IT UNIVERSITY OF COPENHAGEN

Encoder-Decoder Models

Barbara Plank
ITU, Copenhagen, Denmark

AthNLP

September 21, 2019,
#AthNLP2019 Athens, Greece

Motivation

(most?!) interesting data is

sequential in nature

[It, is, all, about, sequences]

[s, e, q, u, e, n, c, e, s]

Inspired by slides from S.Gouws & D. Hovy.

Challenge: Language is ambiguous

Galactic bubbles offer clues to **dark matter**
Space Daily - 22 hours ago

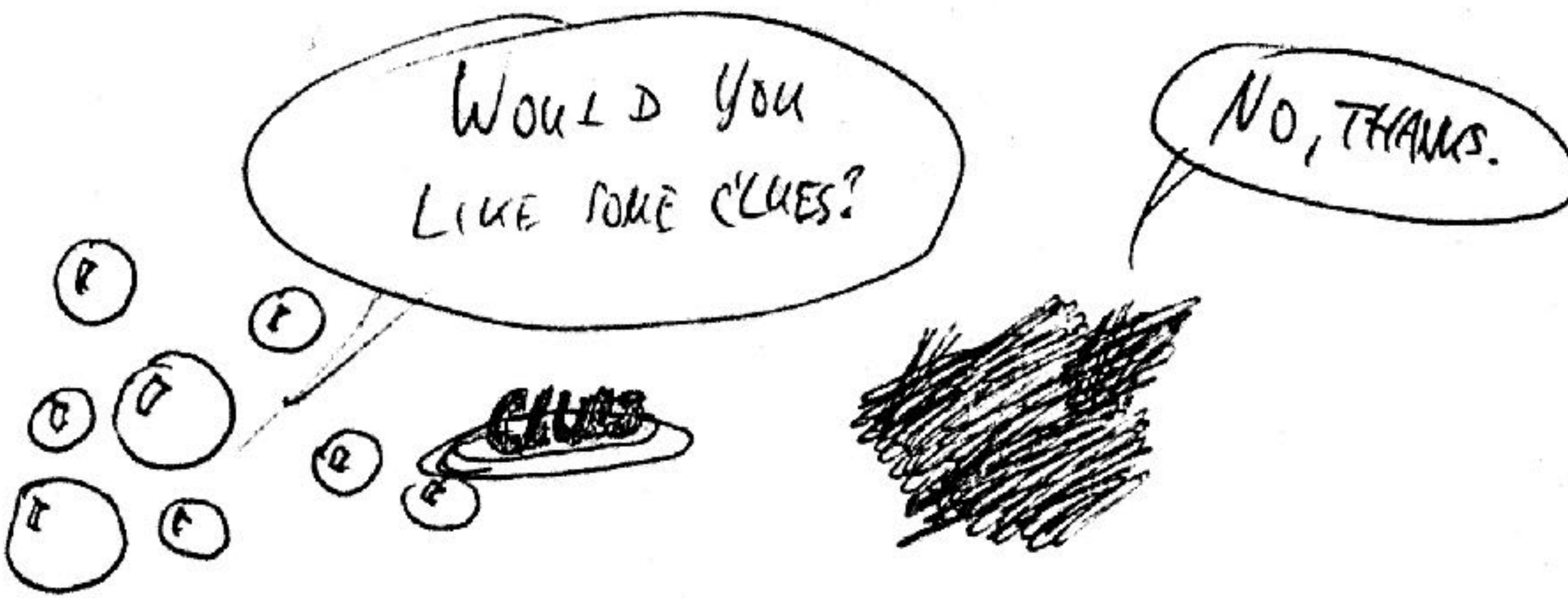


Illustration drawn by Dirk Hovy.

Challenge: Language is productive

How to sunny-day Saturday in Seattle:
✅ pop out of bed and fling open the drapes
✅ brew coffee ☕ and grab your **go-cup**
✅ get outside asap
✅ dog walk, hike, run, bike, kayak, sail
✅ 🍺 soak up the sun in your favorite beer garden
✅ 😃👍✅

2 22

https://twitter.com/seattlekim/status/1106687539279101954

Sequence Modeling Tasks

- In some applications, we want to **condition** on sequential data to make a prediction

condition problems: how to represent a sequence?

AUTHOR ATTRIBUTES

Female

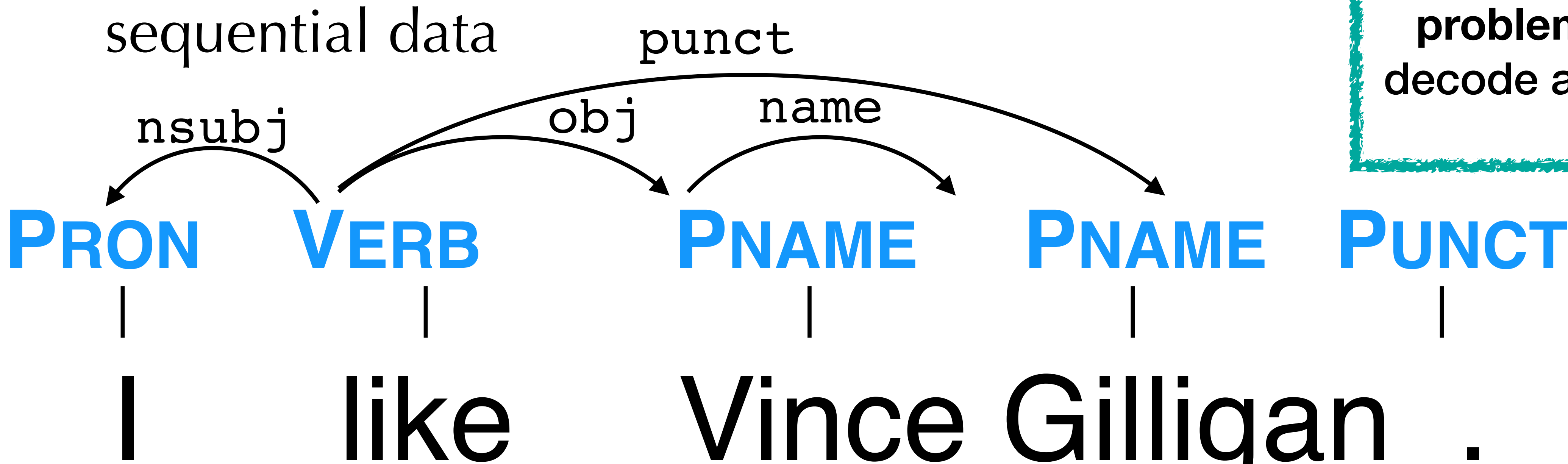
30+

SENTIMENT ANALYSIS

positive

- In other applications, we want to **generate** sequential data

generation problems: how to decode a sequence?



Inspired by slides from Chris Dyer & Dirk Hovy.

A step back…
How did the field evolve?

NLP ❤️ Machine Learning

- Early approaches in NLP: **symbolic & rule-based**
- In the late 80s: development of annotated corpora (especially the well-known Penn Treebank Wall Street Journal)

- And corresponding emergence of **statistical approaches**
- Common evaluation corpora and measures pushed the field

80s

First big jump: statistical learning

Symbolic Processing

e.g. **Brill's**
rule-based tagger (1992)

Epoch 1

from hand-crafted rules to ML

Statistical NLP

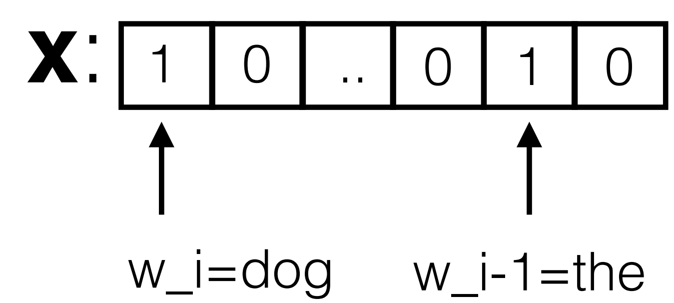
e.g. **Collin's**
structured perceptron (2002)

Epoch 2

approx. 1980s

x: the dog barks

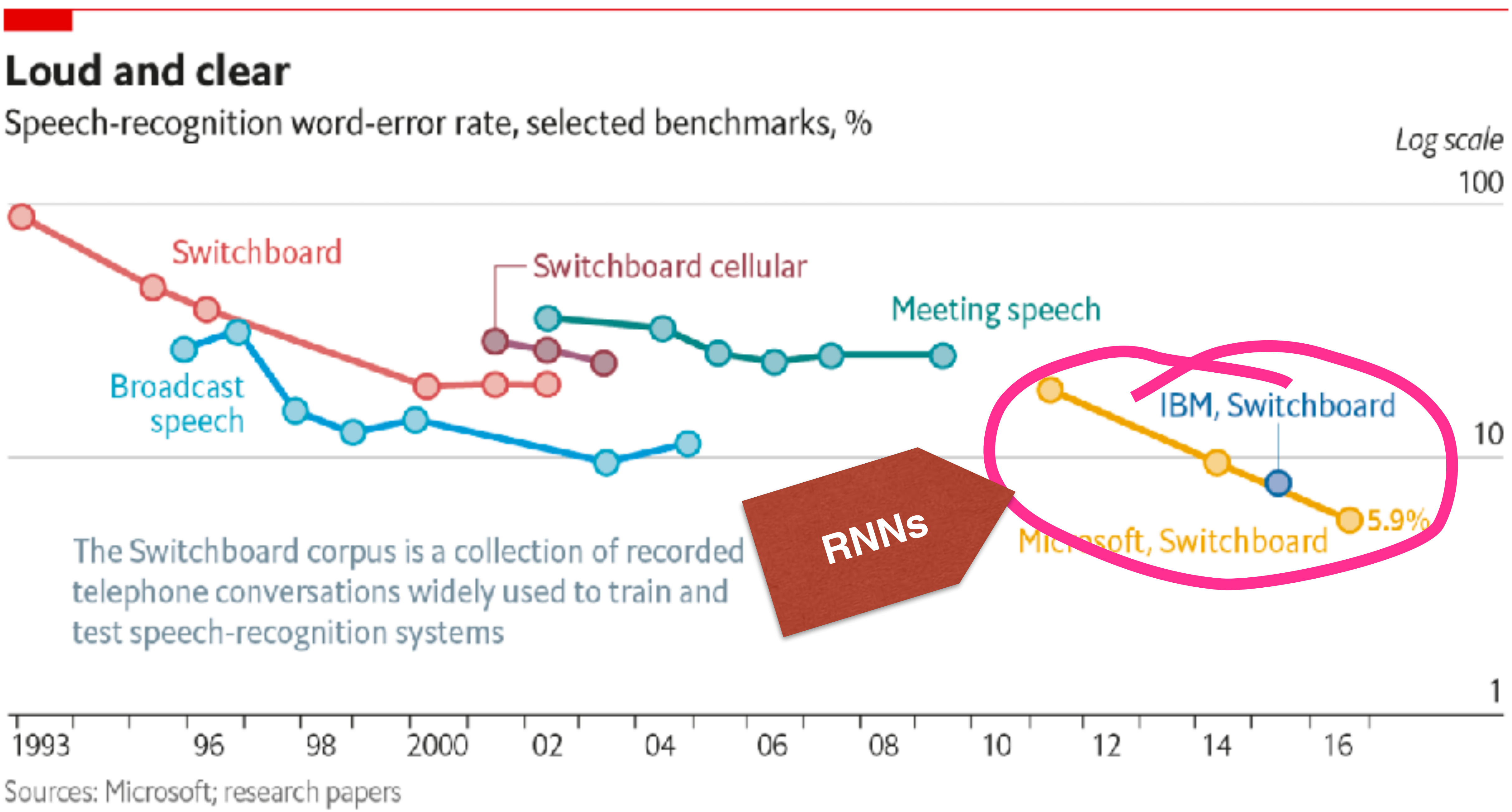
```
if prev_w = DET and ..
tag=NOUN
```



classic **sparse** 'n-hot' encoding

The emergence of deep learning (in NLP)

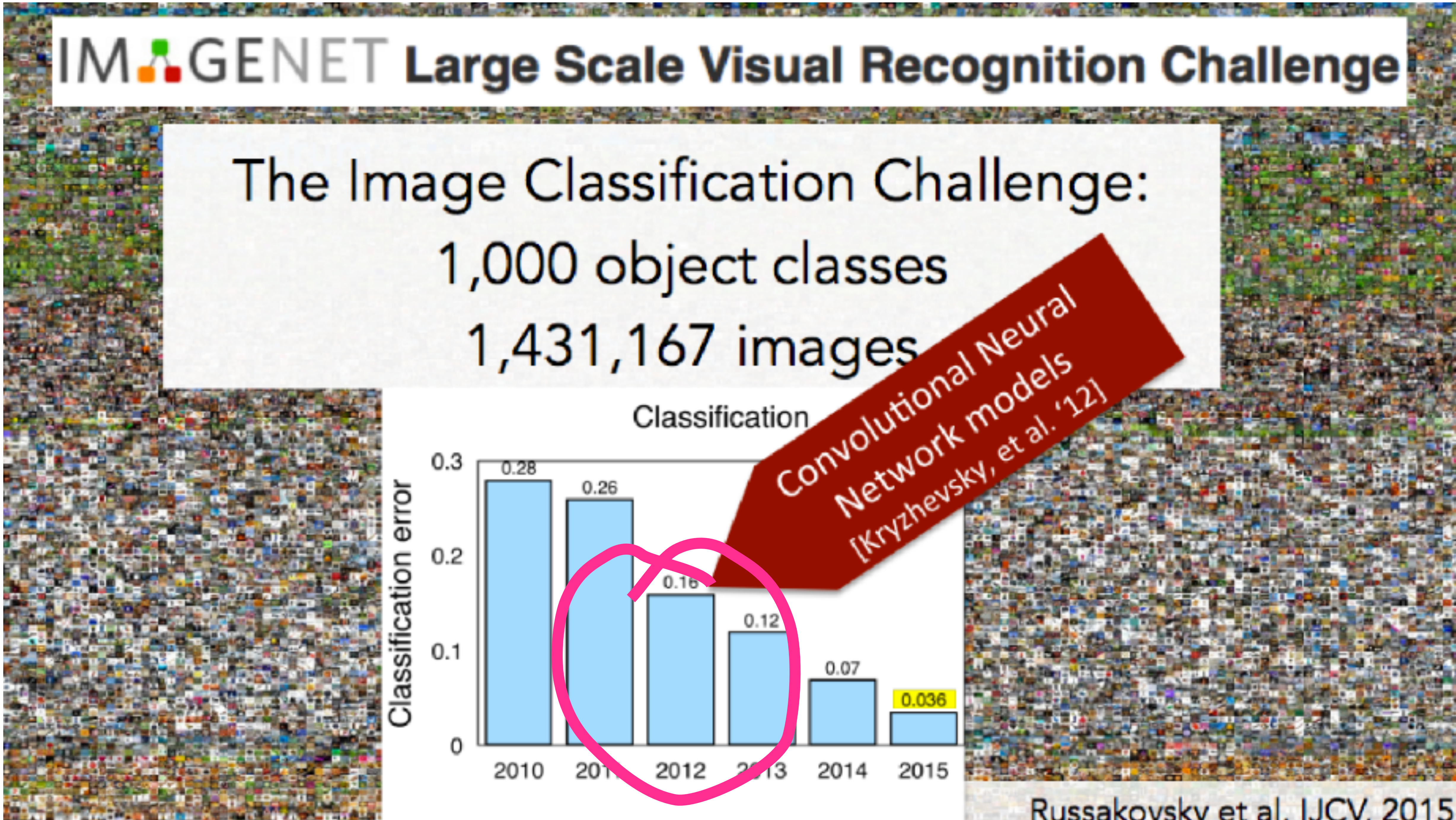
In Speech Recognition



(Source: The Economist)

2010

In Computer Vision

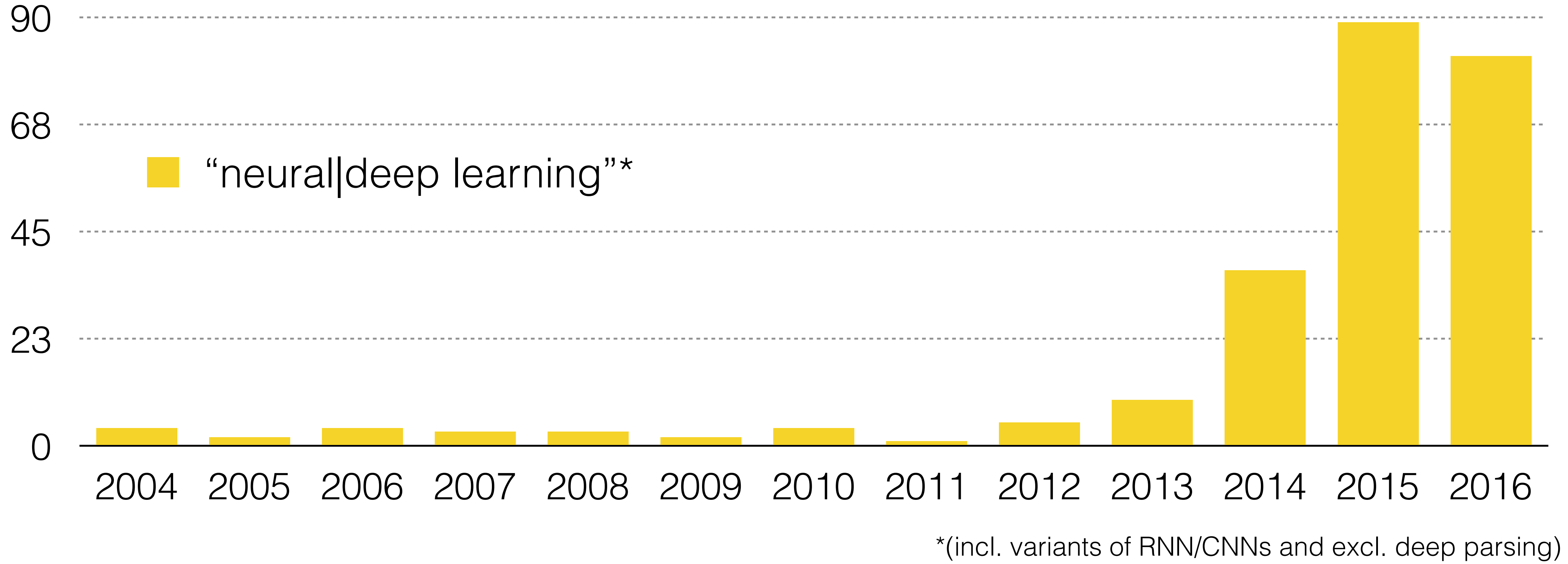


(src: slide by Fei-Fei Li)

2012

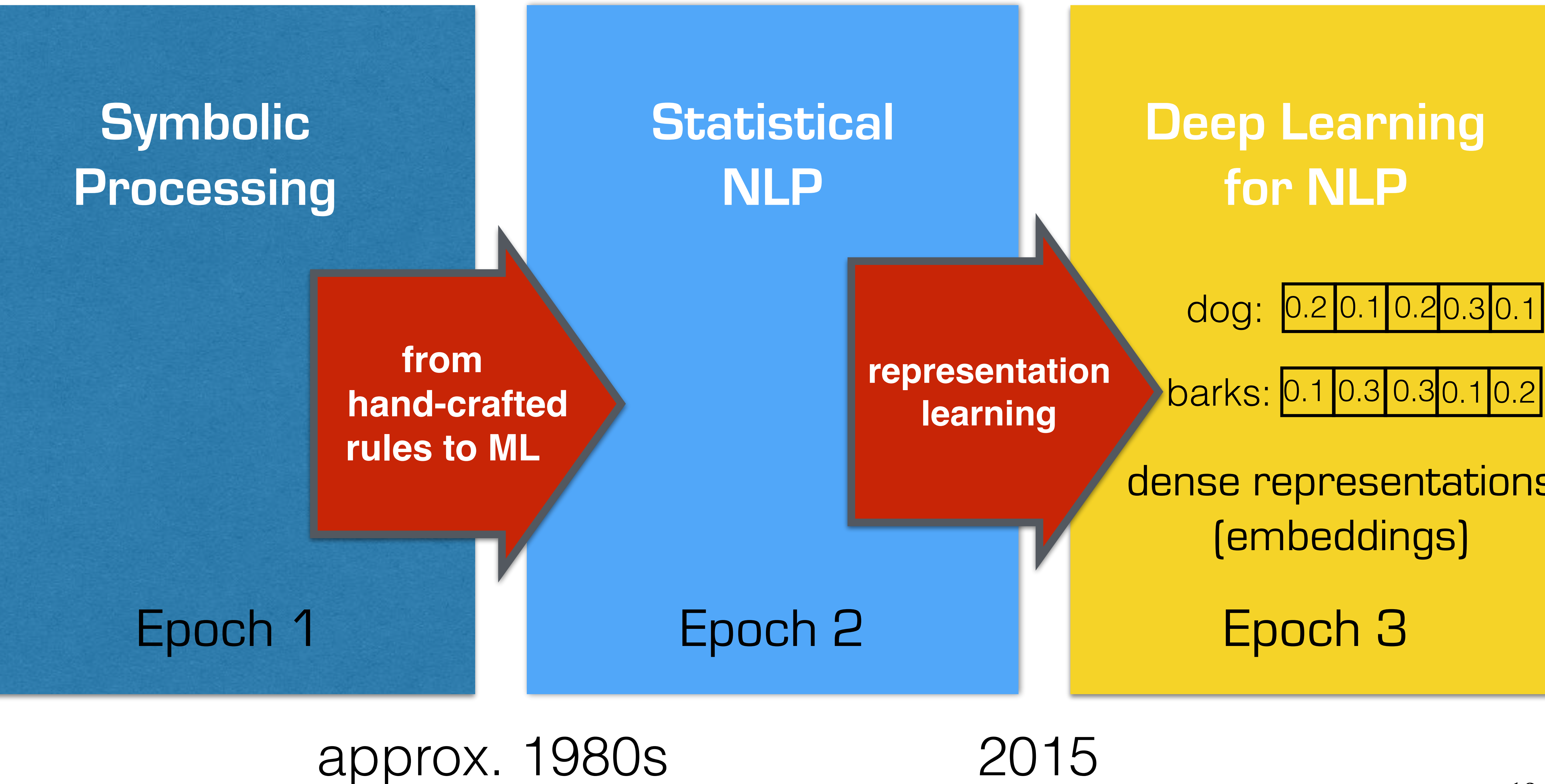
Papers: Deep learning in NLP

"2015 seems like the year when the full force of the tsunami hit the major NLP conferences"
—Chris Manning (2015)



Titles of papers in ACL anthology (from 2004)

NLP ❤ Deep Learning



Overview

foundations

Motivation, Brief History, Overview

Back to the roots: Language Models

Feedforward NNs (FFNNs)

representations

What's the input? Representations

Contextualized Embeddings, Fine-tuning

beyond FFNNs

Convolutional Neural Networks (CNNs)

Recurrent Neural Network (RNNs)

Advanced RNNs, Decoders

Attention? Attention!

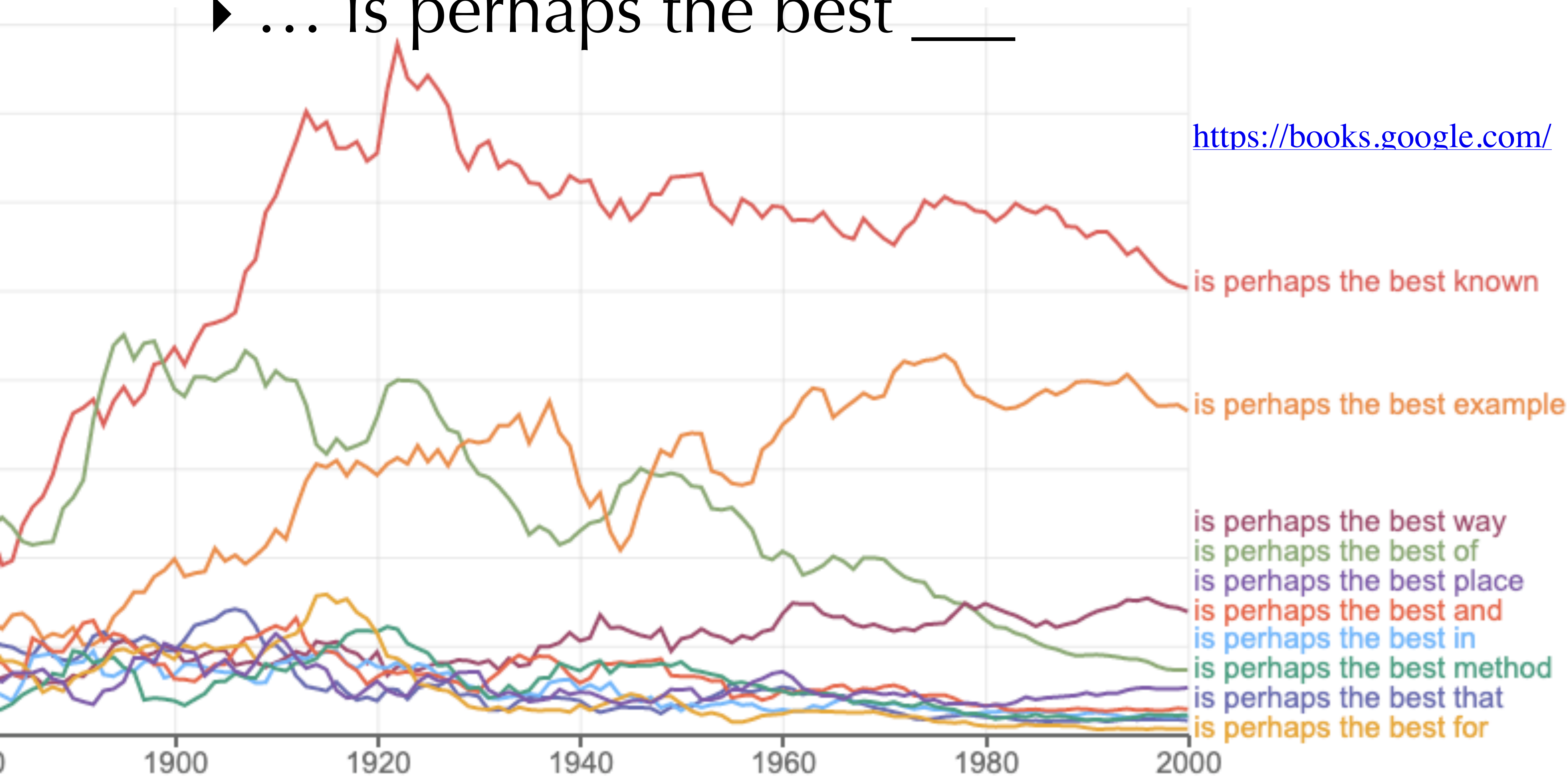
Predicting the next word: A Simple (?) Exercise

- www.mentimeter.com
  Room: (see code)

* (title inspired by Graham Neubig)

More examples

- Recurrent Neural ___
- … is perhaps the best ___

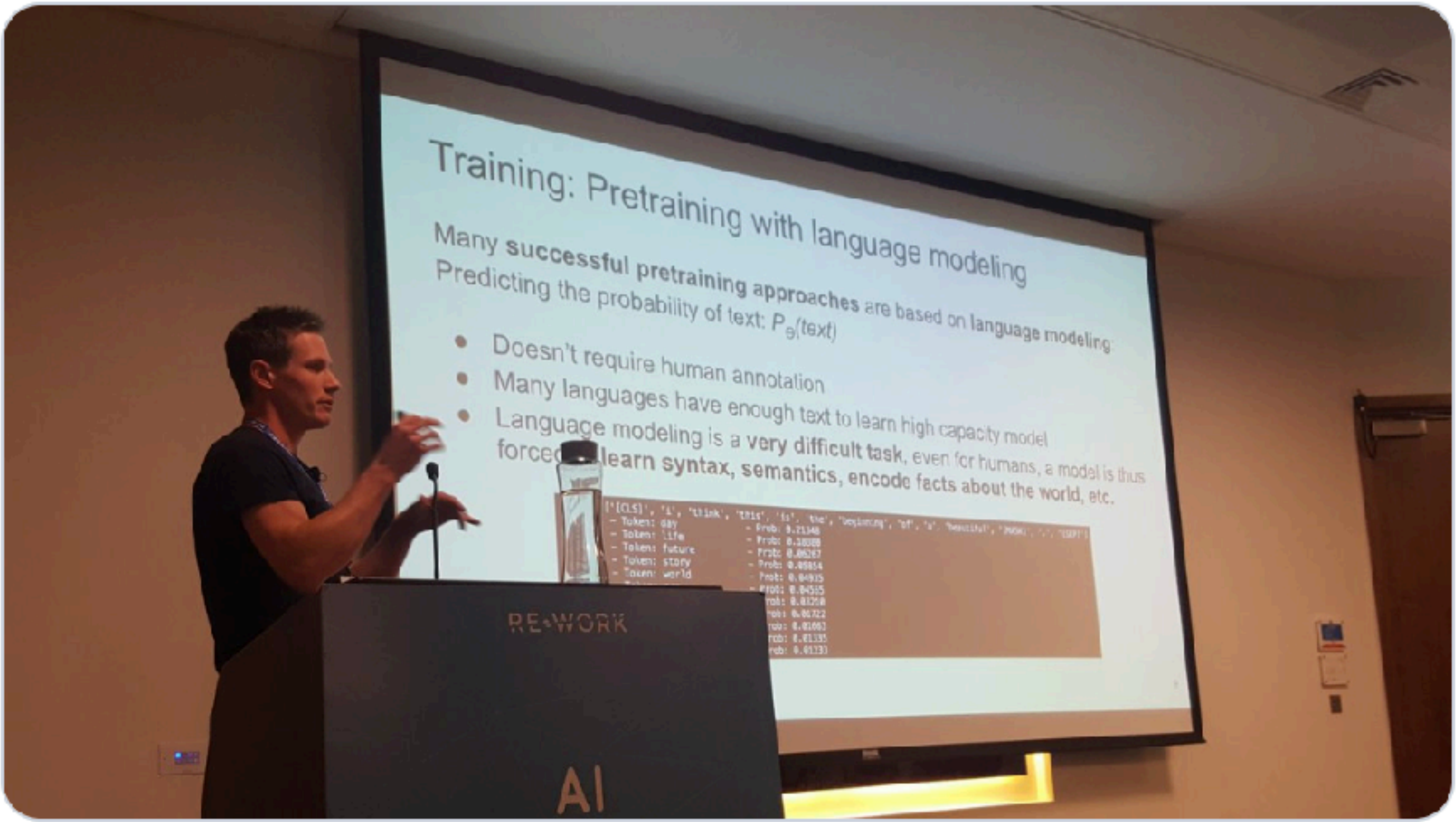


https://books.google.com/

Why care about LMs?

Verena Rieser @verena_rieser · 3m

Great talk by @Thom_Wolf on transfer learning @reworkAI !



https://twitter.com/verena_rieser/status/1174694748310953984

So let’s look deeper at LMs: from traditional LMs to contextualized embeddings

What is a Language Model (LM)?

- A computational model that can be used to either of the following two tasks is called a Language Model (LM):

  - to compute the probability of a text*

$$P(\textbf{today is a great day}) = ??$$

  - to compute the probability of the next word

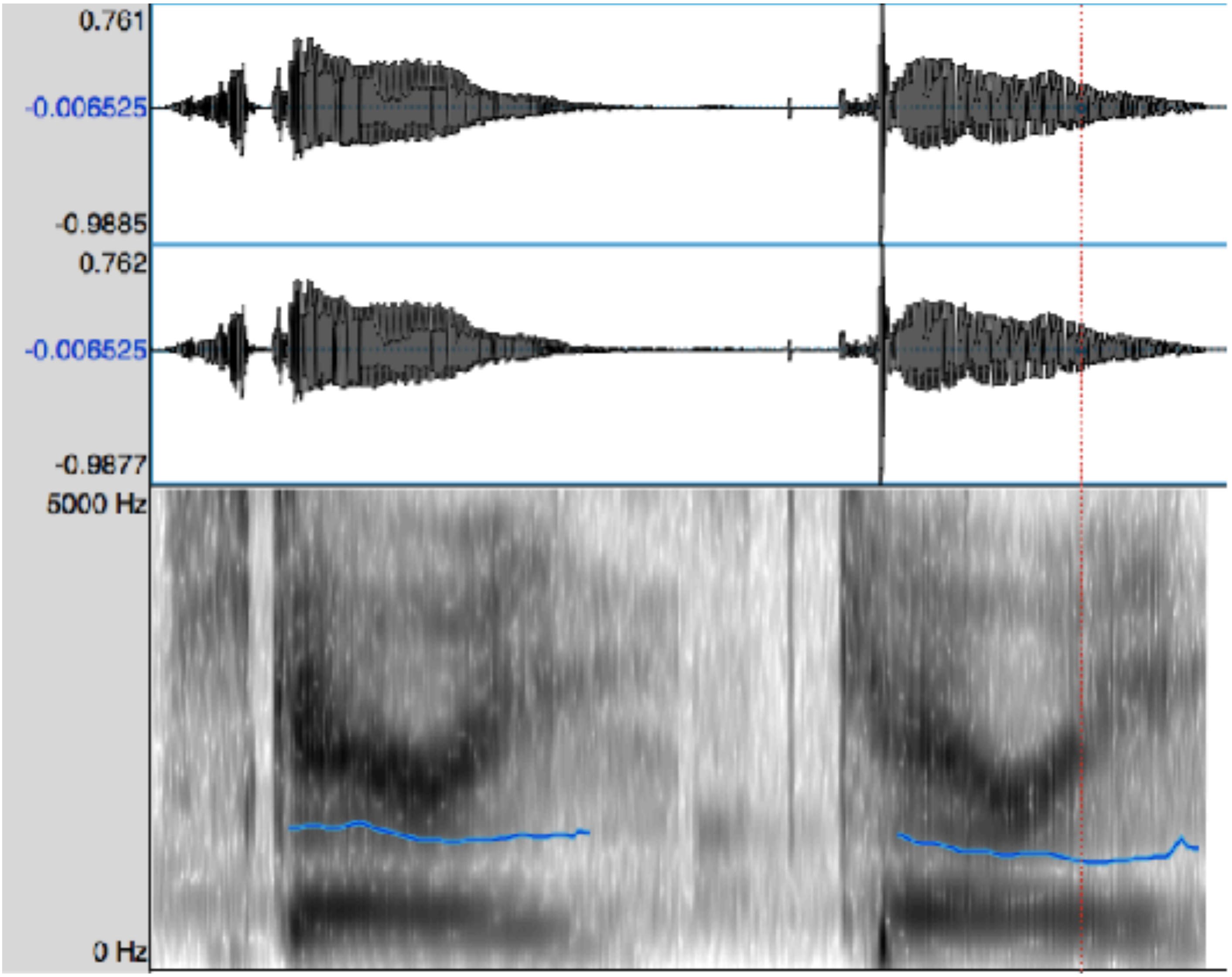
$$P(\textbf{day} \mid \textbf{today is a great}) = ??$$

* (can be a text, sentence, phrase,…)

Why?
Example Use Cases

Speech Recognition

P(**where is the nearest beach**) $>$ P(**where is the nearest breach**)



http://www.personal.psu.edu/ejp10/blogs/tlt/2011/09/understanding-speech-recogniti.html

Spelling Correction

You probably use a LM every day…

what is the mo

what is the mo - Google Search

what is the mo**st spoken language in the world**

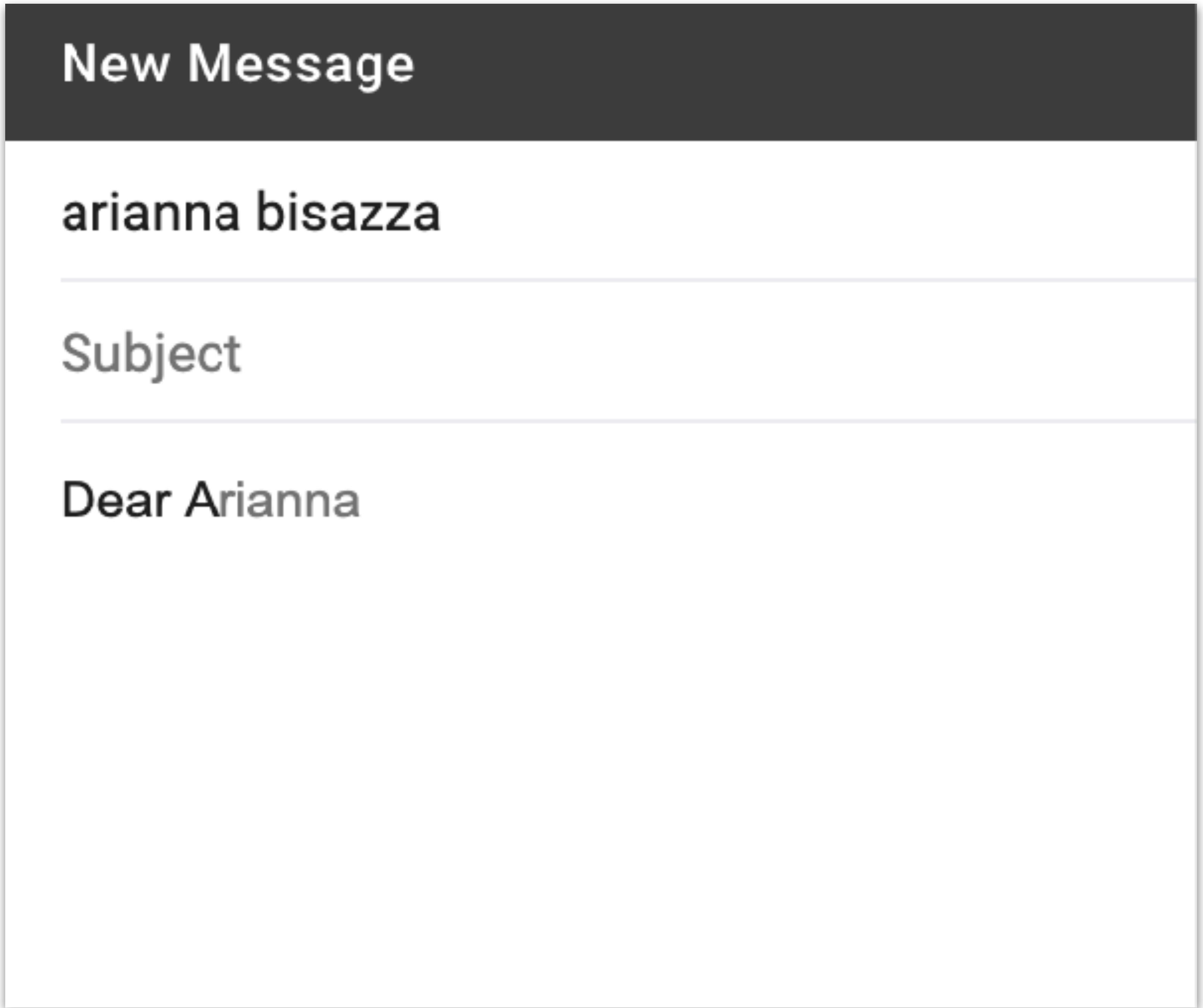
what is the mo**st played game in the world**

what is the mo**st dangerous animal in the world**

what is the mo**st expensive car in the world**

what is the mo**on made of**

You probably use a LM every day…



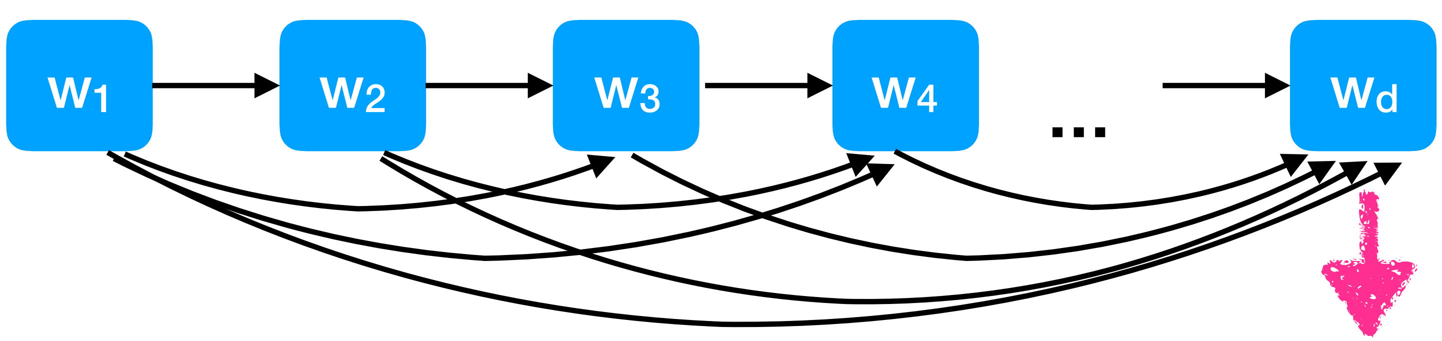
A Language Model - Formally

- Given a sequence of words: $(w_1, \ldots, w_d)$
- LM models the probability: $P(w_1, \ldots, w_d)$
- Without loss of generality (**Chain Rule**):

$$P(w_1, \ldots, w_d) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2)\ldots$$

history

$$= P(w_1)\prod_{i=2}^{d} P(w_i \mid \boxed{w_1, \ldots, w_{i-1}})$$



$P($**Athens|an awesome summer school this year is in**$)$

Markov assumption

- A common assumption in sequence modeling is to make the **Markov assumption**:

$$P(x_1, \ldots, x_d) = \prod_{i=1}^{d} P(x_i \mid x_1, \ldots, x_{i-1}) \approx \prod_{i=1}^{d} P(x_i \mid x_{i-(n-1)}, \ldots, x_{i-1})$$

$$\begin{aligned} p(\boldsymbol{w}) = & p(w_1) \times \\ & p(w_2 \mid w_1) \times \\ & p(w_3 \mid \cancel{w_1}, w_2) \times \\ & p(w_4 \mid \cancel{w_1, w_2}, w_3) \times \\ & \cdots \end{aligned}$$

Markov: forget "distant" past

Valid for language? No**...**

Is it practical? Often**!**

n-th order Markov assumption: history of n-1 words

Adapted from Chris Dyer

How to learn a LM?

- **(Pre-deep learning) era:** Learn an **n-gram** Language Model
- **n-gram**: a chunk of consecutive words
  - n=2 (bigram): “to buy”, “buy a”, “a house”…
  - n=3 (trigram): “to buy a”, “buy a house”,…
- Key method: **collect statistics** of n-grams from a corpus to estimate the parameters of the model (maximum likelihood)

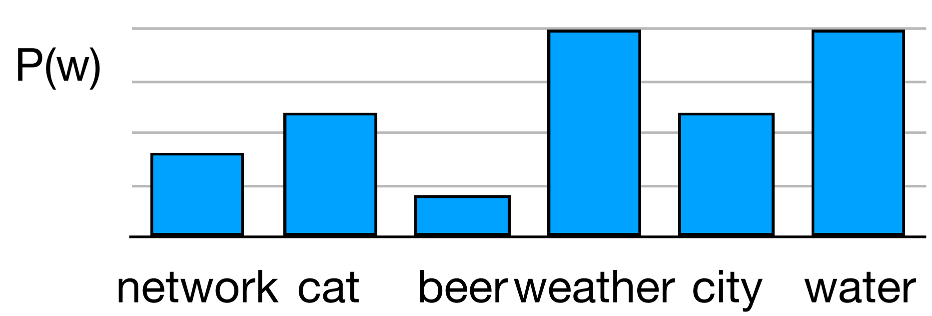
Unigram LM (1st order)

- n=1, 1st order Markov assumption, history (n-1): 0

$$P(w_1, \ldots, w_d) = P(w_1)P(w_2)P(w_3)\ldots$$

$$= \prod_{i=1}^{d} P(w_i)$$

$P(\textbf{started})$ unigram LM



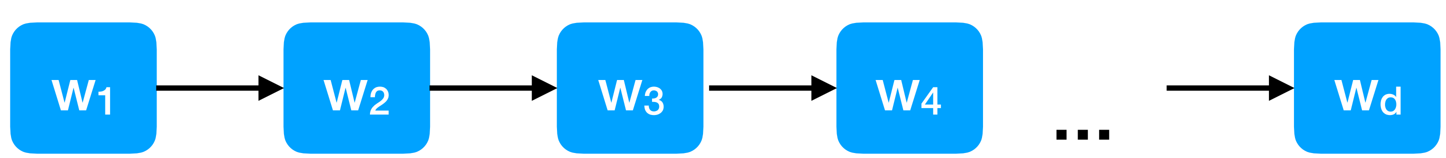
Bigram Language Model

- n=2, 2nd order Markov assumption, history (n-1): 1

$$P(w_1, \ldots, w_d) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_2)\ldots$$

$$= P(w_1)\prod_{i=2}^{d} P(w_i \mid w_{i-1})$$

$P(\textbf{started} \mid \textbf{has})$ bigram LM



Higher-order LMs

- A **bigram** model conditions on the previous word (n=2; or: a window of 2 words)

$$P(x_i \mid x_{i-1})$$

- A **trigram** model uses a history of 2 words (n=3, history 2)

$$P(x_i \mid x_{i-2}, x_{i-1})$$

- E.g. a 5-gram LM

Sparsity problems with n-gram LMs

C=count()

<u>Sparsity problem 1:</u> If "its water is so w" never occurred in the corpus: prob for w is 0!

What can we do? **Smoothing** (add small count to every w)

$$P(\mathbf{w} \,|\, \textbf{its water is so}) = \frac{C(\textbf{its water is so } \mathbf{w})}{C(\textbf{its water is so})}$$

<u>Sparsity problem 2:</u> If "its water is so" never occurred: prob for *any* w is is 0.

What can we do? **Backoff** (condition on lower-level n-grams)

<u>In general:</u> Increasing n-gram size makes sparsity problem worse.

See for more details chapter 3 of Jurafsky & Martin.

<u>Adapted from Abigail See</u>

Further issues with n-gram LMs

- What about similar words?
  - she *bought* a bicycle
  - she *purchased* a bicycle

cannot share strength among similar words

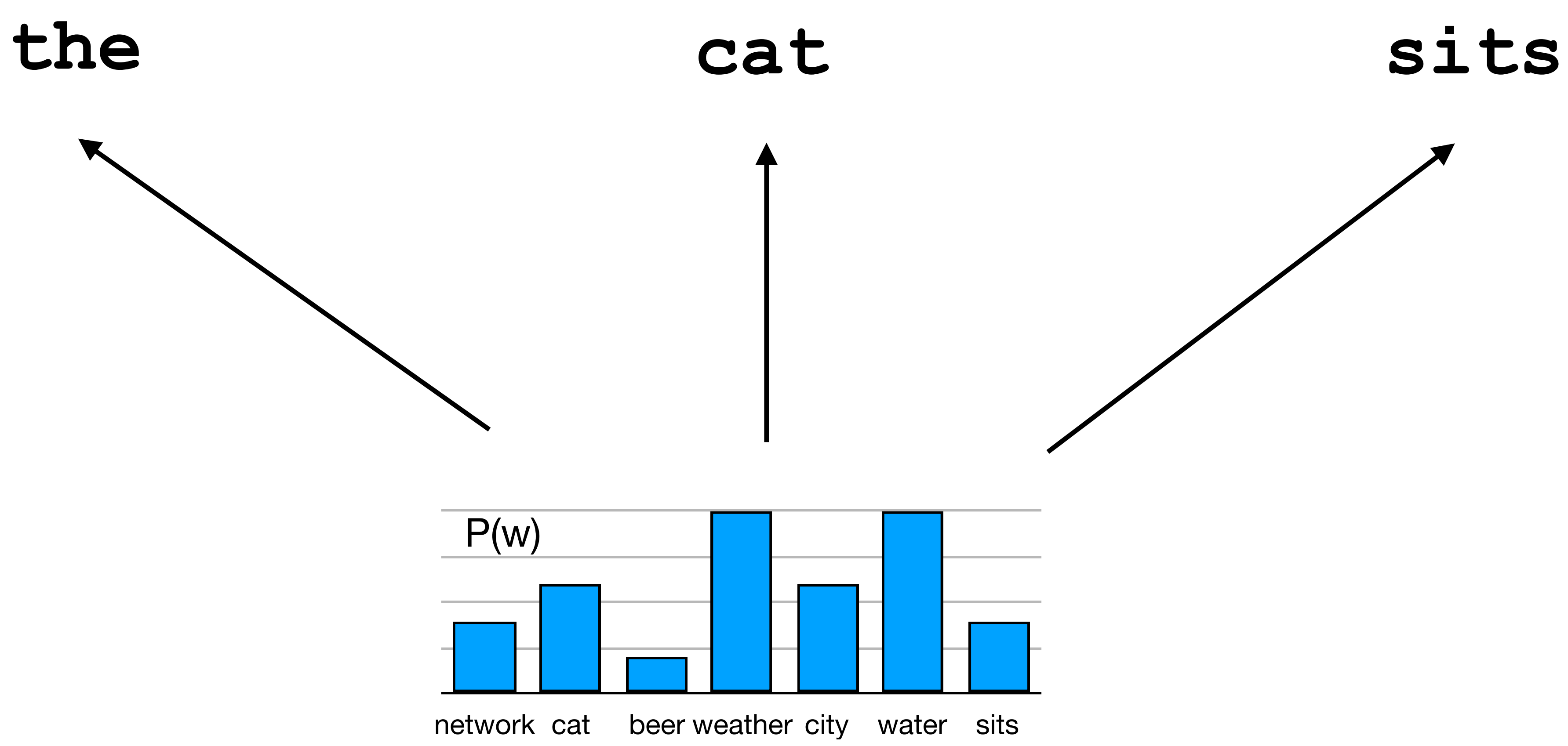
- Long-distance dependencies?
  - for *programming* she yesterday purchased her own brand new *laptop*
  - for *running* she yesterday purchased her brand new *sportswatch*

cannot handle long-distance dependencies

Generating text from a LM

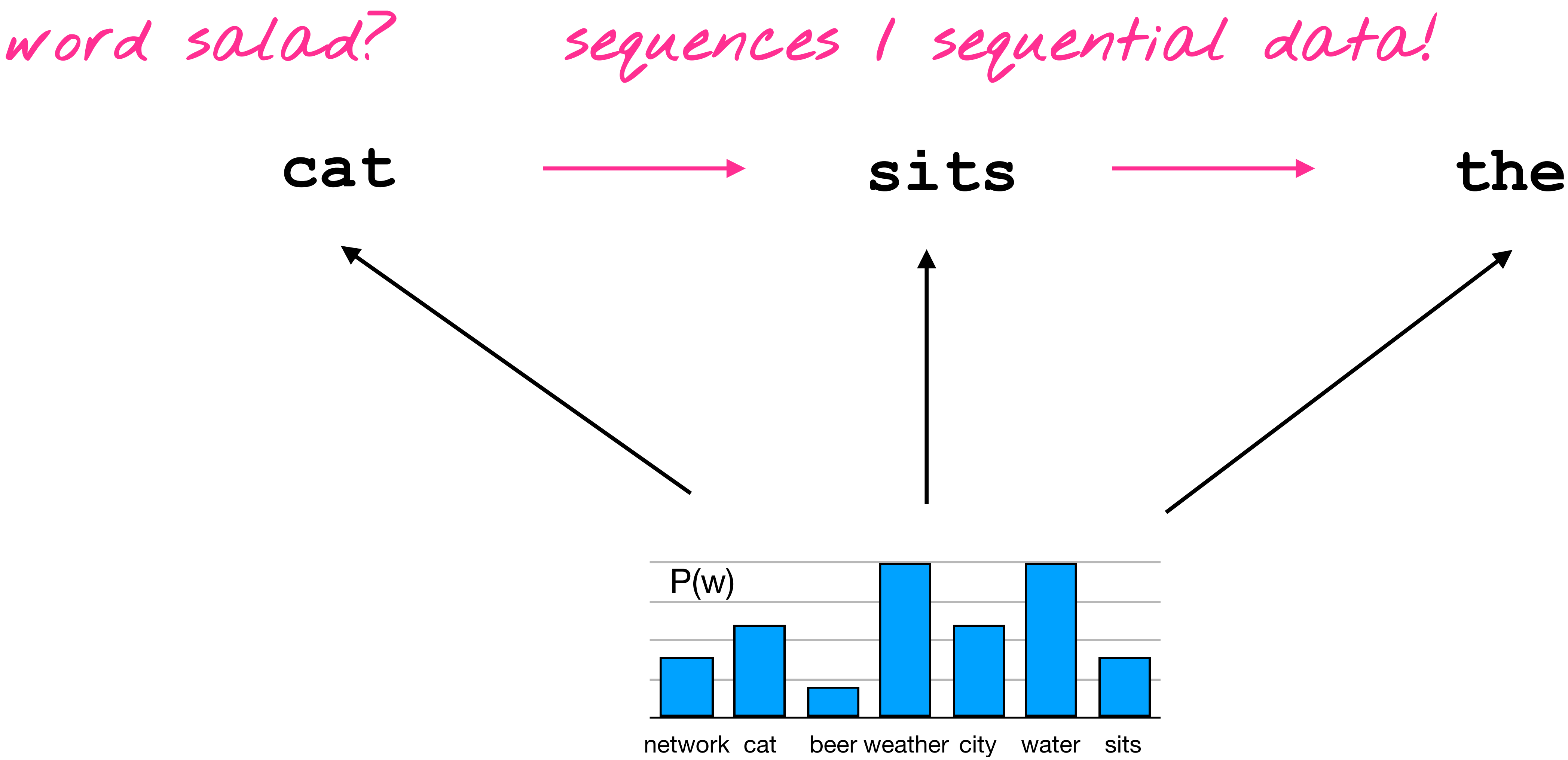
Sample from a unigram LM

- We can sample **incrementally** from a Language Model, one word at a time



Sample from a unigram LM

- We can sample **incrementally** from a Language Model, one word at a time



Outlook: Why RNNs are so great for Language

- **No more Markov assumptions**
- Great fit for *sequences / sequential data*
- Arbitrary length input

```
the ──→ cat ──→ sits
```

```
the cat sits there

the sleepy cat sits there

the sleepy cat which chased the dog sits
```

How to learn a *neural* LM?

