CSC 578
Neural Networks and Deep Learning

8. Recurrent Neural Networks
-- More

(Some slides adapted from Ming Li at U Waterloo)

Recurrent Neural Networks

1. Some implementation hints on RNN in Keras
    1. Recurrent dropout
    2. Stateful-ness
2. Encoder-Decoder
3. Attention Model
    1. Neural Machine Translation (NMT)
    2. Image caption generation
    3. [supplement] Machine Translation

1.1 Recurrent dropout

- Keras has two dropout-related arguments:
  - 'dropout' specifies the dropout rate for input units of the layer,
  - 'recurrent_dropout' specifies the dropout rate of the recurrent units.

```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.GRU(32,
                     dropout=0.2,
                     recurrent_dropout=0.2,
                     input_shape=(None, float_data.shape[-1])))
model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mae')
history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=40,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```


https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/6.3-advanced-usage-of-recurrent-neural-networks.ipynb

1.2 Stateful-ness of Recurrent Networks

- In Keras, internal states (memories) obtained after processing a batch of samples are discarded after the batch, and a set of new states with zeros will be created for the next batch.
- For longer sequences or sequences that have repeating patterns over rows, you may want to re-use the learned states from the previous batch.

When using stateful RNNs, it is therefore assumed that:

- all batches have the same number of samples
- If `x1` and `x2` are successive batches of samples, then `x2[i]` is the follow-up sequence to `x1[i]`, for every `i`.

To use statefulness in RNNs, you need to:

- explicitly specify the batch size you are using, by passing a `batch_size` argument to the first layer in your model. E.g. `batch_size=32` for a 32-samples batch of sequences of 10 timesteps with 16 features per timestep.
- set `stateful=True` in your RNN layer(s).
- specify `shuffle=False` when calling fit().

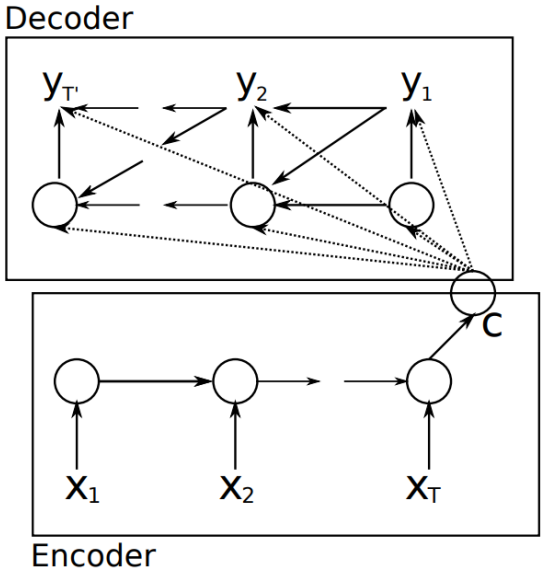
https://keras.io/getting-started/sequential-model-guide/

```python
from keras.models import Sequential
from keras.layers import LSTM, Dense
import numpy as np

data_dim = 16
timesteps = 8
num_classes = 10
batch_size = 32

# Expected input batch shape: (batch_size, timesteps, data_dim)
# Note that we have to provide the full batch_input_shape since the network is stateful.
# the sample of index i in batch k is the follow-up for the sample i in batch k-1.
model = Sequential()
model.add(LSTM(32, return_sequences=True, stateful=True,
               batch_input_shape=(batch_size, timesteps, data_dim)))
model.add(LSTM(32, return_sequences=True, stateful=True))
model.add(LSTM(32, stateful=True))
model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

# Generate dummy training data
x_train = np.random.random((batch_size * 10, timesteps, data_dim))
y_train = np.random.random((batch_size * 10, num_classes))

# Generate dummy validation data
x_val = np.random.random((batch_size * 3, timesteps, data_dim))
y_val = np.random.random((batch_size * 3, num_classes))

