# Deep Learning for NLP

Instructor: Wei Xu
Ohio State University
CSE 5525

Many slides from Greg Durrett

# Outline

- Motivation for neural networks
- Feedforward neural networks
- Applying feedforward neural networks to NLP
- Convolutional neural networks
- Application examples
- Tools

# Sentiment Analysis

*the movie was very good* 👍

# Sentiment Analysis with Linear

| Example | Label | Feature | Type |
|---|---|---|---|
| *the movie was very good* | 👍 | $\mathbb{I}[\textit{good}]$ | Unigrams |
| *the movie was very bad* | 👎 | $\mathbb{I}[\textit{bad}]$ | Unigrams |
| *the movie was not bad* | 👍 | $\mathbb{I}[\textit{not bad}]$ | Bigrams |
| *the movie was not very good* | 👎 | $\mathbb{I}[\textit{not very good}]$ | Trigrams |
| *the movie was not really very enjoyable* | | | 4-grams! |

# Drawbacks

- More complex features capture interactions but scale badly (13M unigrams, 1.3B 4-grams in Google *n*-grams)
- Can we do better than seeing every *n*-gram once in the training data?

*not very good* *not so great*

- Instead of more complex linear functions, let's use *simpler nonlinear functions*, namely neural networks

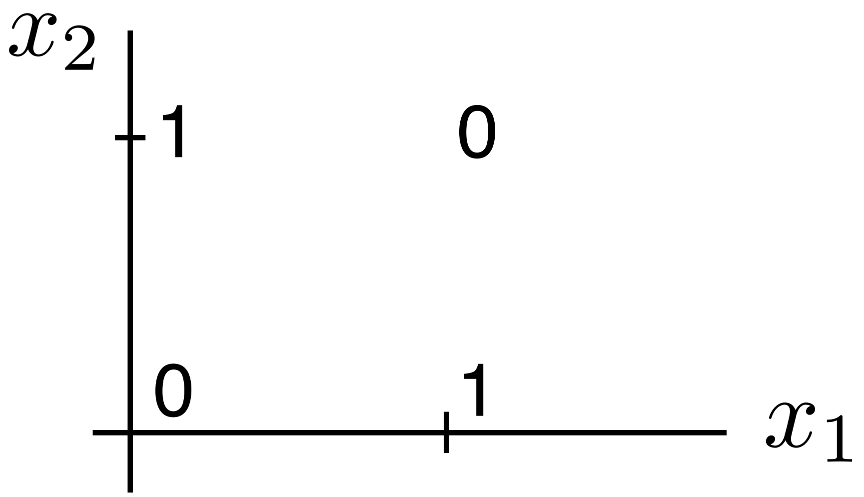
*the movie was not really very enjoyable*

# Neural Networks: XOR

- Let's see how we can use neural nets to learn a simple nonlinear function
- Inputs $x_1, \; x_2$

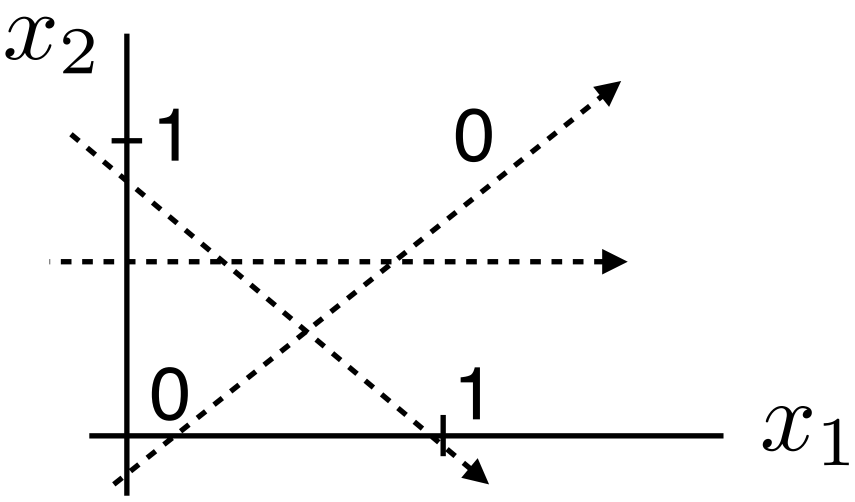
  $(\text{generally } \mathbf{x} = (x_1, \ldots, x_m))$
- Output $y$

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




| $x_1$ | $x_2$ | $y = x_1 \text{ XOR } x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# Neural Networks: XOR



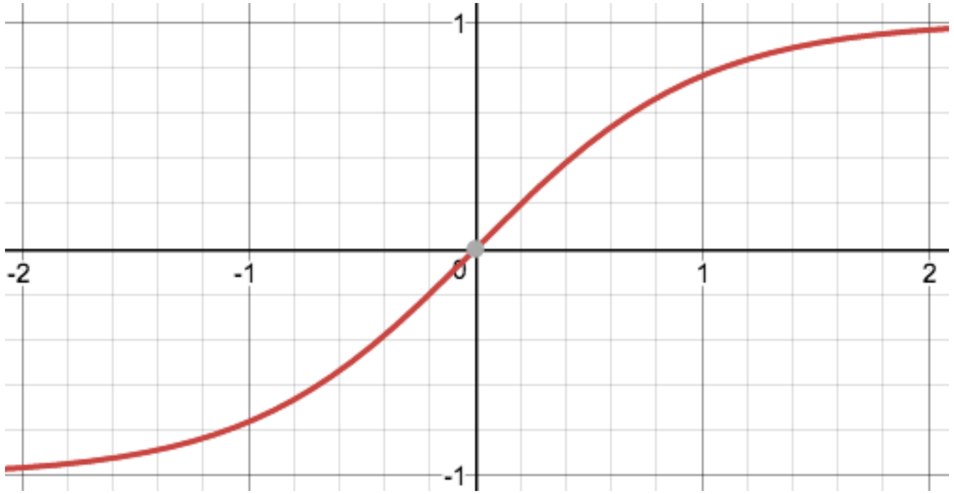

$y = a_1 x_1 + a_2 x_2$ ✗

$y = a_1 x_1 + a_2 x_2 + a_3 \tanh(x_1 + x_2)$ ✓

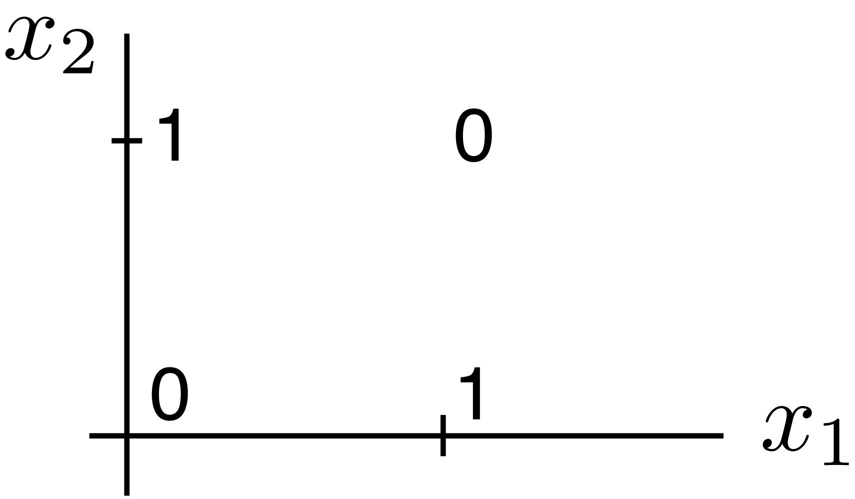
"or"

| $x_1$ | $x_2$ | $x_1$ XOR $x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

(looks like action potential in neuron)



# Neural Networks: XOR



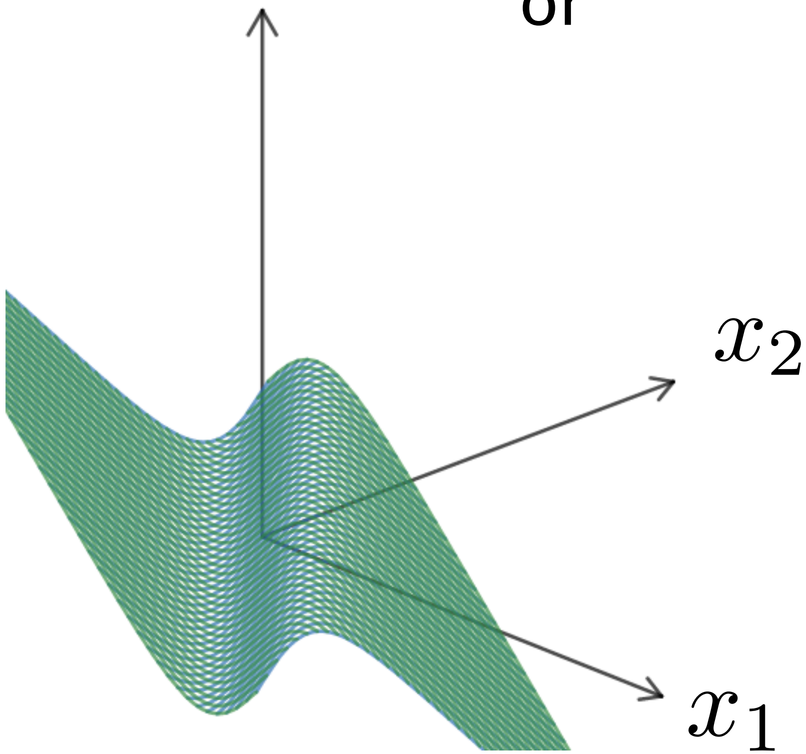

| $x_1$ | $x_2$ | $x_1$ XOR $x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$$y = a_1 x_1 + a_2 x_2$$ ✗

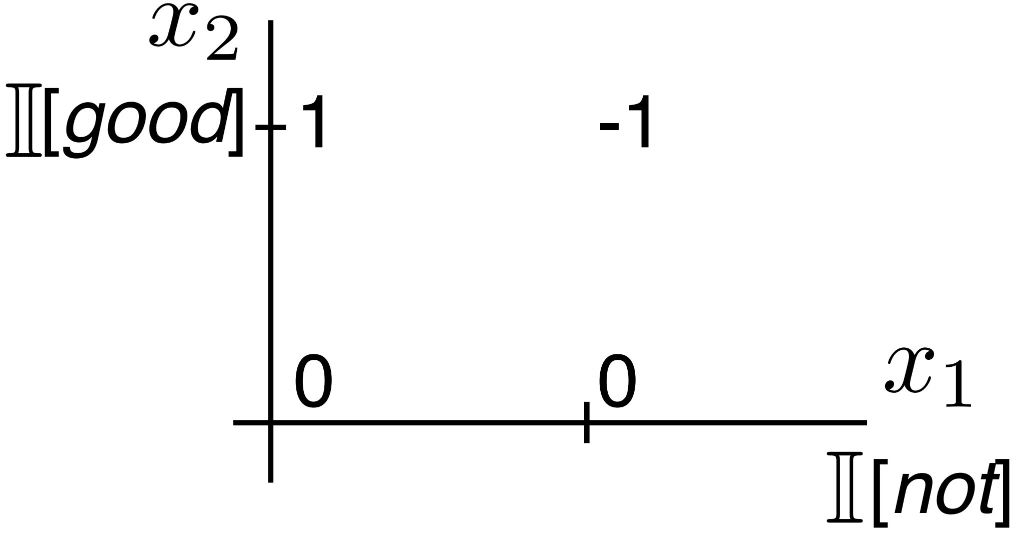
$$y = a_1 x_1 + a_2 x_2 + a_3 \tanh(x_1 + x_2)$$ ✓

$$y = -x_1 - x_2 + 2 \tanh(x_1 + x_2)$$

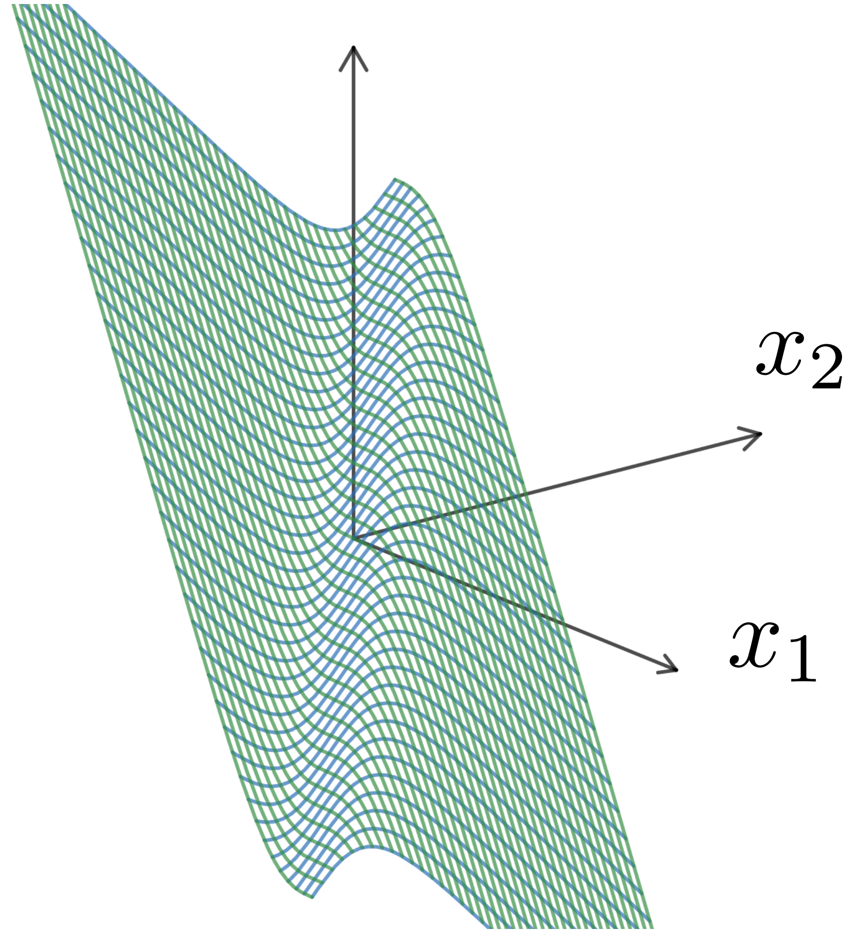
"or"



# Neural Networks: XOR




$$y = -2x_1 - x_2 + 2\tanh(x_1 + x_2)$$



# Neural Networks

(Linear model: $y = \mathbf{w} \cdot \mathbf{x} + b$)

