Natural Language Processing

AI51701/CSE71001

Lecture 11

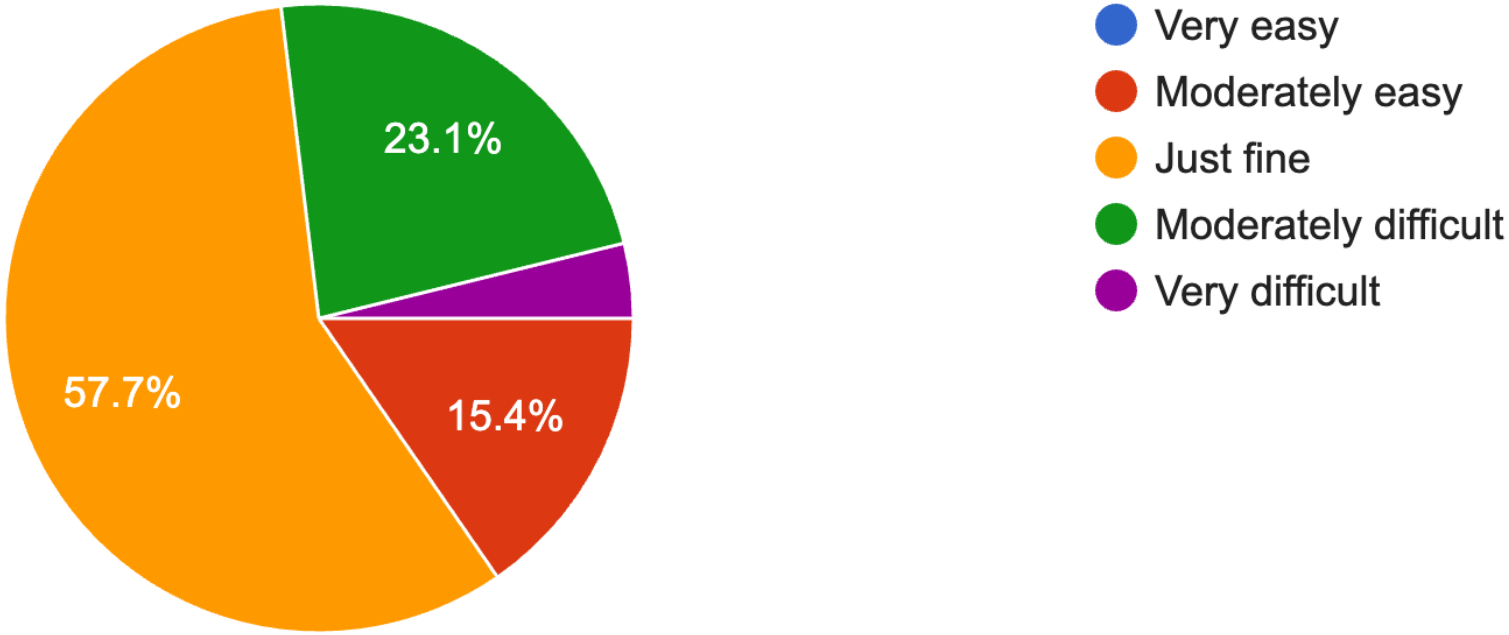
10/24/2023

Instructor: Taehwan Kim

Survey Results

Do you think the contents in this course are easy or difficult to you?

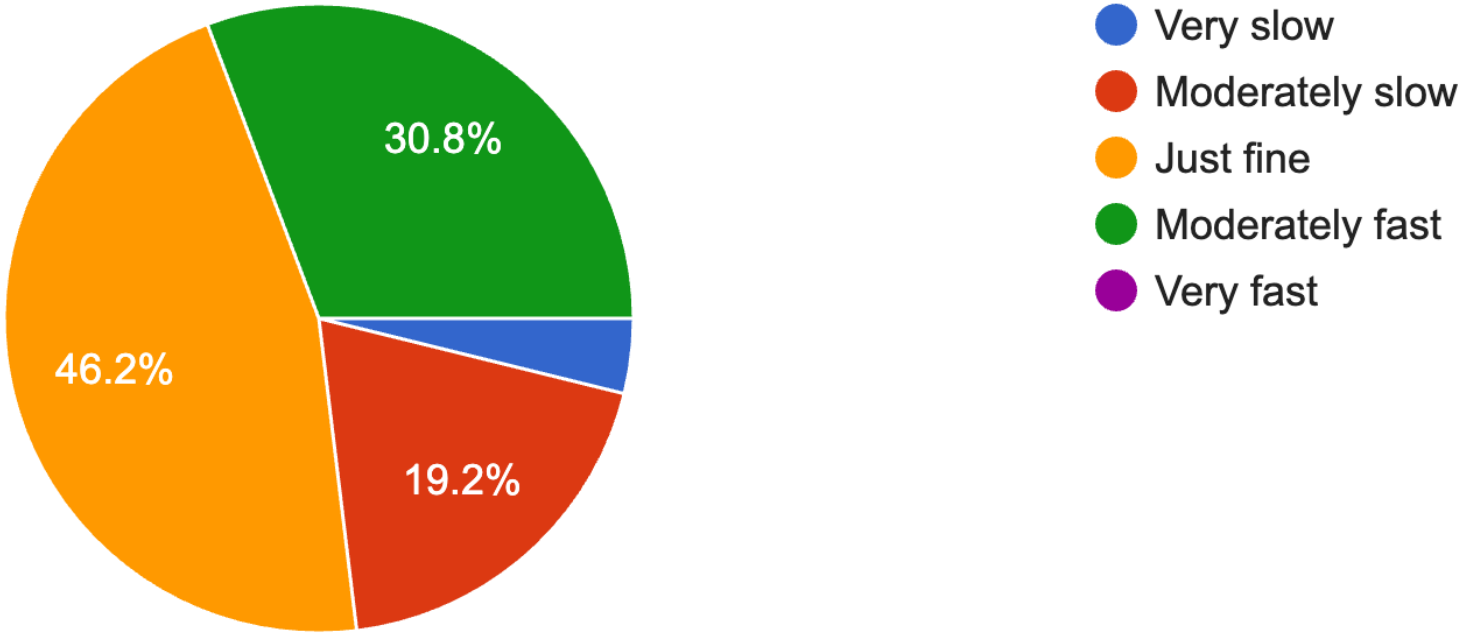
26 responses



Survey Results

Do you think the teaching in this course is too fast or slow to you?

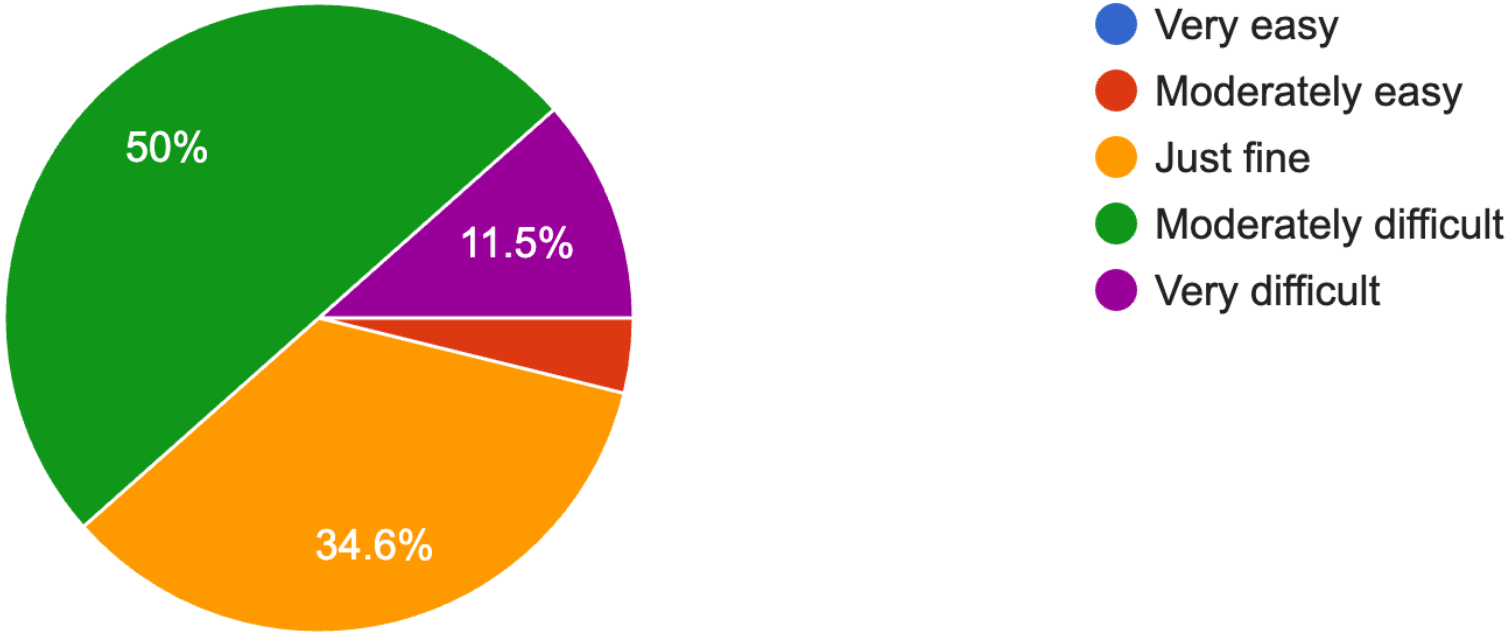
26 responses



Survey Results

Do you think the assignment in this course is easy or difficult for you?

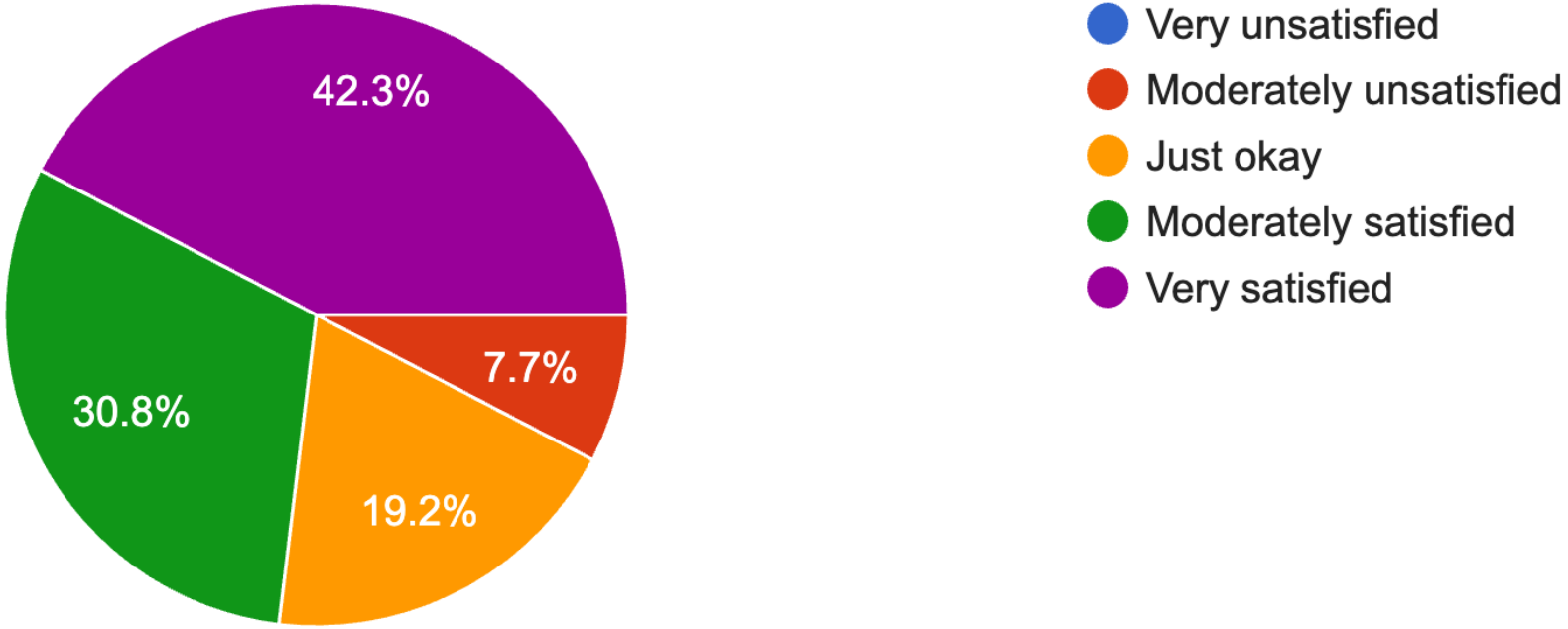
26 responses



Survey Results

In overall, are you currently satisfied with this course?

26 responses



Some Comments (summarized)

- Page limits of final project proposal and report
  - It's 3-4 pages for the proposal
- Use more example/practice when explaining some difficult concepts
- Being unable to complete the previous quiz
  - Let me know

Long Short-Term Memory RNNs (LSTMs)

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the vanishing gradients problem.
  - Everyone cites that paper but really a crucial part of the modern LSTM is from Gers et al. (2000)

- On step t, there is a **hidden state** $h^{(t)}$ *and* a **cell state** $c^{(t)}$
  - Both are vectors length n
  - The cell stores **long-term information**
  - The LSTM can **read**, **erase**, and **write** information from the cell
    - The cell becomes conceptually rather like RAM in a computer

"Long short-term memory", Hochreiter and Schmidhuber, 1997. https://www.bioinf.jku.at/publications/older/2604.pdf
"Learning to Forget: Continual Prediction with LSTM", Gers, Schmidhuber, and Cummins, 2000. https://dl.acm.org/doi/10.1162/089976600300015015

Gated Recurrent Units (GRU)

- Proposed by Cho et al. in 2014 as a simpler alternative to the LSTM.
- On each timestep t we have input $\boldsymbol{x}^{(t)}$ and hidden state $\boldsymbol{h}^{(t)}$ (no cell state)

"Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", Cho et al. 2014, https://arxiv.org/pdf/1406.1078v3.pdf

How does LSTM solve vanishing gradients?

- The LSTM architecture makes it **easier** for the RNN to **preserve information over many timesteps**
  - e.g., if the forget gate is set to 1 for a cell dimension and the input gate set to 0, then the information of that cell is preserved indefinitely.
  - In contrast, it's harder for a vanilla RNN to learn a recurrent weight matrix W_h that preserves info in the hidden state

- LSTM doesn't *guarantee* that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

LSTMs: real-world success

- In 2013–2015, LSTMs started achieving state-of-the-art results
  - Successful tasks include handwriting recognition, speech recognition, machine translation
  - **LSTMs** became the **dominant approach** for most NLP tasks
- Now (2019–2022), other approaches (e.g., **Transformers**) have become dominant for many tasks
  - For example, in **WMT** (a Machine Translation conference + competition):
    - In WMT 2014, there were 0 neural machine translation systems (!)
    - In WMT 2016, the summary report contains "**RNN**" 44 times (and these systems won)
    - In WMT 2019: "**RNN**" 7 times, "**Transformer**" 105 times

Is vanishing/exploding gradient just a RNN problem?

- No! It can be a problem for all neural architectures (including **feed-forward** and **convolutional**), especially **very deep** ones.
  - Due to chain rule/choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
  - Thus, lower layers are learned very slowly (hard to train)
- Solution: lots of new deep feedforward/convolutional architectures **add more direct connections** (thus allowing the gradient to flow)
  - For example:
  - **Residual connections** aka "ResNet"
  - Also known as **skip-connections**
  - The **identity connection preserves information** by default
  - This makes **deep** networks much **easier to train**

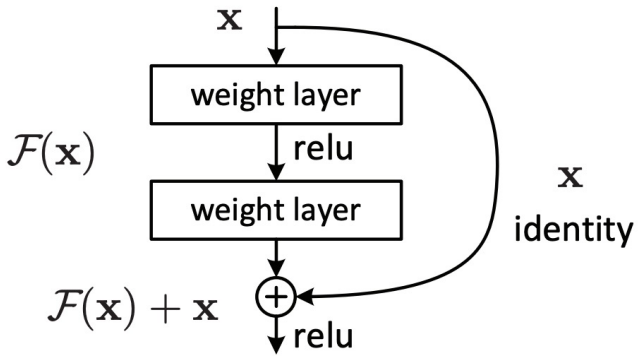


Figure 2. Residual learning: a building block.

"Deep Residual Learning for Image Recognition", He et al, 2015. https://arxiv.org/pdf/1512.03385.pdf

Is vanishing/exploding gradient just a RNN problem?

- Other methods:
  - Dense connections aka “DenseNet”
    - Directly connect each layer to all future layers!
  - Highway connections aka “HighwayNet”
    - Similar to residual connections, but the identity connection vs the transformation layer is controlled by a dynamic gate
    - Inspired by LSTMs, but applied to deep feedforward/convolutional networks

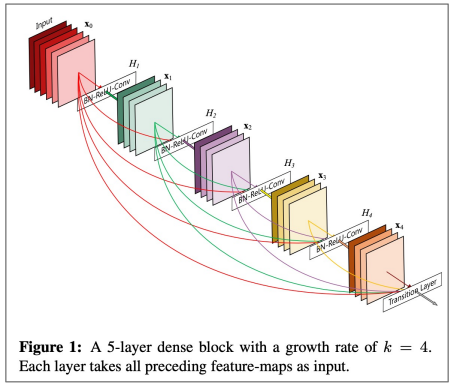
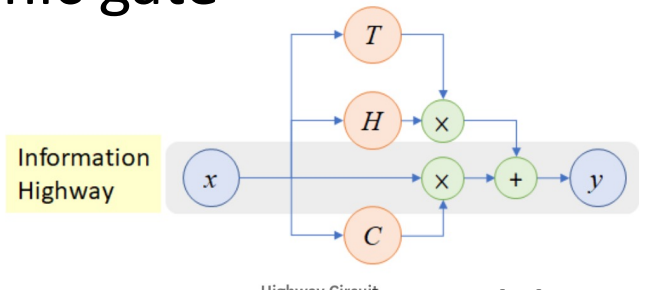


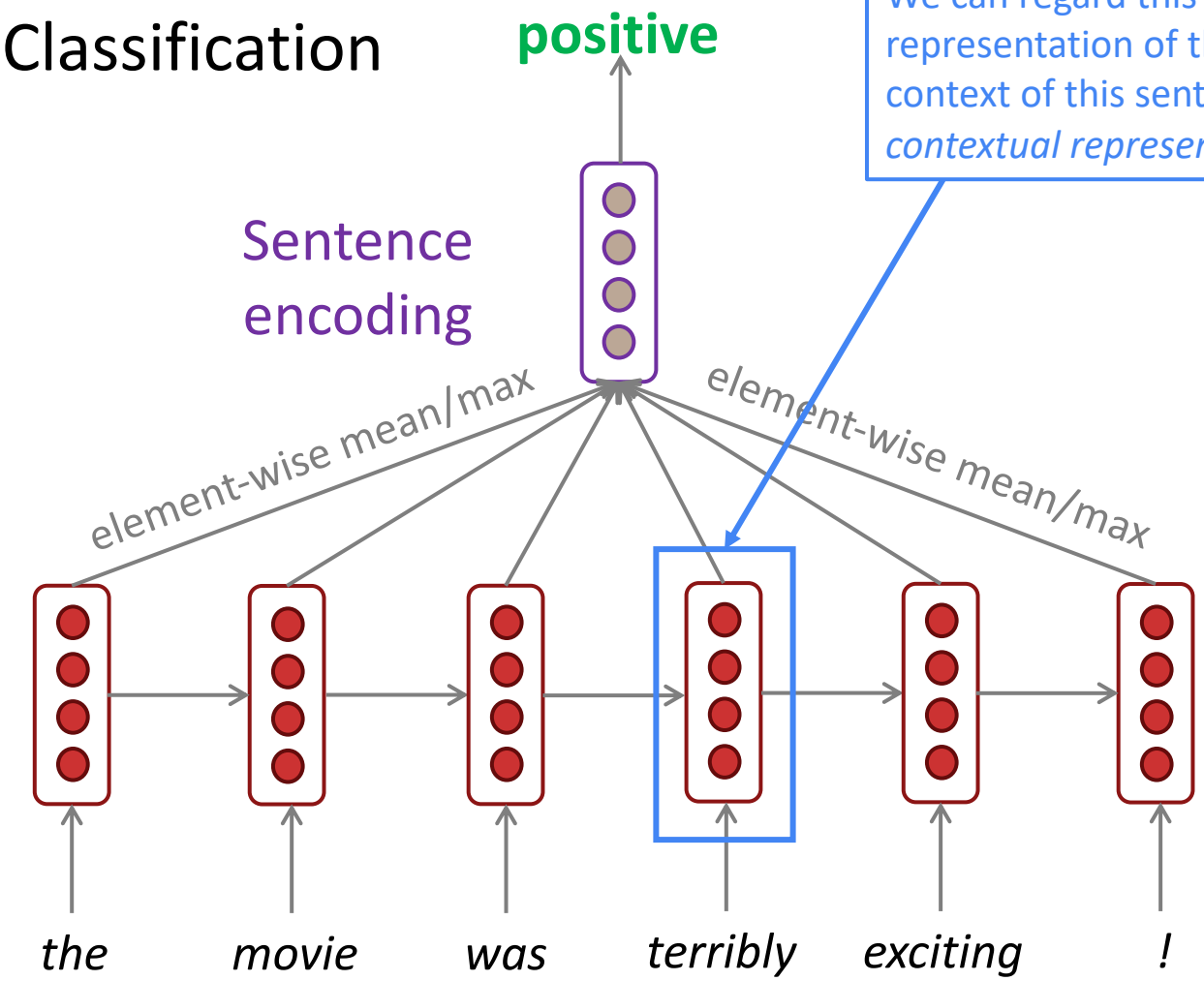
Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.



