Lecture 02: Word2vec technical details, Convolutional Neural Networks, CNN for texts

Radoslav Neychev
Anastasia Ianina

Harbour.Space University
15.06.2021, Barcelona, Spain

Outline

- Convolutional Neural Networks
  - Intro/recap
  - Main definitions
- CNNs for text processing

Word2vec

One-hot vectors

- **One-hot vectors:**

```
                 Paris
          Rome                                  word V
Rome   = [1, 0, 0, 0, 0, 0, …, 0]
Paris  = [0, 1, 0, 0, 0, 0, …, 0]
Italy  = [0, 0, 1, 0, 0, 0, …, 0]
France = [0, 0, 0, 1, 0, 0, …, 0]
```

Problems:

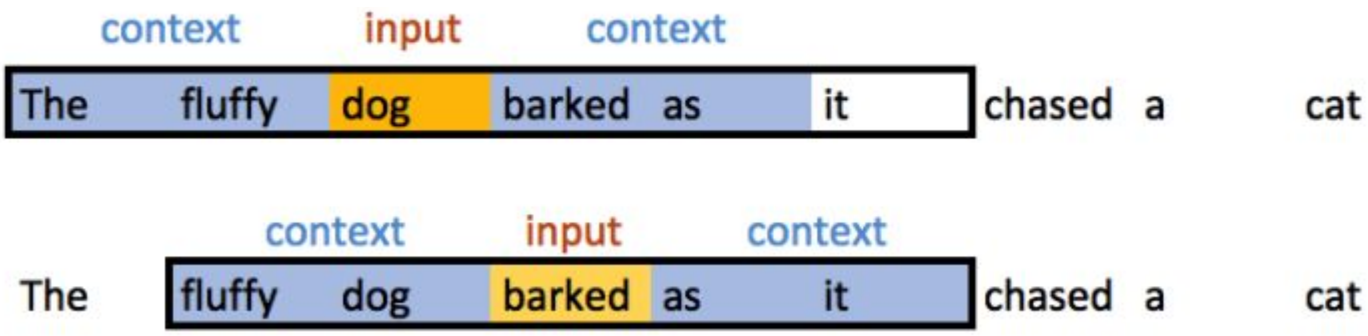
- Huge vectors
- VERY sparse
- No semantics or word similarity information included

Distributional semantics

Does vector similarity imply semantic similarity?

"You shall know a word by the company it keeps"

Firth, 1957

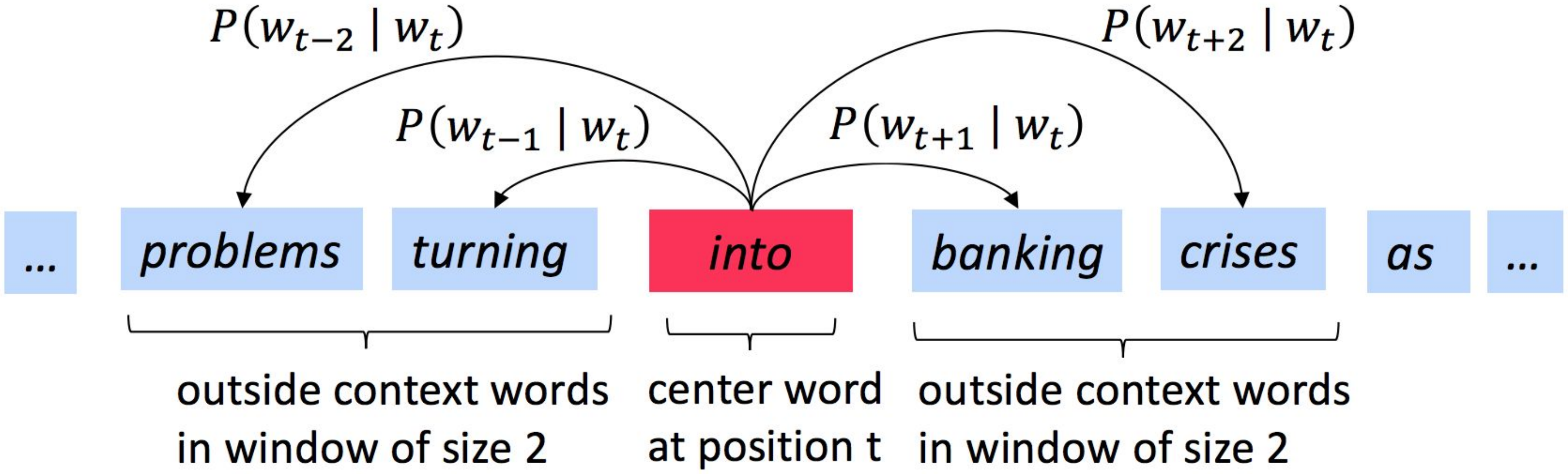


Embeddings: word2vec

Source Text		Training Samples
The quick brown fox jumps over the lazy dog.	→	(the, quick) (the, brown)
The **quick** brown fox jumps over the lazy dog.	→	(quick, the) (quick, brown) (quick, fox)
The quick **brown** fox jumps over the lazy dog.	→	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown **fox** jumps over the lazy dog.	→	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

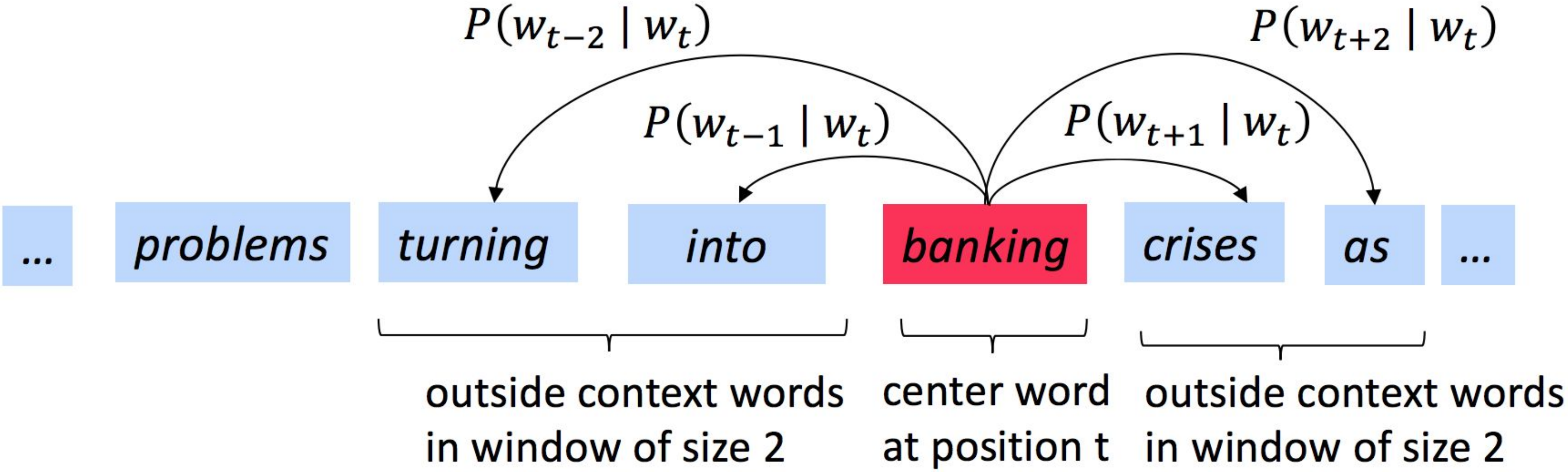
Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Embeddings: word2vec



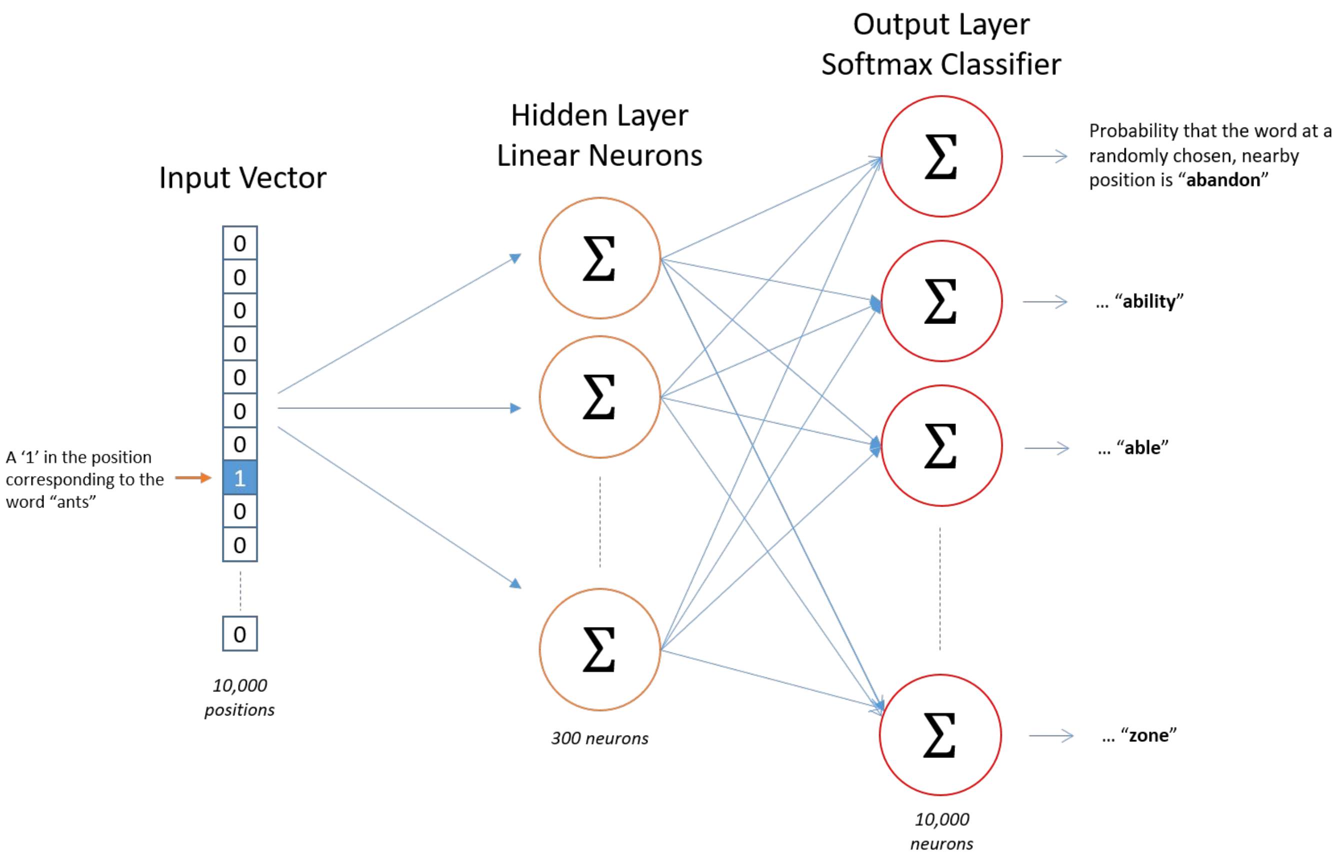
Source: CS224n: http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture01-wordvecs1.pdf

