Embedding Layers and Word Embeddings

Elliott Ash

Bocconi, 2018

Word Embeddings

- Word embeddings:
  - a hot topic in NLP since arrival of Word2Vec in 2013.
  - refers to a class of statistical models that **represent words or phrases as points in a vector space**.
- The key idea is to represent the meaning of words by the neighboring words – their **contexts**.
- Many people use "word embeddings" and "word2vec" interchangeably, although word2vec technically refers to a particular implementation of a word embedding model.
  - the other well-known implementation is gloVe, which has similar performance/applications

What is an Embedding?

- An "embedding" is a tool in machine learning that is quite a bit different than anything we have in econometrics.
  - It is a vector representation of a categorical variable.
  - where the spatial location of the vector encodes predictive information about the category.
- U.S. states:
  - instead of including a fifty-dimensional categorical variable, include two-dimensional latitude and longitude
  - or initialize each state to a random two-dimensional vector, and let the model decide where to move the states to improve prediction on your task (e.g. responses to trade shocks).

An embedding layer is just matrix multiplication

- An embedding layer can be represented as

$$\underbrace{x}_{n\times 1} = \underbrace{\Omega'}_{n\times m}\underbrace{w}_{m\times 1}$$

  - w, a categorical variable, a one-hot-encoded vector that is all zeros, with a single item equaling one.
    - The input to the embedding layer.
  - x, a dense representation of the variable.
    - The output of the embedding layer.
  - An embedding matrix Ω.
    - the model learns the weights of this matrix.

The Embedding Matrix Ω

- The model learns the weights of the embedding matrix in the same way that it would learn any model parameters.
- The rows of the matrix correspond to vectors for the n categories.
  - These are the "word vectors" that people talk about when they mention word embeddings or Word2Vec.

Embedding Layers versus Dense Layers

- A fully-connected dense layer, with sparse data set as input and linear activation, would emulate an embedding layer.
- Why use an embedding layer rather than a dense layer?
  - the main reason is that embedding layers are much faster for this purpose.
  - a related reason is that batch updating with regularization and dropout do not work well on sparse data.

Categorical Embeddings in Keras

- Prep categorical data (in this case, judge identity)

```
# make integer IDs for judges
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
J = encoder.fit_transform(df1['author_id'])
num_judges = max(J)+1
```

Categorical Embeddings: Setup Model

```
from keras.models import Sequential
from keras.layers import Dense, Embedding, Flatten

model = Sequential()
model.add(Embedding(num_judges, # number of categories
                    2, # dimensions of embedding
                    input_length=1))
model.add(Flatten()) # needed after Embedding
model.add(Dense(2))
model.add(Dense(1,activation='sigmoid'))
model.compile(optimizer='adam',loss='binary_crossentropy')
model.summary()
```

Categorical Embeddings: Visualize/Train

```
import ggplot as gg
judge_cites = dict(Y.groupby(J).mean())
df2 = pd.DataFrame(J,columns=['judge']).drop_duplicates()
df2.sort_values('judge',inplace=True)
df2['cites'] = df2['judge'].apply(lambda x: judge_cites[x])

for i in range(5):
    if i > 0:
        model.fit(J,Y,epochs=1, validation_split=.2)

    judge_vectors = model.layers[0].get_weights()[0]
    df2['x'] = judge_vectors[:,0]
    df2['y'] = judge_vectors[:,1]
    chart = gg.ggplot(df2, gg.aes(x='x', y='y', color='cites')) \
                    + gg.geom_point(size=10, alpha=.8)
    chart.show()
```

Word Embeddings

- With word embeddings, embedding layer is a dictionary mapping word indexes to dense vectors.
- We have a set of documents, which are lists of word indexes $\{w_1, w_2, ..., w_L\}$
  - normalize all documents to the same length – shorter documents can be padded with a null token.
  - this requirement can be relaxed with recurrent neural networks
- The embedding layer replaces the list of word indexes with

