Natural Language Processing

Sequence to Sequence Learning

Instructor: Yangqiu Song

Slides credits: Haitham Elmarakeby, Richard Socher

Sequence to Sequence

- Speech recognition

http://nlp.stanford.edu/courses/lsa352/

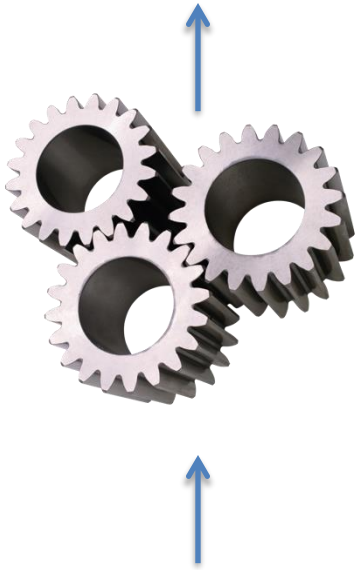
Sequence to Sequence

- Question answering

Sequence to Sequence

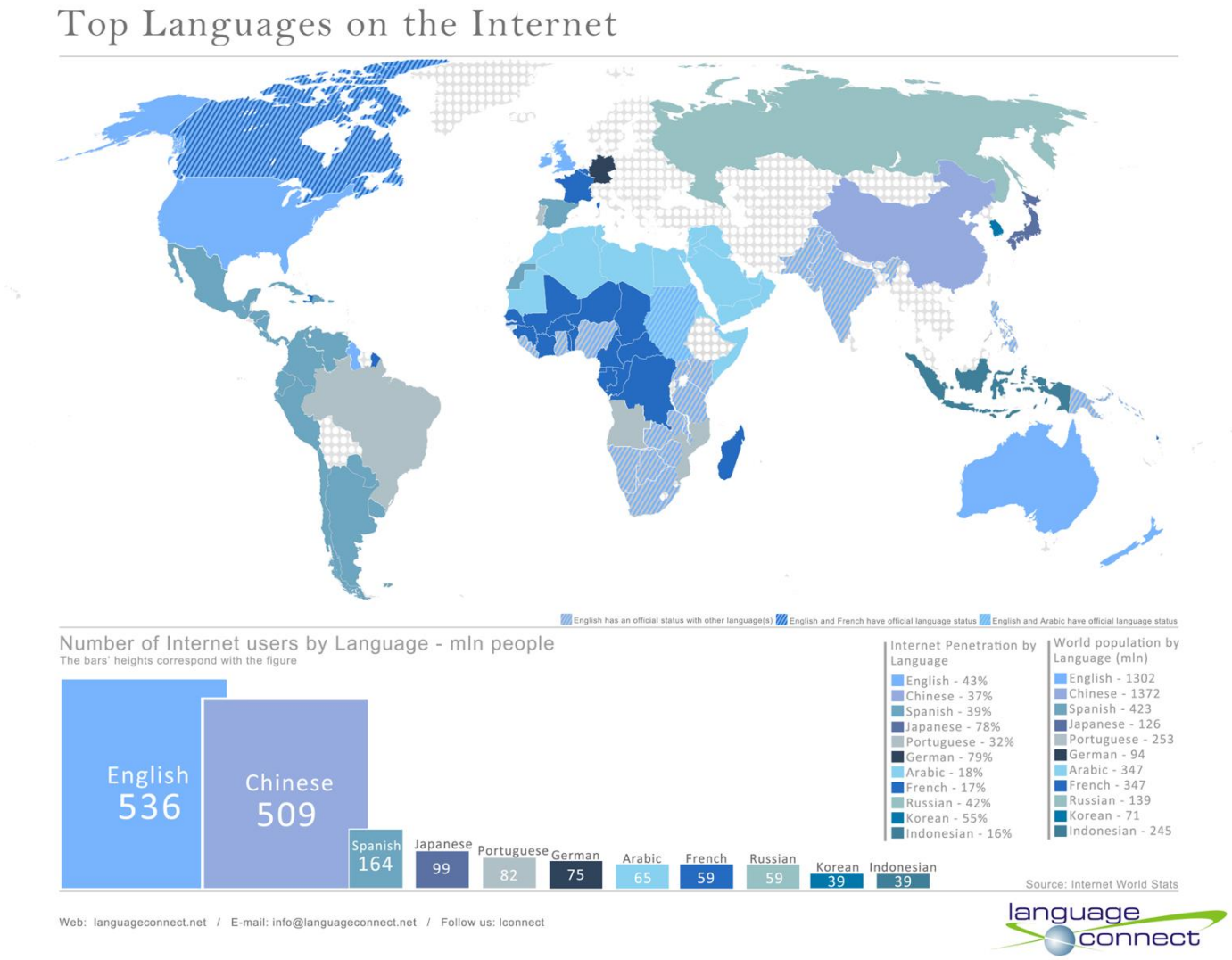
- Machine translation

مرحبا بكم في درس التعلم العميق



Welcome to the deep learning class

7 billion people, 7000 languages



Machine Translation

- **Machine Translation (MT)** is the task of translating a sentence *x* from one language (the source language) to a sentence *y* in another language (the target language).

- **1950s: Early Machine Translation**
  - Mostly Russian → English (motivated by the Cold War!)
  - Systems were mostly rule-based, using a bilingual dictionary to map Russian words to their English counterparts

Source: https://youtu.be/K-HfpsHPmvw

1990s-2010s: Statistical Machine Translation

- Core idea: Learn a probabilistic model from data
- Suppose we're translating Language X → English.
- We want to find best English sentence *y*, given Language X sentence *x*

$$\text{argmax}_y P(y|x)$$

- Use Bayes Rule to break this down into two components to be learnt separately:

$$= \text{argmax}_y P(x|y)P(y)$$

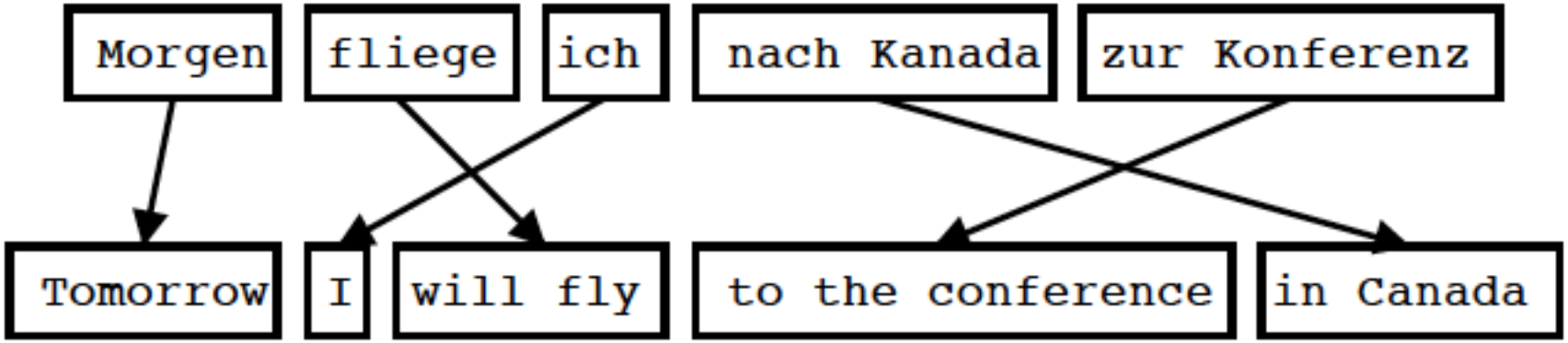
Translation Model
Models how words and phrases should be translated.
Learnt from parallel data.

Language Model
Models how to write good English.
Learnt from monolingual data.

Statistical Machine Translation

- Translation model

- Input is Segmented in Phrases
- Each Phrase is Translated into English
- Phrases are Reordered



Koehn 2004

Statistical Machine Translation

- Language Model

Goal of the Language Model: Detect good English

For Example: Trigram Model

```
Mary did not slap the green witch
Mary  =>  p(Mary)
Mary did  =>  p(did|Mary)
Mary did not  =>  p(not|Mary did)
     did not slap  =>  p(slap|did not)
         not slap the  =>  p(the|not slap)
             slap the green  =>  p(green|slap the)
                 the green witch  =>  p(witch|the green)
```

Knight and Koehn 2003

1990s-2010s: Statistical Machine Translation

$$\text{argmax}_y P(x|y)P(y)$$

Question: How to compute this argmax?

Translation Model

Language Model

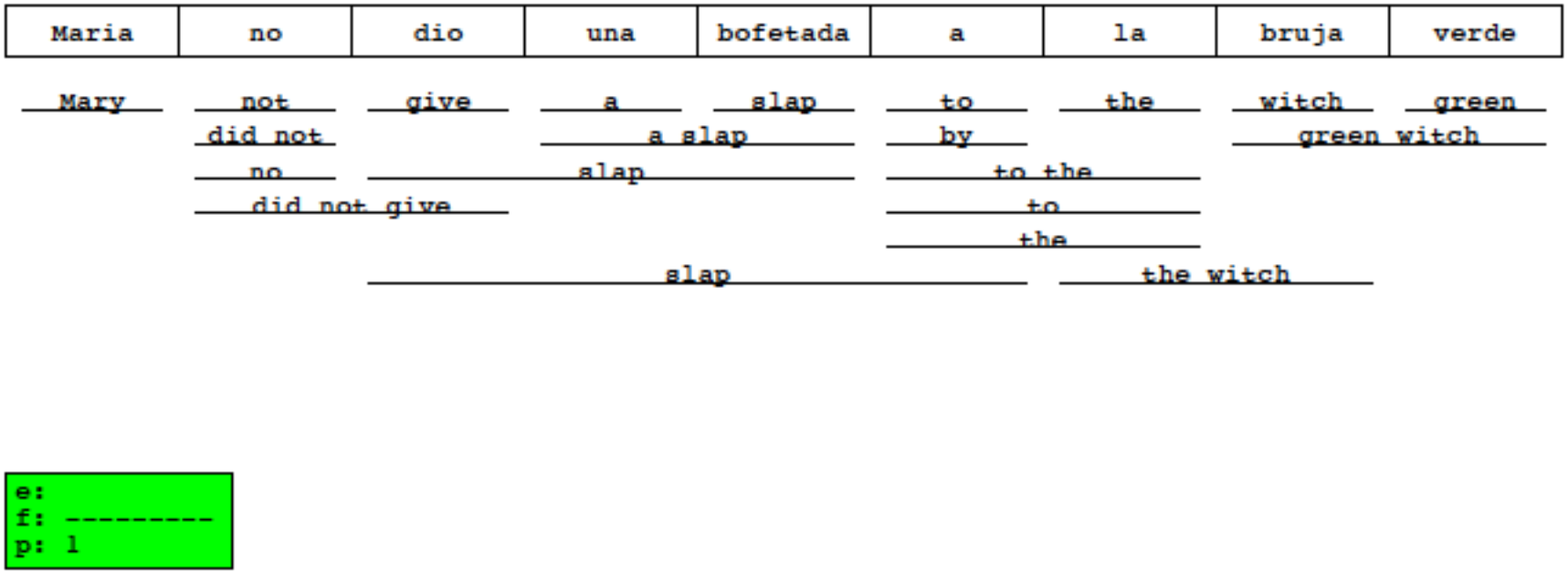
- We could enumerate every possible *y* and calculate the probability? → Too expensive!

- **Answer:** Use a heuristic search algorithm to gradually build up the translation, discarding hypotheses that are too low probability

Statistical Machine Translation

- Decoding

Goal of the decoding algorithm: Put models to work, perform the actual translation

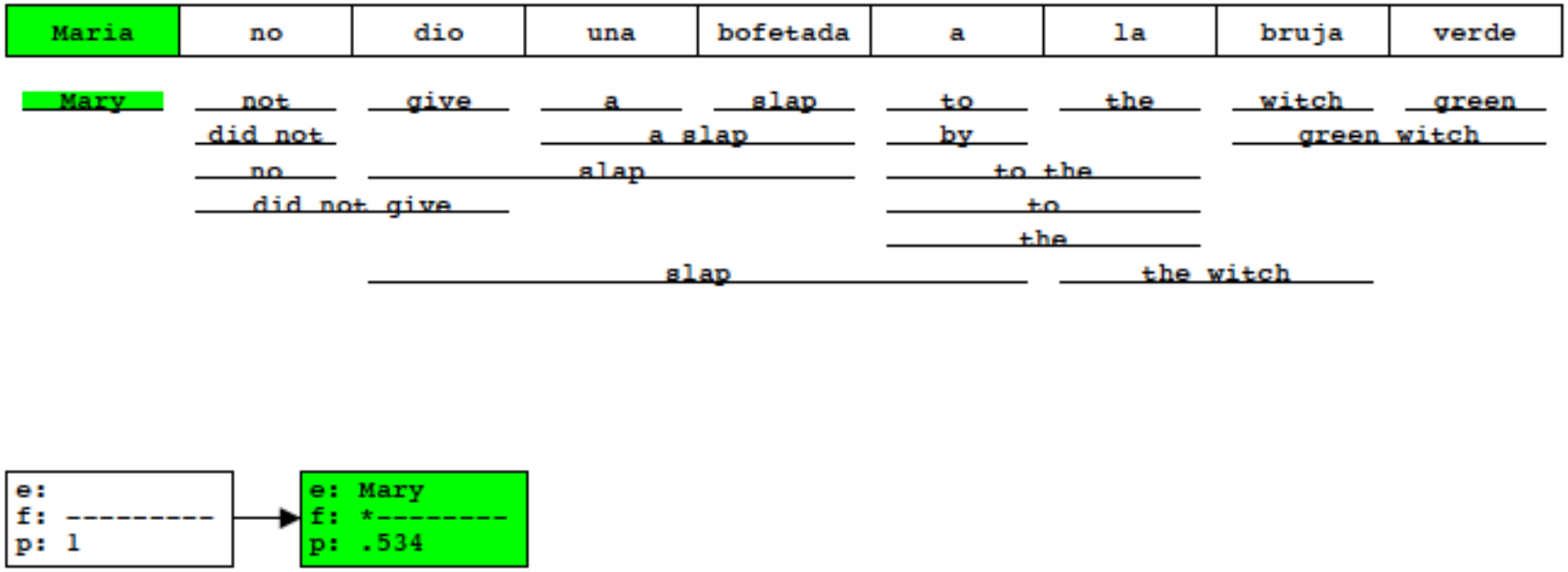


Koehn 2004

Statistical Machine Translation

- Decoding

Goal of the decoding algorithm: Put models to work, perform the actual translation

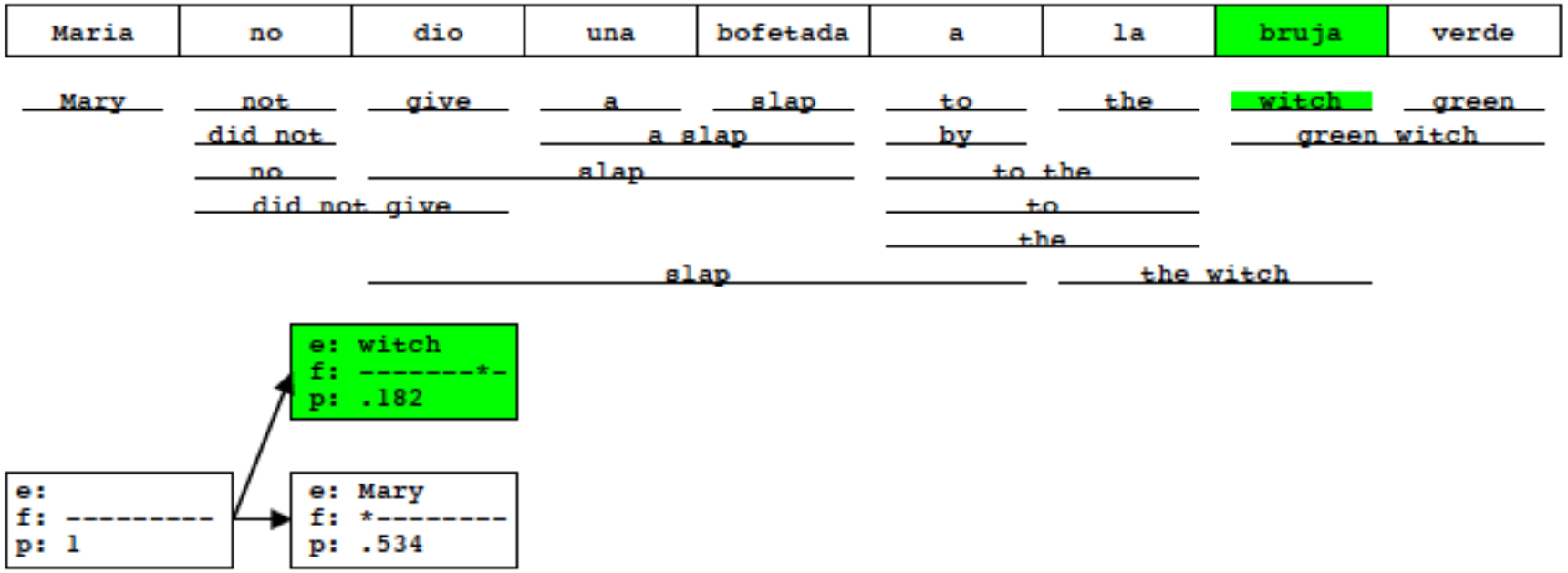


Koehn 2004

Statistical Machine Translation

- Decoding

Goal of the decoding algorithm: Put models to work, perform the actual translation

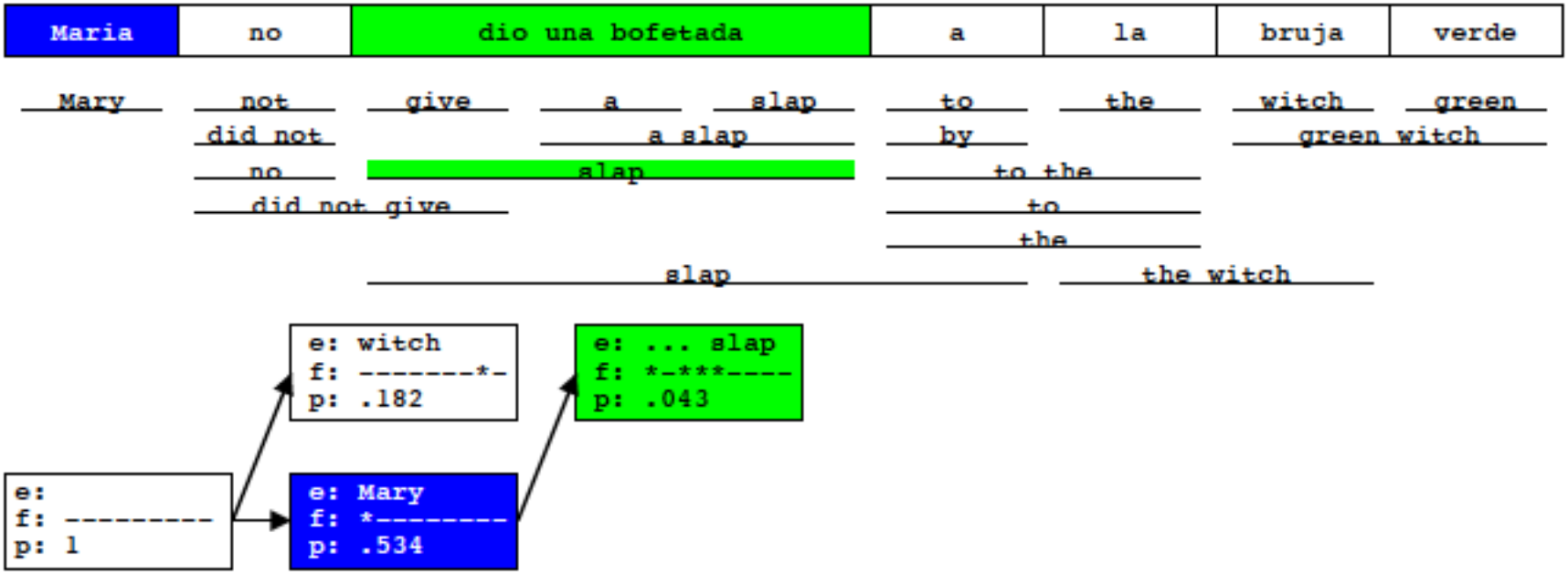


Koehn 2004

Statistical Machine Translation

- Decoding

Goal of the decoding algorithm: Put models to work, perform the actual translation

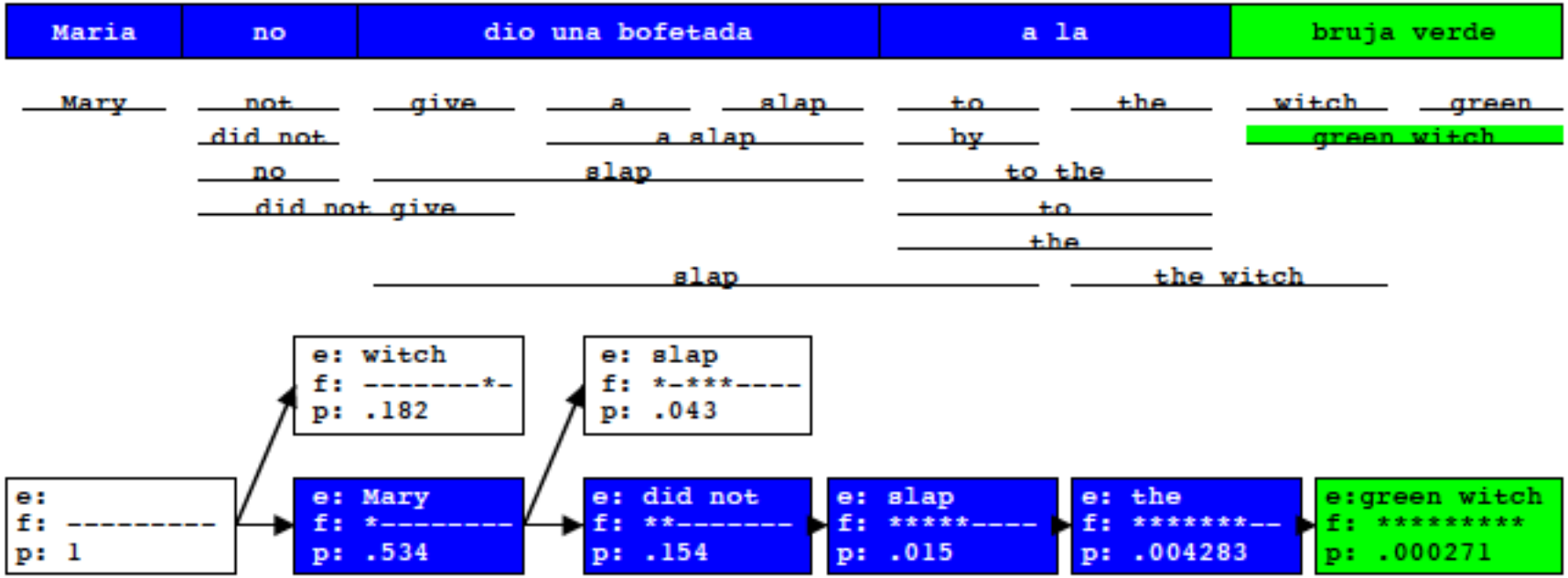


Koehn 2004

Statistical Machine Translation

- Decoding

Goal of the decoding algorithm: Put models to work, perform the actual translation

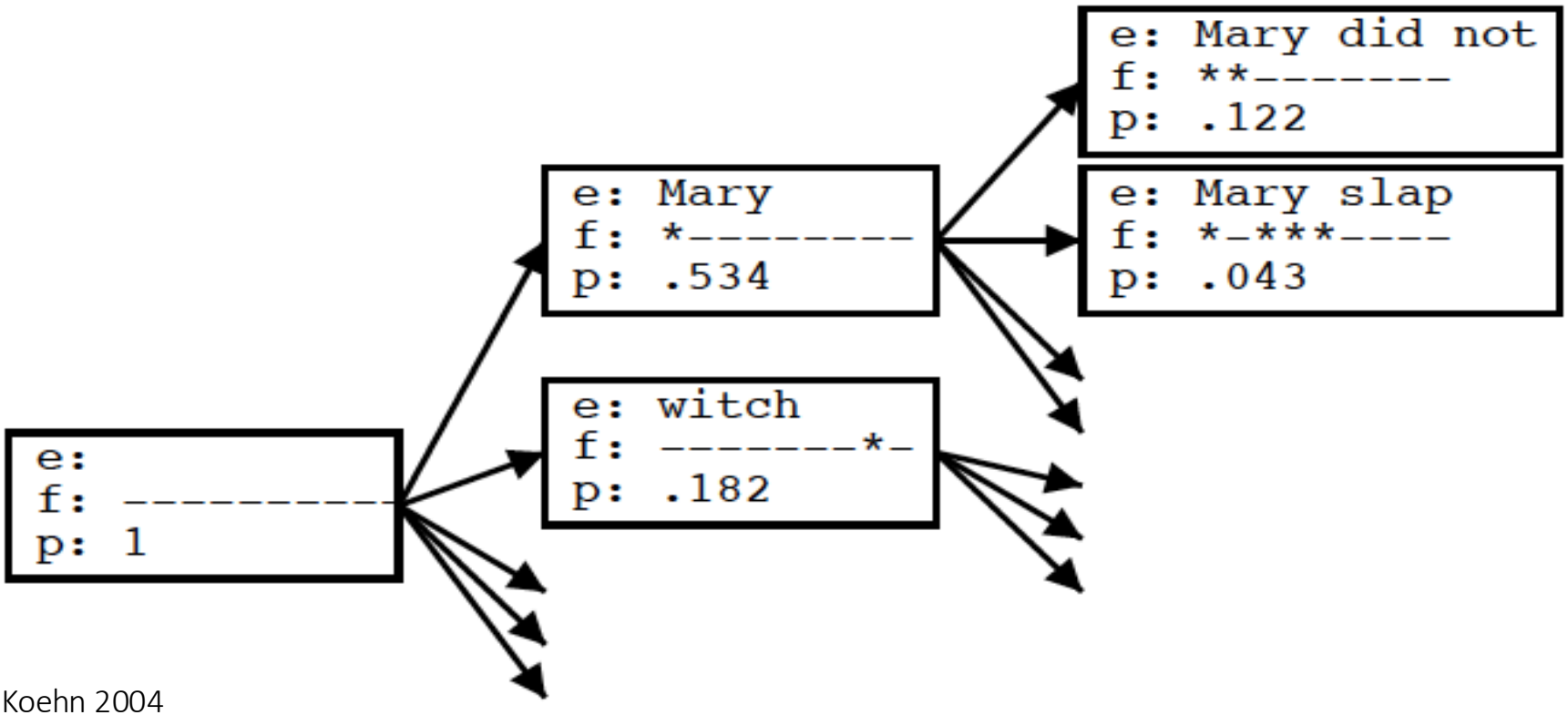


Koehn 2004

Statistical Machine Translation

- Decoding

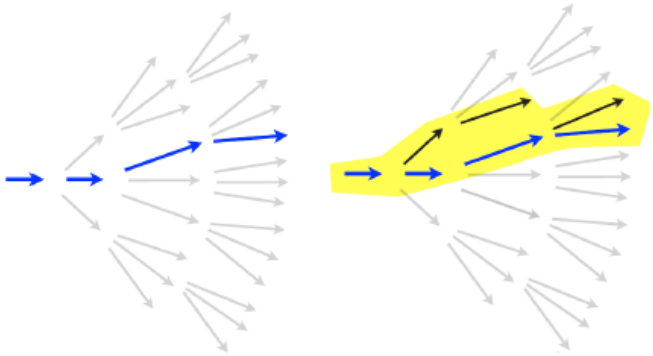
Goal of the decoding algorithm: Put models to work, perform the actual translation



Koehn 2004

Decoding

- Global solution: Dynamic programming
- Approximate solutions: Beam inference
  - At each position keep the top k complete sequences
  - Extend each sequence in each local way
  - The extensions compete for the k slots at the next position



(a) Greedy (b) Beam Search

- Advantages
  - Fast; beam sizes of 3-5 are almost as good as exact inference in many cases
  - Easy to implement (no dynamic programming required)
- Disadvantage
  - Inexact: the globally best sequence can fall off the beam

1990s-2010s: Statistical Machine Translation

- SMT is a huge research field
- The best systems are extremely complex
  - Hundreds of important details we haven't mentioned here
  - Systems have many separately-designed subcomponents
  - Lots of feature engineering
    - Need to design features to capture particular language phenomena
  - Require compiling and maintaining extra resources
    - Like tables of equivalent phrases
  - Lots of human effort to maintain
    - Repeated effort for each language pair!

Neural Machine Translation

- Neural Machine Translation (NMT) is a way to do Machine Translation with a *single neural network*

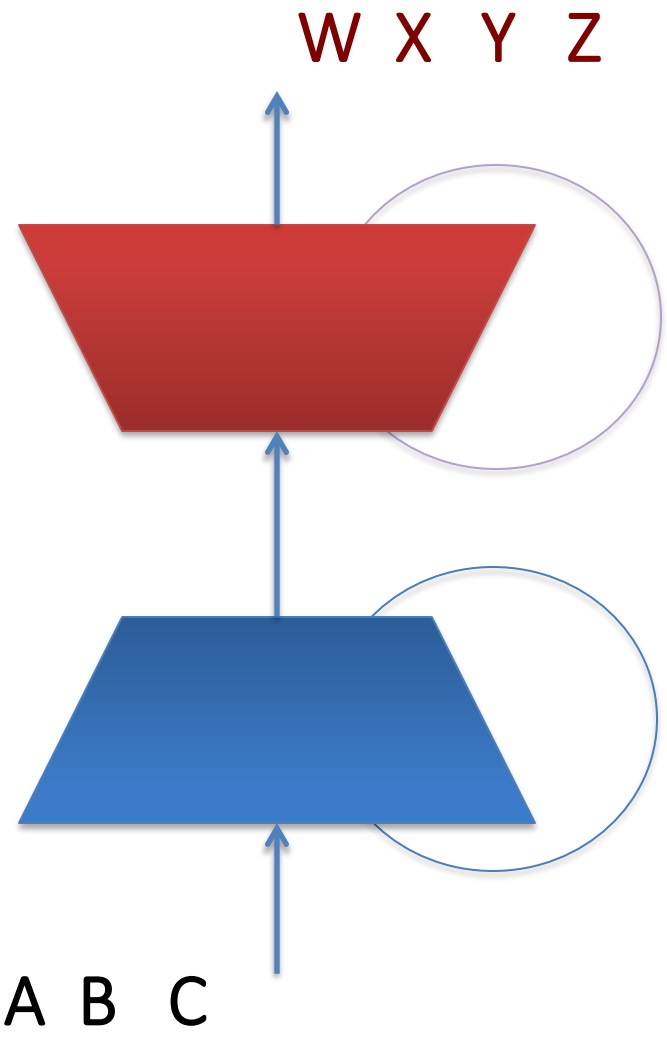
- The neural network architecture is called sequence-to-sequence (aka seq2seq) and it involves *two* RNNs.

Sequence to Sequence Learning with Neural Networks

- Ilya Sutskever, Oriol Vinyals, Quoc V. Le

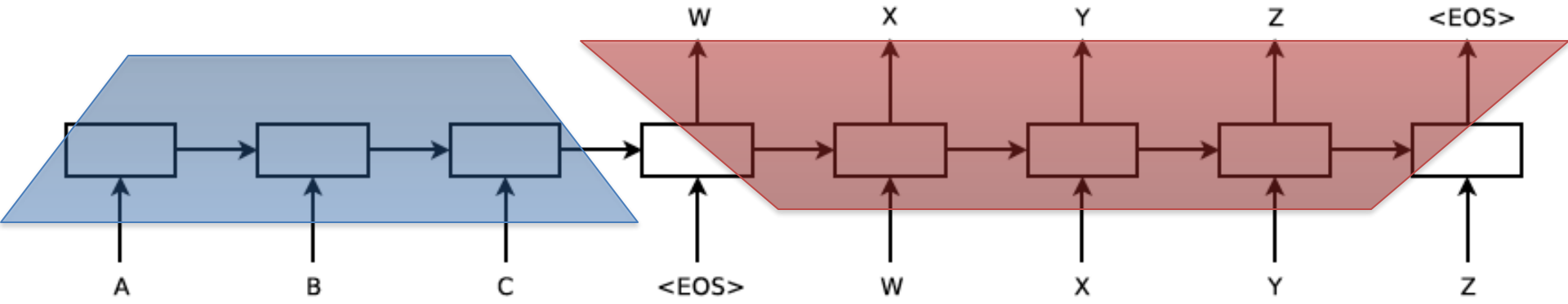
Neural Machine Translation

- Model



Neural Machine Translation

- Model



Sutskever et al. 2014

Neural Machine Translation (NMT)

- The sequence-to-sequence model is an example of a **Conditional Language Model**.
  - **Language Model** because the decoder is predicting the next word of the target sentence *y*
  - **Conditional** because its predictions are *also* conditioned on the source sentence *x*

- NMT directly calculates $P(y|x)$

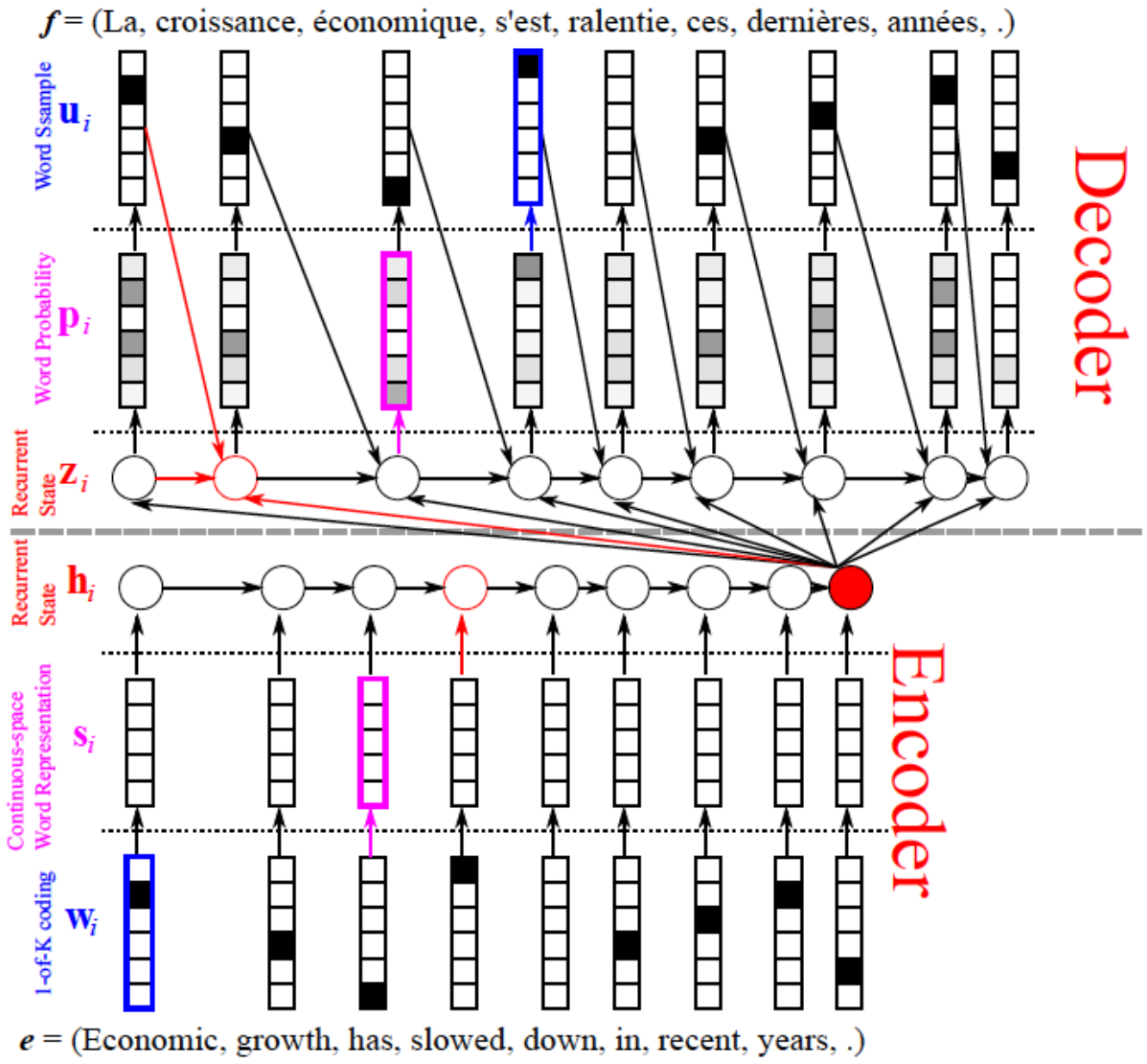
$$P(y|x) = P(y_1|x)\, P(y_2|y_1, x)\, P(y_3|y_1, y_2, x) \ldots, P(y_T|y_1, \ldots, y_{T-1}, x)$$

Probability of next target word, given target words so far and source sentence *x*

- **Question**: How to train a NMT system?
- **Answer**: Get a big parallel corpus…