Embeddings: word2vec



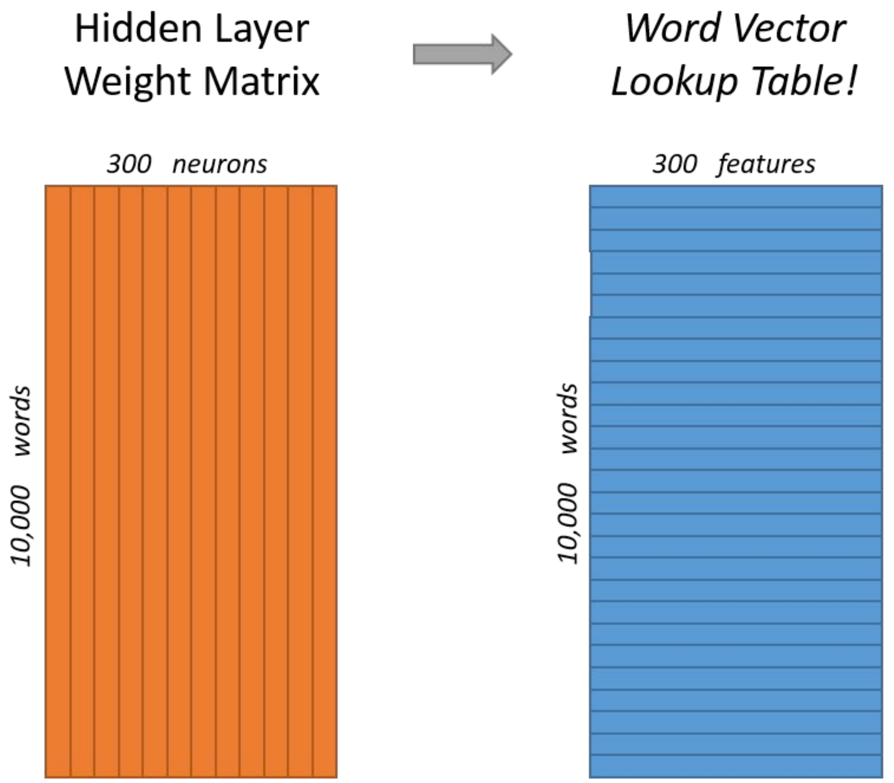
Source: CS224n: http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture01-wordvecs1.pdf

Embeddings: word2vec



Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Embeddings: word2vec



Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Embeddings: word2vec

- Word vectors with 300 components
- Vocabulary of 10,000 words.
- Weight matrix with 300 x 10,000 = 3 million weights each!

Training is too long and computationally expensive

How to fix this?

Embeddings: word2vec

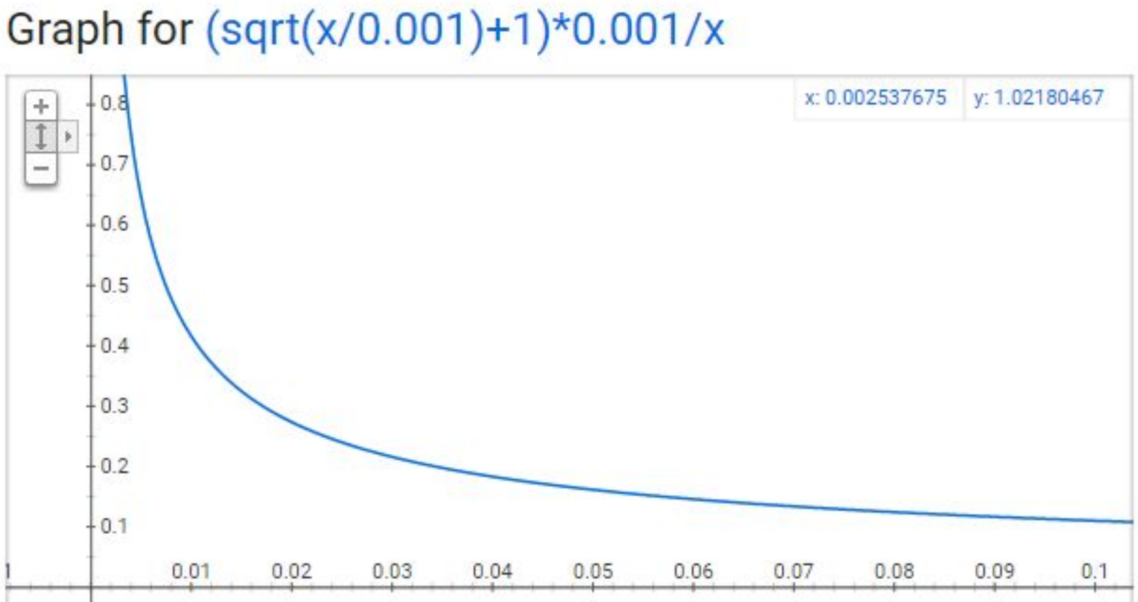
Basic approaches:

1. Treating common word pairs or phrases as single “words” in their model.
2. Subsampling frequent words to decrease the number of training examples.
3. Modifying the optimization objective with a technique they called “Negative Sampling”, which causes each training sample to update only a small percentage of the model’s weights.

Embeddings: word2vec

Subsampling frequent words.

w_i is the word, $z(w_i)$ is the fraction of this word in the whole text



$P(w_i)$ is the probability of *keeping* the word:

$$P(w_i) = \left(\sqrt{\frac{z(w_i)}{0.001}} + 1\right) \cdot \frac{0.001}{z(w_i)}$$

Source: http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/

Embeddings: negative sampling

Negative Sampling idea: only few words error is computed. All other words have zero error, so no updates by the backprop mechanism.

More frequent words are selected to be negative samples more often.The probability for a selecting a word is just it’s weight divided by the sum of weights for all words.

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n}\left(f(w_j)^{3/4}\right)}$$

Word2vec: two models

Continuous BOW (CBOW)

