CS 589 Fall 2021 Lecture 9

Encoder-decoder framework

Monday 6:30-9:00
Babbio 122

All zoom links in Canvas
Most slides adapted from Stanford CS224

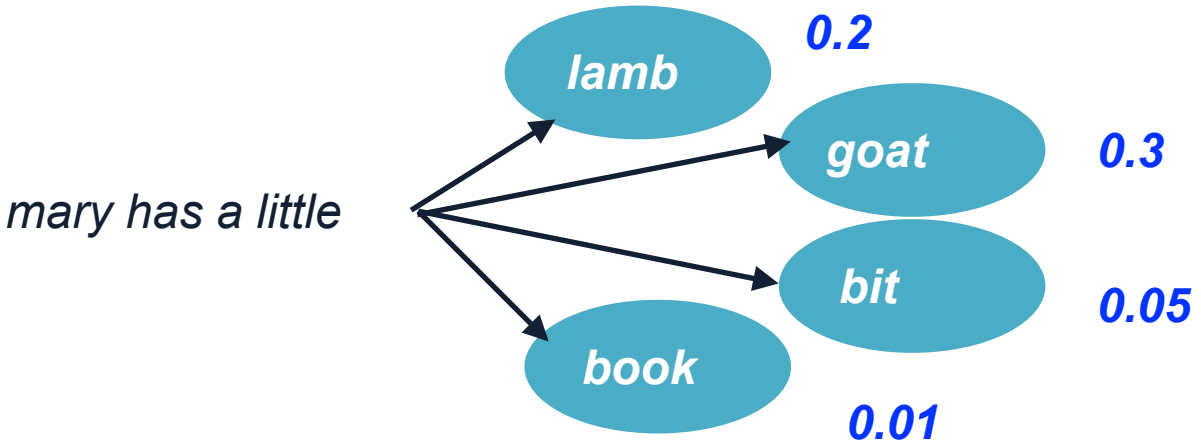
photo: https://www.scubedstudios.com/information-retrieval/

Today's lecture

- Machine translation and attention mechanism
- Transformer
- Encoder representation network
  - Elmo
  - BERT

Review of language models

- Language modeling is the task of **predicting what word comes next**



- Given a sequence of words x1, x2, …, xt, compute the probability distribution for the next word xt+1

$$p(x_{t+1}|x_1, x_2, \cdots, x_t)$$

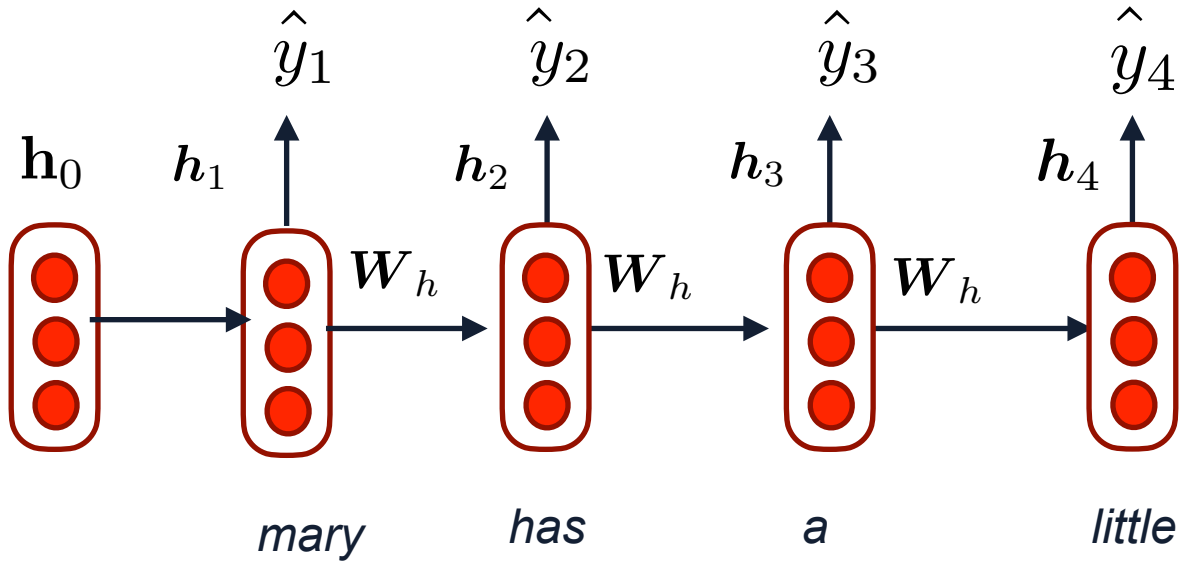
where xt+1 can be any word in the vocabulary $V = \{w_1, \cdots, w_{|V|}\}$

Training RNN method 1: Language modeling

- Minimizes the cross entropy loss:

$$J_t(\theta) = CE(y_t, \hat{y}_t)$$
$$= -\sum_{w \in V} y_{t,w} \log \hat{y}_{t,w}$$
$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\theta)$$
$$= \frac{1}{T}(-\log 0.85$$
$$-\log 0.62$$
$$-\log 0.79$$
$$-\log 0.82$$
$$+ \cdots)$$

a	[0.01,	[0.09,	**[0.79,**	[0.09,
...	...	**0.62,**	...	...
little	0.01,	...	0.01,	**0.82,**
...				
mary	**0.85]**	0.10]	0.10]	0.10]



Training RNN method 2: Text classification

output distribution

$$\hat{y}_t = \text{softmax}\left(Uh_t + b_2\right) \in \mathbb{R}^3$$

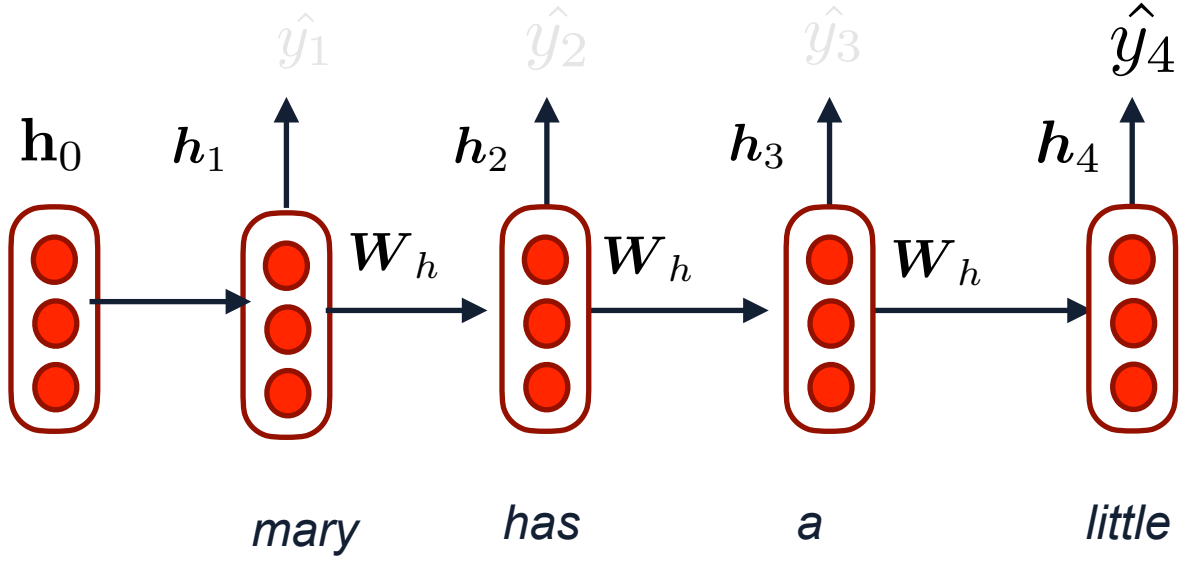
- Cross entropy loss (min):

$$J_s(\theta) = CE(y_s, \hat{y}_s)$$
$$= -\sum_{label \in \{pos, neutral, neg\}} \mathbb{1}[y_s == label] \log \hat{y}_{s,label}$$

- Accuracy (max):

$$J_s(\theta) = Acc(y_s, \hat{y}_s)$$
$$= \sum_{label \in \{pos, neutral, neg\}} \mathbb{1}[y_s == label] \mathbb{1}[\hat{y}_{s,label} > 0.333]$$

pos	[0.2,
neutral	**0.5,**
neg	0.3]



Machine translation

- Translating a sentence from one language (**source language**) to another language (**target language**)

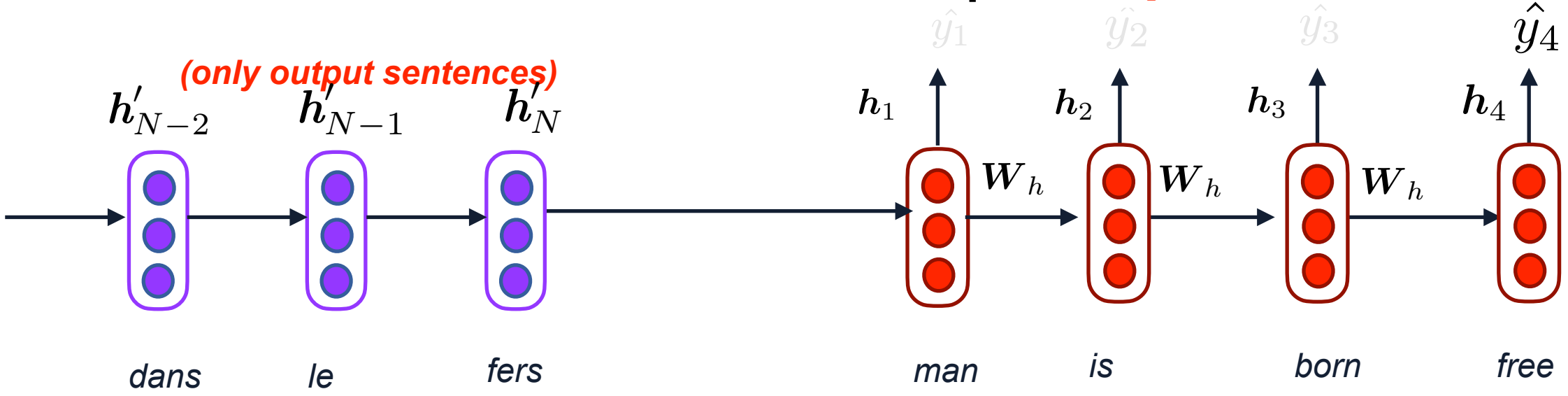
x: *L'homme est né libre, et partout il est dans les fers*

y: *Man is born free, but everywhere he is in chains*

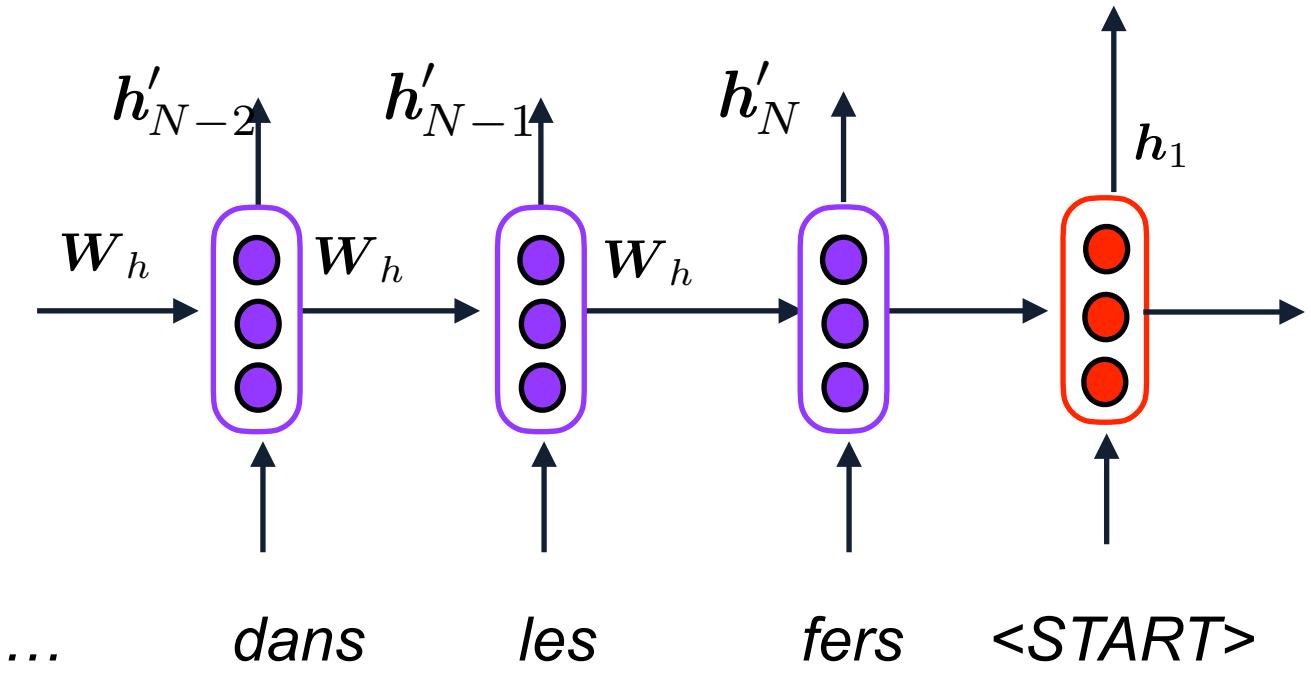
Training NN method 3: Machine Translation

- Minimizes the cross entropy loss:

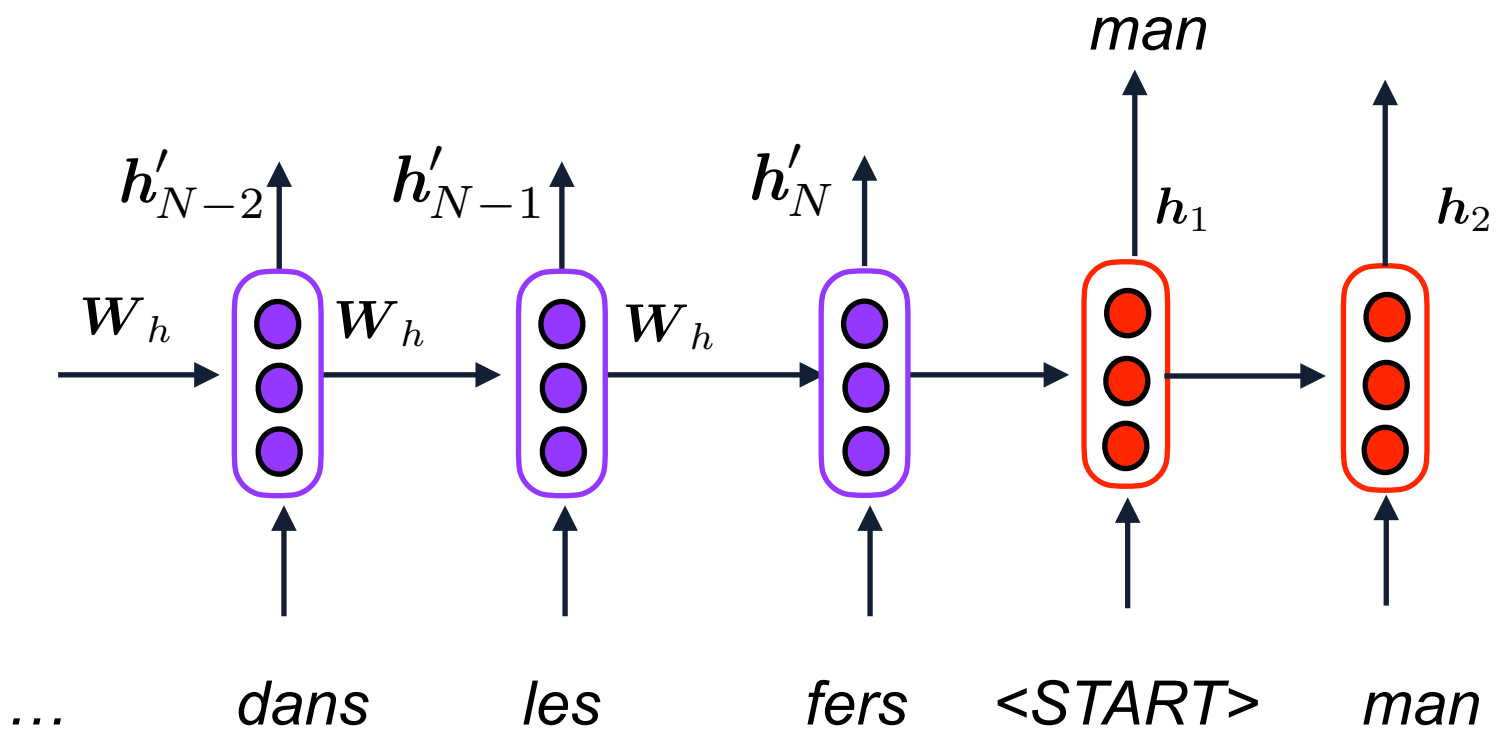
born	[0.01,	[0.09,	**[0.79,**	[0.09,
...	...		...	...
free	0.01,	...	0.01,	**0.82,**
...	**0.85,**			
is	0.01]	**0.62]**	0.10]	0.10]



