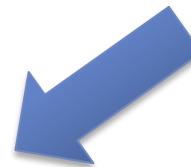


Named Entity Recognition (NER) with Deep Learning

Shobeir Fakhraei
University of Southern California

Slides based on material provided by many people!

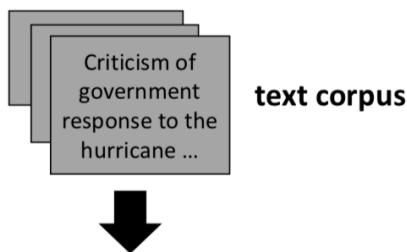
Where are we?



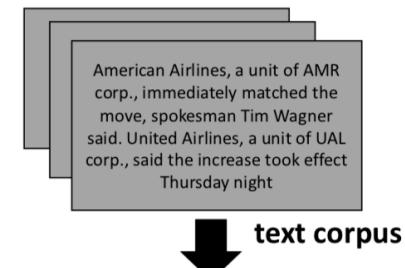
- Information Extraction
 - Semi-structured: Web, HTML
 - **Unstructured: Text, Ads, Tweets, ...**
- Entity Linkage, Data Cleaning, Normalization
- Logical Data Integration
 - Mediators, Query Rewriting
 - Warehouse, Logical Data Exchange
- Automatic Source Modeling/Learning Schema Mappings
- Semantic Web
 - RDF, SPARQL, OWL, Linked Data
- Advanced Topics
 - Geospatial Data Integration, Knowledge Graphs

Landscape of IE Tasks (1): Types of Extractions

Named Entity Recognition



Relationship Extraction



NER Approaches

1. Rule/Pattern-Based

2. Standard Classifiers

KNN, decision tree, naïve Bayes, SVM, ...

3. Sequence Models

HMMs, CRFs, LSTM-CRF

NER Approaches

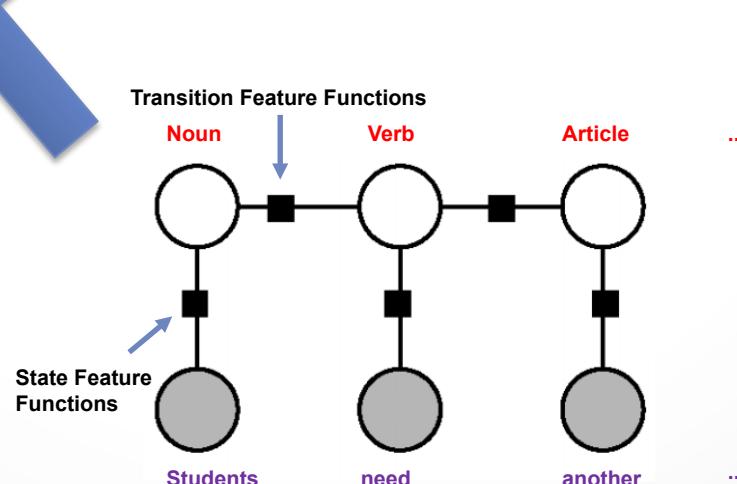
1. Rule/Pattern-Based

2. Standard Classifiers

KNN, decision tree, naïve Bayes, SVM, ...

3. Sequence Models

HMMs, CRFs, LSTM-CRF



NER Approaches

1. Rule/Pattern-Based

2. Standard Classifiers

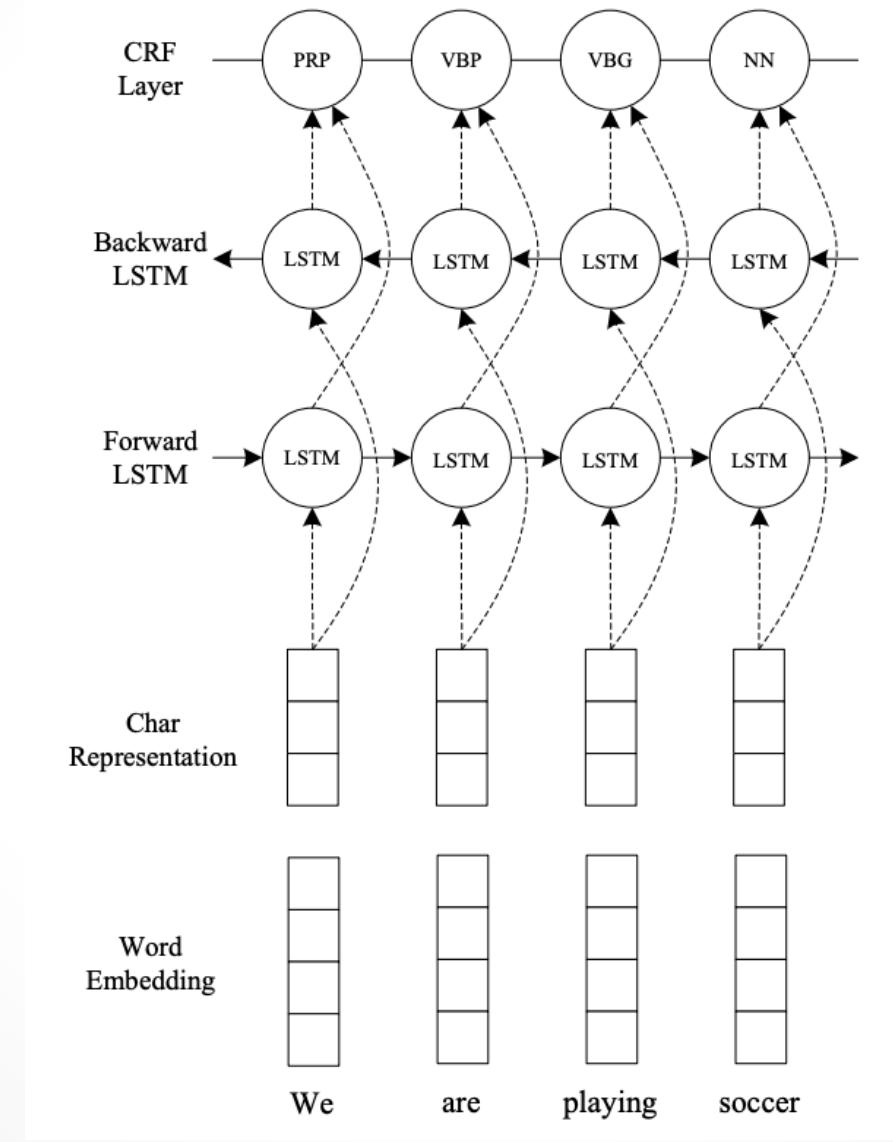
KNN, decision tree, naïve Bayes, SVM, ...

3. Sequence Models

HMMs, CRFs, **LSTM-CRF**



Bi-directional LSTM-CNN-CRF



Outline

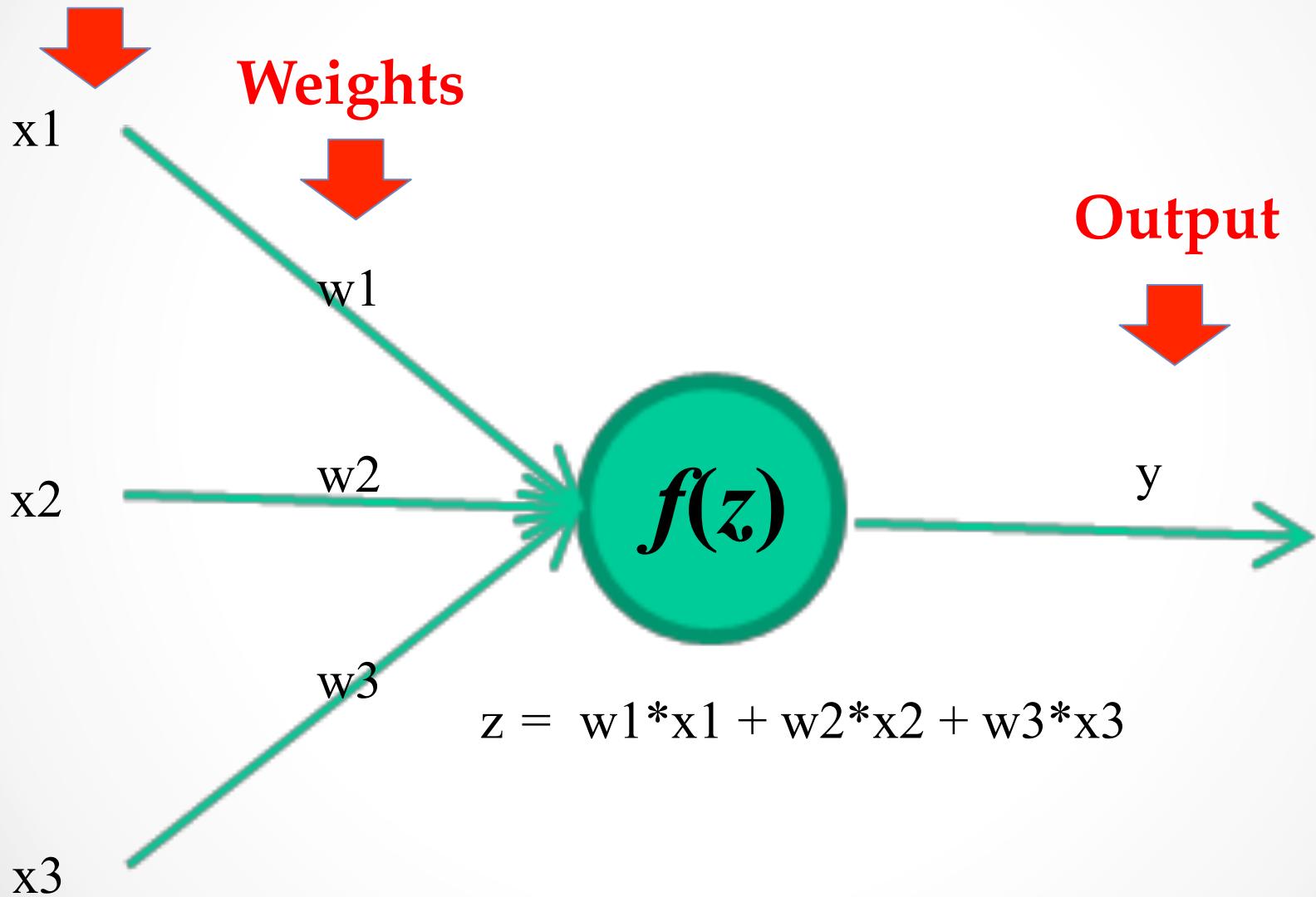
- Deep Learning Background
 - Activation functions, Back propagation, Dropout, CNN
- Sequence Models
 - Recurrent Neural Networks (RNN)
 - Long Short Term Memory (LSTM)
- Word and Character Embedding
 - Word2Vec, GloV
- NER Models with Deep Learning
 - CNN-BiLSTM-CRF



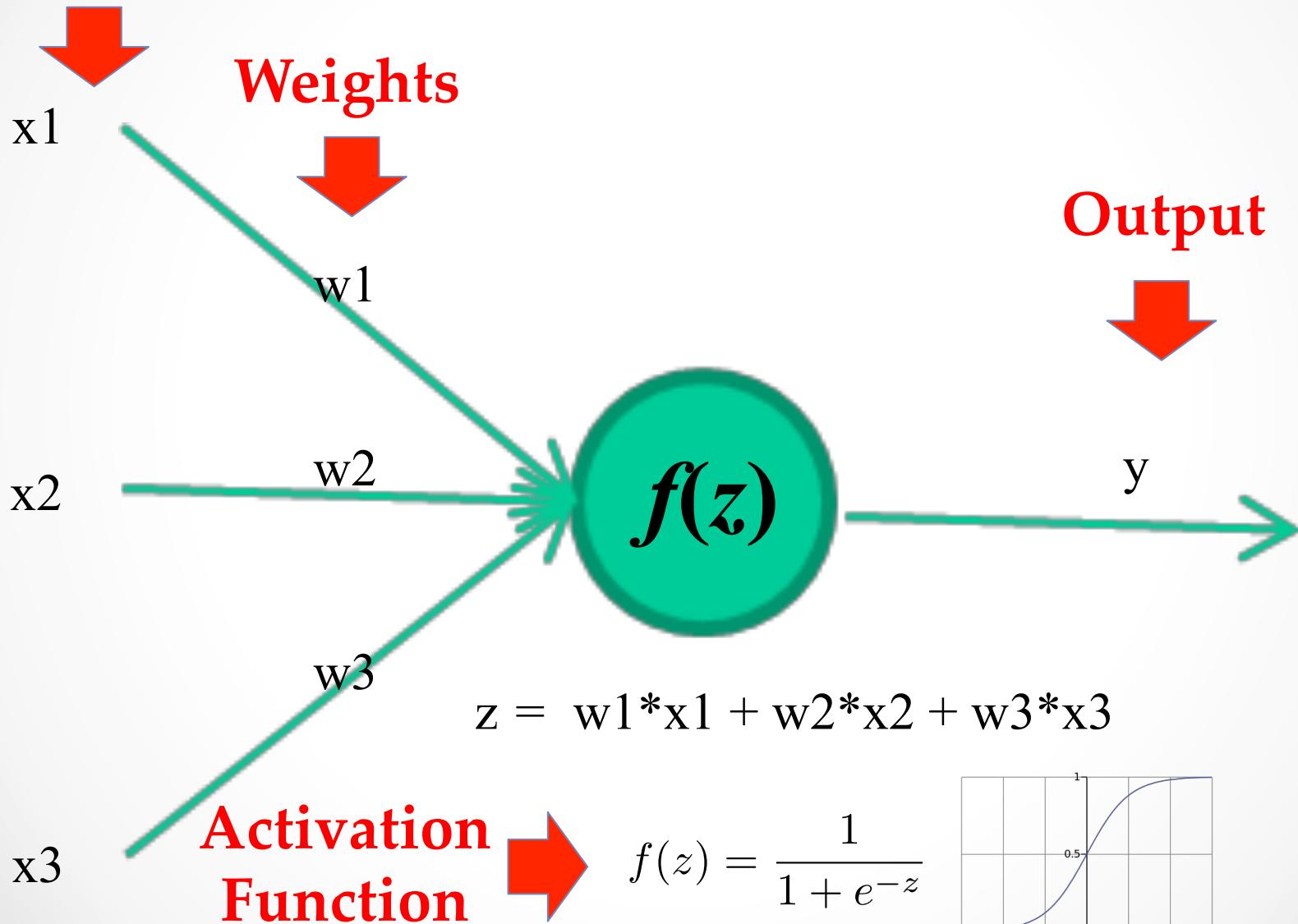
Background: Neural Networks

...

Inputs A neuron is just function!



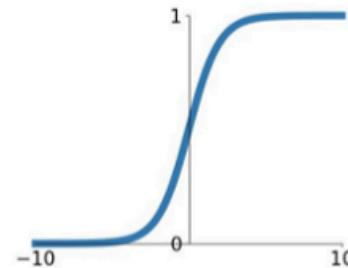
Inputs A neuron is just function!



Activation Functions

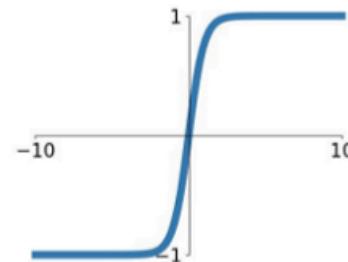
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



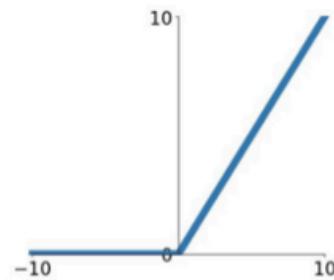
tanh

$$\frac{e^x - e^{-x}}{e^x + e^{-x}}$$

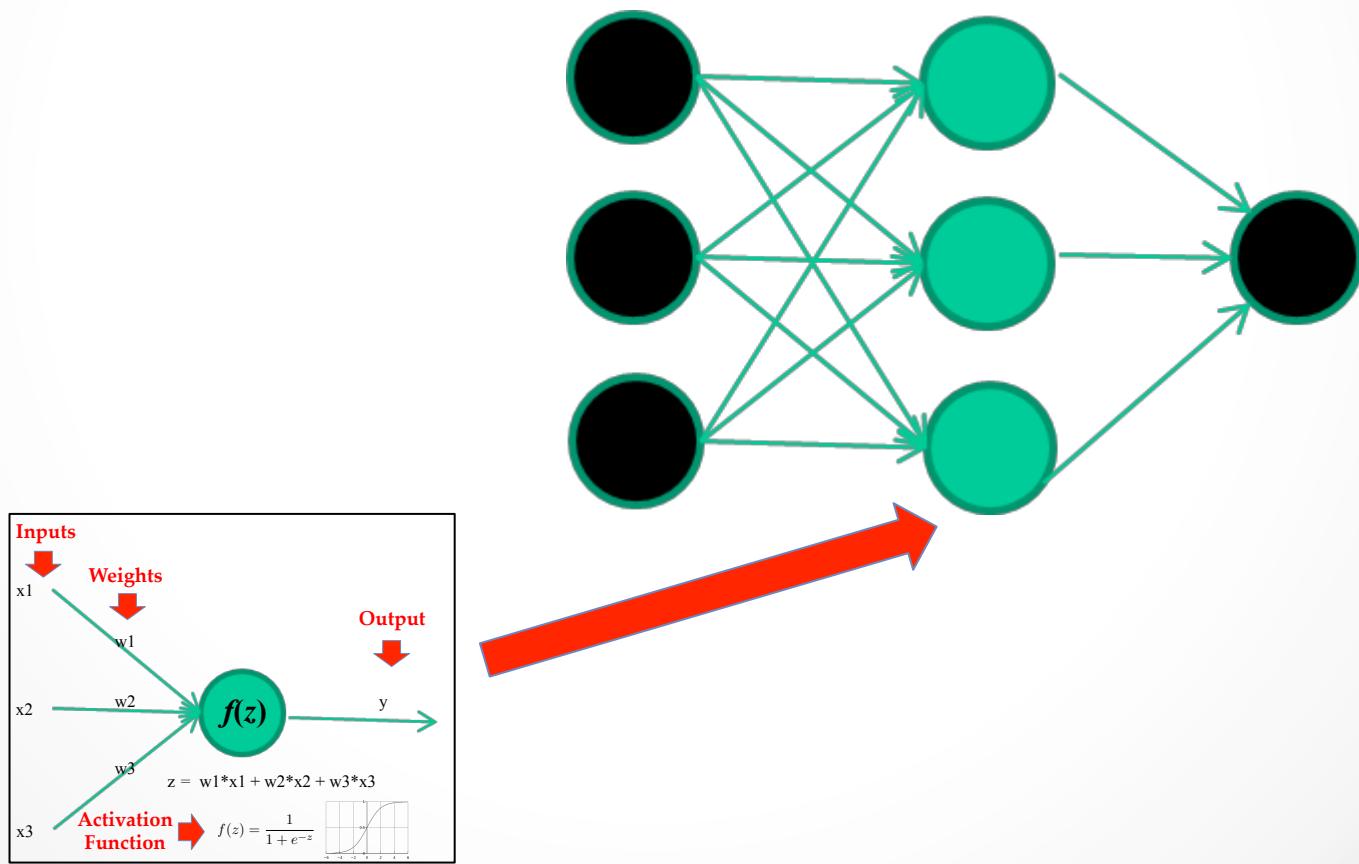


ReLU

$$\max(0, x)$$



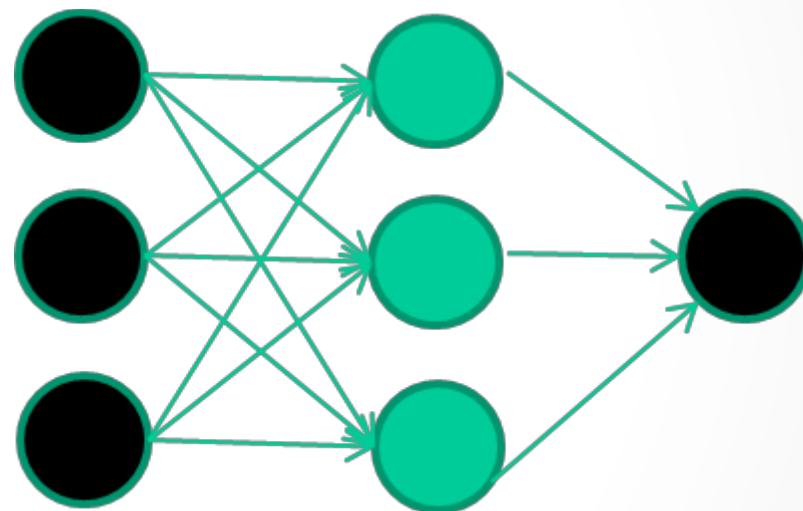
Artificial Neural Network



Training ANN

A dataset

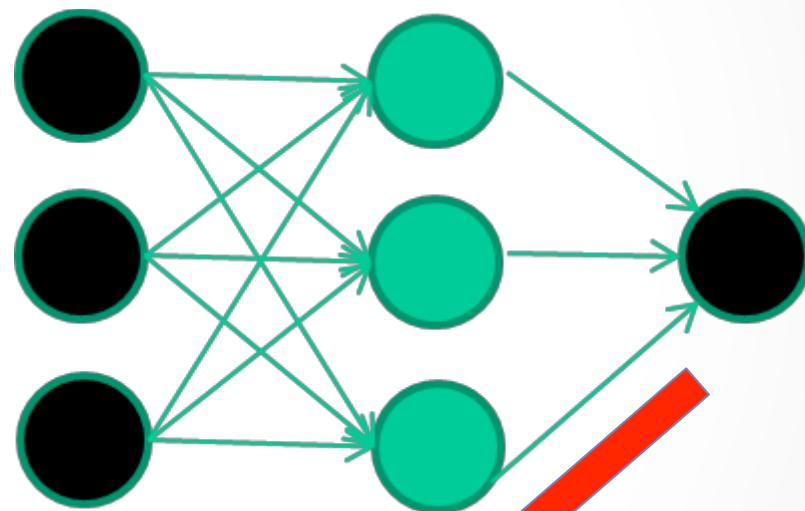
<i>Fields</i>	<i>class</i>
1.4 2.7 1.9	0
3.8 3.4 3.2	0
6.4 2.8 1.7	1
4.1 0.1 0.2	0
etc ...	