$$\begin{bmatrix} x_1 & \dots & x_L \end{bmatrix}$$

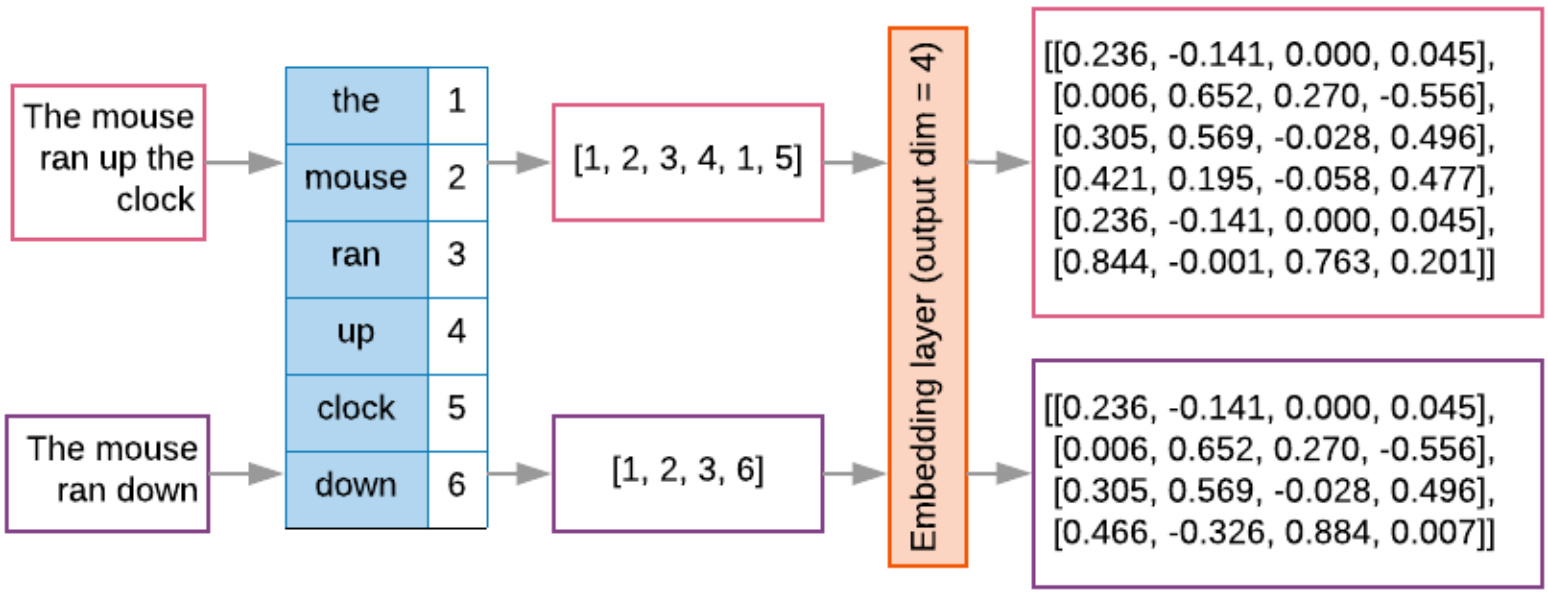
where the words have been replaced with the vectors

$$\underbrace{x}_{n\times 1} = \underbrace{\Omega'}_{n\times m}\underbrace{w}_{m\times 1}$$

as columns.

- This matrix is then flattened into a $L * n$ vector for input to the next layer

Illustration



Word Embeddings: Data Prep

```
# convert documents to sequences of word indexes
from keras.preprocessing.text import Tokenizer
num_words = 200
tokenizer = Tokenizer(num_words=num_words)
tokenizer.fit_on_texts(df1['snippet'])
sequences = tokenizer.texts_to_sequences(df1['snippet'])

# represent data as numrows x maxlen matrix
from keras.preprocessing.sequence import pad_sequences
maxlen = max([len(sent) for sent in sequences])
X = pad_sequences(sequences, maxlen=maxlen)
```

Word Embeddings: Model Setup

```
model = Sequential()
model.add(Embedding(num_words,
                    2,
                    input_length=maxlen)) # sequence length
model.add(Flatten()) # 86*2 = 172 dims
model.add(Dense(2))
model.add(Dense(1))
model.compile(optimizer='adam',loss='binary_crossentropy')
dot = model_to_dot(model,show_shapes=True,show_layer_names=False)
SVG(dot.create(prog='dot', format='svg'))
```

Word Embeddings: Visualize/Train Vectors

```
# show the vectors
df3 = pd.DataFrame(list(tokenizer.word_index.items()),
                                columns=['word', 'word_index'])
df3 = df3.sort_values('word_index')[:num_words]

for i in range(5):
    if i > 0:
        model.fit(X,Y,epochs=1, validation_split=.2)

    word_vectors = model.layers[0].get_weights()[0]
    df3['x'] = word_vectors[:,0]
    df3['y'] = word_vectors[:,1]
    chart = gg.ggplot(df3, gg.aes(x='x',y='y',label='word'))\
            + gg.geom_text(size=10,alpha=.8, label='word')
    chart.show()
```

Word Similarity

```
from scipy.spatial.distance import cosine

vec_defendants = word_vectors[tokenizer.word_index['defendants']-1
vec_convicted = word_vectors[tokenizer.word_index['convicted']-1]
vec_against = word_vectors[tokenizer.word_index['against']-1]

print(1-cosine(vec_defendants, vec_convicted))
print(1-cosine(vec_defendants, vec_against))
```

Word Embeddings and Word2Vec

- Word2Vec , GloVe, and other popular embeddings vectors are trained the same way as the word embeddings we just made for citation counts.
  - rather than predicting some metadata (such as citations) they predict the co-occurence of neighboring words.

Why word vectors?

- Once words are represented as vectors, we can use linear algebra to understand the relationships between words:
  - Words that are geometrically close to each other are similar: e.g. "student" and "pupil."
  - More intriguingly, word2vec algebra can depict conceptual, analogical relationships between words.
    - Consider the analogy: **man is to king as woman is to _____**
    - With word2vec, we have

$$vec(king) - vec(man) + vec(woman) \approx vec(queen)$$

How are word embeddings different from topic models?

- Ben Schmidt:
  - Topic models reduce words to core meanings to understand documents more clearly.
  - Word embedding models ignore information about individual documents to better understand the relationships between words.

