CS 589 Fall 2020

Attention mechanism

Encoder representation models

Instructor: Susan Liu
TA: Huihui Liu

Stevens Institute of Technology

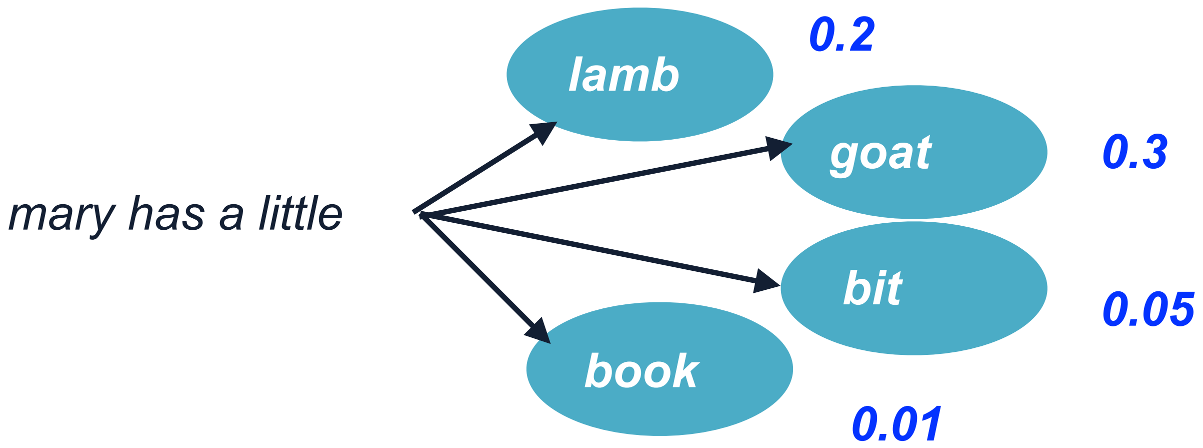
Most slides come from Stanford CS276

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 $V = \{w_1, \cdots, w_{|V|}\}$

Review of language models

- You can train an RNN-LM on any kind of text, then generate in that style

- RNN trained on the first 4 Harry Porter books:

> "Sorry," Harry shouted, panicking
> — "I'll leave those brooms in London, are they?"
> "No idea," said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry's shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn't felt it seemed. He reached the teams too.

source: https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6

Machine translation

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

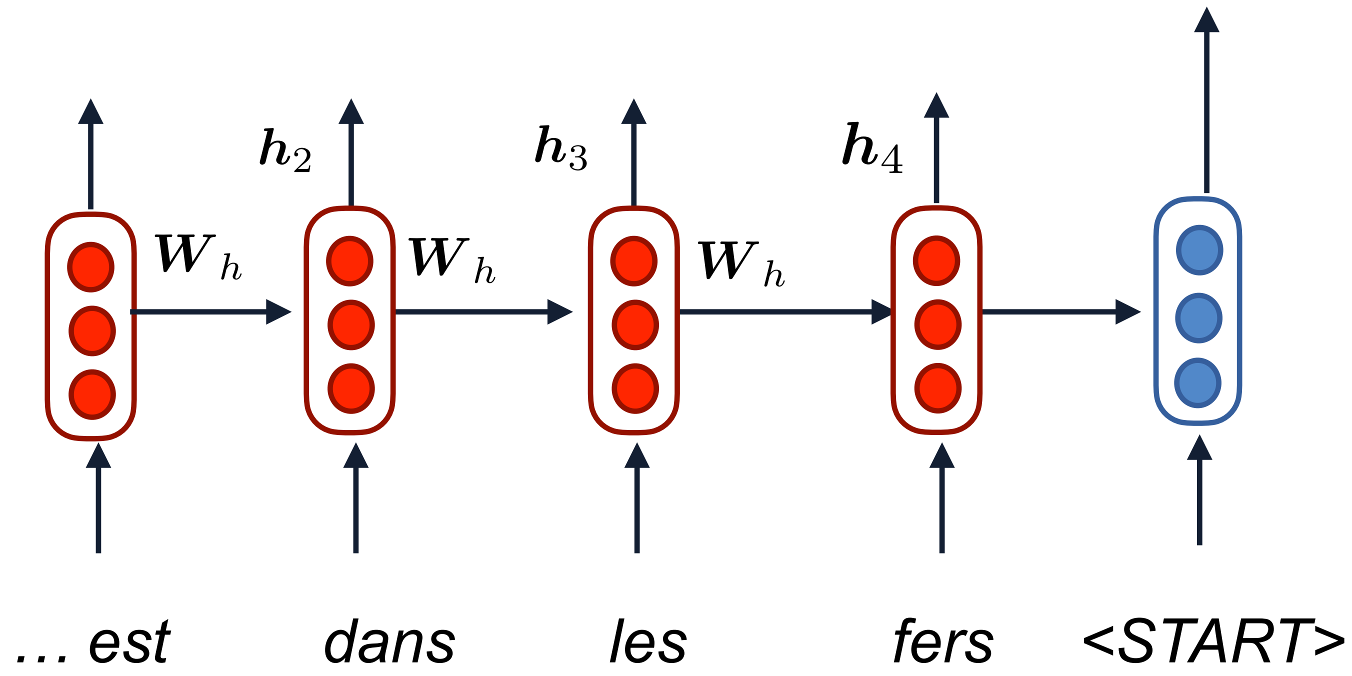
x: *L'homme est né libre, et partout il est dans les fers*

↓

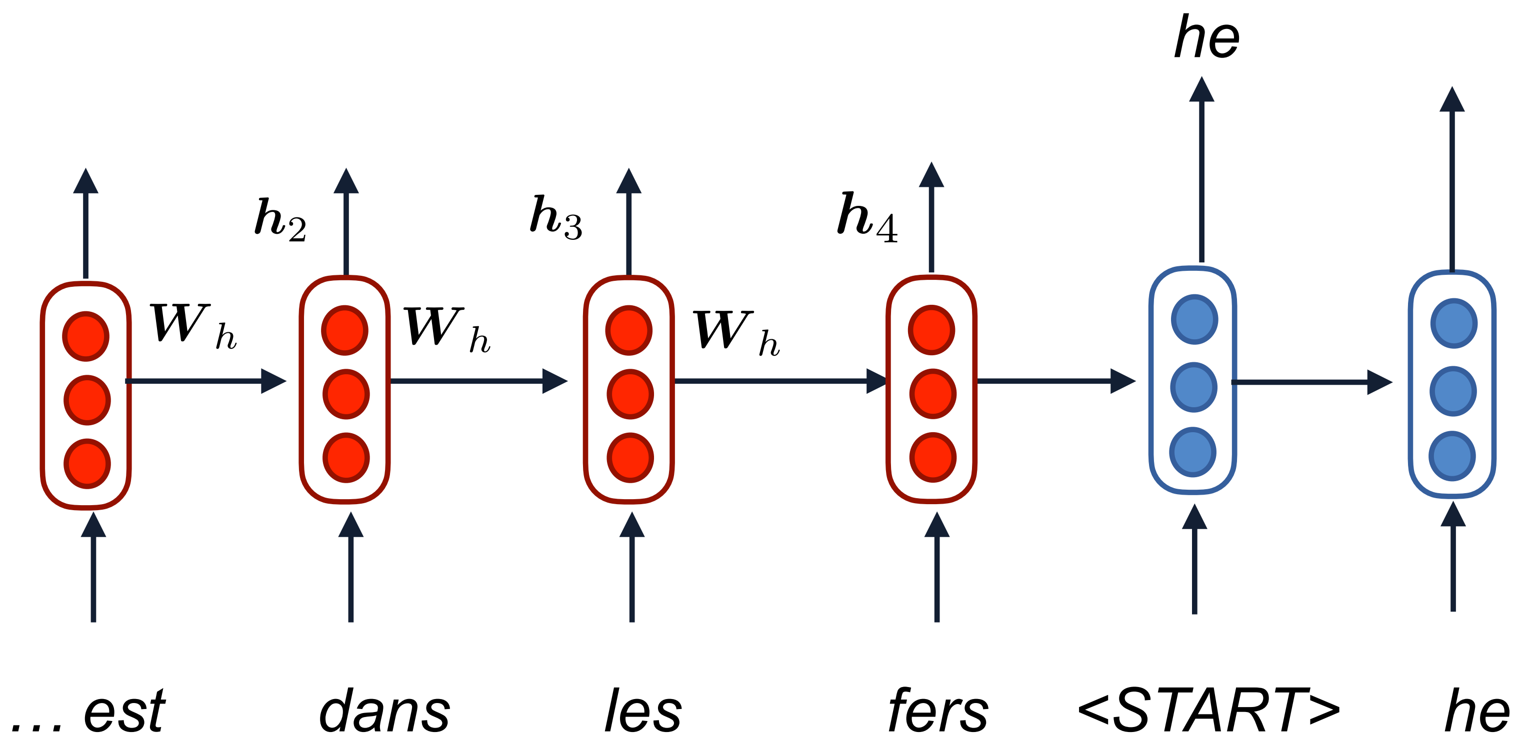
y: *Man is born free, but everywhere he is in chains*

