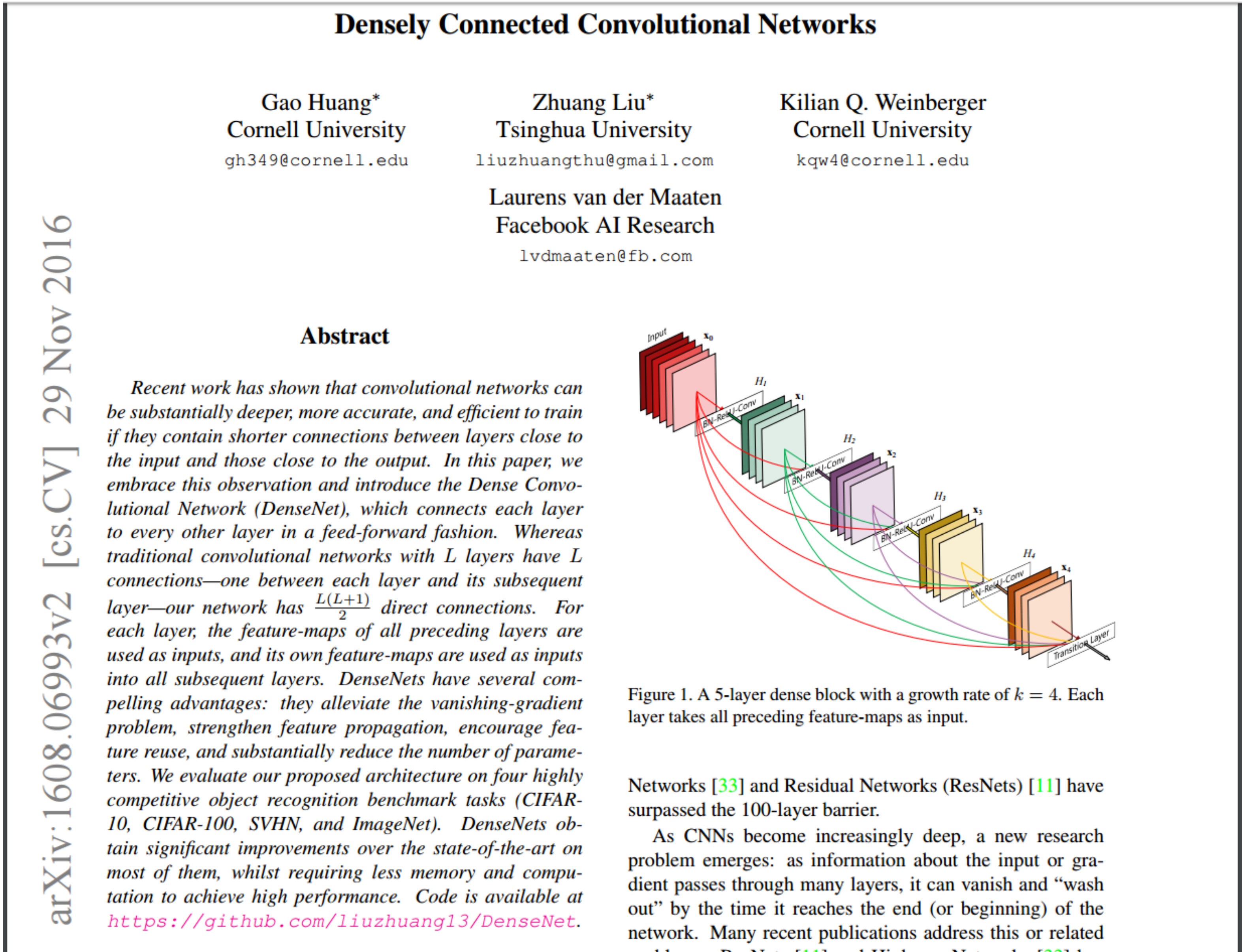
Advanced Prediction Models

CVPR 2017 Best Paper



Densely Connected Convolutional Networks

Gao Huang*
Cornell University
gh349@cornell.edu

Zhuang Liu*
Tsinghua University
liuzhuangthu@gmail.com

Kilian Q. Weinberger
Cornell University
kqw4@cornell.edu

Laurens van der Maaten
Facebook AI Research
lvdmaaten@fb.com

Abstract

Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. In this paper, we embrace this observation and introduce the Dense Convolutional Network (DenseNet), which connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections—one between each layer and its subsequent layer—our network has $\frac{L(L+1)}{2}$ direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. We evaluate our proposed architecture on four highly competitive object recognition benchmark tasks (CIFAR-10, CIFAR-100, SVHN, and ImageNet). DenseNets obtain significant improvements over the state-of-the-art on most of them, whilst requiring less memory and computation to achieve high performance. Code is available at https://github.com/liuzhuang13/DenseNet.

Figure 1. A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Networks [33] and Residual Networks (ResNets) [11] have surpassed the 100-layer barrier.

As CNNs become increasingly deep, a new research problem emerges: as information about the input or gradient passes through many layers, it can vanish and "wash out" by the time it reaches the end (or beginning) of the network. Many recent publications address this or related

Today's Outline

- Introduction to Natural Language Processing
- Models with Simple Representations
- Word Embeddings and Word2Vec

Introduction to Natural Language Processing

Natural Language Processing (NLP)

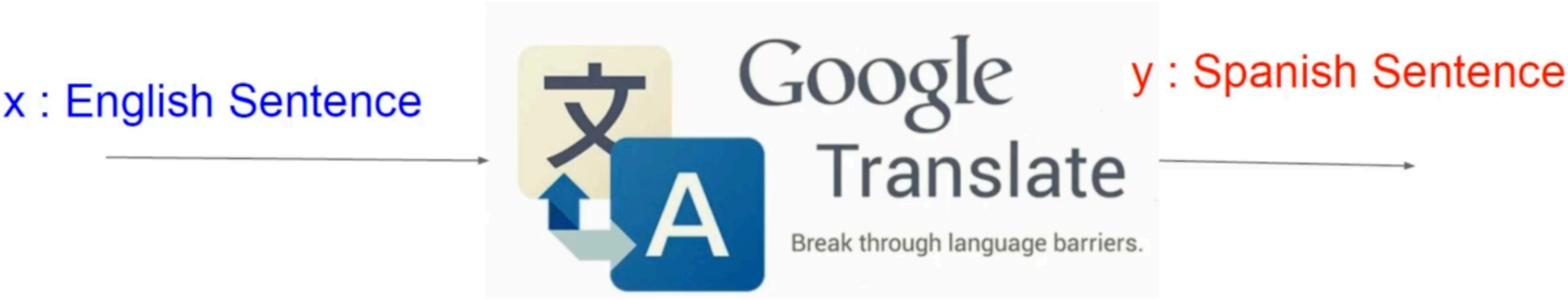
- Concerns will all aspects of natural languages

- *We will only sample a very narrow set of topics in this area*

- We will sample a few ways to deal with text
  - Text is a sequence of symbols
  - Naïve way: represent them as one-hot encoded vectors
  - We will see some better methods today

[1] Figure: https://github.com/oxford-cs-deepnlp-2017/lectures

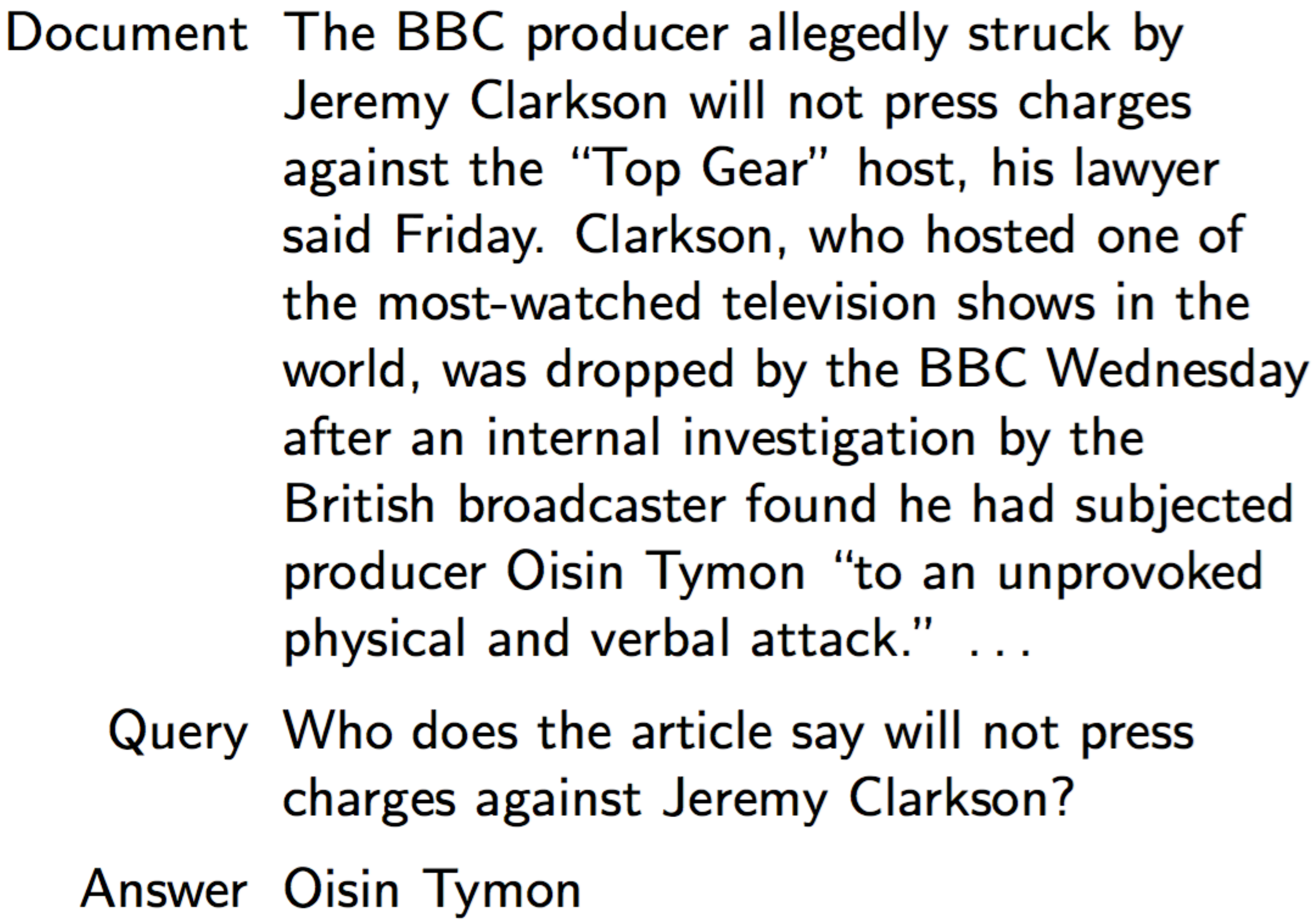
Motivation: Machine Translation


x : English Sentence

y : Spanish Sentence

[1] Figure: https://github.com/oxford-cs-deepnlp-2017/lectures

Motivation: Query Answering



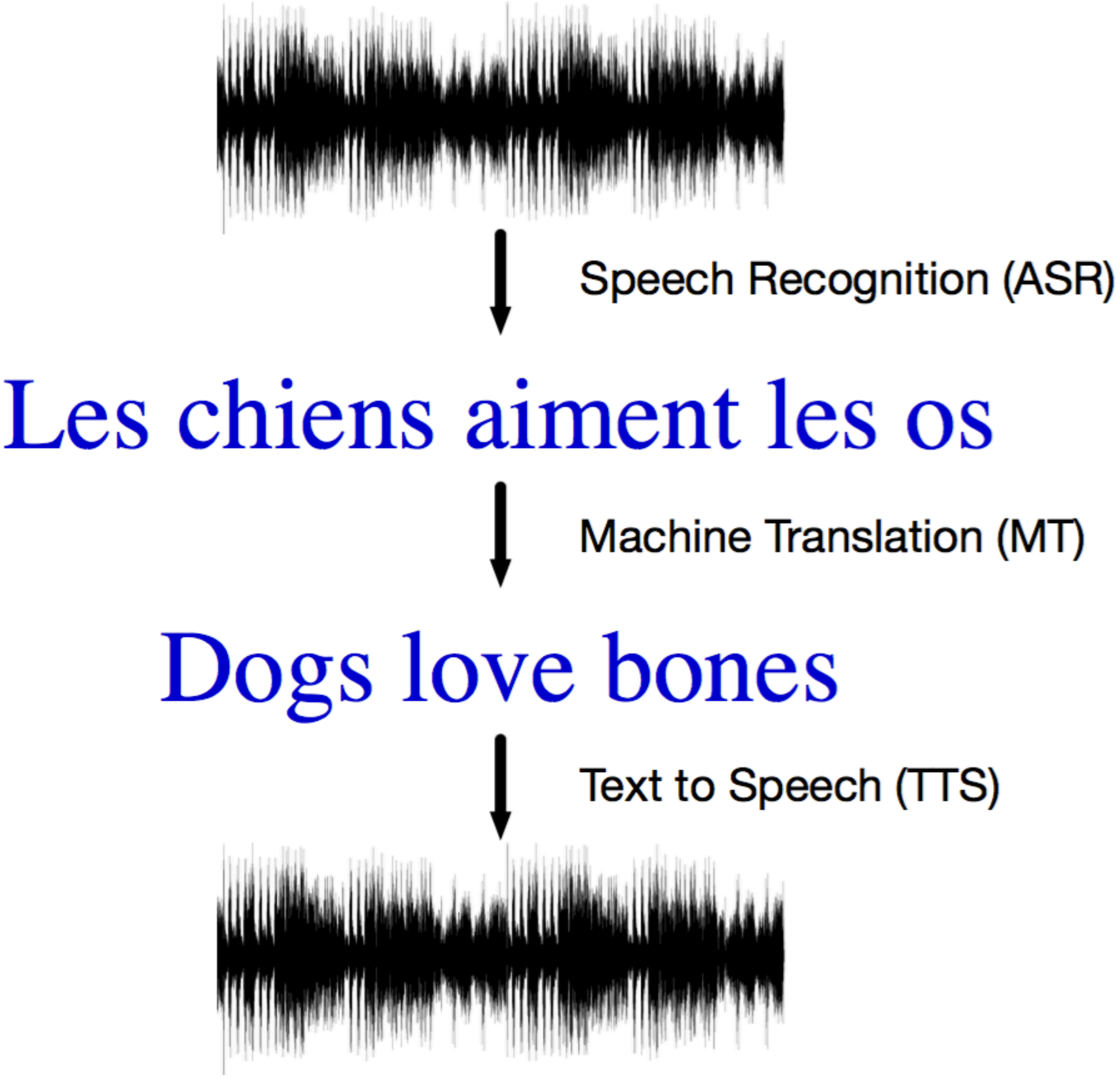
Document The BBC producer allegedly struck by Jeremy Clarkson will not press charges against the “Top Gear” host, his lawyer said Friday. Clarkson, who hosted one of the most-watched television shows in the world, was dropped by the BBC Wednesday after an internal investigation by the British broadcaster found he had subjected producer Oisin Tymon “to an unprovoked physical and verbal attack.” . . .

Query Who does the article say will not press charges against Jeremy Clarkson?