Word Function $\longleftrightarrow$ Word Neighbors

- "The meaning of a word is its use in the language"

  - Ludwig Wittgenstein, *Philosophical Investigation*, 1953

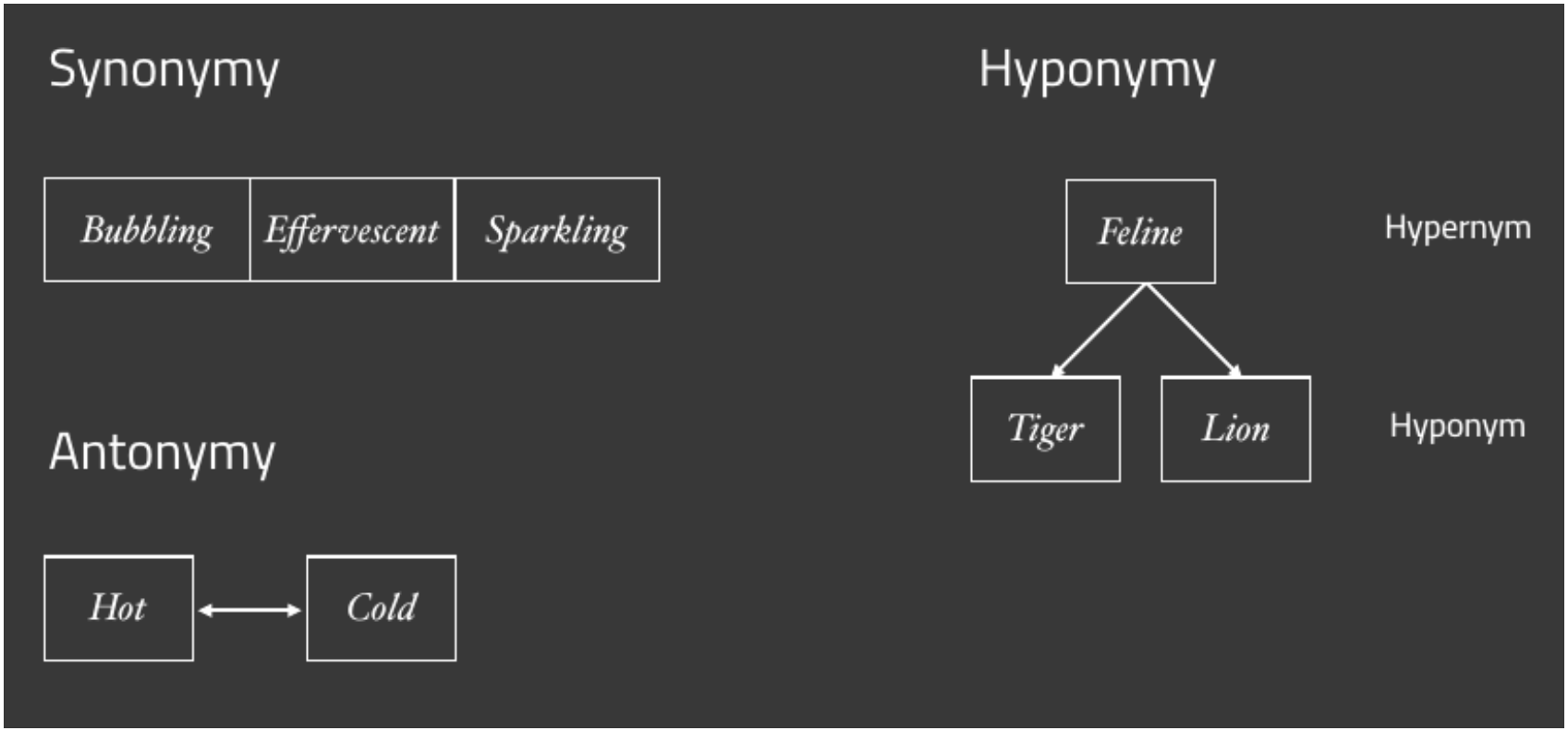
- "You shall know a word by the company it keeps"