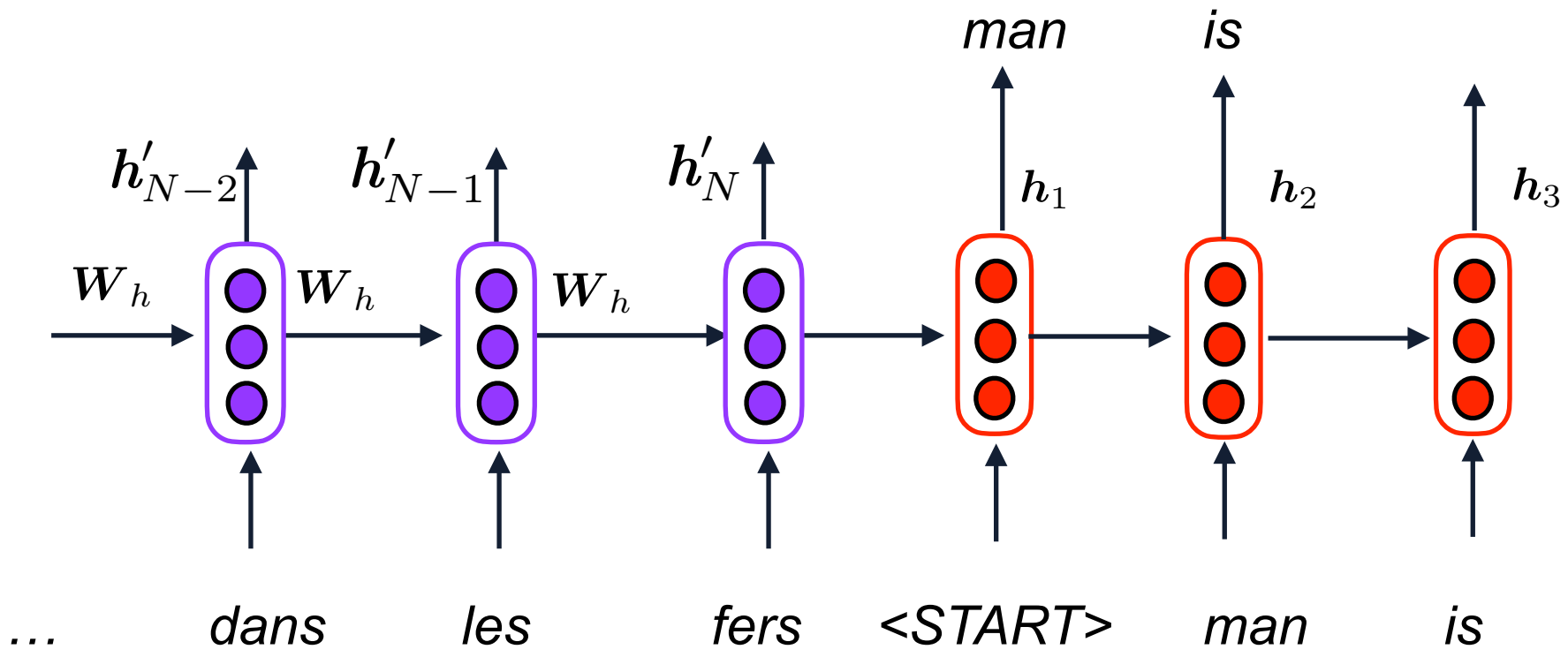
Encoder-decoder framework



Encoder-decoder framework

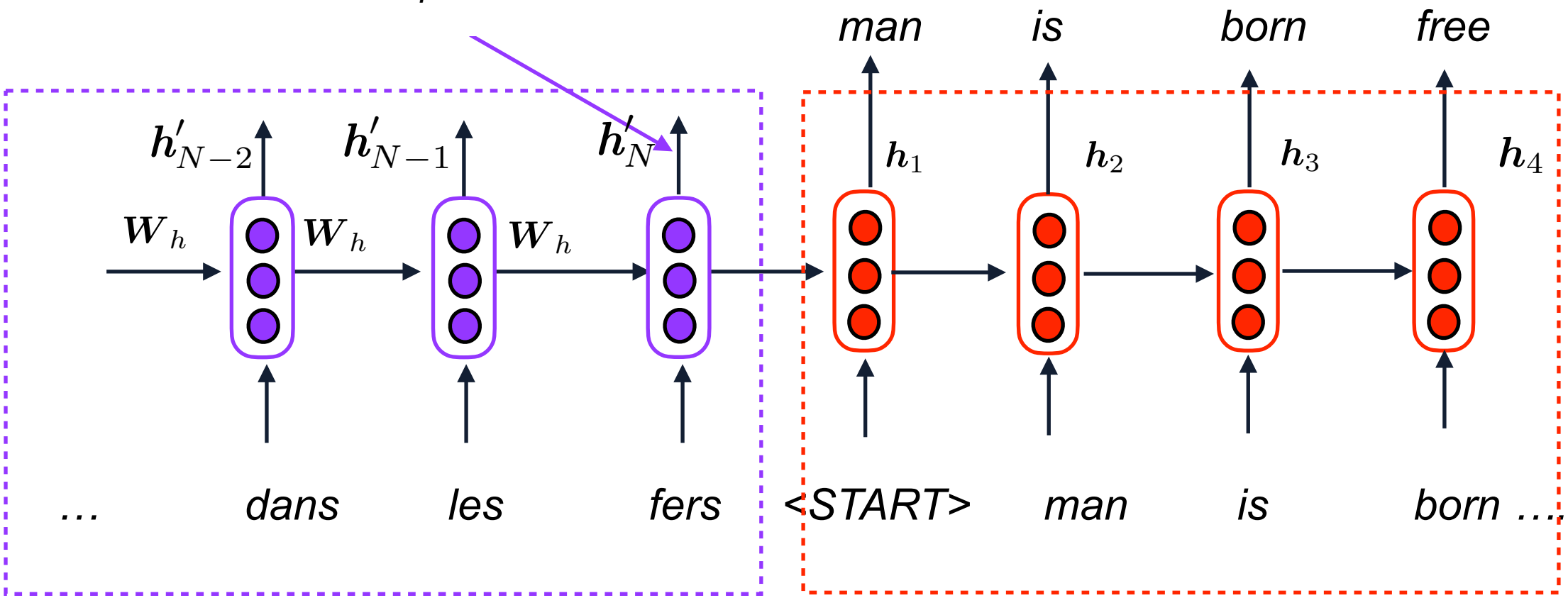


Encoder-decoder framework



Encoder-decoder framework

*hN' **encodes** the info of input French sentence*



encoder RNN

decoder RNN

Training NN task 3: Neural machine translation

- Translating a sentence from one language (**source language**) to another language (**target language**)

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

train: $max_y \sum_{(x^i, y^i)} \log P(y_i|x_i)$ ***similar training objective as language modeling***

predict: $max_y P(y|x)$

Encoder-decoder framework: Other applications

x: *L'homme est né libre, et partout il est dans les fers*

y: *Man is born free, but everywhere he is in chains*

machine translation

x: *What does SIT stand for?*

y: *Stevens Institute of Technology*

factoid question answering

x: *Open the file file.txt and read it line by line*

y: *with open('file.txt', 'r') as fin:*
lines = fin.readlines()

natural language to code

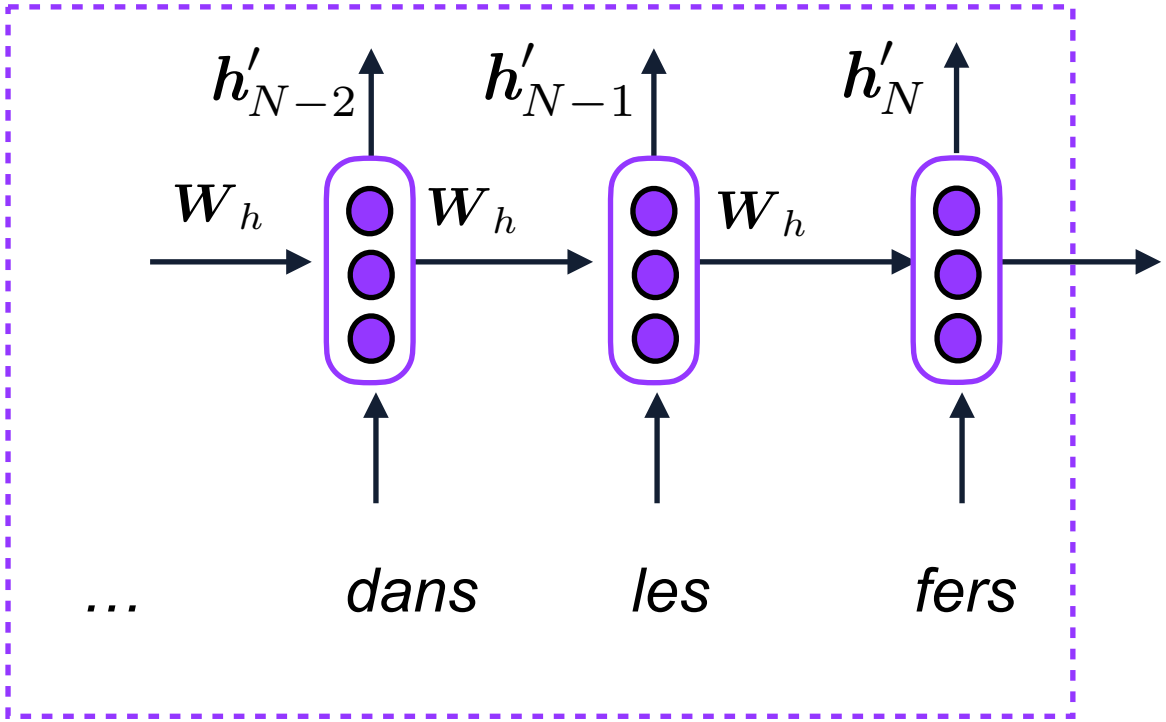
x: *Can you show me the ticket from NY to LA?*

y: *OK, what are the dates of your flight?*

conversational agent

Encoder (Input Network)

*hN' **encodes** the info of input French sentence*



encoder RNN

LSTM encoder:

$$\tilde{c}_t = \tanh\left(\boldsymbol{W}_c h_{t-1} + \boldsymbol{U}_c x_t + b_c\right)$$
$$c_t = \boldsymbol{f}_t \circ c_{t-1} + \boldsymbol{i}_t \circ \tilde{c}_t$$
$$h_t = \boldsymbol{o}_t \circ \tanh c_t$$

Other encoders:

GRU

*LSTM/GRU + **attention***

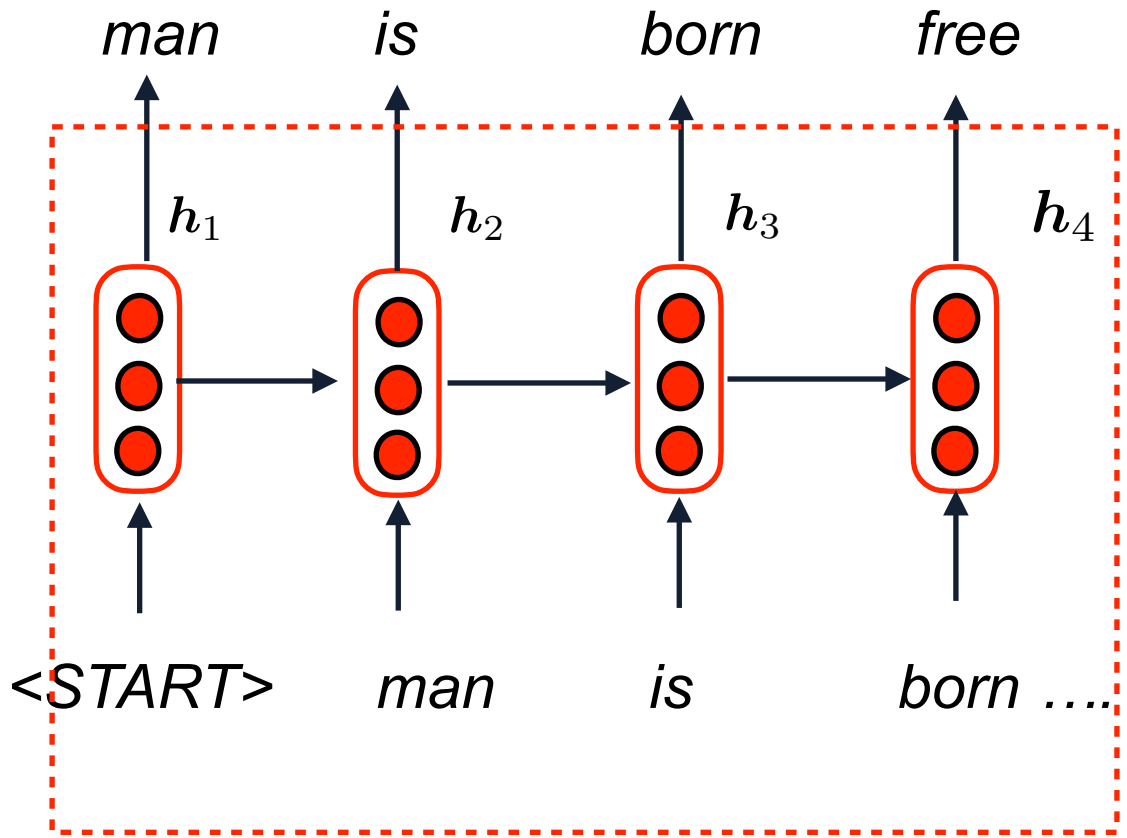
Elmo

BERT

…

Decoder (Output network)

- Decoder selects the word to generate in the target sentence
- Decoder framework can be different from encoder framework
- Training:
  - Train by objective on decoder
- Prediction:
  - Predict the output sequence using the decoder



decoder RNN

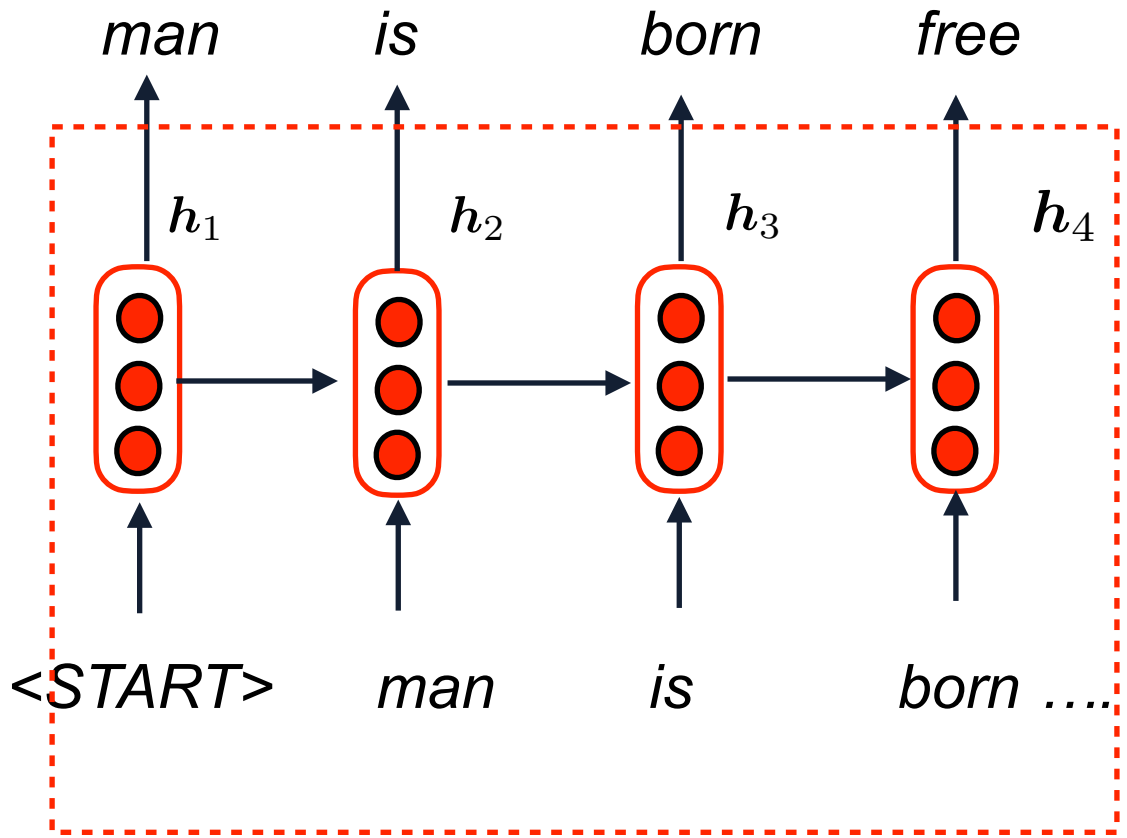
Decoding (Prediction)

