Computational Intelligence Laboratory

Word Embeddings & Sentiment Classification

An Bian, Mikhail Karasikov

19-20 April 2018

ETH Zurich – cil.inf.ethz.ch

Overview

Word Embeddings

Suppose we are given a dictionary of words $\mathcal{V} = \{w_1, w_2 \dots\}$.

The i-th word

$$w_i \in \mathcal{V}$$

is represented by an embedding

$$\mathbf{x}_{w_i} \in \mathbb{R}^d,$$

a d-dimensional latent vector.

Embeddings capture the meaning of the words:

- Similar words should have similar embeddings $\mathbf{x}_{w_i} \approx \mathbf{x}_{w_j}$
- Angles and distances between embeddings relate to comparing meaning $\langle \mathbf{x}_{w_i}, \mathbf{x}_{w_j} \rangle$

The embedding is a way of representing (word) meaning

Discrete Representation

Represent a vector by its index in the vocabulary

$$[0\ 0\ 0\ 1\ 0\ 0\ 0\ \dots\ 0\ 0\ 0]$$

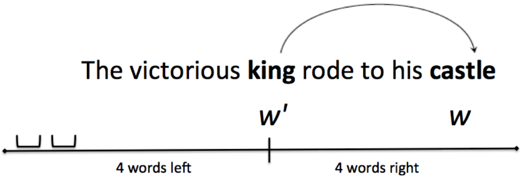
— "one-hot" vector representation.

Problems:

- Dimensionality
  Wikipedia + Gigaword (400K vocab),
  English Language (1M vocab),
  Twitter-2B tweets (1.2M vocab)
- Do not capture similarity of words
  good $= [0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$
  great $= [0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$
  milk $= [0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$
  (good AND great) $=$ (good AND milk) $= 0$

Distributional Similarity Based Representations

Represent a word by its neighbors is the key idea in modern NLP



- **Close words have common context**

A Basic Latent Vector Model

- Latent vector representation of words = embedding

$$w \mapsto (\mathbf{x}_w, b_w) \in \mathbb{R}^{D+1}, \quad \text{(vector + bias)}$$

- Define **log-bilinear** model

$$\log p_\theta(w \,|\, w') = \langle \mathbf{x}_w, \mathbf{x}_{w'} \rangle + b_w + \text{const}$$

  - symmetric bilinear form fitted to log-probabilities
  - normalization constant (see below)

A Basic Latent Vector Model (cont'd)

- Exponentiating $\Longrightarrow$ **soft-max**

$$p_\theta(w \,|\, w') = \frac{\exp\left[\langle \mathbf{x}_w, \mathbf{x}_{w'} \rangle + b_w\right]}{Z_\theta(w')}$$

  - partition function (normalization constant):

$$Z_\theta(w') := \sum_{v \in \mathcal{V}} \exp\left[\langle \mathbf{x}_v, \mathbf{x}_{w'} \rangle + b_v\right]$$

  - the model parameters:

$$\theta = ((\mathbf{x}_w, b_w)_{w \in \mathcal{V}}) \in \mathbb{R}^{(D+1)\cdot|\mathcal{V}|}$$

GloVe: Co-occurrence Matrix

Summarize data in **co-occurrence matrix**

$$\mathbf{N} = (n_{ij}) \in \mathbb{R}^{|\mathcal{V}|\cdot|\mathcal{C}|},$$

$$n_{ij} = \#\text{ occurrences of } w_i \in \mathcal{V} \text{ in context of } w_j \in \mathcal{C}$$

Example corpus:

- The king rode to his castle.
- The king lives in the castle.

counts	the	king	rode	lives	to	in	his	castle
the	0	2	0	0	0	1	0	1
king	2	0	1	1	0	0	0	0
rode	0	1	0	0	1	0	0	0
lives	0	1	0	0	0	1	0	0
to	0	0	1	0	0	0	1	0
in	1	0	0	1	0	0	0	0
his	0	0	0	0	1	0	0	1
castle	1	0	0	0	0	0	1	0

GloVe Objective

Weighted least squares fit of log-counts

$$\mathcal{H}(\theta; \mathbf{N}) = \sum_{i,j} f(n_{ij}) \left(\underbrace{\log n_{ij}}_{\text{target}} - \underbrace{\log \tilde{p}_\theta(w_i|w_j)}_{\text{model}} \right)^2,$$

with **unnormalized** distribution

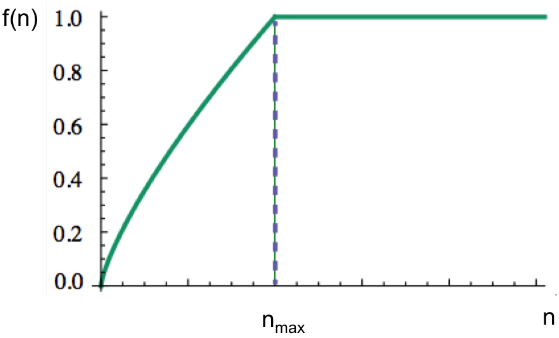
$$\tilde{p}_\theta(w_i|w_j) = \exp\left[\langle \mathbf{x}_i, \mathbf{y}_j \rangle + b_i + d_j\right]$$

and **weighting function** f.