Neural Machine Translation

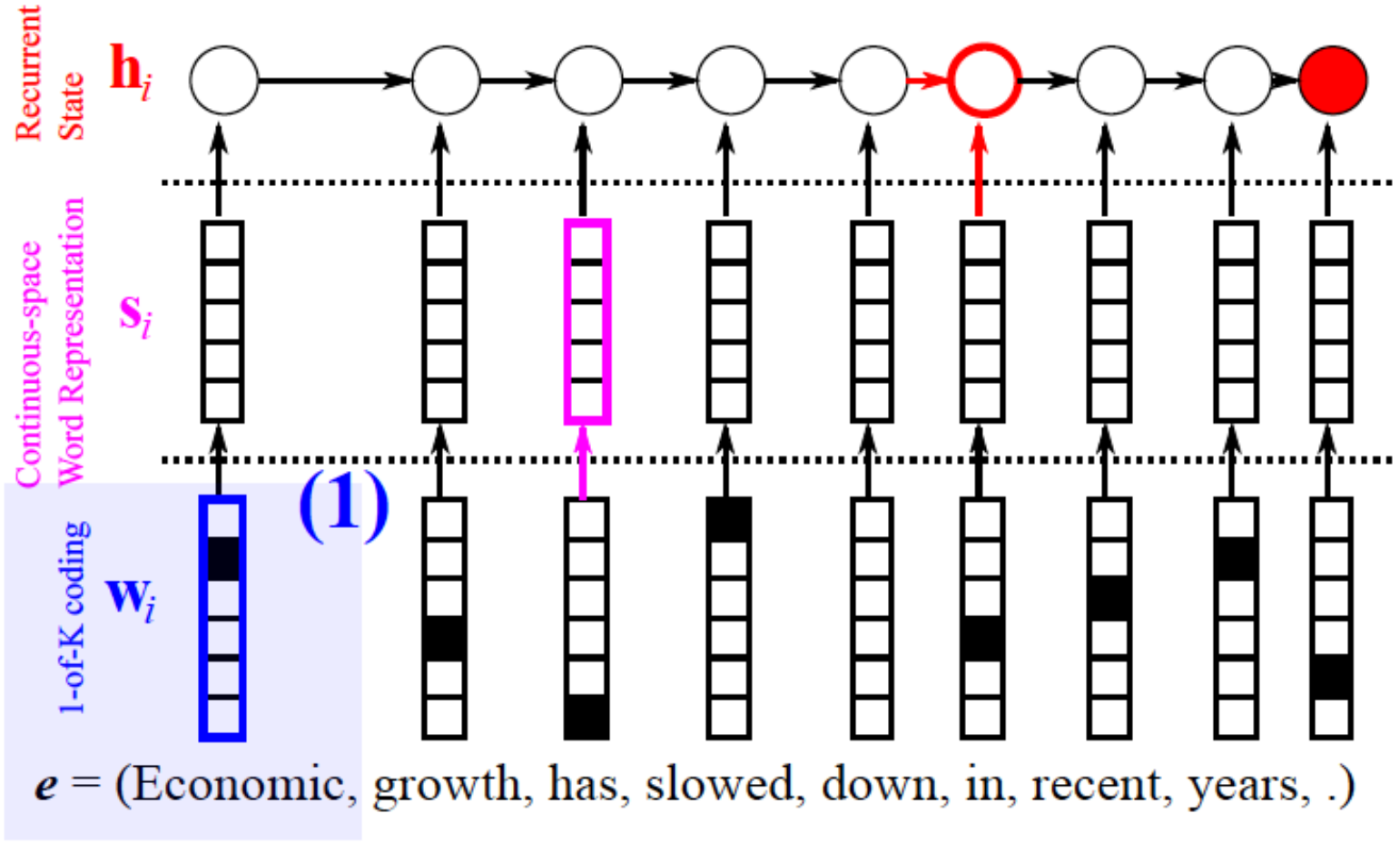
- Model



Cho: From Sequence Modeling to Translation

Neural Machine Translation

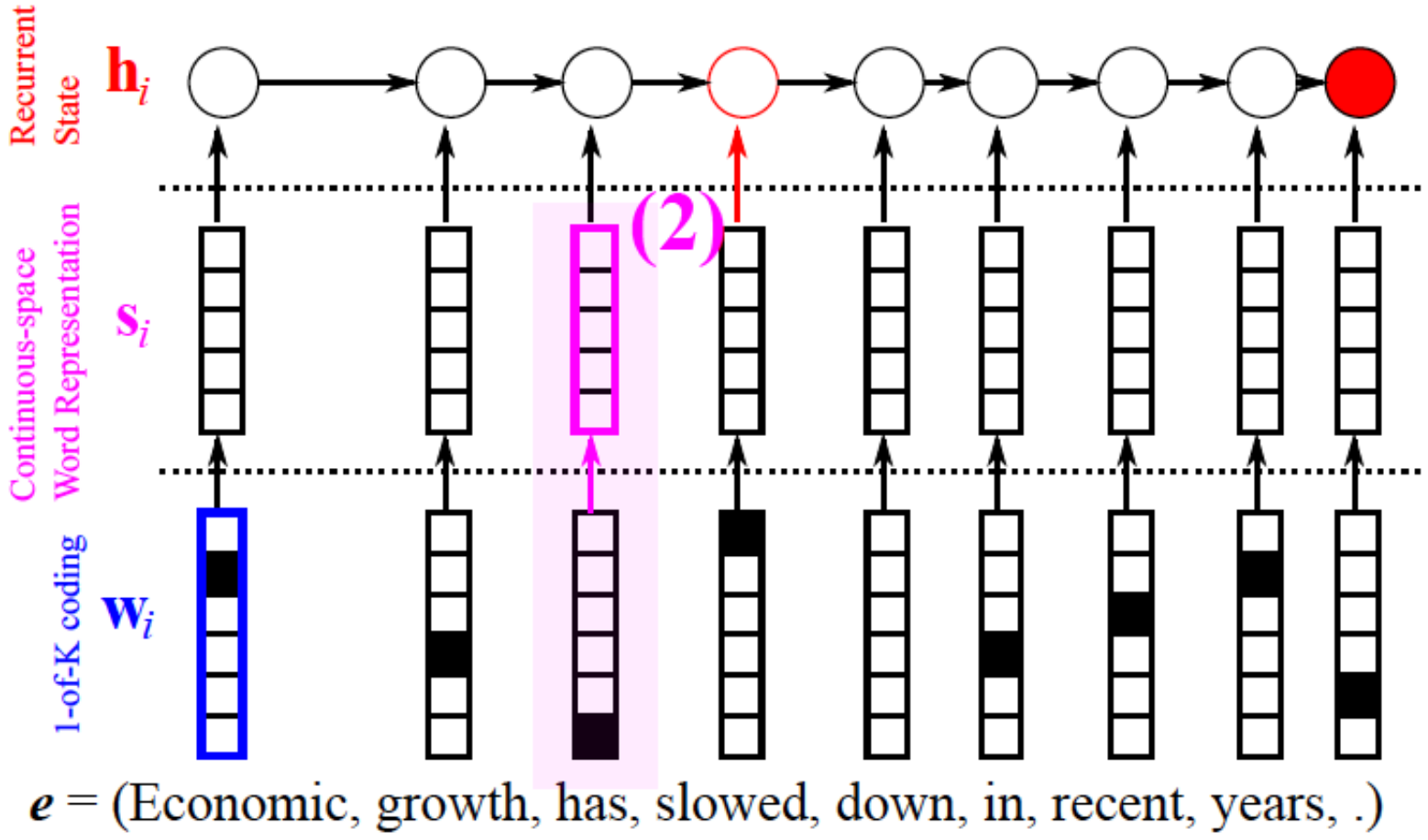
- Model- *encoder*



Cho: From Sequence Modeling to Translation

Neural Machine Translation

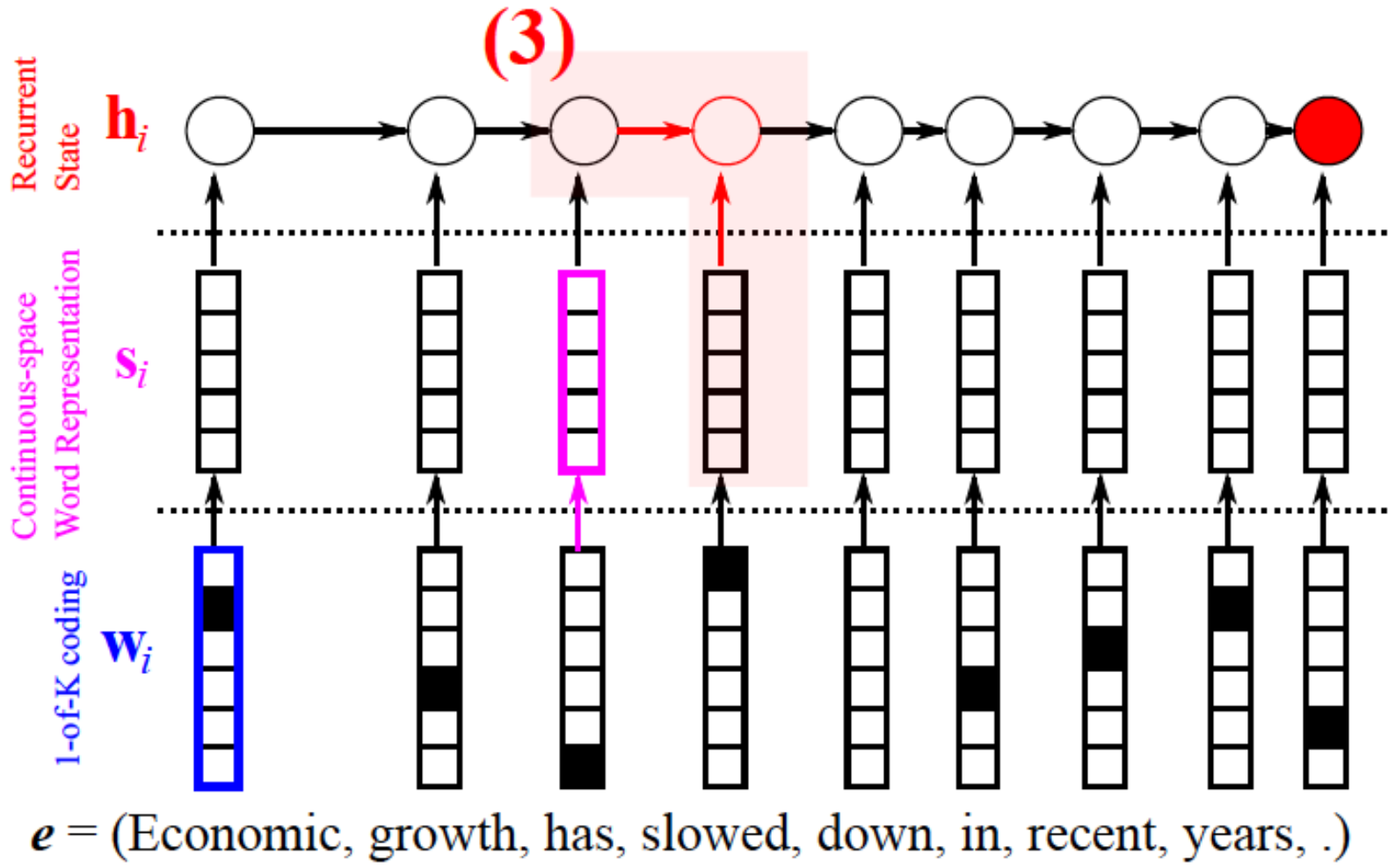
- Model- *encoder*



Cho: From Sequence Modeling to Translation

Neural Machine Translation

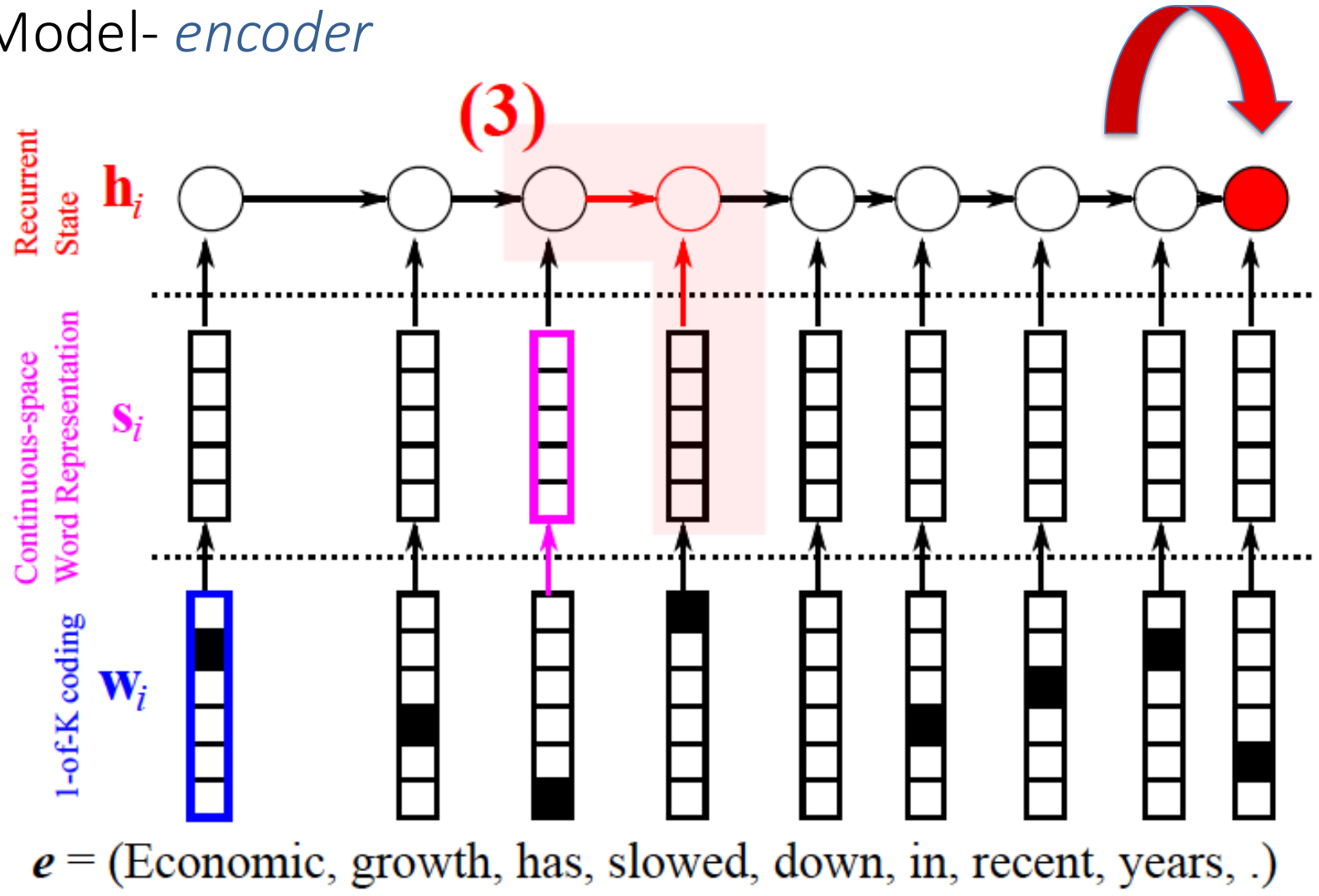
- Model- *encoder*



Cho: From Sequence Modeling to Translation

Neural Machine Translation

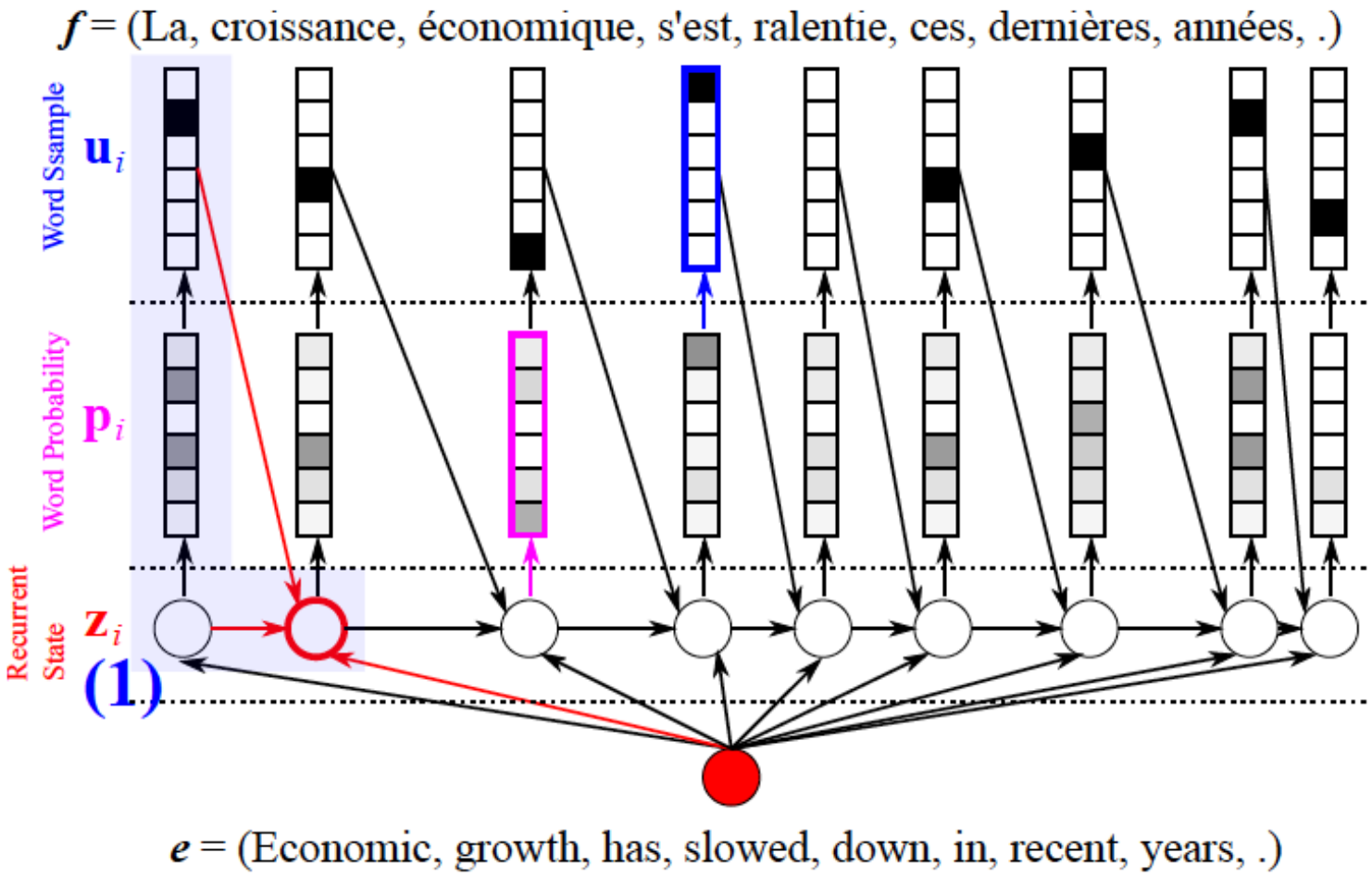
- Model- *encoder*



Cho: From Sequence Modeling to Translation

Neural Machine Translation

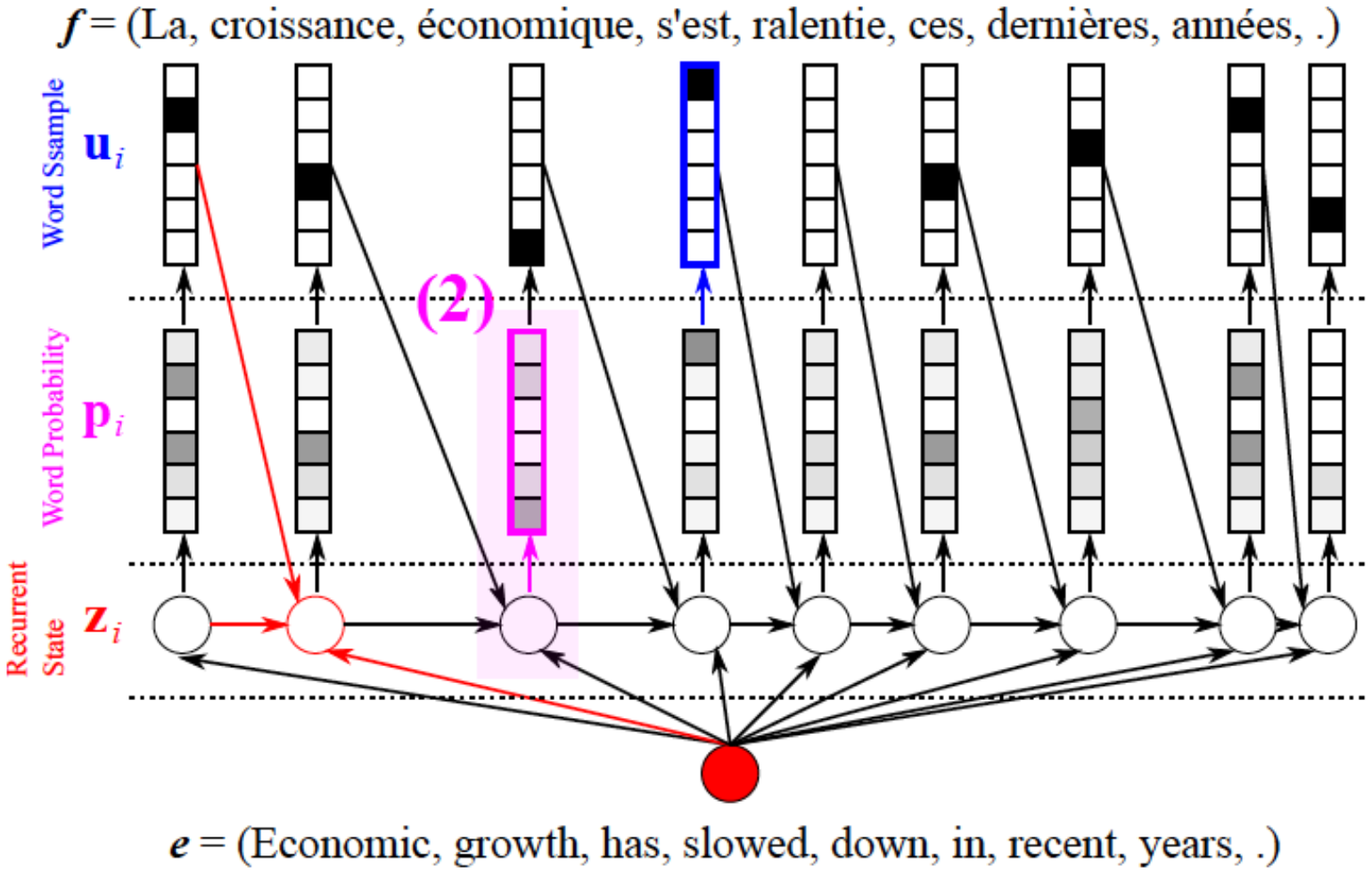
- Model- *decoder*



Cho: From Sequence Modeling to Translation

Neural Machine Translation

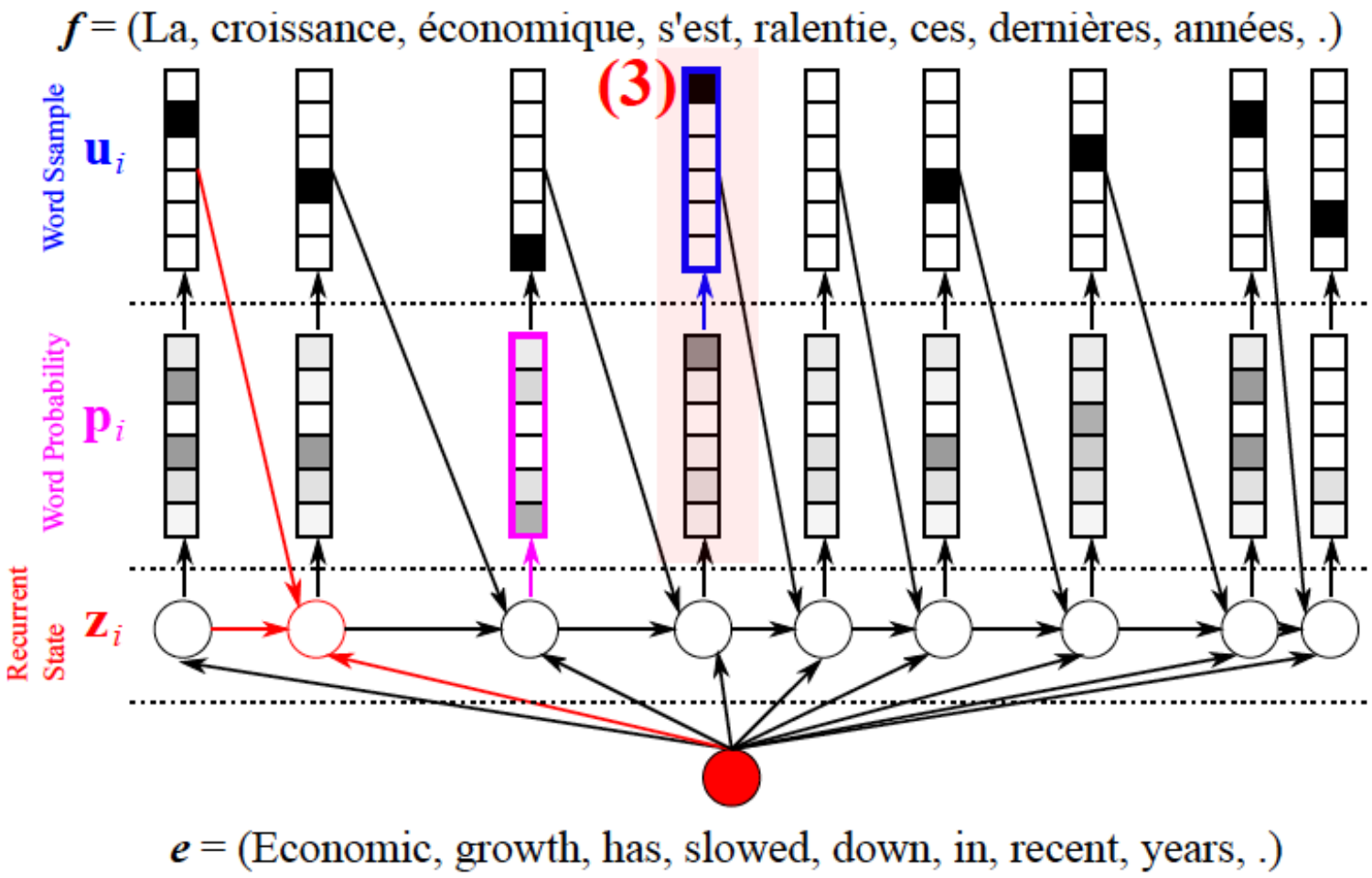
- Model- *decoder*



Cho: From Sequence Modeling to Translation

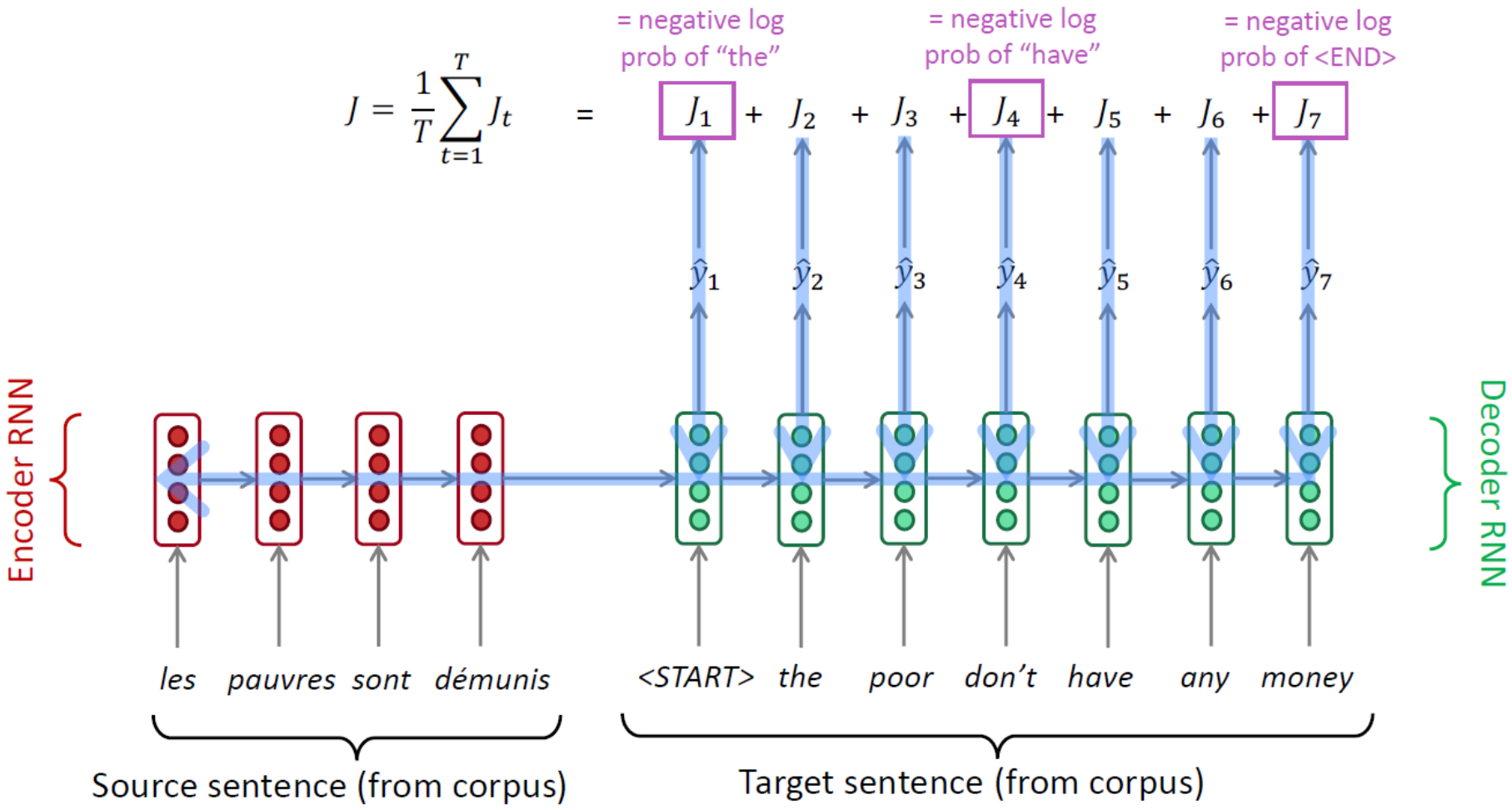
Neural Machine Translation

- Model- *decoder*



Cho: From Sequence Modeling to Translation

Training a Neural Machine Translation system



Seq2seq is optimized as a **single system.**
Backpropagation operates "*end to end*".

Advantages of NMT

Compared to SMT, NMT has many advantages:

- Better performance
  - More fluent
  - Better use of context
  - Better use of phrase similarities
- A single neural network to be optimized end-to-end
  - No subcomponents to be individually optimized
- Requires much less human engineering effort
  - No feature engineering
  - Same method for all language pairs

Disadvantages of NMT?

Compared to SMT:

- NMT is less interpretable
  - Hard to debug
- NMT is difficult to control
  - For example, can't easily specify rules or guidelines for translation
  - Safety concerns!

Evaluation (Machine Translation)

- BLEU (bilingual evaluation understudy) (Papineni et al. (2002))
  - BLEU compares the machine-written translation to one or several human-written translation(s), and computes a similarity score based on:
    - *n*-gram precision (usually up to 3 or 4-grams)
    - Penalty for too-short system translations
  - BLEUs output is always a number between 0 and 1
    - 1 means identical to the reference translations

BLEU is useful but imperfect

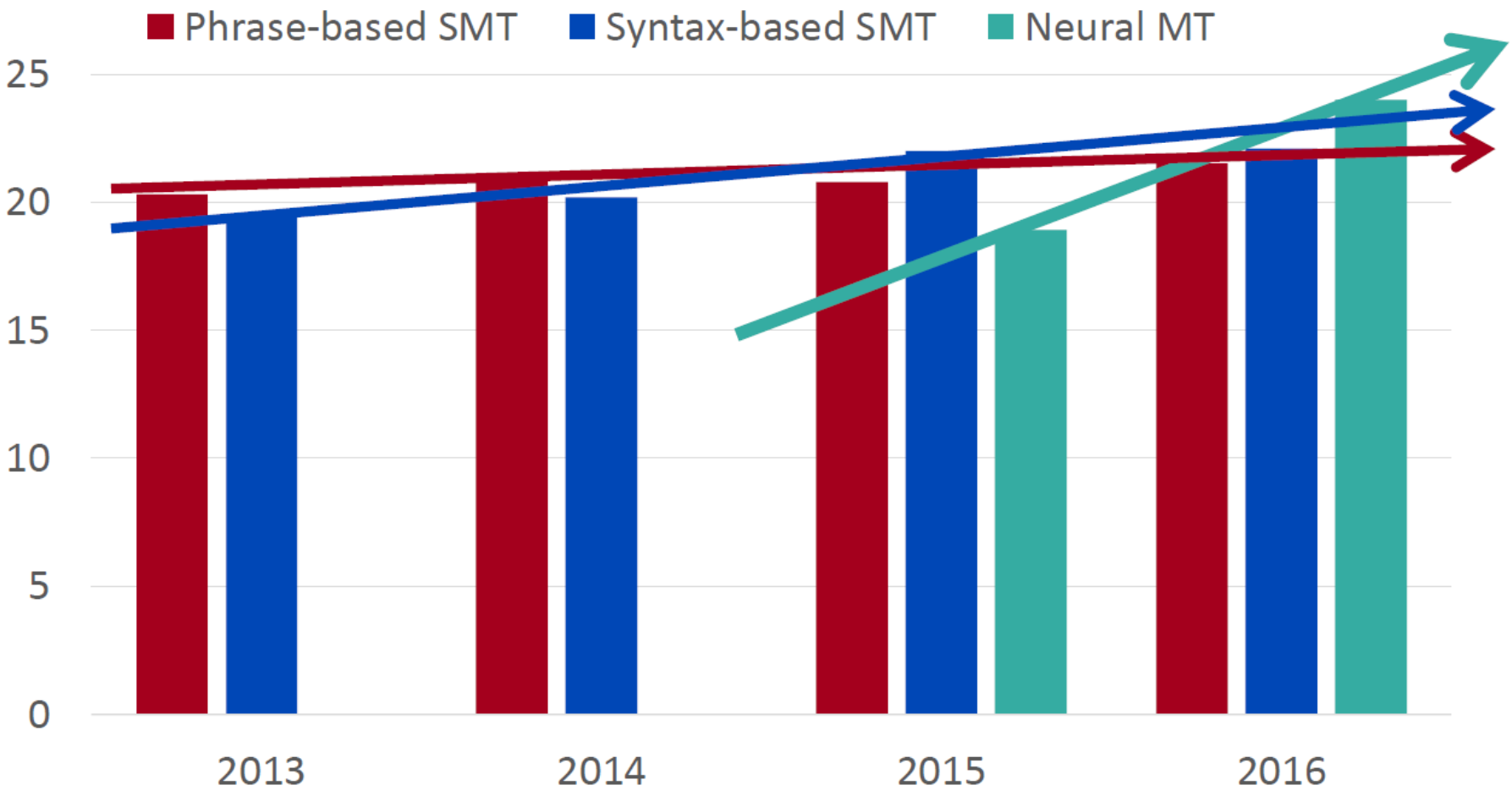
- BLEU was one of the first metrics to claim a high correlation with human judgements of quality, and remains one of the most popular automated and inexpensive metrics

- There are many valid ways to translate a sentence

- So a good translation can get a poor BLEU score because it has low *n*-gram overlap with the human translation L

MT progress over time

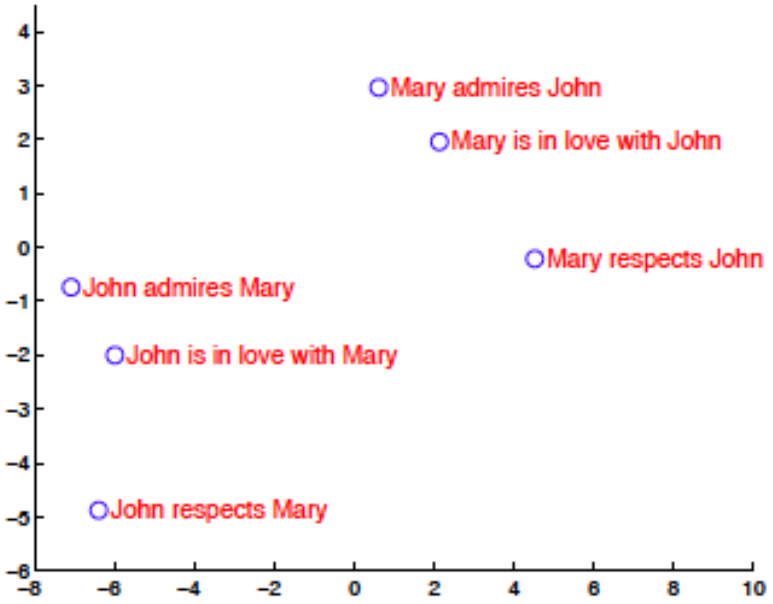
[Edinburgh En-De WMT newstest2013 Cased BLEU; NMT 2015 from U. Montréal]