- Greedy decoding

$$max_{y_i} P(y_i|y_1, \cdots, y_{i-1}, x)$$

- Disadvantage of greedy decoding: cannot undo decisions

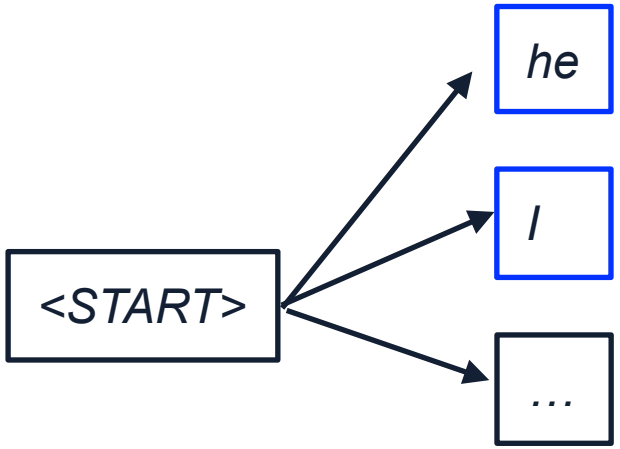
- Exhaustive search? exponential search space



Beam search decoding

- A trade-off between greedy decoding + exhaustive search

- In each step of the decoder, keep track of the k most probable partial translations (**hypotheses**)



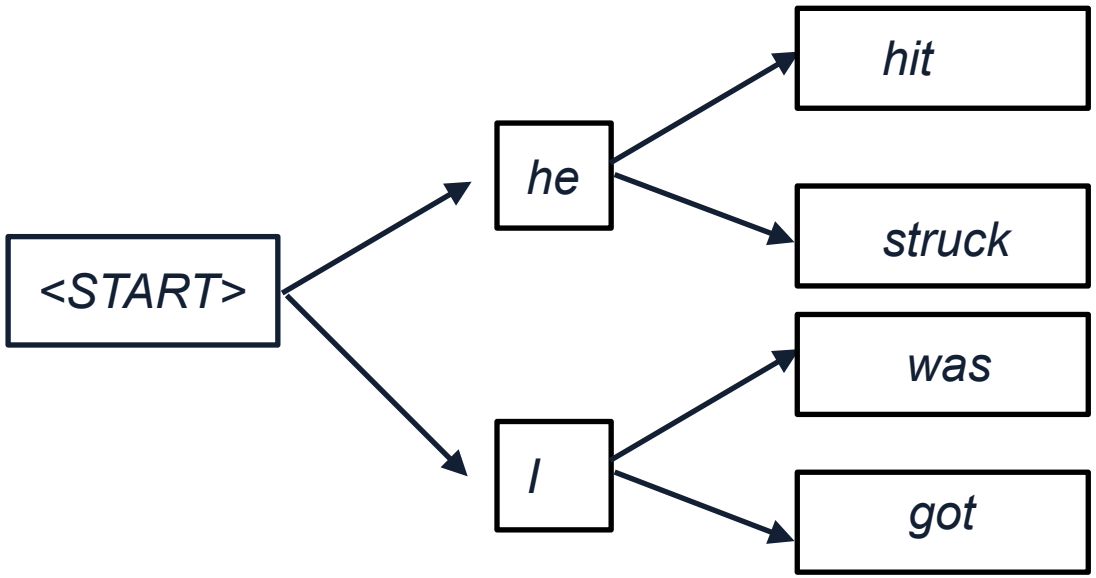
$$-0.7 = \log P_{\text{LM}}\left(he \mid < START >\right)$$

$$-0.9 = \log P_{\text{LM}}\left(I \mid < START >\right)$$

Beam search decoding

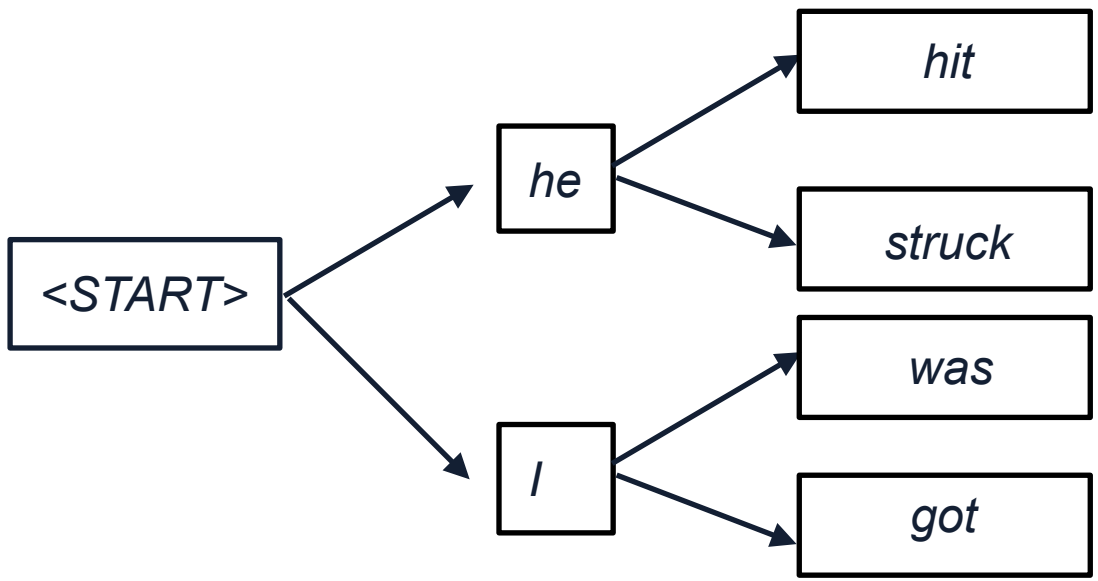
- A trade-off between greedy decoding + exhaustive search

- In each step of the decoder, keep track of the k most probable partial translations (**hypotheses**)



Beam search decoding

- A trade-off between greedy decoding + exhaustive search
- In each step of the decoder, keep track of the k most probable partial translations (**hypotheses**)



$$-1.7 = \log P_{\mathrm{LM}}\,(hit \mid < START > he) + -0.7$$

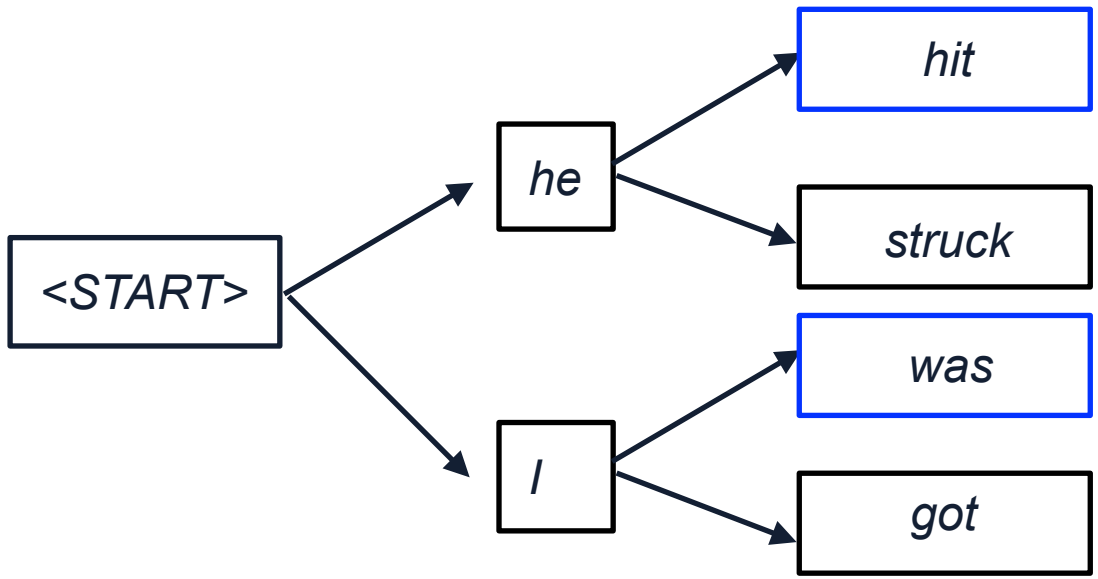
$$-2.9 = \log P_{\mathrm{LM}}\,(struck \mid < START > he) + -0.7$$

$$-1.6 = \log P_{1M}(was \mid < START > I) + -0.9$$

$$-1.8 = \log \mathrm{P}_{\mathrm{LM}}(got \mid < START > I) + -0.9$$

Beam search decoding

- A trade-off between greedy decoding + exhaustive search
- In each step of the decoder, keep track of the k most probable partial translations (**hypotheses**)



$$-1.7 = \log P_{\mathrm{LM}}\,(hit \mid < START > he) + -0.7$$

$$-2.9 = \log P_{\mathrm{LM}}\,(struck \mid < START > he) + -0.7$$

$$-1.6 = \log P_{1M}(was \mid < START > I) + -0.9$$

$$-1.8 = \log \mathrm{P}_{\mathrm{LM}}(got \mid < START > I) + -0.9$$

Decoding: How do we know when to stop?

- When do we stop the generation?
  - Append an <END> token to every target sentence in the training data
  - Train the model, so it knows when to predict <END>
  - During decoding, stop a hypothesis if <END> is predicted

- We continue beam search util
  - We reach time step T, or
  - We have at least n completed hypotheses

Beam search decoding: Selecting output hypothesis

- Each hypothesis has a log probability score
- Longer hypothesis always have lower scores
- Solution: normalize hypotheses by length

$$\frac{1}{t}\sum_{i=1}^{t}\log P_{\mathrm{LM}}\left(y_i \mid y_1, \ldots, y_{i-1}, x\right)$$

Machine translation timeline

- Statistical machine translation:
  - IBM model 1-5 [Brown et al. 1993]

- 2014: first seq2seq paper was published

- 2016: Google Translate switched from SMT to NMT

- The SMT system, built by hundreds of engineers over many years, outperformed by the NMT systems trained by a handful of engineers in a few months

Evaluating machine translation

- How to evaluate the correctness of a translated sentence?
  - Compare the similarity between the prediction and the ground truth
  - Capturing the similarity beyond unigram matching

- BLEU (Bilingual Evaluation Understudy)
  - 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 for 1, 2, 3 and 4-grams)
    - a penalty for too-short system translations

- BLEU is useful but imperfect (non-overlapping ngrams)

BLEU score

- Step 1: computing the percentage of the i-gram tuples in the hypothesis that also occur in the references

$$P(i) = \frac{Matched(i)}{H(i)}$$

$$Matched(i) = \sum_{t_i} \min\{C_h(t_i), \max_j C_{hj}(t_i)\}$$

#times t_i occurs in the hypothesis h

#times t_i occurs in the j-th reference for hypothesis h

- Step 2: averaging over the percentages:

$$BLEU_a = \{\prod_{i=1}^{N} P(i)\}^{1/N}$$

- Penalizing short translations:

$$\rho = \exp\{\min(0, \frac{n-L}{n})\} \qquad BLEU_b = \rho\ BLEU_a$$

n: length of hypothesis L: length of reference

CS 589 Fall 2021 Lecture 9

Attention mechanism

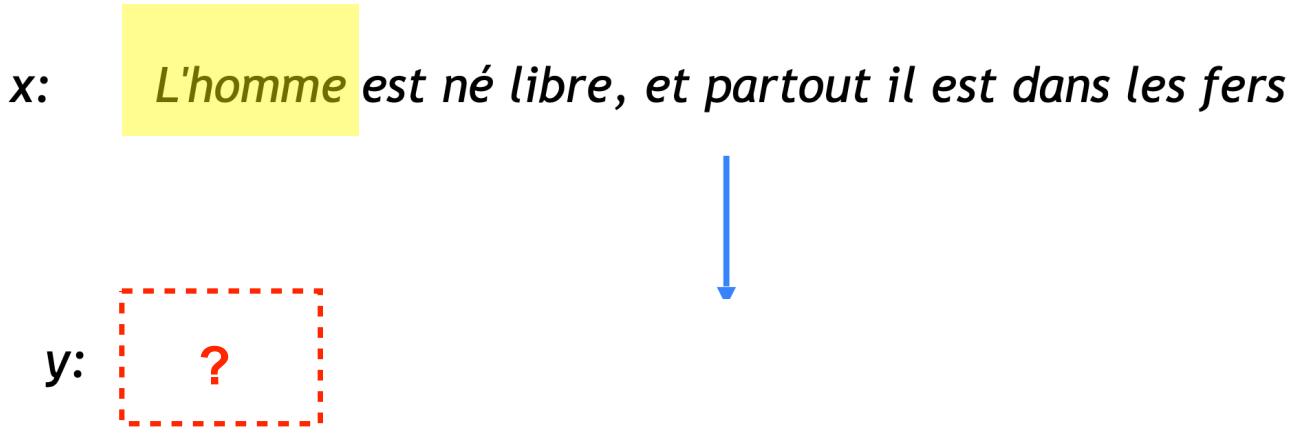
Monday 6:30-9:00
Babbio 122

All zoom links in Canvas
Most slides adapted from Stanford CS224

photo: https://www.scubedstudios.com/information-retrieval/

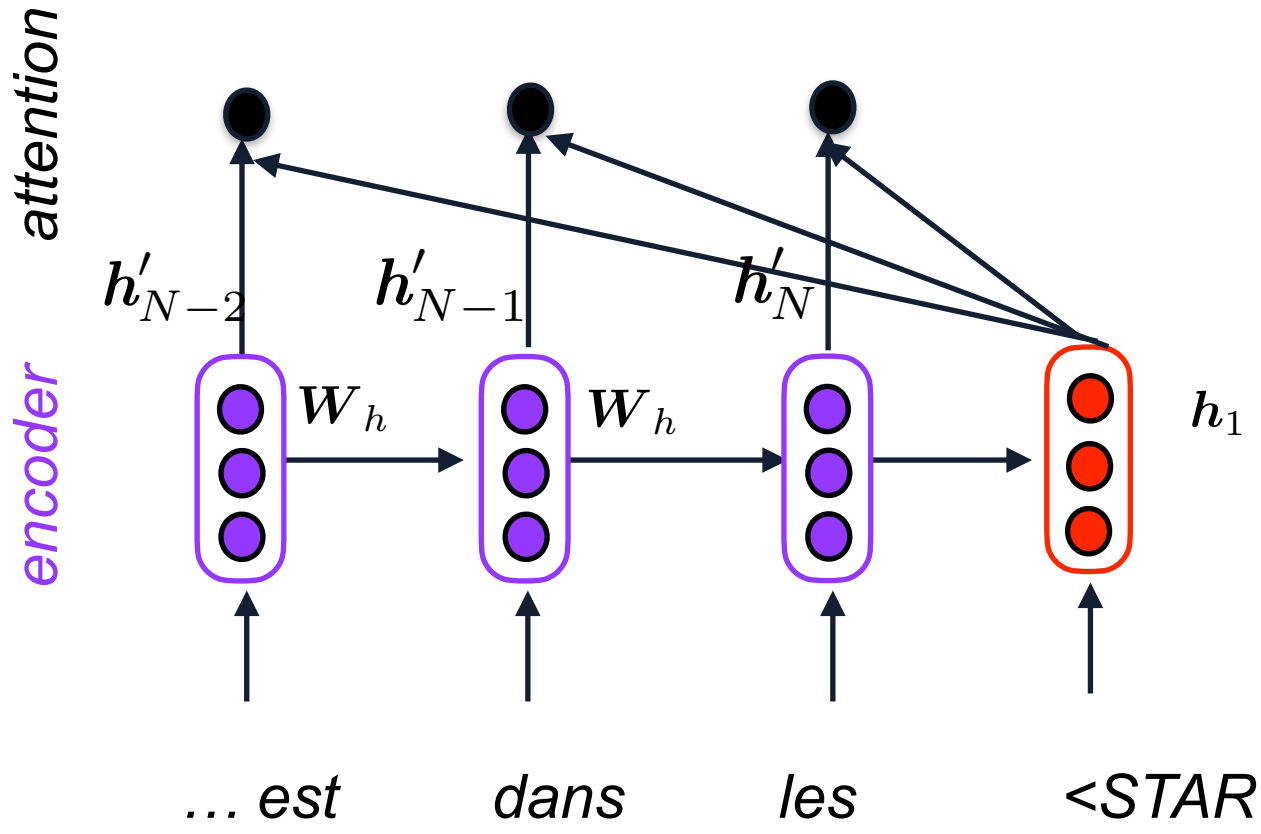
Attention mechanism

- Training the neural network to attend to the most relevant input to the next word to predict