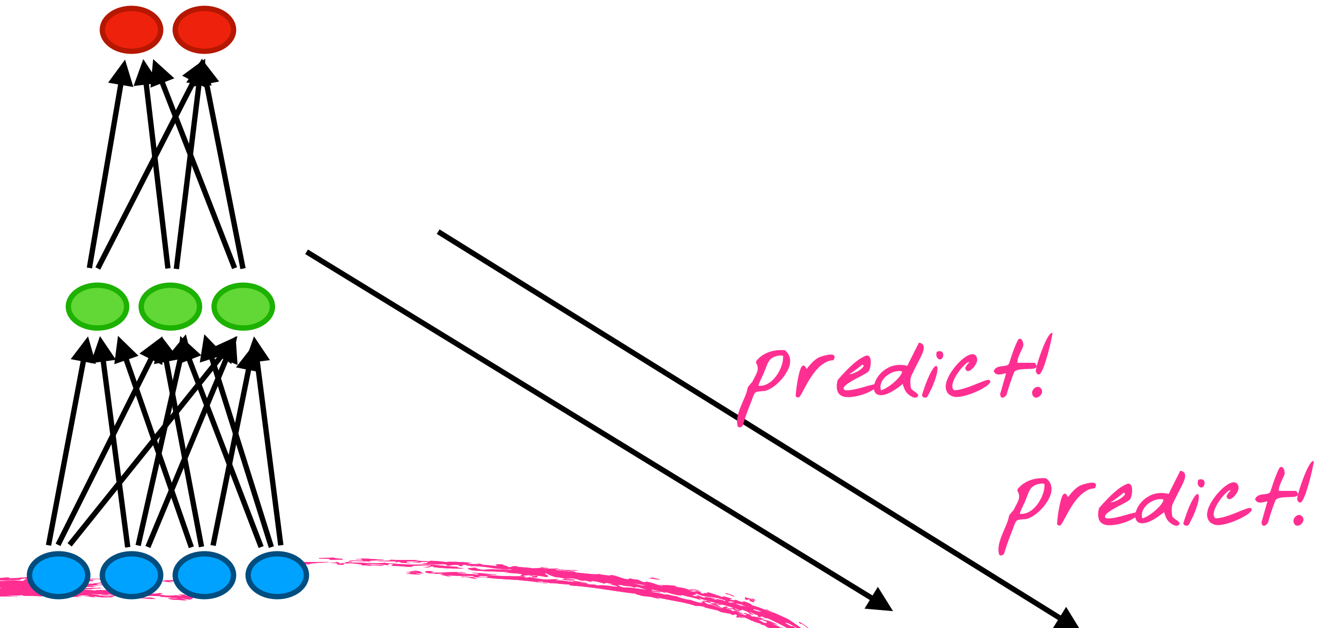
- **Language model task:**
  - input: sequence of words: $(w_1, \ldots, w_d)$
  - output: probability of next word $P(w_{t+1} \mid w_t, \ldots, w_2, w_1)$

- An Early (deep learning era) solution: **a window-based n-gram neural language model** (Bengio et al., 2003)

A neural probabilistic language model
Y Bengio, R Ducharme, P Vincent, C Jauvin - Journal of machine learning ..., **2003** - jmlr.org
A goal of statistical language modeling is to learn the joint probability function of sequences of words in a language. This is intrinsically difficult because of the curse of dimensionality: a word sequence on which the model will be tested is likely to be different from all the word sequences seen during training. Traditional but very successful approaches based on n-grams obtain generalization by concatenating very short overlapping sequences seen in the training set. We propose to fight the curse of dimensionality by learning a distributed ...
Cited by 5046 Related articles All 57 versions

feed-forward neural network

Window-based neural LM via FFNN



~~As the clock~~ rang the students opened the

discard

fixed window of n words

Overview

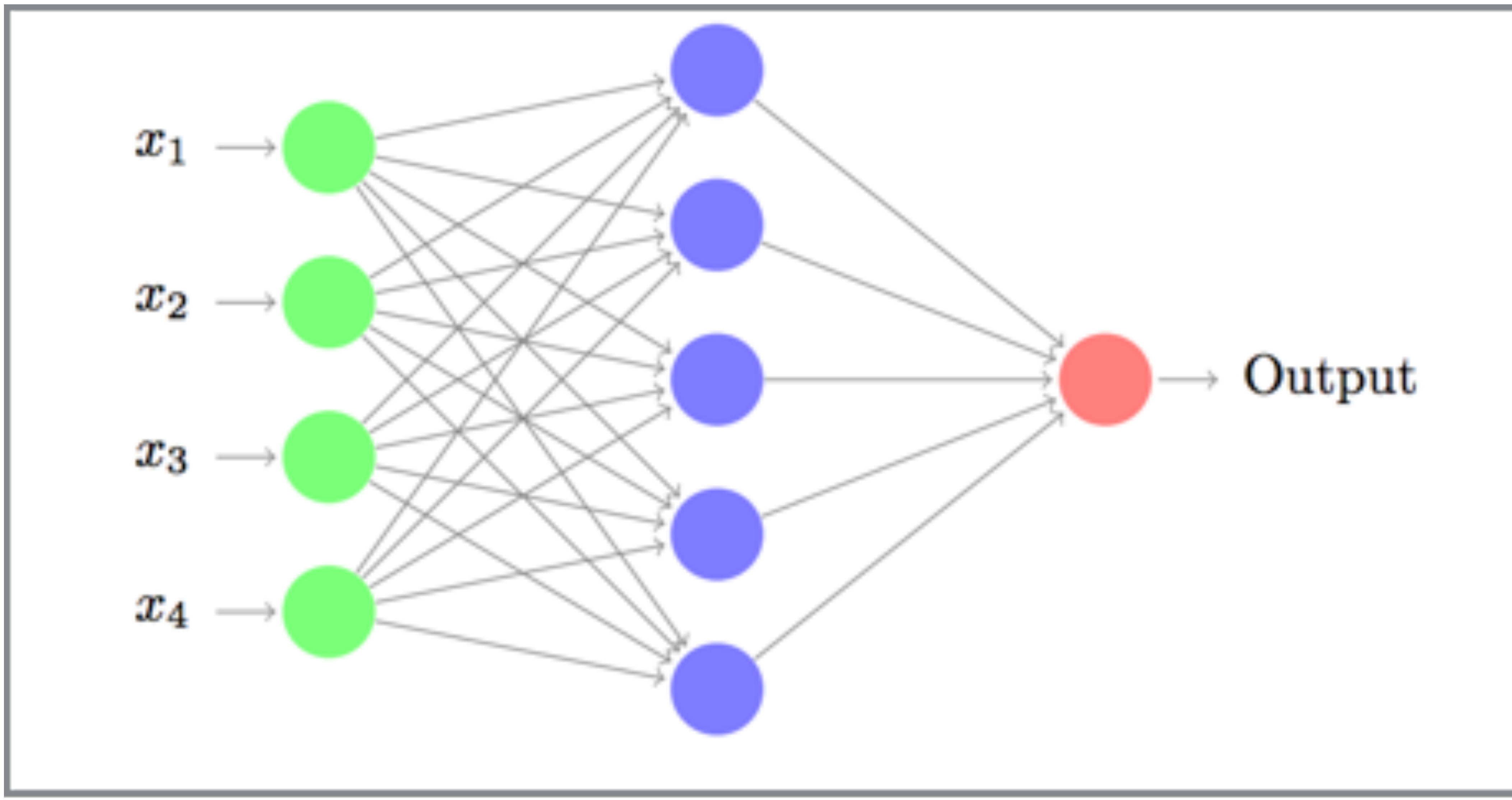
foundations

Motivation,
Brief History, Overview → Back to the roots:
Language Models → Feedforward NNs
(FFNNs)

Feedforward Neural Network (FFNN)

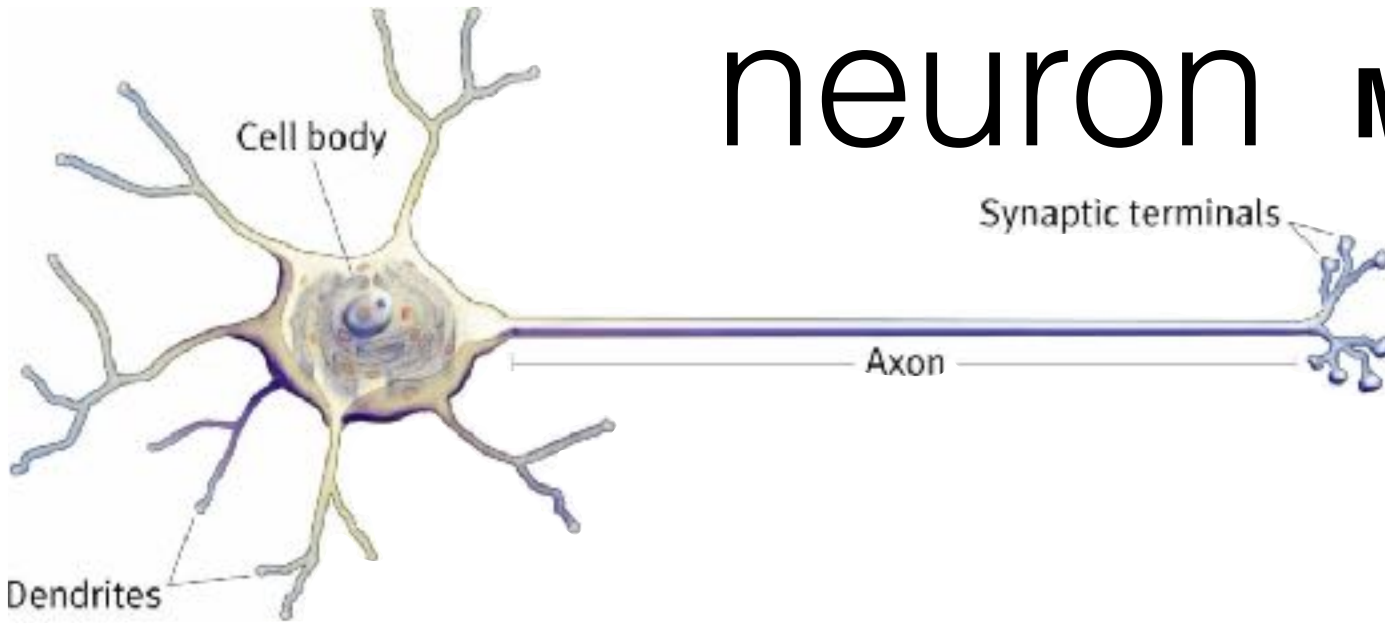
- After this **recap** you should:
  - connect the different views on FFNNs
  - refresh ourselves on how to represent input in NLP

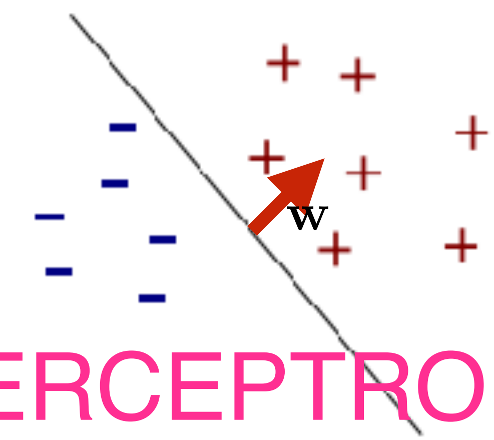
Neural Network



From biological to artificial neuron

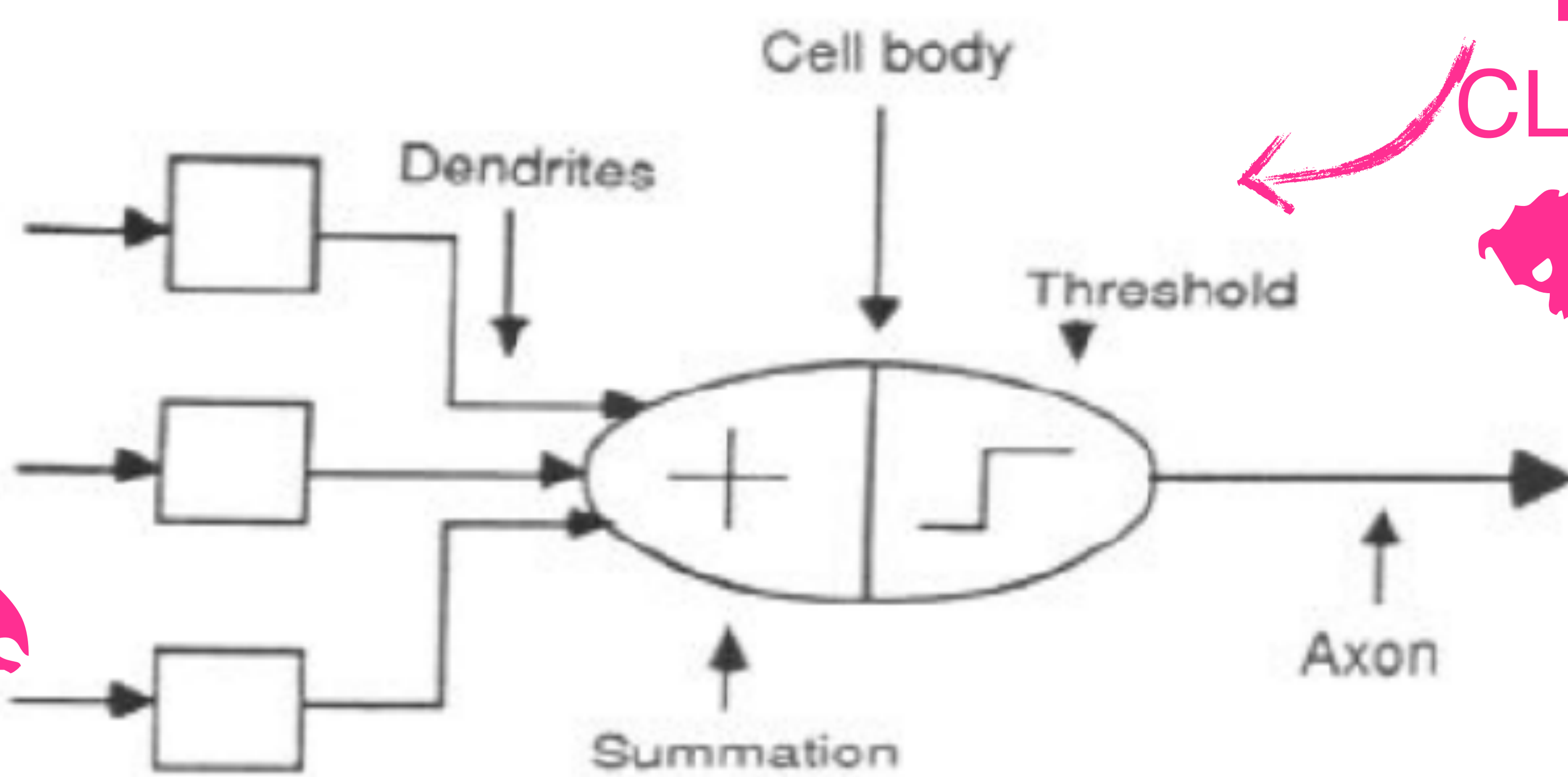
McCulloch & Pitt (1943)





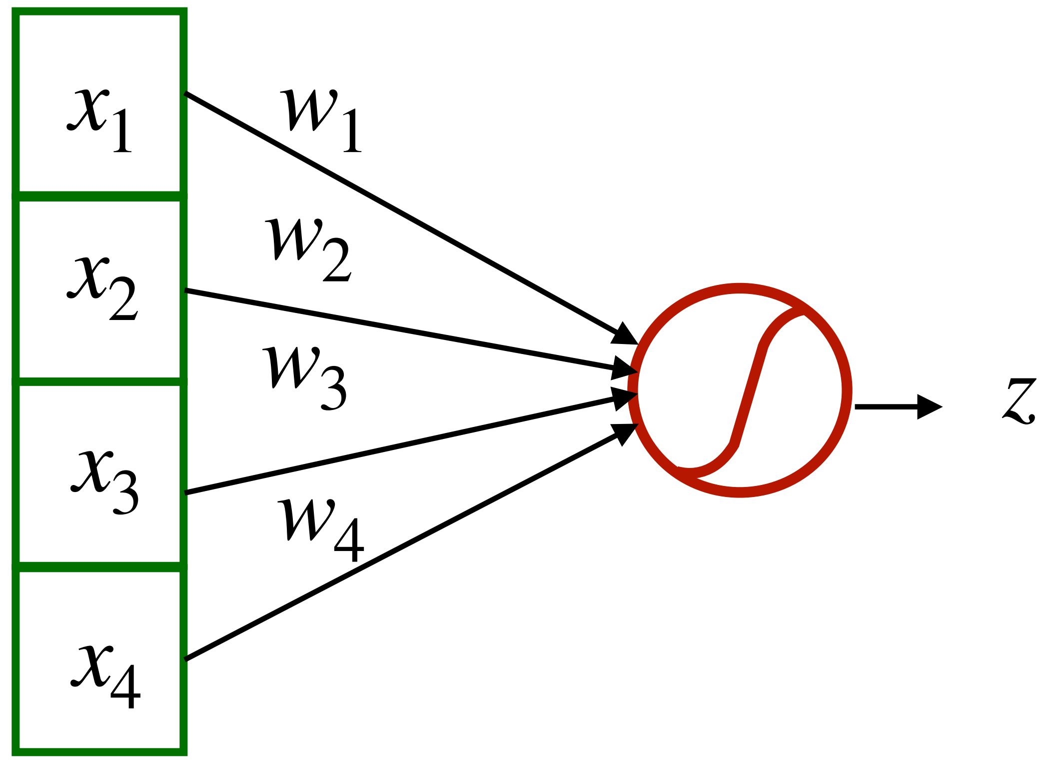
PERCEPTRON: LINEAR CLASSIFIER

x



y

A single neuron



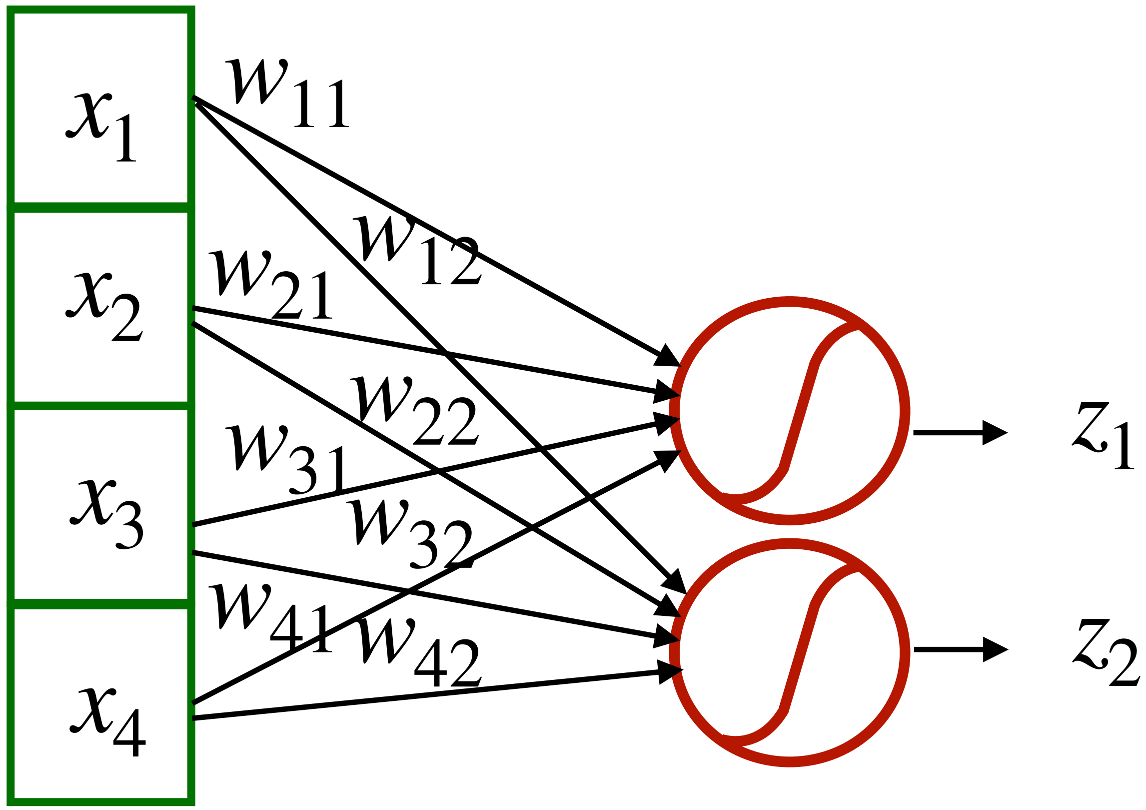
$$z = \sigma(x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 + b)$$

$$= \sigma(\mathbf{x} \cdot \mathbf{w} + b) \quad \textbf{with } \mathbf{x}, \mathbf{w} \in \mathbb{R}^4$$

(bias node omitted in visualization)

Multiple neurons

$$f_\theta : \mathbb{R}^4 \rightarrow \mathbb{R}^2$$



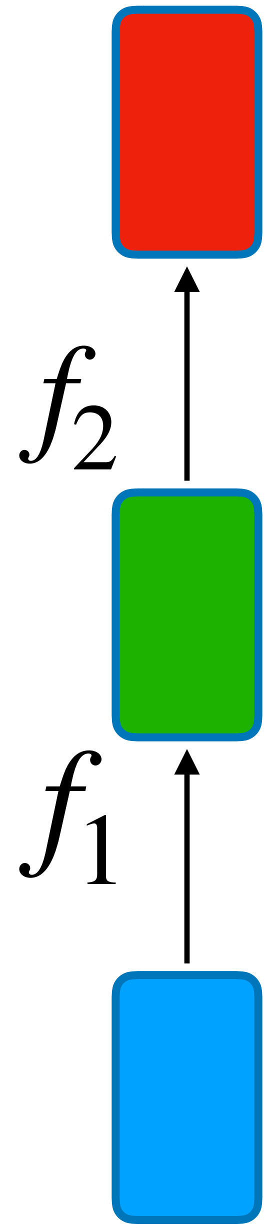
$$z_1 = \sigma(\mathbf{x} \cdot \mathbf{w_1} + b_1)$$

$$z_2 = \sigma(\mathbf{x} \cdot \mathbf{w_2} + b_2)$$

FFNN: abstract & functional view

$\hat{y} = \textbf{softmax}(f_2(f_1(x)))$ $\quad$ $error = \textbf{loss}(y, \hat{y})$

- Each layer is a **function**, acts on the **output** of the layer below (input)
- Final output: **cascade** of functions
- Given the true output y, we compute the **error Er**

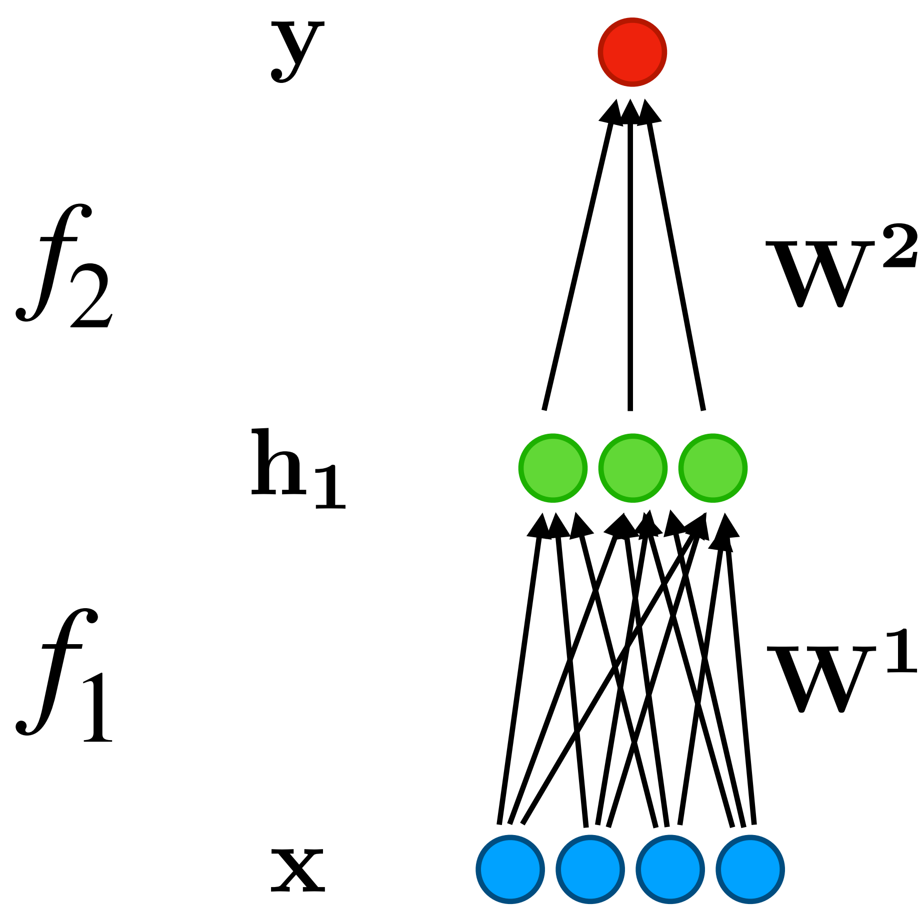


- Using the chain rule, we can compute the derivative (**gradient**) of the Error wrt any of the intermediate layer weights (Lecture 1)

“vanilla” Neural Network

FFNN: Graphical view

$\hat{y} = \textbf{softmax}(f_2(f_1(x)))$



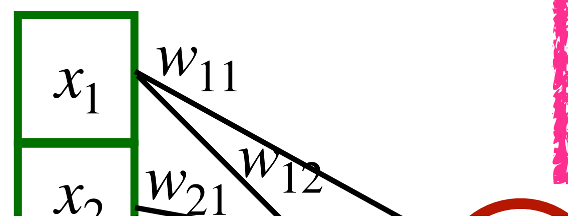
fully-connected layers

Multiple neurons: Vectorization

$z_1 = \sigma(\mathbf{x} \cdot \mathbf{w_1} + b_1)$
$z_2 = \sigma(\mathbf{x} \cdot \mathbf{w_2} + b_2)$

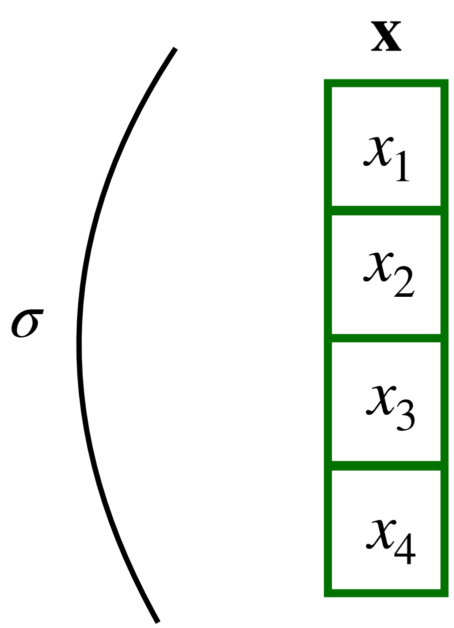
$\longrightarrow$ $\mathbf{z} = \sigma(\mathbf{x} \cdot \mathbf{W} + b)$

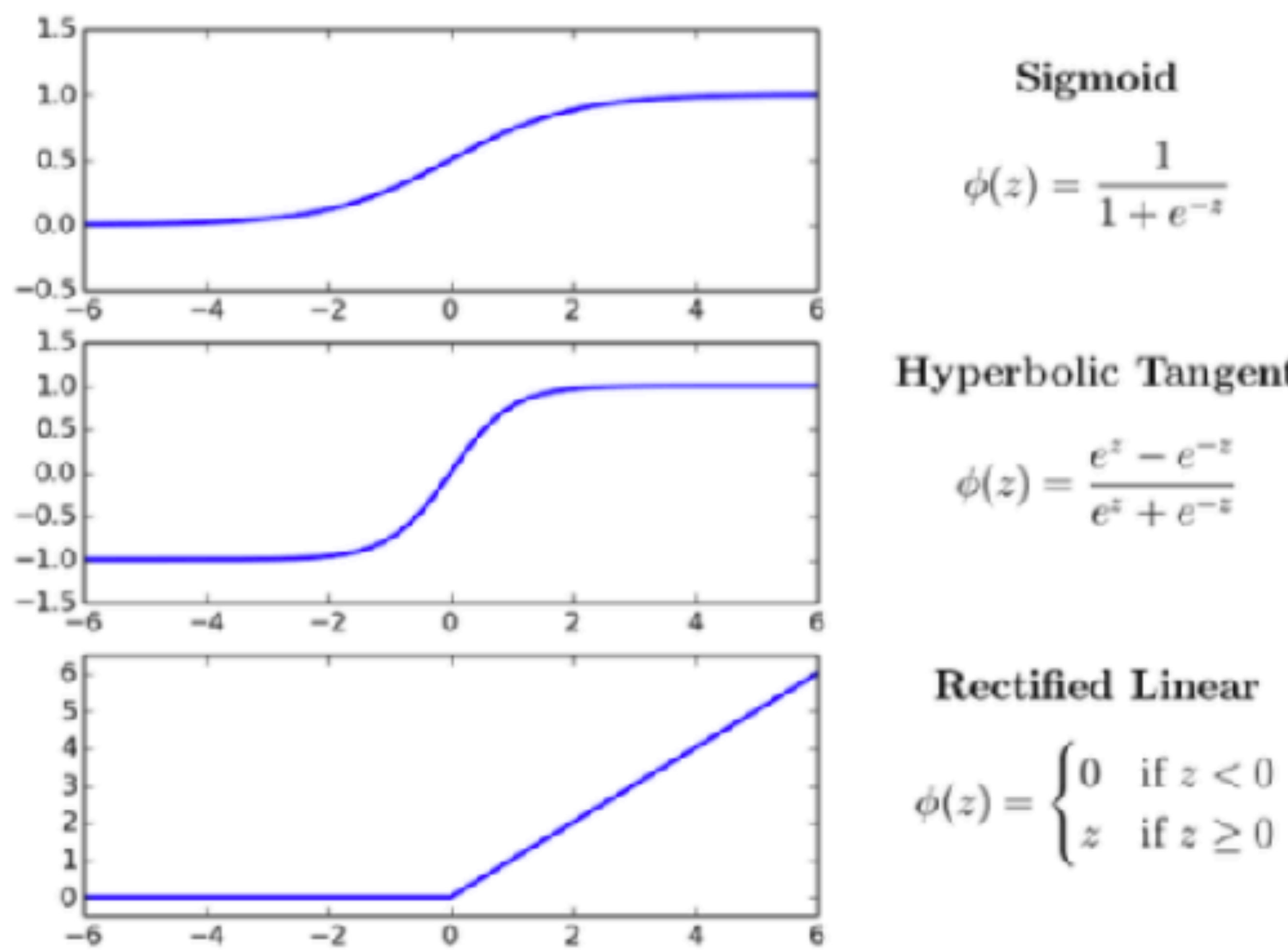
linear projection followed by non-linearity



$f_\theta : \mathbb{R}^4 \rightarrow \mathbb{R}^2$

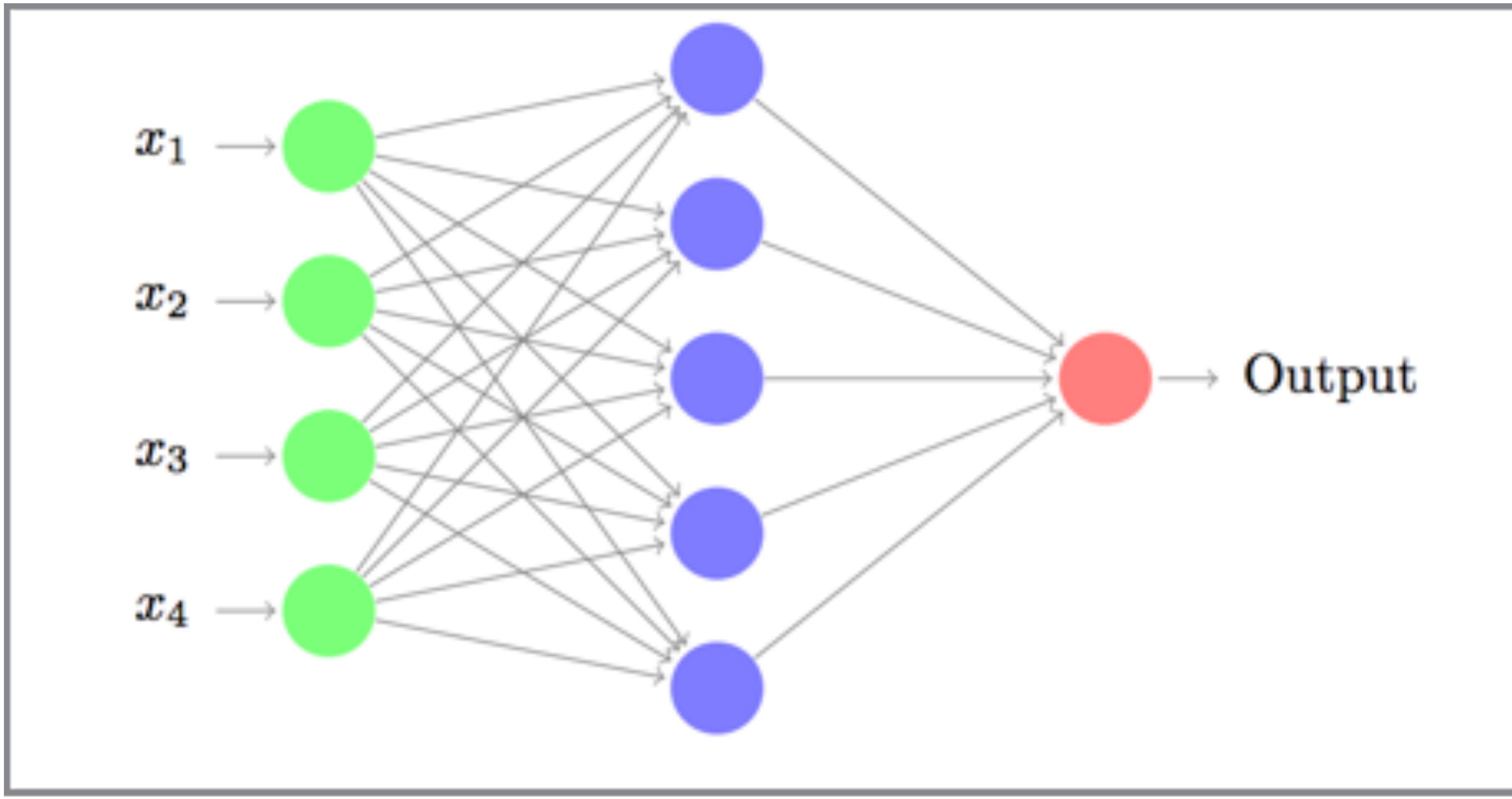
σ is the layer's (non-linear) activation function





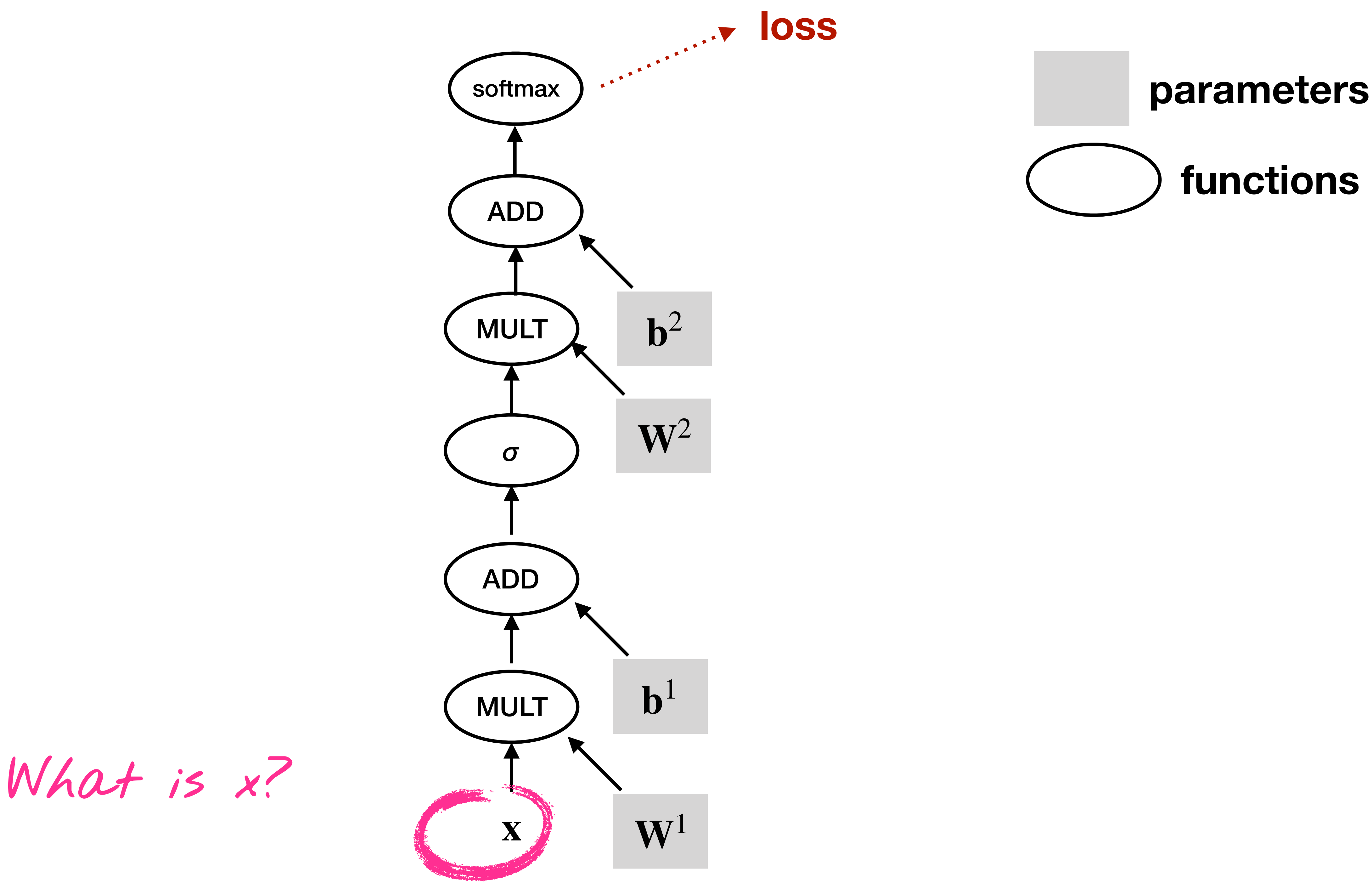
From Hughes and Correll 2016

Connecting the views: FFNNs (MLPs)



$$NN_{MLP1}(\mathbf{x}) = g(\mathbf{x}\mathbf{W}^{\mathbf{1}} + \mathbf{b}^{\mathbf{1}})\mathbf{W}^{\mathbf{2}} + \mathbf{b}^{\mathbf{2}}$$

Computation Graph View



$$NN_{MLP1}(\mathbf{x}) = g(\mathbf{x}\mathbf{W}^{\mathbf{1}} + \mathbf{b}^{\mathbf{1}})\mathbf{W}^{\mathbf{2}} + \mathbf{b}^{\mathbf{2}}$$

Overview

foundations

Motivation,
Brief History, Overview

Back to the roots:
Language Models

Feedforward NNs
(FFNNs)

representations

What's the input?
Representations

What does 'eienskappe' mean?

left context	KWIC	right context
wat help lei het tot sy ontdekking. Die asteroïde se wentelbaan het chaotiese	eienskappe	, wat langtermynvoorspellings bemoeilik. Voor 500n.C. kon dit by die Son-Aar
beperk. 1943 – Albert Hofmann skryf sy eerste verslag van die hallucinogenic	eienskappe	van LSD. 1954 – VSA senator Joseph McCarthy begin verhore om die VSA le
ou klere is vuil, jou vroutjie die huil". In "Die spinnekop en vlieg" beskryf hy die	eienskappe	van die spinnekop en hoe hy die vlieg met listigheid vang, waarna hy die vlieg
r betrokke was. As gevolg hiervan is 'n patroon en vuurwapen geseek wat die	eienskappe	van 'n submasjiengeweer (hoëkapasiteit-magasyn en ten volle outomatiese v
assaproduksie. Die AK-47 kan gesien word as 'n samesmelting van die beste	eienskappe	van die M1 Garand en die StG44. Met die aanvanklike vervaardiging was daa
yne as dekoratiewe plante, word die plant ook gebruik vir die geneeskragtige	eienskappe	wat daaraan toegeskryf word. Hierdie plante word ook in rotstekeninge van di
se Ebers-papirus uit ongeveer 1552 v.C. verwys na die aalwyn se medisinale	eienskappe	en die gebruik daarvan in die balsem van lyke. Die meeste van Suid-Afrikaans
rende Kerstyd ryp word. Die Khoikhoi het reeds hierdie plant vir sy medisinale	eienskappe	gewaardeer en die Boere het dit oorgeneem. Dit is veral as purgeermiddel ge
erte is vir gryp aangepas; terwyl altwee groepe se oë voorwaarts gerig is. Die	eienskappe	van elke aapgroep is beter verstaanbaar deur dit as 'n afsonderlike spesie te b
iemetaalstowwe gebruik het wat wateronoplosbaar en verhittingsbestand is –	eienskappe	wat ook hierdie oksiede het. Die besef dat hierdie aardes nie elemente was ni
raling van die Son en maak so lewe op land moontlik.[19] Die Aarde se fisiese	eienskappe	, geologiese geskiedenis en posisie in die Sonnestelsel maak die volgehoue b
astelandsplaat is onder seevlak. Hierdie onderwater oppervlak het bergagtige	eienskappe	, met bergreekse, vulkane, trôe, skeurdalle, plato's en vlaktes. Die oorblywend
slag in 'n gebied word bepaal deur die dominante windrigting, die topologiese	eienskappe	en die temperatuurverskille.[69] Ondanks die plaaslike verskille kan die Aarde
Wetenskaplikes maak weer eens van seismiese golfsnelheid gebruik om- die	eienskappe	van die rotslae te bepaal. Die vernaamste kenmerke van die gesteentes word
die warm mantelgesteentes die rug binnegedring het, verloor dit die plastiese	eienskappe	, en by dié "stolling" verkry dit die magnetiese eienskappe van die heersende
oor dit die plastiese eienskappe, en by dié "stolling" verkry dit die magnetiese	eienskappe	van die heersende magneetveld, dit word met ander woorde gepolariseer. Die
by die poolstreke binne – vandaar die pooligte. Sou die aarde se magnetiese	eienskappe	met dié van 'n gewone staafmagneet vergelyk word, sal die as wat van pool to
te ontwikkel) om aan besondere kriteria te voldoen, soos veiligheid, estetiese	eienskappe	, ekonomiese geleenthede, die bewaring van die bestaande natuurlike erfenis

Keyword in Context (KWIC)

Distributional Hypothesis

"You shall know a word by the company it keeps"
(Firth, J. R. 1957:11)

The company it keeps

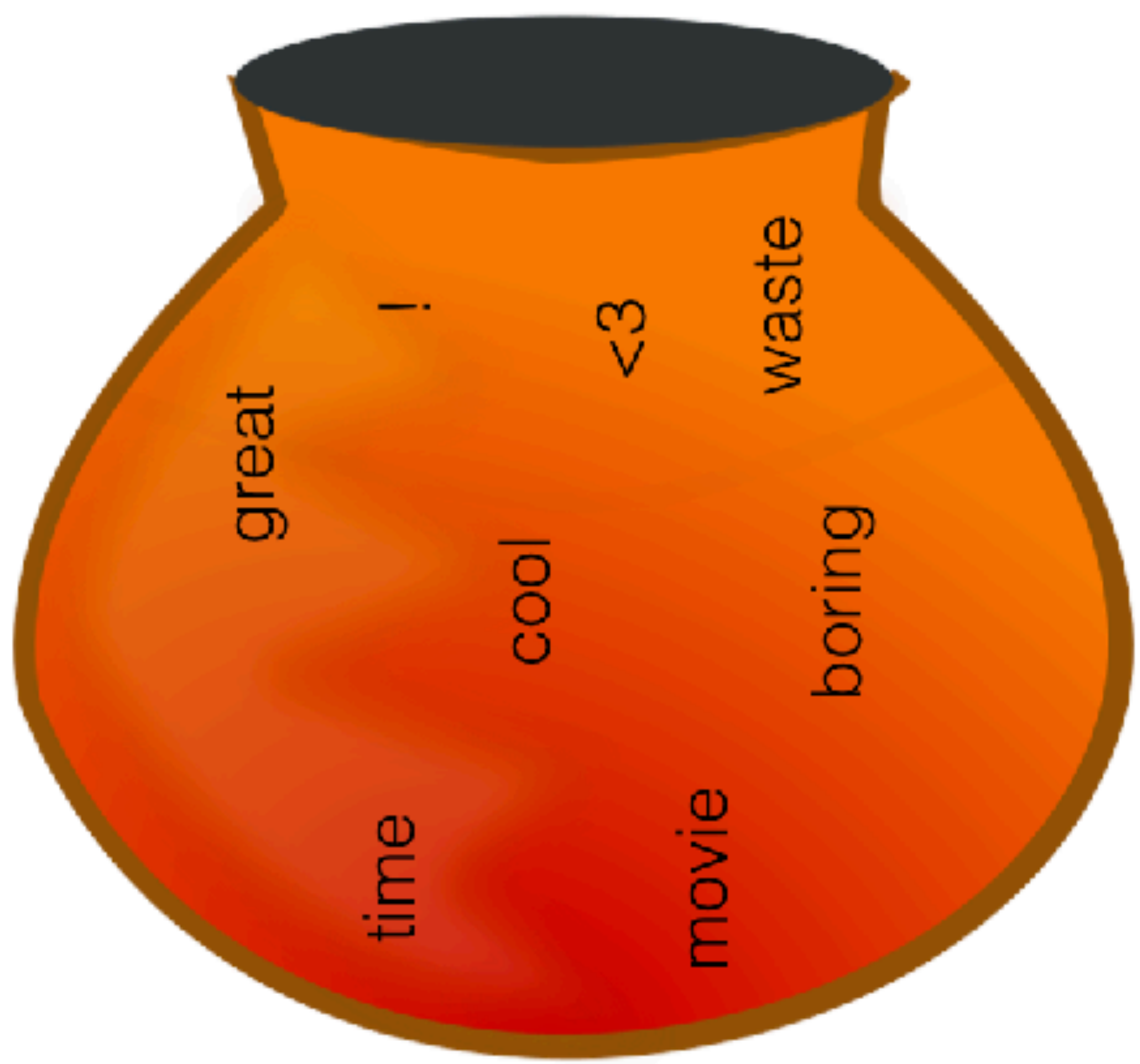
- Key idea in NLP: the meaning of a word is represented by the words which occur frequently close to it
- One of the most successful ideas in NLP
- Nowadays, we talk about **representations**

What are good representations?

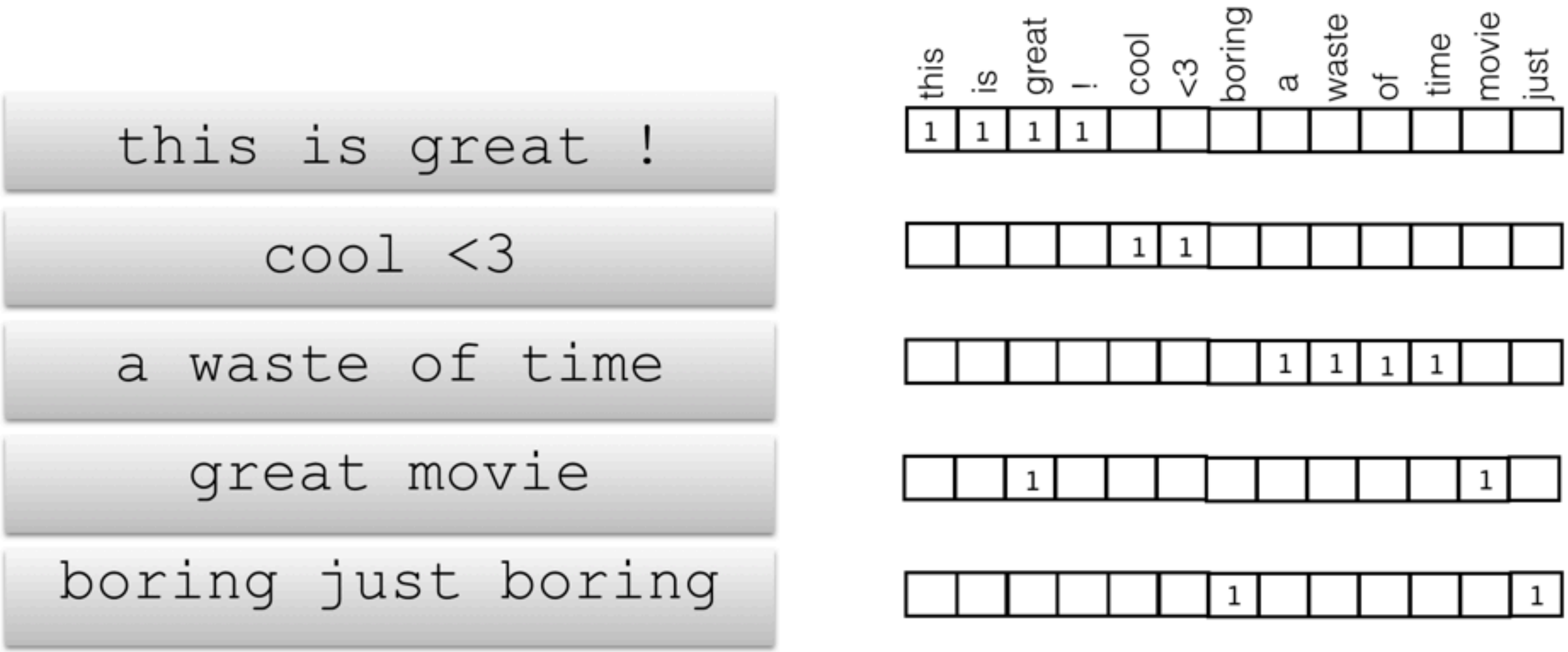
- Representations are **distinct**
- Similar words (or units) have **similar** representations

Traditional sparse text encoding: BOW

bag-of-words



Sparse binary text encoding: BOW



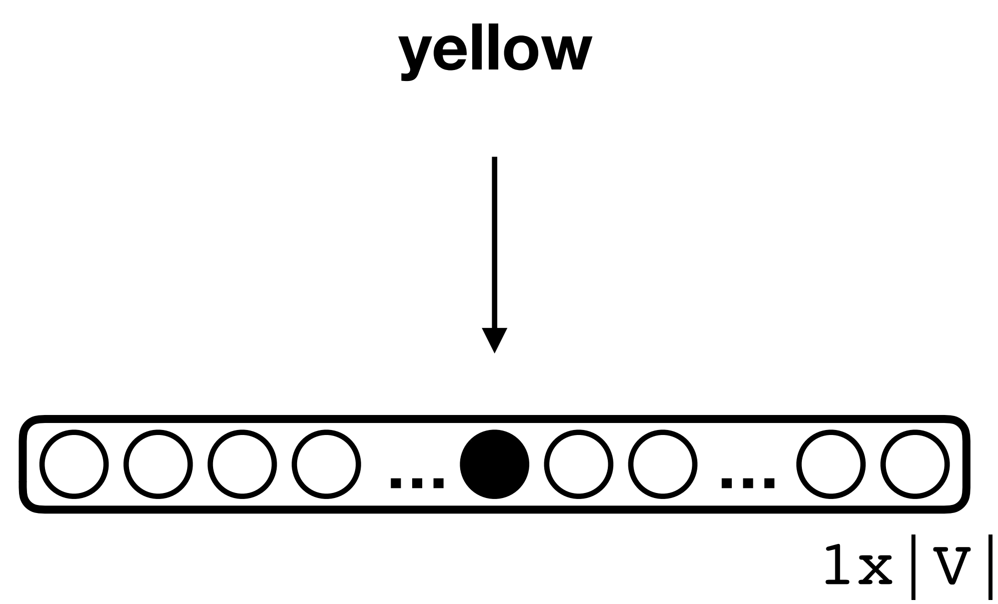
n-hot encoding

One-hot encoding

- Sparse high-dimensional vector of dimension |V| (=size of vocabulary)

Symbol (word, char,..)

one-hot vector
(length V, one entry is 1)



One-hot encoding: Sparse binary repr.