Answer Oisin Tymon

[1] Figure: https://github.com/oxford-cs-deepnlp-2017/lectures

Motivation: Speech to Speech



Speech Recognition (ASR)

Les chiens aiment les os

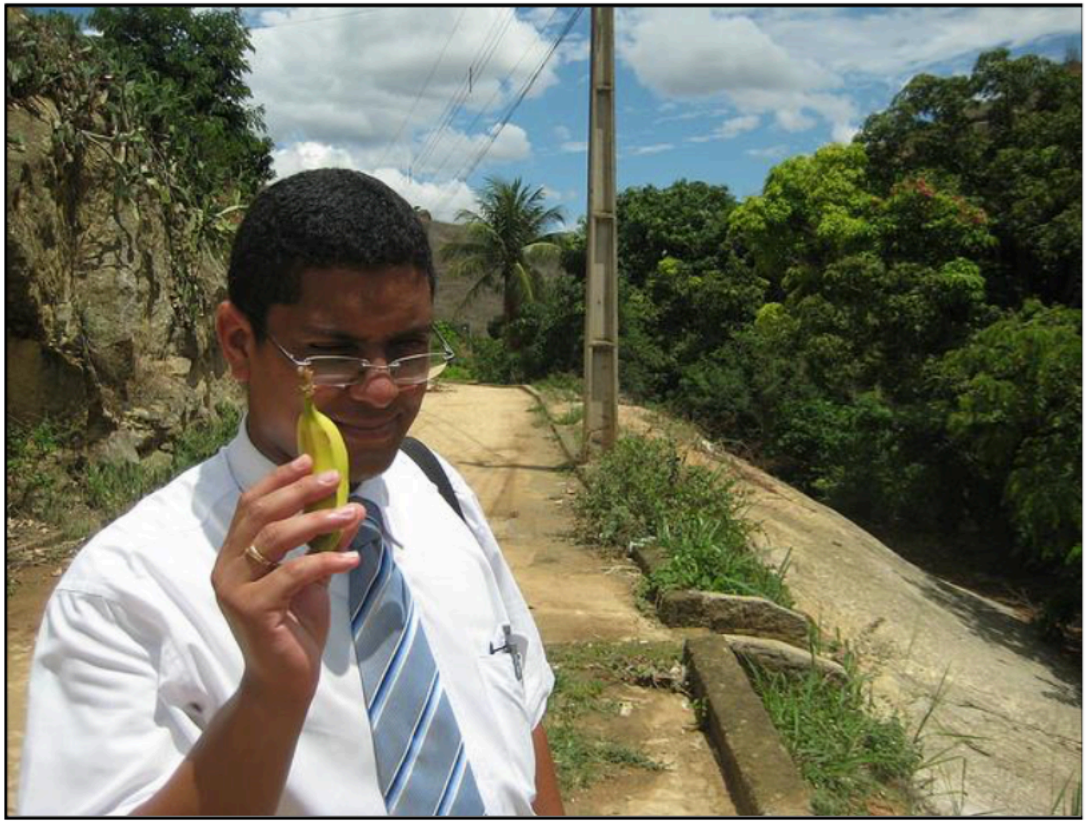
Machine Translation (MT)

Dogs love bones

Text to Speech (TTS)

[1] Figure: https://github.com/oxford-cs-deepnlp-2017/lectures

Motivation: Visual Query Answering



What is the man holding?
Does it appear to be raining?
Does this man have 20/20 vision?

[1] Figure: https://github.com/oxford-cs-deepnlp-2017/lectures

Motivation: Harder Text Problems

Sense

I saw her duck

Idioms

He kicked a goal

He kicked the ball

He caught the ball

He kicked the bucket

Reference

The ball did not fit in the box because **it** was too [big/small].

[1]Figure: https://github.com/oxford-cs-deepnlp-2017/lectures

Models with Simple Representations

Side-stepping Word-word Relationships

- We will look at a few models that
  - Don’t explicitly account for word-word relationships

- These are:
  - Naïve Bayes Spam Filter
  - Markov Language Model
  - Latent Dirichlet Allocation
  - Conditional Random Field based Classifier (appendix)
  - CNN based Sentence Classifier

Naïve Bayes Spam Filter

- **Key assumption**
  Words occur independently of each other given the label of the document

$$p(w_1, \ldots, w_n | \text{spam}) = \prod_{i=1}^{n} p(w_i | \text{spam})$$

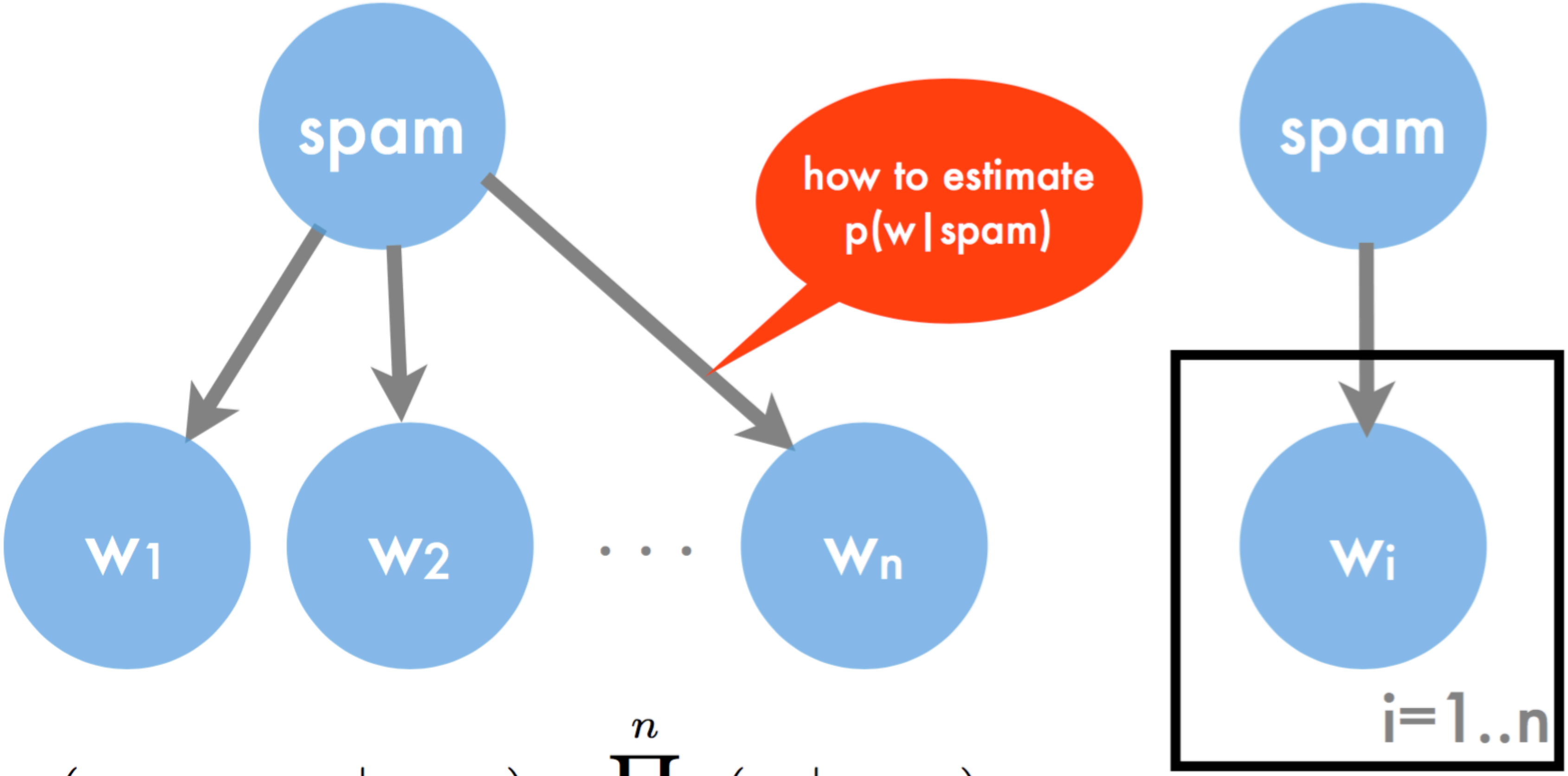
- Spam classification via Bayes Rule

$$p(\text{spam} | w_1, \ldots, w_n) \propto p(\text{spam}) \prod_{i=1}^{n} p(w_i | \text{spam})$$

- Parameter estimation
  Compute spam probability and word distributions for spam and ham

[1] Reference: Alex Smola (2011)

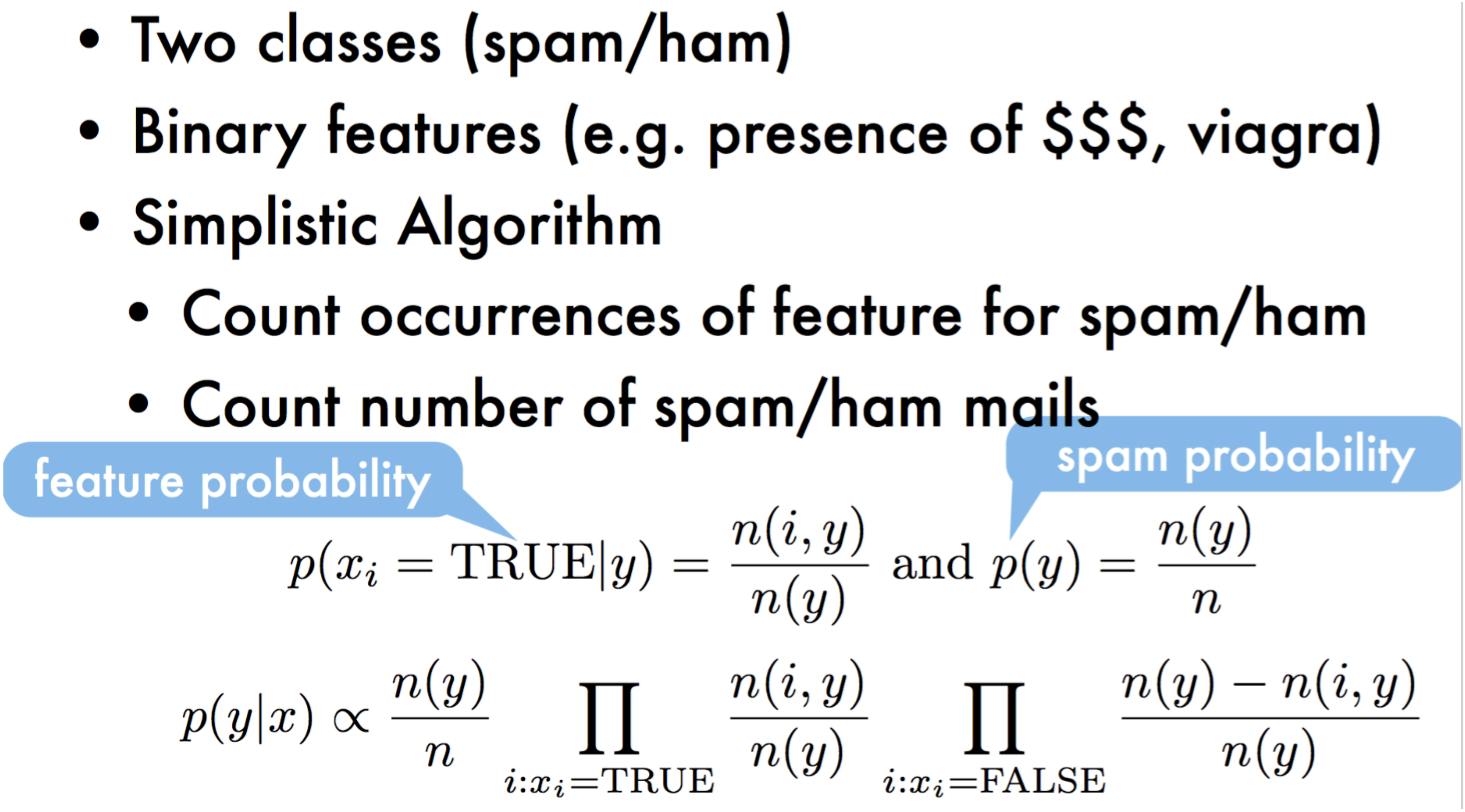
Naïve Bayes Spam Filter



$$p(w_1, \ldots, w_n|\text{spam}) = \prod_{i=1}^{n} p(w_i|\text{spam})$$

[1] Reference: Alex Smola (2011)

Naïve Bayes Spam Filter



- Two classes (spam/ham)
- Binary features (e.g. presence of $$$, viagra)
- Simplistic Algorithm
  - Count occurrences of feature for spam/ham
  - Count number of spam/ham mails

feature probability

spam probability

$$p(x_i = \mathrm{TRUE}|y) = \frac{n(i,y)}{n(y)} \text{ and } p(y) = \frac{n(y)}{n}$$

$$p(y|x) \propto \frac{n(y)}{n} \prod_{i:x_i=\mathrm{TRUE}} \frac{n(i,y)}{n(y)} \prod_{i:x_i=\mathrm{FALSE}} \frac{n(y)-n(i,y)}{n(y)}$$

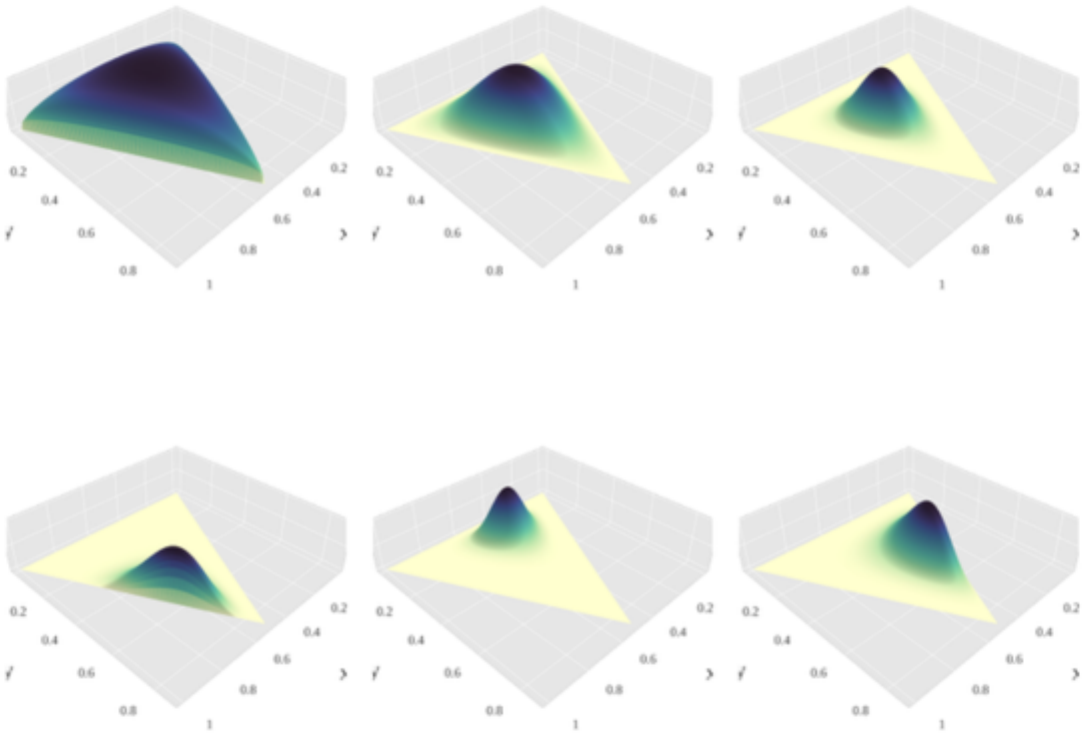
[1] Reference: Alex Smola (2011)

A Character-level Language Markov Model

- Character-level language model allows you to generate new text

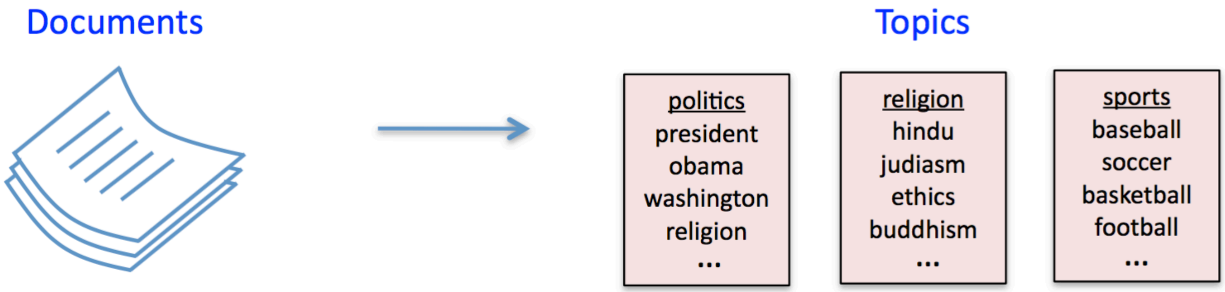
- It can be modeled using a maximum likelihood based method
- Pick a fixed order = 2
- For a training sequence, e.g., {h,e,l,l,o}
  - Compute $P(\{l\{|\{h, e\}) = \frac{\#\{l,h,e\}}{\#\{h,e\}}$
  - Do this for every three characters in the vocabulary
- Generate new text by sampling!