$$p(w_i | w_{i-h}, \ldots, w_{i+h})$$

Predict center word from (bag of) context words

- Predicting one word each time
- Relatively fast

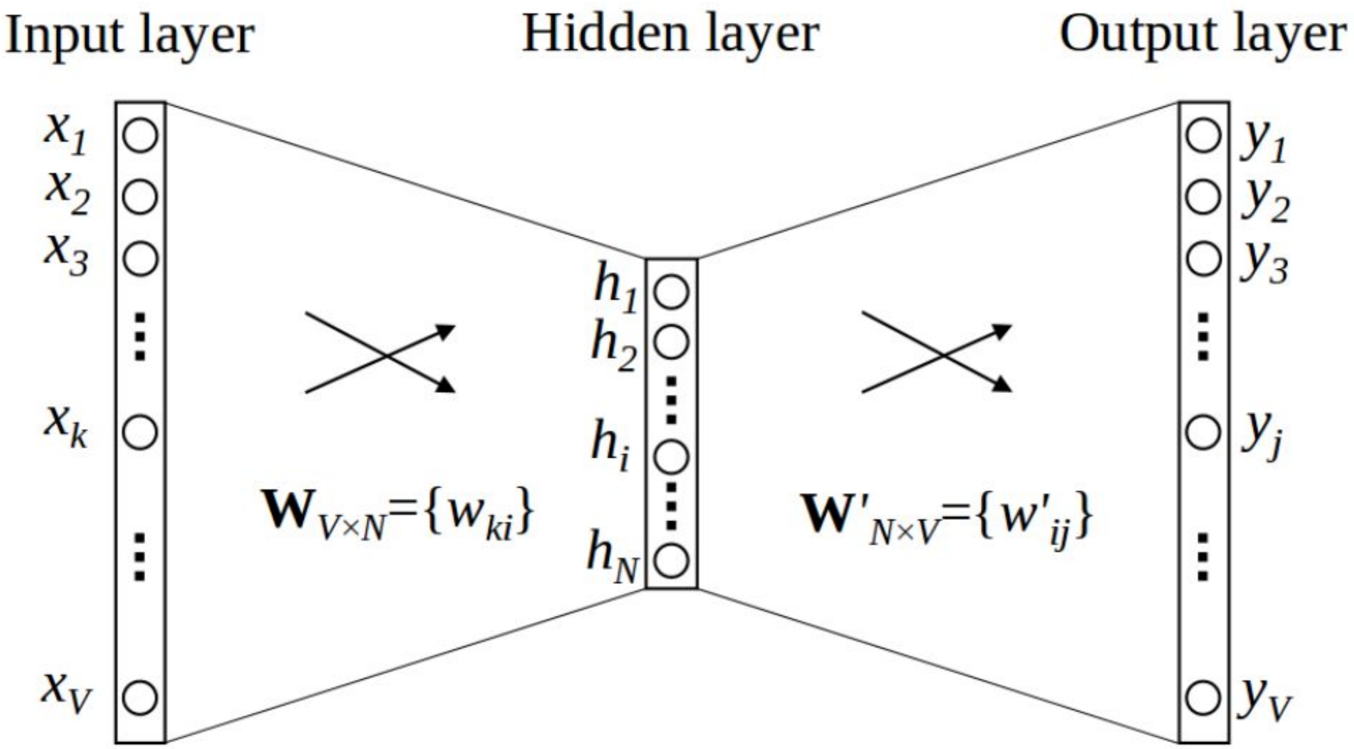
Skip-gram

$$p(w_{i-h}, \ldots w_{i+h} | w_i)$$

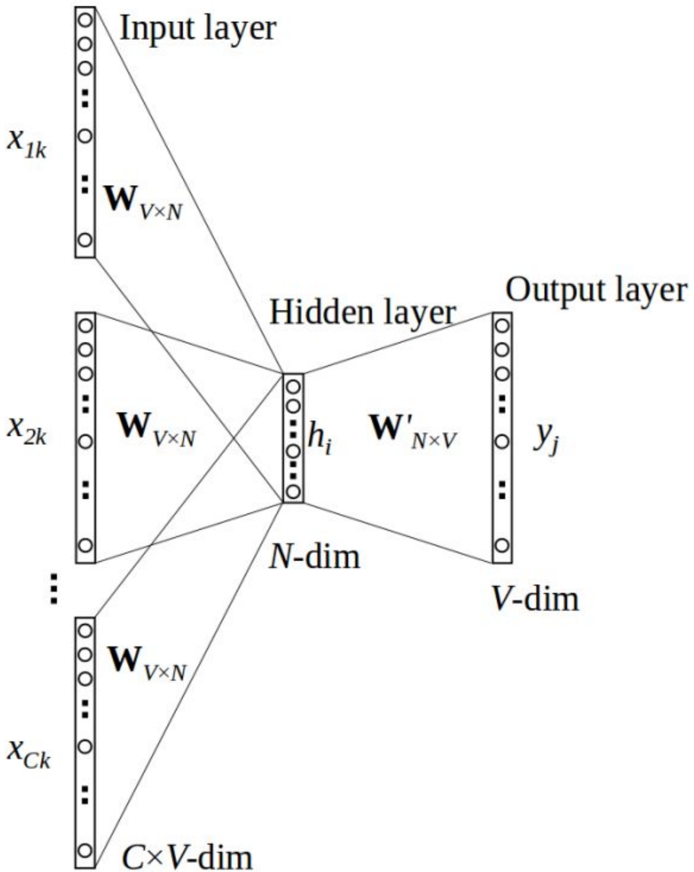
Predict context (”outside”) words (position independent) given center word

- Predicting context by one word
- Much slower
- Better with infrequent words

CBOW

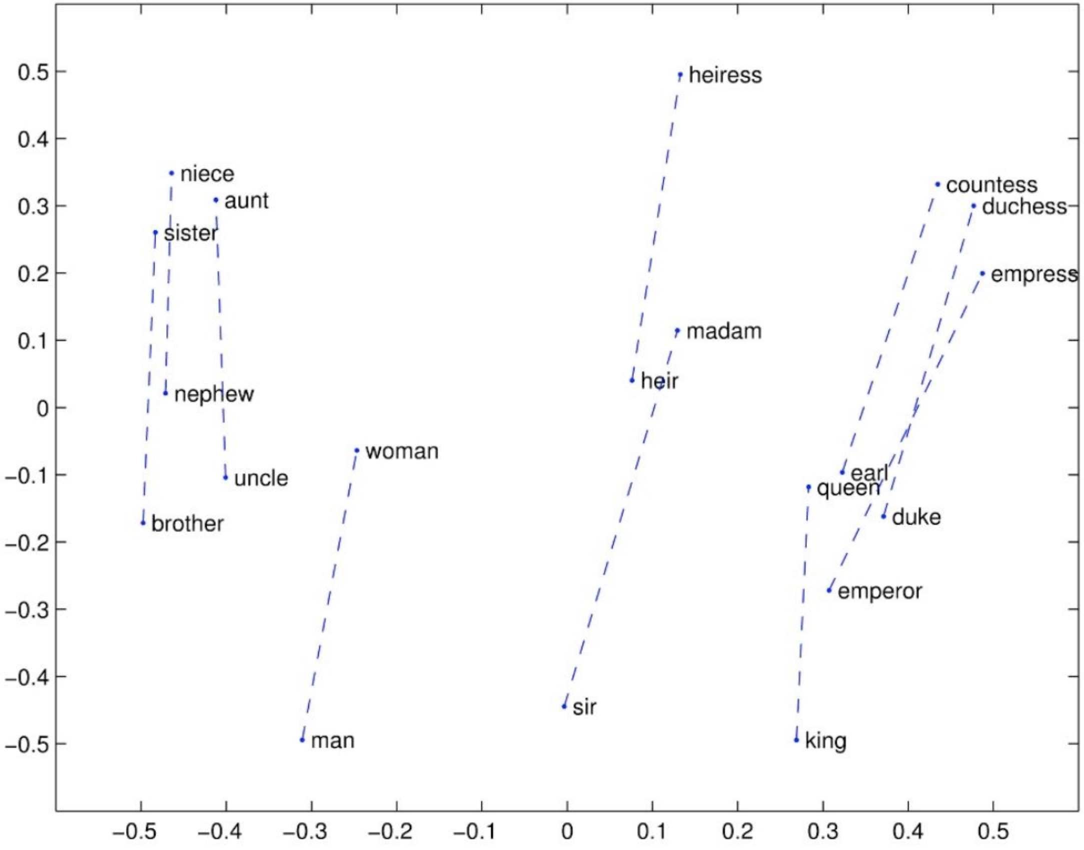

Input layer
Hidden layer
Output layer
x_1
x_2
x_3
x_k
x_V
h_1
h_2
h_i
h_N
y_1
y_2
y_3
y_j
y_V
$\mathbf{W}_{V\times N}=\{w_{ki}\}$
$\mathbf{W}'_{N\times V}=\{w'_{ij}\}$

Skip-gram

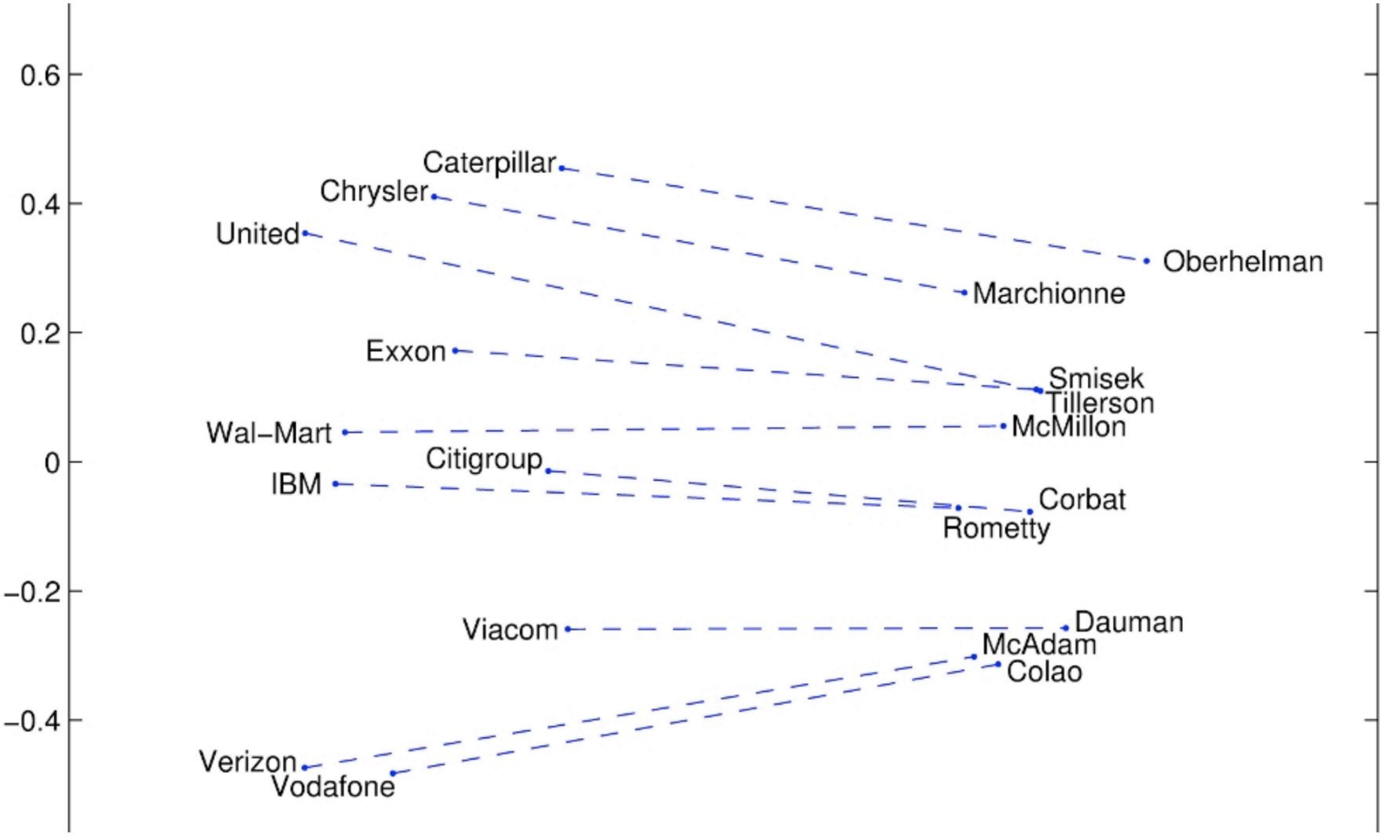


Input layer
x_{1k}
$\mathbf{W}_{V\times N}$
x_{2k}
$\mathbf{W}_{V\times N}$
x_{Ck}
$\mathbf{W}_{V\times N}$
$C\times V$-dim
Hidden layer
h_i
N-dim
$\mathbf{W}'_{N\times V}$
Output layer
y_j
V-dim