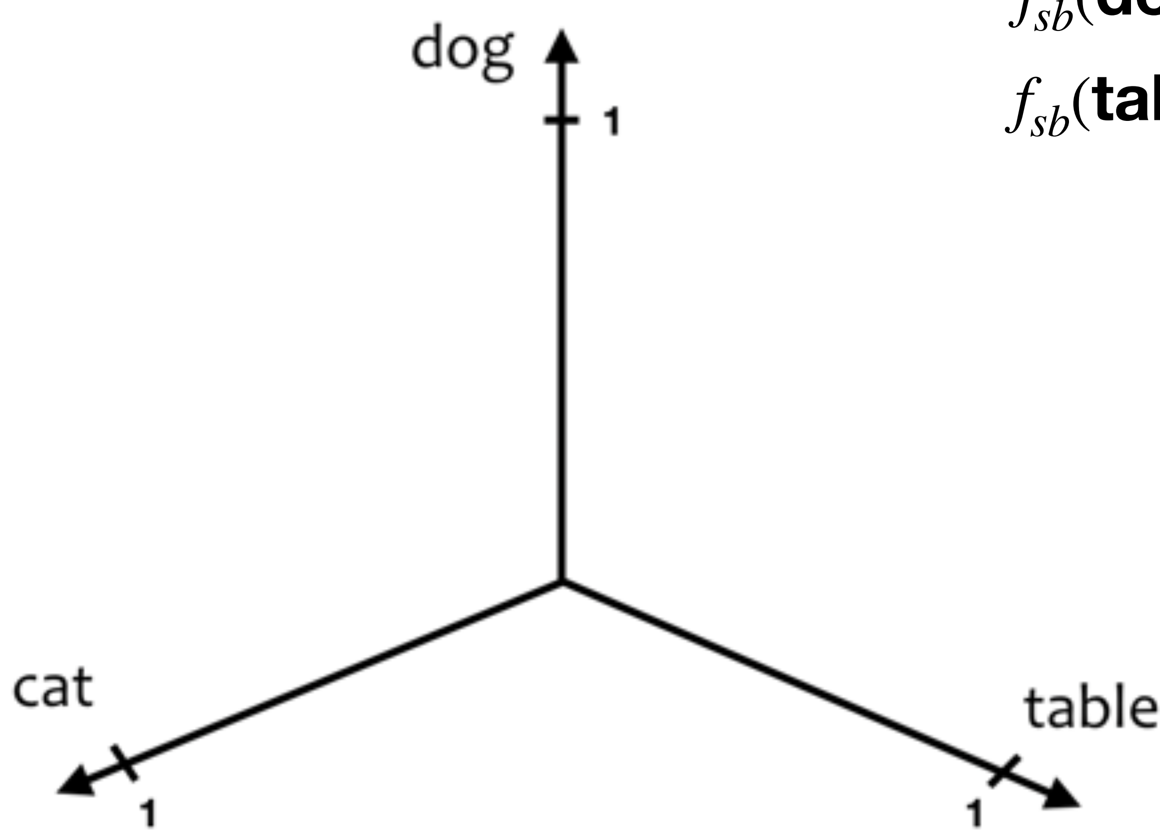
- sb: sparse binary representation

$\mathbb{V} = \{\textbf{cat}, \textbf{dog}, \textbf{table}\}$

$f_{sb}(\textbf{cat}) = [1,0,0]$

$f_{sb}(\textbf{dog}) = [0,1,0]$

$f_{sb}(\textbf{table}) = [0,0,1]$



$cos(f_{sb}(\textbf{cat}), f_{sb}(\textbf{dog})) = 0$

(Illustration adapted from S. Riedel)

Sparse binary representations

- Representations are distinct 
- Similar symbols have similar representations 

- Despite of this, n-hot representations are often very **powerful** for text classification.

From sparse high-dim to continuous low-dim

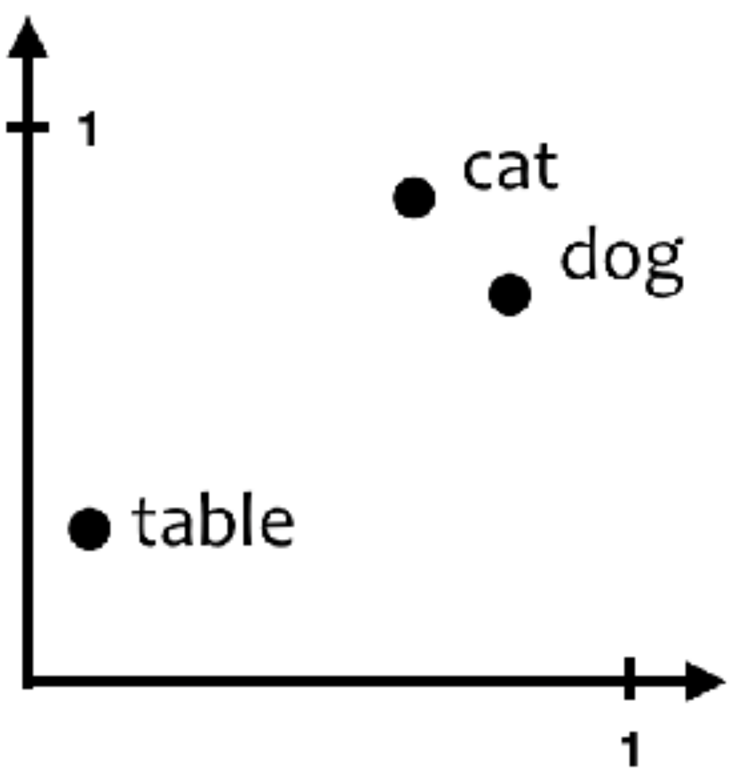
Dense continuous: Embeddings

- "Embed" symbol $f_{dc}(w) \mapsto \mathbb{R}^d$
  in dense low-dimensional space (d << |V|)
- Dimensionality d (hyperparameter)

$\mathbb{V} = \{\textbf{cat}, \textbf{dog}, \textbf{table}\}$

$d = 2$

$E \in \mathbb{R}^{3\times2}$



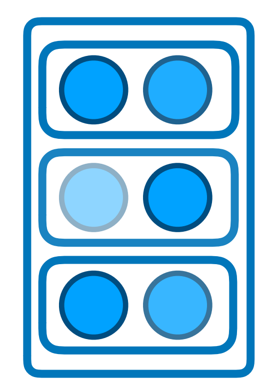
$f_{sb}(\textbf{cat}) = [0.7,0.8]$

$f_{sb}(\textbf{dog}) = [0.75,0.6]$

$f_{sb}(\textbf{table}) = [0.1,0.15]$

=

E



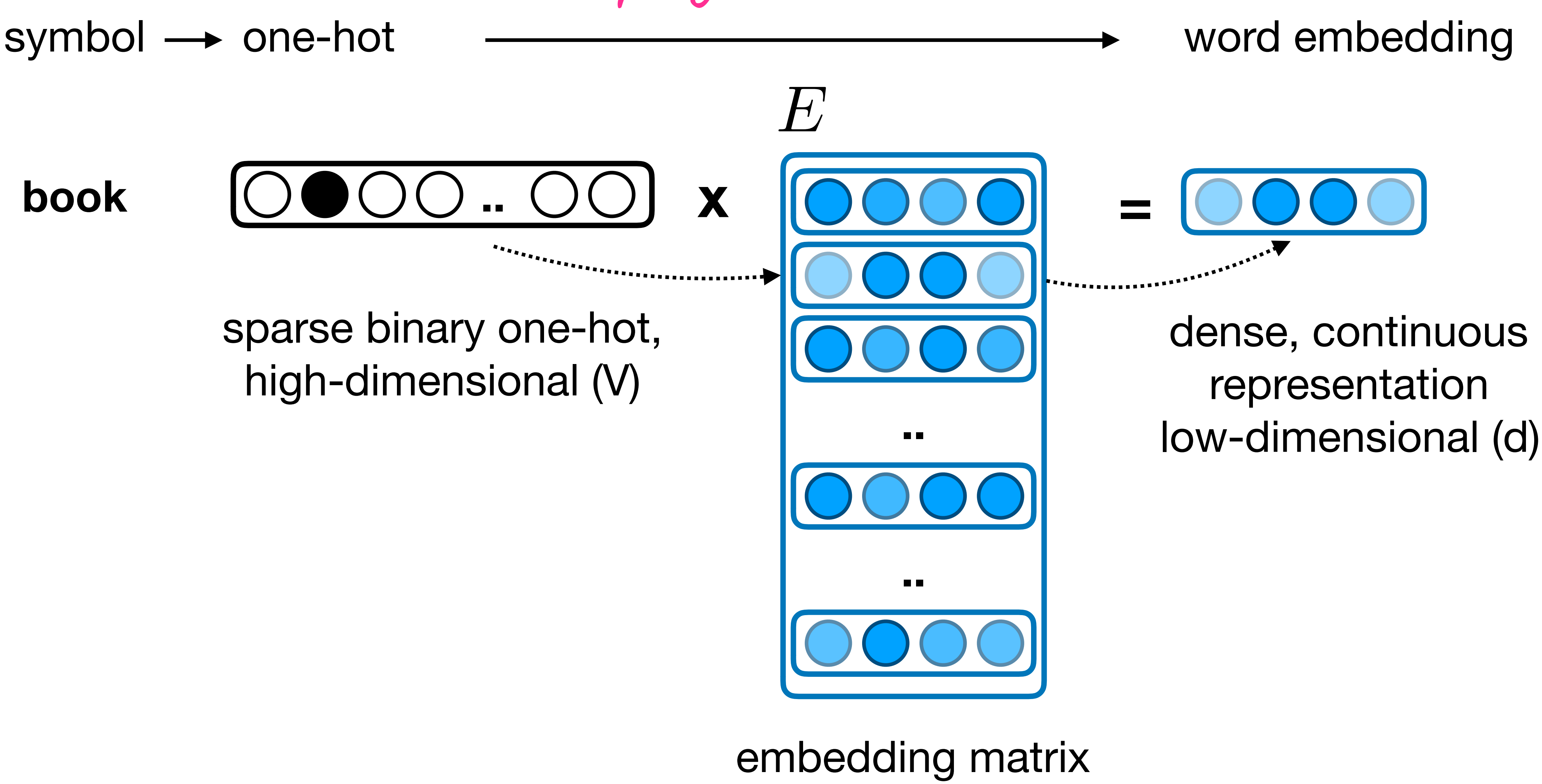
|V|xd

embedding matrix

Note: d < |V|

Lookup: Representing a symbol

linear projection from V->d

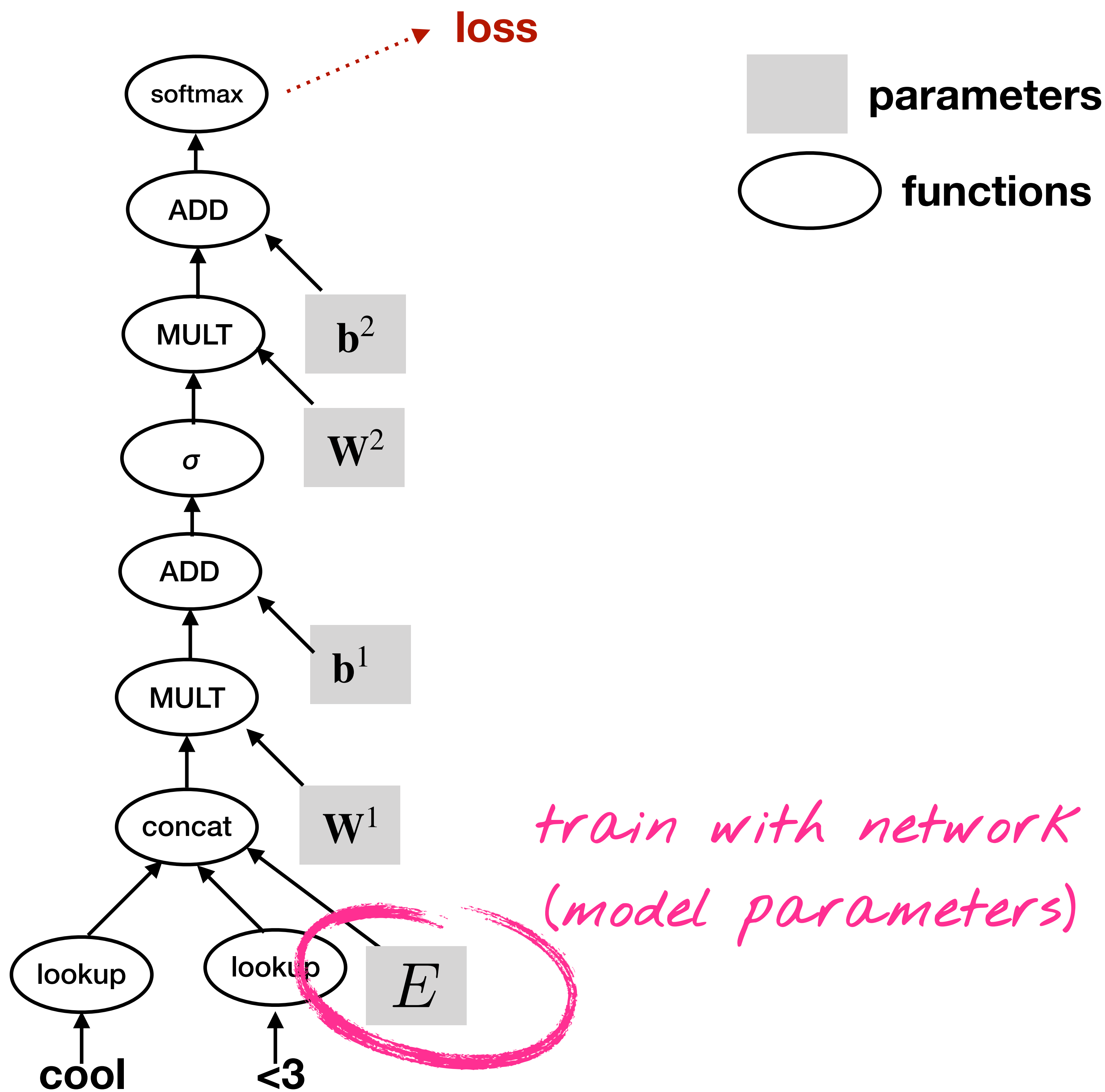


Adapted from Lecture 5 (Riedel) http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture06-rnnlm.pdf

In general, the neural way for extracting features:

- Extract **core linguistic features** $f_1, ..f_n$
- **Define** a **vector** for each feature (lookup Embedding table)
  - Can train representation E together with the network

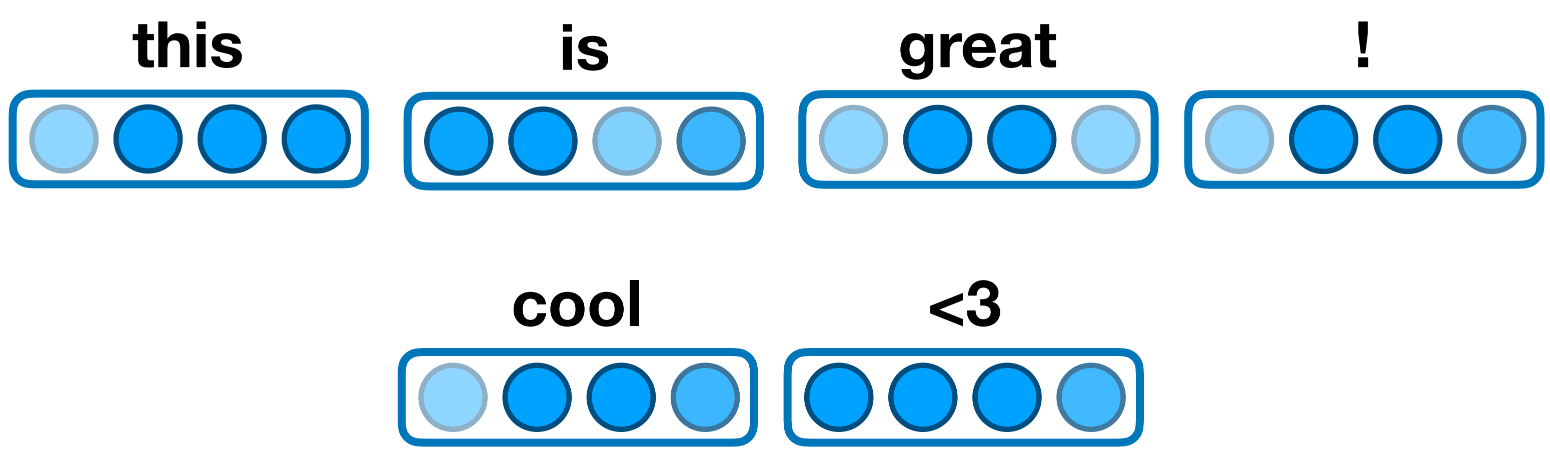
Computational Graph View



Dense continuous text encodings

`this is great !`

`cool <3`



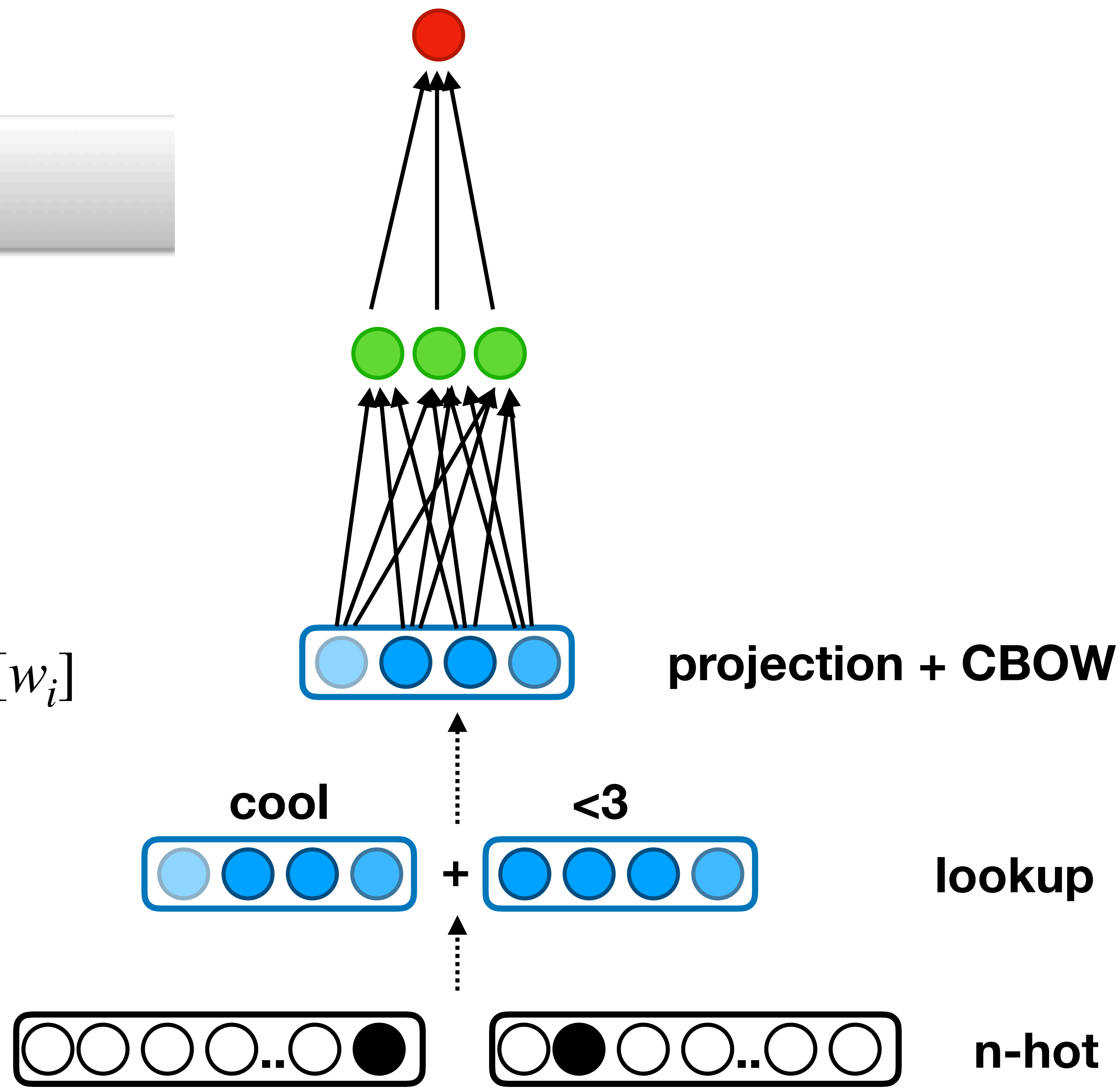
How to combine word embeddings?

Dense continuous text encoding: e.g. continuous BOW (CBOW)

Example input document 1:

`cool <3`

$$\textbf{CBOW}(w_i, \ldots, w_n) = \sum_{i}^{n} E[w_i]$$

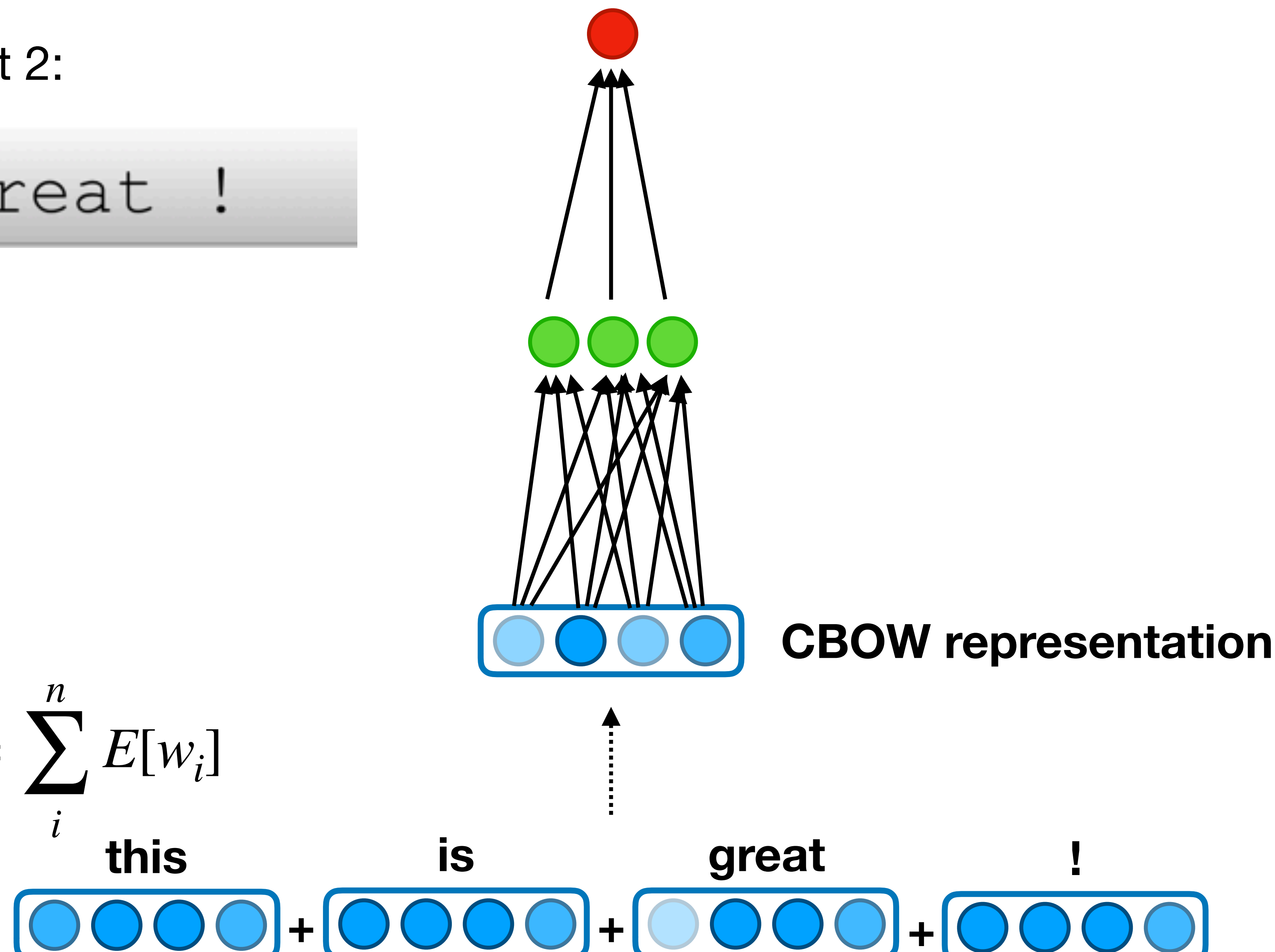


Dense continuous text encoding: e.g. continuous BOW (CBOW)

Example input document 2:

`this is great !`

$$\textbf{CBOW}(w_i, \ldots, w_n) = \sum_{i}^{n} E[w_i]$$



Limitation of BOW

- What's the biggest limitation of BOW/CBOW?
  - Similar to unigram model: It **disregards the order of items** (e.g. words in a sequence)
  - Example:
    - "it was not good, it was actually terrible"
    - "it was not terrible, it was actually good"
- A simple solution?

Possible Improvement

- Bag of n-grams
  - “not good”, …
- Problems:
  - Parameter explosion (BOW/n-hot) or even more averaging (CBOW)
  - No sharing between similar words & n-grams

Where to get task-specific E from? From Scratch vs Pre-trained

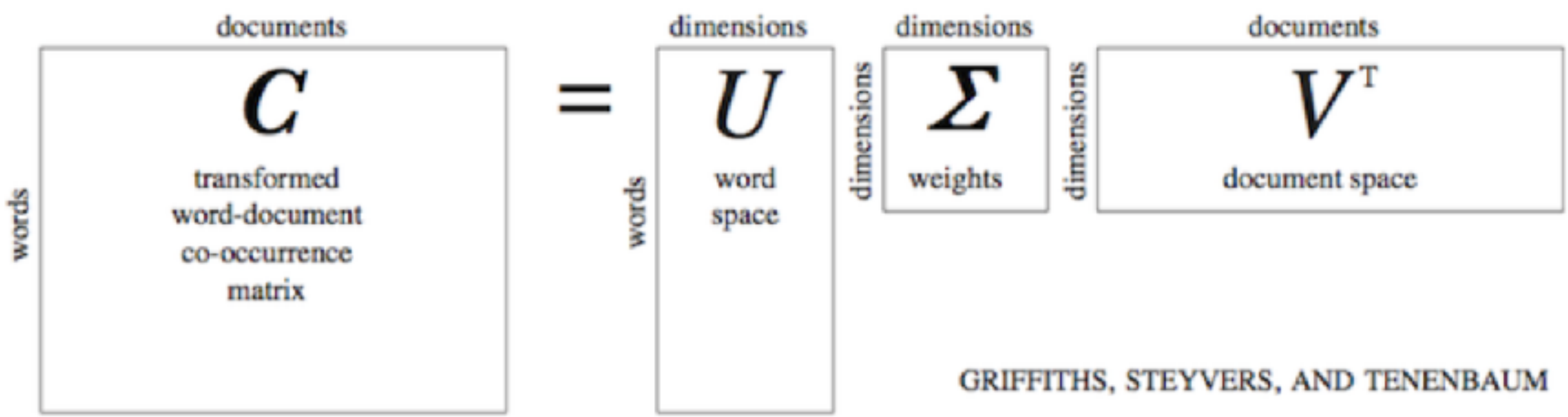
- Embedding layer *E*:
  - trained with network from **scratch** - task-specific
  - **initialized** with off-the-shelf pre-trained word embeddings (e.g., Glove, Polyglot, fastText)
- Pre-trained embedding **initialization** typically leads to performance gains. Why?
  - train on **more words**
  - implicitly **more data**
- Ways to obtain off-the-shelf embeddings? (word vector space representation?)

Embeddings: New? No!

- **Two major methods:**
  - **Count!** (pre-deep learning method, aka “word vector space models”)
  - **Predict!** (core idea underlying word2vec - lecture 1)

Count-based methods

- Represent the “company” of a word in terms of a word **co-occurrence-matrix,** get the statistics (counts)
- E.g. **Latent Semantic Analysis** (LSA) (Deerwester et al., 1990) SVD decomposition over co-occurence matrix to reduce to lower-dimensional space (matrix U where dim < |docs|)



https://simonpaarlberg.com/posts/2012-06-28-latent-semantic-analyses/box2.png

Prediction-based methods

- **Key idea:** predict the context of a word (instead of capturing co-occurrence statistics in matrix C) to directly learn the low-dimensional word vector representation
- ***Word2vec*** (family of methods) Mikolov et al. (2013) [Lecture 1 by Ryan] - scales well to large data

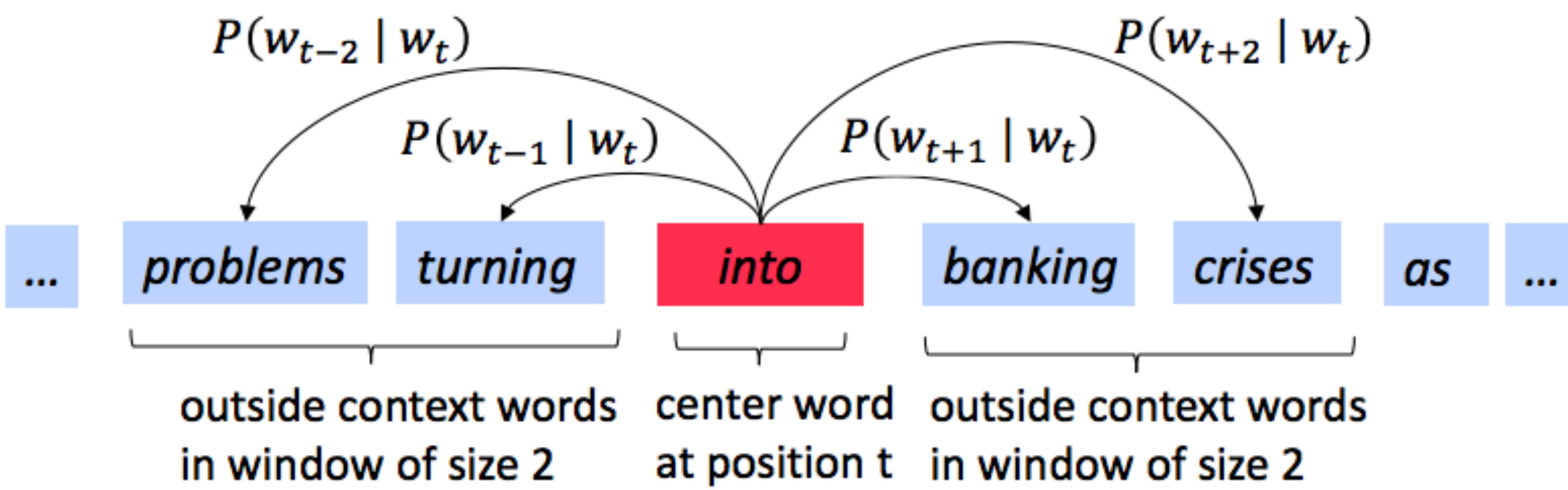
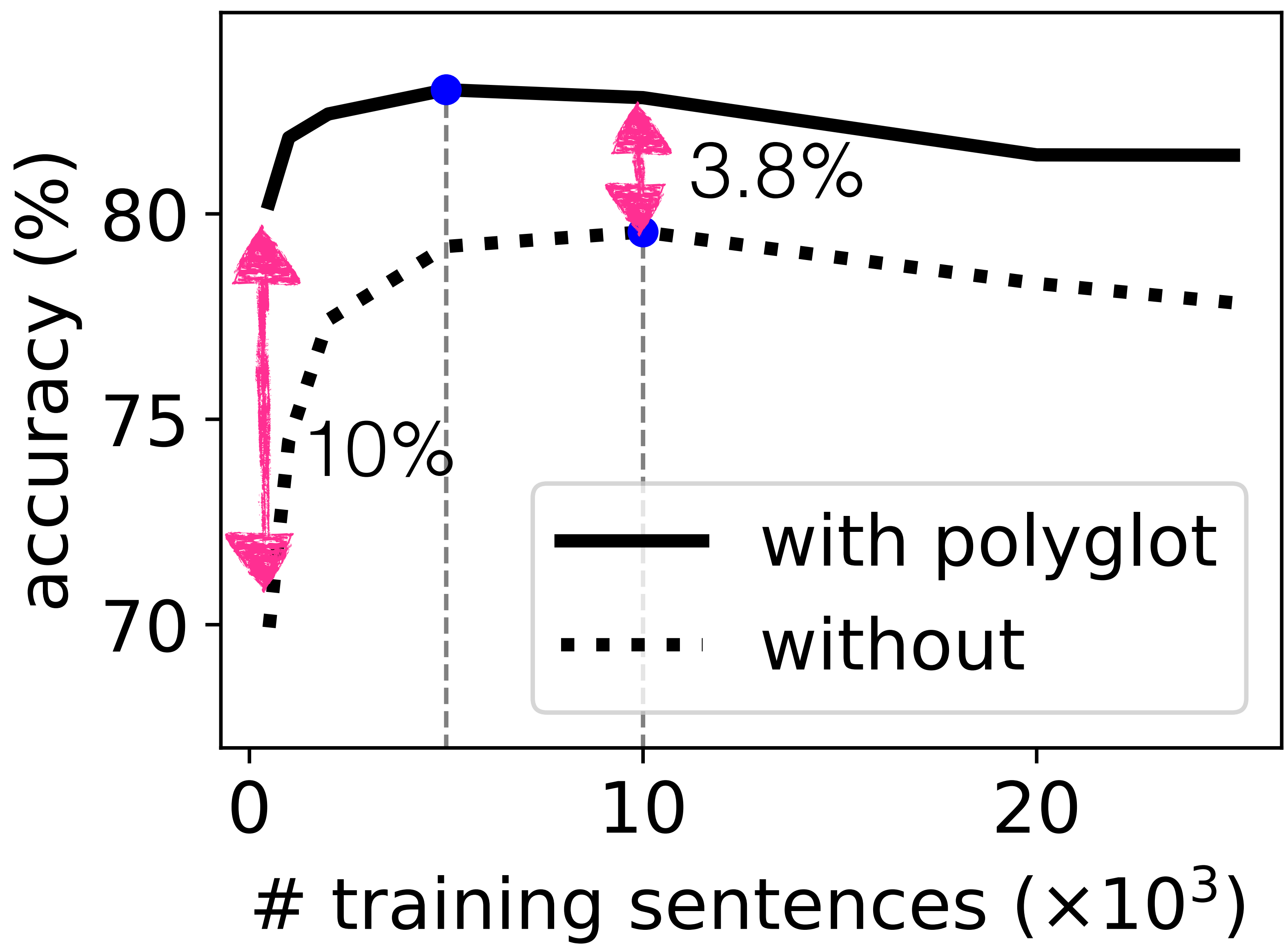


Illustration by Chris Manning

Example: Cross-lingual POS tagging-Word embedding initialization

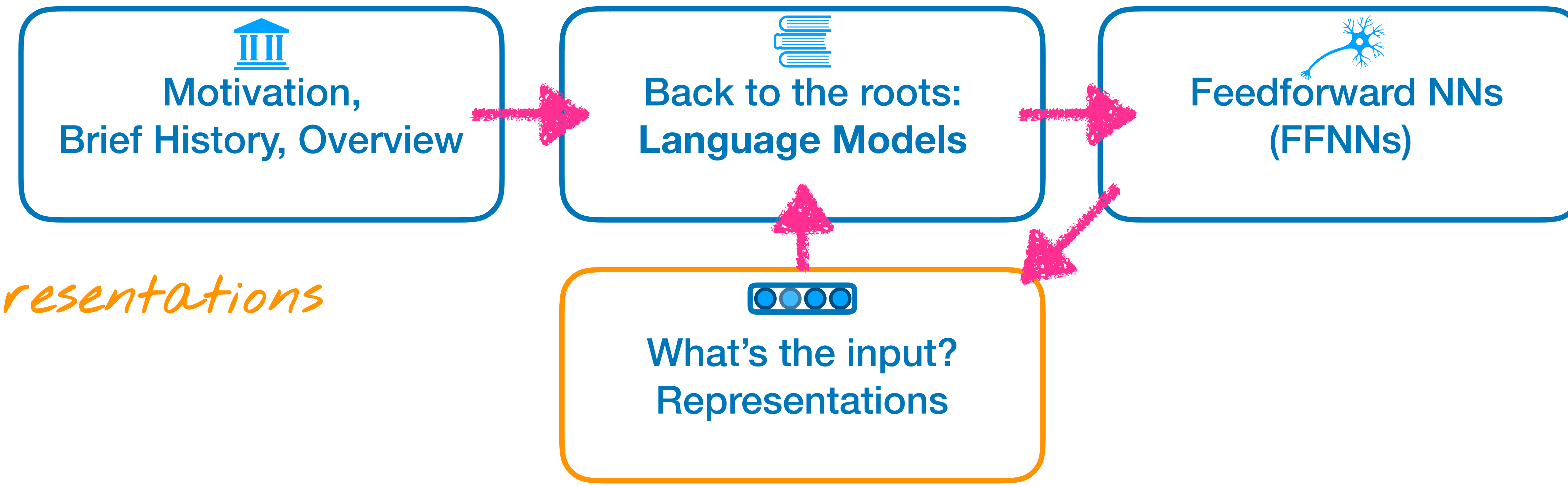
(Plank & Agic, 2018)



Mean over 21 languages

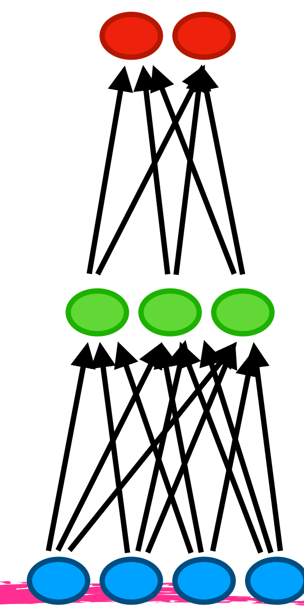
Overview

foundations



representations

Window-based neural LM via FFNN



predict!

~~As the clock~~ rang the students opened the

fixed window of n words

A fixed-window based neural LM

output distribution

$\hat{\boldsymbol{y}} = \mathrm{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$

hidden layer

$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$

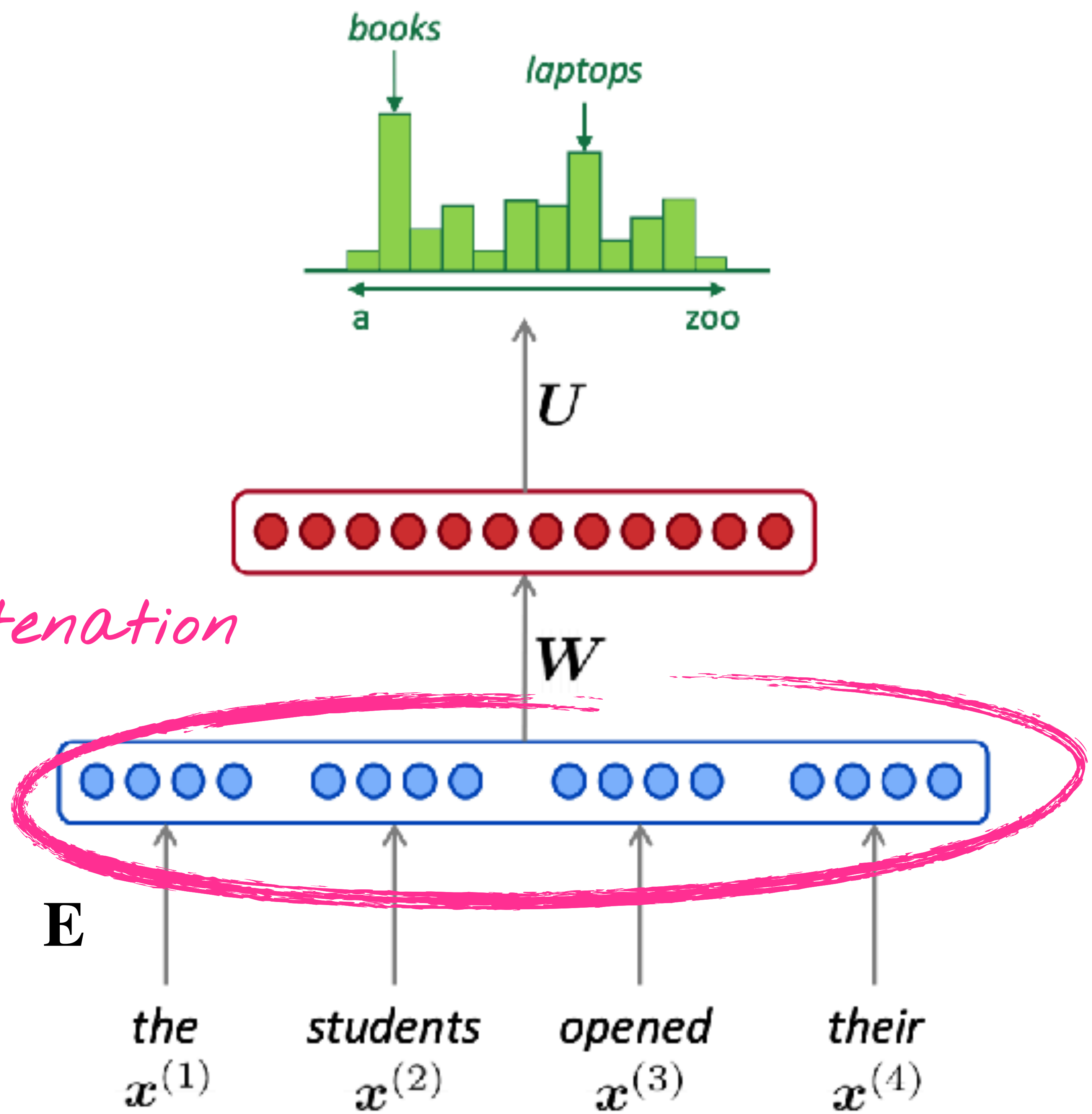
concatenation

concatenated word embeddings

$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$

words / one-hot vectors

$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$



Training the Neural n-gram LM

- Iteratively move the n-gram window through a very large corpus to predict the next word at each time step
- Cross-entropy loss (negative log-likelihood):

$$L = -\,logp(w_t \mid w_{t-1}\,..\,w_{t-n+1})$$

- Note: typically very large vocabulary (softmax)
    - Workaround: negative sampling (lecture 1)

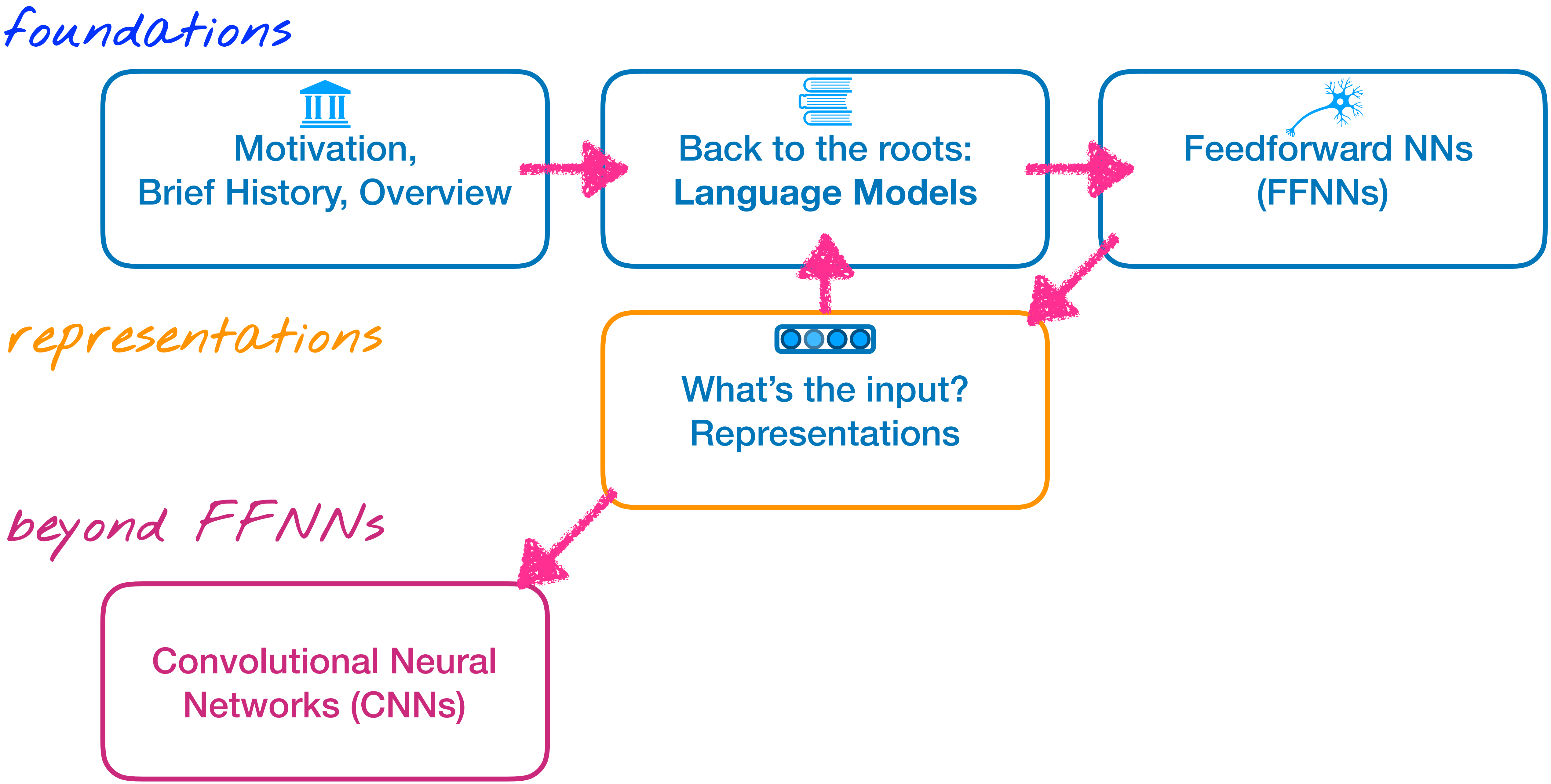
What about these issues?

