Word2Vec and Sentiment Analysis

An implementation from scratch, using only plain python

S.Shankar

2 May 2017

Overview

- This project demonstrates a sentiment analysis application.

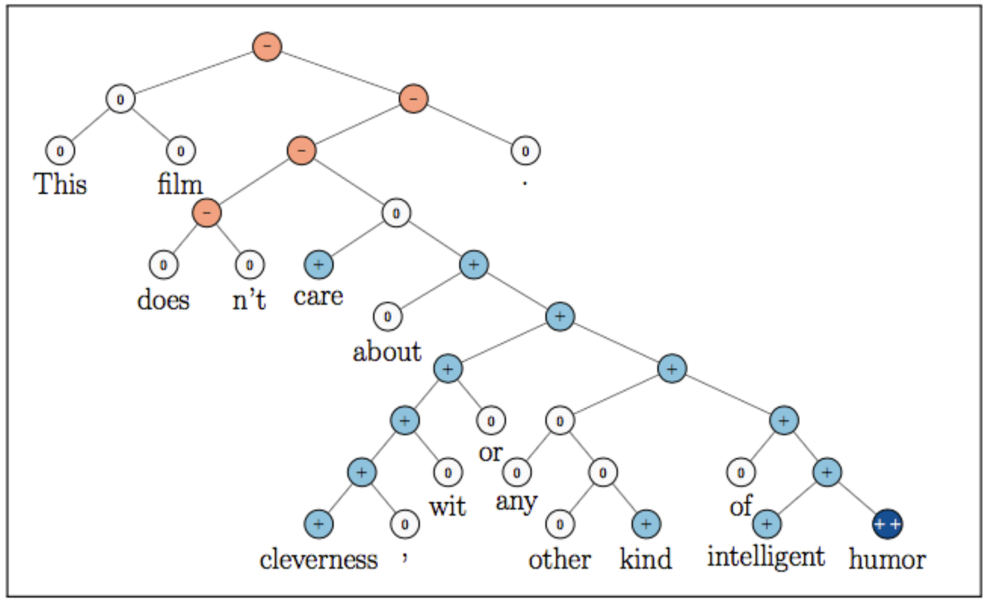
- Here is the catch: word2vec is implemented from scratch, using only plain python primitives.

Stanford Sentiment Treebank (SST)

- SST offers a fine grained evaluation of sentiment analysis algorithms.

- Plausible phrases are parsed by the Stanford parser.

- Sample output of parse tree:



an artist|48897

an artist who is simply tired --|192486

an artist who is simply tired -- of fighting the same fights|192487

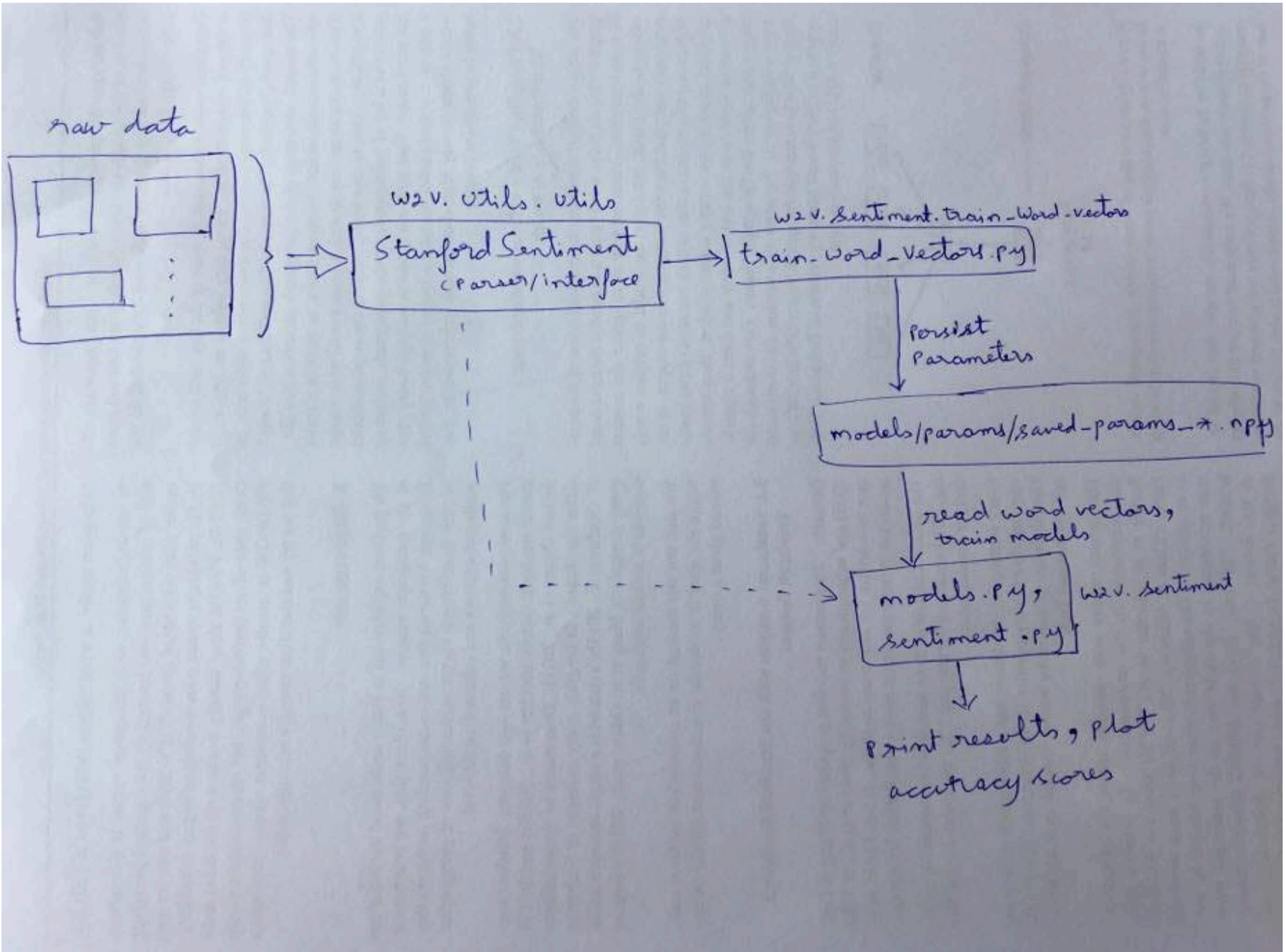
an artist who is simply tired -- of fighting the same fights , of putting the weight of the world on his shoulders|192489

an artist who is simply tired -- of fighting the same fights , of putting the weight of the world on his shoulders , of playing with narrative form|192491

an artistic collaboration|48900

- Reference: "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank"
  See references for paper
  See w2v/utils/utils.py for parsing code

Sentiment analysis pipeline



Sentiment analysis:

- Train word vectors using the word2vec package
- Learn a softmax-regression model: map from word vectors to sentiment scores
- Use L2 regularisation!
- Softmax regression cost function: 1/N * sum(cross_entropy(x_i, y_i)) + 1/2*|w|^2