model.fit(x_train, y_train,
          batch_size=batch_size, epochs=5, shuffle=False,
          validation_data=(x_val, y_val))
```

2 Encoder-Decoder

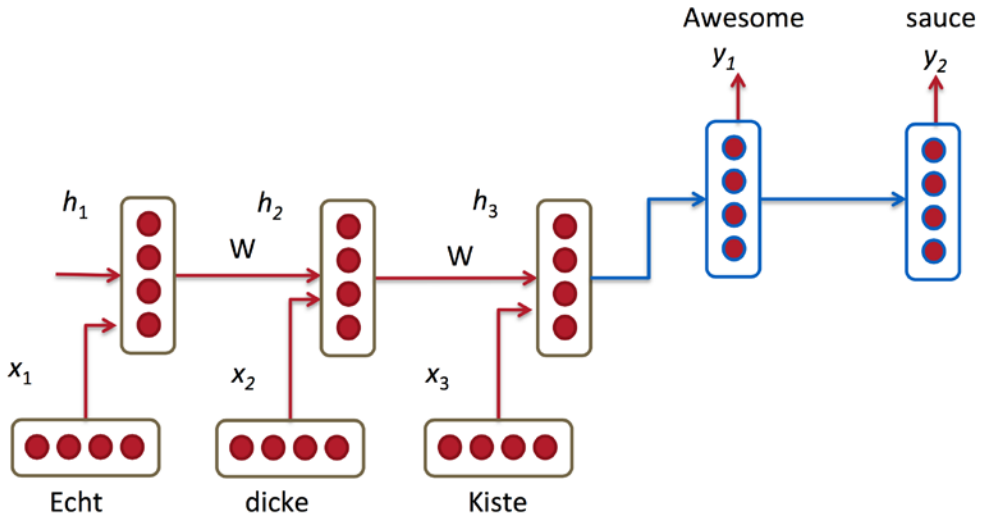
- The **Encoder-Decoder** mechanism for RNNs is suited for sequence-to-sequence problems, especially when input sequences differ in length from output sequences.

- From a high-level, the model is comprised of two sub-models: an encoder and a decoder.
  - **Encoder**: The encoder is responsible for stepping through the input time steps and encoding the entire sequence into a fixed length vector called a ***context vector***.
  - **Decoder**: The decoder is responsible for stepping through the output time steps while reading from the context vector.



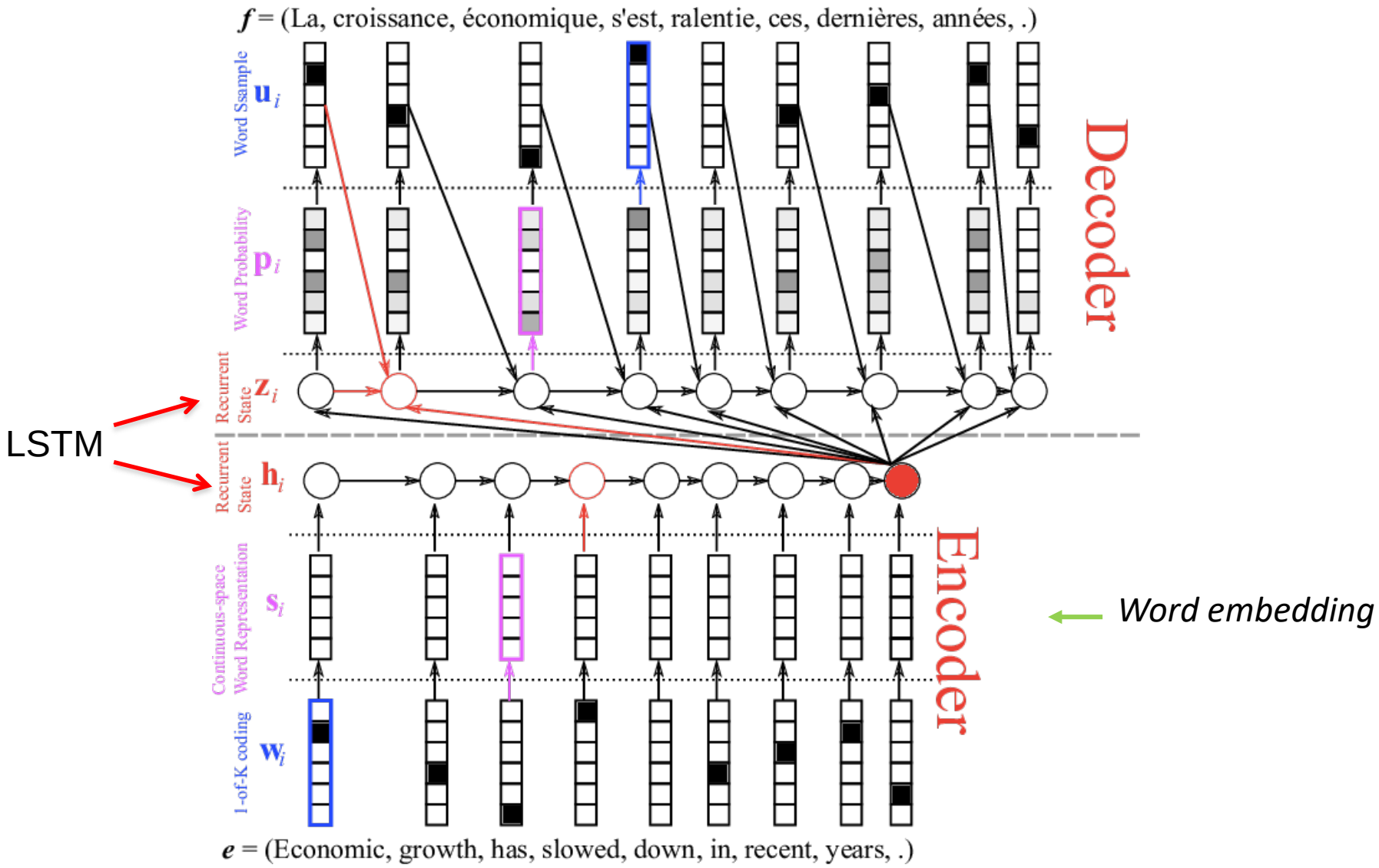
https://machinelearningmastery.com/how-does-attention-work-in-encoder-decoder-recurrent-neural-networks/

- For example, Encoder-Decoder models fit very well to Machine Translation, where a sentence (sequence of words) in the source language is mapped to a sentence in the target language.

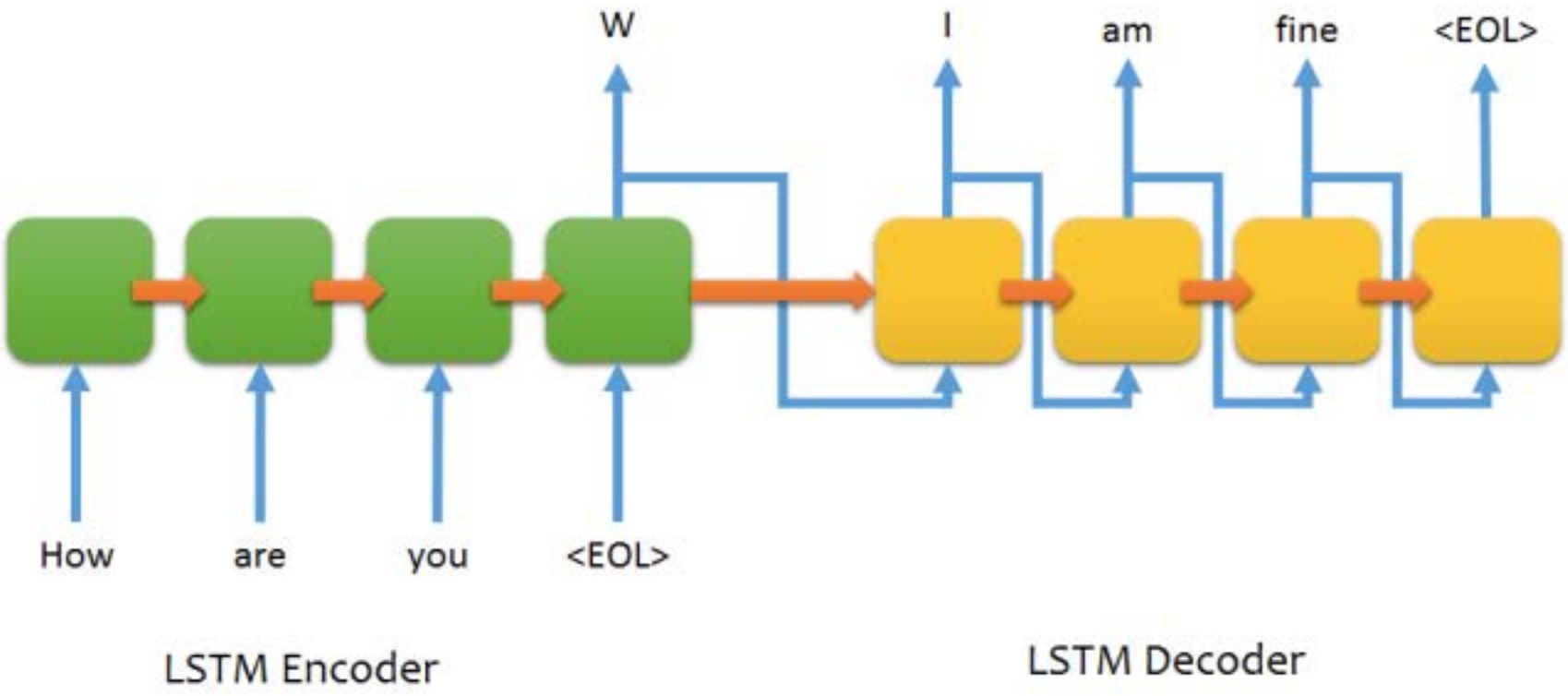


