Sentiment Analysis

Human Language from a Computational Perspective
June 27, 2018

Outline

- Text classification
- Machine learning algorithms

Text classification

Many problems involve **classification**:

- Topic [NEWS, SPORTS, ...]
- Author/gender [MALE, FEMALE]
- Spam filtering [SPAM, NOT-SPAM]

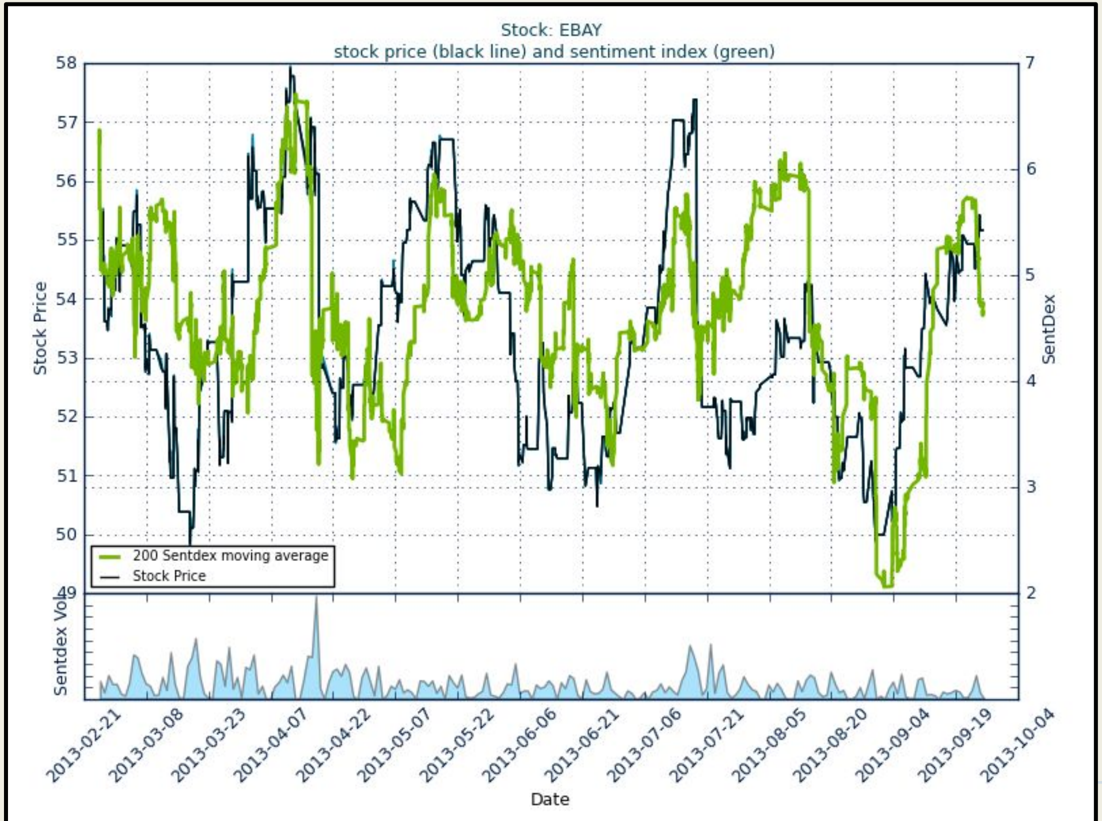
Sentiment classification

Given an input sentence, return its **sentiment label**:

+1 (POSITIVE) 0 (NEUTRAL) -1 (NEGATIVE)

(or a number on a finer scale, e.g. 1-5)

News and stocks



sentdex.com

Twitter

V_FSD: RT @FreeMemesKids: Love my new **fidget spinner** https://t.co/IcA6Ui6fsV

MeqzEdits: @23Duckk Oh no the **fidget spinner**?

fffaraa: @ibrahimyussop Omg.. not the **fidget spinner**

boybrunch: RT @a1andar: The Born This Way album cover except Gaga is a **fidget spinner**

eve_bertie: I bought a **fidget spinner** and I think it's the best thing I've ever done

Hamza0207: I broke this little girls **fidget spinner** and I legit feel so bad ??

sentiment140.com

Algorithm requirements

Input: sequence of tokens

[OH, NO, THE, FIDGET, SPINNER, ?]