Training ANN

A dataset

<i>Fields</i>	<i>class</i>
1.4 2.7 1.9	0
3.8 3.4 3.2	0
6.4 2.8 1.7	1
4.1 0.1 0.2	0
etc ...	



**Cost
Function**

$$C = \frac{1}{n} \sum (y_t - y_p)^2$$

Training ANN

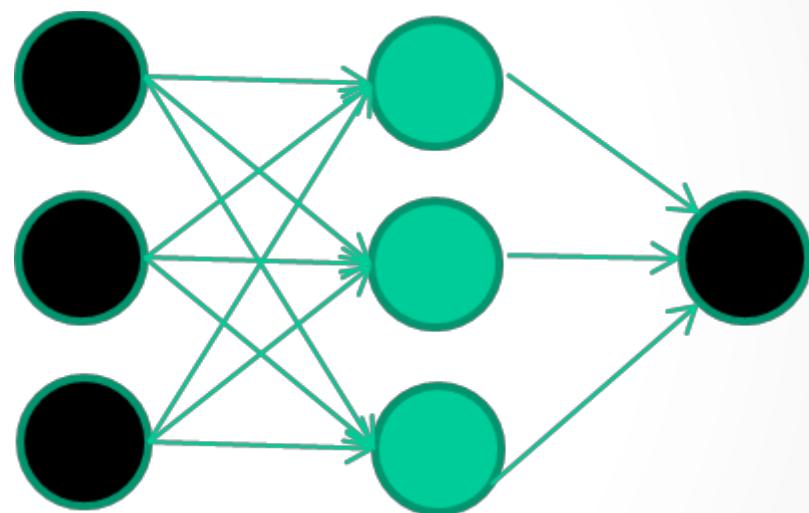
Training data

Fields

	<i>class</i>		
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc ...

Initialise with random weights

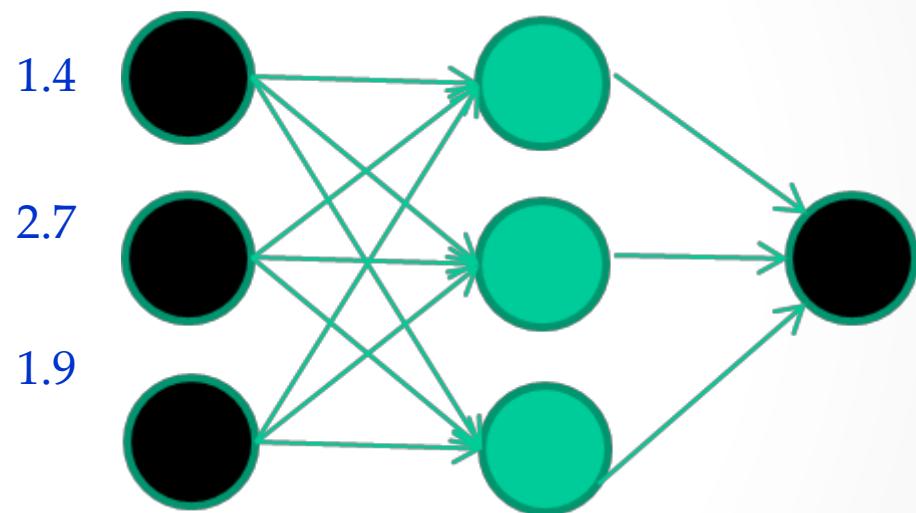


Training ANN

Training data

<i>Fields</i>	<i>class</i>		
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc ...			

Input a training instance

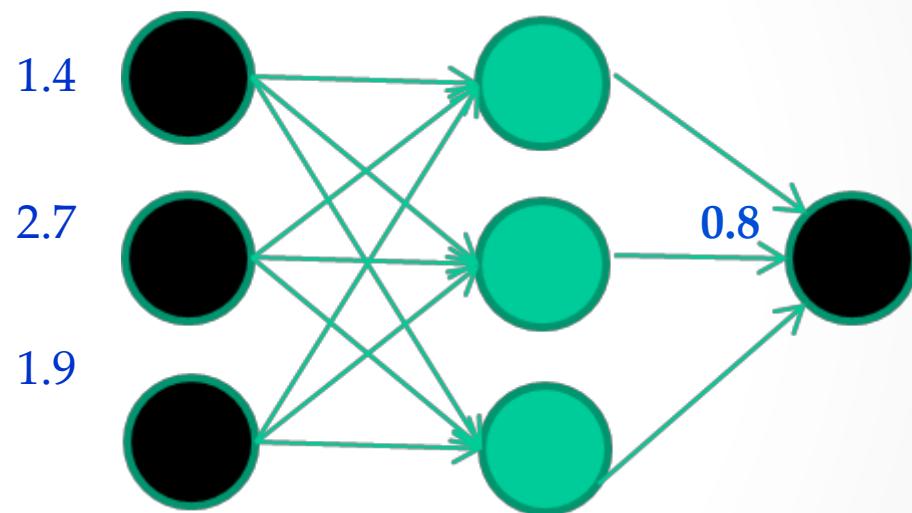


Training ANN

Training data

<i>Fields</i>	<i>class</i>		
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc ...			

Feed it through to get output

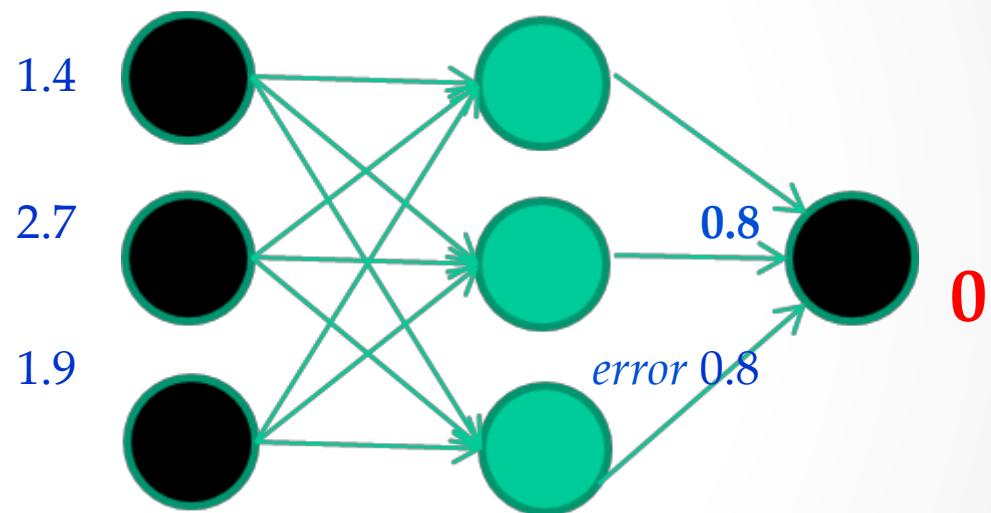


Training ANN

Training data

Fields	class		
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc ...			

Compare with target output

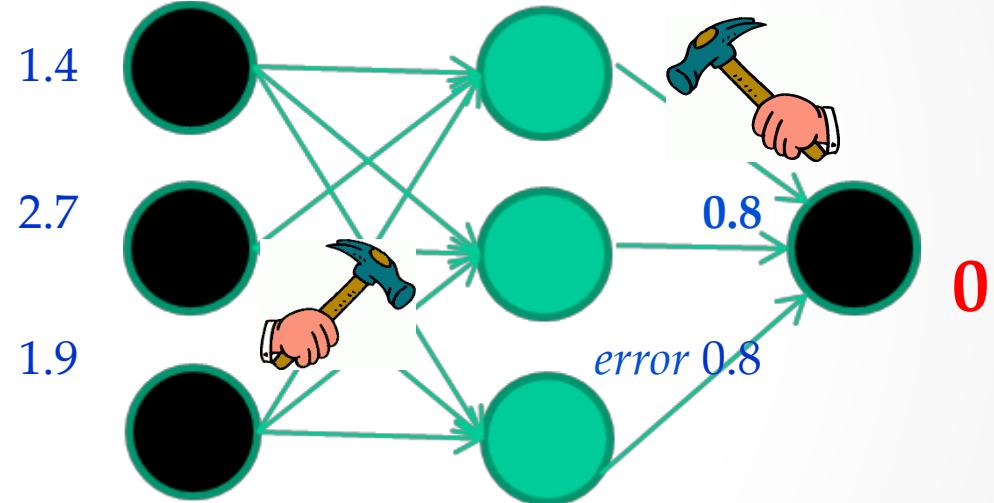


Training ANN

Training data

<i>Fields</i>	<i>class</i>		
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc ...			

Adjust weights based on error



Training ANN

Training data

Fields *class*

1.4 2.7 1.9 0

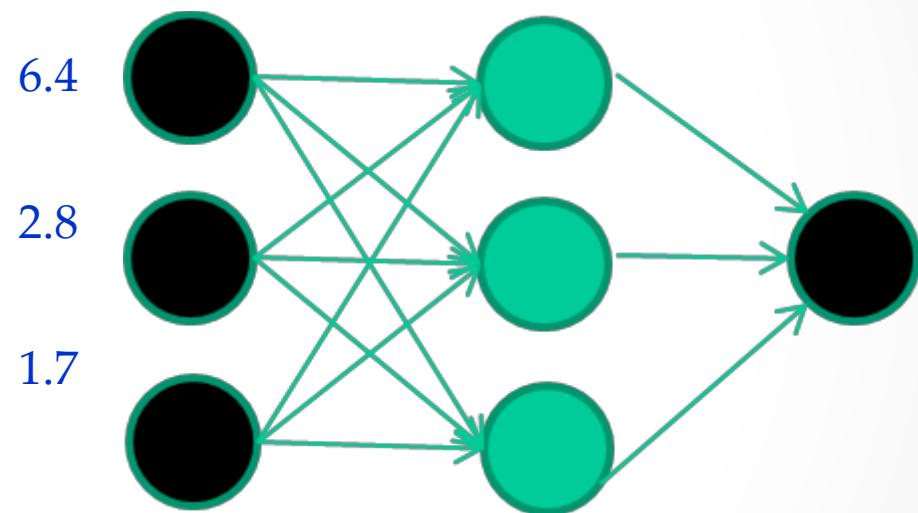
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...

Input a training example



Training ANN

Training data

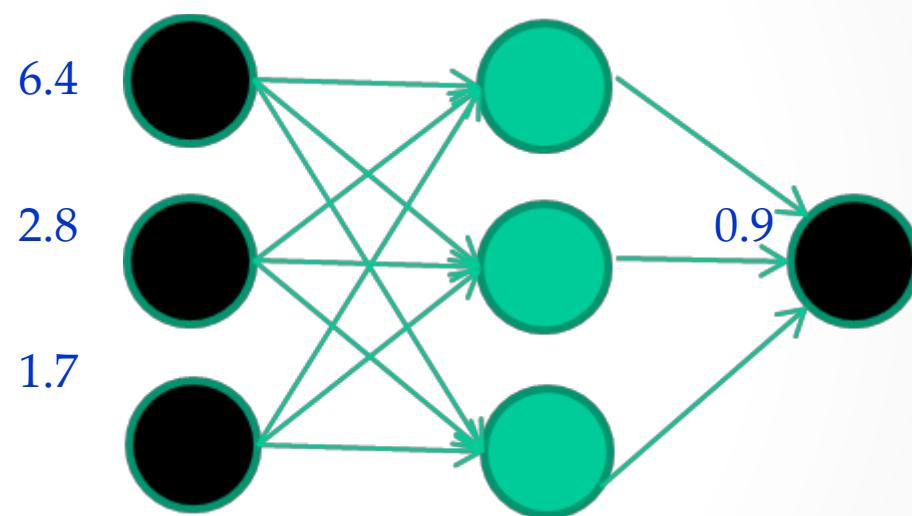
Fields

class

1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc ...

Feed it through to get output



Training ANN

Training data

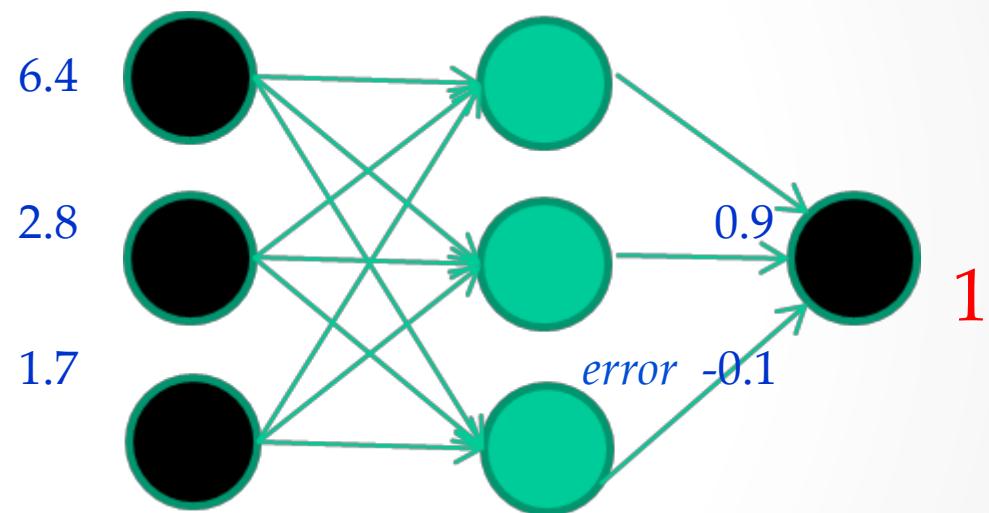
Fields

class

1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc ...

Compare with target output



Training ANN

Training data

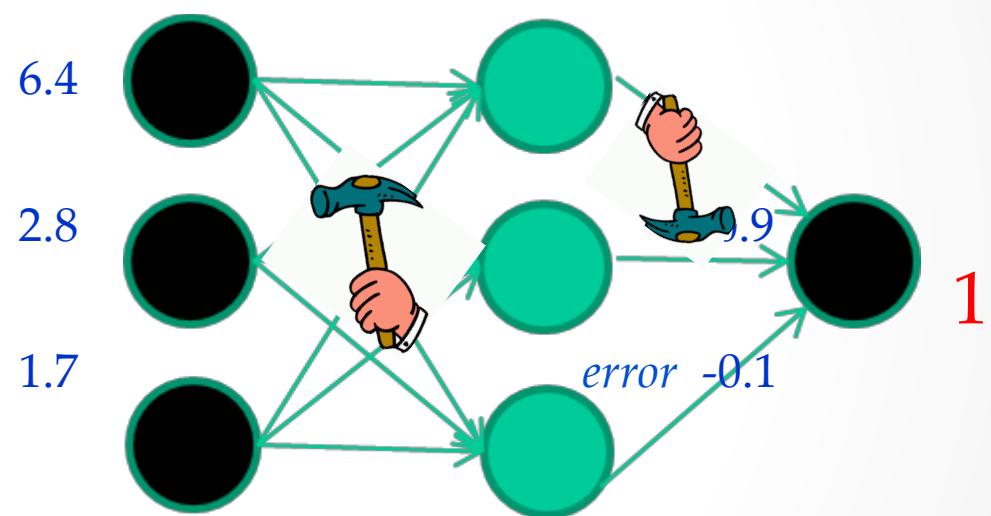
Fields

class

1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc ...

Adjust weights based on error



Training ANN

Training data

Fields *class*

1.4 2.7 1.9 0

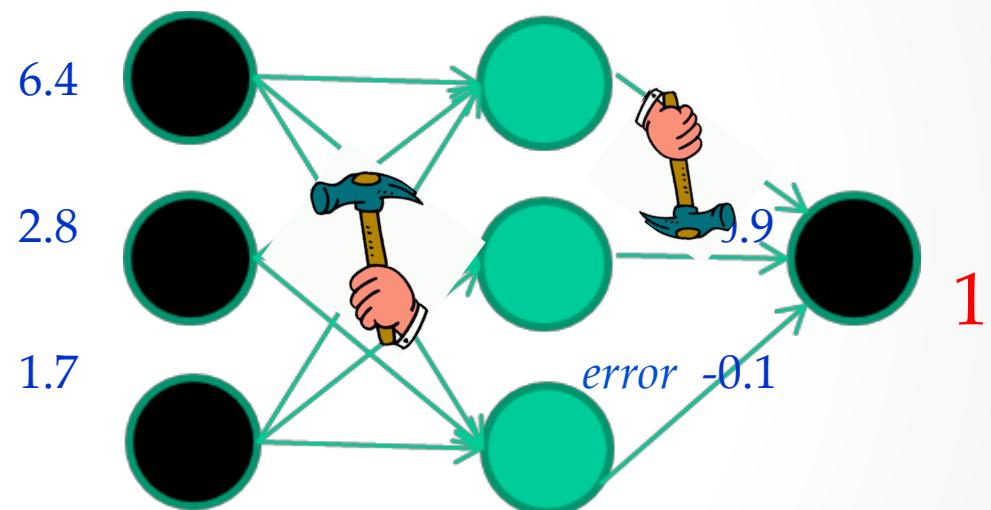
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

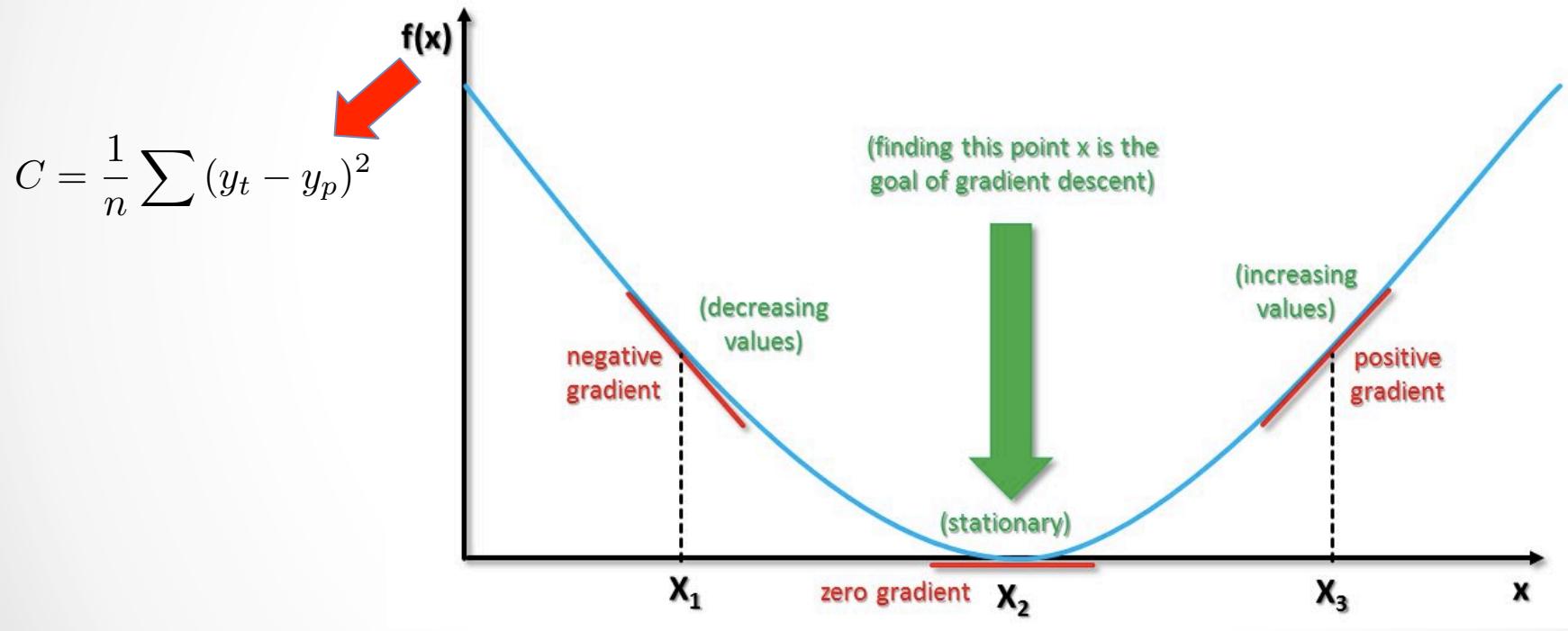
etc ...