source: https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6

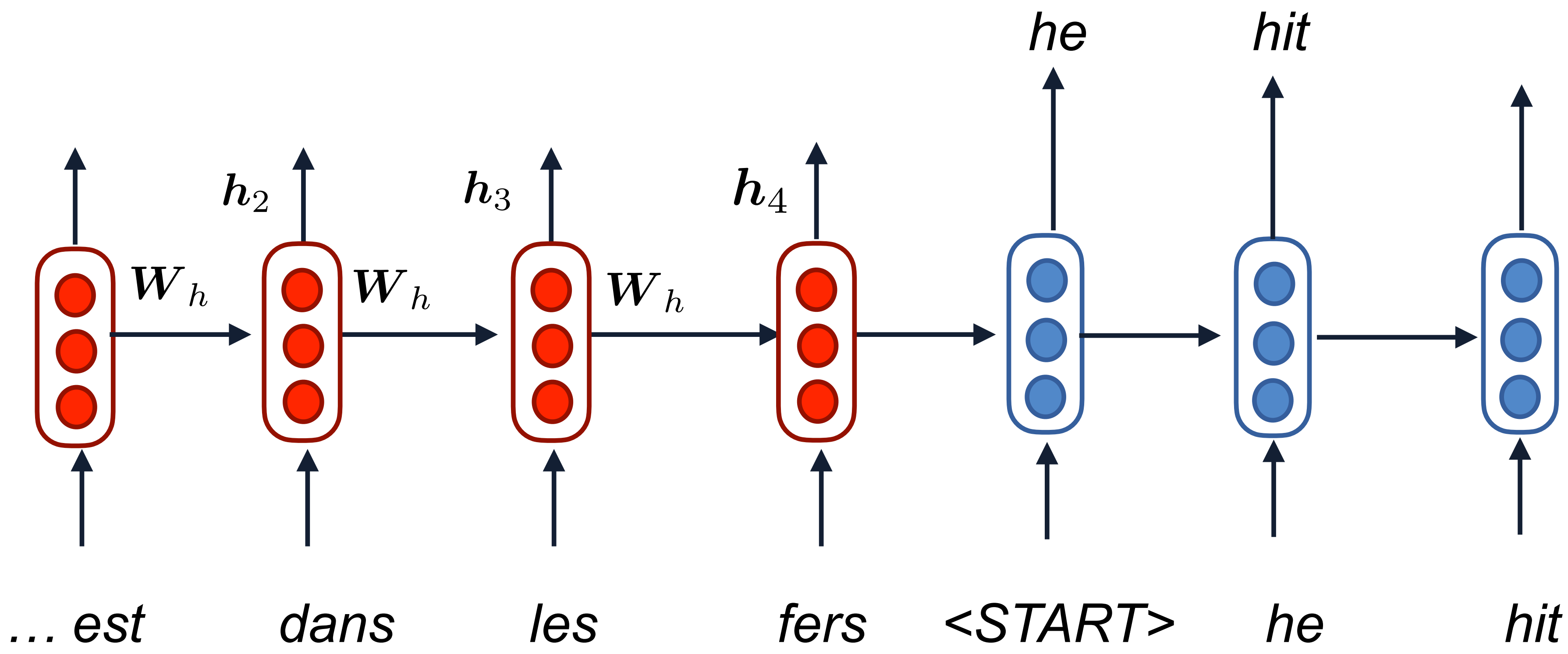
Neural machine translation



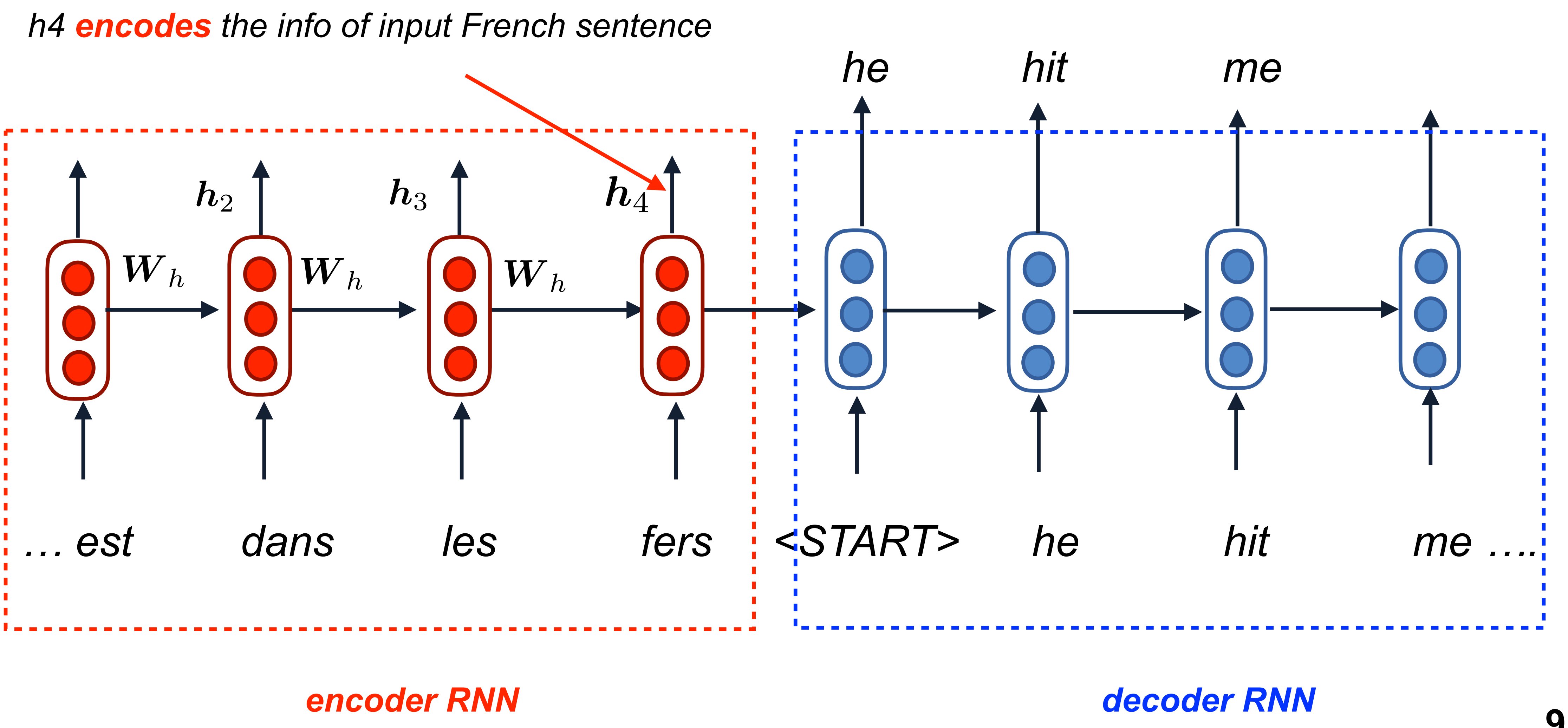
Neural machine translation



Neural machine translation



Neural machine translation



Neural machine translation

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

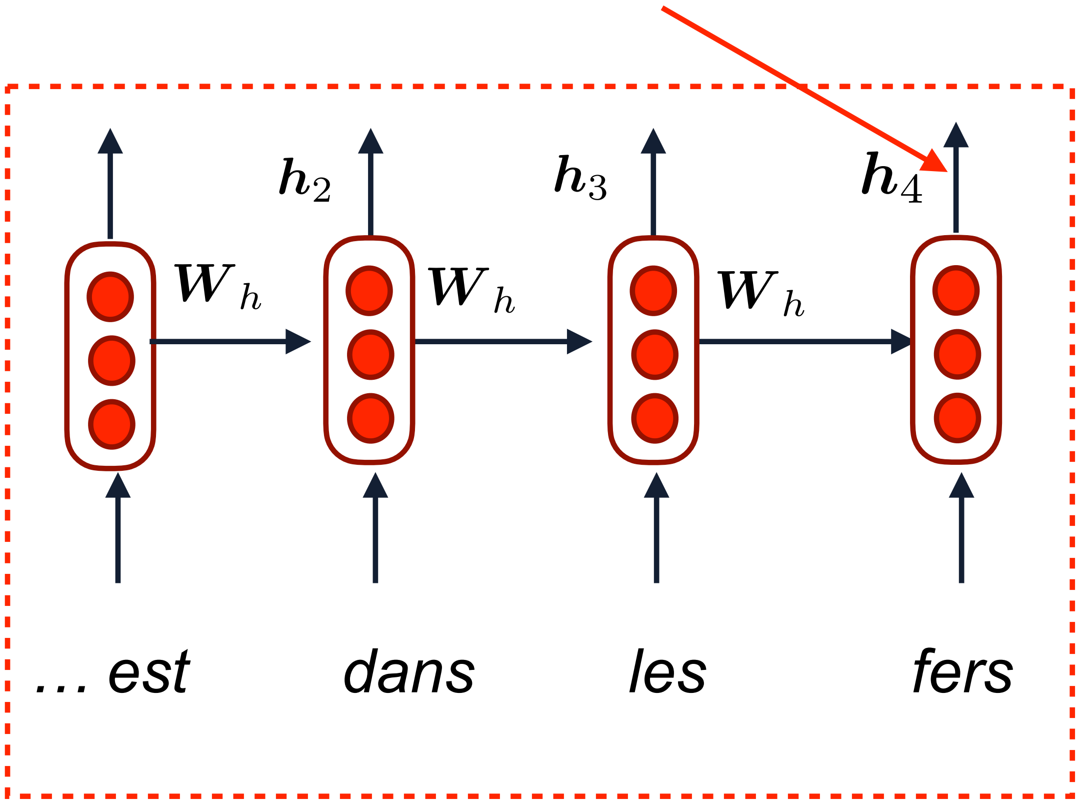
$$P(y \mid x) = P(y_1 \mid x) P(y_2 \mid y_1, x) P(y_3 \mid y_1, y_2, x) \ldots P(y_T \mid y_1, \ldots, y_{T-1}, x)$$

train: $$max_y \sum_{(x^i, y^i)} \log P(y_i|x_i)$$

predict: $$max_y P(y|x)$$

Encoder

h4 ***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

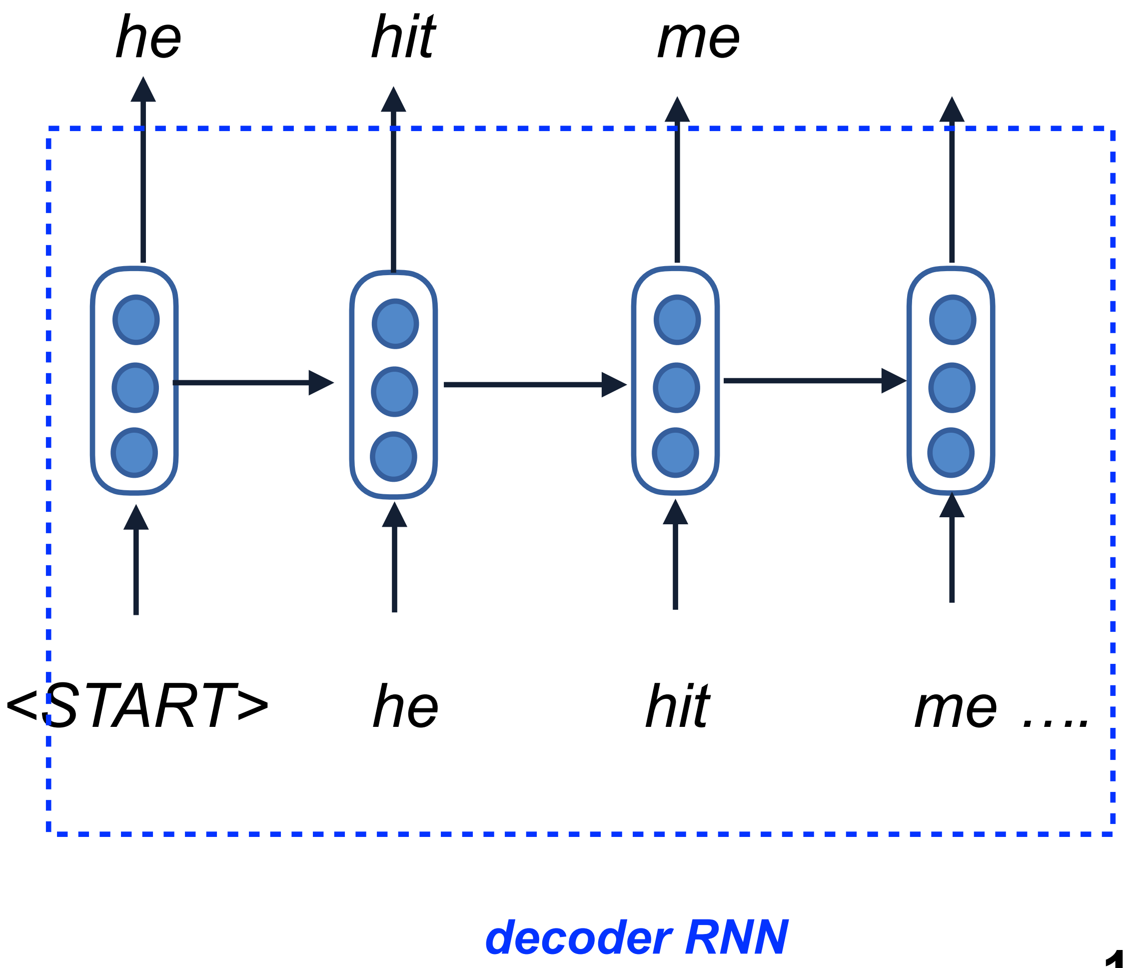
- Decoder selects the word to generate in the target sentence

