Word Embeddings for Language Modelling

Connie Trojan,
Supervisor: Jonathan Cumming

24th February 2021

Background - Language Models

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, grammar checking, and translation.

Background - Language Models

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, grammar checking, and translation.

A key issue in language modelling is the large **vocabulary size** - we must model the meanings and interactions of millions of unique words. It would be useful to exploit similarities between words.

Background - Word Embeddings

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.

Background - Word Embeddings

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.

Background - Word Embeddings

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.

We will focus on **skip-gram with negative sampling**, which obtains embeddings by assuming that words with similar meanings appear in similar contexts.

Skip-Gram with negative Sampling

- Embeddings will be obtained by training a model to distinguish between real context words and random ones

Skip-Gram with negative Sampling

- Embeddings will be obtained by training a model to distinguish between real context words and random ones
- A real **context word** or **positive sample** for word w_i will be defined as any word that appears within a window of length l around w_i anywhere in the training text

Skip-Gram with negative Sampling

- Embeddings will be obtained by training a model to distinguish between real context words and random ones
- A real **context word** or **positive sample** for word w_i will be defined as any word that appears within a window of length l around w_i anywhere in the training text
- We randomly generate **negative samples** from the rest of the vocabulary. For each positive sample (w, w^+), we generate k negative samples (w, w_j^-).

Example sentence: “*Sing, and the hills will answer.*”

If we set $l = 2$ the observed context words for “hills” are “the” and “will”. We make the assumption that the order of the context words is not important, representing these target-context pairs as the **skip-grams** (*hills*,*the*) and (*hills*,*will*).

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** $\boldsymbol{c}_{w_j}$. The dot product $\boldsymbol{v}_{w_i} \cdot \boldsymbol{c}_{w_j}$ will be used to determine similarity.

The probability that w_j is a true context word for w_i is modelled by passing this into the **logistic** function, σ:

$$\mathbb{P}(w_j | w_i) = \sigma(\boldsymbol{v}_{w_i} \cdot \boldsymbol{c}_{w_j}) = \frac{1}{1 + e^{-\boldsymbol{v}_{w_i} \cdot \boldsymbol{c}_{w_j}}}$$

σ takes inputs in $(-\infty, \infty)$ and outputs values in $(0, 1)$, so this is a valid probability. The probability of w_j not being a context word for w_i is simply:

$$1 - \sigma(\boldsymbol{v}_{w_i} \cdot \boldsymbol{c}_{w_j}) = \sigma(-\boldsymbol{v}_{w_i} \cdot \boldsymbol{c}_{w_j})$$

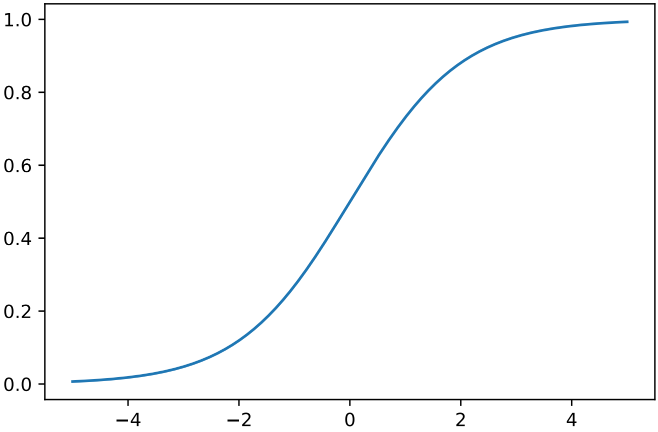


Figure: The logistic function, $y = \frac{1}{1+e^{-x}}$.

Making the assumption that context words occur independently of each other, the likelihood for one positive observation (w, w^+) and the corresponding k negative (w, w_j^-) is:

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

The total **log likelihood** for a dataset is obtained by summing the log of this expression over each positive sample (w, w^+) in the set of observed context words C_w , for each word in the vocabulary V:

$$\sum_{w\in V}\;\sum_{w^+\in C_w}\left(\log\sigma(\boldsymbol{c}_{w^+}\cdot\boldsymbol{v}_w)+\sum_{j=1}^{k}\log\sigma(-\boldsymbol{c}_{w_j^-}\cdot\boldsymbol{v}_w)\right)$$

Note that this has **no unique maximum** (rotating the vectors preserves the log likelihood), but can still be optimised numerically.

Gradient Descent

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.

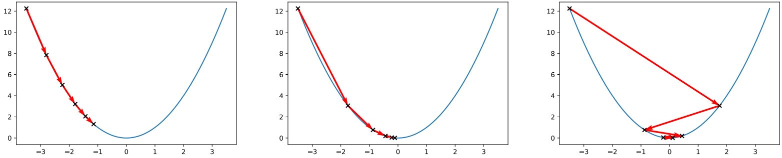


Figure: Using gradient descent to find the minimum of $y = x^2$.