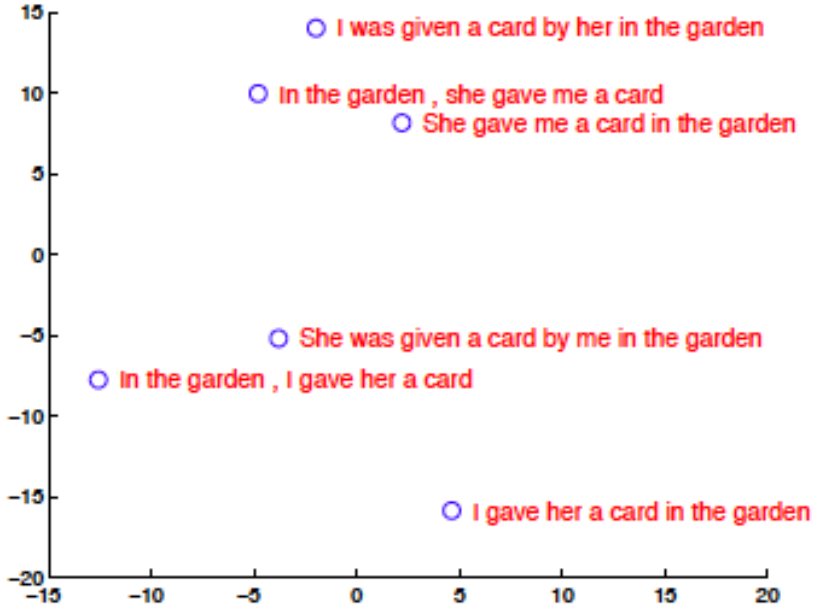


Source: http://www.meta-net.eu/events/meta-forum-2016/slides/09_sennrich.pdf

NMT Analysis

- Model Analysis

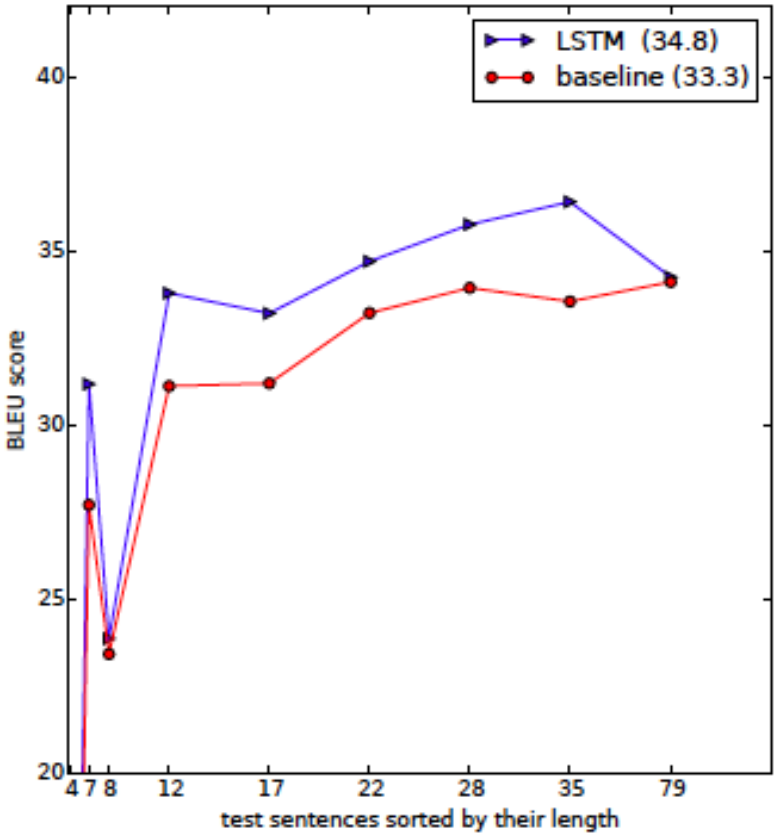


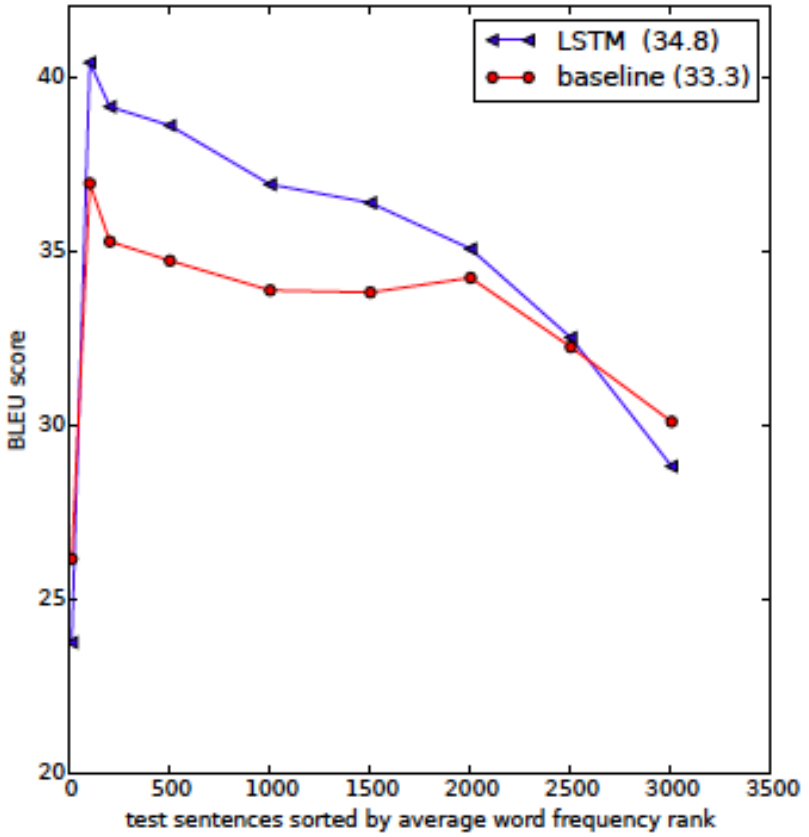


Sutskever et al. 2014

NMT Analysis

- Long sentences





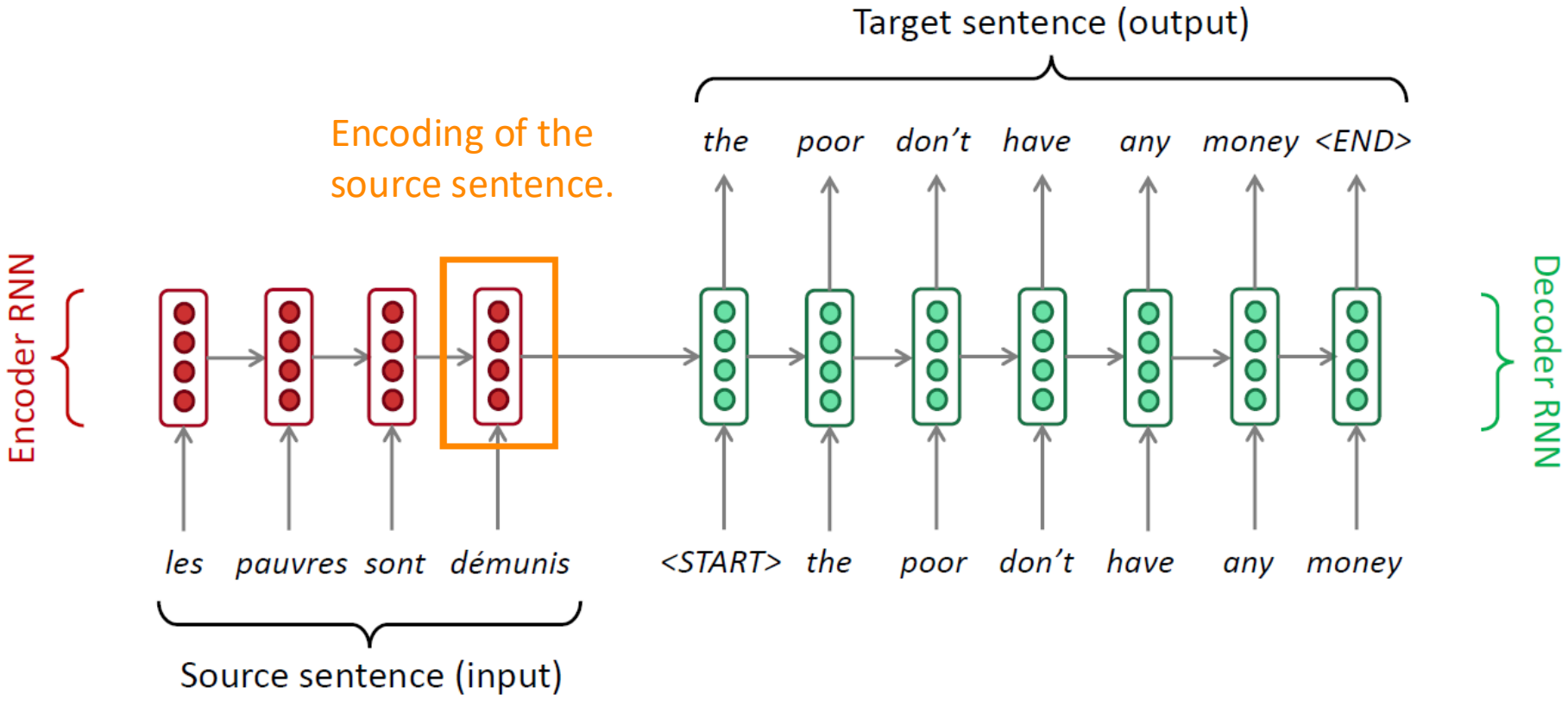
Sutskever et al. 2014

NMT: the biggest success story of NLP Deep Learning

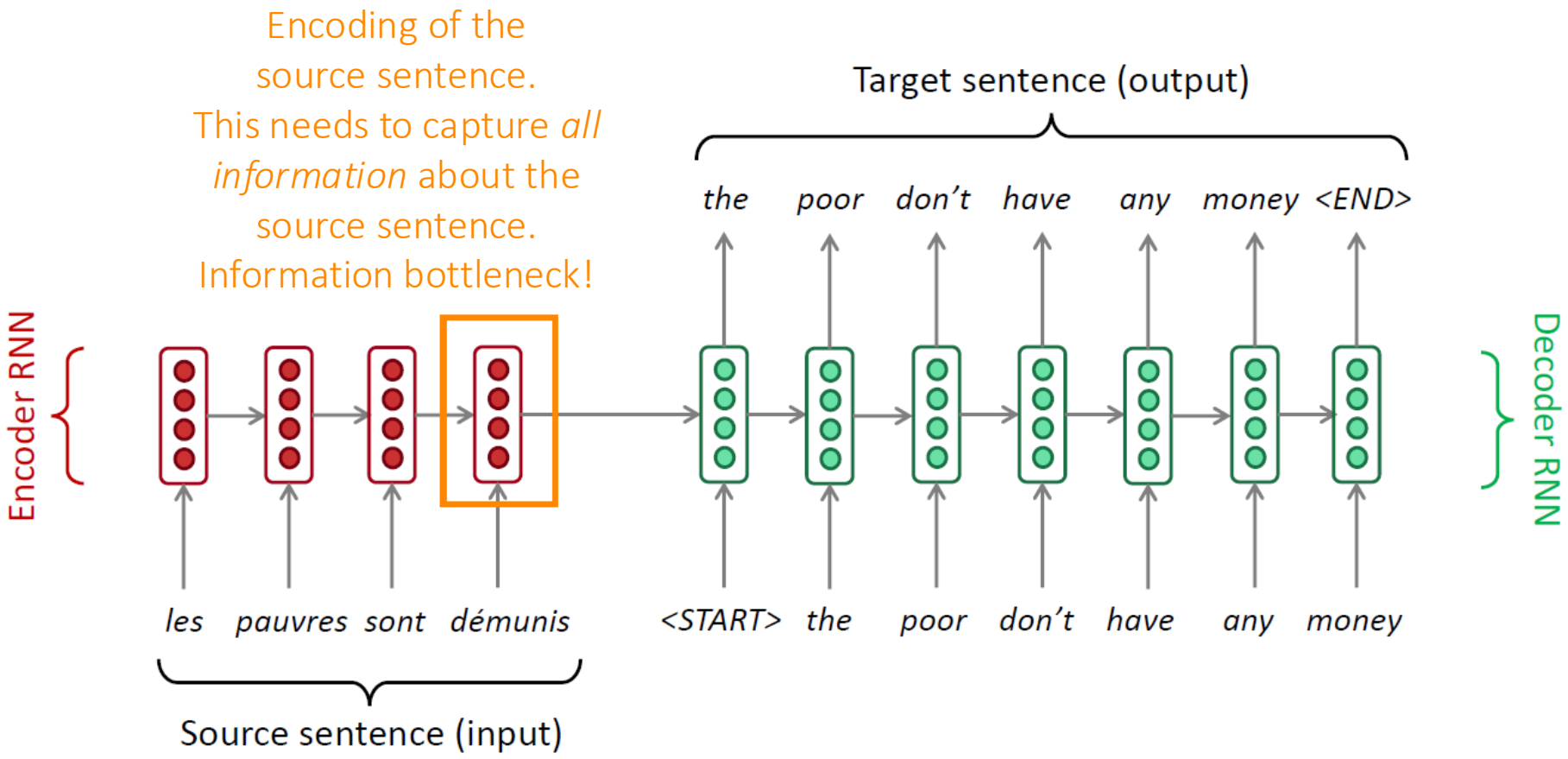
- Neural Machine Translation went from a fringe research activity in **2014** to the leading standard method in **2016**
  - **2014**: First seq2seq paper published
  - **2016**: Google Translate switches from SMT to NMT

- This is amazing!
  - **SMT** systems, built by hundreds of engineers over many years, outperformed by NMT systems trained by a handful of engineers in a few months

Sequence-to-sequence: the bottleneck problem



Sequence-to-sequence: the bottleneck problem



Attention

- **Attention** provides a solution to the bottleneck problem.

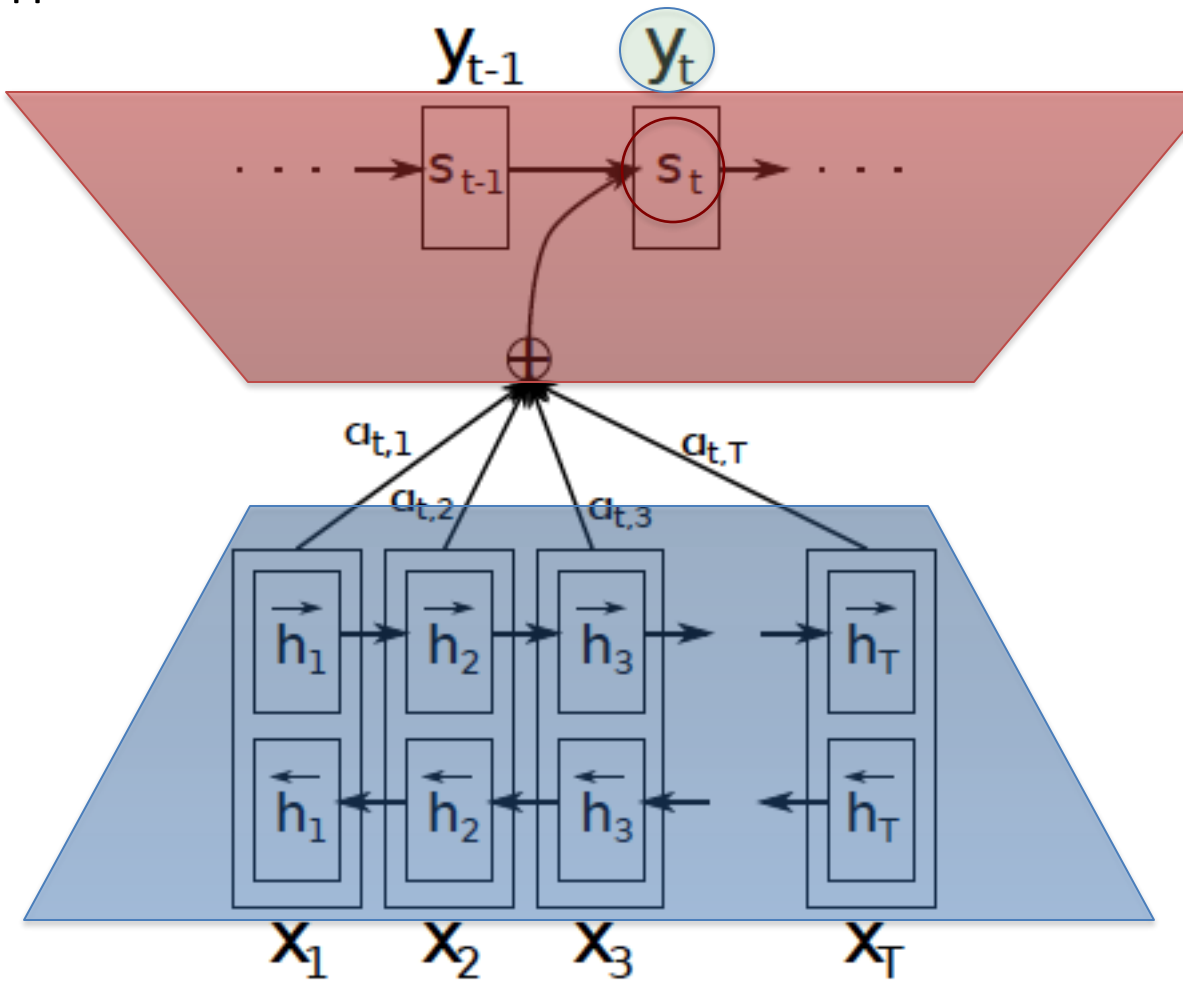
- Core idea: on each step of the decoder, *focus on a particular part* of the source sequence

Jointly Learning to Align and Translate

- Attention mechanism

$$p(y_t|y_{t-1}, \dots, y_1, \boldsymbol{x}) = f(\boldsymbol{s}_t)$$

$$\boldsymbol{s}_t = g(\boldsymbol{s}_{t-1}, y_{t-1}, \boldsymbol{c}_t)$$



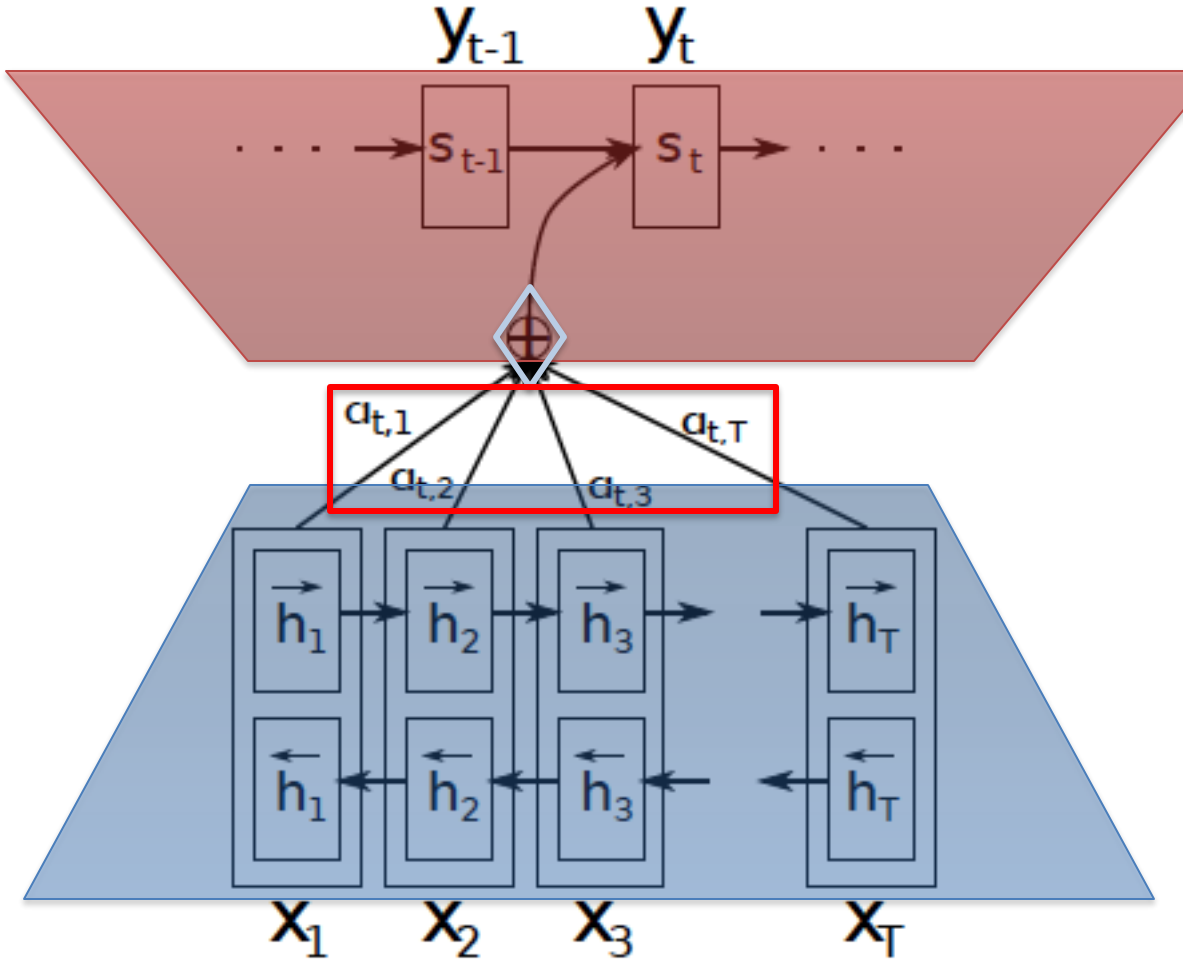
Jointly Learning to Align and Translate

- Attention mechanism

$$p(y_t|y_{t-1}, \dots, y_1, \boldsymbol{x}) = f(\boldsymbol{s}_t)$$

$$\boldsymbol{s}_t = g(\boldsymbol{s}_{t-1}, y_{t-1}, \boldsymbol{c}_t)$$

$$\boldsymbol{c}_t = \sum_{j=1}^{T} \alpha_{t,j} \boldsymbol{h}_j$$



Jointly Learning to Align and Translate

- Attention mechanism

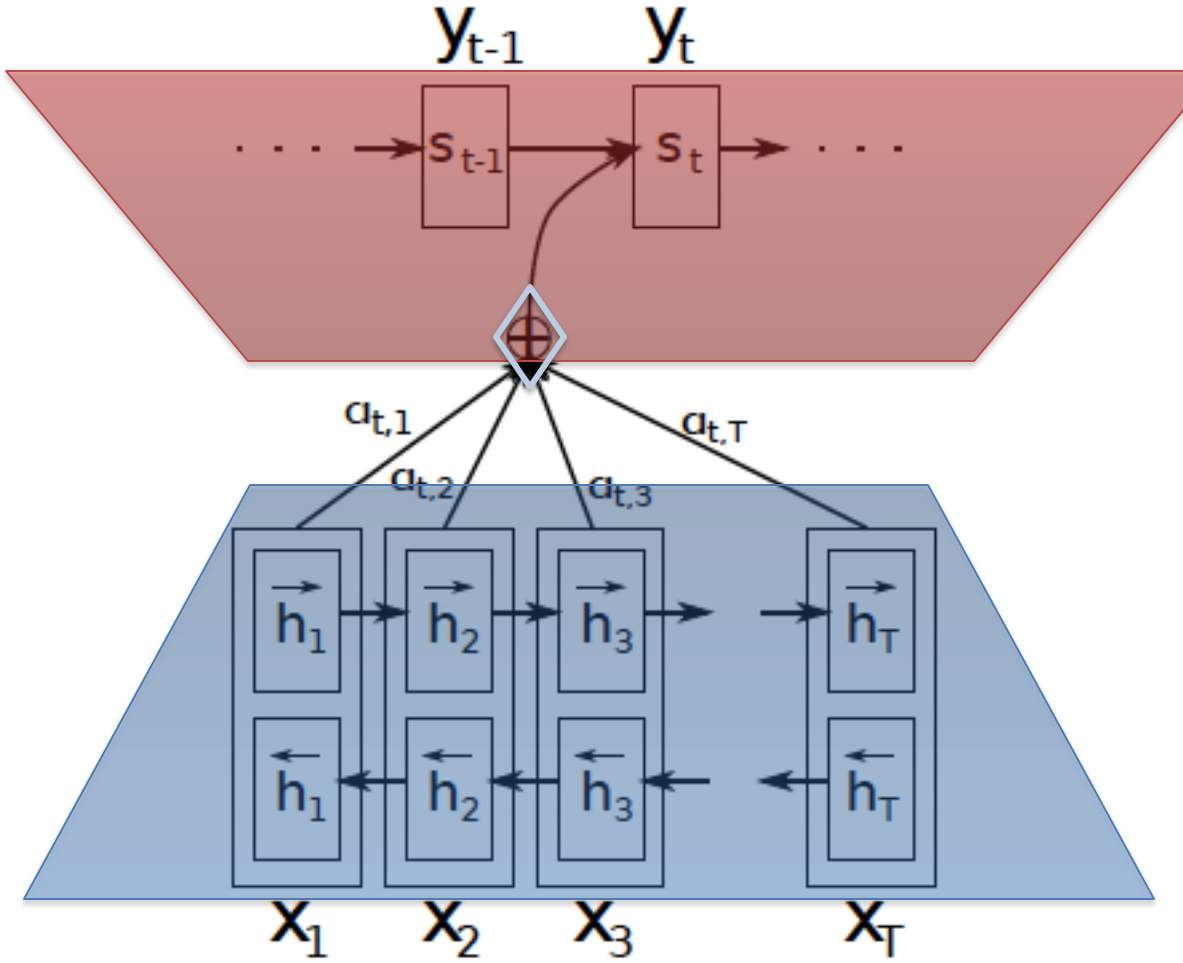
$$p(y_t|y_{t-1},\dots,y_1,\boldsymbol{x}) = f(\boldsymbol{s}_t)$$

$$\boldsymbol{s}_t = g(\boldsymbol{s}_{t-1}, y_{t-1}, \boldsymbol{c}_t)$$

$$\boldsymbol{c}_t = \sum_{j=1}^{T} \alpha_{t,j}\boldsymbol{h}_j$$

$$\alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^{T}\exp(e_{t,k})}$$

$$e_{t,j} = a(\boldsymbol{s}_{t-1}, \boldsymbol{h}_j)$$



Jointly Learning to Align and Translate

- Attention mechanism

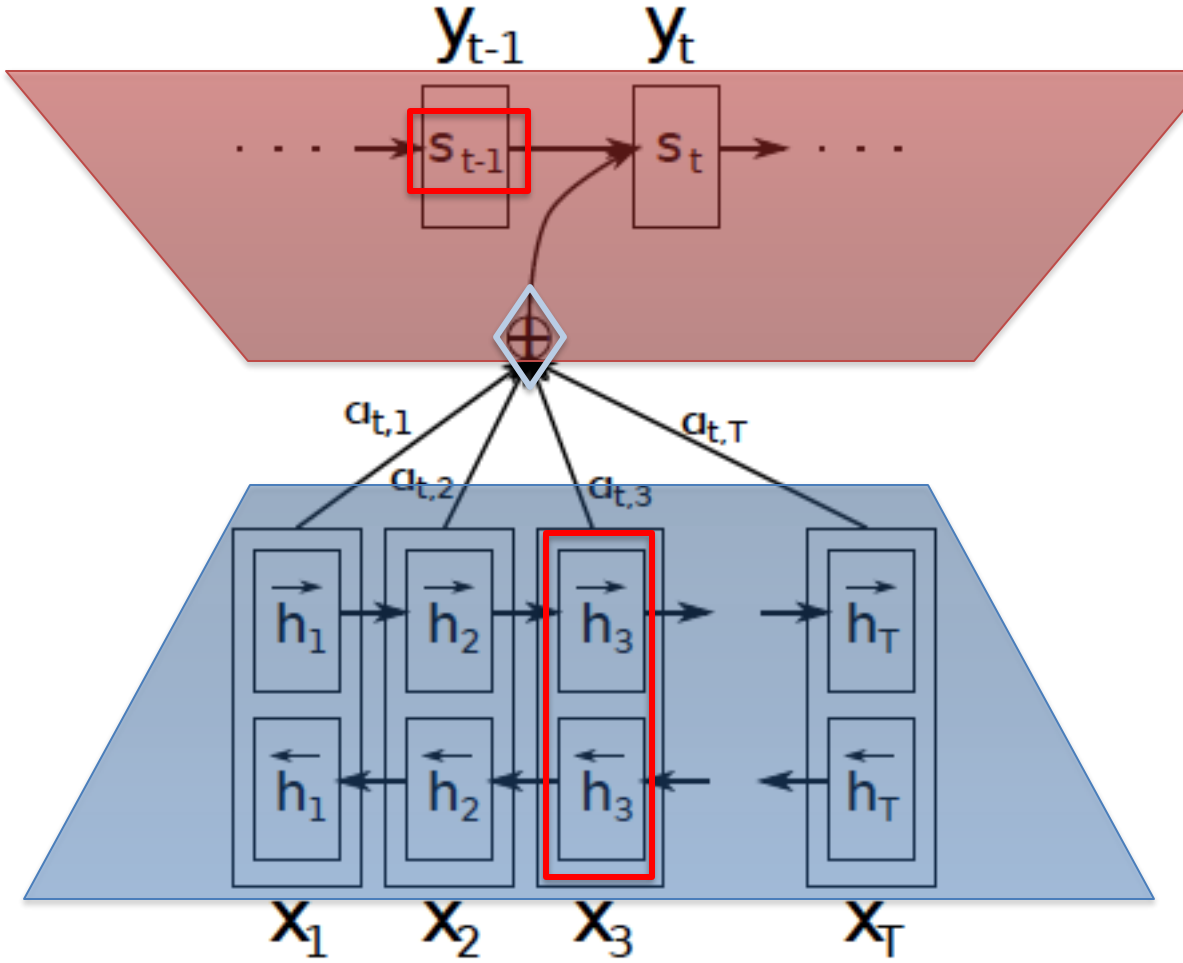
$$p(y_t|y_{t-1},\dots,y_1,\boldsymbol{x}) = f(\boldsymbol{s}_t)$$

$$\boldsymbol{s}_t = g(\boldsymbol{s}_{t-1}, y_{t-1}, \boldsymbol{c}_t)$$

$$\boldsymbol{c}_t = \sum_{j=1}^{T} \alpha_{t,j}\boldsymbol{h}_j$$

$$\alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^{T}\exp(e_{t,k})}$$

$$e_{t,j} = a(\boldsymbol{s}_{t-1}, \boldsymbol{h}_j)$$



Attention variants

$$p(y_t|y_{t-1}, \dots, y_1, \boldsymbol{x}) = f(\boldsymbol{s}_t)$$

$$\boldsymbol{s}_t = g(\boldsymbol{s}_{t-1}, y_{t-1}, \boldsymbol{c}_t)$$

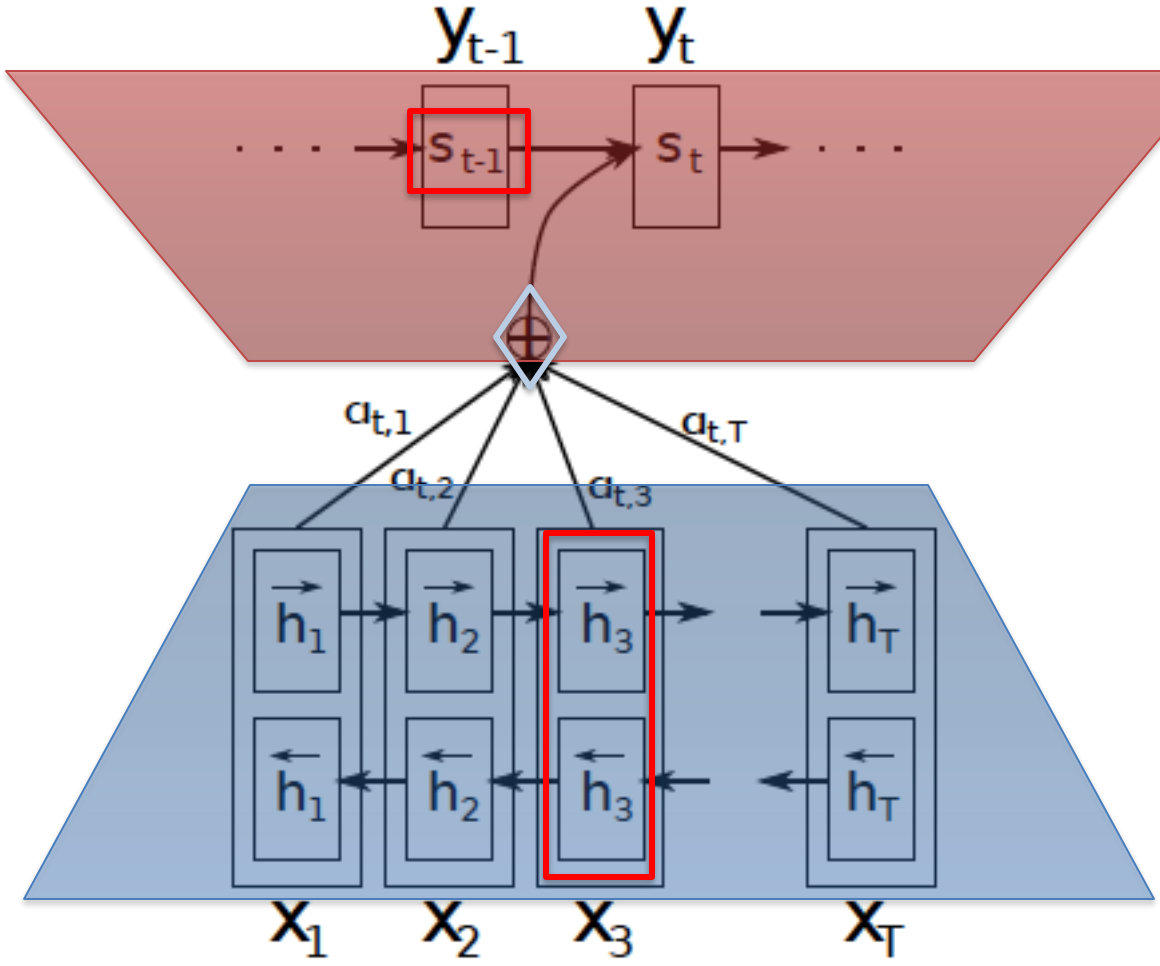
$$\boldsymbol{c}_t = \sum_{j=1}^{T} \alpha_{t,j} \boldsymbol{h}_j$$

$$\alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^{T} \exp(e_{t,k})}$$

$$e_{t,j} = a(\boldsymbol{s}_{t-1}, \boldsymbol{h}_j)$$

Basic dot-product attention

$$e_{tj} = s_{t-1}^T h_j$$



Attention variants

$$p(y_t|y_{t-1}, \dots, y_1, \boldsymbol{x}) = f(\boldsymbol{s}_t)$$

$$\boldsymbol{s}_t = g(\boldsymbol{s}_{t-1}, y_{t-1}, \boldsymbol{c}_t)$$

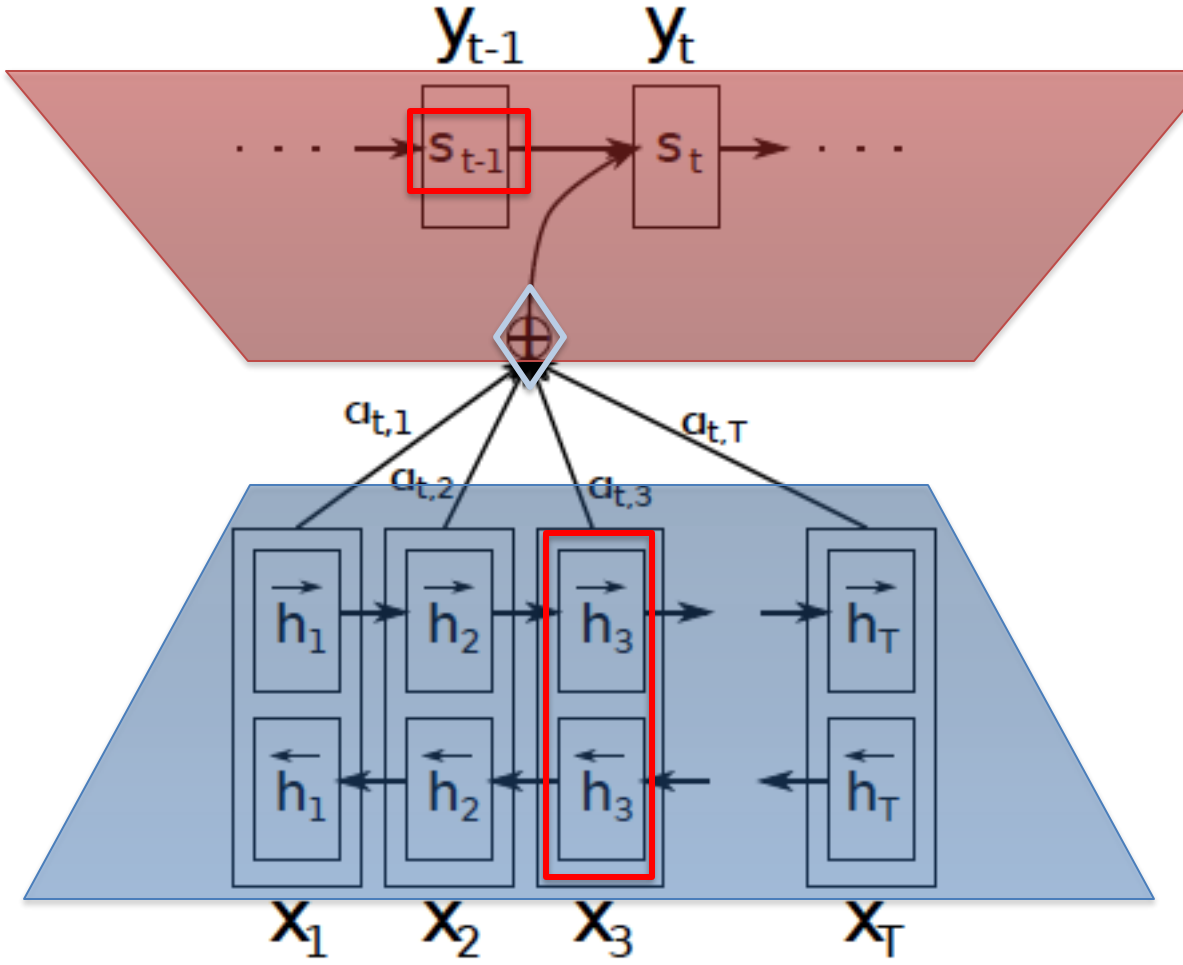
$$\boldsymbol{c}_t = \sum_{j=1}^{T} \alpha_{t,j} \boldsymbol{h}_j$$

$$\alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^{T} \exp(e_{t,k})}$$

$$e_{t,j} = a(\boldsymbol{s}_{t-1}, \boldsymbol{h}_j)$$

Multiplicative attention .
(bilinear mode)

$$e_{tj} = s_{t-1}^T W h_j$$



Attention variants

$$p(y_t|y_{t-1}, \dots, y_1, \boldsymbol{x}) = f(\boldsymbol{s}_t)$$

$$\boldsymbol{s}_t = g(\boldsymbol{s}_{t-1}, y_{t-1}, \boldsymbol{c}_t)$$

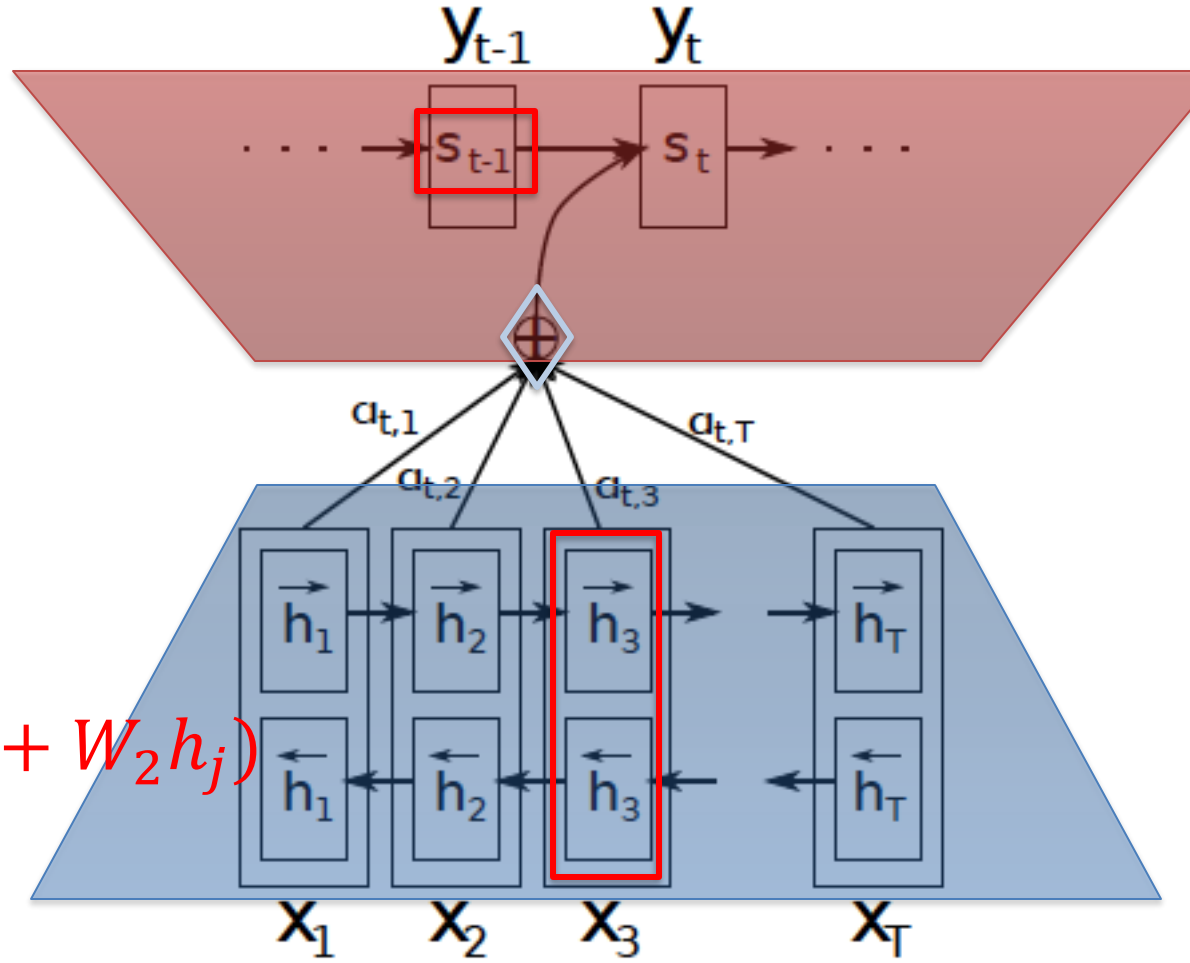
$$\boldsymbol{c}_t = \sum_{j=1}^{T} \alpha_{t,j} \boldsymbol{h}_j$$

$$\alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^{T} \exp(e_{t,k})}$$

$$e_{t,j} = a(\boldsymbol{s}_{t-1}, \boldsymbol{h}_j)$$

Additive attention

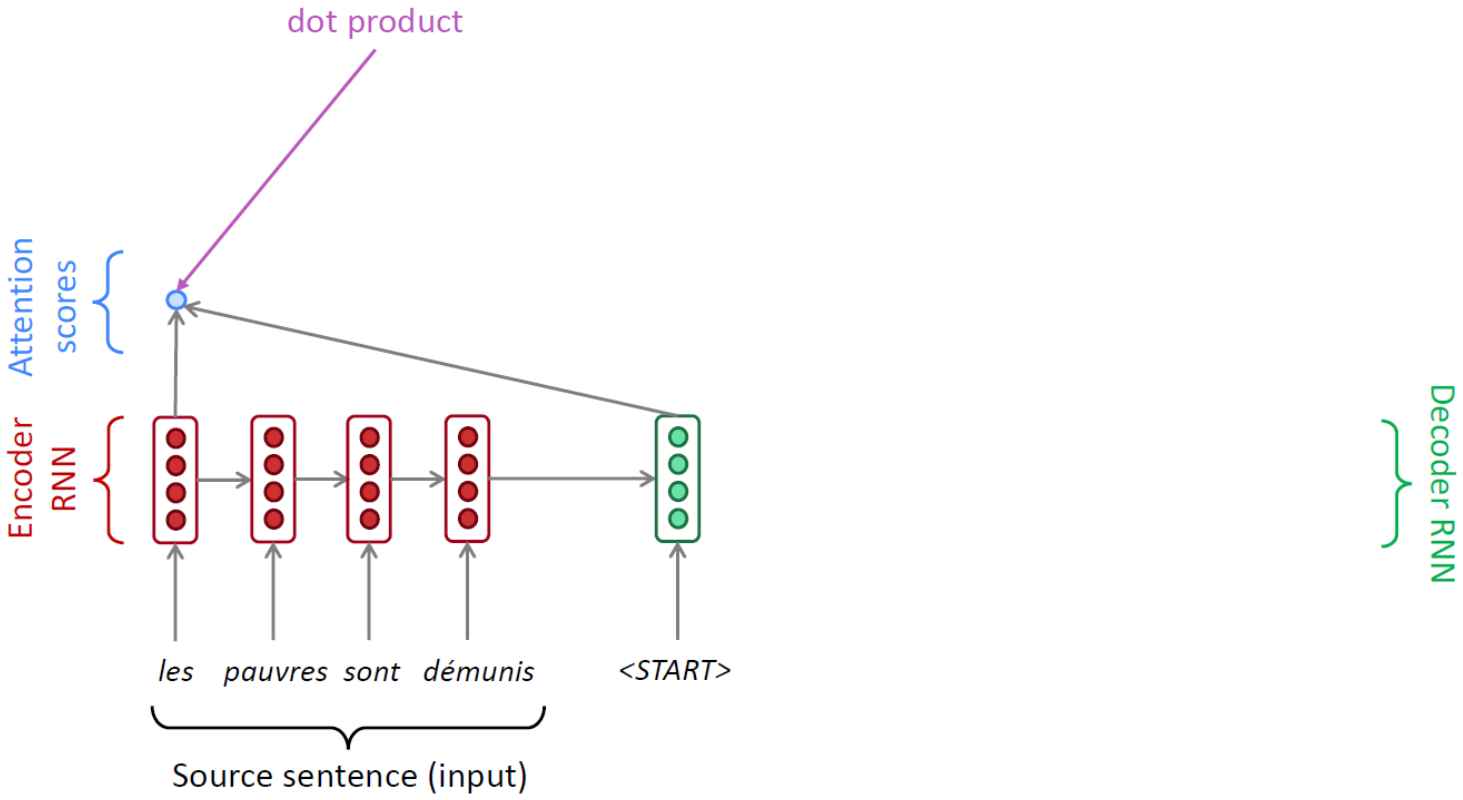
$$e_{tj} = v^T \tanh(W_1 s_{t-1}^T + W_2 h_j)$$