- However, if you look closely, you can see that the decoder is supposed to generate a translation solely based on the last hidden state (h_3 above) from the encoder…
- This issue is believed to be more of a problem when **decoding long sequences**.

Neural machine translation (NMT)

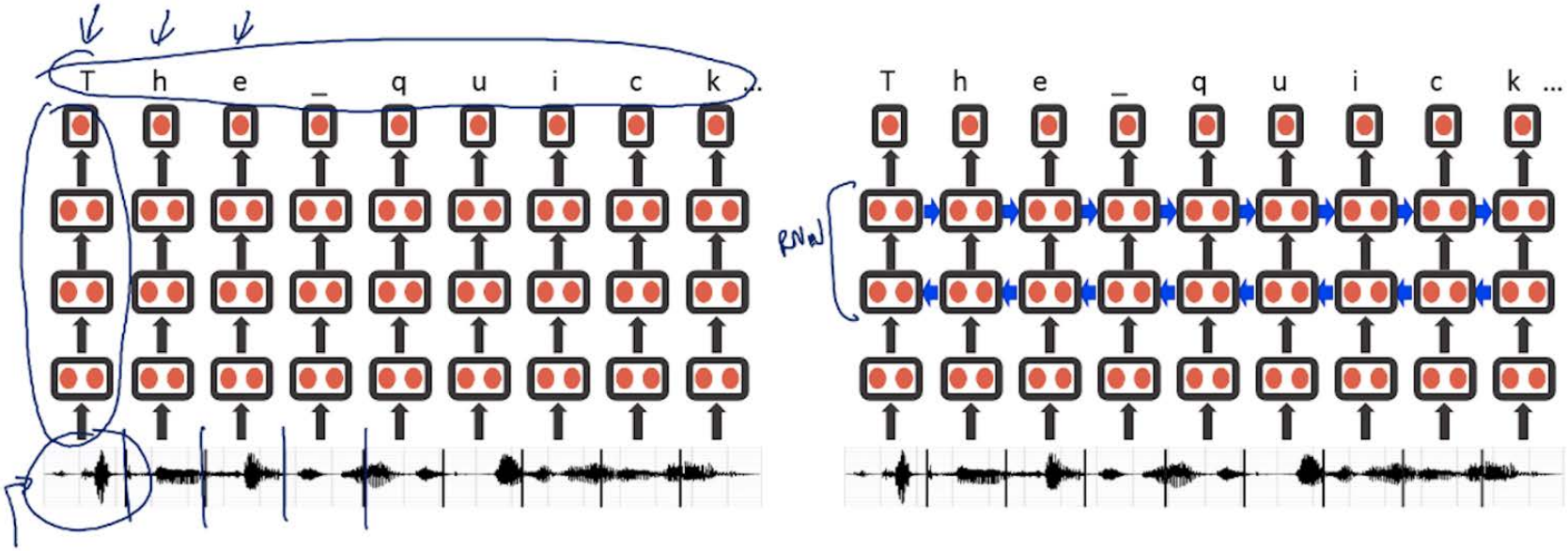


Sequence to sequence chat model



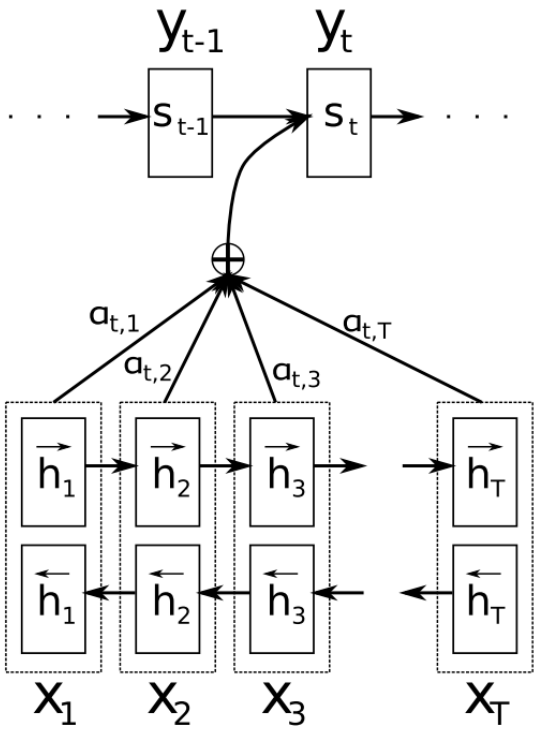
Baidu's speech recognition using RNN

Speech recognition example (Deep Speech)



3 Attention Model

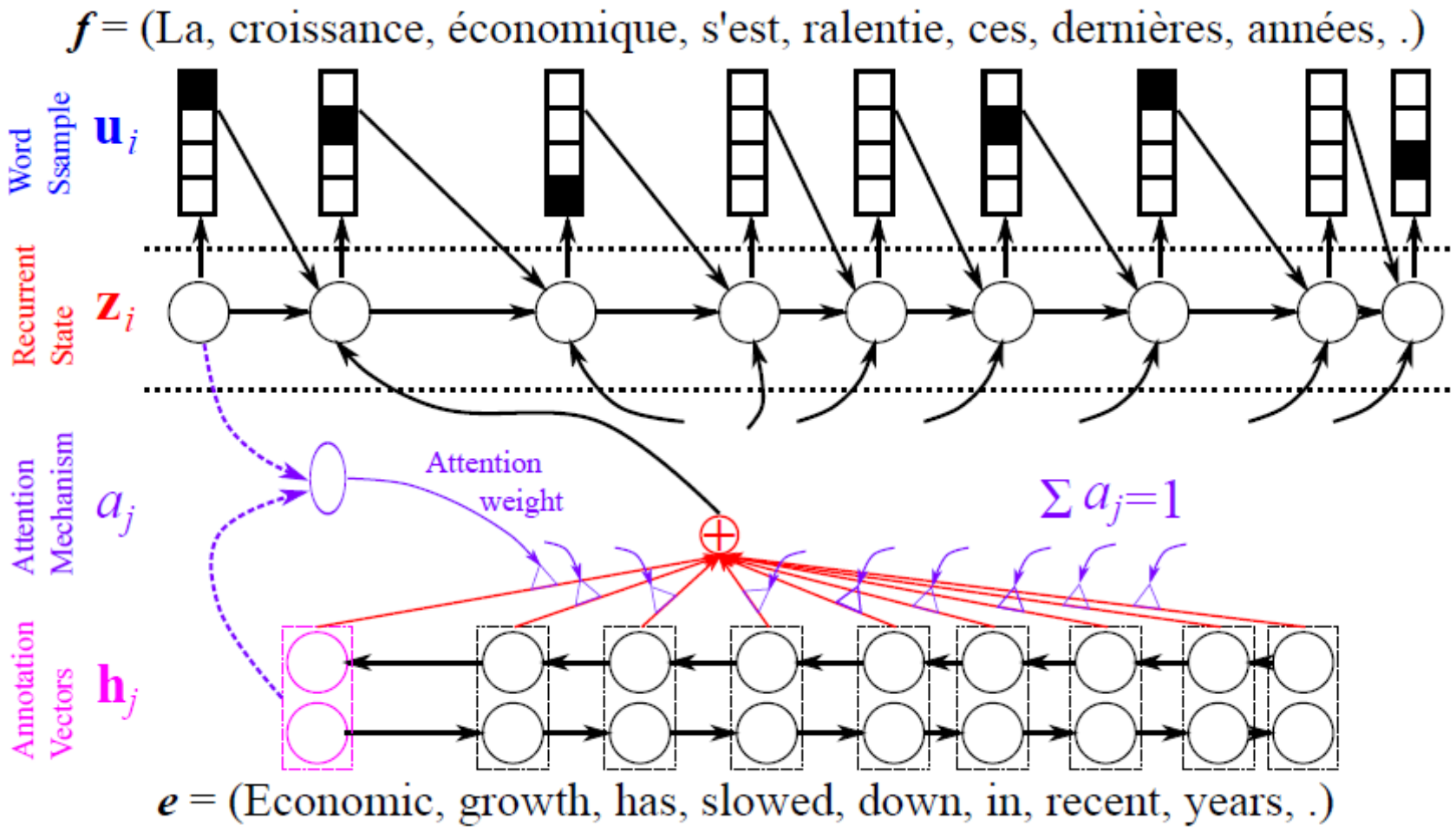
- An **Attention Model** tries to solve the Encoder-Decoder's long sequence generation problem by allowing the decoder to "attend" to different parts of the source sequence at each step of the output generation.



Each decoder output word y_t now depends on a **weighted combination** of all the input states, not just the last state.

The a's are weights that define how much of each input state should be considered for each output.

3.1 Neural Machine Translation (NMT)



From Y. Bengio CVPR 2015 Tutorial

Soft Attention for Translation

Context vector (input to decoder):

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

Mixture weights:

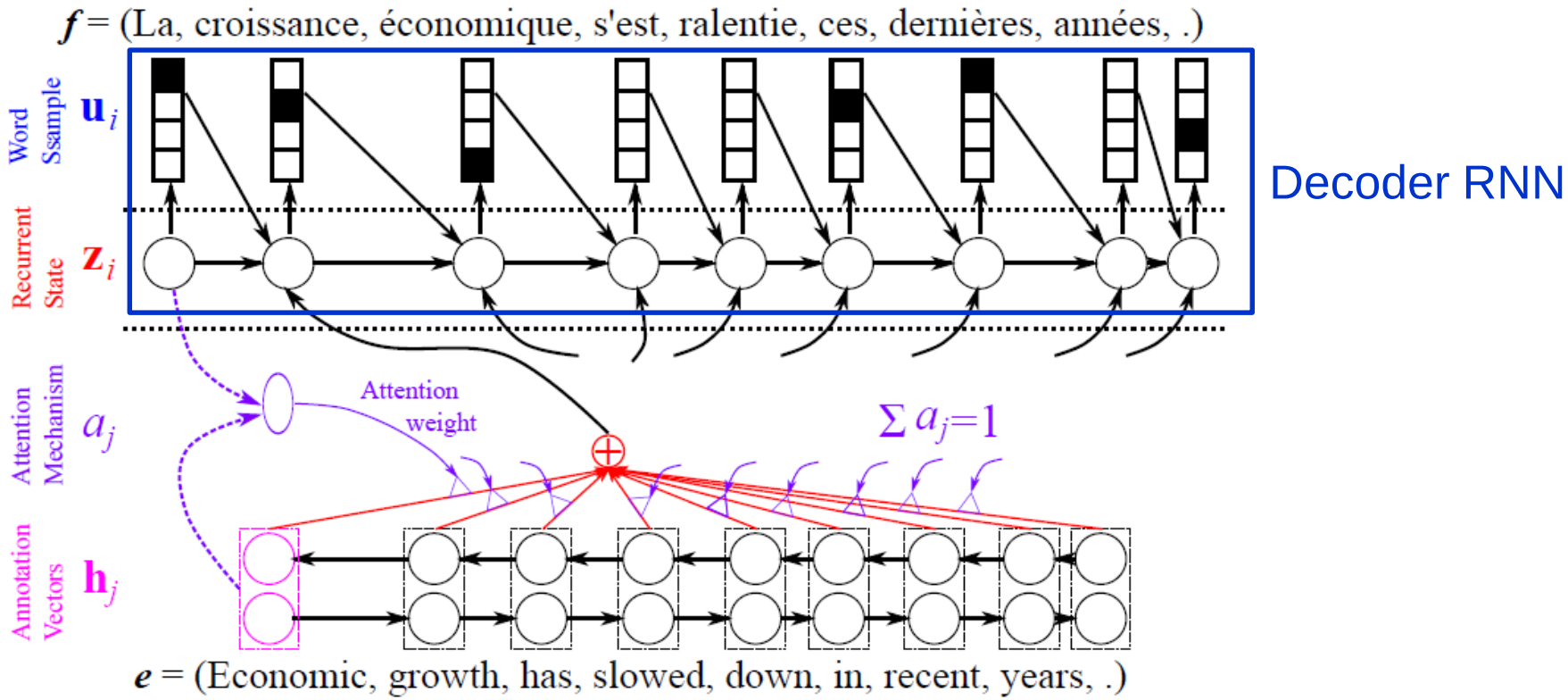