Sentiment analysis pipeline

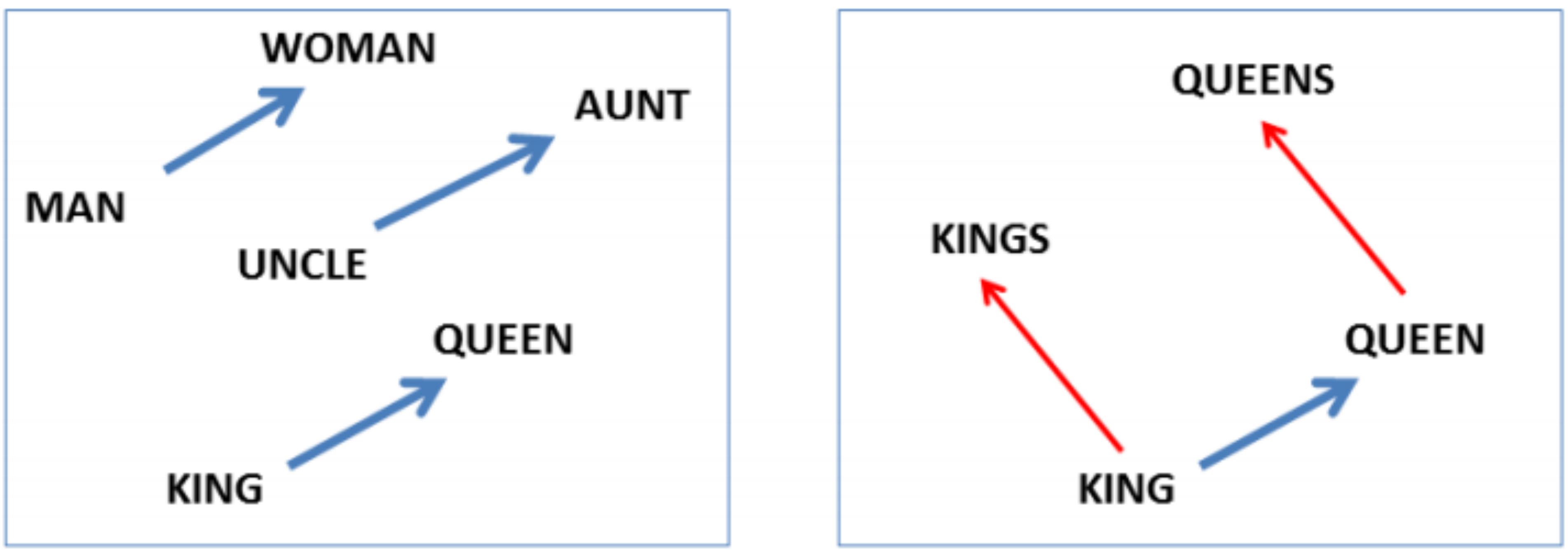
- 1-button run project:
    - make sentiment

Word2Vec

- Word2Vec: an algorithm to represent words with vectors:
  - Fundamental step in NLP: first step in many, many downstream application such as search, classification, sentiment, translation, question-answering, etc.

