Connie Trojan
Supervisor: Jonathan Cumming

Word Embeddings for Language Modelling

Durham University

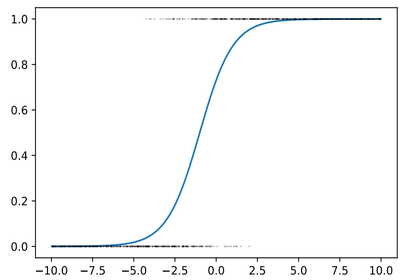
Background

A **language model** aims to assign a probability to a sequence of words, based on how likely they are to be spoken by a native speaker. They have many uses in text analysis - for example, next word prediction or grammar checking.

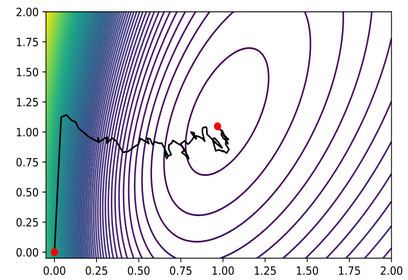
A key issue in language modelling is the large **vocabulary size** - we must model the meanings and interactions of millions of unique words. Models that consider words to be a list of discrete items in a vocabulary suffer from data sparsity issues even with large datasets, and cannot take advantage of similarities between words.

One solution to this is **word embeddings**: we map words to vectors in $\mathbb{R}^d$, designing the map so that words with similar meanings are mapped to vectors that are close together. These vectors will be low dimensional ($100 \leq d \leq 1000$) compared to the vocabulary size.

This poster focuses on **skip-gram with negative sampling**, which obtains embeddings by assuming that words with similar meanings appear in similar contexts. Embeddings will be obtained by using a **logistic regression** to distinguish between real context words and random ones, and stochastic gradient descent will be used to fit the model.



(a) Toy data for a logistic regression



(b) Fitting a logistic regression to the toy data with SGD

Figure 1

Logistic Regression

Logistic regression models the probability that an observation $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$ belongs to a class $y^{(i)} \in \{0, 1\}$ by passing a linear expression to the **sigmoid** function, σ:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

This takes inputs in $(-\infty, \infty)$ and outputs values in $(0, 1)$, allowing us to model the probability of observation i belonging to class 1 as follows:

$$\hat{p}_i := \mathbb{P}(y^{(i)} = 1 \,|\, \boldsymbol{x}^{(i)}) = \sigma(\boldsymbol{\beta} \cdot \boldsymbol{x}^{(i)}),$$

with parameter $\boldsymbol{\beta} \in \mathbb{R}^d$. The probability that the class is 0 is simply defined as:

$$1 - \hat{p}_i = \sigma(-\boldsymbol{\beta} \cdot \boldsymbol{x}^{(i)}),$$

The model can be fitted using **maximum likelihood estimation**. Assuming observations are independent Bernoulli random variables, the log likelihood for N observations is:

$$L(\boldsymbol{\beta}) = \sum_{i=1}^{N} L(\boldsymbol{\beta}; \boldsymbol{x}^{(i)}, y^{(i)}) = \sum_{i=1}^{N} y^{(i)} \log \hat{p}_i + (1 - y^{(i)}) \log(1 - \hat{p}_i)$$

The sigmoid function will be used to model the probability of some word w_j appearing in the context of another word w_i. The pair (w_i, w_j) will be labelled as class 1 if w_j was observed near w_i and 0 otherwise.

Skip-Gram with Negative Sampling

A **context word** for w_i will be defined as any word appearing within a window of length l around w_i anywhere in the training text. Consider the sentence:

Sing, and the hills will answer.

If we set $l = 2$ the observed context words, or **positive samples**, for "hills" are "the" and "will". Ignoring word order, we can represent these as the target-context pairs or **skip-grams** (*hills*,*the*) and (*hills*,*will*).

Training data for our model is created by treating all words observed as context words of the target as positive samples, and randomly generating **negative samples** from the rest of the vocabulary. For each positive sample (w, w^+), we generate k negative samples (w, w_j^-). In our example, (*hills*,*sing*) and (*hills*,*answer*) are possible negative samples since "sing" and "answer" were not observed within our context window for "hills".

We define two sets of embeddings, the **target embeddings** $\boldsymbol{v}_{w_i}$ and the **context embeddings** $\boldsymbol{c}_{w_j}$. Using logistic regression, the probability of w_j being a real context word for w_i will be defined in terms of these embeddings as:

$$\mathbb{P}(w_j | w_i) = \sigma(\boldsymbol{v}_{w_i} \cdot \boldsymbol{c}_{w_j})$$

The context embeddings can be discarded after training, although in practice they are often added or concatenated to the target embeddings instead. Making the assumption that context words occur independently of each other, the log likelihood for one positive observation and k negative is:

$$\log\Big(\mathbb{P}(w^+|w) \prod_{j=1}^{k} (1 - \mathbb{P}(w_j^-|w))\Big) = \log \sigma(\boldsymbol{c}_{w^+} \cdot \boldsymbol{v}_w) + \sum_{j=1}^{k} \log \sigma(-\boldsymbol{c}_{w_j^-} \cdot \boldsymbol{v}_w)$$

The total log likelihood for a dataset is obtained by summing over each positive sample for every observation of the word, for each word in the vocabulary.