Output: label (number)

-1

Bag of words

Simple approach: ignore the order, and just look for indicative tokens:

NOT	GREAT	DAMN	LOVE	HATE	…

Assign a weight to each token:

NOT	GREAT	DAMN	LOVE	HATE	…
0	+1	-1	+2	-2	…

Bag of words: classification

Calculate score for sentence:

THE ONLY REASON I LOVE MONDAYS … ZUMBA !!!!

0 + 0 + 0 + 0 + (+2) + (-1) + 0 + (+1) + (+1)

= +3

Since 3 > 0, predict:

+1 (POSITIVE)

Bag of words algorithm

```
classify(L, W):
s ← 0
i ← 1
while i ≤ len(L):
    s ← s + W[L[i]]
    i ← i + 1
return sign(s)
```

▷ L: input sentence,
W: table of weights for words

$$\text{sign}(s) = \begin{cases} +1 & \text{if } s > 0 \\ 0 & \text{if } s = 0 \\ -1 & \text{if } s < 0 \end{cases}$$

Learning a model

How to determine the word weights?

In language models and POS tagging,

we **learned** the statistics as Counts.

Here we can learn Weights.

Perceptron

The perceptron is a **learning** algorithm.

Input: list of samples (sentence + label)

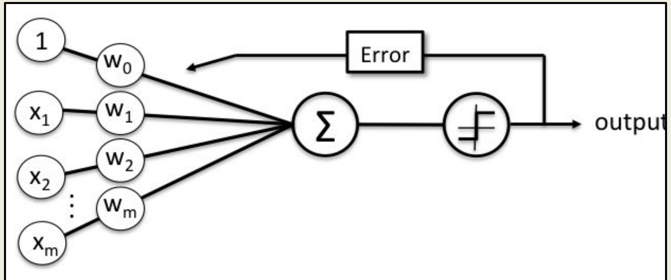
Labels given as numbers: `+1` `0` `-1`

Output: table of weights for each word

(which can then be used to classify new sentences.)

Perceptron

The algorithm goes over all samples repeatedly, until there are no errors. Whenever there is an error, it updates the weights of all tokens in the sample.



Perceptron algorithm

```
train(X, Y):                  ▷ X: list of input sentence (samples)
W ← [0 for all words]           Y: list of labels (one per sample)
while W is changing:          ▷ training iterations
    i ← 1
    while i ≤ len(X):         ▷ go through all samples
        if classify(X[i], W) ≠ Y[i]:     ▷ if the model is wrong,
            update(X[i], Y[i], W)          update its weights
        i ← i + 1
return W                      ▷ return final learned weights
```

Perceptron algorithm

```
update(L, y, W):           ▷ L: input sentence, y: label,
i ← 1                         W: table of weights for words
while i ≤ len(L):          ▷ go through sentence tokens
    W[L[i]] ← W[L[i]] + y  ▷ update weight by
    i ← i + 1                 adding y to it
```

Example

$X = [$THIS IS GREAT, THIS IS AWFUL$]$

$Y = [1, -1]$

Example

First iteration, first sample

X[1] = THIS IS GREAT

$0 + 0 + 0 = 0$

W =

THIS	0
IS	0
GREAT	0
AWFUL	0

Result: 0 (NEUTRAL)

Y[1] = 1 $\neq 0$ ✗

Example

First iteration, first sample

X[1] = THIS IS GREAT

0 + 0 + 0 = 0

W =

THIS	**1**
IS	**1**
GREAT	**1**
AWFUL	0

Result: 0 (NEUTRAL)

Y[1] = 1 $\neq$ 0 ✗

Updating weights

Example

First iteration, second sample

X[2] = THIS IS AWFUL

$1 + 1 + 0 = 0$

Result: 1 (POSITIVE)

Y[2] = -1 $\neq$ 1 ✗

W =

THIS	1
IS	1
GREAT	1
AWFUL	0

Example

First iteration, second sample

X[2] = THIS IS AWFUL

1 + 1 + 0 = 0

W =

THIS	0
IS	0
GREAT	1
AWFUL	-1

Result: 1 (POSITIVE)

Y[2] = -1 ≠ 1 ✗

Updating weights

Example

Second iteration, first sample

X[1] = THIS IS GREAT

0 + 0 + 1 = 1

W =

THIS	0
IS	0
GREAT	1
AWFUL	-1

Result: 1 (POSITIVE)

Y[1] = 1 ✓

Example

Second iteration, second sample

X[2] = THIS IS AWFUL

$0 + 0 + -1 = -1$

W =

THIS	0
IS	0
GREAT	1
AWFUL	-1

Result: -1 (NEGATIVE)

Y[2] = -1 ✓

Example

Finished iteration without updating weights, so return W.