Word2Vec

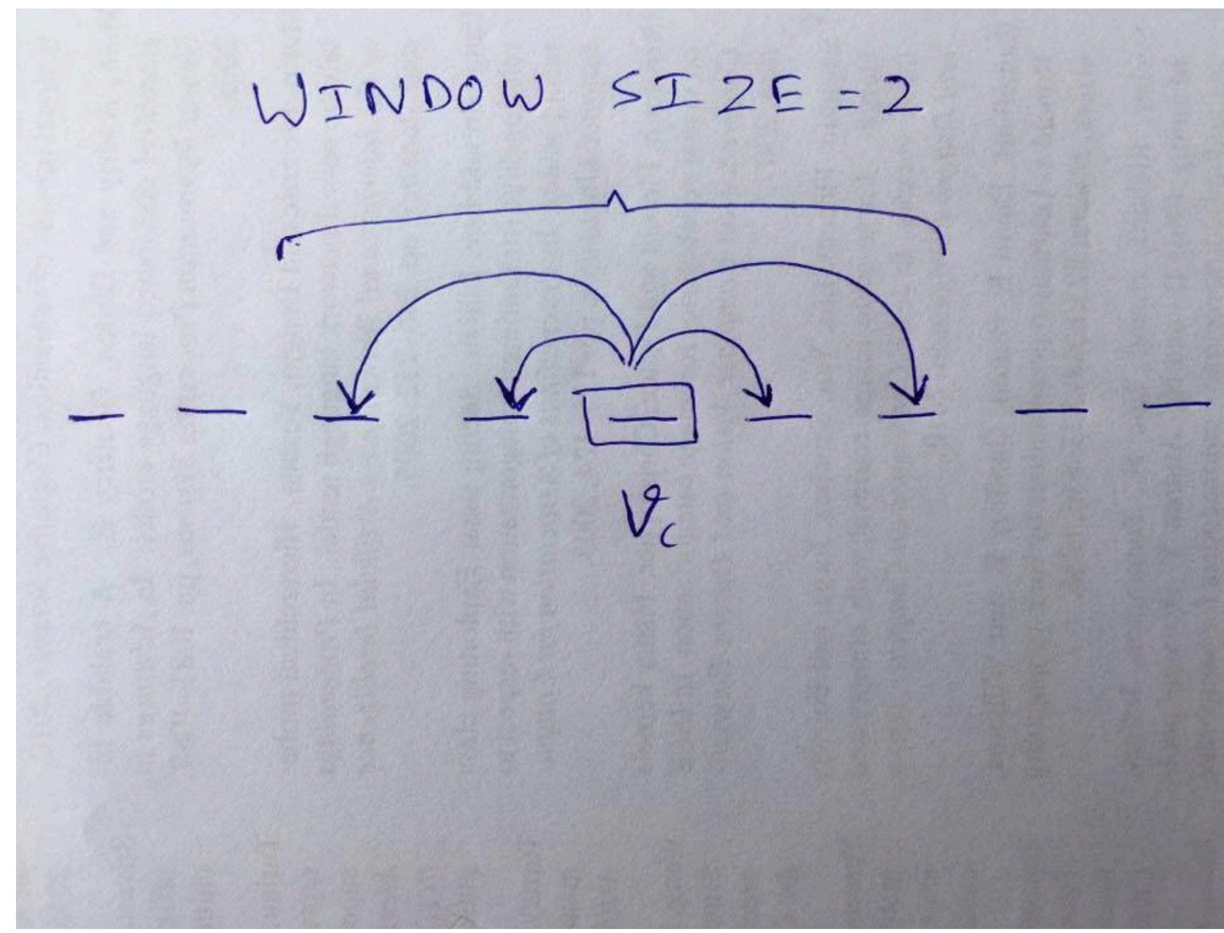
- man:woman :: king: ? —> queen



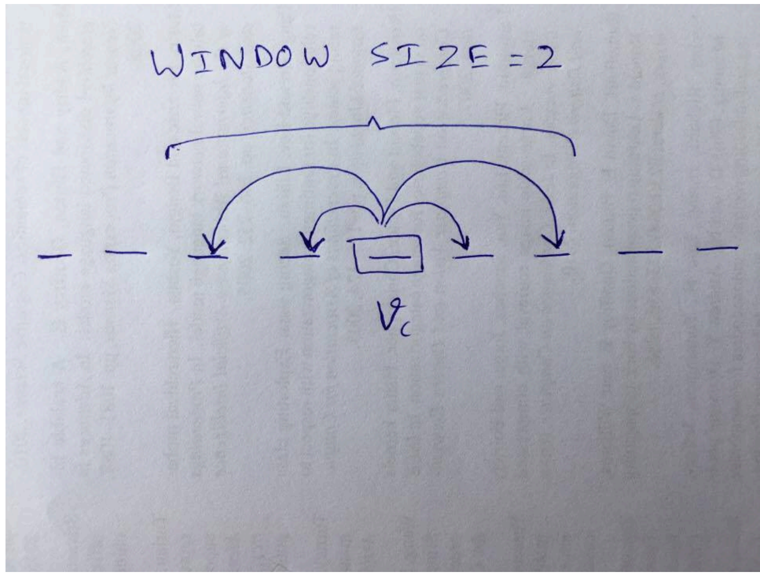
(Mikolov et al., NAACL HLT, 2013)

Word2Vec: Intuition

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



Word2Vec: Intuition

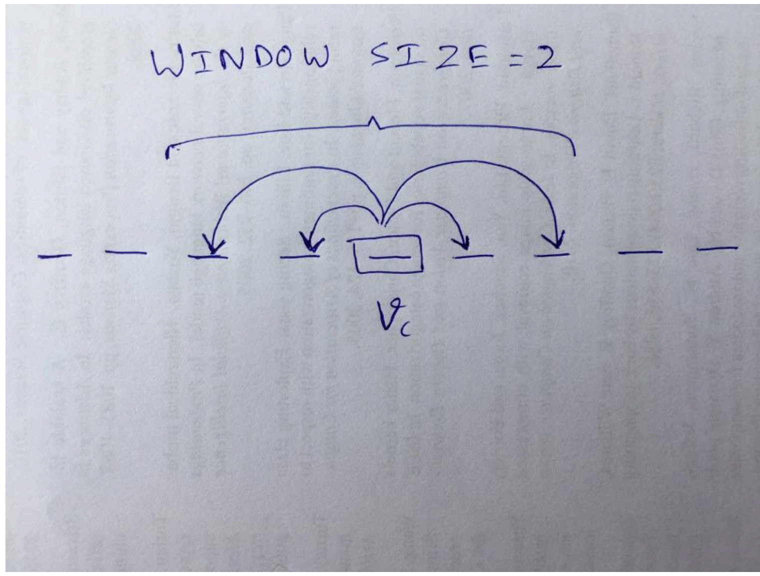


- Increase the probability of correct word co-occurring, decrease the probability of other words co-occurring.