A Running Example (Slightly Different From Bengio's Original Method)

$$\alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^{T} \exp(e_{t,k})}$$

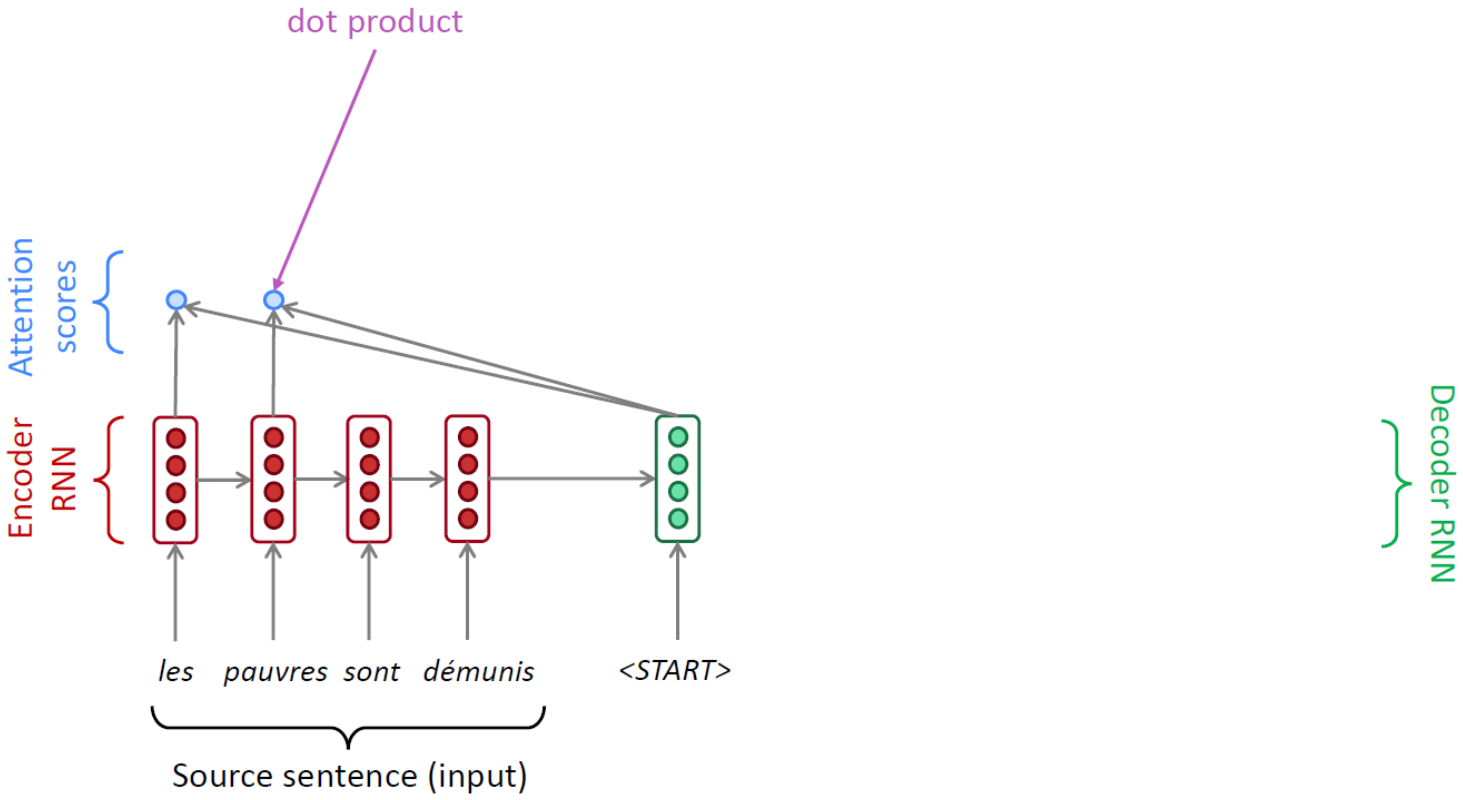
$$e_{t,j} = a(\boldsymbol{s}_t, \boldsymbol{h}_j)$$



A Running Example

$$\alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^{T} \exp(e_{t,k})}$$

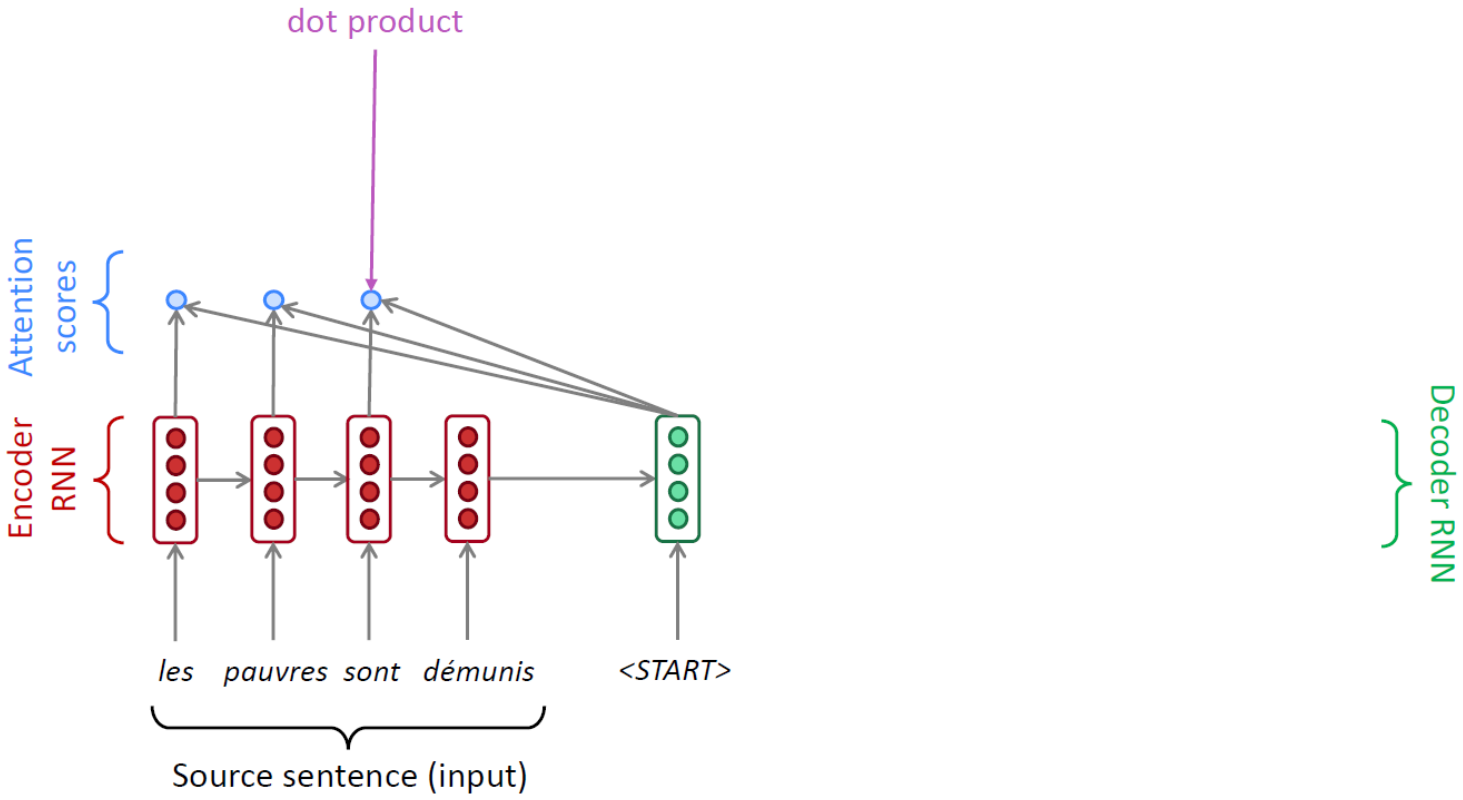
$$e_{t,j} = a(\boldsymbol{s}_t, \boldsymbol{h}_j)$$



A Running Example

$$\alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^{T} \exp(e_{t,k})}$$

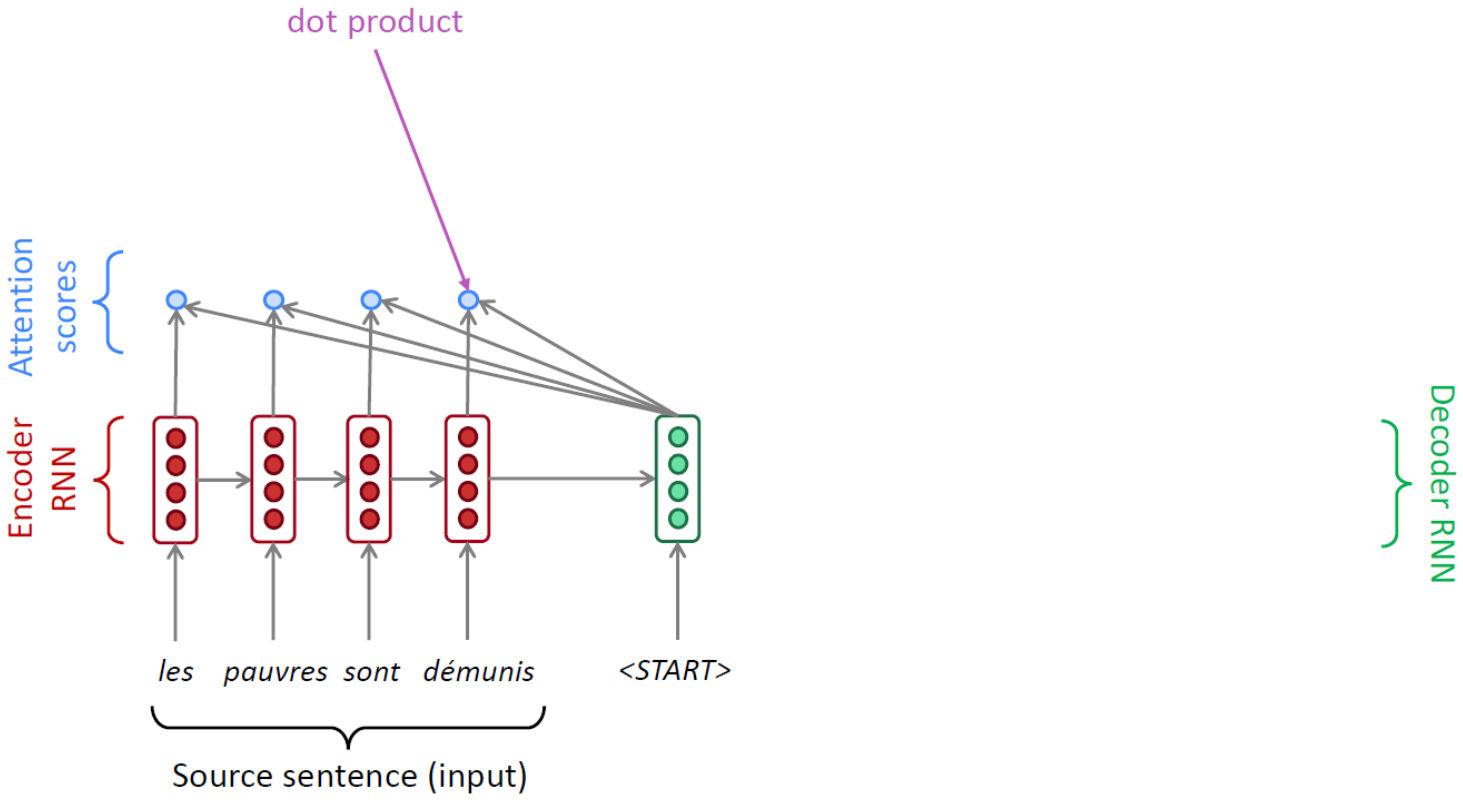
$$e_{t,j} = a(\boldsymbol{s}_t, \boldsymbol{h}_j)$$



A Running Example

$$\alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^{T} \exp(e_{t,k})}$$

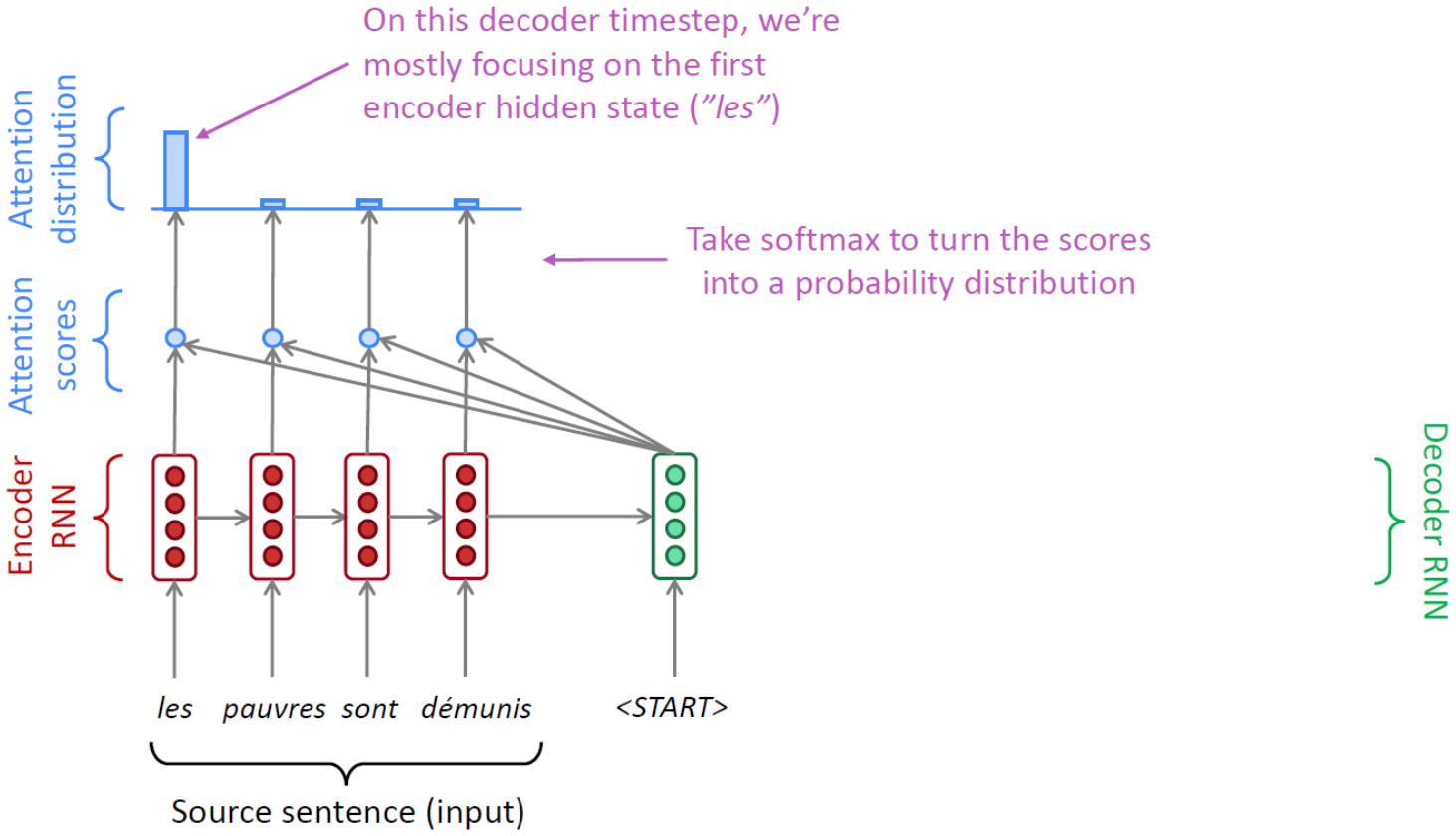
$$e_{t,j} = a(\boldsymbol{s}_t, \boldsymbol{h}_j)$$



A Running Example

$$\alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^{T} \exp(e_{t,k})}$$

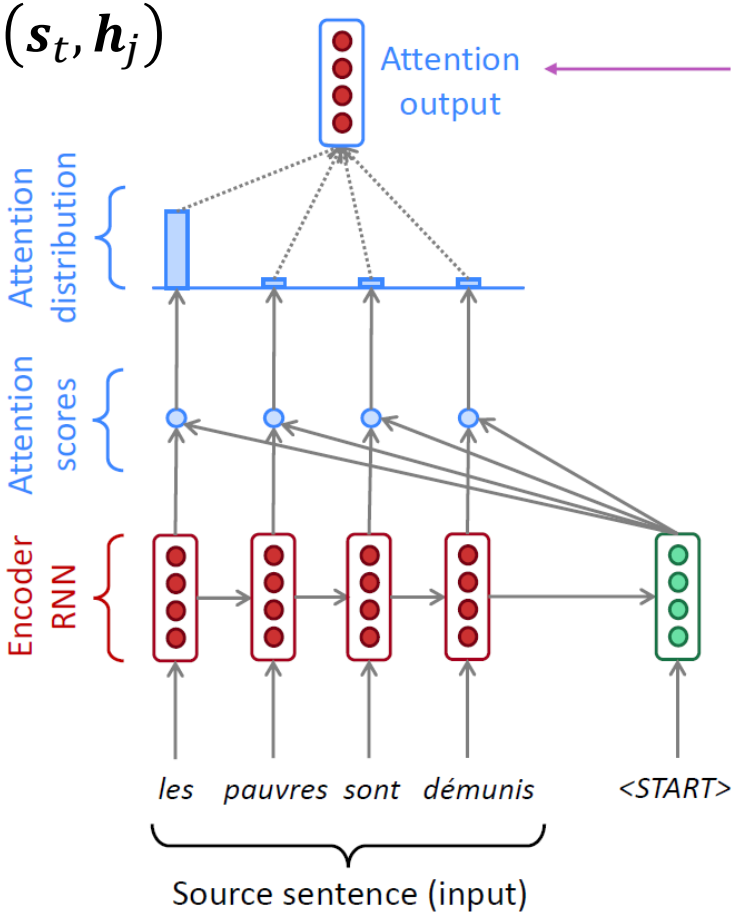
$$e_{t,j} = a(\boldsymbol{s}_t, \boldsymbol{h}_j)$$



A Running Example

$$\alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^{T} \exp(e_{t,k})} \qquad \boldsymbol{c}_t = \sum_{j=1}^{T} \alpha_{t,j} \boldsymbol{h}_j$$

$$e_{t,j} = a(\boldsymbol{s}_t, \boldsymbol{h}_j)$$



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

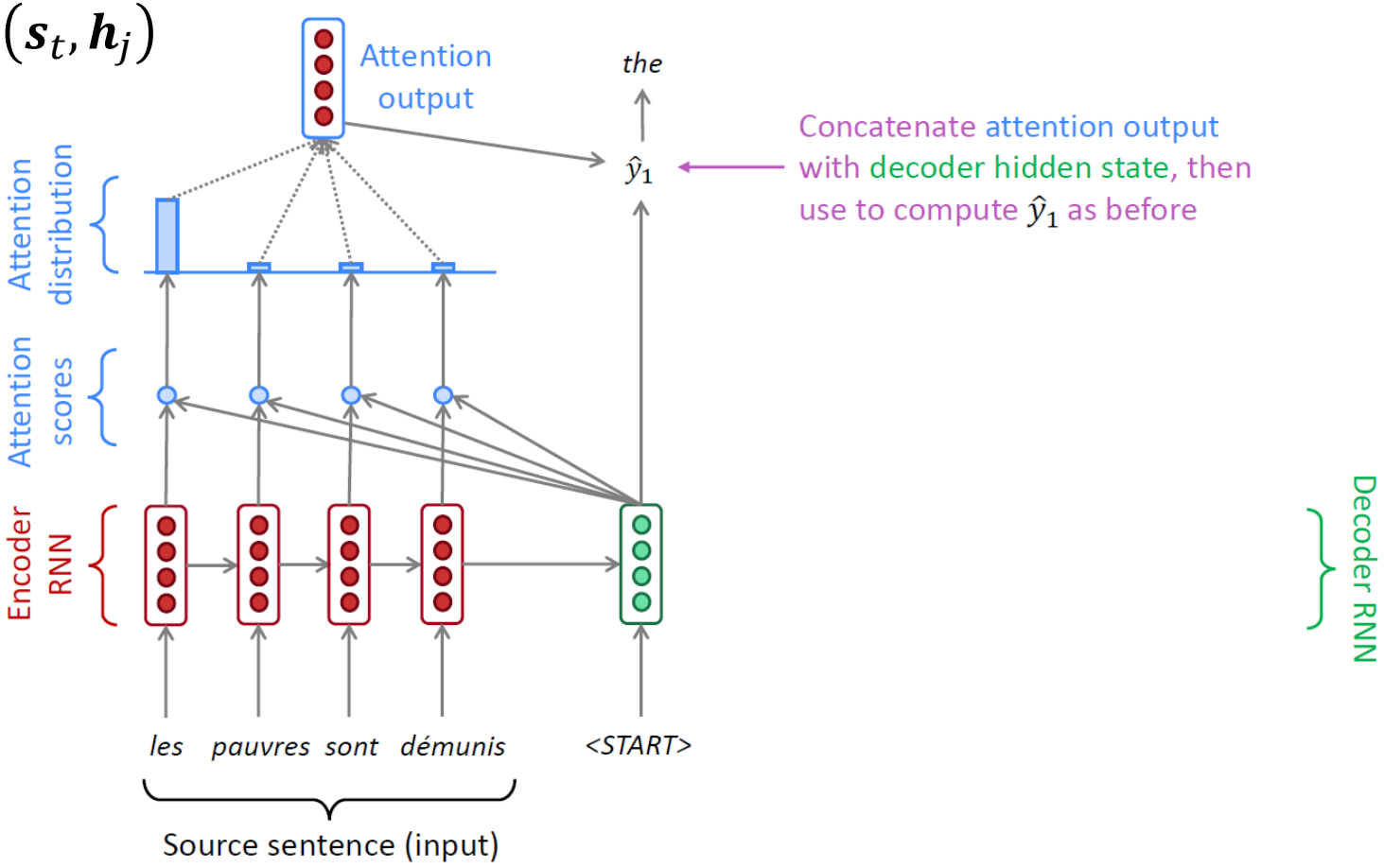
The attention output mostly contains information the hidden states that received high attention.

Decoder RNN

A Running Example

$$\alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^{T} \exp(e_{t,k})} \qquad \boldsymbol{c}_t = \sum_{j=1}^{T} \alpha_{t,j} \boldsymbol{h}_j \qquad p(y_t | y_{t-1}, \dots, y_1, \boldsymbol{x}) = f(y_{t-1}, \boldsymbol{s}_t, \boldsymbol{c}_t)$$

$$e_{t,j} = a(\boldsymbol{s}_t, \boldsymbol{h}_j)$$



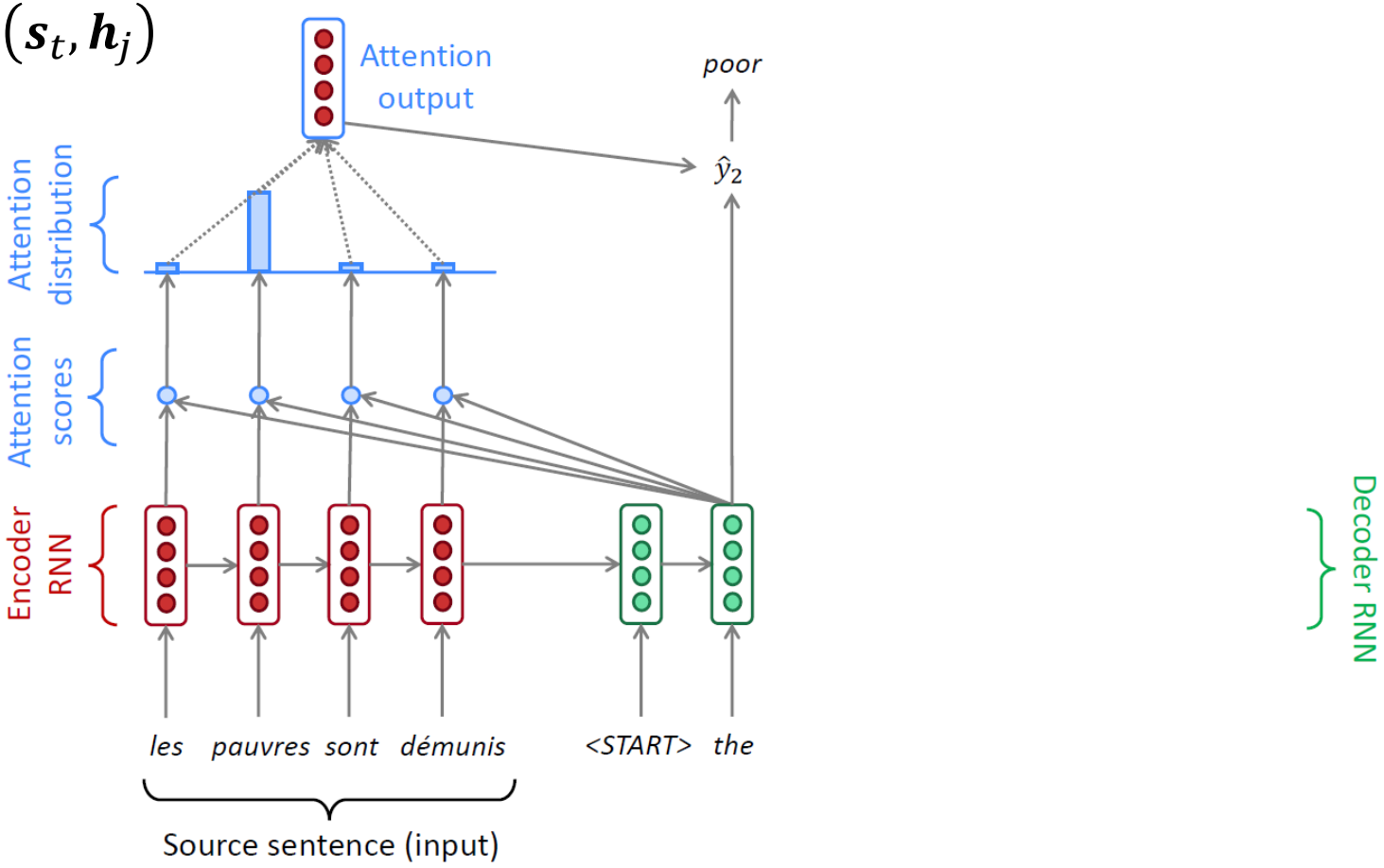
A Running Example

$$\alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^{T} \exp(e_{t,k})} \qquad \boldsymbol{c}_t = \sum_{j=1}^{T} \alpha_{t,j} \boldsymbol{h}_j$$

$$p(y_t | y_{t-1}, \dots, y_1, \boldsymbol{x}) = f(y_{t-1}, \boldsymbol{s}_t, \boldsymbol{c}_t)$$

$$\boldsymbol{s}_{t+1} = g(\boldsymbol{s}_t, y_t)$$

$$e_{t,j} = a(\boldsymbol{s}_t, \boldsymbol{h}_j)$$

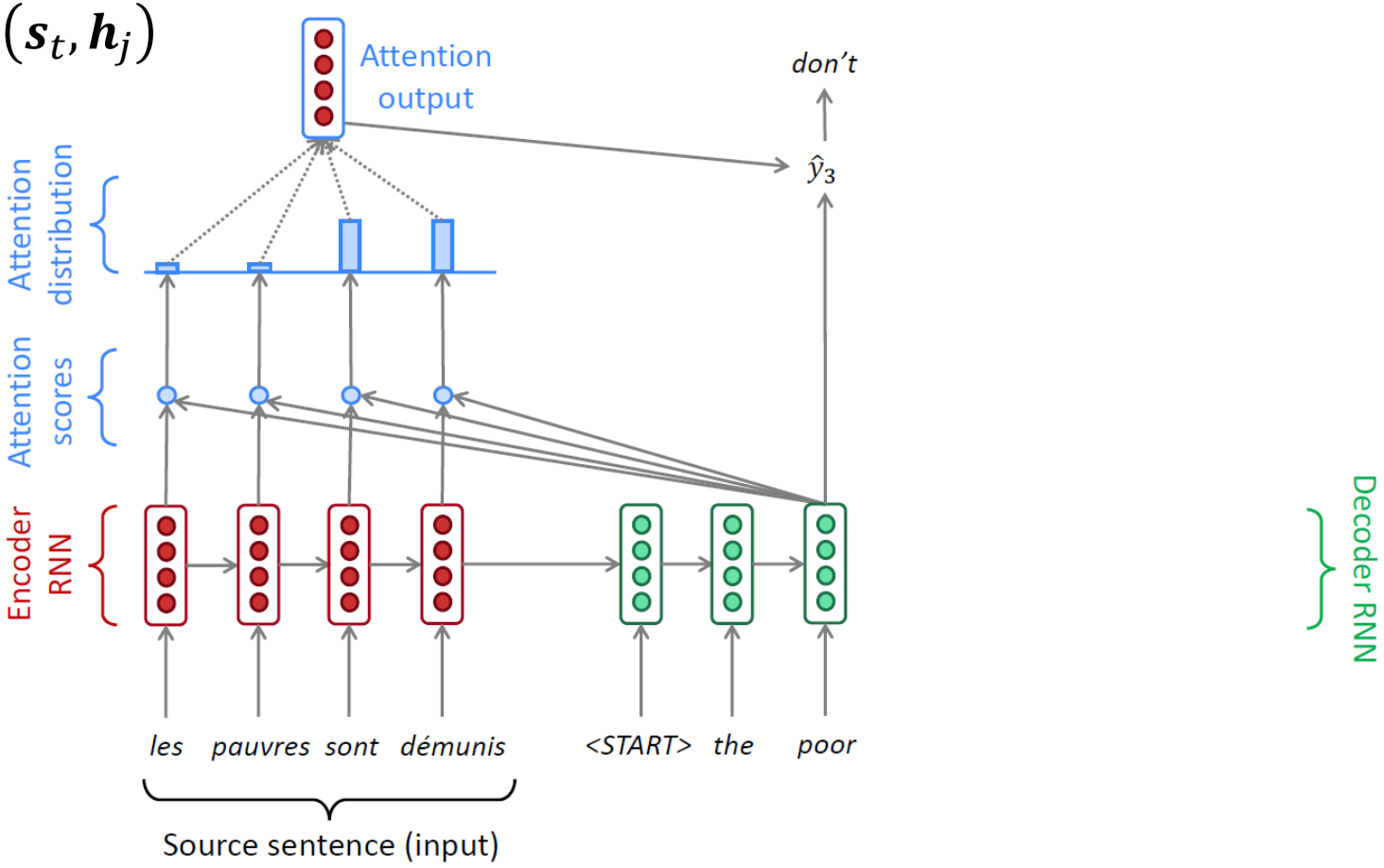


A Running Example

$$\alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^{T} \exp(e_{t,k})} \qquad \boldsymbol{c}_t = \sum_{j=1}^{T} \alpha_{t,j} \boldsymbol{h}_j \qquad p(y_t | y_{t-1}, \dots, y_1, \boldsymbol{x}) = f(y_{t-1}, \boldsymbol{s}_t, \boldsymbol{c}_t)$$

$$\boldsymbol{s}_{t+1} = g(\boldsymbol{s}_t, y_t)$$

$$e_{t,j} = a(\boldsymbol{s}_t, \boldsymbol{h}_j)$$

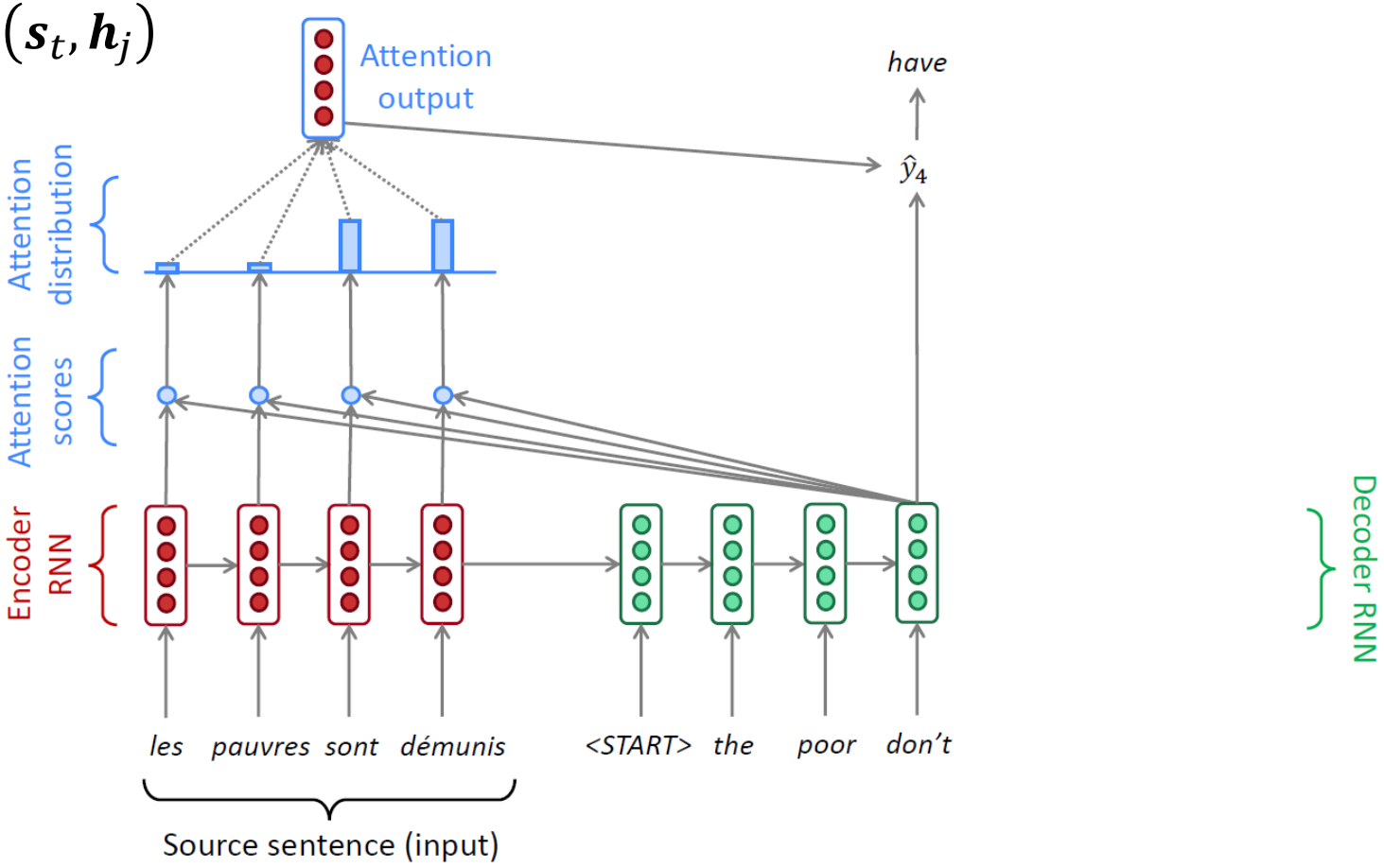


A Running Example

$$\alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^{T} \exp(e_{t,k})} \qquad \boldsymbol{c}_t = \sum_{j=1}^{T} \alpha_{t,j} \boldsymbol{h}_j \qquad p(y_t | y_{t-1}, \dots, y_1, \boldsymbol{x}) = f(y_{t-1}, \boldsymbol{s}_t, \boldsymbol{c}_t)$$

$$\boldsymbol{s}_{t+1} = g(\boldsymbol{s}_t, y_t)$$

$$e_{t,j} = a(\boldsymbol{s}_t, \boldsymbol{h}_j)$$

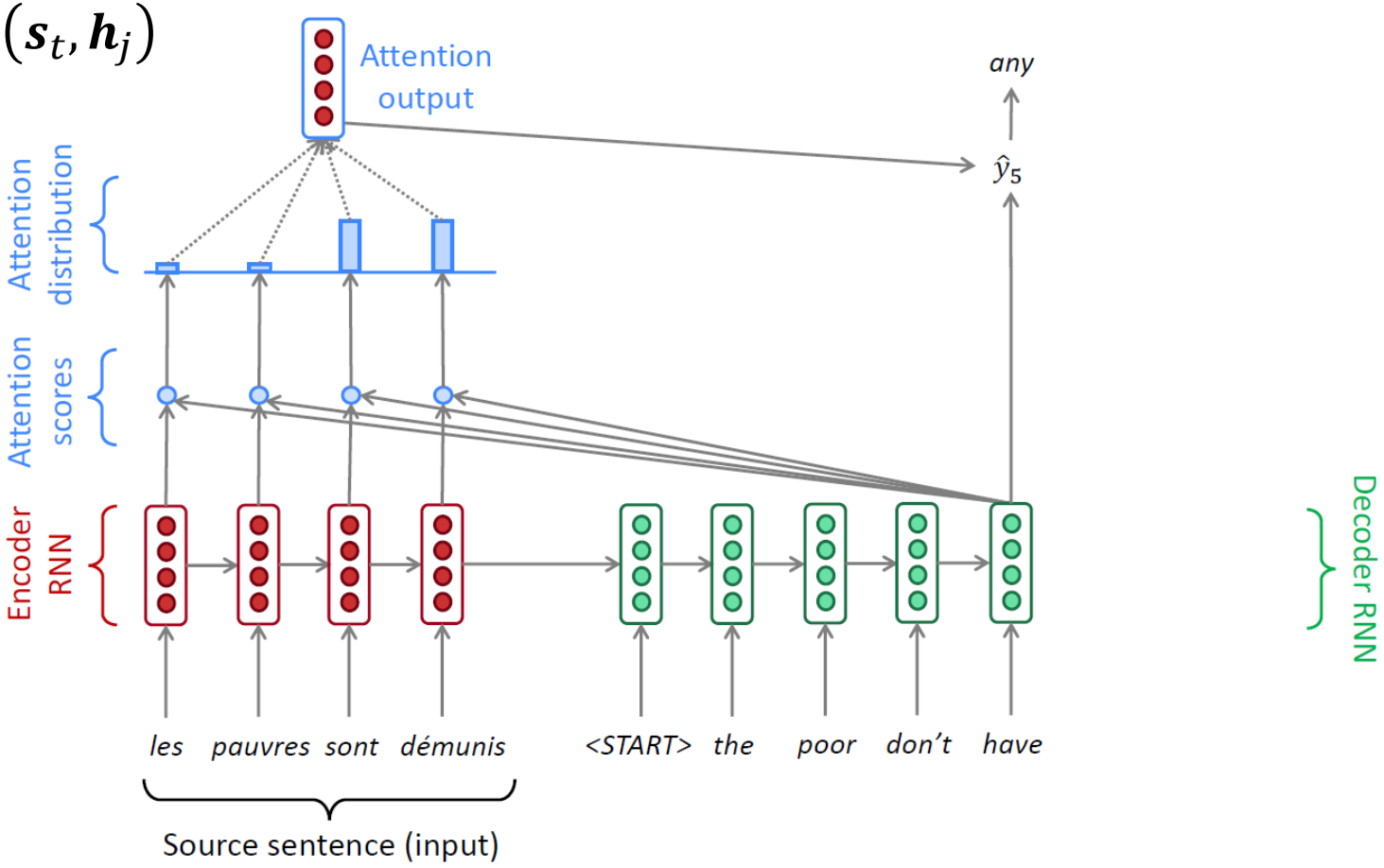


A Running Example

$$\alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^{T} \exp(e_{t,k})} \qquad \boldsymbol{c}_t = \sum_{j=1}^{T} \alpha_{t,j} \boldsymbol{h}_j \qquad p(y_t | y_{t-1}, \dots, y_1, \boldsymbol{x}) = f(y_{t-1}, \boldsymbol{s}_t, \boldsymbol{c}_t)$$