And so on



Repeat this thousands of times – each time
taking a random training instance, and making slight
weight adjustments

Gradient Descent



$$C = \frac{1}{n} \sum (y_t - y_p)^2$$

Weights (w)

Gradient Descent

(minimization: subtract gradient term because we move towards local minima)

$$b = a - \gamma \nabla f(a)$$

(old position before the step) (new position after the step)

(the derivative of f with respect to a)

(gradient term is steepest ascent)

(weighting factor known as step-size, can change at every iteration, also called learning rate)

$f(x)$

x

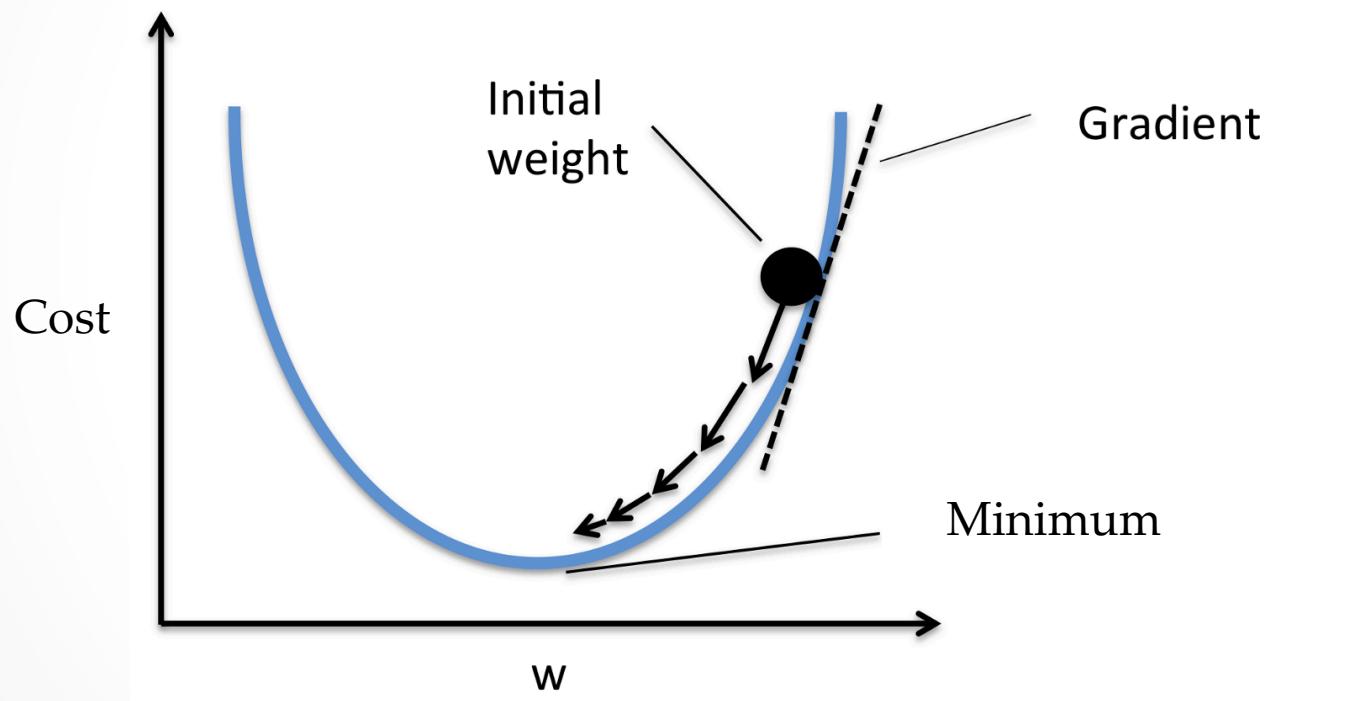
position a (current position)

(one step towards local minimum)

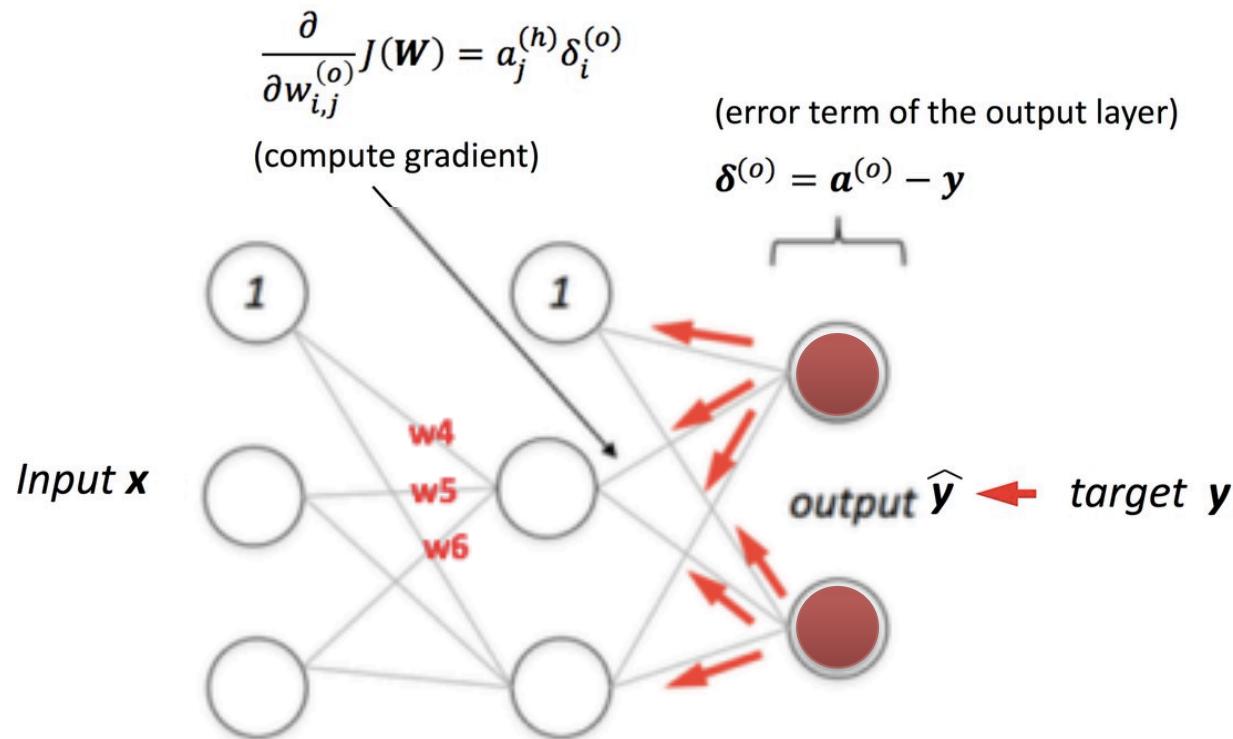
position b (next position)

x_1 x_2

Gradient Descent



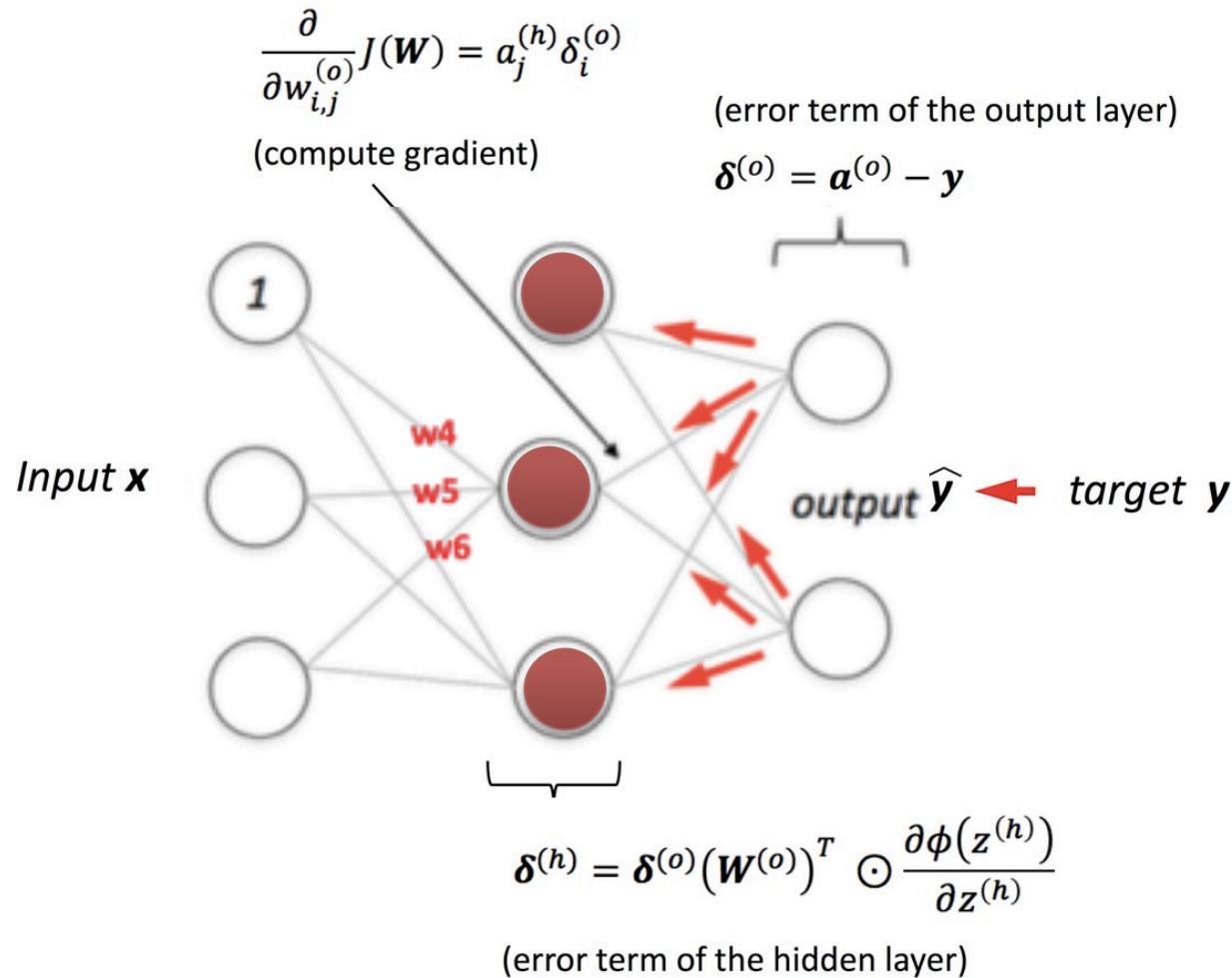
Back Propagation



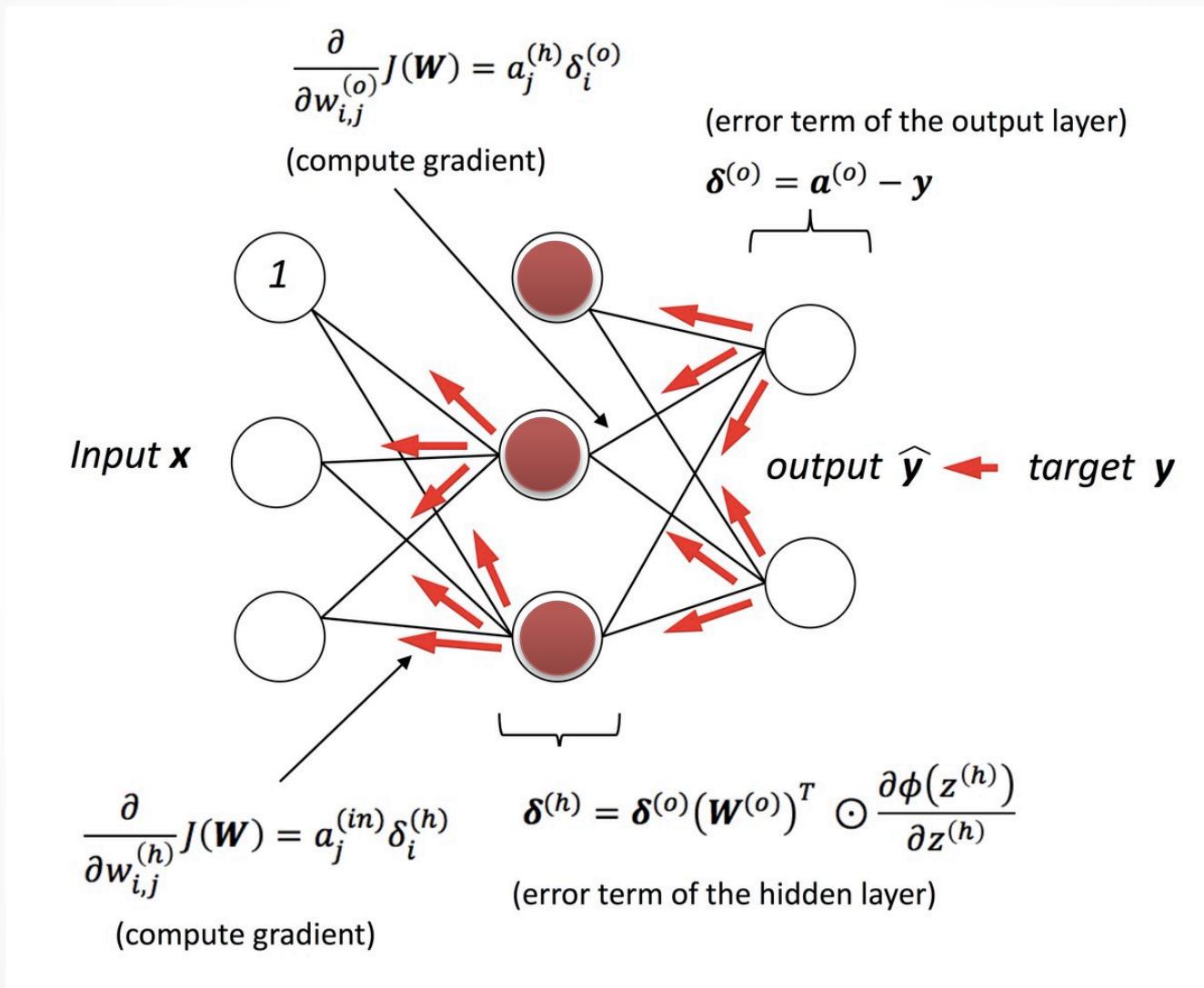
Demo:

<https://google-developers.appspot.com/machine-learning/crash-course/backprop-scroll/>

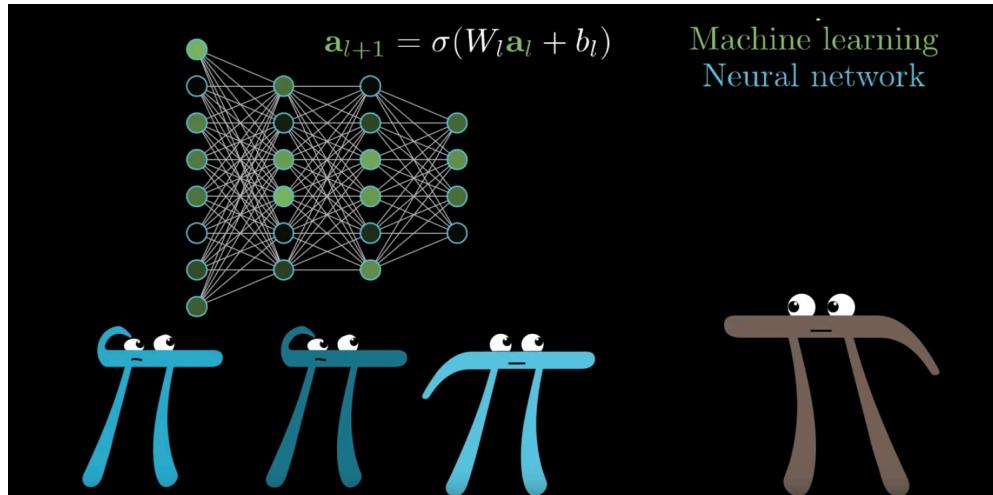
Back Propagation



Back Propagation



Watch the video series!



3BLUE1BROWN Deep Learning Video Series:

[https://www.youtube.com/watch?](https://www.youtube.com/watch?v=aircArUvnKk&list=PLZHQBObOWTQDNU6R1_67000Dx_ZCJB-3pi)

[v=aircArUvnKk&list=PLZHQBObOWTQDNU6R1_67000Dx_ZCJB-3pi](https://www.youtube.com/watch?v=aircArUvnKk&list=PLZHQBObOWTQDNU6R1_67000Dx_ZCJB-3pi)

One Perspective of Neural Networks History

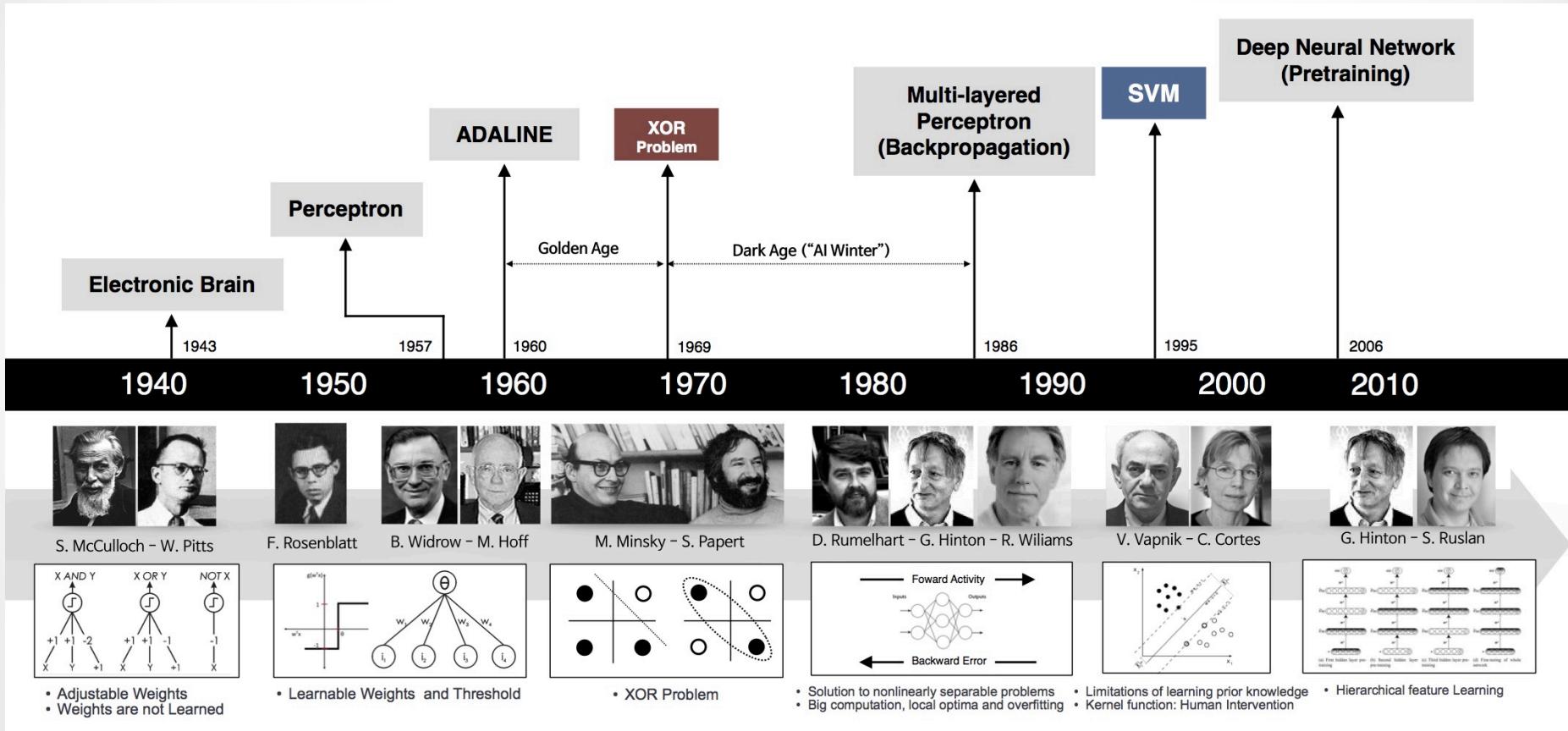


Figure by Andrew Beam

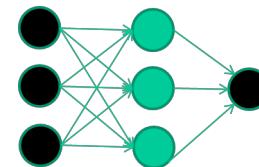
What's New?

**Multilayer neural networks have been around for many years.
What's actually new?**

What's New?