- **Conclusion**: Though vanishing/exploding gradients are a general problem, RNNs are particularly unstable due to the repeated multiplication by the same weight matrix [Bengio et al, 1994]

Bidirectional and Multi-layer RNNs: motivation

- Task: Sentiment Classification



We can regard this hidden state as a representation of the word *"terribly"* in the context of this sentence. We call this a *contextual representation.*

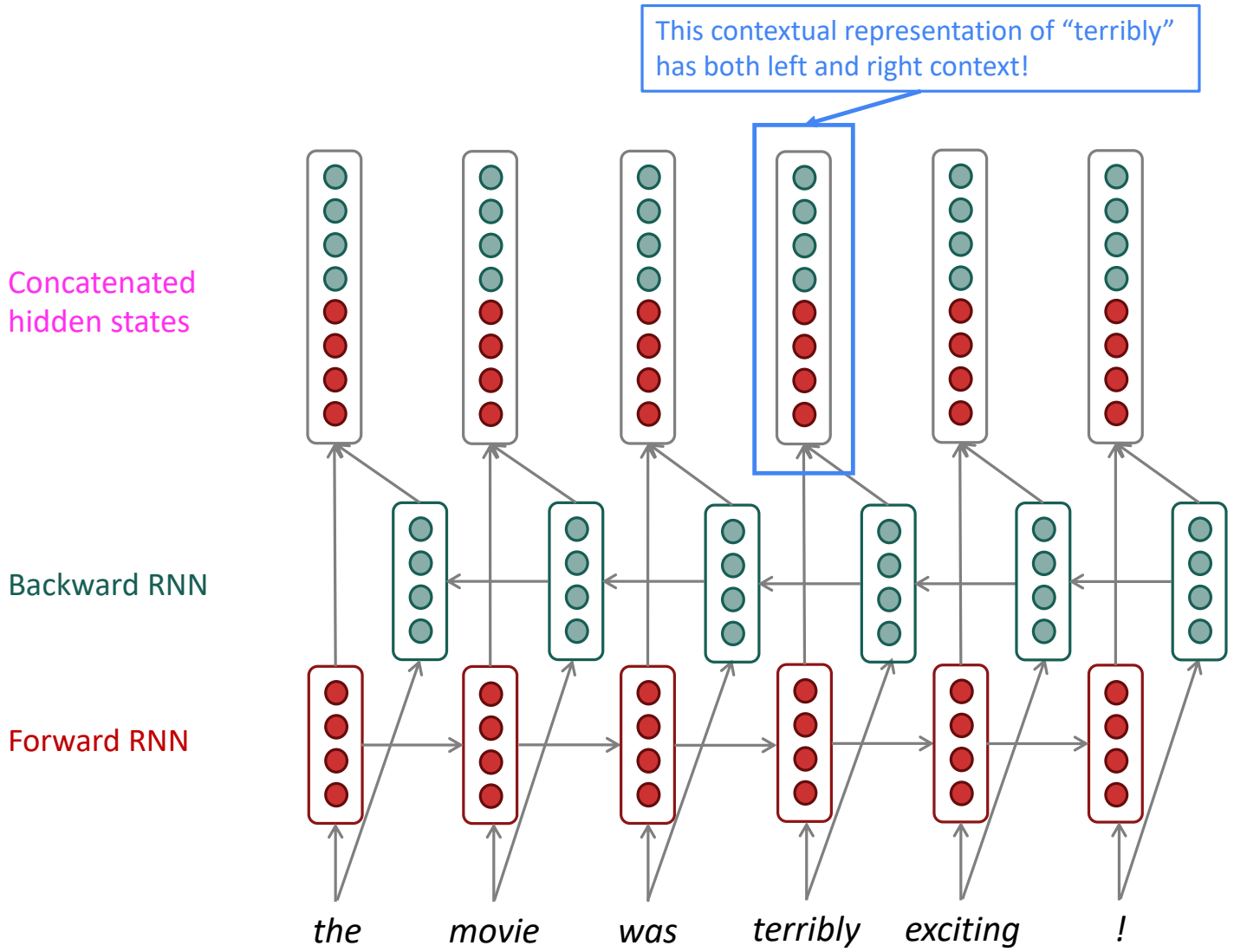
These contextual representations only contain information about the *left* context (e.g. *"the movie was"*).

What about *right* context?

In this example, *"exciting"* is in the right context and this modifies the meaning of *"terribly"* (from negative to positive)

From Stanford CS224n course

Bidirectional RNNs



Bidirectional RNNs

On timestep t:

This is a general notation to mean "compute one forward step of the RNN" – it could be a simple, LSTM, or other (e.g., GRU) RNN computation.

Forward RNN $\overrightarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{FW}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$

Backward RNN $\overleftarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)})$

Generally, these two RNNs have separate weights

Concatenated hidden states $\boldsymbol{h}^{(t)} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$

We regard this as "the hidden state" of a bidirectional RNN. This is what we pass on to the next parts of the network.

Bidirectional RNNs

- Note: bidirectional RNNs are only applicable if you have access to the **entire input sequence**
  - They are **not** applicable to Language Modeling, because in LM you *only* have left context available.
- If you do have entire input sequence (e.g., any kind of encoding), **bidirectionality is powerful** (you should use it by default).
- For example, **BERT** (*Bidirectional* Encoder Representations from Transformers) is a powerful pretrained contextual representation system **built on bidirectionality**.
  - You will learn more about **transformers**, including BERT, in a few weeks!

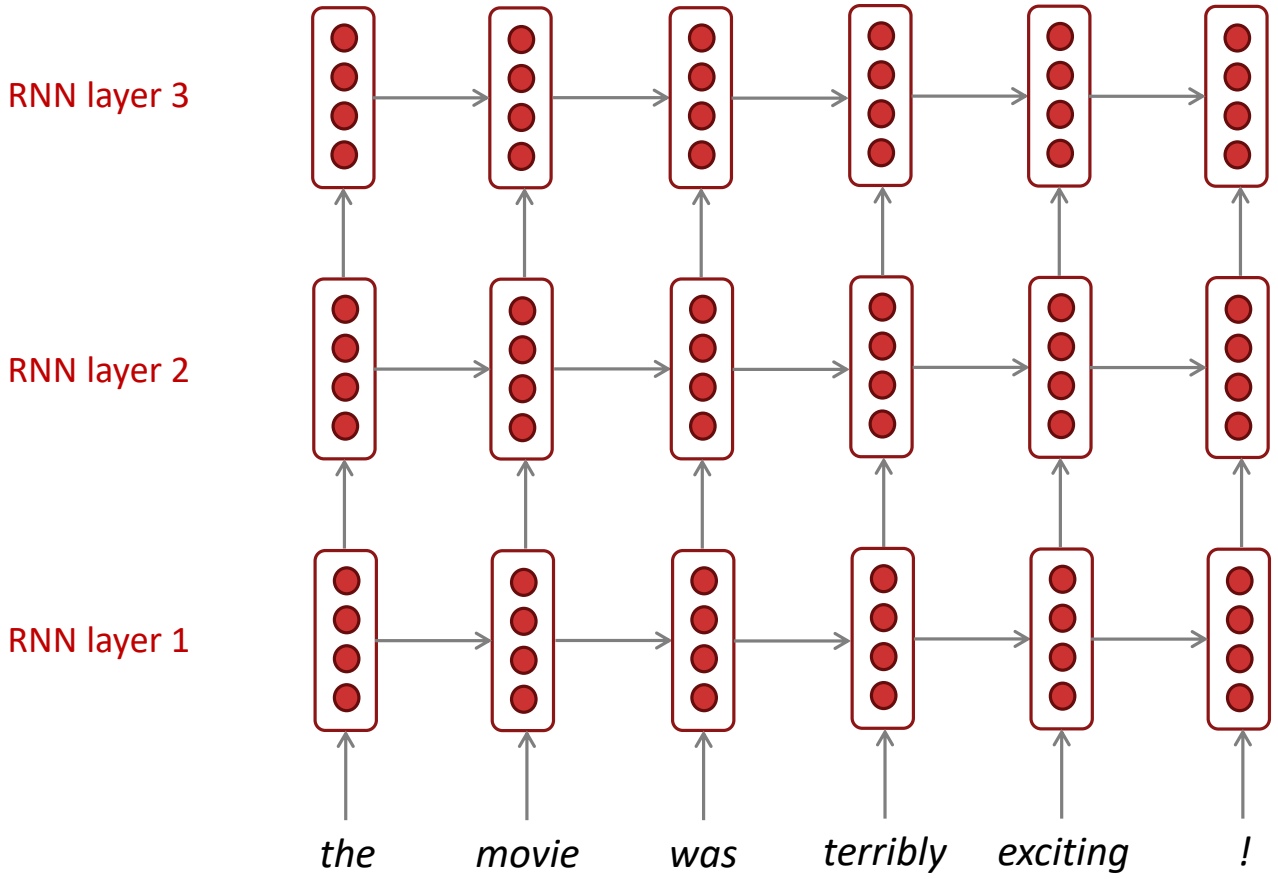
Multi-layer RNNs

- RNNs are already “deep” on one dimension (they unroll over many timesteps)
- We can also make them “deep” in another dimension by **applying multiple RNNs** – this is a multi-layer RNN.
- This allows the network to compute **more complex representations**
  - The **lower RNNs** should **compute lower-level features** and the **higher RNNs** should compute **higher-level features**.

- Multi-layer RNNs are also called *stacked RNNs*.

Multi-layer RNNs

The hidden states from RNN layer *i* are the inputs to RNN layer *i*+1



Multi-layer RNNs in practice

- **High-performing RNNs are usually multi-layer** (but aren't as deep as convolutional or feed-forward networks)
- For example: In a 2017 paper, Britz et al. find that for Neural Machine Translation, **2 to 4 layers** is best for the encoder RNN, and **4 layers** is best for the decoder RNN
  - Often 2 layers is a lot better than 1, and 3 might be a little better than 2
  - Usually, **skip-connections/dense-connections** are needed to train deeper RNNs (e.g., **8 layers**)
- **Transformer**-based networks (e.g., BERT) are usually deeper, like **12 or 24 layers**.
  - You will learn about Transformers later; they have a lot of skipping-like connections

"Massive Exploration of Neural Machine Translation Architecutres", Britz et al, 2017. https://arxiv.org/pdf/1703.03906.pdf

Introduction to PyTorch

From 2021 Fall AI Toolkit course in AIGS,
also adapted from Sung Kim’s lecture slides at KHUST

What is Python?

- Python is an **easy to learn**, **powerful** programming language.
- It has **efficient high-level** data structures and a **simple but effective** approach to **object-oriented programming**.
- Simple syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and **rapid application development** in many areas on most platforms.

Why Python for AI?

- Simple syntax & less coding
  - Very less coding and simple syntax among other programming languages such as C++.
- Inbuilt libraries for AI projects
  - Great inbuilt libraries for AI.
    – Mathematical library: Numpy, Pandas
    – Machine learning library: Scikit-learn
    – Computer vision library: OpenCV
    – NLP library: Natural Language Toolkit (NLTK)
    – Deep learning library: PyTorch, Tensorflow, Keras
  - Open source
  - Can be used for broad range of programming, from small shell script to enterprise web applications.

What is PyTorch?

- PyTorch is a python package that provides two high-level features:
  - **Tensor computation** (like NumPy) with strong GPU acceleration
  - Deep Neural Networks built on a tape-based **autograd** system

The Steps in PyTorch

- Your steps in PyTorch

Design your model using class

Construct loss and optimizer (select from PyTorch API)

Training cycle (forward, backward, update)

Classifying Diabetes

-0.411765	0.165829	0.213115	0	0	-0.23696	-0.894962	-0.7	1
-0.647059	-0.21608	-0.180328	-0.353535	-0.791962	-0.0760059	-0.854825	-0.833333	0
0.176471	0.155779	0	0	0	0.052161	-0.952178	-0.733333	1
-0.764706	0.979899	0.147541	-0.0909091	0.283688	-0.0909091	-0.931682	0.0666667	0
-0.0588235	0.256281	0.57377	0	0	0	-0.868488	0.1	0
-0.529412	0.105528	0.508197	0	0	0.120715	-0.903501	-0.7	1
0.176471	0.688442	0.213115	0	0	0.132638	-0.608027	-0.566667	0
0.176471	0.396985	0.311475	0	0	-0.19225	0.163962	0.2	1

```
xy = np.loadtxt('data-diabetes.csv', delimiter=',', dtype=np.float32)

x_data = torch.from_numpy(xy[:, 0:-1])
y_data = torch.from_numpy(xy[:, [-1]])

print(x_data.data.shape) # torch.Size([759, 8])
print(y_data.data.shape) # torch.Size([759, 1])
```

Classifying Diabetes

```
class Model(torch.nn.Module):

    def __init__(self):
        """
        In the constructor we instantiate three nn.Linear module
        """
        super(Model, self).__init__()
        self.l1 = torch.nn.Linear(8, 6)
        self.l2 = torch.nn.Linear(6, 4)
        self.l3 = torch.nn.Linear(4, 1)

        self.sigmoid = torch.nn.Sigmoid()

    def forward(self, x):
        """
        In the forward function we accept a Variable of input data and we must return
        a Variable of output data. We can use Modules defined in the constructor as
        well as arbitrary operators on Variables.
        """
        out1 = self.sigmoid(self.l1(x))
        out2 = self.sigmoid(self.l2(out1))
        y_pred = self.sigmoid(self.l3(out2))
        return y_pred
```

Classifying Diabetes

```
xy = np.loadtxt('data-diabetes.csv', delimiter=',', dtype=np.float32)
x_data = torch.from_numpy(xy[:, 0:-1])
y_data = torch.from_numpy(xy[:, [-1]])

class Model(torch.nn.Module):
    def __init__(self):
        """
        In the constructor we instantiate two nn.Linear module
        """
        super(Model, self).__init__()
        self.l1 = torch.nn.Linear(8, 6)
        self.l2 = torch.nn.Linear(6, 4)
        self.l3 = torch.nn.Linear(4, 1)

        self.sigmoid = torch.nn.Sigmoid()

    def forward(self, x):
        """
        In the forward function we accept a Variable of input data and we must return
        a Variable of output data. We can use Modules defined in the constructor as
        well as arbitrary operators on Variables.
        """
        out1 = self.sigmoid(self.l1(x))
        out2 = self.sigmoid(self.l2(out1))
        y_pred = self.sigmoid(self.l3(out2))
        return y_pred

# our model
model = Model()

# Construct our loss function and an Optimizer. The call to model.parameters()
# in the SGD constructor will contain the learnable parameters of the two
# nn.Linear modules which are members of the model.
criterion = torch.nn.BCELoss(size_average=True)
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)

# Training loop
for epoch in range(100):
        # Forward pass: Compute predicted y by passing x to the model
    y_pred = model(x_data)

    # Compute and print loss
    loss = criterion(y_pred, y_data)
    print(epoch, loss.data[0])

    # Zero gradients, perform a backward pass, and update the weights.
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Design your model using class

Construct loss and optimizer (select from PyTorch API)

Training cycle (forward, backward, update)

Part-of-Speech Tagging

Part-of-Speech Tagging

Some questioned if Tim Cook ’s first product

would be a breakaway hit for Apple .

Part-of-Speech Tagging

determiner	verb (past)	prep.	proper noun	proper noun	poss.	adj.	noun
Some	questioned	if	Tim	Cook	’s	first	product

modal	verb	det.	adjective	noun	prep.	proper noun	punc.
would	be	a	breakaway	hit	for	Apple	.

Part-of-Speech (POS)

- Functional category of a word:
  - noun, verb, adjective, etc.
  - how is the word functioning in its context?
- Dependent on context like word sense, but different from sense:
  - sense represents word meaning, POS represents word function
  - sense uses a distinct category of senses per word, POS uses same set of categories for all words

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	*and, but, or*	SYM	symbol	+,%, &
CD	cardinal number	*one, two*	TO	"to"	*to*
DT	determiner	*a, the*	UH	interjection	*ah, oops*
EX	existential 'there'	*there*	VB	verb base form	*eat*
FW	foreign word	*mea culpa*	VBD	verb past tense	*ate*
IN	preposition/sub-conj	*of, in, by*	VBG	verb gerund	*eating*
JJ	adjective	*yellow*	VBN	verb past participle	*eaten*
JJR	adj., comparative	*bigger*	VBP	verb non-3sg pres	*eat*
JJS	adj., superlative	*wildest*	VBZ	verb 3sg pres	*eats*
LS	list item marker	*1, 2, One*	WDT	wh-determiner	*which, that*
MD	modal	*can, should*	WP	wh-pronoun	*what, who*
NN	noun, sing. or mass	*llama*	WP$	possessive wh-	*whose*
NNS	noun, plural	*llamas*	WRB	wh-adverb	*how, where*
NNP	proper noun, sing.	*IBM*	$	dollar sign	*$*
NNPS	proper noun, plural	*Carolinas*	#	pound sign	*#*
PDT	predeterminer	*all, both*	"	left quote	' or "
POS	possessive ending	*'s*	"	right quote	' or "
PRP	personal pronoun	*I, you, he*	(	left parenthesis	[, (, {, <
PRP$	possessive pronoun	*your, one's*	)	right parenthesis	],), }, >
RB	adverb	*quickly, never*	,	comma	,
RBR	adverb, comparative	*faster*	.	sentence-final punc	. ! ?
RBS	adverb, superlative	*fastest*	:	mid-sentence punc	: ; ... – -
RP	particle	*up, off*			

Description	Example
symbol	+,%, &
"to"	*to*
interjection	*ah, oops*
verb base form	*eat*
verb past tense	*ate*
verb gerund	*eating*
verb past participle	*eaten*
verb non-3sg pres	*eat*
verb 3sg pres	*eats*
wh-determiner	*which, that*
wh-pronoun	*what, who*
possessive wh-	*whose*
wh-adverb	*how, where*
dollar sign	*$*
pound sign	*#*
left quote	' or "
right quote	' or "
left parenthesis	[, (, {, <
right parenthesis	],), }, >
comma	,
sentence-final punc	. ! ?
mid-sentence punc	: ; ... – -

POS Ambiguity in Penn Treebank

- Word that can be both noun and verb?
  - more often noun than verb:
    - increase: 248 NN vs. 127 VB (and 4 VBP)
    - place: 134 NN vs. 14 VB (and 4 VBP)
  - more often verb than noun:
    - makes: 182 VBZ vs. 5 NNS
    - transfer: 22 VB vs. 16 NN

POS Ambiguity in Penn Treebank

- Word that can be both a singular noun and a plural noun?
  - "savings", e.g.:

DT NN VBD VBN RB
The savings was given incorrectly …

DT JJ NN NN
a Belgian savings bank

POS Ambiguity in Penn Treebank

- Word that can be both a common noun and a proper noun?
  - "Earth": 16 NNP vs. 5 NN
  - annotation inconsistencies: nothing in the context indicates which tag is used
  - these kinds of inconsistencies are common in annotated datasets, so it's usually not possible to get perfect accuracy

POS Ambiguity in Penn Treebank

- Word that can be both a common noun and a proper noun?
  - "Chapter": 21 NNP vs. 41 NN
  - annotation inconsistencies:

VB	VBG	IN	NNP	NNP	NN	NN
consider	filing	for	Chapter	11	bankruptcy	protection

NNP	VBD	IN	NN	CD	NN	NN
Continental	filed	for	Chapter	11	bankruptcy	protection

How many tags can a word have?

- Words in Penn Treebank with the most unique tags:

7 down	6 back
6 that	5 vs.
6 set	5 the
6 put	5 spread
6 open	5 split
6 hurt	5 say
6 cut	
6 bet	

How many tags can a word have?

- Tag counts for *down*:

```
353 down  RB  adverb
214 down  RP  particle
142 down  IN  preposition
 10 down  JJ  adjective
  1 down  VBP verb (past tense)
  1 down  RBR comparative adverb
  1 down  NN  singular noun
```

RP tag: particle

- Test for verb particle:
- Can you insert a modifier between the verb and its particle without it sounding weird?
  - `take the trash out immediately`
  - `*take the trash immediately out`

  - `take the trash outside immediately`
  - `take the trash immediately outside`
- *out* is a particle here, while *outside* is not

Universal Tag Set

- Many use smaller sets of coarser tags
- E.g., "universal tag set" containing 12 tags:
  - noun, verb, adjective, adverb, pronoun, determiner/article, adposition (preposition or postposition), numeral, conjunction, particle, punctuation, other

sentence:	The	oboist	Heinz	Holliger	has	taken	a	hard	line	about	the	problems	.
original:	DT	NN	NNP	NNP	VBZ	VBN	DT	JJ	NN	IN	DT	NNS	.
universal:	DET	NOUN	NOUN	NOUN	VERB	VERB	DET	ADJ	NOUN	ADP	DET	NOUN	.

Figure 1: Example English sentence with its language specific and corresponding universal POS tags.

Petrov, Das, McDonald (2011)

word sense vs. part-of-speech

	word sense	**part-of-speech**
semantic or syntactic?	semantic: indicates meaning of word in its context	syntactic: indicates function of word in its context
number of categories	$\|V\|$ words, ~5 senses each → $5\|V\|$ categories!	typical POS tag sets have 12 to 45 tags
inter-annotator agreement	low; some sense distinctions are highly subjective	high; relatively few POS tags and function is relatively shallow / surface-level
independent or joint classification of nearby words?	independent: can classify a single word based on context words; structured prediction is rarely used	joint: strong relationship between tags of nearby words; structured prediction often used

How might POS tags be useful?

- ❑ Text classification
- ❑ Machine translation
- ❑ Question answering
- ❑ Speech synthesis (pronounce “contract”)
- ❑ …

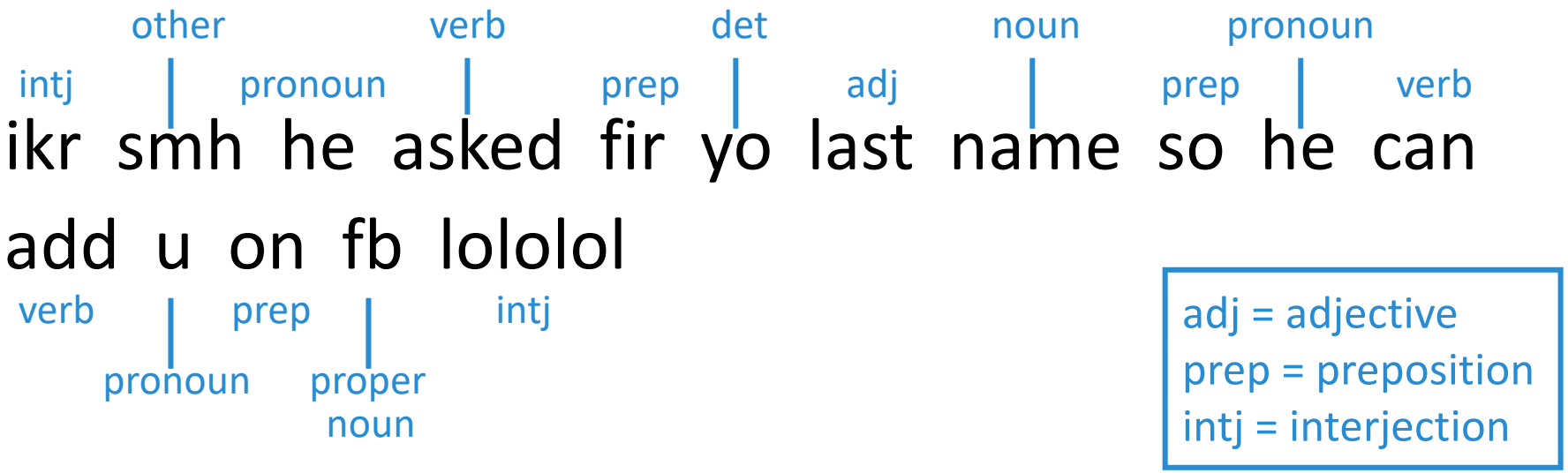
Models for POS Tagging

- Today we'll discuss simple models that do not use structured prediction
- These are often called "local" models
- They predict a tag for each word in a sequence, (and can use the entire word sequence to make each prediction)
- But they do not use information about previous *predictions* to make later predictions
- By contrast, **structured prediction**:
  - predict structures
  - or: make multiple predictions jointly

Twitter Part-of-Speech Tagging

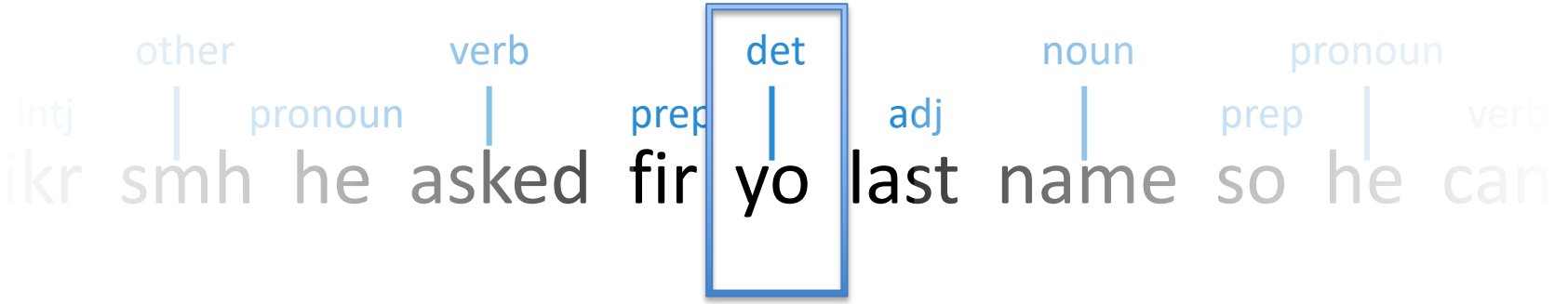
- We removed some fine-grained POS tags, then added Twitter-specific tags:
  - hashtag
  - @-mention
  - URL / email address
  - emoticon
  - Twitter discourse marker
  - other (multi-word abbreviations, symbols, garbage)

Feed-Forward Neural Networks for Twitter POS Tagging



- We use a neural network classifier to predict a word's POS tag based on its context

Feed-Forward Neural Networks for Twitter POS Tagging

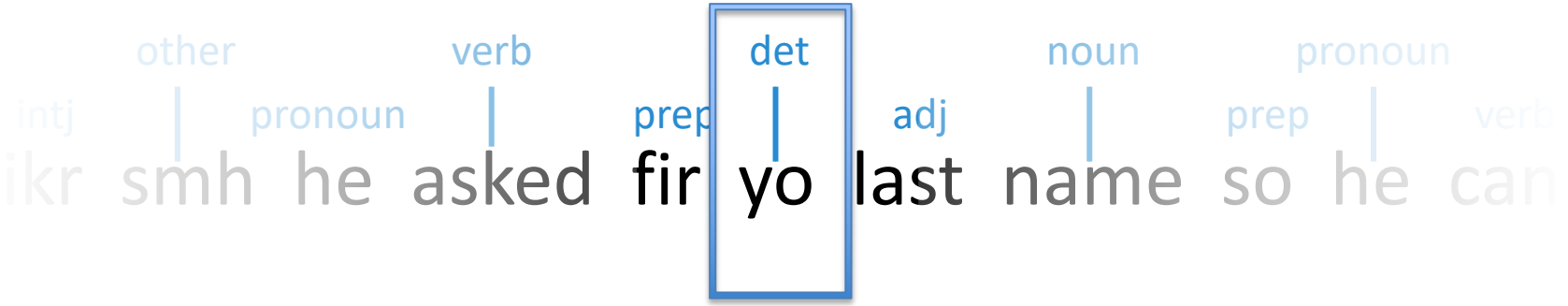


- E.g., predict tag of *yo* given context
- What should the input **x** be to the neural network?

$$\mathbf{x} = [0.4 \;\; 0.1 \;\; \ldots \;\; 0.9]^\top$$

word vector for *yo*

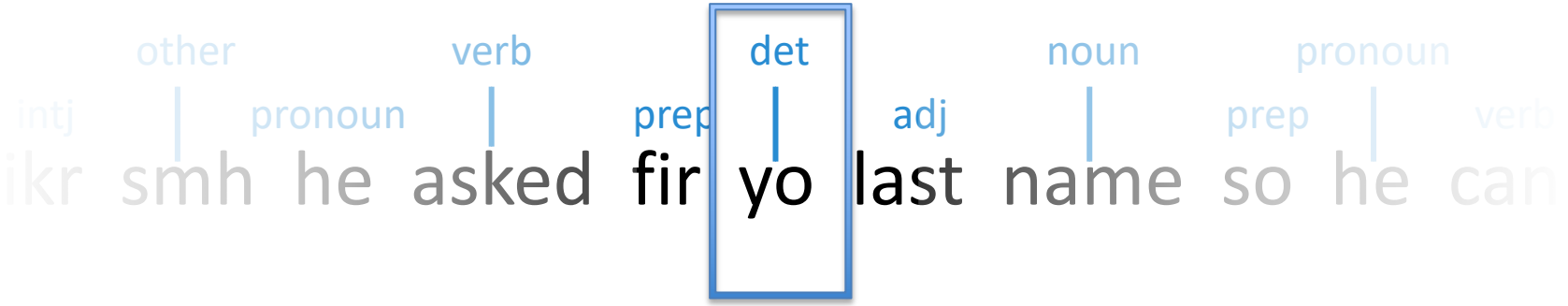
Feed-Forward Neural Networks for Twitter POS Tagging



- E.g., predict tag of *yo* given context
- What should the input **x** be to the neural network?

$$\mathbf{x} = [\underbrace{-0.2 \;\; 0.5 \;\; \ldots \;\; 0.8}_{\text{word vector for } \textit{fir}} \;\; \underbrace{0.4 \;\; 0.1 \;\; \ldots \;\; 0.9}_{\text{word vector for } \textit{yo}}]^\top$$