  - J.R. Firth, Papers in Linguistics,1957

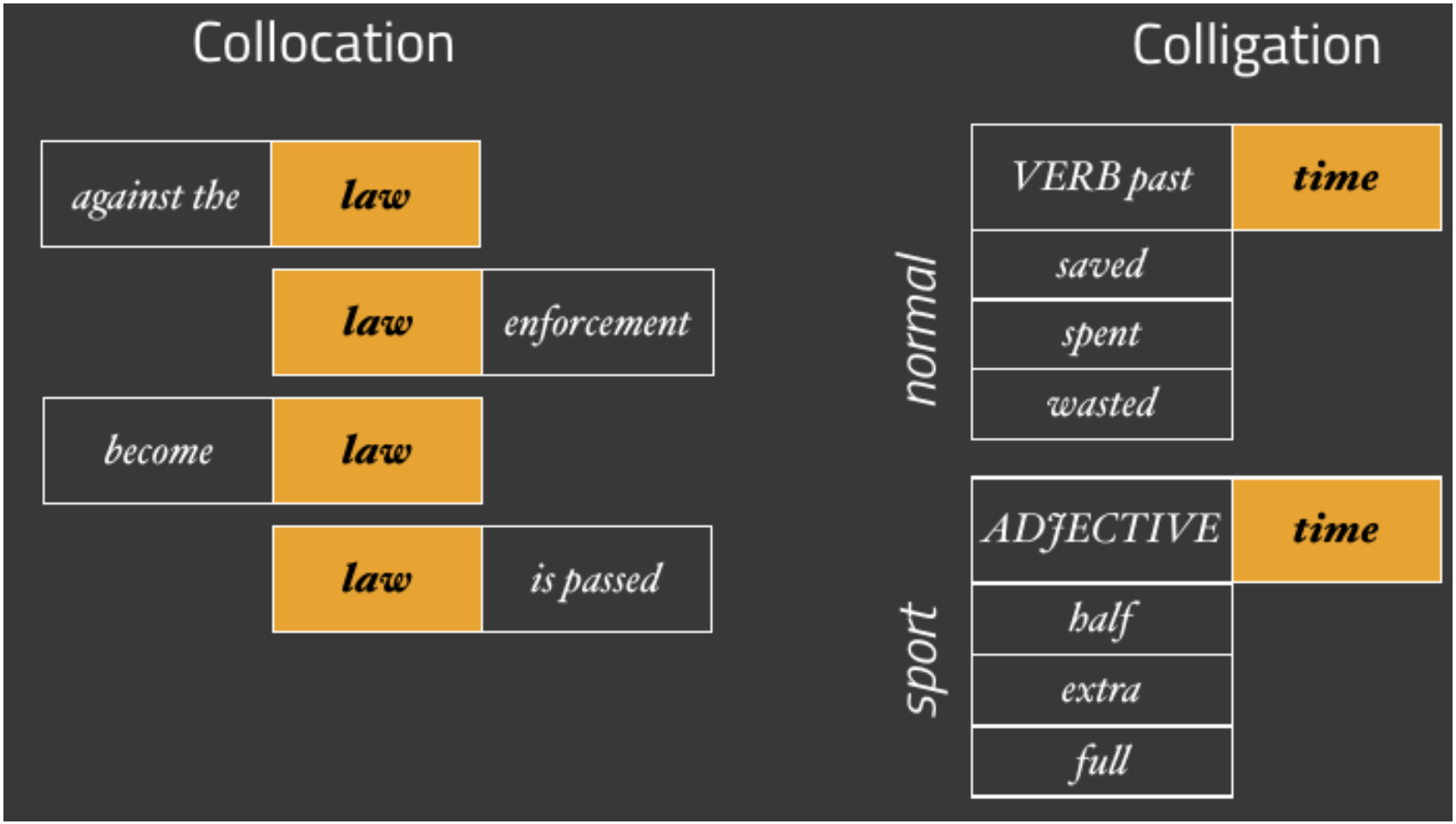
I've never seen this word before, but...

- He filled the **wampimuk**, passed it around and we all drunk some
- We found a little, hairy **wampimuk** sleeping behind the tree

Linguistic Relations



Collocational Relations

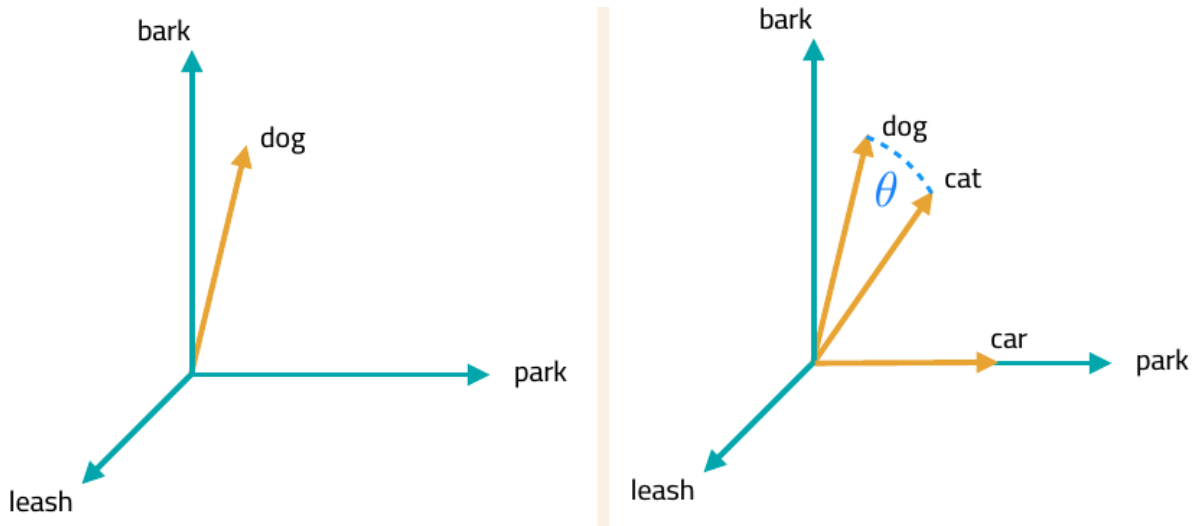


Similarity vs. Relatedness

- Semantic **similarity**: words sharing salient attributes / features
  - synonymy (car / automobile)
  - hypernymy (car / vehicle)
  - co-hyponymy (car / van / truck)
- Semantic **relatedness**: words semantically associated without necessarily being similar
  - function (car / drive)
  - meronymy (car / tire)
  - location (car / road)
  - attribute (car / fast)

(Budansky and Hirst, 2006)