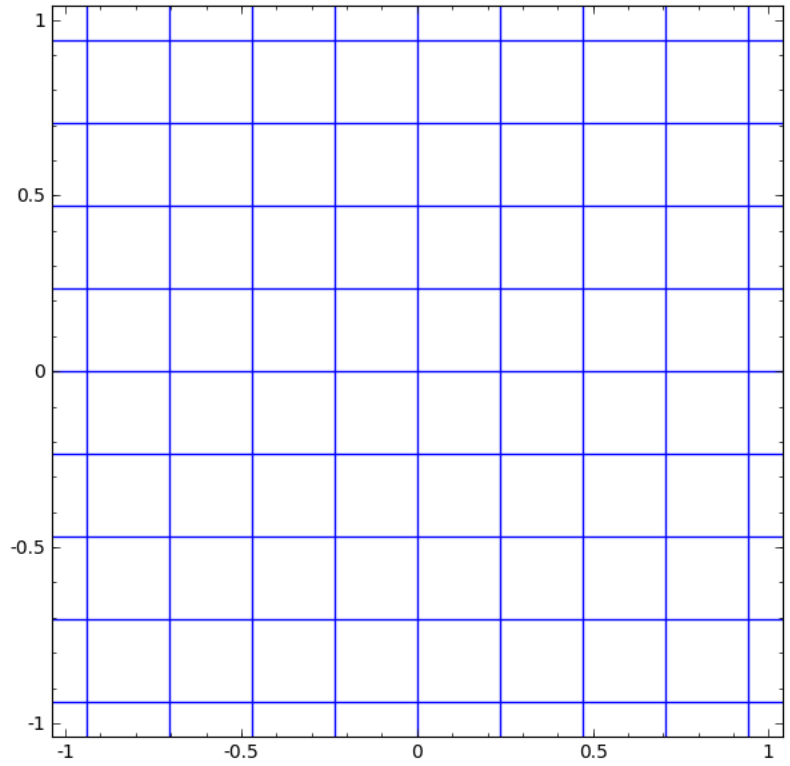
$$y = g(\mathbf{w} \cdot \mathbf{x} + b)$$

$$\mathbf{y} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Nonlinear transformation

Warp space

Shift



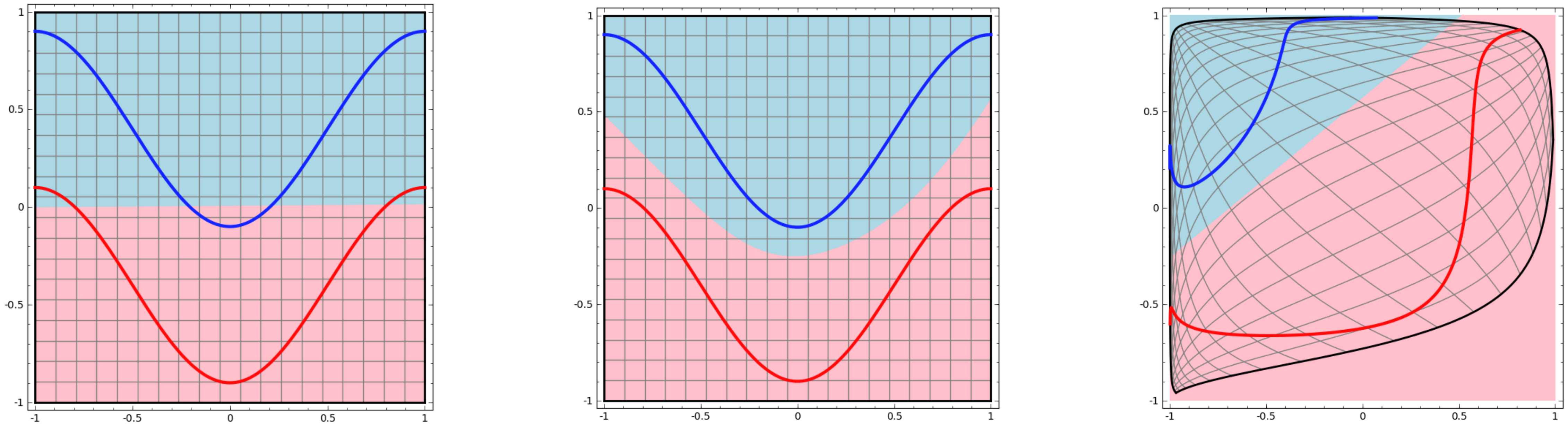

Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

# Neural Networks

Linear classifier

Neural network

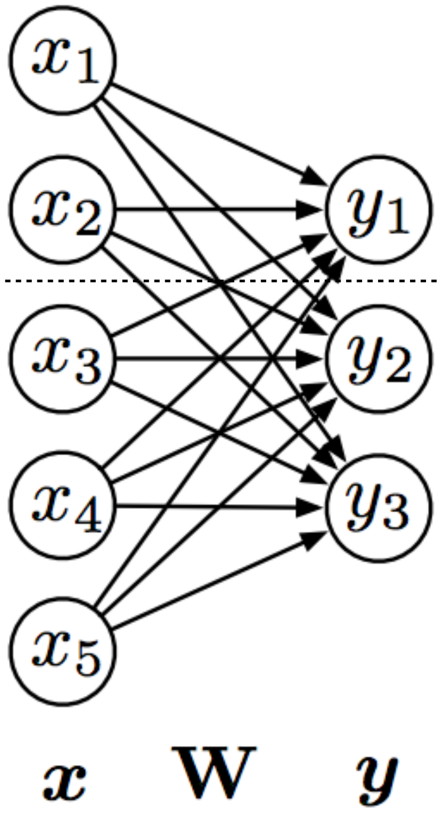
…possible because we transformed the space!



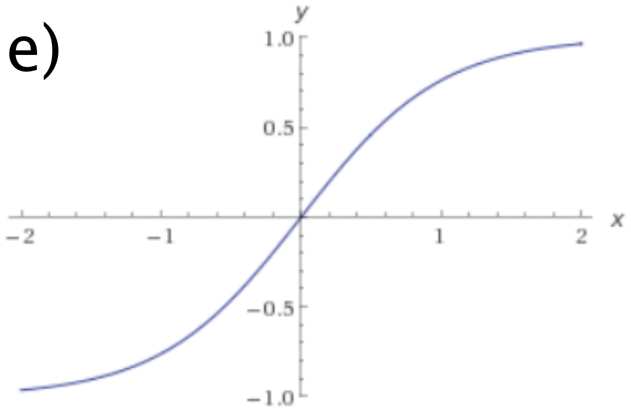

Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

# Deep Neural Networks

(this was our neural net from the XOR example)

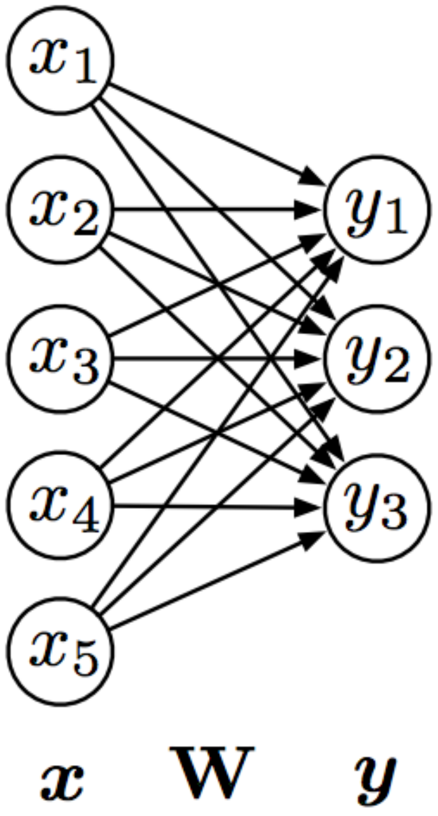



$$y_1 = g(\mathbf{w}_1 \cdot \mathbf{x} + b_1)$$

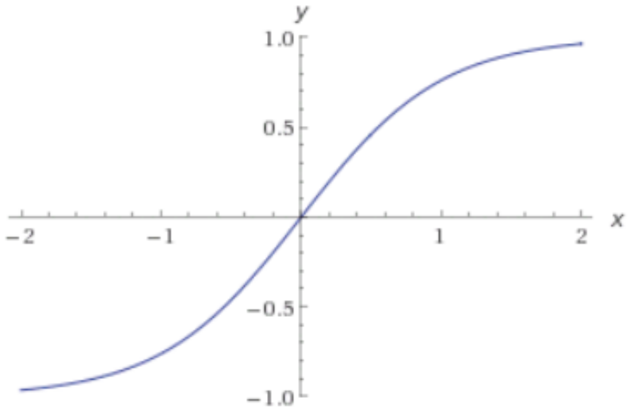



Adopted from Chris Dyer

# Deep Neural Networks



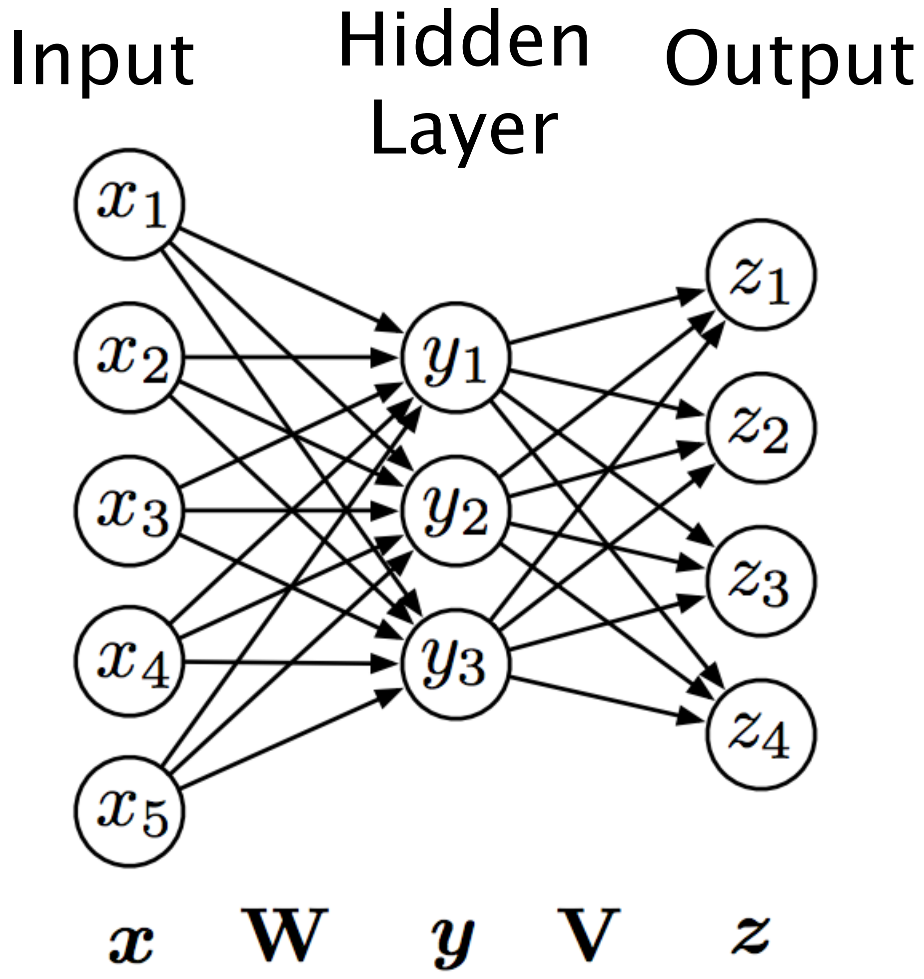

$$y_1 = g(\mathbf{w}_1 \cdot \mathbf{x} + b_1)$$




$$\boldsymbol{y} = g(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$$

Adopted from Chris Dyer

# Deep Neural Networks




$$\boldsymbol{y} = g(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$$

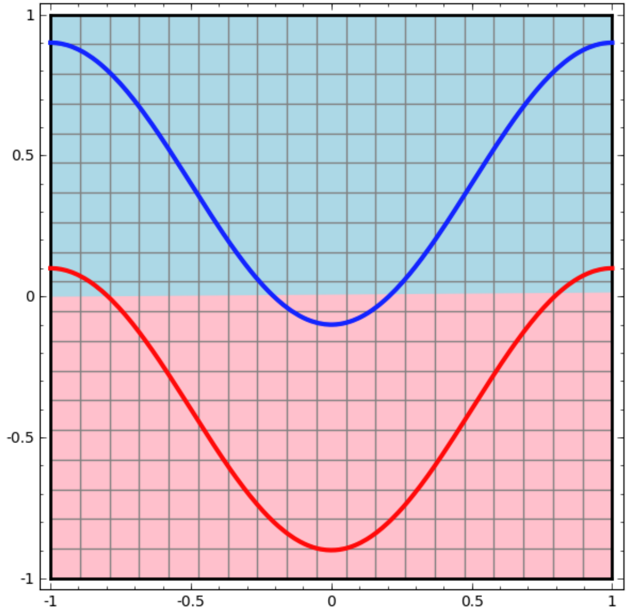
$$\mathbf{z} = g(\mathbf{V}\underbrace{g(\mathbf{W}\mathbf{x} + \mathbf{b})}_{\text{output of first layer}} + \mathbf{c})$$

$$\mathbf{z} = g(\mathbf{V}\mathbf{y} + \mathbf{c})$$

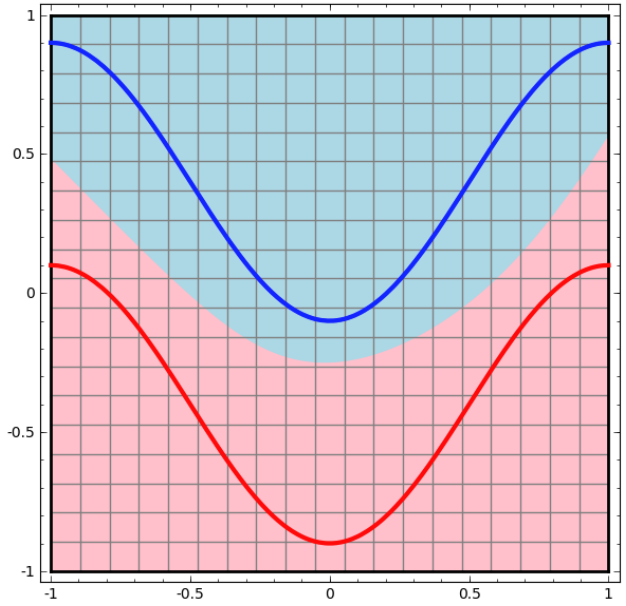
Adopted from Chris Dyer

# Neural Networks

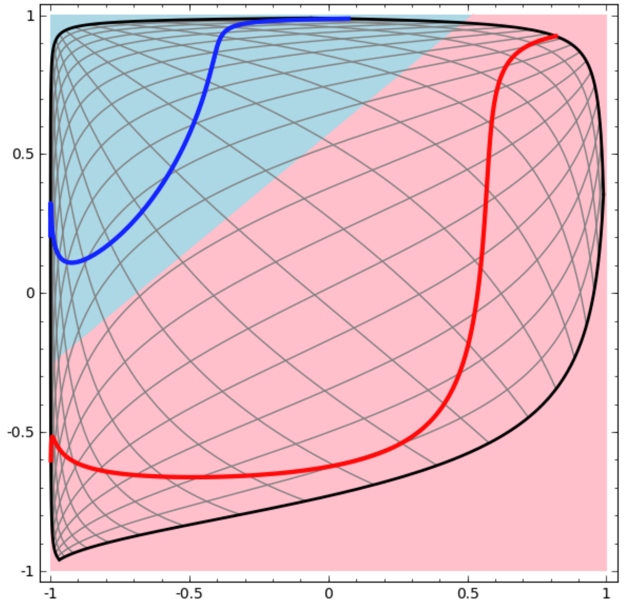
Linear classifier




Neural network

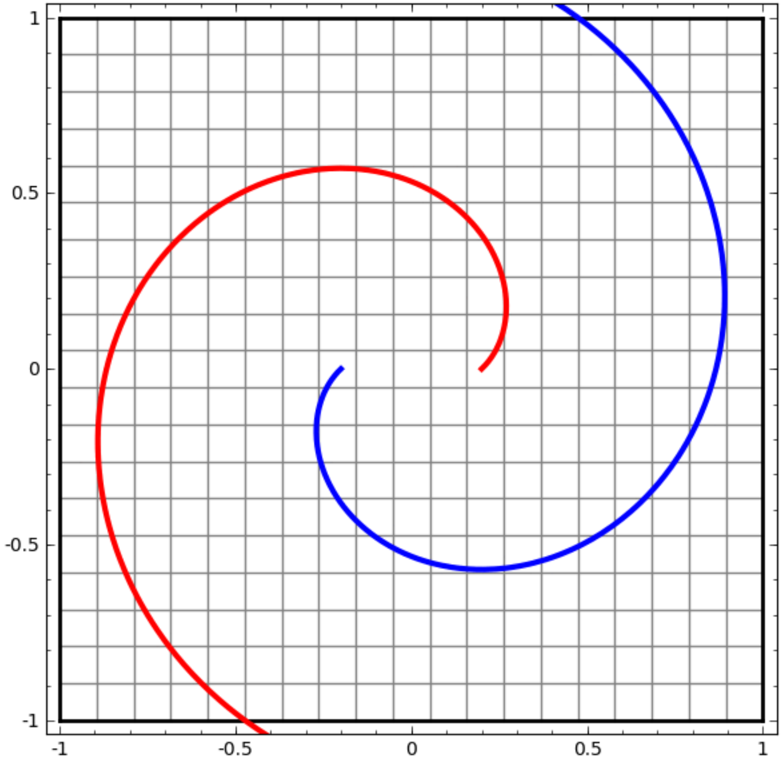



…possible because we transformed the space!