- Greedy decoding

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

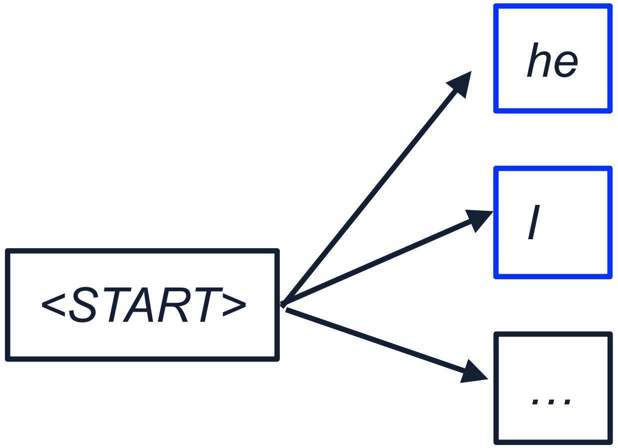
- Disadvantage of greedy decoding: cannot undo decisions

- 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**)



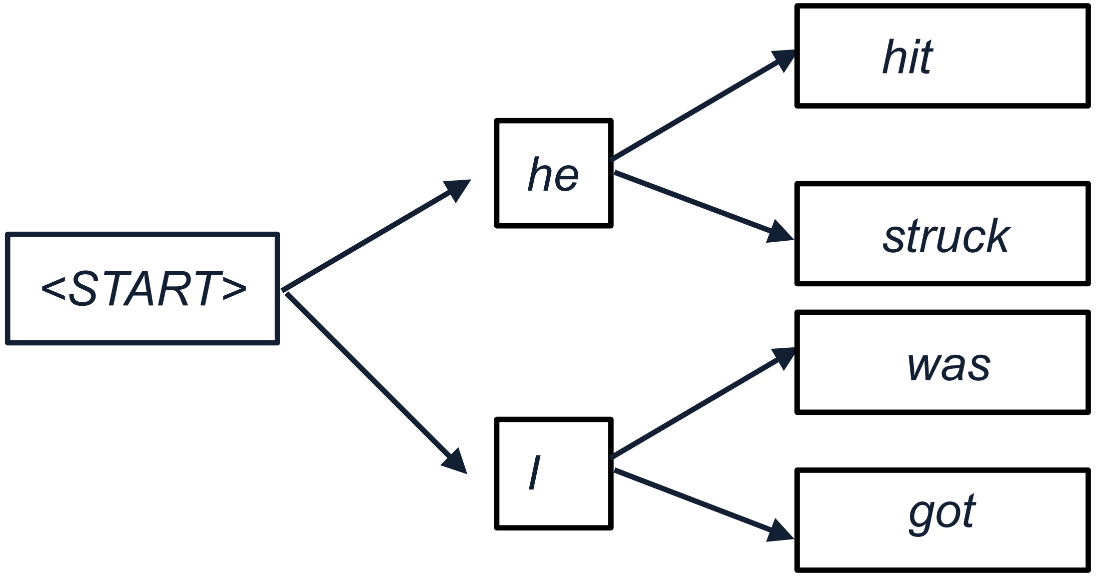
$$-0.7 = \log P_{\text{LM}}\left(he \mid < START >\right)$$

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

Beam search decoding

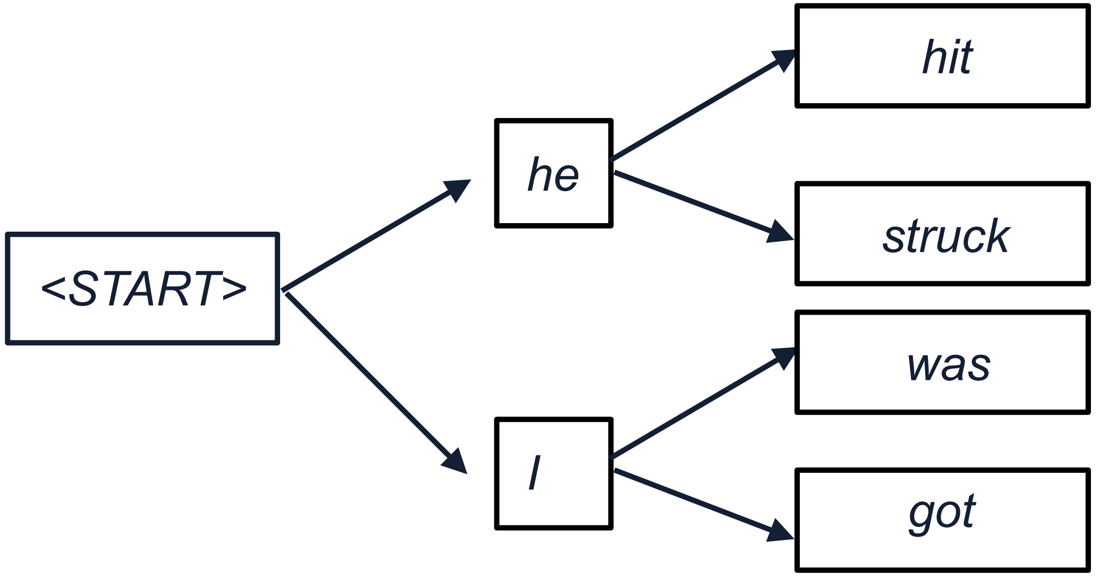
- A trade-off between greedy decoding + exhaustive search

- 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**)



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

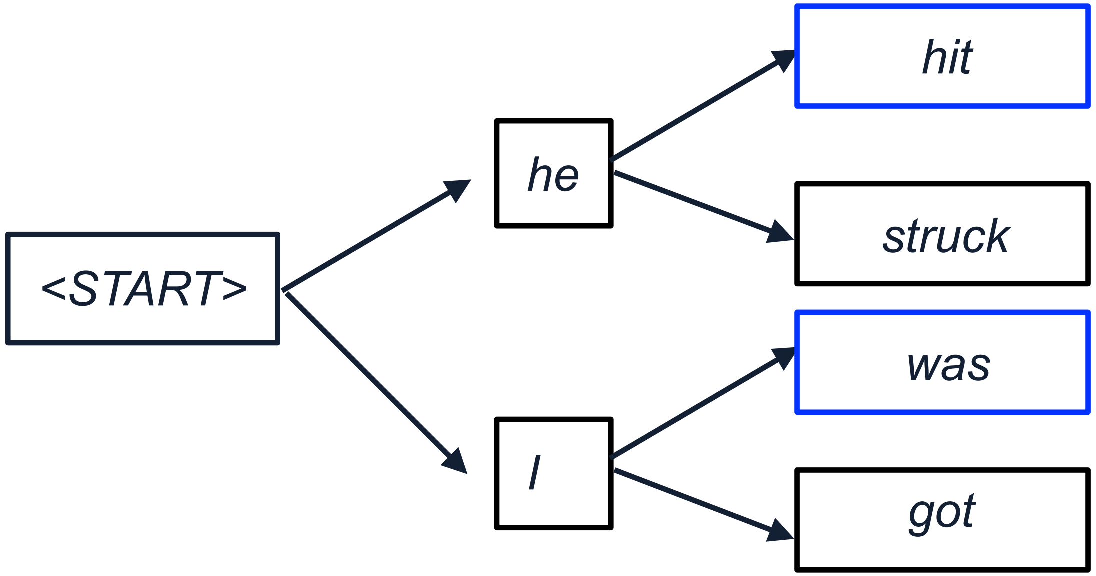
$$-2.9 = \log P_{\mathrm{LM}}\ (struck\ | < START > he) + -0.7$$

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

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

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**)



$$-1.7 = \log P_{\text{LM}}\ (hit\ |< START > he) + -0.7$$

$$-2.9 = \log P_{\text{LM}}\ (struck\ |< START > he) + -0.7$$

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

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

Beam search decoding: Stopping criterion

- In sequence to sequence, 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

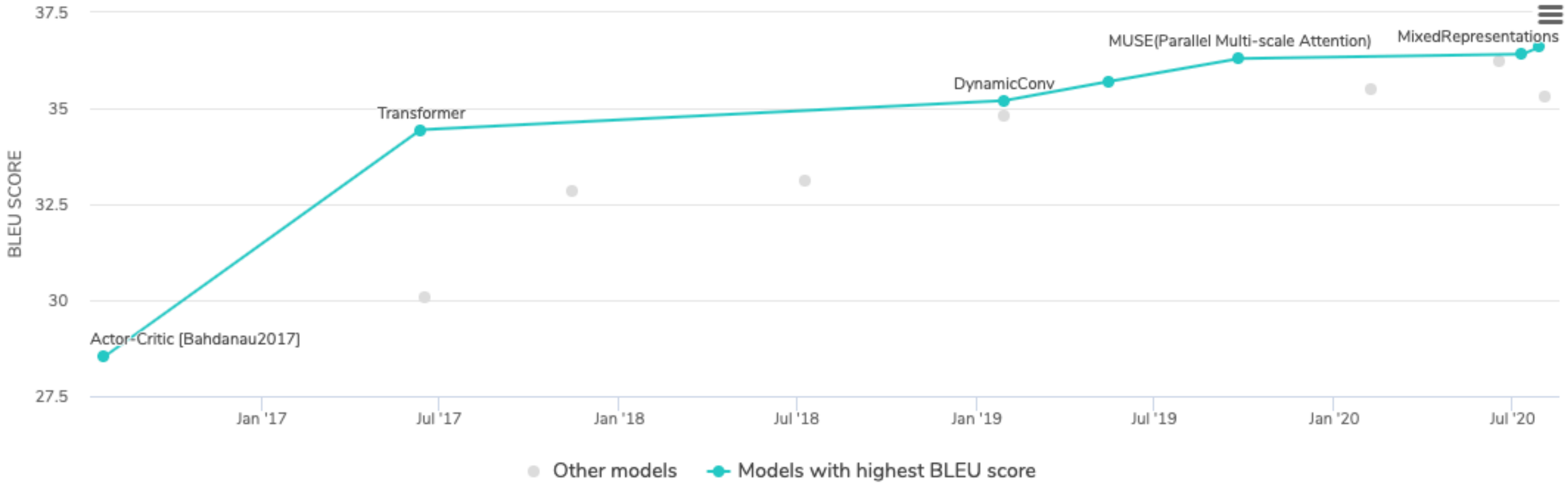
- Each hypothesis has a log probability score
- Longer hypothesis always have lower scores
- Solution: normalize hypotheses by length

