

CSE Machine Learning Paper Review

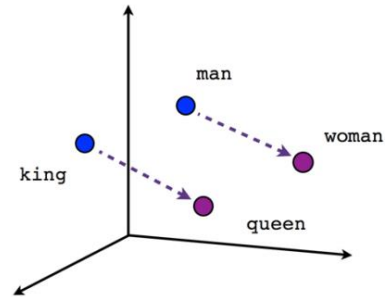
GloVe: Global Vectors for Word Representation

By: Jeffrey Pennington, Richard Socher, Christopher Manning

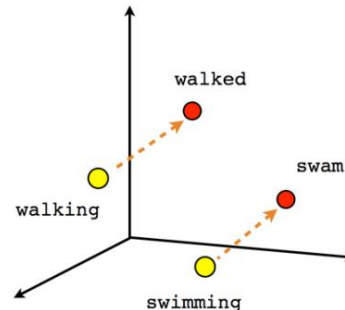
<https://nlp.stanford.edu/pubs/glove.pdf>

A bit about word embeddings

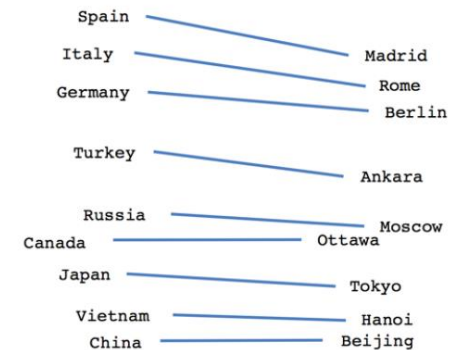
- An important requirement for many NLP tasks is a measure of semantic similarity between words within a corpus (or across corpora)
- This requires a representation for words where such semantic attributes can be encoded
 - Text representations or numeric tokenized forms (based on a dictionary) are not useful for such a representation
- If a corpus is viewed as a high dimensional vector space, words can be viewed as points in that space i.e. vectors
 - Distance/Angle measures between such vectors can then represent semantic correlation
- These word vectors are also otherwise known as “embeddings”
 - The words get “embedded” into the vector space
- GloVe is a model that allows deriving embeddings for words in a corpus
 - You use vector dimensions of your choice and leverage a global (corpus wide) cooccurrence matrix
- GloVe is not your usual model training task (i.e. a predictor or a classifier etc.) – it is more a pre-training task to transform inputs for a downstream model that performs NLP tasks like NER etc.



Male-Female



Verb tense



Country-Capital

Cooccurrence Matrix & Probability Ratios

- A cooccurrence matrix records the cooccurrence counts of context words with respect to a center word within a specific window

The Cat Sat On The Mat

	The	Cat	Sat	On	Mat
The	0	0	0	0	0
Cat	0	0	0	0	0
Sat	0	0	0	0	0
On	0	0	0	0	0
Mat	0	0	0	0	0

Window Width = 2 (on each side of the center word)

Cooccurrence Matrix & Probability Ratios

- A cooccurrence matrix records the cooccurrence counts of context words with respect to a center word within a specific window

The Cat Sat On The Mat

	The	Cat	Sat	On	Mat
The	1	1	1	0	0
Cat	0	0	0	0	0
Sat	0	0	0	0	0
On	0	0	0	0	0
Mat	0	0	0	0	0

Window Width = 2 (on each side of the center word)

Cooccurrence Matrix & Probability Ratios

- A cooccurrence matrix records the cooccurrence counts of context words with respect to a center word within a specific window

The **Cat** Sat On The Mat

	The	Cat	Sat	On	Mat
The	1	1	1	0	0
Cat	1	0	1	1	0
Sat	0	0	0	0	0
On	0	0	0	0	0
Mat	0	0	0	0	0

Window Width = 2 (on each side of the center word)

Cooccurrence Matrix & Probability Ratios

- A cooccurrence matrix records the cooccurrence counts of context words with respect to a center word within a specific window

The Cat **Sat** On The Mat

	The	Cat	Sat	On	Mat
The	1	1	1	0	0
Cat	1	0	1	1	0
Sat	2	1	1	1	0
On	0	0	0	0	0
Mat	0	0	0	0	0

Window Width = 2 (on each side of the center word)

Cooccurrence Matrix & Probability Ratios

- A cooccurrence matrix records the cooccurrence counts of context words with respect to a center word within a specific window

The Cat Sat **On** The Mat

	The	Cat	Sat	On	Mat
The	1	1	1	0	0
Cat	1	0	1	1	0
Sat	2	1	1	1	0
On	1	1	1	1	1
Mat	0	0	0	0	0

Window Width = 2 (on each side of the center word)

Cooccurrence Matrix & Probability Ratios

- A cooccurrence matrix records the cooccurrence counts of context words with respect to a center word within a specific window

The Cat Sat On The Mat

	The	Cat	Sat	On	Mat
The	2	1	2	1	1
Cat	1	0	1	1	0
Sat	2	1	1	1	0
On	1	1	1	1	1
Mat	0	0	0	0	0