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

Alignment score (how well do input words near j match output words at position i):

$$e_{ij} = a(s_{i-1}, h_j)$$

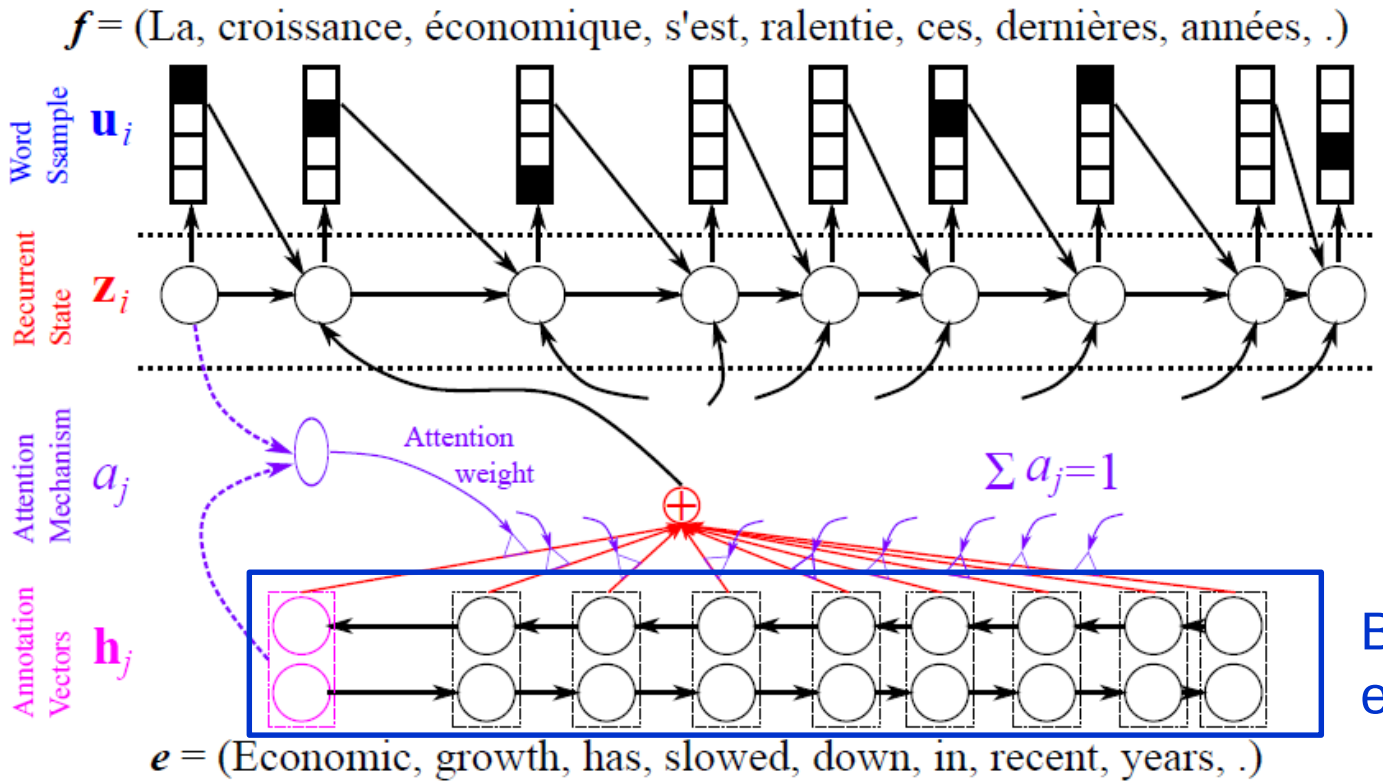
Bahdanau et al, “Neural Machine Translation by Jointly Learning to Align and Translate”, ICLR 2015

Soft Attention for Translation



From Y. Bengio CVPR 2015 Tutorial

Soft Attention for Translation

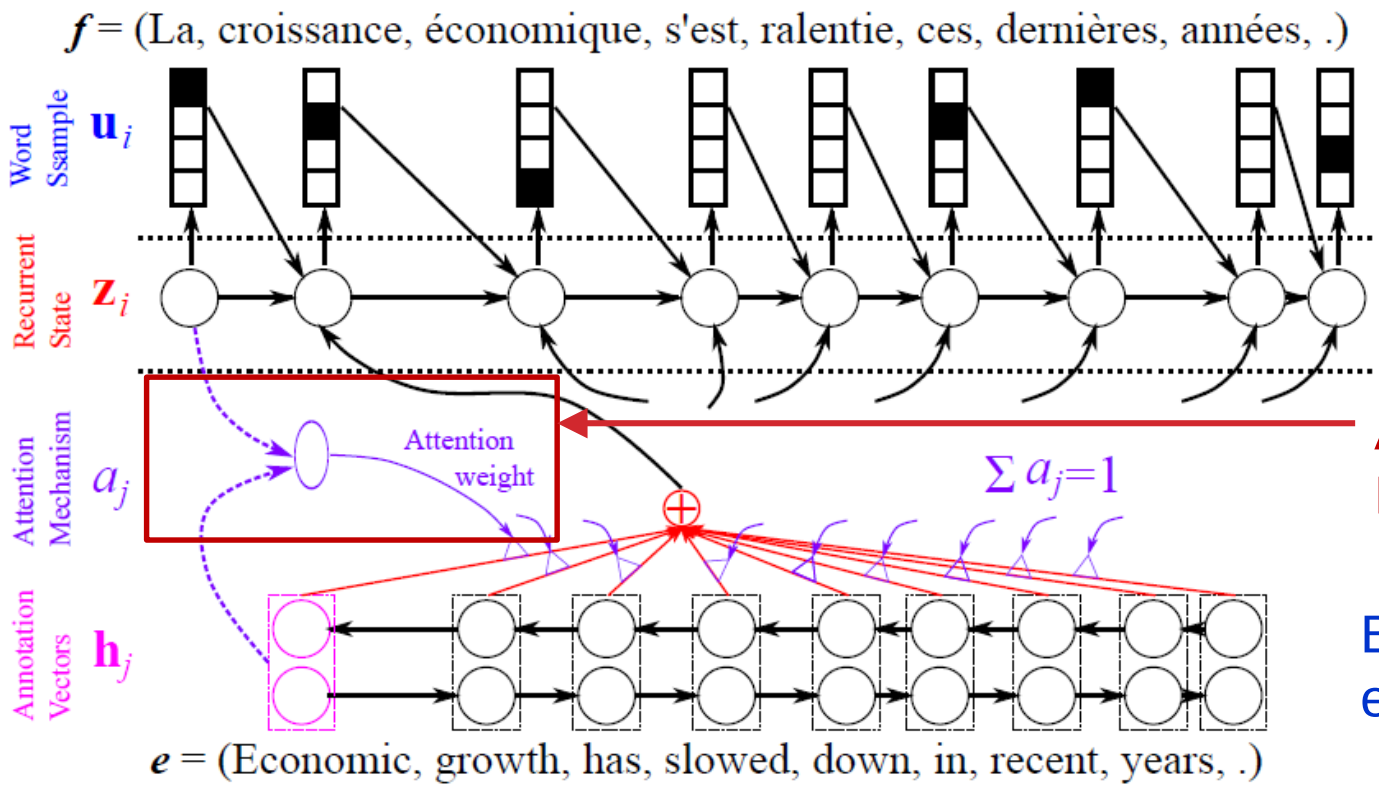


Decoder RNN

Bidirectional encoder RNN

From Y. Bengio CVPR 2015 Tutorial

Soft Attention for Translation



Decoder RNN

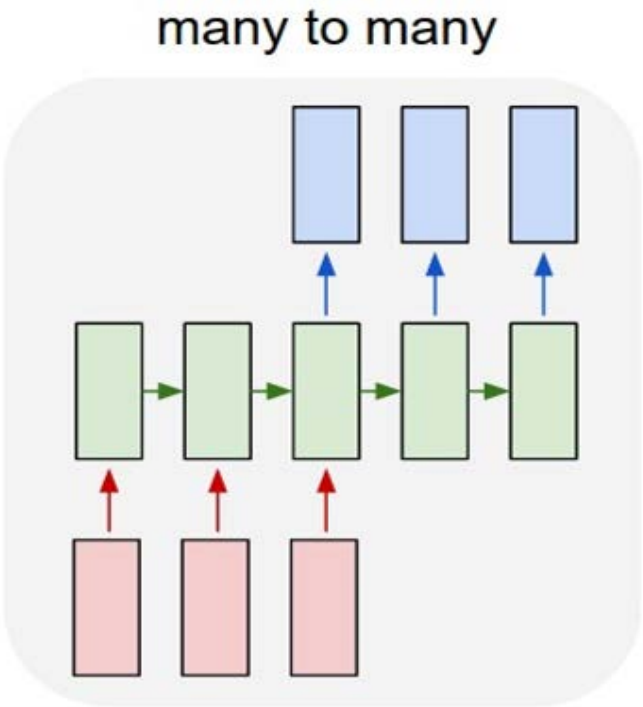
Attention Model

Bidirectional encoder RNN

From Y. Bengio CVPR 2015 Tutorial

Soft Attention for Translation

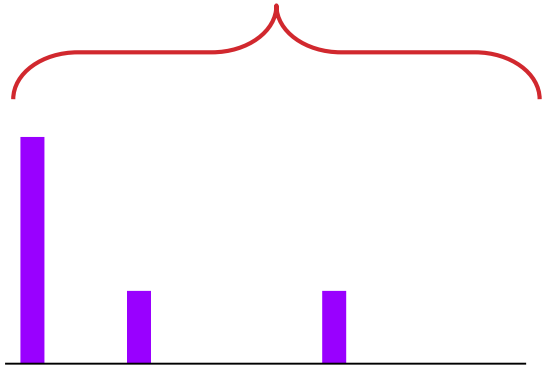
“I love coffee” -> “Me gusta el café”



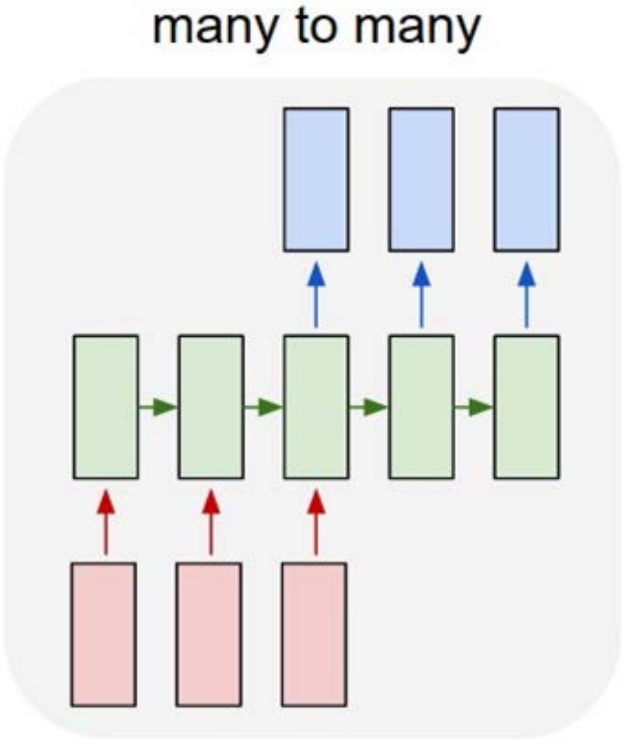
Bahdanau et al, “Neural Machine Translation by Jointly Learning to Align and Translate”, ICLR 2015

Soft Attention for Translation

Distribution over input words



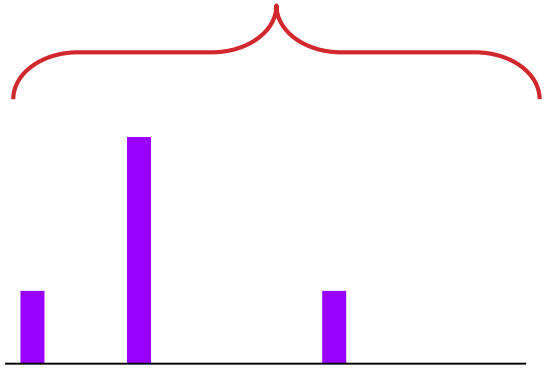
“I love coffee” -> “Me gusta el café”



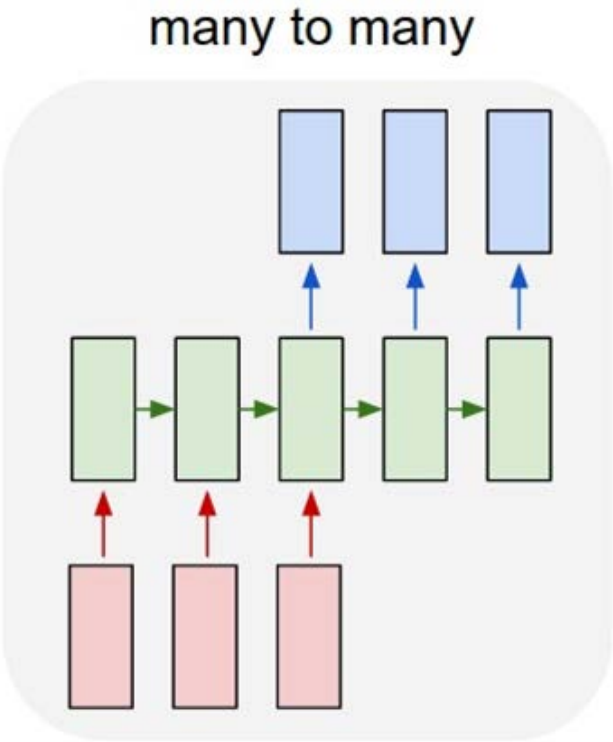
Bahdanau et al, “Neural Machine Translation by Jointly Learning to Align and Translate”, ICLR 2015