Aside: Dirichlet Distribution

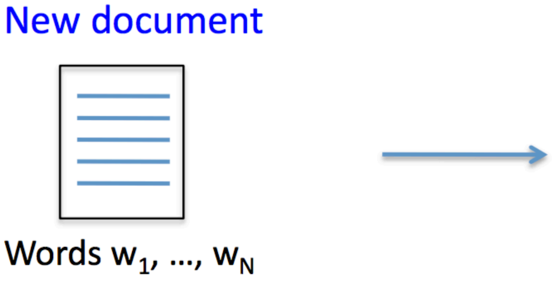


Latent Dirichlet Allocation

- **Topic models** are powerful tools for exploring large data sets and for making inferences about the content of documents



- Many applications in information retrieval, document summarization, and classification



What is this document about?

weather	.50
finance	.49
sports	.01

Distribution of topics θ

- LDA is one of the simplest and most widely used topic models

[1] Reference: David Sontag (2013)

Latent Dirichlet Allocation

1. Sample the document's **topic distribution** θ (aka topic vector)

$$\theta \sim \mathrm{Dirichlet}(\alpha_{1:T})$$

where the $\{\alpha_t\}_{t=1}^T$ are fixed hyperparameters. Thus θ is a distribution over T topics with mean $\theta_t = \alpha_t / \sum_{t'} \alpha_{t'}$

2. For $i = 1$ to N, sample the **topic** z_i of the i'th word

$$z_i | \theta \sim \theta$$

3. ... and then sample the actual **word** w_i from the z_i'th topic

$$w_i | z_i \sim \beta_{z_i}$$

where $\{\beta_t\}_{t=1}^T$ are the *topics* (a fixed collection of distributions on words)

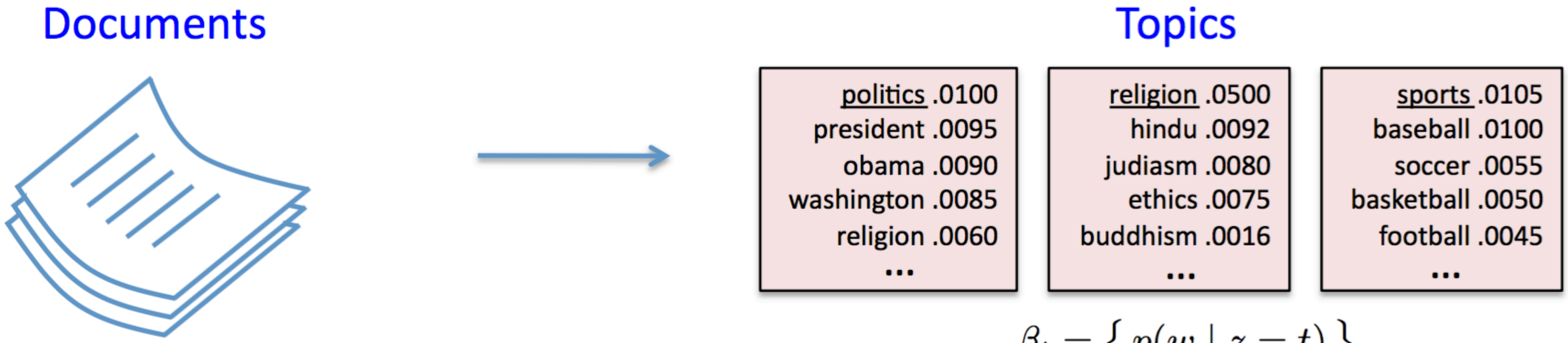
[1] Reference: David Sontag (2013)

Latent Dirichlet Allocation

… and then sample the actual **word** w_i from the z_i'th topic

$$w_i | z_i \sim \beta_{z_i}$$

where $\{\beta_t\}_{t=1}^{T}$ are the *topics* (a fixed collection of distributions on words)



Documents

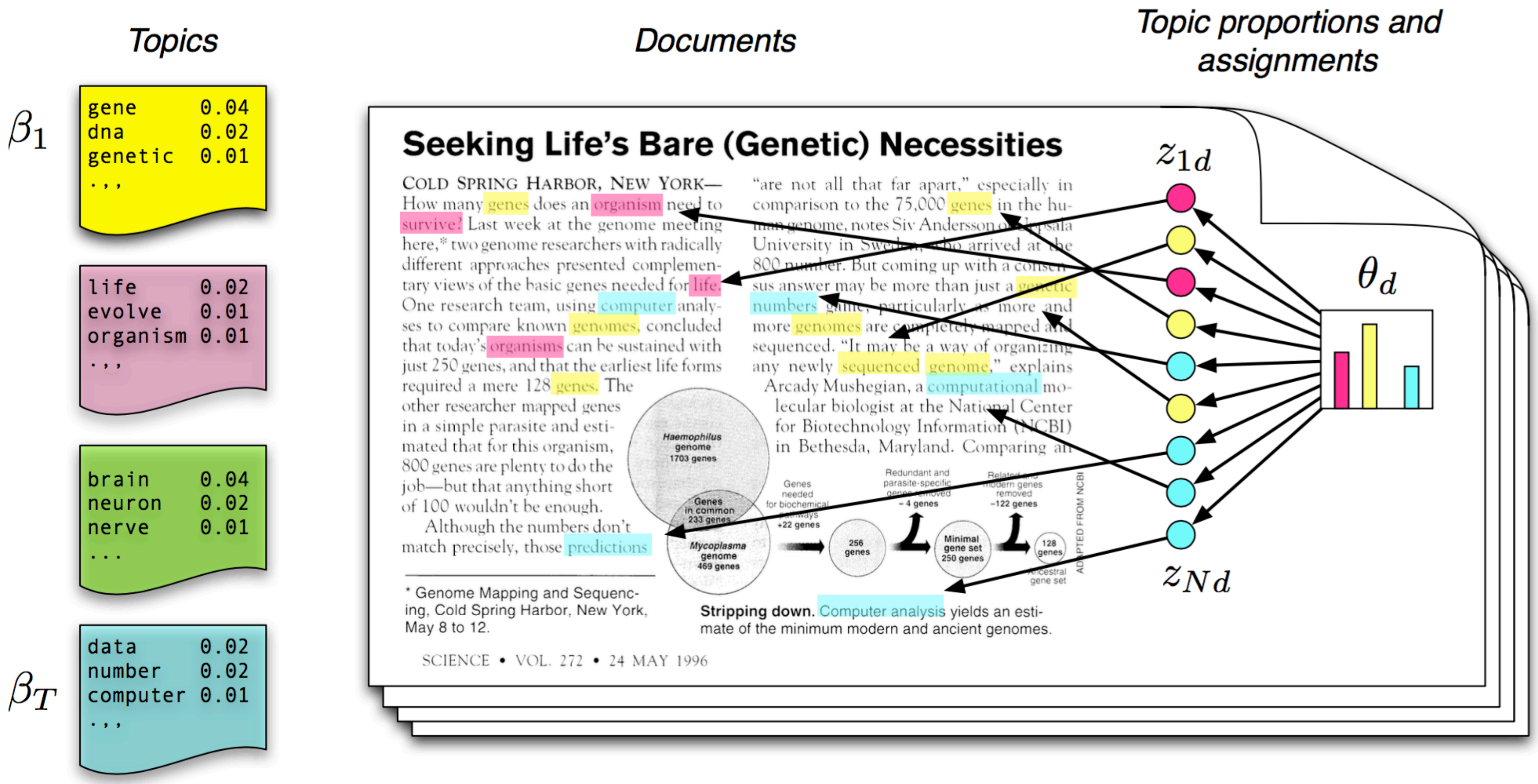
Topics

politics .0100	religion .0500	sports .0105
president .0095	hindu .0092	baseball .0100
obama .0090	judiasm .0080	soccer .0055
washington .0085	ethics .0075	basketball .0050
religion .0060	buddhism .0016	football .0045
...	...	...

$$\beta_t = \{\, p(w \mid z = t) \,\}$$

[1] Reference: David Sontag (2013)

Latent Dirichlet Allocation



gene	0.04
dna	0.02
genetic	0.01
...	

life	0.02
evolve	0.01
organism	0.01
...	

brain	0.04
neuron	0.02
nerve	0.01
...	

data	0.02
number	0.02
computer	0.01
...	

Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK— How many genes does an organism need to survive? Last week at the genome meeting here,* two genome researchers with radically different approaches presented complementary views of the basic genes needed for life. One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest life forms required a mere 128 genes. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those predictions "are not all that far apart," especially in comparison to the 75,000 genes in the human genome, notes Siv Andersson of Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a genetic numbers game, particularly as more and more genomes are completely mapped and sequenced. "It may be a way of organizing any newly sequenced genome," explains Arcady Mushegian, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an

Haemophilus genome 1703 genes
Genes in common 233 genes
Mycoplasma genome 469 genes
Genes needed for biochemical pathways +22 genes
256 genes
Redundant and parasite-specific genes removed – 4 genes
Minimal gene set 250 genes
Related and modern genes removed –122 genes
128 genes
Ancestral gene set
ADAPTED FROM NCBI

Stripping down. Computer analysis yields an estimate of the minimum modern and ancient genomes.

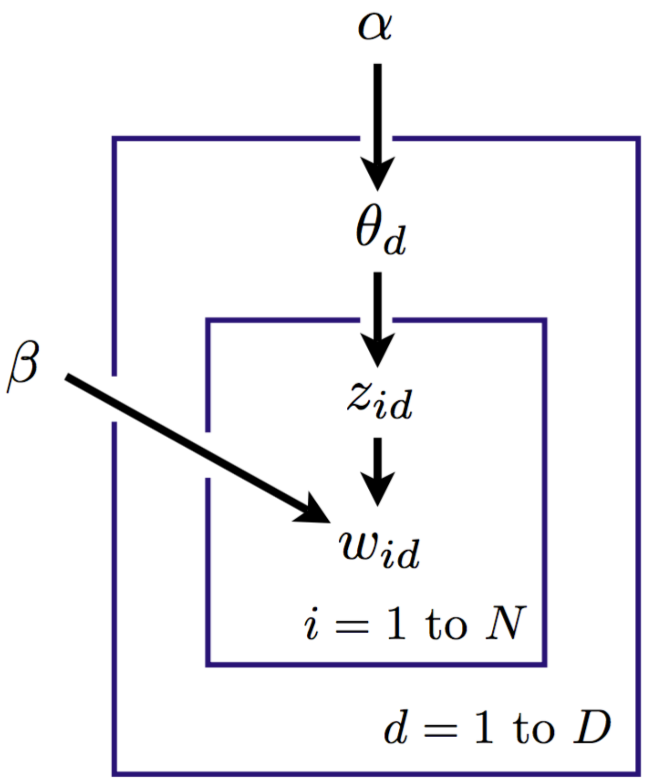
* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

SCIENCE • VOL. 272 • 24 MAY 1996

(Blei, *Introduction to Probabilistic Topic Models*, 2011)

[1] Reference: David Sontag (2013)

Latent Dirichlet Allocation



Dirichlet hyperparameters

Topic distribution for document

Topic-word distributions

Topic of word *i* of doc *d*

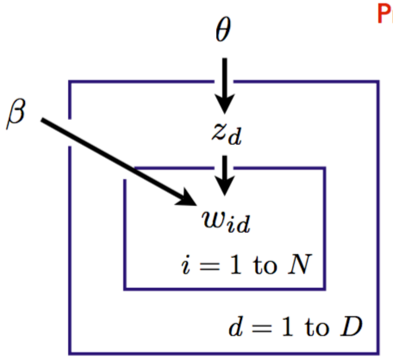
Word

Variables within a plate are replicated in a conditionally independent manner

[1] Reference: David Sontag (2013)

Latent Dirichlet Allocation

Topic-word distributions

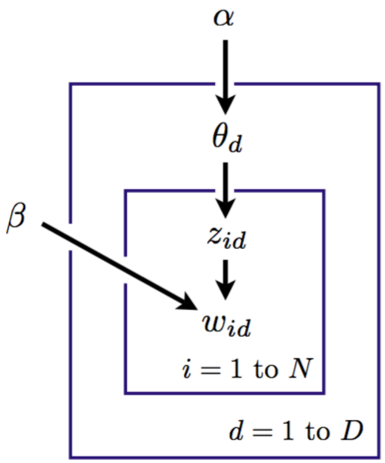


Prior distribution over topics

Topic of doc *d*

Word

Topic-word distributions



Dirichlet hyperparameters

Topic distribution for document

Topic of word *i* of doc *d*

Word

- Model on left is a **mixture model**
  - Called *multinomial* naive Bayes (a word can appear multiple times)
  - Document is generated from a single topic
- Model on right (LDA) is an **admixture model**
  - Document is generated from a distribution over topics

[1] Reference: David Sontag (2013)

CNN based Sentence Classification

- Input is a sequence of words (variable)
- Output is a class label (fixed)

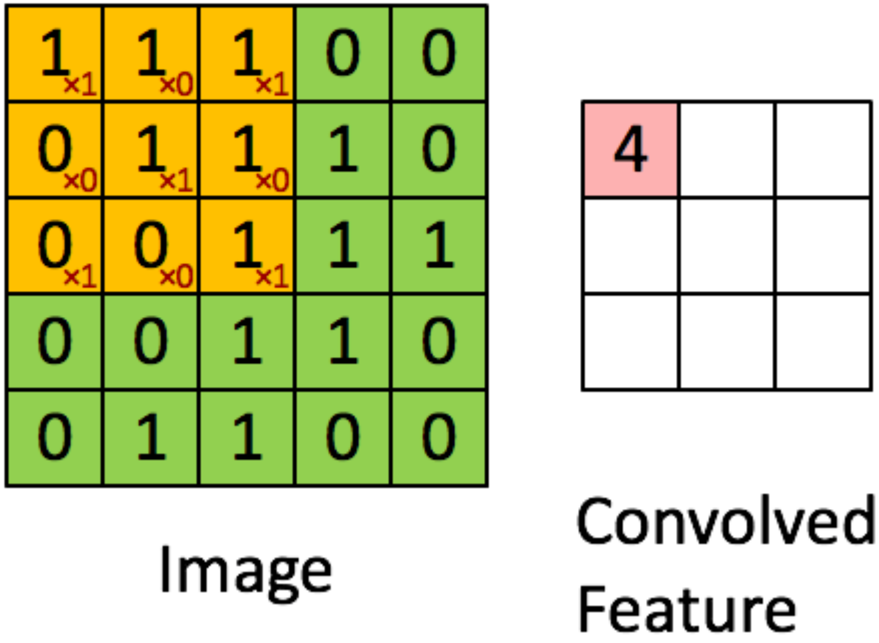
- Baseline 1:
  - Ignore sequence
  - Ignore semantic information
  - Treat input as a fixed length bag of words
  - This is a fixed size input and output classifier

CNN based Sentence Classification

- Baseline 2:
  - Weighted average of the word vectors as a vector for the sentence
  - Still loses word order
  - Retains some semantic information
  - Again, a fixed size input and output classifier

CNN based Sentence Classification

- Can also use Convolutional Neural Network!
  - For NLP, they became popular 2014
  - Less prominent currently due to other techniques
- Recall