$\mathbf{x}_i$: word embeddings

$\mathbf{y}_i$: context embeddings

GloVe Weighting



- Scalable to large corpora
- Fast training
- Limit influence of large counts (very frequent words)

How to optimize the objective of GloVe?

- Non-convex problem: hard to find global minimum
- Goal: Minimize the objective/cost function
- Use gradient descent method

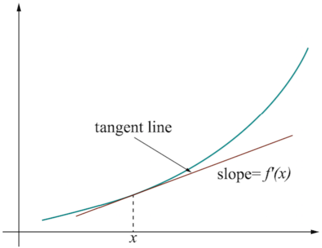
Trivial example: Find a local minimum of the function $f(x) = x^2 - 6x$, where $f'(x) = 2x - 6$.

```python
x_old = 0
x_new = 6
eps = 0.01
precision = 0.00001

def f_derivative(x):
    return 2*x - 6

while abs(x_new - x_old) > precision:
    x_old = x_new
    x_new = x_old - eps * f_derivative(x_old)

print("Local minimum occurs at ", x_new)
```



Full-Batch GloVe Optimization (no!)

$$\mathcal{H}(\theta; \mathbf{N}) = \sum_{i,j} f(n_{ij}) \left(\underbrace{\log n_{ij}}_{\text{target}} - \underbrace{\log \tilde{p}_\theta(w_i|w_j)}_{\text{model}} \right)^2 \to \min_\theta$$

- Non-convex objective: hard to find global minimum
- Gradient descent (aka **steepest descent**)

$$\theta^{\text{new}} \leftarrow \theta^{\text{old}} - \eta \nabla_\theta \mathcal{H}(\theta; \mathbf{N}), \quad \eta > 0 \text{ (step size)}$$

$$\theta = ((\mathbf{x}_w, b_w)_{w \in \mathcal{V}}, (\mathbf{y}_w, d_w)_{w \in \mathcal{C}}), \text{ embeddings are the parameters!}$$

Minimization over the full batch (the entire training data) would require us to compute gradients for **all entries** in the co-occurence matrix w.r.t. **all parameters**

SGD GloVe Optimization (yes!)

There might be billions of entries in the co-occurence matrix!
Long waiting time before each single update of parameters

- Non-convex problem: hard to find global minimum
- Use stochastic optimization to find local minimum
- Stochastic gradient descent (SGD):
  1. **sample** (i, j) such that $n_{ij} > 0$ uniformly at random
  2. perform "cheap" **update** (single entry and sparse)

$$\mathbf{x}_i^{\text{new}} \leftarrow \mathbf{x}_i + 2\eta f(n_{ij}) \left(\log n_{ij} - \langle \mathbf{x}_i, \mathbf{y}_j \rangle\right) \mathbf{y}_j$$

$$\mathbf{y}_j^{\text{new}} \leftarrow \mathbf{y}_j + 2\eta f(n_{ij}) \left(\log n_{ij} - \langle \mathbf{x}_i, \mathbf{y}_j \rangle\right) \mathbf{x}_i$$

How to Evaluate Word Vectors?

Intrinsic evaluation (on a specific or intermediate task)

- Fast to compute
- Leads to better understanding of the system
- Usefulness depends on correlation to the realistic task

Extrinsic evaluation (on a real task)

- Can take a long time to compute accuracy
- Requires an expert assessment and user comments

Practical Tips

- What to do with the two types of embeddings:
  - $\mathbf{x}_i$: word embeddings
  - $\mathbf{y}_i$: context embeddings
- Both capture similar co-occurence
- To get the final embeddings, a simple and efficient way is to sum them up:

$$\mathbf{x}_i := \mathbf{x}_i + \mathbf{y}_i$$

- Different variations (Pennington et al. (2014))

Practical Tips

Deal with ambiguity

- Homonyms (e.g. address) are captured as one vector
- The vector is pulled to different directions
  - Cluster context windows of words using K-means, and retrain with each word assigned to multiple clusters

Retrain word vectors

Visualization

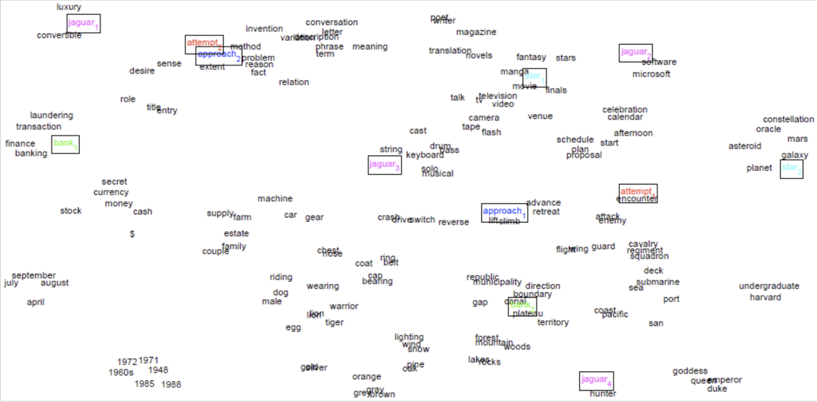


Figure: Huang et al. 2012

Pen and Paper Assignment

Problem 1:

1) The objective of GloVe is

$$\mathcal{H}(\theta; \mathbf{N}) = \sum_{i,j} f(n_{ij}) \left(\underbrace{\log n_{ij}}_{\text{target}} - \underbrace{\log \tilde{p}_\theta(w_i|w_j)}_{\text{model}} \right)^2 .$$

Suppose $f(\cdot) \equiv 1$ for all arguments, and $m_{ij} = \log n_{ij}$.