Soft Attention for Translation

Distribution over input words



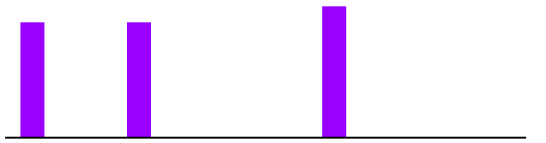
“I love coffee” -> “Me gusta el café”



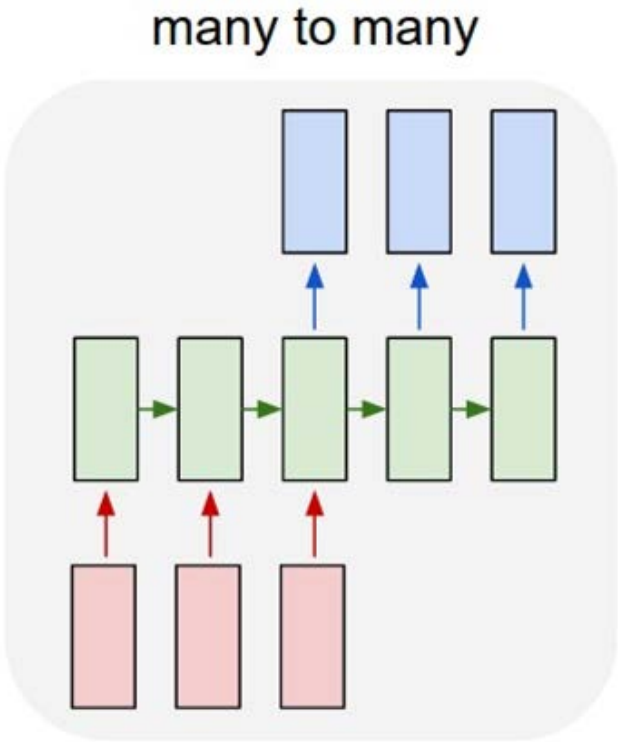
Bahdanau et al, “Neural Machine Translation by Jointly Learning to Align and Translate”, ICLR 2015

Soft Attention for Translation

Distribution over input words



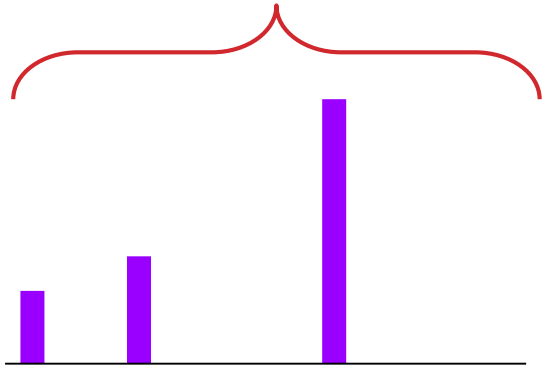
“I love coffee” -> “Me gusta el café”



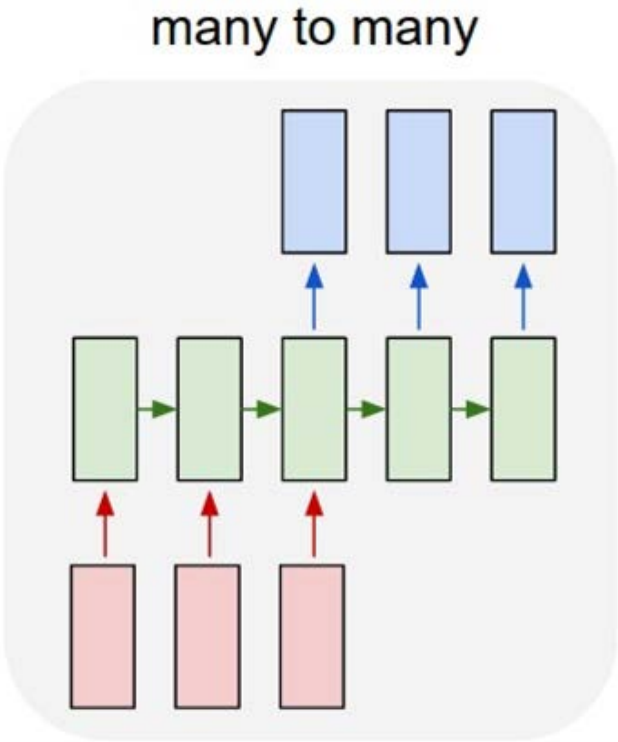
Bahdanau et al, “Neural Machine Translation by Jointly Learning to Align and Translate”, ICLR 2015

Soft Attention for Translation

Distribution over input words

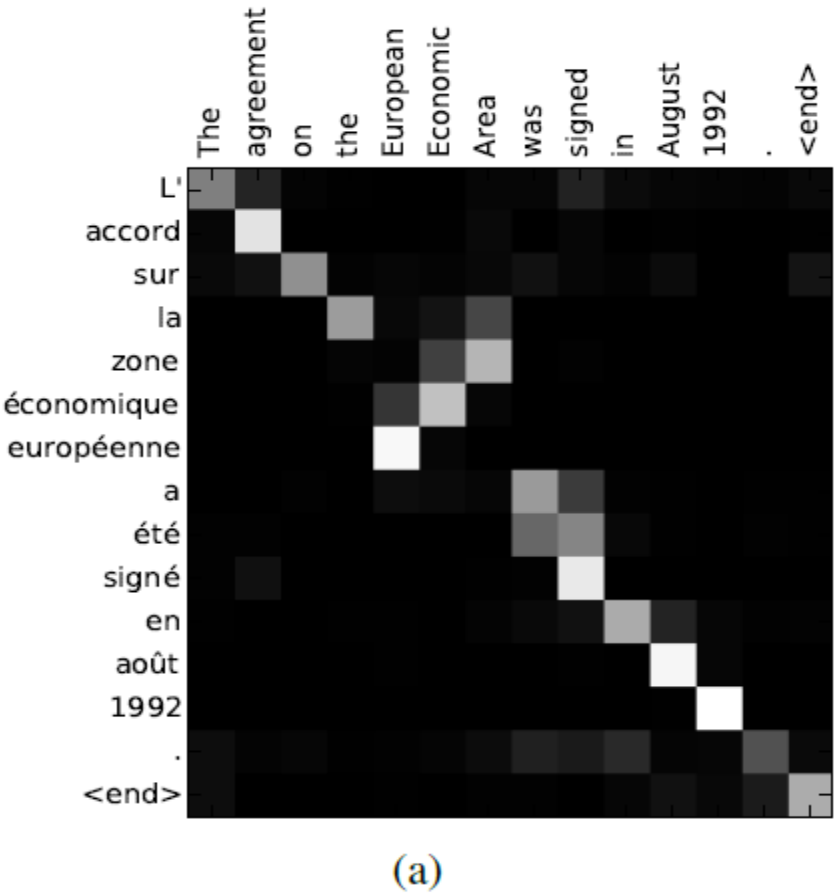


“I love coffee” -> “Me gusta el café”

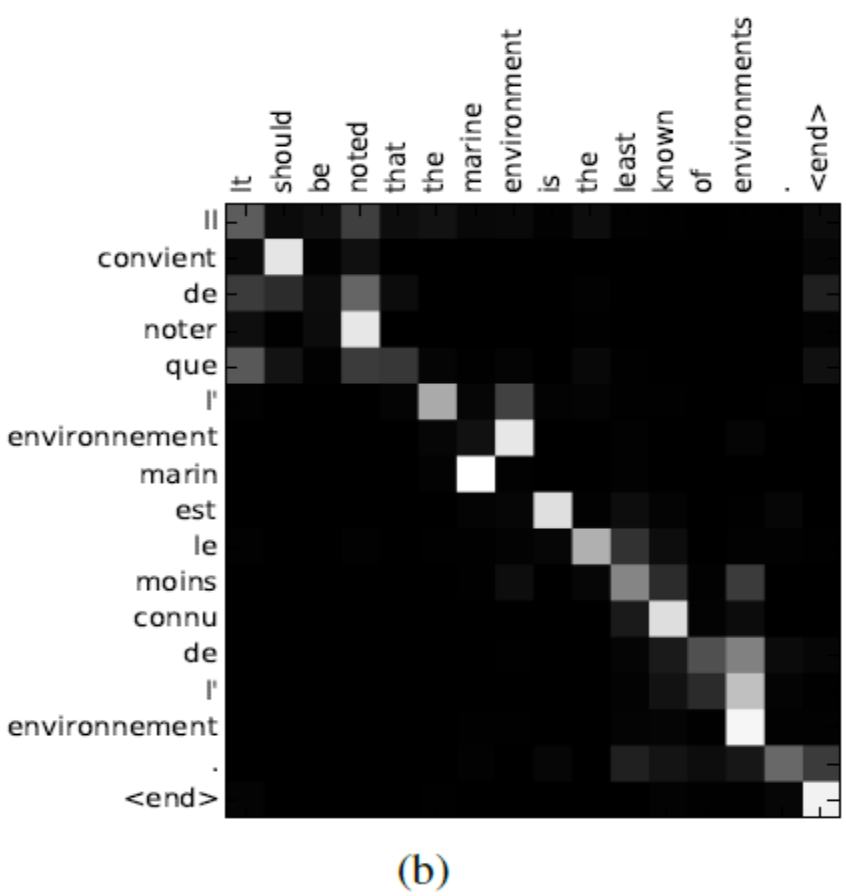


Bahdanau et al, “Neural Machine Translation by Jointly Learning to Align and Translate”, ICLR 2015

Soft Attention for Translation



(a)



(b)

Bahdanau et al, “Neural Machine Translation by Jointly Learning to Align and Translate”, ICLR 2015

- Video [“Attention Model Intuition”](#) by Andrew Ng (Coursera)

- Implementation example ("A ten-minute introduction to sequence-to-sequence learning in Keras"):

