Sathe et al. (2022). Overview of Image Caption Generators and Its Applications. ICCSA. https://doi.org/10.1007/978-981-19-0863-7_8

- Each word of the sentence is encoded/decoded using word2vec.
- The output of the LSTM at time t becomes its new input at time $t + 1$.

One to Many: image caption generation

↑ a living room with a couch and a television

↑ a man riding a bike on a beach

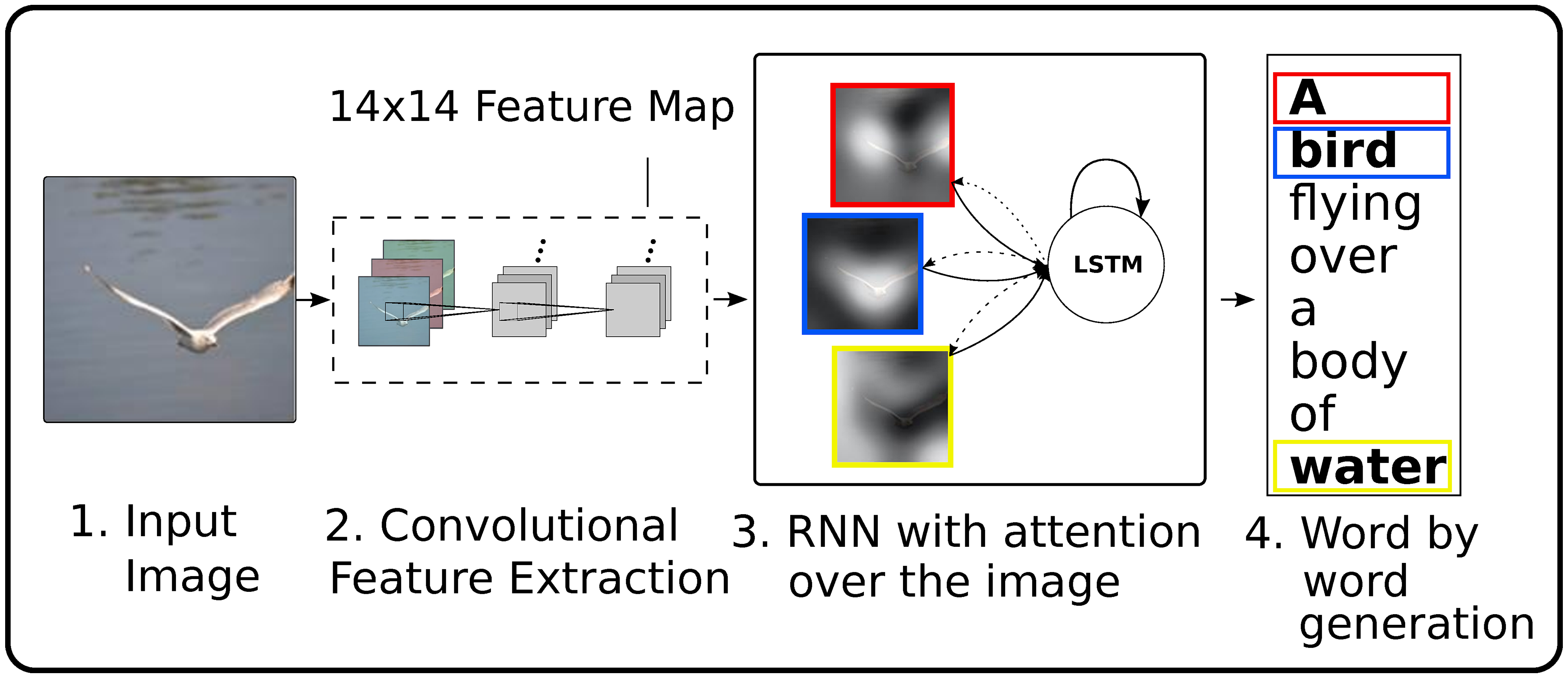


a man is walking down the street with a suitcase ↗

Xu et al. (2015). Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. NIPS.

One to Many: image caption generation

- **Show, attend and tell** uses attention to focus on specific parts of the image when generating the sentence.