Attention mechanism

- Attention mechanism is designed for each word to **focus** on **one or a just a few words** in the source sentences



dot product [Luong et al. 2015];

RNNSearch [Bahdanau et al. 2014]

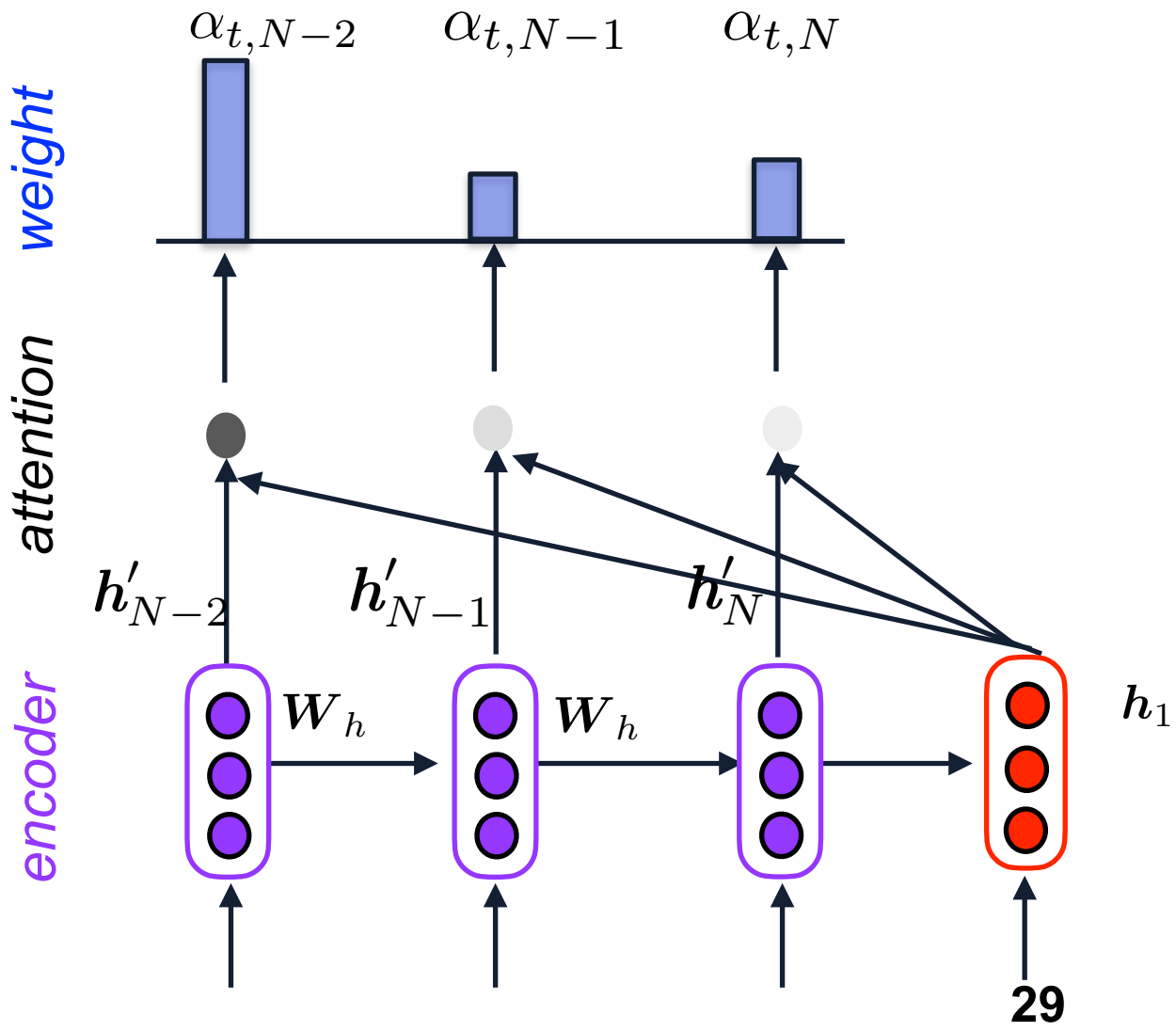
Attention mechanism: Dot-product attention

[Luong et al. 2015]

- Step 1: Getting the attention weight

$$\text{score}\left(\boldsymbol{h}_t, \boldsymbol{h}_i\right) = \boldsymbol{h}_t^\top \boldsymbol{h}_i$$

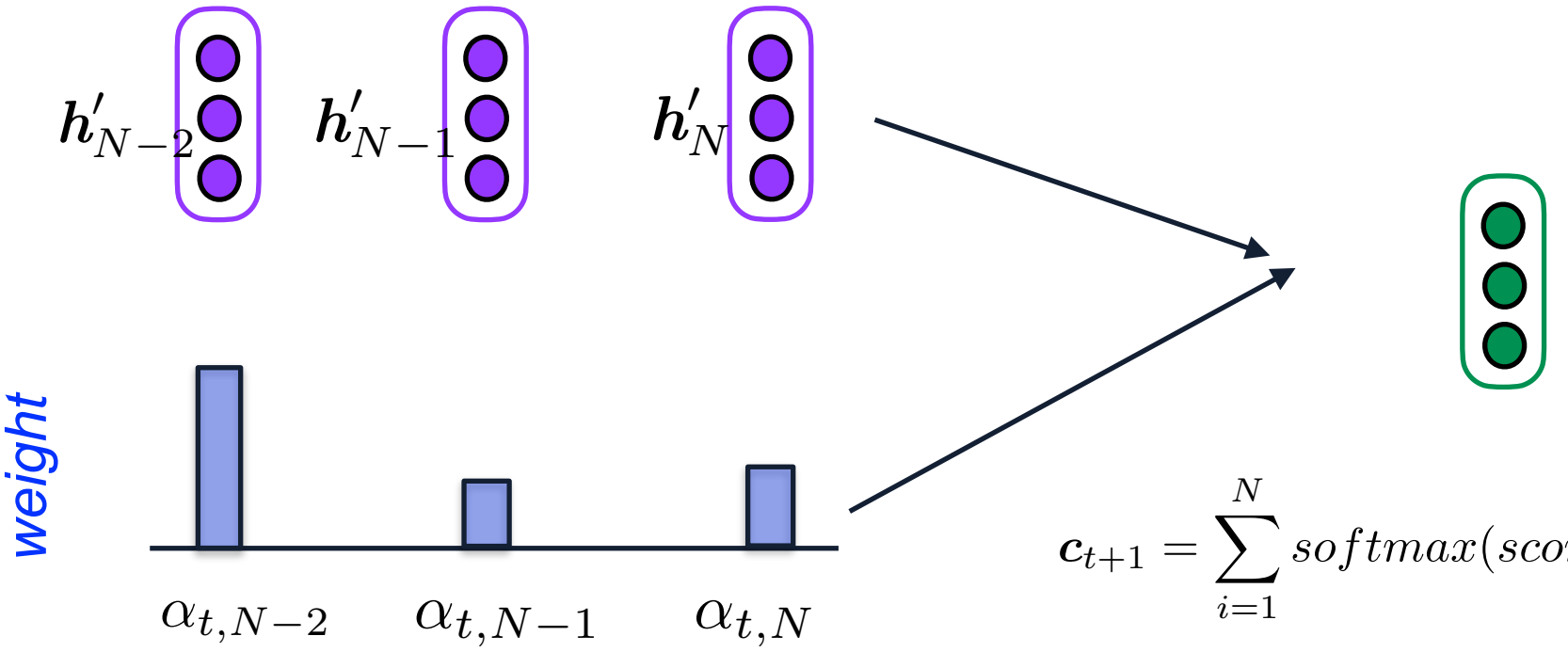
- Using the hidden vector to compute the attention weight



Attention mechanism: Dot-product attention

[Luong et al. 2015]

- Step 2: Computing the memory vector ct using the integration of the hidden vector



$$c_{t+1} = \sum_{i=1}^{N} softmax(score(h_i, h_t))h_i \in \mathbb{R}^h$$

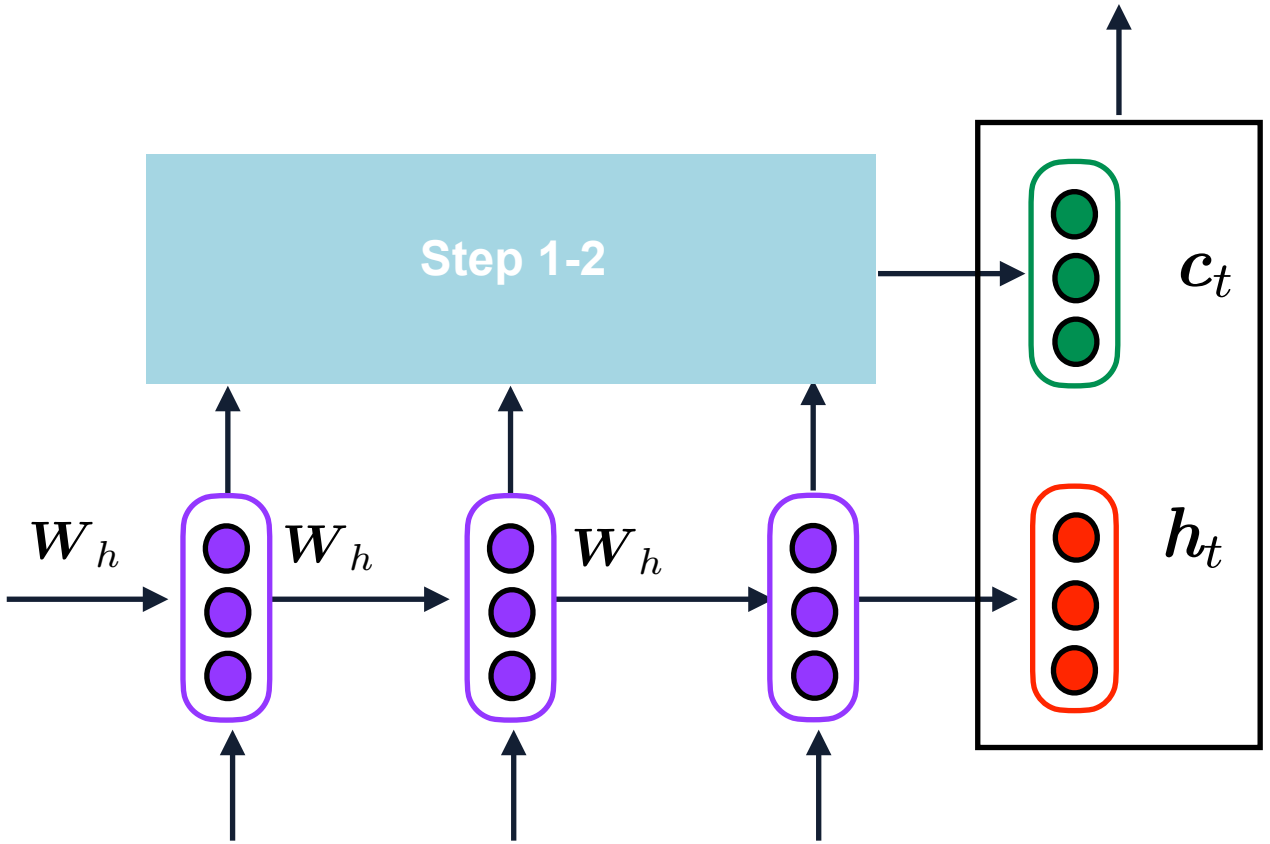
Attention mechanism: Dot-product attention

[Luong et al. 2015]

- Step 3: Using the combination of memory vector ct and hidden vector for prediction

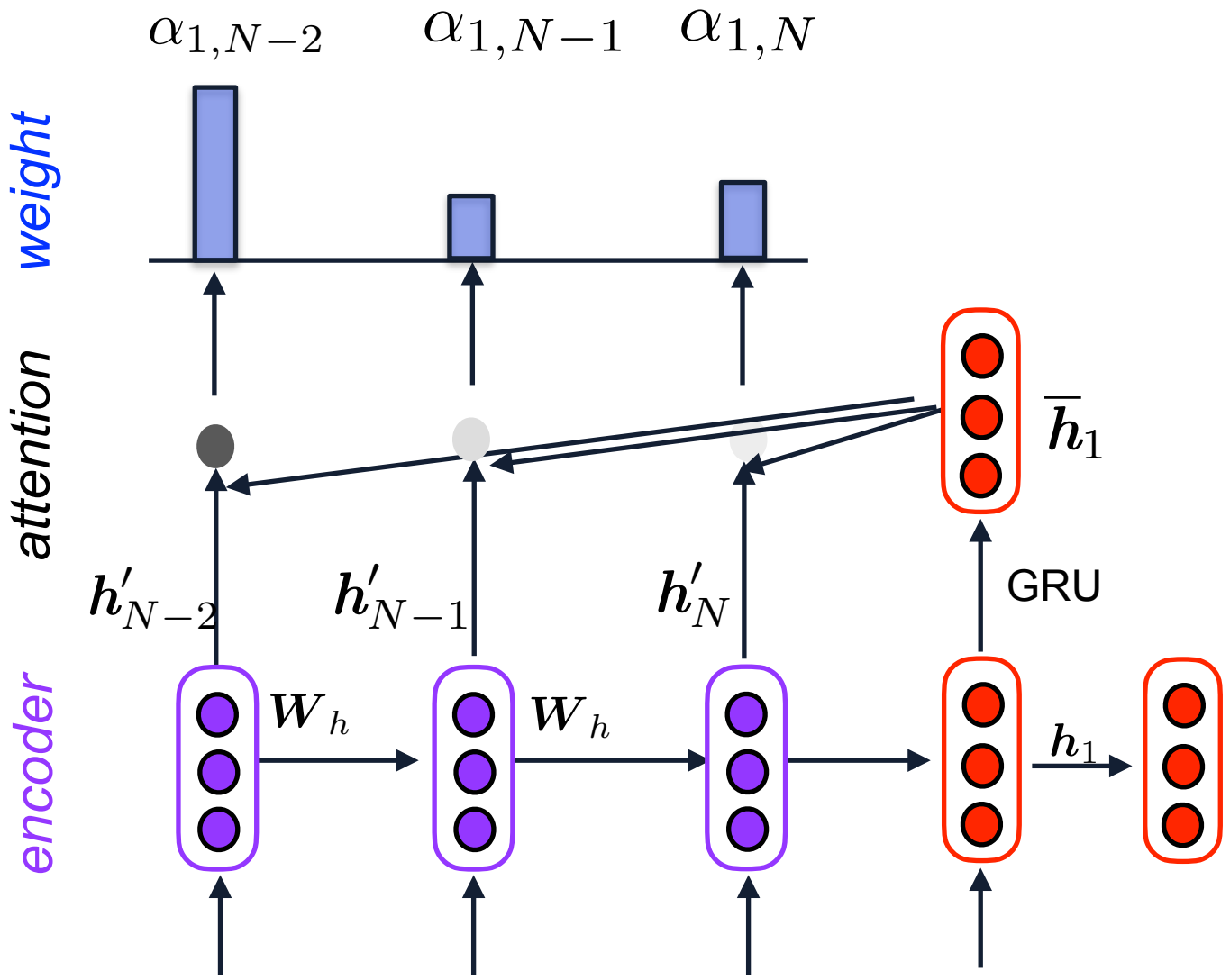
$$\tilde{\boldsymbol{h}}_t = \tanh\left(\boldsymbol{W}_{\boldsymbol{c}}\left[\boldsymbol{c}_t; \boldsymbol{h}_t\right]\right)$$

$$p\left(y_t \mid y_{<t}, x\right) = \operatorname{softmax}\left(\boldsymbol{W}_{\boldsymbol{s}}\tilde{\boldsymbol{h}}_t\right)$$