```
from keras.models import Model
from keras.layers import Input, LSTM, Dense

# Define an input sequence and process it.
encoder_inputs = Input(shape=(None, num_encoder_tokens))
encoder = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_inputs)
# We discard `encoder_outputs` and only keep the states.
encoder_states = [state_h, state_c]

# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None, num_decoder_tokens))
# We set up our decoder to return full output sequences,
# and to return internal states as well. We don't use the
# return states in the training model, but we will use them in inference.
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_inputs,
                                     initial_state=encoder_states)
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

# Define the model that will turn
# `encoder_input_data` & `decoder_input_data` into `decoder_target_data`
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

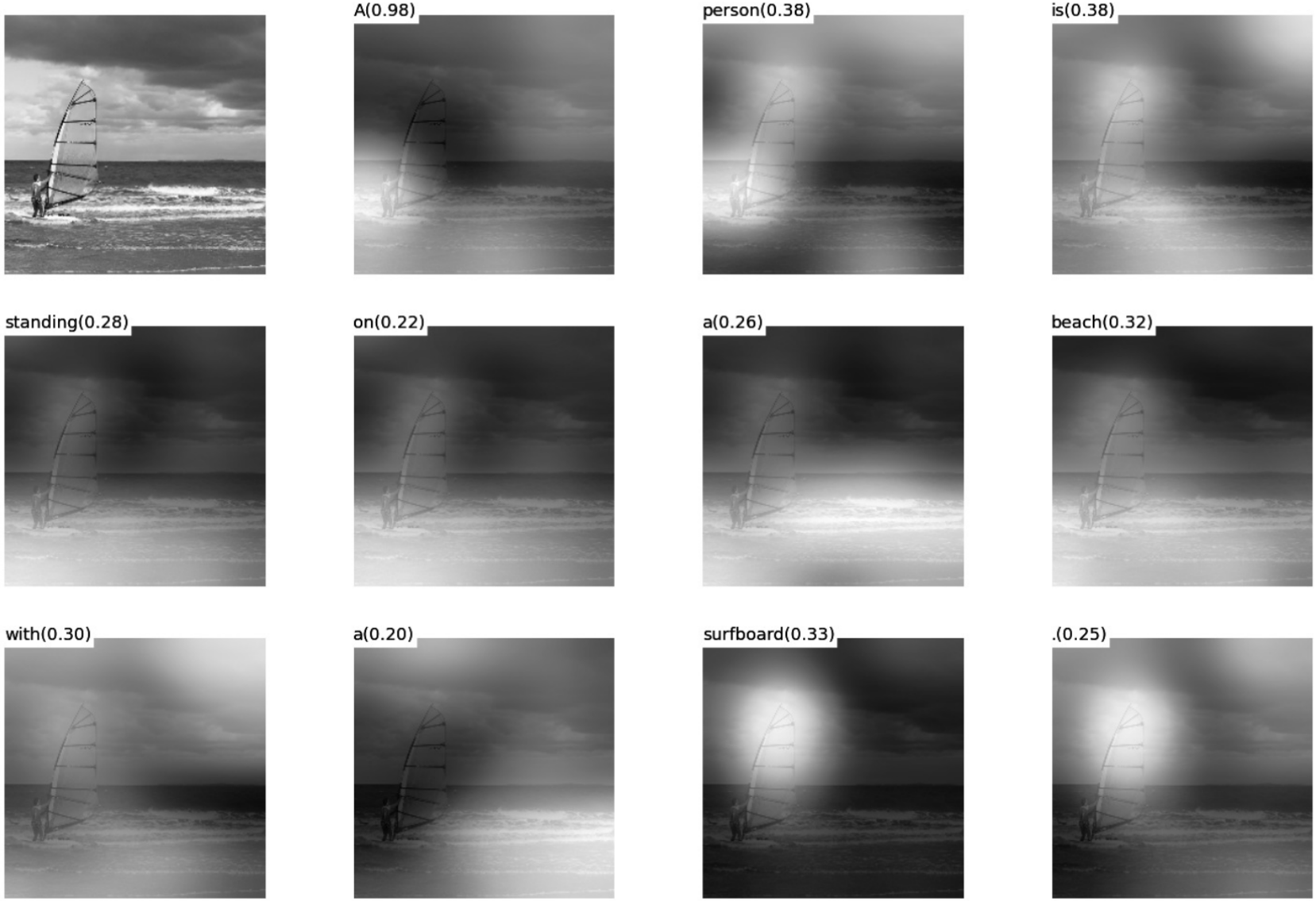
- The **cost** of attention: Attention comes at a cost. We need to calculate an attention value for each combination of input and output word. If you have a 50-word input sequence and generate a 50-word output sequence that would be 2500 attention values.
- That's quite counterintuitive. Human attention is something that's supposed to **save** computational resources.
- However, attention mechanisms have become quite popular and shown good performance on many tasks.
- An alternative approach to attention is to use *Reinforcement Learning* to predict an approximate location to focus to.

http://www.wildml.com/2016/01/attention-and-memory-in-deep-learning-and-nlp/

Other applications of Attention:

In "Show, Attend and Tell", the authors apply attention mechanisms to the problem of generating image descriptions. They use a Convolutional Neural Network to "encode" the image, and a Recurrent Neural Network with attention mechanisms to generate a description.

- Code example: Tensorflow tutorial "Image Captioning with Attention" (under 'Generative Models')



(b) A person is standing on a beach with a surfboard.

Image caption generation using attention (From CY Lee lecture)

