Distributed word representations: Retrofitting

Christopher Potts

Stanford Linguistics

CS224u: Natural language understanding
Central goals

- Distributional representations are powerful and easy to obtain, but they tend to reflect only similarity (synonymy, connotation).

- Structured resources are sparse and hard to obtain, but they support learning rich, diverse semantic distinctions.

- Can we have the best aspects of both? Retrofitting is one way of saying, “Yes”.

- Retrofitting is due to Faruqui et al. (2015).
The retrofitting model

\[ \sum_{i \in V} \alpha_i \|q_i - \hat{q}_i\|^2 + \sum_{(i,j,r) \in E} \beta_{ij} \|q_i - q_j\|^2 \]

- Balances **fidelity to the original vector** \( \hat{q}_i \)
- **against looking more like** one’s graph neighbors.
- **Forces are balanced with** \( \alpha = 1 \) and \( \beta = \frac{1}{\text{Degree}(i)} \)

Figure 1: Word graph with edges between related words showing the observed (grey) and the inferred (white) word vector representations.
Simple retrofitting examples

\[
\sum_{i \in V} \alpha_i \| q_i - \hat{q}_i \|^2 + \sum_{(i,j,r) \in E} \beta_{ij} \| q_i - q_j \|^2
\]
Simple retrofitting examples

\[
\sum_{i \in V} \alpha_i \|q_i - \hat{q}_i\|^2 + \sum_{(i,j,r) \in E} \beta_{ij} \|q_i - q_j\|^2
\]
Simple retrofitting examples

\[ \sum_{i \in V} \alpha_i \| q_i - \hat{q}_i \|^2 + \sum_{(i,j,r) \in E} \beta_{ij} \| q_i - q_j \|^2 \]

\[ \alpha = 0 \]
Extensions

Drop the assumption that every edge means ‘similar’:

- Mrkšić et al. (2016) AntonymRepel, SynonymAttract, and VectorSpacePreservation for different edge types.

- Lengerich et al. (2018): functional retrofitting to learn the semantics of any edge types.

- This work is closely related to **graph embedding** (learning distributed representations for nodes), for which see Hamilton et al. 2017.
Code snippets

[1]: import pandas as pd
   from retrofitting import Retrofitter

[2]: Q_hat = pd.DataFrame(
    [[0.0, 0.0],
     [0.0, 0.5],
     [0.5, 0.0]],
    columns=['x', 'y'])

   edges = {0: {1, 2}, 1: set(), 2: set()}

[3]: Q_hat

[4]: retro = Retrofitter(verbosetrue)

[5]: X_retro = retro.fit(Q_hat, edges)

   Converged at iteration 2; change was 0.0000

[6]: X_retro

[6]: x y
   0 0.125 0.125
   1 0.000 0.500
   2 0.500 0.000

[7]: # For an application to WordNet, see `usm_03_retrofitting`.