$$\boldsymbol{s}_{t+1} = g(\boldsymbol{s}_t, y_t)$$

$$e_{t,j} = a(\boldsymbol{s}_t, \boldsymbol{h}_j)$$

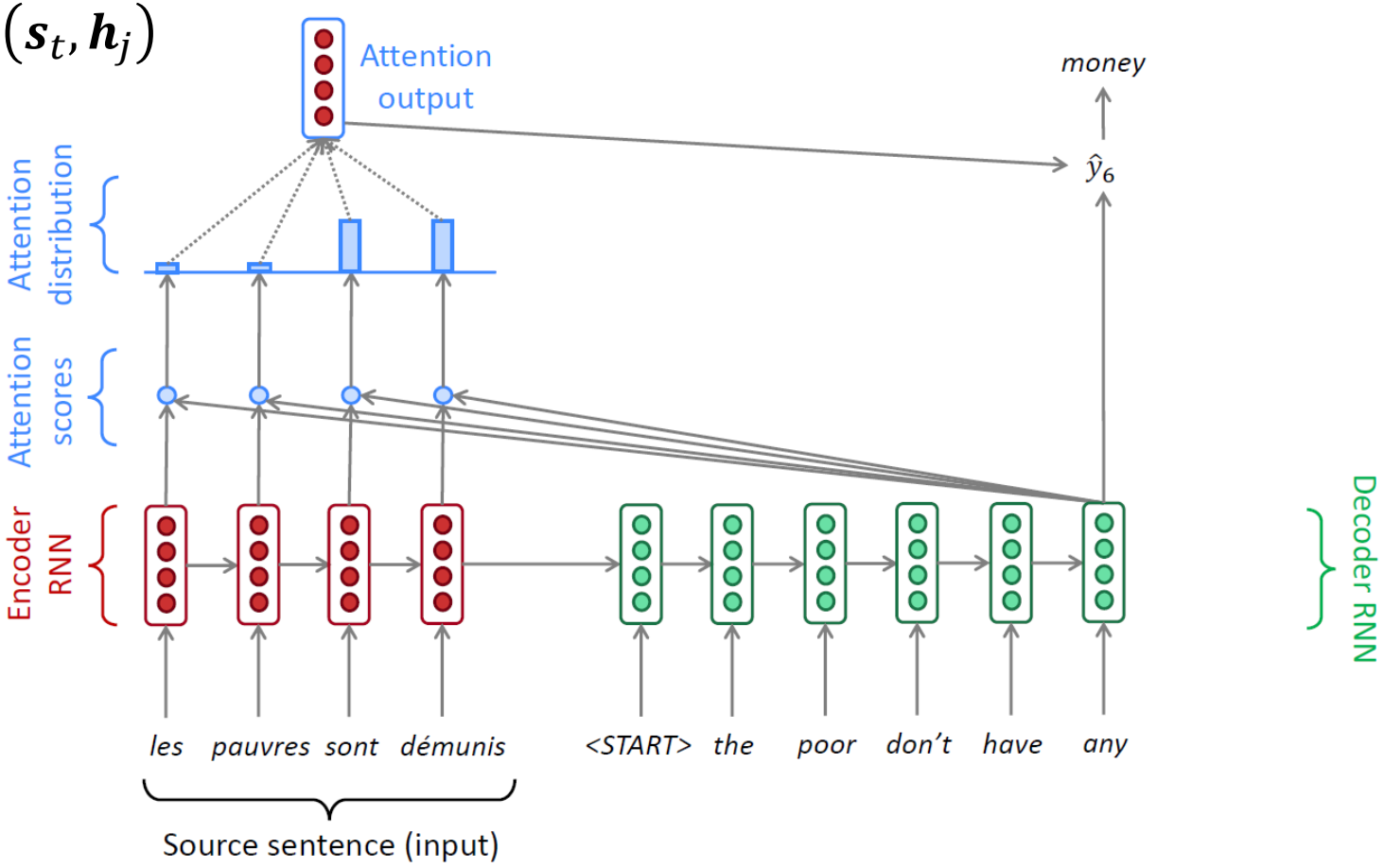


A Running Example

$$\alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^{T} \exp(e_{t,k})} \qquad \boldsymbol{c}_t = \sum_{j=1}^{T} \alpha_{t,j} \boldsymbol{h}_j \qquad p(y_t | y_{t-1}, \dots, y_1, \boldsymbol{x}) = f(y_{t-1}, \boldsymbol{s}_t, \boldsymbol{c}_t)$$

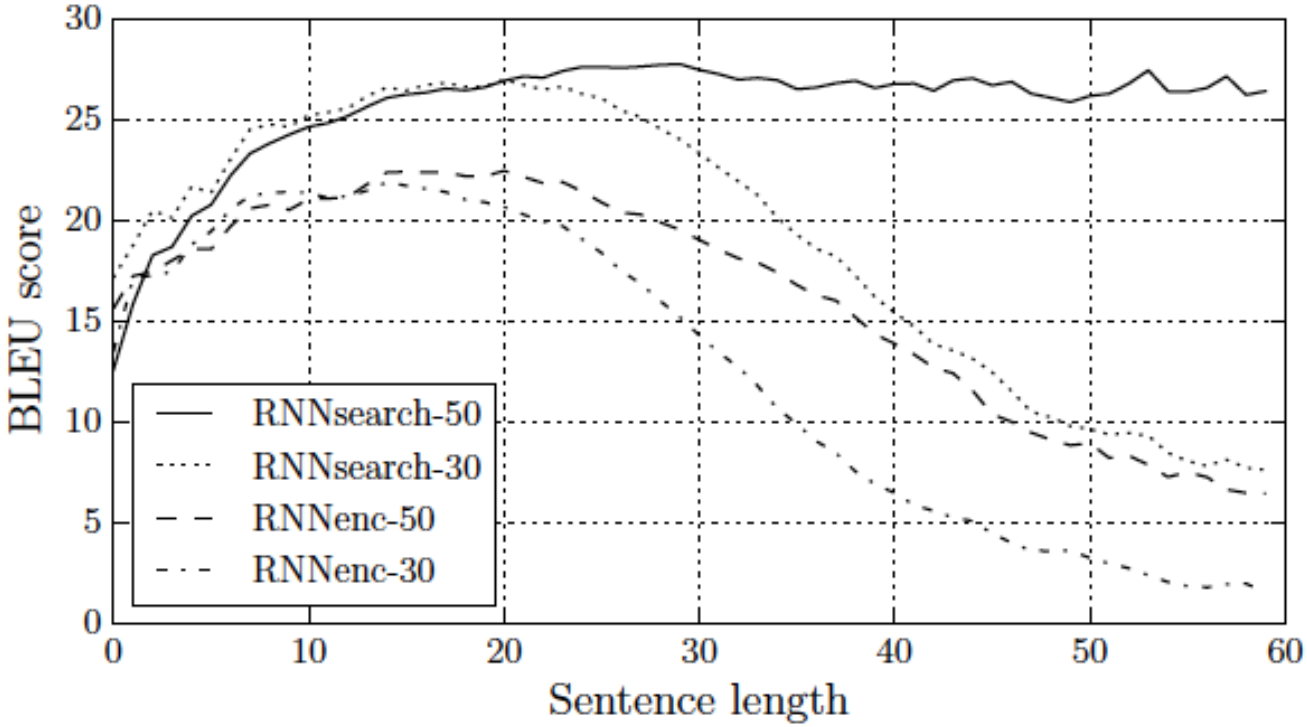
$$\boldsymbol{s}_{t+1} = g(\boldsymbol{s}_t, y_t)$$

$$e_{t,j} = a(\boldsymbol{s}_t, \boldsymbol{h}_j)$$



Jointly Learning to Align and Translate

- No performance drop for long sentences



"We train each model twice: first with the sentences of length up to 30 words (RNNencdec-30, RNNsearch-30) and then with the sentences of length up to 50 word (RNNencdec-50, RNNsearch-50)."

RNNencdec: Cho et al. 2014 https://www.aclweb.org/anthology/D14-1179
RNNsearch: Bahdanauet al, ICLR 2015, https://arxiv.org/pdf/1409.0473.pdf

OpenNMT

- http://opennmt.net/

Google Neural Machine Translation (Wu et al. (2016))

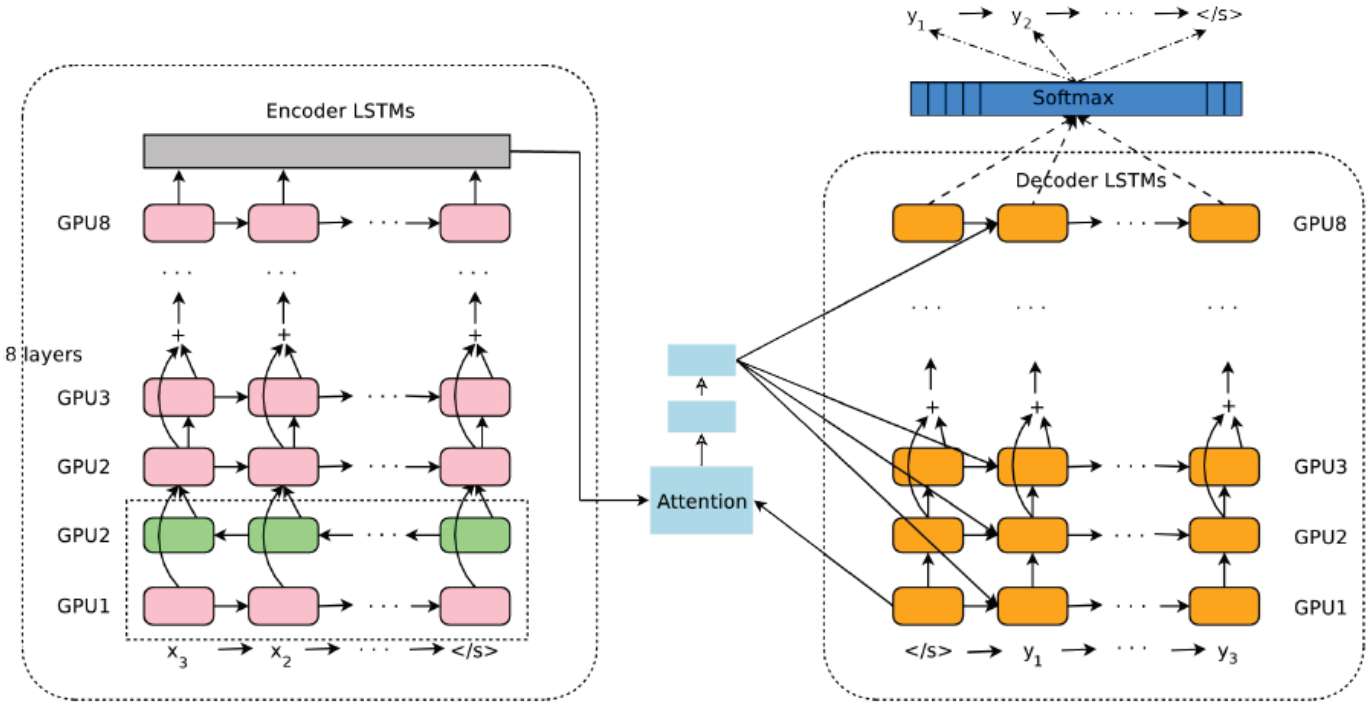
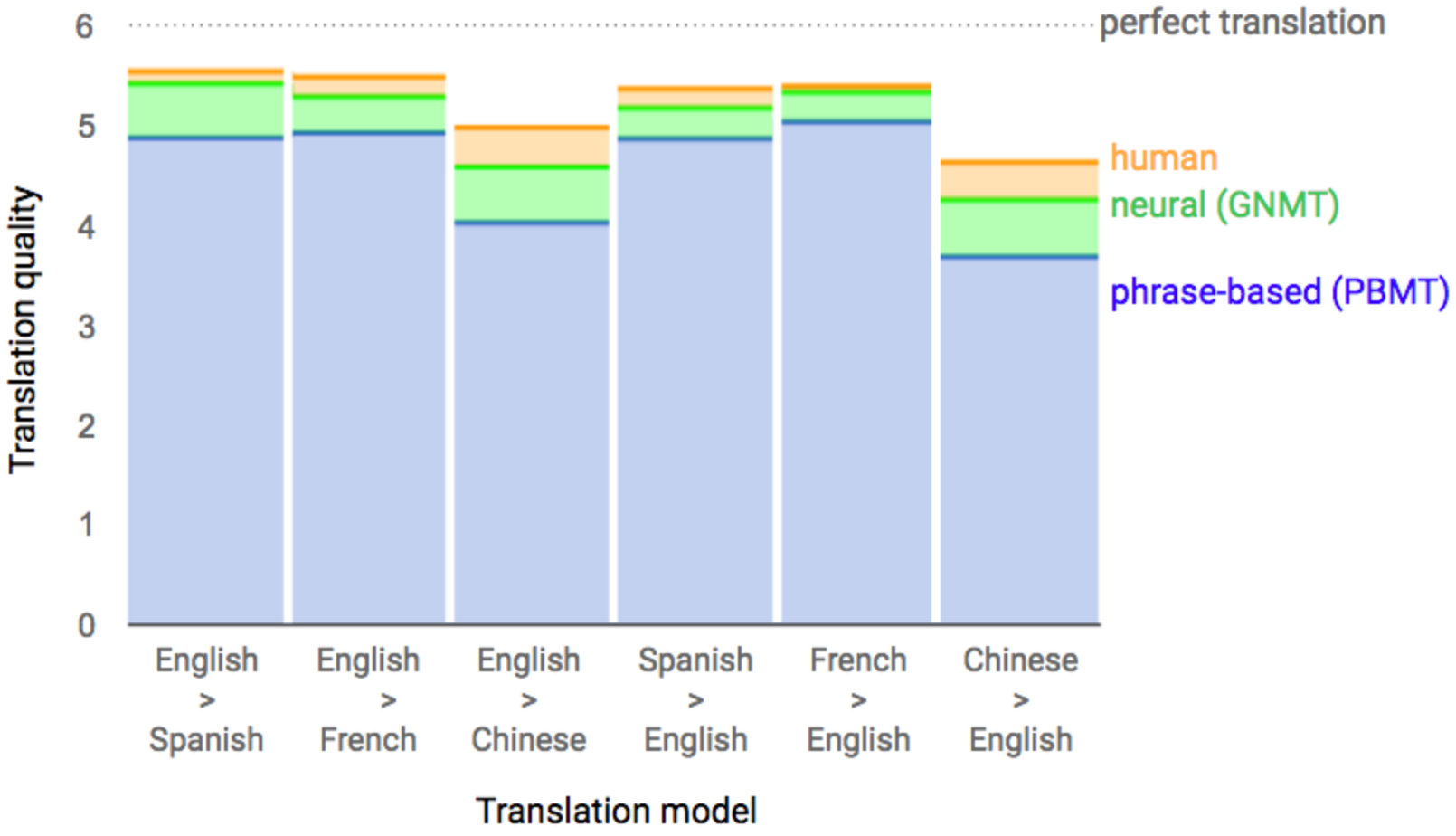


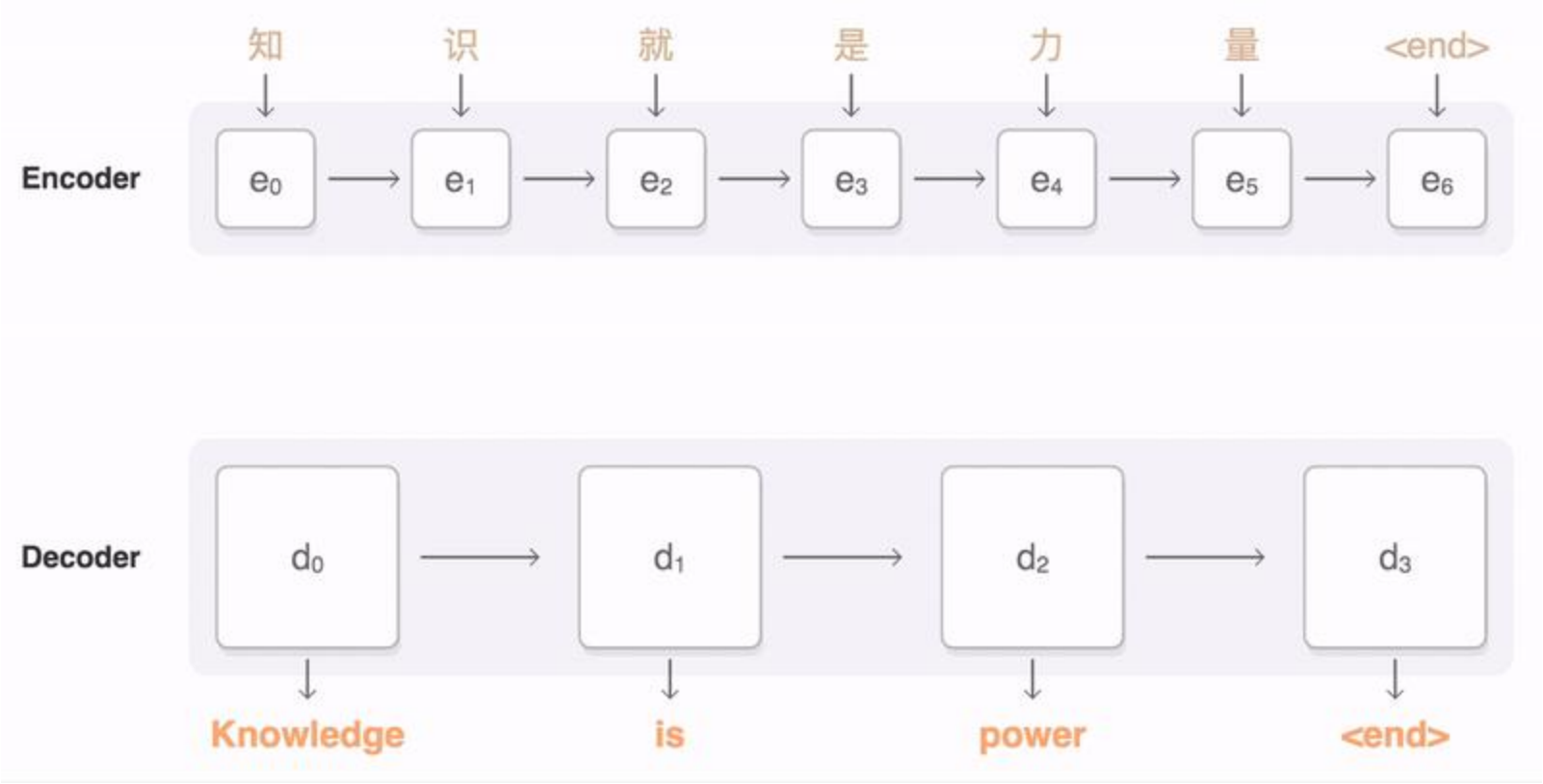
Figure 1: The model architecture of GNMT, Google's Neural Machine Translation system. On the left is the encoder network, on the right is the decoder network, in the middle is the attention module. The bottom encoder layer is bi-directional: the pink nodes gather information from left to right while the green nodes gather information from right to left. The other layers of the encoder are uni-directional. Residual connections start from the layer third from the bottom in the encoder and decoder. The model is partitioned into multiple GPUs to speed up training. In our setup, we have 8 encoder LSTM layers (1 bi-directional layer and 7 uni-directional layers), and 8 decoder layers. With this setting, one model replica is partitioned 8-ways and is placed on 8 different GPUs typically belonging to one host machine. During training, the bottom bi-directional encoder layers compute in parallel first. Once both finish, the uni-directional encoder layers can start computing, each on a separate GPU. To retain as much parallelism as possible during running the decoder layers, we use the bottom decoder layer output only for obtaining recurrent attention context, which is sent directly to all the remaining decoder layers. The softmax layer is also partitioned and placed on multiple GPUs. Depending on the output vocabulary size we either have them run on the same GPUs as the encoder and decoder networks, or have them run on a separate set of dedicated GPUs.

Google Neural Machine Translation (Wu et al. (2016))



https://ai.googleblog.com/2016/09/a-neural-network-for-machine.html

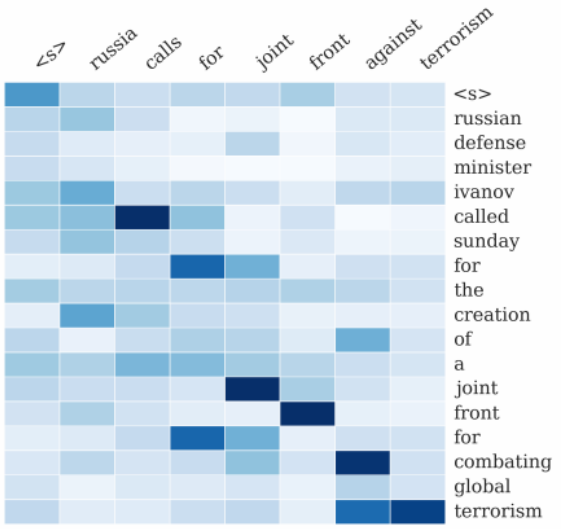
Google Neural Machine Translation (Wu et al. (2016))



https://ai.googleblog.com/2016/09/a-neural-network-for-machine.html

Summary

- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on
  - We get alignment for free!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself



Sequence-to-sequence is versatile!

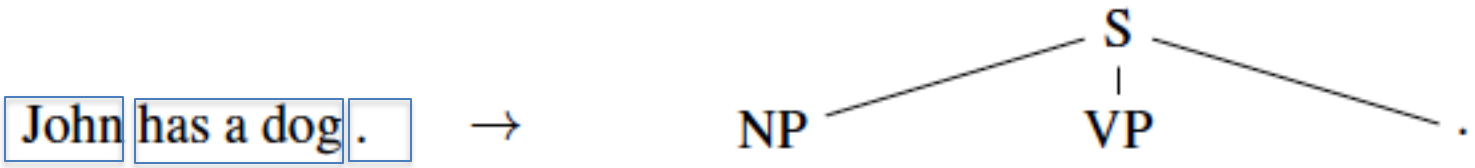
- Sequence-to-sequence is useful for *more than just MT*
- Many NLP tasks can be phrased as sequence-to-sequence:
  - Summarization (long text → short text)
  - Dialogue (previous utterances → next utterance)
  - Parsing (input text → output parse as sequence)
  - Code generation (natural language → Python code)

Grammar as a Foreign Language

- Vinyals et al., 2015

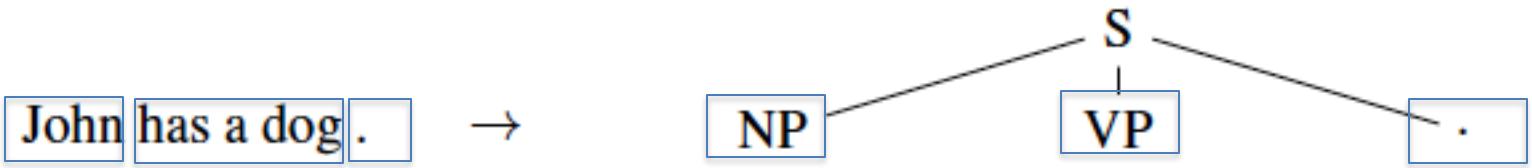
Grammar as a Foreign Language

Parsing tree



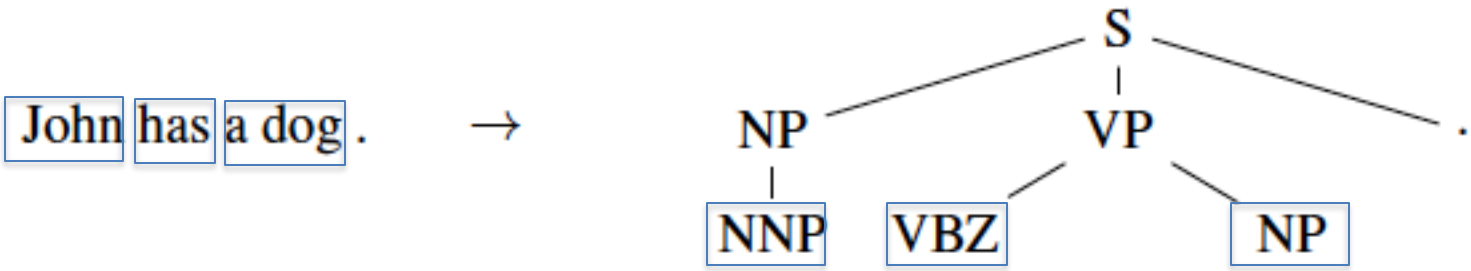
Grammar as a Foreign Language

Parsing tree



Grammar as a Foreign Language

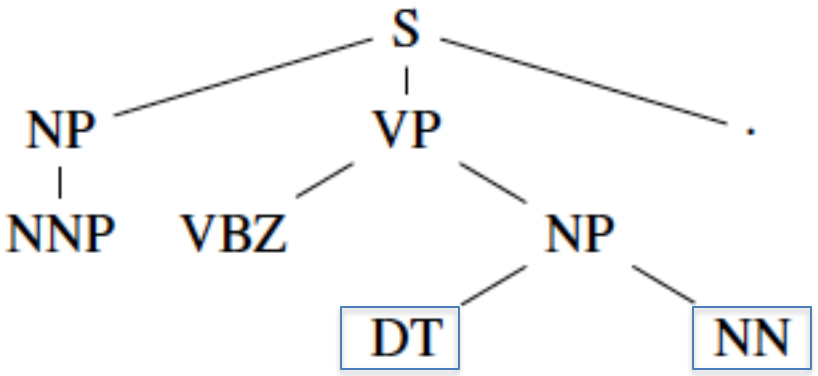
Parsing tree



Grammar as a Foreign Language

Parsing tree

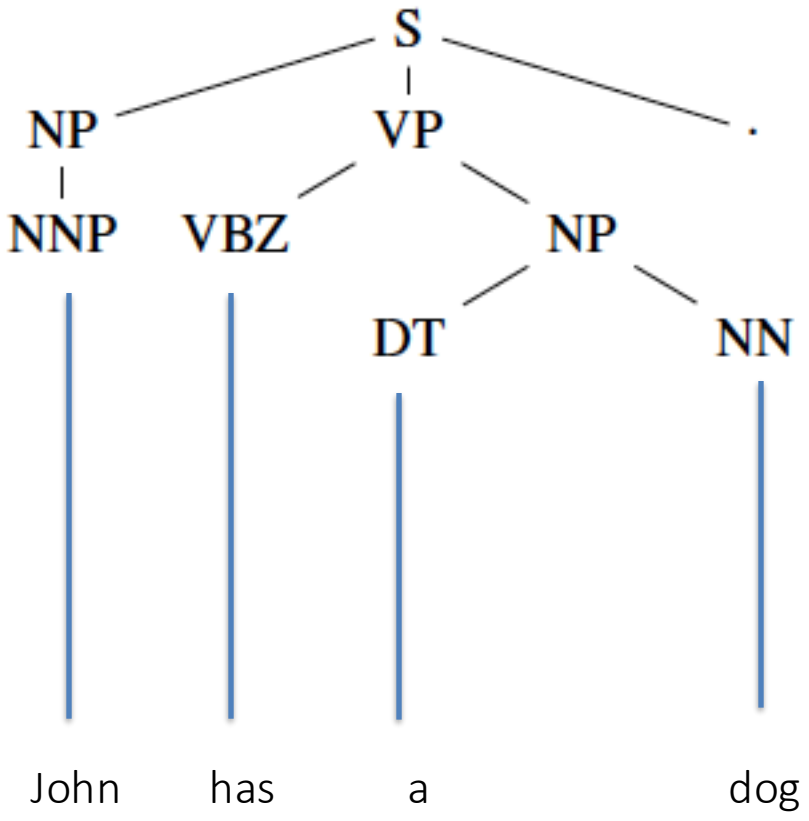
John has a dog . →



Grammar as a Foreign Language

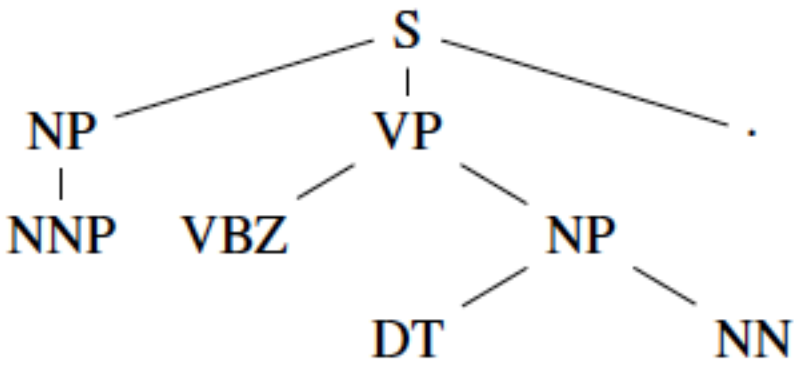
Parsing tree

John has a dog . →



Grammar as a Foreign Language

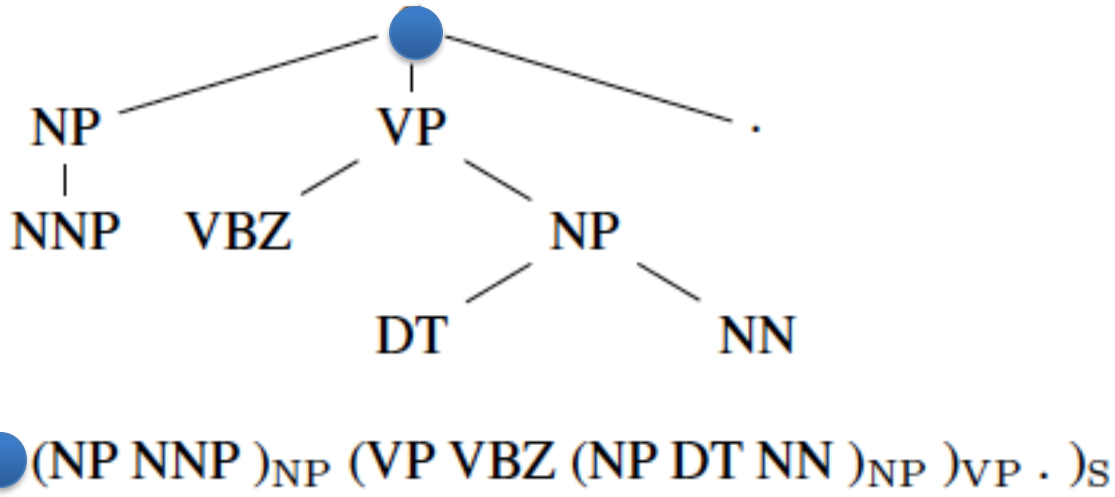
Converting tree to sequence



$$(\text{S (NP NNP)}_{\text{NP}} \text{ (VP VBZ (NP DT NN)}_{\text{NP}} \text{)}_{\text{VP}} \text{ .)}_{\text{S}}$$

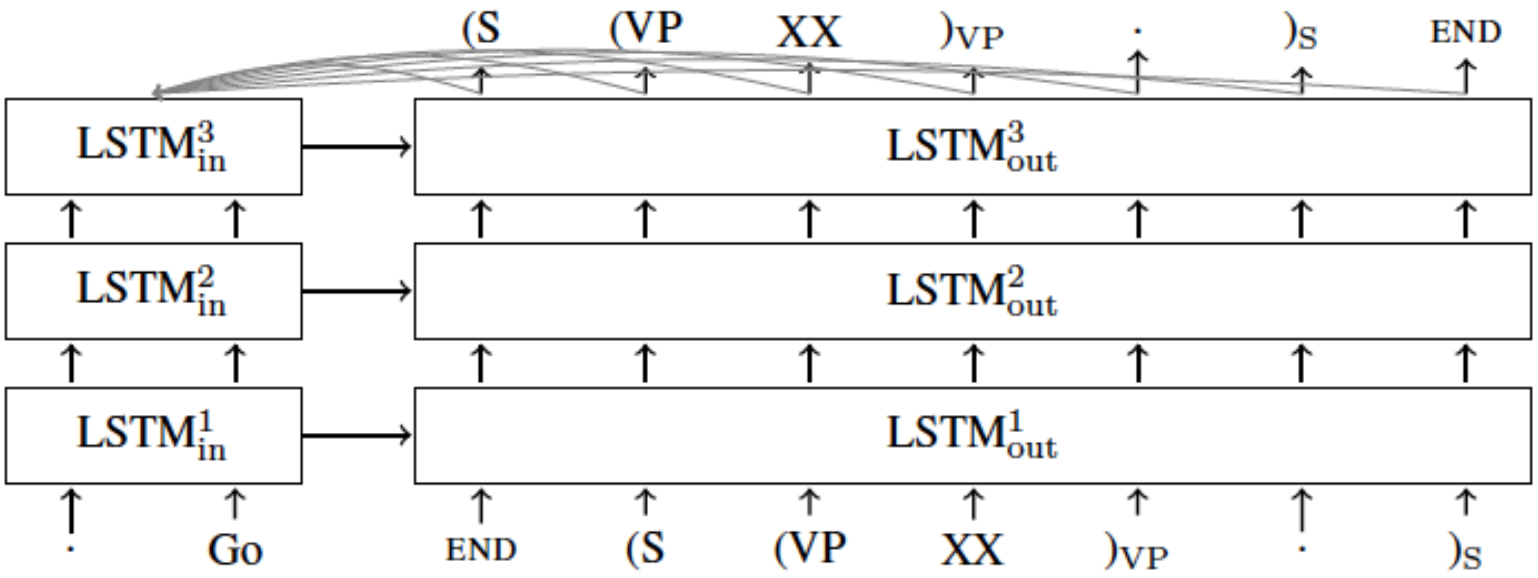
Grammar as a Foreign Language

Converting tree to sequence



Grammar as a Foreign Language

Model



Grammar as a Foreign Language

Results

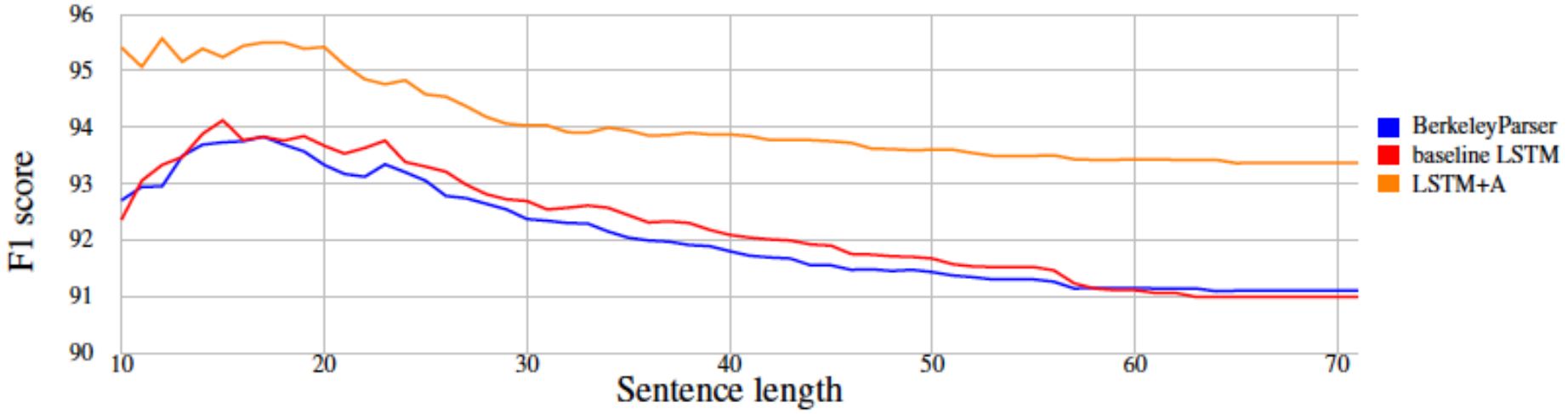


Image Captioning

- Neural image captioning (Donahue et al. (2014); Vinyals et al. (2015))
  - Encode image
  - Decode text