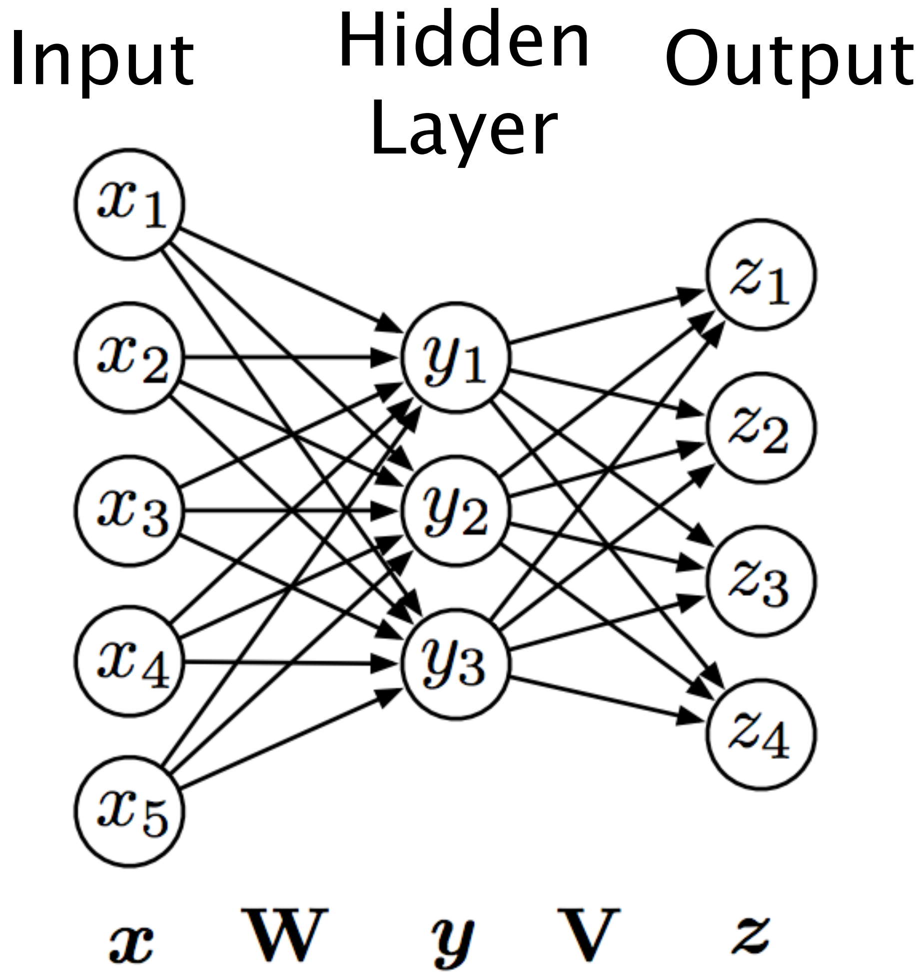

Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology

# Deep Neural Networks




Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

# Deep Neural Networks




$$\boldsymbol{y} = g(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$$

$$\mathbf{z} = g(\mathbf{V}\underbrace{g(\mathbf{W}\mathbf{x} + \mathbf{b})}_{\text{output of first layer}} + \mathbf{c})$$

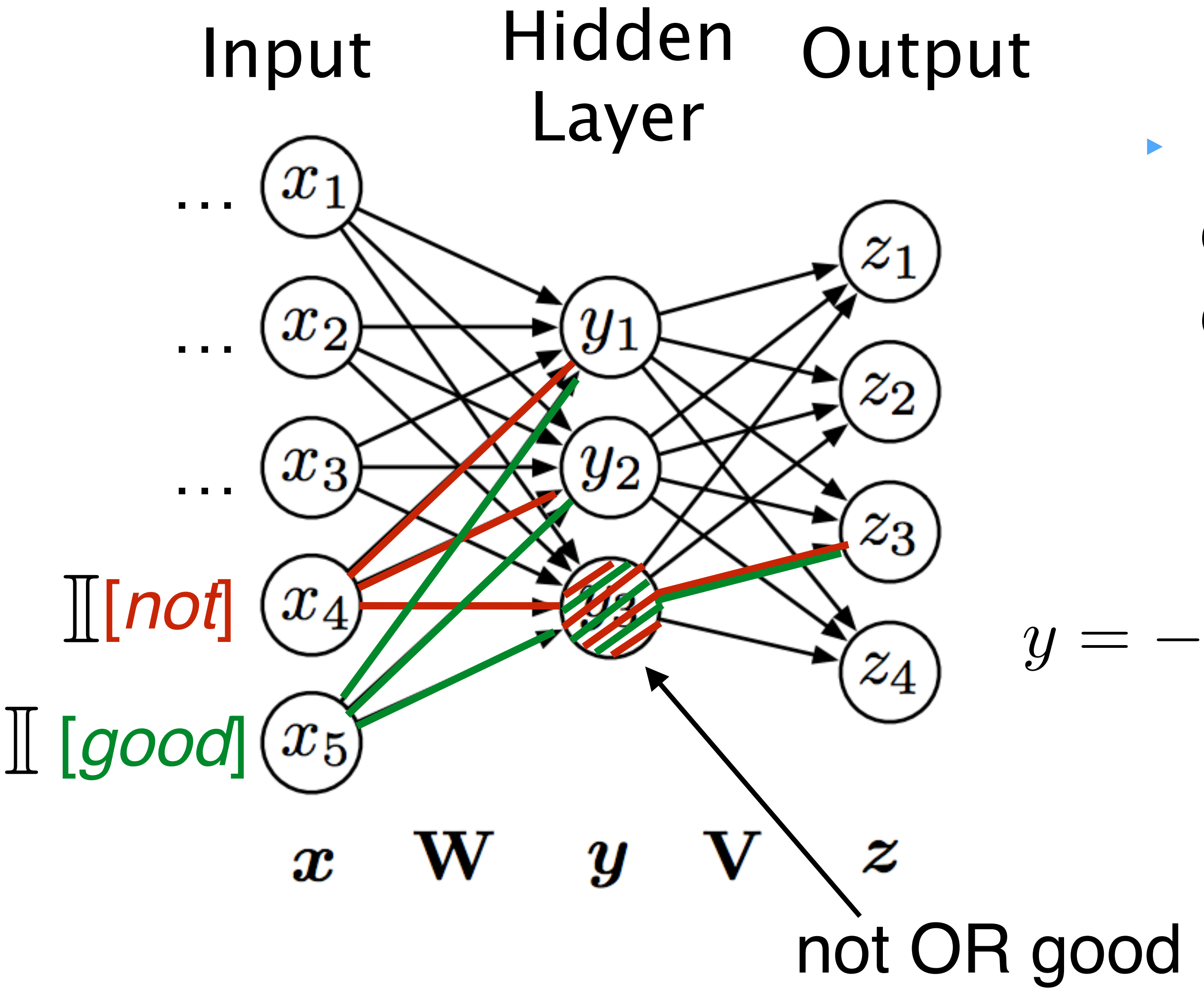
With no nonlinearity:

$$\mathbf{z} = \mathbf{VWx} + \mathbf{Vb} + \mathbf{c}$$

Equivalent to $\mathbf{z} = \mathbf{Ux} + \mathbf{d}$

Adopted from Chris Dyer

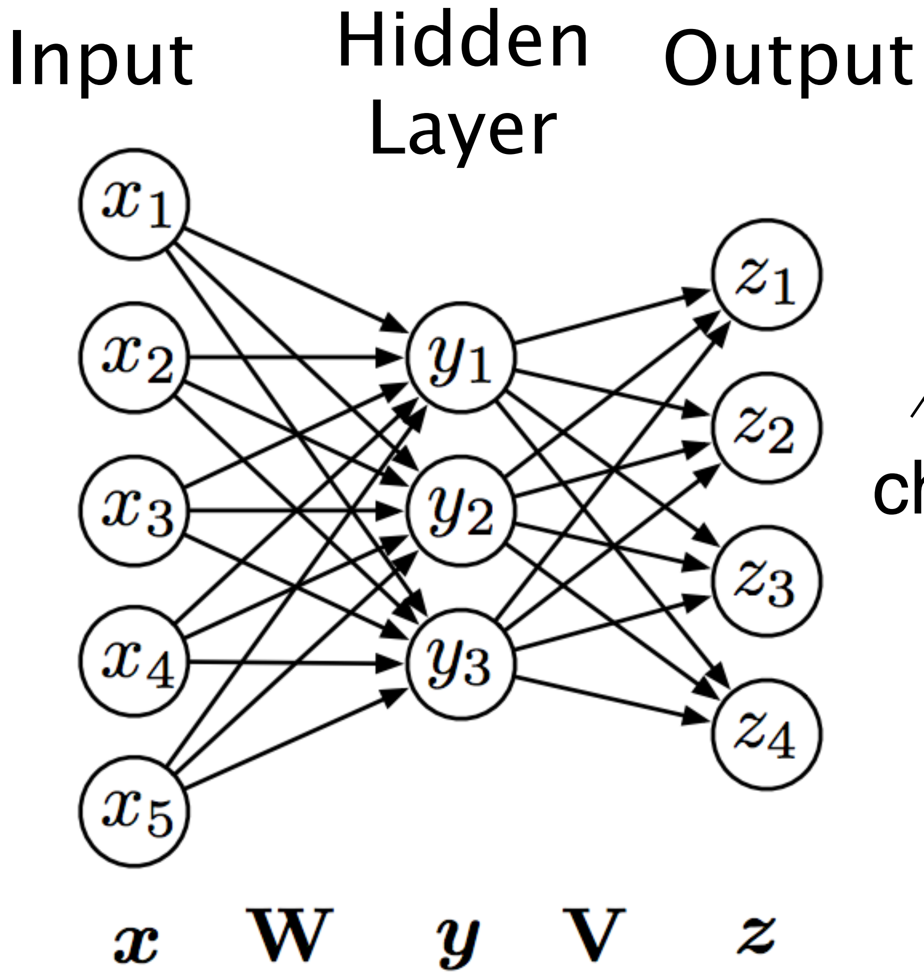
# Deep Neural Networks




- Nodes in the hidden layer can learn interactions or conjunctions of features

$$y = -2x_1 - x_2 + 2\tanh(x_1 + x_2)$$

# Learning Neural Networks




change in output w.r.t. hidden

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

change in hidden w.r.t. input

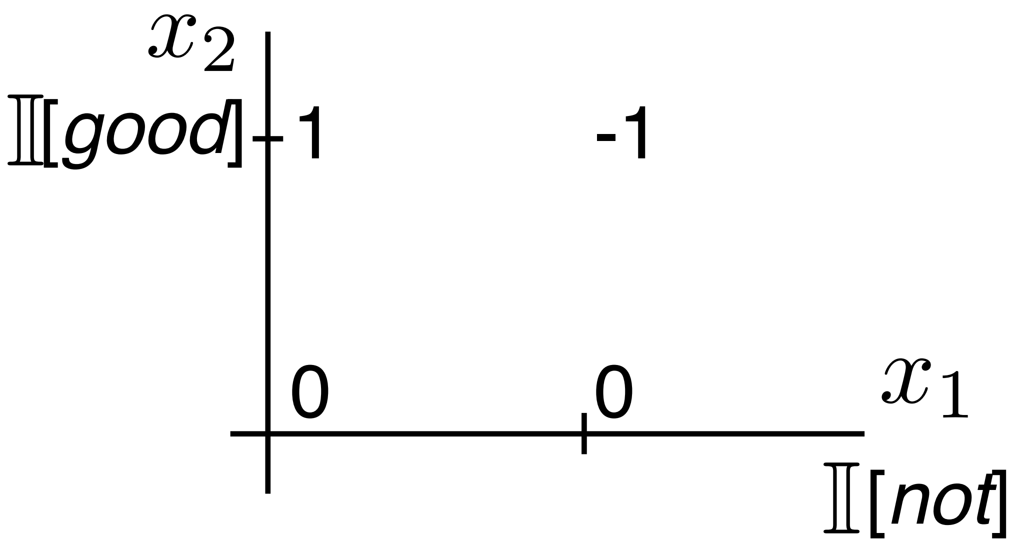
change in output w.r.t. input

- Computing these looks like running this network in reverse (backpropagation)

# Outline

- Motivation for neural networks
- Feedforward neural networks
- Applying feedforward neural networks to NLP
- Convolutional neural networks
- Application examples
- Tools