[1] Figure: http://deeplearning.stanford.edu/

Example I: Sentence Classification

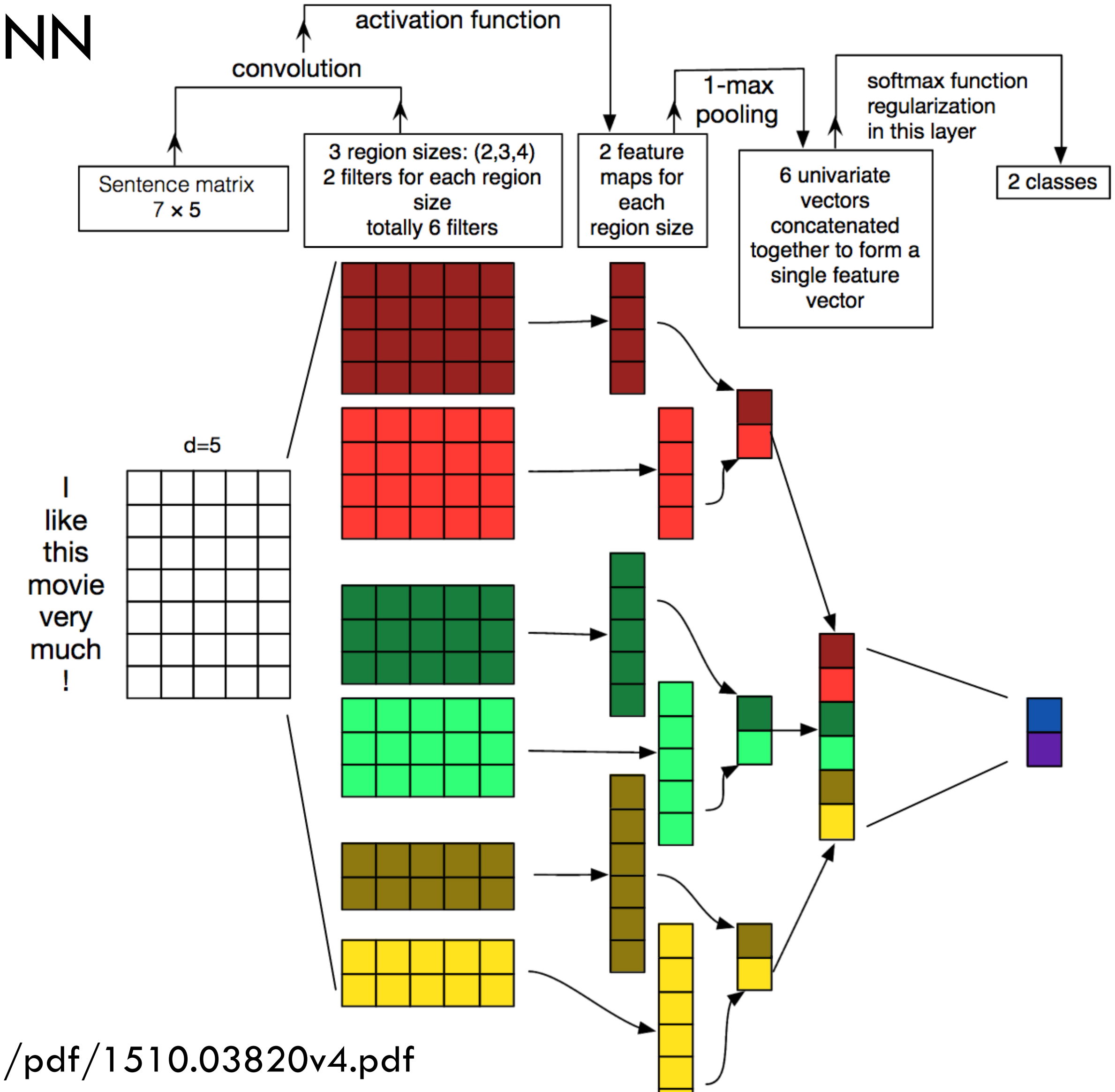
- As you already know, CNNs capture
  - Location invariance
    - Example: In images, don't care where the 'cat' is in the input
  - Compositionality
    - Example: In images, lower level features to higher level patterns

- We will represent the sentence as a matrix
  - Each row for one word

[1]Reference: https://arxiv.org/pdf/1408.5882.pdf

Example I: Sentence Classification

- Example CNN


activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

[1] Reference: https://arxiv.org/pdf/1510.03820v4.pdf

Embeddings

A Different Way of Dealing with Words

- Want semantically similar words to be represented similarly
  - This is the idea behind Vector Space Models in NLP

- Distributional Hypothesis (Firth 1957)
  - Words that appear in the same contexts share semantic meaning
- Two types of approaches:
  - Count based (PCA based)
  - Prediction based (creating auxiliary task etc)

[1] Reference: https://www.tensorflow.org/tutorials/word2vec/

Dealing with Words

- A word embedding W: words → $\mathbb{R}^n$ is a function
  - Parametric
- Dimension n can be high: e.g., 300

- Example:
  - W(‘university’) $= (0.3, -0.1, 2.0, 1.1, -1.5, \dots)$
  - W(‘class’) $= (0.5, 1.1, -0.7, 2.5, 0.2, \dots)$

Learning an Embedding

- How do we learn a good W?

- Start with the same intuitive idea as before
  - Initialize such that W outputs random vectors for each word
  - Change the parameters such that the embedding vectors are meaningful for a task

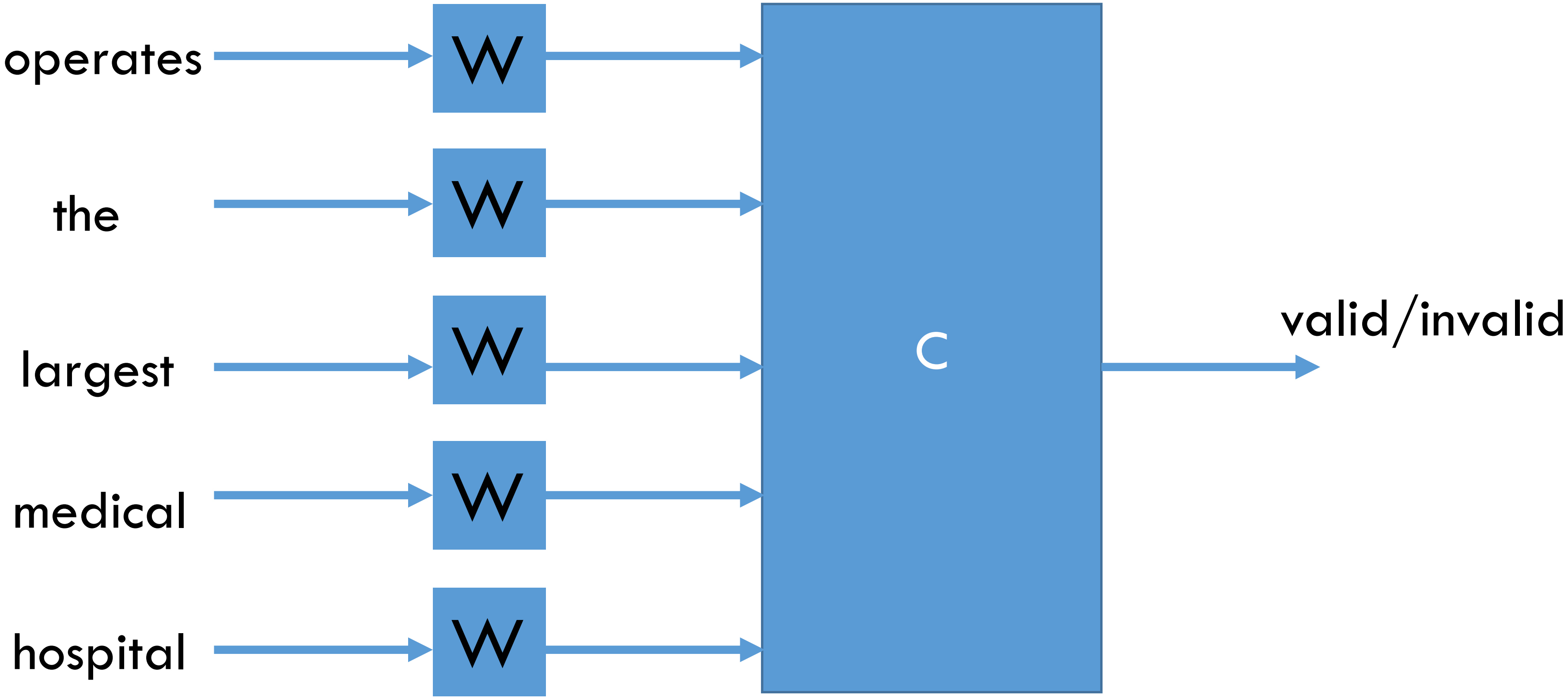
Which Task? (I)

- Train a network to classify whether an input sequence of 5 words is valid or not
  - The input sequence is called an N-gram (5-gram)

- We get data, say from Wikipedia
  - Example: “operates the largest medical school”
- Break ‘half’ of them by replacing a word in each sequence with a random word
  - Example: “operates the consistently medical school”

Which Task? (II)

- Pass each word through W to get the vectors
- Pass the vectors through C (a classifier)



Which Task? (III)

- In order to do the classification correctly, parameters for W and C should be good

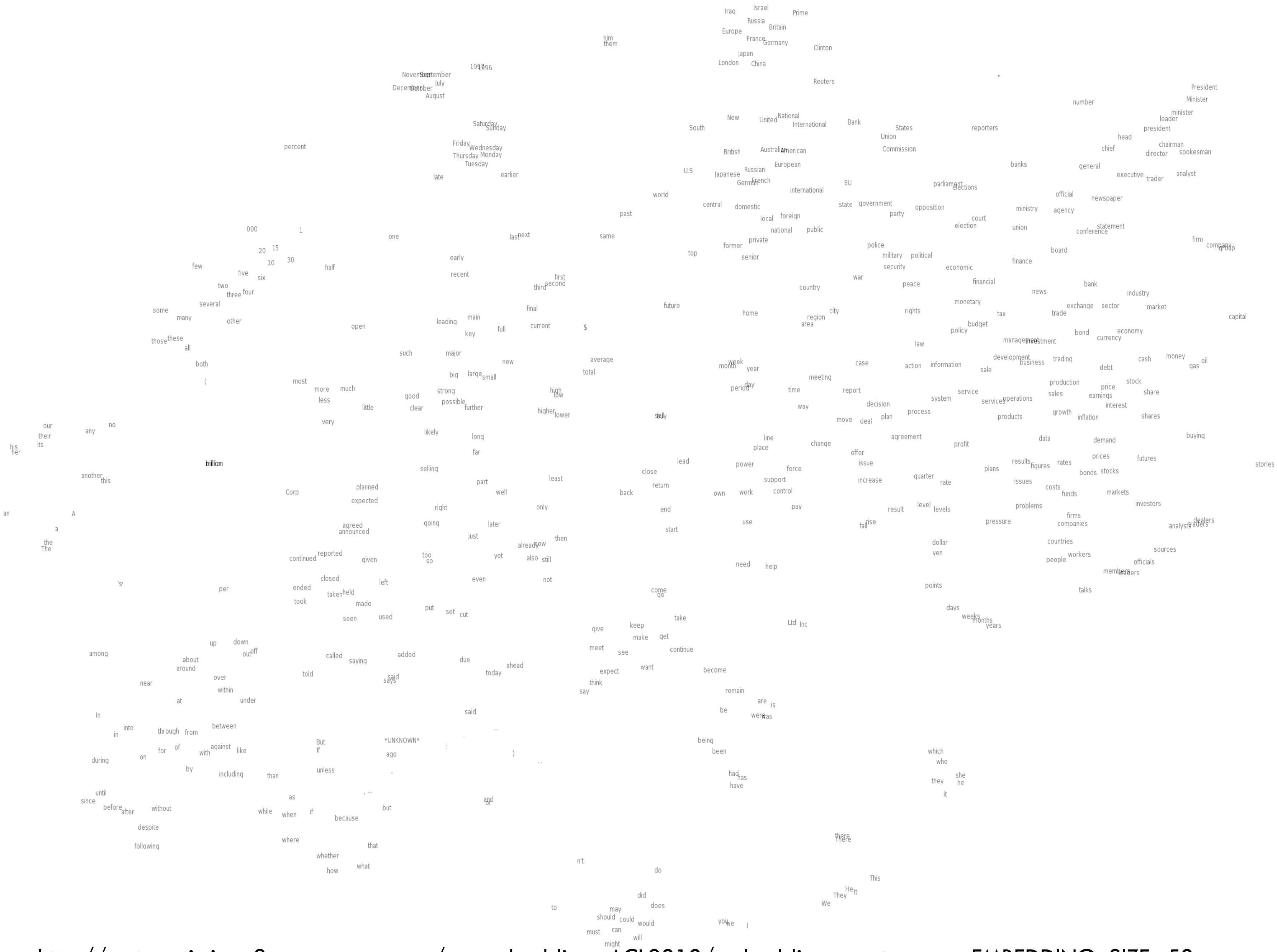
- The task itself is uninteresting and inconsequential
  - We could have defined a different task

- Our objective is to learn a good W

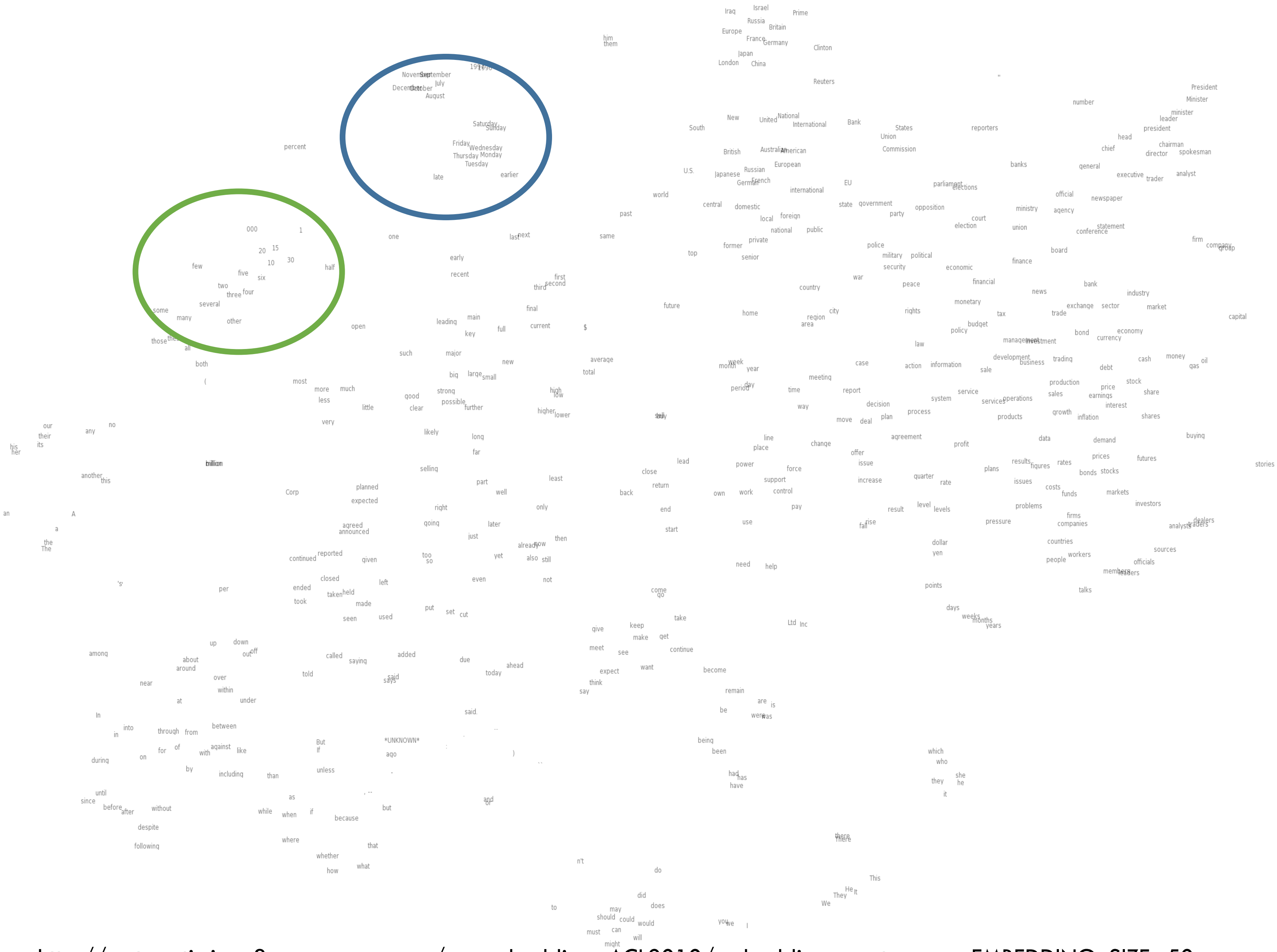
Quality of Embedding (I)

- Say, we learned a good W
  - See Word2Vec or GloVe (for pretrained embeddings)
- How to visualize? Use t-SNE