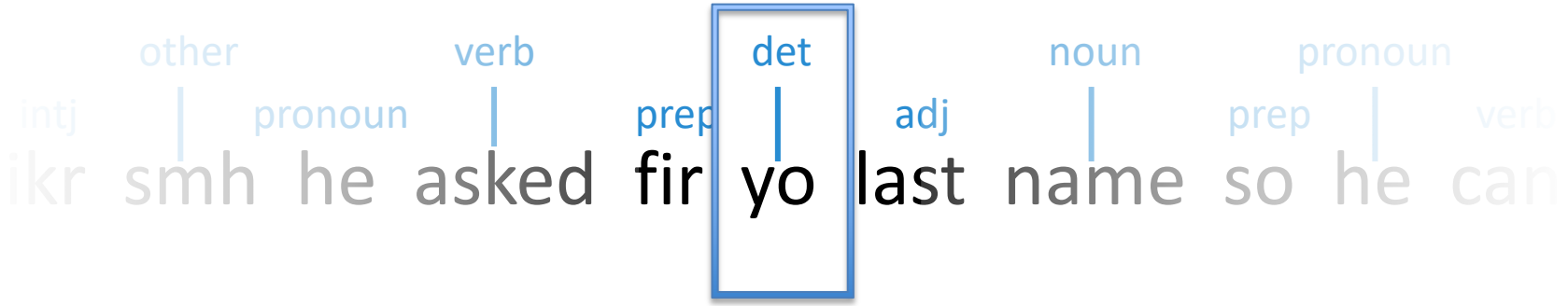
Feed-Forward Neural Networks for Twitter POS Tagging



- When using word vectors as part of input, we can also treat them as more parameters to be learned!
- This is called "updating" or "fine-tuning" the vectors (since they are initialized using something like *word2vec*)

$$\mathbf{x} = [\underbrace{-0.2 \;\; 0.5 \;\; \ldots \;\; 0.8}_{\text{word vector for } \textit{fir}} \;\; \underbrace{0.4 \;\; 0.1 \;\; \ldots \;\; 0.9}_{\text{word vector for } \textit{yo}}]^\top$$

Feed-Forward Neural Networks for Twitter POS Tagging



- Let's use the center word + two words to the right:

$$\mathbf{x} = [0.4 \ \ldots \ 0.9 \ \ 0.2 \ \ldots \ 0.7 \ \ 0.3 \ \ldots \ 0.6]^\top$$

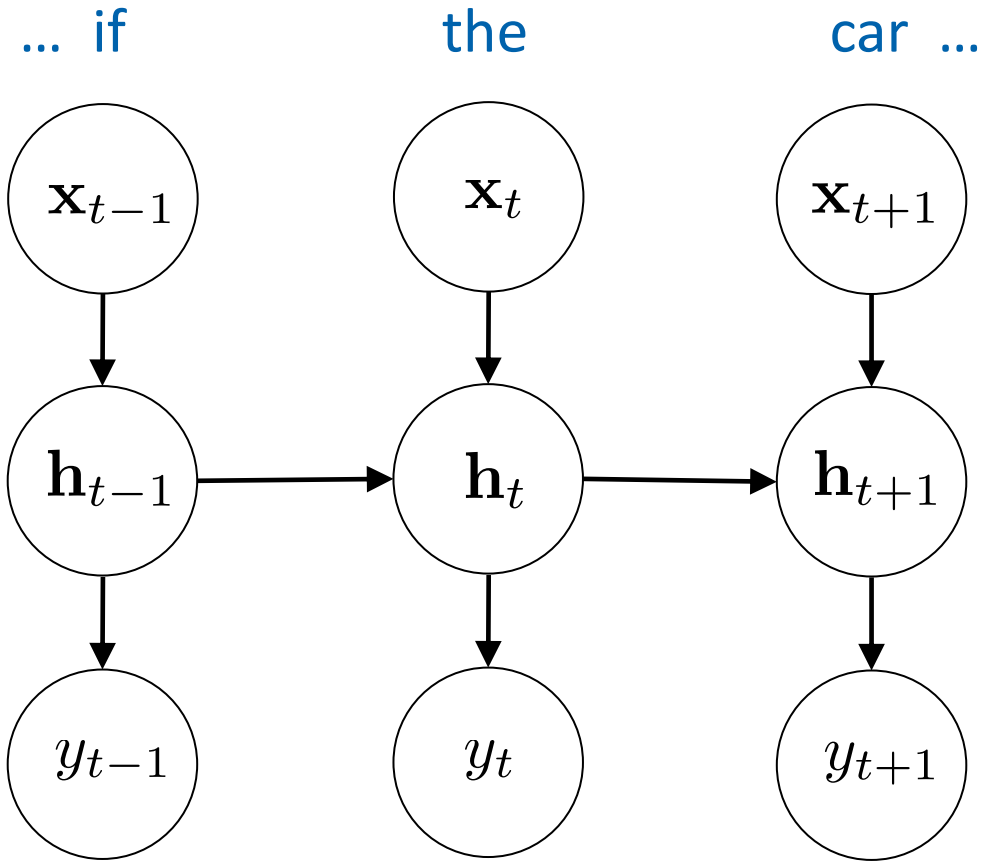
vector for *yo* vector for *last* vector for *name*

- If *name* is to the right of *yo*, then *yo* is probably a form of *your*
- But our **x** above uses separate dimensions for each position!
  - i.e., name is two words to the right
  - What if name is one word to the right?

Feed-Forward Neural Networks for POS Tagging

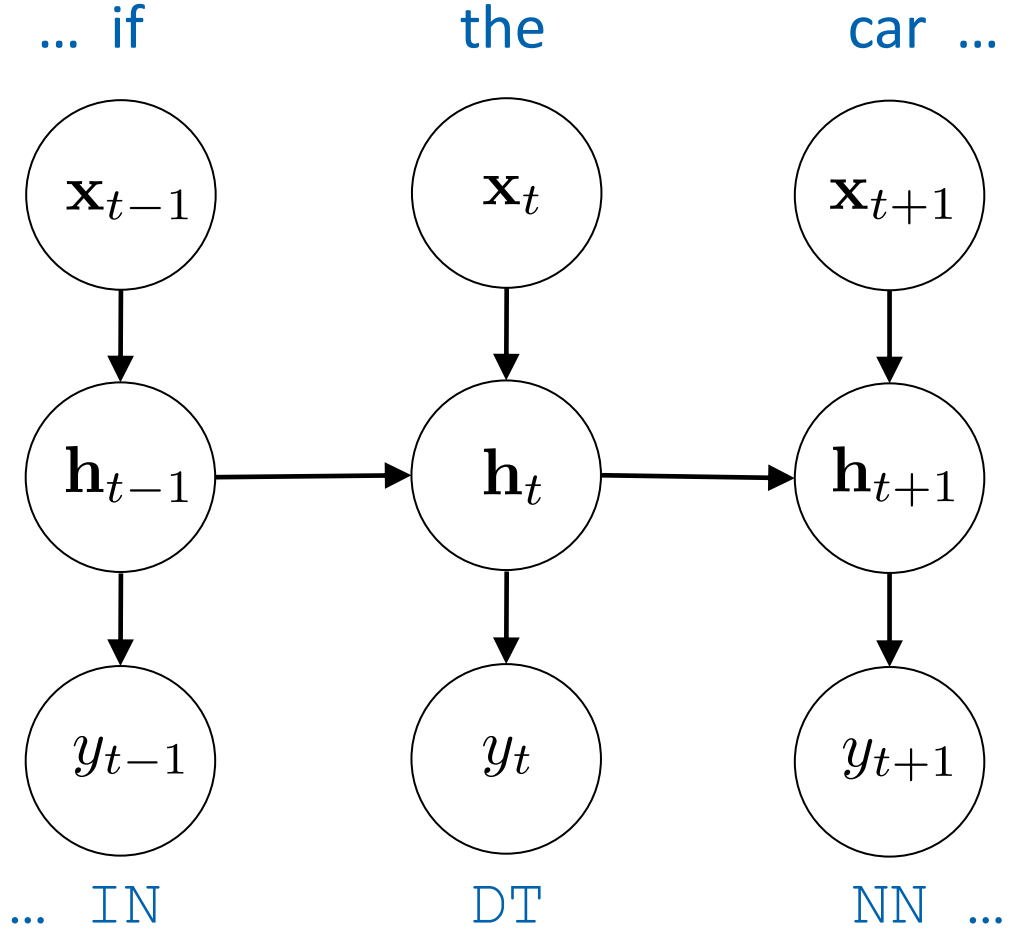
- Feed-forward networks are OK for tagging
- They tend to work best with very small contexts (e.g., 1 word to left & right)
- With larger windows, probably not enough data to learn a good model
- Also, distant words not very informative for POS tagging
- Can also use convolutional networks defined on a window centered on the target word

RNNs for Part-of-Speech Tagging



- input: a word sequence

RNNs for Part-of-Speech Tagging



- target output at each position: POS tag for corresponding word

RNN Taggers

- RNN POS taggers are simple and effective
- Most common is to use some sort of bidirectional RNN, like a BiLSTM or BiGRU

RNN Taggers

- Note: RNN taggers are not structured predictors
- Yes, a structure is being predicted, but predictions for neighboring words are independent!
- BiRNN taggers do compute input representations that depend on the sentence context
- But they do not make any predictions jointly; each prediction is independent of all others

Sequence Labeling

- Roughly: for each item in an input sequence, predict a label
- Many sequence labeling tasks in NLP and other areas
  - computational biology, speech processing, video processing, etc.
- Related class of tasks: segmentation, possibly with labeling of segments

Formulating segmentation tasks as sequence labeling via B-I-O labeling:

Named Entity Recognition

O	O	O	B-PERSON	I-PERSON	O	O	O
Some	questioned	if	Tim	Cook	’s	first	product

O	O	O	O	O	O	B-ORGANIZATION	O
would	be	a	breakaway	hit	for	Apple	.

B = “begin”
I = “inside”
O = “outside”

- There are many downloadable part-of-speech taggers and named entity recognizers:
  - Stanford POS tagger, NER labeler
  - TurboTagger, TurboEntityRecognizer
  - Illinois Entity Tagger
  - CMU Twitter POS tagger
  - Alan Ritter's Twitter POS/NER labeler

Stanford CoreNLP

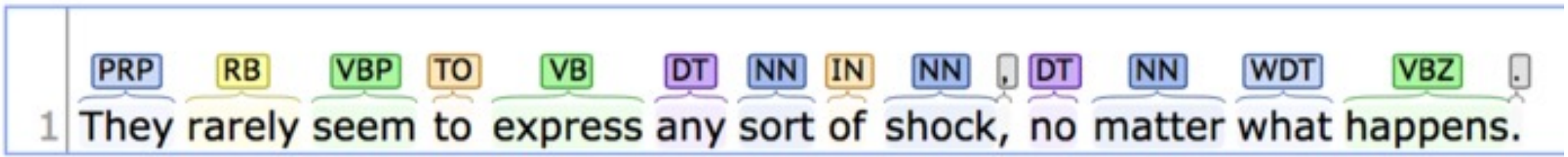
Output format: Visualise

Please enter your text here:

They rarely seem to express any sort of shock, no matter what happens.

Submit Clear

Part-of-Speech:



Stanford Named Entity Tagger

Classifier: english.all.3class.distsim.crf.ser.gz

Output Format: highlighted

Preserve Spacing: no

Please enter your text here:

Some questioned if Tim Cook's first product would be a breakaway hit for Apple.

Submit Clear

Some questioned if Tim Cook's first product would be a breakaway hit for Apple.

Potential tags:

ORGANIZATION

LOCATION

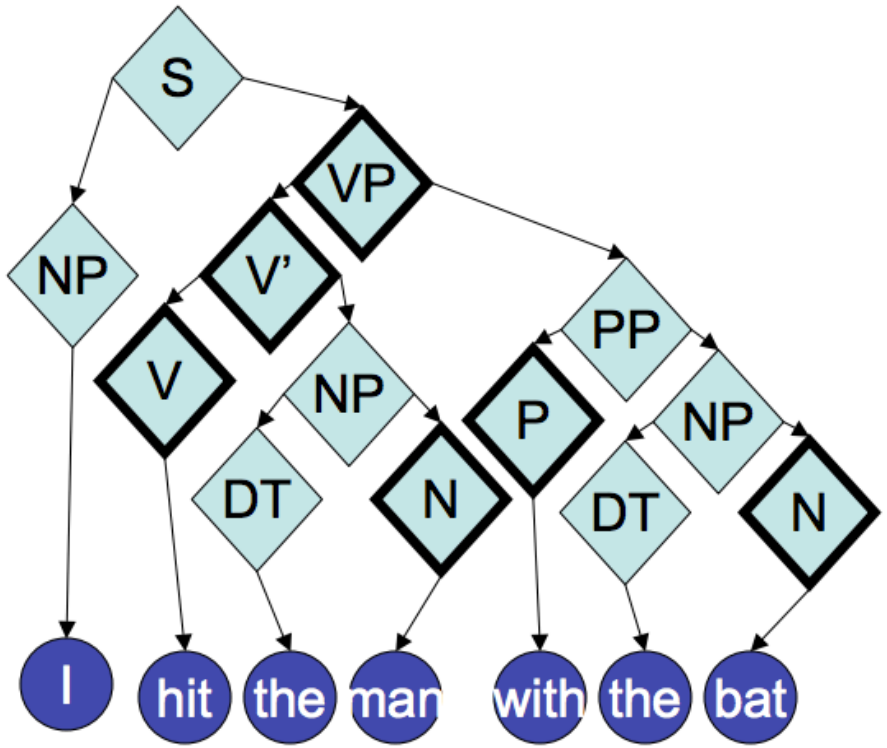
PERSON

Syntax and Syntactic Parsing