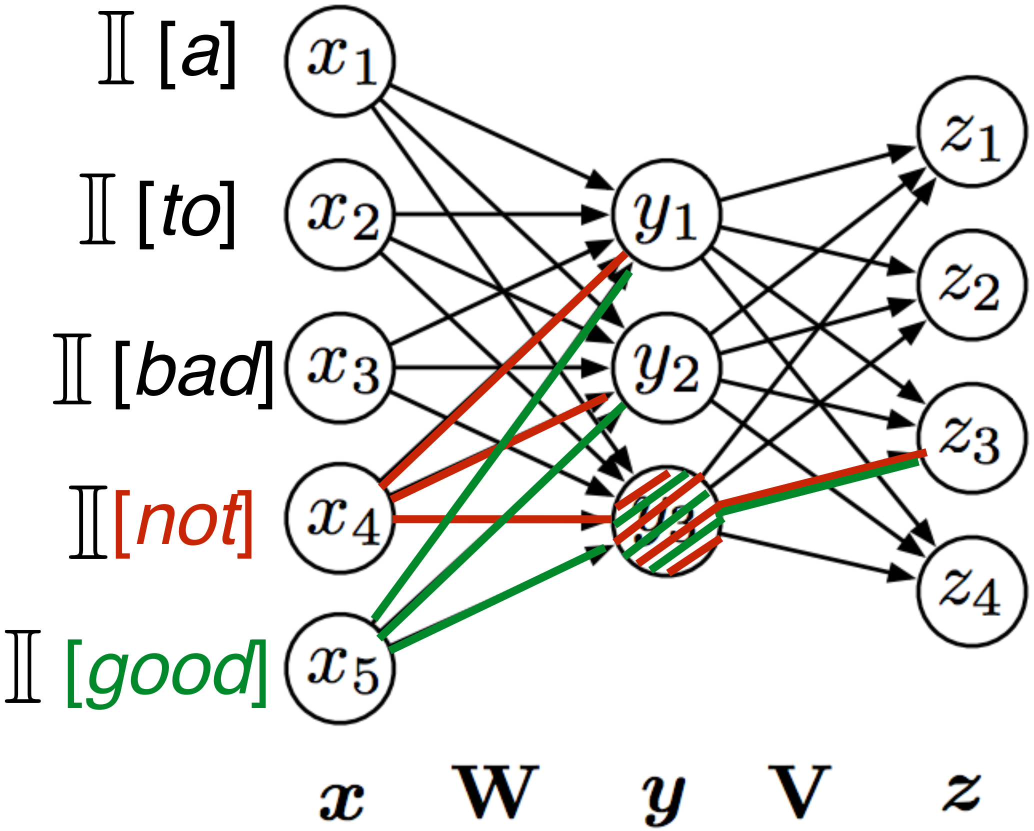
# Feedforward Bag-of-words




$$\mathbf{y} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

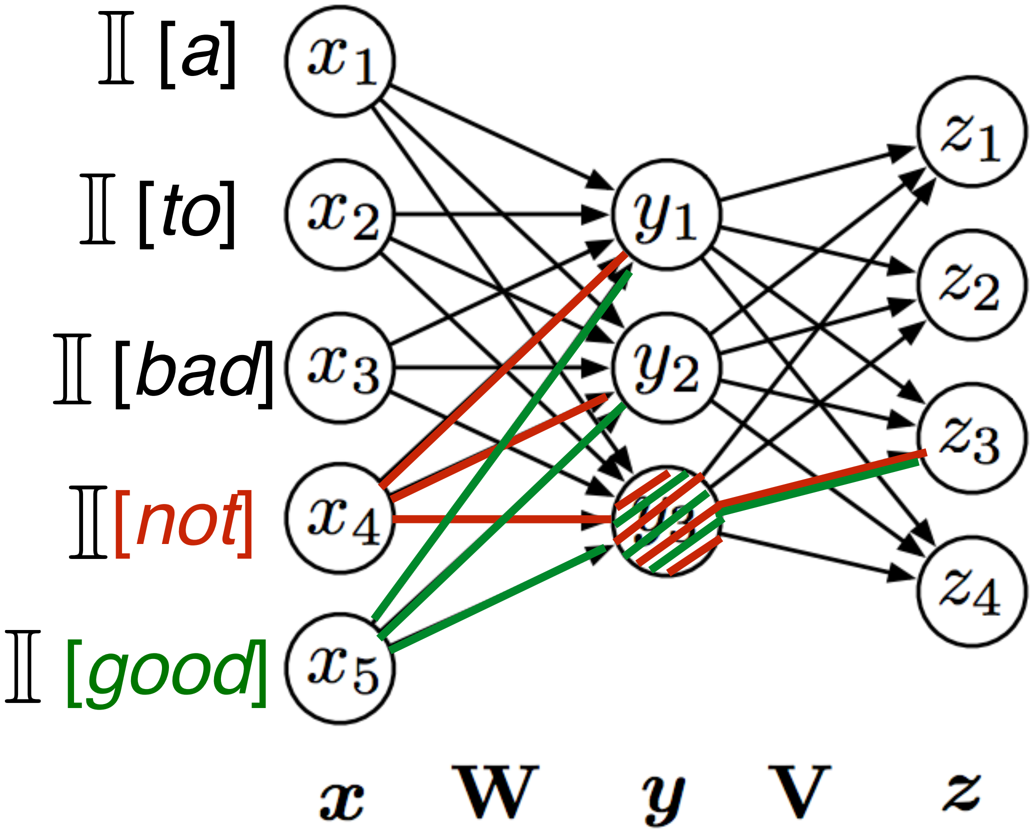
real-valued matrix,
dims = vocabulary size (~10k) x
hidden layer size (~100)

binary vector,
length = vocabulary size



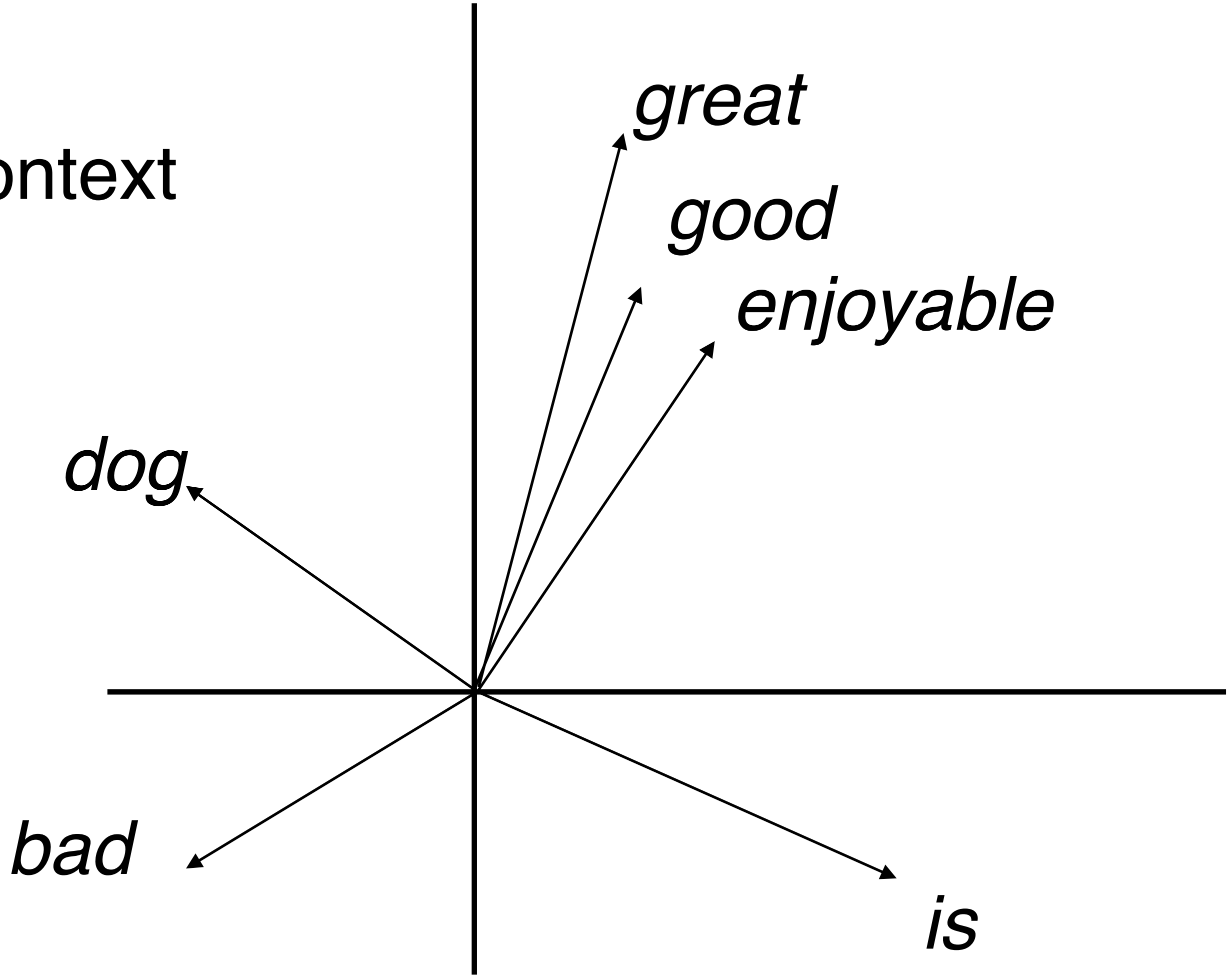
# Drawbacks to FFBoW

- Lots of parameters to learn
- Doesn't preserve ordering in the input
- *really not very good* and *really not very enjoyable* — we don't know the relationship between *good* and *enjoyable*

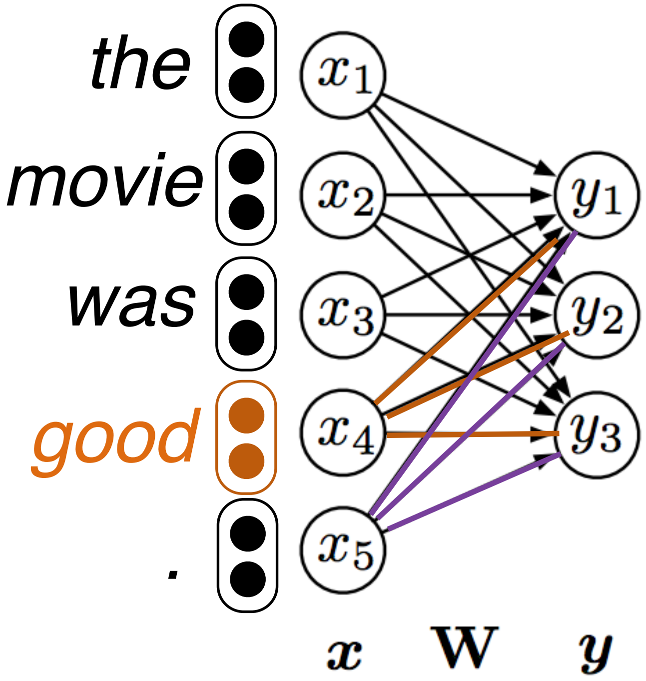


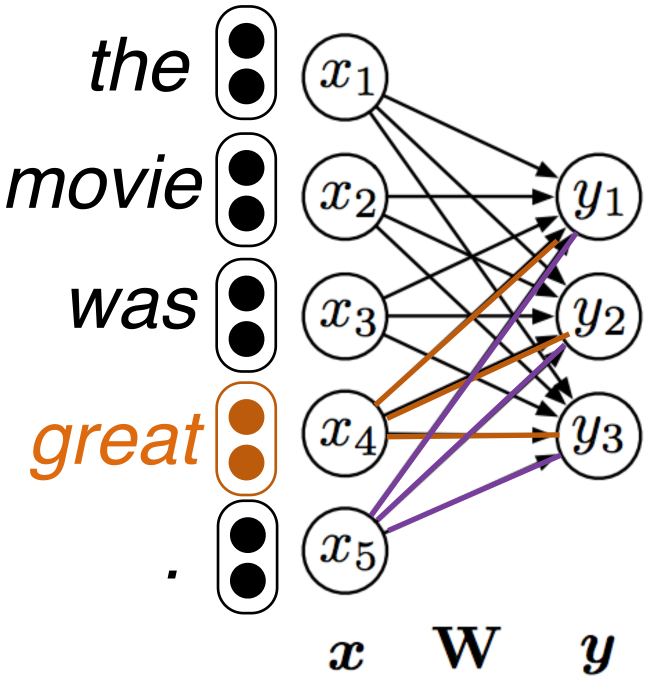
# Word Embeddings

- word2vec: turn each word into a 100-dimensional vector
- Context-based embeddings: find a vector predictive of a word's context
- Words in similar contexts will end up with similar vectors



# Feedforward with word vectors



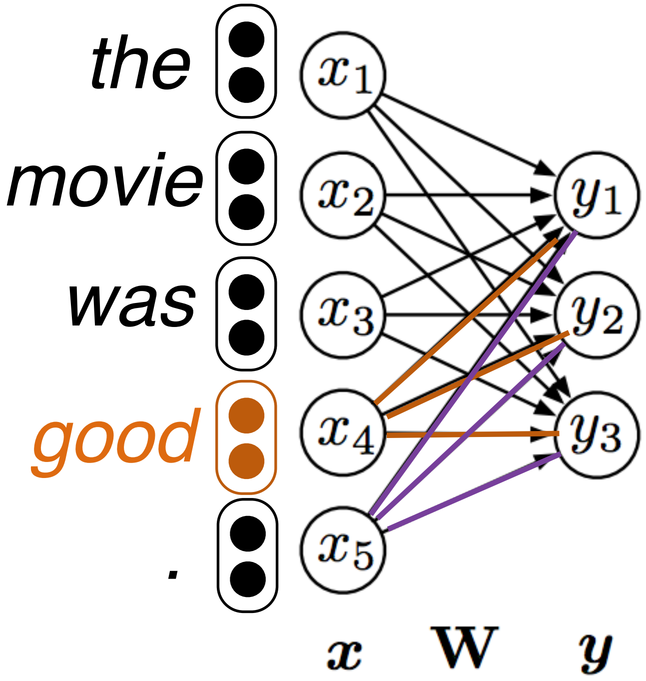

$$\mathbf{y} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

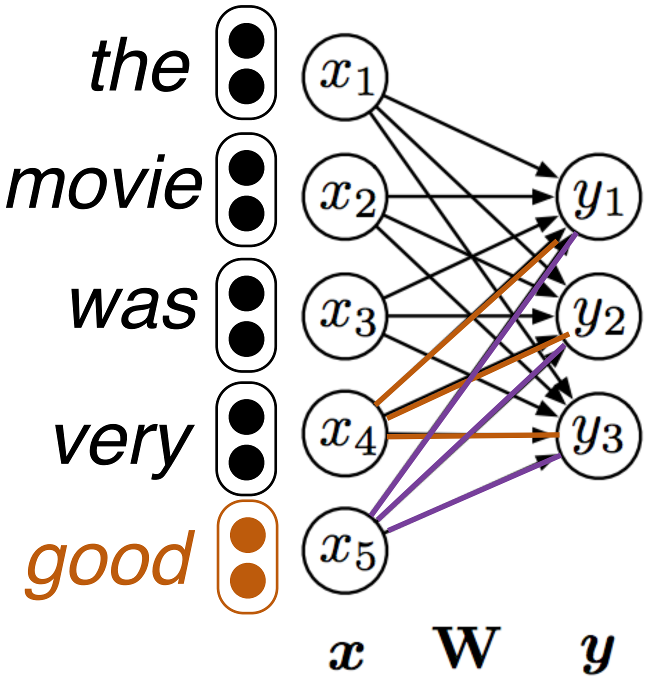
hidden layer size ~100 x (sentence length (~10) x vector size (~100))