[1] Reference for t-SNE: http://lvdmaaten.github.io/tsne/



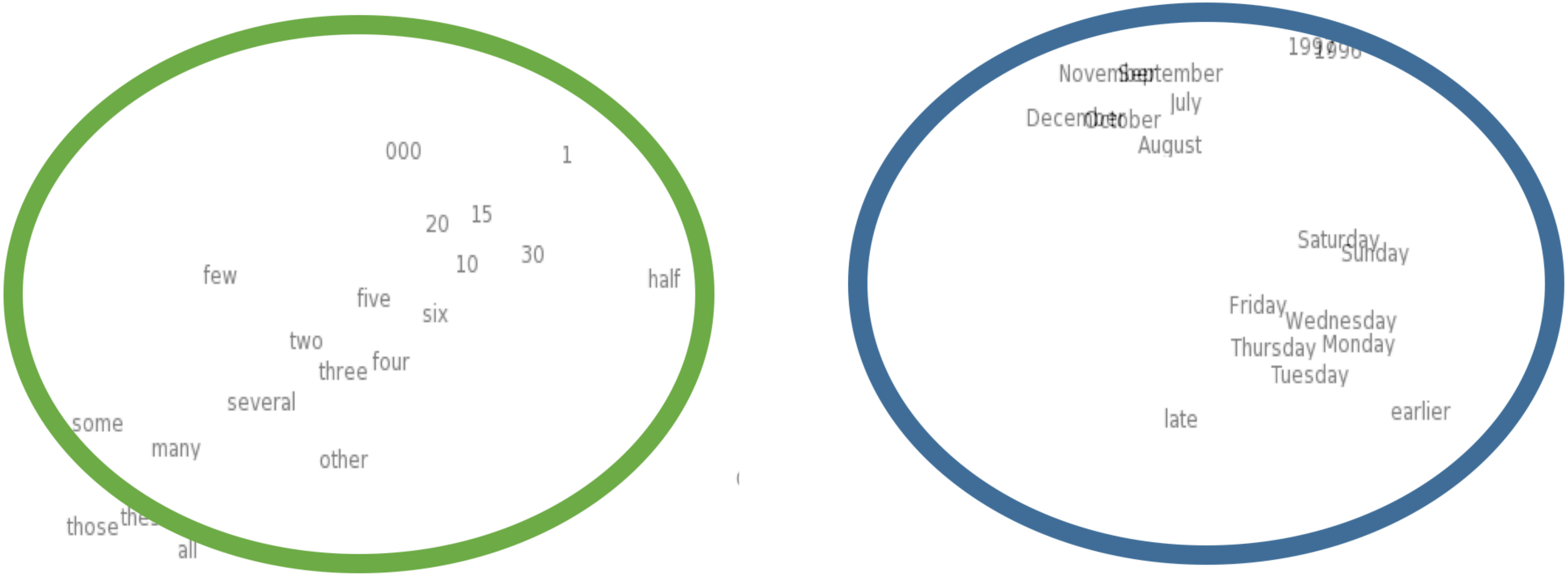
[1] Figure: http://metaoptimize.s3.amazonaws.com/cw-embeddings-ACL2010/embeddings-mostcommon.EMBEDDING_SIZE=50.png



[1] Figure: http://metaoptimize.s3.amazonaws.com/cw-embeddings-ACL2010/embeddings-mostcommon.EMBEDDING_SIZE=50.png

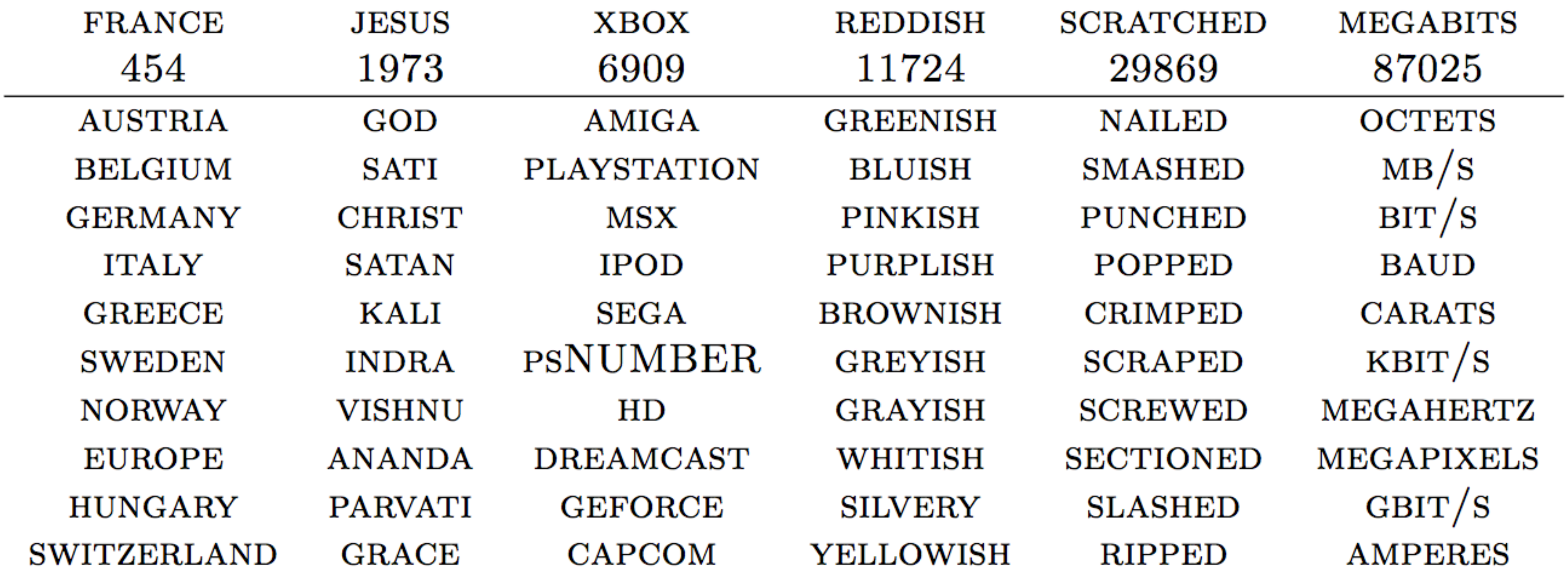
Quality of Embedding (II)

- We see similar words are close together



Quality of Embedding (III)

- Look at words closest in the embedding to a given word
- 10 nearest neighbors are listed here



FRANCE 454	JESUS 1973	XBOX 6909	REDDISH 11724	SCRATCHED 29869	MEGABITS 87025
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

[1] Figure: https://arxiv.org/pdf/1103.0398v1.pdf Page 23

An Attempt at Intuition (I)

- Is it natural for words with similar meanings to have similar vectors (hence nearest neighbors)?

- Example:
  - Change "operates the largest medical school" to "operates the biggest medical school"
  - If W maps biggest and largest close by
  - Then classifier C should still be able to work

[1] Reference: http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/

An Attempt at Intuition (II)

- Similar words getting mapped to close by vectors is great!
- We are not just limited to synonyms
- Example 1: “the inside wall is blue” to “the inside wall is red”
- Example 2: “the inside wall is blue” to “the inside ceiling is red”

[1] Reference: http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/

How Much Data?

- Clearly, we have to see all words (for whom we need embeddings)

- But we need not see their combinations

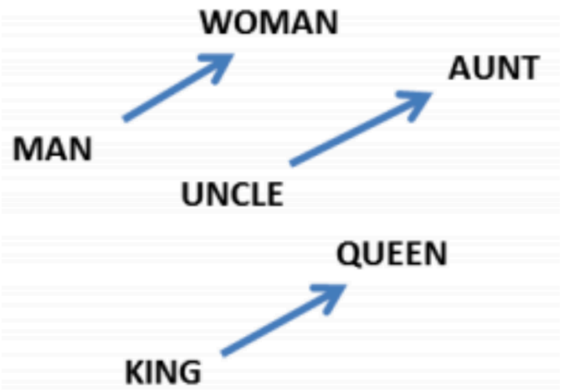
- Analogies allow us to generalize to new combination of words

- This is similar to humans: we have seen all words but have not seen all sentences with those words

[1] Reference: http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/

Difference of Vectors Property (I)

- Many word embeddings exhibit the following property as a side-effect:
  - Analogies are encoded in difference vectors



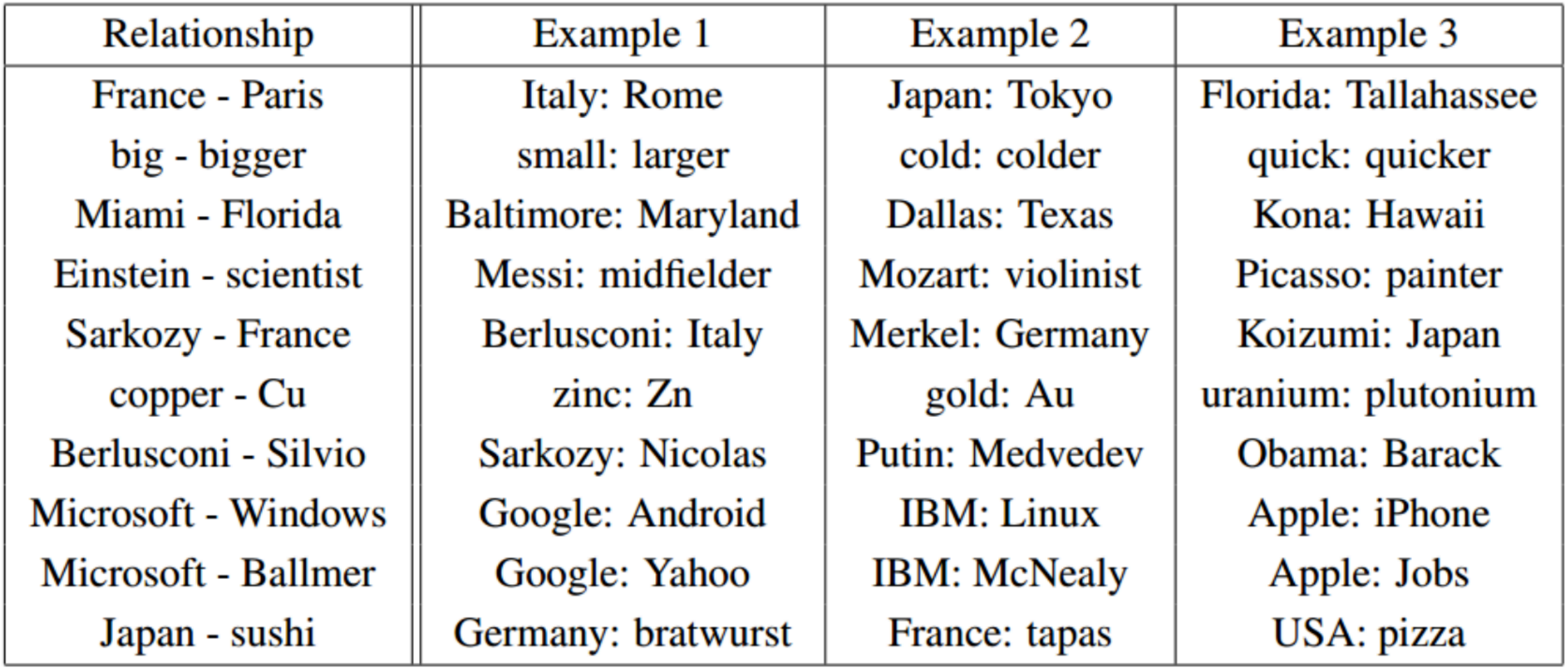
$$W(\text{``woman''}) - W(\text{``man''}) \simeq W(\text{``aunt''}) - W(\text{``uncle''})$$

$$W(\text{``woman''}) - W(\text{``man''}) \simeq W(\text{``queen''}) - W(\text{``king''})$$

[1] Reference: https://www.aclweb.org/anthology/N/N13/N13-1090.pdf

Difference of Vectors Property (II)

- For a pair of words, subtract their difference and add to another word. For example,
  - W(“France”) – W(“Paris”) $+$ W(“Rome”) $\approx$ W(“Italy”)



Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

[1] Reference: https://arxiv.org/pdf/1301.3781.pdf

Use of Embeddings (I)

- Embeddings represent unstructured data automatically in such a way that a subsequent task's performance is good
  - We have already seen image embeddings
  - Here, we are seeing word embeddings

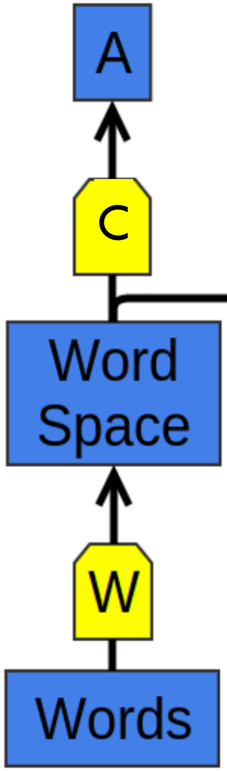
Use of Embeddings (I)

- Embeddings represent unstructured data automatically in such a way that a subsequent task's performance is good
  - We have already seen image embeddings
  - Here, we are seeing word embeddings

- Once a word embedding W is learned, we can use it for many other NLP (Natural Language Processing) tasks
  - Transfer learning (just like for images!)

Use of Embeddings (II)

- Learn a good representation (i.e., W) on some task and use it for other tasks



W and F learn to perform task A. Later, G can learn to perform B based on W.

> *The use of word representations… has become a key "secret sauce" for the success of many NLP systems in recent years, across tasks including named entity recognition, part-of-speech tagging, parsing, and semantic role labeling.* (Luong *et al.* (2013))

[1] Figure: http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/

Use of Embeddings (III)

- Key benefit of W
  - Can train using more than one kind of data

- Thus, we can learn a way to map multiple kinds of data into a single representation!

- Example: Bilingual word embedding[1]
  - English words
  - Mandarin words
  - Embed both words in the same space

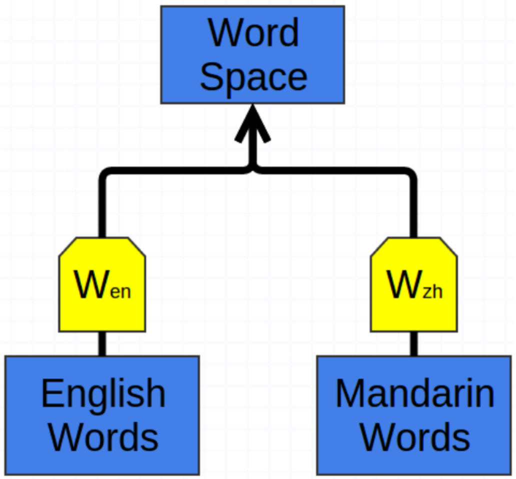
[1] Reference: http://ai.stanford.edu/~wzou/emnlp2013_ZouSocherCerManning.pdf

Bilingual Word Embedding (I)

- Train W_{en} and W_{zh} simultaneously
  - Impose the following: words that we know are close translations should be close together
  - Example:

$$W_{en}(\text{‘university’}) = \qquad (0.3, -0.1, 2.0, 1.1, -1.5, \ldots)$$

$$W_{zh}(\text{‘ 大学 ’}) = \qquad (0.2, -0.1, 2.2, 1.0, -1.4, \ldots)$$



[1] Pronounciation of 大学: Dàxué

[2] Figure: http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/

Bilingual Word Embedding (II)