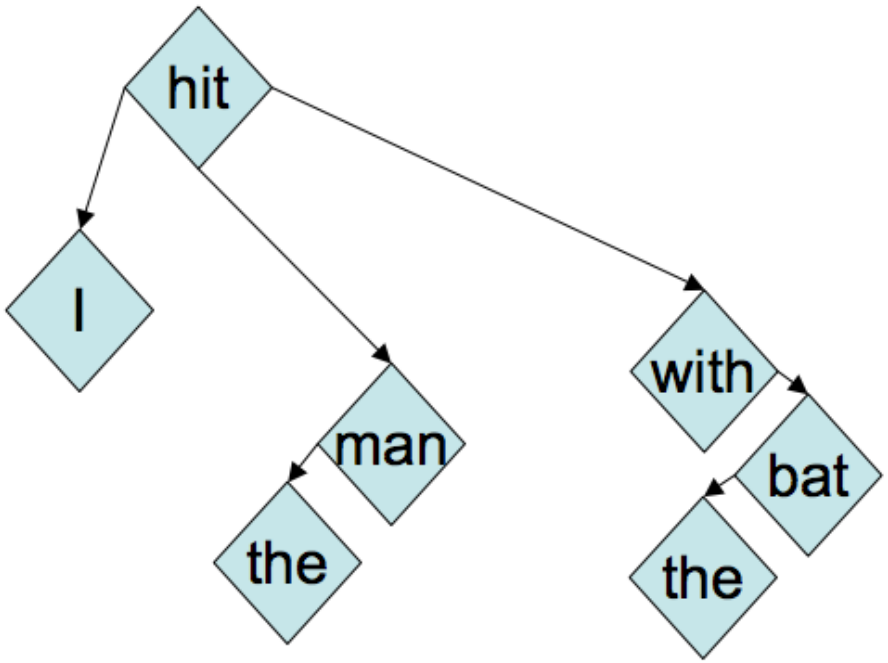
What is Syntax?

- Rules, principles, processes that govern sentence structure of a language
- Can differ widely among languages
- But every language has systematic structural principles

constituent parse:

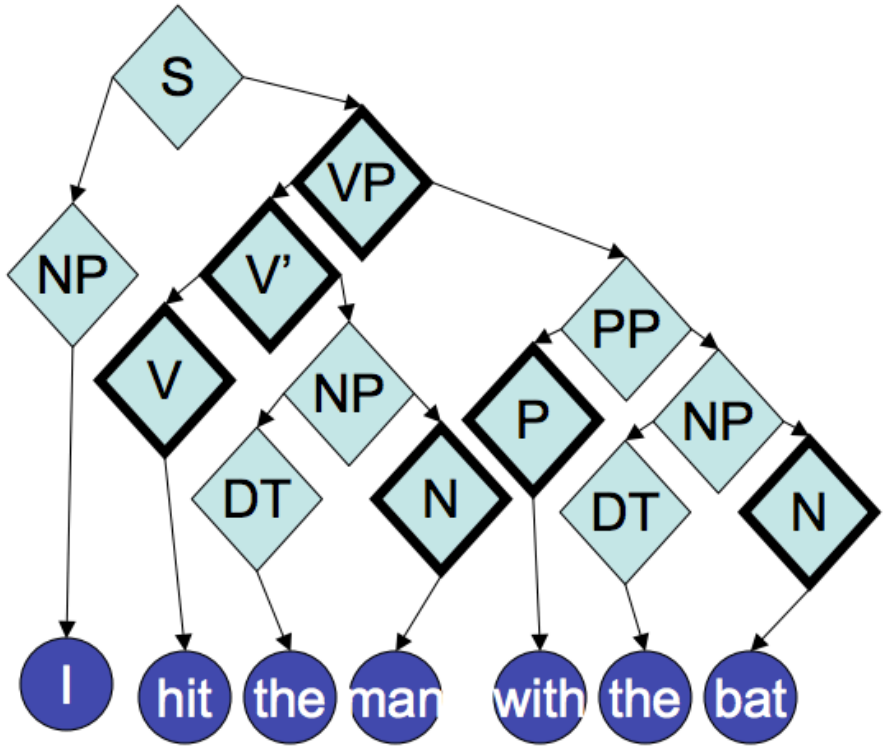


dependency parse:

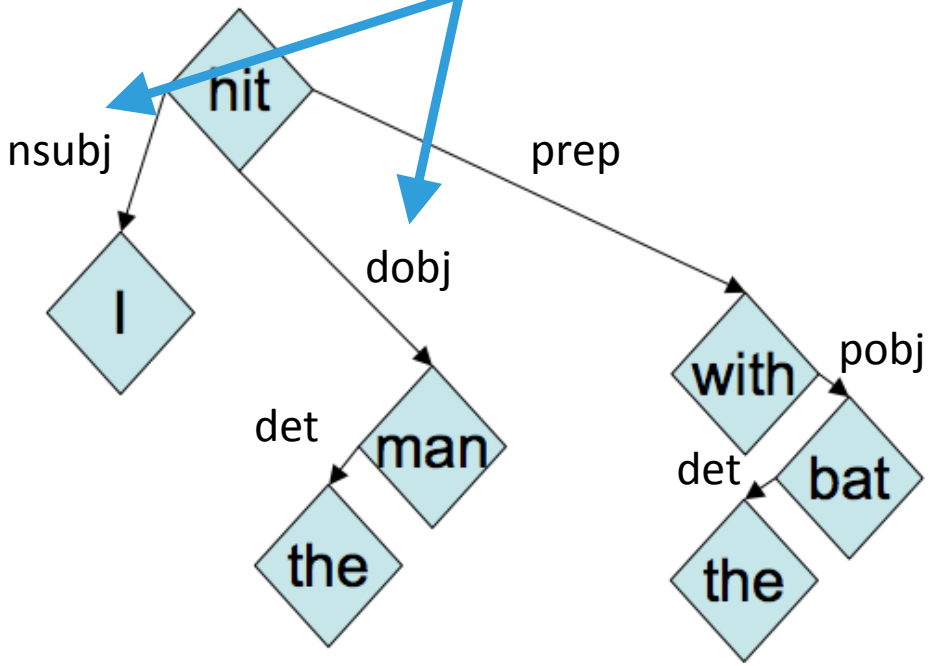


from Noah Smith

constituent parse:



captures some semantic relationships



nsubj = “nominal subject”
dobj = “direct object”
prep = “preposition modifier”
pobj = “object of preposition”
det = “determiner”

from Noah Smith

Machine Translation

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**).

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

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

– Rousseau

People rely on machine translation!

People rely on machine translation!

1990s-2010s: Statistical Machine Translation

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

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

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

$$= \text{argmax}_y \underbrace{P(x|y)}_{\text{Translation Model}} \underbrace{P(y)}_{\text{Language Model}}$$

Translation Model

Models how words and phrases should be translated (*fidelity*). Learnt from parallel data.

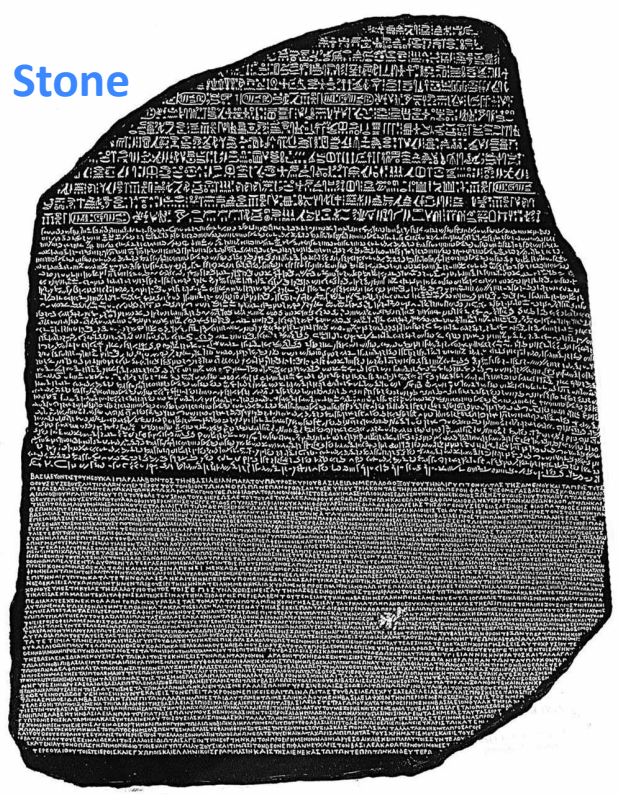
Language Model

Models how to write good English (*fluency*). Learnt from monolingual data.

Statistical Machine Translation

- Question: How to learn translation model $P(x|y)$?
- First, need large amount of **parallel data**
  (e.g., pairs of human-translated French/English sentences)

The Rosetta Stone



Ancient Egyptian

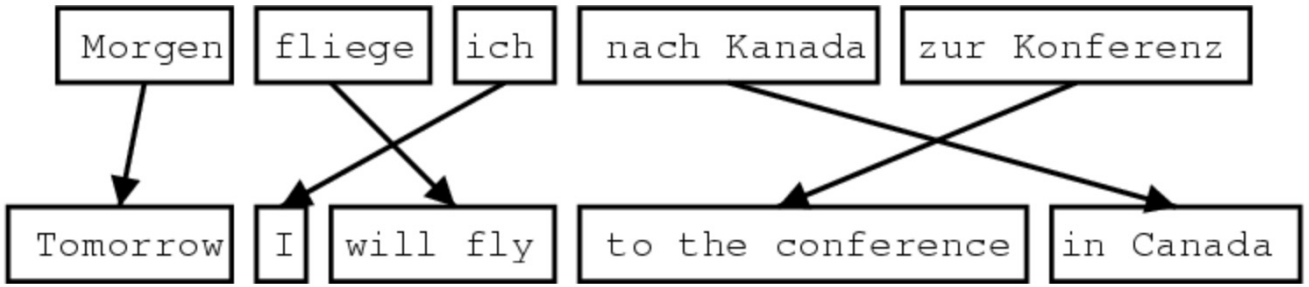
Demotic

Ancient Greek

Learning alignment for SMT

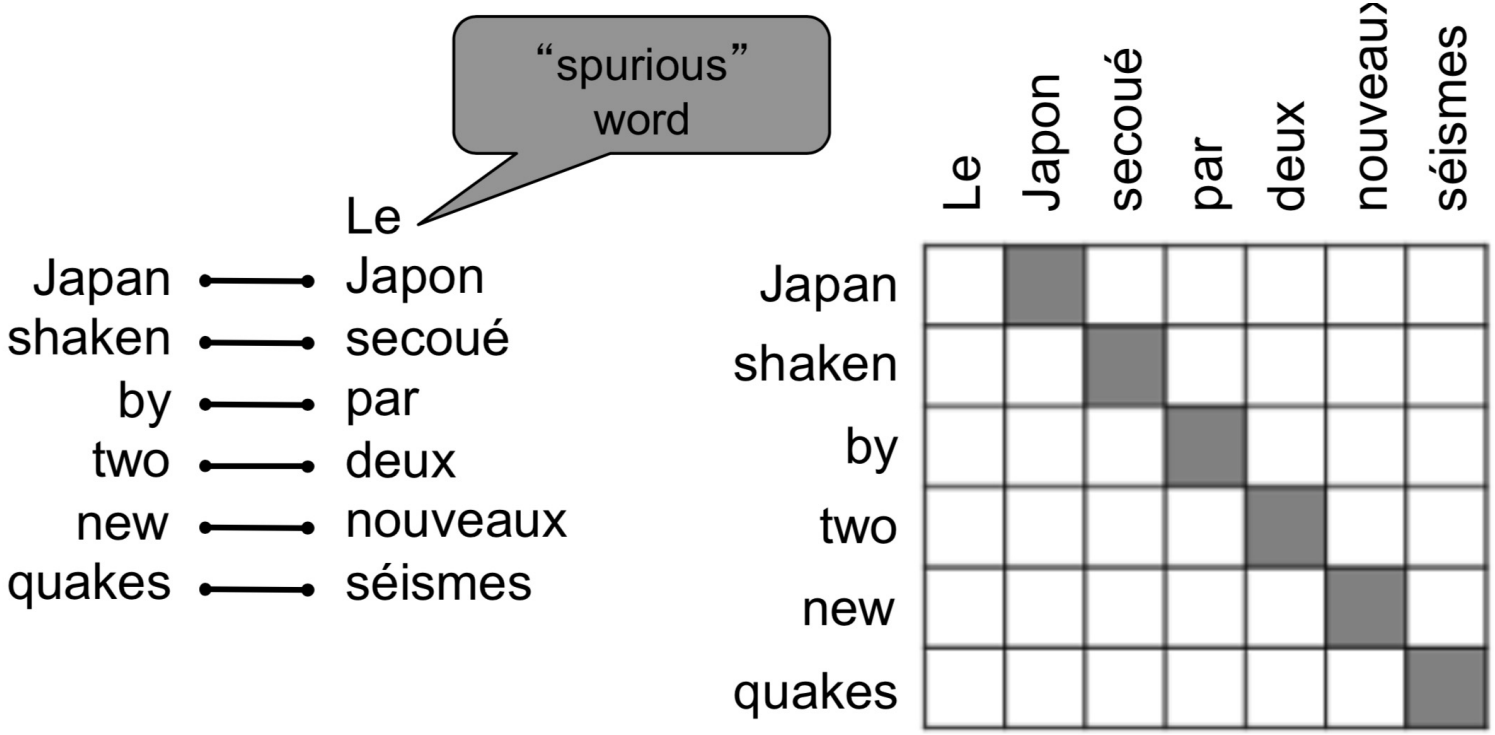
- Question: How to learn translation model $P(x|y)$ from the parallel corpus?
- Break it down further: Introduce latent *a* variable into the model: $P(x, a|y)$

where *a* is the **alignment**, i.e. word-level correspondence between source sentence *x* and target sentence *y*



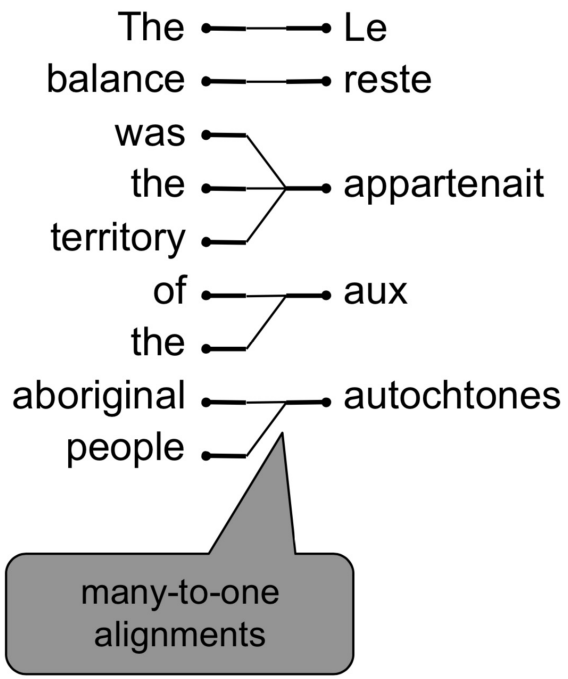
What is alignment?

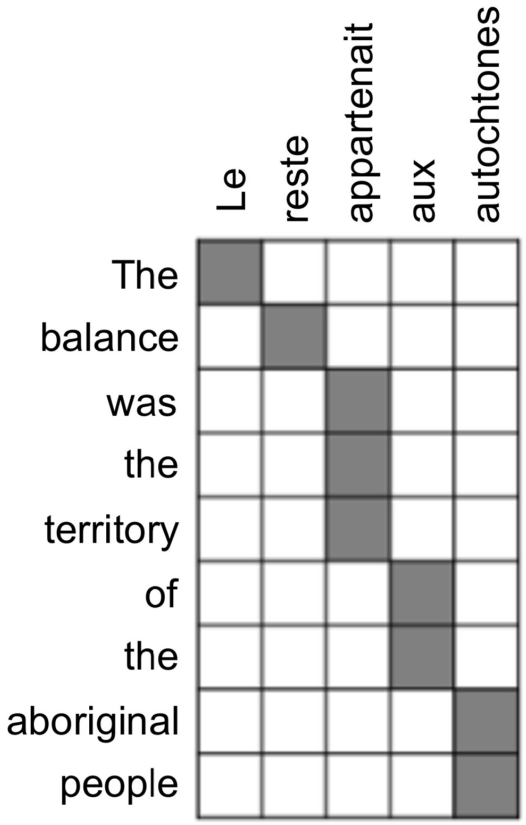
- Alignment is the **correspondence between particular words** in the translated sentence pair.
  - **Typological differences** between languages lead to complicated alignments!
  - Note: Some words have **no counterpart**



Alignment is complex

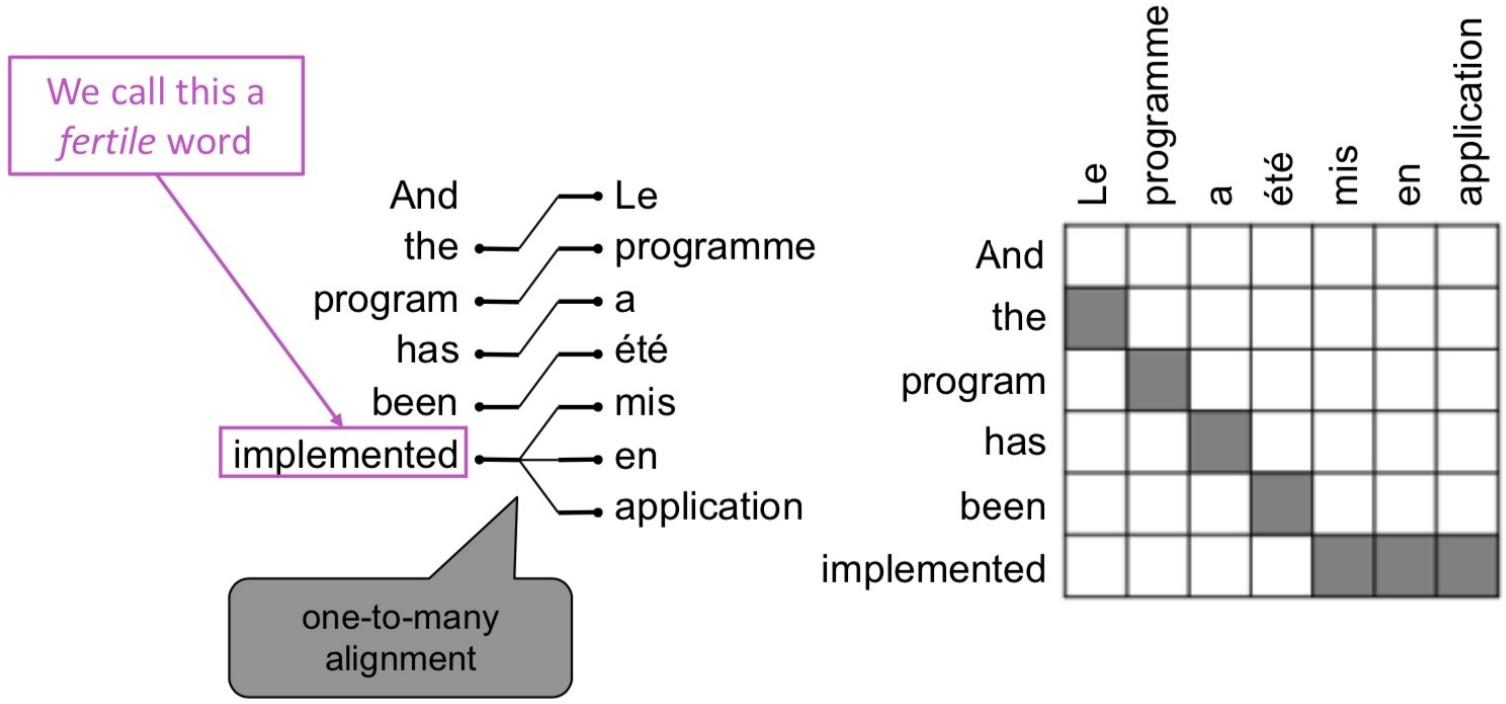
- Alignment can be **many-to-one**





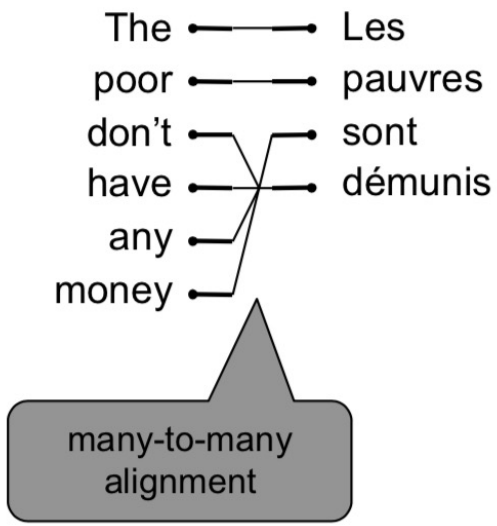
Alignment is complex

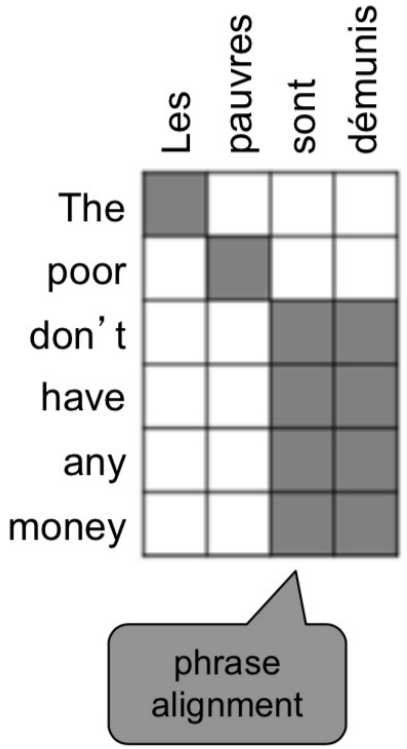
- Alignment can be **one-to-many**



Alignment is complex

- Alignment can be **many-to-many** (phrase-level)

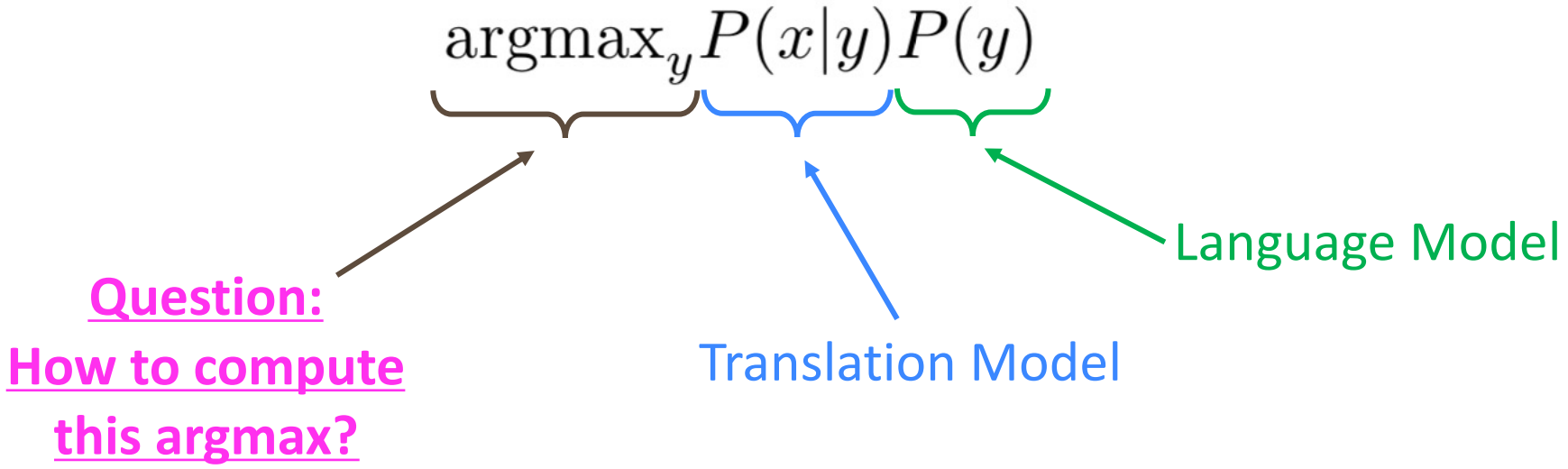




Alignment is complex

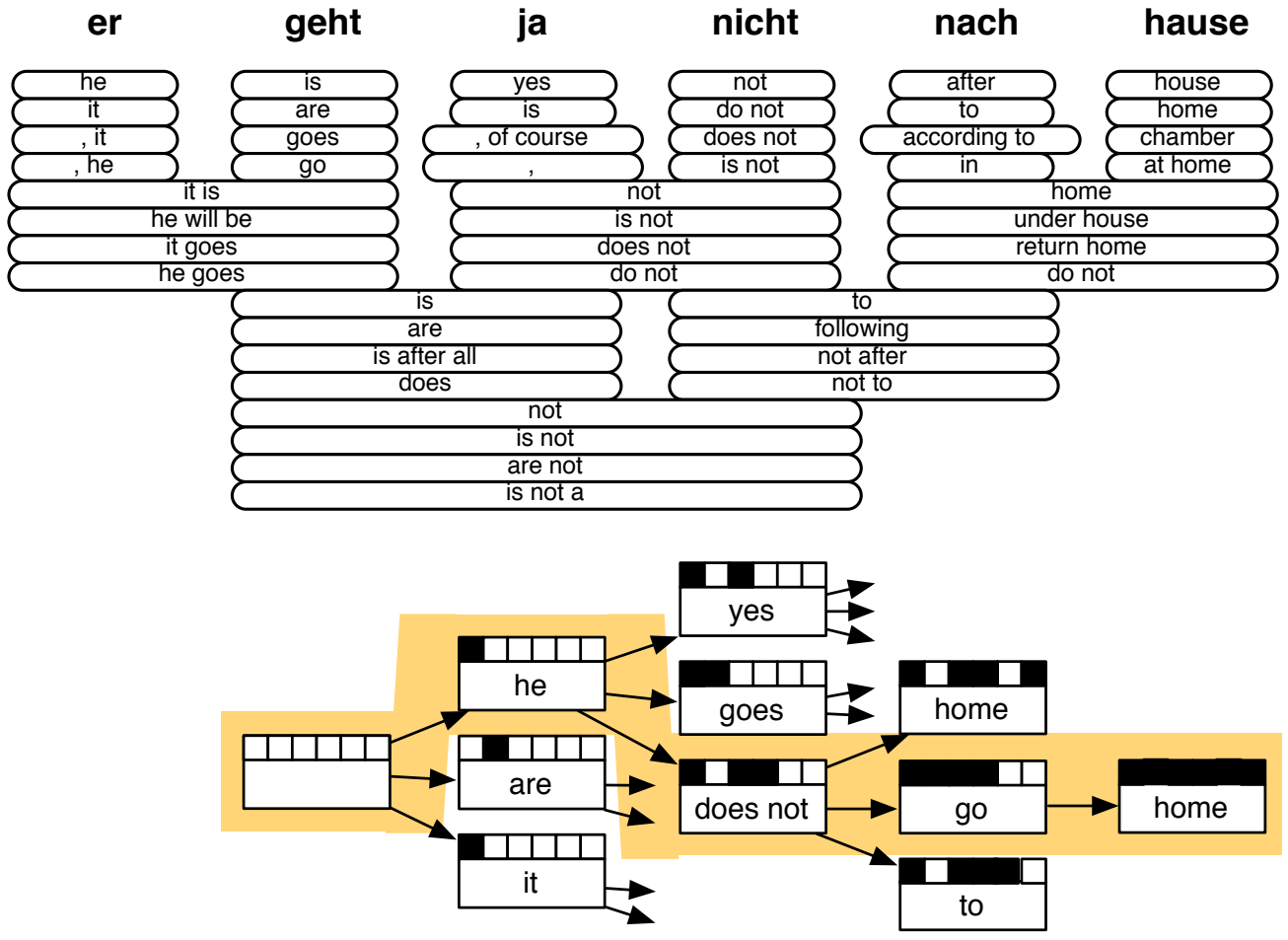
- We learn $P(x, a|y)$ as a combination of several factors, including:
  - Probability of particular words aligning (also depends on position in sent)
  - Probability of particular words having a particular fertility (number of corresponding words)
  - etc.
- Alignments *a* are **latent variables**: They aren't explicitly specified in the data!
  - Require the use of special learning algorithms (like Expectation-Maximization) for learning the parameters of distributions with latent variables

Decoding for SMT



- We could enumerate every possible *y* and calculate the probability?
  → Too expensive!
- Answer: Impose strong **independence assumptions** in model, use dynamic programming for globally optimal solutions (e.g. Viterbi algorithm).