binary vector,
length = sentence length x vector size

- Each *x* now represents multiple bits of input
- Can capture word similarity

# Feedforward with word vectors




$$\mathbf{y} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

- Need our model to be shift-invariant, like bag-of-words is

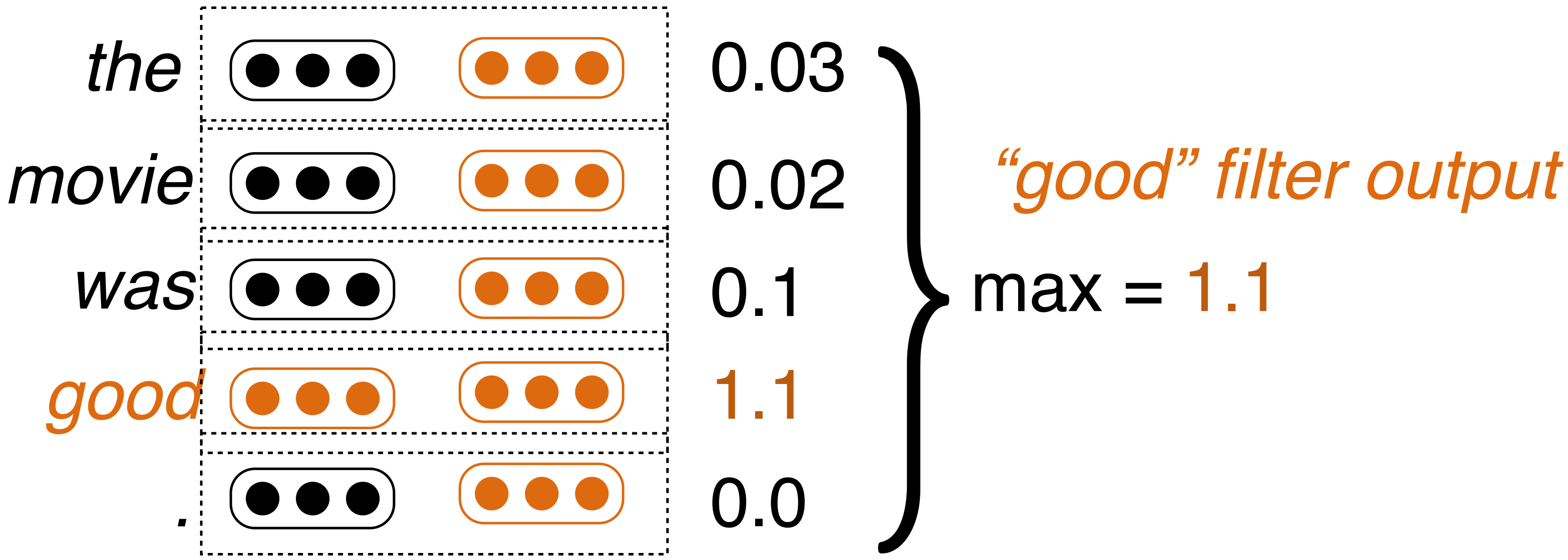
# Comparing Architectures

- Instead of more complex linear functions, let's use *simpler nonlinear functions*
- Feedforward bag-of-words: didn't take advantage of word similarity, lots of parameters to learn
- Feedforward with word vectors: our parameters are attached to particular indices in a sentence
- Solution: *convolutional* neural nets

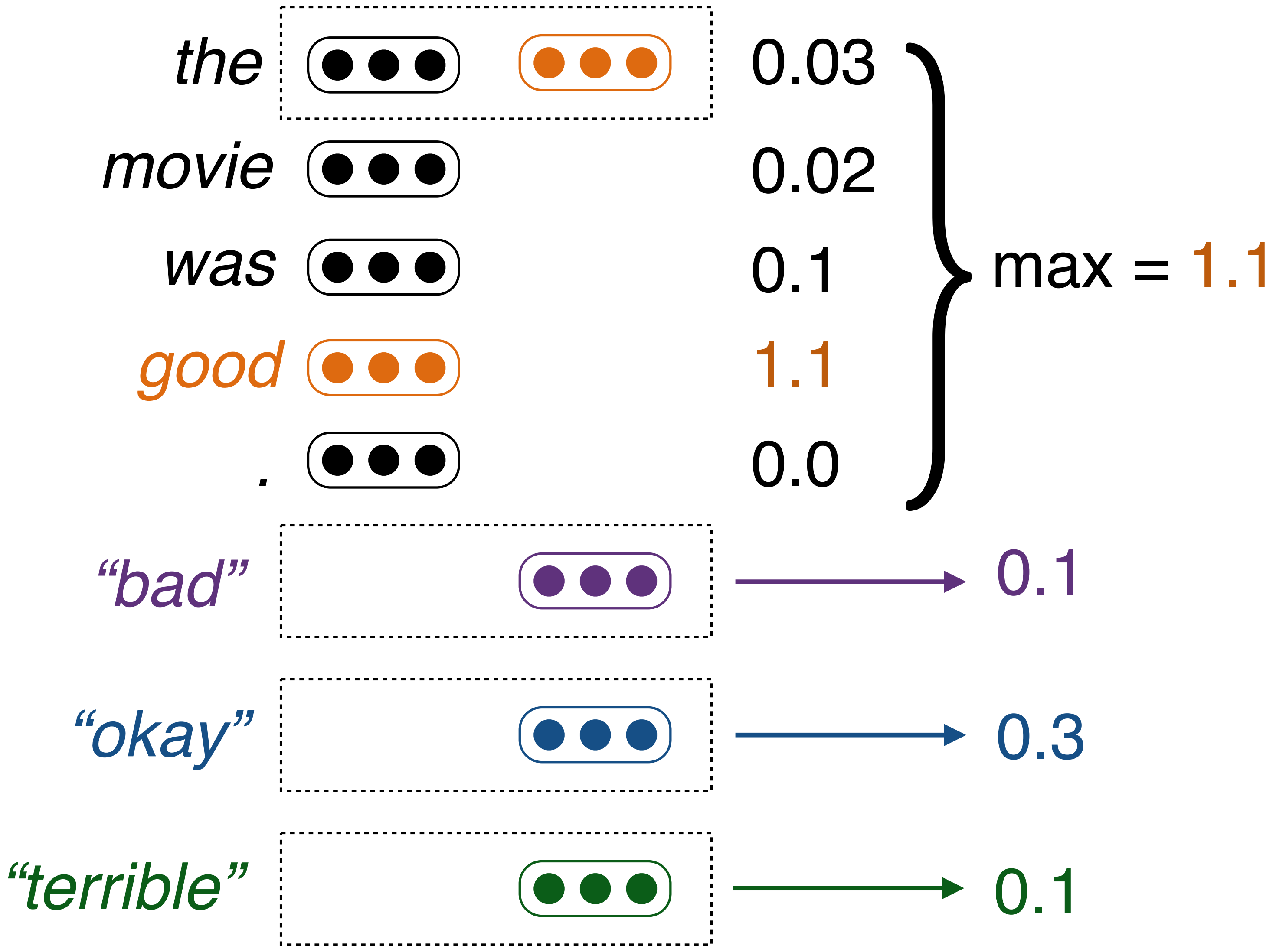
# Outline

- Motivation for neural networks
- Feedforward neural networks
- Applying feedforward neural networks to NLP
- Convolutional neural networks
- Application examples
- Tools

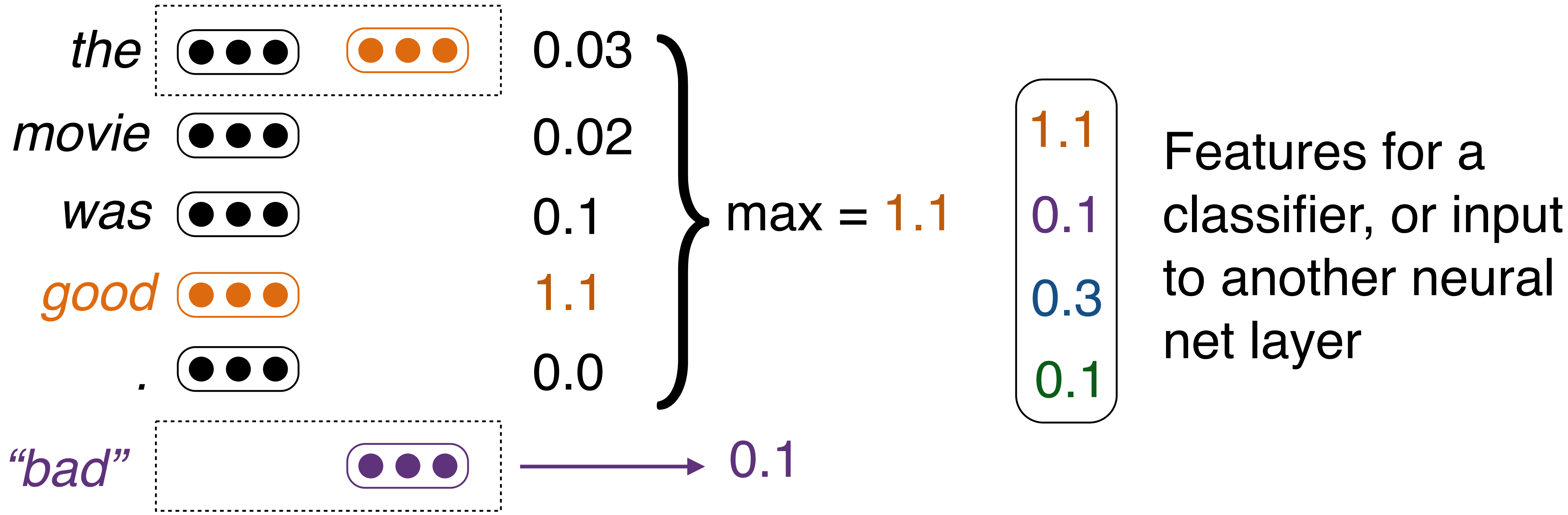
# Convolutional Networks



# Convolutional Networks

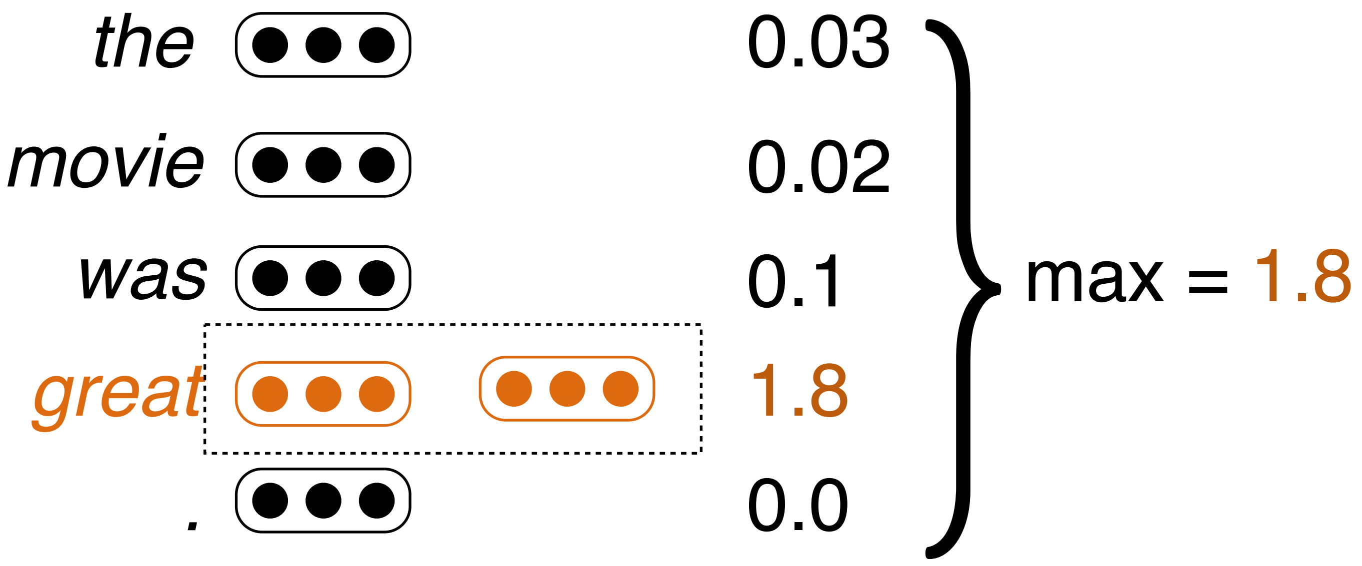


# Convolutional Networks



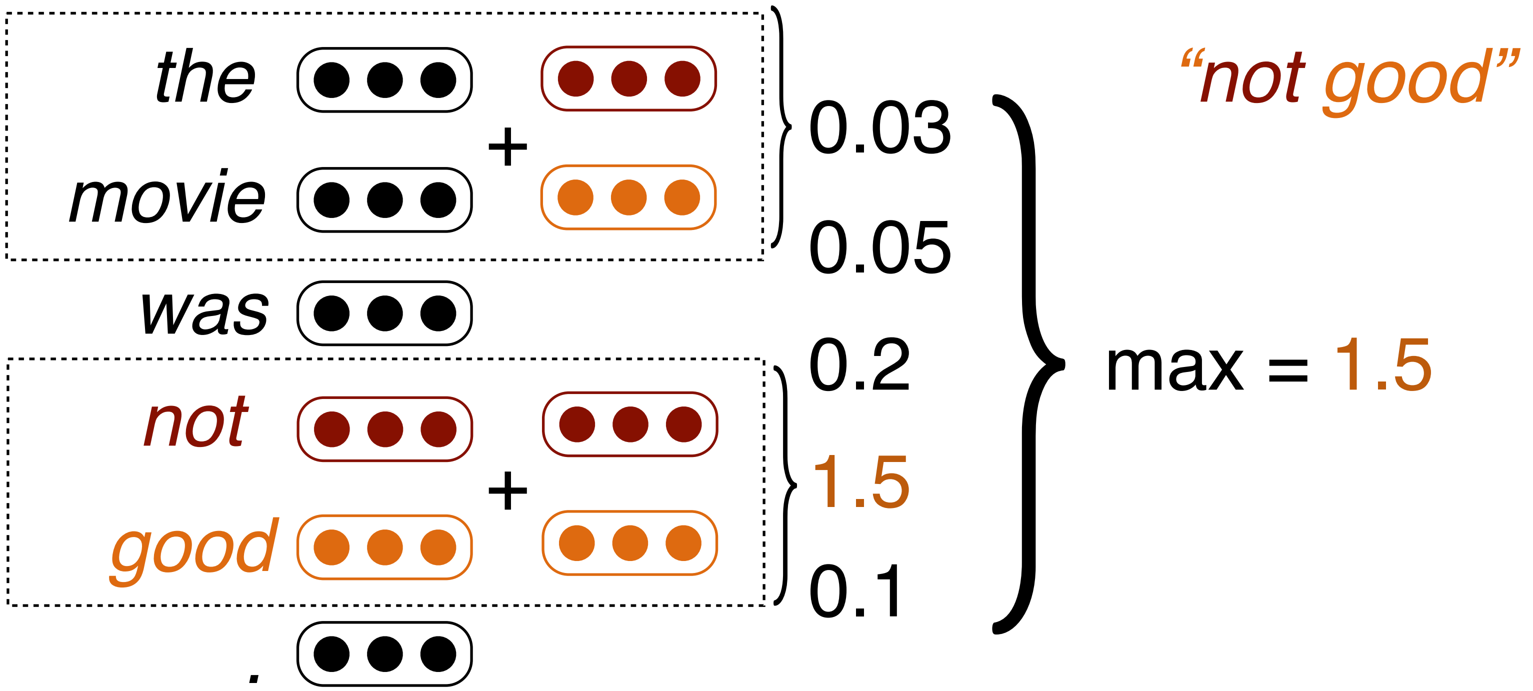

- Input: $n$ vectors of length $m$ each
  $k$ filters of length $m$ each → $k$ filter outputs of length 1 each
- Takes variable-length input and turns it into fixed-length output
- Filters are initialized randomly and then learned