W can now be used to classify new sentences.

W =

THIS	0
IS	0
GREAT	1
AWFUL	-1

Problem: order matters

THAT IS NOT TRUE , IT IS GREAT

has the same tokens as

THAT IS TRUE , IT IS NOT GREAT

Will be classified the same

Generalizing

Instead of single words, use features.

Examples for features:

- first token is GREAT?
- contains NOT followed by BAD?
- contains TASTE with part of speech NN?

Generalizing

In Bag of Words, all features are:

- contains <w>?

where <w> is some word.

This is also called **unigram features**.

Generalizing

We can also use **bigram features**:

- contains `<w1>` followed by `<w2>`?

where `<w1>` and `<w2>` are words. Or:

- contains `<w>` with part-of-speech `<p>`?

etc.

Linear classifier

```
classify(L, F, W):         ▷ L: input sentence, F: features to check,
                             W: table of weights for features
s ← 0
i ← 1
while i ≤ len(F):          ▷ go through possible features
    f ← F[i]
    if f(L):               ▷ the feature f is true for L
        s ← s + W[f]       ▷ add the feature's weight to the score
    i ← i + 1
return sign(s)             ▷ 1 if s > 0,   0 if s = 0,   1 if s > 0
```

Linear classifier

The same as the Bag of Words classifier, but using general features and not just words.

Might be even millions of features!

Example

MeqzEdits: @23Duckk Oh no the **fidget spinner**?

The unigram features OH and NO may not be clearly negative (weights will stay ~0), but the bigram feature OH NO is negative.

sentiment140.com

Multi-class classifier

So far we discussed **binary** classifiers,

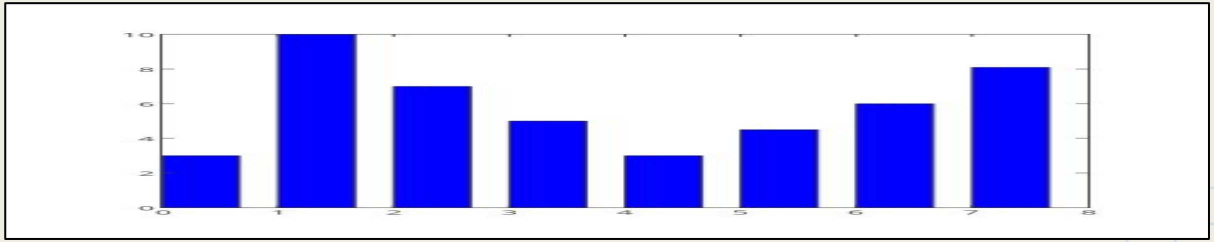
but there can be more than two labels.

Example:

2 (VERY POSITIVE), 1 (POSITIVE), 0, -1 (NEGATIVE), -2 (VERY NEGATIVE)

Multi-class classifier

Instead of a single score, the classifier will have a score for each possible label, and then choose the label with the top score.

