Distributed Representation
Word Embeddings

Md Shad Akhtar
IIT Patna
Why do we need word representation?

• Many Machine Learning algorithms do not understand text data, they require input to be numeric. E.g. SVM, NN etc.
Different Representations

• Local Representation
  – One hot
    • Cat = [0,0,0,0,1,0,0,0,0,0]
    • Sparse
    • No semantics
    • Curse of Dimensionality

• Distributed Representation
  – Word embeddings
    • Cat = [2.4, 1.0, 3.1, 5.3]
    • Dense
    • Very good at capturing semantic relations.
    • Low Dimensionality
Word Embeddings: word2vec

• Word2vec is a tool which computes vector representations of words.
• Word meaning and relationships between words are encoded spatially.
• learns from input texts.
• Developed by Mikolov, Sutskever, Chen, Corrado and Dean in 2013 at Google Research
What is word2vec? (cont’d…)

• Word2vec is a two-layer neural net that processes text.
• Its input is a text corpus and its output is a set of vectors: feature vectors for words in that corpus.
• While Word2vec is not a deep neural network, it turns text into a numerical form that deep nets can understand.
Similar words are closer together

- spatial distance corresponds to word similarity
- words are close together ⇔ their "meanings" are similar
- notation: word $w \rightarrow \text{vec}[w]$ its point in space, as a position vector.
- e.g. $\text{vec}[\text{woman}] = (0.1, -1.3)$
word2vec

Input: one document

Model:

vector space

word vectors

word2vec

most_similar('france'):

spania 0.678515
belgium 0.665923
netherlands 0.652428
italy 0.633130

highest cosine distance values in vector space of the nearest words
Learning from text

- word2vec learns from input text
- considers each word $w_0$ in turn, along with its context $C$
- context = neighbouring words (here, for simplicity, 2 words forward and back)

<table>
<thead>
<tr>
<th>sample #</th>
<th>$w_0$</th>
<th>context $C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>once</td>
<td>{upon, a}</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>time</td>
<td>{upon, a, in, a}</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Two approaches: CBOW and Skip-gram

word2vec can learn the word vectors via two distinct learning tasks, **CBOW** and **Skip-gram**.

- **CBOW**: predict the current word $w_0$ given only $C$
- **Skip-gram**: predict words from $C$ given $w_0$
- Skip-gram produces better word vectors for infrequent words
- CBOW is faster by a factor of window size – more appropriate for larger corpora
Two model

CBOW

Skip-gram
An example

```python
from keras.models import Sequential
from keras.layers import Dense
import numpy

seed = 7  # fix random seed for reproducibility
numpy.random.seed(seed)

# load dataset
dataset = numpy.loadtxt("train.csv", delimiter="",")
# split into input (X) and output (Y) variables
X = dataset[:,0:8]
Y = dataset[:,8]

# create model
model = Sequential()
model.add(Dense(12, input_dim=8, init='uniform', activation='relu'))
model.add(Dense(8, init='uniform', activation='relu'))
model.add(Dense(1, init='uniform', activation='sigmoid'))

# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

model.fit(X, Y, nb_epoch=150, batch_size=10)  # Fit the model
scores = model.evaluate(X, Y)  # evaluate the model
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```