# Convolutional Networks




- Word vectors for similar words are similar, so convolutional filters will have similar outputs

# Convolutional Networks




- Analogous to bigram features in bag-of-words models

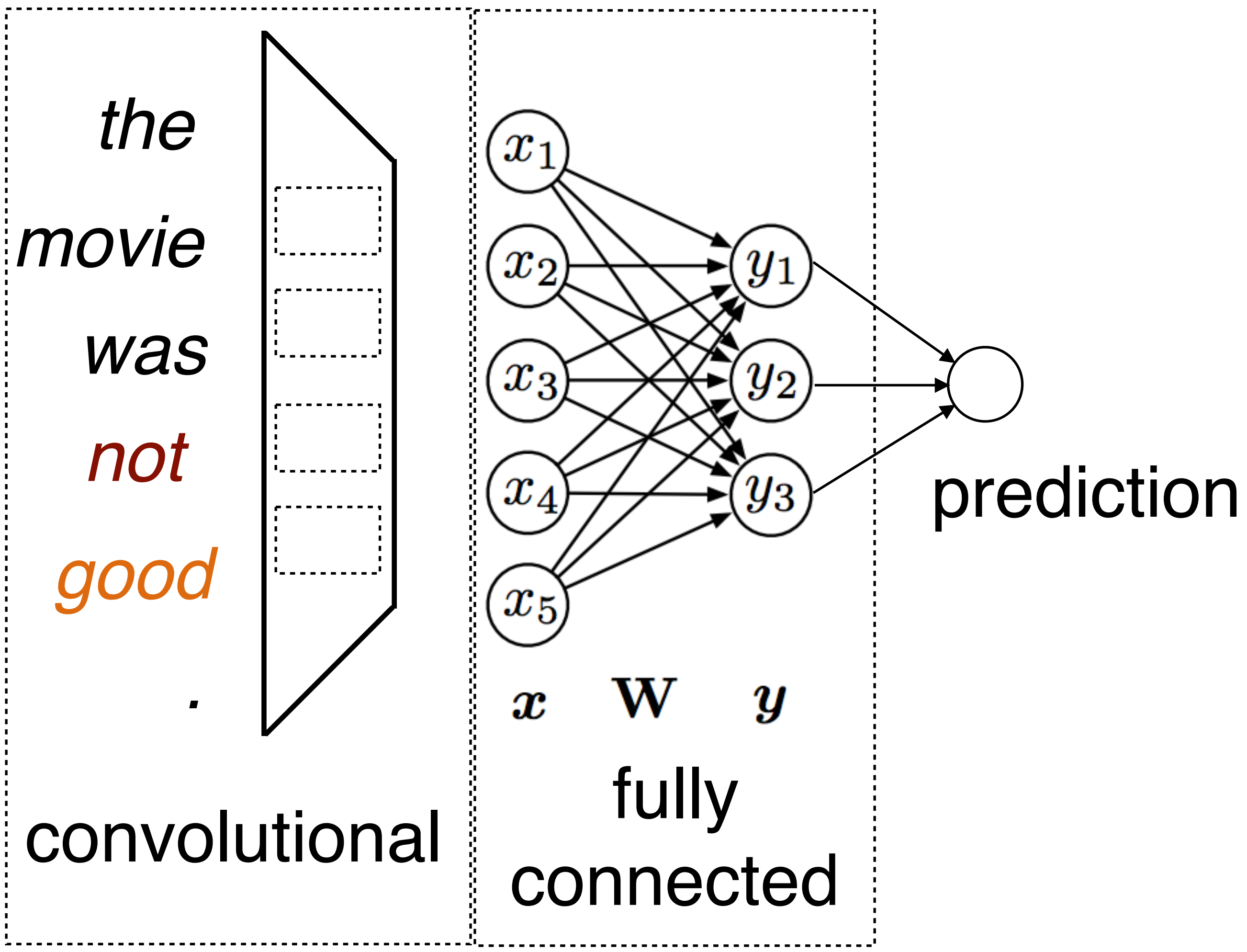
# Comparing Architectures

- Instead of more complex linear functions, let's use *simpler nonlinear functions* ✓
- Convolutional networks let us take advantage of word similarity ✓
- Convolutional networks are translation-invariant like bag-of-words ✓
- Convolutional networks can capture local interactions with filters of width > 1 (i.e. *"not good"* ) ✓

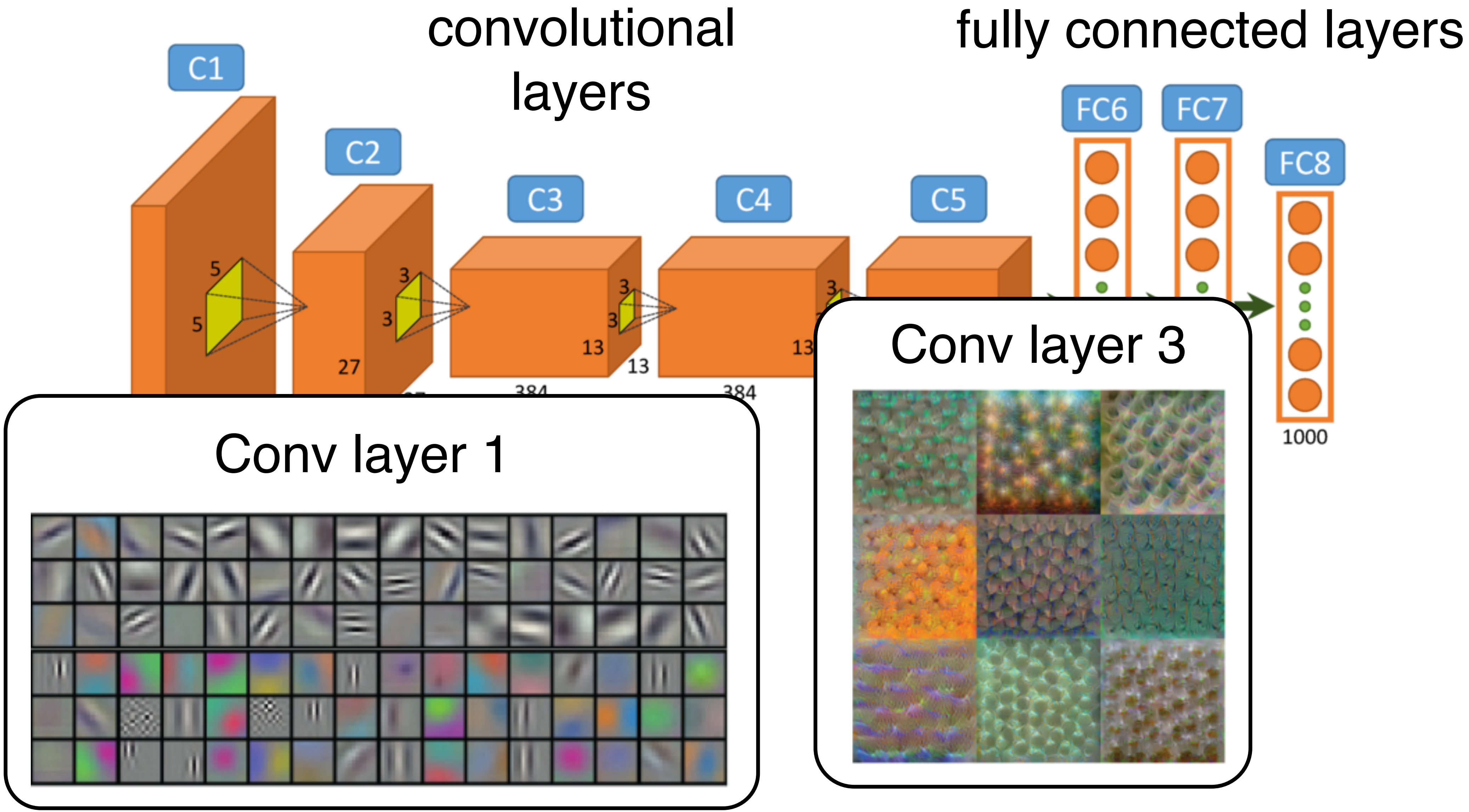
# Outline

- Motivation for neural networks
- Feedforward neural networks
- Applying feedforward neural networks to NLP
- Convolutional neural networks
- Application examples
- Tools

# Sentence Classification



# Object Recognition

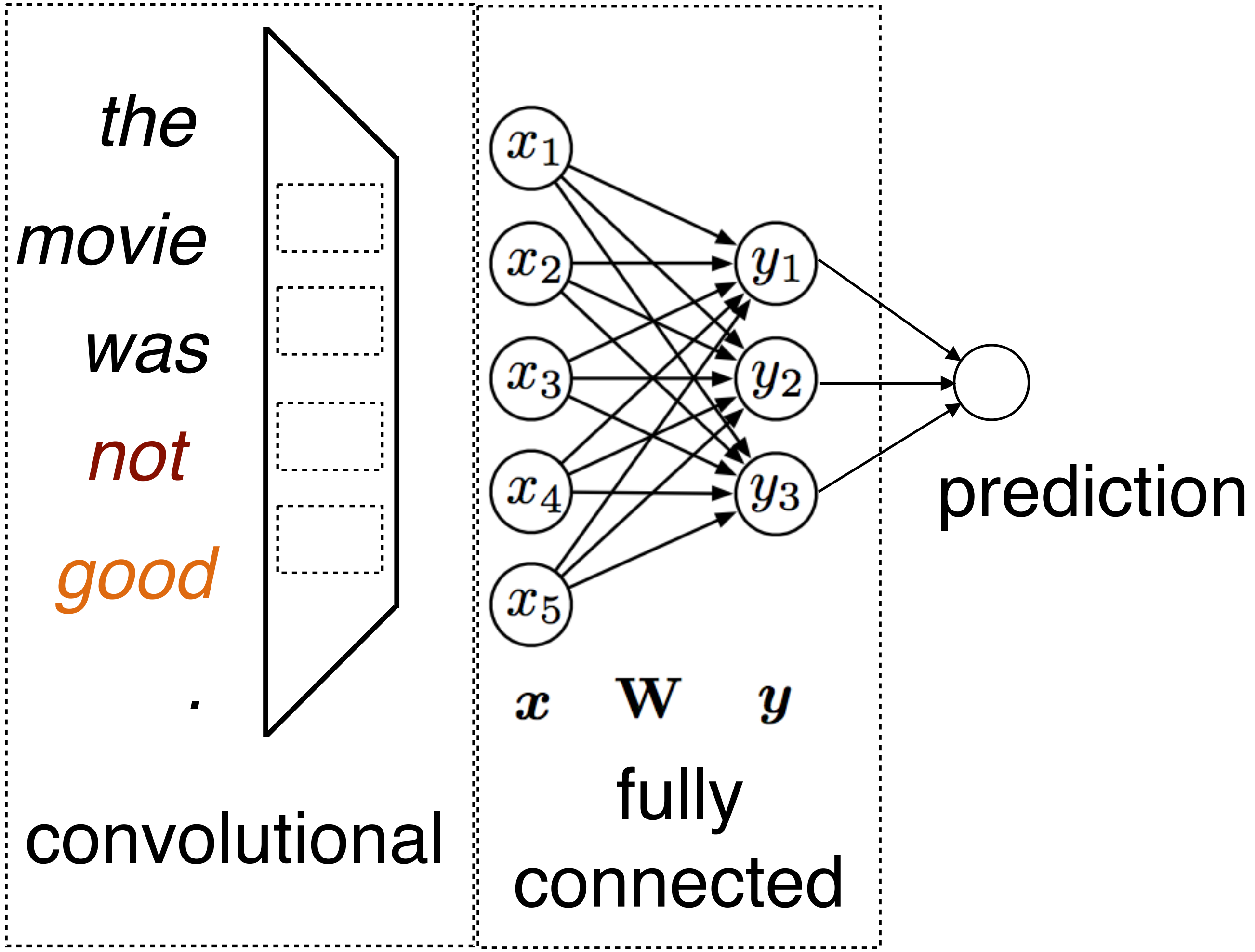



AlexNet (2012)

# Neural networks are

- NNs are built from convolutional layers, fully connected layers, and some other types
- Can chain these together into various architectures
- Any neural network built this way can be learned from data!

# Sentence Classification



# Sentence Classification

movie review sentiment

subjectivity/objectivity detection

product reviews

| Model | MR | SST-1 | SST-2 | Subj | TREC | CR | MPQA |
|---|---|---|---|---|---|---|---|
| CNN-multichannel | 81.1 | 47.4 | **88.1** | 93.2 | 92.2 | **85.0** | 89.4 |
| NBSVM (Wang and Manning, 2012) | 79.4 | – | – | 93.2 | – | 81.8 | 86.3 |

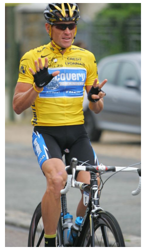
question type classification

- Outperforms highly-tuned bag-of-words model

Taken from Kim (2014)

# Entity Linking

Although he originally won the event, the United States Anti-Doping Agency announced in August 2012 that they had disqualified **Armstrong** from his seven consecutive Tour de France wins from 1999–2005.



Lance Edward Armstrong is an American former professional road cyclist

Armstrong County is a county in Pennsylvania…

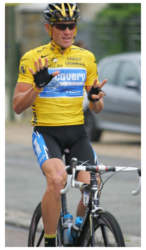
? ?

- Conventional: compare vectors from tf-idf features for overlap
- Convolutional networks can capture many of the same effects: distill notions of topic from n-grams

Francis-Landau, Durrett, and Klein (NAACL 2016)

# Entity Linking

Although he originally won the event, the United States Anti-Doping Agency announced in August 2012 that they had disqualified **Armstrong** from his seven consecutive Tour de France wins from 1999–2005.