Source: http://kelvinxu.github.io/projects/capgen.html

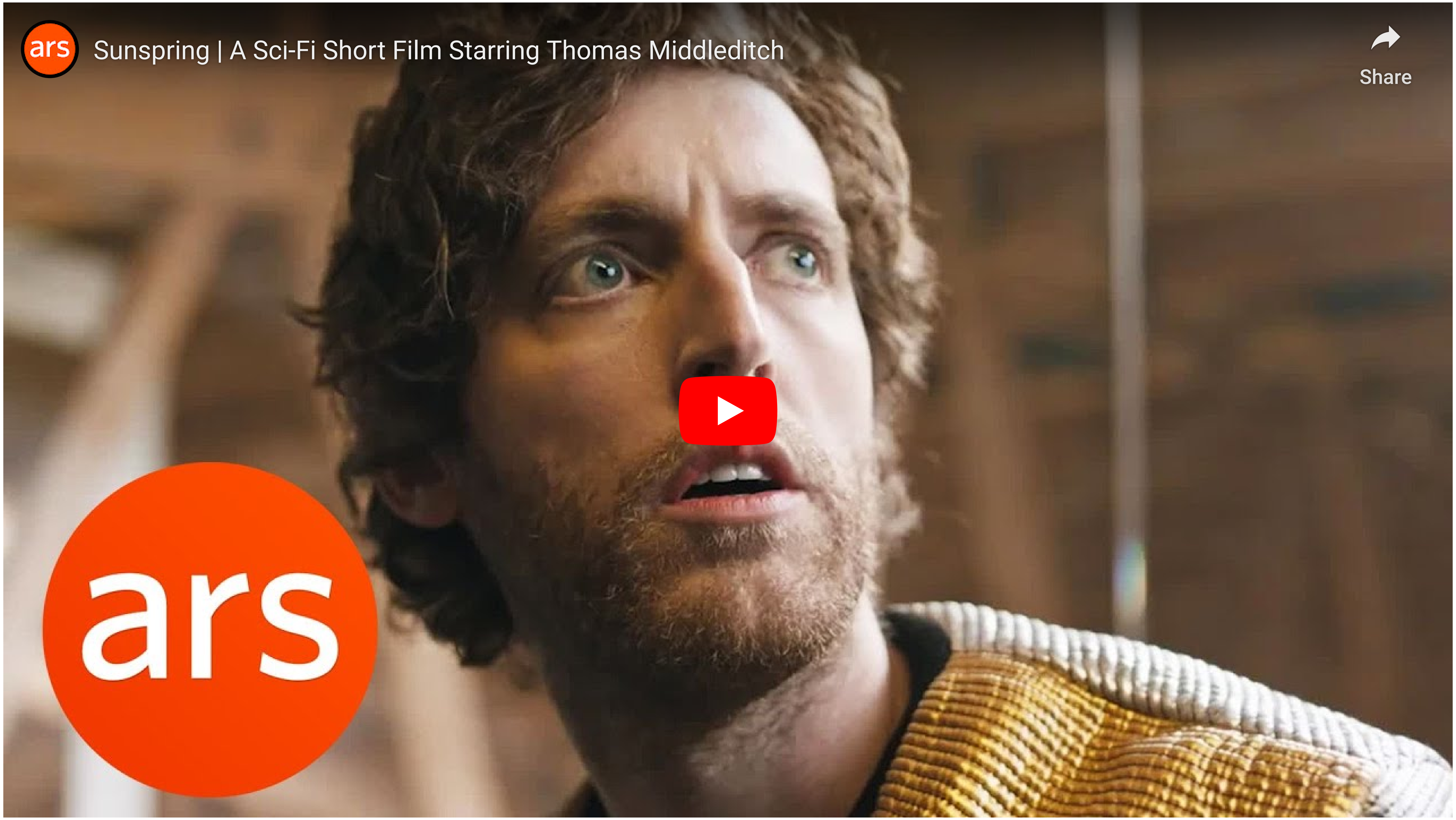
Many to One: next character/word prediction

```
PANDARUS:
Alas, I think he shall be come approached and
the day
When little srain would be attain'd into being
never fed,
And who is but a chain and subjects of his
death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my
soul,
Breaking and strongly should be buried, when I
perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.
```

- Characters or words are fed one by one into a LSTM.
- The desired output is the next character or word in the text.
- Example:
  - Inputs: **To, be, or, not, to**
  - Output: **be**
- The text on the left was generated by a LSTM having read the entire writings of William Shakespeare.
- Each generated word is used as the next input.