Stochastic Gradient Descent (SGD)

Gradient descent aims to find the parameters $\boldsymbol{\theta} \in \mathbb{R}^d$ that minimise a **loss function** $L(\boldsymbol{\theta})$ by updating θ in the opposite direction to the gradient ∇L. We assume L is the average loss per observation, taking the form

$$L(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} L(\boldsymbol{\theta}; x^{(i)}, y^{(i)}), \text{ with } \nabla L(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta L(\boldsymbol{\theta}; x^{(i)}, y^{(i)})$$

Computing ∇L is very computationally expensive for large datasets, since a calculation is required for every observation. A more efficient method is **stochastic gradient descent**, which takes a random sample at each iteration to compute an unbiased estimate of the gradient:

Set starting value $\boldsymbol{\theta}_0$ and step size $\eta > 0$. Iterate the following for $t \geq 1$:

1. Take random sample b of size m from the dataset
2. Estimate gradient: $\boldsymbol{g}_t \leftarrow \frac{1}{m} \sum_{x,y \in b} \nabla_\theta L(\boldsymbol{\theta}_{t-1}; x, y)$
3. Update parameters: $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \, \boldsymbol{g}_t$

By setting the loss function to be the $\frac{1}{N}$ times the negative log likelihood, we can use SGD for maximum likelihood estimation, as seen in Figure 1(b).

The erratic behaviour of the walk can make it easier to escape suboptimal local minima and saddle points when the loss function is non-convex. Note that this is the case for skip-gram - applying a rotation to all of the vectors will preserve dot products and hence the log likelihood, so there is no unique minimum.

Training Word Embeddings

The dataset was the text of all public domain **Sherlock Holmes** novels and short stories, available on Project Gutenberg. To simplify the task, all punctuation was removed, splitting contractions like "I've" and "he's" into two words instead of treating them as distinct words. Capitalisation was also removed and numbers replaced with a # token.

In total the dataset had around 580 000 words, with a vocabulary size of 17 500. Since it was a relatively small dataset, a small embedding dimension $d = 100$ was chosen, with $k = 5$ and $l = 10$. Any words with less than 10 occurrences in the dataset were discarded.

Since the loss function was non-convex, results varied a lot depending on the random initialisation and training was repeated several times with different random seeds.

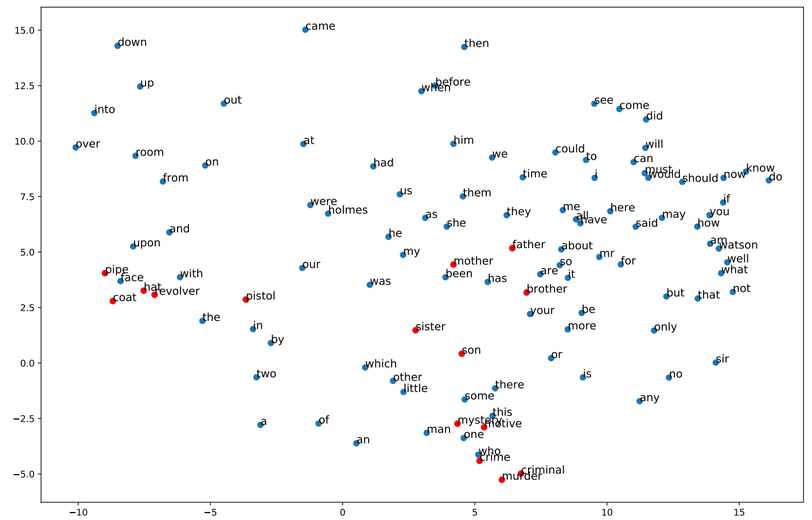


Figure 2: The embeddings for 110 common words from Sherlock Holmes, normalised and then projected into 2D with principal component analysis. Some clusters with high cosine similarity are highlighted in red.

Conclusions and Future Work

The model did well at grouping words with similar meanings together, using the **cosine similarity** - the cosine of the angle between vectors - to measure vector similarity. Some examples were: **you** was most similar to **yourself**, **brother** was most similar to **son**, and **crime** was most similar to **murder**. This worked best for more common words - due to the small amounts of training data for rarer words, their embeddings did not move far from their random initialisation.

An embedding layer is typically included as the first layer in **neural network** models for language, which is what this project will focus on next. Embeddings can be trained as part of the network or trained separately to cut down on training time - skip-gram with negative sampling is an efficient way of doing this.

References

D. Jurafsky and J. H. Martin.
Speech and Language Processing. Prentice Hall, 2020 3rd edition draft. https://web.stanford.edu/~jurafsky/slp3

T. Mikolov, K. Chen, G. Corrado, and Jeffrey Dean.
Efficient Estimation of Word Representations in Vector Space. https://arxiv.org/abs/1301.3781

T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and Jeffrey Dean.
Distributed Representations of Words and Phrases and their Compositionality. https://arxiv.org/abs/1310.4546