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

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

Machine translation timeline



source: https://paperswithcode.com/sota/machine-translation-on-iwslt2014-german

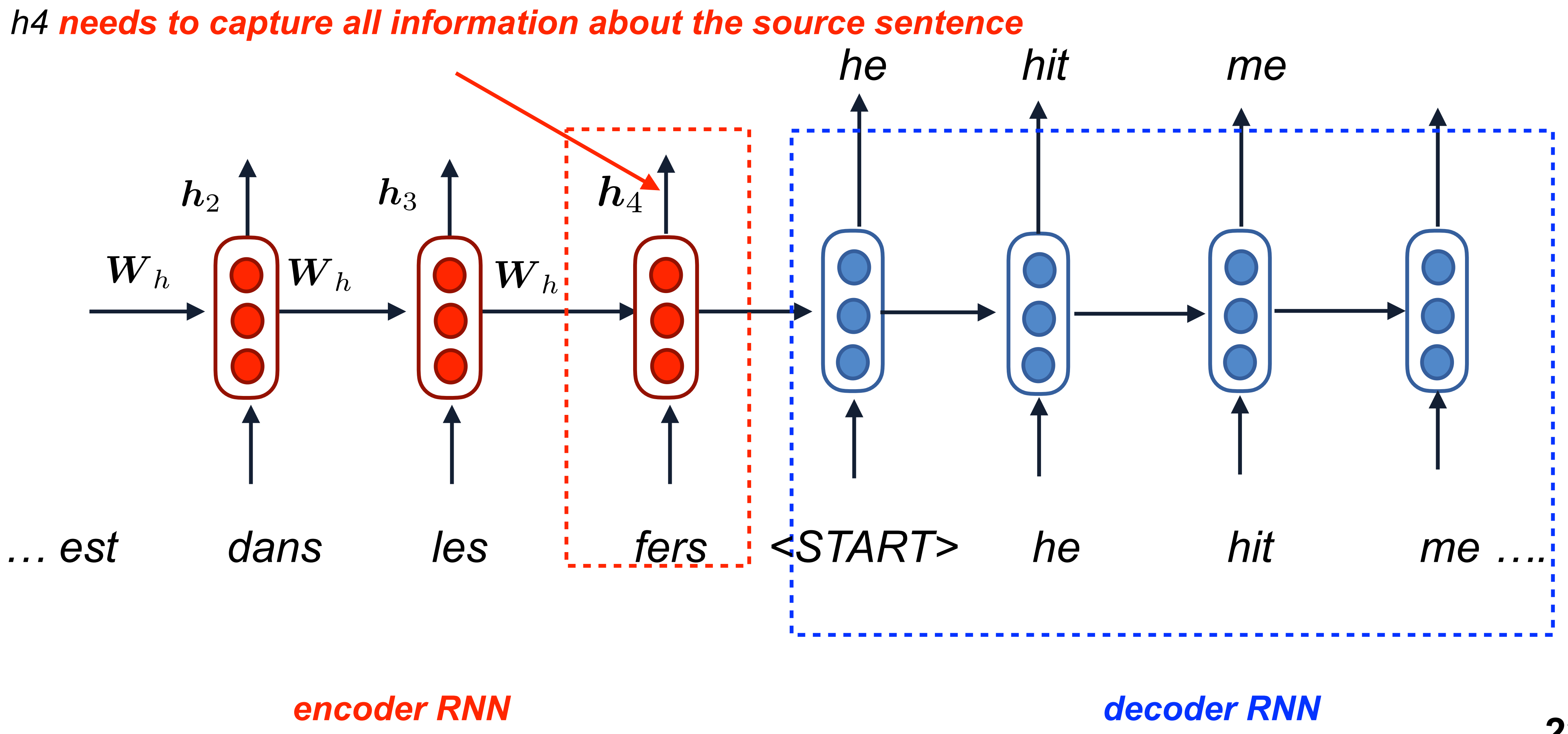
Evaluating machine translation

- How to evaluate the correctness of a translated sentence?
  - We need to compare the similarity between two sentences
  - The two sentences may be semantically equivalent yet contain different tokens

- 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

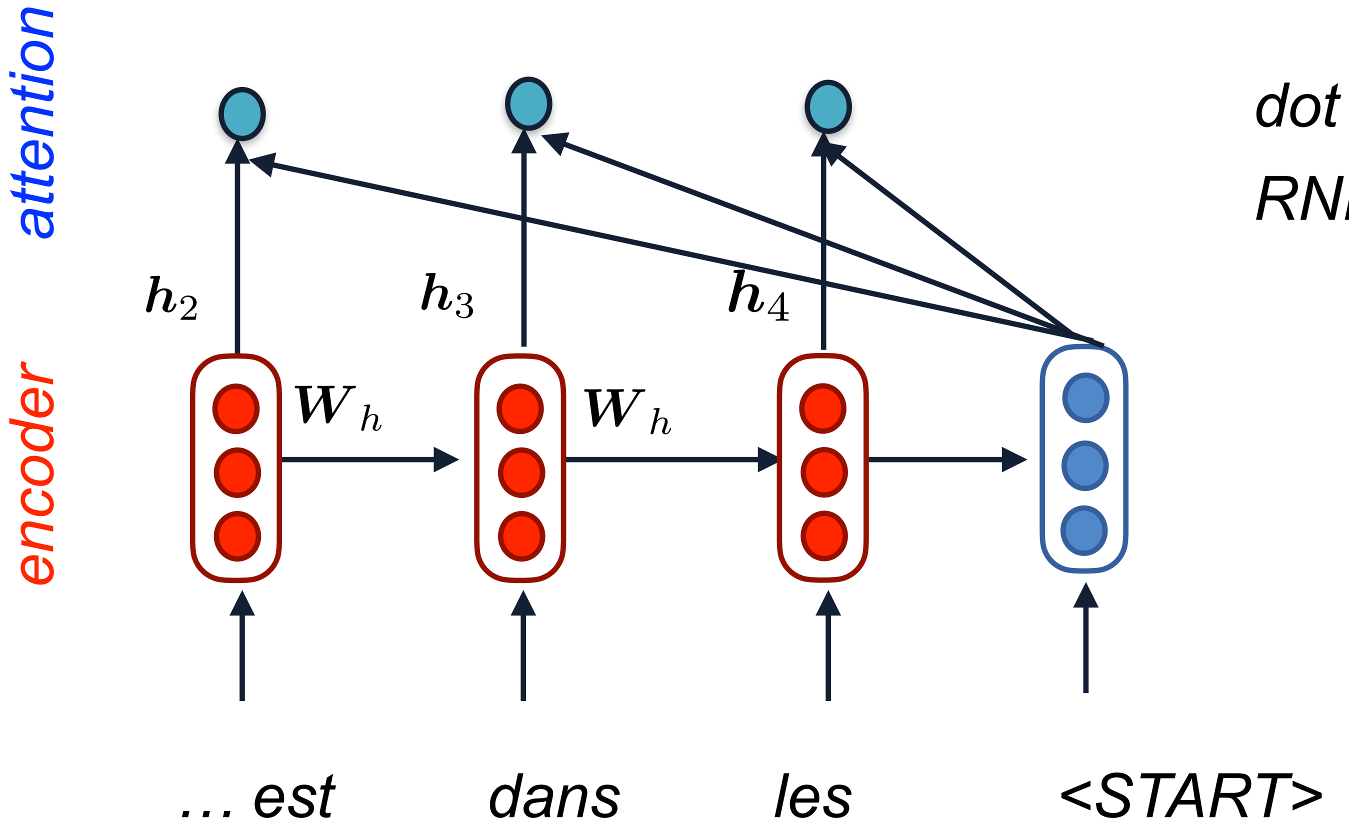
- BLEU is useful but imperfect (non-overlapping ngrams)

Attention mechanism



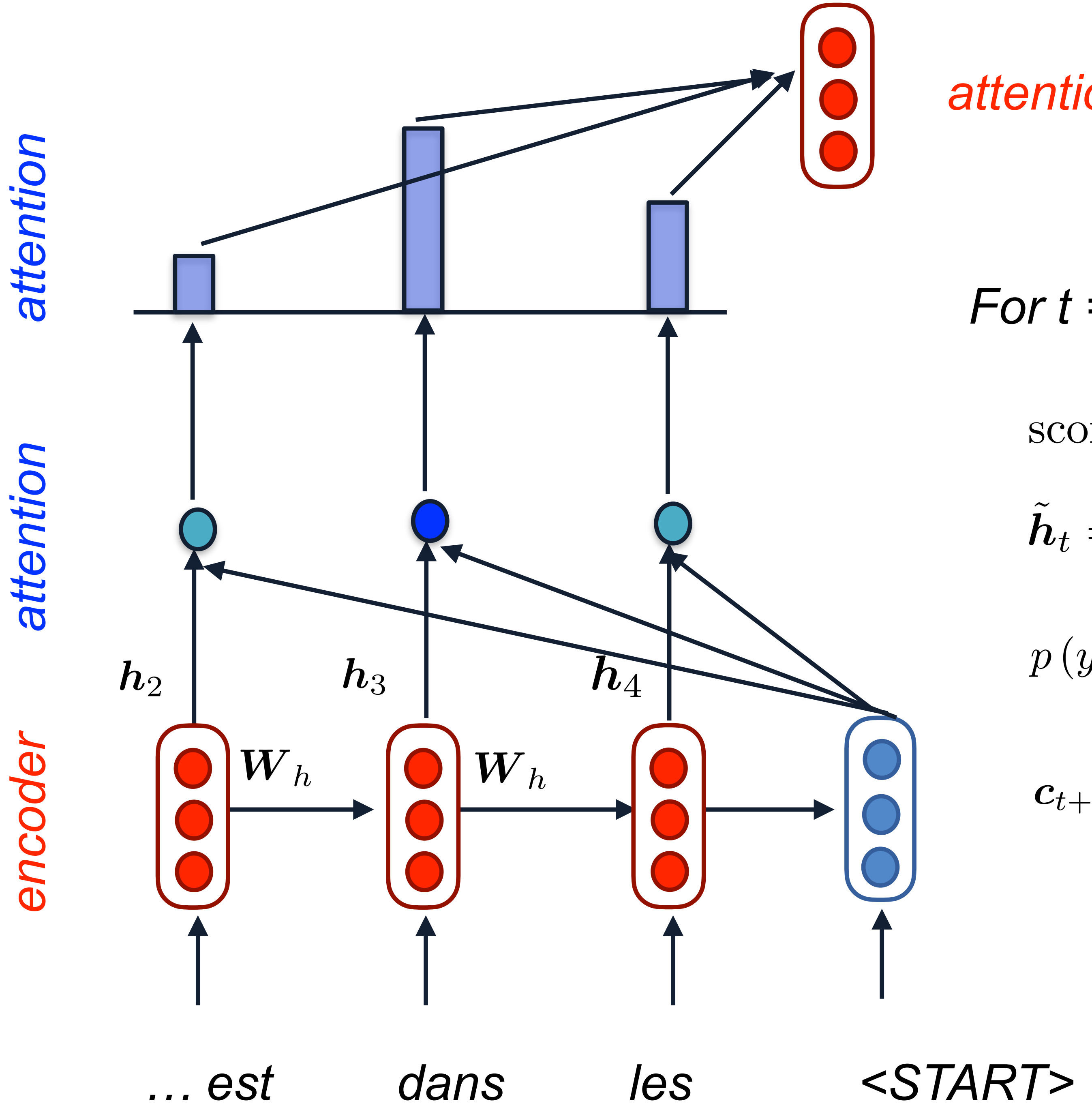
Attention mechanism

- 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]



For t = 1,…, T:

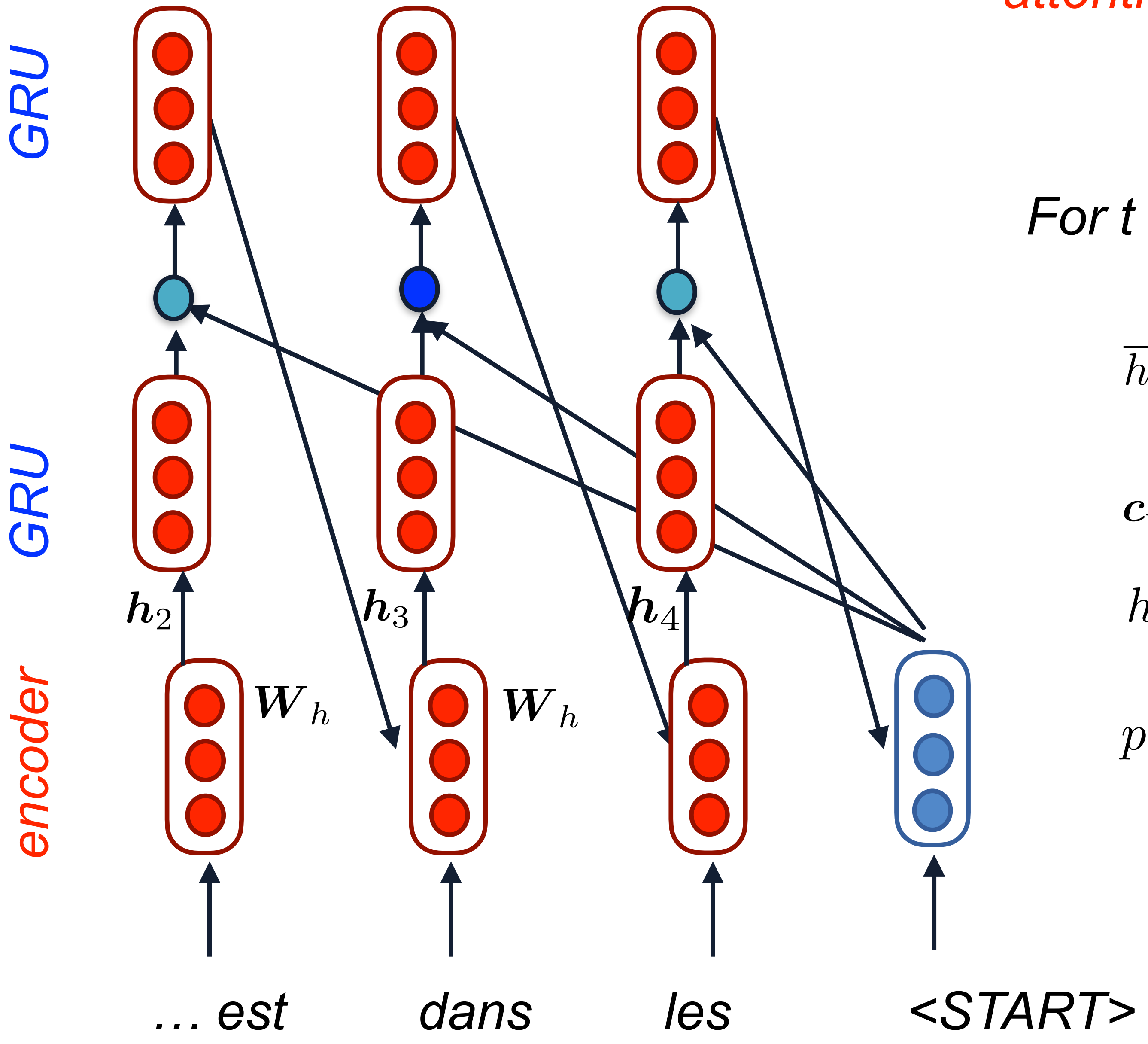
$$\text{score}\left(\boldsymbol{h}_t, \boldsymbol{h}_i\right) = \boldsymbol{h}_t^\top \boldsymbol{h}_i$$

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

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

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

dot product attention



attention output

For t = 1,…, T:

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

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

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

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

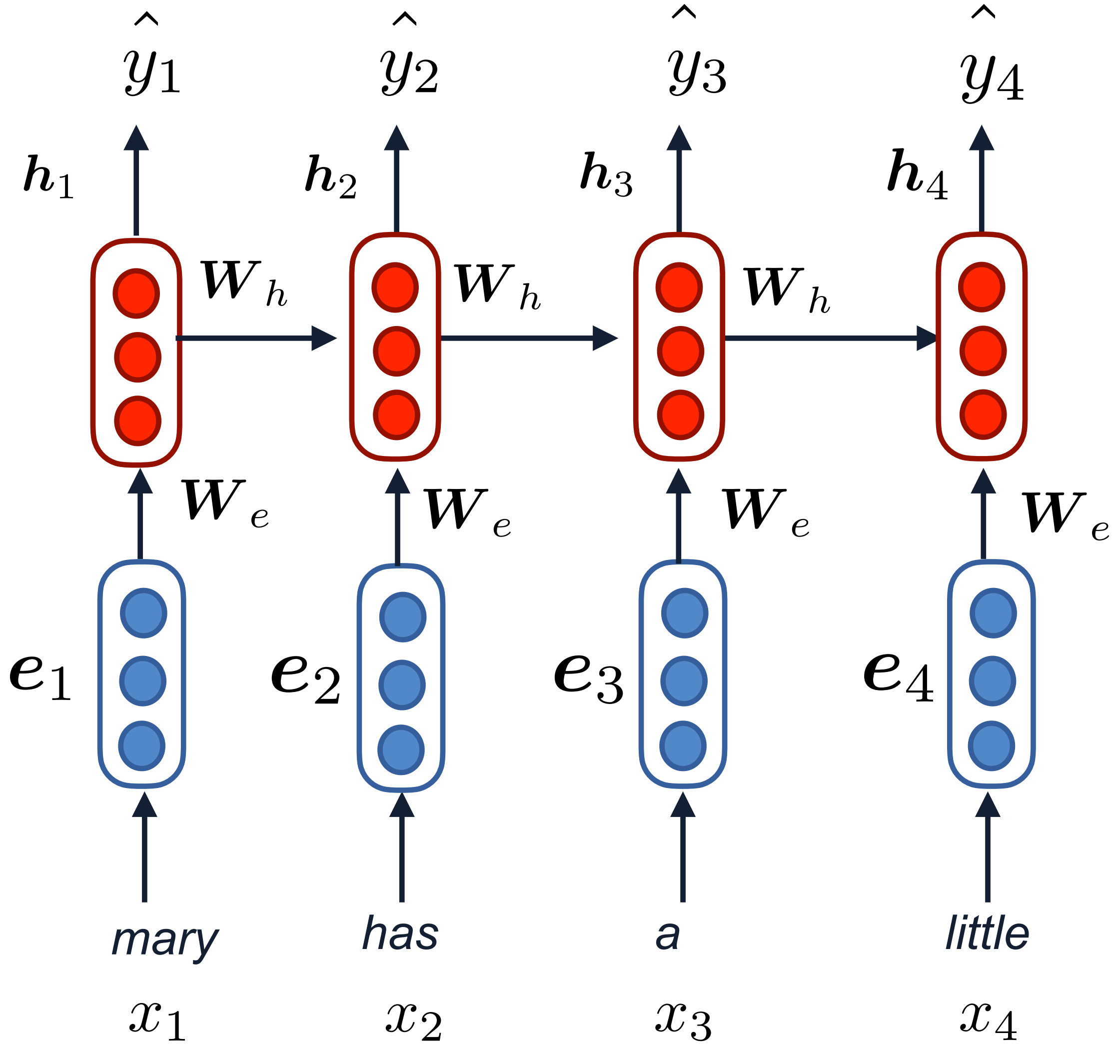
RNN search attention

Performance of attention

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

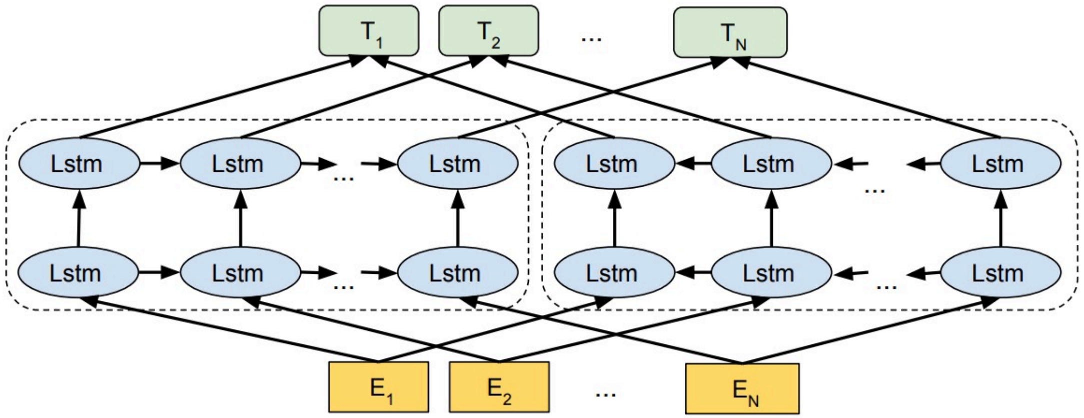
- Pretrained encoder representation:
  - Neural network layers which takes an input sentence and encodes it into a layer output to be fed into another network, the encoder representation can be fine tuned
  - A pretrained LSTM on auto encoder & next word prediction can be used as a starting point to 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 $[\mathbf{h}_k; \mathbf{ELMo}_k^{task}]$

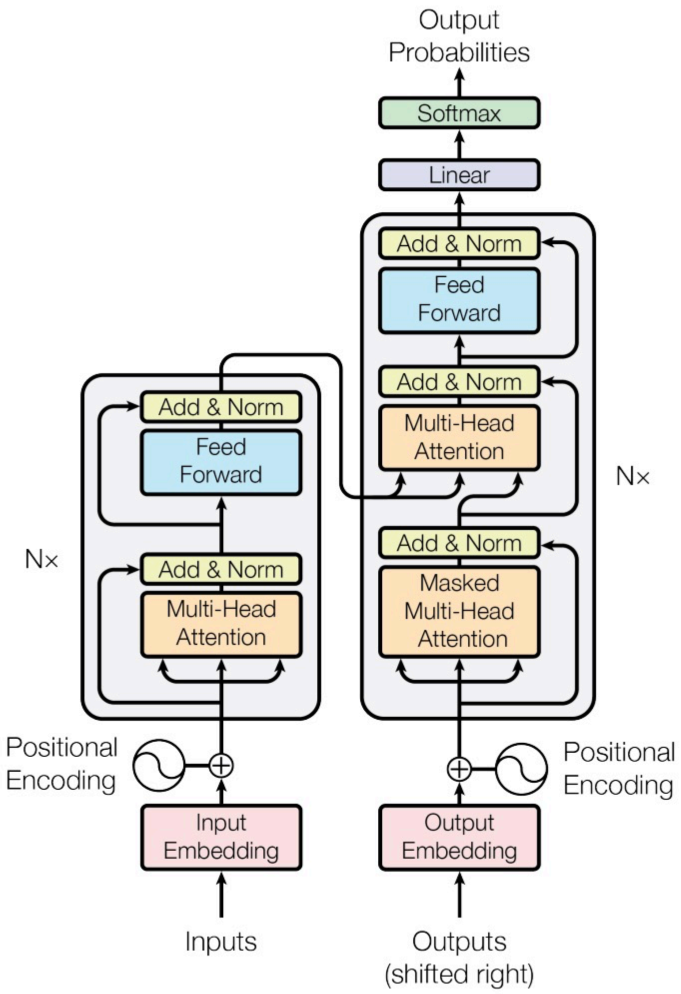


Transformer

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

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

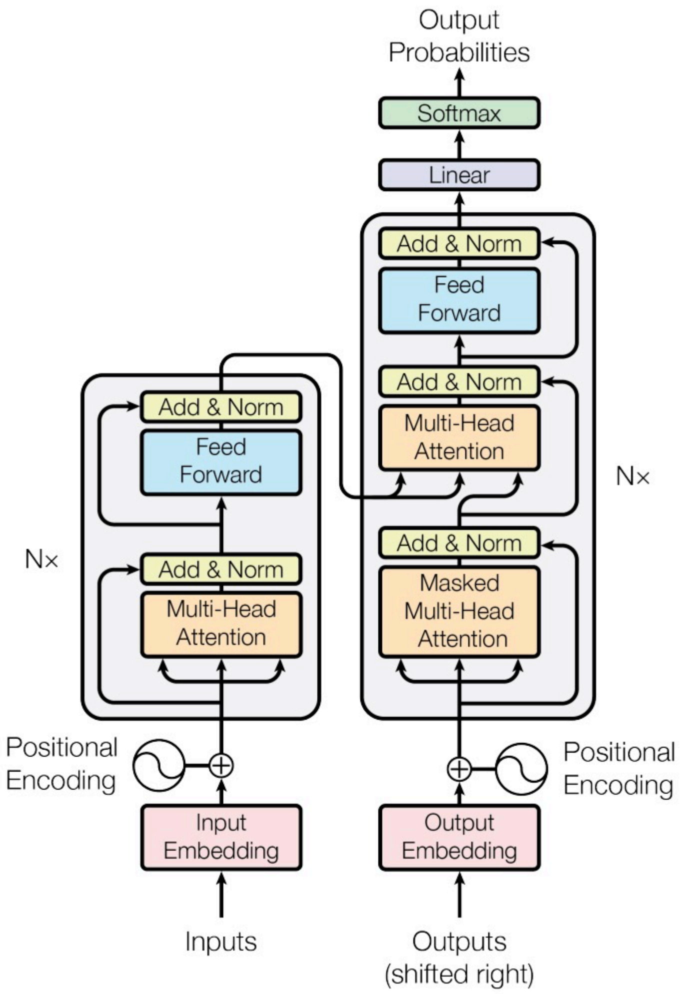
- Position encoding [Gehring et al. 2017]