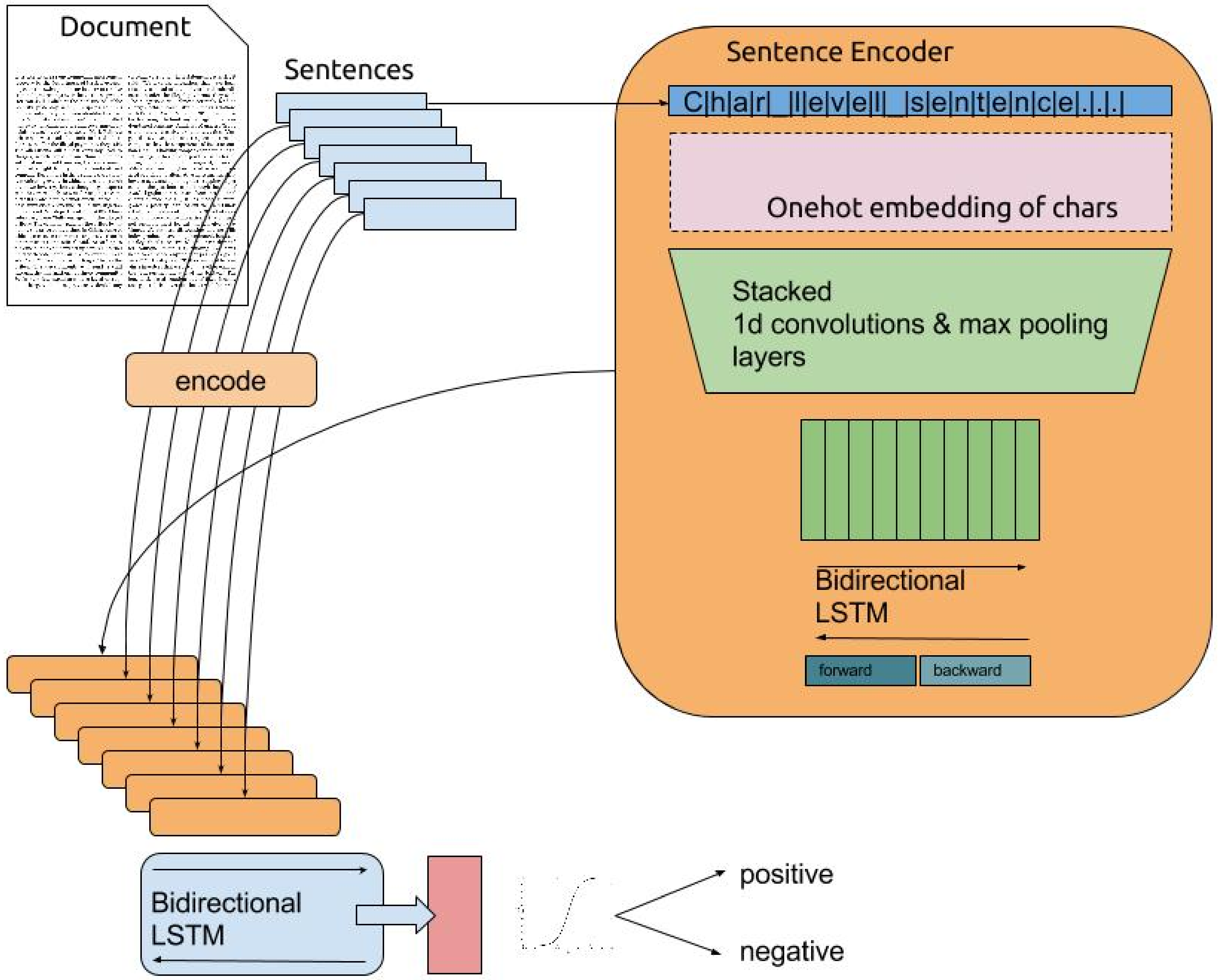
Source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Many to one: Sunspring SciFi movie