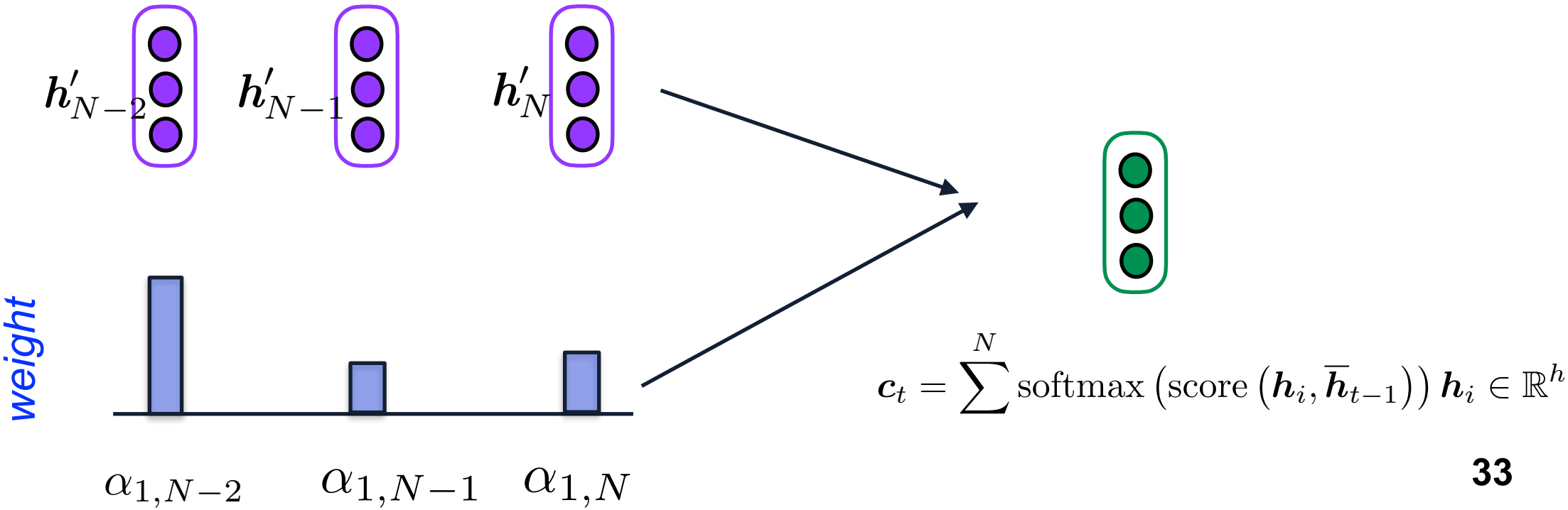
Attention mechanism: RNNSearch

[Bahdanau et al. 2014]



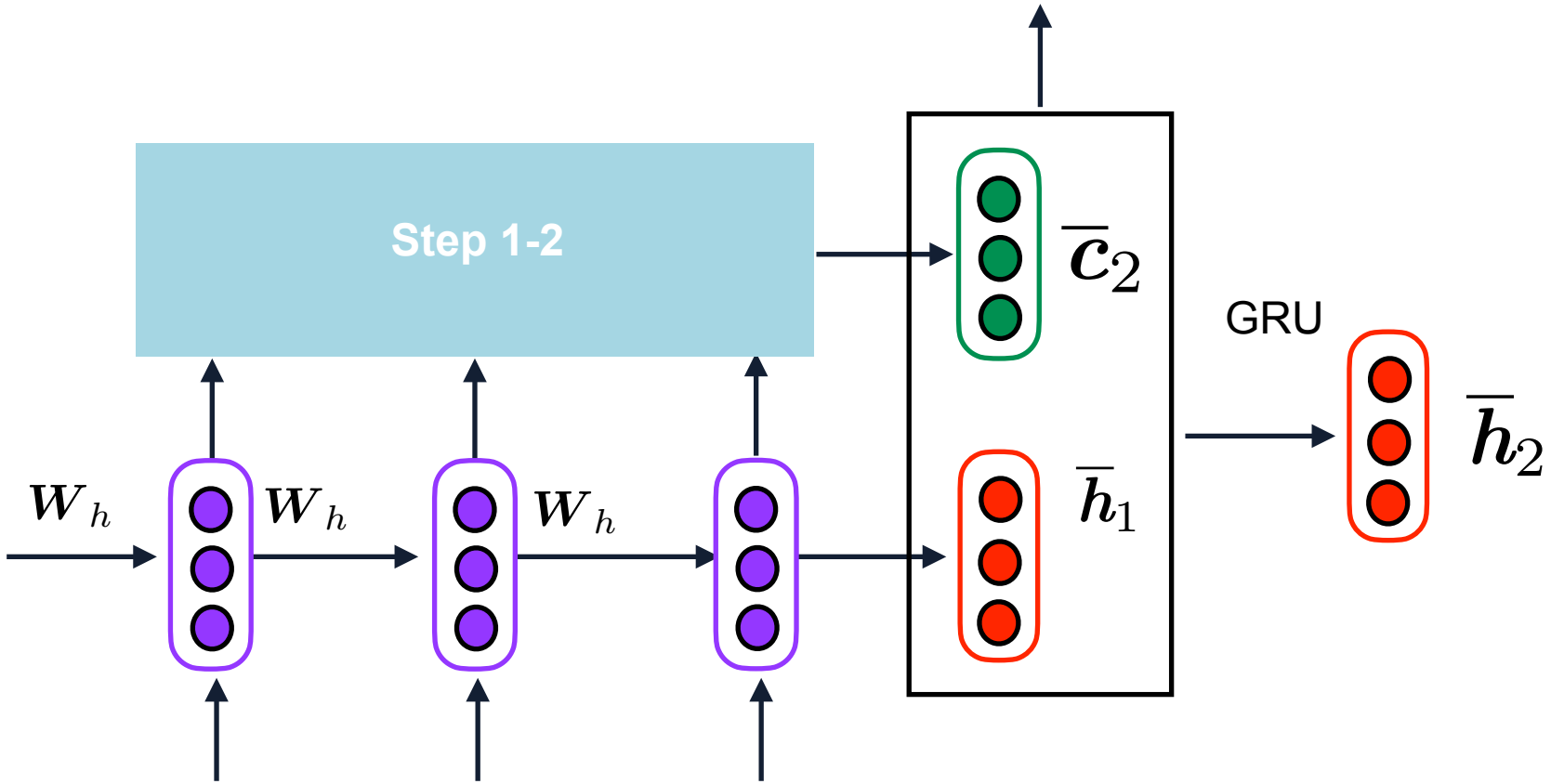
Attention mechanism: RNNSearch

[Bahdanau et al. 2014]



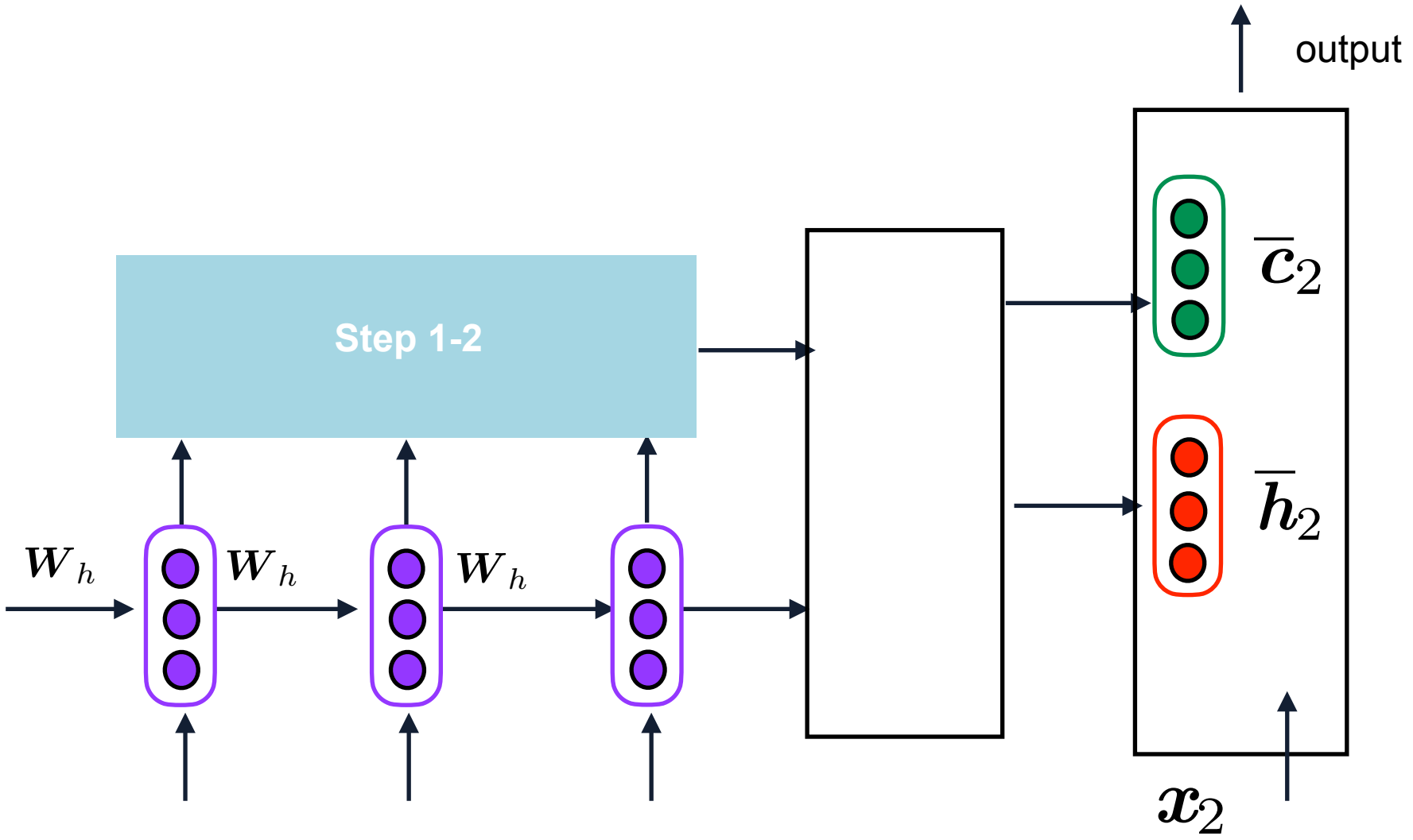
Attention mechanism: RNNSearch

[Bahdanau et al. 2014]



Attention mechanism: RNNSearch

[Bahdanau et al. 2014]



Attention mechanism: Dot-product attention vs RNNSearch

For t = 1,…, T:

$$\text{score}\left(\boldsymbol{h}_t, \boldsymbol{h}_i\right) = \boldsymbol{h}_t^\top \boldsymbol{h}_i$$

$$\boldsymbol{c}_t = \sum_{i=1}^{N} softmax(score(\boldsymbol{h}_i, \boldsymbol{h}_t))\boldsymbol{h}_i \in \mathbb{R}^h$$

$$\tilde{\boldsymbol{h}}_t = \tanh\left(\boldsymbol{W}_{\boldsymbol{c}}\left[\boldsymbol{c}_t; \boldsymbol{h}_t\right]\right)$$

$$p\left(y_t \mid y_{<t}, x\right) = \text{softmax}\left(\boldsymbol{W}_{\boldsymbol{s}}\tilde{\boldsymbol{h}}_t\right)$$

dot product attention

For t = 1,…, T:

$$\overline{h}_{t-1} = GRU(h_{t-1})$$

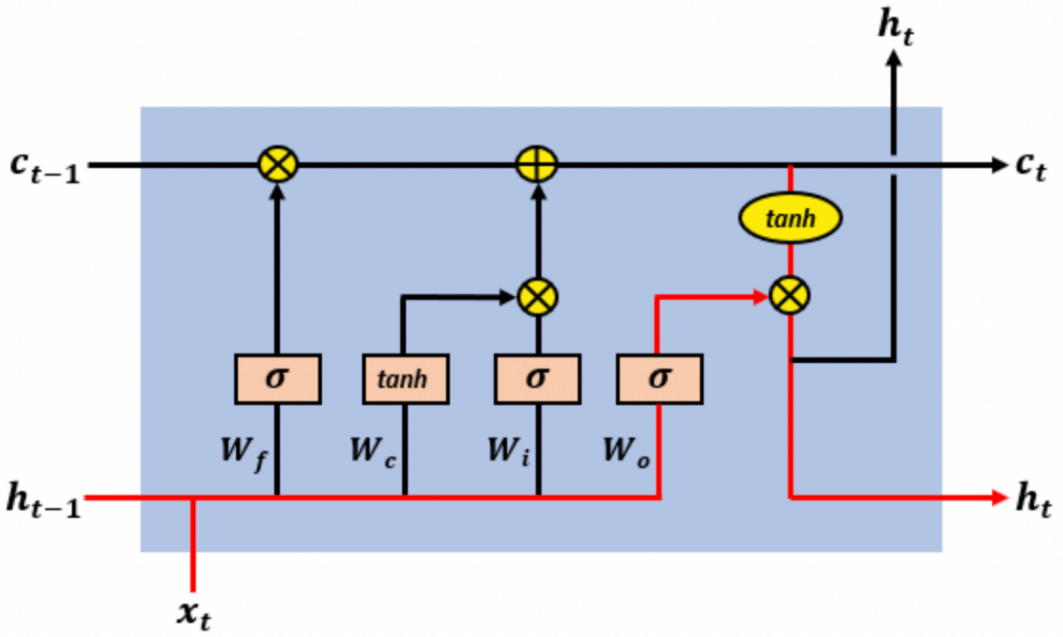
$$\boldsymbol{c}_t = \sum^{N} \text{softmax}\left(\text{score}\left(\boldsymbol{h}_i, \overline{\boldsymbol{h}}_{t-1}\right)\right)\boldsymbol{h}_i \in \mathbb{R}^h$$