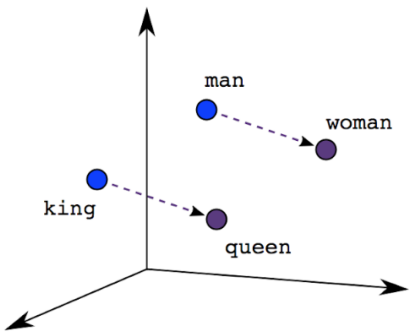
Words as Vectors



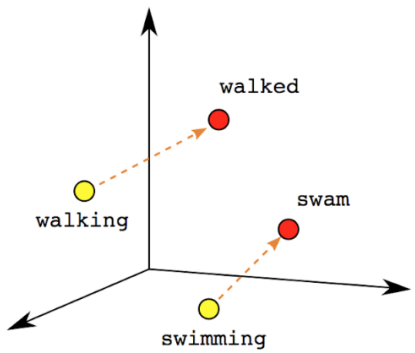
- Use cosine similarity as a measure of relatedness:

$$\cos\theta = \frac{v_1 \cdot v_2}{||v_1||||v_2||}$$

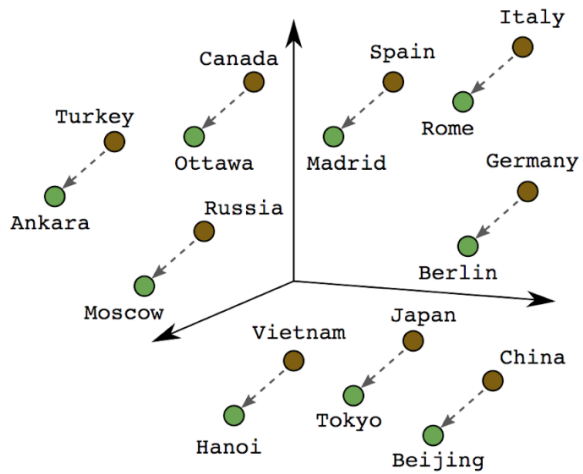
Vector Directions $\leftrightarrow$ Meaning



Male-Female



Verb Tense



Country-Capital

Evaluation of Word Embeddings

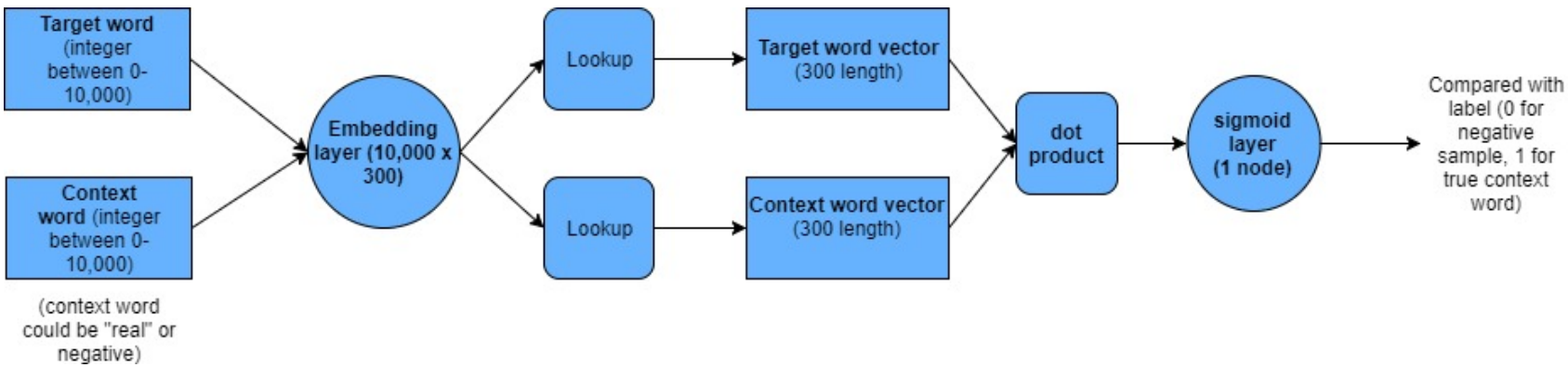
- Intrinsic:
  - evaluate word-pairs similarities $\rightarrow$ compare with similarity judgments given by humans
  - evaluate on analogy tasks ("Paris is to France as Tokyo is to ____")
- Extrinsic:
  - use the vectors in a downstream task (classification, translation, ...) and evaluate the final performance on the task

SGNS: Skip-gram with negative sampling

- When people mention "word2vec", they are usually talking about SGNS: "skip gram with negative sampling."
  - This is a particular word-embedding model with good performance on a range of analogy and prediction tasks.
- How does it learn the meaning of the word "fox":

  The quick brown **fox** jumps over the lazy dog
- Word2Vec reads in every example of the word "fox" and tries to predict what other words will be in the context window.
  - the prediction weights on these other words (after dimension reduction) are the word vectors