More info: http://www.thereforefilms.com/sunspring.html

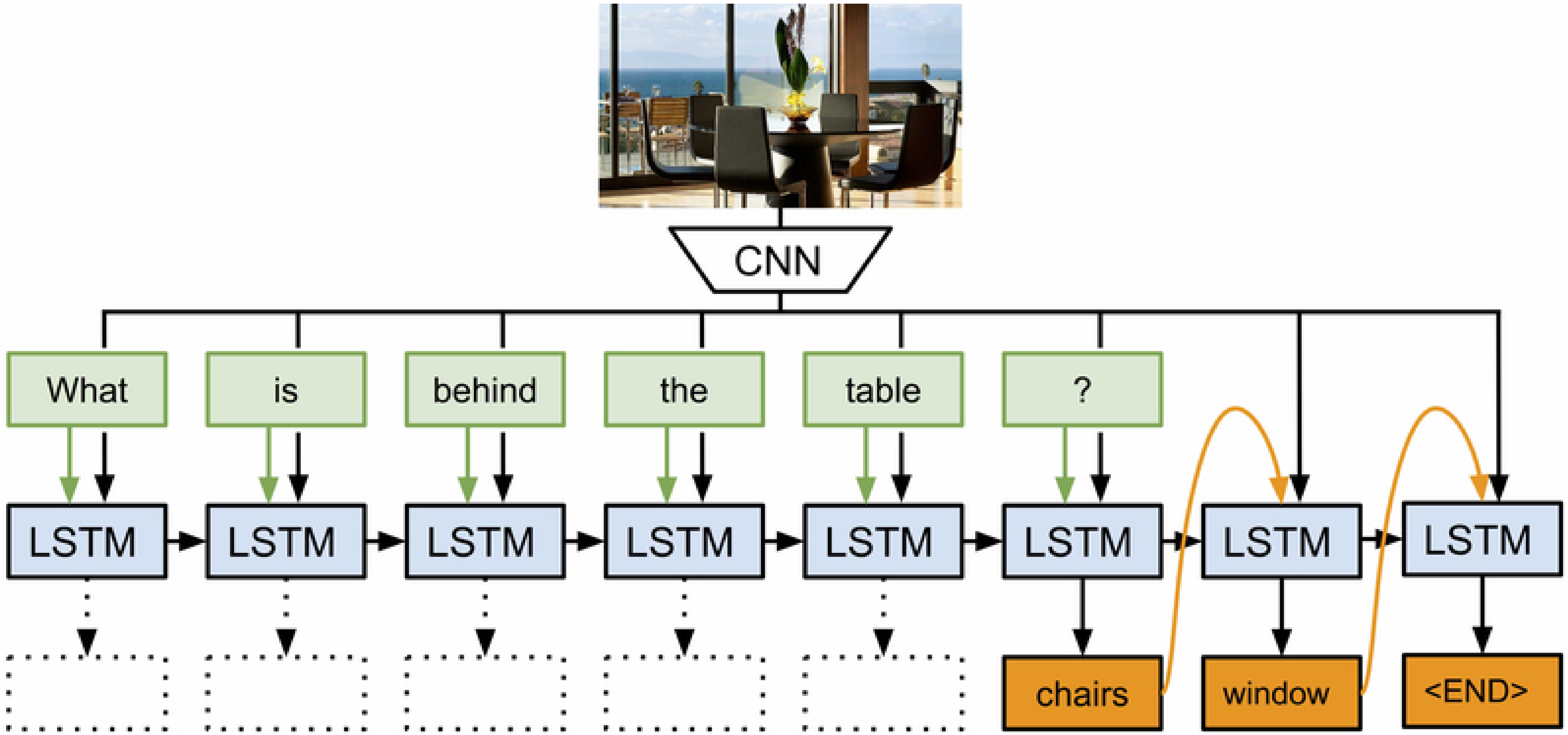
Many to One: sentiment analysis



- To obtain a vector from a sentence, **one-hot encoding** is used (alternative: word2vec).
- A 1D convolutional layers “slides” over the text.
- The bidirectional LSTM computes a state vector for the complete text.
- A classifier (fully connected layer) learns to predict the sentiment of the text (positive/negative).

Source: https://offbit.github.io/how-to-read/

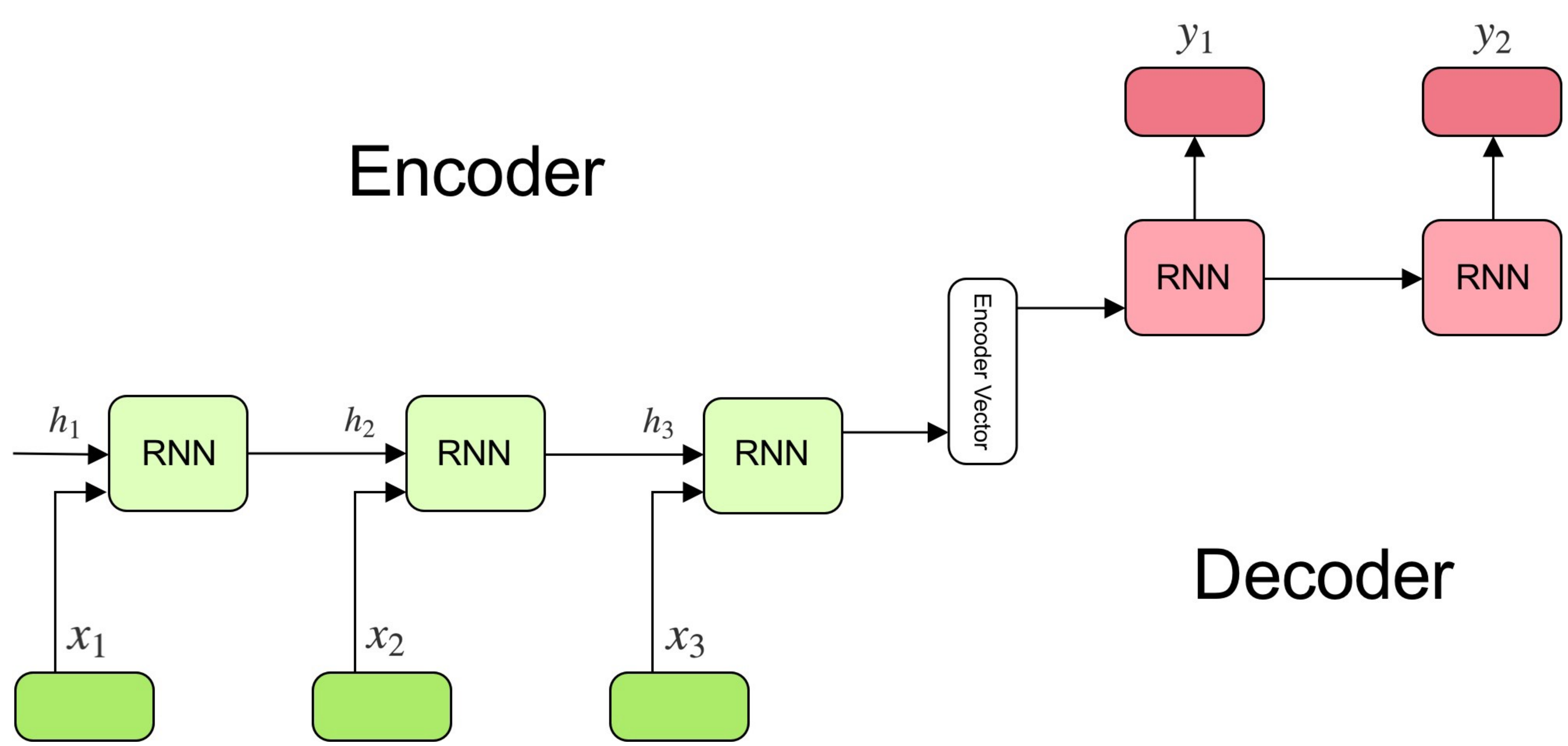
Many to Many: Question answering / Scene understanding



- A LSTM can learn to associate an image (static) plus a question (sequence) with the answer (sequence).
- The image is abstracted by a CNN trained for object recognition.