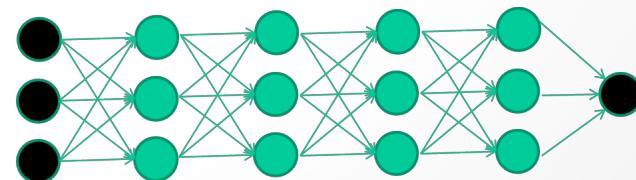
**Multilayer neural networks have been around for many years.
What's actually new?**

we have always had good algorithms for learning the weights in networks with 1 hidden layer

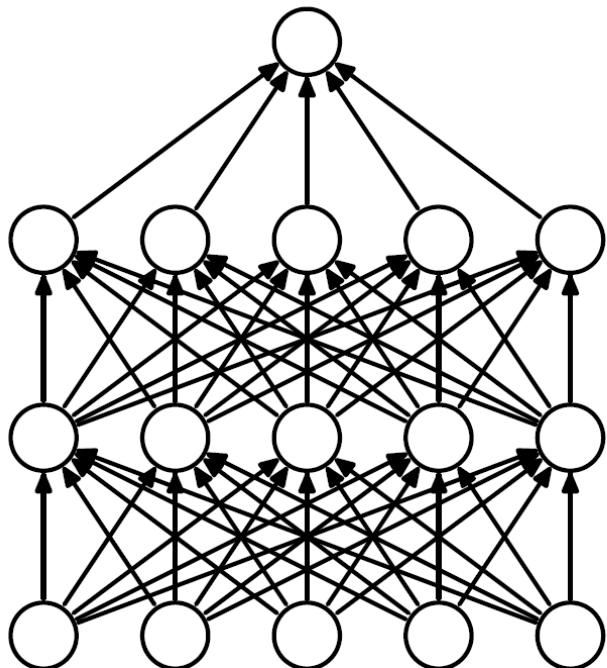


but these algorithms are not good at learning the weights for networks with more hidden layers

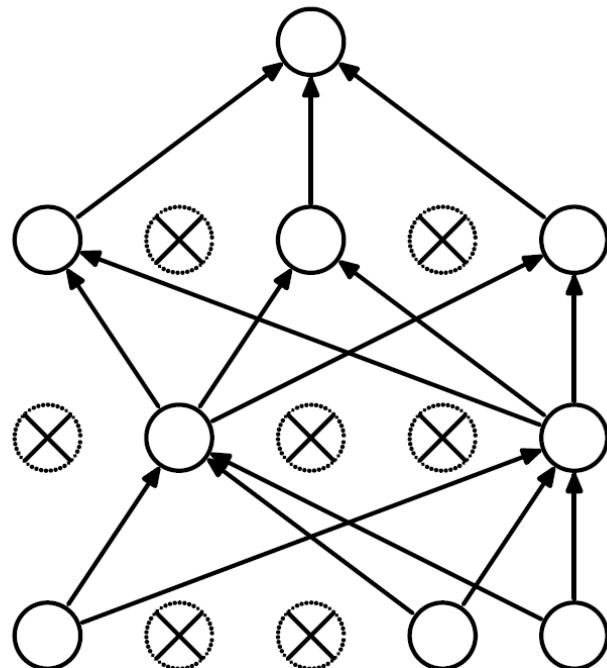
what's new is: algorithms for training many-layer (deep) networks



Dropout to avoid over-fitting



(a) Standard Neural Net



(b) After applying dropout.

Convolutional Neural Networks



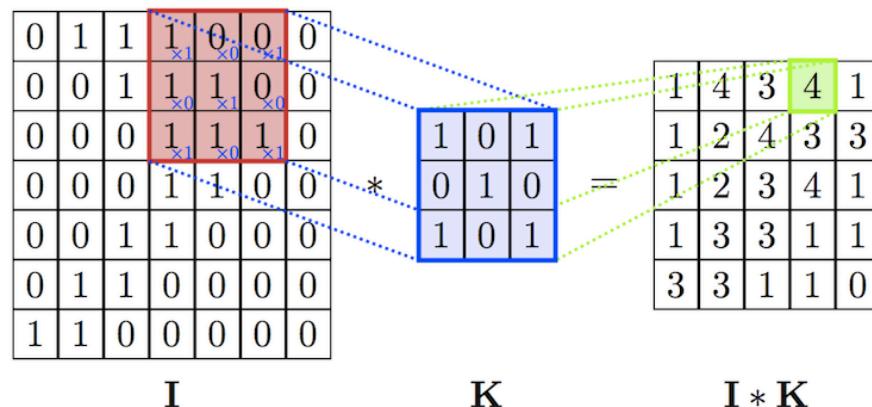
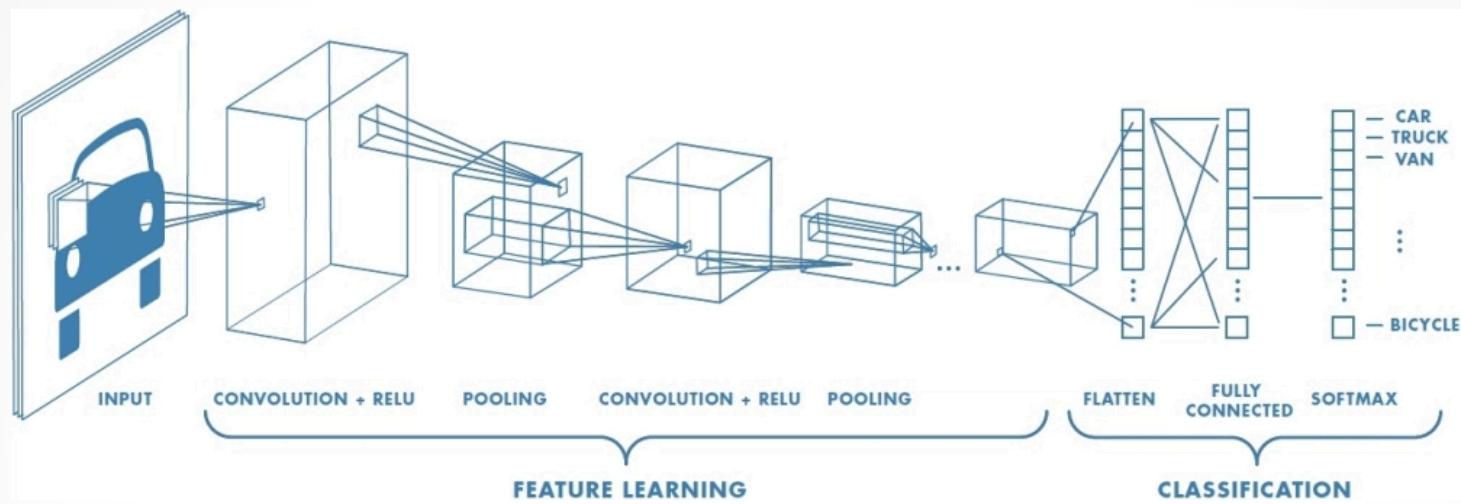
05	02	22	97	38	15	00	40	00	75	04	05	07	78	52	12	50	77	9	58
49	49	99	40	17	81	18	57	60	87	17	40	98	43	69	45	61	56	62	00
81	49	31	73	55	79	14	29	93	71	40	67	53	85	30	03	49	13	36	65
52	70	95	23	04	60	11	42	68	24	68	56	01	32	56	71	37	02	36	91
22	31	16	71	51	67	03	59	41	92	36	54	22	40	40	28	66	33	13	80
24	47	34	60	99	03	45	02	44	75	33	53	78	36	84	20	35	17	12	50
32	98	81	28	64	23	67	10	26	38	40	67	59	54	70	66	18	38	64	70
67	26	20	68	02	62	12	20	95	63	94	39	63	08	40	91	66	49	94	21
24	55	58	05	66	73	99	26	97	17	78	78	96	83	14	88	34	89	63	72
21	36	23	09	75	00	76	44	20	45	35	14	00	61	33	97	34	31	33	95
78	17	53	28	22	75	31	67	15	94	03	80	04	62	16	14	09	53	56	92
16	39	05	42	96	35	31	47	55	58	88	24	00	17	54	24	36	29	85	57
86	56	00	48	35	71	89	07	05	44	44	37	44	60	21	58	51	54	17	58
19	80	81	68	05	94	47	69	28	73	92	13	86	52	17	77	04	89	55	40
04	52	01	63	97	35	99	16	07	97	57	32	16	26	26	79	33	27	98	66
66	44	68	87	57	62	20	72	03	46	33	67	46	55	12	32	63	93	53	69
04	42	16	73	55	35	39	11	24	94	72	18	08	46	29	32	40	62	76	36
20	69	36	41	72	30	23	88	34	72	99	69	82	67	59	85	74	04	36	16
20	73	35	29	78	31	90	01	74	31	49	71	48	88	81	16	23	57	05	54
01	70	54	71	83	51	54	69	16	92	33	48	61	43	52	01	89	23	67	48

What the computer sees

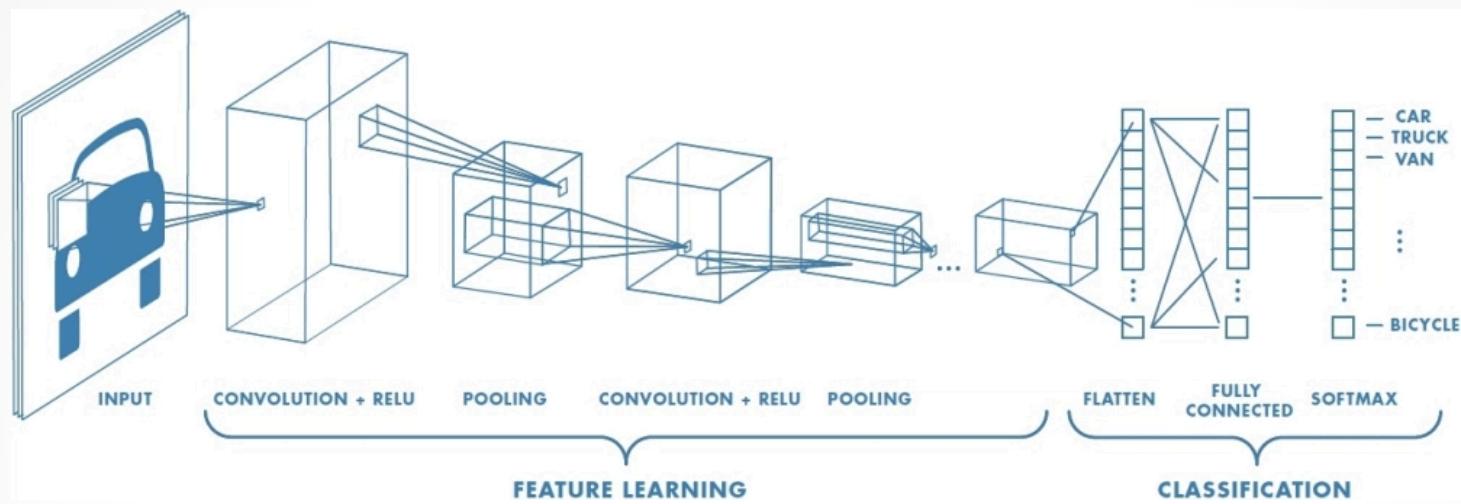
image classification

82% cat
15% dog
2% hat
1% mug

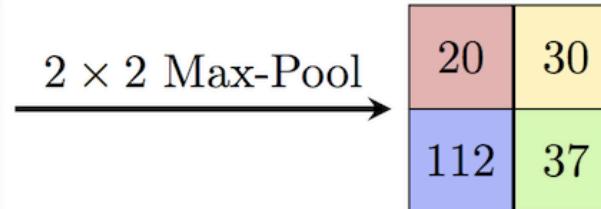
Convolutional Neural Networks



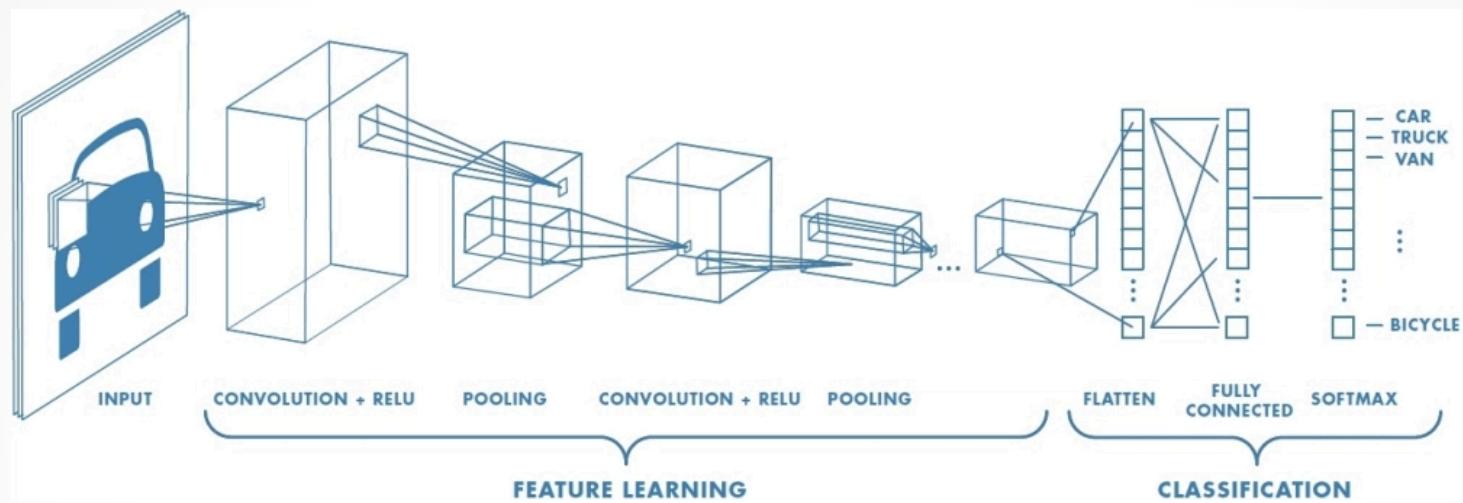
Convolutional Neural Networks



12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

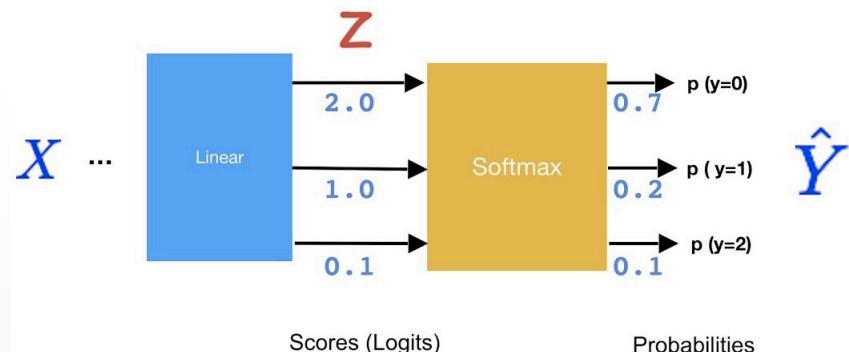


Convolutional Neural Networks



Meet Softmax

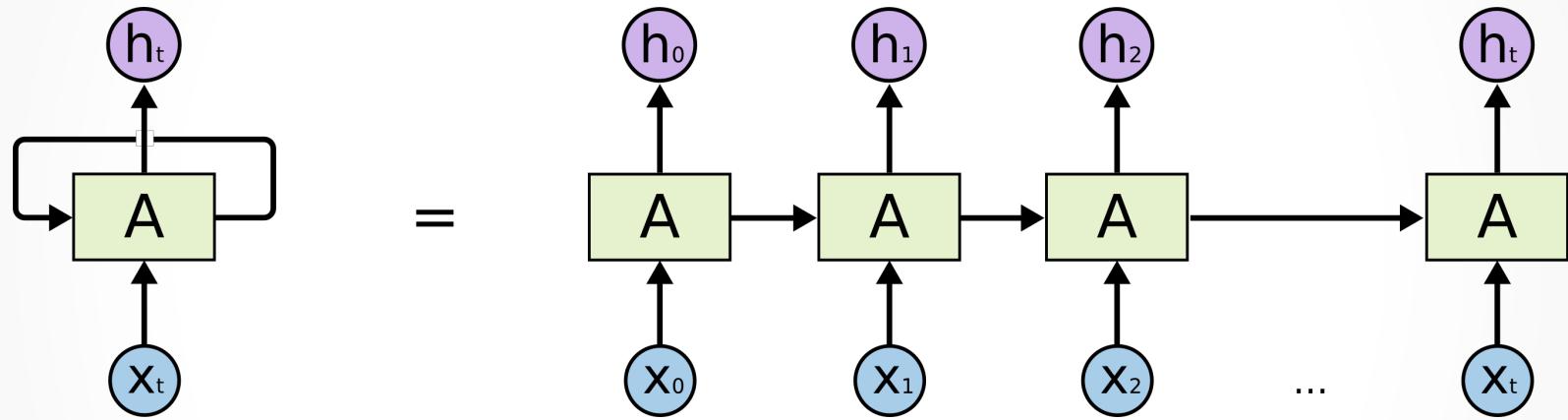
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$



RNN and LSTM

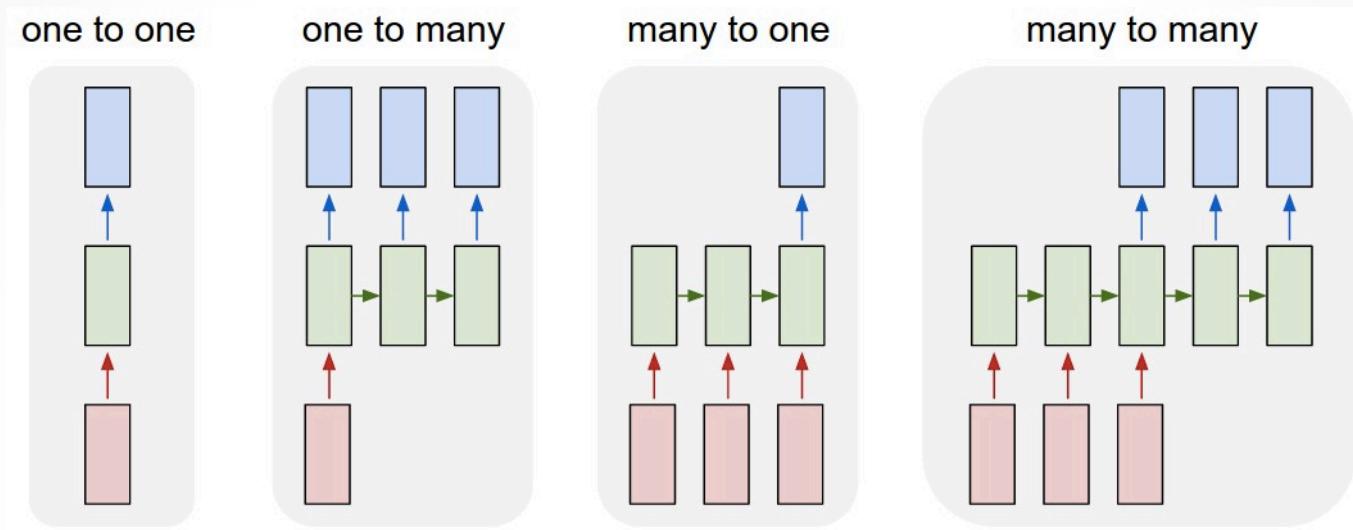
...

Recurrent Neural Network (RNN)



Humans don't start their thinking from scratch every time.
Why should machines?

Applications of RNN



(1) Fixed-sized input to fixed-sized output

(e.g. image classification)

(2) Sequence output

(e.g. image captioning takes an image and outputs a sentence of words)

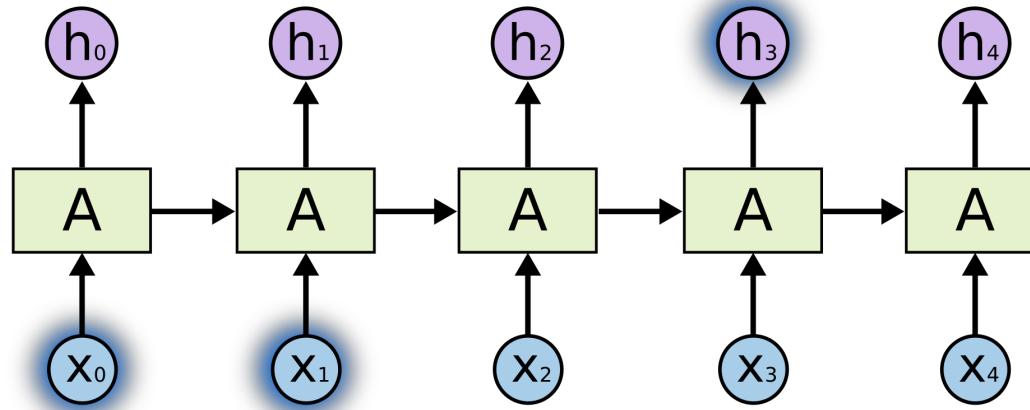
(3) Sequence input

(e.g. sentiment analysis)

(4) Sequence input and sequence output

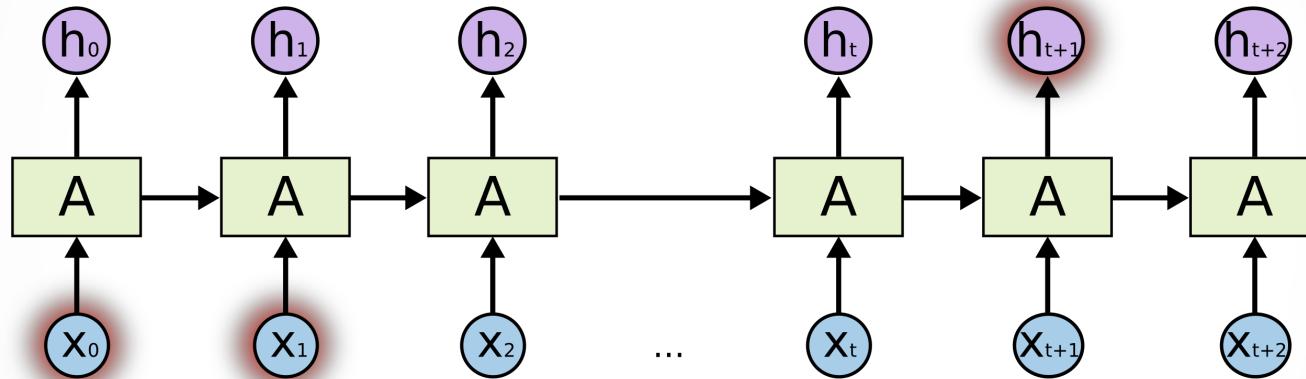
(e.g. Machine Translation: a sentence in English to a sentence in French)