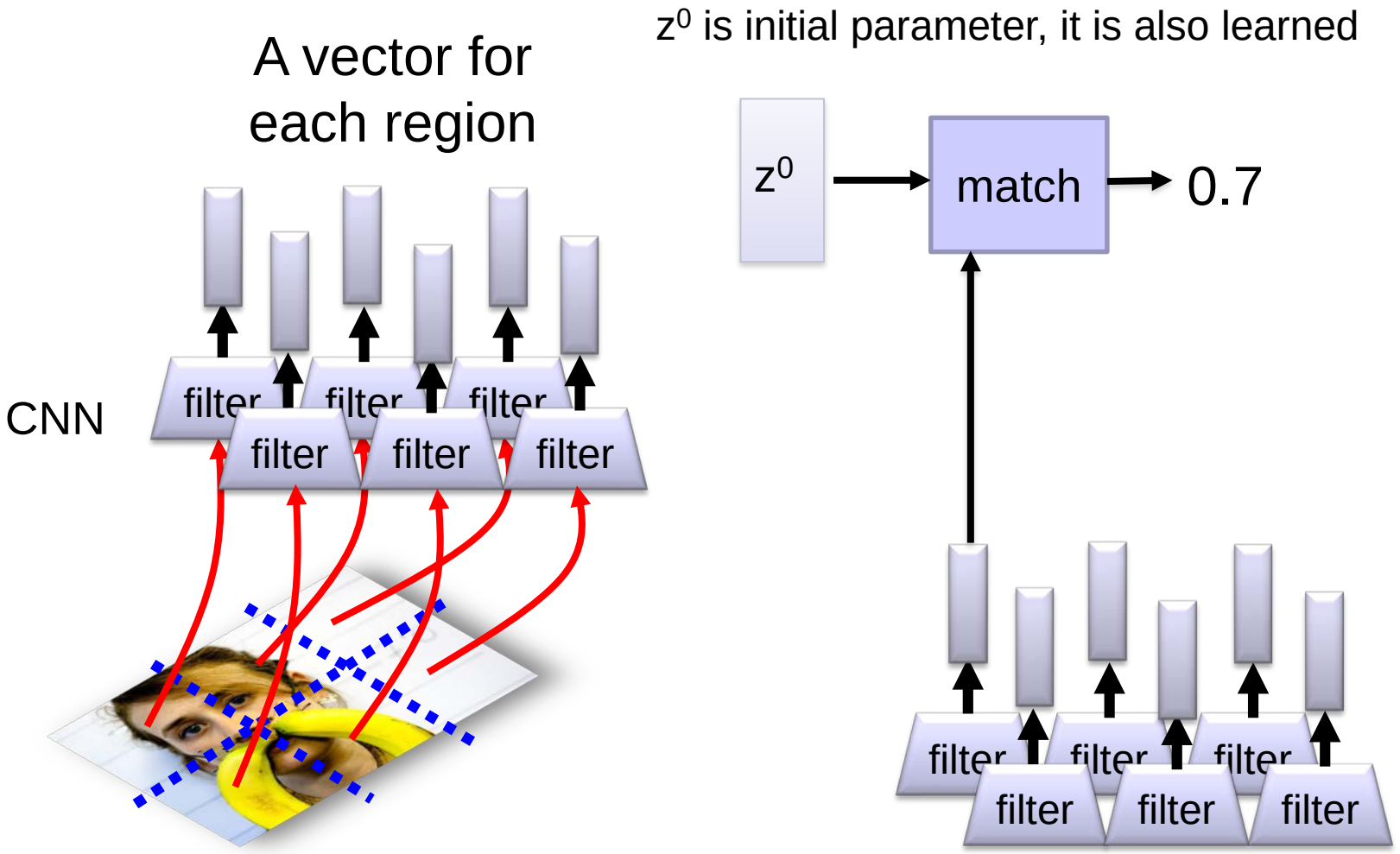


Image Caption Generation

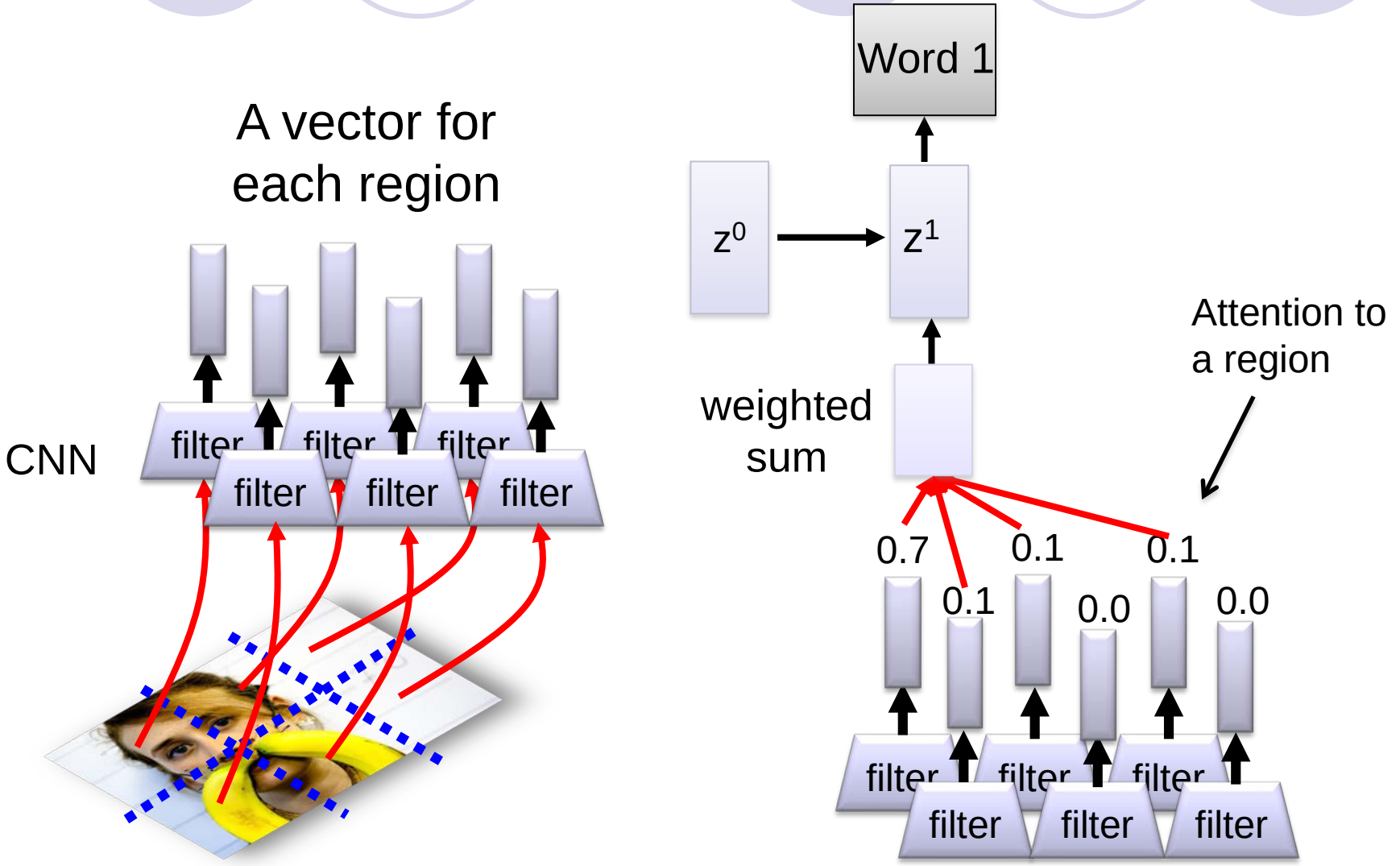


Image Caption Generation

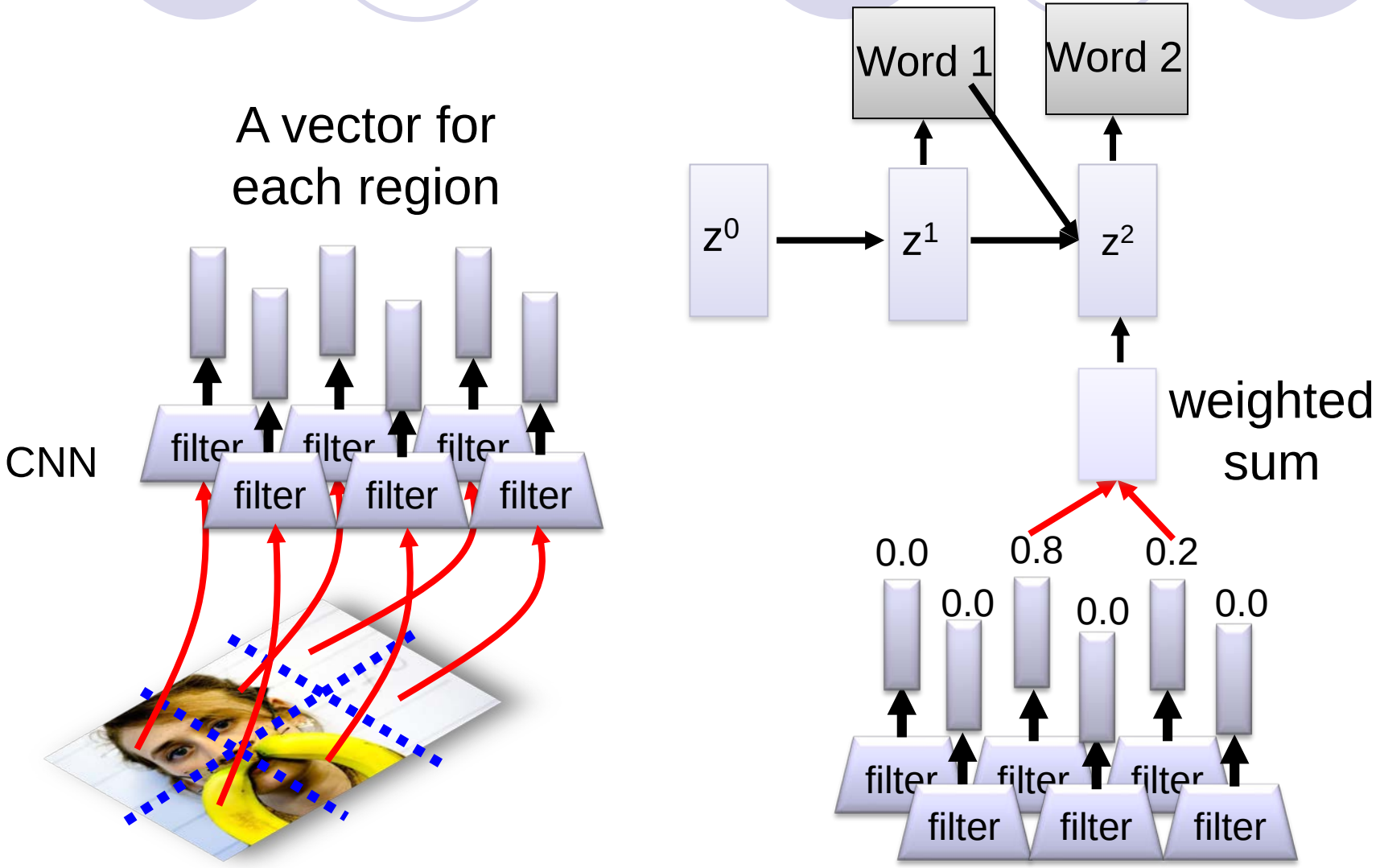


Image Caption Generation

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, Yoshua Bengio, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML, 2015

Image Caption Generation

A large white bird standing in a forest.

A woman holding a clock in her hand.

A man wearing a hat and a hat on a skateboard.

A person is standing on a beach with a surfboard.

A woman is sitting at a table with a large pizza.

A man is talking on his cell phone while another man watches.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, Yoshua Bengio, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML, 2015

[supplement] Machine Translation (not Neural)

- Machine Translation (MT) is one of the oldest applications of computers!! (from the 1950's)
- It is the task of automatically translate one natural language into another.