Word2Vec Schema



gensim implemention

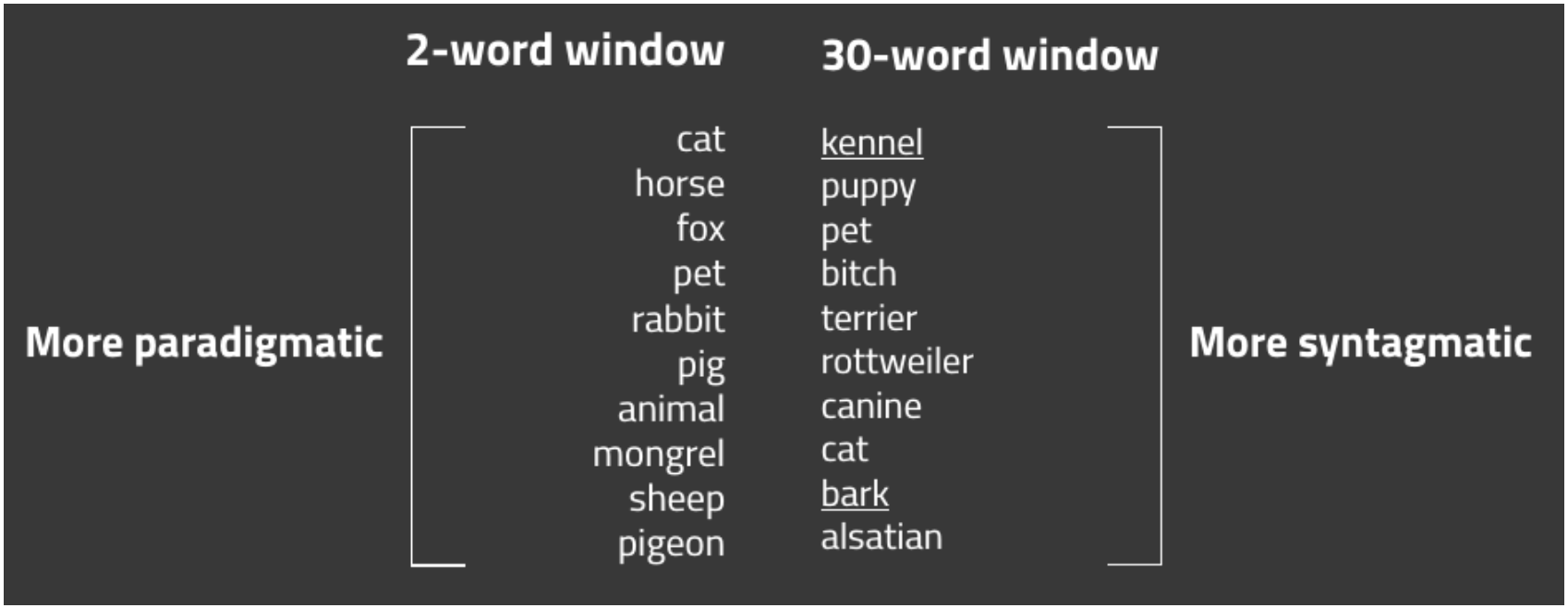
- gensim's implementation of word2vec has the same benefits as the LDA implementation:
  - intuitive, streaming, and fast/parallellized

```
from gensim.models import Word2Vec
w2v = Word2Vec(sentences, # list of tokenized sentences
               workers = 8, # Number of threads to run
               size=300,  # Word vector dimensionality
               min_count =  5, # Minimum word count
               window = 5, # Context window size
               sample = 1e-3  # Downsample for frequent words
               )
```

Using Word2Vec

```
w2v.save('w2v-vectors.pkl')

w2v.wv['judg'] # vector for "judge"

w2v.wv.similarity('judg','juri') # similarity

w2v.wv.most_similar('judg') # most similar words

# analogies: judge is to man as __ is to woman
w2v.wv.most_similar(positive=['judg','man'],
                    negative=['woman'])
```

Most similar words to dog, depending on window size



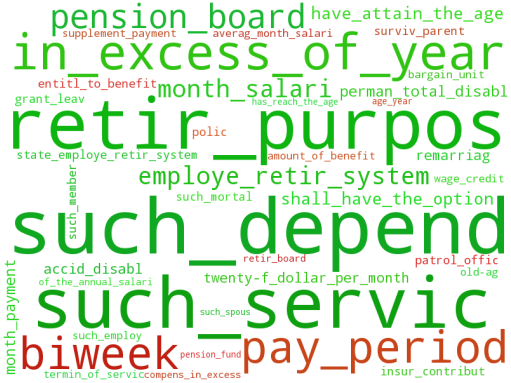
- Small windows pick up substitutable words; large windows pick up topics.

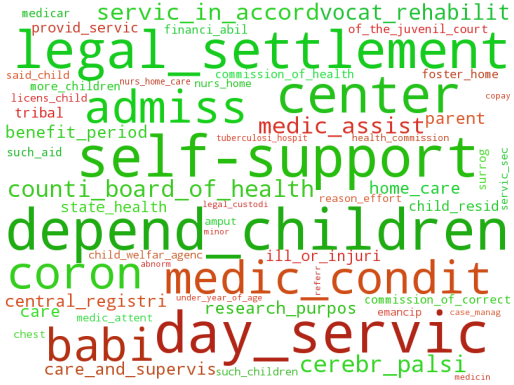
Word2Vec: K-means clustering

```
from sklearn.cluster import KMeans
kmw = KMeans(n_clusters=50)
kmw.fit(w2v.wv.vectors)
judge_clust = kmw.labels_[w2v.wv.vocab['judg'].index]
for i, cluster in enumerate(kmw.labels_):
    if cluster == judge_clust:
        print(w2v.wv.index2word[i])
```