RNNs are good for short dependencies



The Clouds are in the *sky*

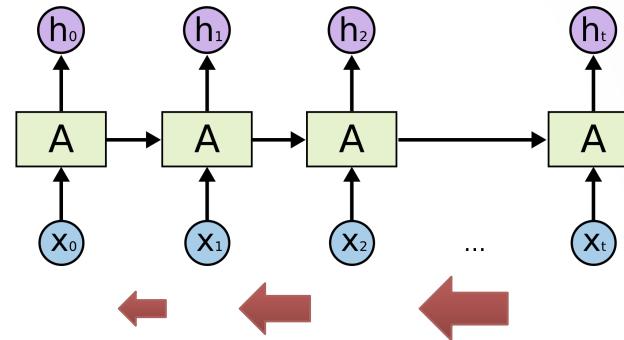
RNNs are problematic with longer dependencies



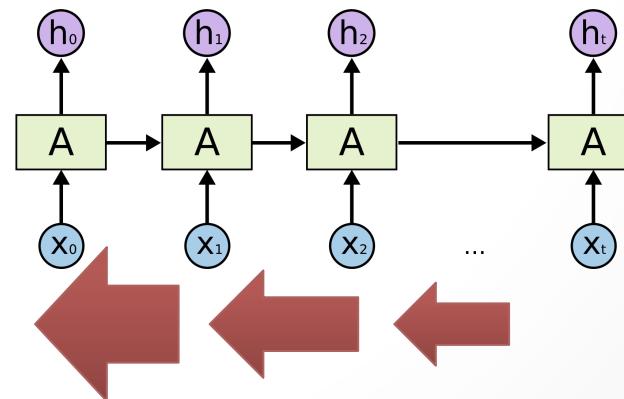
I grew up in France ... I speak fluent *French*

Problems with RNNs

Vanishing Gradients



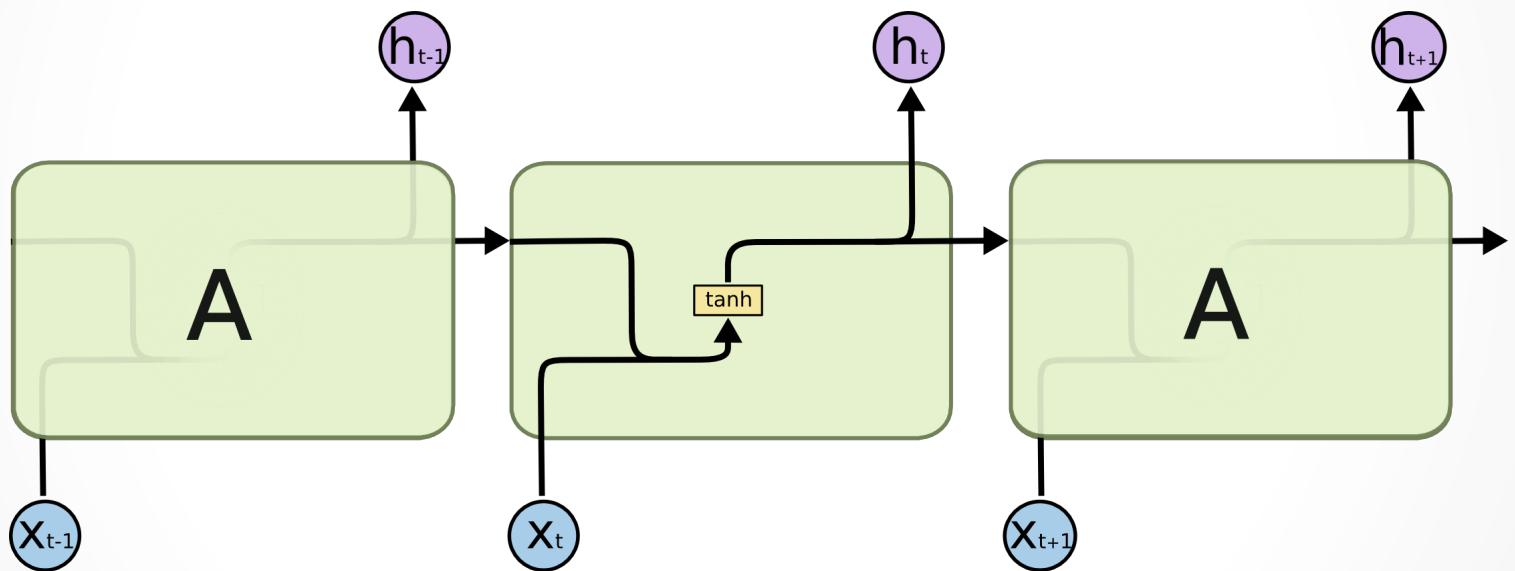
Exploding Gradients



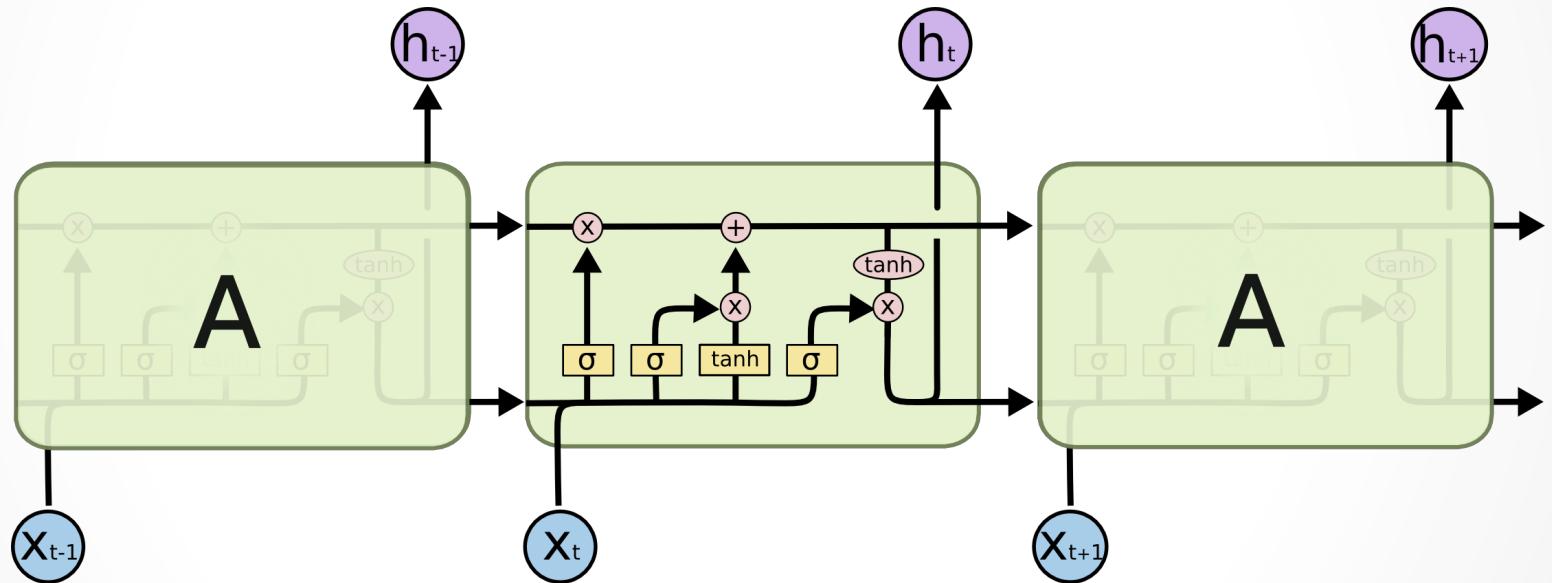
The Solution?

Long Short Term Memory
networks (LSTM)

Inside RNN units

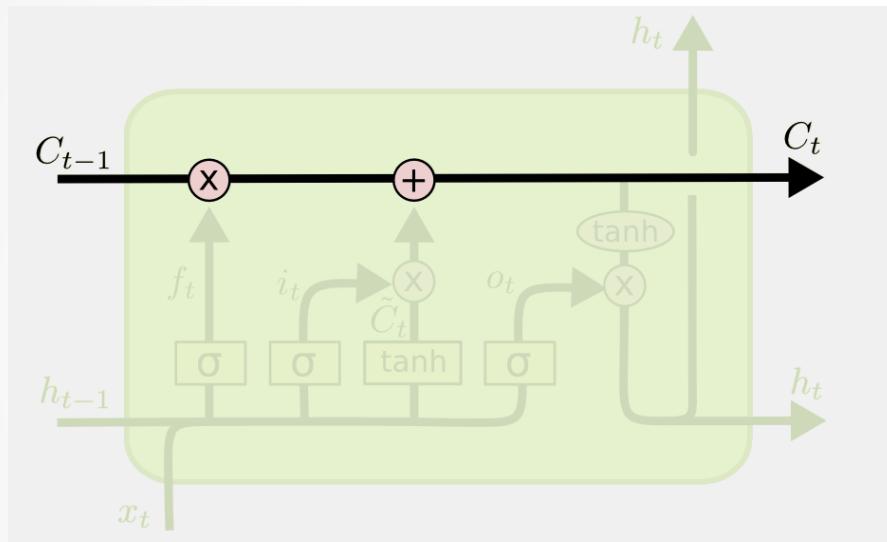


Inside LSTM units

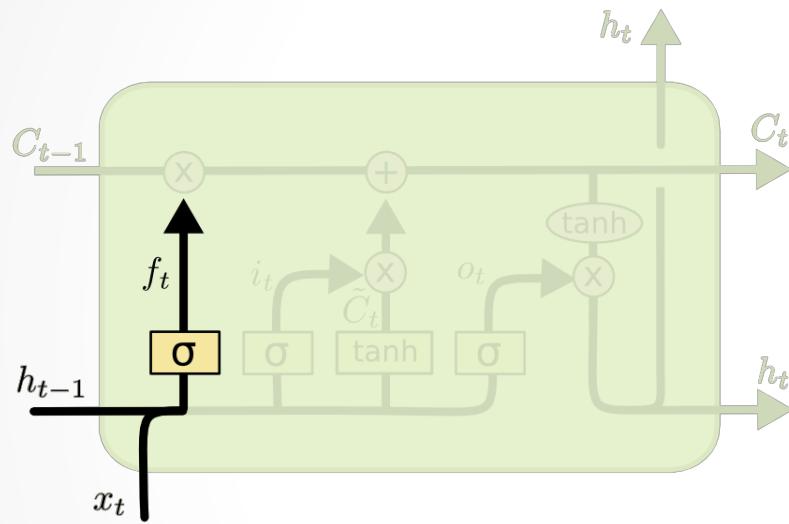


Gates to control adding and removing information

Cell State



Forget Gate



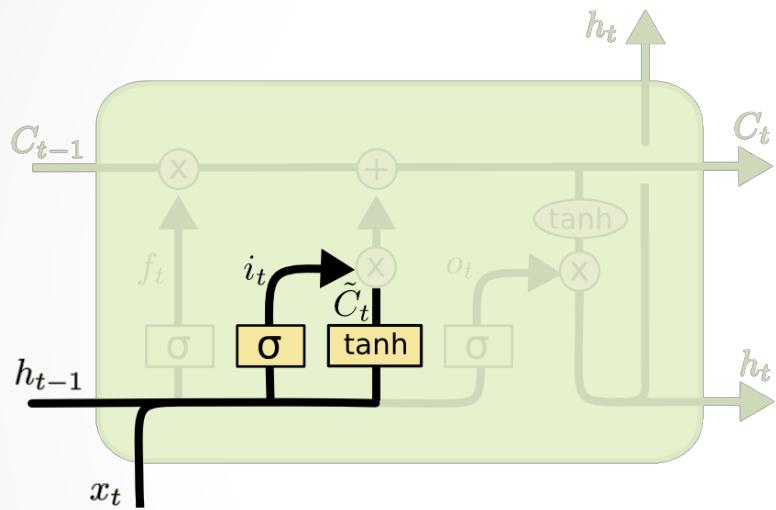
$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

e.g.,

The cell state might include the gender of the present subject, so that the correct pronouns can be used.

When we see a new subject, we want to forget the gender of the old subject.

Input Gate Layer



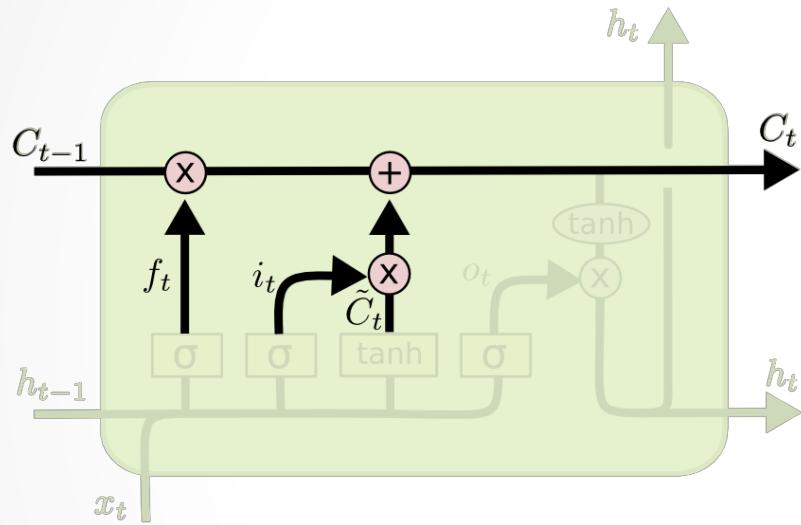
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

e.g.,

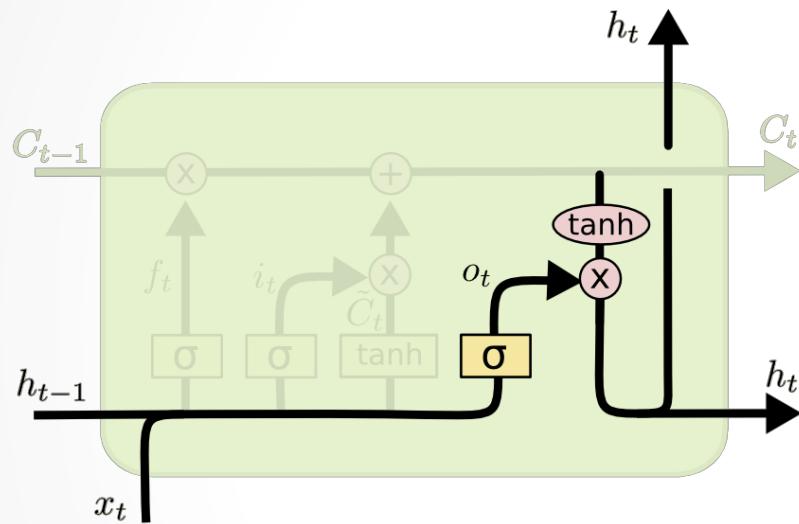
we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.

Applying the updates



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output gate



$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

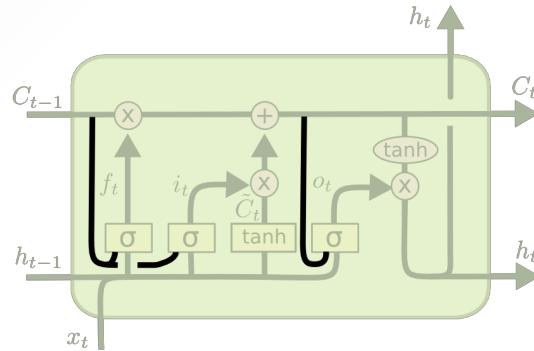
e.g.,

The network just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next.

For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.

Variants on Long Short Term Memory

Peephole
connections



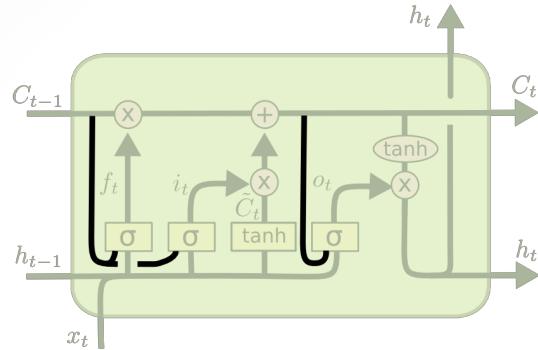
$$f_t = \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o)$$

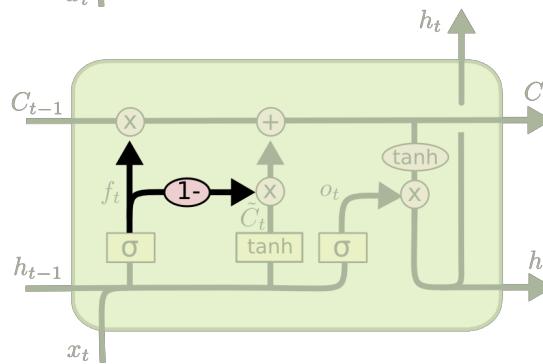
Variants on Long Short Term Memory

Peephole
connections



$$f_t = \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i)$$
$$o_t = \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o)$$

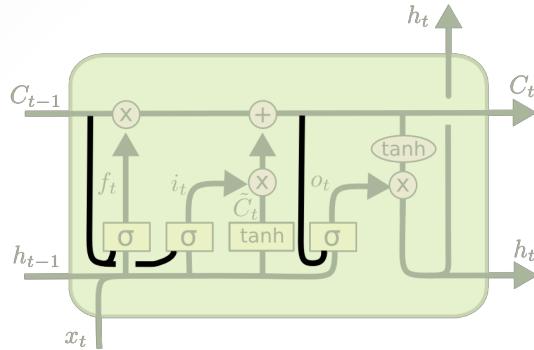
Coupled
forget and
input gates



$$C_t = f_t * C_{t-1} + (1 - f_t) * \tilde{C}_t$$

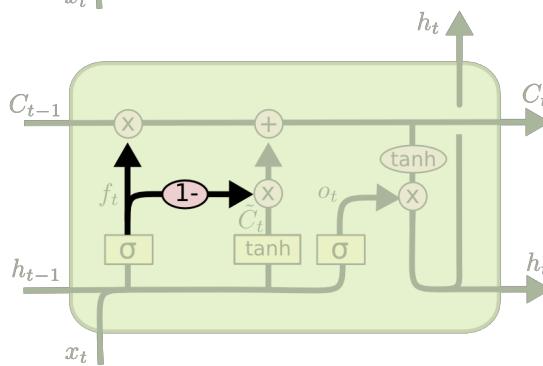
Variants on Long Short Term Memory

Peephole
connections



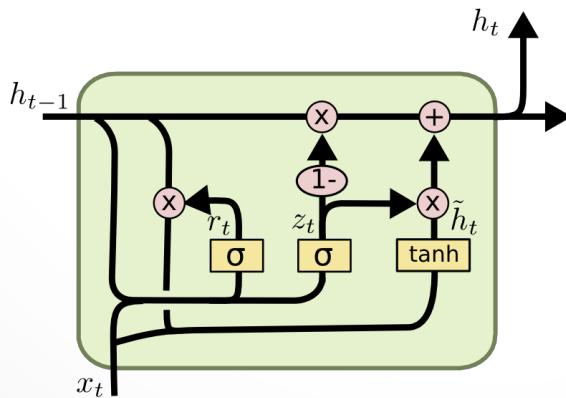
$$f_t = \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i)$$
$$o_t = \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o)$$

Coupled
forget and
input gates



$$C_t = f_t * C_{t-1} + (1 - f_t) * \tilde{C}_t$$

Gated
Recurrent
Unit (GRU)



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

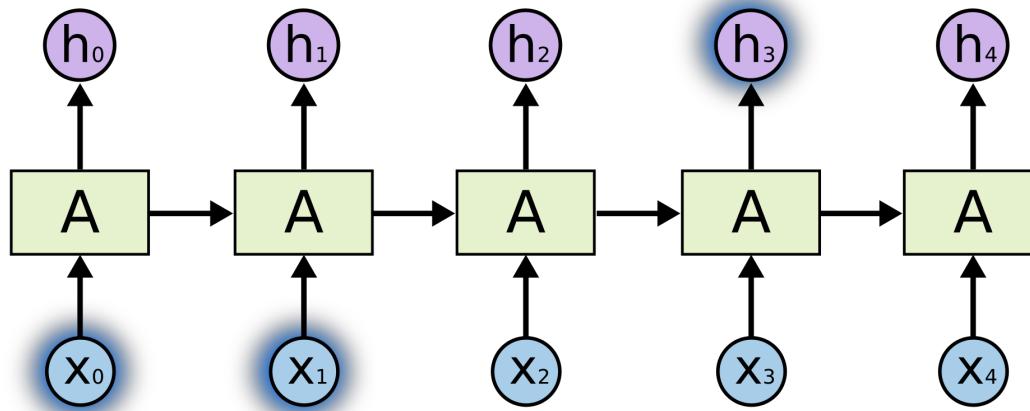
$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Word Embedding

...

Our inputs are strings!



The Clouds are in the *sky*

How to Represent a Word?