- This process is called **decoding**

Decoding for SMT



Source: ”Statistical Machine Translation", Chapter 6, Koehn, 2009.

Statistical Machine Translation

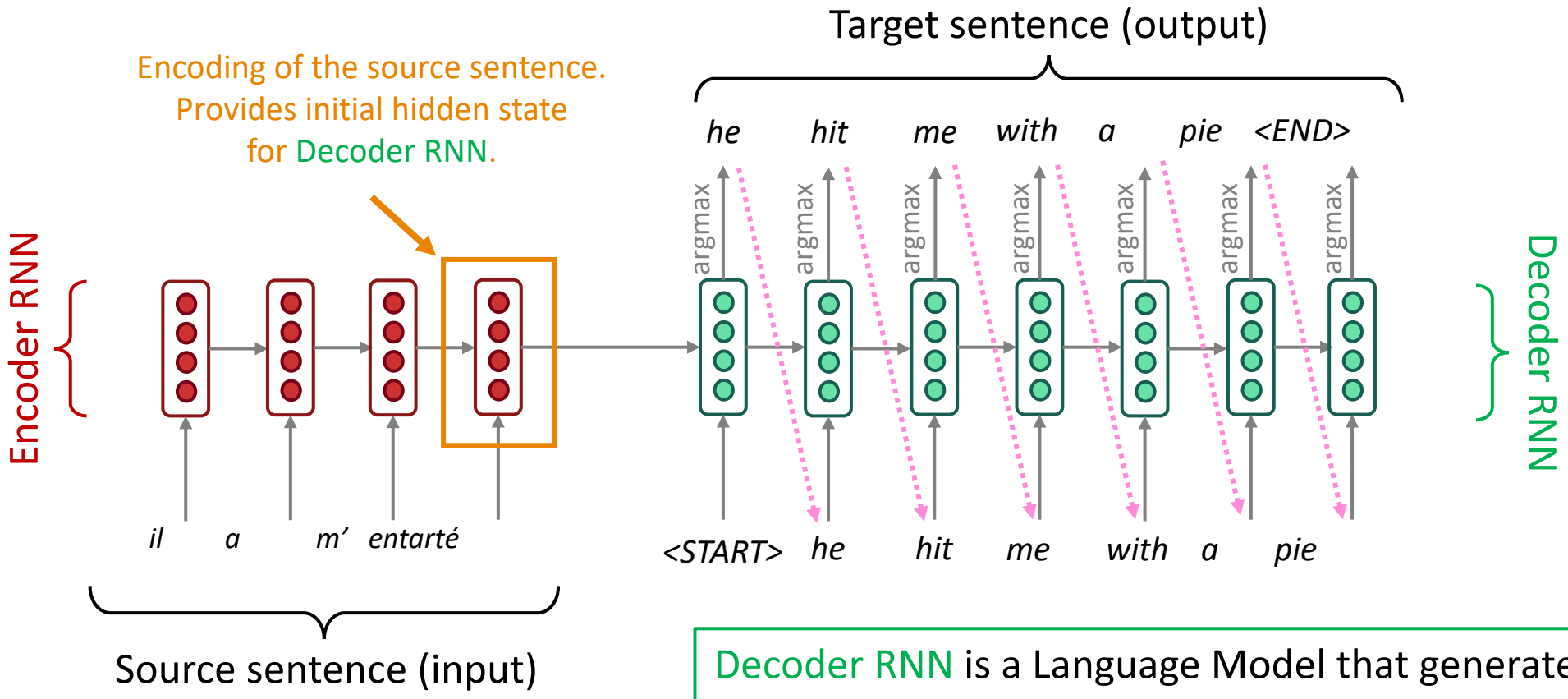
- SMT was a **huge research field**
- The best systems were **extremely complex**
  - Hundreds of important details we haven't mentioned here
- Systems had many **separately-designed subcomponents**
  - Lots of **feature engineering**
    - Need to design features to capture particular language phenomena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

What is Neural Machine Translation?

- **Neural Machine Translation (NMT)** is a way to do Machine Translation with a *single end-to-end neural network*
- The neural network architecture is called a **sequence-to-sequence** model (aka **seq2seq**) and it involves ***two*** **RNNs**

Neural Machine Translation (NMT)

The sequence-to-sequence model



Encoder RNN produces an encoding of the source sentence.

Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

Note: This diagram shows **test time** behavior: decoder output is fed in ⋯→ as next step's input

From Stanford 224n course

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." *Advances in neural information processing systems* 27 (2014).

Sequence-to-sequence is versatile!

- The general notion here is an **encoder-decoder** model
  - One neural network takes input and produces a neural representation
  - Another network produces output based on that neural representation
  - If the input and output are sequences, we call it a seq2seq model
- 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)

Neural Machine Translation (NMT)

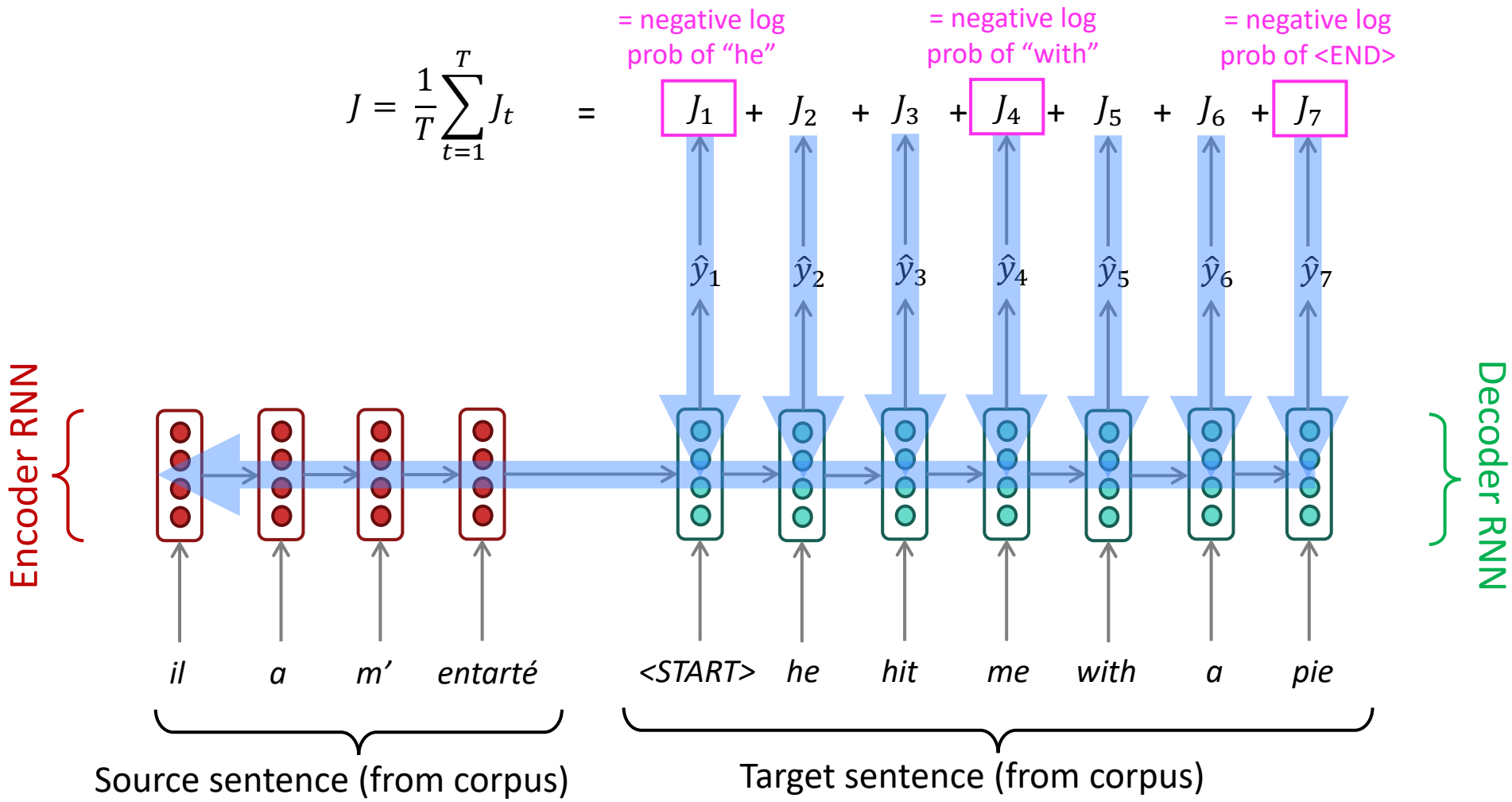
- The **sequence-to-sequence** model is an example of a **Conditional Language Model**
  - Language Model because the decoder is predicting the next word of the target sentence *y*
  - Conditional because its predictions are also conditioned on the source sentence *x*
- NMT directly calculates $P(y|x)$:

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

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

- Question: How to train an NMT system?
- (Easy) Answer: Get a big parallel corpus...
  - But there is now exciting work on "unsupervised NMT", data augmentation, etc.

Training a Neural Machine Translation system

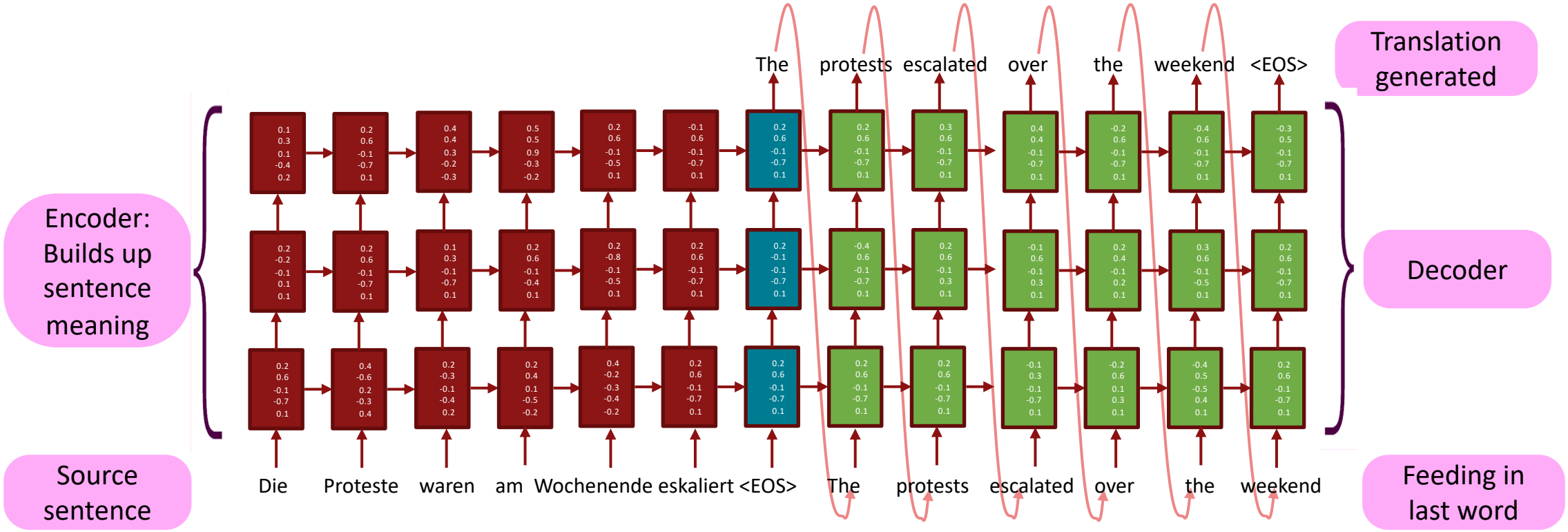


Seq2seq is optimized as a **single system.** Backpropagation operates "*end-to-end*".

Multi-layer deep encoder-decoder machine translation net

[Sutskever et al. 2014; Luong et al. 2015]

The hidden states from RNN layer *i* are the inputs to RNN layer *i*+1



Conditioning = Bottleneck

Multi-layer RNNs in practice

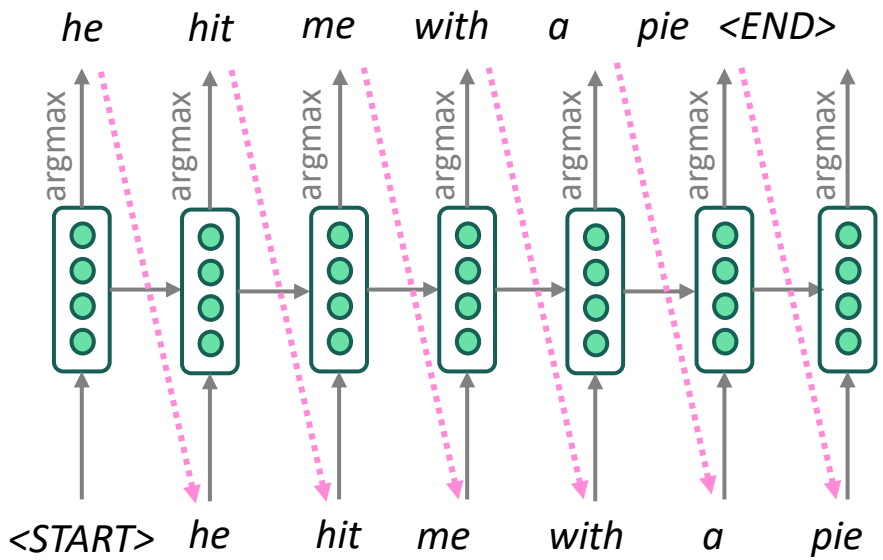
- Multi-layer or stacked RNNs allow the network to compute **more complex representations**
  - The lower RNNs should compute lower-level features and the higher RNNs should compute higher-level features.

- **High-performing RNNs are usually multi-layer** (but aren't as deep as convolutional or feed-forward networks)
- For example: In a 2017 paper, Britz et al. find that for Neural Machine Translation, 2 to 4 layers is best for the encoder RNN, and 4 layers is best for the decoder RNN
  - Often 2 layers is a lot better than 1, and 3 might be a little better than 2
  - Usually, skip-connections/dense-connections are needed to train deeper RNNs (e.g., 8 layers)

- **Transformer**-based networks (e.g., BERT) are usually deeper, like 12 or 24 layers.
  - You will learn about Transformers later; they have a lot of skipping-like connections