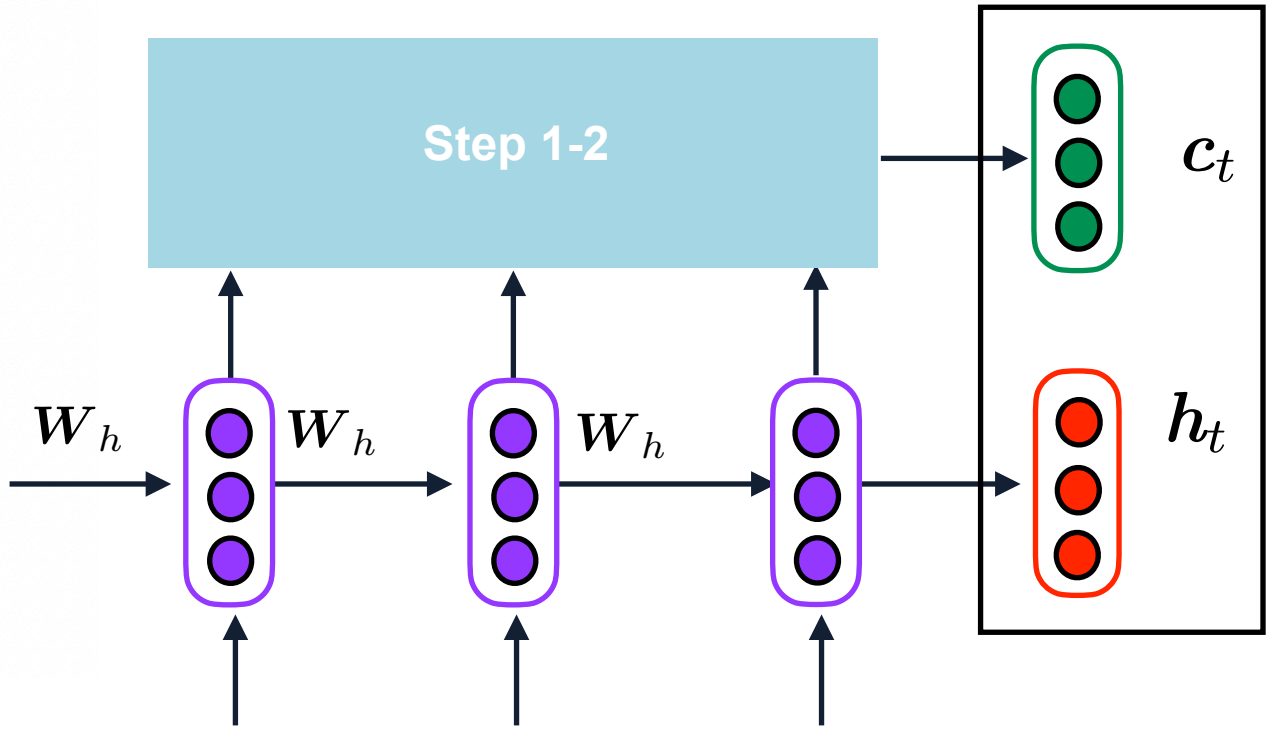
$$h_t = GRU(\overline{h}_{t-1}, c_t)$$

$$p\left(y_t \mid y_{<t}, \mathbf{x}\right) = g\left(y_{t-1}, h_t, c_t\right)$$

RNN search attention

Comparing attention with LSTM



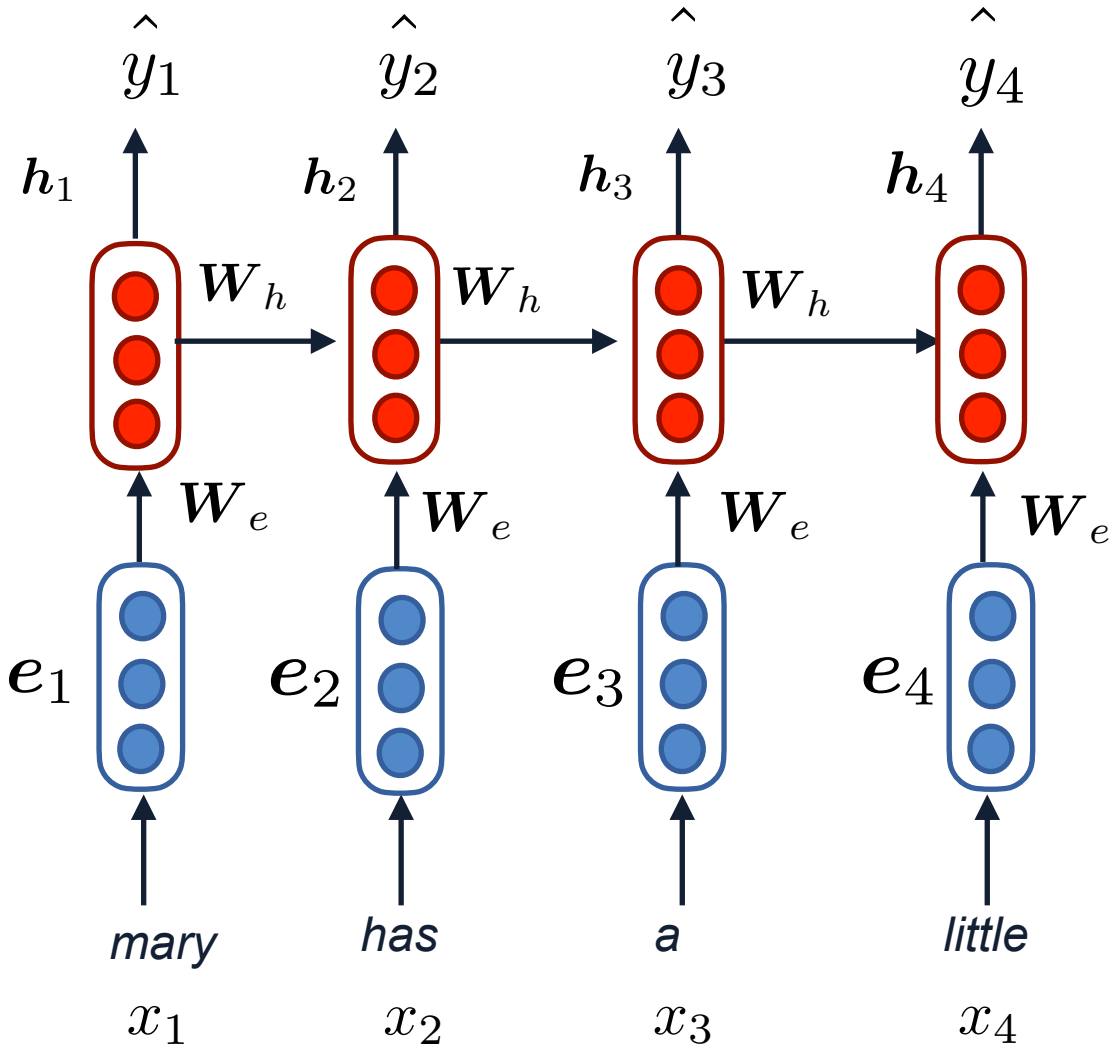


Performance of attention [Luong et al. 2015]

System	Ppl	BLEU
Winning WMT'14 system – *phrase-based* + *large LM* (Buck et al., 2014)		20.7
Existing NMT systems		
RNNsearch (Jean et al., 2015)		16.5
RNNsearch + unk replace (Jean et al., 2015)		19.0
RNNsearch + unk replace + large vocab + *ensemble* 8 models (Jean et al., 2015)		**21.6**
Our NMT systems		
Base	10.6	11.3
Base + reverse	9.9	12.6 (+*1.3*)
Base + reverse + dropout	8.1	14.0 (+*1.4*)
Base + reverse + dropout + global attention (*location*)	7.3	16.8 (+*2.8*)
Base + reverse + dropout + global attention (*location*) + feed input	6.4	18.1 (+*1.3*)
Base + reverse + dropout + local-p attention (*general*) + feed input	5.9	19.0 (+*0.9*)
Base + reverse + dropout + local-p attention (*general*) + feed input + unk replace		20.9 (+*1.9*)
Ensemble 8 models + unk replace		**23.0 (**+*2.1*)

Pre-trained word embeddings

- In RNN, we randomly initialize the weight, use the back propagation to update this weight
- The random initialization may not give us a good starting point
- We can choose a better starting point with embedding vectors **pertained in a large unlabelled corpus to help the initialization** [Mikolov et al. 13]



From pretrained embedding to pretrained encoder

- Pretrained embeddings: fixed vectors

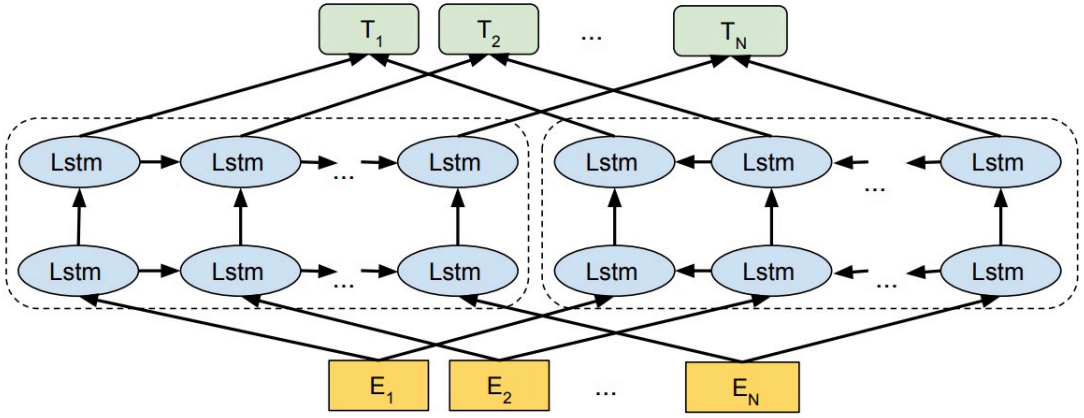
- Pretrained encoder representation:
  - Neural network layers which takes an input sentence and encodes it into a layer output to be fed into another network, the encoder representation can be fine tuned
  - **A pretrained LSTM** on auto encoder & next word prediction can be used as a starting point to improve the performance of downstream supervised LSTM tasks [Dai & Le 2015]
  - ELMo [Peters et al. 2018]

ELMo [Peters et al. 2018]

- Train an L-layered bidirectional LSTM model, i.e., predicting the next word & the previous word
- Collapse all the 2L+1 representations into a single layer

$$\mathbf{ELMo}_k^{task} = E\left(R_k; \Theta^{task}\right) = \gamma^{task} \sum_{i=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}$$

- Concatenate the ELMo weights into task specific models
  - Concatenate with input $[\mathbf{x}_k; \mathbf{ELMo}_k^{task}]$
  - Concatenate with output $[\mathbf{h}_k; \mathbf{ELMo}_k^{task}]$

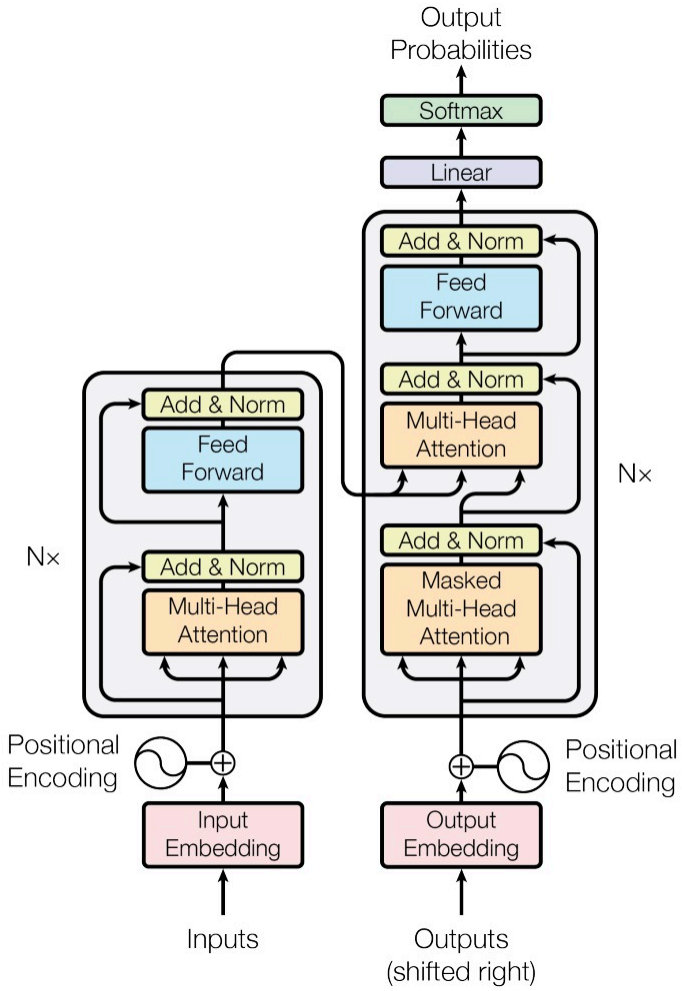


Transformer [2017]

- **Non-recurrent** sequence to sequence encoder-decoder model
- Encoder and decoder architectures
- Multi-head self attention [Lin et al. 2017]

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

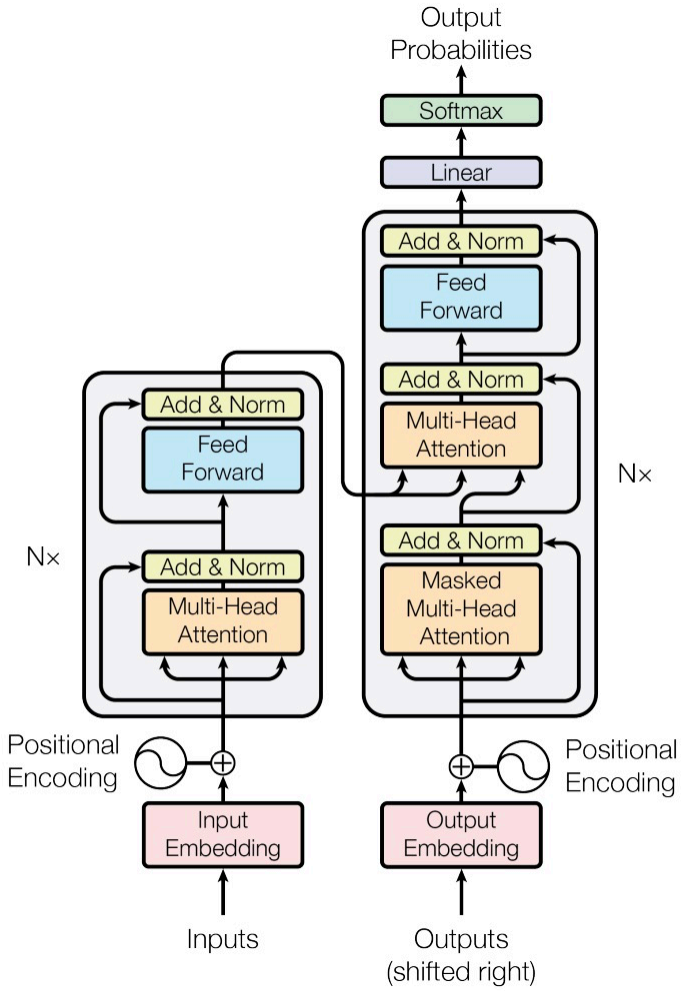
- Position encoding [Gehring et al. 2017]



Encoder/decoder in transformer

- Encoder: 6 identical layers
  - First layer: multi-head self attention
  - Second layer: fully connected feed forward network, followed by layer normalization

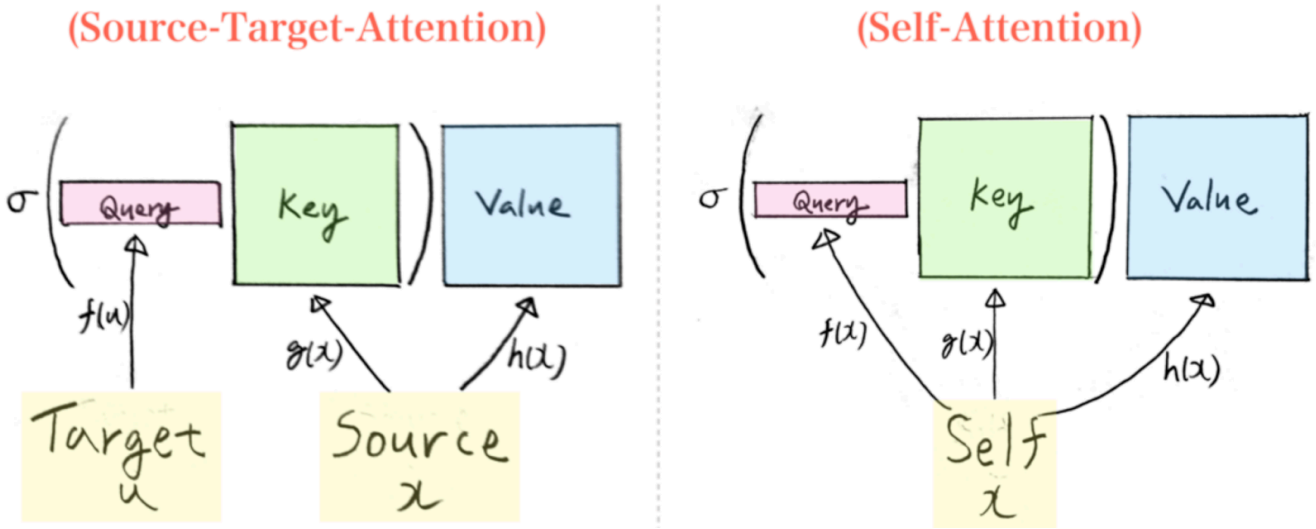
- Decoder: 6 identical layers
  - First & second: multi-head self attention
  - Third layer: fully connected feed forward