Lance Edward Armstrong is an American former professional road cyclist

Armstrong County is a county in Pennsylvania…

convolutional

convolutional

convolutional

topic vector

topic vector

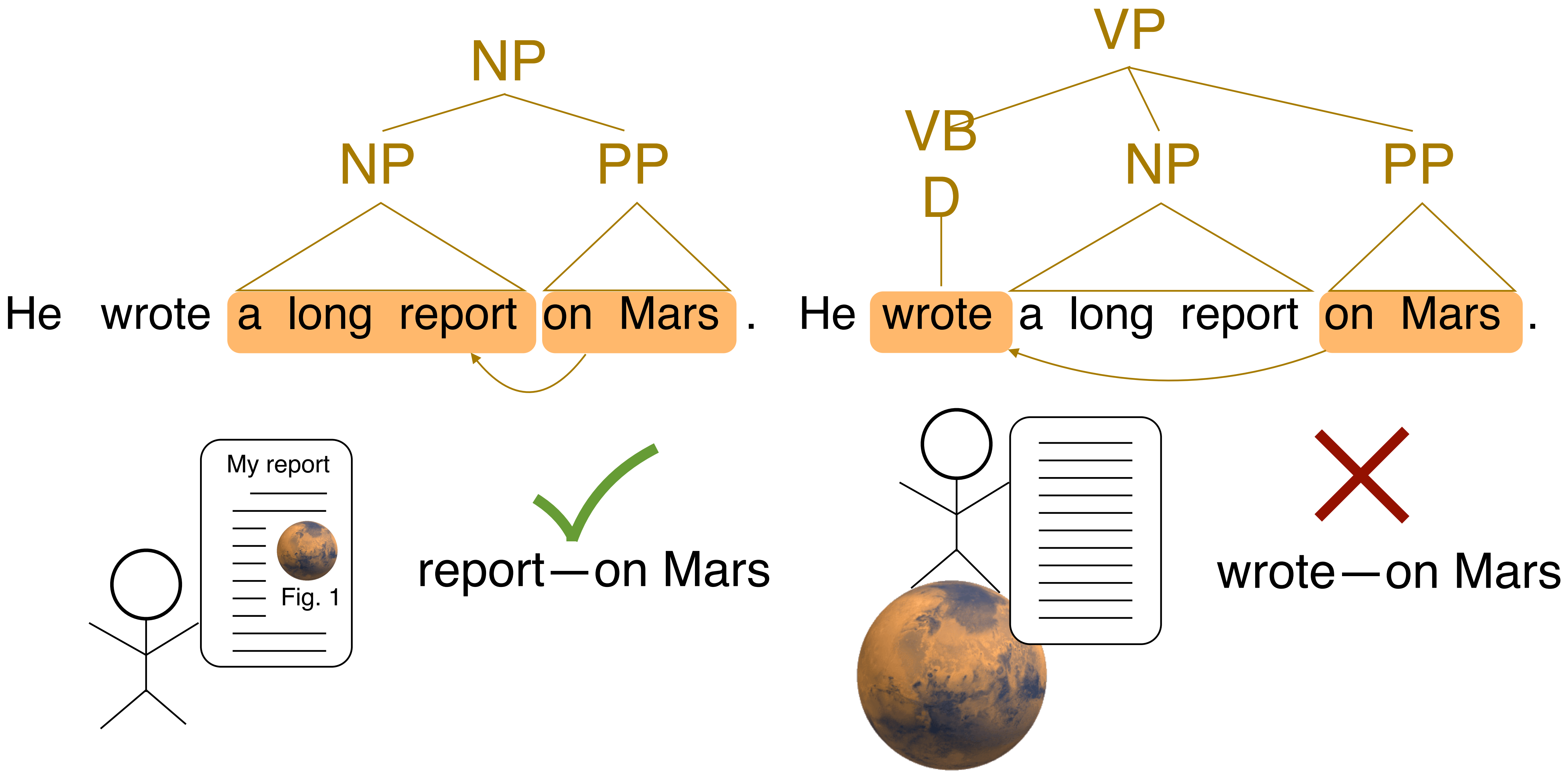
topic vector

similar — probable link

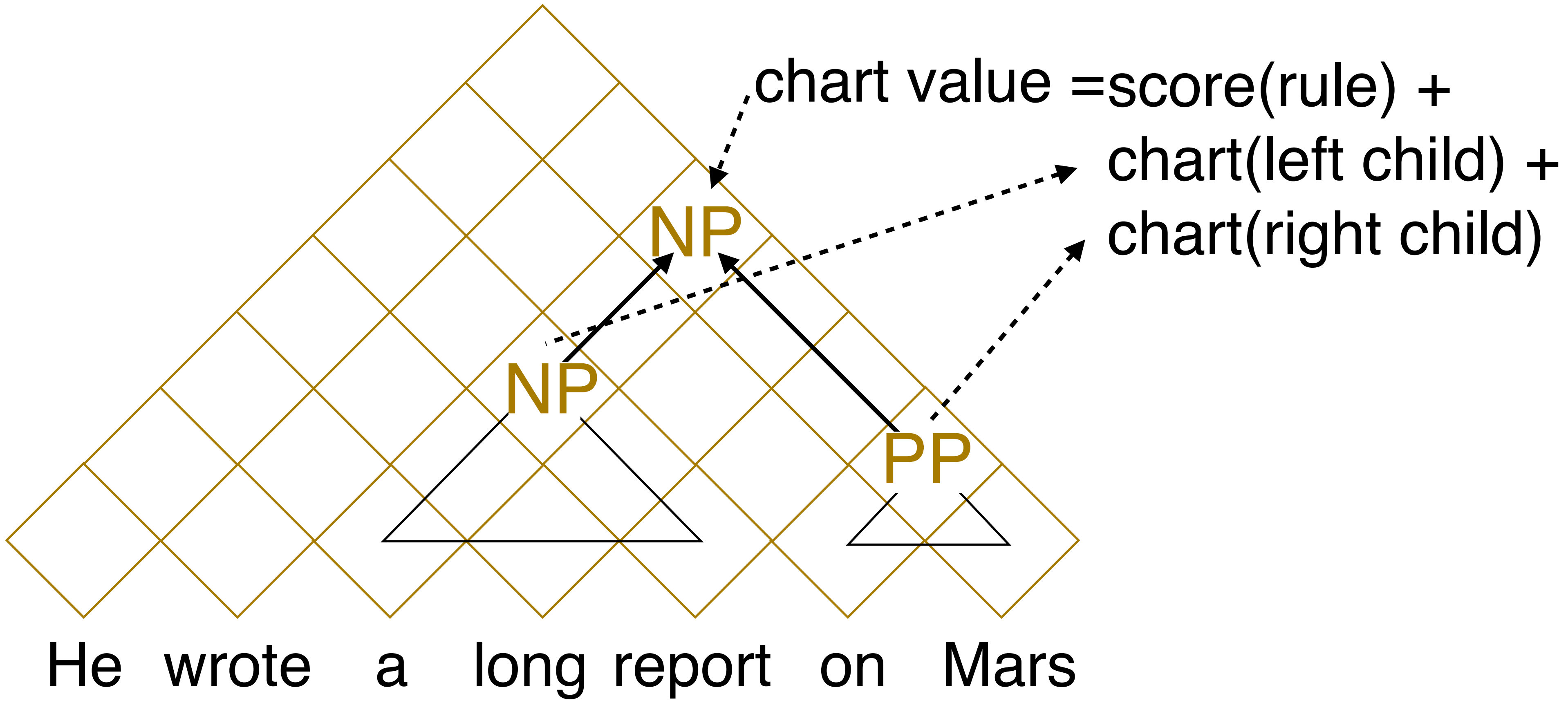
dissimilar — improbable link

Francis-Landau, Durrett, and Klein (NAACL 2016)

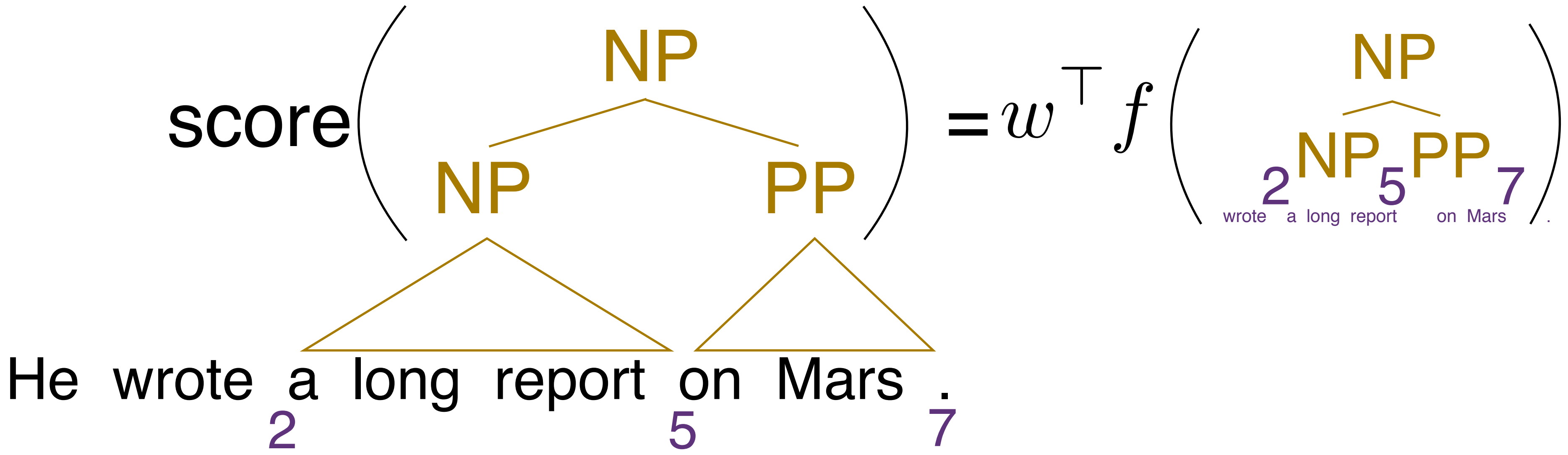
# Syntactic Parsing



# Chart Parsing



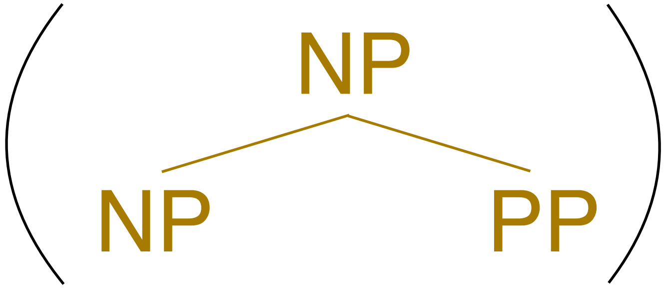
# Syntactic Parsing



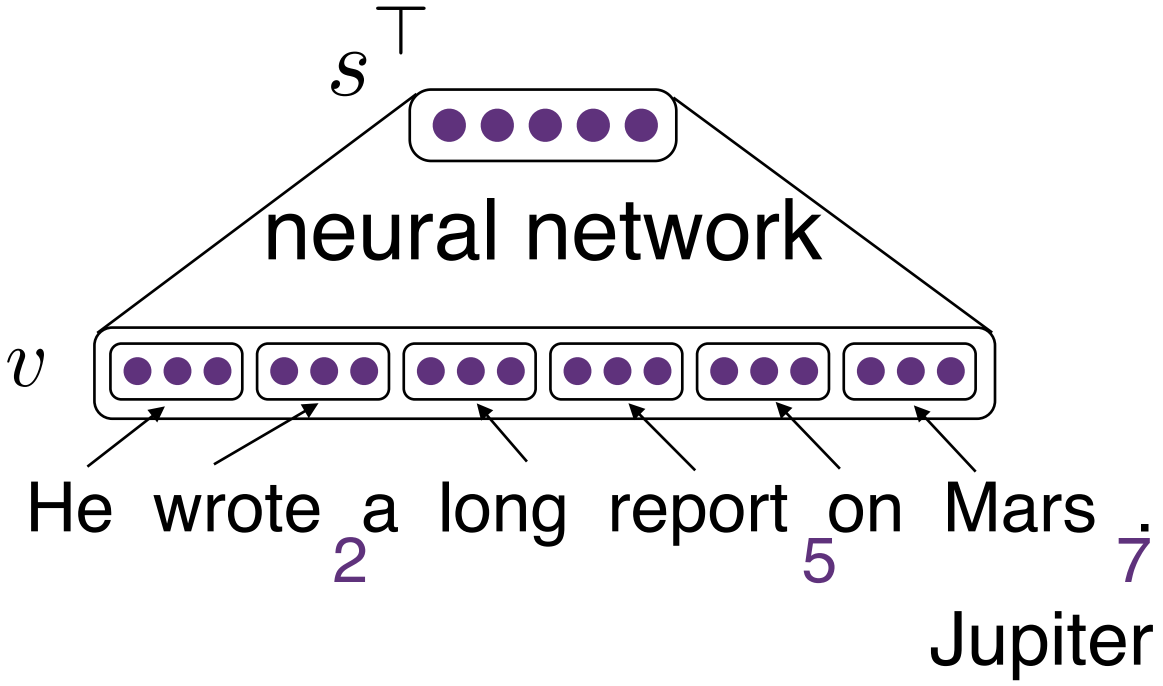

feat = $\mathbb{I}\left[\text{Left child last word} = \textit{report} \wedge \text{NP} \rightarrow \text{NP PP}\right]$

- Features need to combine surface information and syntactic information, but looking at words directly ends up being very sparse

# Scoring parses with neural nets

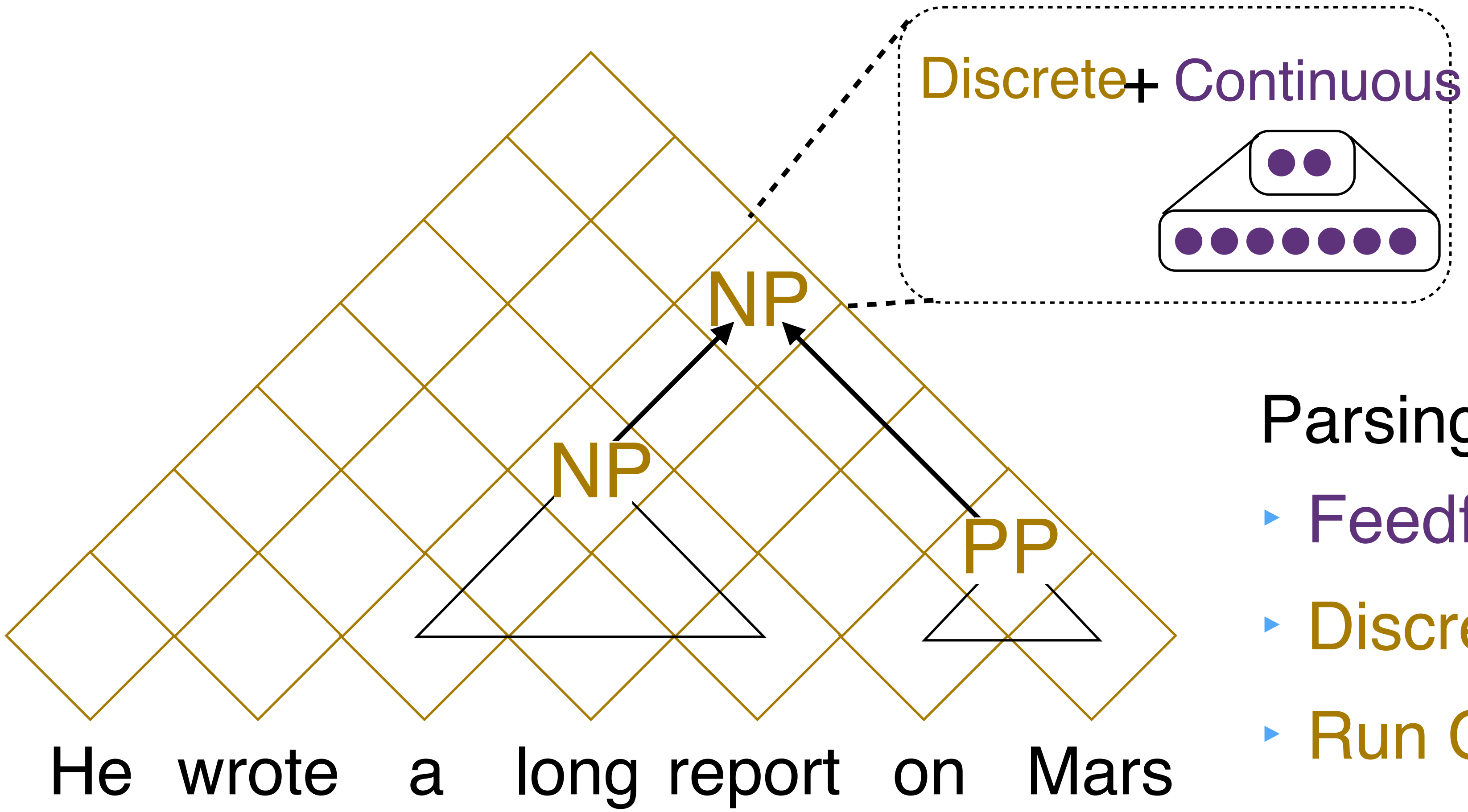
score( 

) = $s^\top \cdot$ vector representation of rule being applied




Durrett and Klein (ACL 2015)