Greedy decoding

- We saw how to generate (or "decode") the target sentence by taking argmax on each step of the decoder



- This is **greedy decoding** (take most probable word on each step)
- Problems with this method?

Greedy decoding

- Greedy decoding has no way to undo decisions!
  - Input: *il a m'entarté* *(he hit me with a pie)*
  - → *he* ____
  - → *he hit* ____
  - → *he hit a* ____ *(whoops! no going back now...)*

- How to fix this?

Exhaustive search decoding

- Ideally, we want to find a (length T) translation *y* that maximizes

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

- We could try computing **all possible sequences** *y*
  - This means that on each step *t* of the decoder, we're tracking V^t possible partial translations, where V is vocab size
  - This $O(V^T)$ complexity is **far too expensive**!

Beam search decoding

- Core idea: On each step of decoder, keep track of the k most probable partial translations (which we call ***hypotheses***)
  - *K* is the **beam size**(in practice around 5 to 10, in NMT)

- A hypothesis $y_1, \ldots, y_t$ has a score which is its log probability:

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

  - Scores are all negative, and higher score is better
  - We search for high-scoring hypotheses, tracking top *k* on each step

- Beam search is **not guaranteed** to find optimal solution
- But **much more efficient** than exhaustive search!

Beam search decoding: example

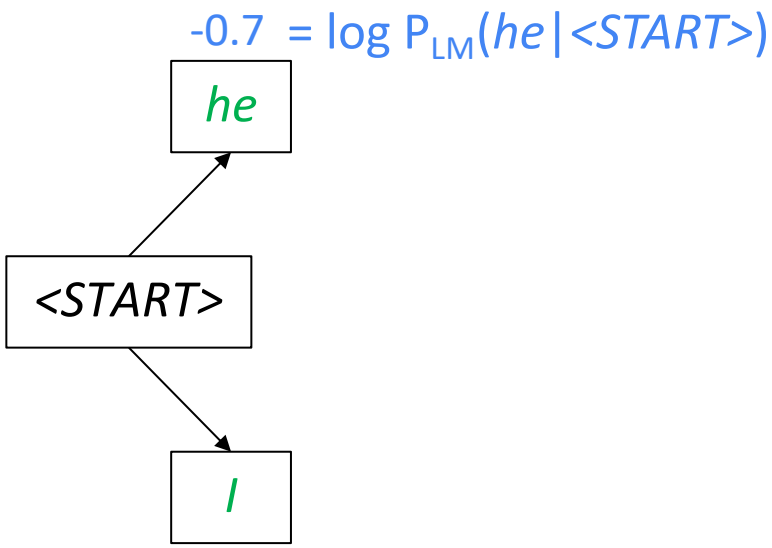
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

<START>

Calculate prob dist of next word

Beam search decoding: example

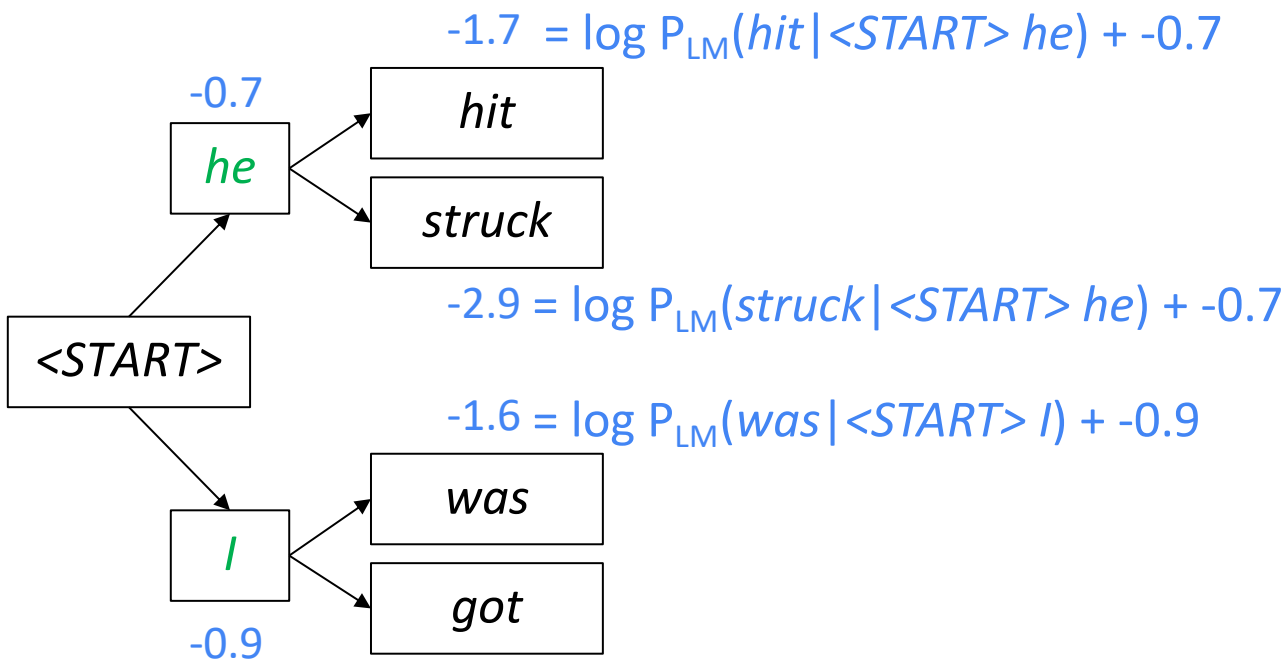
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Take top *k* words and compute scores

Beam search decoding: example

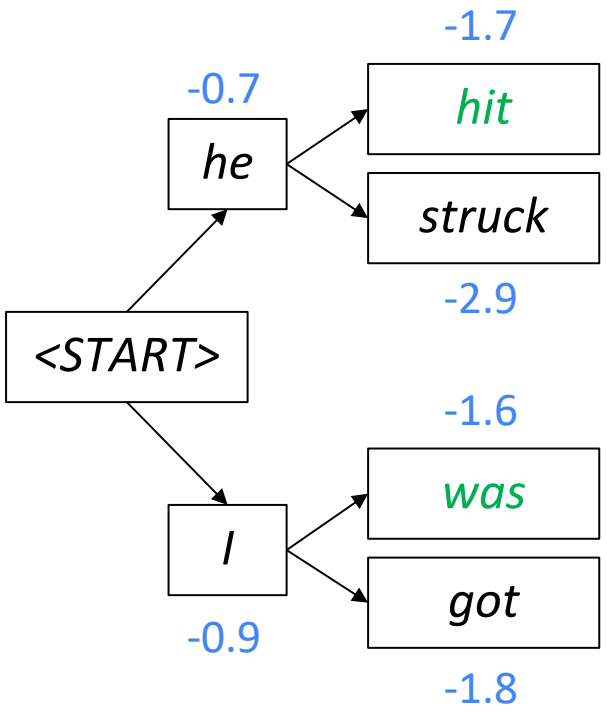
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding: example

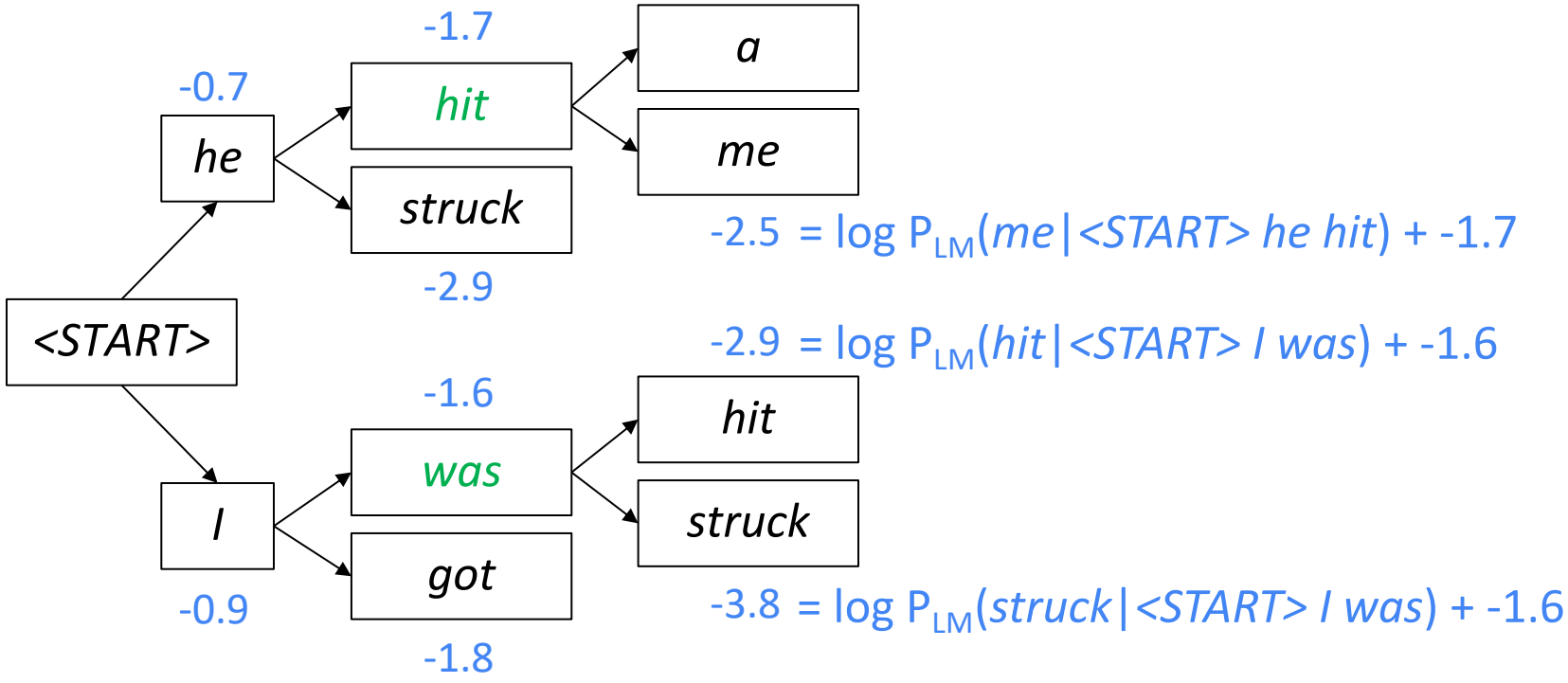
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Of these k^2 hypotheses,
just keep k with highest scores

Beam search decoding: example

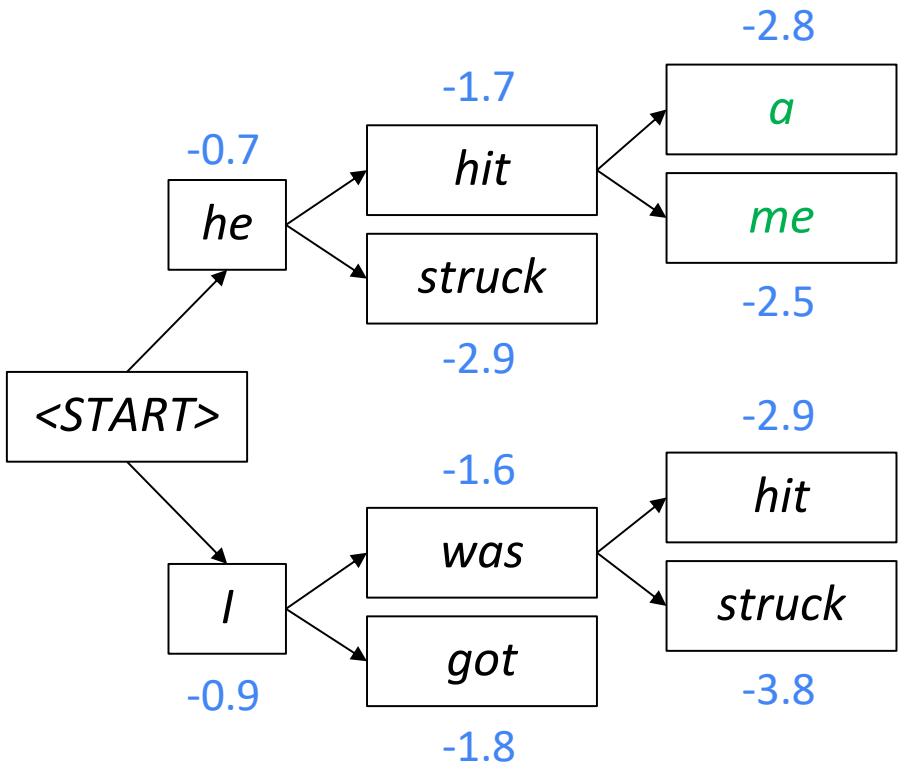
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding: example

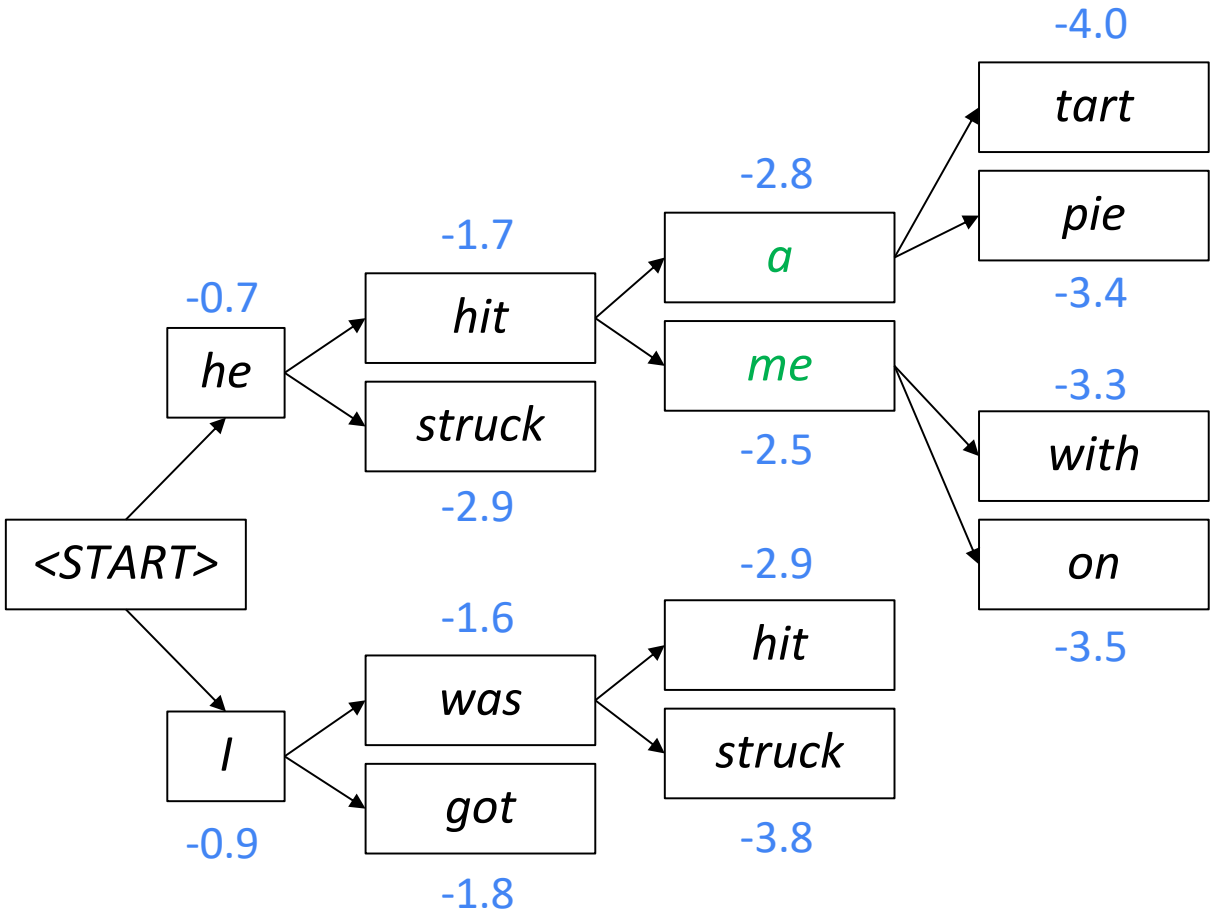
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Of these k^2 hypotheses,
just keep k with highest scores

Beam search decoding: example

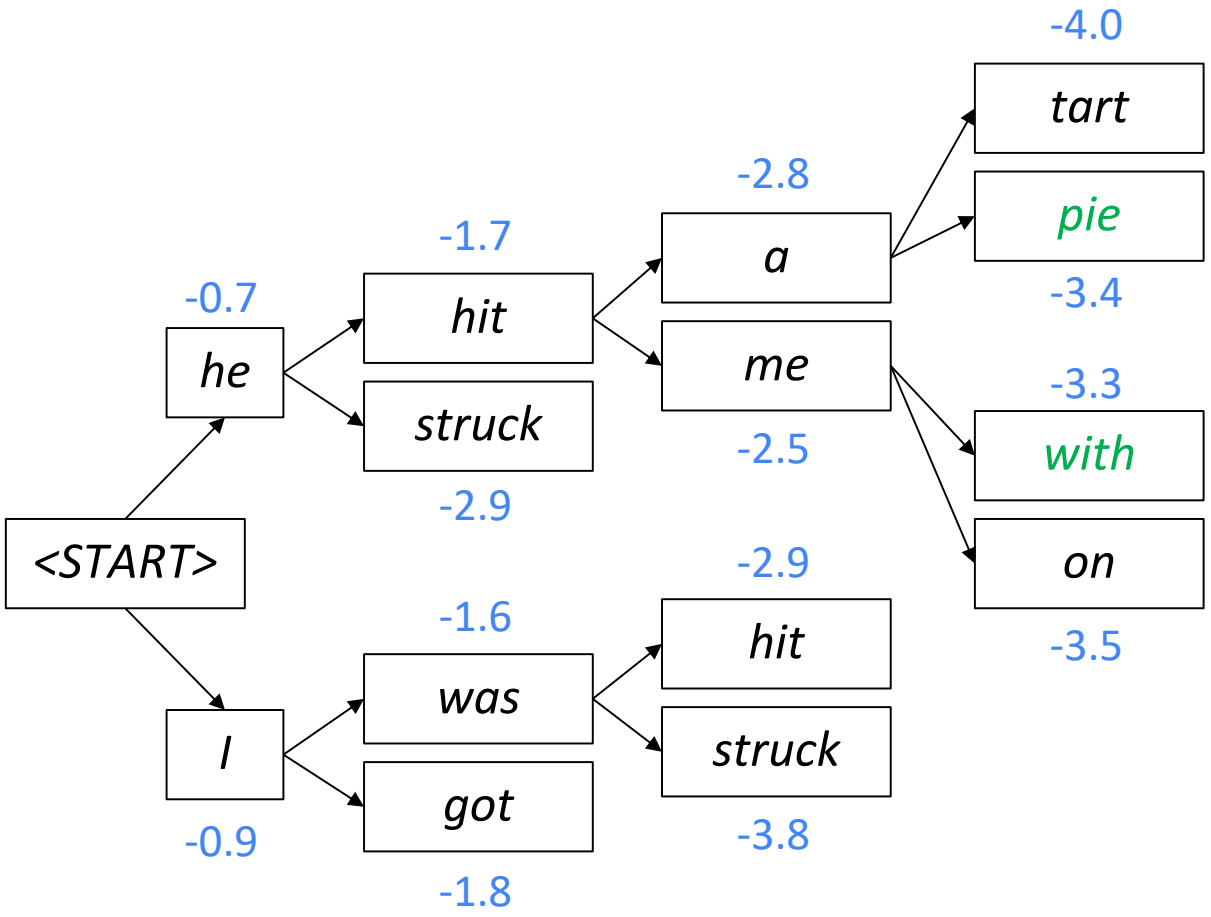
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding: example

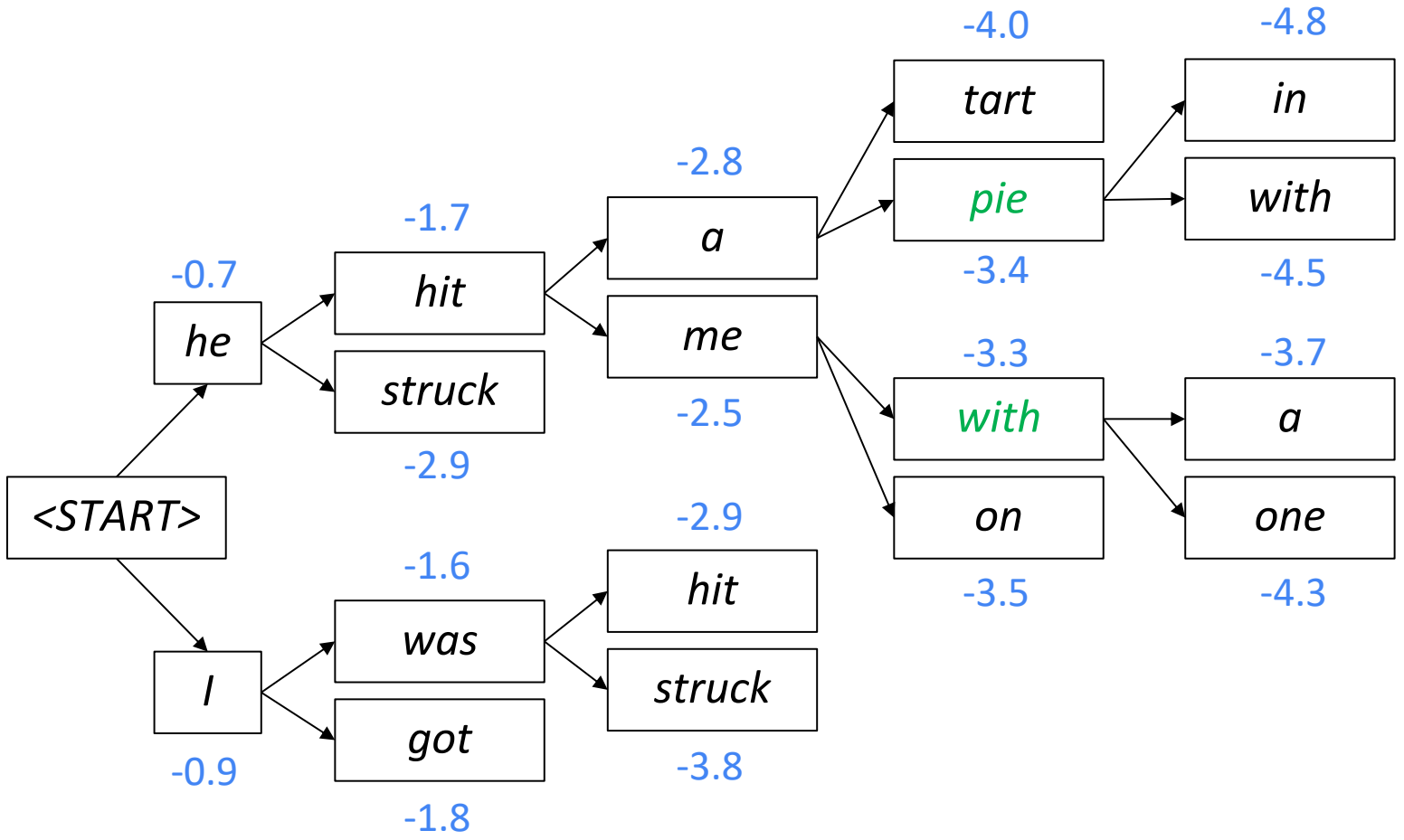
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Of these k^2 hypotheses,
just keep k with highest scores

Beam search decoding: example

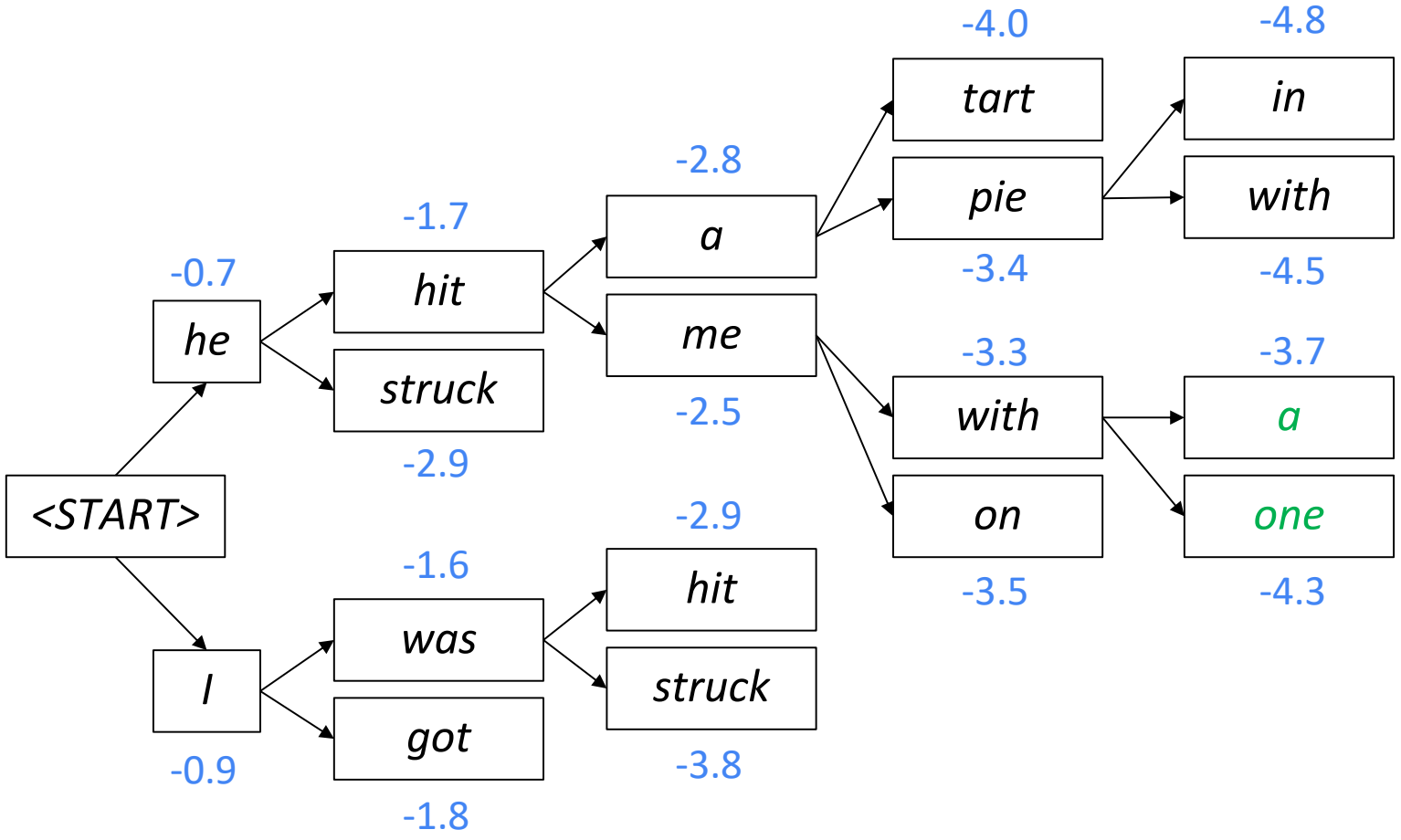
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding: example

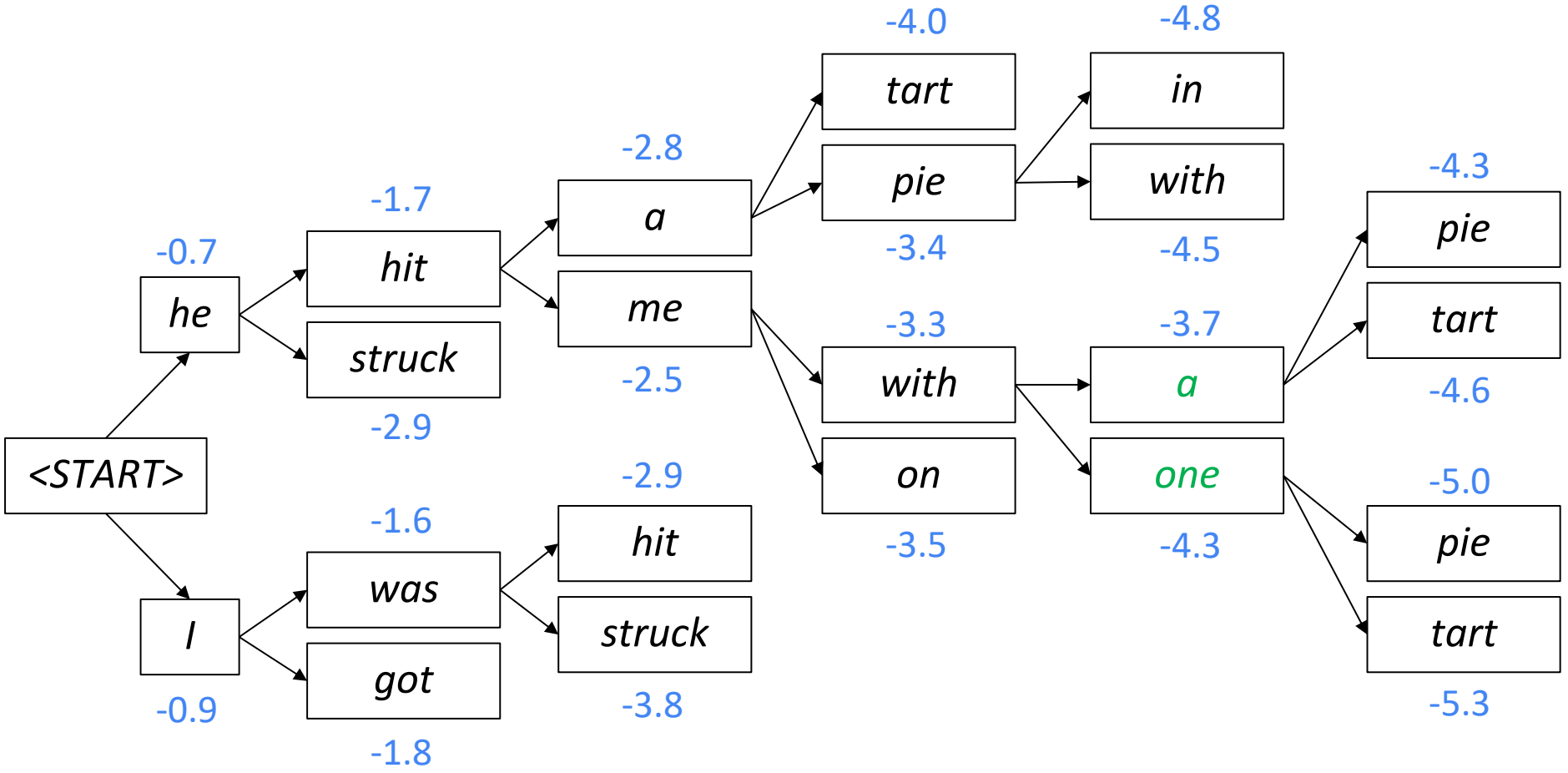
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Of these k^2 hypotheses, just keep k with highest scores

Beam search decoding: example

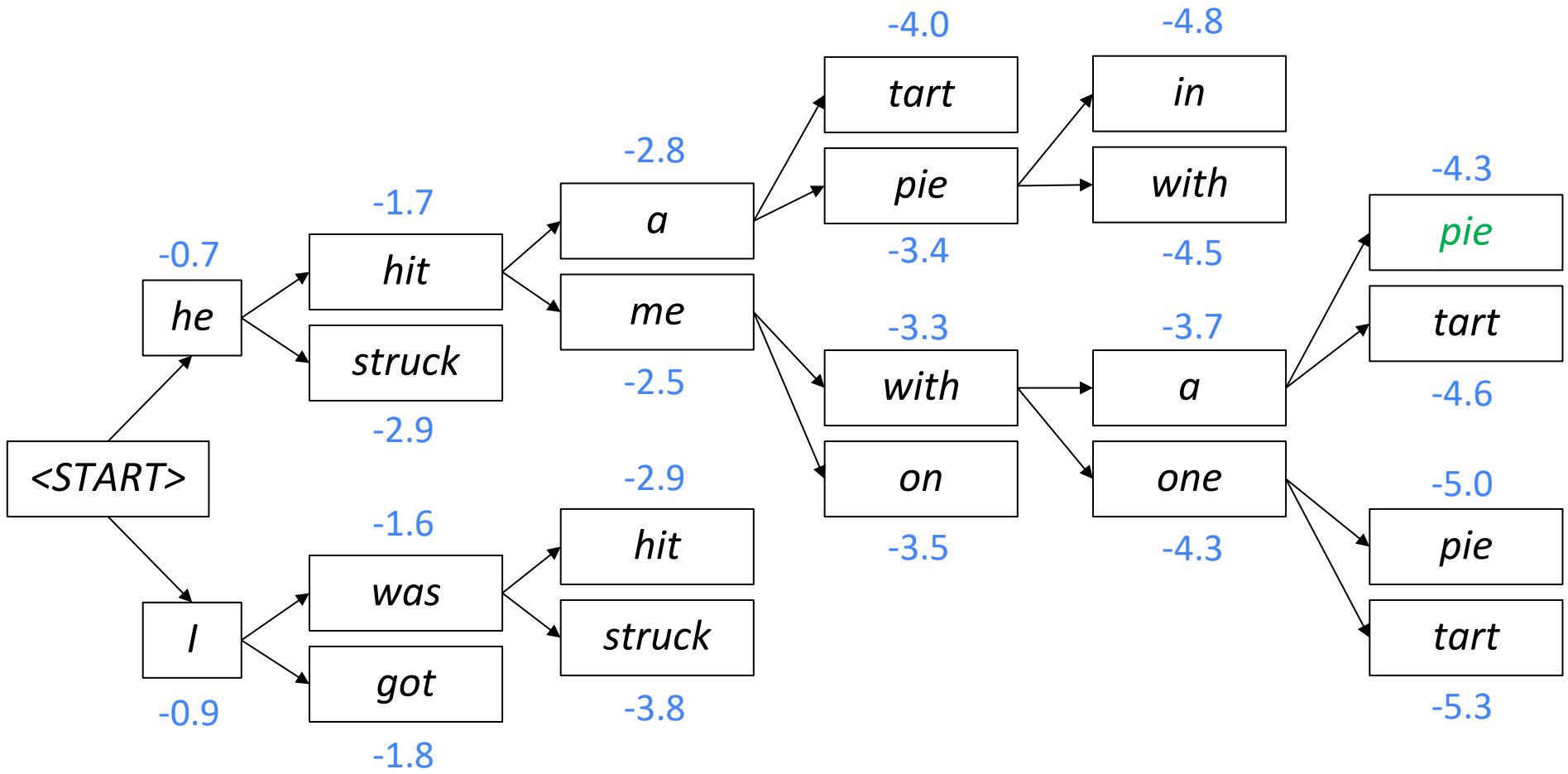
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding: example

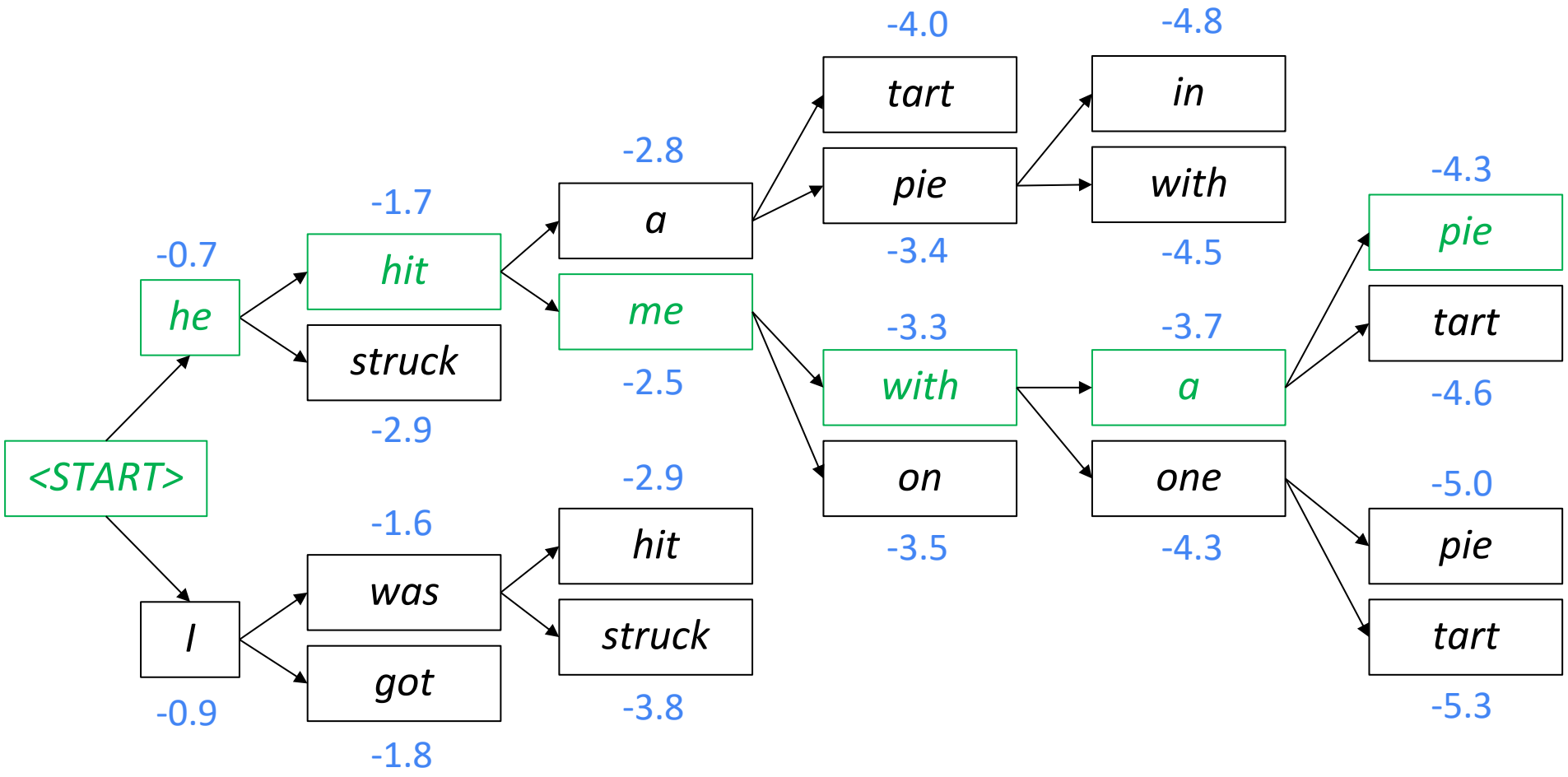
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



This is the top-scoring hypothesis!

Beam search decoding: example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Backtrack to obtain the full hypothesis

Beam search decoding: stopping criterion

- In **greedy decoding**, usually we decode until the model produces an **<END> token**
  - For example: <START> he hit me with a pie <END>

- In **beam search decoding**, different hypotheses may produce <END> tokens on **different timesteps**