Self attention

[Lin et al. 2017]

- What do we mean by self attention?
  - In dot-product attention, target attends to source only
  - Source shall never attend to target (**why?**)
  - In self attention, target can attend to source, **source itself can also attend to source, target can also attend to target, so called internal attention**

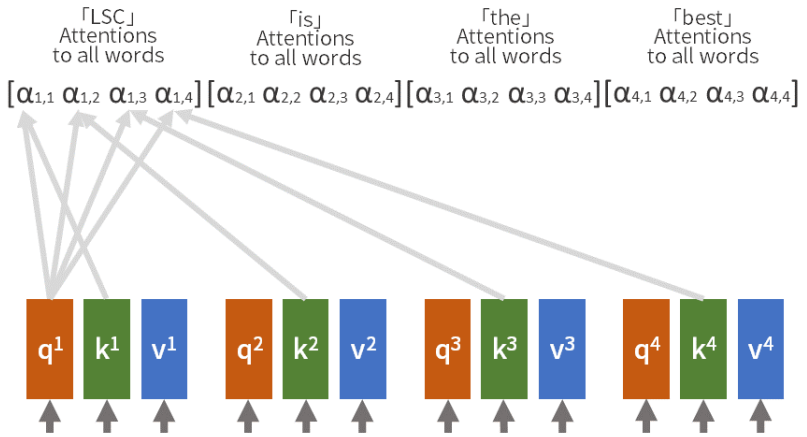


source: https://pretteyandnerdy.wordpress.com/2019/04/26/attention-is-all-you-need/

Self attention

[Lin et al. 2017]

- Step 1: Computing the self attention weights
  - The vector at each position is replicated as 3 vectors: Q, K, V vectors
  - Computing alpha is fast because it can be done using matrix multiplication (whose complexity is O(n^2.37) as of 2020)



$$\alpha_{i,j} = \frac{q^i \cdot k^j}{\sqrt{d}}$$

d: dimension of q, k

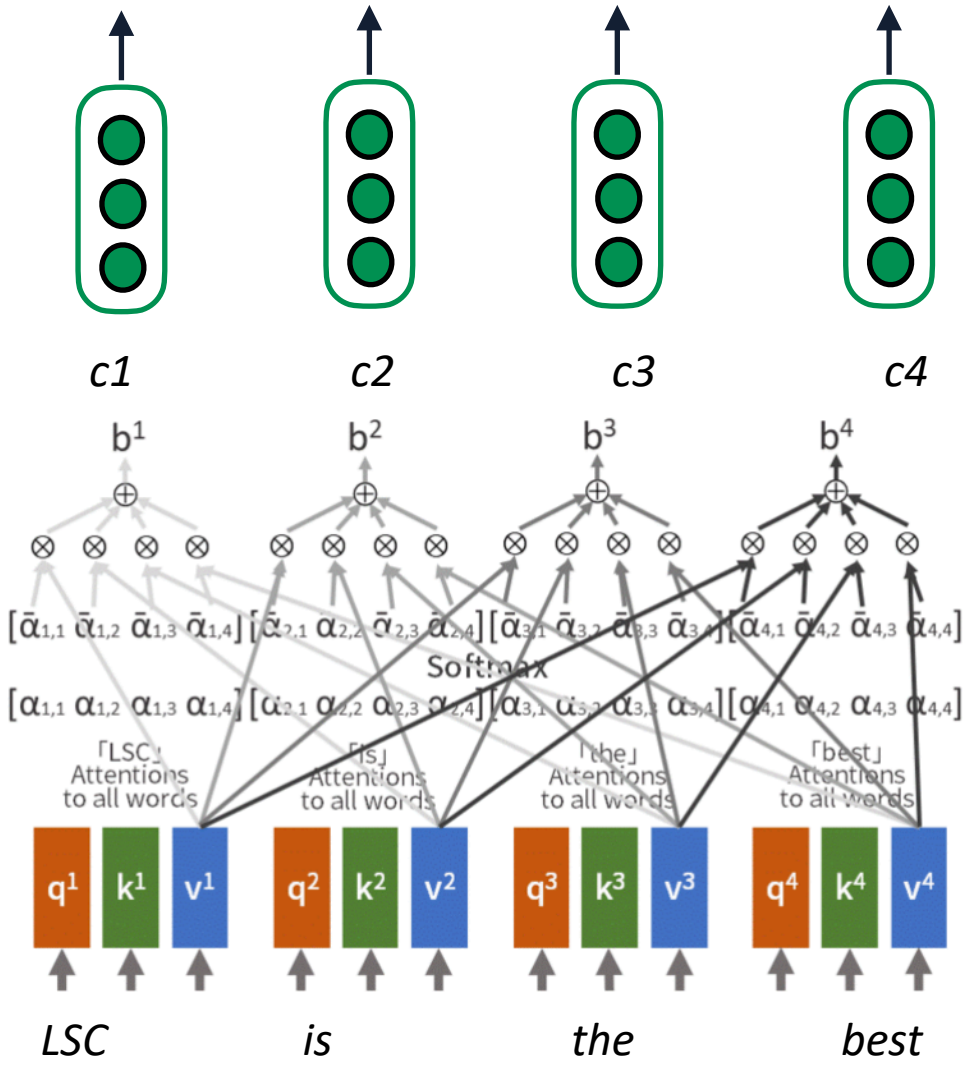
$$A = \begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} & \alpha_{1,4} \\ \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} & \alpha_{2,4} \\ \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} & \alpha_{3,4} \\ \alpha_{4,1} & \alpha_{4,2} & \alpha_{4,3} & \alpha_{4,4} \end{bmatrix}$$

Attention Matrix

Self attention

[Lin et al. 2017]

- Step 2: Computing the context vector ct which will be used for prediction



Attention vs. Self Attention

- Attention is often applied to RNN, whereas self attention does not require RNN (i.e., attention is all you need)
- Attention happens from decoder -> encoder, self attention also looks from encoder to encoder state, and from decoder state to decoder state
- Self attention time cost increase quadratically with the sequence length

Position embedding

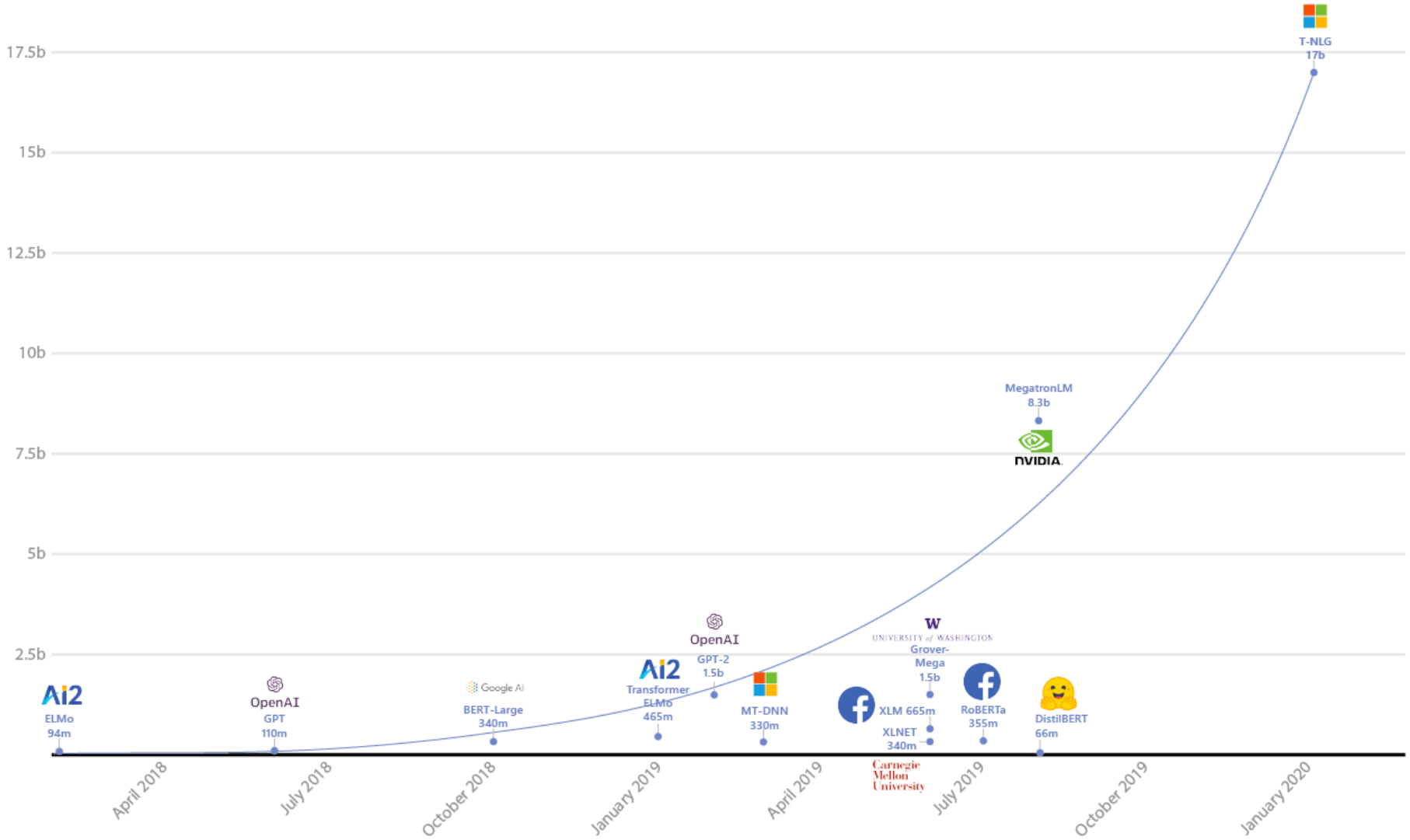
- So far we have not considered the order between words
- Word orders are important for encoding sentence semantics
- How to solve the problem? Position embedding

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

- For example, with 4-dimensional embedding, dmodel = 6:

$$\begin{aligned} e'_w &= e_w + \left[\sin\left(\frac{pos}{10000^0}\right), \cos\left(\frac{pos}{10000^0}\right), \sin\left(\frac{pos}{10000^{2/6}}\right), \cos\left(\frac{pos}{10000^{2/6}}\right)\right] \\ &= e_w + \left[\sin(pos), \cos(pos), \sin\left(\frac{pos}{100}\right), \cos\left(\frac{pos}{100}\right)\right] \end{aligned}$$

Competition of pretrained language models



source: https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/

BERT: Bidirectional Encoder Representations

Pre-training BERT

- **Task #1: Masked LM**

- Mask out k% input words
- Predict the masked words given all its context (“cloze” style)
- Enables representation to fuse the left and right context

BERT: Bidirectional Encoder Representations

Pre-training BERT

- **Task #1: Masked LM**

- Problem: pre-training and fine-tuning mismatch
  - [MASK] will never appear in downstream task training
- Solution: not always use [MASK]
  - 80% use [MASK]
  - 10% use a random token
  - 10% use the original token

BERT: Bidirectional Encoder Representations

Pre-training BERT

- Task #1: Masked LM
- **Task #2: Next Sentence Prediction (NSP)**

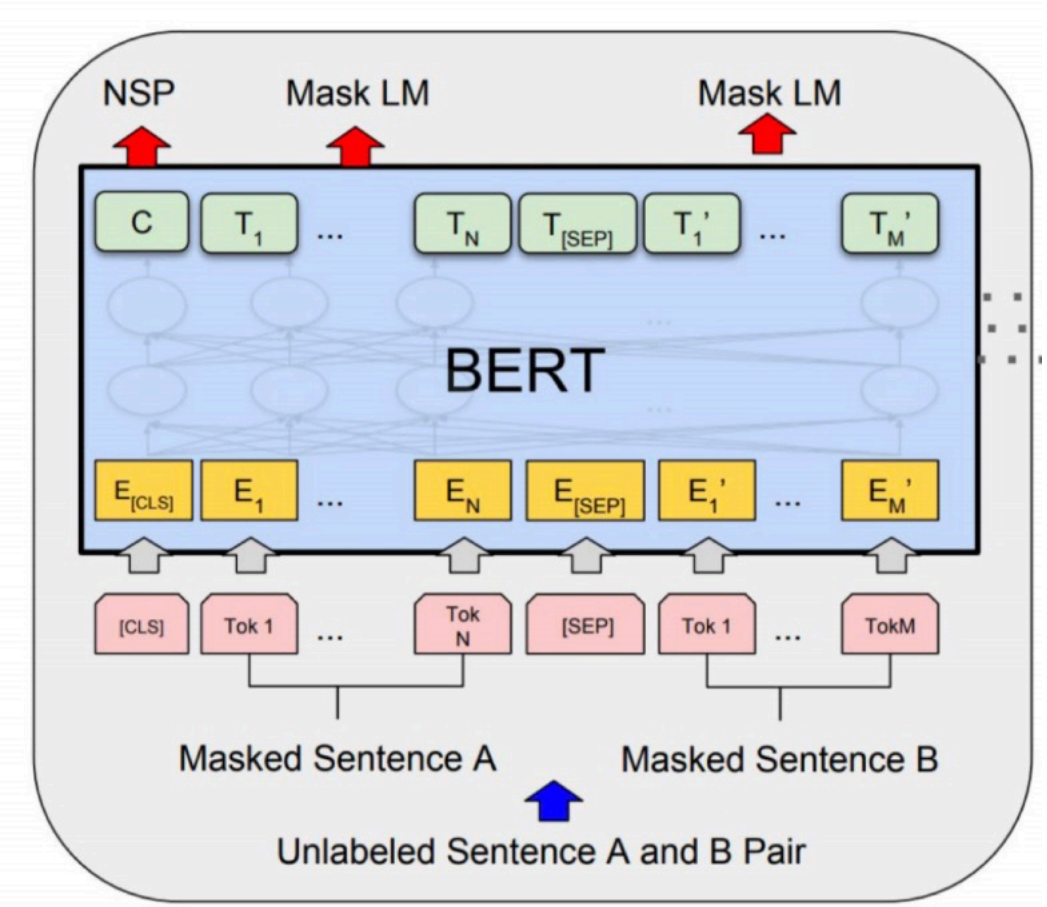
Sentence A = `The man went to the store.`
Sentence B = `He bought a gallon of milk.`
Label = `IsNextSentence`

- Predict whether B is the actual sentence that proceeds A (True / False)
- To learn relationship between sentences

BERT: Bidirectional Encoder Representations

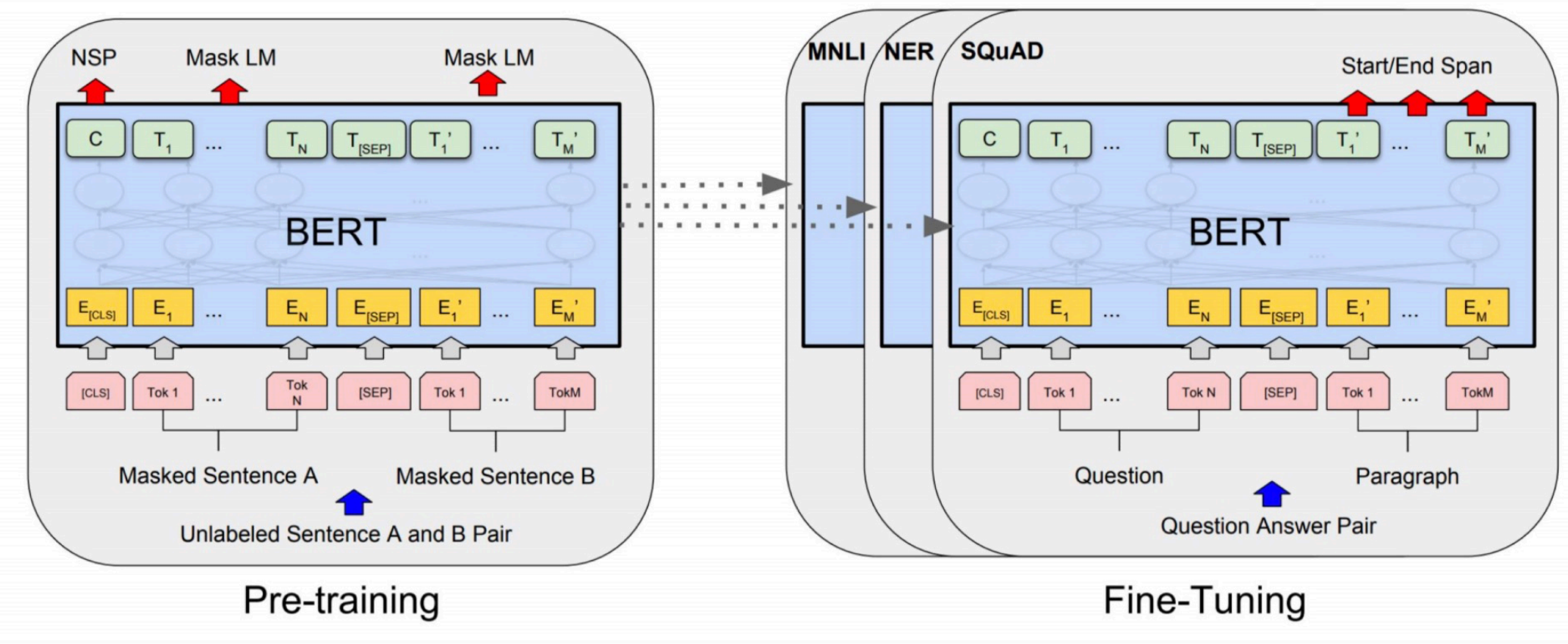
Pre-training BERT

- Input: unlabeled sentence pair
- Training
  - Masked LM
  - NSP (label position [CLS])