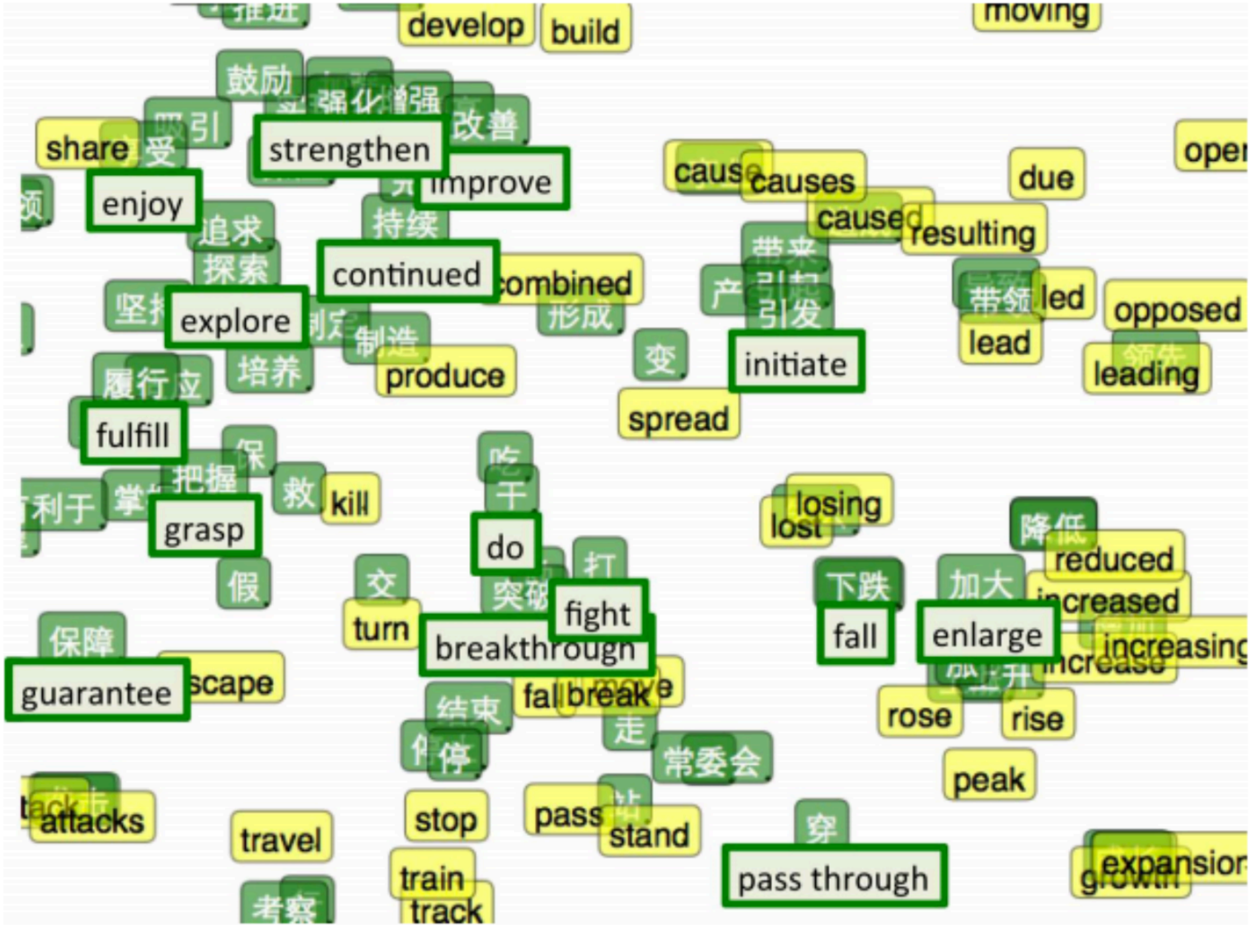
- After training W_{en} and W_{zh} we observe:
  - Words that we didn't know were translations end up close together
  - Example:
    - We did not know 商业 and business are translations. Still we get:

$$W_{en}(\text{'business'}) = \quad (0.7, -0.4, 1.0, 1.8, -0.8, \dots)$$

$$W_{zh}(\text{'商业'}) = \quad (0.8, -0.3, 0.9, 2.0, -0.9, \dots)$$

[1] Pronounciation of 商业: Shāngyè

Bilingual Word Embedding (III)



t-SNE visualization of the bilingual word embedding. Green is Chinese, Yellow is English.

[1] Reference: http://ai.stanford.edu/~wzou/emnlp2013_ZouSocherCerManning.pdf

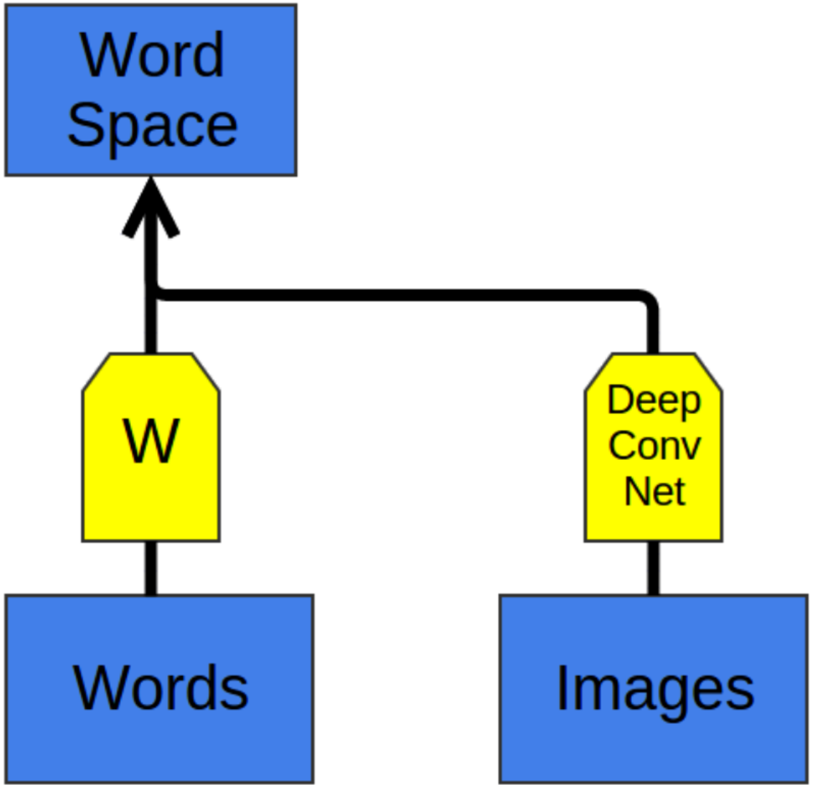
General Shared Embeddings

- We can also embed very different kinds of data into the same space

- Example:
  - Images and words
    - Map the image of an object near the object word vector
    - Map the image of a dog near the dog word vector

General Shared Embeddings

- Essentially the output of the image classifier is not a score vector but a vector in the range(W)

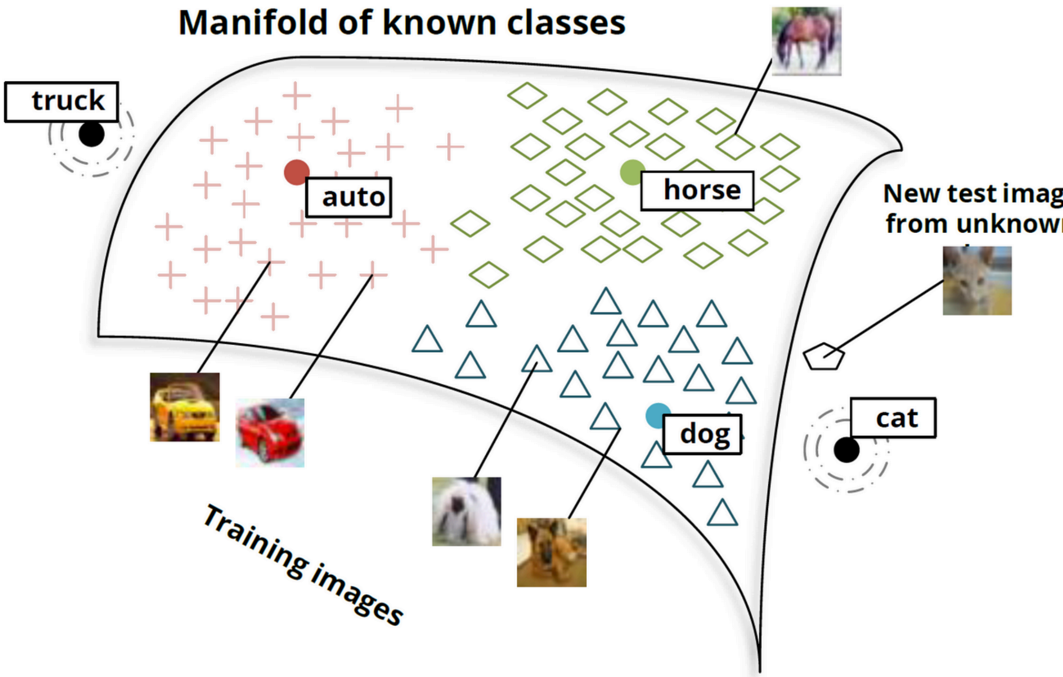


[1] Figure: http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/

General Shared Embeddings

- When we test the model on new classes of images
  - Note: new means *not seen in training*

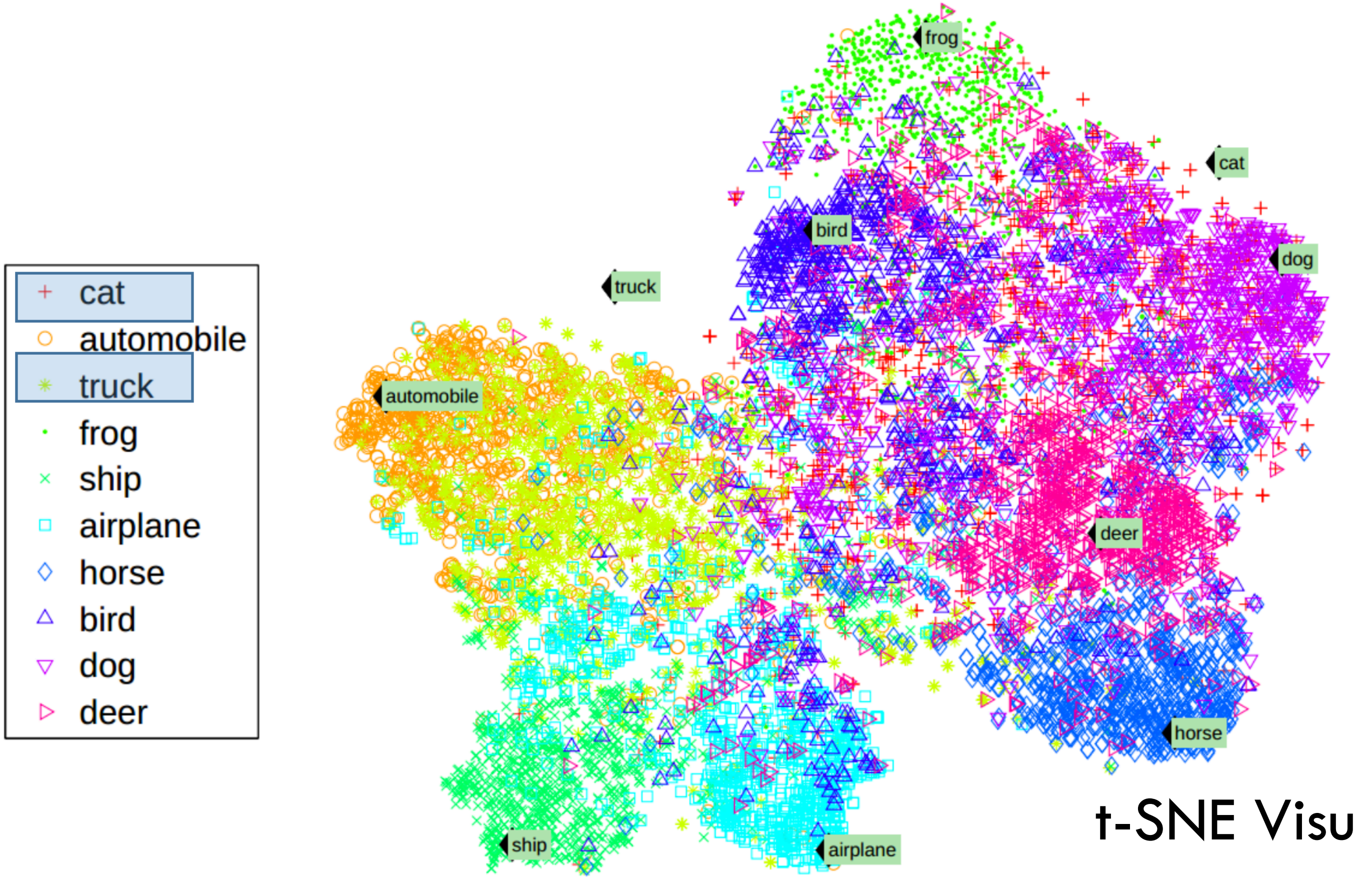
- For example, we didn't have images of cats



[1] Figure: http://nlp.stanford.edu/~socherr/SocherGanjooManningNg_NIPS2013.pdf

General Shared Embeddings

- *Images of cats are mapped to regions where dog vectors are!*



t-SNE Visualization

[1] Figure: http://nlp.stanford.edu/~socherr/SocherGanjooManningNg_NIPS2013.pdf

Questions?

Word2Vec In Detail

Word2Vector

- A technique proposed by Google in 2013
- Is a predictive method rather than a count based method

- Objective: Vector representations of words that capture their co-occurence statistics

Word2Vector: Two Versions

- Continuous Bag of Words and Skip-Gram

- Lets go through the skip-gram model in some detail now

W2V: The Skip-Gram Version

- This is a very simple neural network model to learn W
- We will train a single hidden layer NN to perform an auxiliary task
- The goal will be to just learn the weights of the network
  - This will give us W

[1] Reference: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

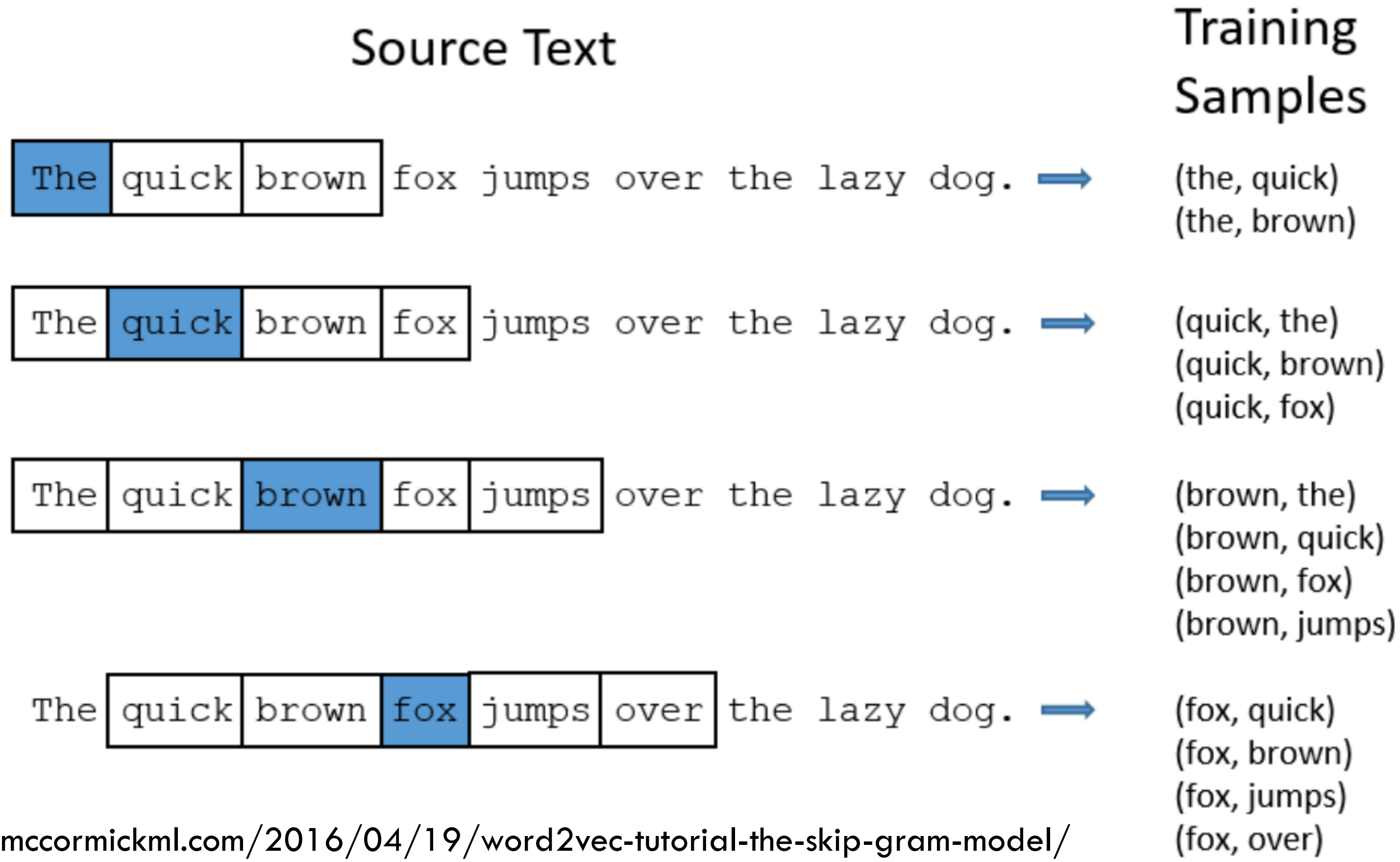
W2V: The Auxiliary Task

- Task:
  - Pick a word in the middle of a sentence
  - Pick one of the nearby words at random
  - Make network learn probability of every word in our vocab of being this nearby word
- Input: a word pair (one hot encoded)
- Output: normalized scores (of length: vocab size)