  - When a hypothesis produces <END>, that hypothesis is complete.
  - Place it aside and continue exploring other hypotheses via beam search.

- Usually we continue beam search until:
  - We reach timestep *T* (where *T* is some pre-defined cutoff), or
  - We have at least *n* completed hypotheses (where *n* is pre-defined cutoff)

Beam search decoding: finishing up

- We have our list of completed hypotheses.
- How to select top one with highest score?
- Each hypothesis $y_1, \ldots, y_t$ on our list has a score

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

- **Problem with this:** longer hypotheses have lower scores
- **Fix:** Normalize by length. Use this to select top one instead:

$$\frac{1}{t} \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

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!

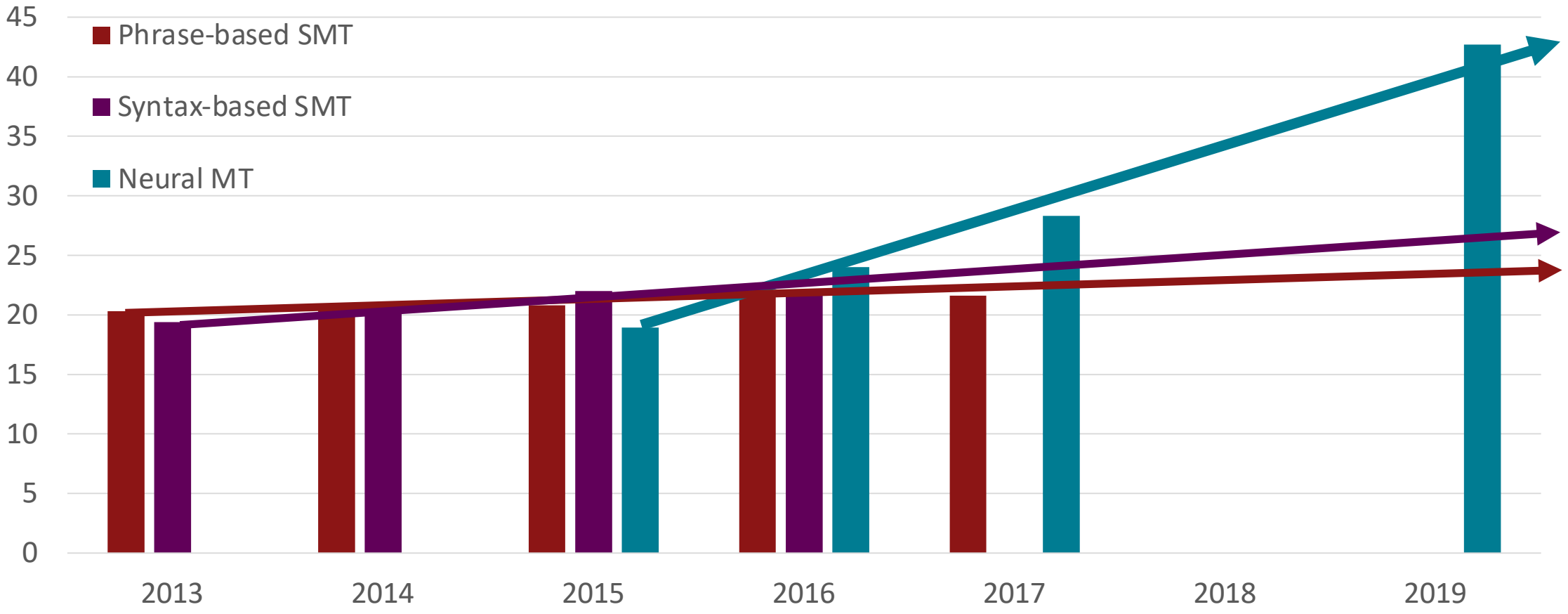
How do we evaluate Machine Translation?

- **BLEU** (**Bil**ingual **E**valuation **U**nderstudy)
  - 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)
    - Plus a penalty for too-short system translations
  - BLEU is **useful** but **imperfect**
    - 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

Source: "BLEU: a Method for Automatic Evaluation of Machine Translation", Papineni et al, 2002. http://aclweb.org/anthology/P02-1040

MT progress over time

[Edinburgh En-De WMT newstest2013 Cased BLEU; NMT 2015 from U. Montréal; NMT 2019 FAIR on newstest2019]



Sources: http://www.meta-net.eu/events/meta-forum-2016/slides/09_sennrich.pdf & http://matrix.statmt.org/

NMT: perhaps the biggest success story of NLP Deep Learning?

- Neural Machine Translation went from a **fringe research attempt** in 2014 to the **leading standard method** in 2016
  - 2014: First seq2seq paper published
  - 2016: Google Translate switches from SMT to NMT – and by 2018 everyone has

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

So, is Machine Translation solved?

- Nope!
- Many difficulties remain:
  - Out-of-vocabulary words
  - Domain mismatch between train and test data
  - Maintaining context over longer text
  - Low-resource language pairs
  - Failures to accurately capture sentence meaning
  - Pronoun (or zero pronoun) resolution errors
  - Morphological agreement errors

Further reading: *"Has AI surpassed humans at translation? Not even close!"*
https://www.skynettoday.com/editorials/state_of_nmt

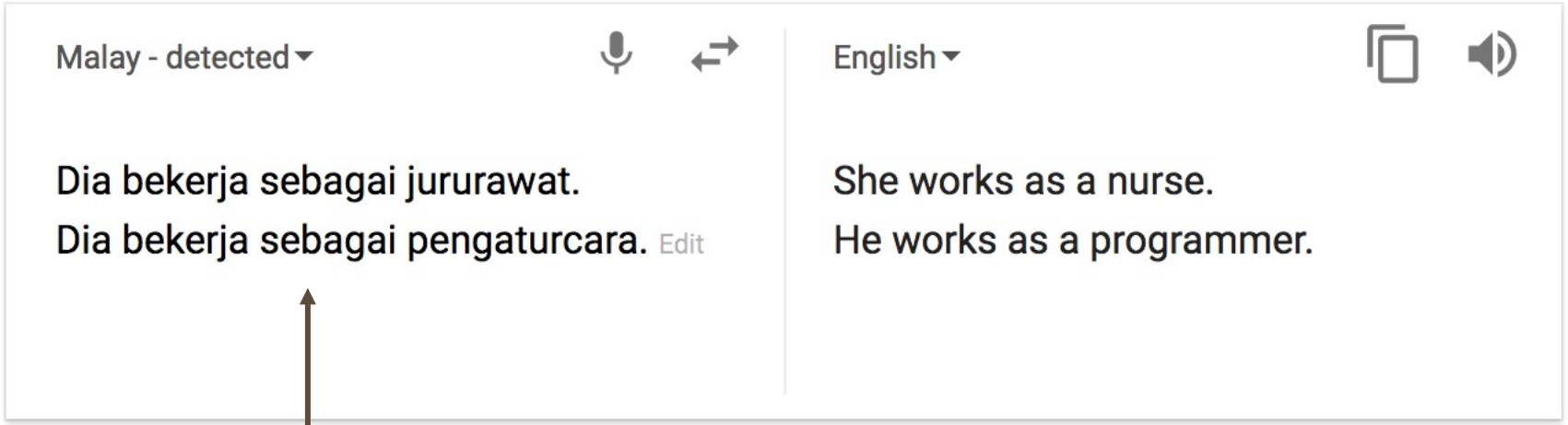
So, is Machine Translation solved?

- Nope!
- Using **common sense** is still hard

?

So, is Machine Translation solved?

- Nope!
- NMT picks up biases in training data



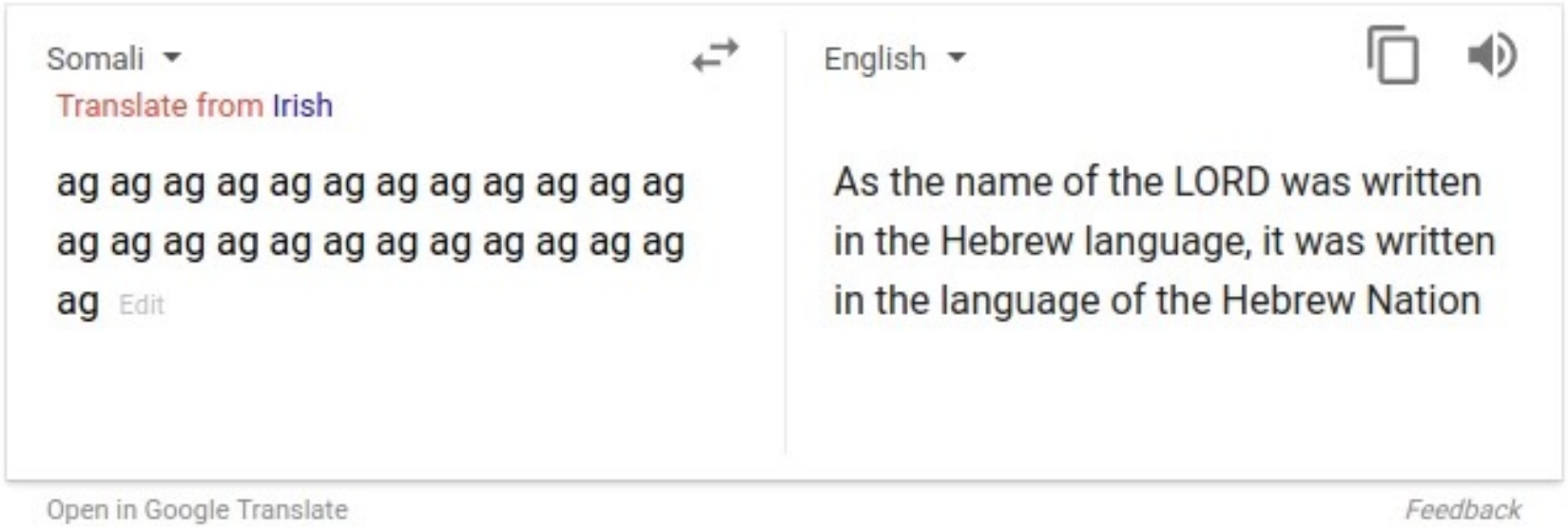
Didn’t specify gender

Source: https://hackernoon.com/bias-sexist-or-this-is-the-way-it-should-be-ce1f7c8c683c

So, is Machine Translation solved?

- Nope!
- **Uninterpretable** systems do **strange things**

(But this problem might have been fixed in Google Translate by 2021.)



Picture source: https://www.vice.com/en_uk/article/j5npeg/why-is-google-translate-spitting-out-sinister-religious-prophecies
Explanation: https://www.skynettoday.com/briefs/google-nmt-prophecies

NMT research continues

- NMT is a **flagship task** for NLP Deep Learning
  - NMT research has **pioneered** many of the recent **innovations** of NLP Deep Learning
  - NMT research continues to **thrive**
    - Researchers have found **many, many improvements** to the "vanilla" seq2seq NMT system we've just presented
    - But we'll present next **one improvement** so integral that it is the new vanilla...

ATTENTION