BERT: Bidirectional Encoder Representations

Fine-tuning BERT



BERT experimental results

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

MultiNLI

Premise: Hills and mountains are especially sanctified in Jainism.
Hypothesis: Jainism hates nature.
Label: Contradiction

CoLa

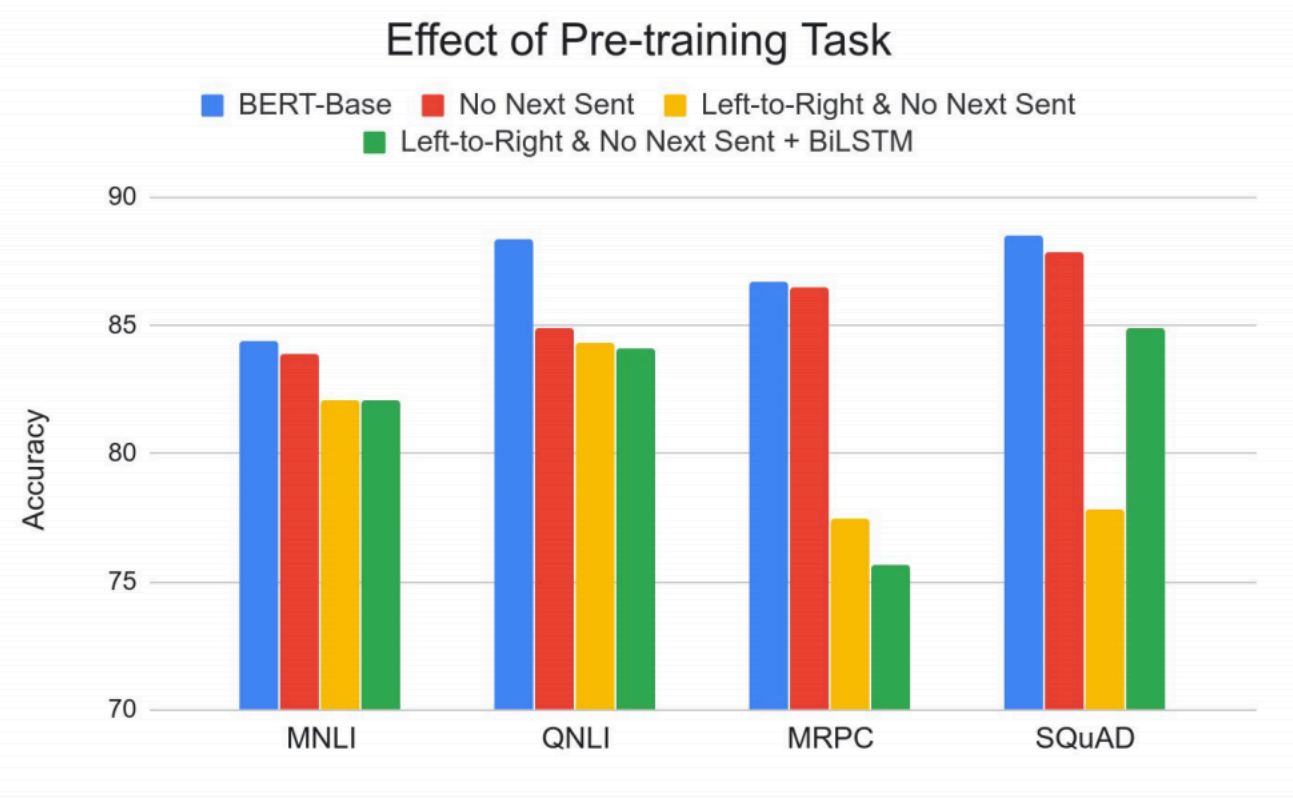
Sentence: The wagon rumbled down the road.
Label: Acceptable

Sentence: The car honked down the road.
Label: Unacceptable

Effects of pretraining tasks

- Masked LM (compared to left-to-right LM) is very important on some tasks, Next Sentence Prediction is important on other tasks

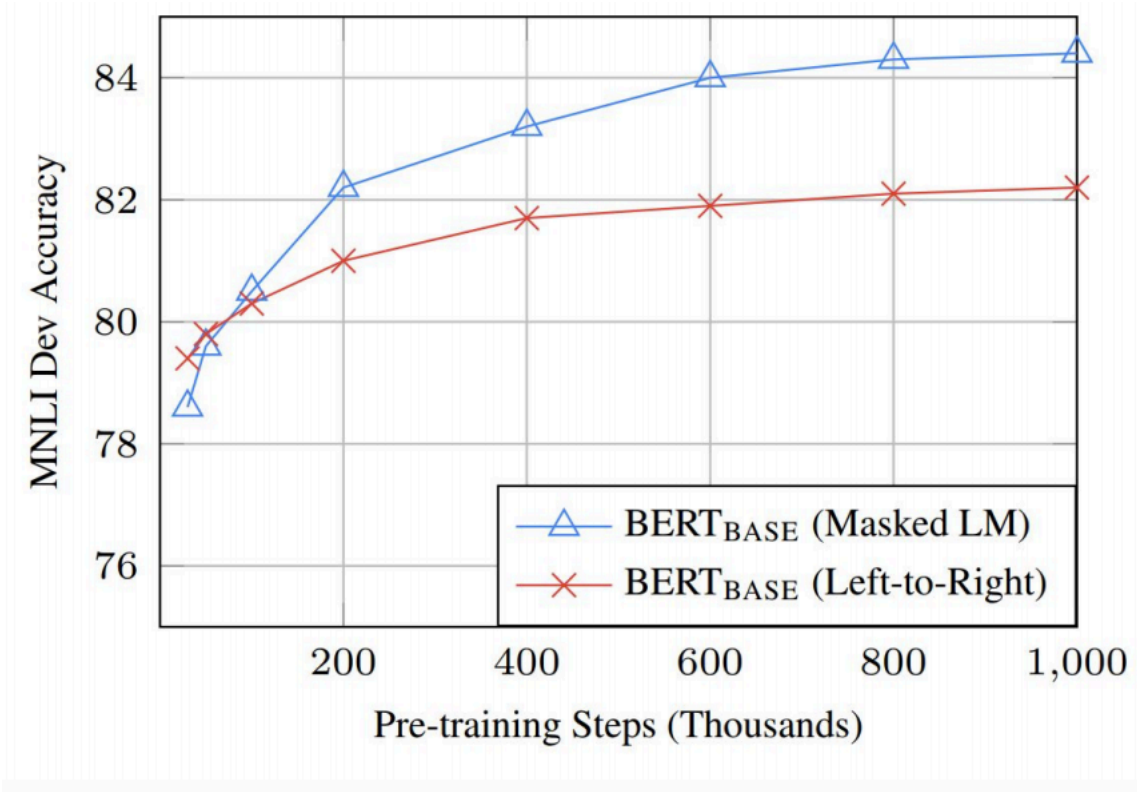
- Left-to-right model does very poorly on word-level task (SQuAD), although this is mitigated by BiLSTM



Effects of dimensionality and training time

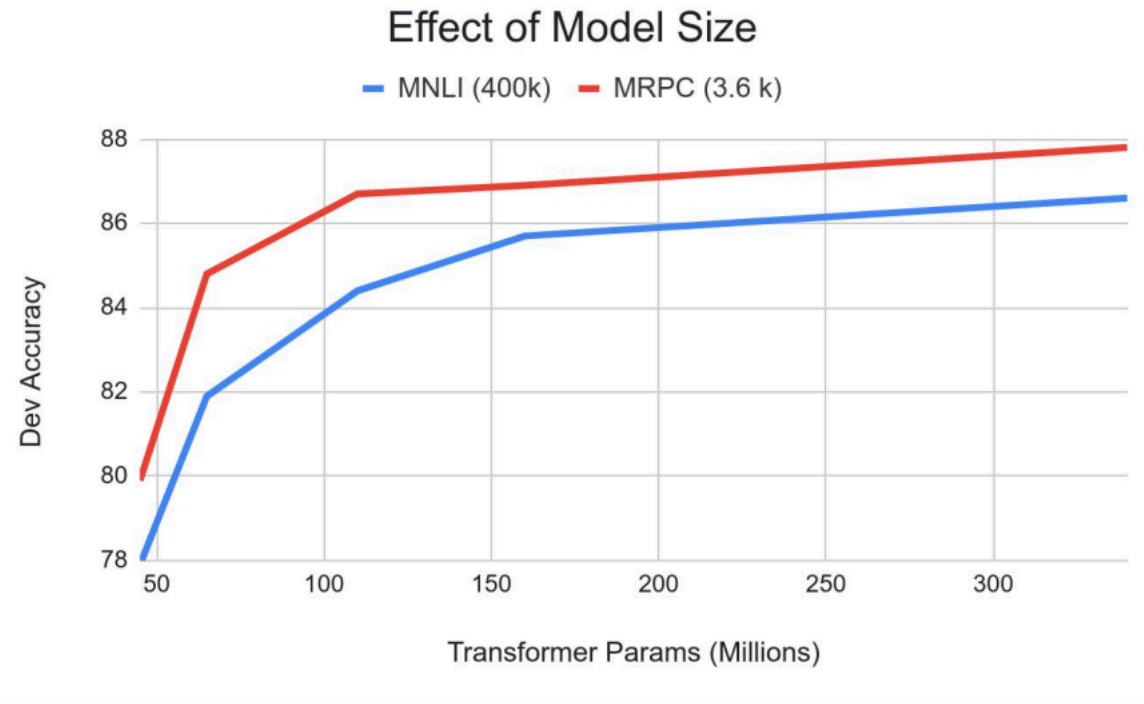
- Masked LM takes slightly longer to converge because we only predict 15% instead of 100%

- But absolute results are much better almost immediately



Effects of dimensionality and training time

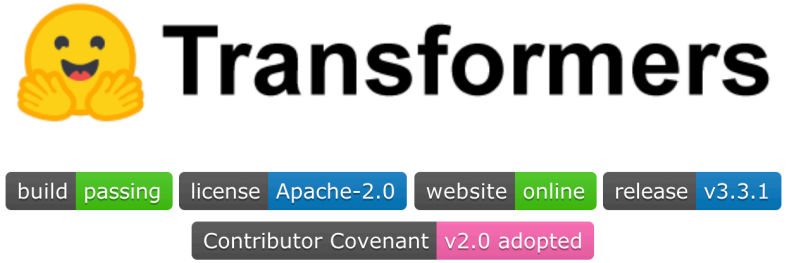
- Big models help a lot
- Going from 110M -> 340M params helps even on datasets with 3,600 labeled examples
- Improvements have not asymptoted



BERT: Open source release

One reason for BERT's success was the open source release

- Minimum dependency
- Abstracted so people could including a single file to use model
- Thorough README
- Idiomatic code
- Well-documented code
- Good support (for the first few months)



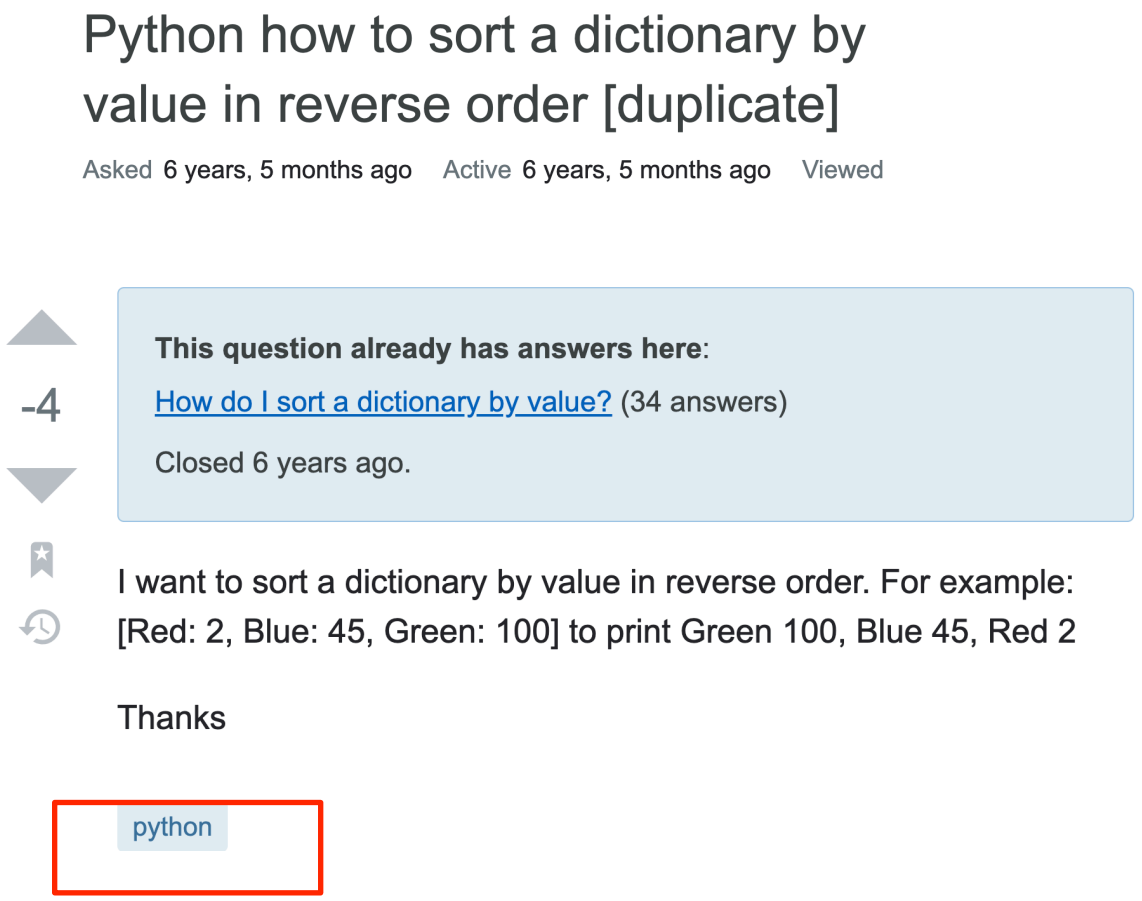

```
from transformers import
AutoTokenizer,
AutoModelForMaskedLM

tokenizer =
AutoTokenizer.from_pretrain
ed("bert-base-uncased")

model =
AutoModelForMaskedLM.from_p
retrained("bert-base-
uncased")
```

Homework 4

- Using Google colab to train a text classification model
  - Supports keras, tensor flow and pytorch
  - Free Tesla K80 GPU
- Homework 4: StackOverflow programming language prediction using hugging face's transformer library



Python how to sort a dictionary by value in reverse order [duplicate]

Asked 6 years, 5 months ago Active 6 years, 5 months ago Viewed

-4

This question already has answers here:

How do I sort a dictionary by value? (34 answers)

Closed 6 years ago.

I want to sort a dictionary by value in reverse order. For example: [Red: 2, Blue: 45, Green: 100] to print Green 100, Blue 45, Red 2

Thanks

python