- Meaning of ‘nearby’:
  - Essentially defined using a window size
  - Example: Window size 2 means 2 words to left and 2 to right of the input word are nearby

W2V: The Auxiliary Task

- Feed word pairs
- Example: “The quick brown fox jumps over the lazy dog”



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)

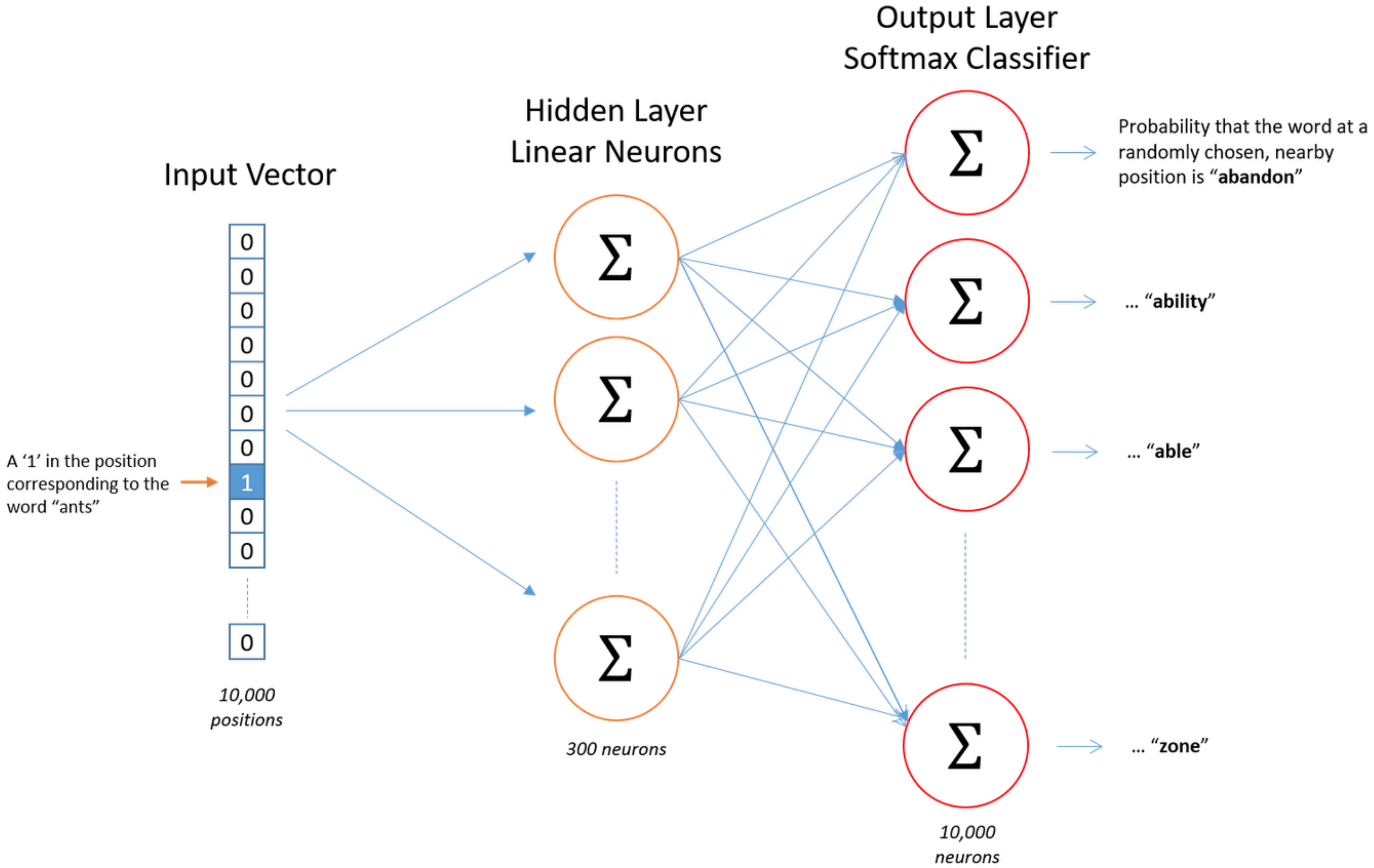
[1] Figure: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

W2V: The Network

- Say we have 10000 words in our vocab
- Then the input word is 10000 dimensional vector
  - Example: Cat word will have 'Cat' coordinate 1, everything else 0
- The true label (word) is also 10000 dimensional vector
- Network outputs 10000 scores which pass through softmax
  - Each coordinate is the probability that a particular word is the randomly selected nearby word

W2V: The Network

- Notice: No nonlinearity in the hidden layer!



[1] Figure: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

W2V: The Objective

- The objective is to maximize the normalized score (recall normalization by softmax operation) of the correct context word

- Say our training data is made with T words, each having a context window size 2
  - That is, each word is associated with 4 other words
  - Total training data is $4T$

- The objective is $\frac{1}{T}\sum_{t=1}^{T}\sum_{j\in\{-2,-1,1,2\}}\log p(w_{t+j}|w_t)$

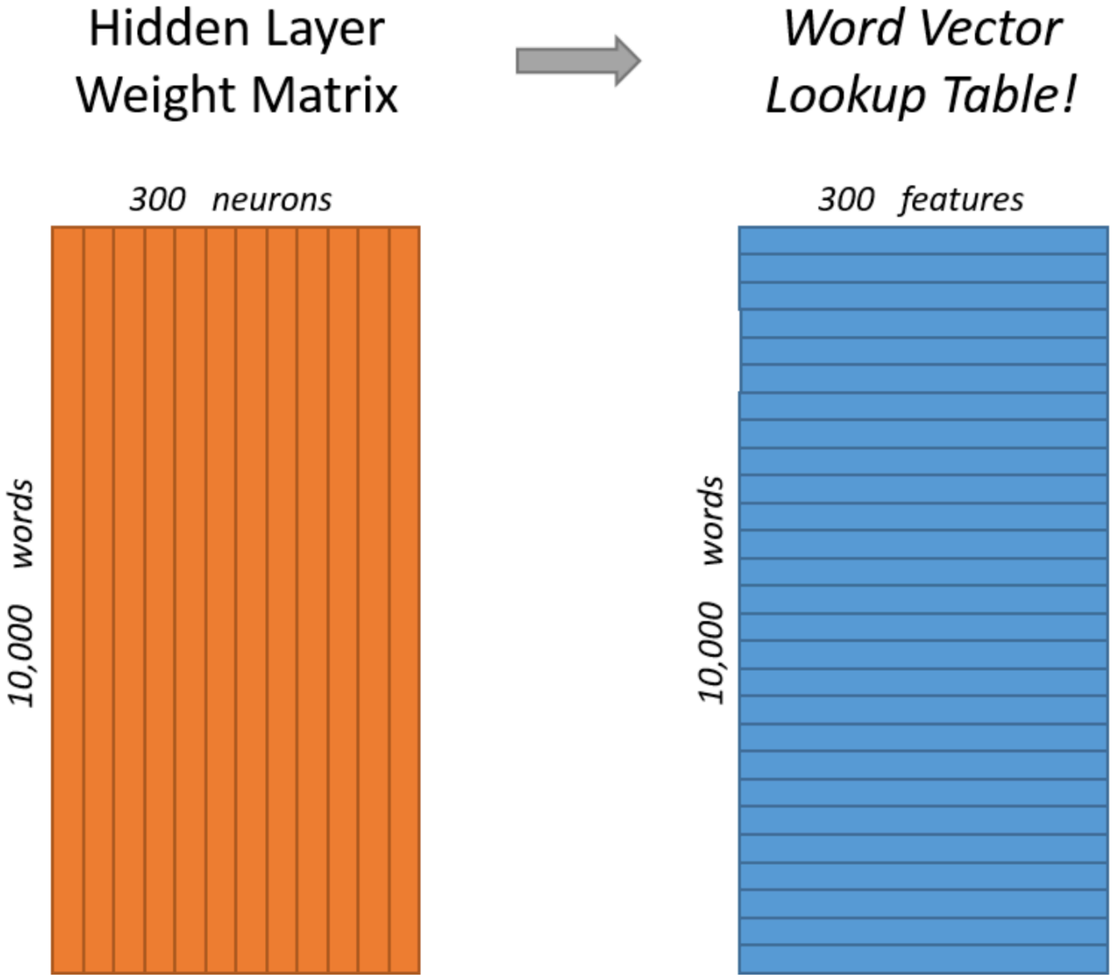
W2V: The Hidden Layer

- Is represented by a weight matrix W_0
  - Lets represent it by its transpose (just for convenience)
    - $h^T \ = \ x^T W_0^T = x^T W$ for each example
  - Number of rows of W is 10000
  - Number of columns of W is 300

- Then the rows of W are our word vectors!

W2V: The Hidden Layer

- Our real goal was just to learn the hidden layer weights



[1] Figure: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

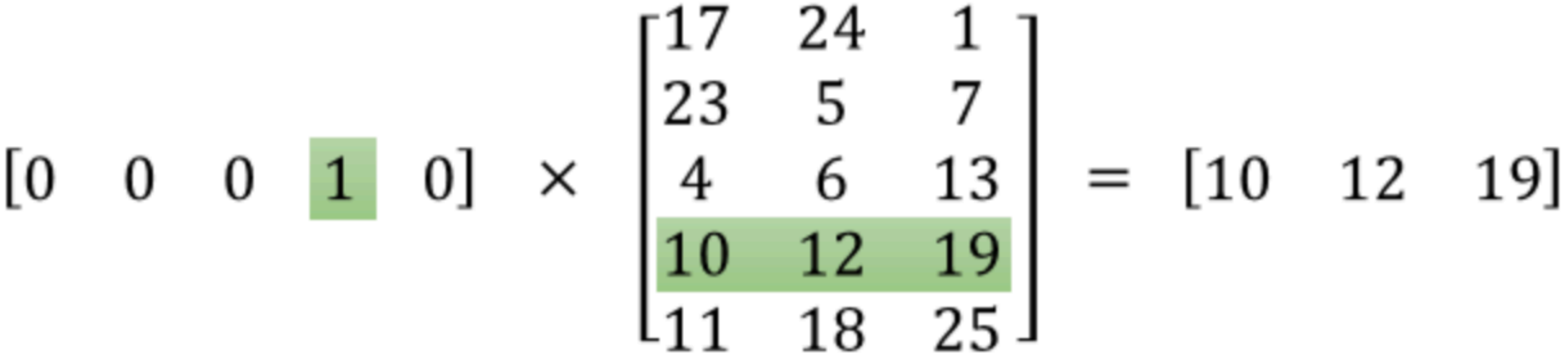
W2V: The Lookup

- Say word ‘Cat’ has coordinate c for some $c \in \{1, \ldots, 10{,}000\}$
- If we multiply the 1*10000 dim one hot vector for the word ‘Cat’ with W
- It will just select the c^{th} row of W
  - The output of the hidden layer is the word vector!

[1] Figure: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

W2V: The Lookup

- Say word ‘Cat’ has coordinate c for some $c \in \{1, \ldots, 10{,}000\}$
- If we multiply the 1*10000 dim one hot vector for the word ‘Cat’ with W
- It will just select the c^{th} row of W
  - The output of the hidden layer is the word vector!
- Example visualization

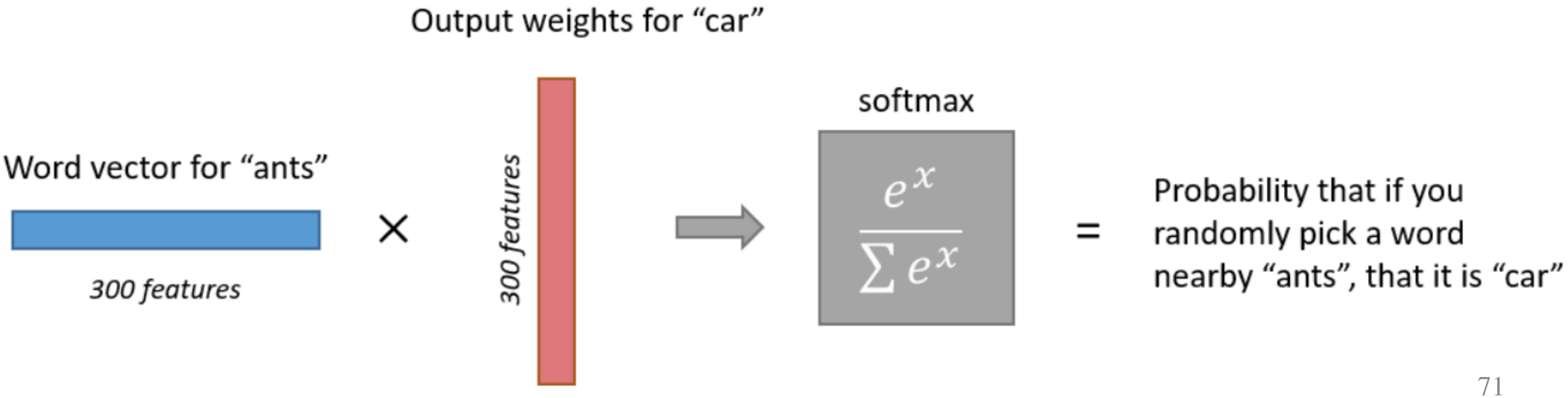


$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = \begin{bmatrix} 10 & 12 & 19 \end{bmatrix}$$

[1] Figure: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

W2V: Auxiliary Task Again

- Output of the network is a bunch of normalized scores (i.e., probabilities)
  - Denote the probability that the this word is a nearby word
- Example:
  - Pick the word vector for ‘ants’
  - Pic the output neuron for word ’car’



[1]Figure: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

W2V: Intuition for Vectors

- If two different words have similar “contexts”
  - Words that are likely to appear around them
  - Then the output probability vector should be similar
- For output vector to be similar
  - The word vector (weights of hidden layer) should be similar
  - Since the inputs are 1-hamming distance apart always

W2V: Intuition for Vectors

- Word2Vec is capturing nothing but the co-occurence statistics!

- Example:
  - Words like ‘university’ and ‘masters’ would have similar contexts, hence similar word vectors

- This will also handle stemming!
  - Example: words like ‘car’ and ‘cars’ will have similar vectors because contexts would be similar

W2V: Practice

- The network is relatively large
  - Two weight matrices
    - 300*10000 parameters each

- Need a lot of data to train
- And engineering tricks are needed to deal with data

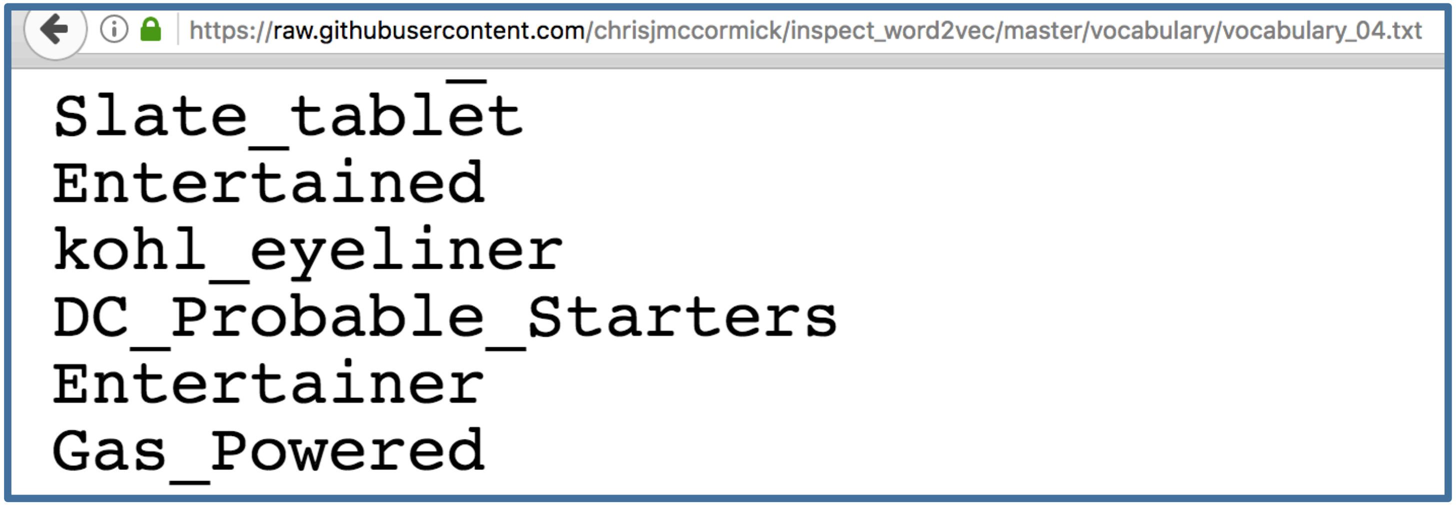
[1] Reference: http://arxiv.org/pdf/1310.4546.pdf

W2V: Engineering Tricks

- Subsample frequent words
  - Example: Too many pairs like (‘the’,…). So delete them proportional to how frequent they are
- Treat common phrases as single ‘words’
- Optimization trick: negative sampling
  - Only update the weights of neurons corresponding to a few (5-20) non-nearby words
  - These few are sampled inversely proportional to their frequency

W2V: From Google

- 100 Billion words from the Google News Dataset
- Vocab size totals about 3 million!