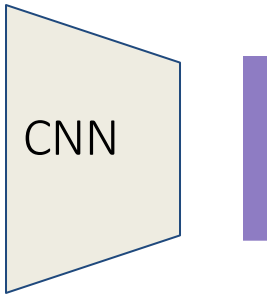
a cat is sitting on a toilet seat
logprob: -7.79

a display case filled with lots of different types of donuts
logprob: -7.78

a group of people sitting at a table with wine glasses
logprob: -6.71

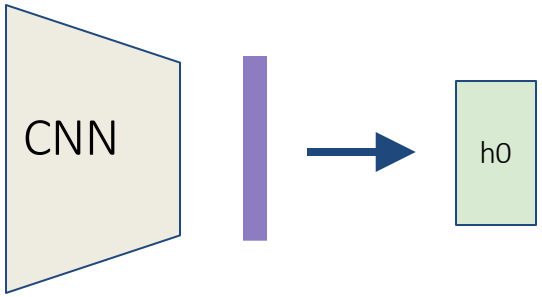
Image:
H x W x 3

Image:
H x W x 3



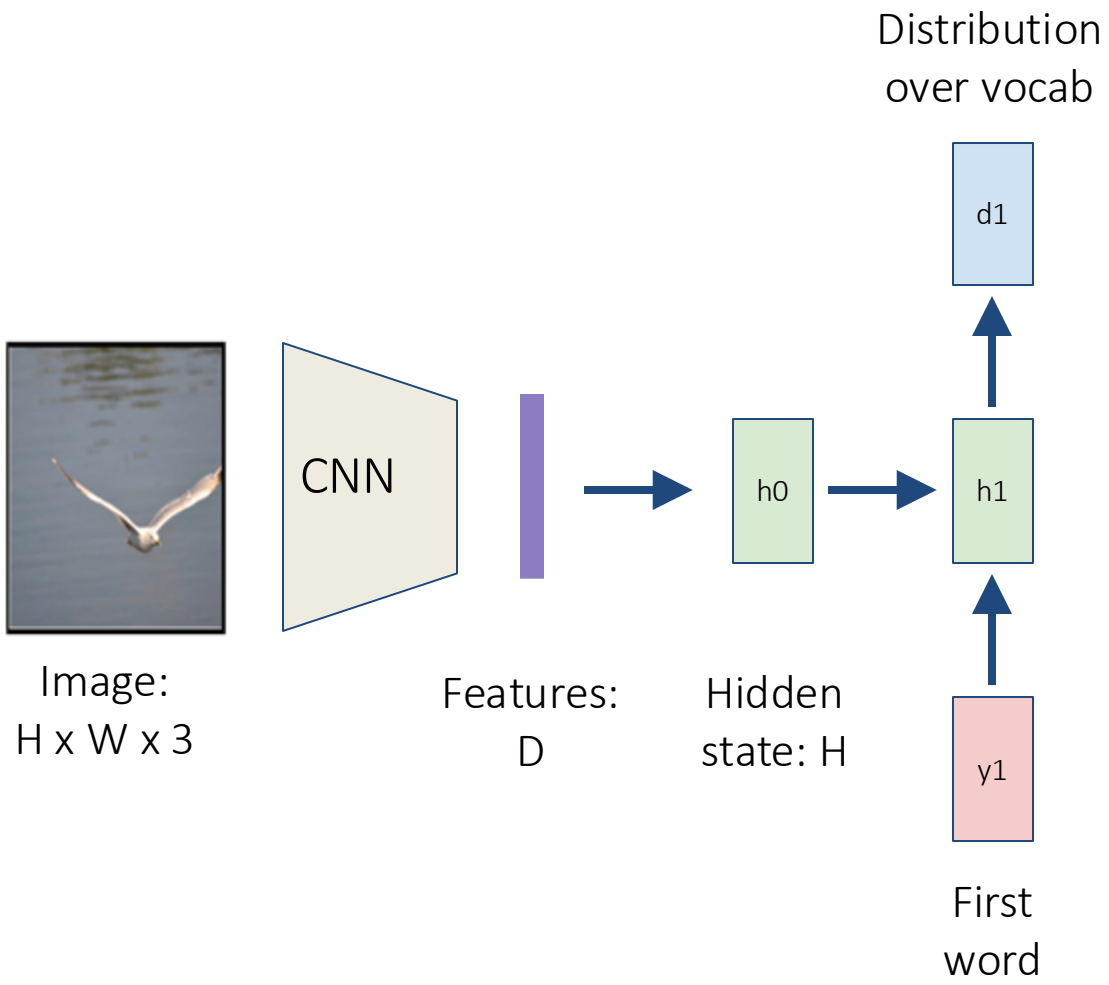
Features:
D

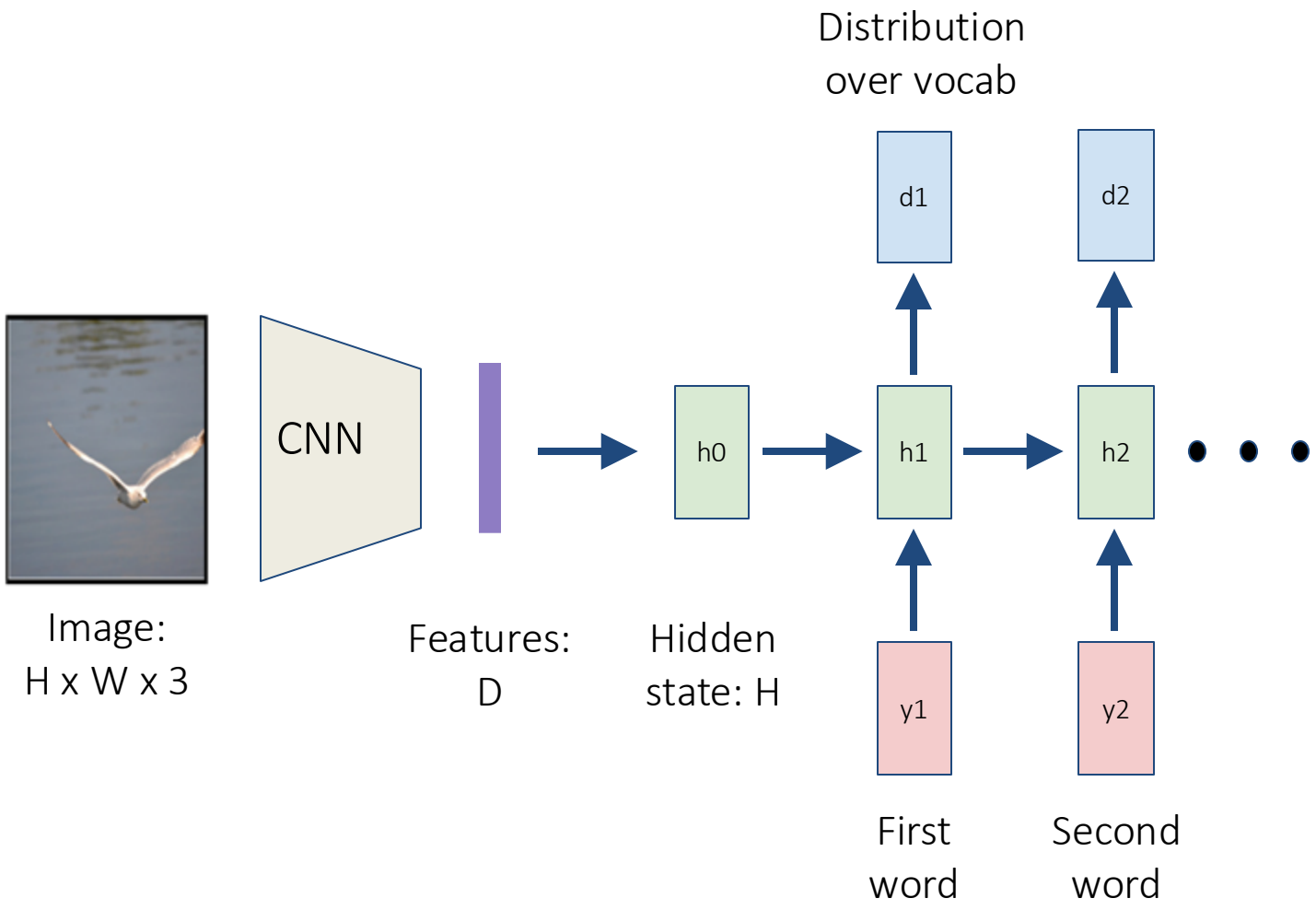
Image:
H x W x 3

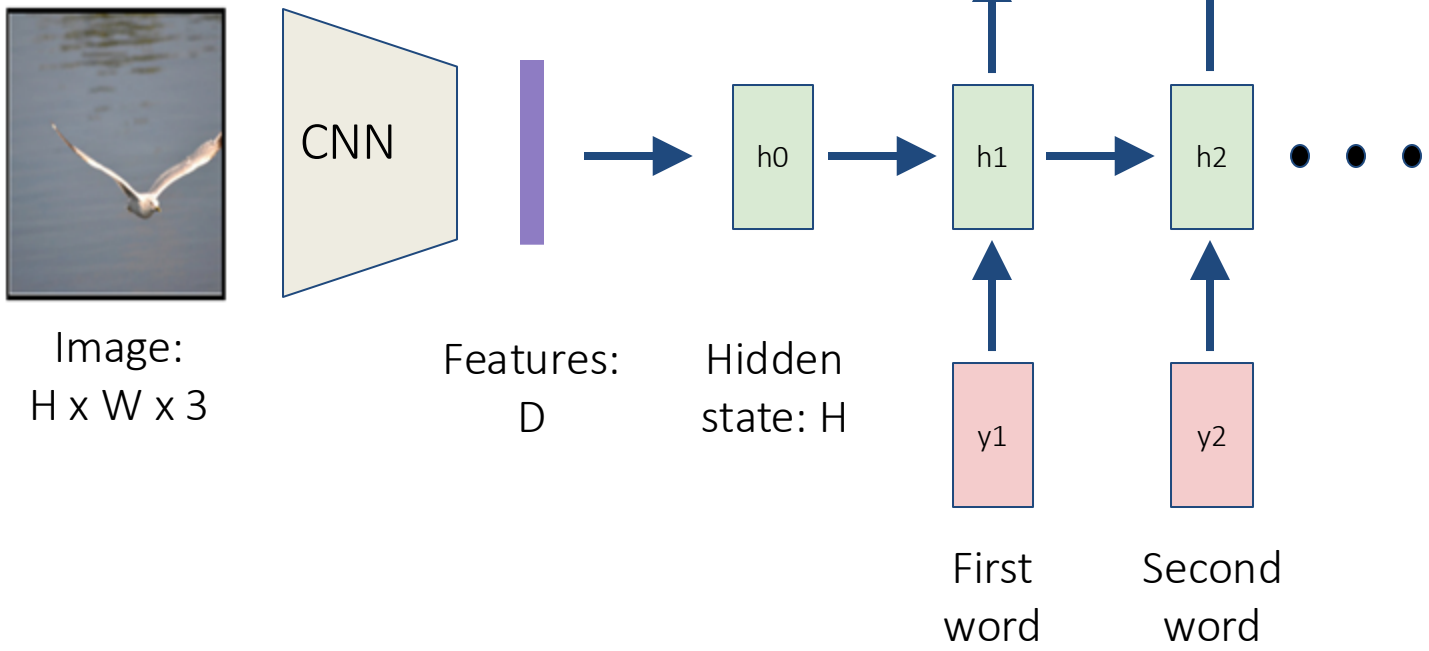


Features:
D

Hidden
state: H


Distribution over vocab
d1
CNN
h0
h1
y1
Image: H x W x 3
Features: D
Hidden state: H
First word


Distribution
over vocab
d1
d2
CNN
h0
h1
h2
y1
y2
Image:
H x W x 3
Features:
D
Hidden
state: H
First
word
Second
word


Distribution over vocab
RNN only looks at whole image, once
d1
d2
CNN
h0
h1
h2
y1
y2
Image: H x W x 3
Features: D
Hidden state: H
First word
Second word

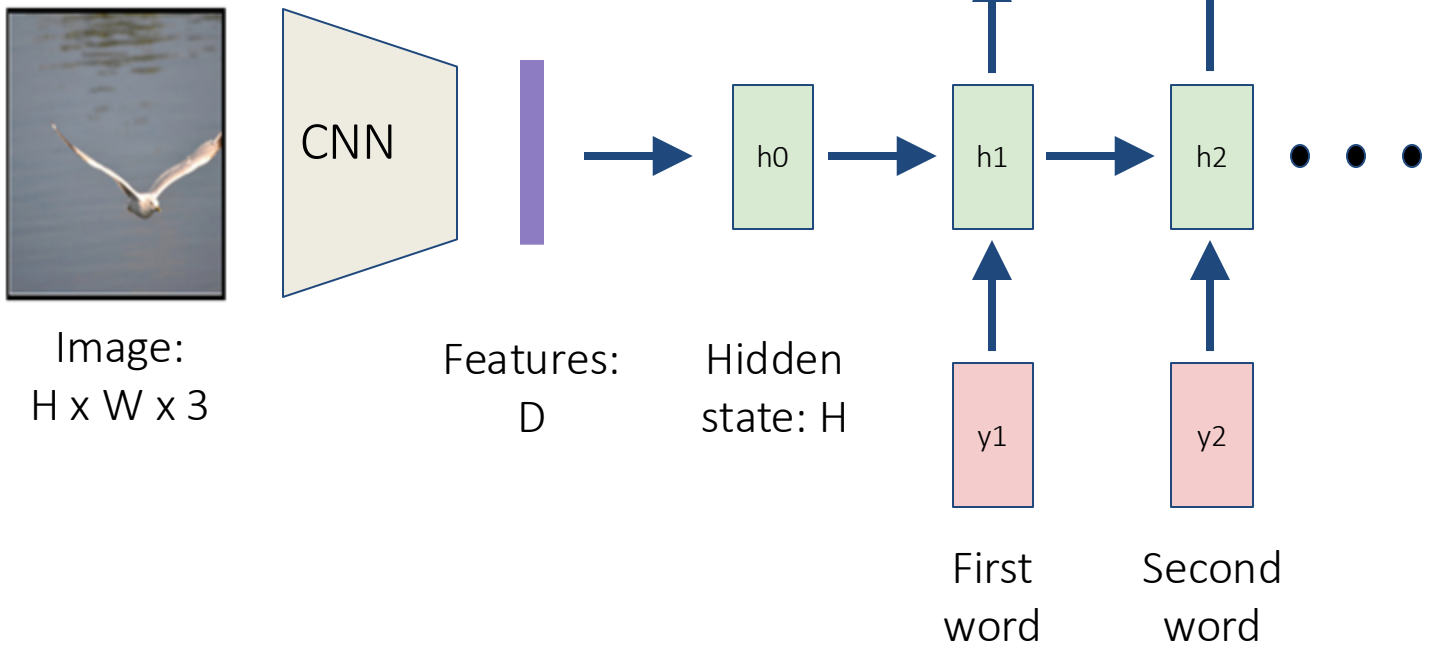
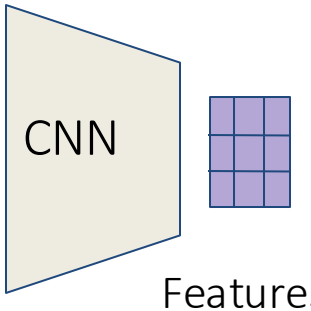
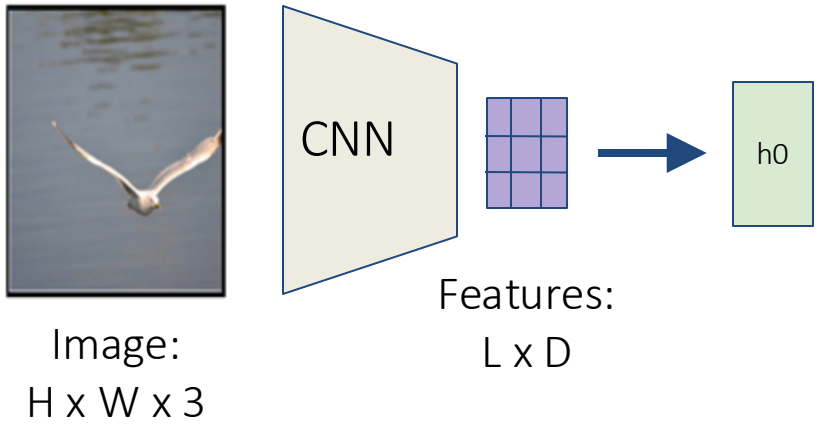

Distribution over vocab
RNN only looks at whole image, once
d1
d2
CNN
h0
h1
h2
Image: H x W x 3
Features: D
Hidden state: H
y1
y2
First word
Second word
What if the RNN looks at different parts of the image at each timestep?

Image:
H x W x 3

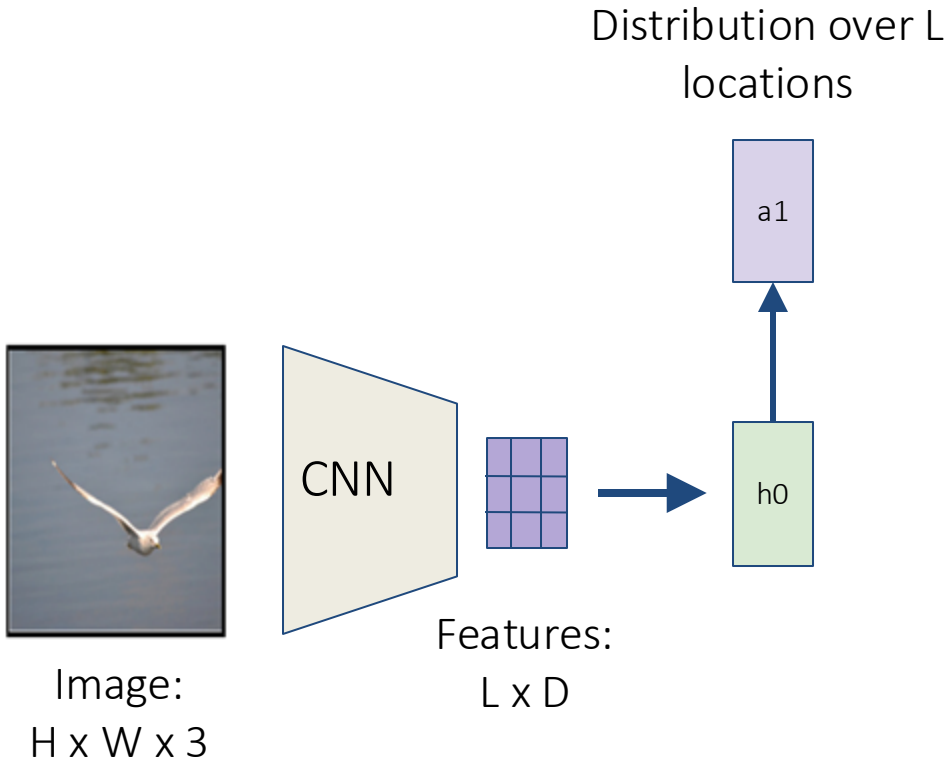


Features:
L x D

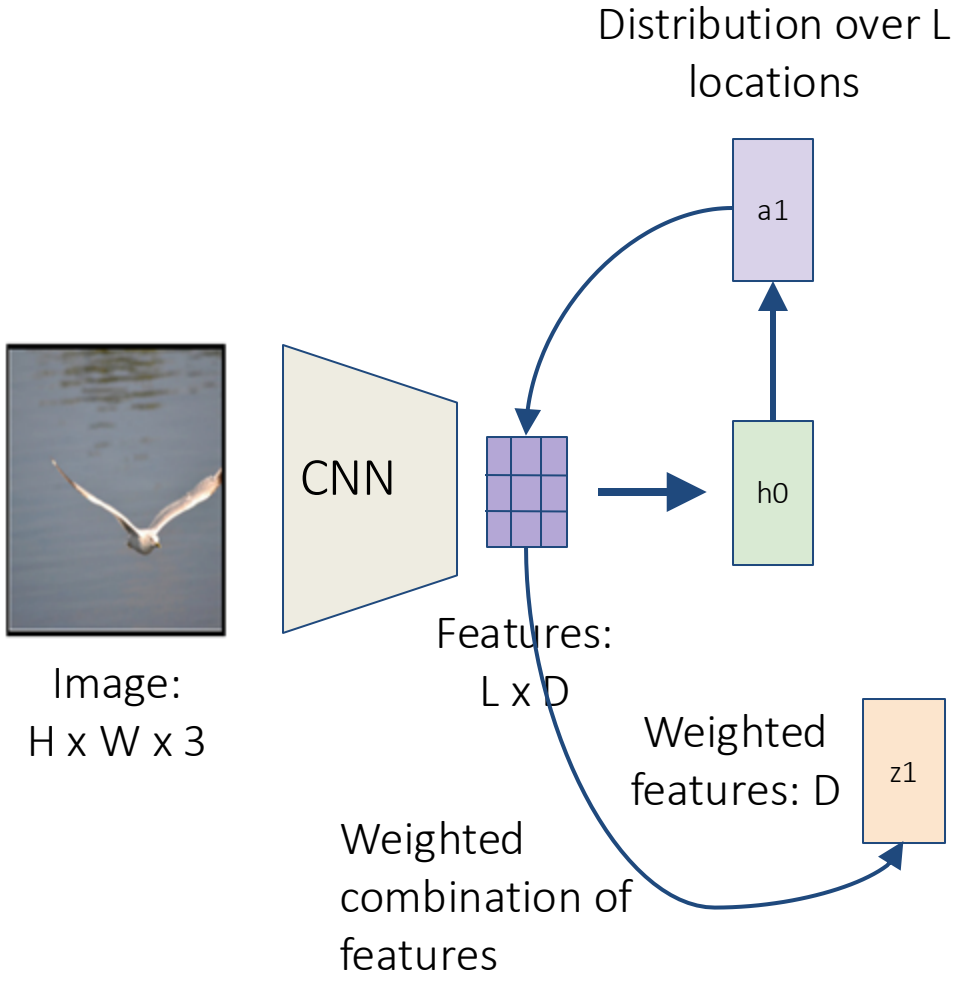
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

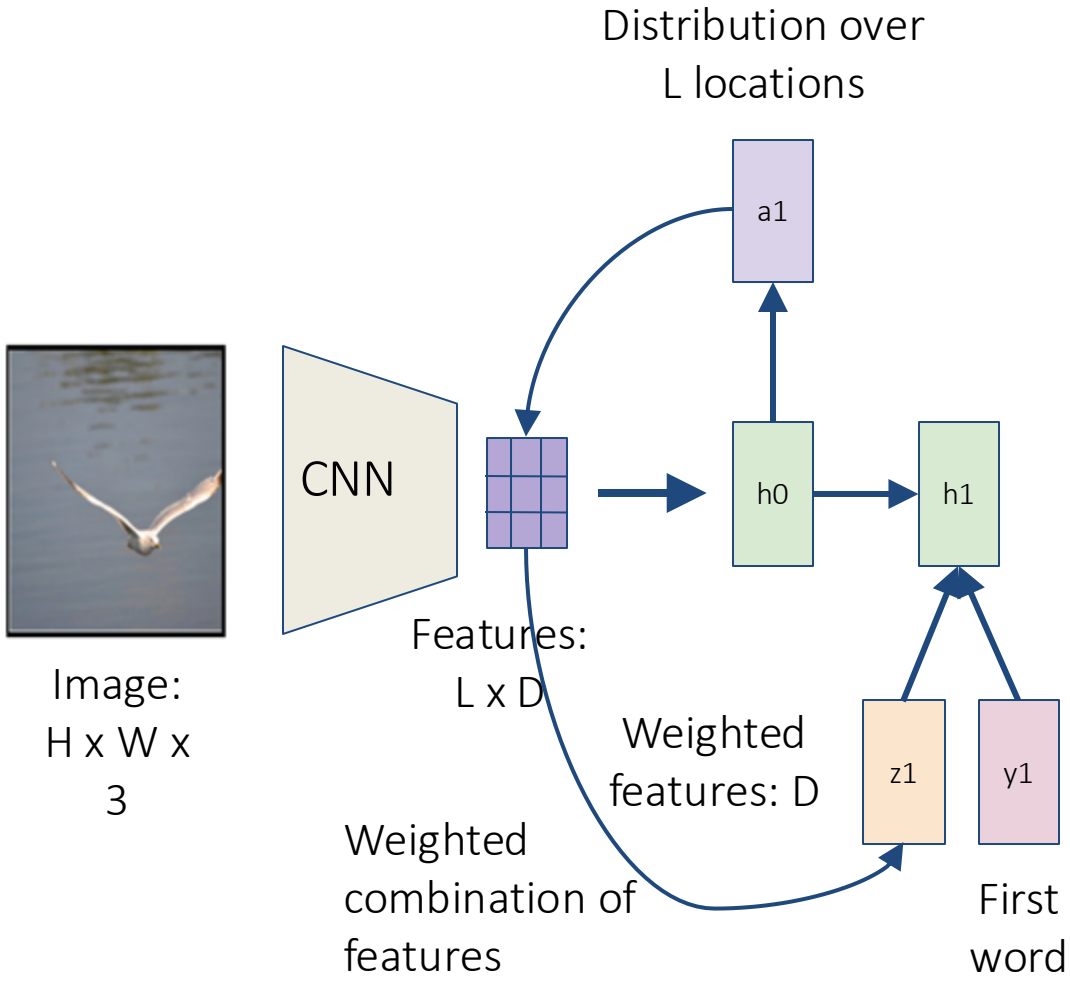


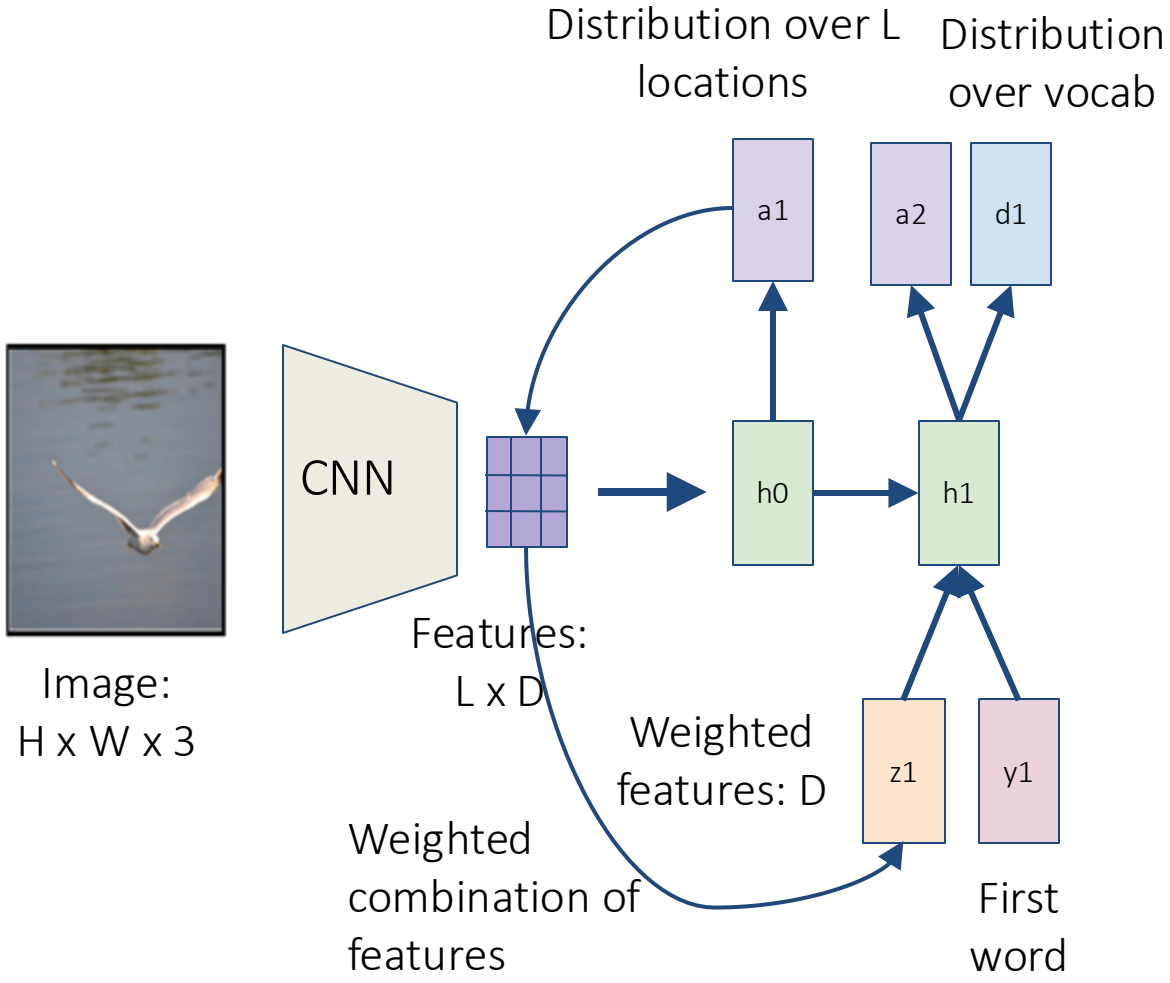
Xu et al, “Show, Attend and Tell:
Neural Image Caption Generation with
Visual Attention”, ICML 2015

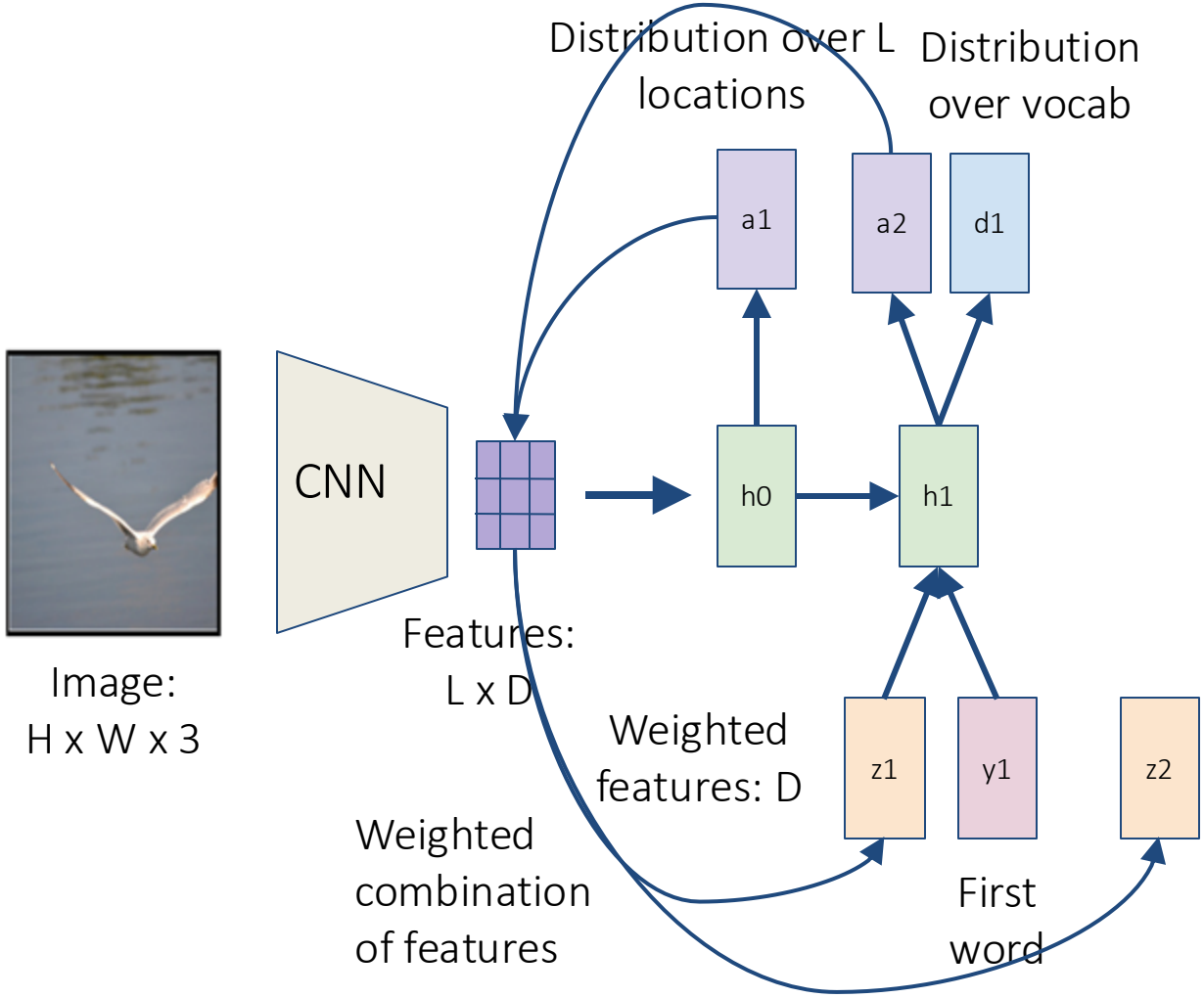


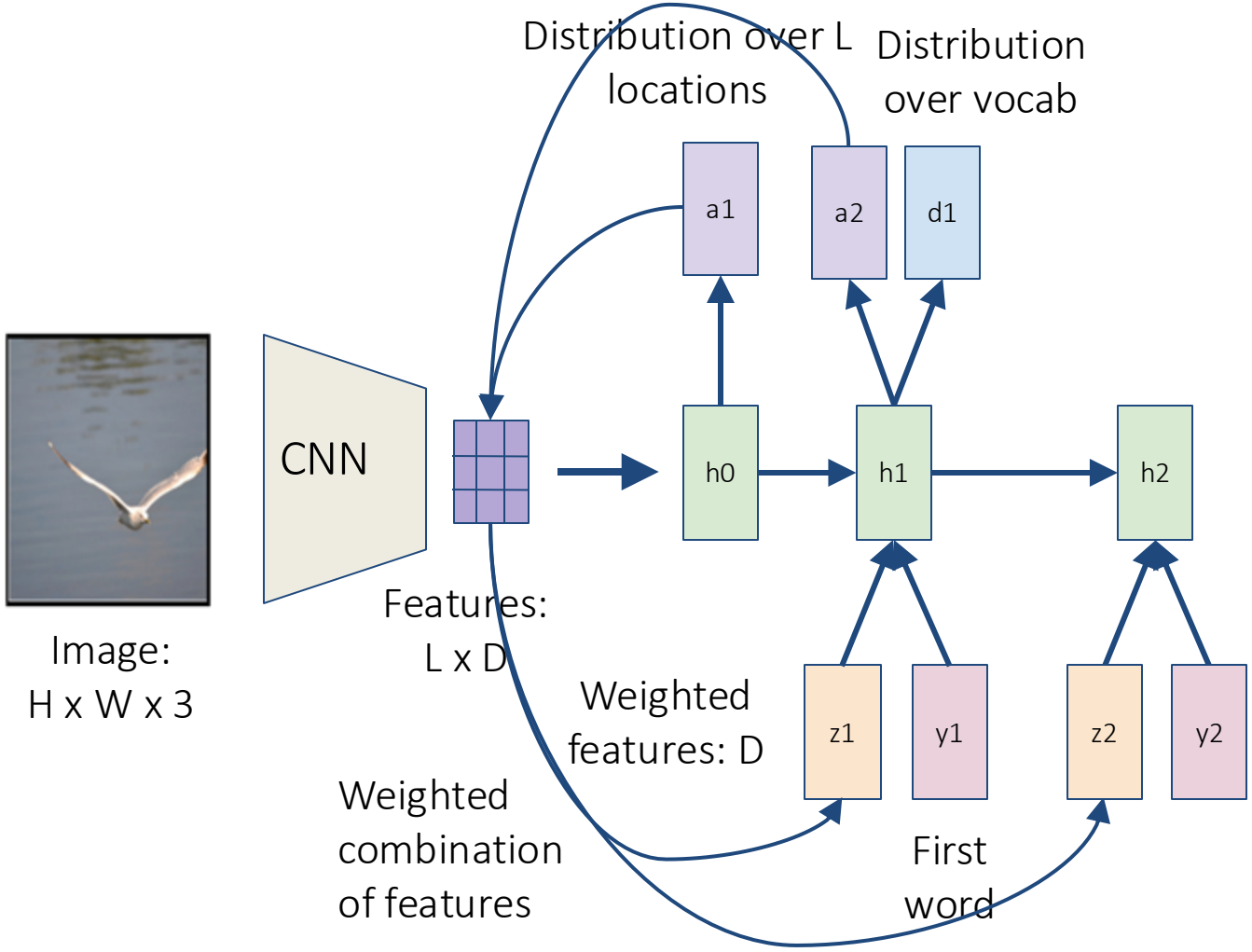
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015


Distribution over L locations
a1
CNN
h0
Image:
H x W x 3
Features:
L x D
Weighted features: D
z1
Weighted combination of features


Distribution over
L locations
a1
CNN
h0
h1
Image:
H x W x
3
Features:
L x D
Weighted
features: D
z1
y1
Weighted
combination of
features
First
word


Distribution over L locations
Distribution over vocab
a1
a2
d1
CNN
h0
h1
Image: H x W x 3
Features: L x D
Weighted features: D
z1
y1
Weighted combination of features
First word


Distribution over L locations
Distribution over vocab
a1
a2
d1
CNN
h0
h1
Image: H x W x 3
Features: L x D
Weighted features: D
z1
y1
z2
Weighted combination of features
First word


Distribution over L locations
Distribution over vocab
a1
a2
d1
CNN
h0
h1
h2
Features: L x D
Image: H x W x 3
Weighted features: D
z1
y1
z2
y2
Weighted combination of features
First word

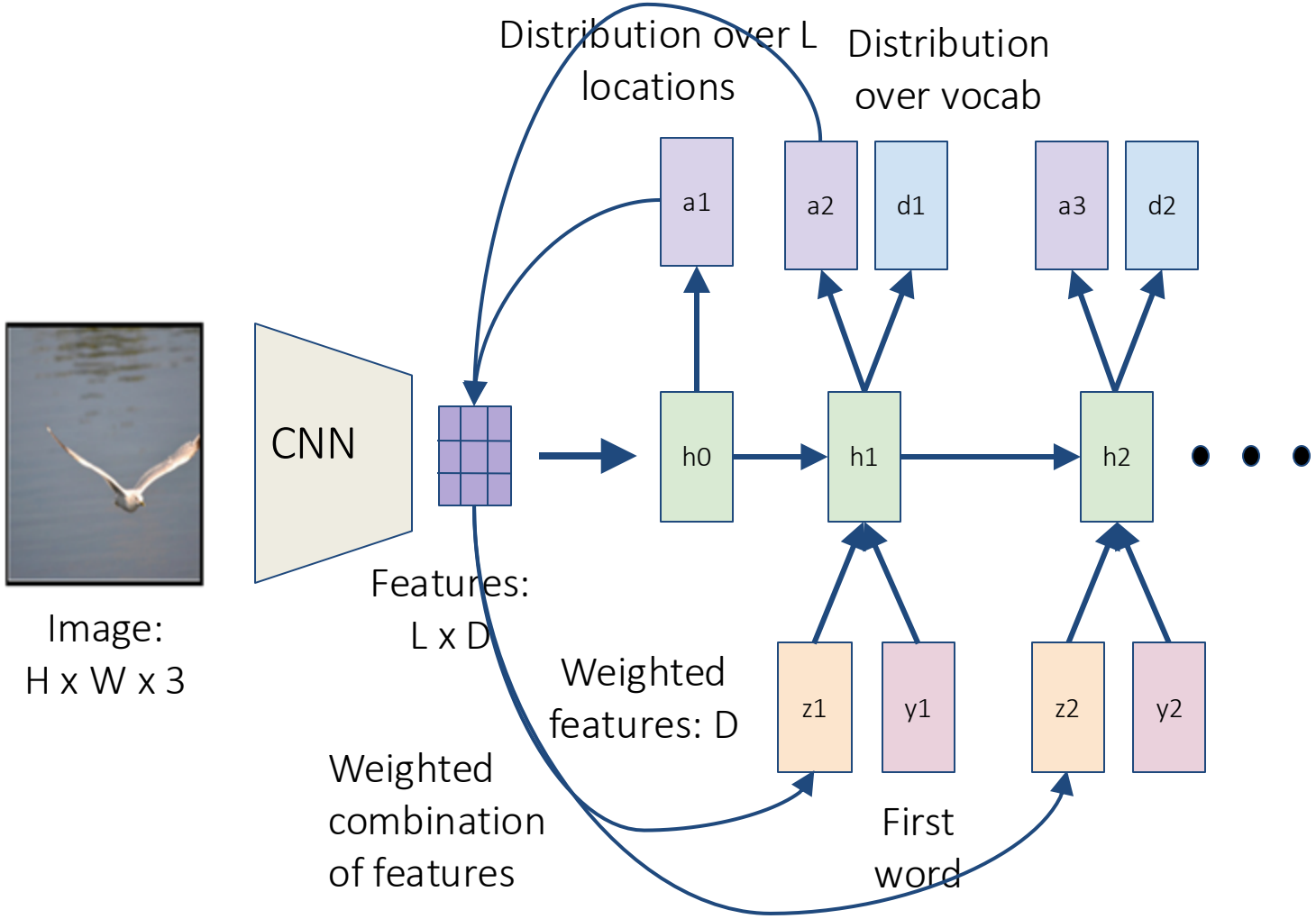

Distribution over L locations
Distribution over vocab
a1
a2
d1
a3
d2
CNN
h0
h1
h2
Image: H x W x 3
Features: L x D
Weighted features: D
z1
y1
z2
y2
Weighted combination of features
First word

Image:
H x W x 3

CNN

a	b
c	d

Grid of features
(Each D-dimensional)

From
RNN:

p_a	p_b
p_c	p_d

Distribution over grid locations

$p_a + p_b + p_c + p_c = 1$

Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image:
H x W x 3

CNN

a	b
c	d

Grid of features
(Each D-dimensional)

Context vector z
(D-dimensional)

From
RNN:

p_a	p_b
p_c	p_d

Distribution over grid locations

$p_a + p_b + p_c + p_c = 1$

Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image:
H x W x 3

CNN

a	b
c	d

Grid of features
(Each D-dimensional)

From
RNN:

p_a	p_b
p_c	p_d

Distribution over grid locations

$p_a + p_b + p_c + p_c = 1$

Context vector z
(D-dimensional)

Soft attention:
Summarize ALL locations
$z = p_a a + p_b b + p_c c + p_d d$

Derivative dz/dp is nice!
Train with gradient descent

Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image:
H x W x 3

CNN

a	b
c	d

Grid of features
(Each D-dimensional)

From
RNN:

p_a	p_b
p_c	p_d

Distribution over grid locations

$p_a + p_b + p_c + p_c = 1$

Context vector z
(D-dimensional)

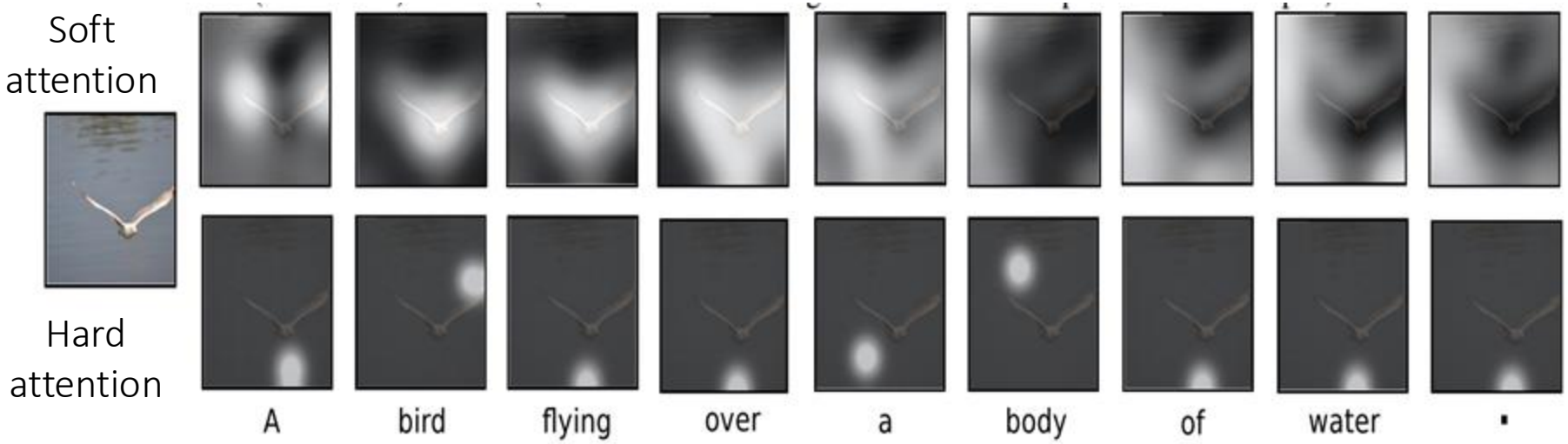
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Soft attention:
Summarize ALL locations
$z = p_a a + p_b b + p_c c + p_d d$

Derivative dz/dp is nice!
Train with gradient descent

Hard attention:
Sample ONE location according to p, z = that vector

With argmax, dz/dp is zero almost everywhere …
Can't use gradient descent; need reinforcement learning


Soft attention
Hard attention
A
bird
flying
over
a
body
of
water
.