Window Width = 2 (on each side of the center word)

Cooccurrence Matrix & Probability Ratios

- A cooccurrence matrix records the cooccurrence counts of context words with respect to a center word within a specific window

The Cat Sat On The **Mat**

	The	Cat	Sat	On	Mat
The	2	1	2	1	1
Cat	1	0	1	1	0
Sat	2	1	1	1	0
On	1	1	1	1	1
Mat	1	0	0	1	0

Window Width = 2 (on each side of the center word)

Cooccurrence Matrix & Probability Ratios

- A cooccurrence matrix records the cooccurrence counts of context words with respect to a center word within a specific window

The Cat Sat On The Mat

	The	Cat	Sat	On	Mat
The	2	1	2	1	1
Cat	1	0	1	1	0
Sat	2	1	1	1	0
On	1	1	1	1	1
Mat	1	0	0	1	0

Window Width = 2 (on each side of the center word)

- Cooccurrence Matrix is calculated on entire corpus prior to training.
- Helps produce cooccurrence probabilities

e.g.

- $P(\text{Sat} \mid \text{Cat}) = 1/3$
- $P(\text{Sat} \mid \text{The}) = 2/7$

Probability of the word k cooccurring with word i :

$$P_{ik} = P(W_k \mid W_i) = X_{ik} / \sum_j X_{ij}$$

Cooccurrence Matrix & Probability Ratios

Probability and Ratio	$k = \text{solid}$	$k = \text{gas}$	$k = \text{water}$	$k = \text{fashion}$
$P(k \text{ice})$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k \text{steam})$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k \text{ice})/P(k \text{steam})$	8.9	8.5×10^{-2}	1.36	0.96

Source: <https://nlp.stanford.edu/pubs/glove.pdf>


- Probability ratios provide a measure of how likely a word k is to cooccur with a word x , compared to its likelihood of cooccurrence with another word y
- From above
 - “solid” is much more likely to cooccur with “ice” than with “steam” i.e. more relevant to “ice”
 - “gas” is much more likely to cooccur with “steam” than with “ice” i.e. more relevant to “steam”
 - “water” and “fashion” are almost equally likely to cooccur with both “ice” and “steam” i.e. equally relevant or irrelevant
- Suggests probability ratios is a better way to gauge correlation between words
- This leads to the abstract GloVe model form:

$$F(w_i, w_j, w_k) = P_{ik} / P_{jk}$$

where the r.h.s is the ratio of the probabilities of word k cooccurring with words i and j respectively and the l.h.s is some function of the word vectors for words i , j and k

- The word vectors (and any related biases) are the “learned” parameters in such a model

GloVe Model – Objective Function

$$F(w_i, w_j, w_k) = P_{ik} / P_{jk}$$


Derivation skipped for brevity

$$\sum_{i,j=1}^V f(x_{ij})(\omega_i^T \omega_j + b_i + b_j - \log x_{ij})^2$$

GloVe cost function (weighted least squares)

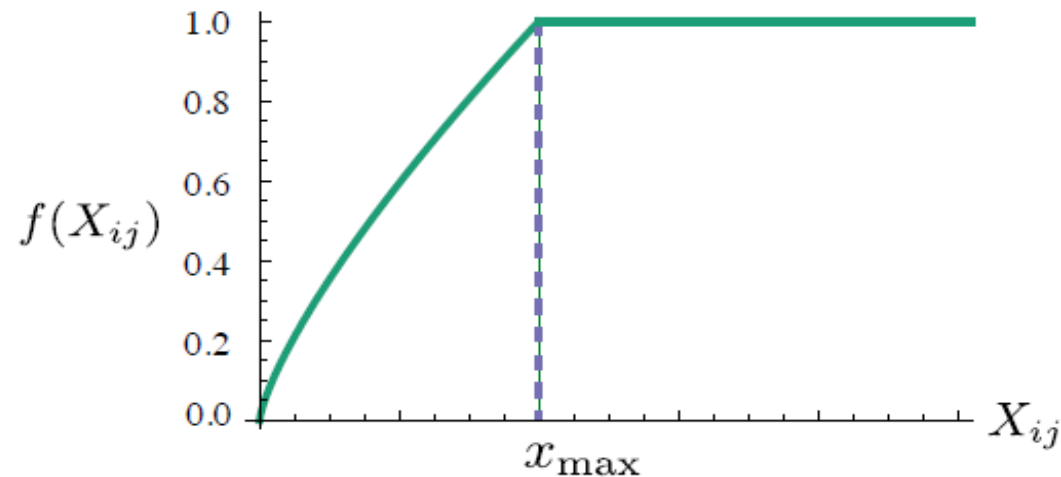
- w_i and w_j are the word vectors for the i^{th} center word (conventionally along the rows of the cooccurrence matrix) and the j^{th} context word (along the columns)
- b_i and b_j are the bias terms respectively associated with those word vectors
- x_{ij} is the cooccurrence count in the i, j^{th} cell of the cooccurrence matrix
- V is the vocabulary size (all unique words in the corpus)
- w_i, w_j, b_i and b_j are all trainable parameters
- The cost is summed over all i and j i.e. you end up with two vectors for each word – once as a center word vector and another as a context word vector
- Post training – sum or average the two vectors for each word or discard the context word vectors
 - All acceptable methods
- $f(x_{ij})$ is a weighting function – more on next slide