GloVe Visualizations



GloVe Visualizations: Company - CEO

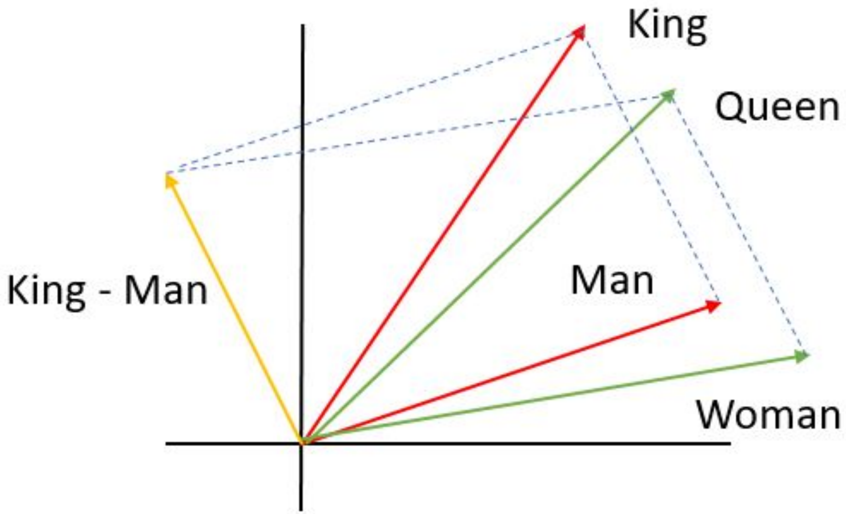


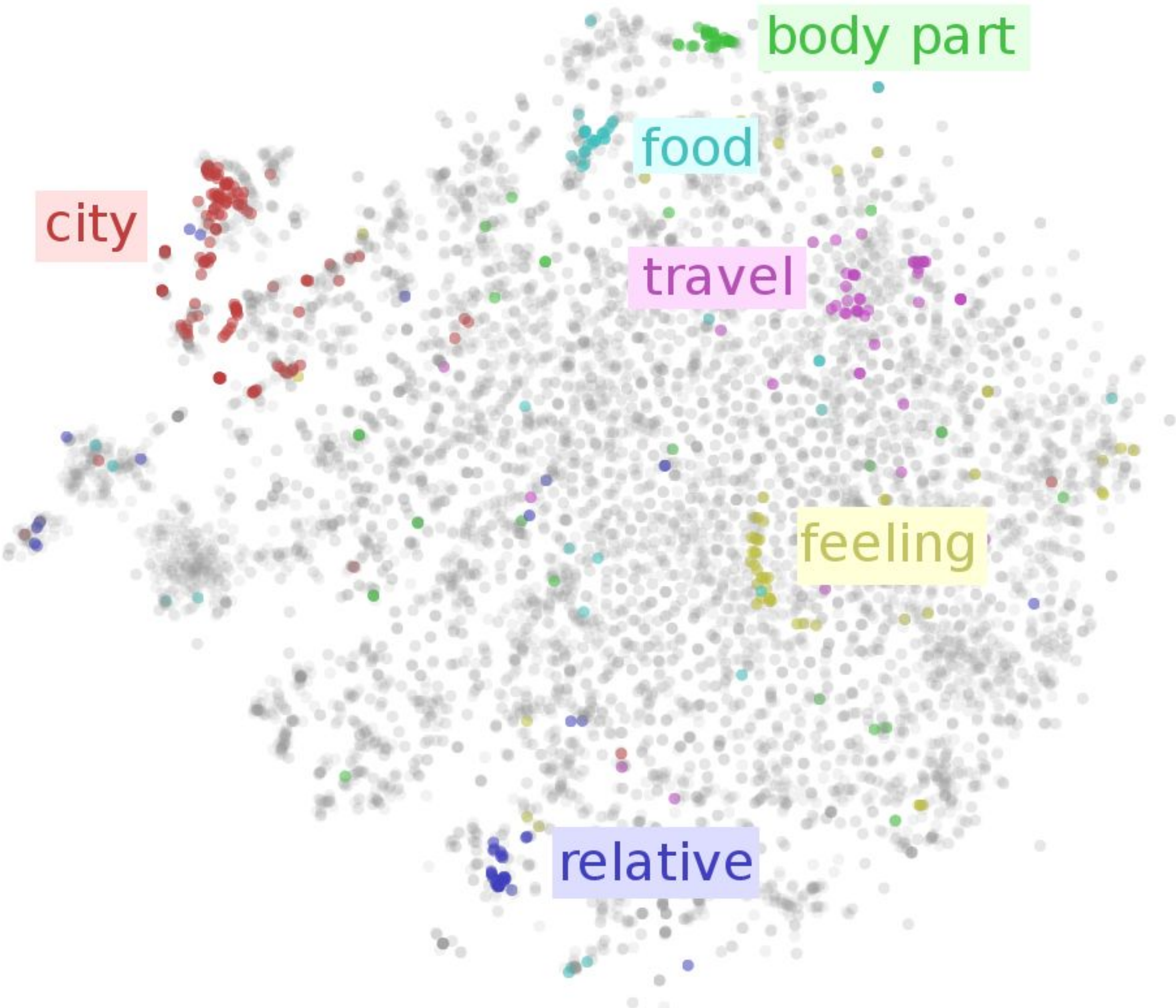
Word2vec: word analogies

King - man + woman = queen

$$\begin{array}{cccc} \text{King} & \text{man} & \text{woman} & \text{queen} \\ \downarrow & \downarrow & \downarrow & \downarrow \\ x & y & y' & target \end{array}$$

$$\cos(x - y + y', target) \to \max_{target}$$




body part
food
city
travel
feeling
relative

Conclusion

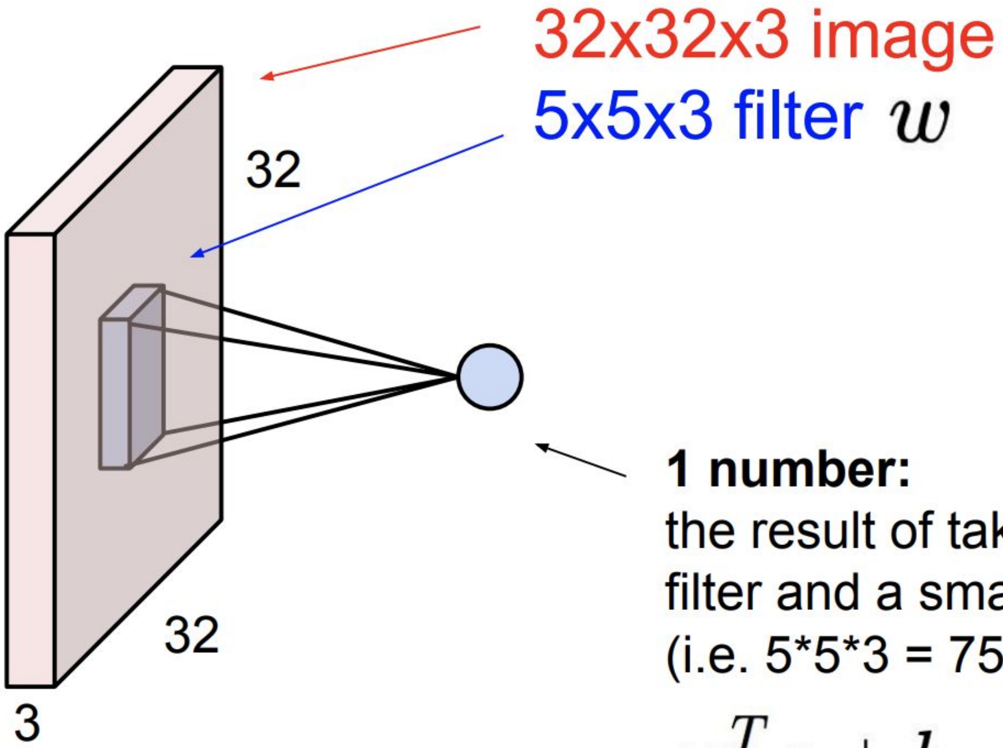
Word vectors are simply vectors of numbers that represent the meaning of a word

Approaches:

- One-hot encoding
- Bag-of-words models
- Counts of word / context co-occurrences
- TF-IDF
- Predictions of context given word (skip-gram neural network models, e.g. word2vec)