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.

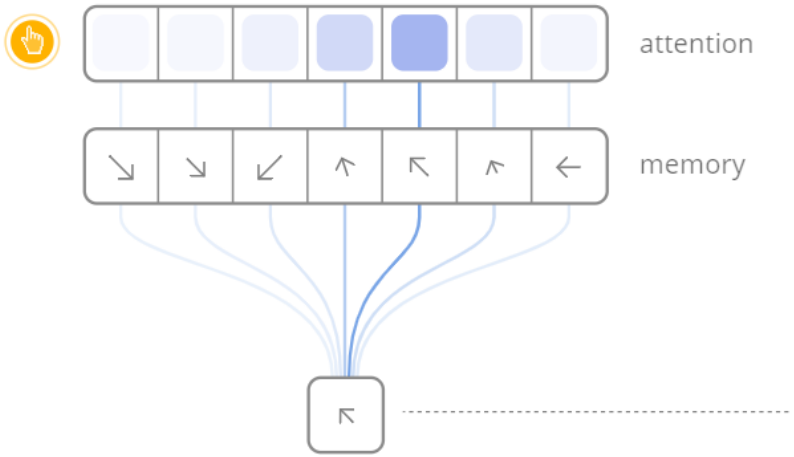
A giraffe standing in a forest with trees in the background.

What did we learn today?

- Attention can be quite useful in a variety of applications
- Attention = a form of memory access and can help with capturing long-term dependencies!

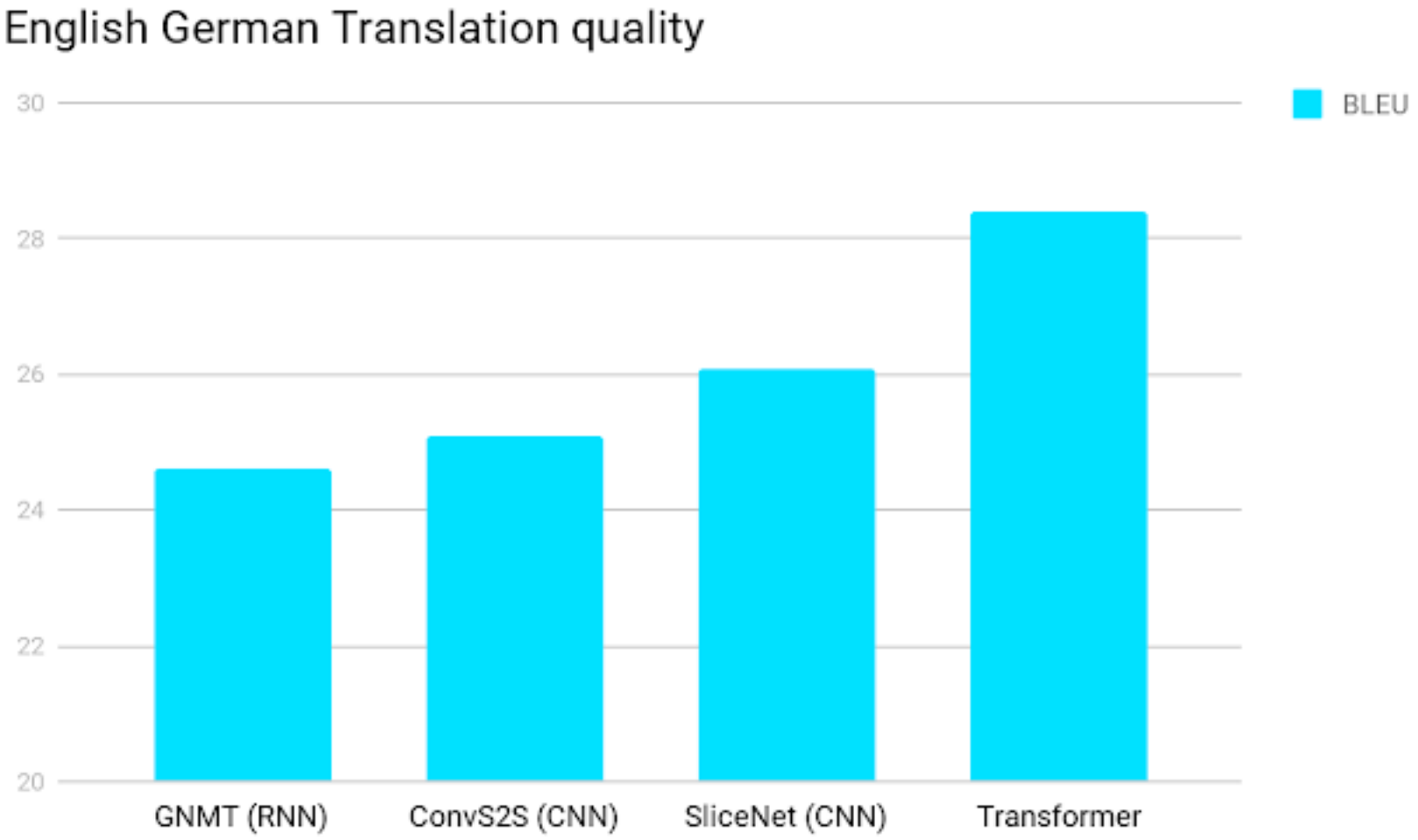
Attention is a *general* deep Learning technique

- More general definition of attention
  - Given a set of vector *values*, and a vector *query*, **attention** is a technique to compute a weighted sum of the values, dependent on the query.
  - We sometimes say that the query *attends to* the values
  - For example, in the seq2seq + attention model, each decoder hidden state *attends to* the encoder hidden states



https://distill.pub/2016/augmented-rnns/

Attention is all you need? – Transformer Networks



https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

Problems with RNNs = Motivation for Transformers

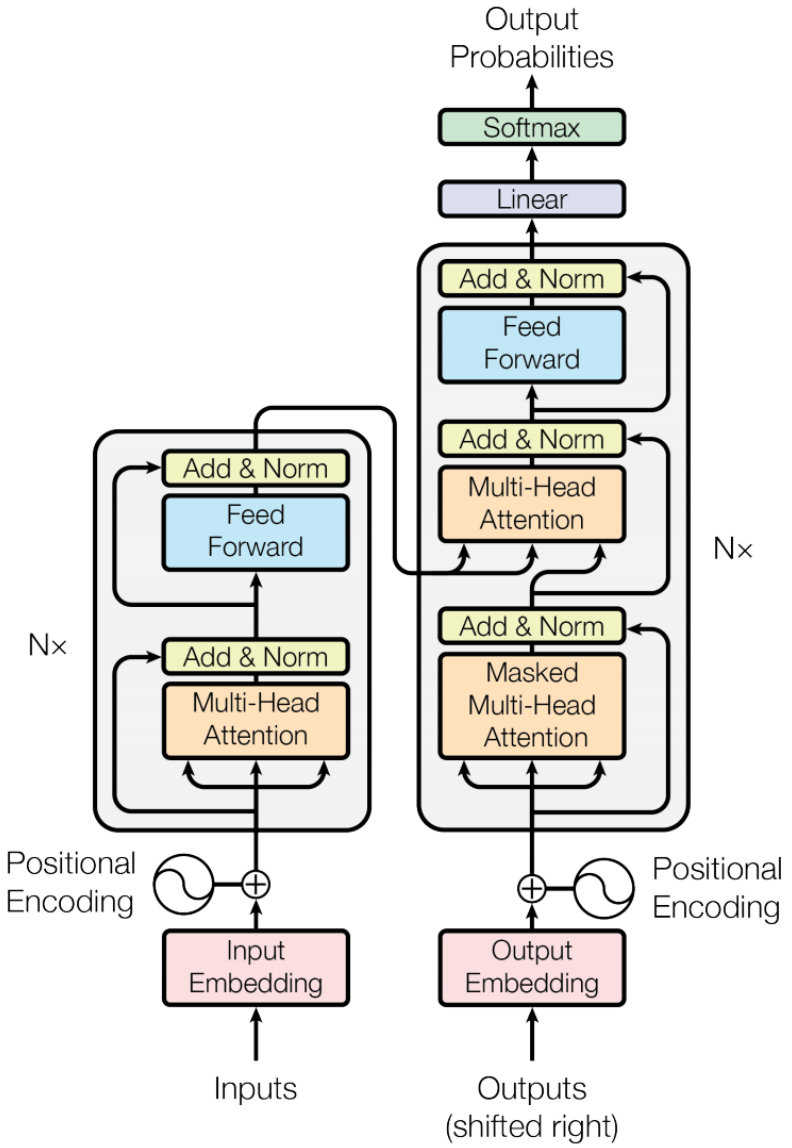
- Recurrent models typically factor computation along the symbol positions of the input and output sequences
  - Sequential computation prevents parallelization
  - Critical at longer sequence lengths, as memory constraints limit batching across examples

- Despite advanced RNNs like LSTMs, RNNs still need attention mechanism to deal with long range dependencies – path length for codependent computation between states grows with sequence

- But if attention gives us access to any state… maybe we don't need the RNN?

Transformer Overview

- Sequence-to-sequence
- Encoder-Decoder
- Task: machine translation with parallel corpus
- Predict each translated word
- Final cost/error function is standard cross-entropy error on top of a softmax classifier



https://arxiv.org/pdf/1706.03762.pdf

Transformer Basics

- Let’s define the basic building blocks of transformer networks first: new attention layers!

Dot-Product Attention (Extending our previous def.)

- Inputs: a query q and a set of key-value (k-v) pairs to an output
  - Query, keys, values, and output are all vectors
- Output is weighted sum of values, where
  - Weight of each value is computed by an inner product of query and corresponding key
  - Queries and keys have same dimensionality

$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$

Dot-Product Attention – Matrix notation

- When we have multiple queries q, we stack them in a matrix Q:

$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$

- Becomes: $A(Q, K, V) = softmax(QK^T)V$

$[|Q| \times d_k] \times [d_k \times |K|] \times [|K| \times d_v]$

softmax row-wise

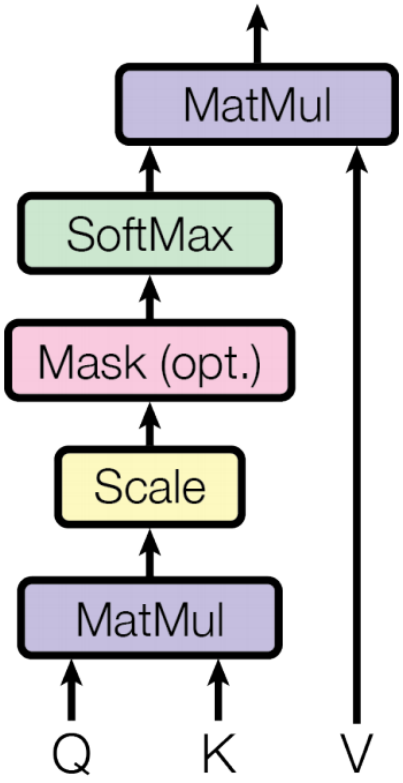
$= [|Q| \times d_v]$

Scaled Dot-Product Attention

- Problem: As d_k gets large, the variance of $q^T k$ increases → some values inside the softmax get large → the softmax gets very peaked → hence its gradient gets smaller.

- Solution: Scale by length of query/key vectors:

$$A(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

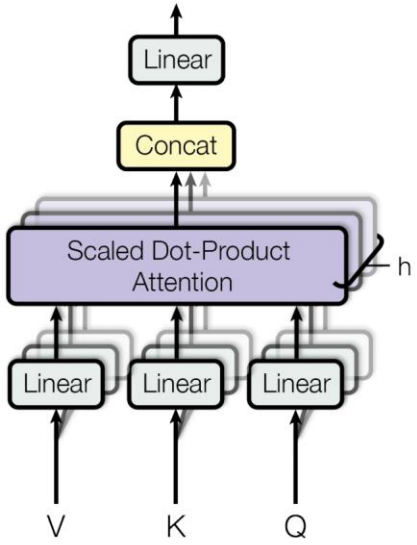


Self-attention and Multi-head attention

- The input word vectors could be the queries, keys and values
  - In other words: the word vectors themselves select each other
  - Word vector stack = Q = K = V
- Problem: Only one way for words to interact with one-another
- Solution: Multi-head attention
  - First map Q, K, V into h many lower dimensional spaces via W matrices
  - Then apply attention, then concatenate outputs and pipe through linear layer

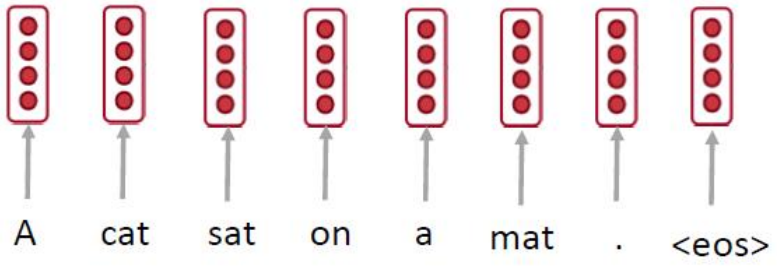
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

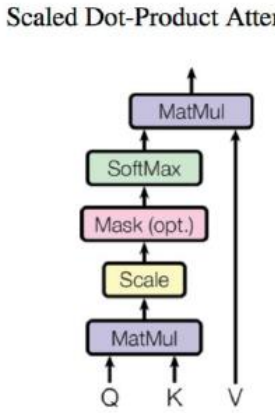
$$\textbf{where } \text{head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



Self-attention: A Running Example

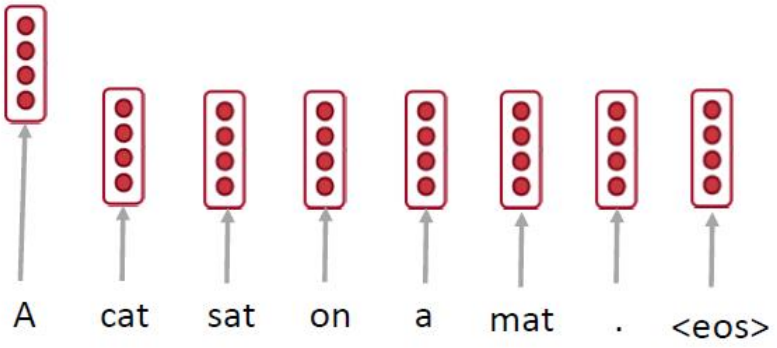
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

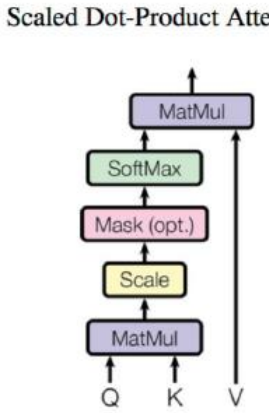




Self-attention: A Running Example

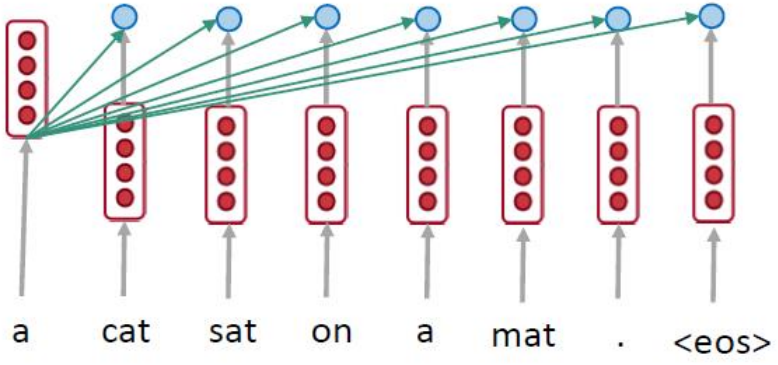
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

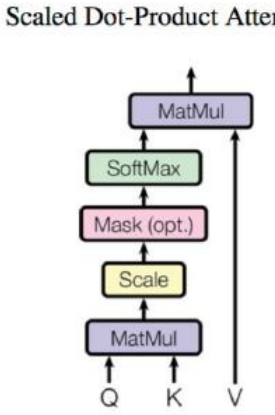




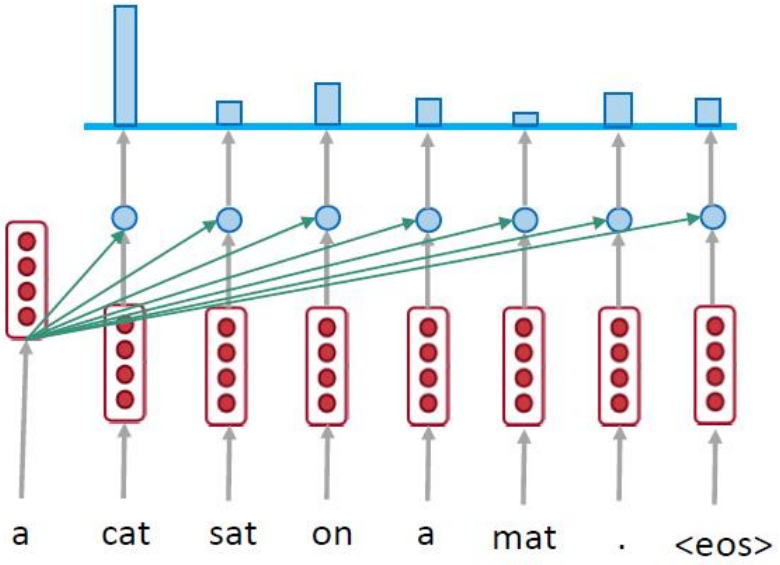
Self-attention: A Running Example

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

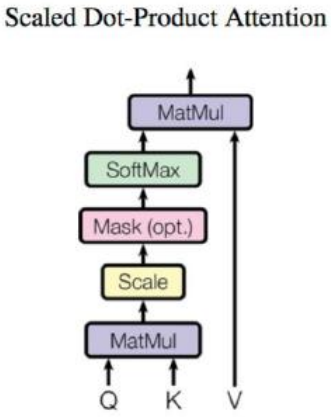




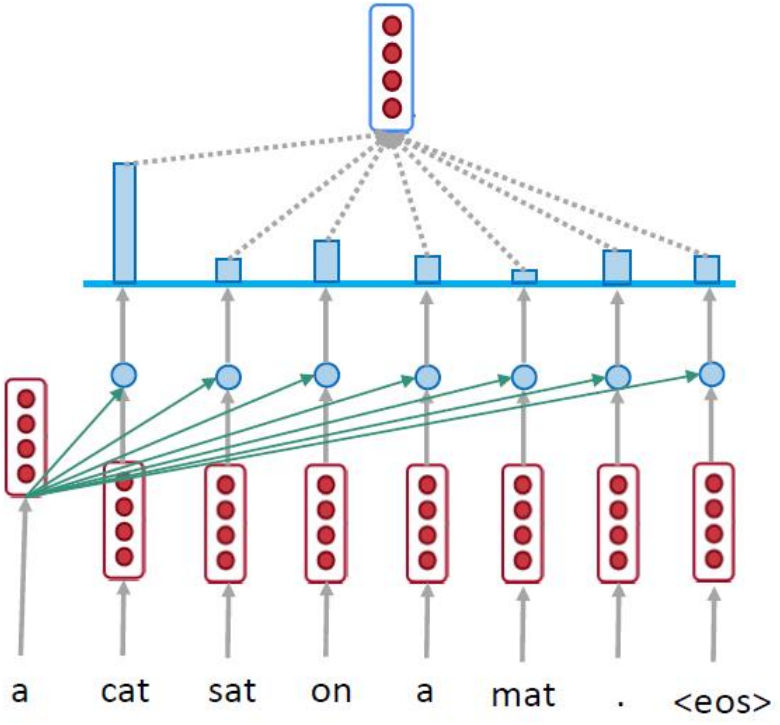
Self-attention: A Running Example



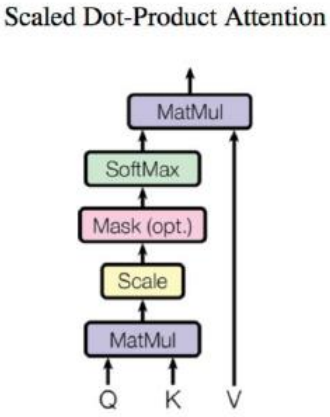
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



Self-attention: A Running Example

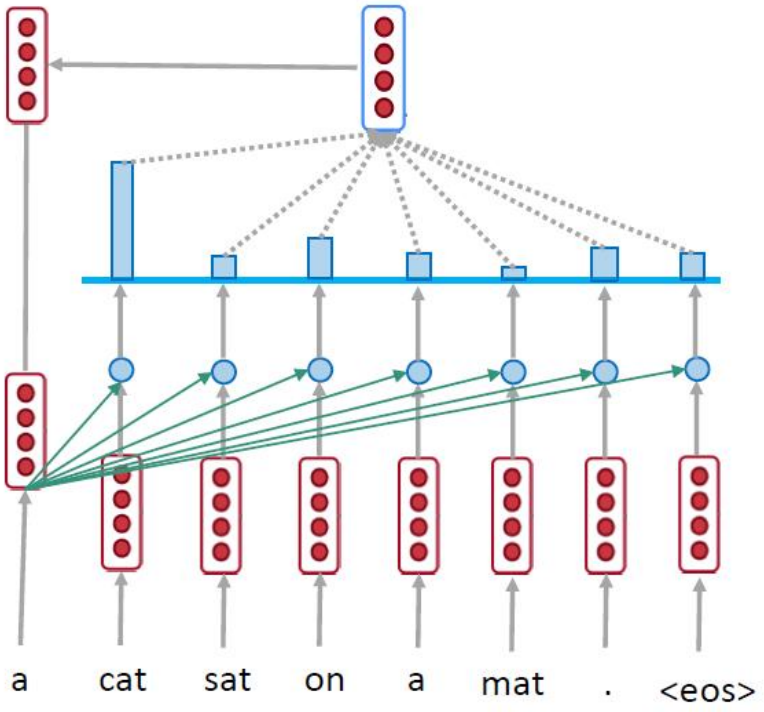


$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



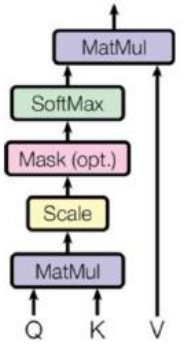
Self-attention: A Running Example

update representation for the word "a"



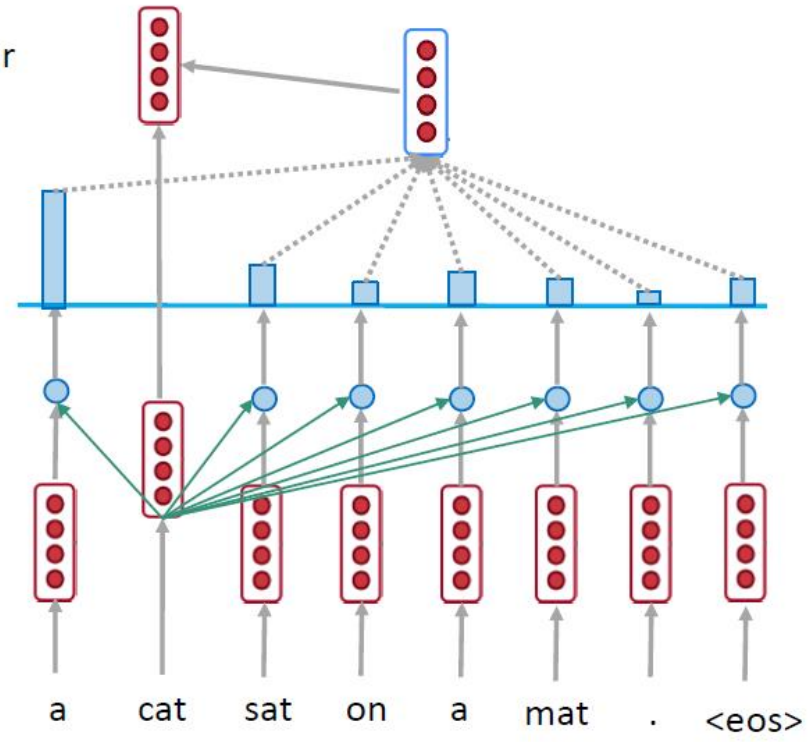
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Scaled Dot-Product Attention



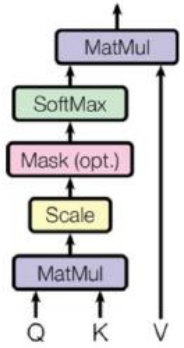
Self-attention: A Running Example

update representation for the word "cat"



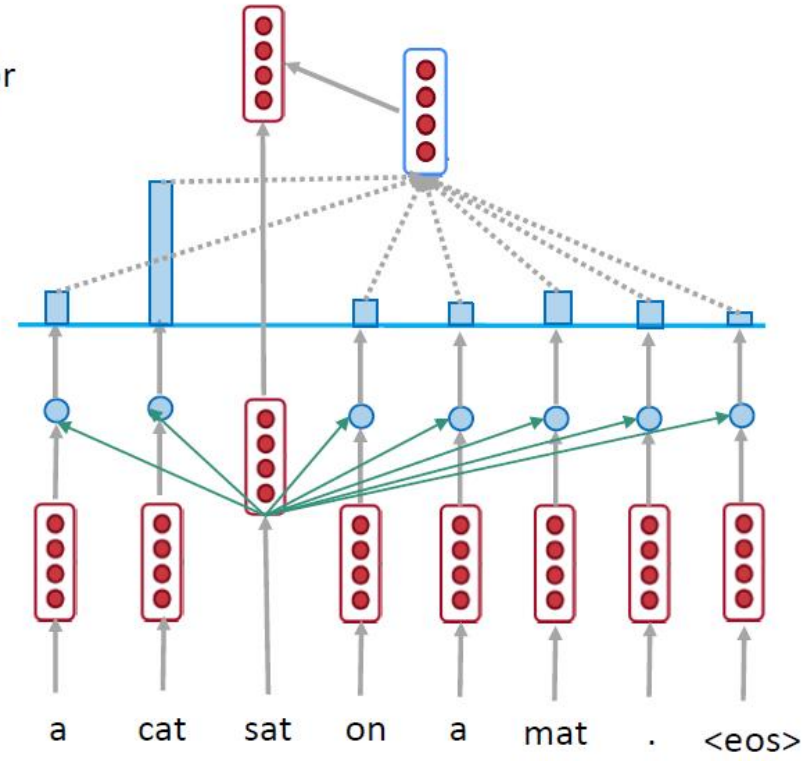
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Scaled Dot-Product Attention



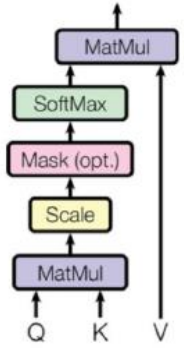
Self-attention: A Running Example

update representation for the word "sat"



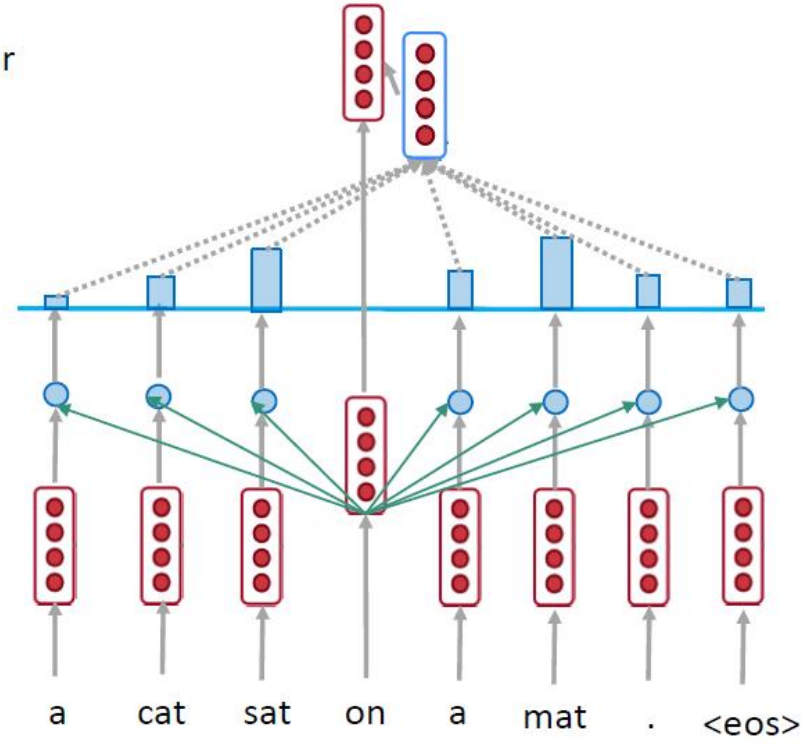
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Scaled Dot-Product Attention



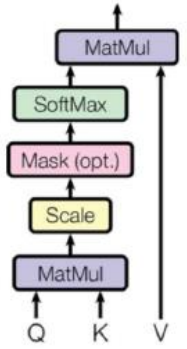
Self-attention: A Running Example

update representation for the word "on"



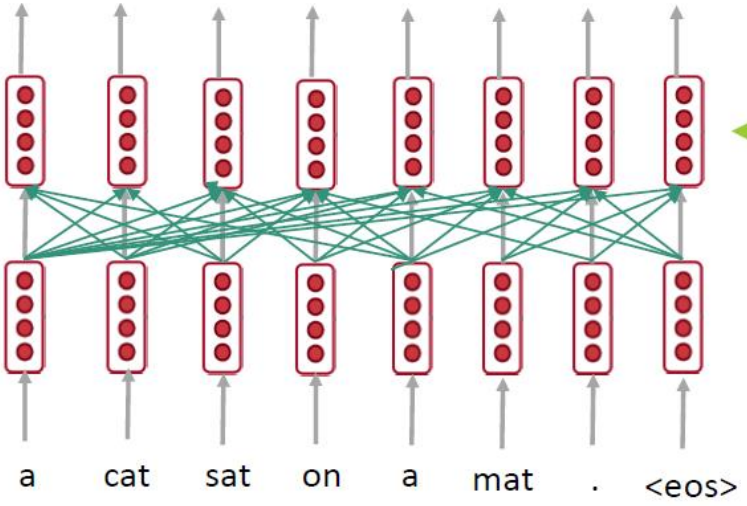
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Scaled Dot-Product Attention



Self-attention: A Running Example

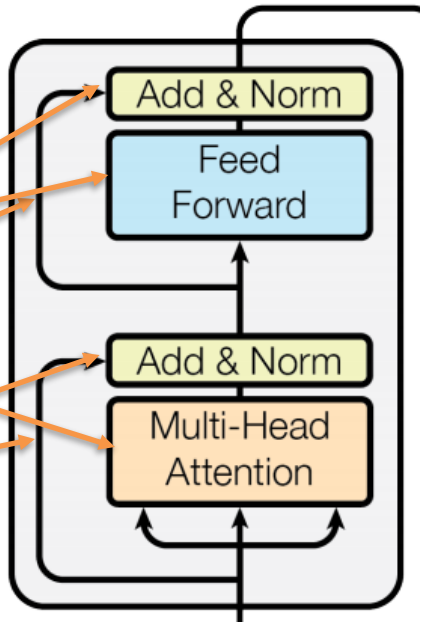
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



Updated representations for each word (one layer)

Complete transformer block

- Each block has two "sublayers"
  - 1. Multihead attention
  - 2. 2 layer feed-forward Nnet (with relu)



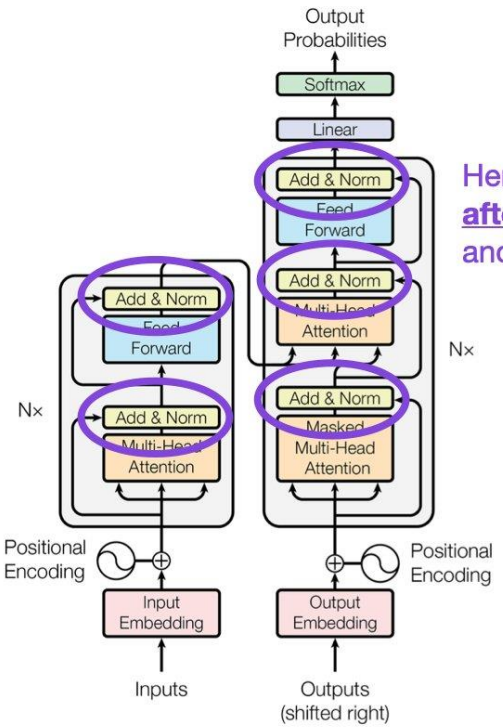
- Each of these two steps also has:
  - Residual (short-circuit) connection and LayerNorm:
  - LayerNorm(x + Sublayer(x))
    - Layernorm changes input to have mean 0 and variance 1, per layer and per training point (and adds two more parameters)

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2} \qquad h_i = f\left(\frac{g_i}{\sigma_i}\left(a_i - \mu_i\right) + b_i\right)$$

Layer Normalization by Ba, Kiros and Hinton, https://arxiv.org/pdf/1607.06450.pdf

New Fun Fact (Found in May 2023)

Post-LN Transformer



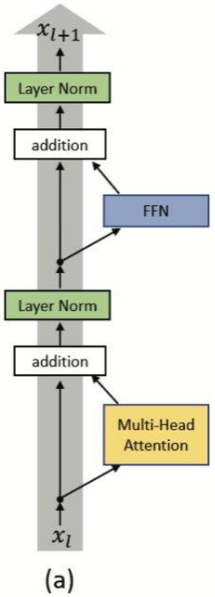
Here, LayerNorms are **after** attention and **after** fully connected layers

Figure 1: The Transformer - model architecture.

Attention Is All You Need, https://arxiv.org/abs/1706.03762

Places the layer normalization between the residual blocks: the expected gradients of the parameters near the output layer are large

Paper: addition + layer norm
Code: dropout + addition + layer norm



On Layer Normalization in the Transformer Architecture, https://arxiv.org/abs/2002.04745

https://twitter.com/rasbt/status/1655575611979489282

New Fun Fact (Found in May 2023)

Better gradients if LayerNorm is placed inside residual connection, before the attention and fully connected layers

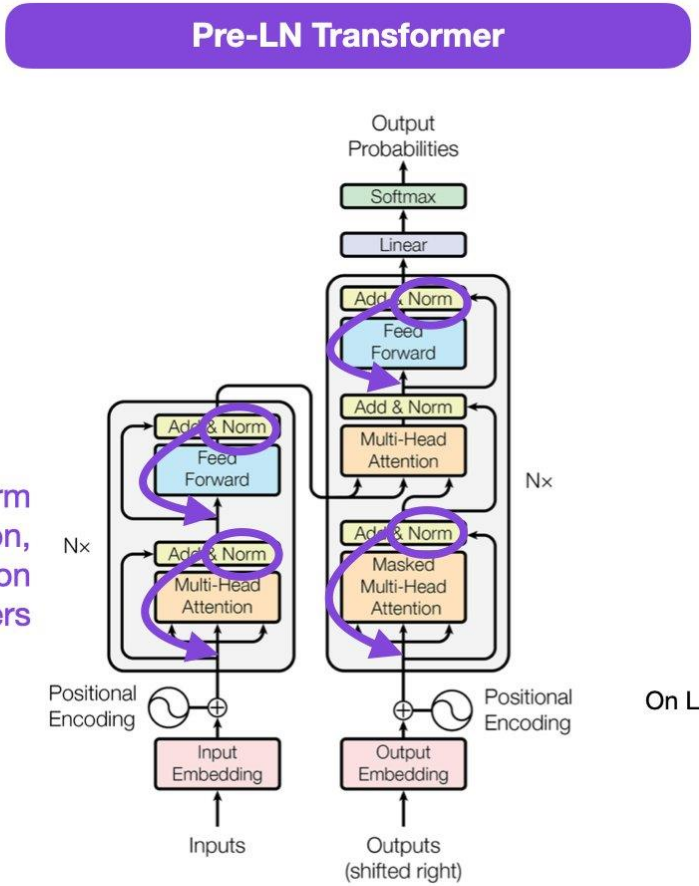
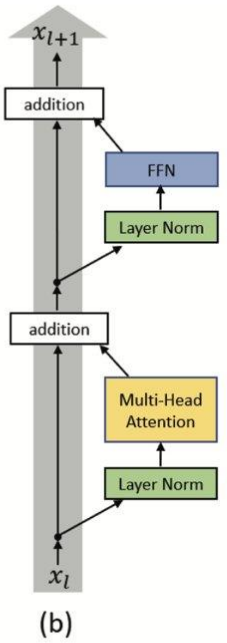


Figure 1: The Transformer - model architecture.