- Probability of words co-occurring:

$$p(o|c) = \frac{\exp\left(u_o^T v_c\right)}{\sum_{w=1}^{W} \exp\left(u_w^T v_c\right)}$$

Word2Vec: Intuition



- Express our intuition in terms of a cost function :

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-m \le j \le m, j \neq 0} \log p(w_{t+j} | w_t)$$

- Minimising the above cost function will result in word vectors that "best" express our intuition.

Word2Vec: Details

- Actually, we have 2 variations of the cost function:

    - Softmax-CE:

$$\hat{\boldsymbol{y}}_o = p(\boldsymbol{o} \mid \boldsymbol{c}) = \frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{w=1}^{W} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)}$$

$$CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_i y_i \log(\hat{y}_i)$$

$$J_{softmax-CE}(\boldsymbol{o}, \boldsymbol{v}_c, \boldsymbol{U}) = CE(\boldsymbol{y}, \hat{\boldsymbol{y}})$$

Word2Vec: Details

- Actually, we have 2 variations of the cost function:

    - Negative sampling:

$$J_{neg-sample}(\boldsymbol{o}, \boldsymbol{v}_c, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))$$

Word2Vec: Details

- We also have 2 models for word2vec:
  - Skipgram:

$$J_{\text{skip-gram}}(\text{word}_{c-m\ldots c+m}) = \sum_{-m \le j \le m, j \ne 0} F(\boldsymbol{w}_{c+j}, \boldsymbol{v}_c)$$

Word2Vec: Details

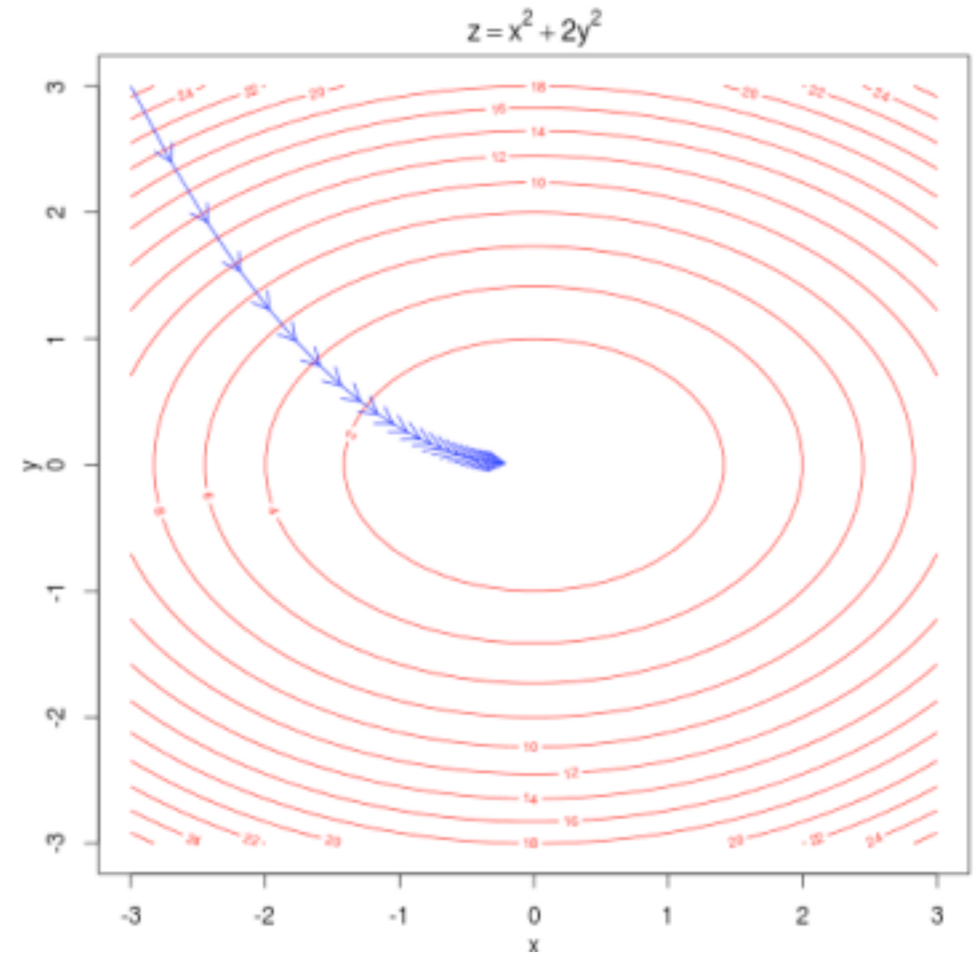
- We also have 2 models for word2vec:
  - CBOW:

$$\hat{\boldsymbol{v}} = \sum_{-m \leq j \leq m, j \neq 0} \boldsymbol{v}_{c+j}$$

$$J_{\text{CBOW}}(\text{word}_{c-m \ldots c+m}) = F(\boldsymbol{w}_c, \hat{\boldsymbol{v}})$$

Word2Vec: SGD

- SGD: numerically minimise a differentiable function



$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

```
while True:
    theta_grad = evaluate_gradient(J,corpus,theta)
    theta = theta - alpha * theta_grad
```

Differentiation

- We compute gradients from scratch
- Empirical gradient checking

Word2Vec: Differentiation

- Gradients for softmax-CE:

$$\frac{\partial J}{\partial \boldsymbol{v}_c} = U^T(\hat{\boldsymbol{y}} - \boldsymbol{y}).$$

$$\frac{\partial J}{\partial \boldsymbol{U}} = \boldsymbol{v}_c(\hat{\boldsymbol{y}} - \boldsymbol{y})^\top$$

Word2Vec: Differentiation

- Gradients for negative sampling:

$$\frac{\partial J}{\partial \boldsymbol{v}_c} = (\sigma(\boldsymbol{u}_o^\top v_c) - 1)\boldsymbol{u}_o - \sum_{k=1}^{K}(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c) - 1)\boldsymbol{u}_k$$

$$\frac{\partial J}{\partial \boldsymbol{u}_o} = (\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c) - 1)\boldsymbol{v}_c$$

$$\frac{\partial J}{\partial \boldsymbol{u}_k} = -(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c) - 1)\boldsymbol{v}_c, \quad \text{for all } k = 1, 2, \ldots, K$$

Word2Vec: Differentiation

- Gradients for skipgram:

$$\frac{\partial J_{\text{skip-gram}}(\text{word}_{c-m\ldots c+m})}{\partial \boldsymbol{U}} = \sum_{-m\leq j\leq m, j\neq 0} \frac{\partial F(\boldsymbol{w}_{c+j}, \boldsymbol{v}_c)}{\partial \boldsymbol{U}},$$

$$\frac{\partial J_{\text{skip-gram}}(\text{word}_{c-m\ldots c+m})}{\partial \boldsymbol{v}_c} = \sum_{-m\leq j\leq m, j\neq 0} \frac{\partial F(\boldsymbol{w}_{c+j}, \boldsymbol{v}_c)}{\partial \boldsymbol{v}_c},$$

$$\frac{\partial J_{\text{skip-gram}}(\text{word}_{c-m\ldots c+m})}{\partial \boldsymbol{v}_j} = \mathbf{0}, \text{for all } j \neq c.$$

Word2Vec: Differentiation

- Gradients for CBOW:

$$\frac{\partial J_{\text{CBOW}}(\text{word}_{c-m\ldots c+m})}{\partial \boldsymbol{U}} = \frac{\partial F(\boldsymbol{w}_c, \hat{\boldsymbol{v}})}{\partial \boldsymbol{U}},$$

$$\frac{\partial J_{\text{CBOW}}(\text{word}_{c-m\ldots c+m})}{\partial \boldsymbol{v}_j} = \frac{\partial F(\boldsymbol{w}_c, \hat{\boldsymbol{v}})}{\partial \hat{\boldsymbol{v}}},$$

$$\frac{\partial J_{\text{CBOW}}(\text{word}_{c-m\ldots c+m})}{\partial \boldsymbol{v}_j} = \boldsymbol{0}, \quad \text{for all } j$$