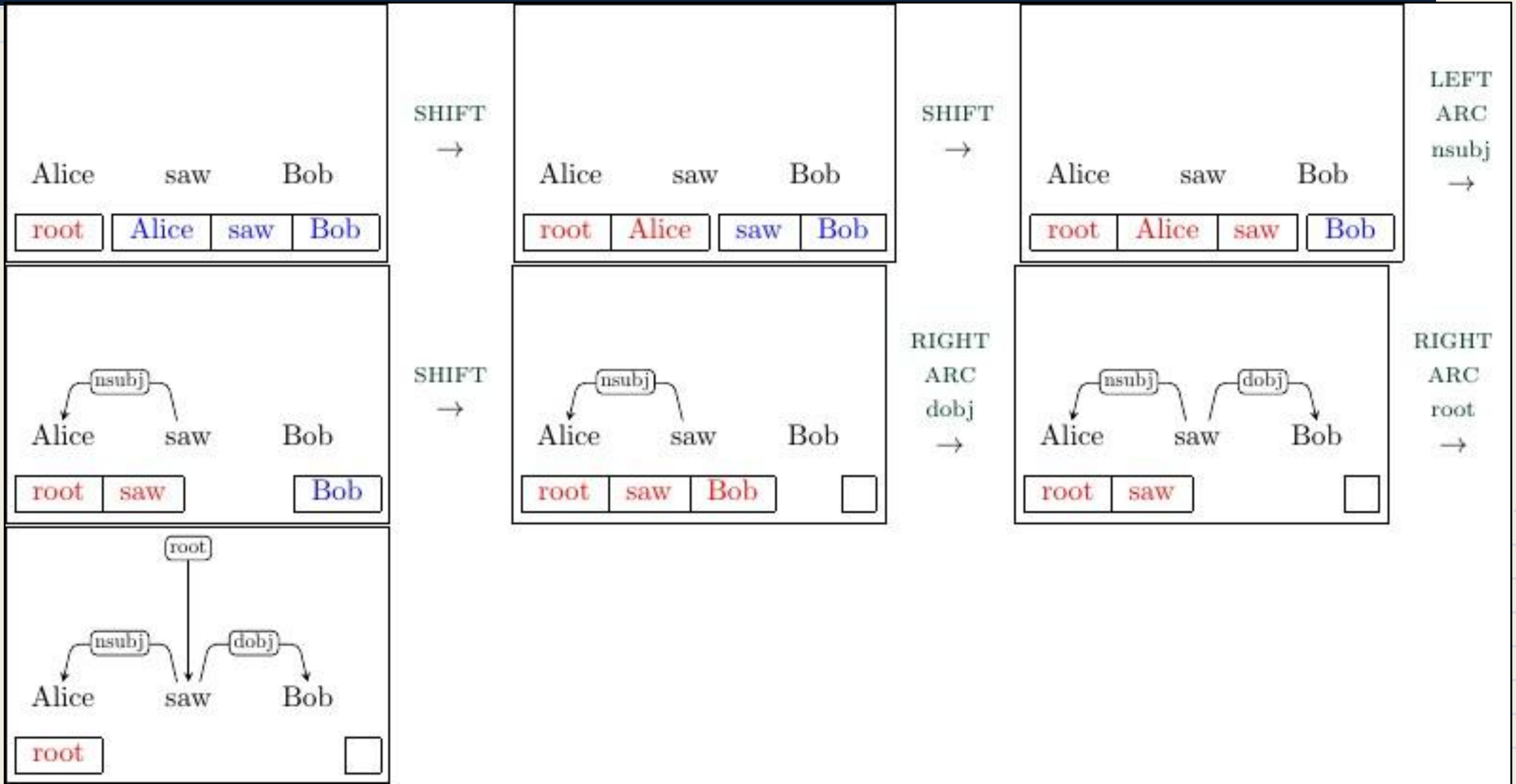


Learning: different weights for each label.

Linear multi-class classifier

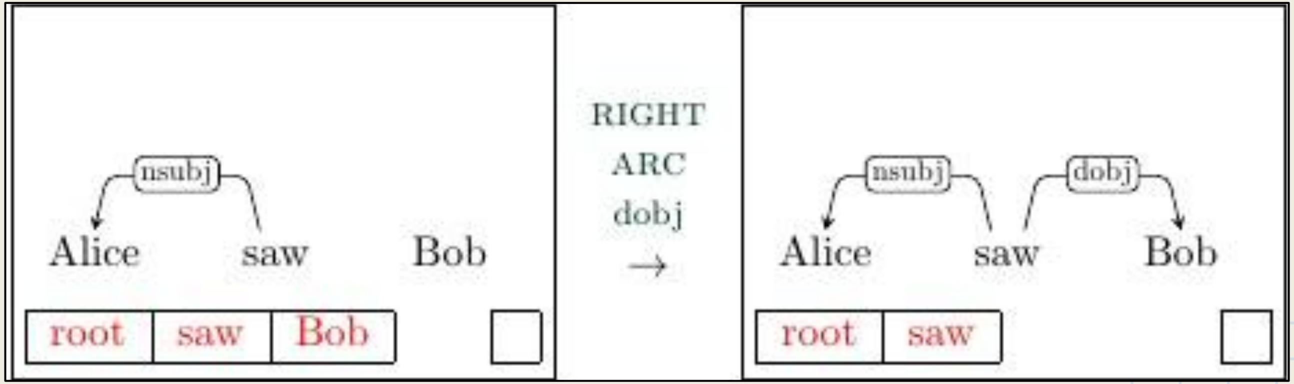
```
classify(L, F, W):          ▷ L: input sentence, F: features to check
S ← [0 for all labels]        W: table of weights for features for each
i ← 1                           label
while i ≤ len(F):           ▷ go through possible features
    f ← F[i]
    if f(L):                ▷ the feature f is true for L
        S ← S + W[f]        ▷ add the feature's weight to the score
    i ← i + 1                  for each label
return argmax(S)
```