- Can it handle similar words?
  - she *bought* a bicycle
  - she *purchased* a bicycle
- Long-distance dependencies?
  - for *programming* she yesterday purchased her own brand new *laptop*
  - for *running* she yesterday purchased her brand new *sportswatch*

Tips for unknown words

- Simplest solution:
  - Train an <UNK> word vector, e.g., map rare words to UNK (count < threshold)
- Problem:
  - Conflates a long tail into the same vector representation
- Subword representations (character-level models) to the rescue!
  - More on these later (after we have seen CNNs)

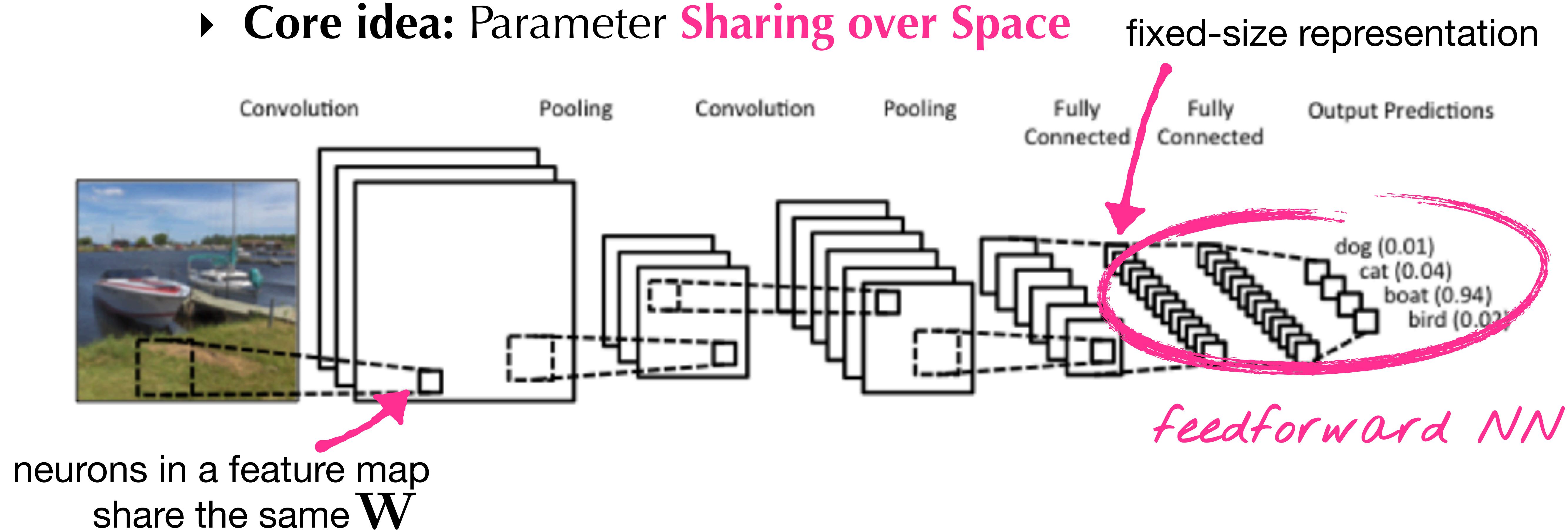
Overview


foundations
Motivation,
Brief History, Overview
Back to the roots:
Language Models
Feedforward NNs
(FFNNs)
representations
What’s the input?
Representations
beyond FFNNs
Convolutional Neural
Networks (CNNs)

CNNs / Convnets

Convolutional Neural Network (CNN)

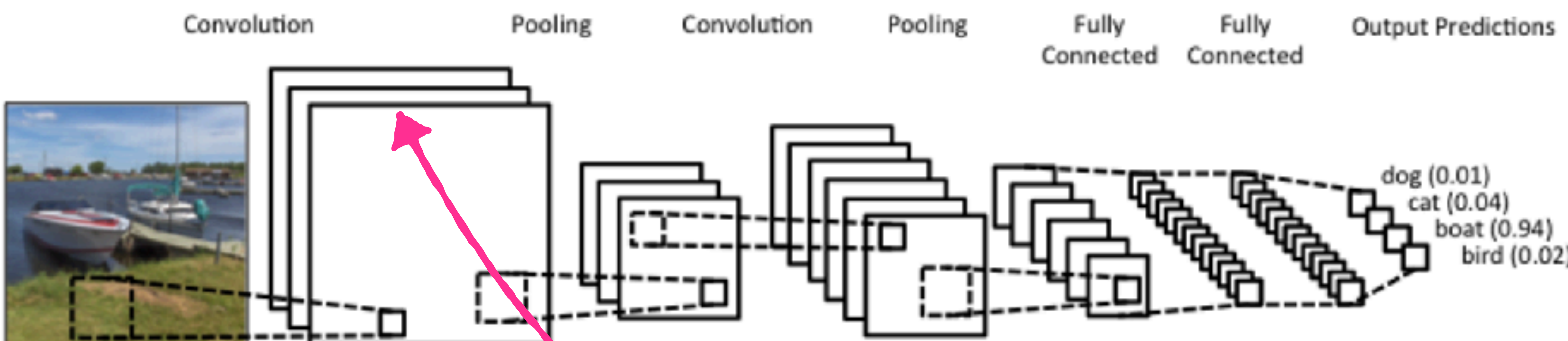
- Neural networks for processing data with a **grid-like typology** (LeCun & Bengio, 1995)
- **Can handle arbitrary-length inputs** and reduce them down to **a fixed size vector** representation
- **Core idea:** Parameter **Sharing over Space**



[1] http://www.deeplearningbook.org/contents/convnets.html

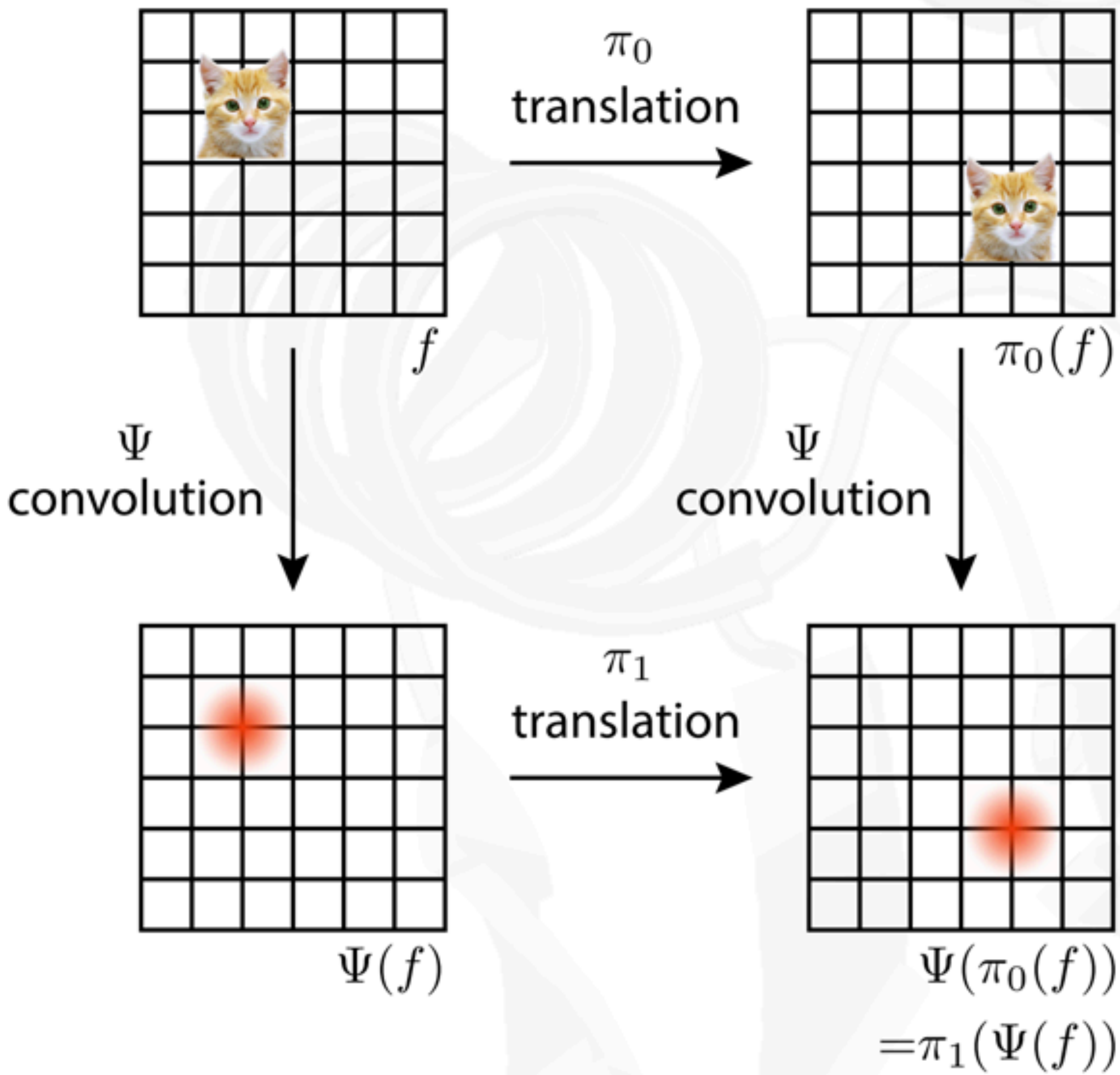
What are CNNs - Terminology

- CNNs use **convolutions** over the input (convolution + pooling)
  - Each convolution applies filters (or kernels; often several hundreds of them) and combines their results via **pooling** (to reduce the resolution of the feature map and the sensitivity of the output to shifts and distortion)



a convolutional layer has typically
several feature maps (with different $\mathbf{W}$)
to extract different features

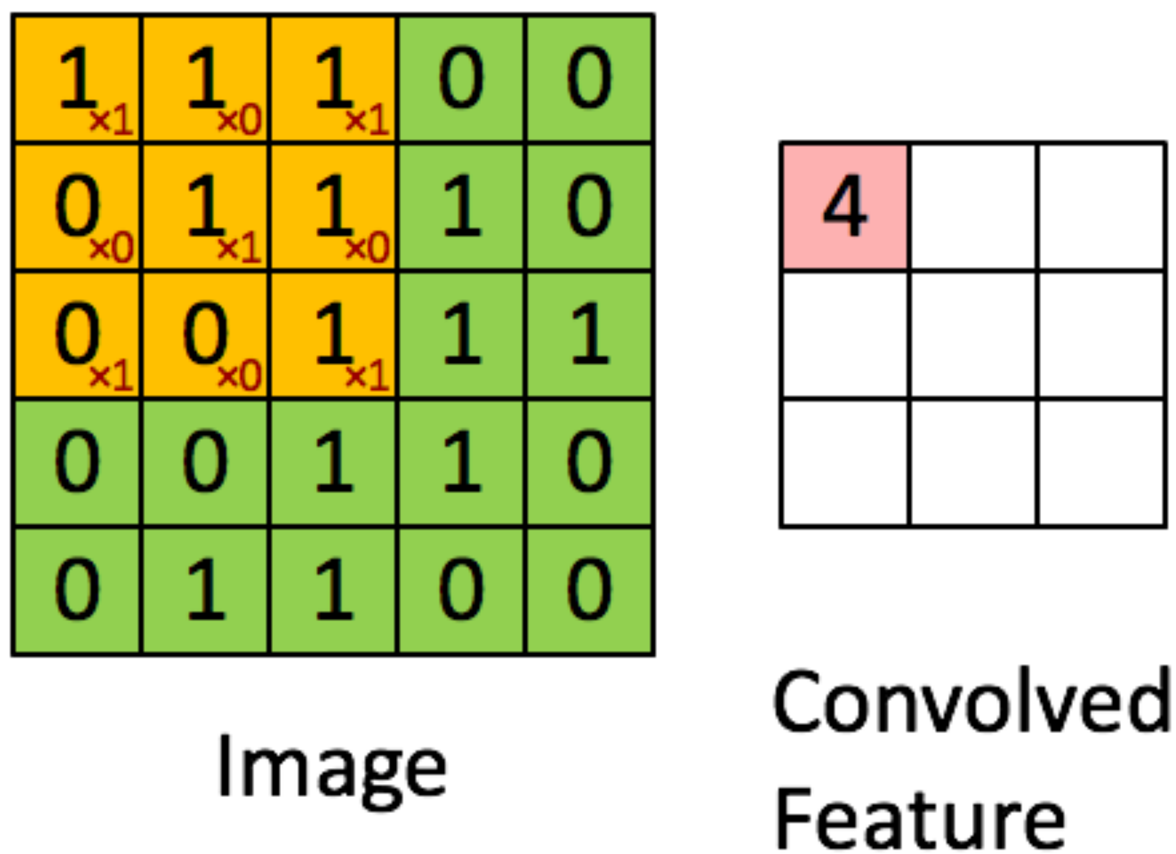
Translational equivalence



Slide by Jes Frellsen

Example of a 2D convolution

- Filter (kernel) of size 3x3
  - “to identify indicative local predictors" (Goldberg, 2015)



Source: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Convolution - Filter (Kernel) example

What is this Kernel doing?

computing a moving average

- Imagine a 1d input vector
  - f: [10, 50, 60, 10, 20, 40, 30]
  - g: [1/3, 1/3, 1/3]

$$(f * g)(i) = \sum_{j=1}^{m} g(j) \cdot f(i - j + m/2)$$

- Let's compute the value at position h(3)

[10, 50, 60, 10, 20, 40, 30]

[0, 1/3, 1/3, 1/3, 0, 0, 0]

$$50 * \frac{1}{3} + 60 * \frac{1}{3} + 10 * \frac{1}{3} = 40$$

$$h(3) = 40$$

$$h(4) = 30$$

Convolutions for Text

Collobert et al. (2011); Kim (2014)

Types of convolution

She likes strong coffee

She likes

likes strong

strong coffee

narrow/“valid” convolution

<s> She likes strong coffee </s>

wide/“same” convolution

(padded)

CNN on Text

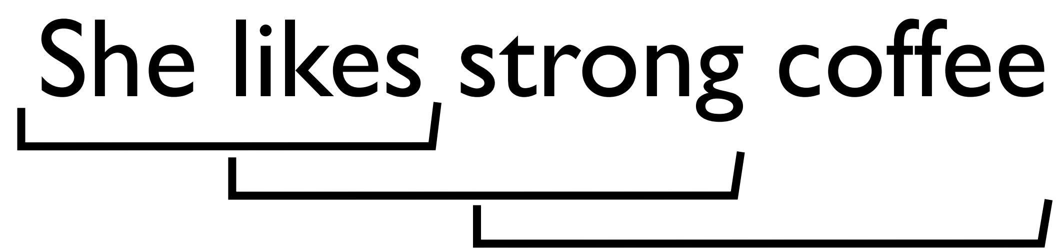
$k=2$ window length

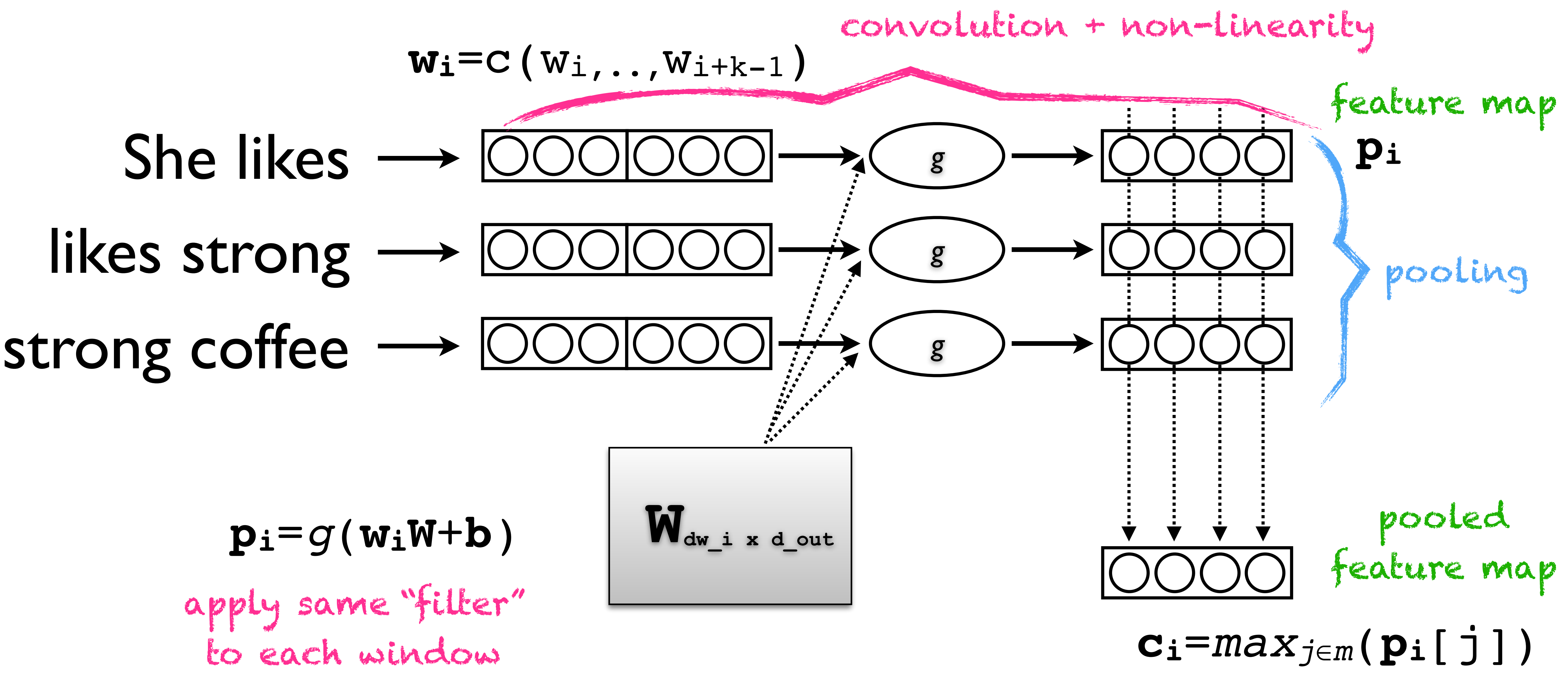
$d_{in}=3$ embedding dim

$d_{w\_i}=kd_{in}=6$ filter width

$d_{out}=4$ conv output dim

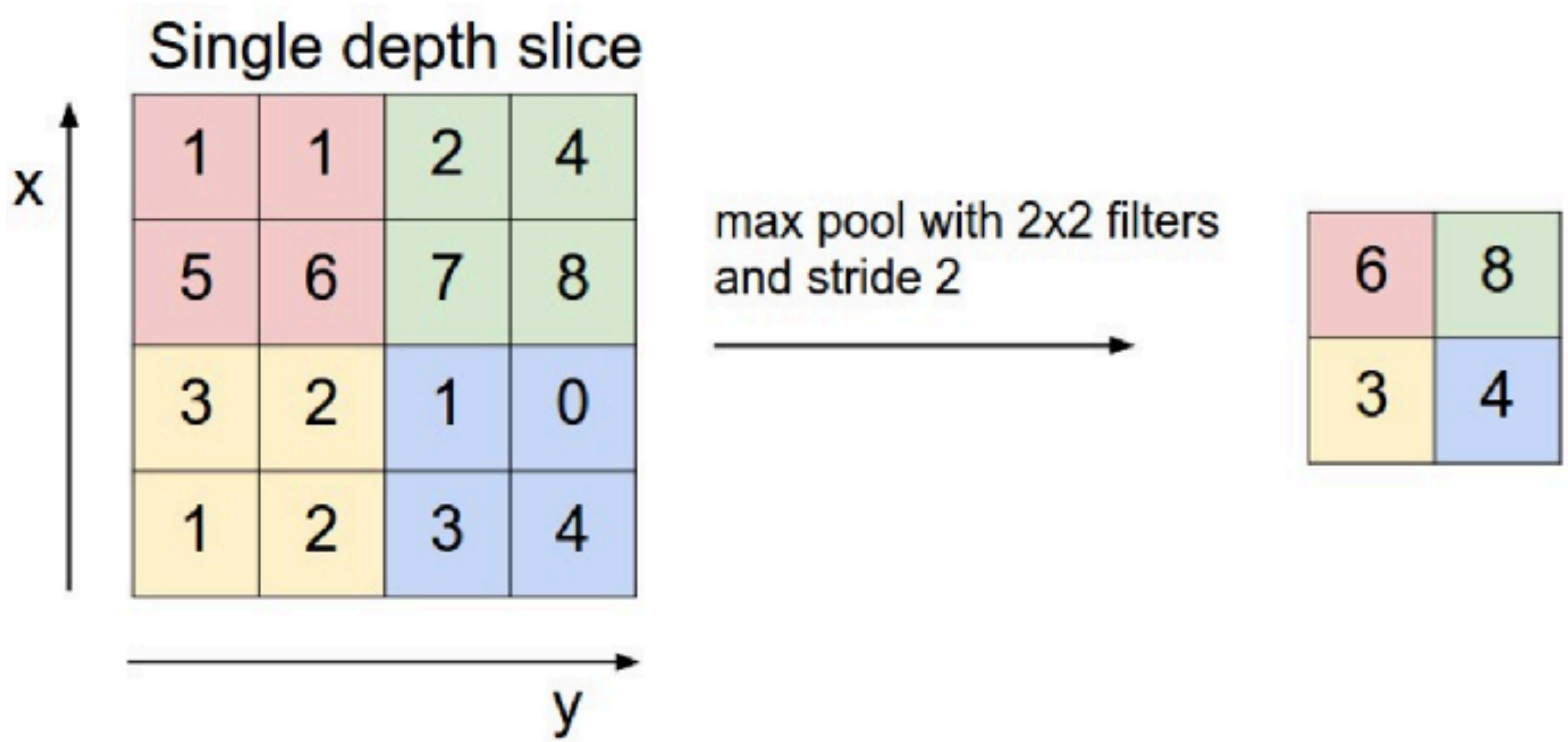
$n=4$ input length

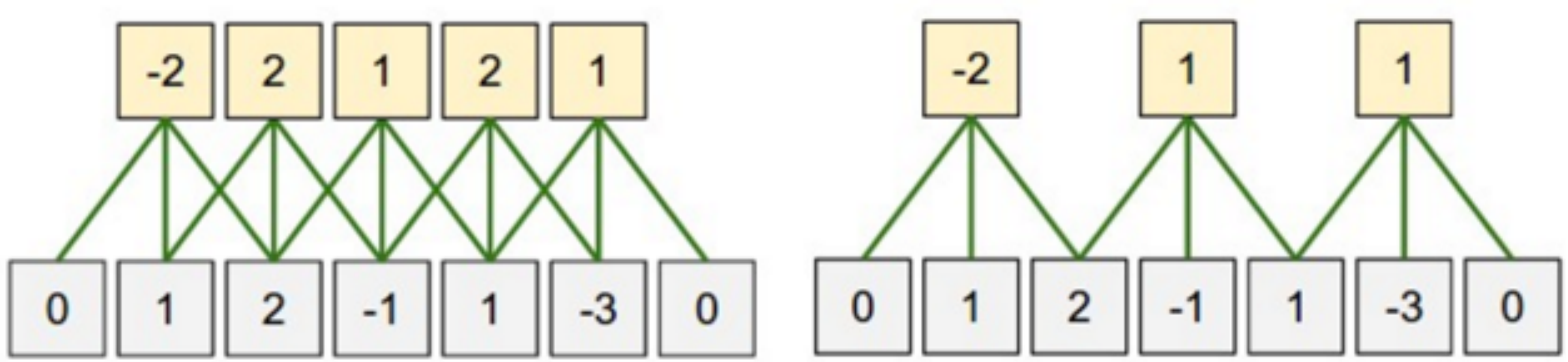




“soft” n-grams

Stride



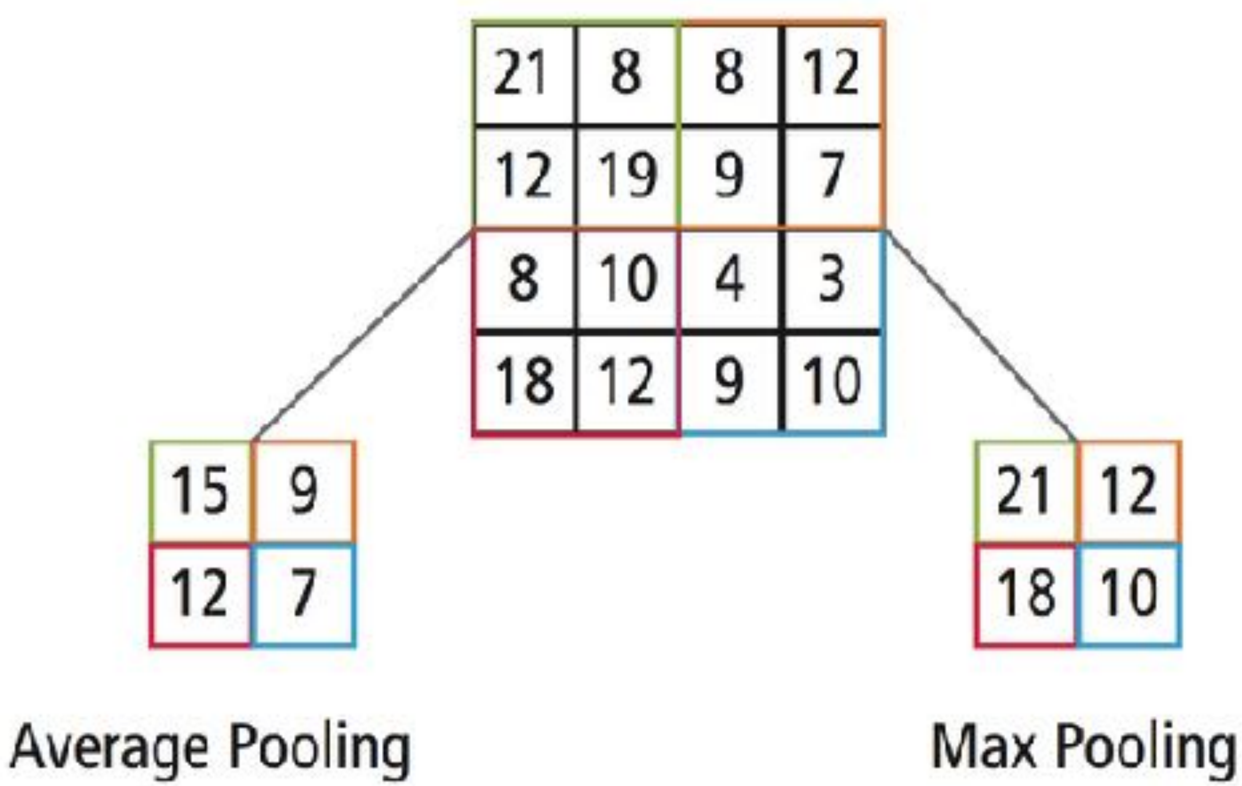


http://d3kbpzbmcynnmx.cloudfront.net/wp-content/uploads/2015/11/Screen-Shot-2015-11-05-at-2.18.38-PM.png

http://d3kbpzbmcynnmx.cloudfront.net/wp-content/uploads/2015/11/Screen-Shot-2015-11-05-at-10.18.08-AM.png

Types of pooling (1/3)

- **Max pooling:** "Did you see this feature anywhere in the range?" (most common) $c_i = max_{j \in m}(p_i[j])$
- **Average pooling:** "How prevalent is this feature over the entire range" $c_i = 1/m \sum_m p_i$



Types of pooling (2/3)

- **k-Max pooling**: “Did you see this feature up to k times?” (Kalchbrenner et al., 2014)
  - retain top k values in each dimension instead of only the best one, while preserving the order in which they appeare

```
1 2 3
9 6 5
2 3 1
7 8 1
3 4 1
```

1-max pooling

```
9 8 5
```

2-max pooling

```
9 6 3
7 8 5
```

~~9 8 5~~
~~7 6 3~~

Types of pooling (3/3)

- **Dynamic pooling**: “Are some parts more informative?” (Johnson & Zhang, 2015)
  - split $\mathbf{p}_i$’s into separate groups based on domain knowledge and apply max-pooling to each region/group
  - e.g. initial sentences more predictive for news topic classification (Johnson & Zhang, 2015)

$$\begin{array}{l}\left.\begin{matrix}1 & 2 & 3\\ 9 & 6 & 5\end{matrix}\right\}\ \text{pooling}\\ \left.\begin{matrix}2 & 3 & 1\\ 7 & 8 & 1\\ 3 & 4 & 1\end{matrix}\right\}\ \text{pooling}\end{array}$$

CNNs for Text Classification (Kim, 2014)

different "channels" for pre-trained & embeddings from scratch

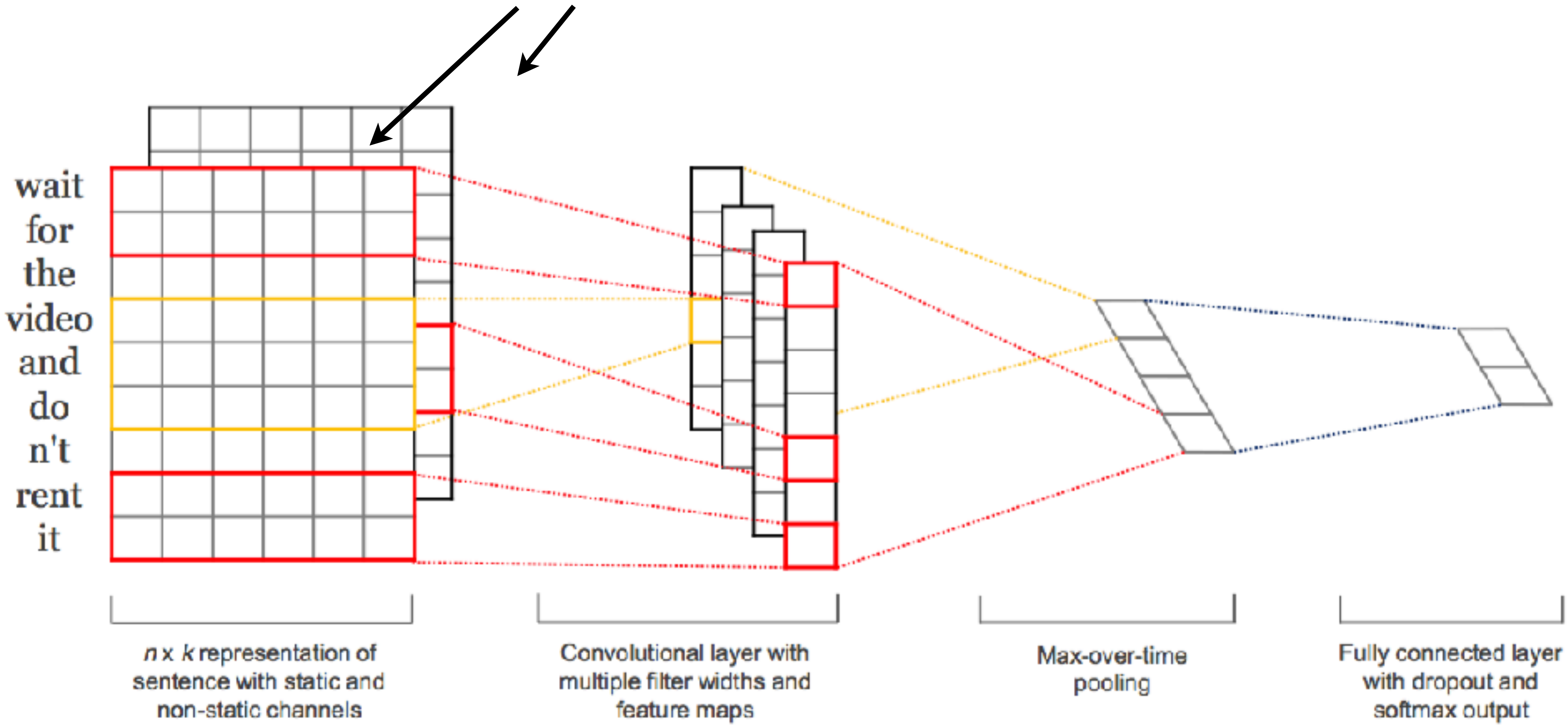


Figure 1: Model architecture with two channels for an example sentence.

CNNs - Interim summary

- **Main idea:** apply the same parametrized function over all n-grams in the sequence.
- This creates a series of m vectors, each representing a particular n-gram in the sequence
- The representation is sensitive to the identity and order of the words in the n-gram, but the same representation will be extracted for a n-gram regardless of its position in the sequence

Two advances in CNNs

Stacked convolutions

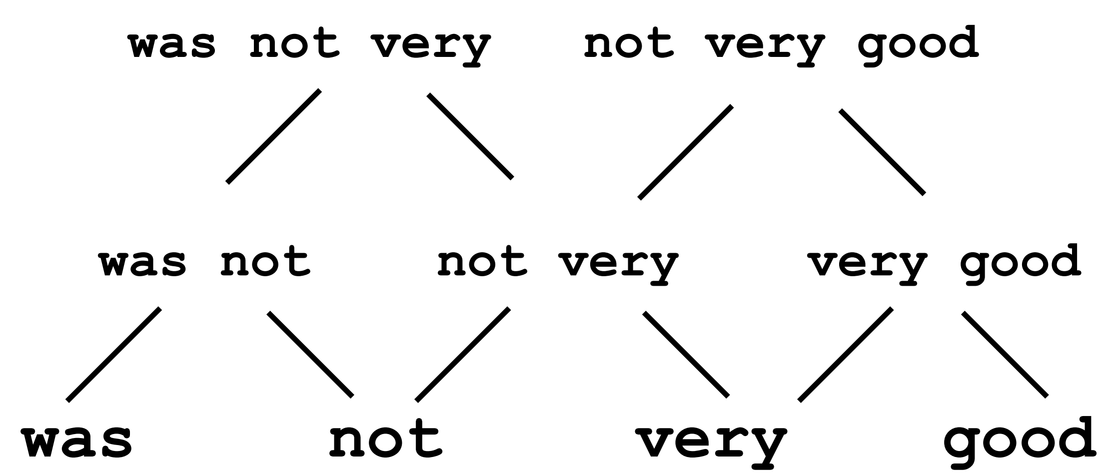
- **Hierarchical convolutions:** apply a sequence of ***r*** convolutions that feed into each other
- Resulting vectors capture increasingly larger windows ("receptive fields")

$$\mathbf{p^1_{1:m_1}} = CONV^{k}1_{U^1,b^1}(\mathbf{w_1} : \mathbf{n})$$

$$\mathbf{p^2_{1:m_2}} = CONV^{k}2_{U^2,b^2}(\mathbf{p_1} : \mathbf{m_1})$$

$$\ldots$$

$$\mathbf{p^r_{1:m_r}} = CONV^{k}r_{U^r,b^r}(\mathbf{p_{r-1}} : \mathbf{m_{r-1}})$$



Dilated convolutions

(Strubell et al., 2017; Kalchbrenner et al,. 2016; Yu and Koltun, 2016)

- Each layer in the hierarchy has a stride size of k-1
  - speed gains over RNNs

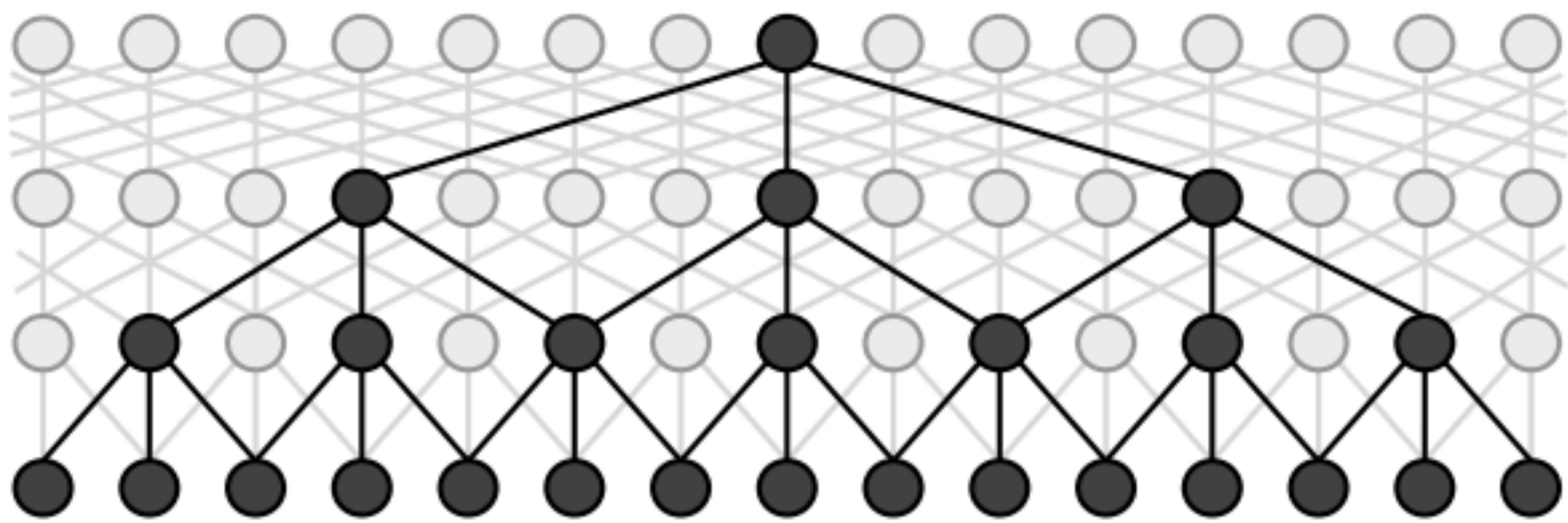
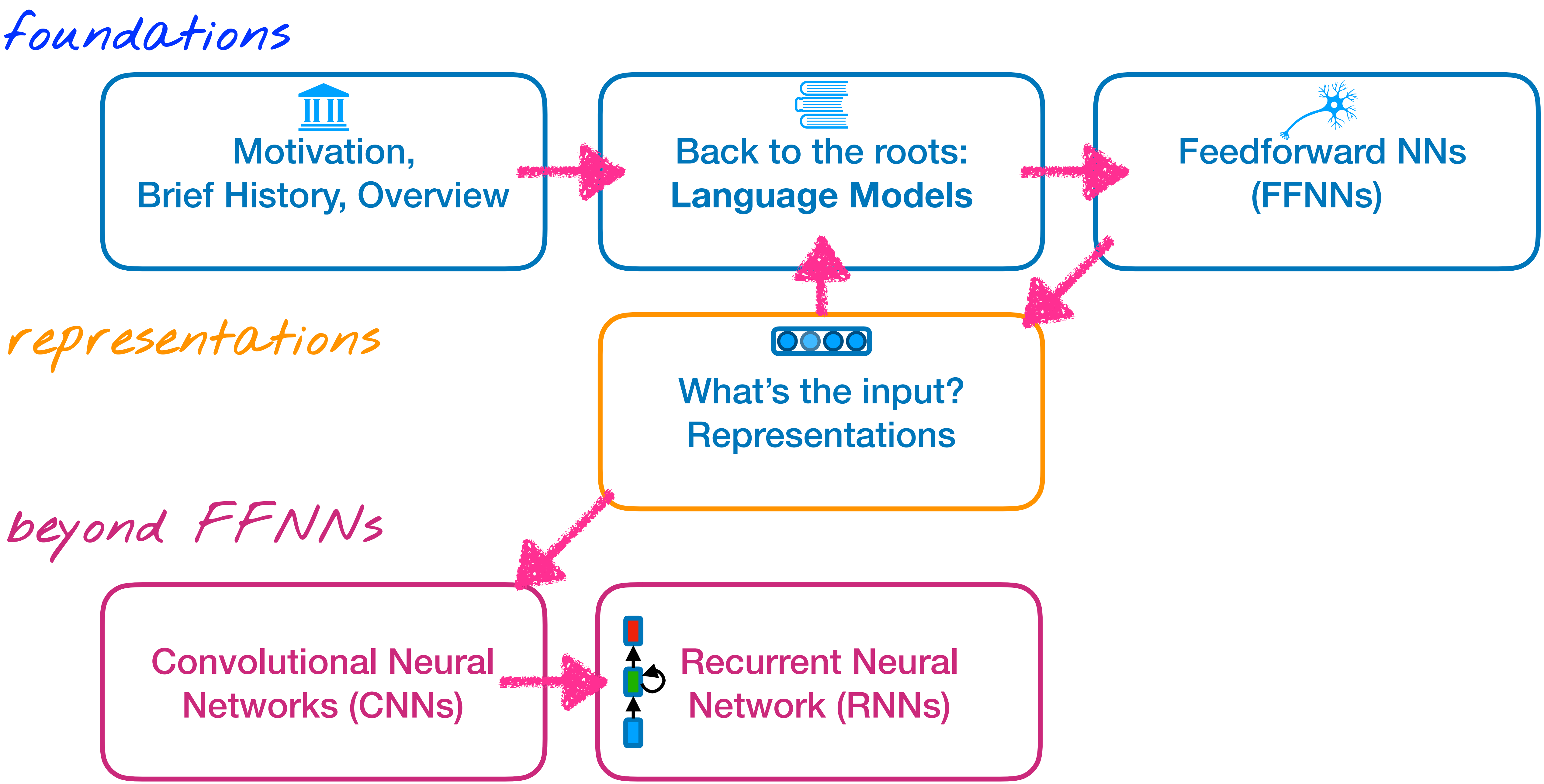


Figure 1: A dilated CNN block with maximum dilation width 4 and filter width 3. Neurons contributing to a single highlighted neuron in the last layer are also highlighted.

Overview


foundations
Motivation,
Brief History, Overview
Back to the roots:
Language Models
Feedforward NNs
(FFNNs)
representations
What’s the input?
Representations
beyond FFNNs
Convolutional Neural
Networks (CNNs)
Recurrent Neural
Network (RNNs)

BREAK

RNNs

Recurrent Neural Networks (RNNs)

Elman, 1990

- RNNs (and their variants) are **one of the most powerful and widespread architectures to date**
- From J.Schmidhuber's homepage:

speech recognition. They learn through gradient descent and / or evolution or both. Compare the RNN Book Preface. LSTM is getting popular: Google, Apple, Microsoft, Facebook, IBM, Baidu, and many other companies use LSTM RNNs to improve large vocabulary speech recognition, machine translation, language identification / time series prediction / text-to-speech synthesis, etc.

Recurrent Neural Networks (RNNs)

- Can handle **arbitrary length inputs** (just like CNNs or a FFNN with a CBOW input representation)
  - Unlike CBOW, they **model the order** in the sequence
  - Unlike vanilla CNNs, they **can deal with long-distance dependencies** (especially the gated RNN variants)
- Do **not need to make the Markov** assumption
  - Opens up for a family of models:
    Conditioned generation models

Key Idea:

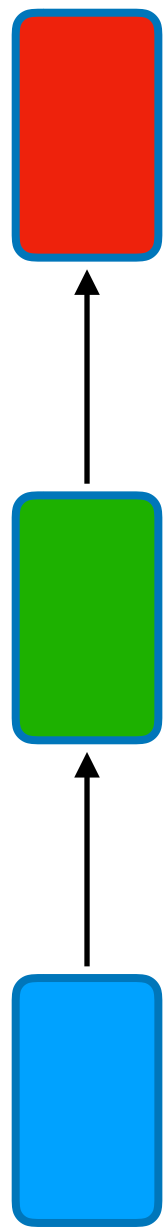
Recurrent Neural Networks (RNNs)

RNNs have an internal "memory" (state) which is updated as the sequence is read

"vanilla" Neural Network

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

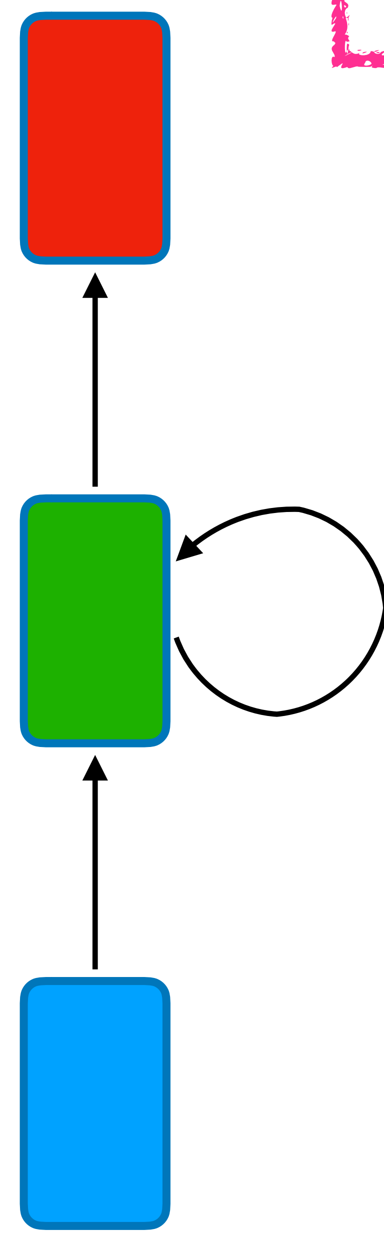
$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{b})$$



RNN

$$\hat{\mathbf{y}}_{\mathbf{t}} = \mathbf{W}\mathbf{h}_{\mathbf{t}} + \mathbf{b}$$

$$\mathbf{h} = g(\mathbf{V}\mathbf{x}_{\mathbf{t}} + \mathbf{U}\mathbf{h}_{\mathbf{t-1}} + \mathbf{c})$$

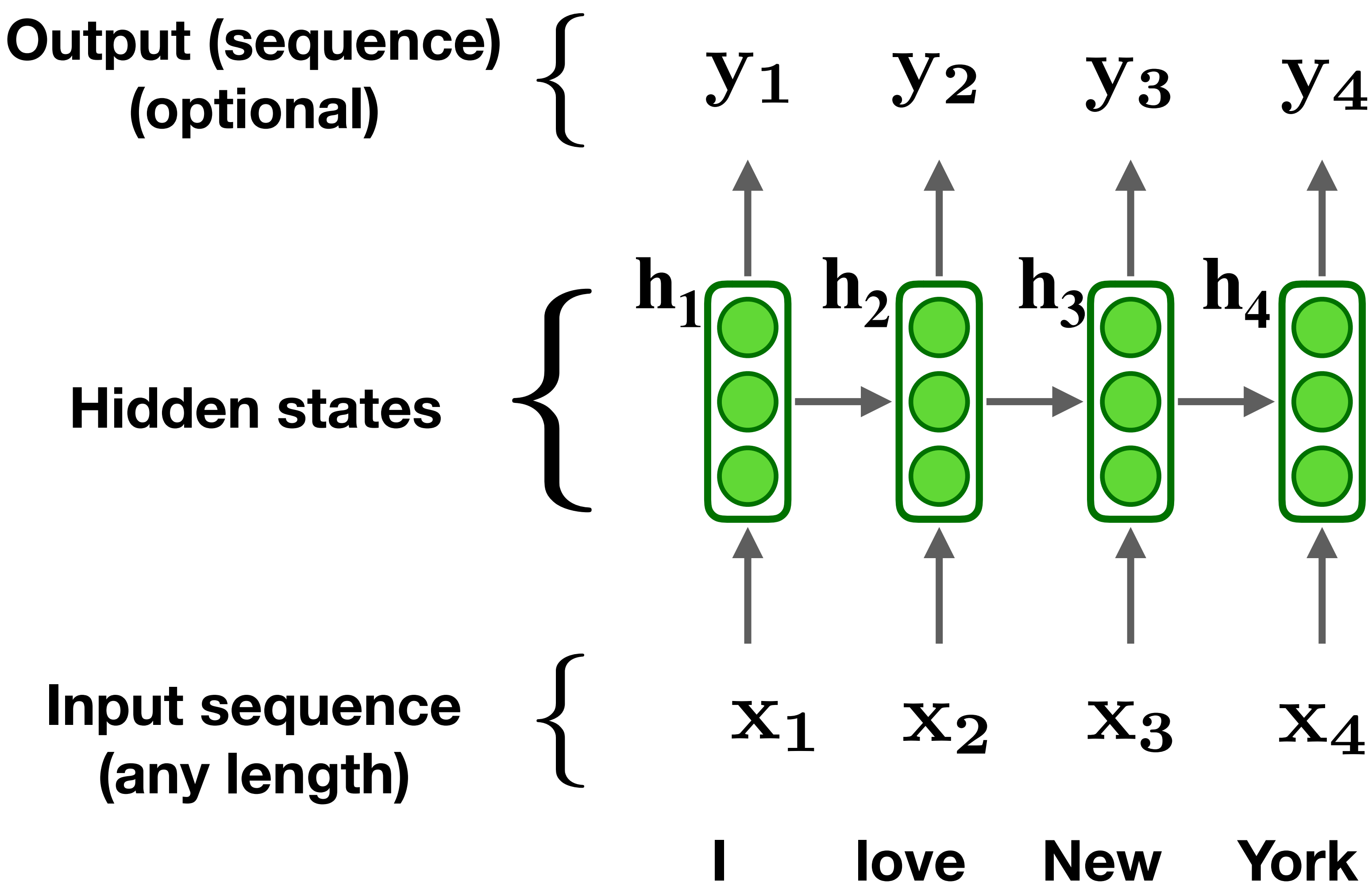


recurrence /
memory / state

Recurrent Neural Networks

A family of recurrent NN architectures

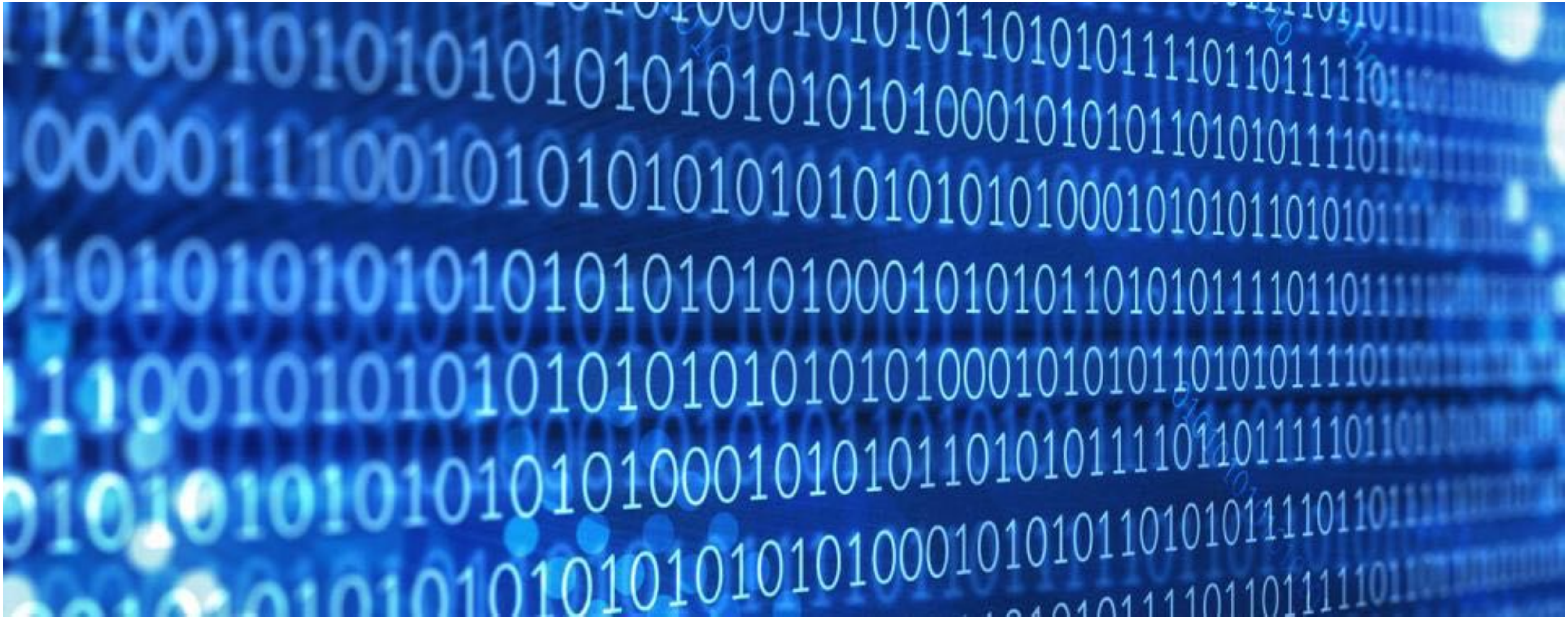
$$\mathbf{h} = g(\mathbf{V}\mathbf{x_t} + \mathbf{U}\mathbf{h_{t-1}} + \mathbf{c})$$



Core idea:
Parameter
Sharing over
Time
(=apply **U**
repeatedly)

Before we dig into details - The RNN abstraction

Count the number of 1s



Example from Cho (2015)

Count the number of 1s

```python
def add1(el,s):
    if el==1: return s+1
    else: return s
```

Two important components:

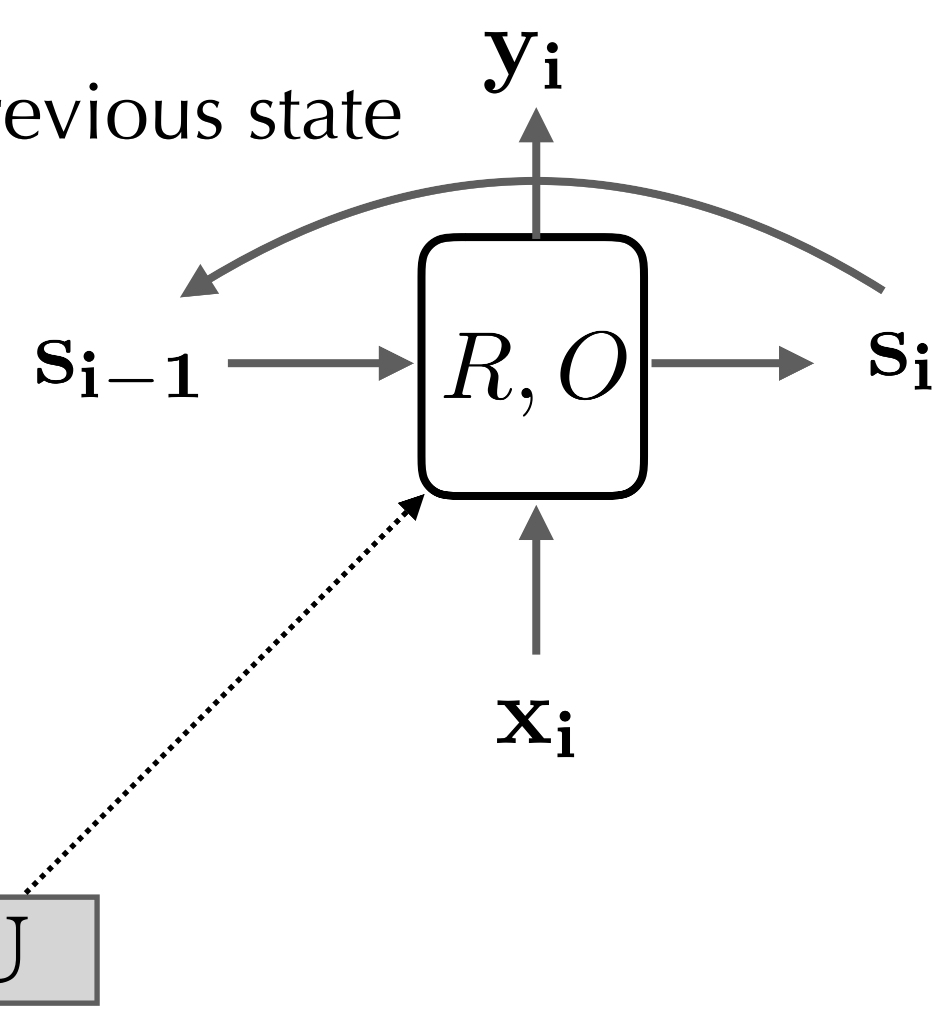
- memory ***s***
- function *add1* is applied to each symbol in the input *one at a time* to update the memory

```python
v=[0,1,0,0,1,1]
```

```python
s=0
for el in v:
    s=add1(el,s)
print("count(1):", s)
```