- Challenge
 - Discrete structure
- Simple representation
 - One-hot representation: a vector with one 1 and a lot of zeroes
 - E.g., Motel =

[0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

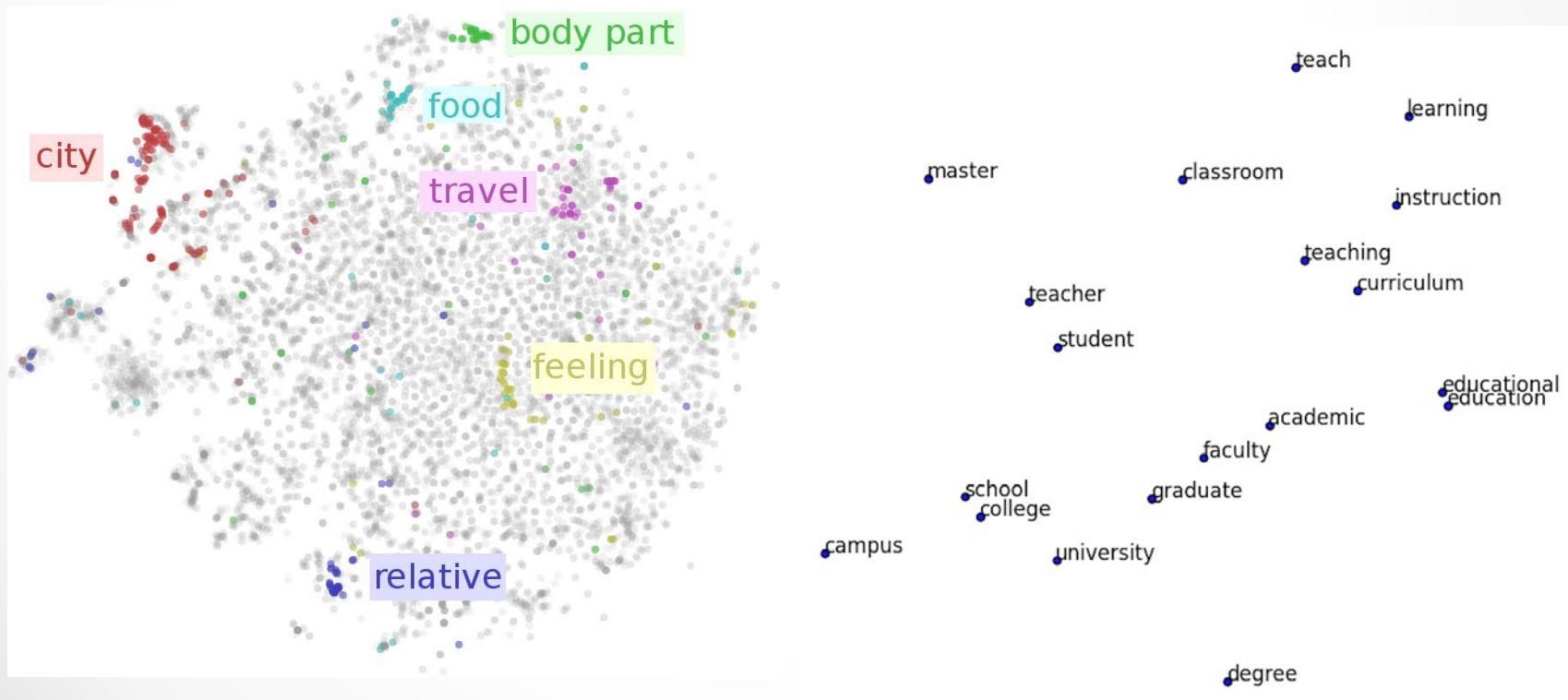
One-Hot Representation

- High dimensionality
 - E.g., for Google news, 13M words
- Sparse
 - Only 1 non-zero value
- It has no semantics
 - E.g.,

motel [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0] AND
hotel [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0] = 0

Word Embedding

- Low dimensional vector representation of every word
 - E.g., motel =[1.3, -1.4] and hotel =[1.2, -1.5]



How to Learn Such Embeddings?

- Using context information!

...he curtains open and the **moon** shining in on the barely...

...ars and the **cold** , close **moon** " . And neither of the w...

...rough the **night** with the **moon** shining so **brightly** , it...

...made in the **light** of the **moon** . It all boils down , wr...

...surely under a **crescent moon** , thrilled by ice-white...

...sun , the **seasons** of the **moon** ? Home , alone , Jay pla...

...m is dazzling snow , the **moon** has risen full and **cold**...

...un and the **temple** of the **moon** , driving out of the hug...

...in the **dark** and now the **moon** rises , **full** and amber a...

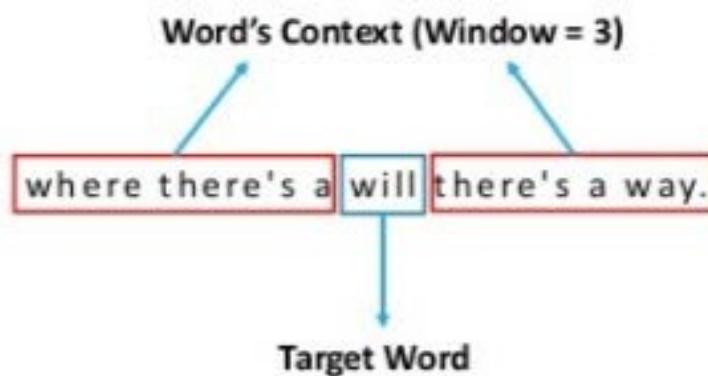
...bird on the **shape** of the **moon** over the **trees** in front...

Word2Vec

- Proposed by Mikolov et al. at Google in 2013
- The most popular word embedding models
- Fast
 - “an optimized single-machine implementation can train on more than 100 billion words in one day”
- Two main architectures:
 - Continuous bag-of-words (CBOW)
 - Skip-gram

Main Idea of Word2Vec

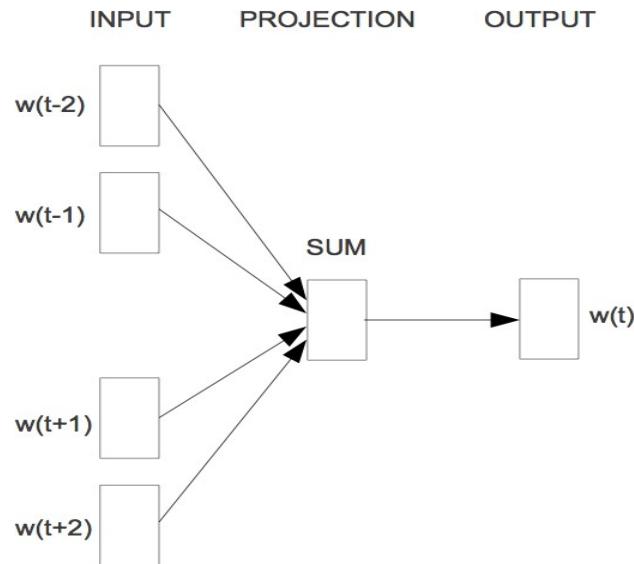
- Consider a local window of a target word



- CBOW:
predict the target words given the neighbors
- Skip-gram:
predict neighbors given the target words

CBOW

- Predicting target using neighbors



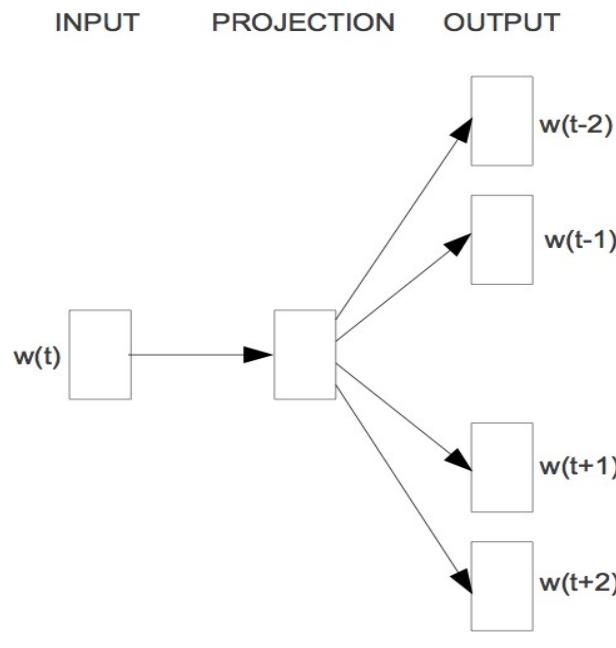
$$J_{\theta} = \frac{1}{T} \sum_{t=1}^T \log p(w_t | w_{t-n}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+n})$$

More details can be found in:

http://www.1-4-5.net/~dmm/ml/how_does_word2vec_work.pdf

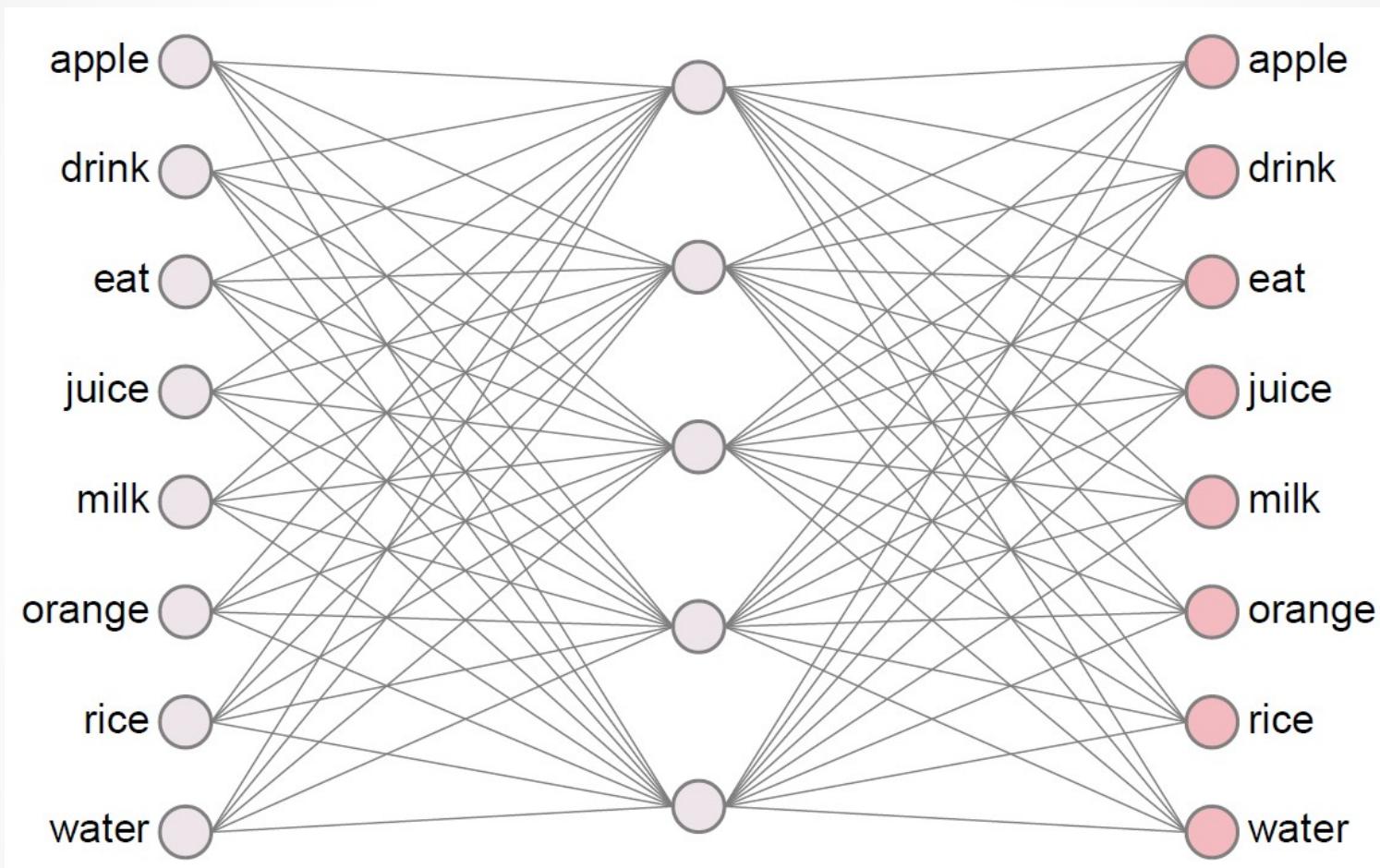
Skip-Gram

- Predicting neighbors using target



$$J_{\theta} = \frac{1}{T} \sum_{t=1}^T \sum_{-n \leq j \leq n, j \neq 0} \log p(w_{t+j} | w_t)$$

A Neural Network Point of View



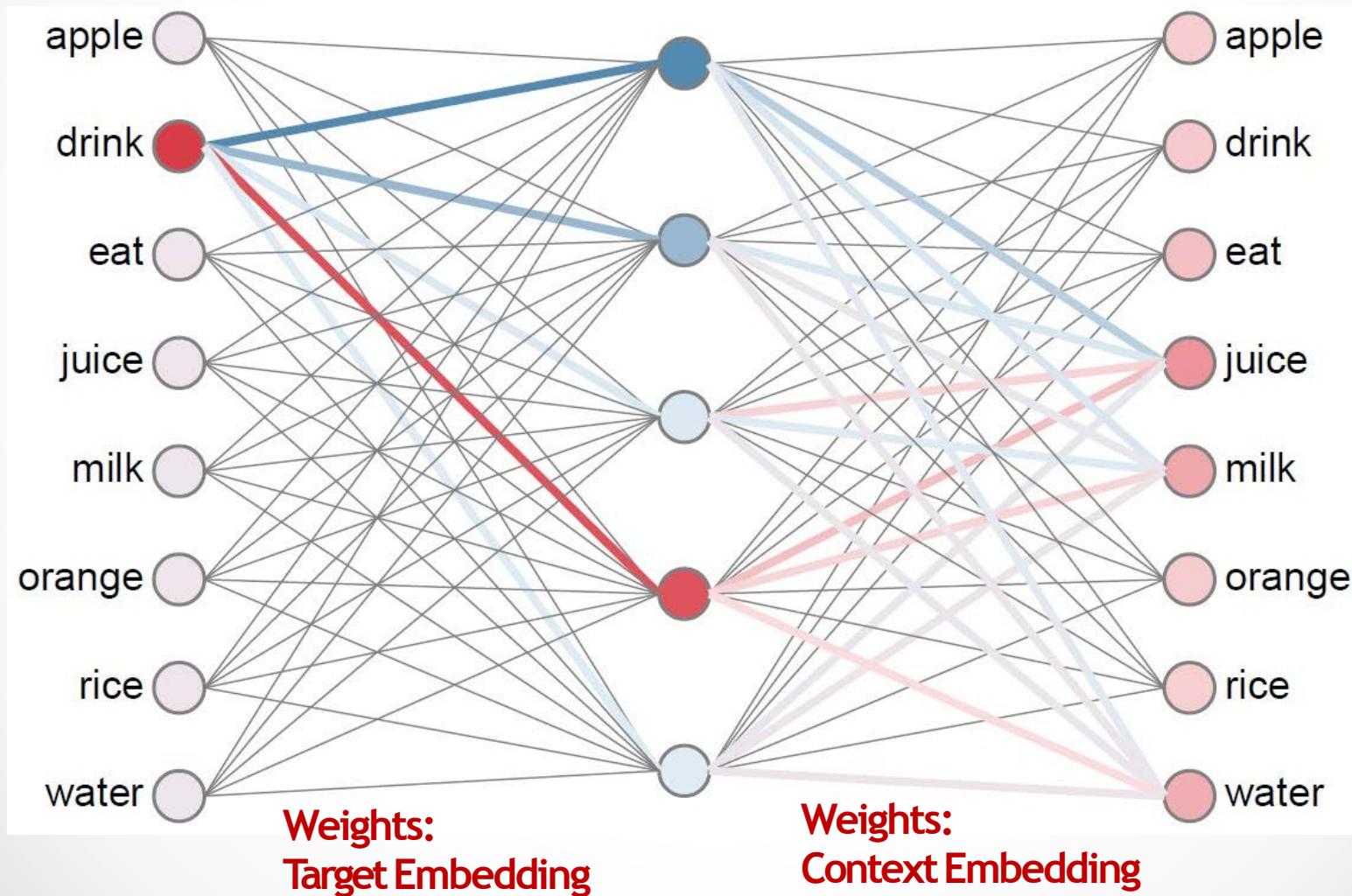
Input Layer:
one-hot vector

Hidden Layer:
Linear (Identity)