- a) Derive the gradient of the objective function of GloVe w.r.t. $\mathbf{x}_i$ and $\mathbf{y}_j$
- b) Derive the stochastic gradient of the objective function of GloVe w.r.t. $\mathbf{x}_i$ and $\mathbf{y}_j$

Pen and Paper Assignment

Problem 1:

2) Show that GloVe with

$$f(n_{ij}) := \begin{cases} 1 & \text{if } n_{ij} > 0, \\ 0 & \text{otherwise.} \end{cases}$$

solves the **matrix completion** problem

$$\min_{\mathbf{X},\mathbf{Y}} \sum_{ij\,:\,n_{ij}>0} \left(m_{ij} - (\mathbf{X}^\top \mathbf{Y})_{ij} \right)^2 .$$

3) Derive the gradient and stochastic gradient of $\mathcal{H}$ for arbitrary weighting function f.

Sentiment Classification

Given a tweet, predict whether it has a positive or negative opinion
e.g.: whether a tweet message contains a :) or :(

Examples:

- "i know android sucks :("
- "twitter is dead right now :("
- "my sis made apple crisp with extra crisp ! it's awesome :)"
- "i hope your wednesday was awesome :)"

Project 2: The Dataset Provided

- Twitter data:
  - train_pos_full.txt $\sim$ 1M tweets that contained :)
  - train_neg_full.txt $\sim$ 1M tweets that contained :(
  - train_pos.txt - 10% from the positive tweets for training
  - train_neg.txt - 10% from the negative tweets for training
  - test_data.txt - 10K unlabeled tweets
- Each tweet contains at most 140 characters
- All tweets are tokenized - words are separated by a single whitespace
- All labels (smileys) are removed
- User mentions replaced with <user>
- Links replaced with <url>

A Simple Baseline via Word Embeddings

1. Average the word embeddings to get an embedding for the whole tweet message
2. Feed the resulting word embedding to a classifier
3. If the tweet has positive sentiment, its embedding is close to the words that represent positive meaning (close to good, great, amazing and far from bad, horrible etc.)

Choosing A Classifier

- Any classifier you like
- Logistic regression, Support Vector Machine (SVM), Gaussian process classifier, neural network, ...
- Ensemble methods
- Deep neural nets give a good performance if the hyper-parameters are carefully tuned
- No need to go deep into classifiers so far, can use libraries :-)
  - scikit-learn: a library for Machine learning in Python
  - xgboost: a library for Gradient Boosting
  - etc

Logistic Regression from scikit-learn

- a linear model for classification rather than regression
- a.k.a. logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier
- The probabilities describing the possible outcomes of a single trial are modeled using a logistic function

The code example for scikit-learn

- Logistic Regression, the Iris dataset

Python source code: plot_iris_logistic.py

Import data

```
print(__doc__)
# Code source: Gael Varoquaux

import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model, datasets

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features.
Y = iris.target
h = .02  # step size in the mesh
```

Train the model

```
logreg = linear_model.LogisticRegression(C=1e5)
logreg.fit(X, Y)
```

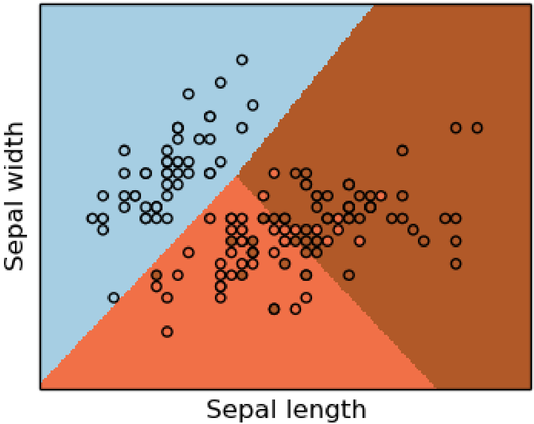
Python source code: show results

```
# Plot the decision boundary.
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)
# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())

plt.show()
```

Python source code: The decision boundary



Why is the baseline overly simplistic?

- If the tweet consists of only positive (negative) words it may correctly predict the sentiment
- However, it fails with double negation or mixed words
  "It was not horrible and definitely not boring"
  "It started pretty bad, but it turned out to be amazing"
- The task is to improve the baseline taking this into consideration

Possible Improvements

Retrain word embeddings

- Initialize word embeddings using GloVe
- Treat them as parameters
- Retrain the word embeddings during the classification