Convolutional Neural Networks: Intro or recap

Convolutional layer



1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

source: http://cs231n.stanford.edu/slides/2016/winter1516_lecture7.pdf

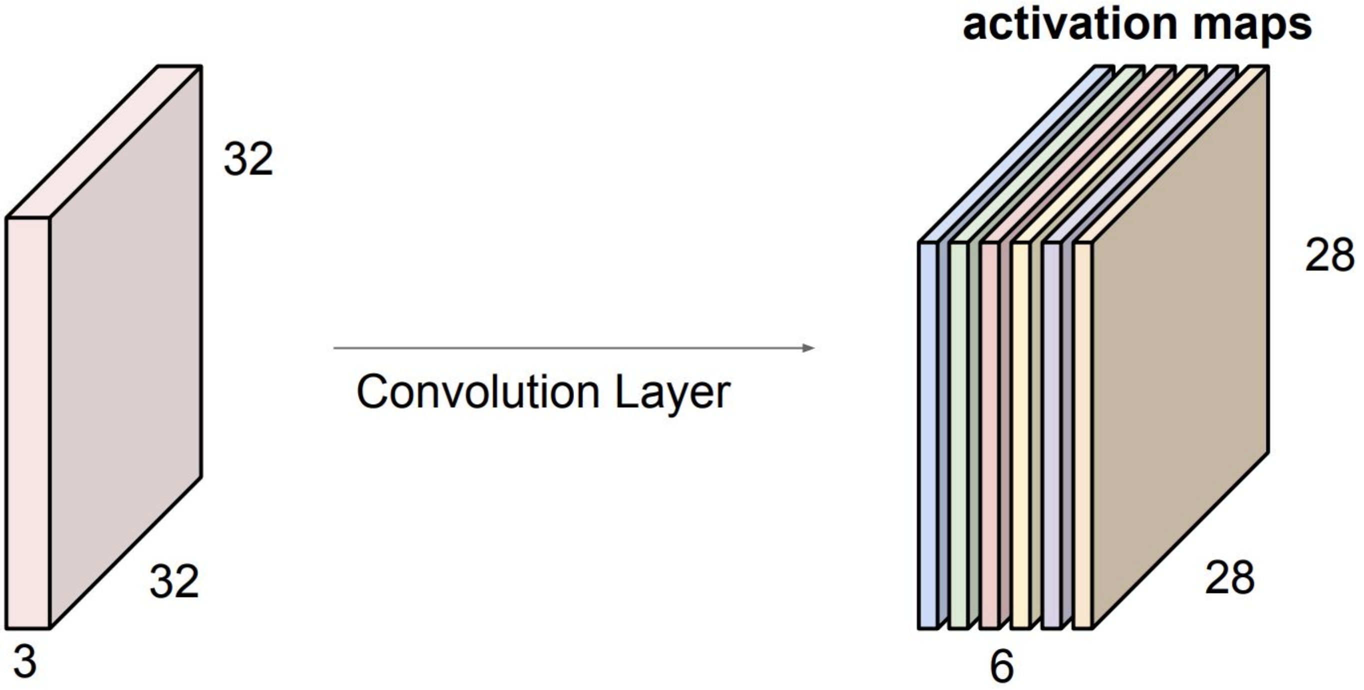
Convolutional layer

convolve (slide) over all spatial locations

source: http://cs231n.stanford.edu/slides/2016/winter1516_lecture7.pdf

Convolutional layer

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

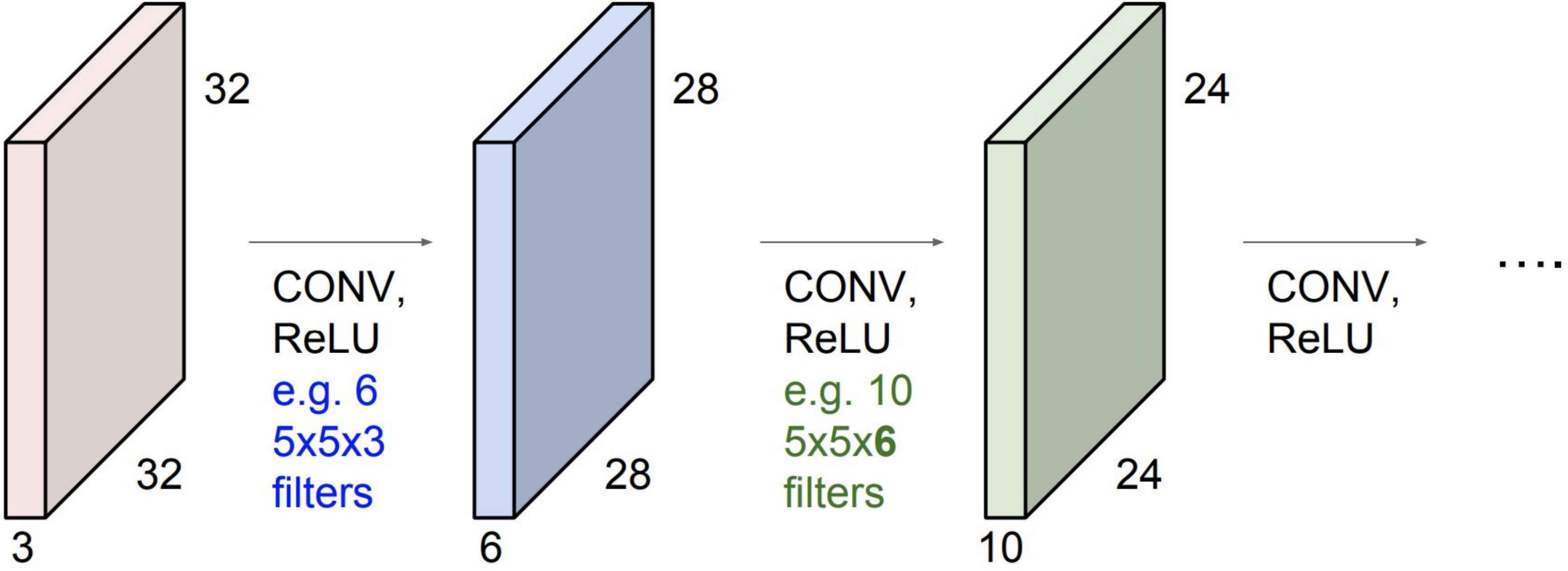


We stack these up to get a "new image" of size 28x28x6!