Encoder/decoder in transformer

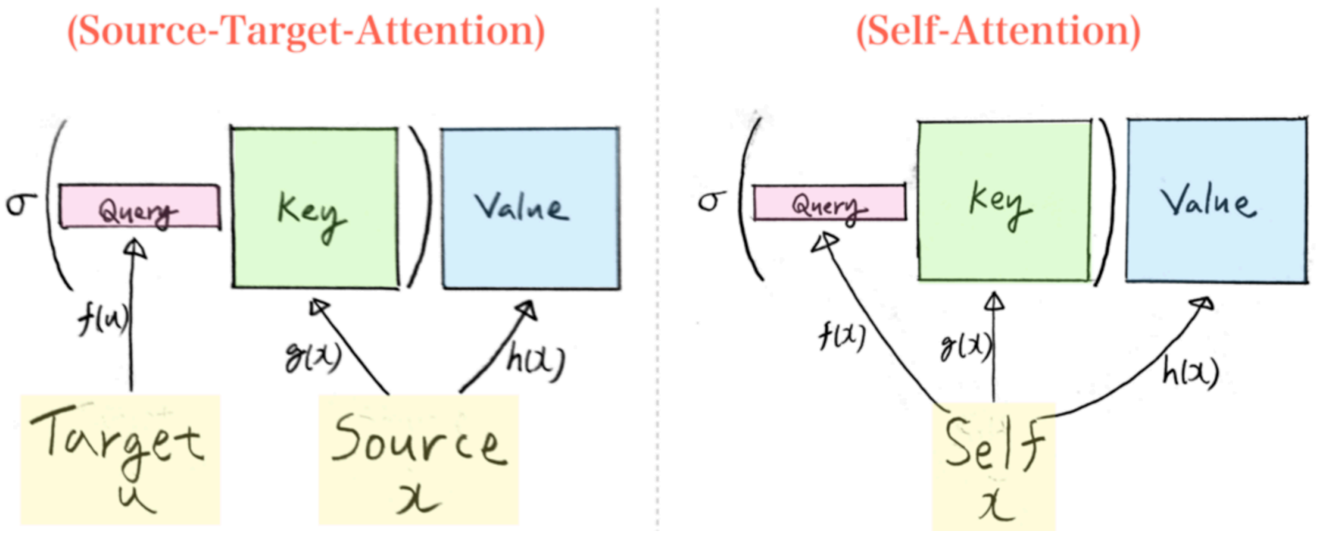
- Encoder: 6 identical layers
  - First layer: multi-head self attention
  - Second layer: fully connected feed forward network, followed by layer normalization

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

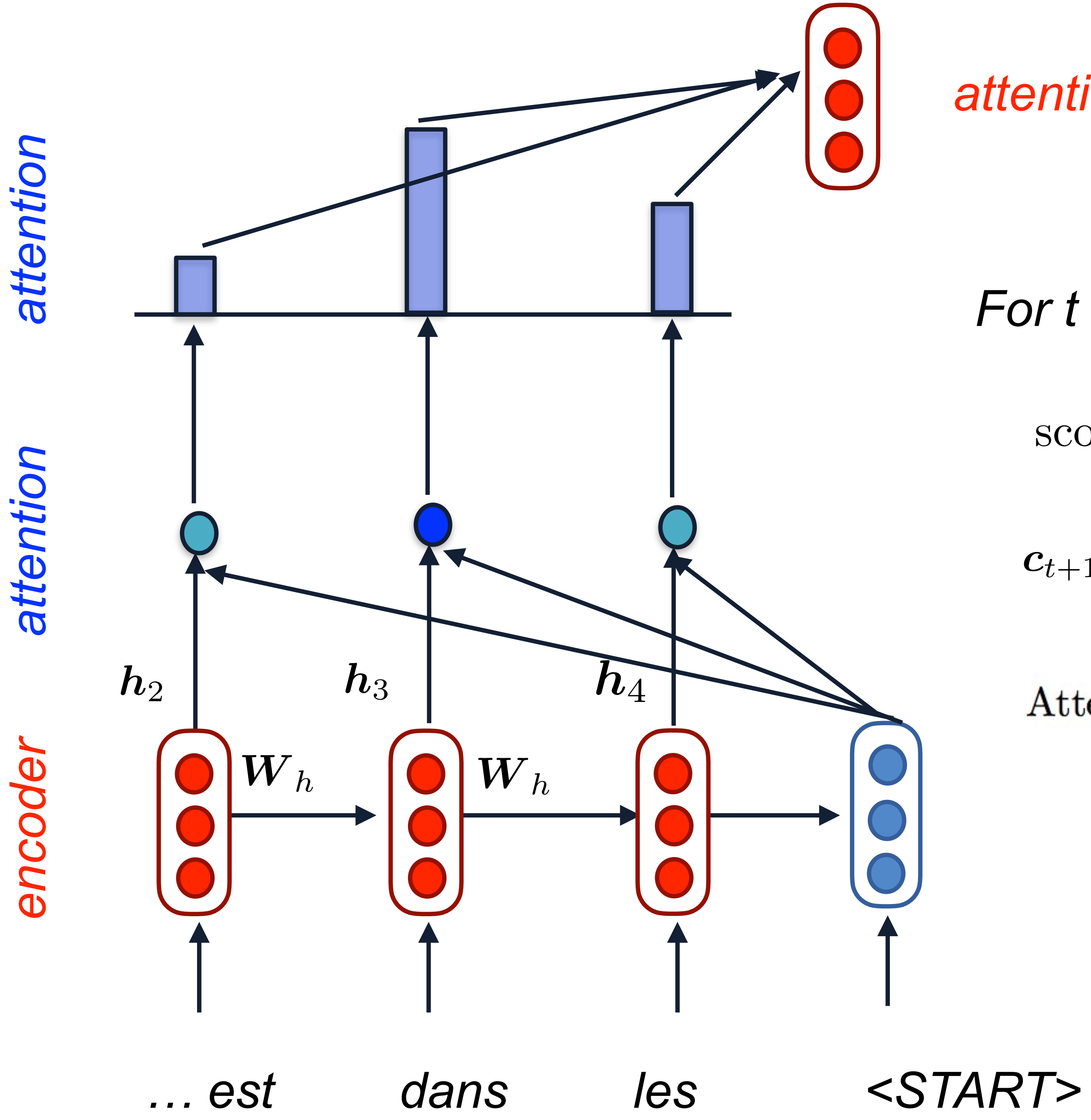


Self attention [Lin et al. 2017]

- 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/



For t = 1,…, T:

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

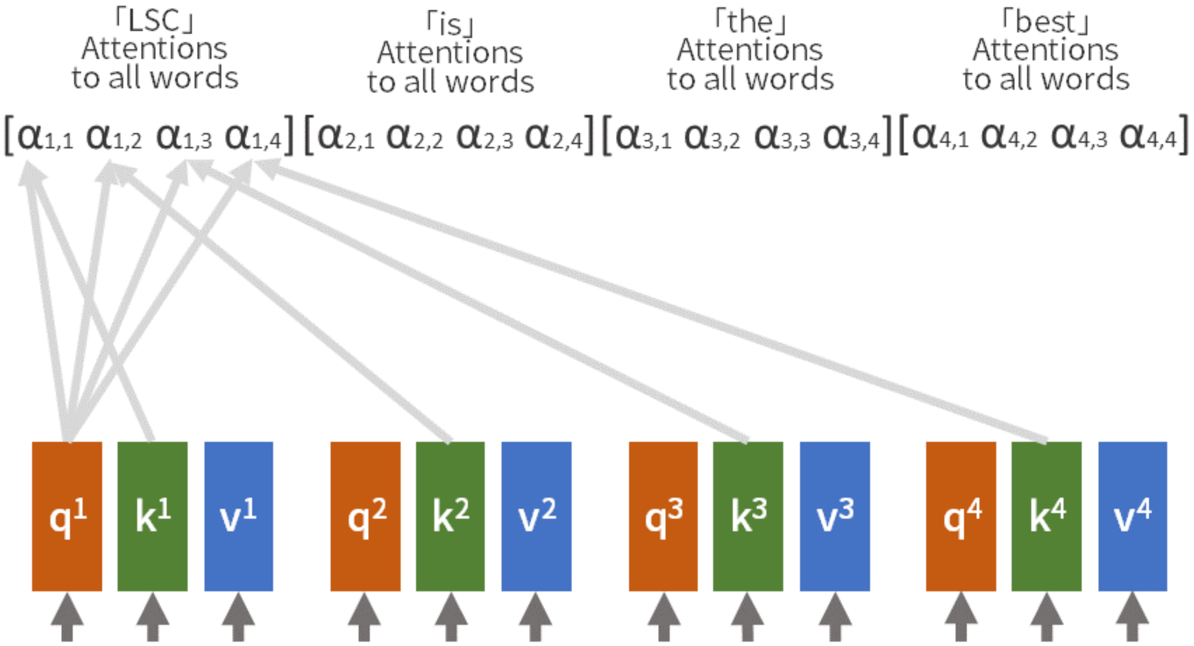
$$\boldsymbol{c}_{t+1} = \sum_{i=1}^{N} softmax(score(\boldsymbol{h}_i, \boldsymbol{h}_t))\boldsymbol{h}_i \in \mathbb{R}^h$$

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

dot product attention

Self attention [Lin et al. 2017]