The RNN abstraction

- Input sequence of vectors: $\mathbf{x_{1:n}}$
- Start state: $\mathbf{s_0}$
- $RNN(\mathbf{s_0}, \mathbf{x_{1:n}})$ consists of two functions:
  - function R consumes input and previous state
  - function O maps states to outputs



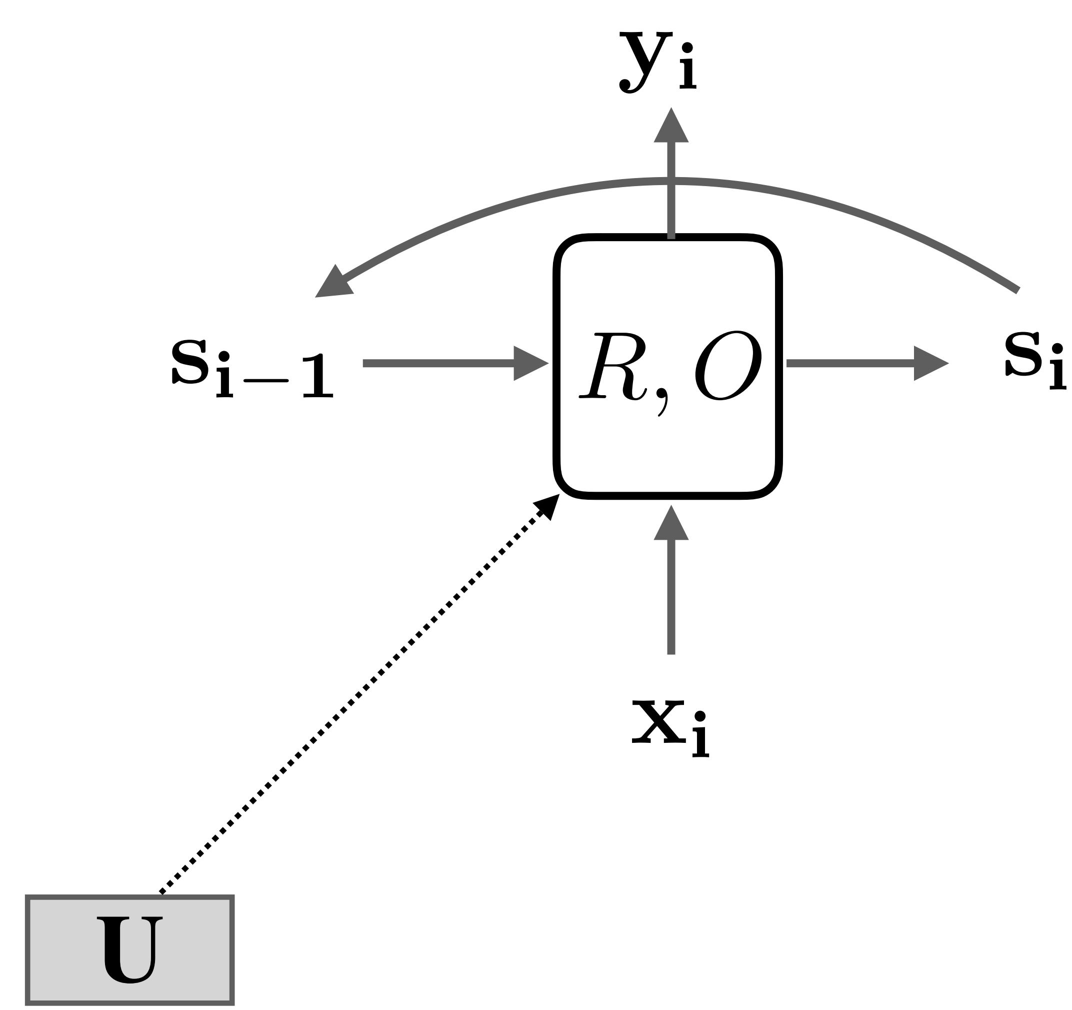
(Graphical illustration - recursive - from Yoav Goldberg's primer, 2015)

The RNN abstraction - More formally

$$RNN(\mathbf{s_0}, \mathbf{x_{1:n}}) = \mathbf{s_{1:n}},\ \mathbf{y_{1:n}}$$

$$\mathbf{s_i} = R(\mathbf{s_{i-1}}, \mathbf{x_i})$$

$$\mathbf{y_i} = O(\mathbf{s_i})$$



(Graphical illustration - recursive - from Yoav Goldberg's primer, 2015)

RNN: Unrolled over time

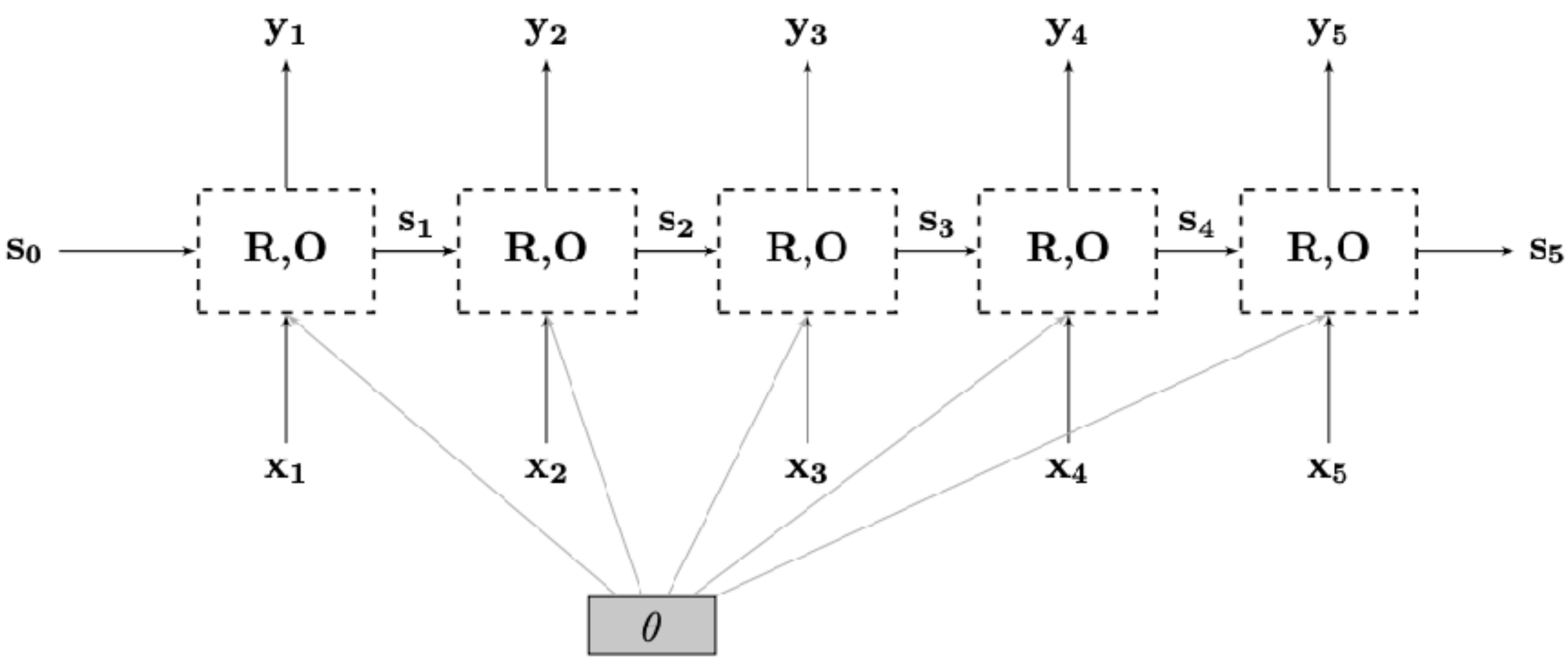


Figure 6: Graphical representation of an RNN (unrolled).

(Graphical illustration - recursive - from Yoav Goldberg's primer, 2015)

Expansion at time step 4

$$
\begin{aligned}
\mathbf{s_4} &= R(\mathbf{s_3}, \mathbf{x_4}) \\
&= R(\overbrace{R(\mathbf{s_2}, \mathbf{x_3})}^{\mathbf{s_3}}, \mathbf{x_4}) \\
&= R(R(\overbrace{R(\mathbf{s_1}, \mathbf{x_2})}^{\mathbf{s_2}}, \mathbf{x_3}), \mathbf{x_4}) \\
&= R(R(R(\overbrace{R(\mathbf{s_0}, \mathbf{x_1})}^{\mathbf{s_1}}, \mathbf{x_2}), \mathbf{x_3}), \mathbf{x_4})
\end{aligned}
$$

(Graphical illustration - recursive - from Yoav Goldberg's primer, 2015)

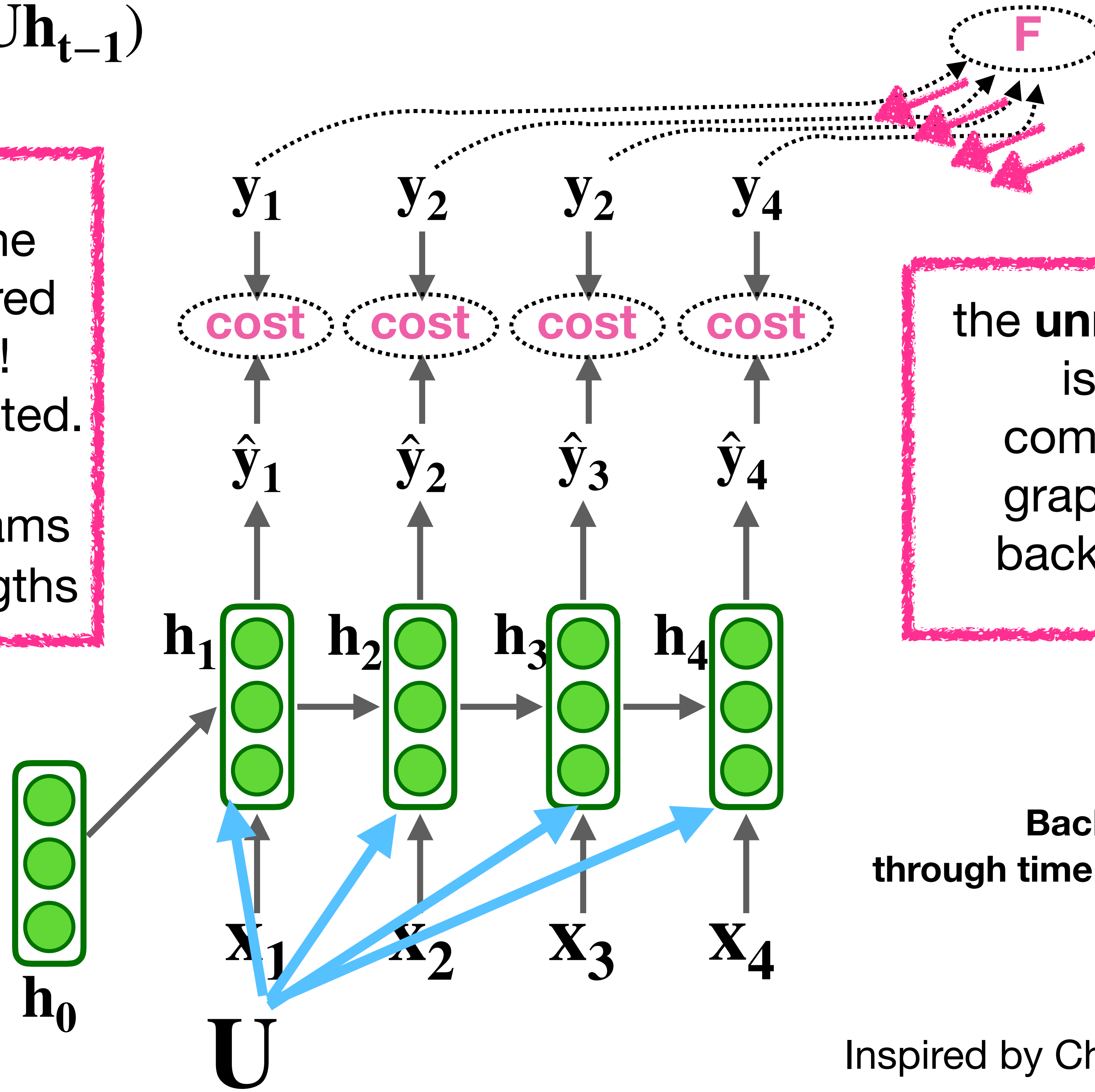
Training a RNN, parameter tying

$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$

$$\mathbf{h} = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1})$$

Parameter **tying**: the parameters are shared across time steps! Derivatives accumulated.

Pros: - reduce #params
- model arbitrary lengths



the **unrolled** graph is a DAG computational graph, we can backprop back

Backpropagation through time (BPTT, Werbos, 1990).

Inspired by Chris Dyer's lecture

A closer look: inside an RNN

- We process a sequence $\mathbf{x}$ by applying a recurrence formula at every time step i:

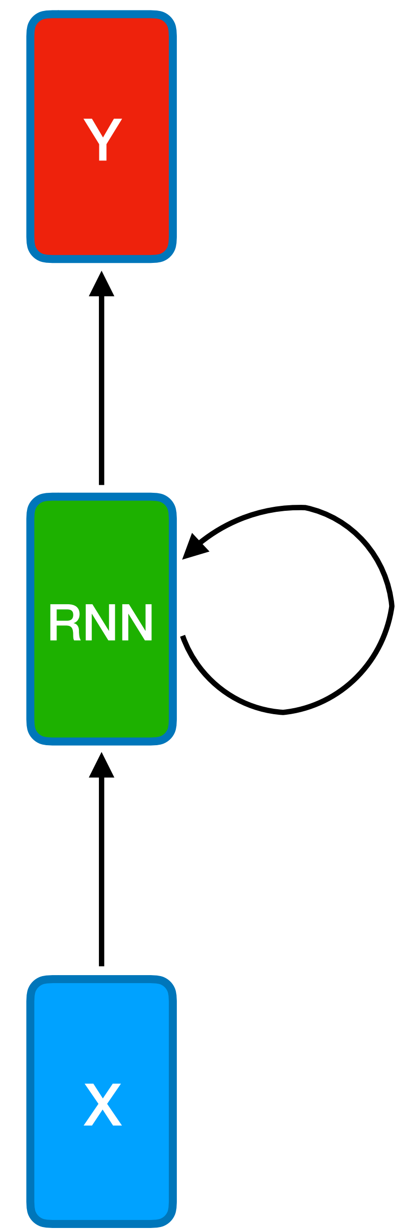
new state

$$\boxed{\mathbf{h}_i} = f_\theta(\boxed{\mathbf{h}_{i-1}}, \boxed{\mathbf{x}_i})$$

function
parametrized by θ

previous state

input



Vanilla RNN

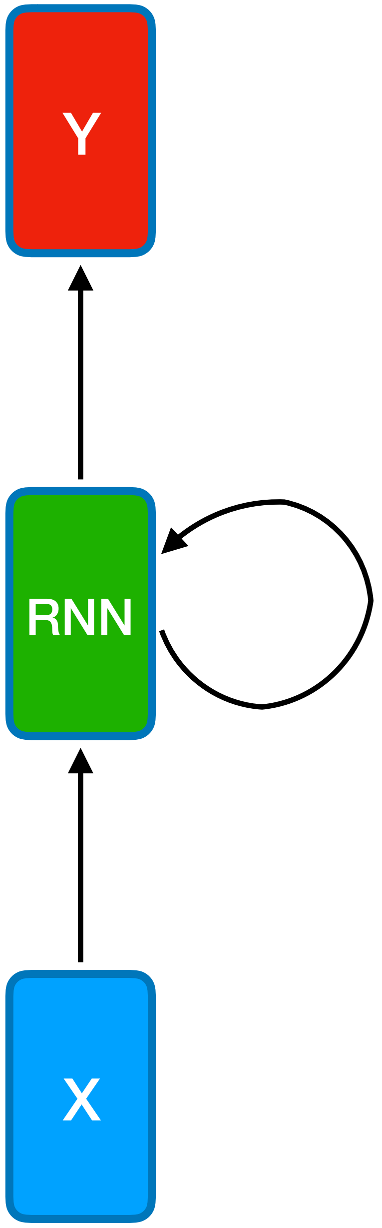
- **Simple vanilla RNN** (Elman, 1990)

$$\mathbf{h}_i = f_\theta(\mathbf{h}_{i-1}, \mathbf{x}_i)$$

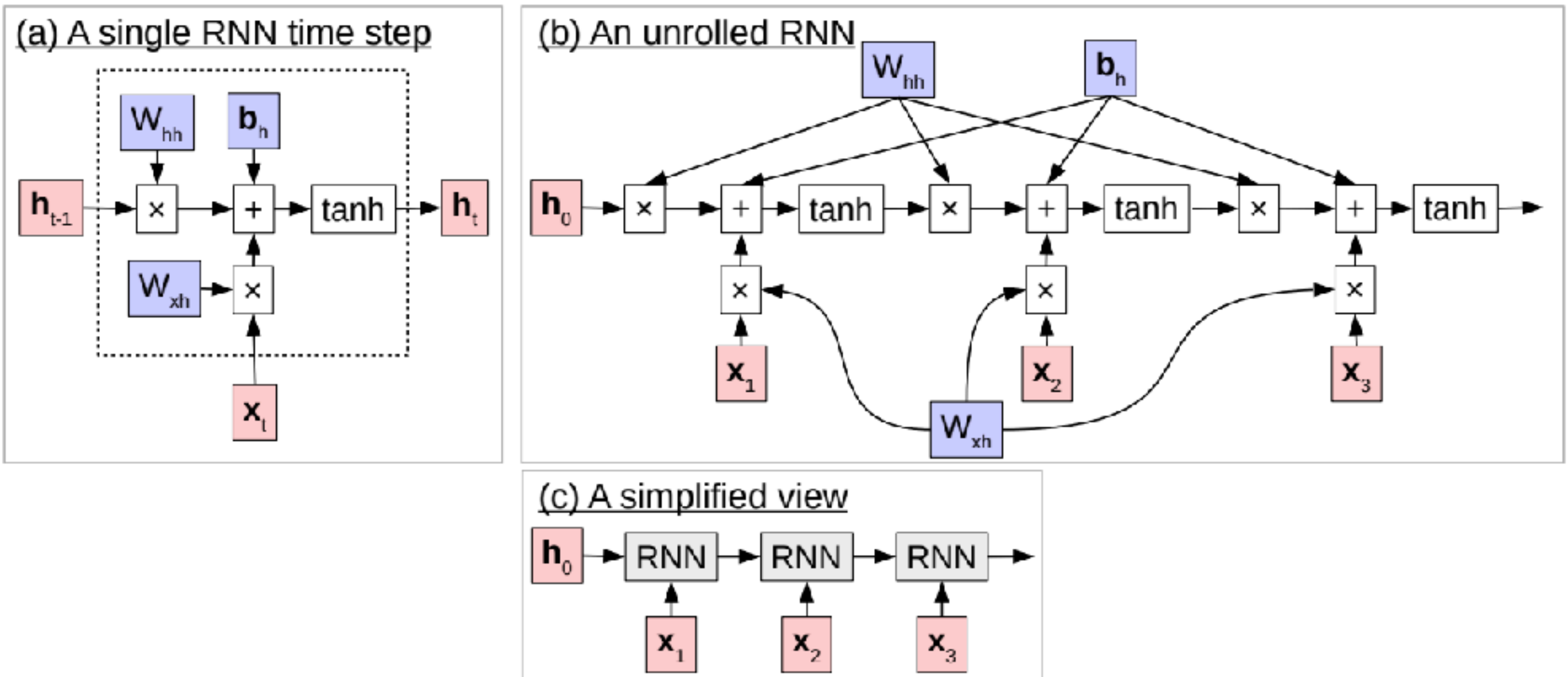
RNN: example instantiation of function parametrized by θ

$$\hat{\mathbf{y}}_\mathbf{t} = \mathbf{W}\mathbf{h}_\mathbf{t} + \mathbf{b}$$

$$\mathbf{h} = g(\mathbf{V}\mathbf{x}_\mathbf{t} + \mathbf{U}\mathbf{h}_{\mathbf{t-1}})$$



Summary of Views:



Illustrations by G.Neubig, 2018

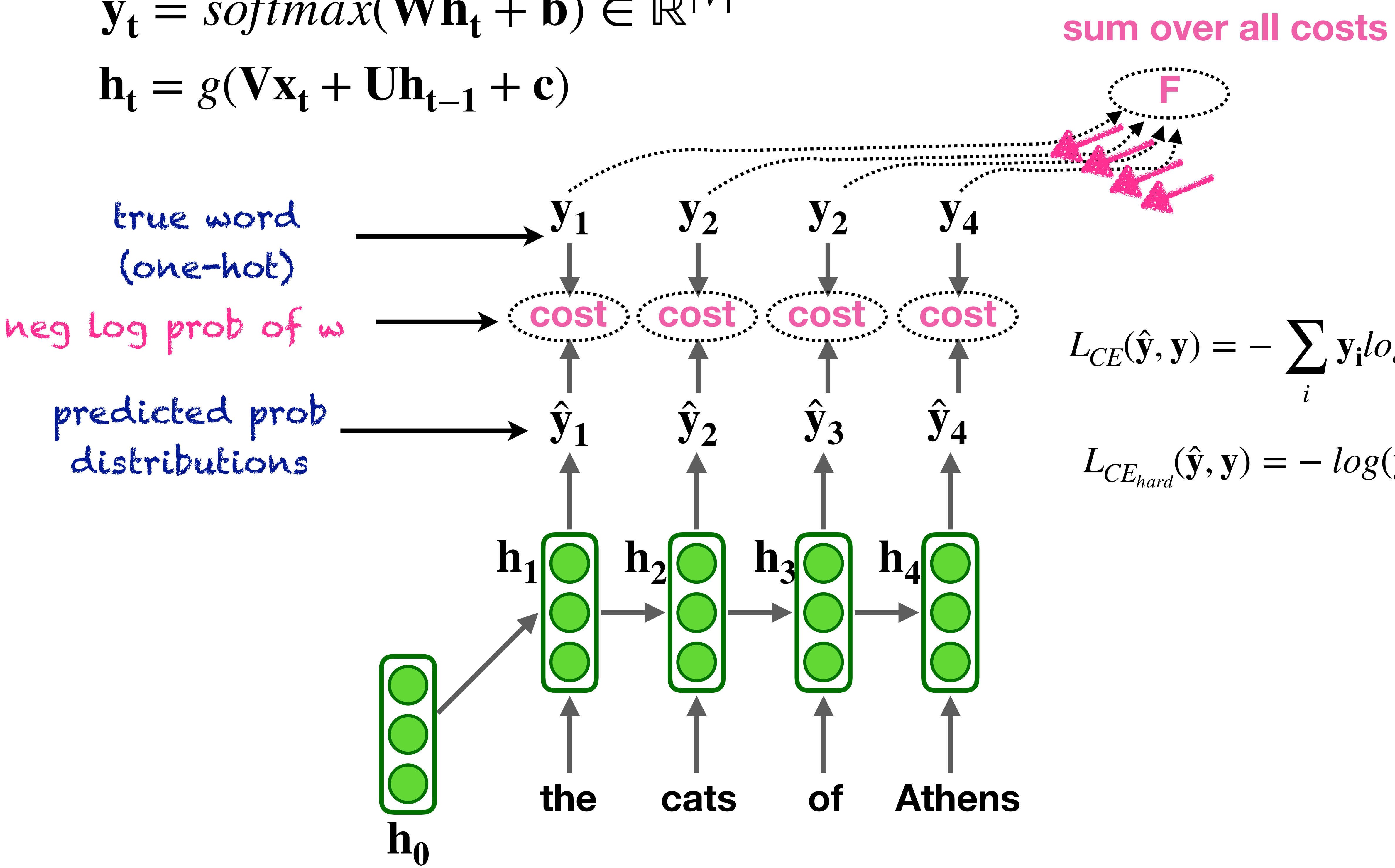
http://www.phontron.com/class/mtandseq2seq2018/assets/slides/mt-fall2018.chapter6.pdf

RNN Language Model

Training a RNN LM

$$\hat{\mathbf{y}}_\mathbf{t} = softmax(\mathbf{W}\mathbf{h}_\mathbf{t} + \mathbf{b}) \in \mathbb{R}^{|V|}$$

$$\mathbf{h}_\mathbf{t} = g(\mathbf{V}\mathbf{x}_\mathbf{t} + \mathbf{U}\mathbf{h}_{\mathbf{t-1}} + \mathbf{c})$$



$$L_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_i \mathbf{y}_\mathbf{i} log(\hat{\mathbf{y}}_\mathbf{i})$$

$$L_{CE_{hard}}(\hat{\mathbf{y}}, \mathbf{y}) = -\,log(\hat{\mathbf{y}}_\mathbf{i})$$

See Yoav Goldberg's book Sec.2.7.1

What about these issues?

- Can it handle similar words?
  - she *bought* a bicycle
  - she *purchased* a bicycle

- Long-distance dependencies?
  - for *programming* she yesterday purchased her own brand new *laptop*
  - for *running* she yesterday purchased her brand new *sportswatch*

However, in practice the vanilla RNN has some trouble.. more soon

Generate with a RNN LM - some fun!

character-level RNN-LM

Generating Baby Names

Lets try one more for fun. Lets feed the RNN a large text file that contains 8000 baby names listed out, one per line (names obtained from here). We can feed this to the RNN and then generate new names! Here are some example names, only showing the ones that do not occur in the training data (90% don't):

Rudi Levette Berice Lussa Hany Mareanne Chrestina Carissy Marylen Hammine Janye Marlise Jacacrie Hendred Romand Charienna Nenotto Ette Dorane Wallen Marly Darine Salina Elvyn Ersia Maralena Minoria Ellia Charmin Antley Nerille Chelon Walmor Evena Jeryly Stachon Charisa Allisa Anatha Cathanie Geetra Alexie Jerin Cassen Herbett Cossie Velen Daurenge Robester Shermond Terisa Licia Roselen Ferine Jayn Lusine Charyanne Sales Sanny Resa Wallon Martine Merus Jelen Candica Wallin Tel Rachene Tarine Ozila Ketia Shanne Arnande Karella Roselina Alessia Chasty Deland Berther Geamar Jackein Mellisand Sagdy Nenc Lessie Rasemy Guen Gavi Milea Anneda Margoris Janin Rodelin Zeanna Elyne Janah Ferzina Susta Pey Castina

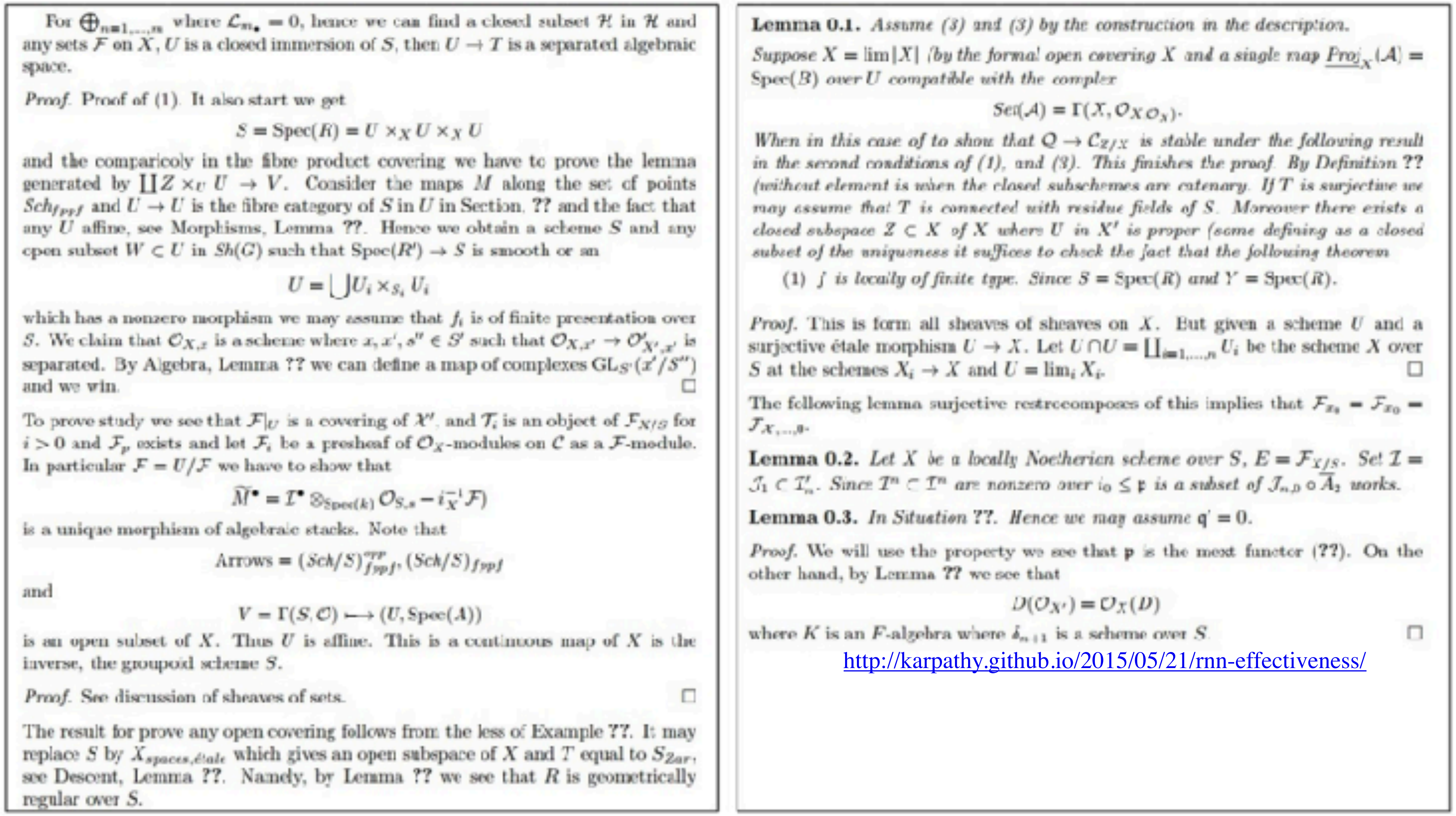
You can see many more here. Some of my favorites include "Baby" (haha), "Killie", "Char", "R", "More", "Mars", "Hi", "Saddie", "With" and "Ahbort". Well that was fun. Of course, you can imagine this being quite useful inspiration when writing a novel, or naming a new startup :)

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Generate with a RNN LM - some fun!

Algebraic Geometry (Latex)

The results above suggest that the model is actually quite good at learning complex syntactic structures. Impressed by these results, my labmate (Justin Johnson) and I decided to push even further into structured territories and got a hold of this book on algebraic stacks/geometry. We downloaded the raw Latex source file (a 16MB file) and trained a multilayer LSTM. Amazingly, the resulting sampled Latex *almost* compiles. We had to step in and fix a few issues manually but then you get plausible looking math, it's quite astonishing:



For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the comparicoly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ??. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longrightarrow (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example ??. It may replace S by $X_{spaces,\acute{e}tale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma ??. Namely, by Lemma ?? we see that R is geometrically regular over S.

Lemma 0.1. *Assume (3) and (3) by the construction in the description.*

Suppose $X = \lim |X|$ (by the formal open covering X and a single map $\underline{Proj}_X(\mathcal{A}) = \mathrm{Spec}(B)$ over U compatible with the complex

$$Set(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If T is surjective we may assume that T is connected with residue fields of S. Moreover there exists a closed subspace $Z \subset X$ of X where U in X' is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem

(1) *f is locally of finite type. Since $S = \mathrm{Spec}(R)$ and $Y = \mathrm{Spec}(R)$.*

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\ldots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{X,\ldots,0}$.

Lemma 0.2. *Let X be a locally Noetherian scheme over S, $E = \mathcal{F}_{X/S}$. Set $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. Since $\mathcal{I}^n \subset \mathcal{I}^n$ are nonzero over $i_0 \leq \mathfrak{p}$ is a subset of $\mathcal{J}_{n,0} \circ \overline{A}_2$ works.*

Lemma 0.3. *In Situation ??. Hence we may assume $\mathfrak{q}' = 0$.*

Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F-algebra where δ_{n+1} is a scheme over S. □

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Sampled (fake) algebraic geometry. Here's the actual pdf.

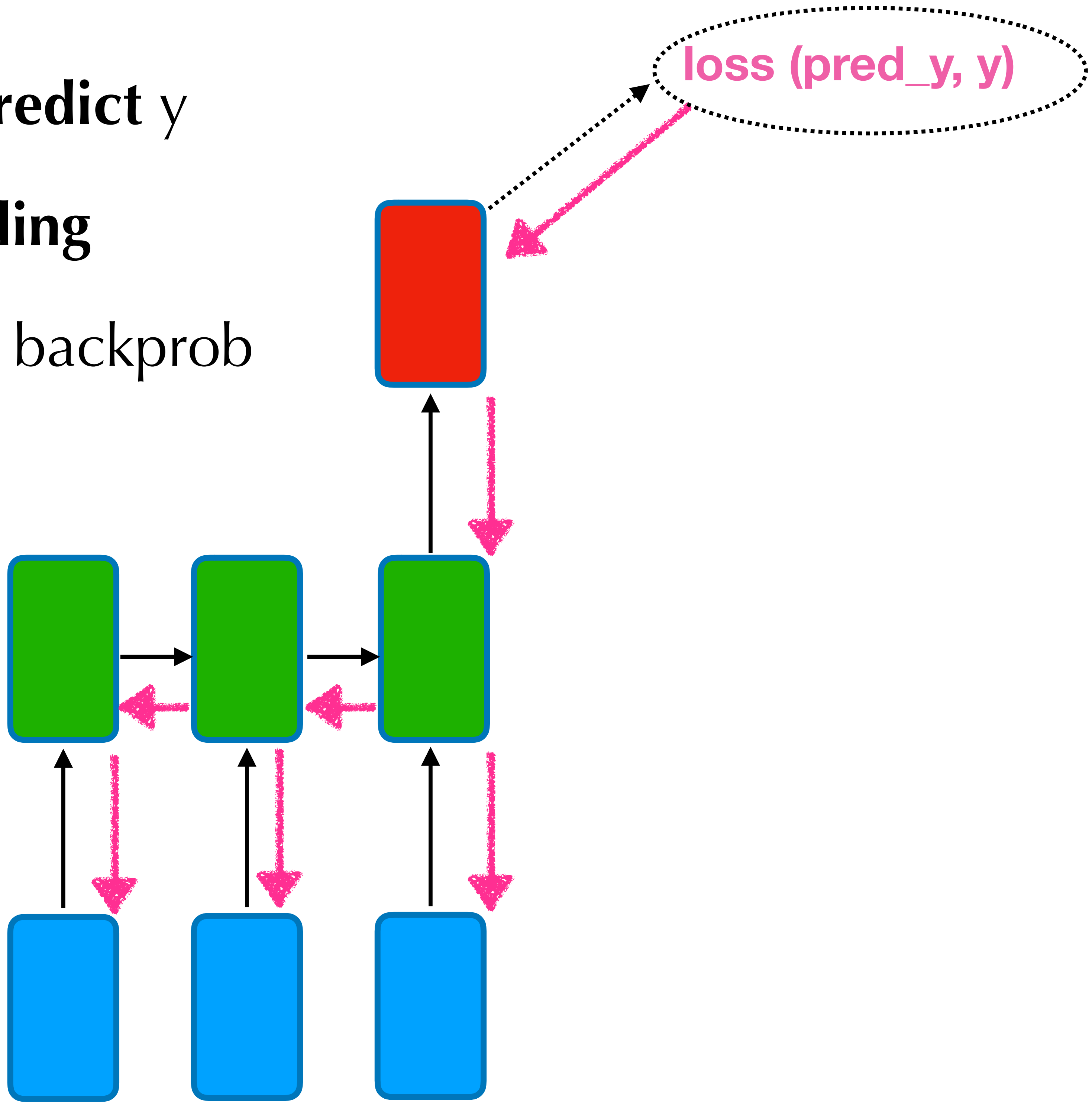
RNNs - Interim summary

- **LM:** a model that predicts the next word
- **RNN:** a family of neural networks
  - to model sequential input of any length
  - apply the same parameters on each time step
  - can optionally produce output at each time step
- RNN's are great as LMs. But they can be used for much more!

Common Usage Patterns

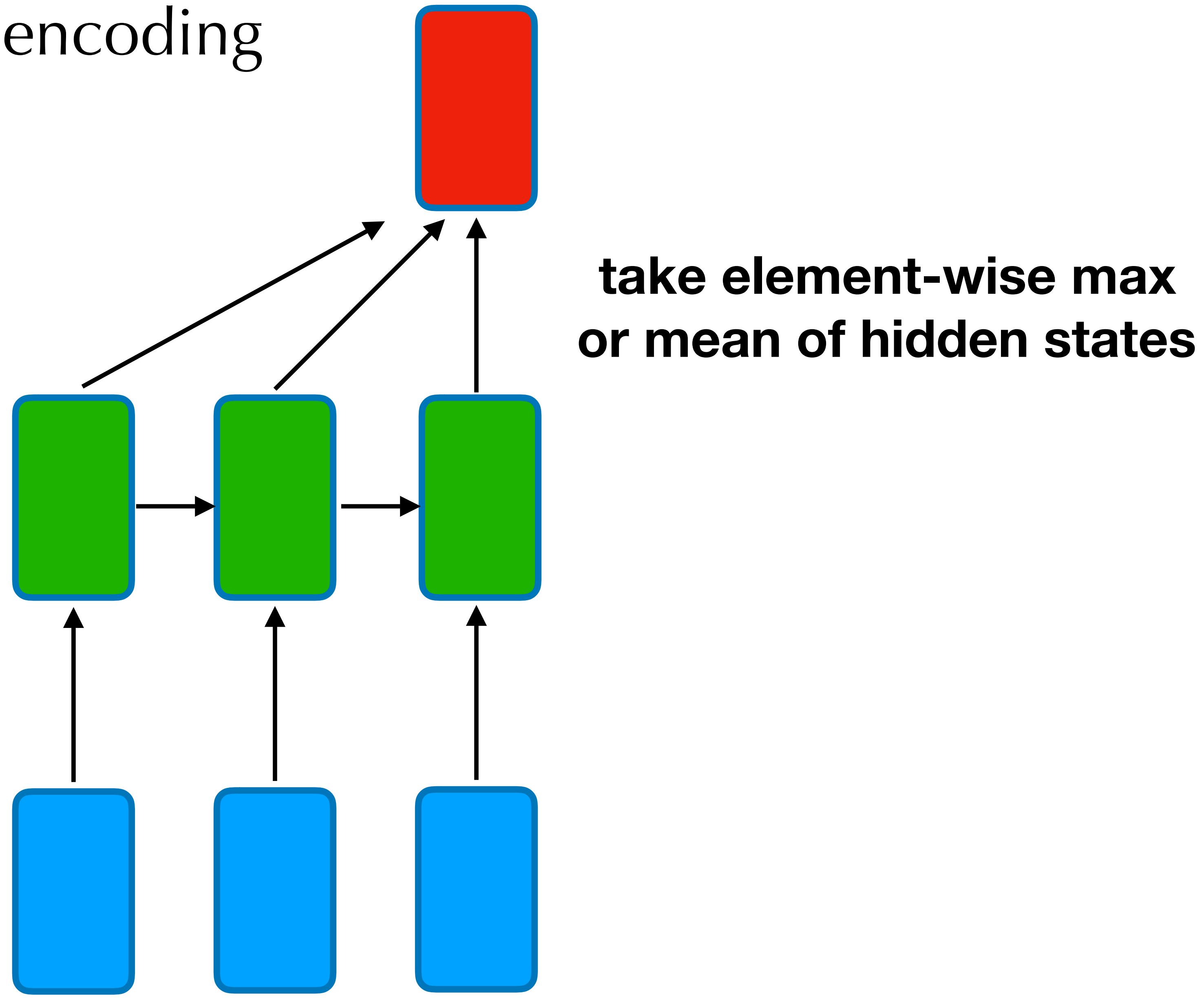
Example: An RNN as acceptor

- Use **last state** to **predict** y
  - **sentence encoding**
- Calculate loss and backprob



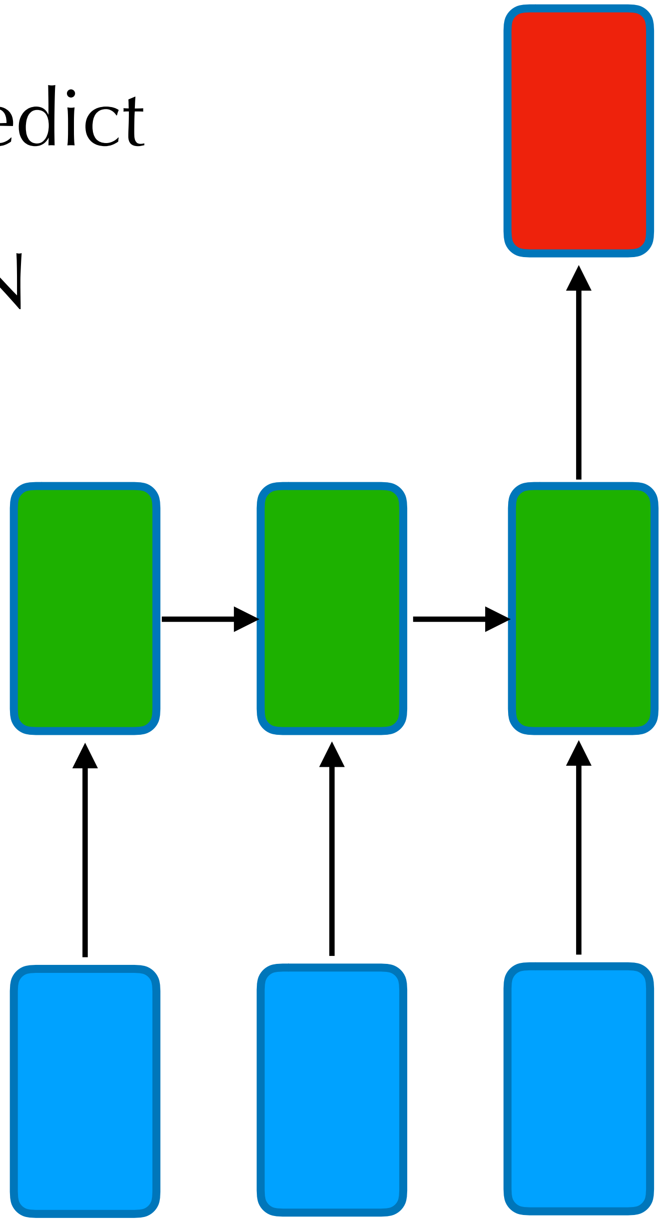
Example: An RNN as acceptor

- Use **average of states** to **predict** y
  - other sentence encoding



Example: An RNN as encoder

- Use **last state** as encoding of the information in the sequence; use as “feature” in other NN
    - encode, not predict
- E.g. character RNN



RNN as Transducer

- predict an output for each time step t
- E.g. Tagging (POS, NER)

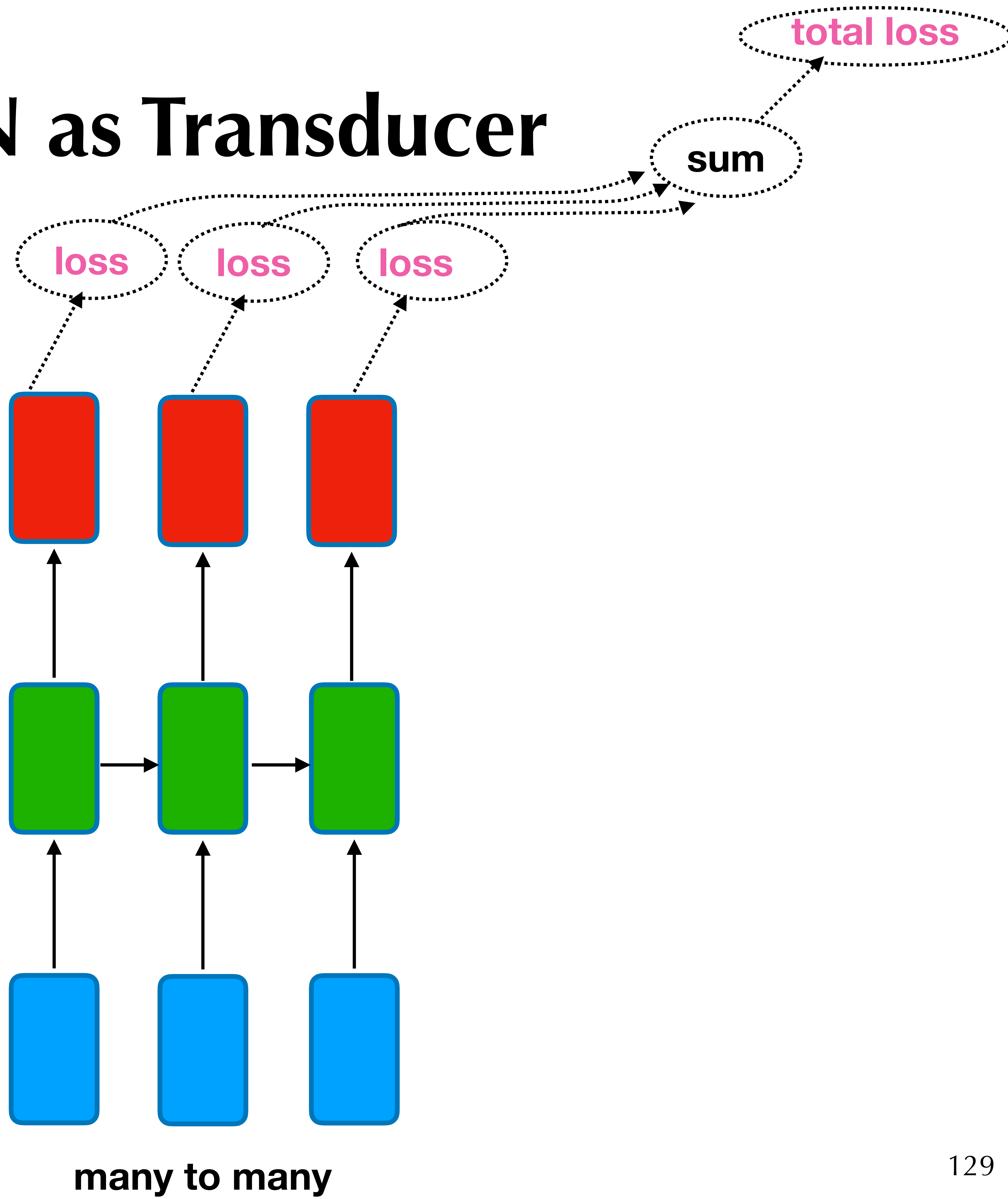
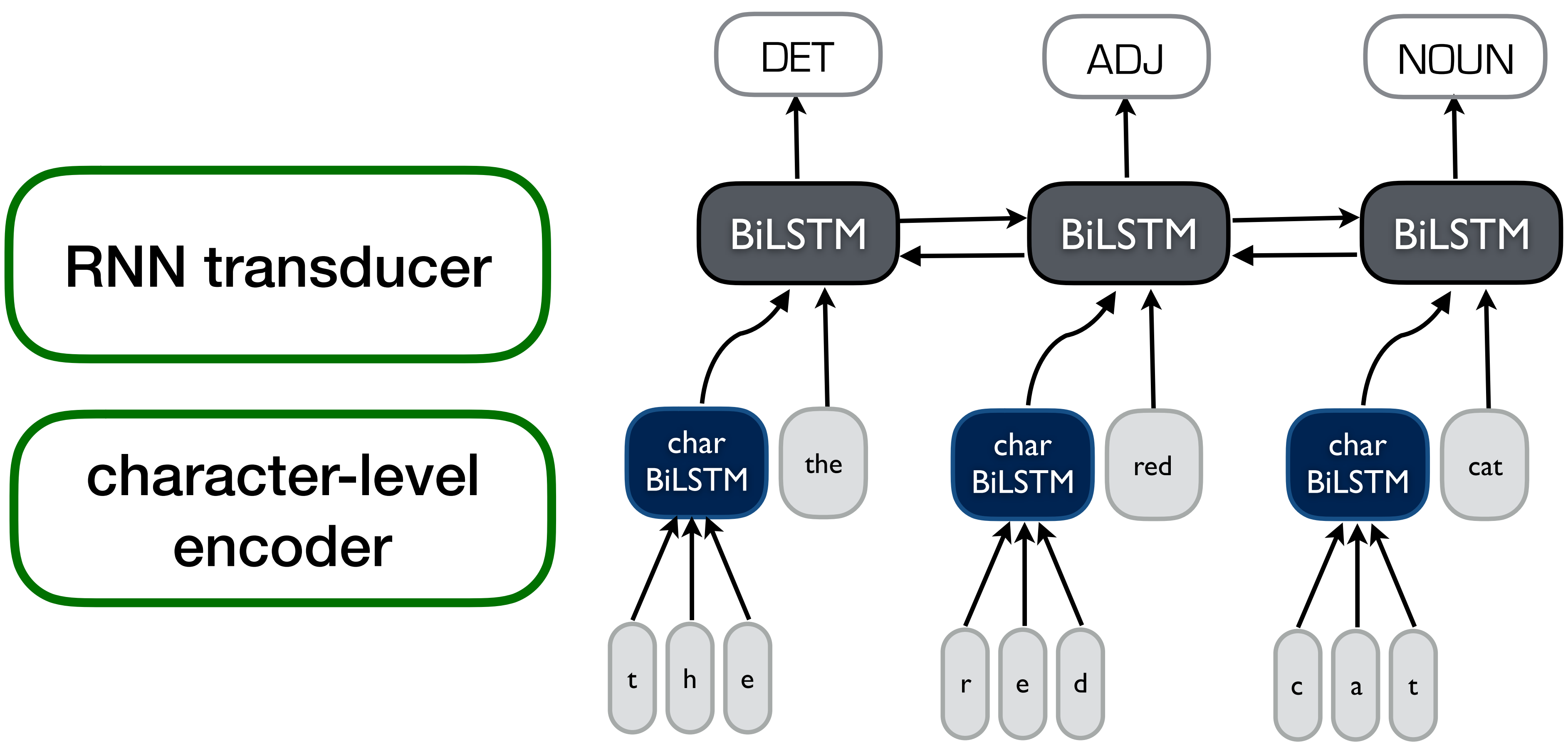


Illustration adapted from Karpathy

Combining them - a hierarchical RNN: Example for POS

- Use **RNN transducer** and lower-level **RNN encoder for characters** (more in a second)



(Plank et al., 2016)

RNN as generator

- Conditional generation
- E.g. image caption generation, speech synthesis

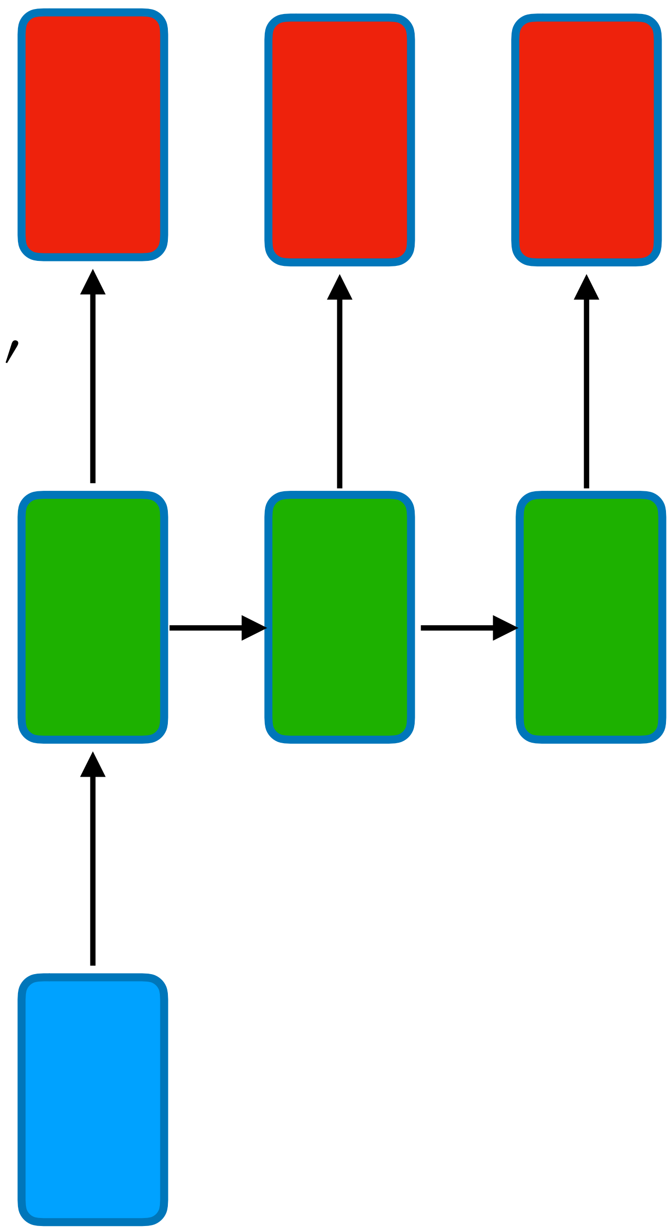


Illustration adapted from Karpathy

RNN encoder-decoder (seq2seq)

- Both input and output are a sequence

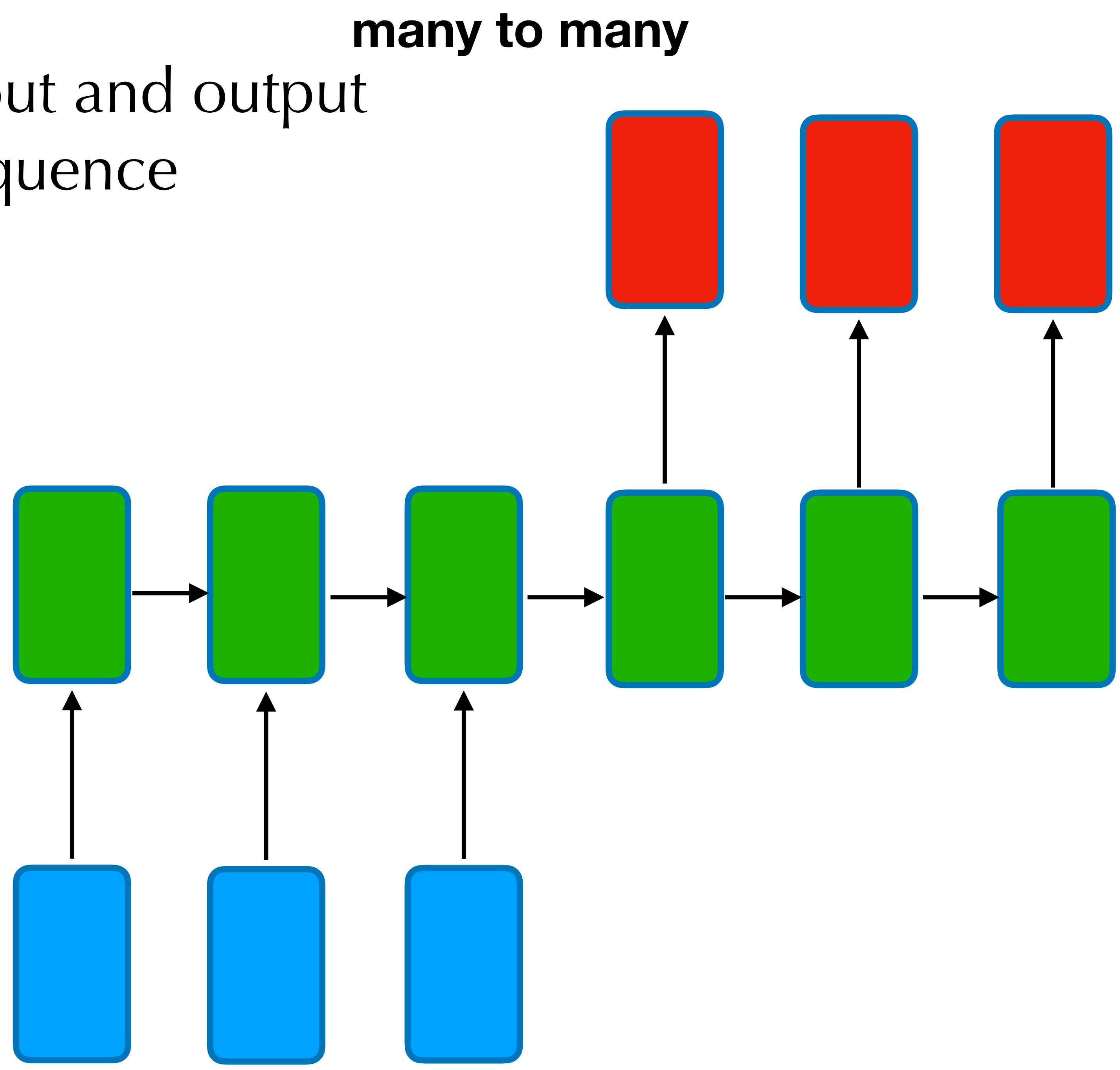


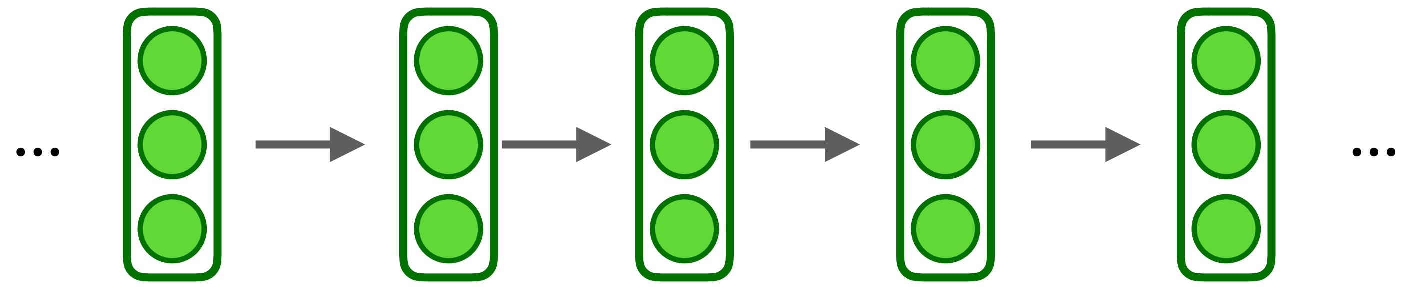
Illustration adapted from Karpathy

Deeper, better models?