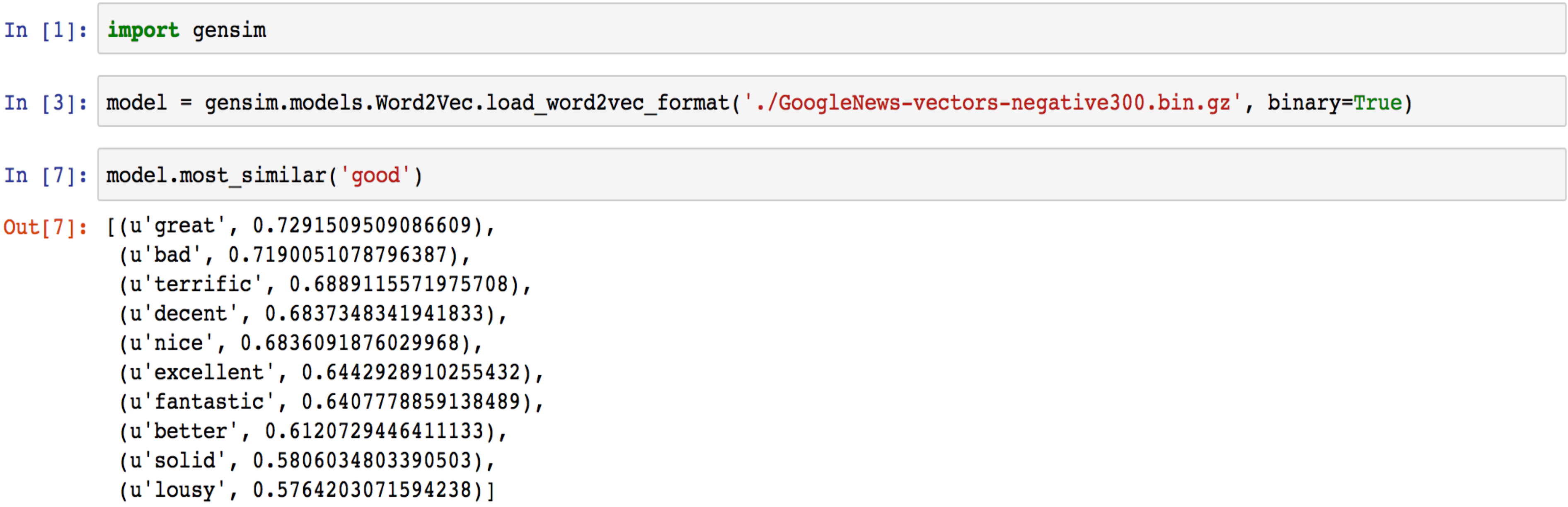
https://raw.githubusercontent.com/chrisjmccormick/inspect_word2vec/master/vocabulary/vocabulary_04.txt

```
Slate_tablet
Entertained
kohl_eyeliner
DC_Probable_Starters
Entertainer
Gas_Powered
```

[1] Reference: https://github.com/chrisjmccormick/inspect_word2vec/tree/master/vocabulary
[2] word2Vec file: https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit

W2V: From Google

- 100 Billion words from the Google News Dataset
- Vocab size totals about 3 million!



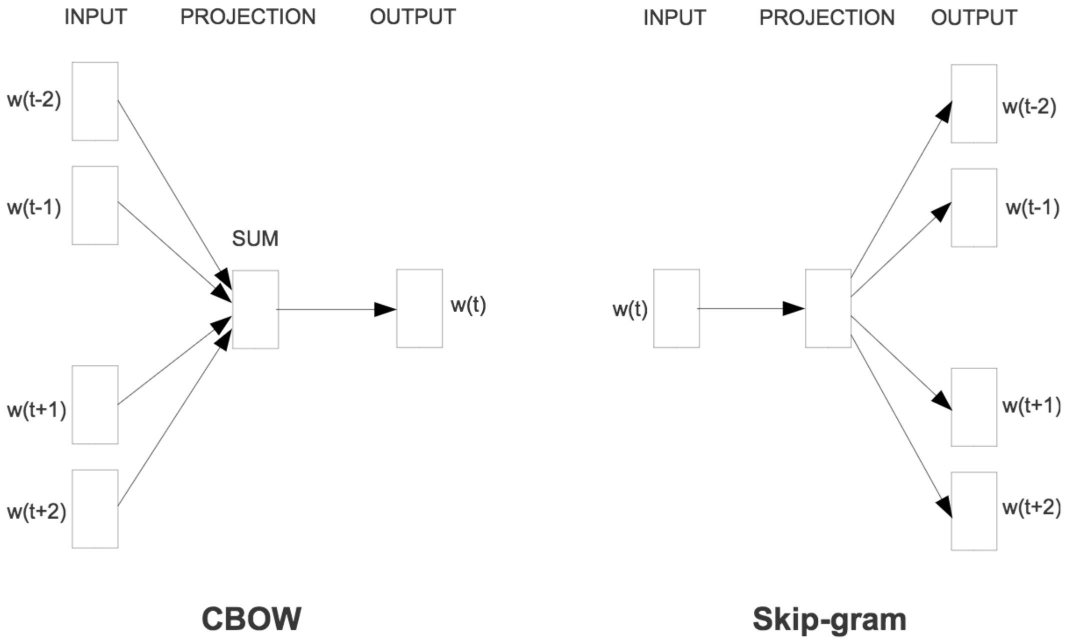
```
In [1]: import gensim

In [3]: model = gensim.models.Word2Vec.load_word2vec_format('./GoogleNews-vectors-negative300.bin.gz', binary=True)

In [7]: model.most_similar('good')

Out[7]: [(u'great', 0.7291509509086609),
          (u'bad', 0.7190051078796387),
          (u'terrific', 0.6889115571975708),
          (u'decent', 0.6837348341941833),
          (u'nice', 0.6836091876029968),
          (u'excellent', 0.6442928910255432),
          (u'fantastic', 0.6407778859138489),
          (u'better', 0.6120729446411133),
          (u'solid', 0.5806034803390503),
          (u'lousy', 0.5764203071594238)]
```

[1] Reference: https://github.com/chrisjmccormick/inspect_word2vec/tree/master/vocabulary

[2] word2Vec file: https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit

W2V: The CBOW Version

- Continuous Bag of Words (CBOW) version of Word2Vec
  - Essentially, a slightly different prediction task



- There is another popular embedding called GLoVe

[1]Figure: https://arxiv.org/pdf/1301.3781.pdf

[2]Reference for GLoVe: http://nlp.stanford.edu/projects/glove/

Word2Vec Example Code

- For an implementation in Python see
  - See https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/tutorials/word2vec/word2vec_basic.py

- Many other pretrained embeddings are also available
  - See https://github.com/3Top/word2vec-api

NLP Ecosystem in Python

- There are many tools to choose from
  - Gensim, NLTK, SpaCy, TextBlob, Pattern

- Also, there are many traditional NLP tasks and techniques that may be helpful to know about:
  - These include tokenizing, stop words, stemming, Parts-of-speech tagging, chunking and chinking, Named Entity Recognition, lemmatizing and knowing the wordnet ecosystem among others.

Questions?

Summary

- Text processing is very useful in multiple applications
- We saw some models that did not need to understand word meanings
  - Naïve bayes, Markov assumption based, CRF, LDA
- The notion of embedding words (or characters or phrases …) is useful and such embeddings can be learned
- We saw how Word2Vec embeddings were created

Appendix

LDA: More Intuition (different notation)

Question: given a text document, what topics is it about?

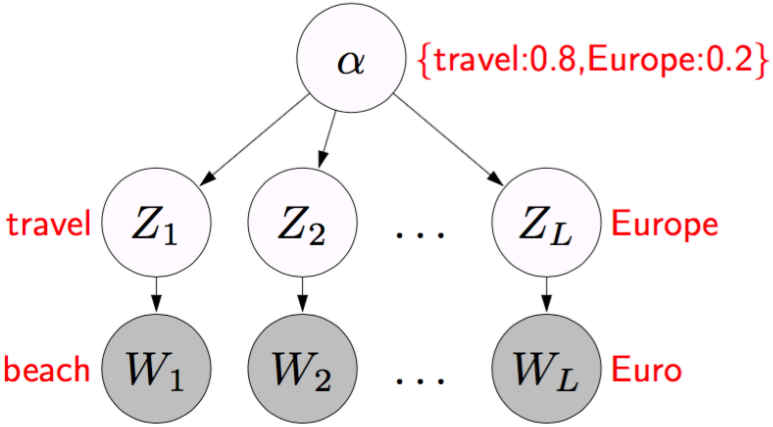
Probabilistic program: latent Dirichlet allocation

Generate a distribution over topics $\alpha \in \mathbb{R}^K$

For each position $i = 1, \ldots, L$:

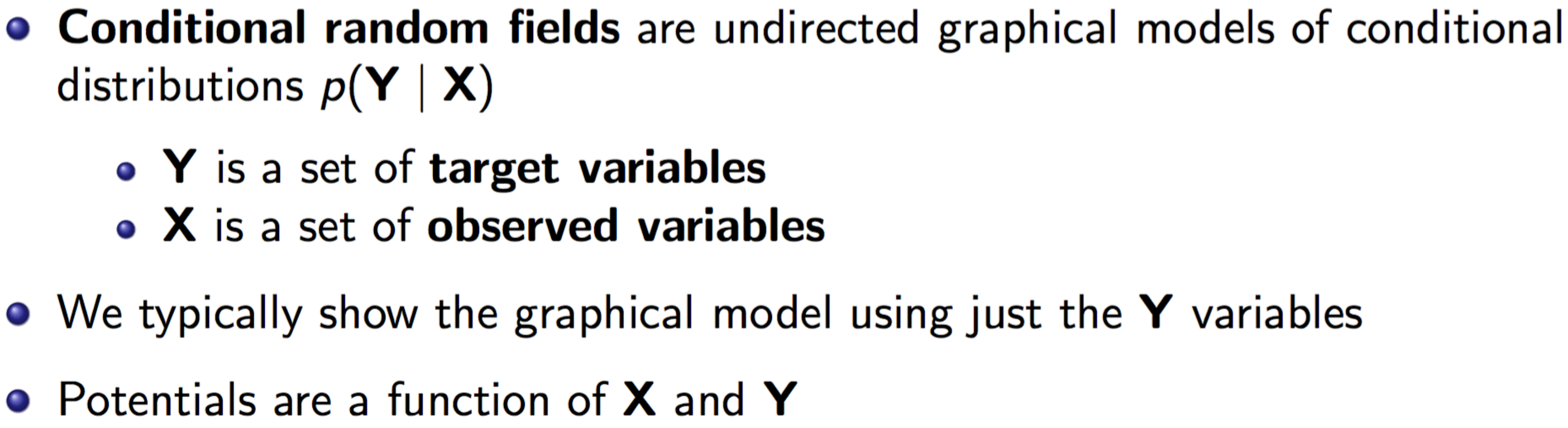
    Generate a topic $Z_i \sim p(Z_i \mid \alpha)$

    Generate a word $W_i \sim p(W_i \mid Z_i)$



[1] Reference: Percy Liang, CS221 (2015)

Conditional Random Field based Classifier



- **Conditional random fields** are undirected graphical models of conditional distributions $p(\mathbf{Y} \mid \mathbf{X})$
  - $\mathbf{Y}$ is a set of **target variables**
  - $\mathbf{X}$ is a set of **observed variables**
- We typically show the graphical model using just the $\mathbf{Y}$ variables
- Potentials are a function of $\mathbf{X}$ and $\mathbf{Y}$

[1] Reference: David Sontag (2013)

Conditional Random Field based Classifier

- A CRF is a Markov network on variables $\mathbf{X} \cup \mathbf{Y}$, which specifies the conditional distribution

$$P(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{c \in C} \phi_c(\mathbf{x}_c, \mathbf{y}_c)$$

with partition function

$$Z(\mathbf{x}) = \sum_{\hat{\mathbf{y}}} \prod_{c \in C} \phi_c(\mathbf{x}_c, \hat{\mathbf{y}}_c).$$

- As before, two variables in the graph are connected with an undirected edge if they appear together in the scope of some factor
- The only difference with a standard Markov network is the normalization term – before marginalized over $\mathbf{X}$ and $\mathbf{Y}$, now only over $\mathbf{Y}$

[1] Reference: David Sontag (2013)

CRF for NLP: Log-linear Terms

- Factors may depend on a large number of variables
- We typically parameterize each factor as a log-linear function,

$$\phi_c(\mathbf{x}_c, \mathbf{y}_c) = \exp\{\mathbf{w} \cdot \mathbf{f}_c(\mathbf{x}_c, \mathbf{y}_c)\}$$

- $\mathbf{f}_c(\mathbf{x}_c, \mathbf{y}_c)$ is a feature vector
- $\mathbf{w}$ is a weight vector which is typically learned – we will discuss this extensively in later lectures

[1] Reference: David Sontag (2013)

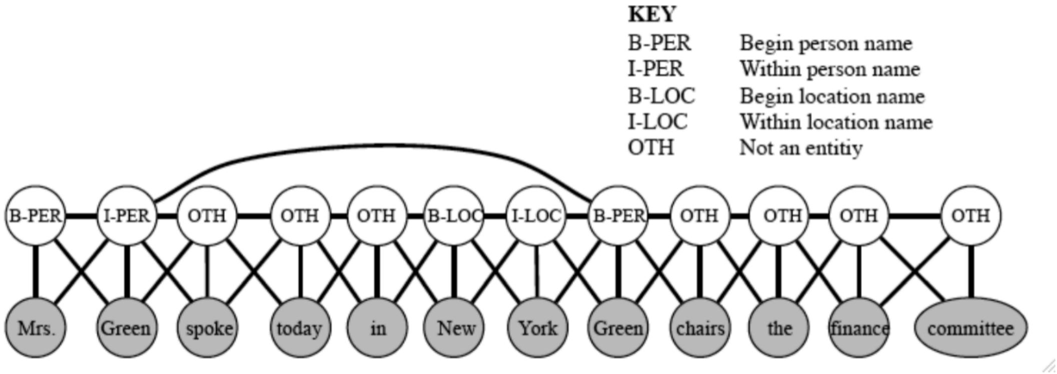
CRF for NLP: The Task

- Given a sentence, determine the people and organizations involved and the relevant locations:
  "Mrs. Green spoke today in New York. Green chairs the finance committee."
- Entities sometimes span multiple words. Entity of a word not obvious without considering its *context*
- CRF has one variable X_i for each word, and Y_i encodes the possible labels of that word
- The labels are, for example, "B-person, I-person, B-location, I-location, B-organization, I-organization"
  - Having beginning (B) and within (I) allows the model to segment adjacent entities

[1] Reference: David Sontag (2013)

CRF for NLP: The Task

The graphical model looks like (called a *skip-chain CRF*):



KEY	
B-PER	Begin person name
I-PER	Within person name
B-LOC	Begin location name
I-LOC	Within location name
OTH	Not an entitiy

There are three types of potentials:

- $\phi^1(Y_t, Y_{t+1})$ represents dependencies between neighboring target variables [analogous to transition distribution in a HMM]
- $\phi^2(Y_t, Y_{t'})$ for all pairs t, t' such that $x_t = x_{t'}$, because if a word appears twice, it is likely to be the same entity
- $\phi^3(Y_t, X_1, \cdots, X_T)$ for dependencies between an entity and the word sequence [e.g., may have features taking into consideration capitalization]

Notice that the graph structure changes depending on the sentence!

[1] Reference: David Sontag (2013)