Many to Many: seq2seq

- The **state vector** obtained at the end of a sequence can be reused as an initial state for another LSTM.
- The goal of the **encoder** is to find a compressed representation of a sequence of inputs.
- The goal of the **decoder** is to generate a sequence from that representation.
- **Sequence-to-sequence** (seq2seq) models are recurrent autoencoders.



Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. arXiv:1409.3215

seq2seq for language translation

- The **encoder** learns for example to encode each word of a sentence in French.
- The **decoder** learns to associate the **final state vector** to the corresponding English sentence.
- seq2seq allows automatic text translation between many languages given enough data.
- Modern translation tools are based on seq2seq, but with attention.

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

3 - Attentional recurrent networks

Attentional recurrent networks

- The problem with seq2seq is that it **compresses** the complete input sentence into a single state vector.

- For long sequences, the beginning of the sentence may not be present in the final state vector:
    - Truncated BPTT, vanishing gradients.
    - When predicting the last word, the beginning of the paragraph might not be necessary.
- Consequence: there is not enough information in the state vector to start translating.

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Attentional recurrent networks

- A solution would be to concatenate the **state vectors** of all steps of the encoder and pass them to the decoder.

- **Problem 1:** it would make a lot of elements in the state vector of the decoder (which should be constant).
- **Problem 2:** the state vector of the decoder would depend on the length of the input sequence.

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

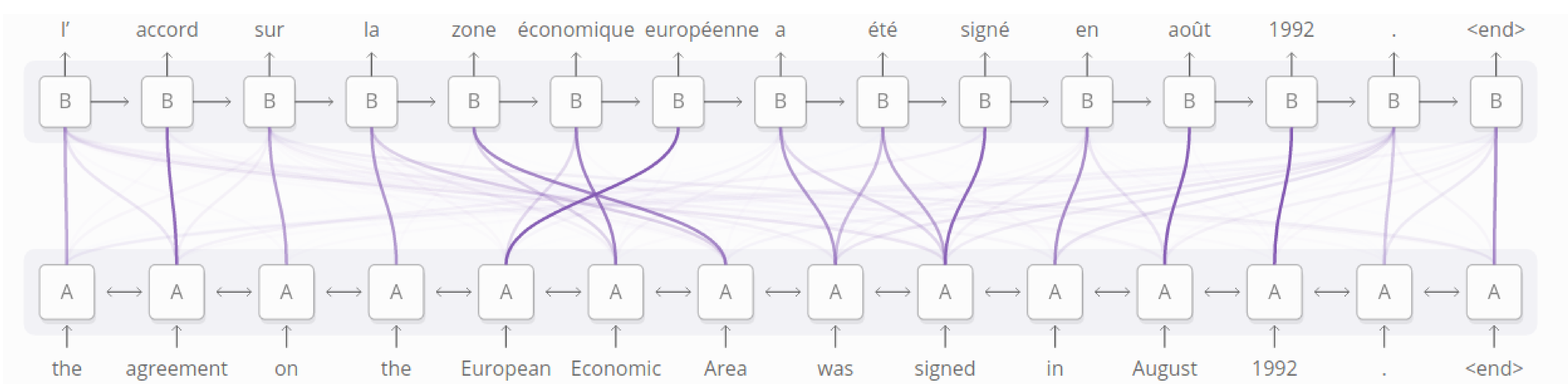
Attentional recurrent networks

- Attentional mechanisms let the decoder decide (by learning) which state vectors it needs to generate each word at each step.

- The **attentional context vector** of the decoder A_t^decoder at time t is a weighted average of all state vectors C_i^encoder of the encoder.

$$A_t^\text{decoder} = \sum_{i=0}^{T} a_i \, C_i^\text{encoder}$$

- The coefficients a_i are called the **attention scores** : how much attention is the decoder paying to each of the encoder's state vectors?



Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio (2014). Neural Machine Translation by Jointly Learning to Align and Translate. arXiv:1409.0473

Attentional recurrent networks

- The attention scores a_i are computed as a **softmax** over the scores e_i (in order to sum to 1):

$$a_i = \frac{\exp e_i}{\sum_j \exp e_j} \Rightarrow A_t^\text{decoder} = \sum_{i=0}^T \frac{\exp e_i}{\sum_j \exp e_j} \, C_i^\text{encoder}$$

- The score e_i is computed using:
  - the previous output of the decoder $\mathbf{h}_{t-1}^\text{decoder}$.
  - the corresponding state vector C_i^encoder of the encoder at step i.
  - attentional weights W_a.

$$e_i = \tanh(W_a \, [\mathbf{h}_{t-1}^\text{decoder}; C_i^\text{encoder}])$$

- Everything is differentiable, these attentional weights can be learned with BPTT.