Only left to right?

`The person who hunts ducks out on the weekends`

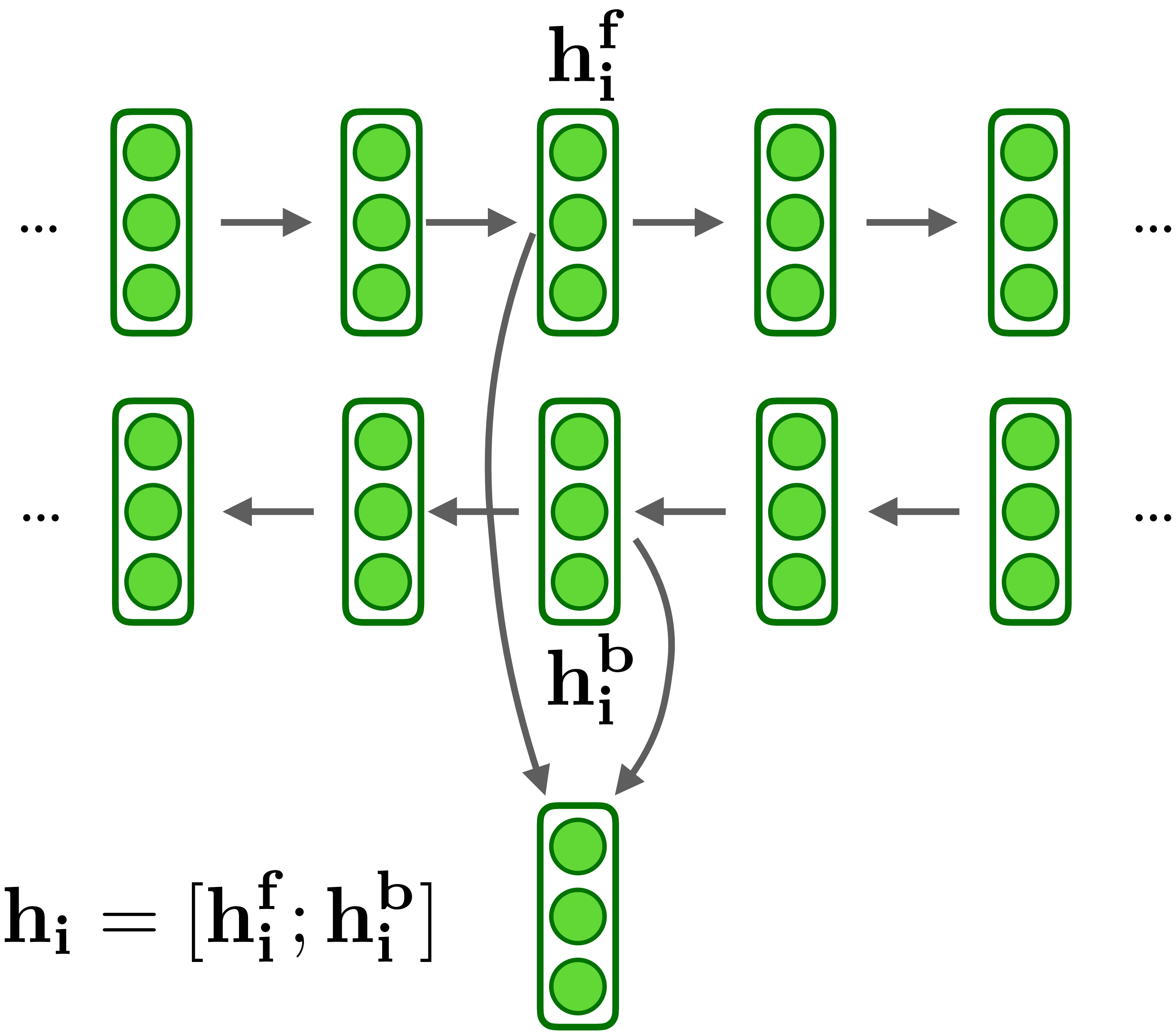
`… person who hunts ducks out …`



Example adapted from Rao & McMahan, 2018 https://en.wikipedia.org/wiki/Garden-path_sentence

Bidirectional RNNs

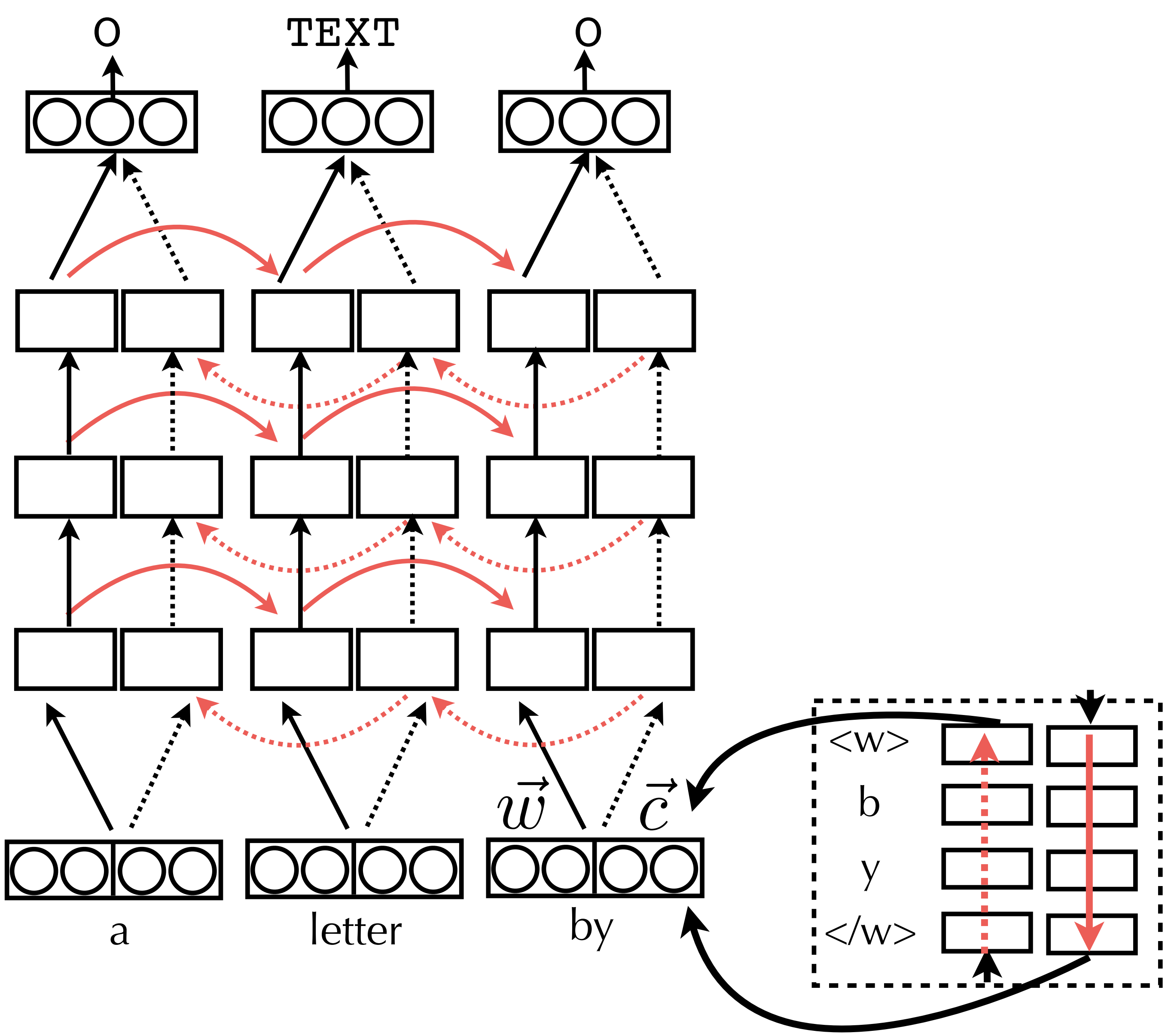
… `person who hunts ducks out` …



$$\mathbf{h_i} = [\mathbf{h_i^f}; \mathbf{h_i^b}]$$

Stacked RNNs

- Multiple layers of RNNs, e.g., bi-RNNs

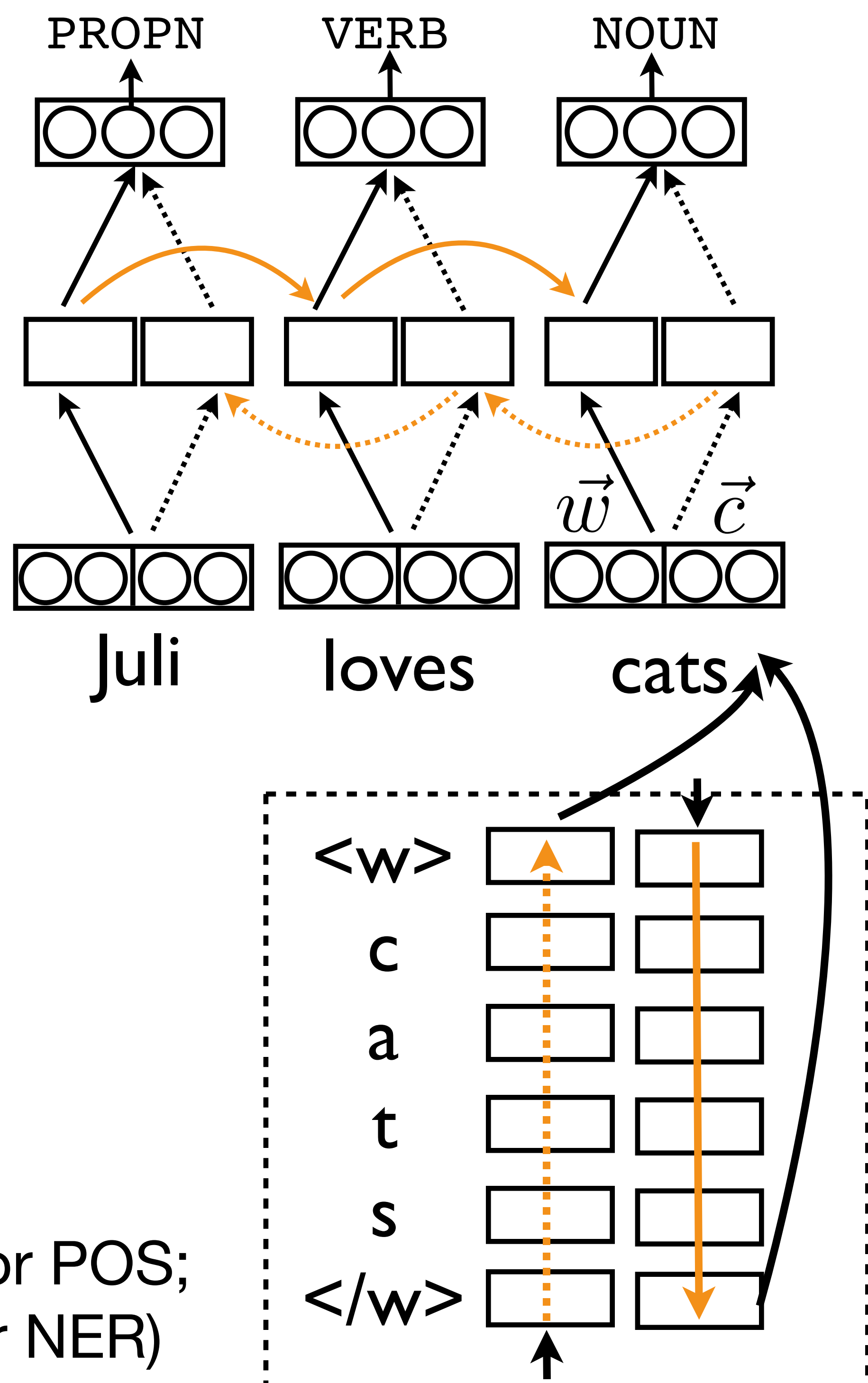


Guards against the long tail?
Subword representations

OOVs (out-of-vocabulary) words

- So far we saw <UNK>
  - But that conflates a lot of information into a single <UNK> representations
- Are we better of modeling at the subword level?

Subword representations: Characters



**able* (98% adj in WSJ)

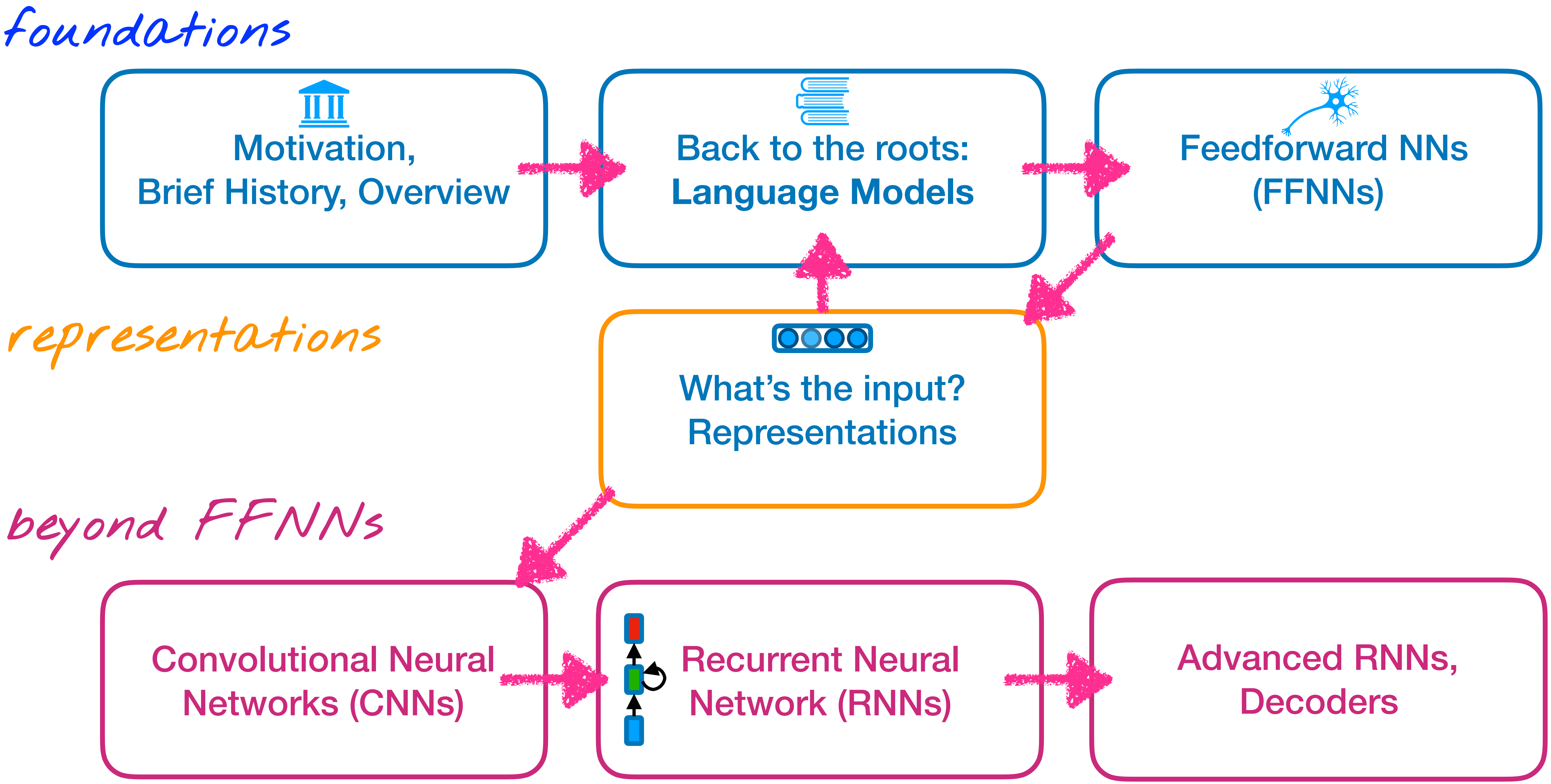
*bi** (85% noun in Danish)

(Plank et al., 2016 for POS; Ling et al., 2015 for NER)

How to model subwords?

- Representations:
  - ***Characters***
  - ***Bytes*** (e.g., Gillick et al., 2015; Plank et al., 2016)
  - ***Byte-Pair Encoding (BPE)*** (Sennerich et al., 2016)
- Modeling choices:
  - RNN-variants, CNNs,…
- How to leverage the representations (only char level, combine, …)

Overview



foundations
Motivation, Brief History, Overview
Back to the roots: Language Models
Feedforward NNs (FFNNs)
representations
What's the input? Representations
beyond FFNNs
Convolutional Neural Networks (CNNs)
Recurrent Neural Network (RNNs)
Advanced RNNs, Decoders

A note on terminology

- RNN = “vanilla” RNN 
- RNN flavors (=gated RNNs):
  - GRU  and LSTMs 
  - Why? Problem of RNNs: Vanishing gradients!

Gated RNN architectures

Vanishing Gradient

- Example:
  - The cat, which ate a …., was full
  - The cats, which … , were full
- Backprop can have difficulties with long sequences: vanishing gradient problem
  - if the gradient becomes very close to zero:
    - is it because there is no dependency in the data?
    - or because of a wrong configuration of the parameters—the vanishing gradient condition?

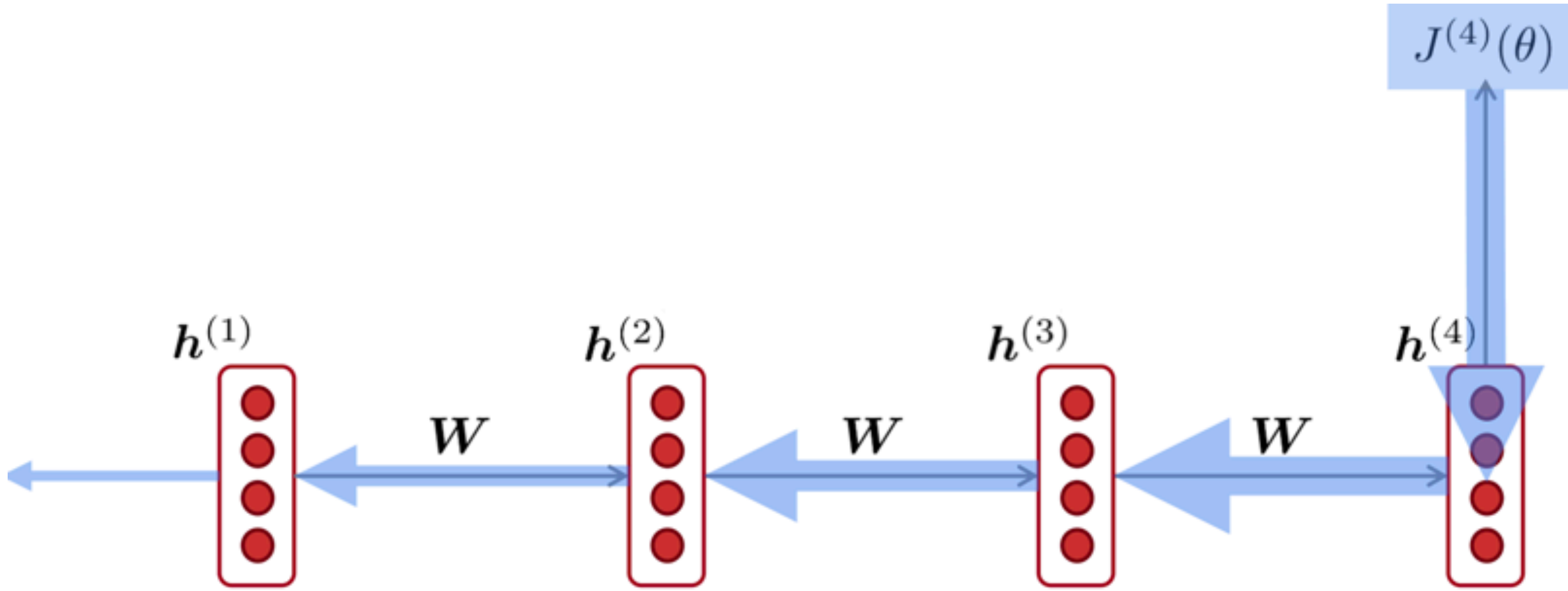
Exploding Gradients

- Easier to catch. If the gradient becomes to big, then the SGD update becomes very large:

$$\theta^{new} = \theta^{old} - \overbrace{\alpha}^{\text{learning rate}} \underbrace{\nabla_\theta J(\theta)}_{\text{gradient}}$$

- This might cause bad updates: too large updates, large loss
- In the worst case, you might get NaNs or Infs
- Solution: **gradient clipping** (scale down before update)

Vanishing Gradient



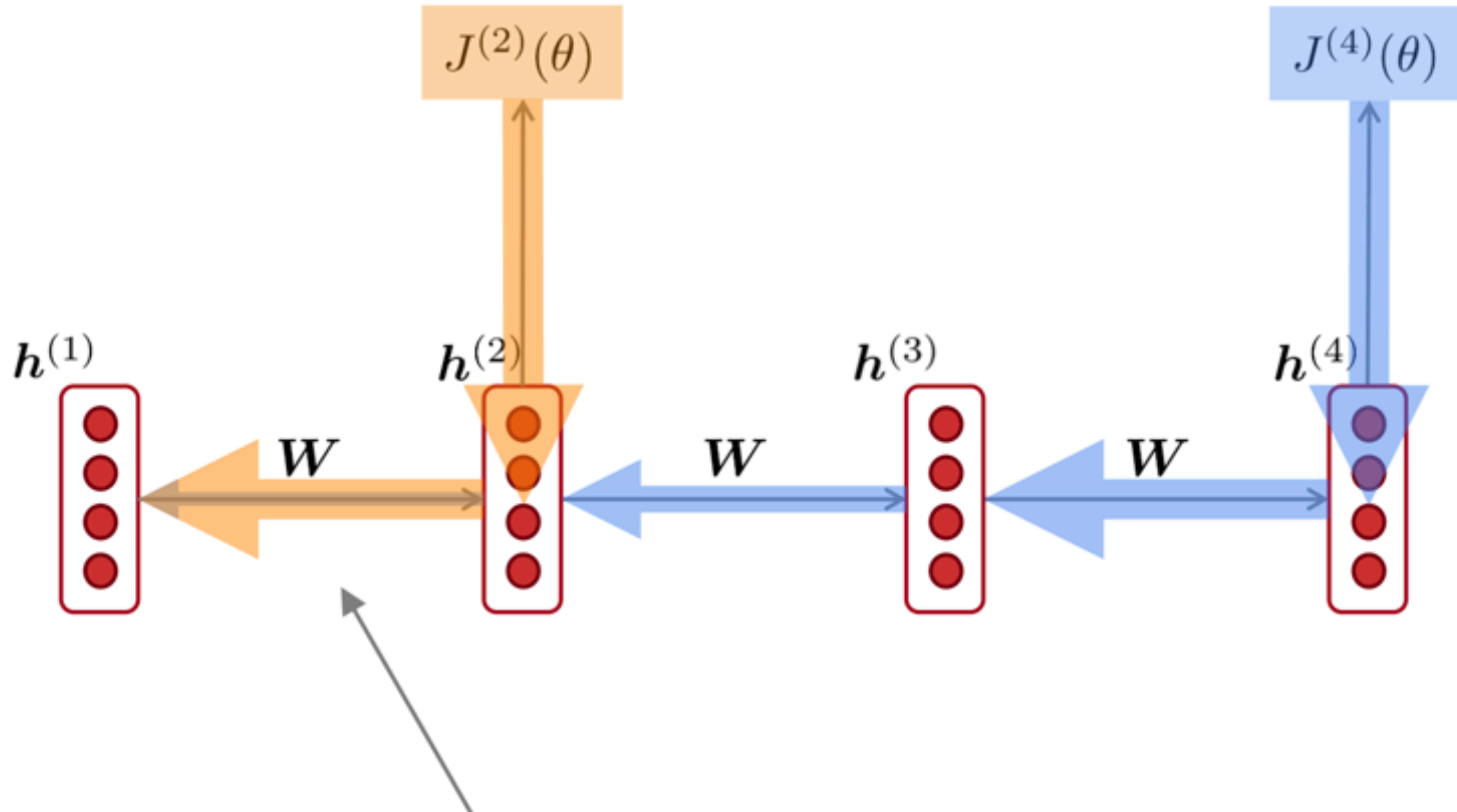
$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \boxed{\frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}}} \times \boxed{\frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}}} \times \boxed{\frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(4)}}$$

What happens if these are small?

<u>Vanishing gradient problem:</u>
When these are small, the gradient signal gets smaller and smaller as it backpropagates further

Slide by Abigail See

Why is Vanishing Gradient a problem?



Gradient signal from faraway is lost because it's much smaller than gradient signal from close-by.

So model weights are only updated only with respect to near effects, not long-term effects.

Slide by Abigail See

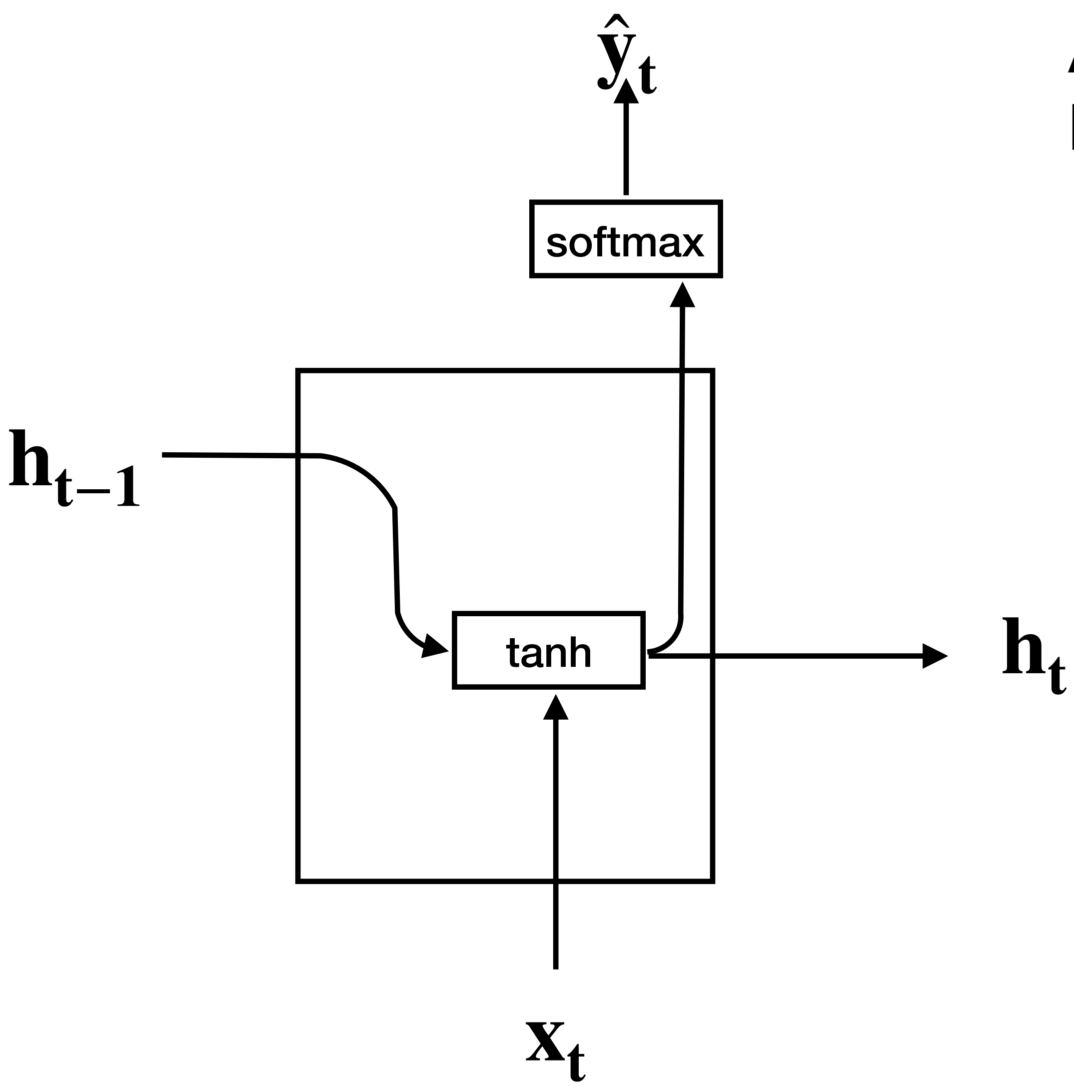
Effect of vanishing gradient

- LM task:

  "When she bought her laptop, she found that the keyboard layout was Danish. She went back to the shop to ask if the owner of the shop had another keyboard layout. Unfortunately this was not the case, so she kept the ____"

- Model needs to learn a dependency to "laptop"
- But if the gradients are small, the model won't learn this
- RNNs are better at syntactic **recency** [Linzen et al., 2016]

RNN unit



At each time step, the hidden state is updated:

$$\mathbf{h}_i = f_\theta(\mathbf{h}_{i-1}, \mathbf{x}_i)$$

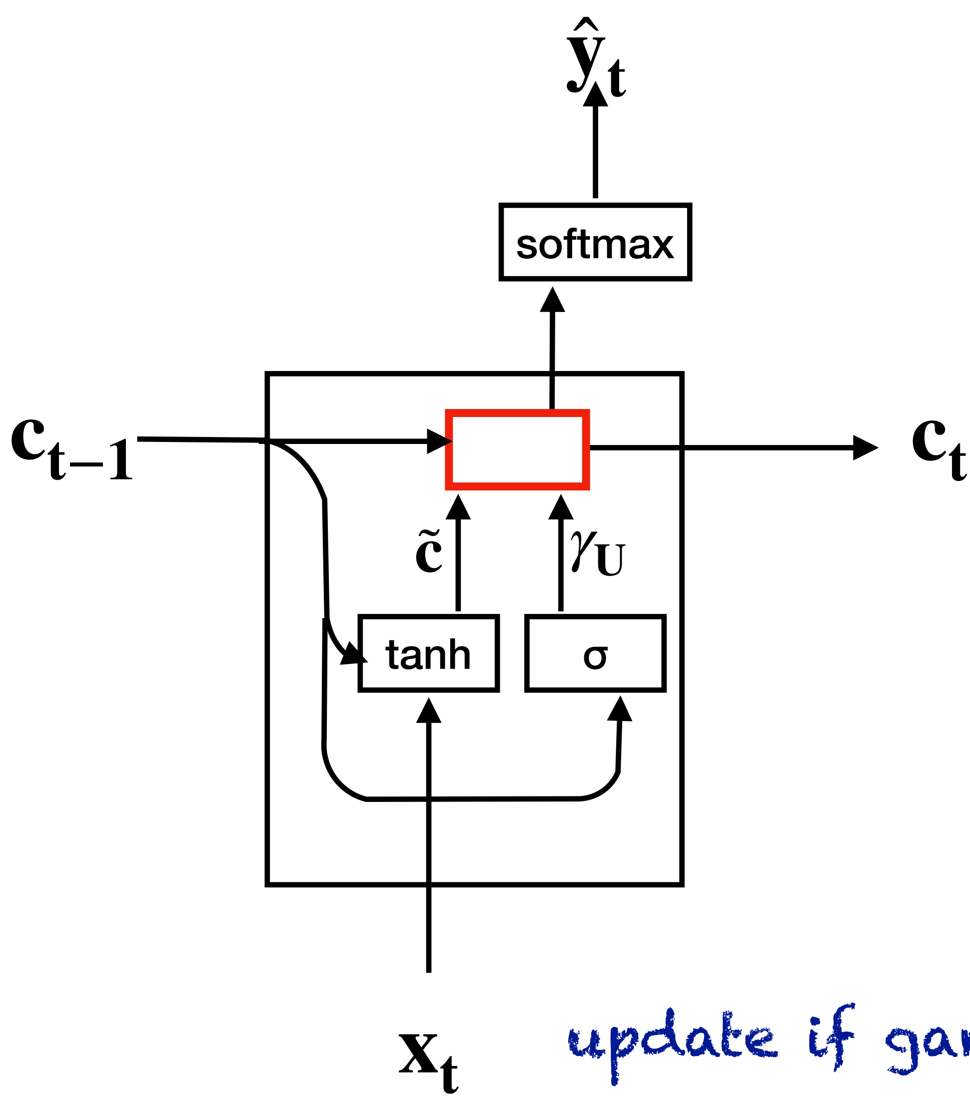
$$\mathbf{h} = tanh(\mathbf{V}\mathbf{x_t} + \mathbf{U}\mathbf{h_{t-1}} + \mathbf{b})$$

in a vanilla RNN the hidden state is constantly being **rewritten**

Gated RNN architectures: RNN flavors with a separate memory

GRU (Gated recurrent Unit) - simplified

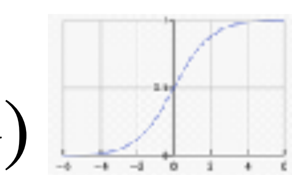
- Cho et al. (2014) - key idea: dynamic memory update $\mathbf{c}$ *(h=c)*
- at every step t, consider overwriting candidate memory c



candidate for overwriting cell

$$\tilde{\mathbf{c}} = tanh(\mathbf{U_c x_t} + \mathbf{W_c c_{t-1}} + \mathbf{b_c})$$

$$\gamma_U = \sigma(\mathbf{U_U x_t} + \mathbf{W_U c_{t-1}} + \mathbf{b_U})$$



"update" gate

sigmoid gate: values between 0 and 1

"choose which bits to update"

$$\mathbf{c_t} = \boxed{\gamma_U \odot \tilde{\mathbf{c}} + (1 - \gamma_U) \odot \mathbf{c_{t-1}}}$$

update if gamma_U > 0

element-wise multiplication

GRU (Gated recurrent Unit) - full

- GRU: creates "adaptive" connections
- perhaps prune some unnecessary connections adaptively

Update gate: controls what parts of the hidden state are updated vs preserved

Reset gate: controls what parts of the previous hidden state are used to compute new content

$$\tilde{\mathbf{c}} = tanh(\mathbf{U_c x_t} + \mathbf{W_c}(\boxed{\gamma_\mathbf{R} \odot} \mathbf{c_{t-1}}) + \mathbf{b_c})$$

$$\gamma_\mathbf{U} = \sigma(\mathbf{U_U x_t} + \mathbf{W_U c_{t-1}} + \mathbf{b_U})$$

$$\gamma_\mathbf{R} = \boxed{\sigma(\mathbf{U_R x_t} + \mathbf{W_R c_{t-1}} + \mathbf{b_R})}$$

$$\mathbf{c_t} = \gamma_\mathbf{U} \odot \tilde{\mathbf{c}} + (1 - \gamma_\mathbf{U}) \odot \mathbf{c_{t-1}}$$

all vectors of same size

How does this help the vanishing gradient problem?
GRUs make it easier to retain info long-term (e.g. by not updating bits)

Slide inspired by Abigail See

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

LSTM (Long-Short Term Memory)

- Introduced by Hochreiter & Schmidhuber 1997
- Separate memory cell c and hidden state h
- Three gates:
  - **forget gate**: controls what is kept and forgotten from previous cell state
  - **input gate:** controls what part of the new cell content are written to the cell
  - **output gate:** controls what part of the new cell content are written to the hidden state

LSTM (Long-Short Term Memory)

We have a sequence of inputs $\boldsymbol{x}^{(t)}$, and we will compute a sequence of hidden states $\boldsymbol{h}^{(t)}$ and cell states $\boldsymbol{c}^{(t)}$. On timestep t:

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

Sigmoid function: all gate values are between 0 and 1

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

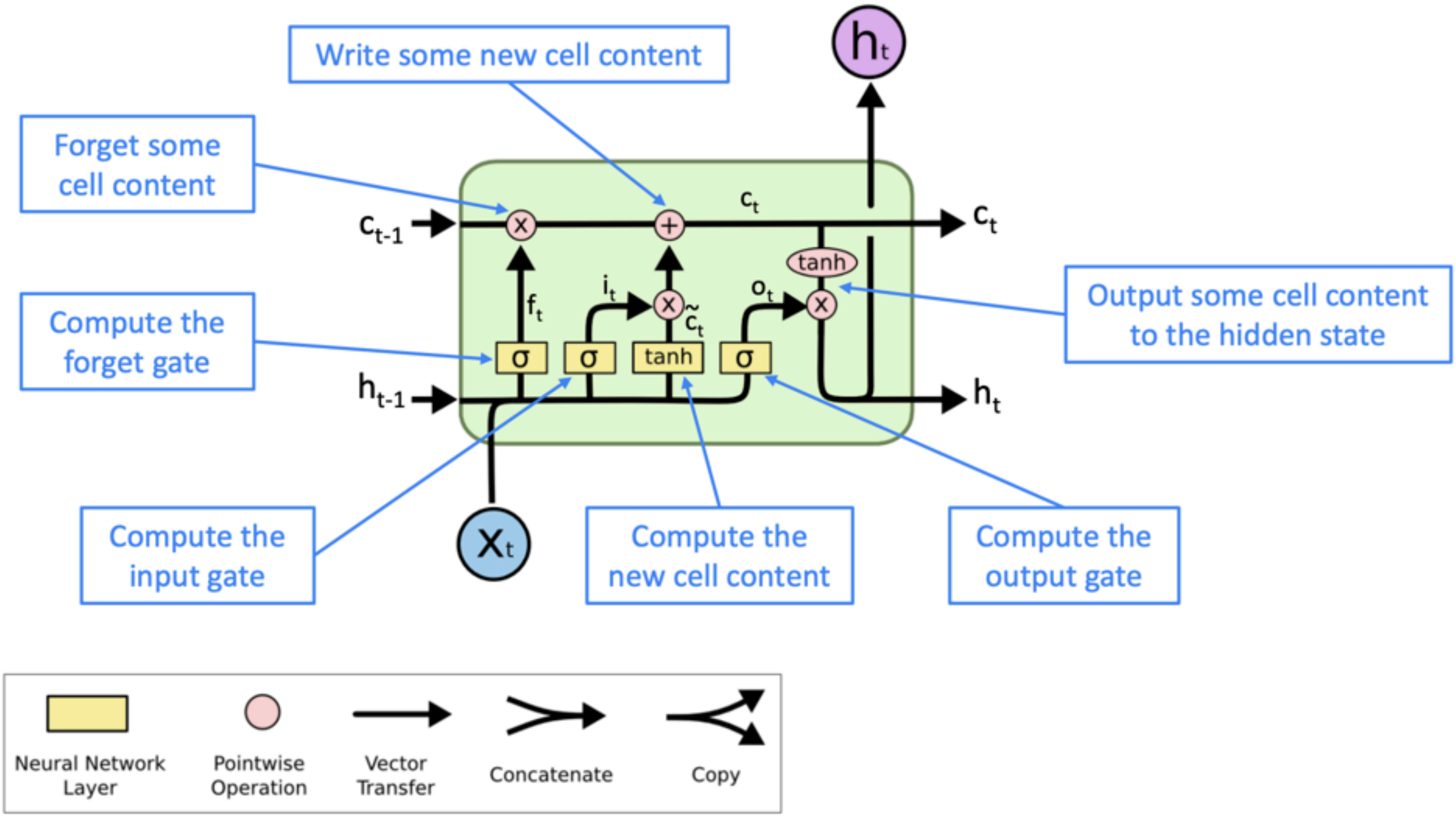
$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

All these are vectors of same length n

Gates are applied using element-wise product

LSTM (Long-Short Term Memory)



Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Slide by Abigail See

GRU vs LSTM

- GRU is more efficient to learn (fewer parameters)
- Which is better?
  - No conclusive evidence that one is always superior to the other
- LSTM is typically a good starting choice
- Suggestion: switch to GRU if you want a more efficient model

Residual connections

- Is the vanishing gradient problem specific to RNNs?
  - No! Also for deep FFNN and ConvNets
- **Solution**: add direct "skip" connections (ResNet, residual connections) - proposed by He et al., (2015)
  - i.e. add F(x) + x, instead of F(x)
  - allows for training deeper models

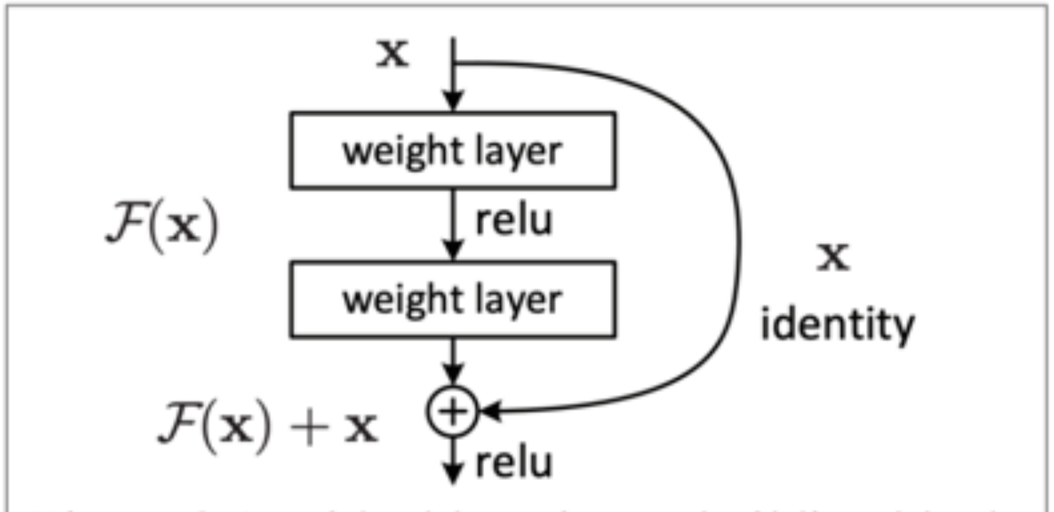


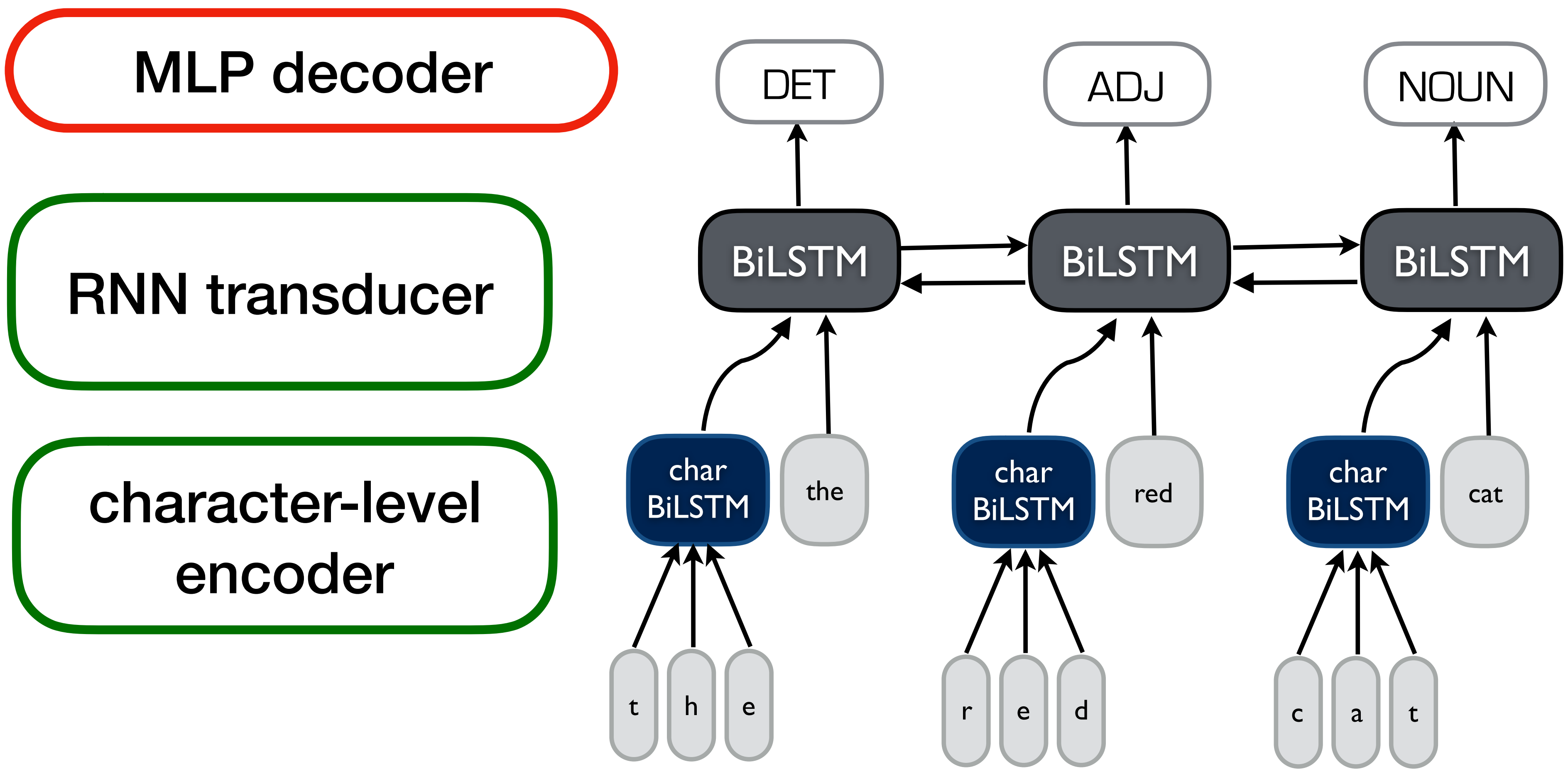
Figure 2. Residual learning: a building block.