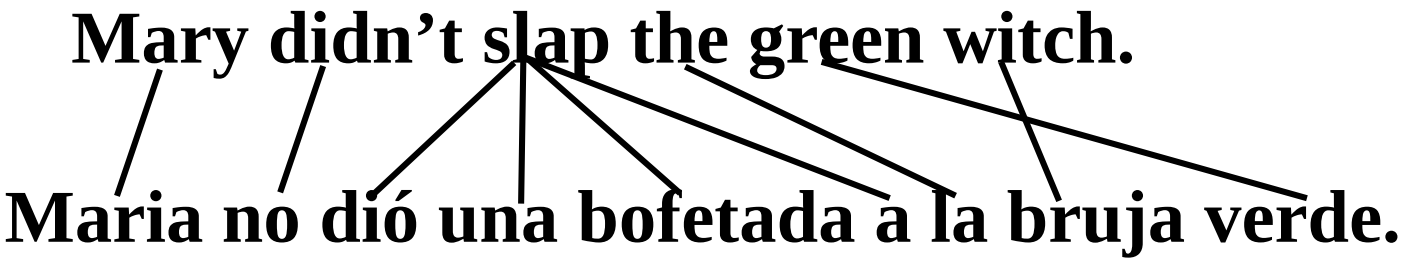
Mary didn't slap the green witch.

Maria no dió una bofetada a la bruja verde.

http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/

Word Alignment

- Shows mapping between words in one language and the other.

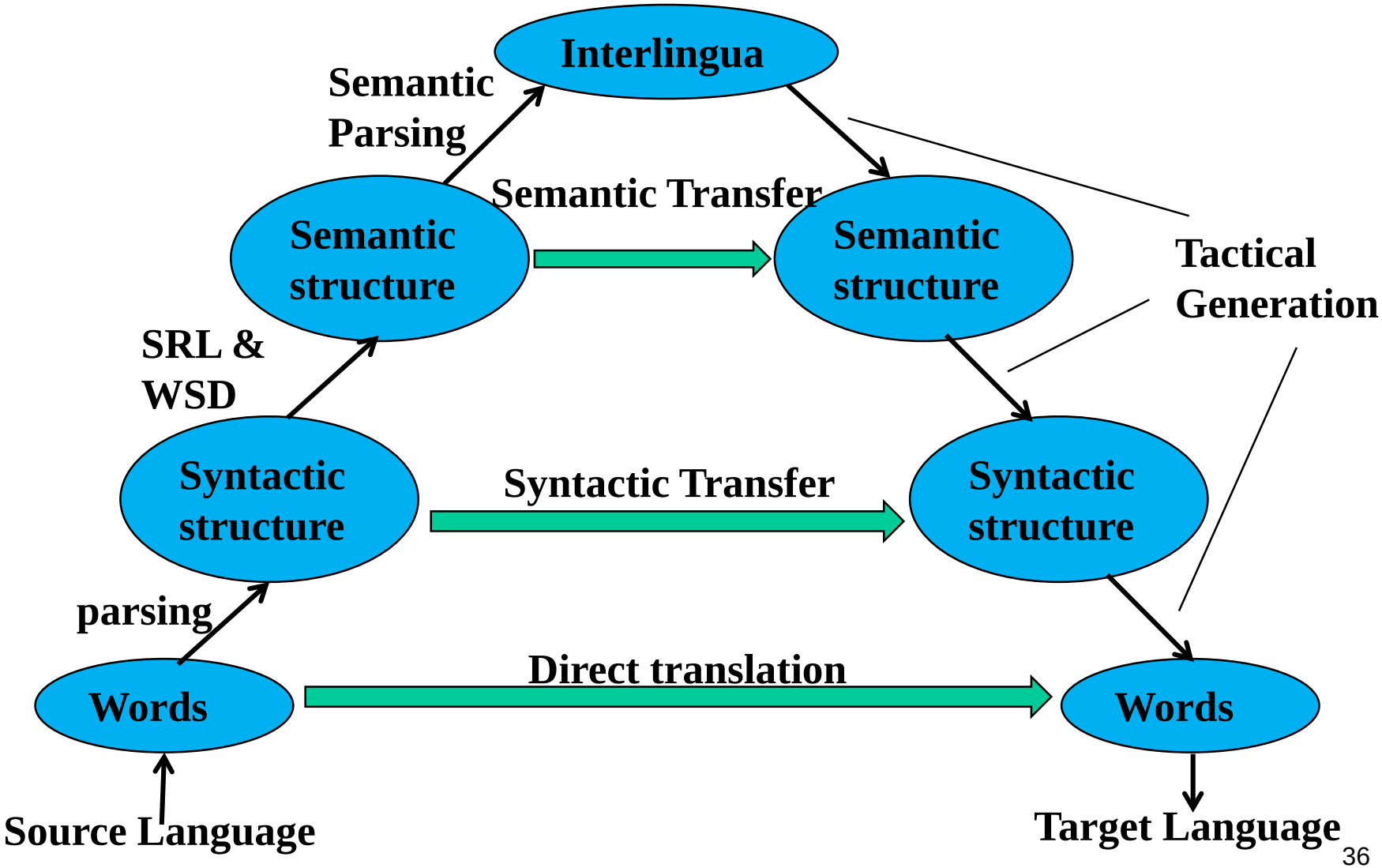


Linguistic Issues Making MT Difficult

- Morphological issues with ***agglutinative***, ***fusion*** and ***polysynthetic*** languages with complex word structure.
- Syntactic variation between ***SVO*** (e.g. English), ***SOV*** (e.g. Hindi), and ***VSO*** (e.g. Arabic) languages.
  - SVO languages use prepositions
  - SOV languages use postpositions
- ***Pro-drop*** languages regularly omit subjects that must be inferred.

•Ray Mooney, https://www.cs.utexas.edu/~mooney/cs388/

Vauquois Triangle



Direct Transfer

- Morphological Analysis
  - Mary didn't slap the green witch. →
    Mary DO:PAST not slap the green witch.
- Lexical Transfer
  - Mary DO:PAST not slap the green witch.
  - Maria no dar:PAST una bofetada a la verde bruja.
- Lexical Reordering
  - Maria no dar:PAST una bofetada a la bruja verde.
- Morphological generation
  - Maria no dió una bofetada a la bruja verde.

Statistical MT

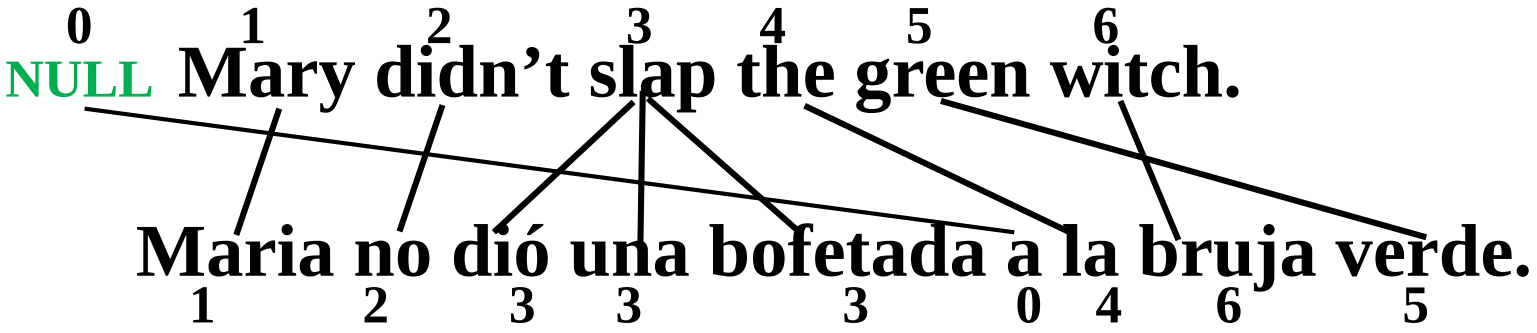
- Manually encoding comprehensive bilingual lexicons and transfer rules is difficult.
- SMT acquires knowledge needed for translation from a ***parallel corpus*** or ***bitext*** that contains the same set of documents in two languages.
- The Canadian Hansards (parliamentary proceedings in French and English) is a well-known parallel corpus.
- First align the sentences in the corpus based on simple methods that use coarse cues like sentence length to give bilingual sentence pairs.

Word Alignment

- Directly constructing phrase alignments is difficult, so rely on first constructing word alignments.
- Can learn to align from supervised word alignments, but human-aligned bitexts are rare and expensive to construct.
- Typically use an unsupervised EM-based approach to compute a word alignment from unannotated parallel corpus.

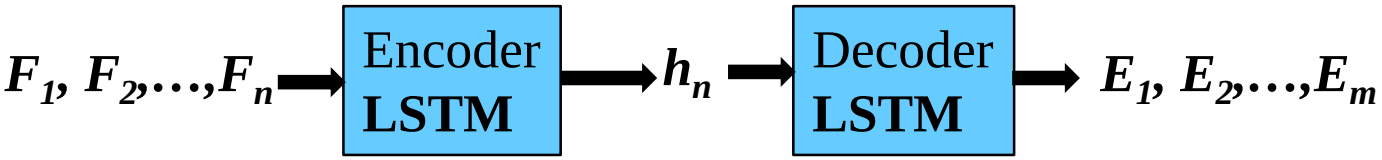
One to Many Alignment

- To simplify the problem, typically assume each word in *F* aligns to 1 word in *E* (but assume each word in *E* may generate more than one word in *F*).
- Some words in *F* may be generated by the NULL element of *E*.
- Therefore, alignment can be specified by a vector *A* giving, for each word in *F*, the index of the word in *E* which generated it.



Neural Machine Translation (NMT)

- Encoder/Decoder framework maps sentence in source language to a "deep vector" then another LSTM maps this vector to a sentence in the target language



- Train model "end to end" on sentence-aligned parallel corpus.