Back to parsing



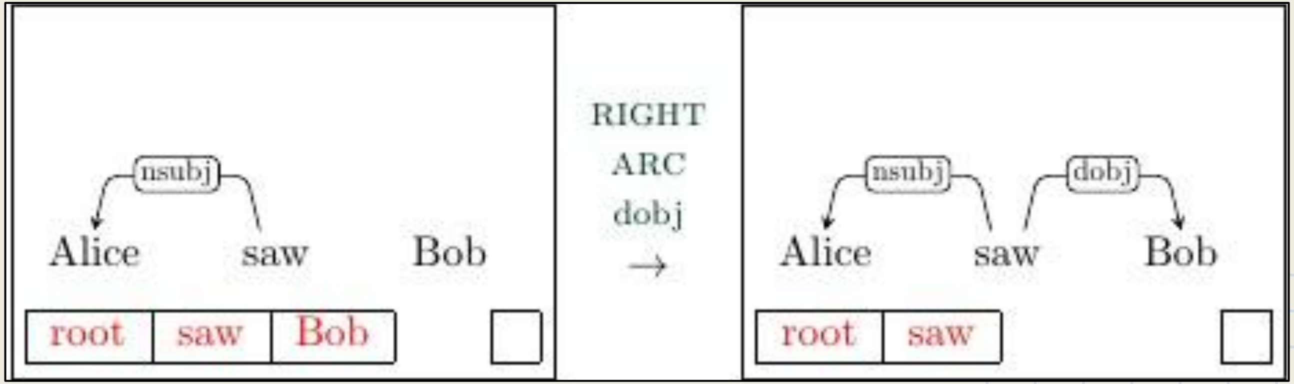
Perceptron for parsing

If we see the state on the left here, we need to know to apply **RIGHT-ARC$_{\text{dobj}}$**



Features for parsing

- rightmost token on the stack is <w>?
- buffer leftmost part-of-speech is <p>?
- second stack token is a parent?
- etc.



Perceptron for parsing

Perceptron + word/part-of-speech unigram, bigram features: pretty good parser.

Parser	UAS (%)	LAS (%)
MaltParser	90.93	88.95
Parsey McParseface	94.41	92.55

Twitter sentiment analysis

Training any model requires labeled data: learning from examples.

Problem: labeling

Manually annotating tweets takes time and money.

Avoiding manual work

Automatically building a training set using the **emoticons** in the tweets:

POSITIVE			NEGATIVE		
:)	:D	:)	:(	=(	: (
:-)	=)	;)	:-(	:/	=/

Alec Go, Richa Bhayani, and Lei Huang. 2009. Twitter sentiment classification using distant supervision. Technical report, Stanford.

Problem: sarcasm

People sometimes say the opposite of what they mean.

Sarcasm recognition

Sarcasm: “saying the opposite of what you mean in a way intended to make someone else feel stupid or show you are angry”.

listening to Andrew Ridgley by Black Box Recorder on @Grooveshark: http://tinysong.com/cO6i #goodmusic

I guess you should expect a WONDERFUL video tomorrow. #sarcasm

Accuracy: 94.7%

1. Oren Tsur, Dmitry Davidov, Ari Rappoport. ICWSM - A Great Catchy Name: Semi-Supervised Recognition of Sarcastic Sentences in Product Reviews. *Fourth International AAAI Conference on Weblogs and Social Media (ICWSM) 2010*.
2. Oren Tsur, Dmitry Davidov, Ari Rappoport. Semi-Supervised Recognition of Sarcastic Sentences in Twitter and Amazon. *Computational Natural Language Learning (CoNLL) 2010*.