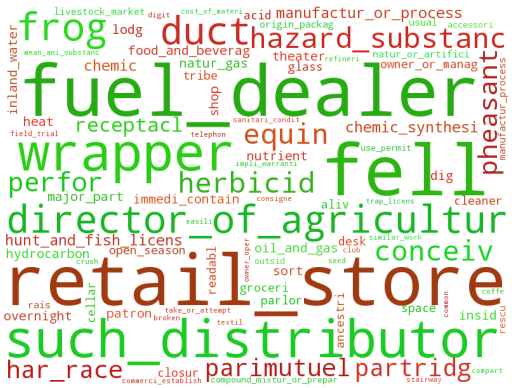
Clustered Phrases Affecting Tax Revenue (Ash 2017)

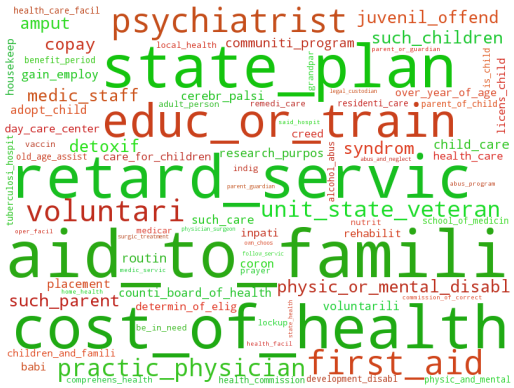
Income Tax (Pensions Topic and Health Care Topic)





Sales Tax (Retail Topic and Health Care Topic)





- Green words tend to increase revenues; red words tend to decrease revenues.

Can robots rewrite the tax code?

- Example statutory substitution recommended by estimates (Virginia Code § 54-377): "person, firm, or corporation" should be replaced with "such person, firm, or corporation"; "failure" should be replaced with "such failure"
  - Adding "such" improves clarity, is considered good legal writing (e.g. Osbeck 2012).
  - Predicted revenue impact: +$16.2 million.

Version 1:

(a) If any person, firm or corporation shall continue after the expiration of a license previously issued without obtaining a new license, person, firm or corporation shall, if failure to obtain a new license continues for one month, be subject to a penalty of ten percentum of the amount of the license tax ... If failure to obtain a new license be continued for a longer period than one month, person, firm or corporation shall be guilty or a misdemeanor. Such conviction shall not relieve any person, firm or corporation from the payment of any license tax and penalty imposed by this article.
(b) If any person, firm or corporation shall commence any business, employment or profession in the city for which a license tax is required under this article, without first obtaining such license, person, firm or corporation shall be guilty of a misdemeanor ... If such violation of law be continued for one month, person, firm or corporation shall be subject to a penalty of ten percentum of the license tax which was due...

Version 2:

(a) If any person, firm or corporation shall continue after the expiration of a license previously issued without obtaining a new license, such person, firm or corporation shall, if such failure to obtain a new license continues for one month, be subject to a penalty of ten percentum of the amount of the license tax ... If such failure to obtain a new license be continued for a longer period than one month, such person, firm or corporation shall be guilty or a misdemeanor. Such conviction shall not relieve any such person, firm or corporation from the payment of any license tax and penalty imposed by this article.
(b) If any person, firm or corporation shall commence any business, employment or profession in the city for which a license tax is required under this article, without first obtaining such license, such person, firm or corporation shall be guilty of a misdemeanor ... If such violation of law be continued for one month, such person, firm or corporation shall be subject to a penalty of ten percentum of the license tax which was due...

Pre-trained word embeddings

- For some NLP tasks, you might not need to train your own vectors.
  - or your corpus might be too small to do so.
- In these cases, you can use a pre-trained model, like spaCy:
  - one million vocabulary entries
  - 300-dimensional vectors
  - trained on the Common Crawl corpus

Using spaCy Pre-Trained Embeddings

```
import spacy
nlp = spacy.load('en_core_web_lg')
apple = nlp('apple')
apple.vector # vector for 'apple'

apple.similarity(apple)

orange = nlp('orange')
apple.similarity(orange)

it = spacy.load('it')
mela = it('mela')
arancia = it('arancia')
mela.similarity(arancia)

mela.similarity(apple)
```

Initialize a Model with Pre-Trained Embeddings

```
embed_dims = len(apple.vector)
embedding_matrix = np.zeros([num_words, embed_dims])
for word, i in tokenizer.word_index.items():
    if i > num_words:
        break
    embedding_vector = en(word).vector
    embedding_matrix[i-1] = embedding_vector

model = Sequential()
model.add(Embedding(num_words,
                    embed_dims,
                    weights=[embedding_matrix],
                    input_length=maxlen,
                    trainable=False)) # frozen layer
model.add(Flatten()) # 86*300 = 25800 dims
```

Visualizing Pre-Trained Embeddings

```
word_vectors = model.layers[0].get_weights()[0]
df3 = pd.DataFrame(list(tokenizer.word_index.items()),
                   columns=['word', 'word_index'])
df3 = df3.sort_values('word_index')[:num_words]

# use t-SNE to reduce to 2 dimensions
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2, verbose=1, perplexity=50, n_iter=300)
wv_tsne = tsne.fit_transform(word_vectors)
df3['x'] = wv_tsne[:,0]
df3['y'] = wv_tsne[:,1]

chart = gg.ggplot( df3, gg.aes(x='x', y='y', label='word') ) \
                  + gg.geom_text(size=10, alpha=.8, label='word')
chart.show()
```