LSTMs are everywhere…

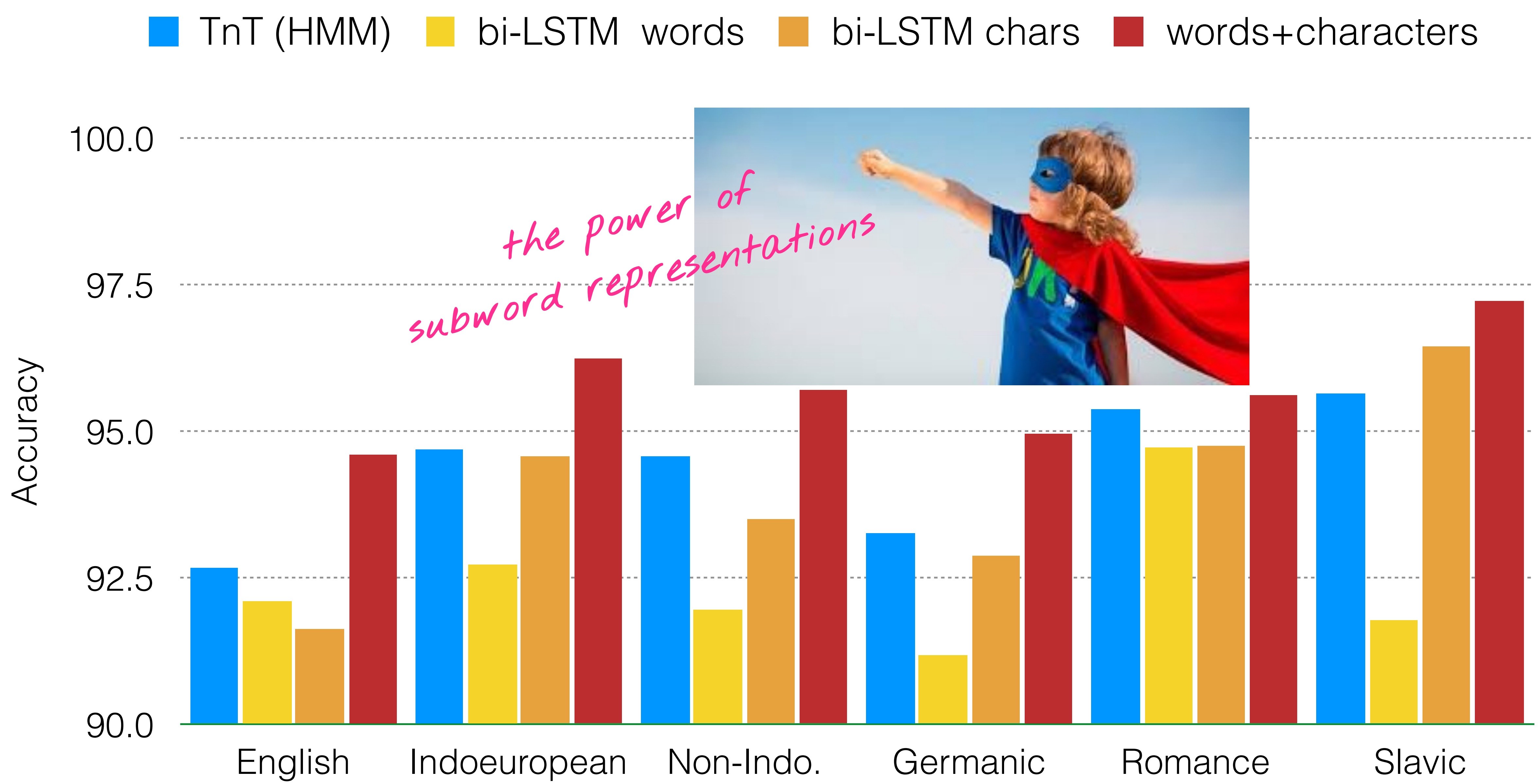
Let’s look briefly at different decoders via examples

A very common POS tagger

- Use **bi-LSTM transducer** with a lower-level **bi-LSTM encoder for characters** and a softmax decoder

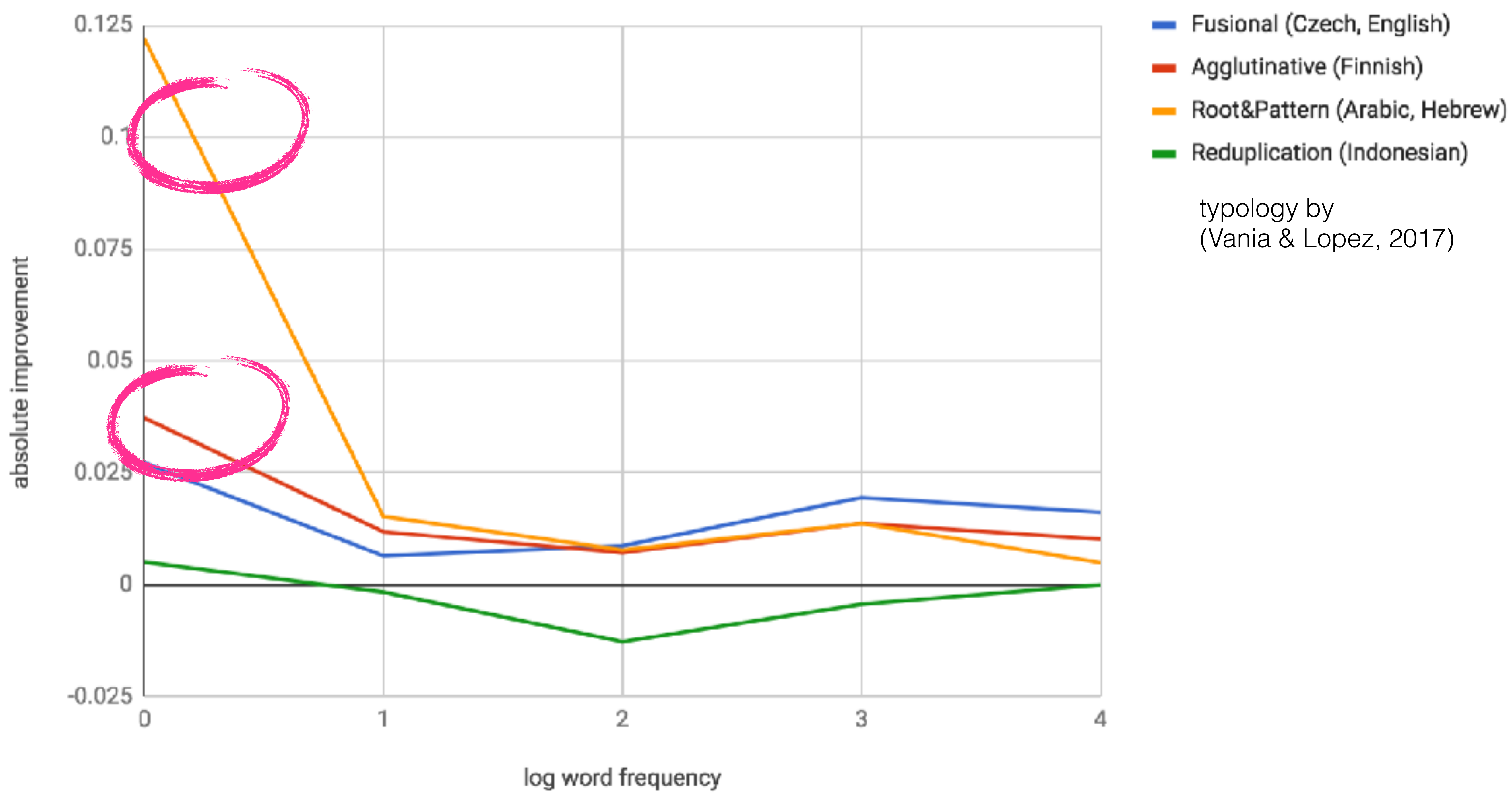


POS tagging on many languages



17 coarse POS tags,
experiments over 22 languages of UD 1.2, (Plank et al., 2016)

A closer look at non-IE languages



typology by
(Vania & Lopez, 2017)

Named Entity Recognition (NER)

- Example: Bill B-PER lives in Athens B-LOC
- (Huang et al., 2015): from RNN to bidirectional LSTM-CRF

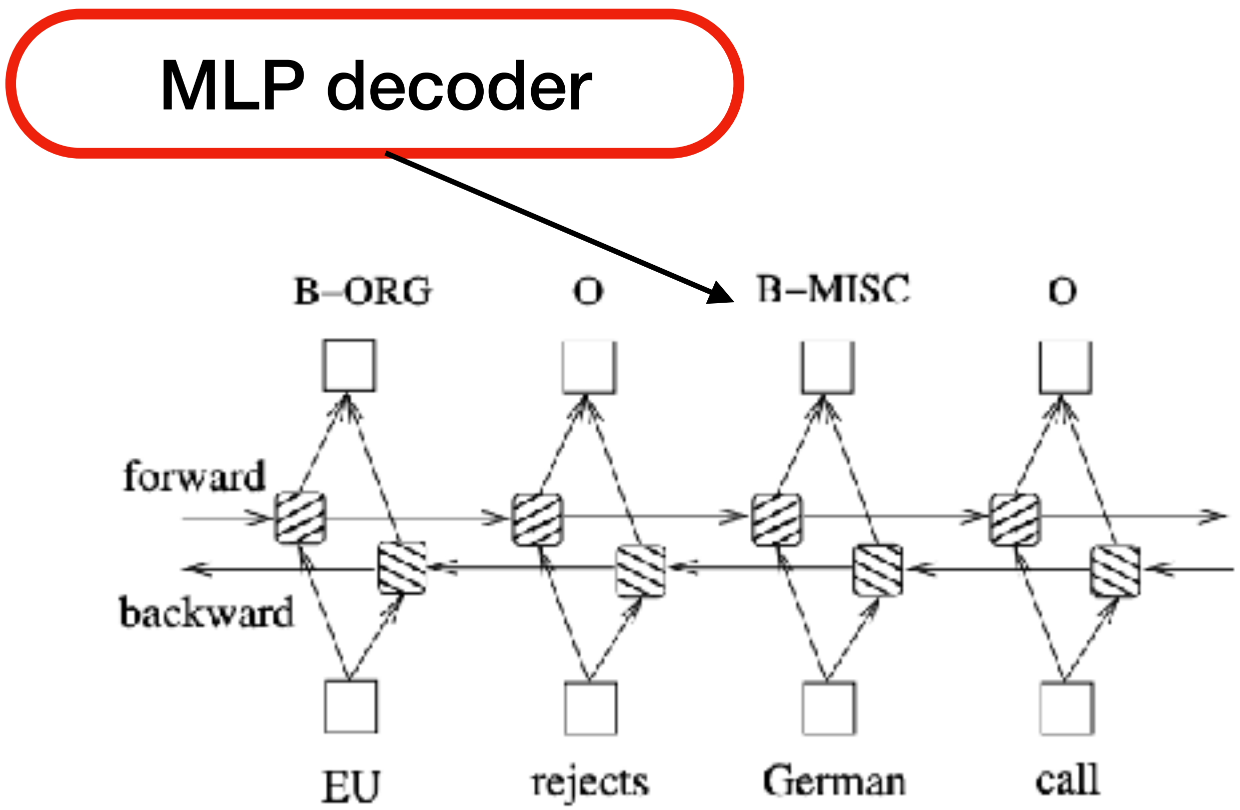


Figure 4: A bidirectional LSTM network.

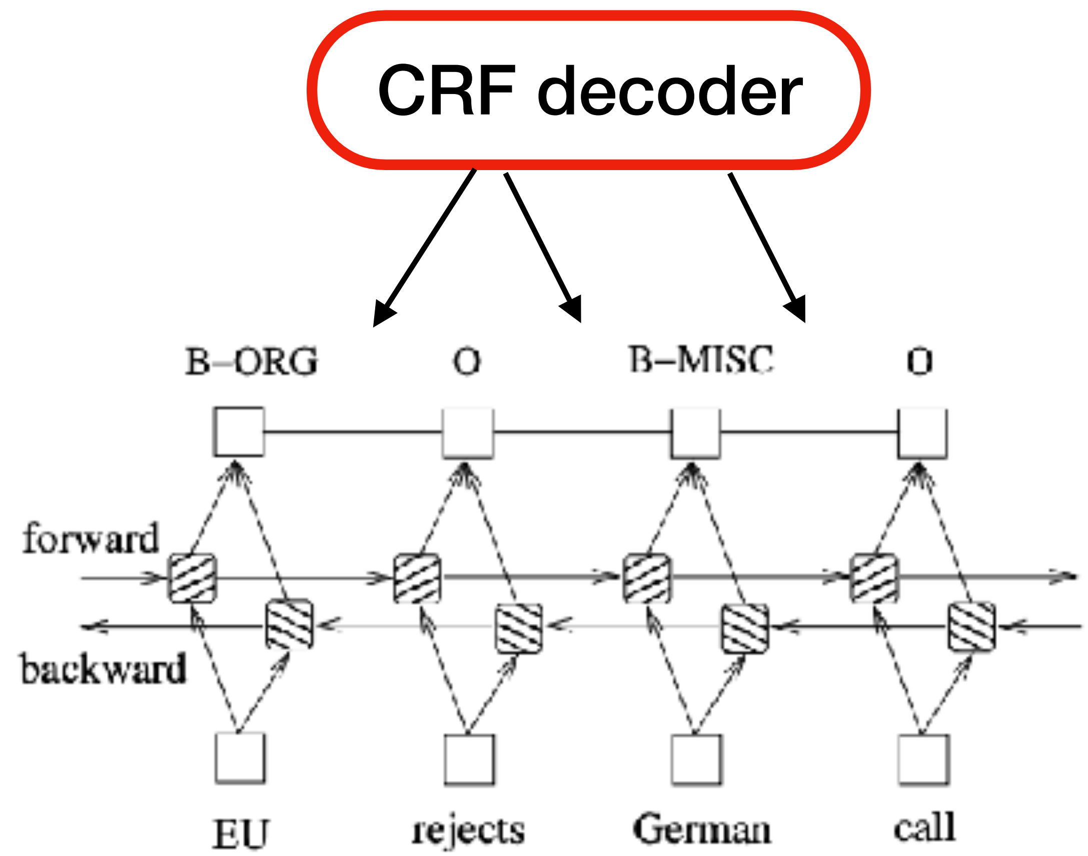


Figure 7: A BI-LSTM-CRF model.

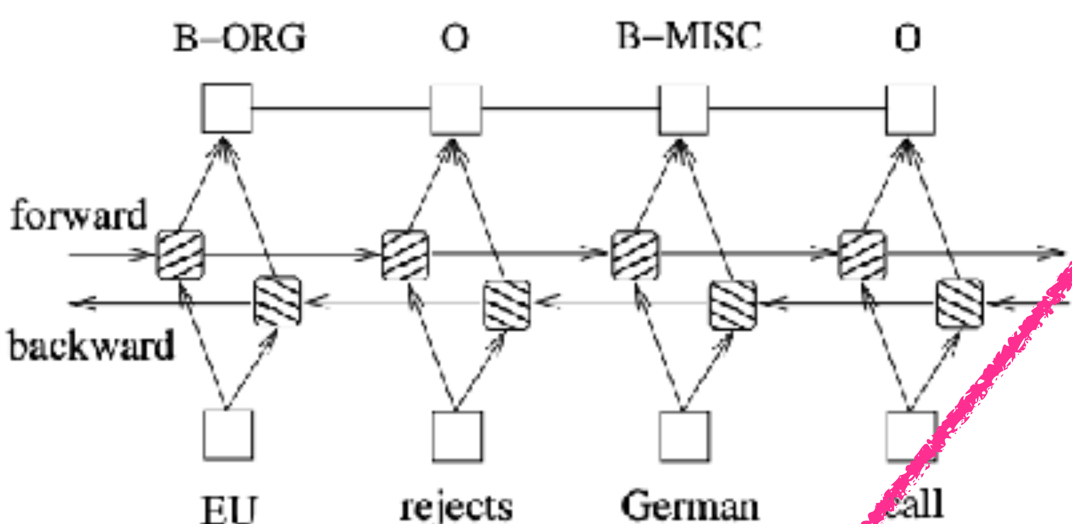
CRF decoder

- Stronger sequential nature (e.g., I-PER after B-PER)

Bill B-PER | Doe I-PER | lives in | Athens B-LOC

Lampe et al., (2016):

$$\mathbf{y} = (y_1, y_2, \ldots, y_n)$$



$$\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n),$$

CRF maintains matrix **A:** transition scores matrix (k x k tags plus start/end)- A_{ij} score from tag i to j

$$s(\mathbf{X}, \mathbf{y}) = \sum_{i=0}^{n} A_{y_i, y_{i+1}} + \sum_{i=1}^{n} P_{i, y_i}$$

P: output of bi-LSTM projected onto hidden layer (of size n x k) - P_{ij}: score of jth tag for i-th word

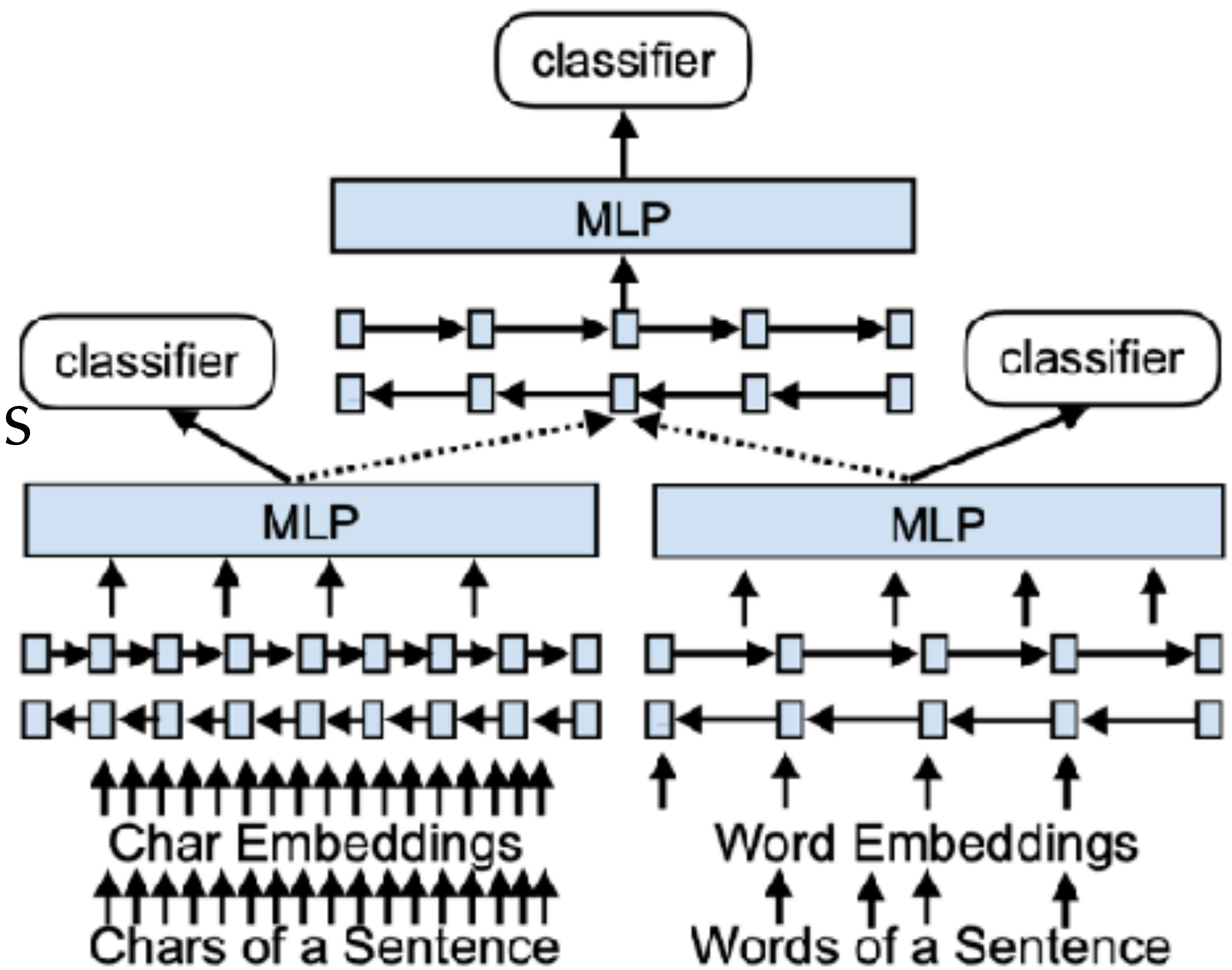
P is input to the CRF layer

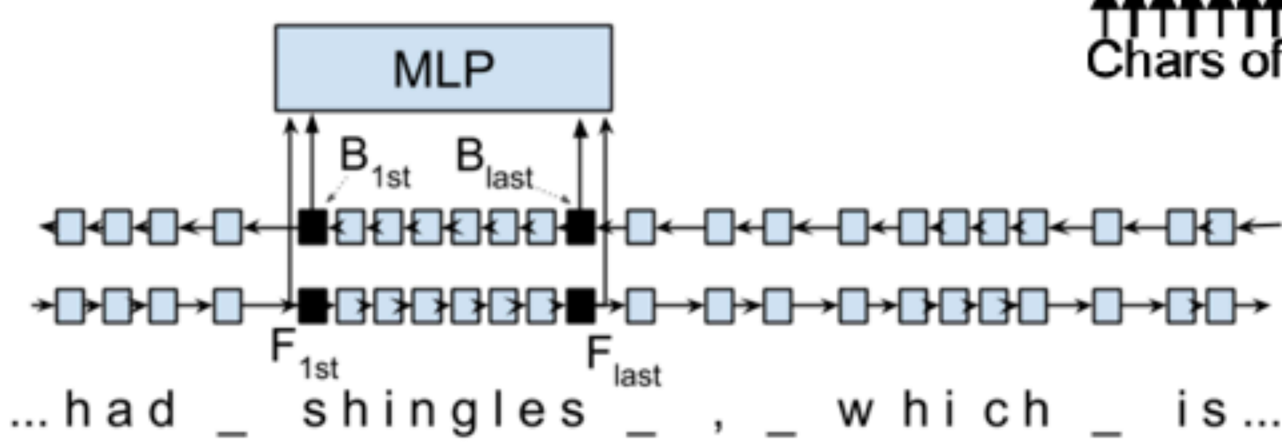
$$p(\mathbf{y}|\mathbf{X}) = \frac{e^{s(\mathbf{X}, \mathbf{y})}}{\sum_{\tilde{\mathbf{y}} \in \mathbf{Y_X}} e^{s(\mathbf{X}, \tilde{\mathbf{y}})}}.$$

Softmax over all possible tag sequences Y; dynamic programming

Meta-BiLSTM

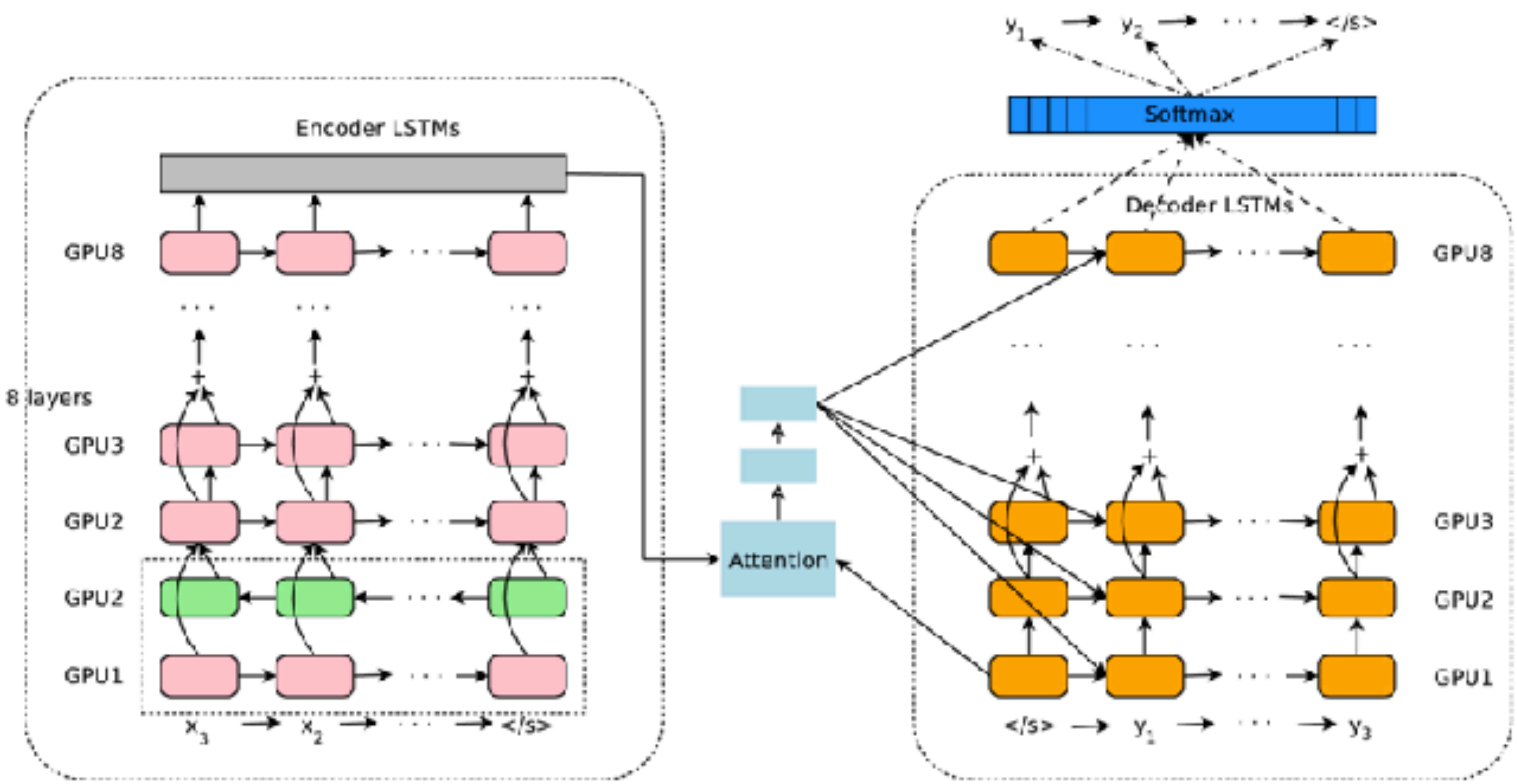
- The model so far is restricted to subwords to within words
- Recent SOTA model
  - context-sensitive character and word representations
  - Models trained synchronously & then combined





(Bohnet et al., 2018)

Google's Neural MT System (Wu et al., 2016)



- **deep bidirectional LSTM** (stacked) with **residual connections** and **attention**
- huge improvements in MT quality

https://arxiv.org/pdf/1609.08144.pdf

- Now (**2019** onwards): other approaches have become dominant for certain NLP tasks (e.g. the **Transformer**) - see more on Monday (Arianna)

Interim summary

- RNNs:
  - Two fancy variants: **LSTM and GRU** to address the vanishing gradient problem

  - Deep RNNs (stacking)
  - Residual connections
- Two more concepts to cover:
  - beyond static word embeddings
  - gluing it all together: attention!

Overview

foundations

Motivation,
Brief History, Overview

Back to the roots:
Language Models

Feedforward NNs
(FFNNs)

representations

What's the input?
Representations

Contextualized
Embeddings, Fine-tuning

beyond FFNNs

Convolutional Neural
Networks (CNNs)

Recurrent Neural
Network (RNNs)

Advanced RNNs,
Decoders

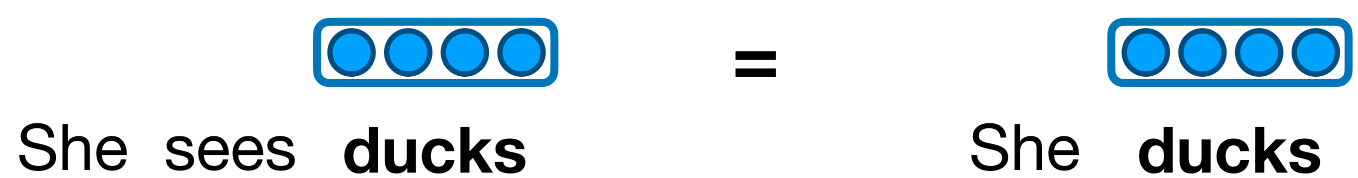
Traditional (“static”) word embeddings

compress all contexts into a single vector

Contextualized word embeddings

Representing a word as vector so far

- Problem:
  - It is a **type-based** representation: always the same vector for a word regardless of its context (e.g. 'ducks')
  - Polysemy is not handled



- Solution: Contextualized embeddings
  - Learn a vector that depends on the context

Language Models to the rescue!

Deep contextualized word representations

Matthew E. Peters[†], **Mark Neumann**[†], **Mohit Iyyer**[†], **Matt Gardner**[†],
{matthewp,markn,mohiti,mattg}@allenai.org

Christopher Clark[*], **Kenton Lee**[*], **Luke Zettlemoyer**[†*]
{csquared,kentonl,lsz}@cs.washington.edu

[†]Allen Institute for Artificial Intelligence
[*]Paul G. Allen School of Computer Science & Engineering, University of Washington

Peters et al., NAACL 2018

Src: Wikipedia

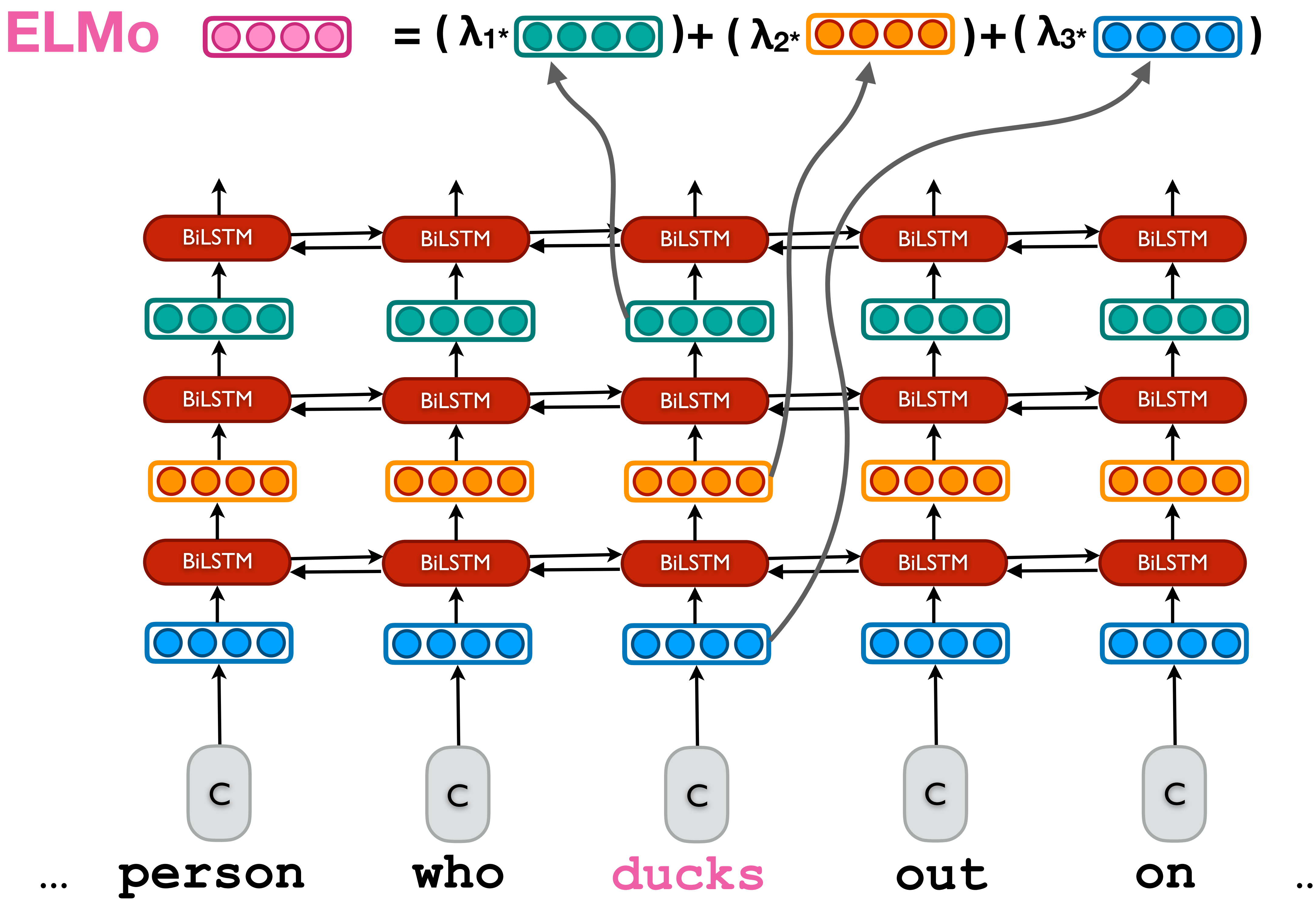
ELMo: Embeddings as Language Models

- Neural LMs embed the left and right context of a word
- We can use a bi-directional LM with the forward and the backward LSTM states

$$\sum_{k=1}^{N} \left(\log p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \log p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s) \right).$$

- **Key Idea:** Learn word token vectors (not type!) using long contexts (not only context *windows*)
- ELMO uses a "deep" model to get different encodings (or "views") from stacked RNNs

Embeddings from Language Models



ELMo - Details

- ELMo: every token is assigned a representation that is a function of the entire input sentence (L=#stacked layers)

$$\begin{aligned} R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\ &= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\}, \end{aligned}$$

- This gives 2L+1 representations - Which to use?
  - Just the top layer (similar to TagLM; Peters et al., 2017)
  - Include all L+1 layers, average
  - All layers, weighted average (best)

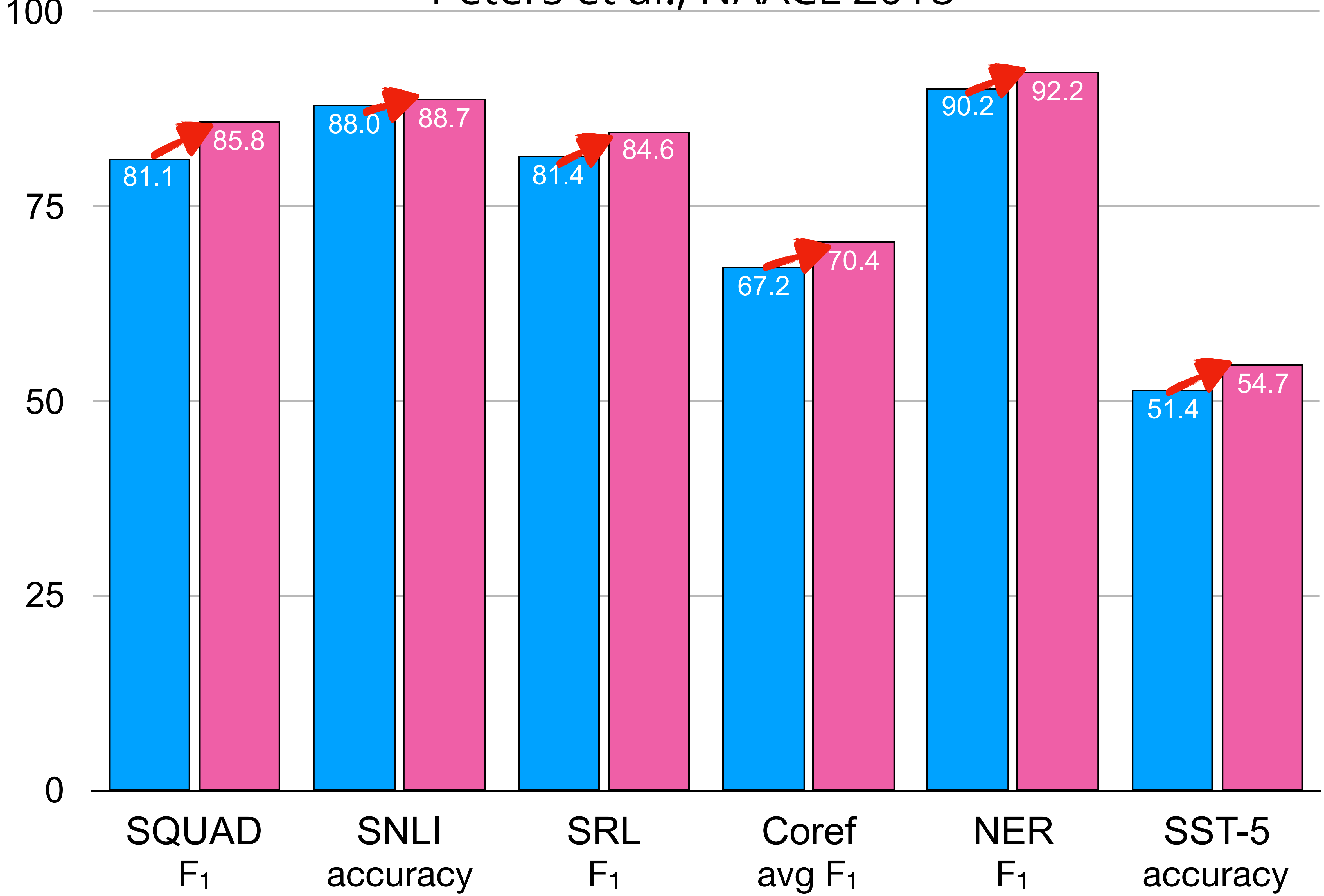
$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}$$

How to use ELMo for your task?

- Recipe: For a given instance
  - Run biLM to get the representations for each word
  - Concatenate ELMo embeddings into task-specific model, e.g.,
    - as additional input to static word embeddings
    - as additional hidden representation
    - … many choices, best might depend on end task

Results over 6 NLP benchmarks

Peters et al., NAACL 2018



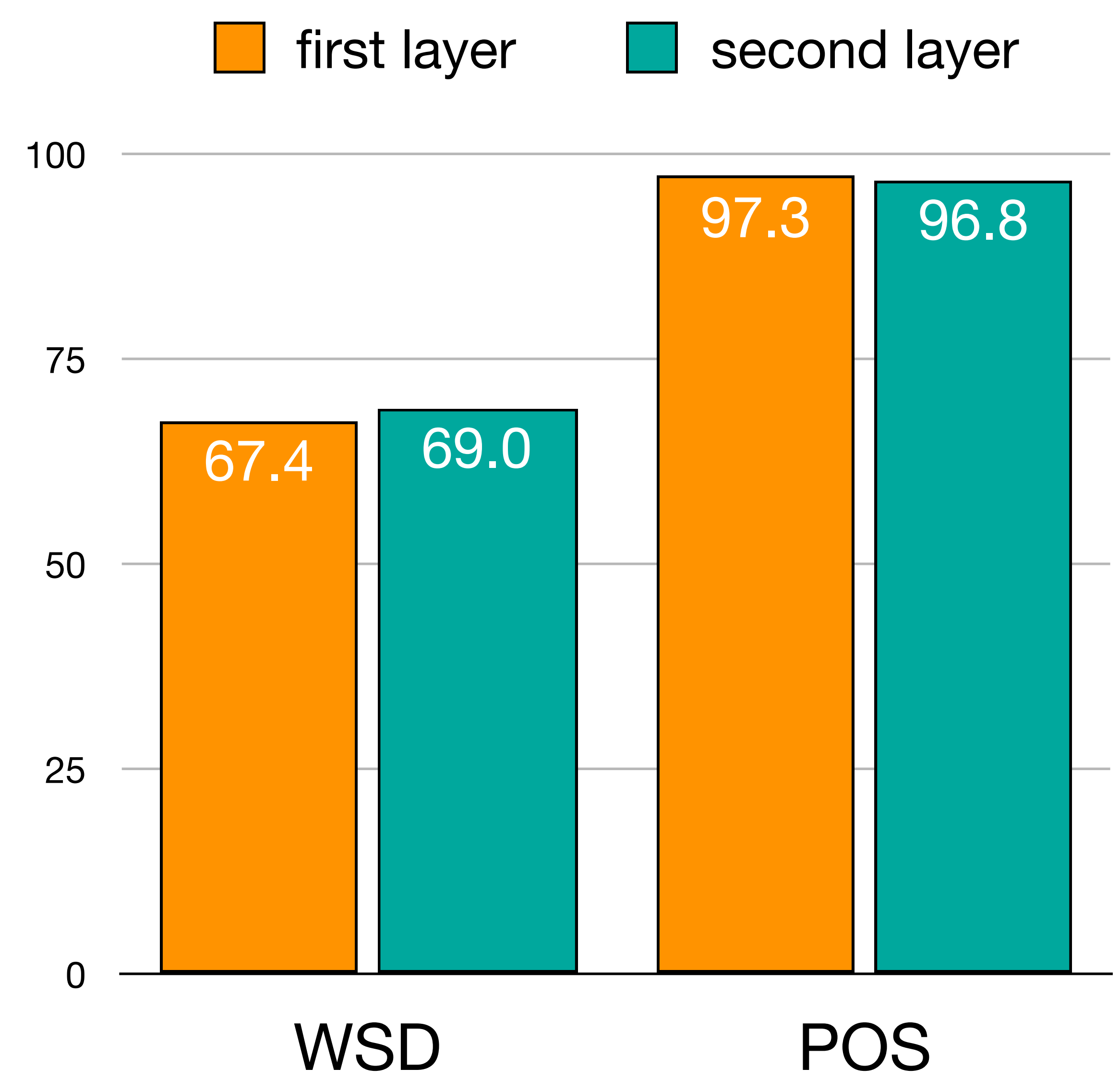
Is ELMo the first such model? No!

- ELMo is *deeper* compared to an earlier model by Peters et al., 2017 ACL (**TagLM**)
- It doesn't require parallel data (as **CoVe** does) by McCann et al., 2017 NeurIPS
  - CoVe: use NMT as encoder (translation is meant to preserve meaning, so why not use it to provide context?)
- It obtained a new SOTA on 6 benchmarks

What’s in a representation?

Probing ELMo representations

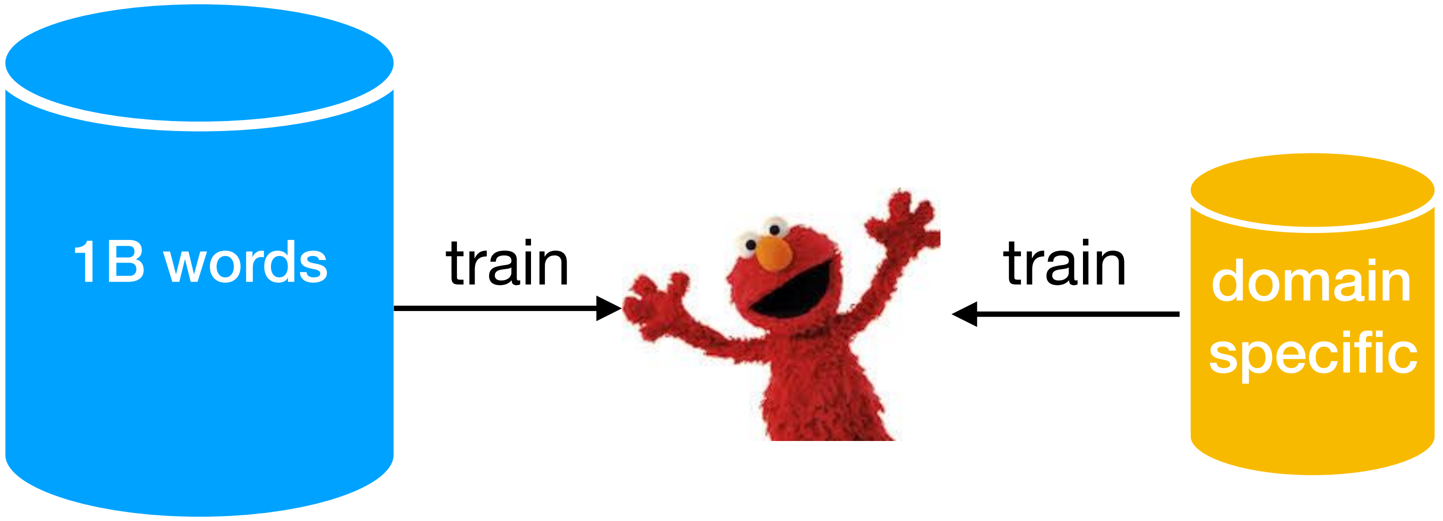
- What do ELMo representations capture?
  - Word Sense Disambiguation (WSD)
  - Part-of-Speech tagging (POS)
- Finding: Different layers encode different kinds of syntactic and semantic information



(Selected related work): Tenney et al., 2019 ACL; Liu et al., 2019 NAACL
Belinkov & Glass, 2019 TAC

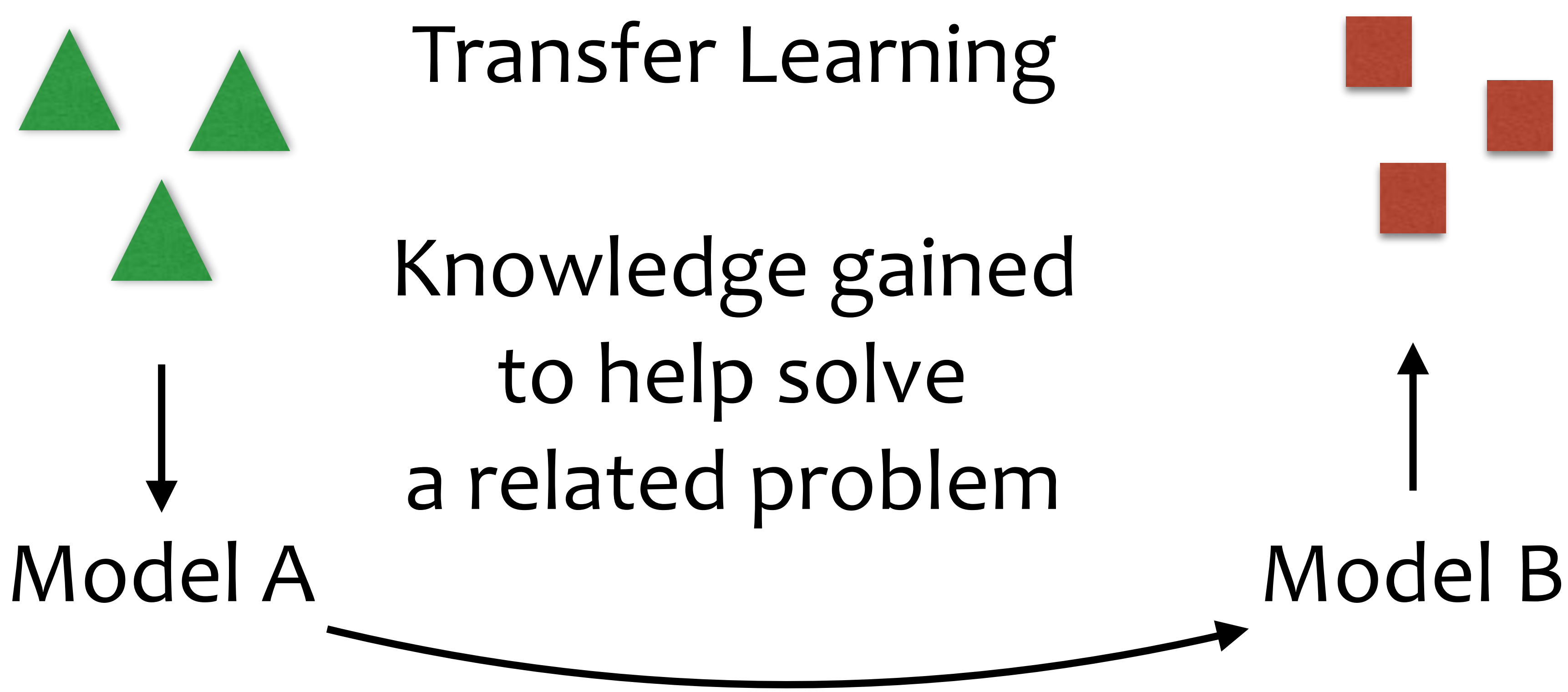
On what was ELMo trained?