Multi-class labels

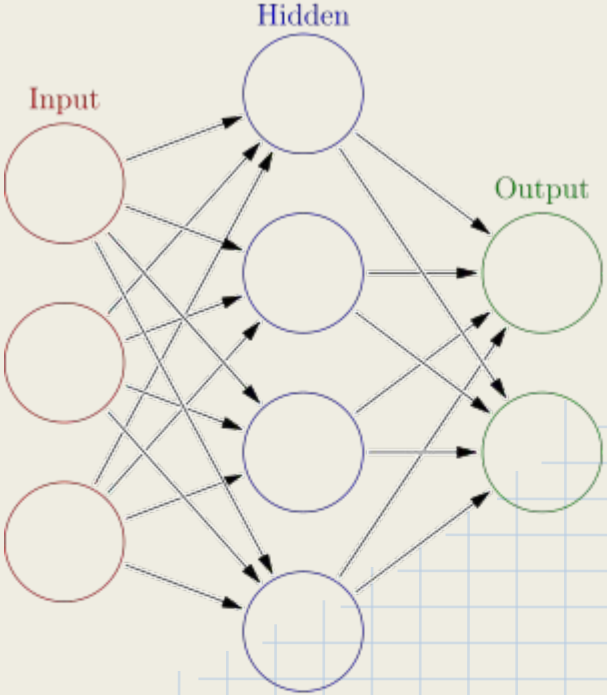
Hashtag- and smiley-based labels: 51 or 16 labels instead of just two (positive/negative)

Hashtags	Smileys
#sad	;)
#crazy	:(
#bored	X(
#fun	:S

Dmitry Davidov, Oren Tsur, Ari Rappoport. [Enhanced Sentiment Learning Using Twitter Hashtags and Smileys](). *COLING 2010*.

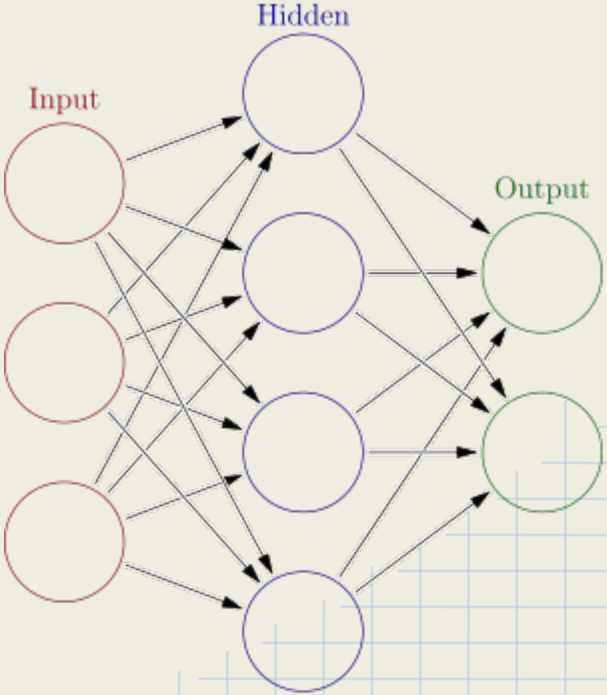
Artificial neural networks

Instead of using the score for classification, give it to another linear classifier ("multi-layer perceptron")



Artificial neural networks

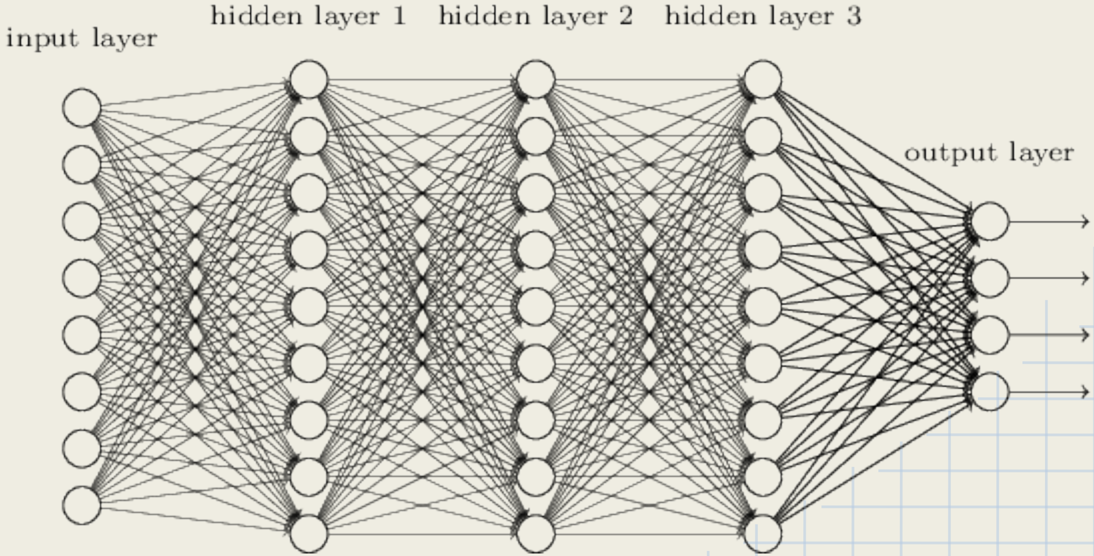
Only the output of the last layer is actually used for classification.