source: http://cs231n.stanford.edu/slides/2016/winter1516_lecture7.pdf

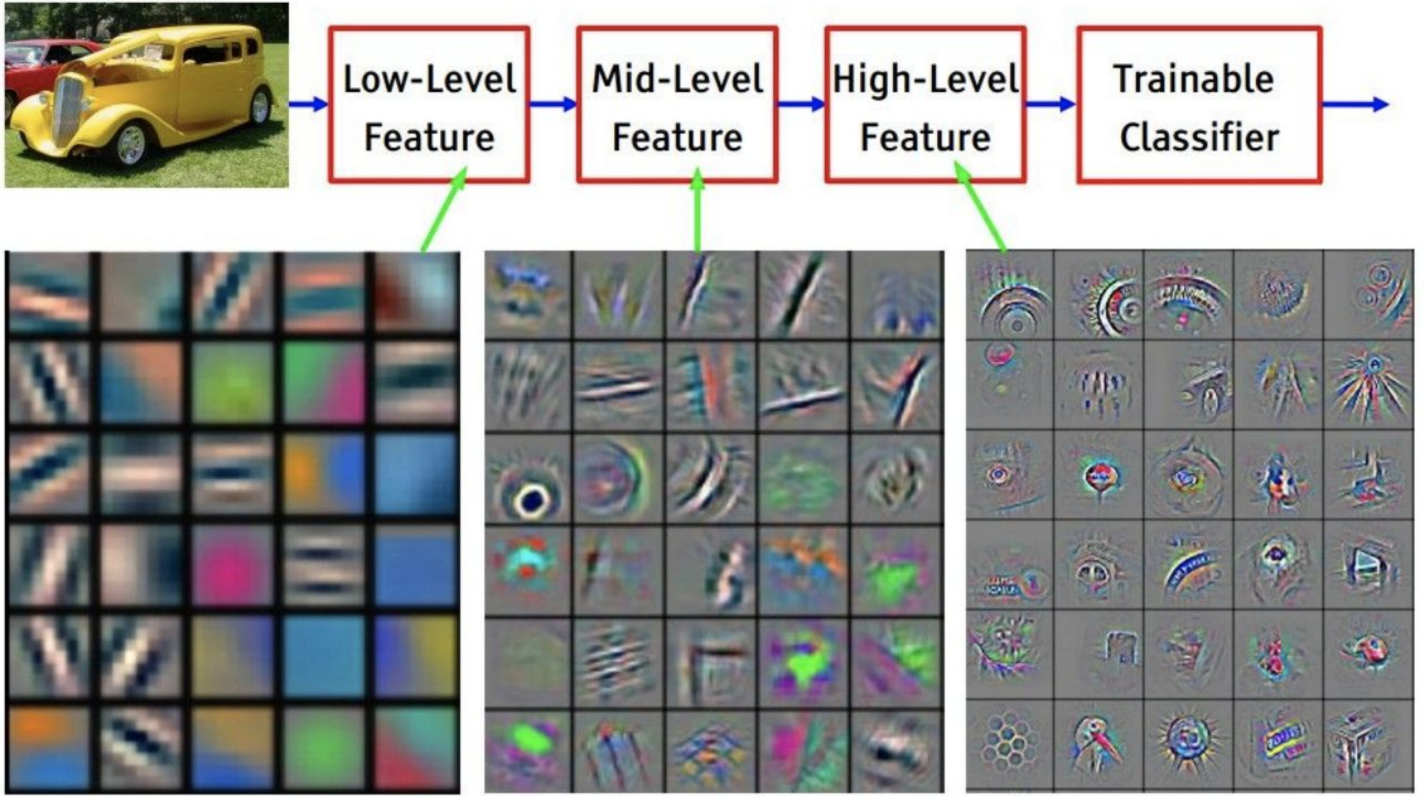
Convolutional layer

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



source: http://cs231n.stanford.edu/slides/2016/winter1516_lecture7.pdf

Convolutional layer

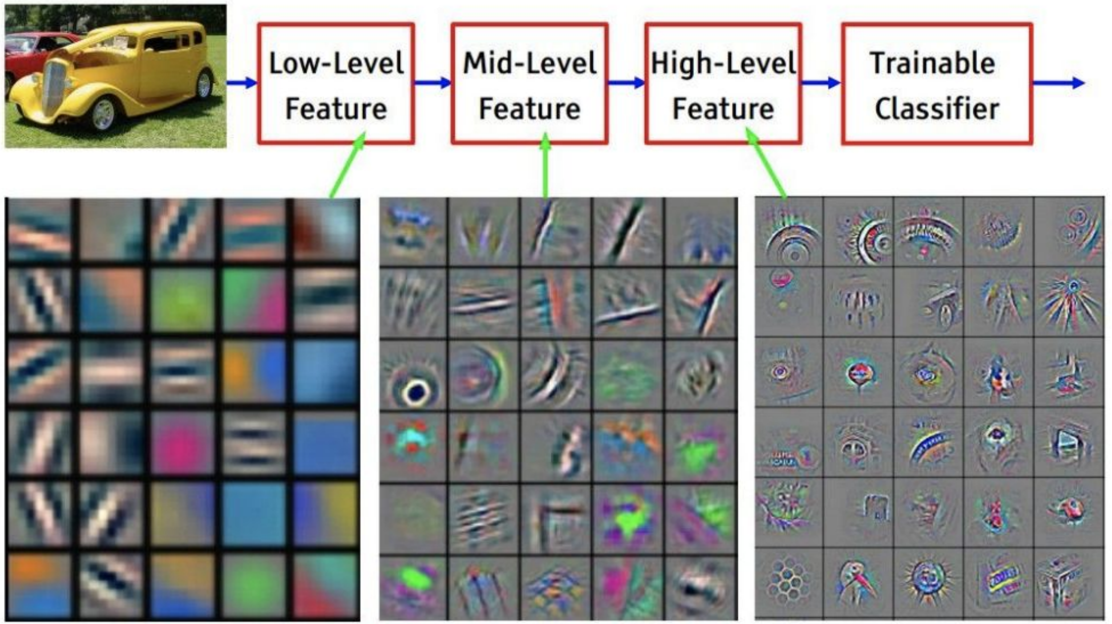


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

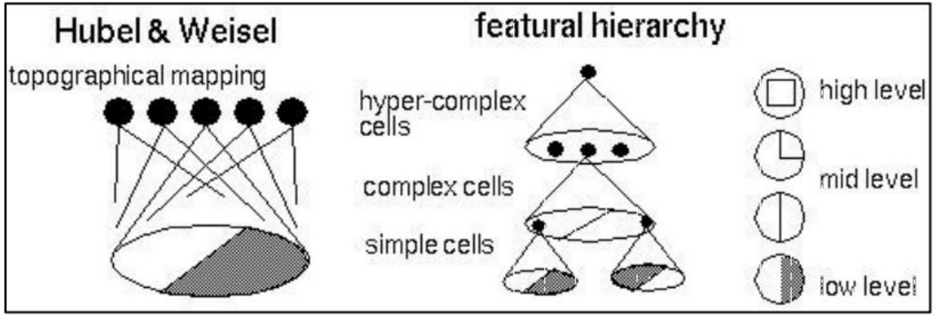
[From Yann LeCun slides]

source: http://cs231n.stanford.edu/slides/2016/winter1516_lecture7.pdf

Convolutional layer and visual cortex



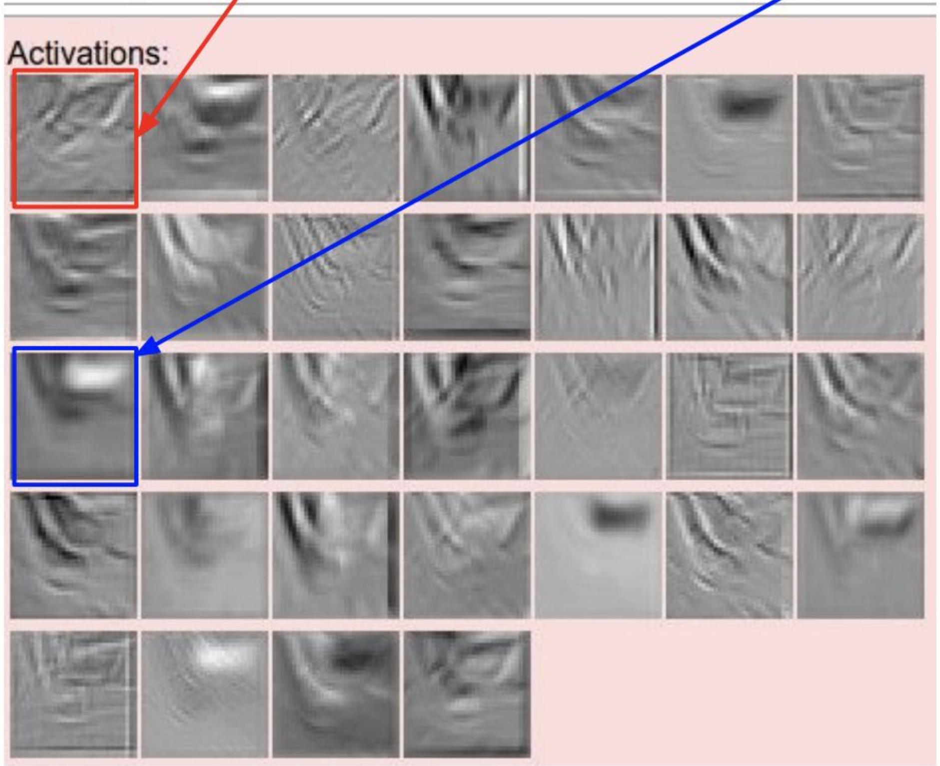
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]



[From Yann LeCun slides]

one filter =>
one activation map

example 5x5 filters
(32 total)



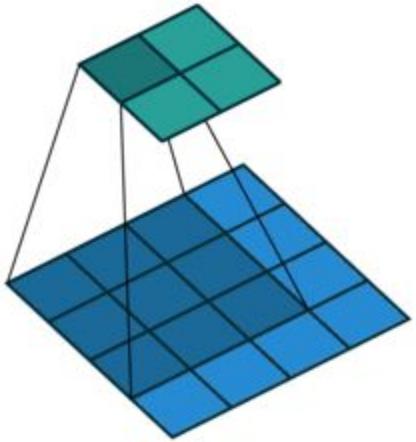
We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

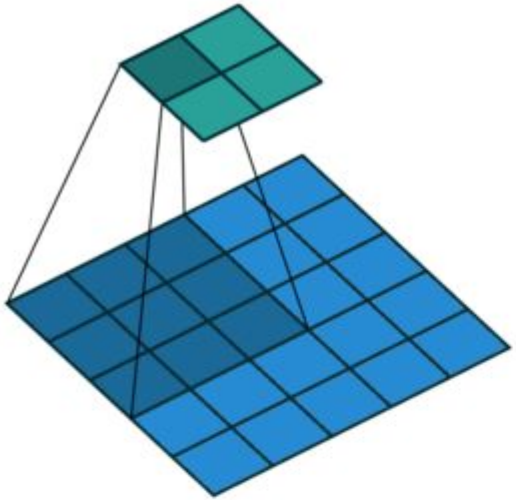
elementwise multiplication and sum of a filter and the signal (image)

source: http://cs231n.stanford.edu/slides/2016/winter1516_lecture7.pdf

Strides, padding in convolutional layer



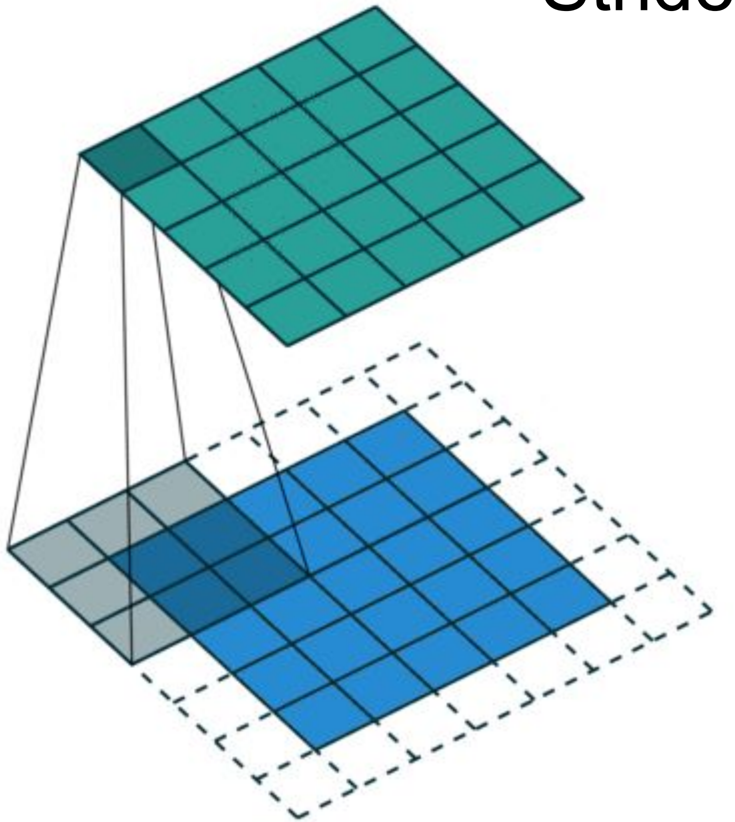
No padding,
no strides



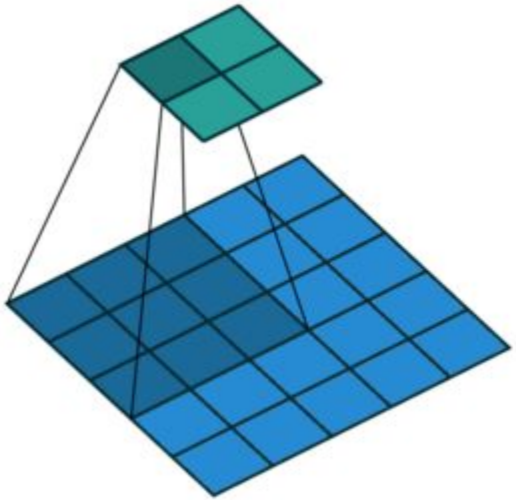
No padding,
with strides

Image source: https://github.com/vdumoulin/conv_arithmetic

Strides, padding in convolutional layer



With padding,
no strides

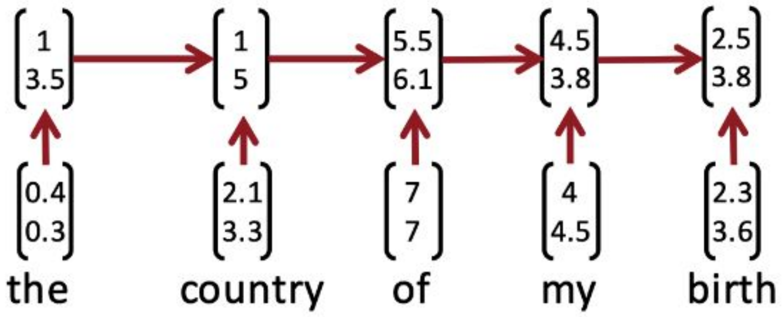


No padding,
with strides

Image source: https://github.com/vdumoulin/conv_arithmetic

Applying CNNs to texts

From RNN to CNN



- Recurrent neural nets can not capture phrases without prefix context and often capture too much of last words in final vector

Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

From RNN to CNN

- RNN: Get compositional vectors for grammatical phrases only

- CNN: What if we compute vectors for every possible phrase?
  - Example: *"the country of my birth"* computes vectors for:
    - *the country, country of, of my, my birth, the country of, country of my, of my birth, the country of my, country of my birth*

- Regardless of whether it is grammatical
- Wouldn't need parser
- Not very linguistically or cognitively plausible

Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

Recap: n-gramms

The brown dog plays with a little cat

The brown

brown dog

dog plays

plays with

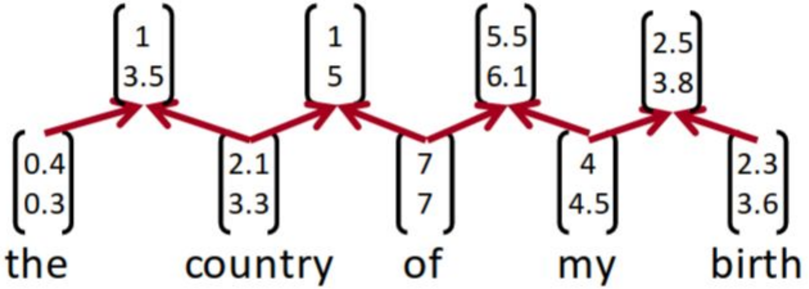
with a

a little

little cat

From RNN to CNN

- Imagine using only bigrams



- Same operation as in RNN, but for every pair

$$p = \tanh\left(W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b\right)$$

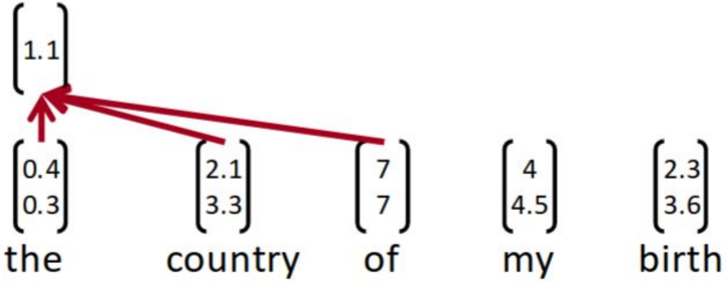
- Can be interpreted as convolution over the word vectors

Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

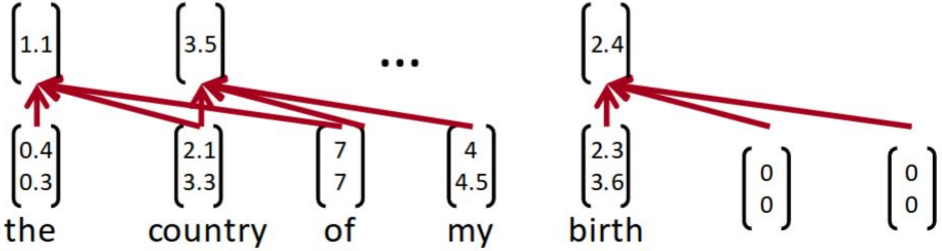
One layer CNN

- Simple convolution + pooling
- Window size may be different (2 or more)
- The feature map based on bigrams:

$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$



$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

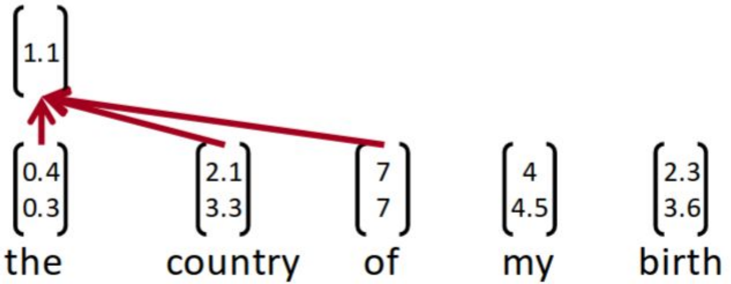


Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

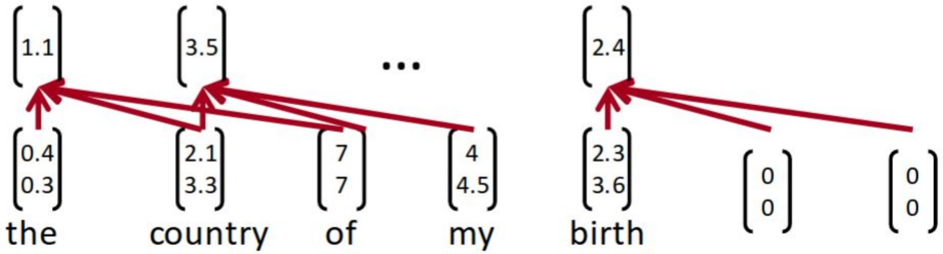
One layer CNN

- Simple convolution + pooling
- Window size may be different (2 or more)
- The feature map based on bigrams:

$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$



$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$



What's next?

We need more features!

Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

One layer CNN

- Feature representation is based on some applied filter:

$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

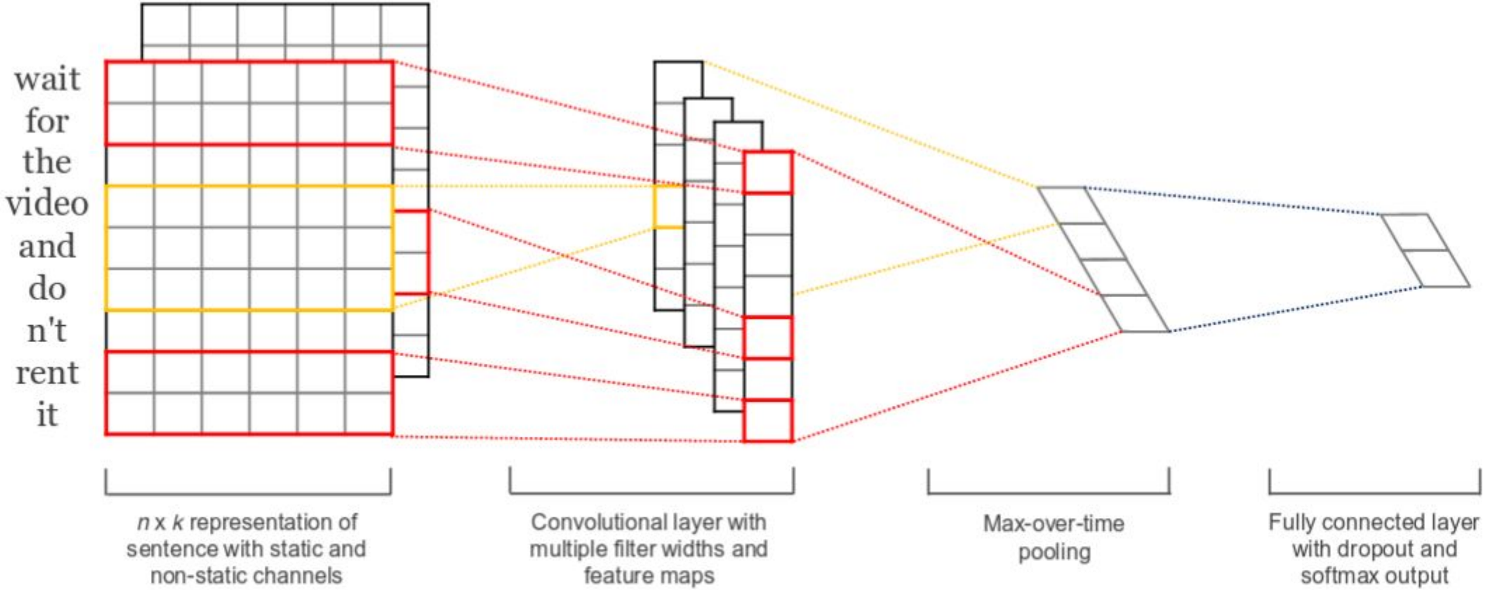
- Let's use pooling: $\hat{c} = \max\{\mathbf{c}\}$

- Now the length of **c** is irrelevant!

- So we can use filters based on unigrams, bigrams, tri-grams, 4-grams, etc.

Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

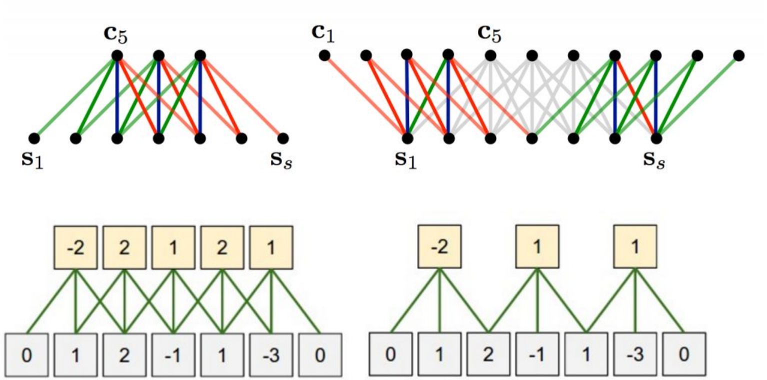
Another example from Kim (2014) paper



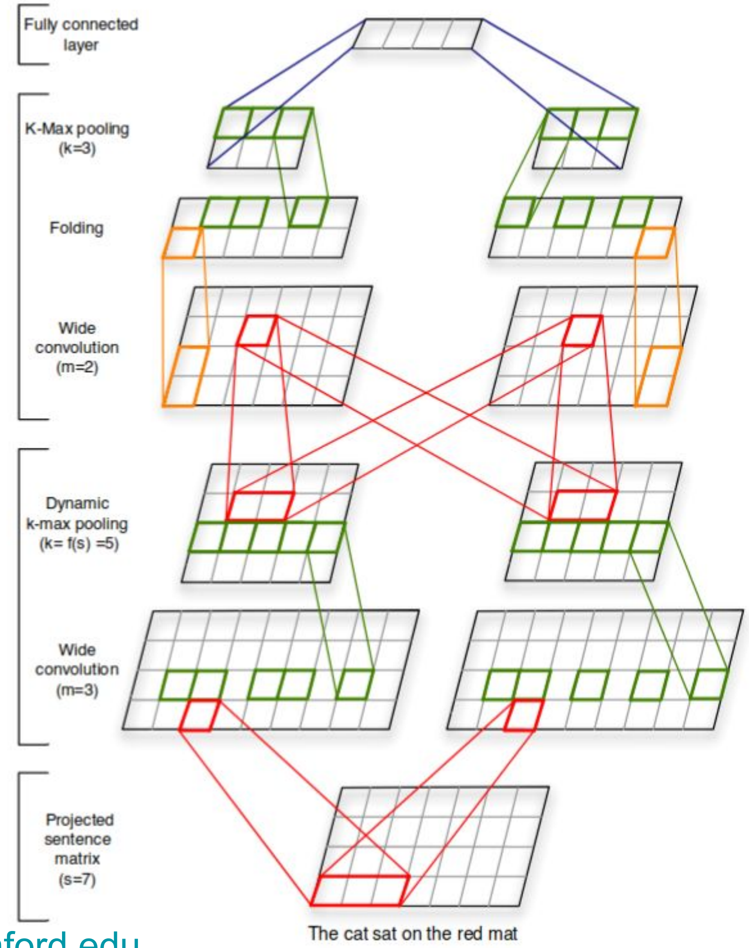
Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

More about CNN

- Narrow vs wide convolution (stride and zero-padding)



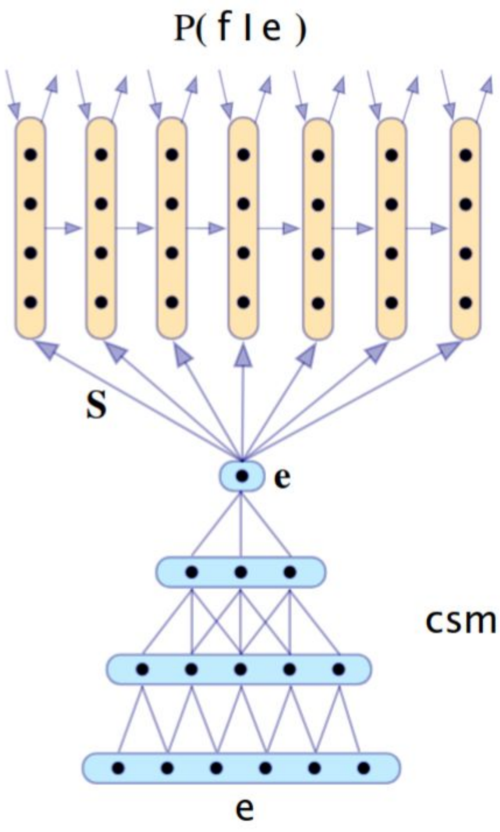
- Complex pooling schemes over sequences
- Great readings (e.g. Kalchbrenner et. al. 2014)



Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

CNN applications

- Neural machine translation: CNN as encoder, RNN as decoder
- Kalchbrenner and Blunsom (2013) “Recurrent Continuous Translation Models”
- One of the first neural machine translation efforts



Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

Approaches comparison

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	–	–	–	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	–	–	–	–
RNTN (Socher et al., 2013)	–	45.7	85.4	–	–	–	–
DCNN (Kalchbrenner et al., 2014)	–	48.5	86.8	–	93.0	–	–
Paragraph-Vec (Le and Mikolov, 2014)	–	**48.7**	87.8	–	–	–	–
CCAE (Hermann and Blunsom, 2013)	77.8	–	–	–	–	–	87.2
Sent-Parser (Dong et al., 2014)	79.5	–	–	–	–	–	86.3
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	–	–	**93.6**	–	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	–	–	93.4	–	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	–	–	**93.6**	–	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	–	–	–	–	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	–	–	–	–	–	82.7	–
SVM_S (Silva et al., 2011)	–	–	–	–	**95.0**	–	–

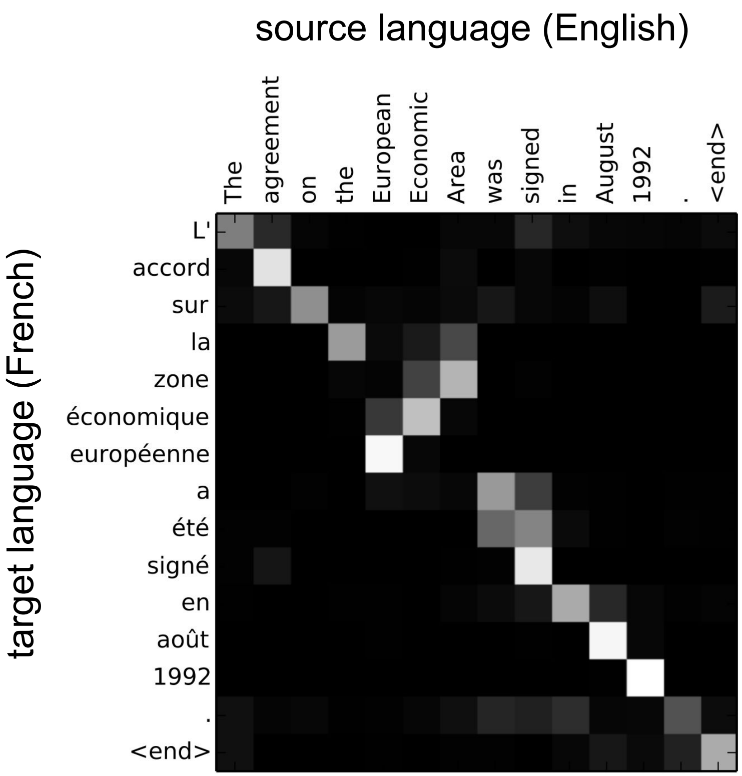
Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

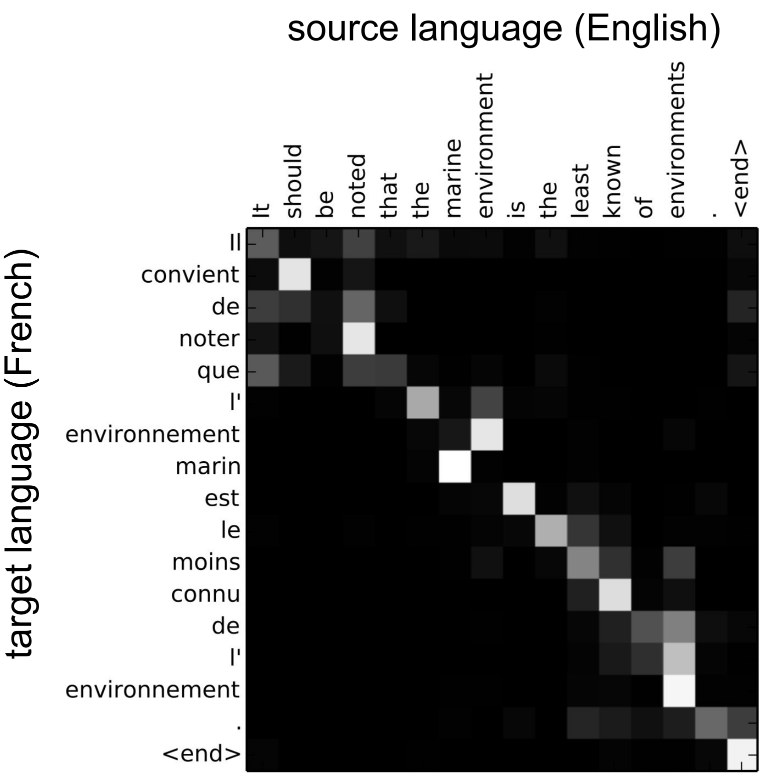
Outro and Q & A

- Vanishing gradient is present not only in RNNs
  - Use some kind of memory or skip-connections
- LSTM and GRU are both great
  - GRU is quicker, LSTM catch more complex dependencies
- Rule of thumb: start with LSTM, but switch to GRU if you want something more efficient
- Clip your gradients
- Combining RNN and CNN worlds? Why not ;)

That's all. Feel free to ask any questions.

Attention outro





Bahdanau et al. "Neural Machine Translation by Jointly Learning to Align and Translate", 2014