GloVe Model – Weighting Function

- Words that cooccur infrequently can introduce noise
- The weighting coefficient $f(x_{ij})$ is a function that penalizes infrequently cooccurring pairs
- $f(x_{ij}) = (x_{ij} / x_{max})^\alpha$ if $x_{ij} < x_{max}$ otherwise 1

where x_{max} is some prefixed cutoff (paper uses 100) and $\alpha = 0.75$ (per paper)

- Multiplying the squared loss by above has the effect of reducing the contribution from cooccurrence counts that are less than the prefixed cutoff while those from over the cutoff remain as is.



GloVe Objective Function (in code)

```
def loss_function(cooccur_mat:torch.tensor,
                  log_cooccur_mat:torch.tensor,
                  center_word_vectors:torch.tensor,
                  context_word_vectors:torch.tensor,
                  center_word_biases:torch.tensor,
                  context_word_biases:torch.tensor,
                  x_max = cooccurence_cutoff):

    ones = torch.ones_like(cooccur_mat,device = torch_device)
    weighting = torch.min(ones,(cooccur_mat/x_max)**(0.75))
    loss = torch.sum(
        weighting*(
(torch.matmul(center_word_vectors,context_word_vectors.transpose(0,1))
        + center_word_biases
        + context_word_biases
        - log_cooccur_mat)**2)
    )

    return loss
```

GloVe Training

- Nothing special – use standard gradient descent (or SGD or minibatch GD) to optimize loss
- Run as many iterations as you need – consider a stopping criterion

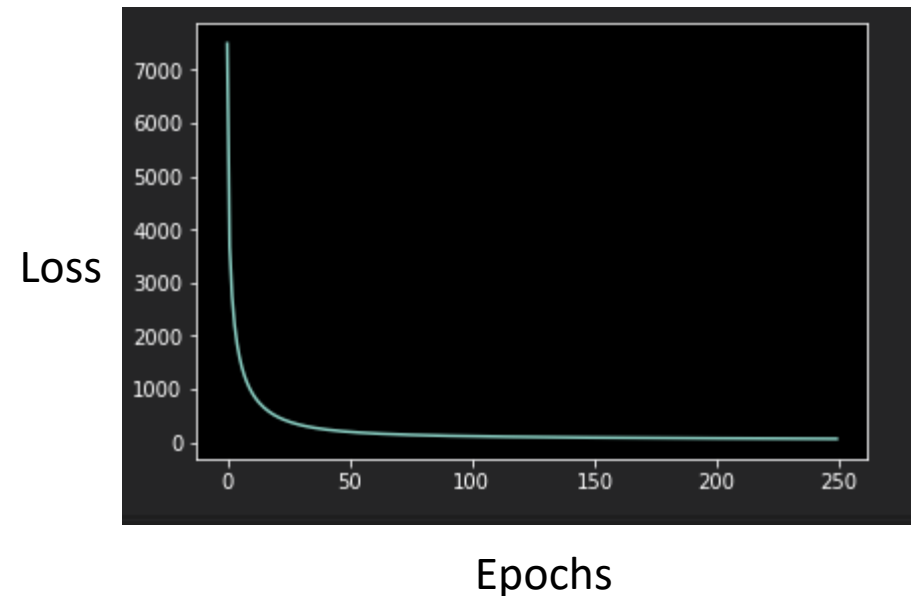
```
optimizer = torch.optim.Adagrad([weight_center,weight_context,bias_center,bias_context])
```

```
for epoch in range(epochs): #need stopping criterion
```

```
optimizer.zero_grad()
```

```
loss_val = loss_function(cooccur_mat,  
                        log_cooccur_mat,  
                        weight_center,  
                        weight_context,  
                        bias_center,  
                        bias_context)
```

```
loss_val.backward()  
optimizer.step()
```



GloVe – pre-trained vectors

- Available at <https://nlp.stanford.edu/projects/glove/>
- Also available via some popular NLP libraries such as SpaCy
 - All SpaCy English models are trained using GloVe on common crawl data
- Using pre-trained vectors

```
model = k.models.Sequential()  
model.add(k.layers.Embedding(max_words, embeddings_dim,  
                             input_length=max_sent_len, name="embedding"))  
embedding_layer = model.get_layer(name="embedding")  
embedding_layer.set_weights([glove_embeddings_matrix]) # of size vocab x word vector dim  
embedding_layer.trainable = False
```

- You should almost never need to train your own word vectors
 - Plenty of pre-trained data available – not just for GloVe
- The only situation where training your own word vectors might come into play is domain specific corpora

Questions ?