Deep neural networks

Many layers sometimes work better.

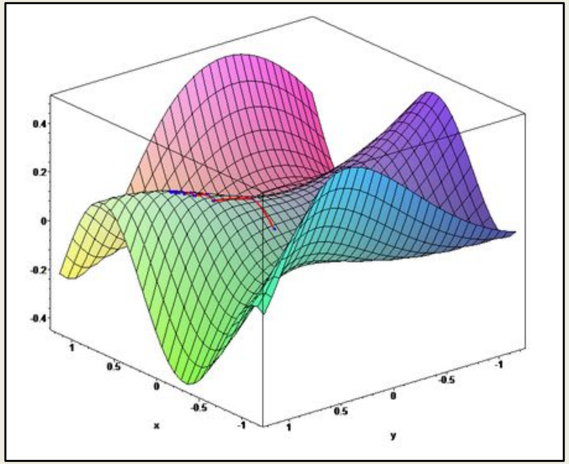
(“deep learning”)

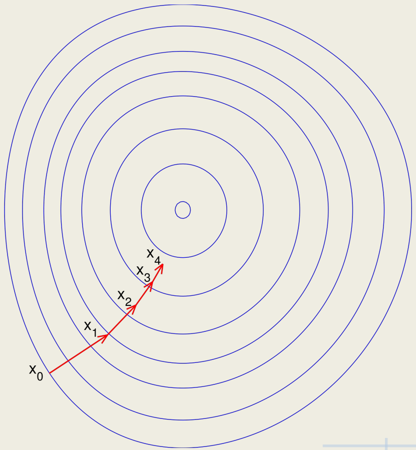


Training neural networks

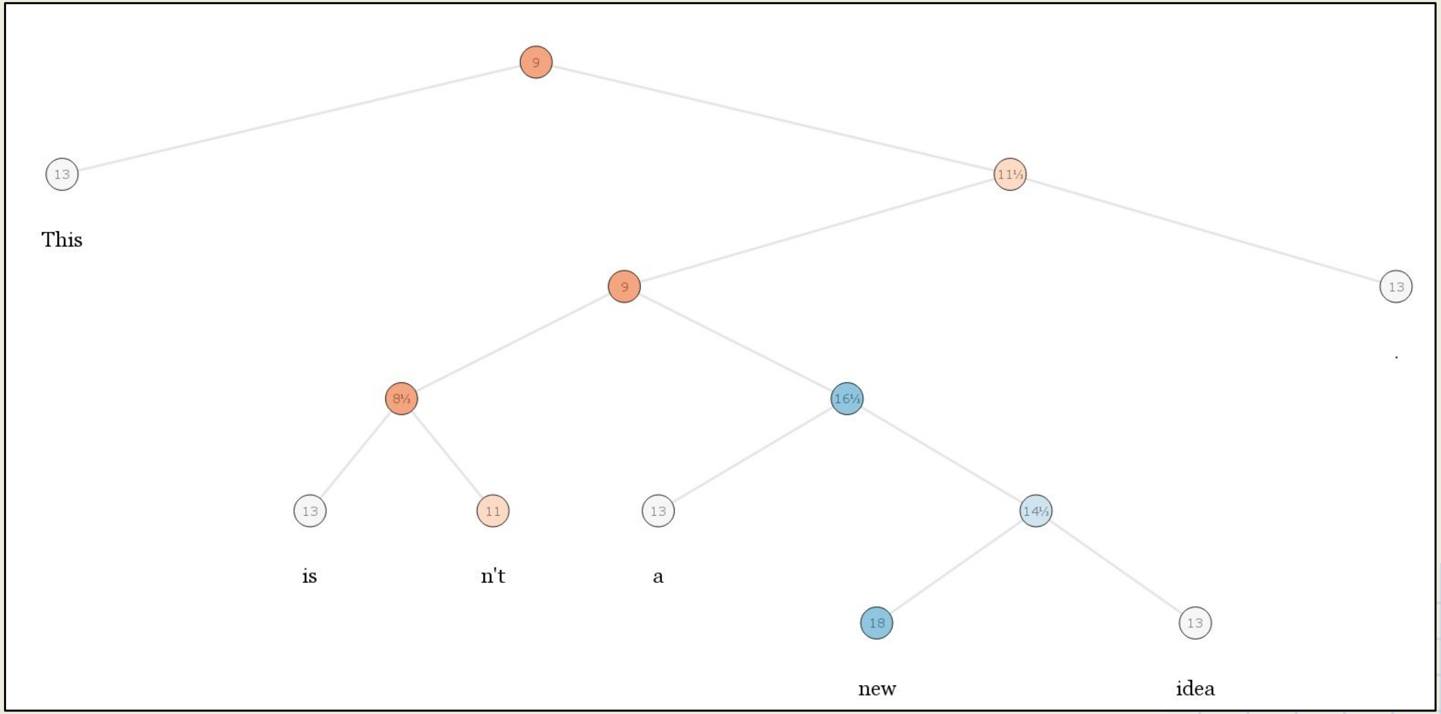
Perceptron only works for one layer.