Attention Is All You Need, https://arxiv.org/abs/1706.03762



(b)

On Layer Normalization in the Transformer Architecture, https://arxiv.org/abs/2002.04745

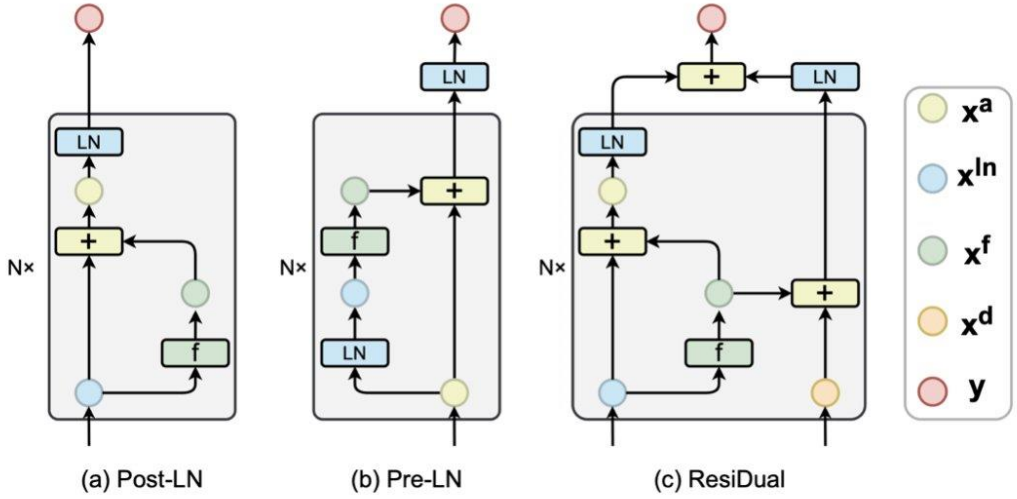
Method	Gradient Vanishing	Representation Collapse
Post-LN	☹	😄
Pre-LN	😄	☹

ResiDual: Transformer with Dual Residual Connections, https://arxiv.org/abs/2304.14802

https://twitter.com/rasbt/status/1655575611979489282

New Fun Fact (Found in May 2023)

Dual-Residual Transformer



No representation collapse

No vanishing gradient

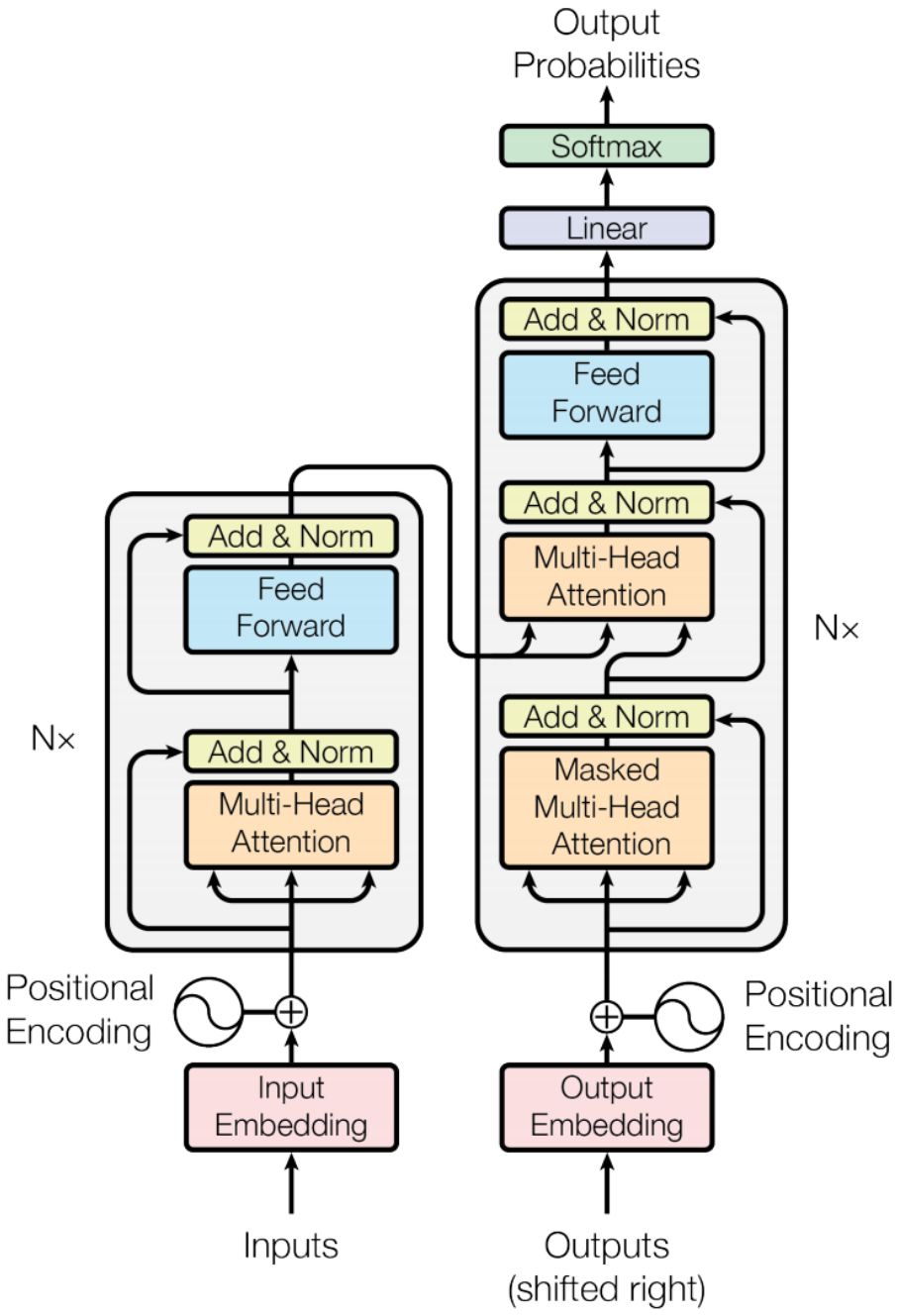
Best of both worlds

Method	Gradient Vanishing	Representation Collapse
Post-LN	☹	😄
Pre-LN	😄	☹
ResiDual	😄	😄

ResiDual: Transformer with Dual Residual Connections,
https://arxiv.org/abs/2304.14802

https://twitter.com/rasbt/status/1655575611979489282

Blocks are repeated
N=6 times

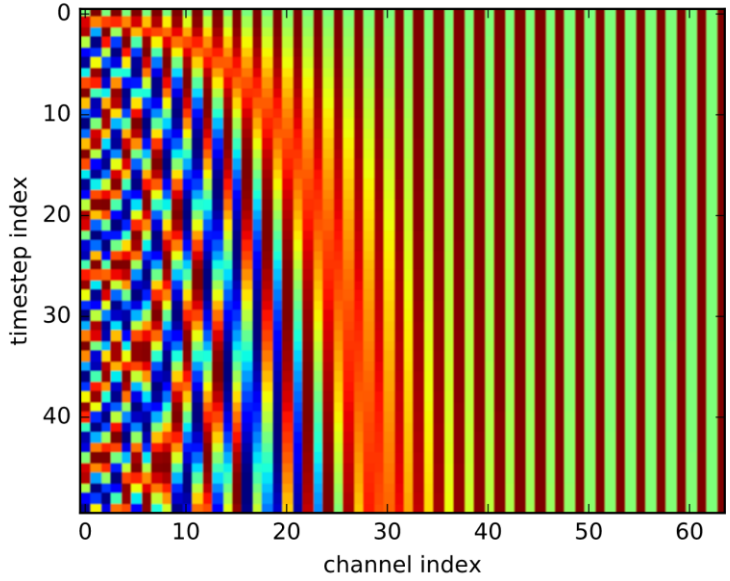


Encoder Input

- Added is a positional encoding so same words at different locations have different overall representations:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

pos is the position of a word
i is the dimension index



https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Encoder Input

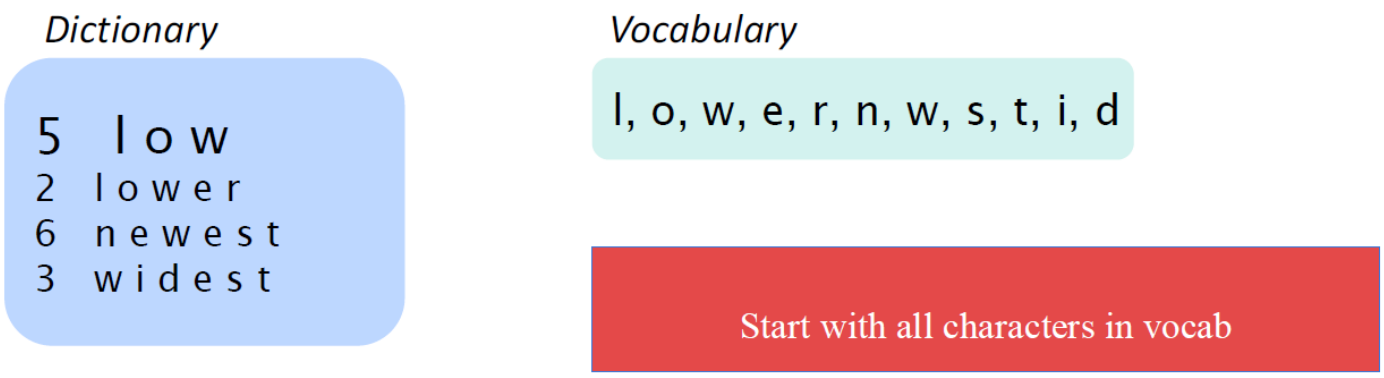
- Actual word representations are byte-pair encodings
  - Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units. ACL 2016.
- Need to handle **large, open vocabulary**
  - Rich morphology:
    - nejneobhospodařovávatelnějšímu - Czech = “to the worst farmable one”
    - Donaudampfschiffahrtsgesellschaftskapitän – German = Danube steamship company captain
  - Informal spelling: goooooood morning !!!!!
- Need to be able to operate at sub-word levels!

Original Byte Pair Encoding for Data Compression

- Suppose the data to be encoded is
  - aaabdaaabac
- The byte pair "aa" occurs most often, so it will be replaced by a byte that is not used in the data, "Z". Now there is the following data and replacement table:
  - ZabdZabac
  - Z=aa
- Then the process is repeated with byte pair "ab", replacing it with Y:
  - ZYdZYac
  - Y=ab
  - Z=aa
- The only literal byte pair left occurs only once, and the encoding might stop here. Or the process could continue with recursive byte pair encoding, replacing "ZY" with "X":
  - XdXac
  - X=ZY
  - Y=ab
  - Z=aa
- This data cannot be compressed further by byte pair encoding because there are no pairs of bytes that occur more than once.

Byte Pair Encoding used in NLP

- A compression algorithm:
  - Most frequent byte pair ↦ a new byte.
  - Replace bytes with character ngrams
- A word segmentation algorithm:
  - Start with a vocabulary of characters.
  - Most frequent ngram pairs ↦ a new ngram.



(Example from Sennrich)

Byte Pair Encoding used in NLP

- A compression algorithm:
  - Most frequent byte pair ↦ a new byte.
  - Replace bytes with character ngrams
- A word segmentation algorithm:
  - Start with a vocabulary of characters.
  - Most frequent ngram pairs ↦ a new ngram.

Dictionary

5 l o w
2 l o w e r
6 n e w **es** t
3 w i d **es** t

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, **es**

Add a pair (e, s) with freq 9

(Example from Sennrich)

Byte Pair Encoding used in NLP

- A compression algorithm:
  - Most frequent byte pair ↦ a new byte.
  - Replace bytes with character ngrams
- A word segmentation algorithm:
  - Start with a vocabulary of characters.
  - Most frequent ngram pairs ↦ a new ngram.

Dictionary

5 l o w
2 l o w e r
6 n e w **est**
3 w i d **est**

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, es, **est**

Add a pair (es, t) with freq 9

(Example from Sennrich)

Byte Pair Encoding used in NLP

- A compression algorithm:
  - Most frequent byte pair ↦ a new byte.
  - Replace bytes with character ngrams
- A word segmentation algorithm:
  - Start with a vocabulary of characters.
  - Most frequent ngram pairs ↦ a new ngram.

Dictionary

5 **lo** w
2 **lo** w e r
6 n e w est
3 w i d est

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, es, est, **lo**

Add a pair (l, o) with freq 7

(Example from Sennrich)

Byte Pair Encoding used in NLP

- Automatically decide vocabs for NMT
  - Top places in WMT 2016!
  - https://github.com/rsennrich/nematus

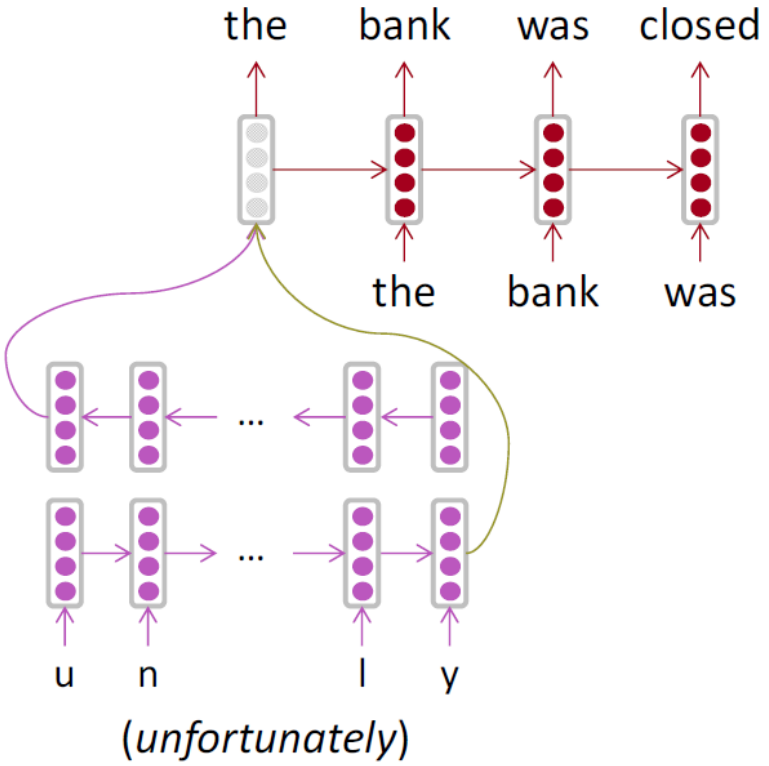
- Google's WordPiece Embeddings (Google Neural Machine Translation System (Wu et al. (2016))

  - **Word**: Jet makers feud over seat width with big orders at stake
  - **wordpieces**: _J et _makers _fe ud _over _seat _width _with _big _orders _at _stake

    - The word "Jet" is broken into two wordpieces "_J" and "et", and the word "feud" is broken into two wordpieces "_fe" and "ud". The other words remain as single wordpieces. "_" is a special character added to mark the beginning of a word.
    - The wordpiece model is generated using a data-driven approach to maximize the language-model likelihood of the training data, given an evolving word definition.

Many ways of using subwords

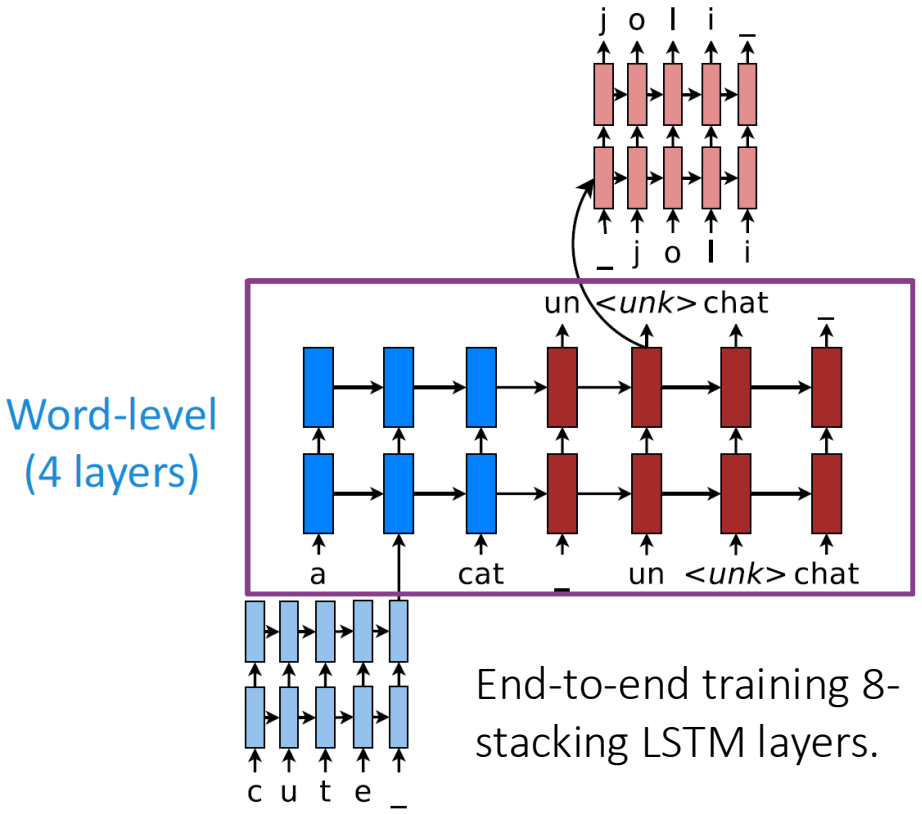
- Character-based LSTM



Ling, Luís, Marujo, Astudillo, Amir, Dyer, Black, Trancoso. Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation. EMNLP'15.

Many ways of using subwords

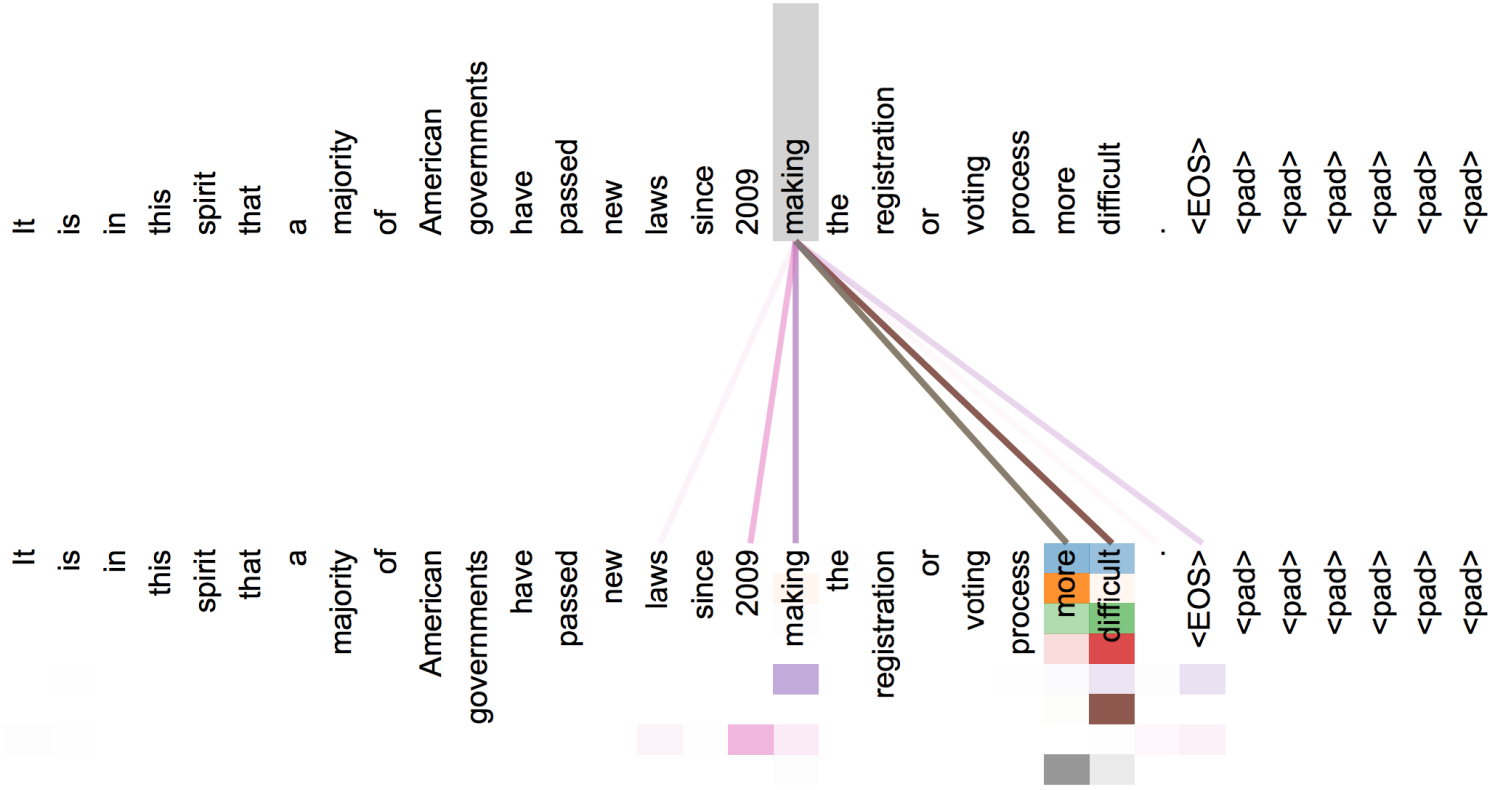
- **Hybrid NMT**
  - A *best-of-both-worlds* architecture:
    - Translate mostly at the word level
    - Only go to the character level when needed.
  - Word-level beam search
  - Char-level beam search for <unk>.
  - More than 2 BLEU improvement over a copy mechanism.



Thang Luong and Chris Manning. Achieving Open Vocabulary Neural Machine Translation with Hybrid Word-Character Models. ACL 2016.

Self Attention Visualization in Layer 5

- Words start to pay attention to other words in sensible ways

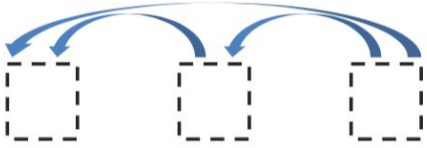


Transformer Decoder

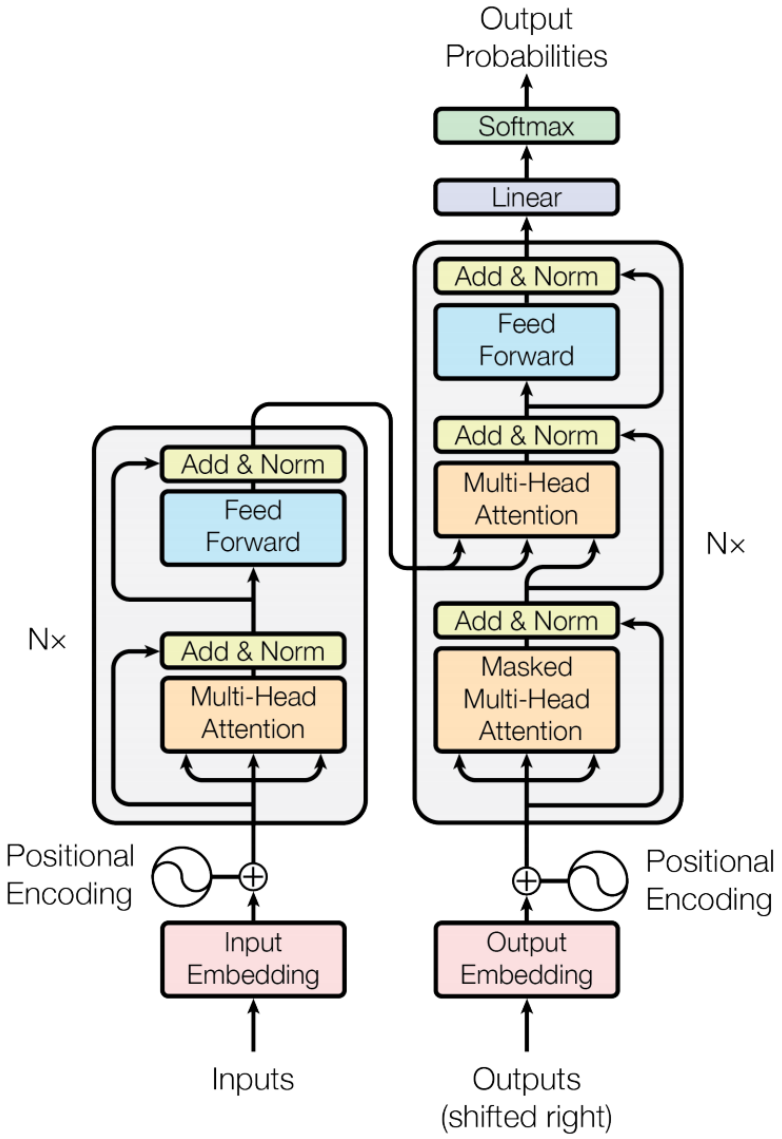
- 2 sublayer changes in decoder
- Masked decoder self-attention
  - Only depends on previous words
- Encoder-Decoder Attention
  - Queries come from previous decoder layer and keys and values come from output of encoder

Encoder-Decoder Attention

Encoder Self-Attention



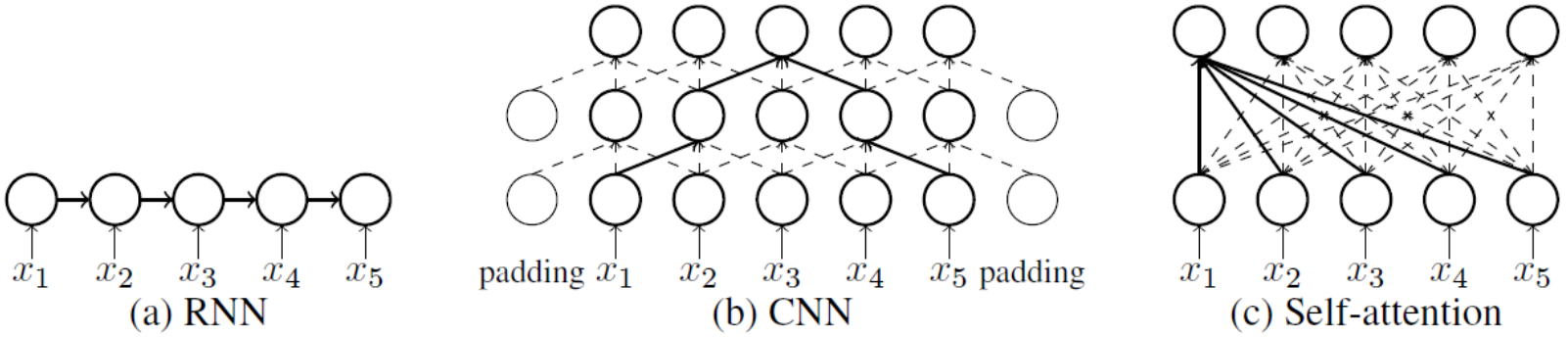
MaskedDecoder Self-Attention



Advantages

- No recurrence: parallel encoding
- Fast training: both encoder and decoder are parallel
- No long range problem: O(1) for all tokens direct connections
- Three attentions: the model does not have to remember too much
- Multi-head attention allows to pay attention to different aspects

- Comparison of self-attention and CNN/RNN



A comparison of RNN, CNN, and self-attention
http://aclweb.org/anthology/D18-1458

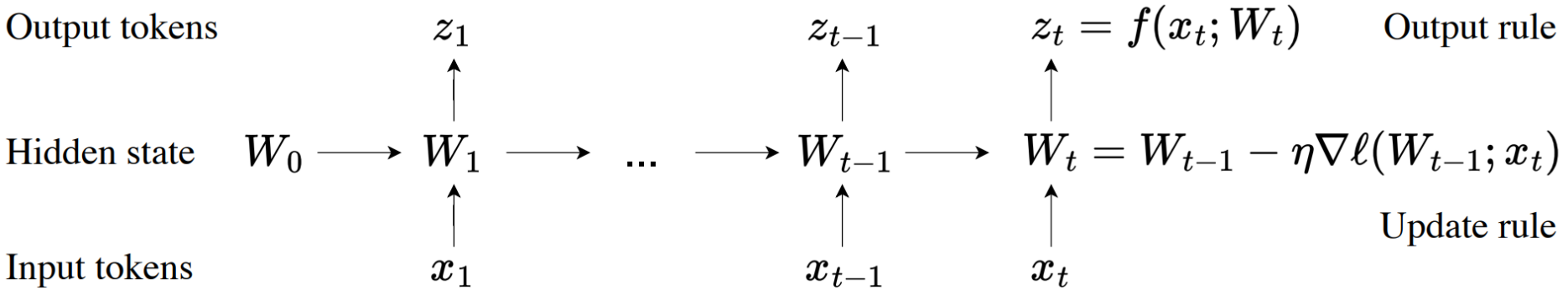
Transformer Networks Visualization

https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

Test Time Training (TTT)

- The hidden state is updated by training even on test sequences
- Loss: denoising autoencoders
  - process x into a corrupted input (low-rank projection)

$$\ell(W; x_t) = \|f(\tilde{x}_t; W) - x_t\|^2.$$



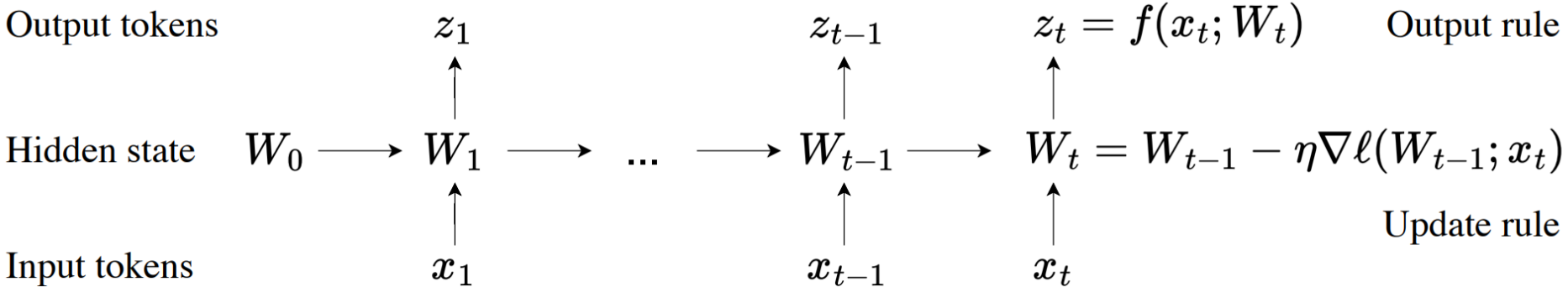
Learning to (Learn at Test Time): RNNs with Expressive Hidden States Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, Carlos Guestrin

Test Time Training (TTT)

- Input corruption by projection using θ_V
- Reconstruction label by another low-rank projection θ_K
- Loss: $\ell(W; x_t) = \left\| f(\theta_K x_t; W) - \theta_V x_t \right\|^2.$

- Output decoding by another projection: θ_Q

$$z_t = f\left(\theta_Q x_t; W_t\right).$$



Test Time Training (TTT)

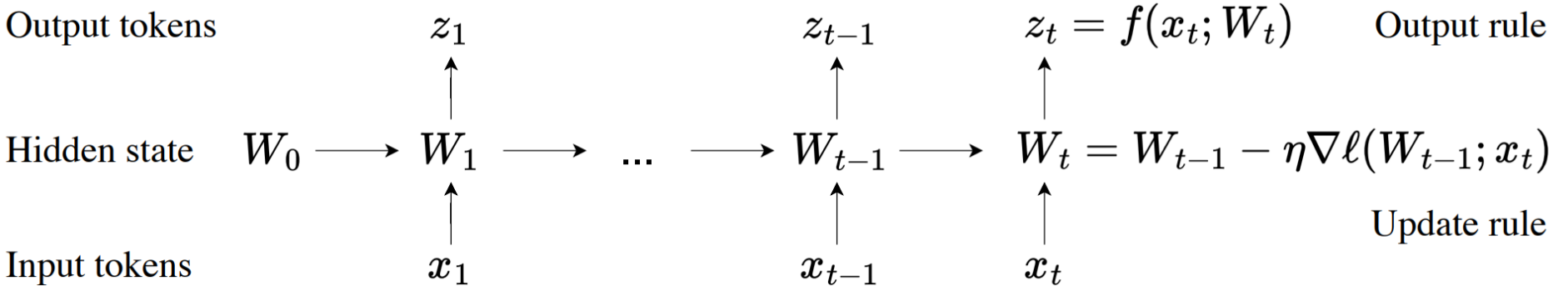
- Loss: $\ell(W;x_t) = \left\| f(\theta_K x_t; W) - \theta_V x_t \right\|^2.$
- Updating rule:

$$\nabla \ell(W_0;x_t) = -2(\theta_V x_t)(\theta_K x_t)^T$$

$$W_t = W_{t-1} - \eta \nabla \ell(W_0;x_t) = W_0 - \eta \sum_{s=1}^{t} \nabla \ell(W_0;x_s) = \sum_{s=1}^{t} (\theta_V x_s)(\theta_K x_s)^T.$$

$$z_t = f\left(\theta_Q x_t; W_t\right) = \sum_{s=1}^{t} (\theta_V x_s)(\theta_K x_s)^T (\theta_Q x_t)$$

Exactly linear transformer



Linear Attention

- Original Transformer

$$\mathbf{Q} = x\mathbf{W}_{\mathbf{Q}}, \qquad \mathbf{K} = x\mathbf{W}_{\mathbf{K}}, \qquad \mathbf{V} = x\mathbf{W}_{\mathbf{V}},$$

$$\mathbf{y}_i = \sum_{j=1}^{i} \frac{\exp\left(\mathbf{Q}_i^\top \mathbf{K}_j / \sqrt{d_{\text{in}}}\right) \mathbf{V}_j}{\sum_{\ell=1}^{i} \exp\left(\mathbf{Q}_i^\top \mathbf{K}_\ell / \sqrt{d_{\text{in}}}\right)},$$

- Linearization

$$\mathbf{y}_i = \sum_{j=1}^{i} \frac{\phi(Q_i^\top K_j)}{\sum_{\ell=1}^{i} \phi(Q_i^\top K_\ell)} V_j = \sum_{j=1}^{i} \frac{\phi(Q_i)^\top \phi(K_j)}{\sum_{\ell=1}^{i} \phi(Q_i)^\top \phi(K_\ell)} V_j = \frac{\phi(Q_i)^\top \sum_{j=1}^{i} \phi(K_j) V_j}{\phi(Q_i)^\top \sum_{\ell=1}^{i} \phi(K_\ell)},$$

- Memory matrix

$$\mathcal{M}_t = \mathcal{M}_{t-1} + K_t^\top V_t \,,$$

$$\mathbf{y}_t = Q_t \mathcal{M}_t \,,$$

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In International conference on machine learning, pages 5156–5165. PMLR, 2020

Titans: Learning to Memorize at Test Time Ali Behrouz† , Peilin Zhong† , and Vahab Mirrokni, 2025

Further Nonlinearization

- Memory matrix

$$\mathcal{M}_t = \mathcal{M}_{t-1} + K_t^\top V_t \,,$$
$$\mathbf{y}_t = Q_t \mathcal{M}_t \,,$$

- Generalization

$$\mathcal{M}_t = f(\mathcal{M}_{t-1}, x_t), \qquad \text{Write Operation}$$
$$\mathbf{y}_t = g(\mathcal{M}_t, x_t), \qquad \text{Read Operation}$$

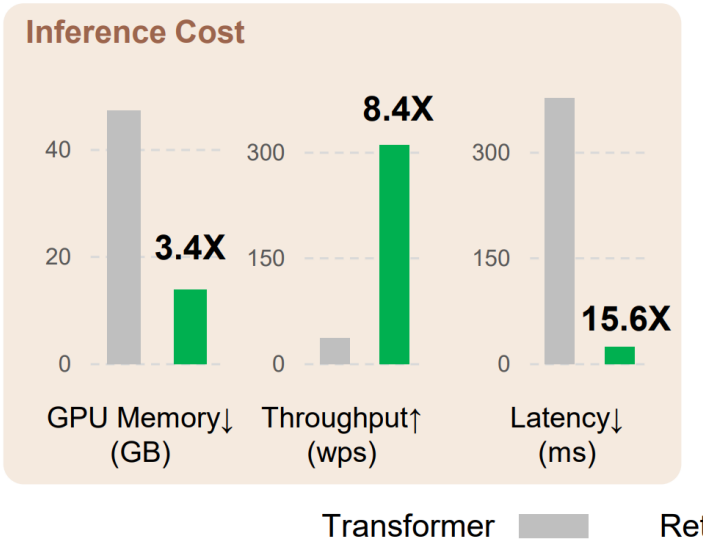
- We can use any kernel function here

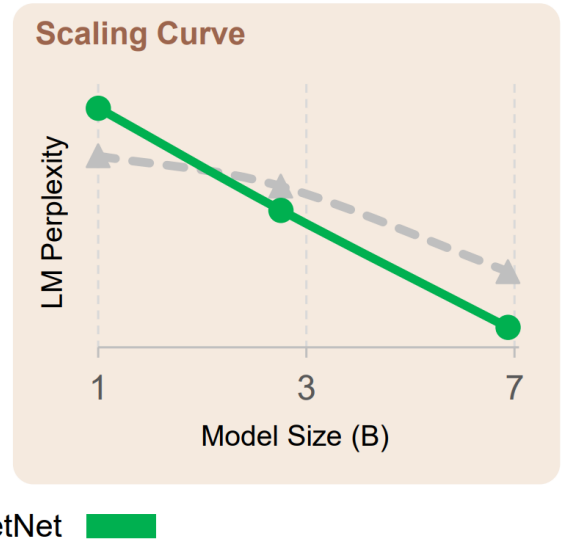
$$\kappa\left(x, x'; \theta_K, \theta_Q\right) \propto e^{(\theta_K x)^T \theta_Q x'}$$

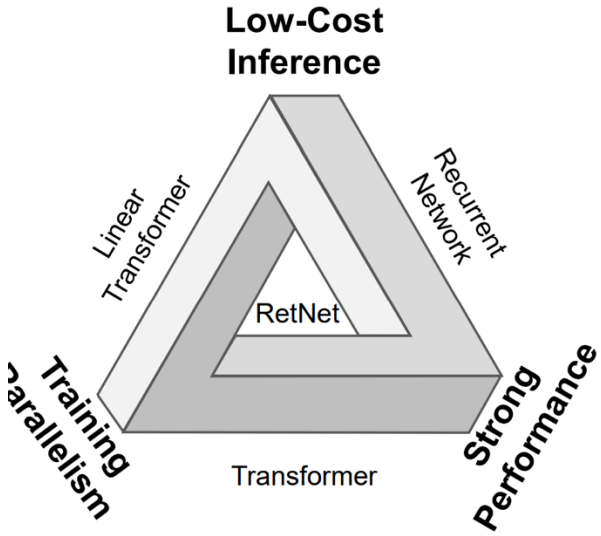
Learning to Memorize at Test Time Ali Behrouz† , Peilin Zhong† , and Vahab Mirrokni, 2025
Learning to (Learn at Test Time): RNNs with Expressive Hidden States Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, Carlos Guestrin

The “Impossible Triangle” of Scaling

- Achieving low-cost inference, efficient long sequence modeling







Retentive Network: A Successor to Transformer for Large Language Models Yutao Sun* †‡ Li Dong* † Shaohan Huang† Shuming Ma† Yuqing Xia† Jilong Xue† Jianyong Wang‡ Furu Wei†⋄, 2023

Test Time Scaling

- Still a very hot topic
  - LLM architecture design, e.g., TTT
  - Reasoning models, e.g., R1, o3
- No SOTA LLM is using TTT so far
  - Possibly because of the sunk cost
  - But it's a promising direction, as the inference cost has become a bottleneck for LLM service