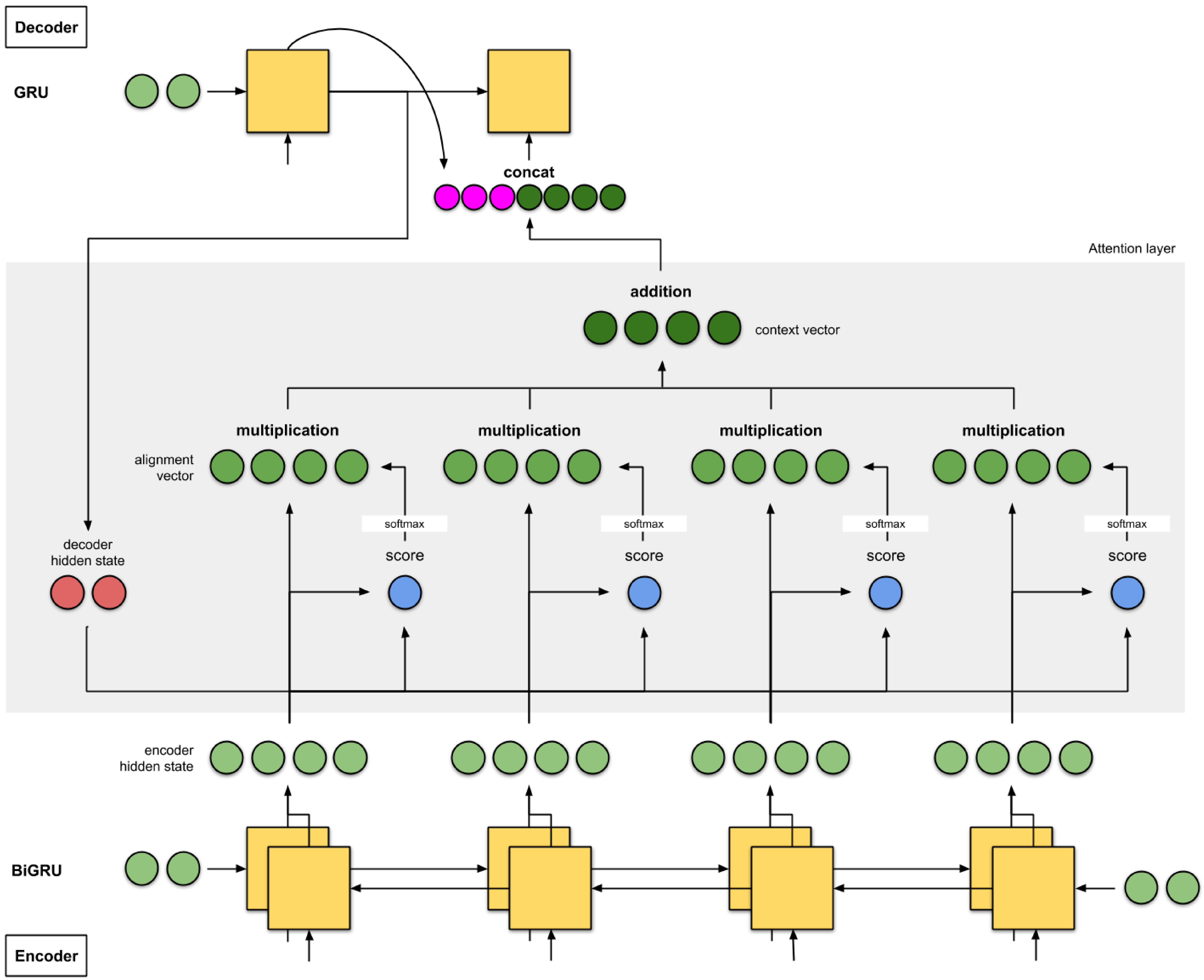
https://towardsdatascience.com/day-1-2-attention-seq2seq-models-65df3f49e263

Attentional recurrent networks

- The attentional context vector A_t^{decoder} is concatenated with the previous output $\mathbf{h}_{t-1}^{\text{decoder}}$ and used as the next input $\mathbf{x}_t^{\text{decoder}}$ of the decoder:

$$\mathbf{x}_t^{\text{decoder}} = [\mathbf{h}_{t-1}^{\text{decoder}}; A_t^{\text{decoder}}]$$