Instead, use gradient descent algorithms.





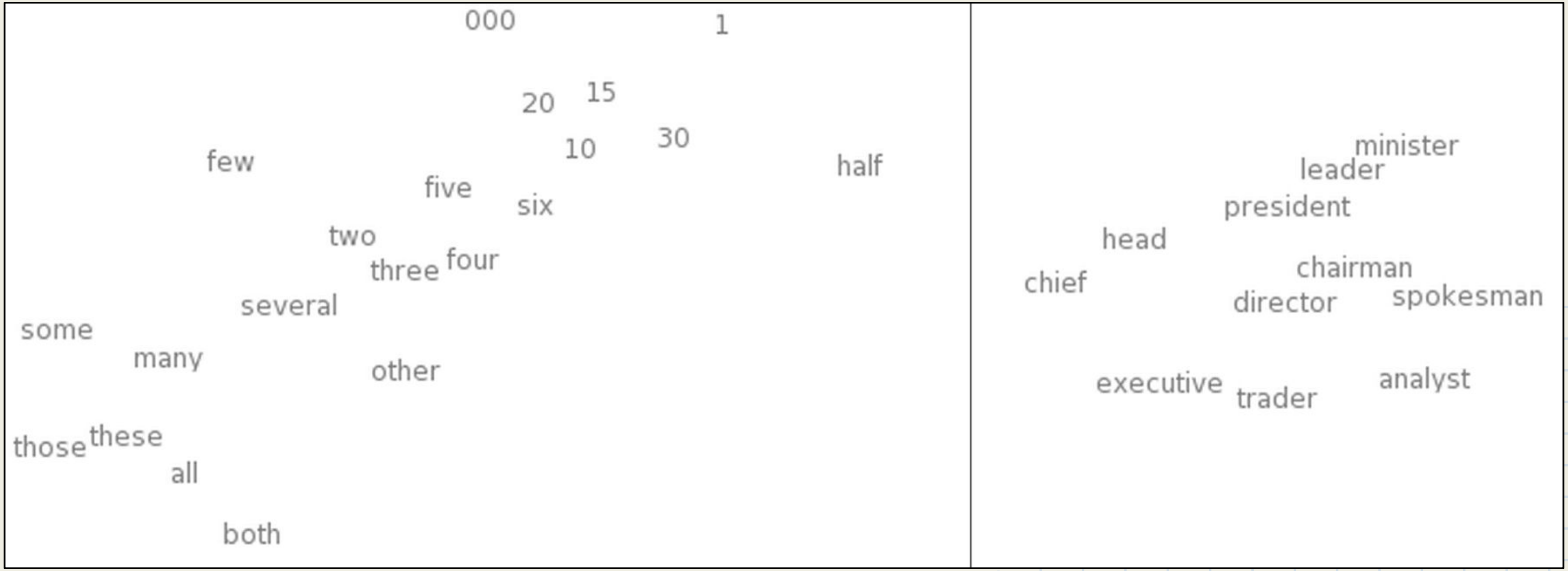
Sentiment treebank



nlp.stanford.edu/sentiment/treebank.html

Word embedding

Use vectors to represent words.



NNs for parsing

NN + word embedding features:

best parsers today.

Parser	UAS (%)	LAS (%)
MaltParser	90.93	88.95
Parsey McParseface	94.41	92.55

Links

- http://www.sciencefriction.net/blog/2010/12/06/1120/
- https://www.aclweb.org/anthology/E/E17/E17-2017.pdf
- https://research.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html
- http://www.cs.huji.ac.il/~danielh/P17-1104.pdf
- https://github.com/ayushoriginal/Sentiment-Analysis-Twitter