- The vector at each position is replicated as 3 vectors: Q, K, V vectors



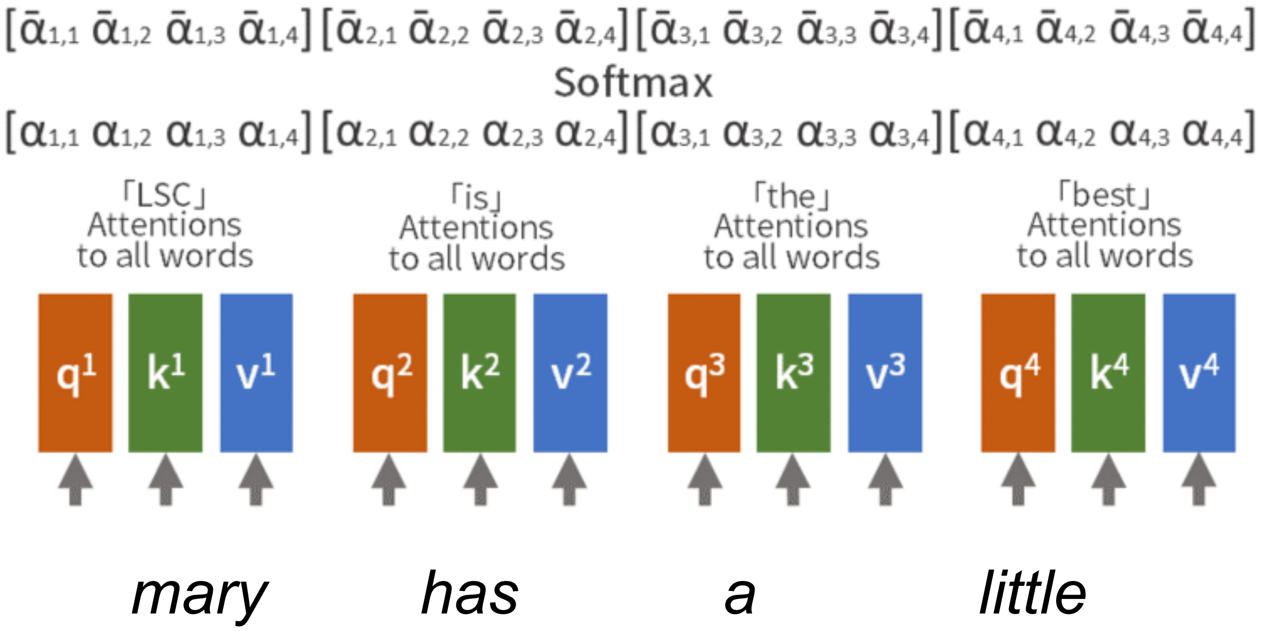
$$\alpha_{i,j} = \frac{q^i \cdot k^j}{\sqrt{d}}$$

d: dimension of q, k

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

Self attention [Lin et al. 2017]

$$b^i = \sum_j \bar{\alpha}_{i,j} v^j$$



Self attention [Lin et al. 2017]

- Self attention can capture the relation between long-distant words

$$b_i = \sum_{j=1}^{n} \alpha_{ij} \left(x_j W^V \right) \qquad e_{ij} = \frac{\left(x_i W^Q \right) \left(x_j W^K \right)^T}{\sqrt{d_z}}$$

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

- Capturing domain knowledge using self-attention, e.g., “C” vs. “Al” in chemistry [Shaw et al. 2018]

$$e_{ij} = \frac{x_i W^Q \left(x_j W^K + a_{ij}^K \right)^T}{\sqrt{d_z}}$$

Position embedding

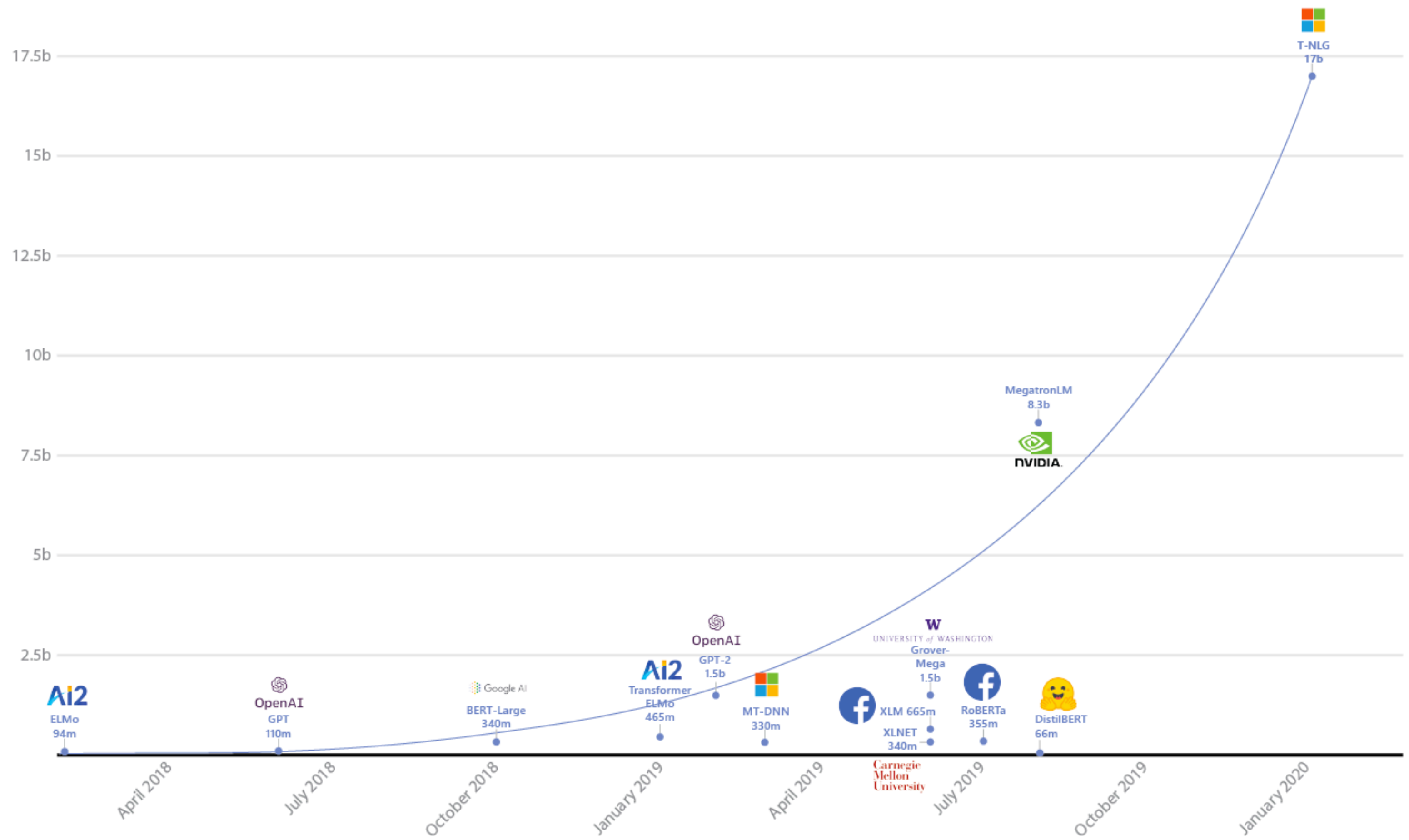
- So far we have not considered the order between words
- Word orders are important for encoding sentence semantics
- How to solve the problem? Position embedding

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

- 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**

store gallon

↑ ↑

the man went to the [MASK] to buy a [MASK] of milk

- 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**

store ↑ gallon ↑

the man went to the [MASK] to buy a [MASK] of milk

- 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)**

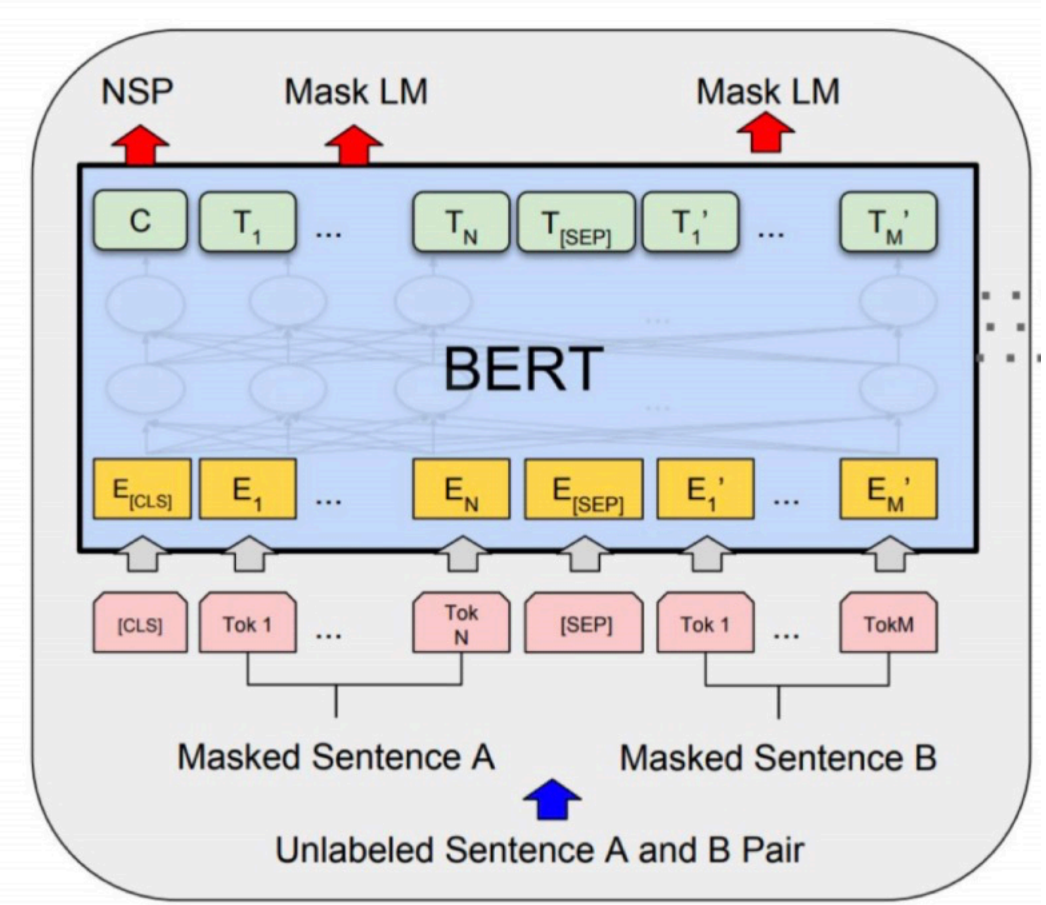
Sentence A = `The man went to the store.`
Sentence B = `He bought a gallon of milk.`
Label = `IsNextSentence`