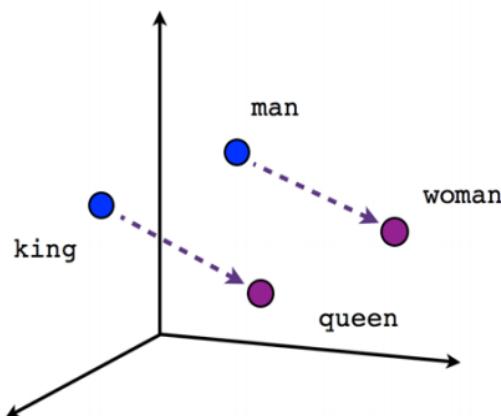
Output Layer:
softmax

Demo

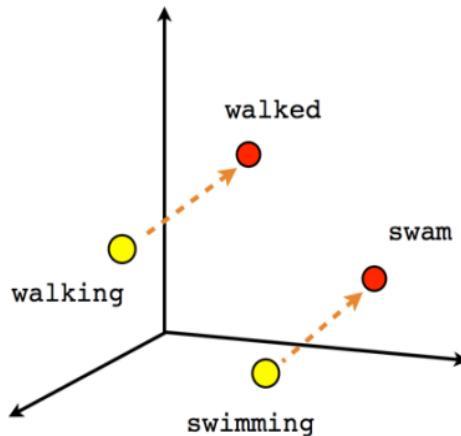
- <https://ronxin.github.io/wevi/>



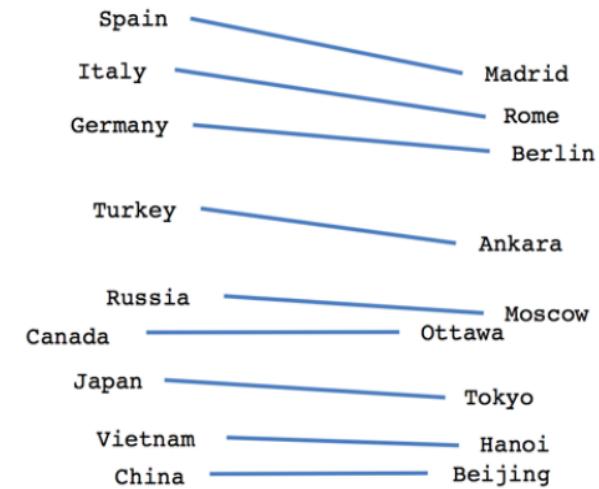
Examples



Male-Female



Verb tense

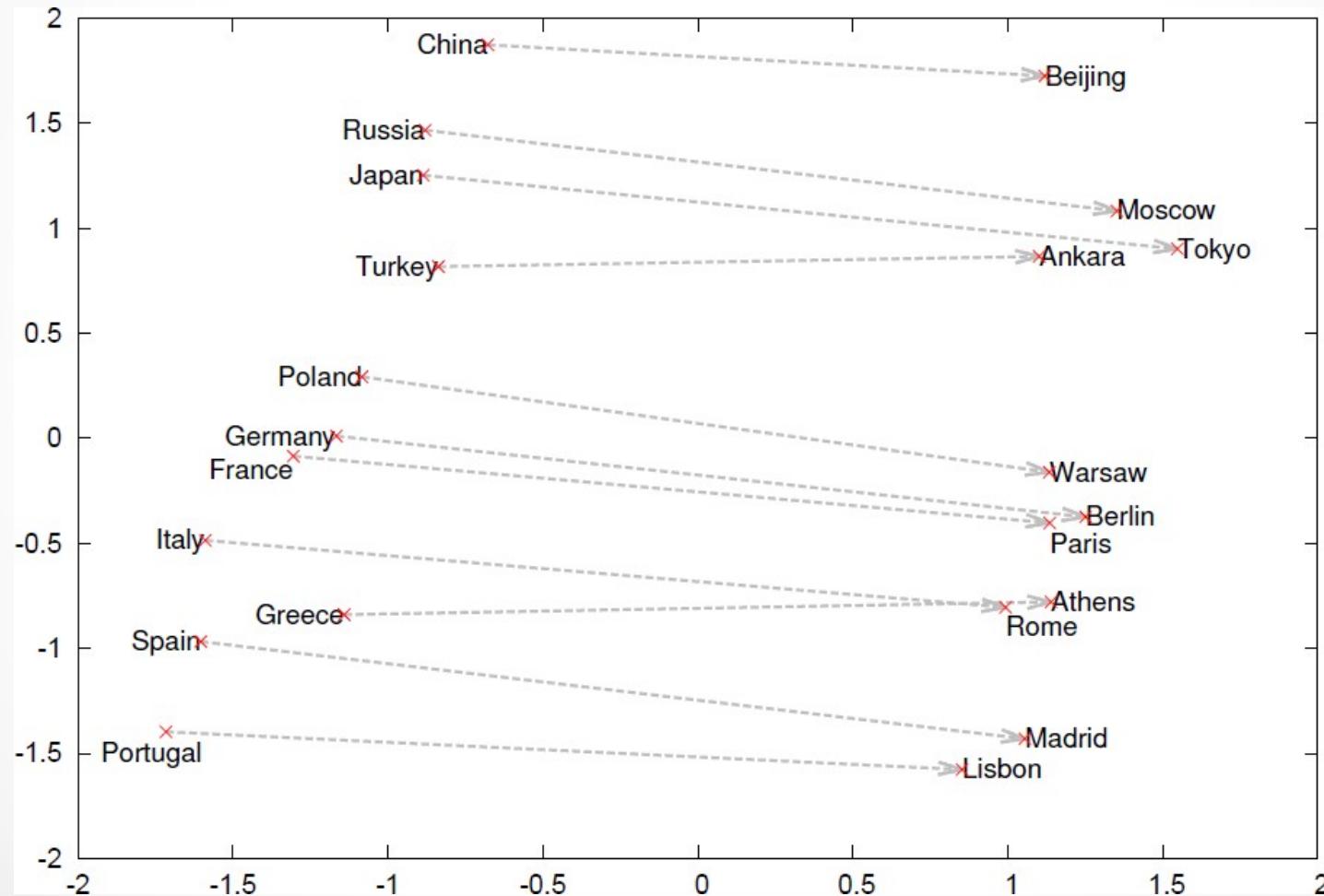


Country-Capital

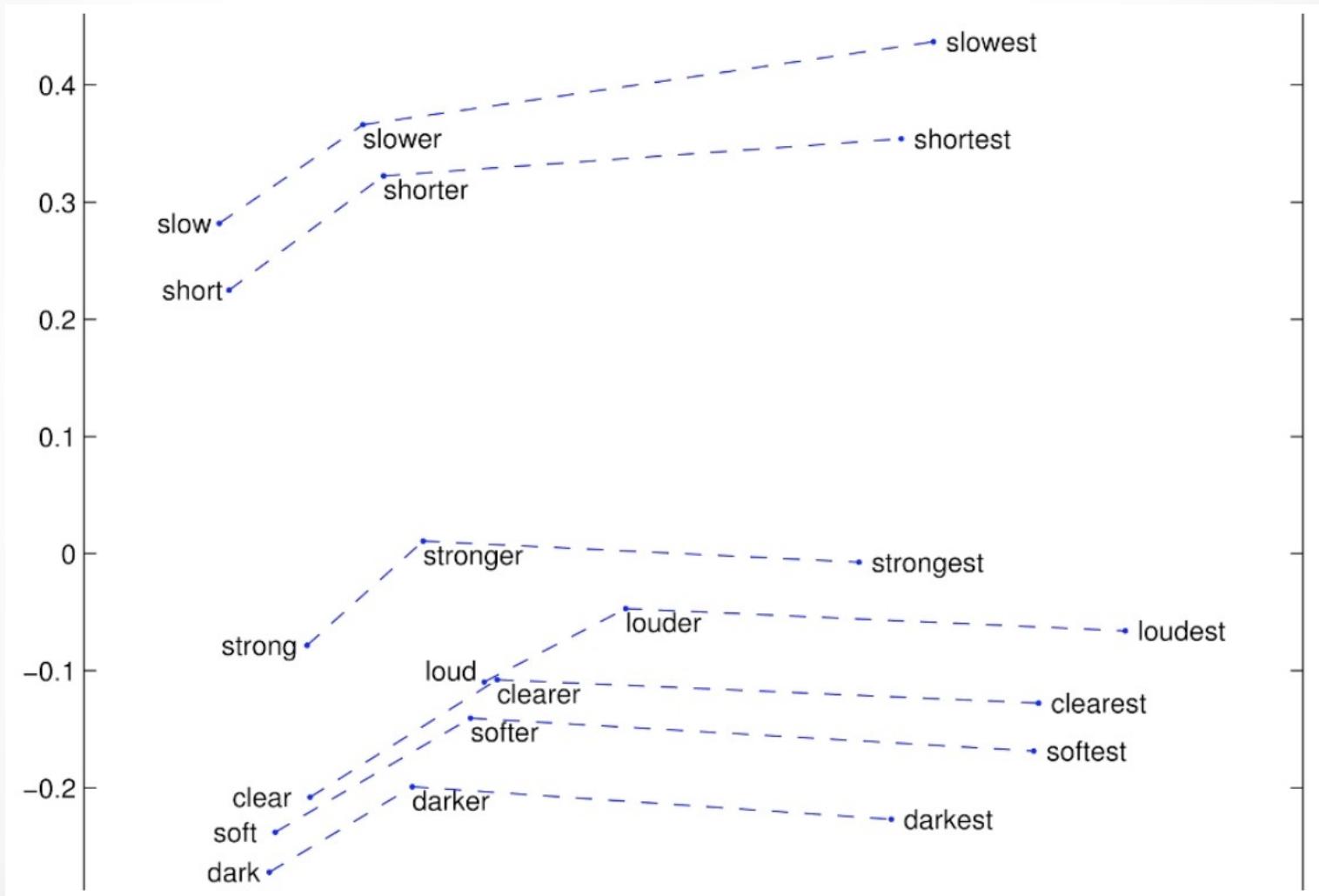
$$\text{vector[Queen]} = \text{vector[King]} - \text{vector[Man]} + \text{vector[Woman]}$$

A Potential Application

- Relationship detection



Some Interesting Results: Superlatives



Word2Vec

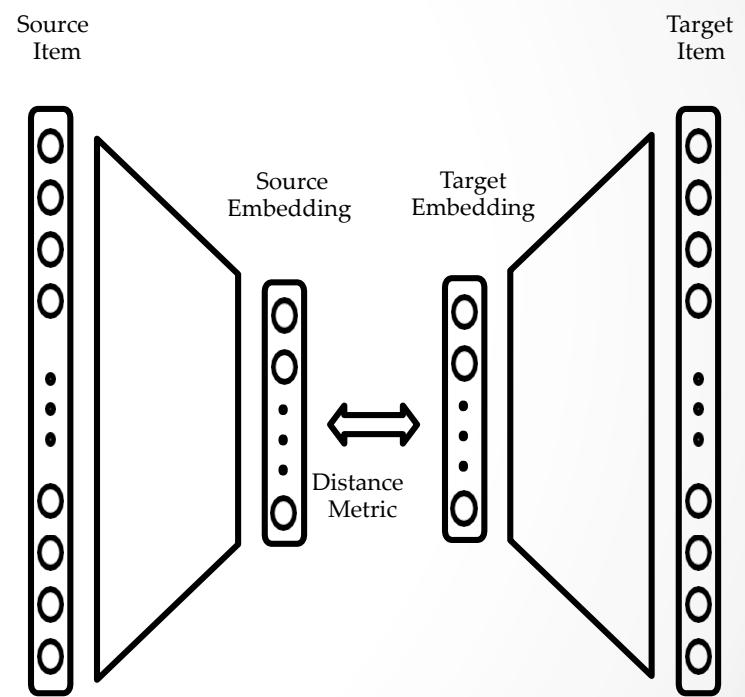
Start with a paired items dataset

[source, target]

Train a neural network

Bottleneck layer gives you a dense vector representation

E.g., word2vec



GloVe

(Global Vectors for Word Representation)

Start with a paired items dataset

[source, target]

Make a Source x Target matrix

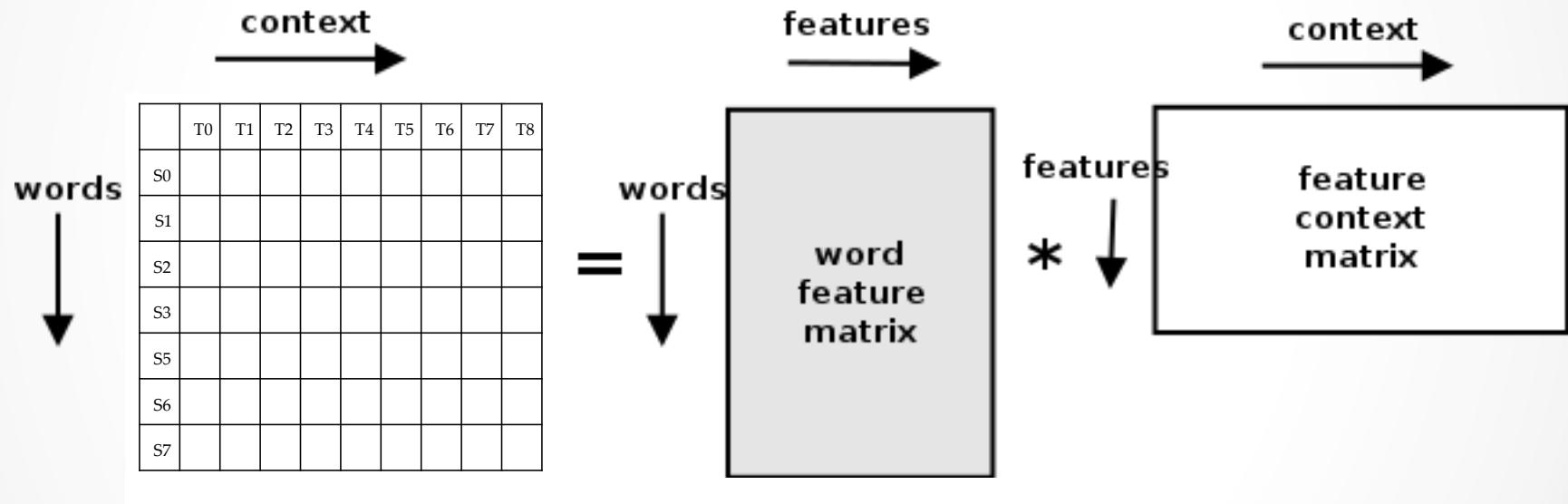
Factorizing the matrix gives you
a dense vector representation

E.g., LSA, GloVe

	T0	T1	T2	T3	T4	T5	T6	T7	T8
S0									
S1									
S2									
S3									
S5									
S6									
S7									

GloVe

(Global Vectors for Word Representation)



Pennington, Socher, and Manning. "Glove: Global Vectors for Word Representation." EMNLP (2014).

Some examples of text embeddings

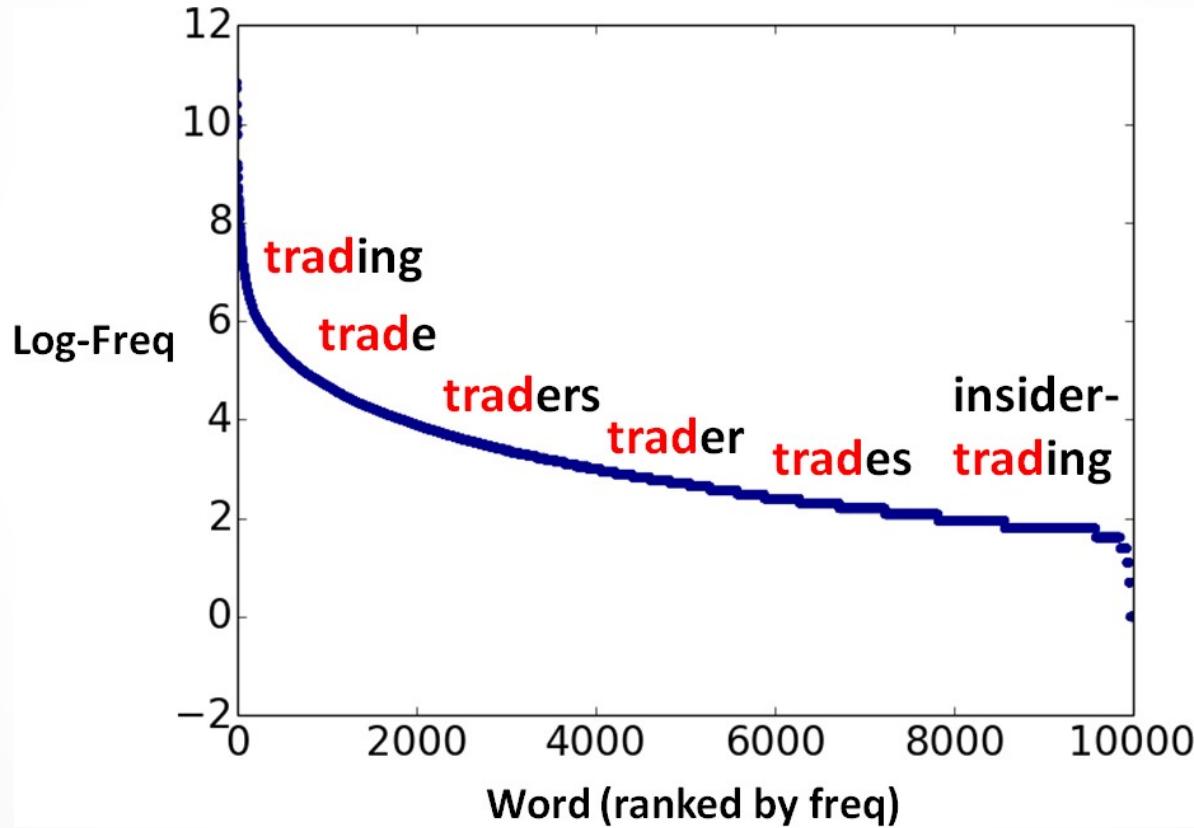
	Embedding for	Source Item	Target Item	Learning Model
Latent Semantic Analysis Deerwester et. al. (1990)	Single word	Word (one-hot)	Document (one-hot)	Matrix factorization
Word2vec Mikolov et. al. (2013)	Single Word	Word (one-hot)	Neighboring Word (one-hot)	Neural Network (Shallow)
Glove Pennington et. al. (2014)	Single Word	Word (one-hot)	Neighboring Word (one-hot)	Matrix factorization
Semantic Hashing (auto-encoder) Salakhutdinov and Hinton (2007)	Multi-word text	Document (bag-of-words)	Same as source (bag-of-words)	Neural Network (Deep)
DSSM Huang et. al. (2013), Shen et. al. (2014)	Multi-word text	Query text (bag-of-trigrams)	Document title (bag-of-trigrams)	Neural Network (Deep)
Session DSSM Mitra (2015)	Multi-word text	Query text (bag-of-trigrams)	Next query in session (bag-of-trigrams)	Neural Network (Deep)
Language Model DSSM Mitra and Craswell (2015)	Multi-word text	Query prefix (bag-of-trigrams)	Query suffix (bag-of-trigrams)	Neural Network (Deep)

Character Embedding

...

An Issue with word embedding

Separate embeddings for “trading”, “trade”, “trades”, etc.



One Solution: Pre-process using morphological segmenter

CNN - Character Embedding

$\mathbf{C} \in \mathbb{R}^{d \times l}$: Representation of *absurdity*

0.4	-0.8	2.2	0.1	0.5	-0.4	0.4	-0.4	0.1
0.1	1.2	1.5	-0.8	-1.5	0.2	0.1	1.2	0.7
0.2	0.1	-1.2	0.2	-0.2	0.3	0.2	-1.3	-0.1
-0.2	-0.5	0.1	0.2	-0.3	0.3	-0.1	1.0	-0.3

a b s u r d i t y

The diagram illustrates the character embedding matrix \mathbf{C} . Below the matrix, the word "absurdity" is written, with each character aligned under its corresponding column in the matrix. Dotted vertical arrows point from each character to the top of its respective column in the matrix, demonstrating how each character is represented by a unique vector in the d -dimensional space.

CNN - Character Embedding

$\mathbf{H} \in \mathbb{R}^{d \times w}$: Convolutional filter matrix of width $w = 3$

-0.1	0.5	2.2
0.7	0.9	0.3
-0.2	-0.2	0.7
1.3	-0.1	-1.1

..... ➔

0.4	-0.8	2.2	0.1	0.5	-0.4	0.4	-0.4	0.1
0.1	1.2	1.5	-0.8	-1.5	0.2	0.1	1.2	0.7
0.2	0.1	-1.2	0.2	-0.2	0.3	0.2	-1.3	-0.1
-0.2	-0.5	0.1	0.2	-0.3	0.3	-0.1	1.0	-0.3

a b s u r d i t y

CNN - Character Embedding

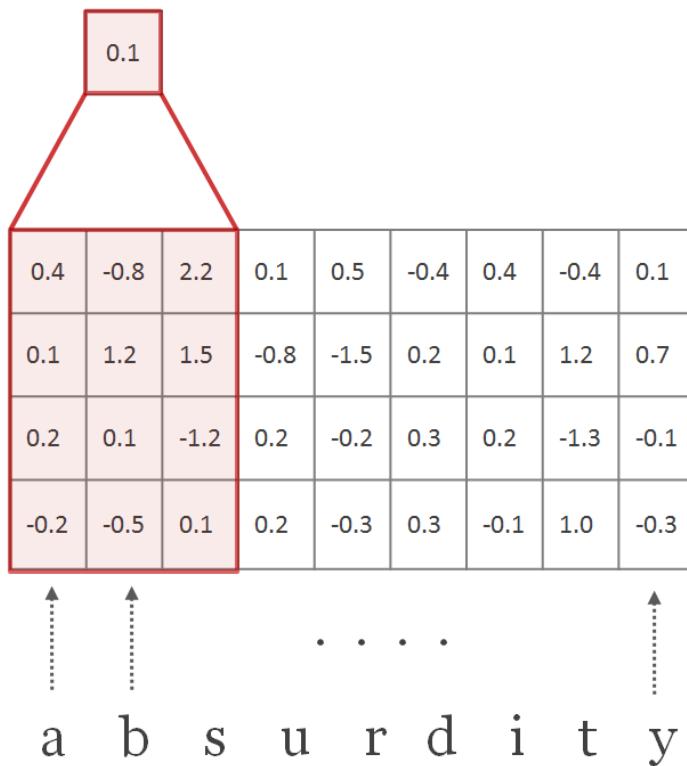
$$\mathbf{f}[1] = (\mathbf{C}[:, 1 : 3], \mathbf{H})$$

0.4	-0.8	2.2	0.1	0.5	-0.4	0.4	-0.4	0.1
0.1	1.2	1.5	-0.8	-1.5	0.2	0.1	1.2	0.7
0.2	0.1	-1.2	0.2	-0.2	0.3	0.2	-1.3	-0.1
-0.2	-0.5	0.1	0.2	-0.3	0.3	-0.1	1.0	-0.3

Diagram illustrating character embeddings for the word "surday". The word is represented by a sequence of characters: a, b, s, u, r, d, i, t, y. Arrows point from each character to its corresponding embedding vector in the matrix above. The first three columns of the matrix are highlighted with a red border, representing the character embeddings for 'a', 'b', and 's' respectively.