- A news corpus of 1B words: the 1-billion word language modeling benchmark (Chelba et al., 2014)
- ELMo can compute representations for any task
- In some cases, **fine-tuning** ELMo on domain-specific data leads to increased downstream performance



Fine-tuning: train on large data, continue training on small data (reuse weights)

Fine-tuning: One way of Transfer Learning



Language models learn transferable contextual representations

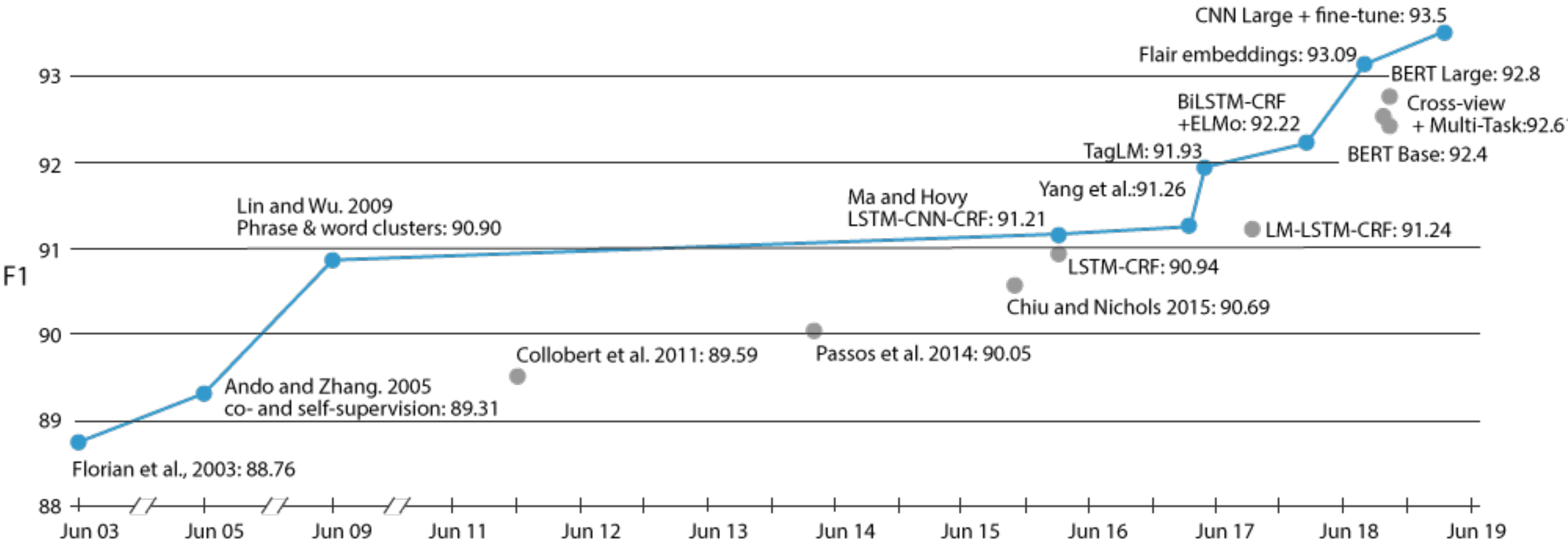
To sum up: ELMo properties

- unsupervised
- contextual
- deep
- character-based
- extremely versatile (new type of word representation)

- Many follow-up words, most of which rely on the transformer model (Lecture 4), e.g., BERT

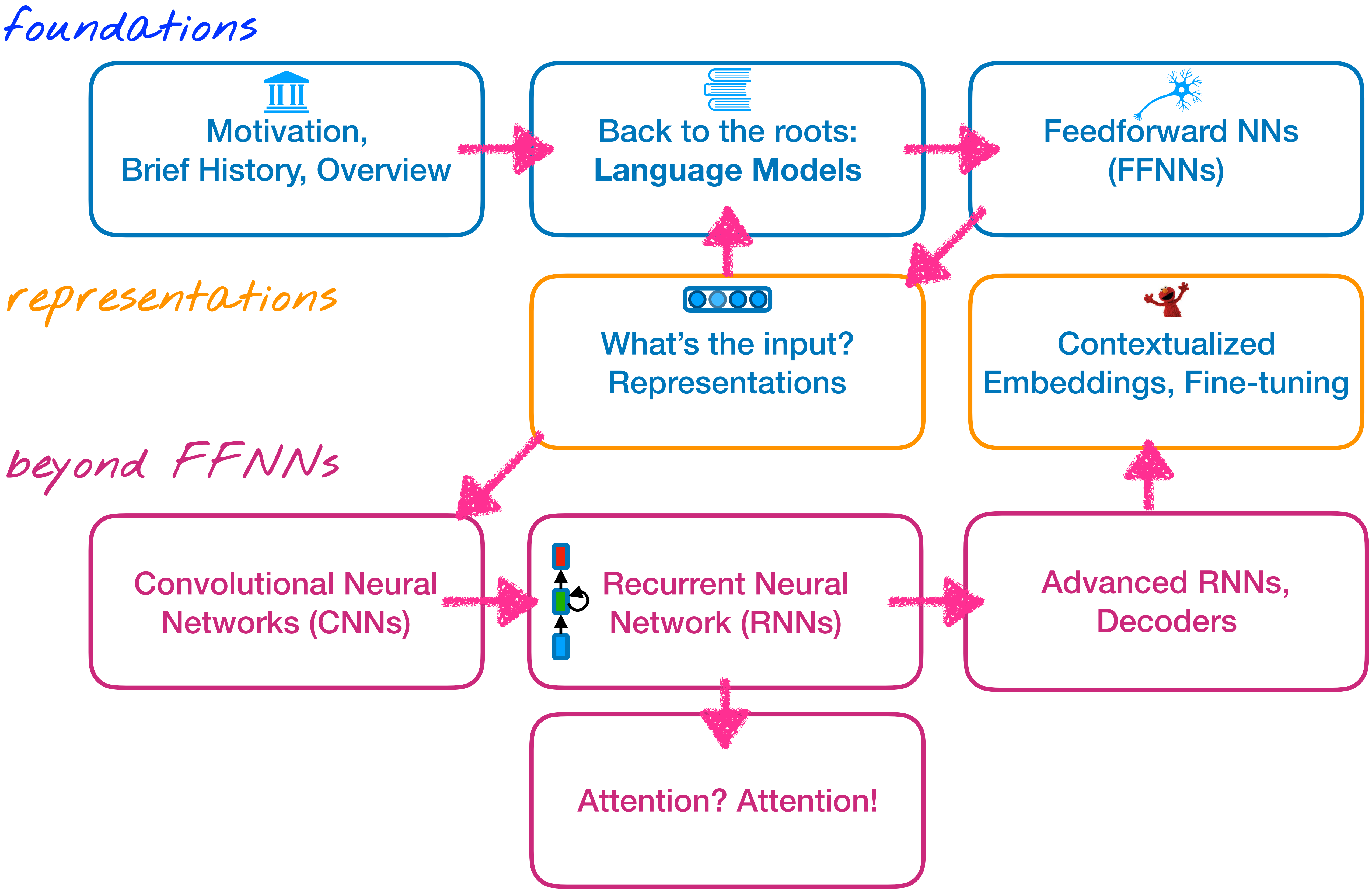
NLP Progress on NER

- From Ruder et al.'s 2019 NAACL tutorial



Performance on Named Entity Recognition (NER) on CoNLL-2003 (English) over time

Overview

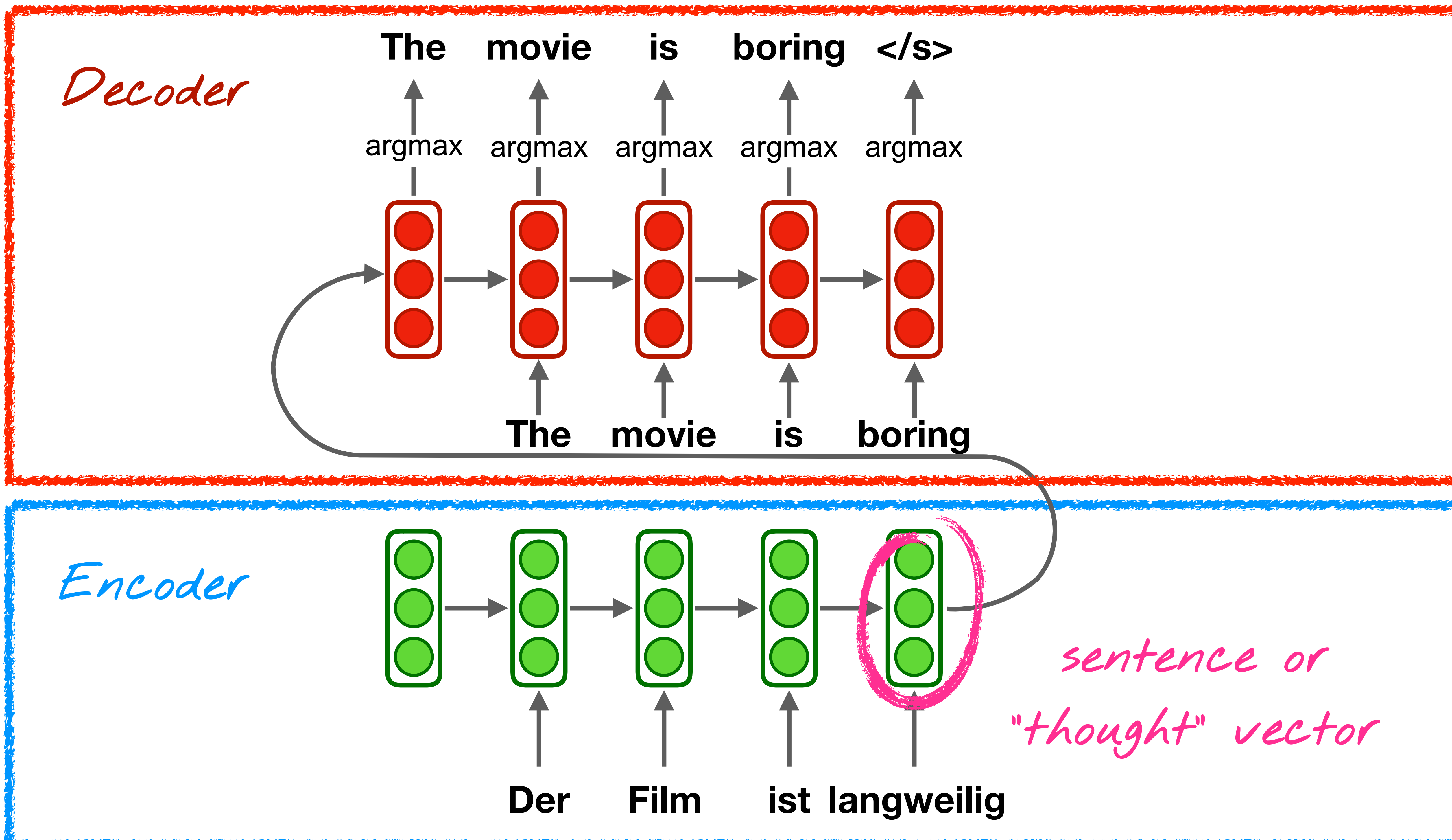

foundations
Motivation,
Brief History, Overview
Back to the roots:
Language Models
Feedforward NNs
(FFNNs)
representations
What's the input?
Representations
Contextualized
Embeddings, Fine-tuning
beyond FFNNs
Convolutional Neural
Networks (CNNs)
Recurrent Neural
Network (RNNs)
Advanced RNNs,
Decoders
Attention? Attention!

Attention? Attention!

Many thanks to Lilian Weng for an awesome tutorial (https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html) and Graham Neubig's NN for NLP class (http://www.phontron.com/class/nn4nlp2019/)

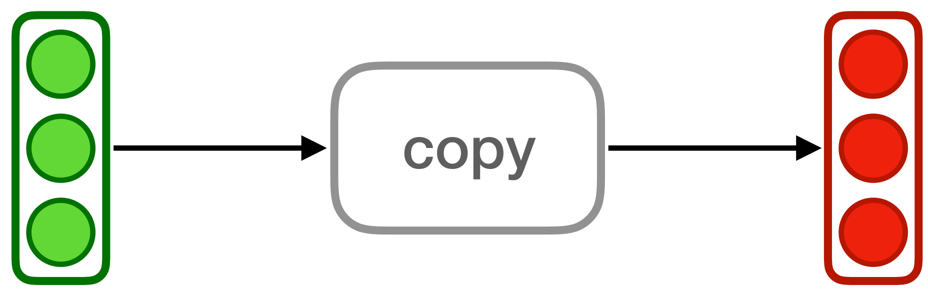
Motivation: Encoder-decoder model

(Sutskever et al., 2014; Cho et al., 2014)



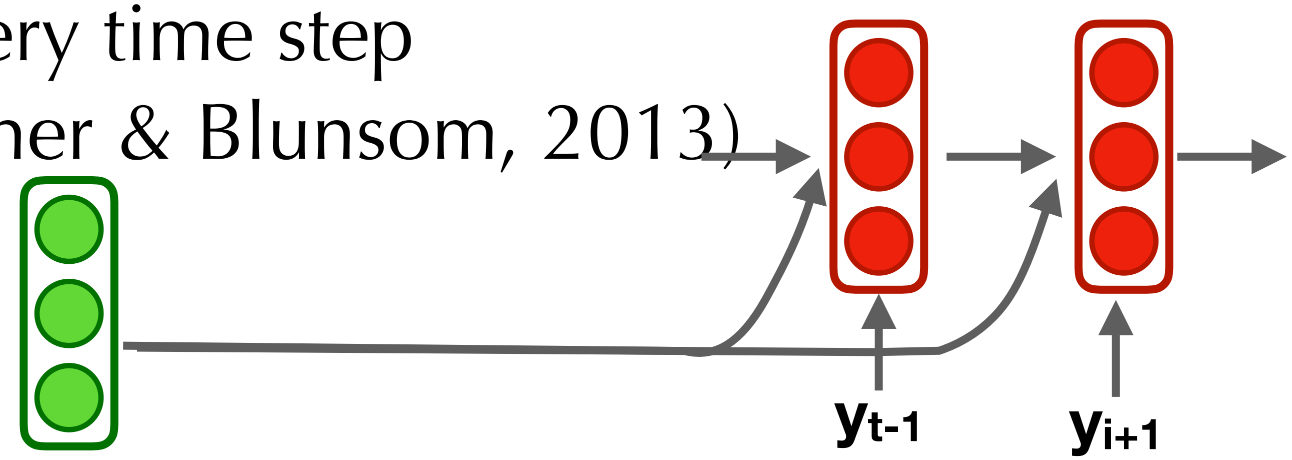
How to pass the sentence vector?

- Initialize decoder with encoder representation (Sutskever et al., 2014)

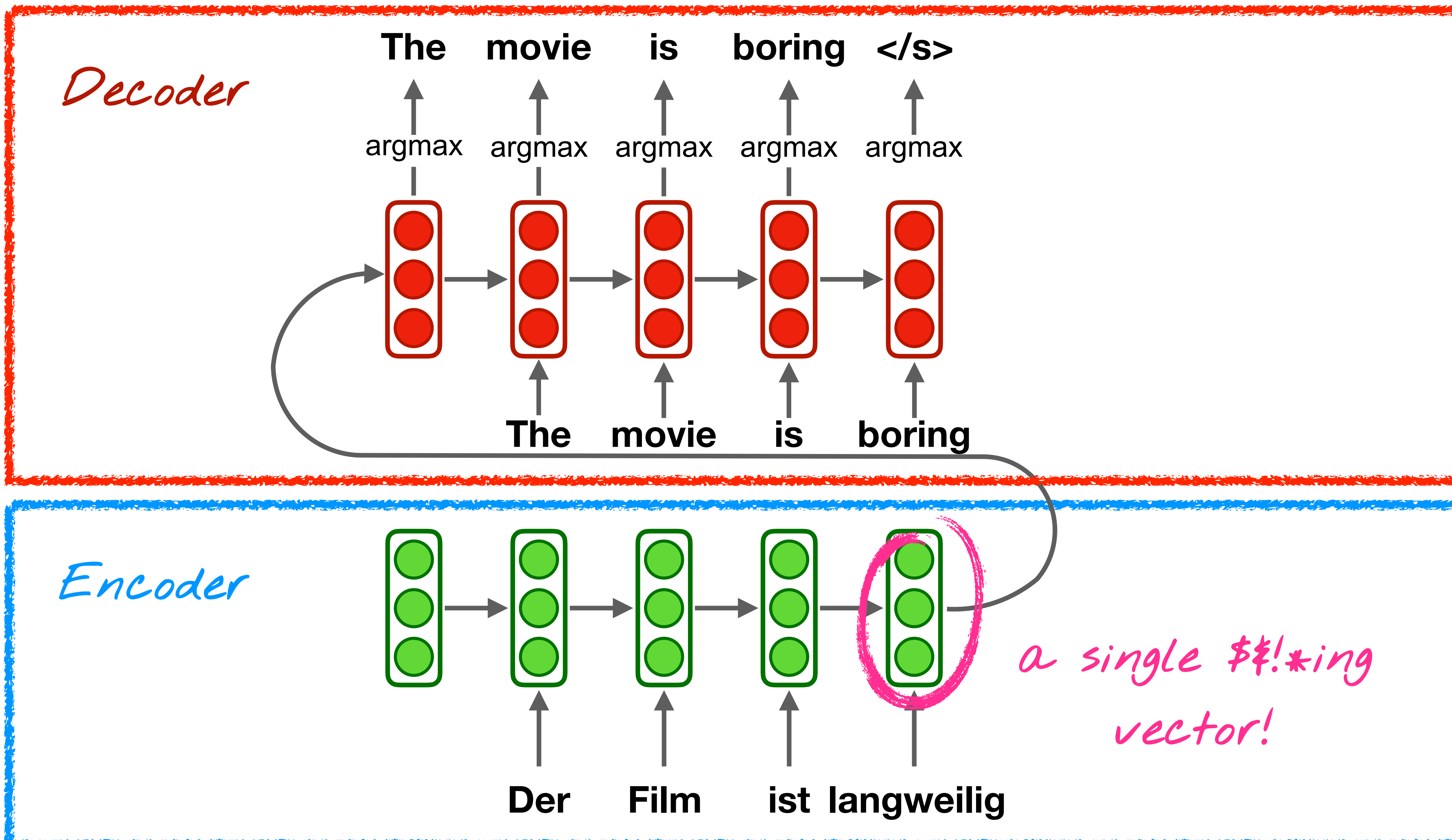


- Transform (change dimensionality)

- Input at every time step (Kalchbrenner & Blunsom, 2013)



But: we're cramming it all into..

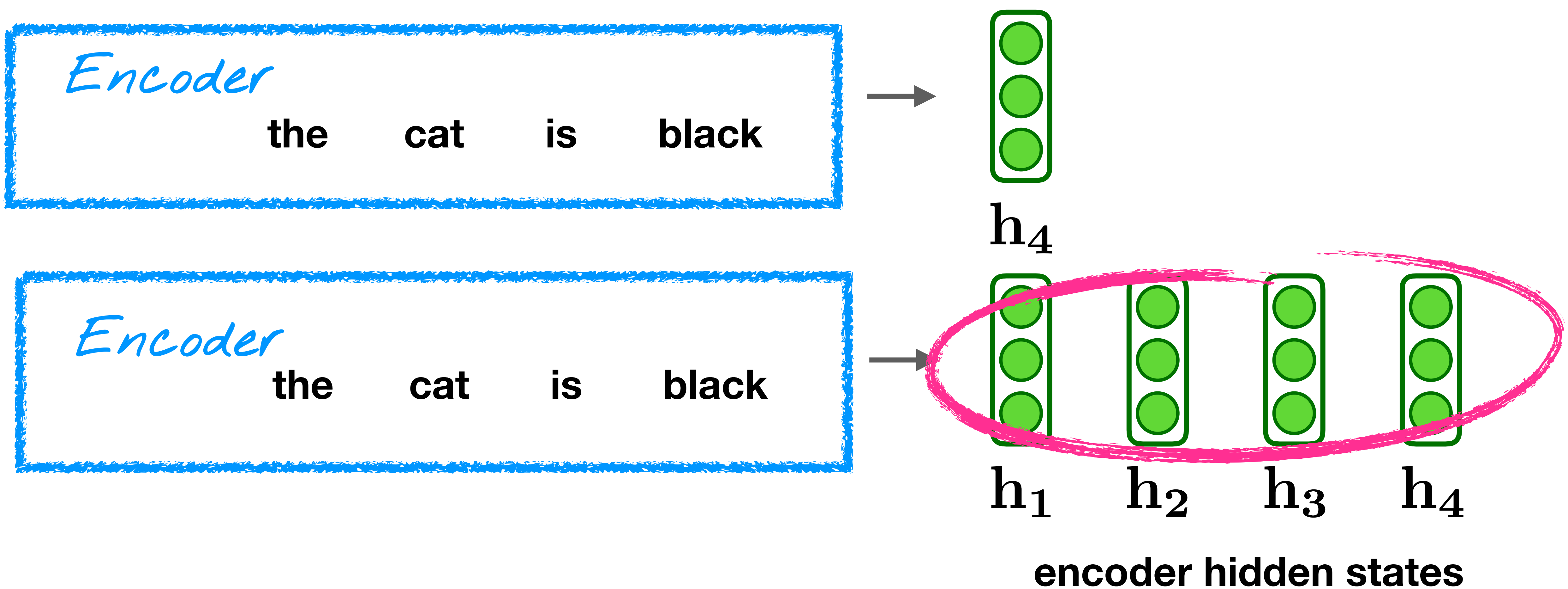


Problem

- The encoder compresses the sentence into a single fixed-size vector. This representation is expected to be a good summary of the entire sentence.
- Disadvantage: incapability of remembering longer sequences.
- "**You can't cram the meaning of a of a whole %&!$ing sentence into a single $&!*ing vector!**" — Ray Mooney

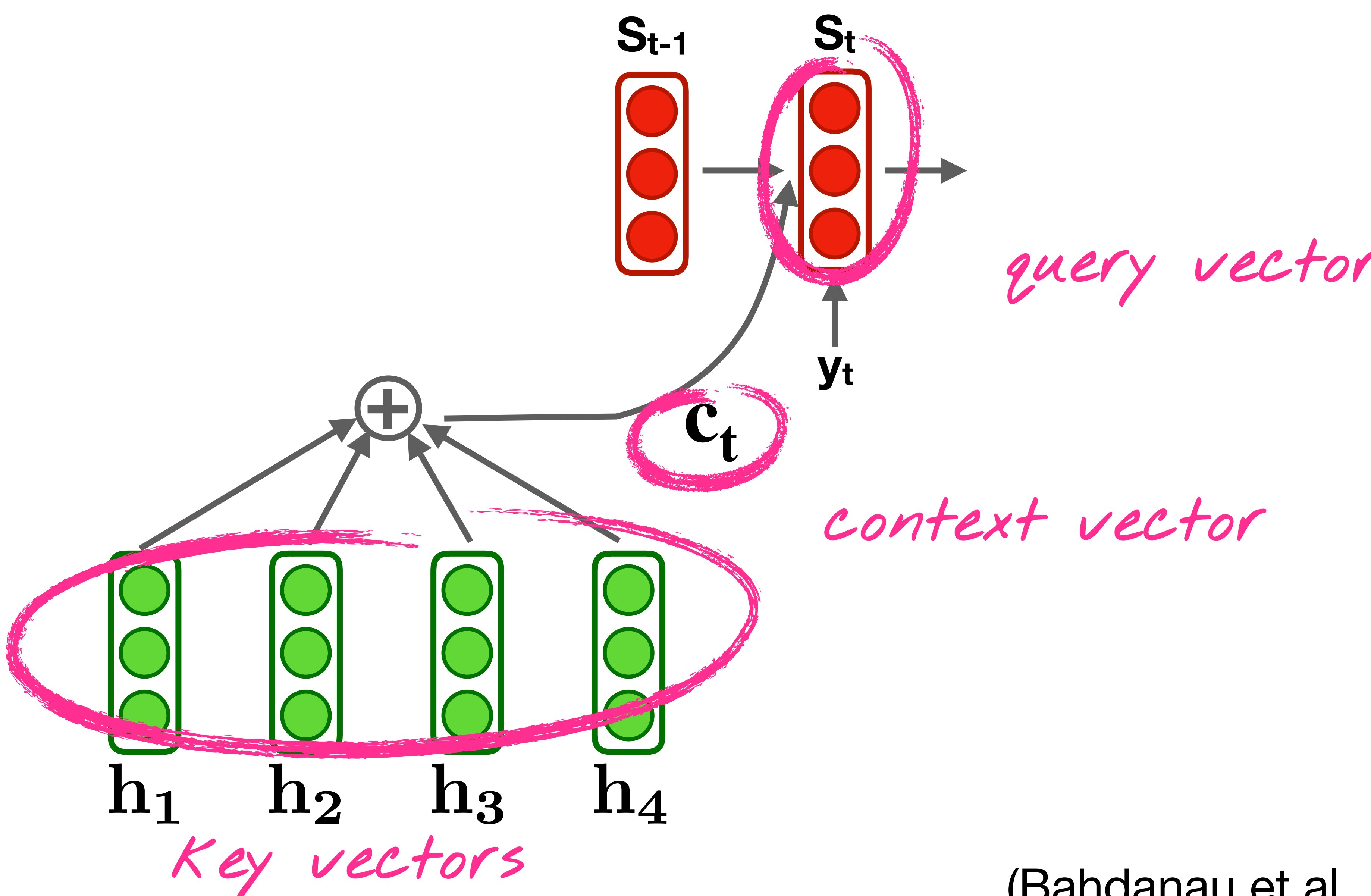
Beyond a single static "crammed" vector

- What if we could use several vectors, based on the length of the input sequence?
- Idea: when we generate the next word in MT, perhaps we can learn to **attend** to the **relevant** source words



Attention: Core Idea

- When decoding, perform a linear combination of the encoded input vectors, weighted by “attention weights”

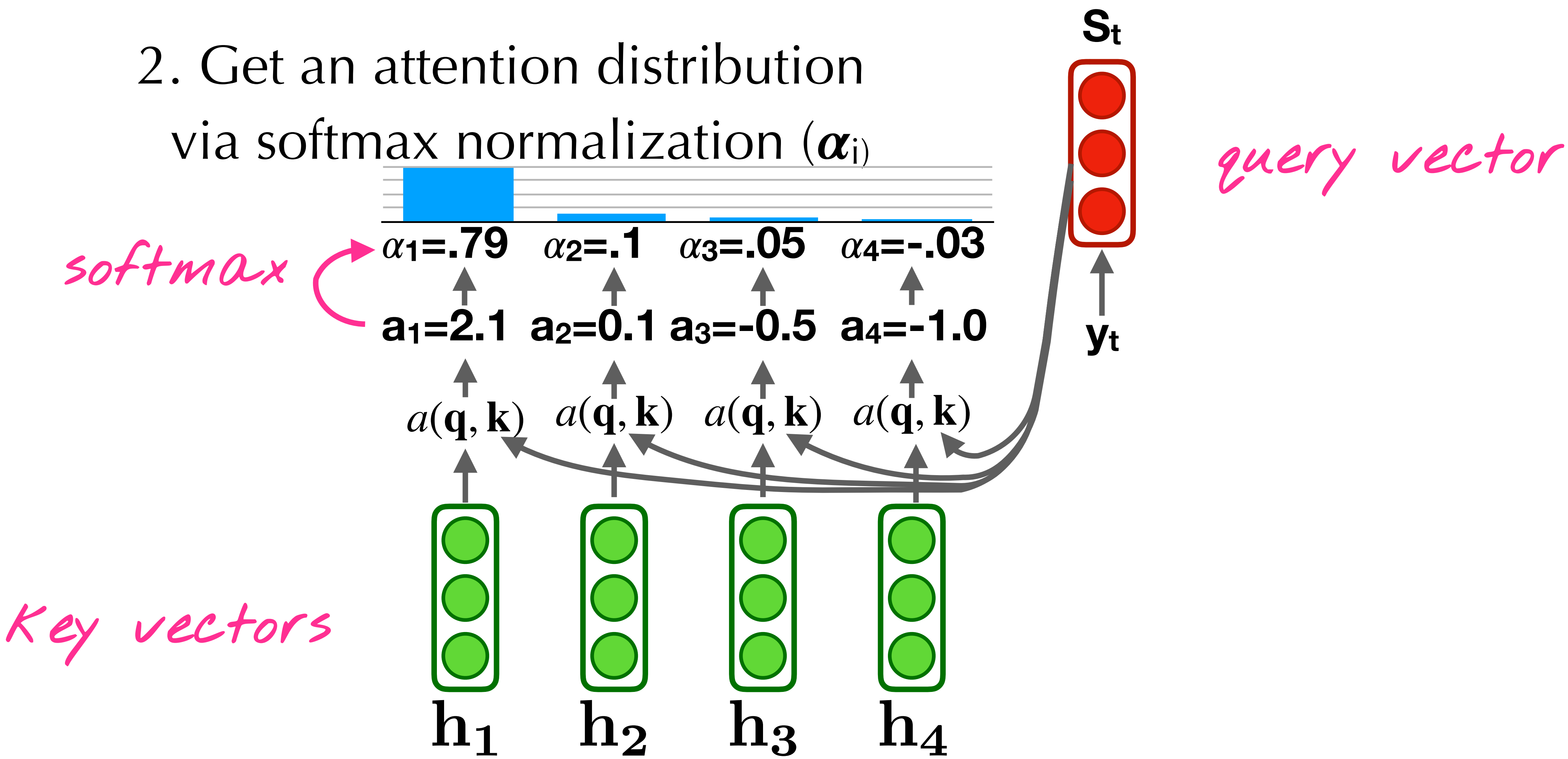


(Bahdanau et al., 2015)

Calculating attention (1/2): Attention weights α

1. For each query-key pair, calculate an **attention score** (a_i)

2. Get an attention distribution
   via softmax normalization ($\boldsymbol{\alpha}_i$)

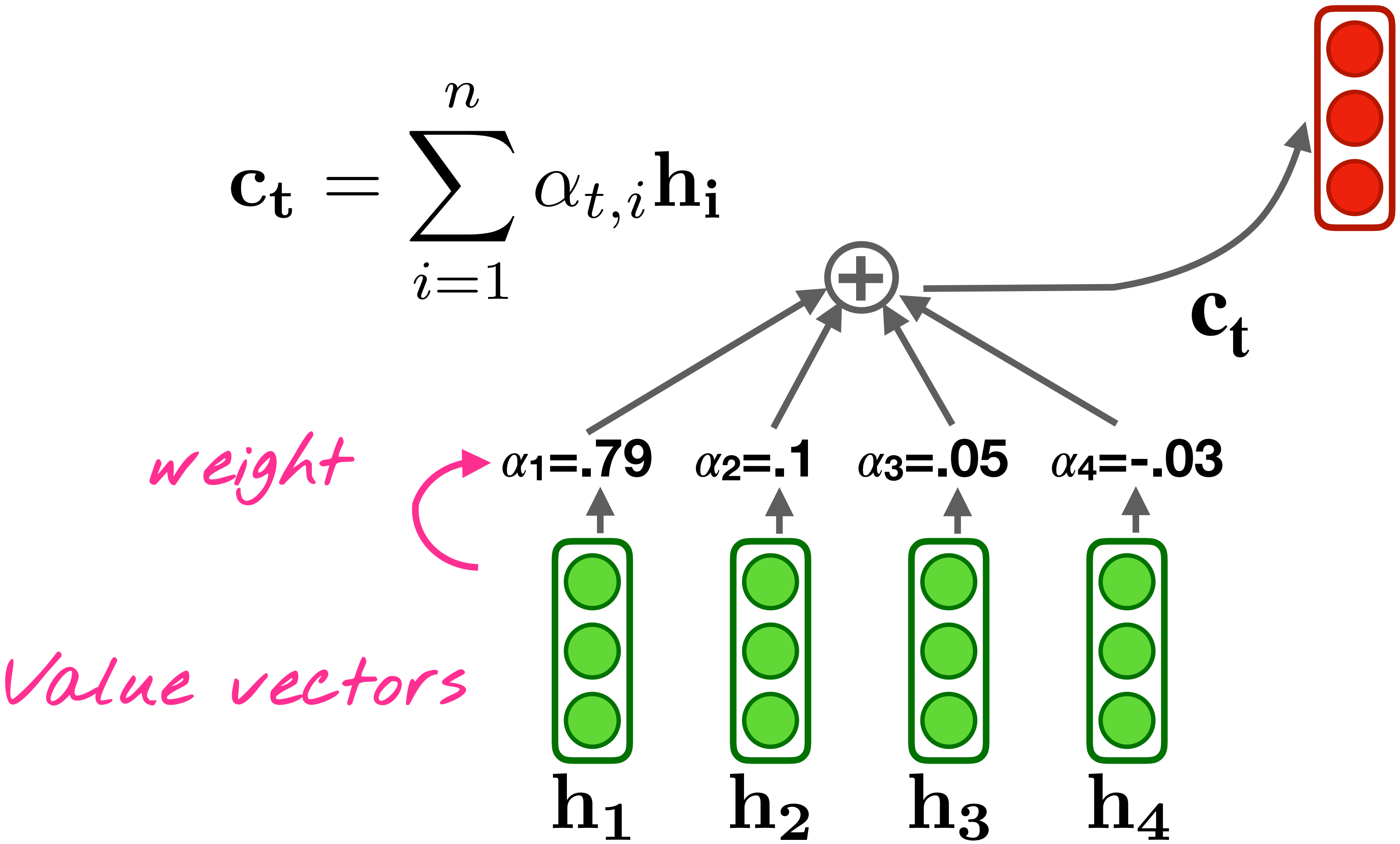


(Bahdanau et al., 2015)

Calculating attention (1/2): Attention weights α

3. Combine together **value vectors** (can be the encoder states, like the key vectors) by taking the weighted sum to get **c**

$$\mathbf{c_t} = \sum_{i=1}^{n} \alpha_{t,i} \mathbf{h_i}$$



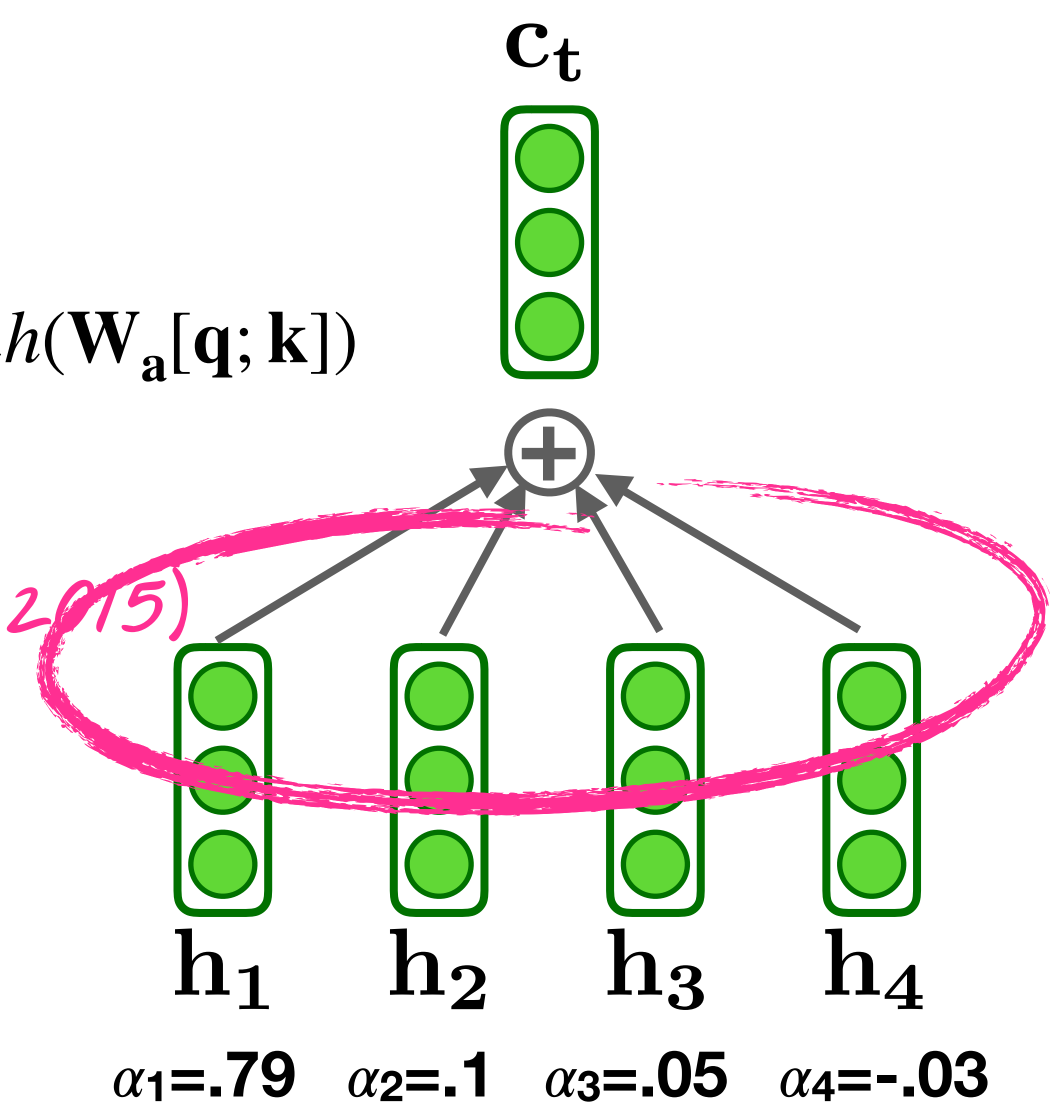
(Bahdanau et al., 2015)

Summary: Additive attention (Bahdanau, 2015)

$$\mathbf{c_t} = \sum_{i=1}^{n} \alpha_{t,i} \mathbf{h_i}$$

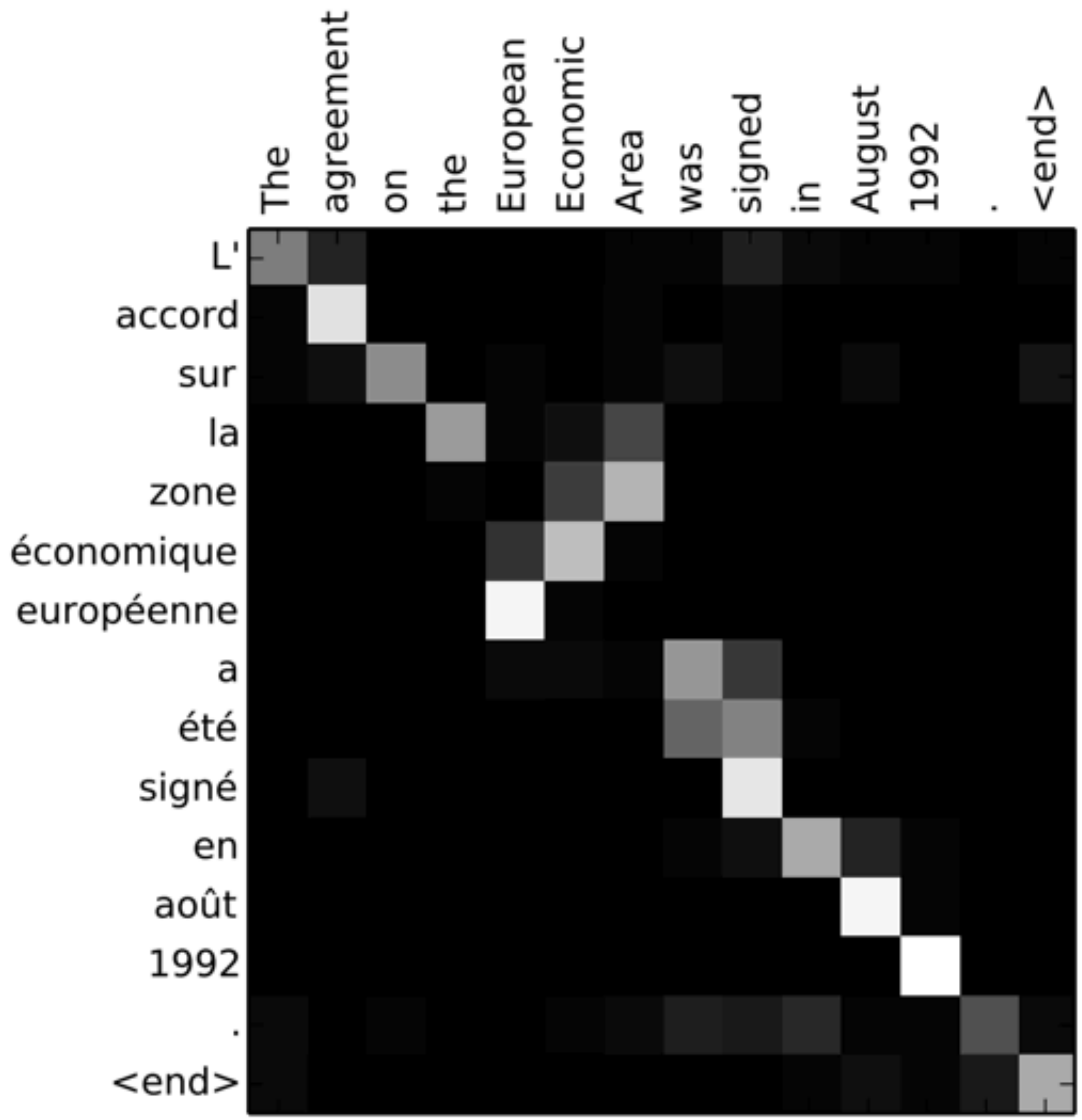
$$a(\mathbf{q}, \mathbf{k}) = \mathbf{v_a^T} \mathit{tanh}(\mathbf{W_a}[\mathbf{q}; \mathbf{k}])$$

FFNN!
(Bahdanau, 2015)



1. For each query-key pair, calculate weight a_i
2. Normalize via softmax
3. Combine together value vectors via weighted sum to get $\mathbf{c_t}$
4. Use in your model in any part you like

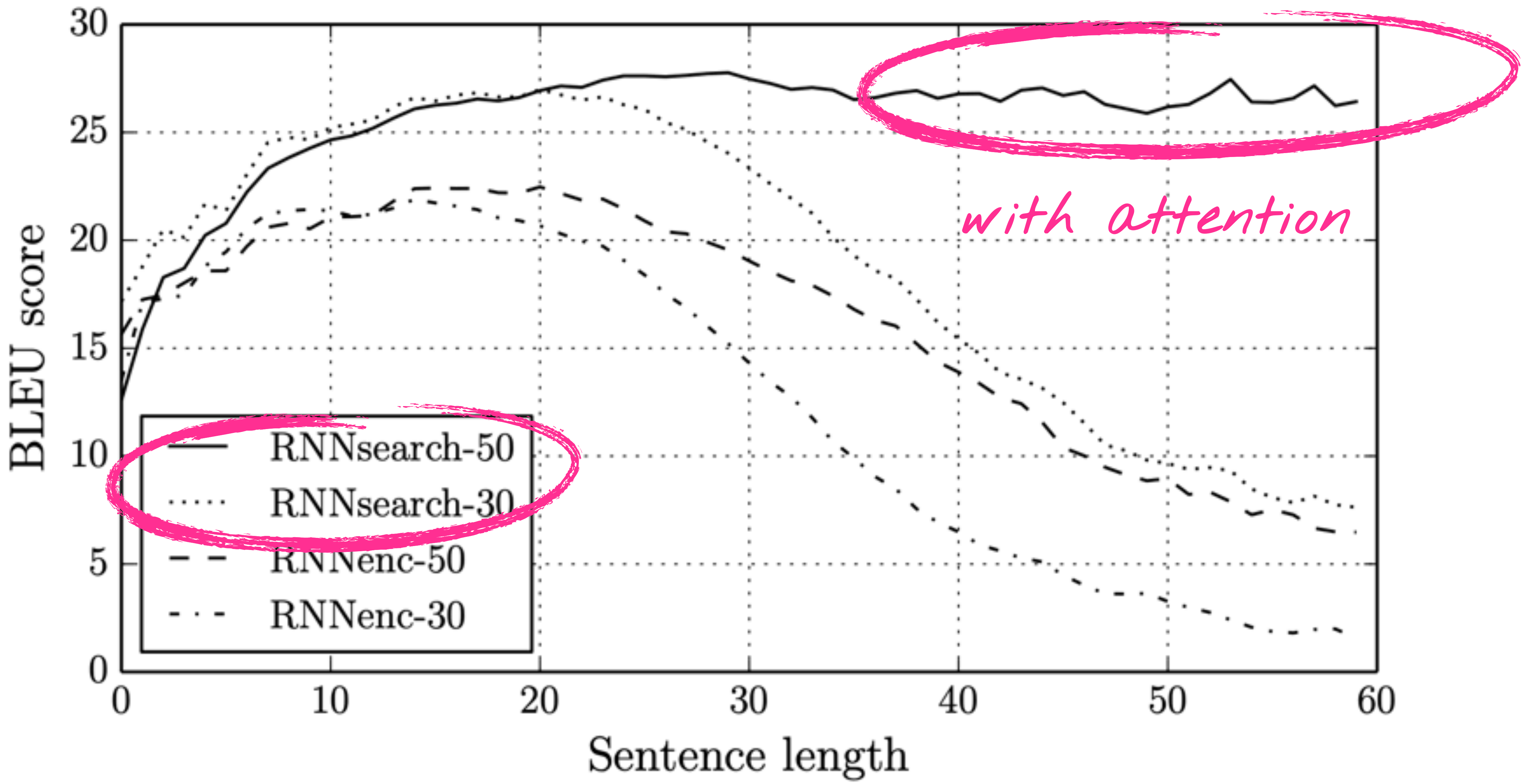
Alignment matrix (Bahdanau, 2015)



https://arxiv.org/pdf/1409.0473.pdf

Enc-dec performance deteriorates rapidly as input sentence length increases

Cho et al., (2014); Bahdanau et al. (2015)



https://arxiv.org/pdf/1409.0473.pdf

Different forms of attention are available (e.g., Luong et al., 2015)

Alignment Functions: What's $a(\mathbf{q}, \mathbf{k})$?

- In **Bahdanau** et al., (2015): the alignment score function is a single **FFNN** (MLP) with a single hidden layer:
  - $a(\mathbf{q}, \mathbf{k}) = \mathbf{v_a^T} tanh(\mathbf{W_a}[\mathbf{q}; \mathbf{k}])$
  - both v_a and W_a are trained with the network

More alignment functions

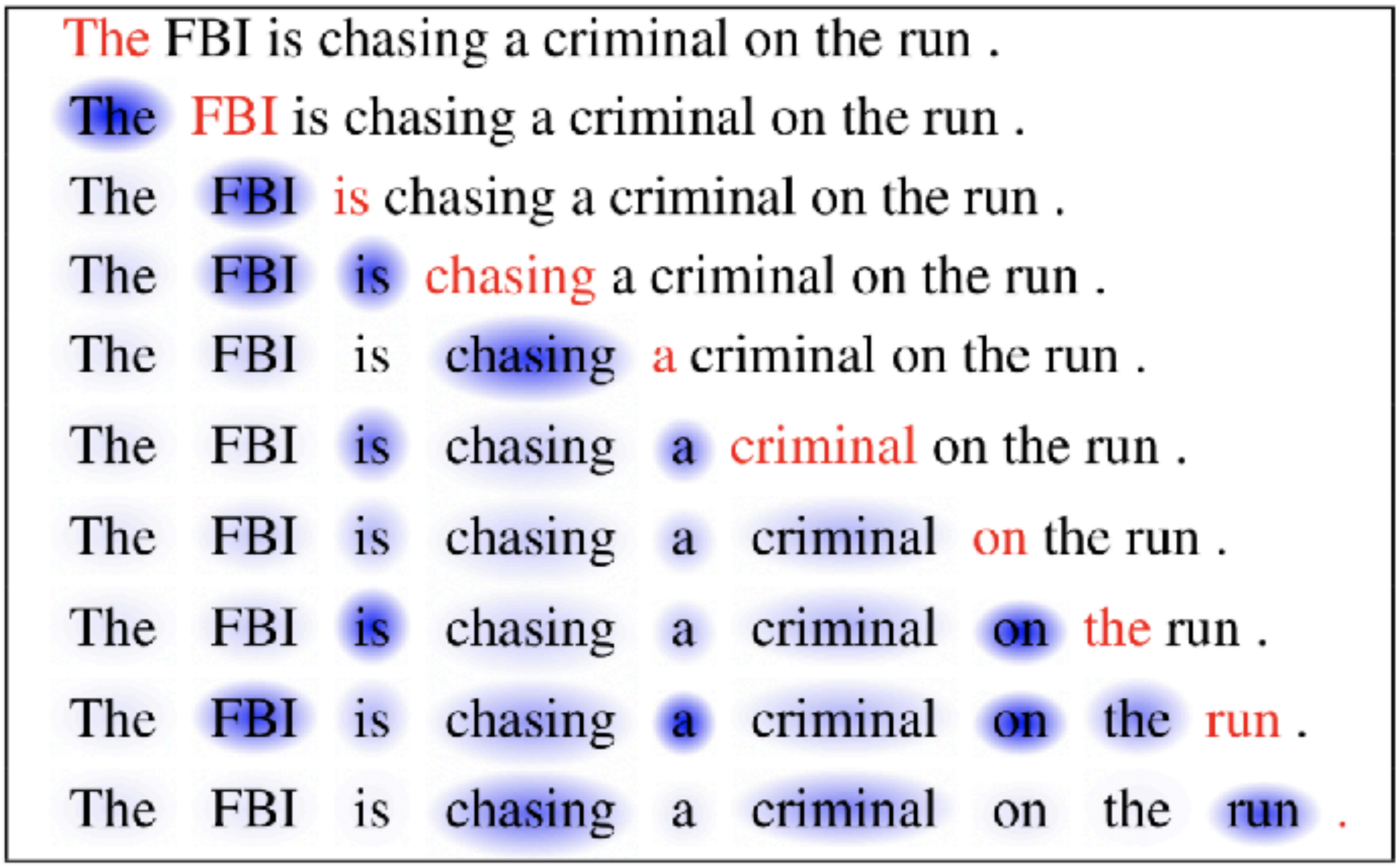
- **Dot product** (Luong et al., 2015)
  - $a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T \mathbf{k}$
  - requires same size; but has no parameters!
- **Bilinear** (Luong et al., 2015)
  - $a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T W \mathbf{k}$
- **Scaled dot product** (Vaswani et al., 2017)
  - $a(\mathbf{q}, \mathbf{k}) = \dfrac{\mathbf{q}^T \mathbf{k}}{\sqrt{|\mathbf{k}|}}$
  - fixes problem of dot product that scale of dot product increases as dimensions get larger

https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

A little more on attention

Self-attention

- Attend to sentence itself (Cheng, Dong, Lapata, 2016)



https://arxiv.org/pdf/1601.06733.pdf

What to attend to?
Some more examples

Image caption generation

- Salient parts of the image (e.g., Xu et at., 2015)

A woman is throwing a frisbee in a park.

Character-level attention

- E.g. Rei et al., 2016

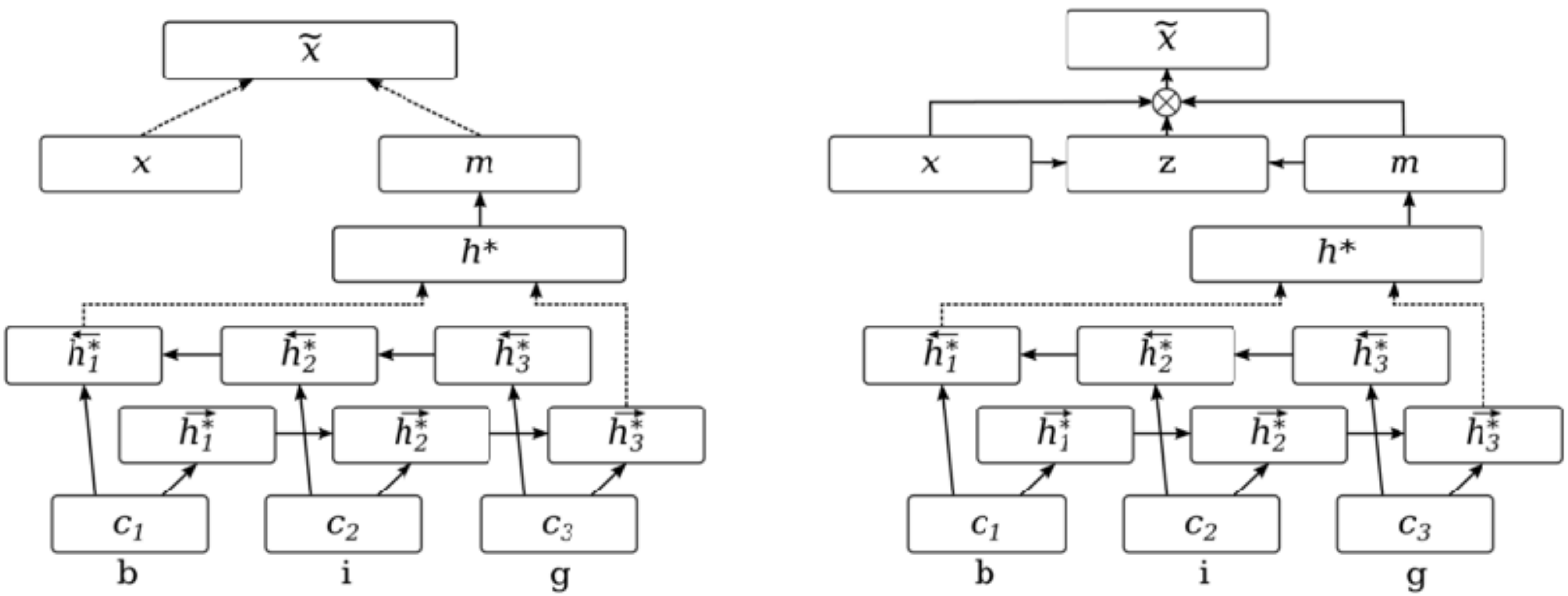


Figure 2: Left: concatenation-based character architecture. Right: attention-based character architecture. The dotted lines indicate vector concatenation.

Attention is everywhere
[and all you need?!] ->
More on Monday :-)

To summarize

To sum up…

foundations

Motivation,
Brief History, Overview

Back to the roots:
Language Models

n-grams, limitations

Feedforward NNs
(FFNNs)

FFNN LM

representations

What's the input?
Representations

n-hot & static
word embeddings

ELMo

Contextualized
Embeddings, Fine-tuning

beyond FFNNs

Convolutional Neural
Networks (CNNs)

variable-size input
& "soft" ngrams

vanilla RNNs

Recurrent Neural
Network (RNNs)

GRU, LSTM

Advanced RNNs,
Decoders

MLP, CRF

Basis for Transformer

Attention? Attention!

IT UNIVERSITY OF COPENHAGEN

Questions?
Thanks!

Barbara Plank
@barbara_plank
ITU, Denmark

Follow us:

EurNLP

Thanks to all the
organizers & sponsors of:

AdNLP

bplank.github.io
https://nlp.itu.dk/

Research is supported by:

NVIDIA amazon

References (incomplete)

- Jurasky & Martin textbook, chapter 3 (n-gram LMs), chapter 7 (neural LMs)
- Graham Neubig (2018): Language Models 4: Recurrent Neural Network Language Models
- Yoav Goldberg (2015): A Primer on Neural Network Models for Natural Language Processing
- Chris Manning & Abigail See (2018) Stanford class