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

BERT: Bidirectional Encoder Representations

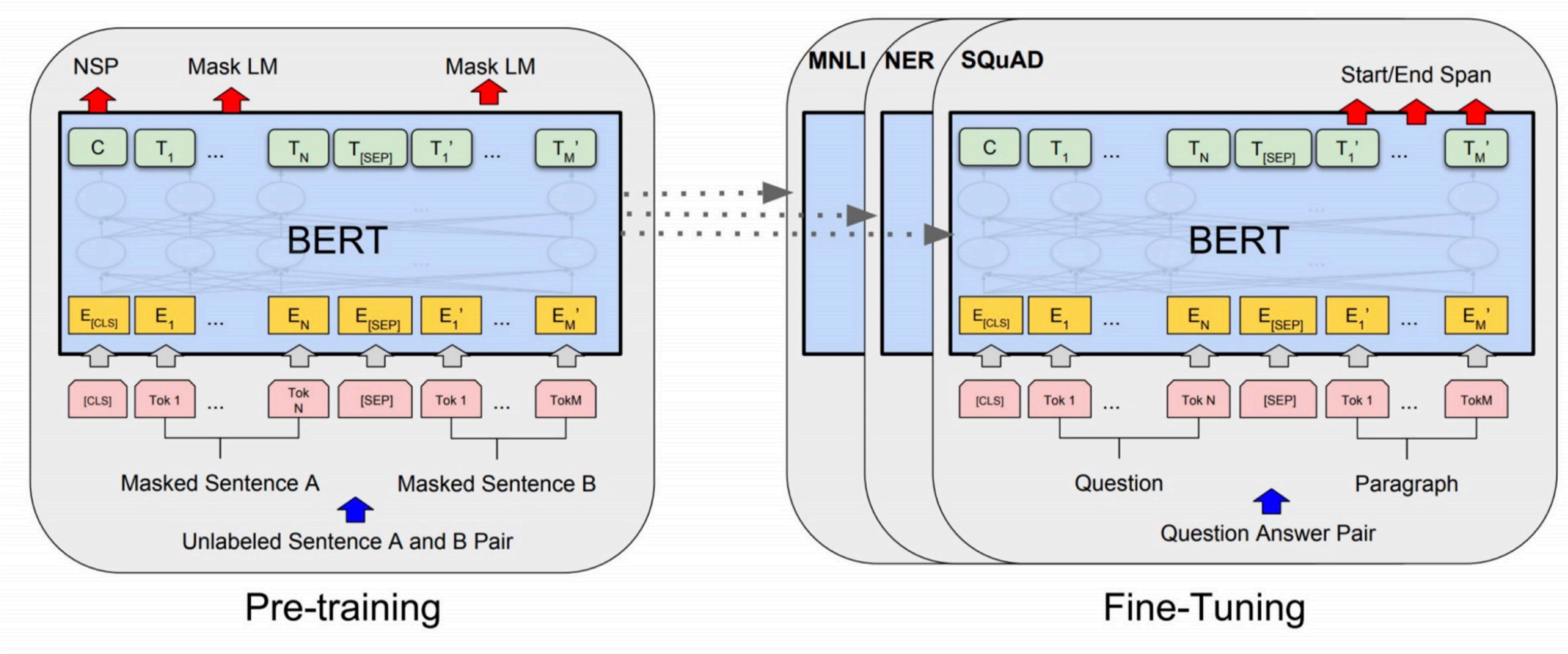
Pre-training BERT

- 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)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
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

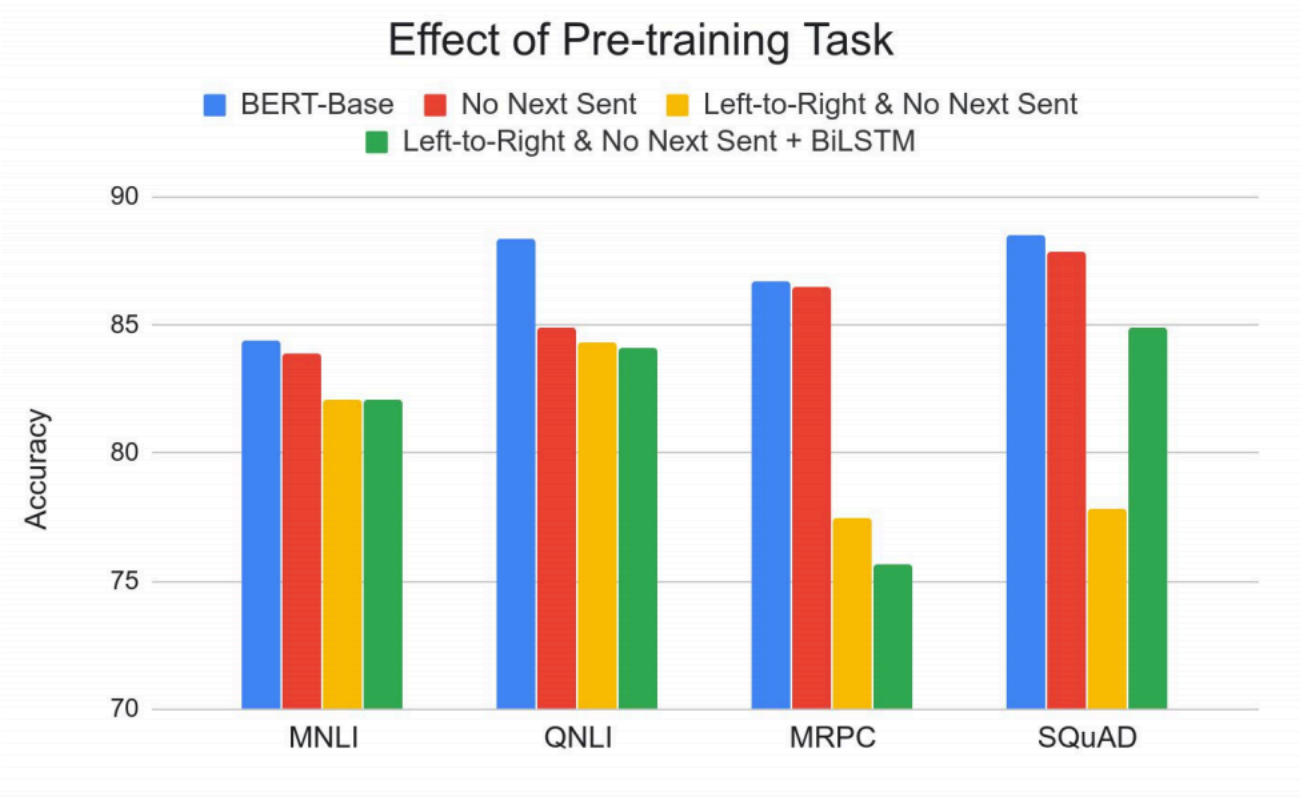
Sentence: The wagon rumbled down the road.
Label: Acceptable

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

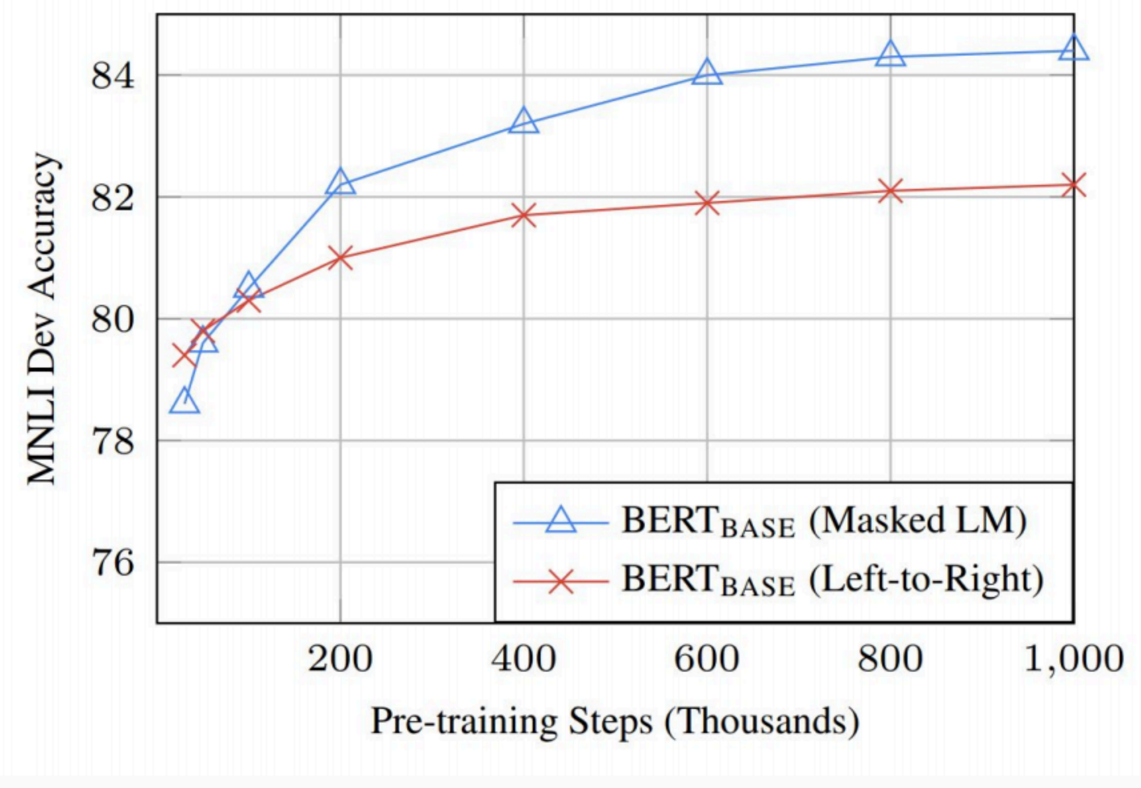
- Left-to-right model does very poorly on word-level task (SQuAD), although this is mitigated by BiLSTM



Effects of dimensionality and training time

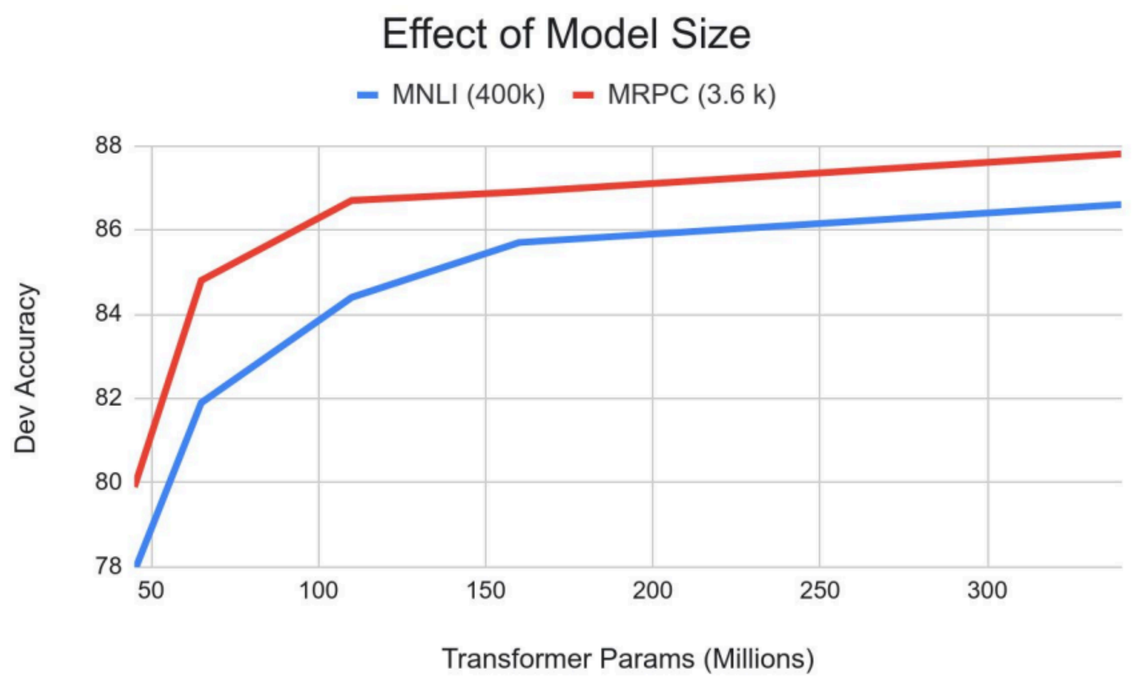
- Masked LM takes slightly longer to converge because we only predict 15% instead of 100%

- 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)

Transformers

build passing | license Apache-2.0 | website online | release v3.3.1

Contributor Covenant v2.0 adopted

```
from transformers import AutoTokenizer, AutoModelForMaskedLM

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

model = AutoModelForMaskedLM.from_pretrained("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 tag prediction using hugging face’s transformer library