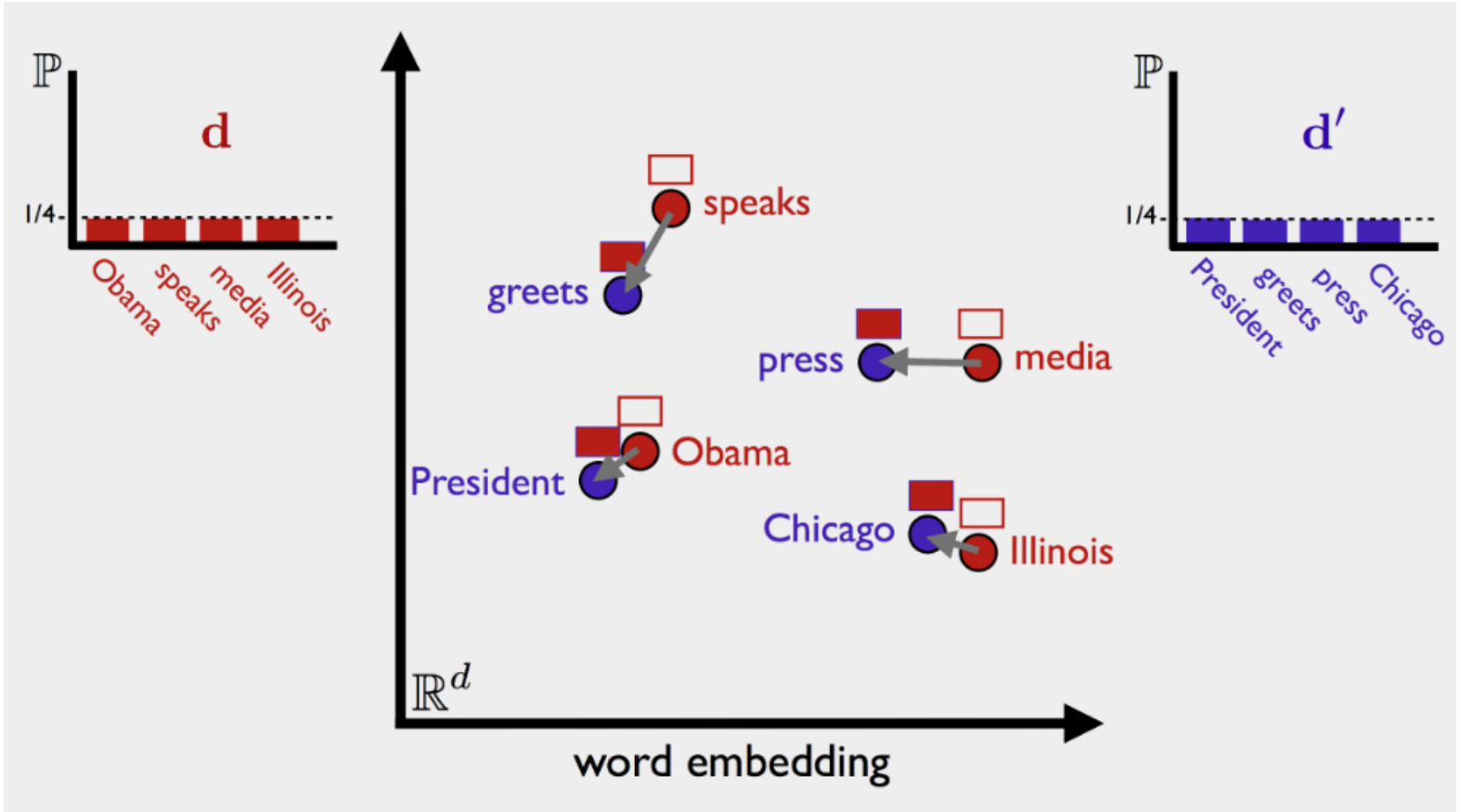
Tips for using pre-trained embeddings

- Split training in two steps:
  - in first run, train the model with the first layer (the pre-trained embeddings) frozen.
  - in second run, un-freeze the embedding layer for fine tuning.

Word Mover Distance

- Cosine distance treats synonyms as just as close as totally unrelated words.
- Word mover distance between two texts is given by:
  - total amount of "mass" needed to move words from one side into the other
  - multiplied by the distance the words need to move
  - Kusner, Sun, Kolkin, and Weinberger (ICML 2015)
- Requires measure of distance between words (word embeddings).

Illustration



- d (obama speaks media illinois) is orthogonal to d' (president greets press chicago):
  - cosine similarity is zero
  - Word mover distance sums the shortest distances between the words in the documents.

WMD In Python

```
import spacy
import wmd
nlp = spacy.load('en',
                 create_pipeline=wmd.WMD.create_spacy_pipeline)
doc1 = nlp("Politician speaks to the media in Illinois.")
doc2 = nlp("The president greets the press in Chicago.")
print(doc1.similarity(doc2))
```

- installing wmd might break your Python installation, so don't try this unless you need it.