# Syntactic Parsing

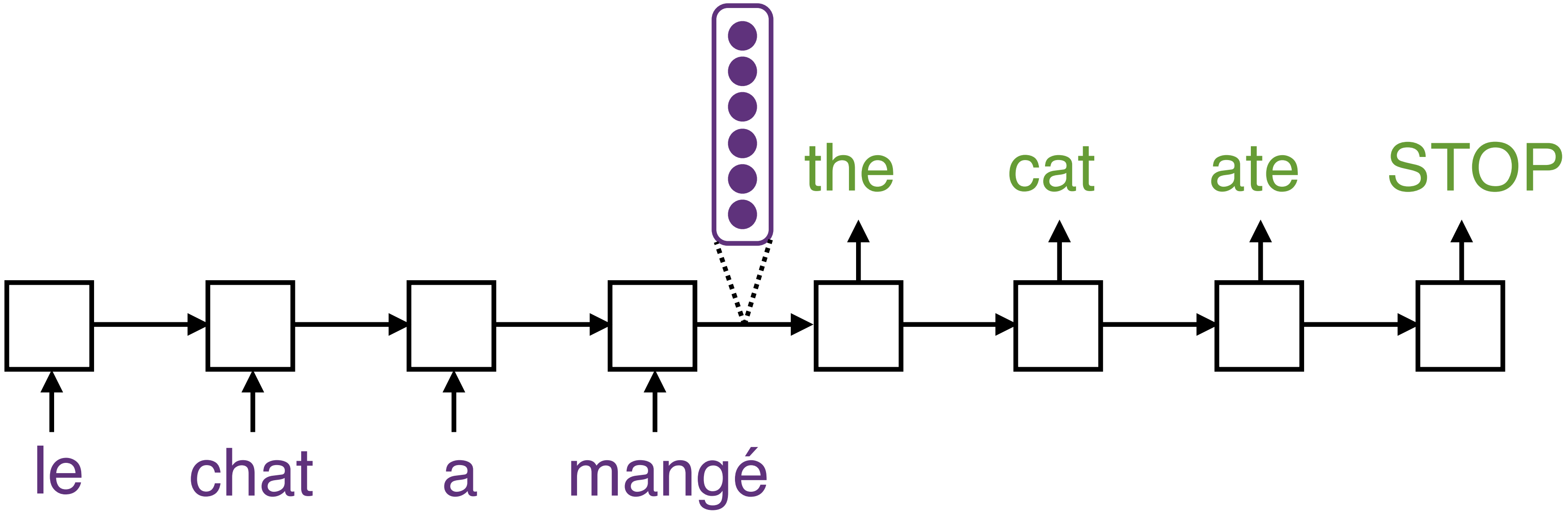



Parsing a sentence:

- Feedforward pass on nets
- Discrete feature computation
- Run CKY dynamic program

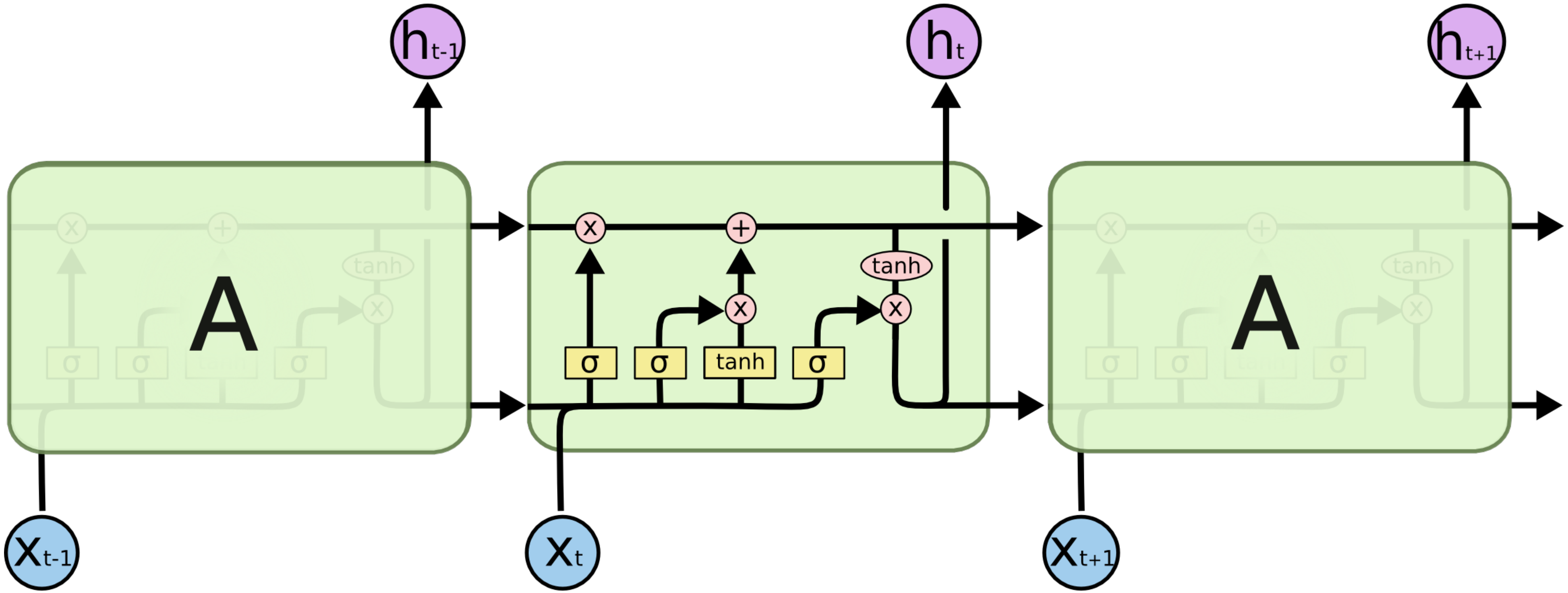
Durrett and Klein (ACL 2015)

# Machine Translation




- Long short-term memory units

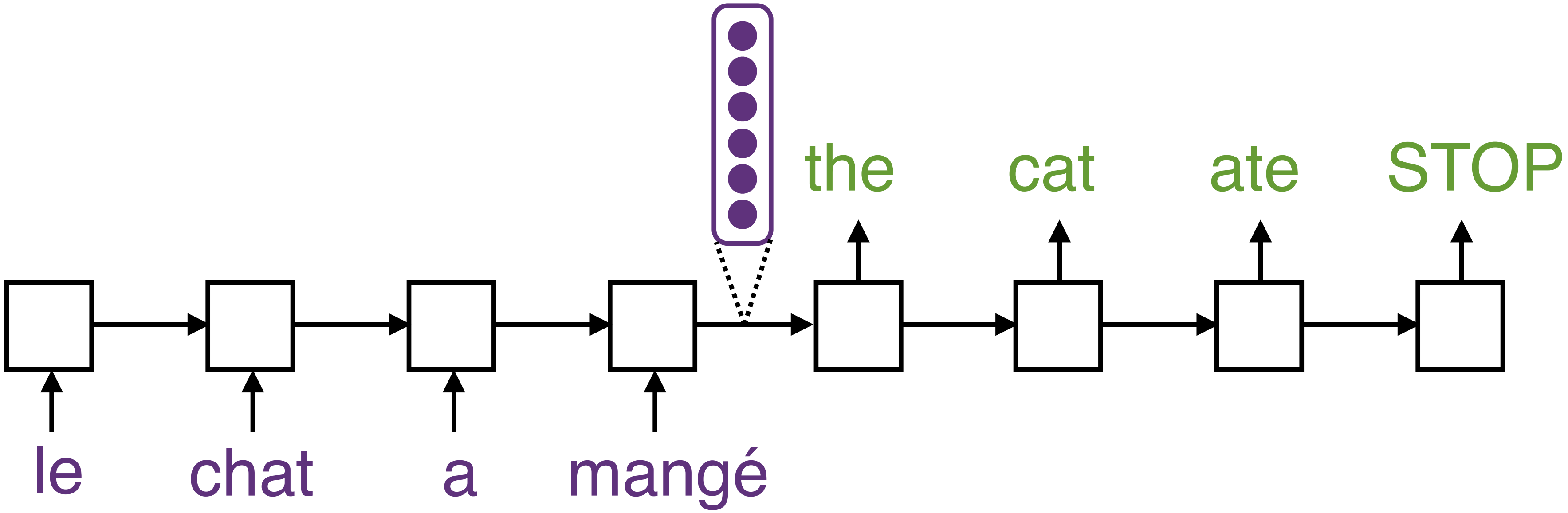
# Long Short-Term Memory Networks




- Map sequence of inputs to sequence of outputs

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

# Machine Translation

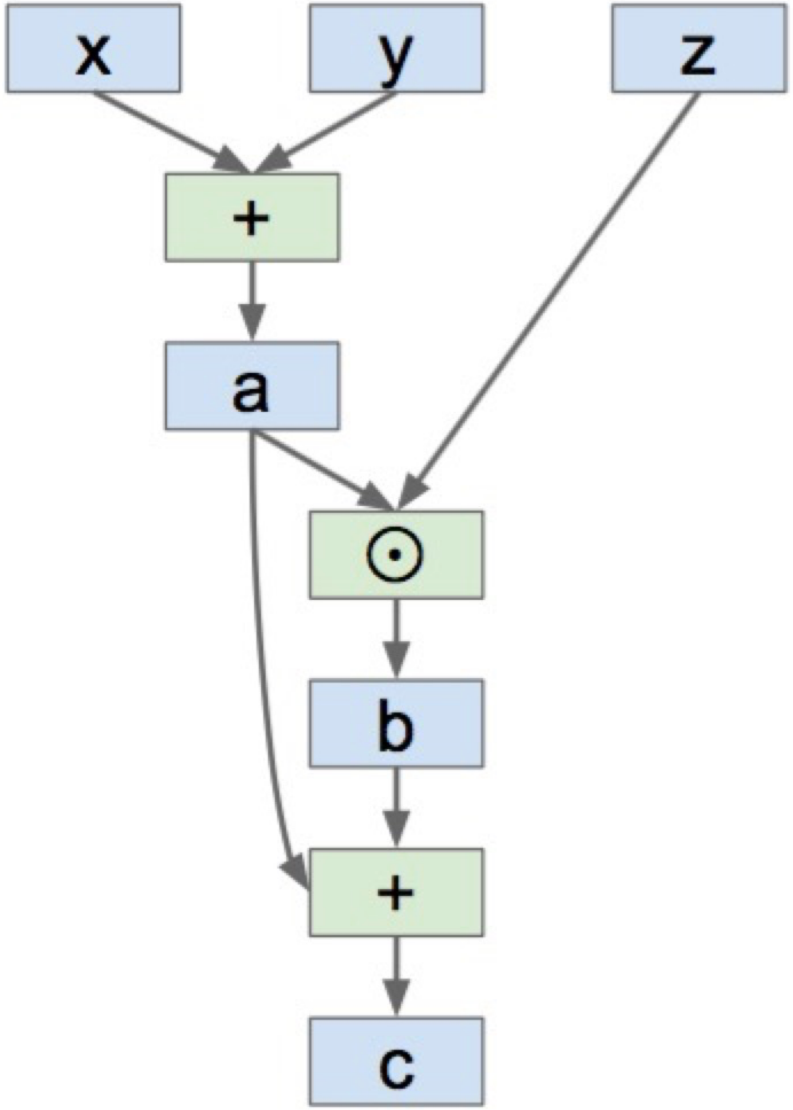



- Google is moving towards this architecture, performance is constantly improving compared to phrase-based methods

# Neural Network

- Tensorflow: https://www.tensorflow.org/
  - By Google, actively maintained, bindings for many languages
- Theano: http://deeplearning.net/software/theano/
  - University of Montreal, less and less maintained
- Torch: http://torch.ch/
  - Facebook AI Research, Lua

# Neural Network




```python
import theano
import theano.tensor as T

# Define symbolic variables
x = T.matrix('x')
y = T.matrix('y')
z = T.matrix('z')

# Compute some other values symbolically
a = x + y
b = a * z
c = a + b

# Compile a function that computes c
f = theano.function(
      inputs=[x, y, z],
      outputs=c
    )

# Evaluate the compiled function
# on some real values
xx = np.random.randn(4, 5)
yy = np.random.randn(4, 5)
zz = np.random.randn(4, 5)
print f(xx, yy, zz)

# Repeat the same computation
# explicitly using numpy ops
aa = xx + yy
bb = aa * zz
cc = aa + bb
```

Compile a function that produces c from x, y, z (generates code)

http://tmmse.xyz/content/images/2016/02/theano-computation-graph.png

# Word Vector Tools

- Word2Vec: https://radimrehurek.com/gensim/models/word2vec.html

  https://code.google.com/archive/p/word2vec/

  - Python code, actively maintained

- GLoVe: http://nlp.stanford.edu/projects/glove/
  - Word vectors trained on very large corpora

# Convolutional Networks

- CNNs for sentence class.: https://github.com/yoonkim/CNN_sentence
  - Based on tutorial from: http://deeplearning.net/tutorial/lenet.html
  - Python code
  - Trains very quickly

# Takeaways

- Neural networks have several advantages for NLP:
  - We can use *simpler nonlinear functions* instead of more complex linear functions
  - We can take advantage of word similarity
  - We can build models that are both position-dependent (feedforward neural networks) and position-independent (convolutional networks)
- NNs have natural applications to many problems
- While conventional linear models often still do well, neural nets are increasingly the state-of-the-art for many tasks