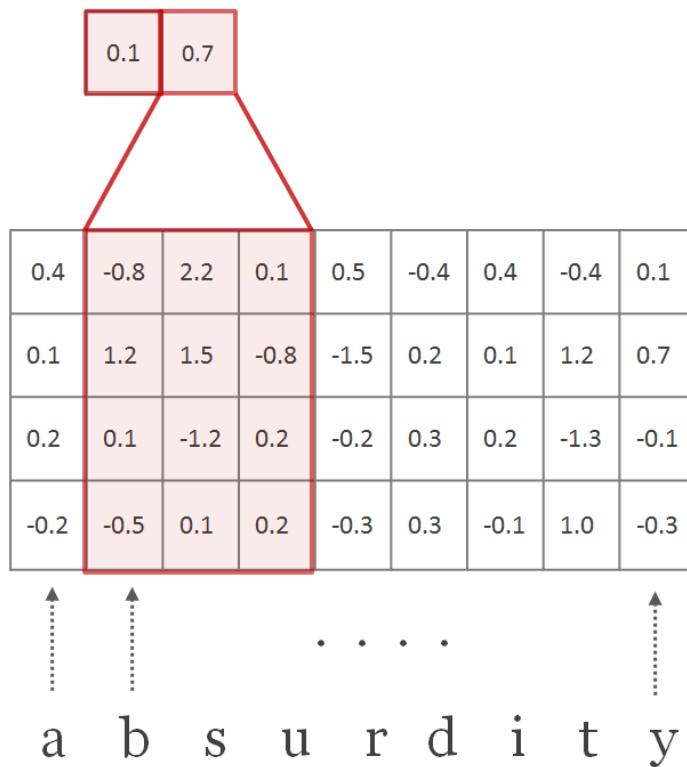
CNN - Character Embedding

$$\mathbf{f}[1] = (\mathbf{C}[:, 1 : 3], \mathbf{H})$$



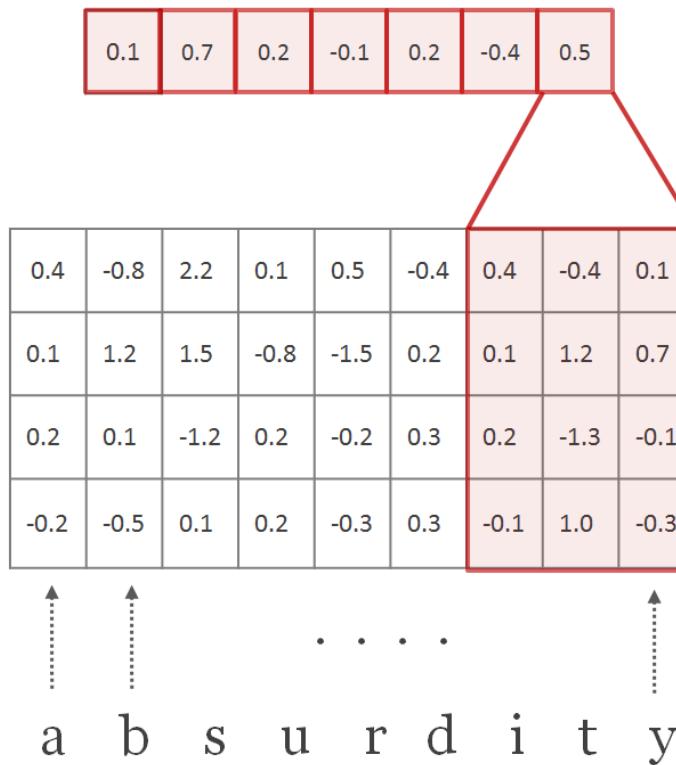
CNN - Character Embedding

$$\mathbf{f}[2] = (\mathbf{C}[:, 2 : 4], \mathbf{H})$$



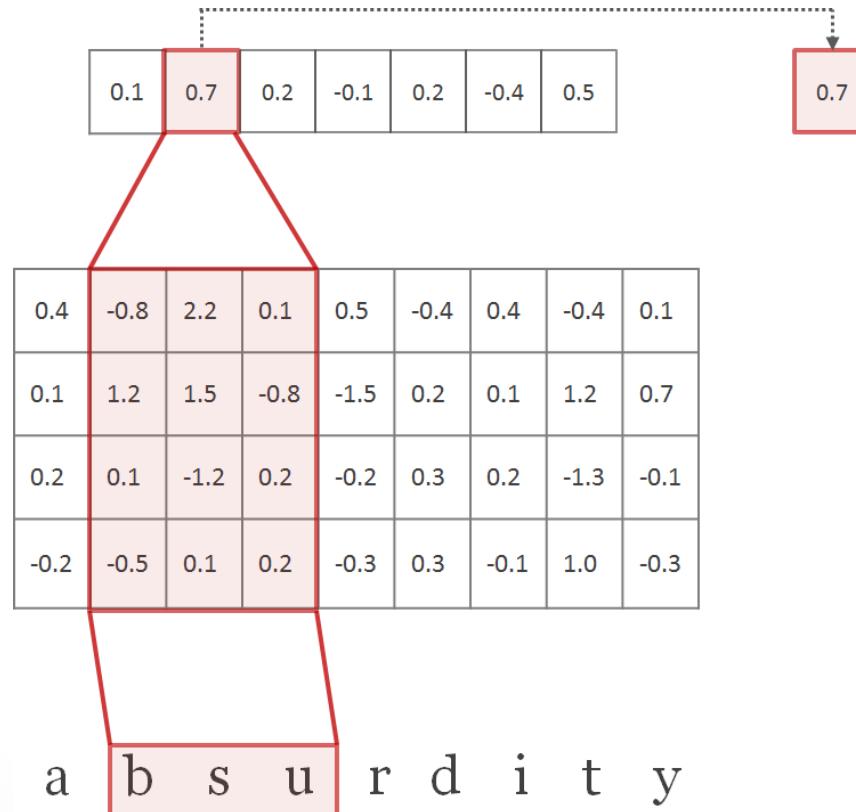
CNN - Character Embedding

$$\mathbf{f}[T-2] = (\mathbf{C}[:, T-2:T], \mathbf{H})$$



CNN - Character Embedding

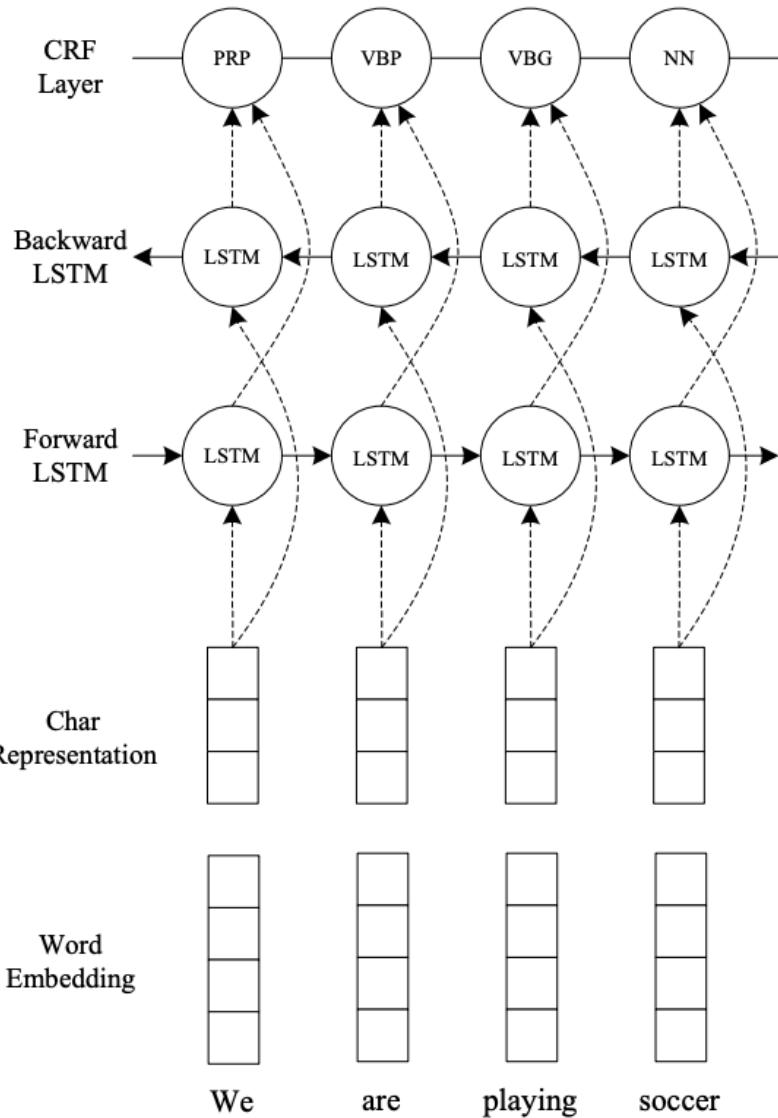
Each filter picks out a character n -gram



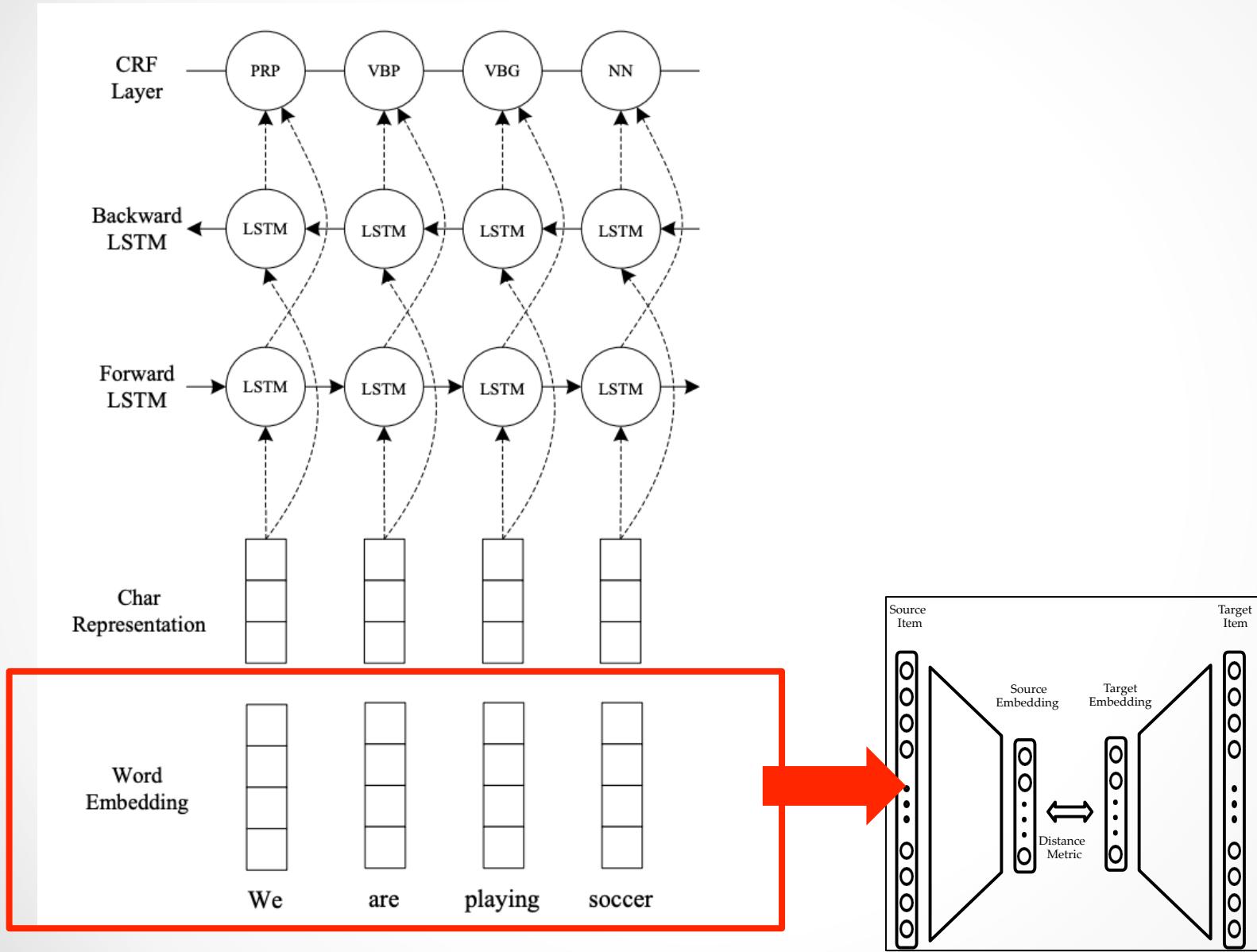
A Deep Learning Model for NER

...

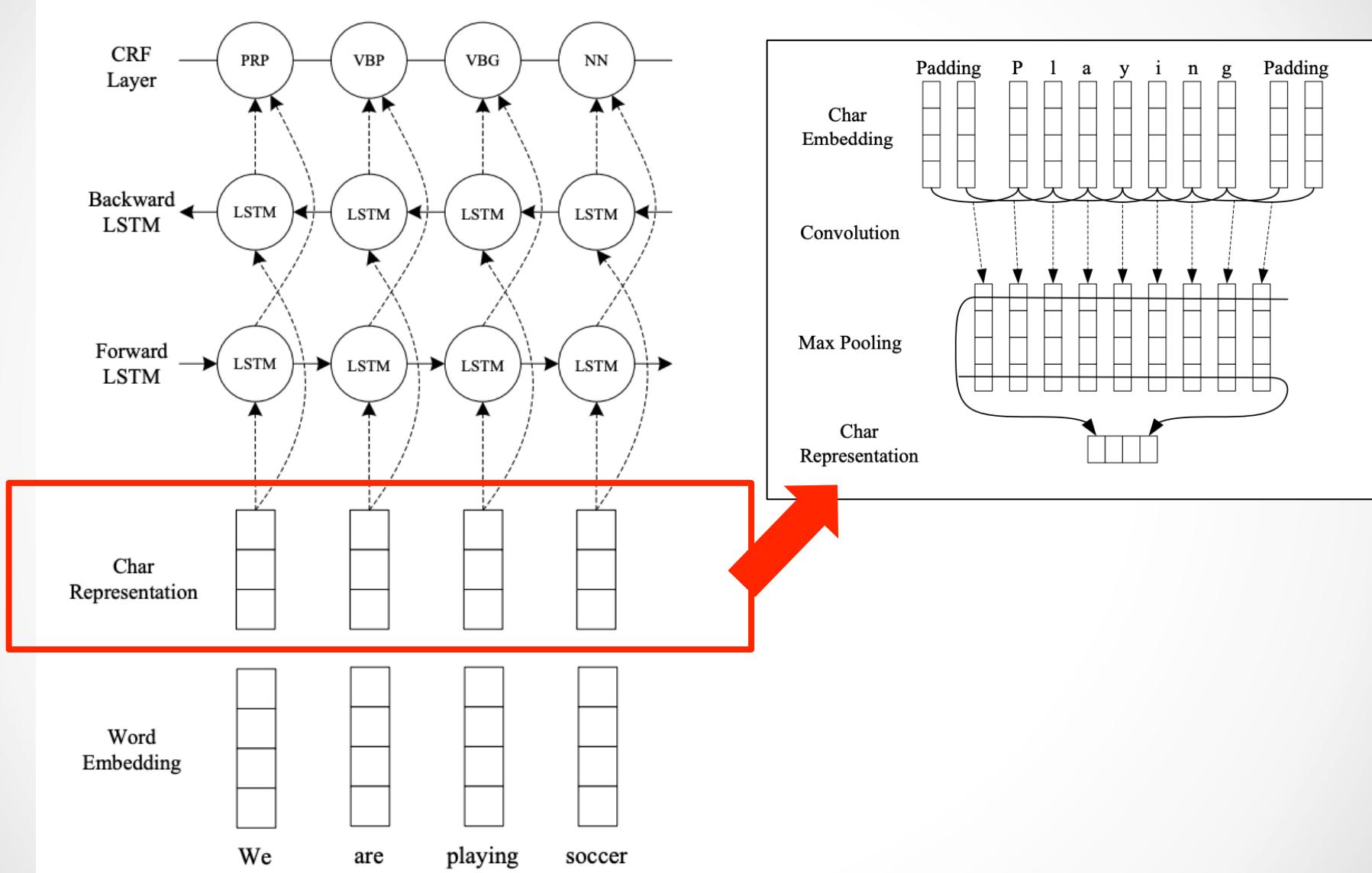
Bi-directional LSTM-CNNs-CRF



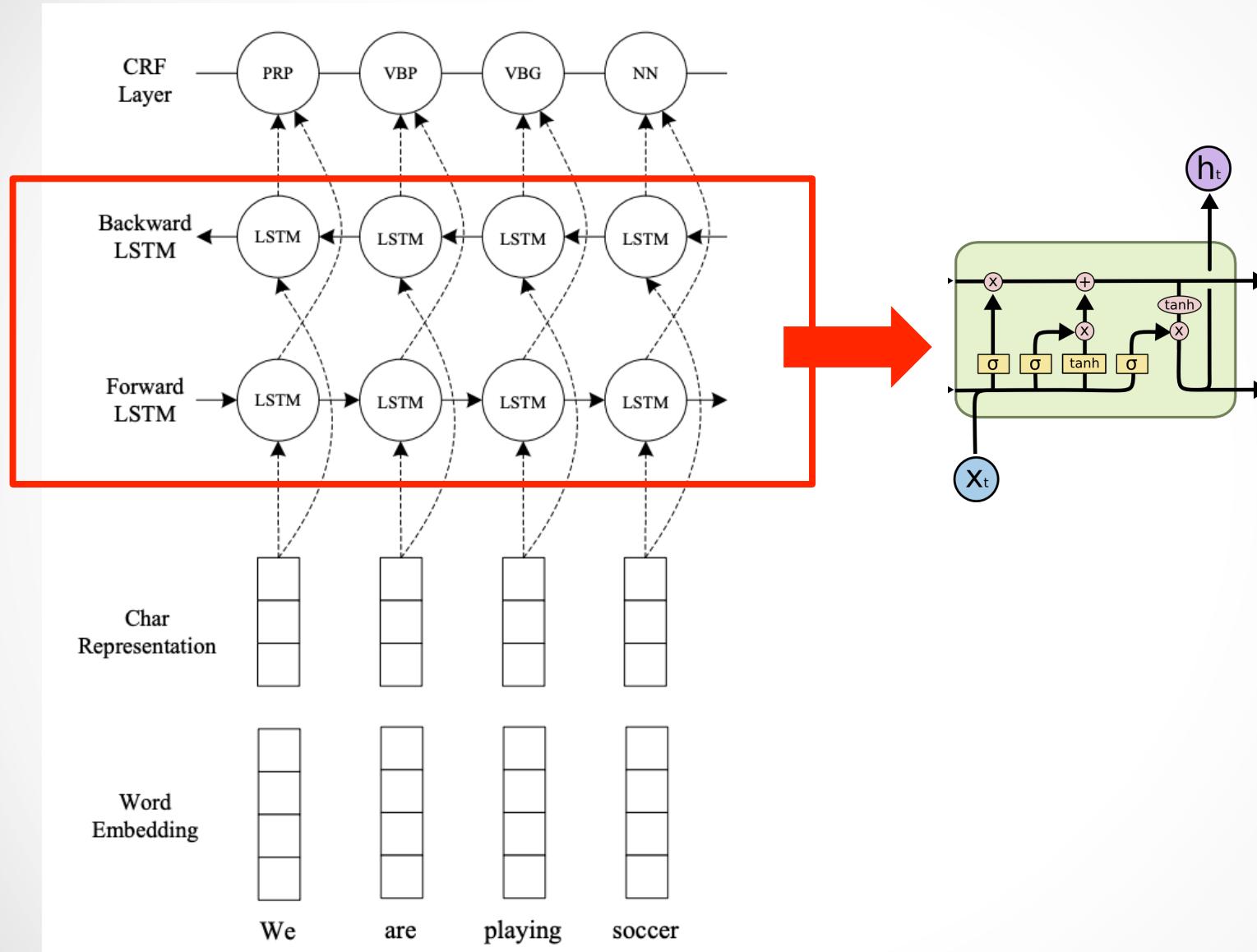
Bi-directional LSTM-CNNs-CRF



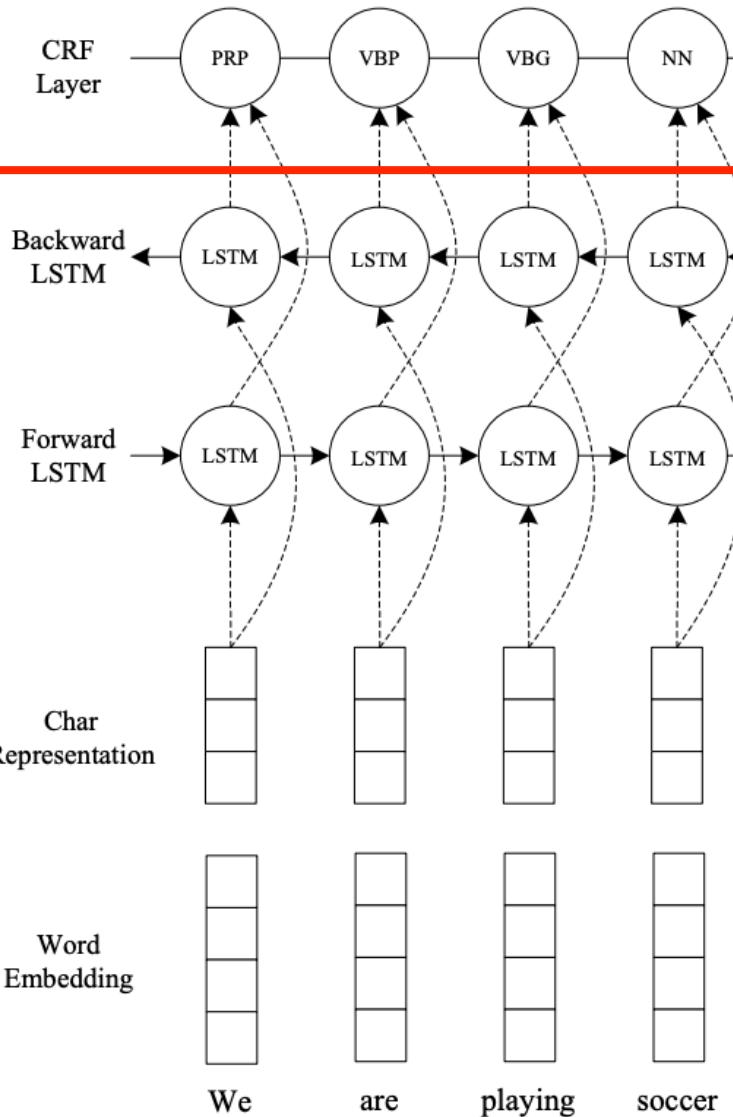
Bi-directional LSTM-CNNs-CRF



Bi-directional LSTM-CNNs-CRF

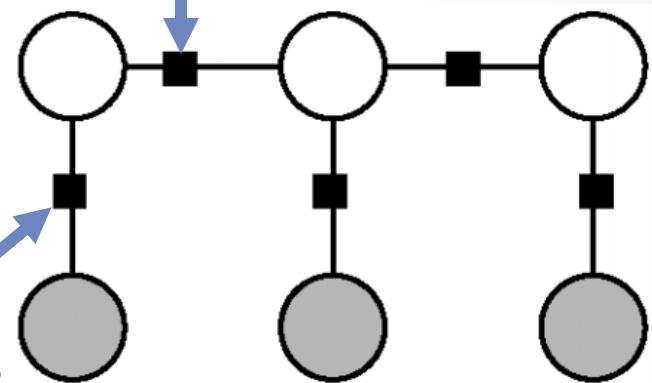


Bi-directional LSTM-CNNs-CRF



Transition
Feature
Functions

State
Feature
Functions



Performance Result

Model	NER		
	Test	Prec.	Recall
BRNN	87.05	83.88	85.44

CoNLL 2003 (PERSON, LOCATION, ORGANIZATION, and MISC)

Performance Result

Model	NER		
	Test	Prec.	Recall
BRNN	87.05	83.88	85.44
BLSTM	87.77	86.23	87.00

Performance Result

Model	NER		
	Test	Prec.	Recall
BRNN	87.05	83.88	85.44
BLSTM	87.77	86.23	87.00
BLSTM-CNN	88.53	90.21	89.36

CoNLL 2003 (PERSON, LOCATION, ORGANIZATION, and MISC)

Performance Result

Model	NER		
	Test	Prec.	Recall
BRNN	87.05	83.88	85.44
BLSTM	87.77	86.23	87.00
BLSTM-CNN	88.53	90.21	89.36
BRNN-CNN-CRF	91.35	91.06	91.21

CoNLL 2003 (PERSON, LOCATION, ORGANIZATION, and MISC)