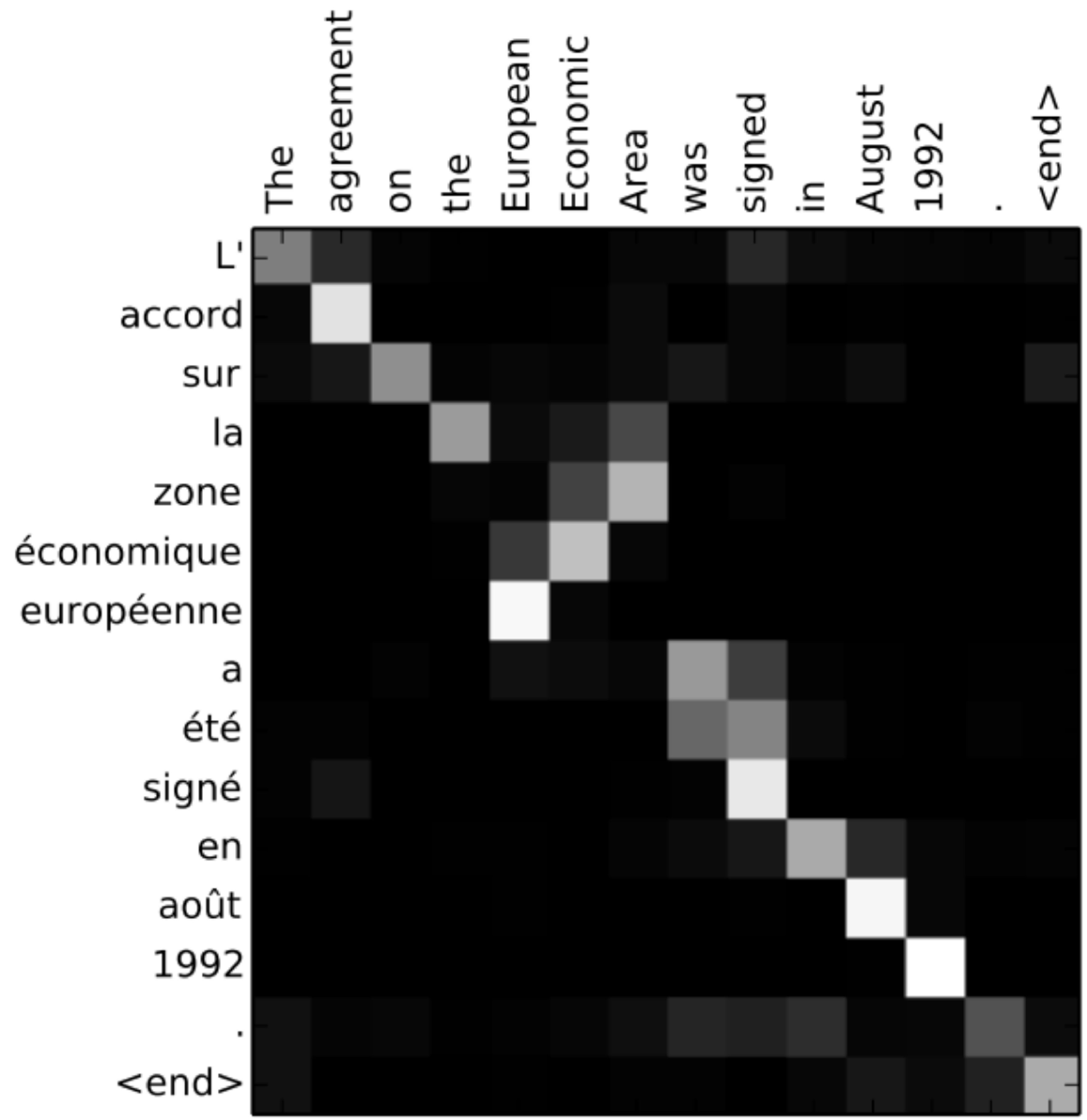
https://towardsdatascience.com/day-1-2-attention-seq2seq-models-65df3f49e263



https://towardsdatascience.com/day-1-2-attention-seq2seq-models-65df3f49e263

Attentional recurrent networks

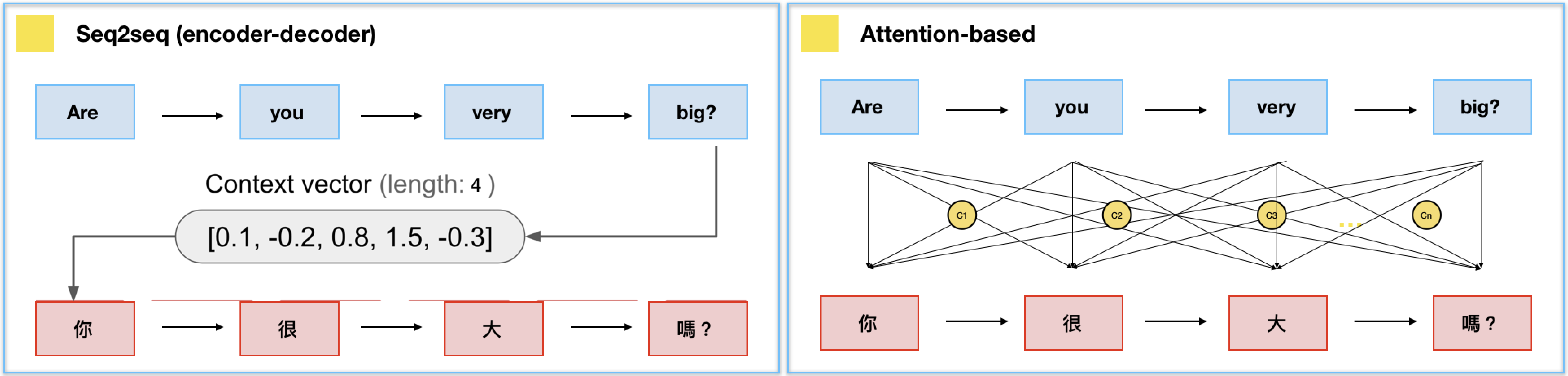
- The attention scores or **alignment scores** a_i are useful to interpret what happened.
- They show which words in the original sentence are the most important to generate the next word.



Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio (2014). Neural Machine Translation by Jointly Learning to Align and Translate. arXiv:1409.0473

Attentional recurrent networks

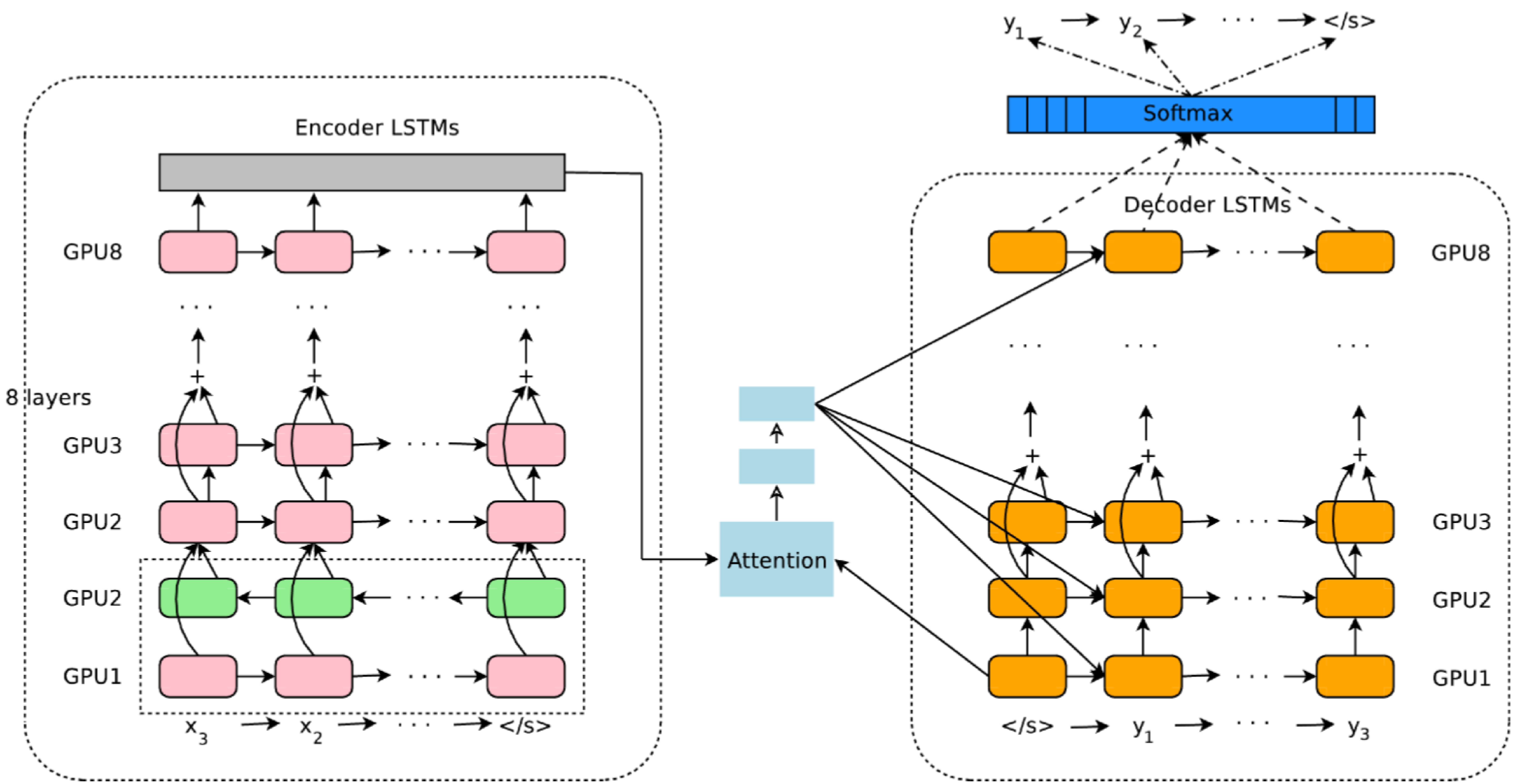
- **Attentional mechanisms** are now central to NLP.



- The whole **history** of encoder states is passed to the decoder, which learns to decide which part is the most important using **attention**.
- This solves the bottleneck of seq2seq architectures, at the cost of much more operations.
- They require to use fixed-length sequences (generally 50 words).

Google Neural Machine Translation (GNMT)

- Google Neural Machine Translation (GNMT) uses an attentional recurrent NN, with bidirectional GRUs, 8 recurrent layers on 8 GPUs for both the encoder and decoder.



Wu et al. (2016). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. arXiv:1609.08144v2