For **likelihood maximisation**, we set the loss function to be $-\frac{1}{N}$ times the log likelihood:

$$L(\boldsymbol{\theta}) = -\frac{1}{N}\ell(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N}\ell(\boldsymbol{\theta}; x^{(i)}, y^{(i)})$$

Then the gradient is:

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

Computing ∇L is very computationally expensive for large datasets.

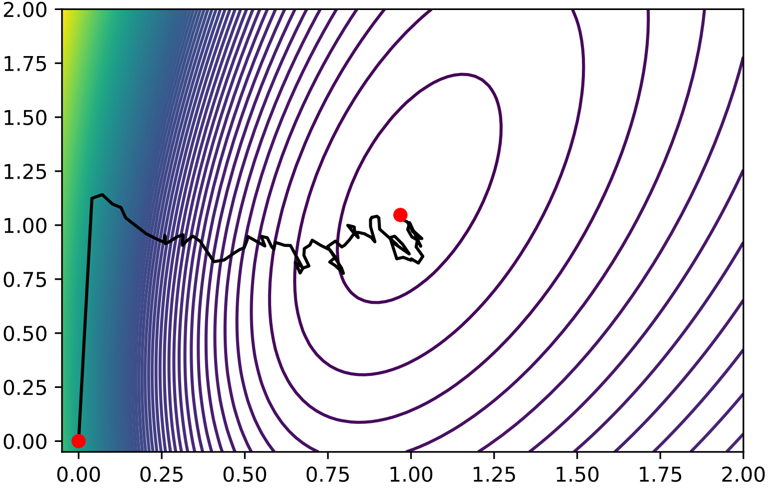
Stochastic Gradient Descent

An efficient method is **stochastic gradient descent** (SGD), which uses a random sample at each iteration to estimate the gradient:

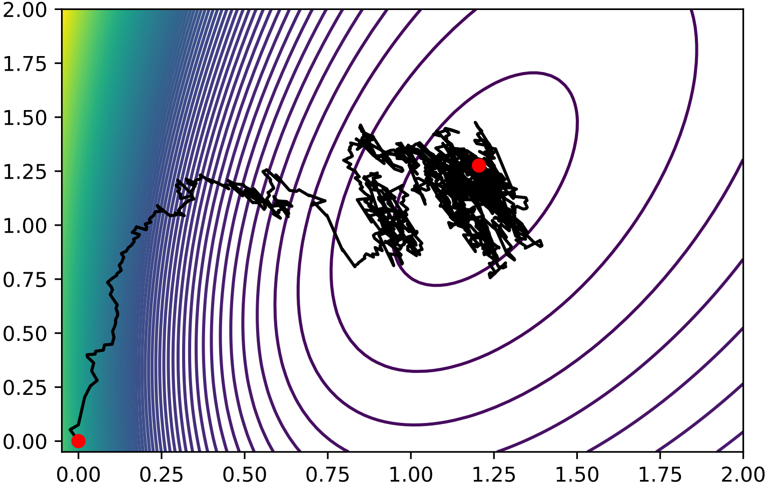
Algorithm: SGD

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 the gradient at $\boldsymbol{\theta}_t$ by :
$$\boldsymbol{g}_t \leftarrow \frac{1}{m} \sum_{x,y \in b} \nabla_\theta L(\boldsymbol{\theta}_{t-1}; x, y)$$
3. Update parameters: $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \, \boldsymbol{g_t}$


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

(a) $m = 50$


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00

(b) $m = 1$

Sherlock Holmes

- The dataset was the text of all public domain Sherlock Holmes novels and short stories

Sherlock Holmes

- The dataset was the text of all public domain Sherlock Holmes novels and short stories
- All **punctuation** was removed, splitting contractions like "I've" and "he's" into two words

Sherlock Holmes

- The dataset was the text of all public domain Sherlock Holmes novels and short stories
- All **punctuation** was removed, splitting contractions like "I've" and "he's" into two words
- In total the dataset had around **580 000** words, with a vocabulary size of **17 500**

Sherlock Holmes

- The dataset was the text of all public domain Sherlock Holmes novels and short stories
- All **punctuation** was removed, splitting contractions like "I've" and "he's" into two words
- In total the dataset had around **580 000** words, with a vocabulary size of **17 500**
- A small embedding dimension $d = 100$ was chosen, with $k = 5$ and $l = 10$

Sherlock Holmes

- The dataset was the text of all public domain Sherlock Holmes novels and short stories
- All **punctuation** was removed, splitting contractions like "I've" and "he's" into two words
- In total the dataset had around **580 000** words, with a vocabulary size of **17 500**
- A small embedding dimension $d = 100$ was chosen, with $k = 5$ and $l = 10$
- Fitted model several times with different random seeds

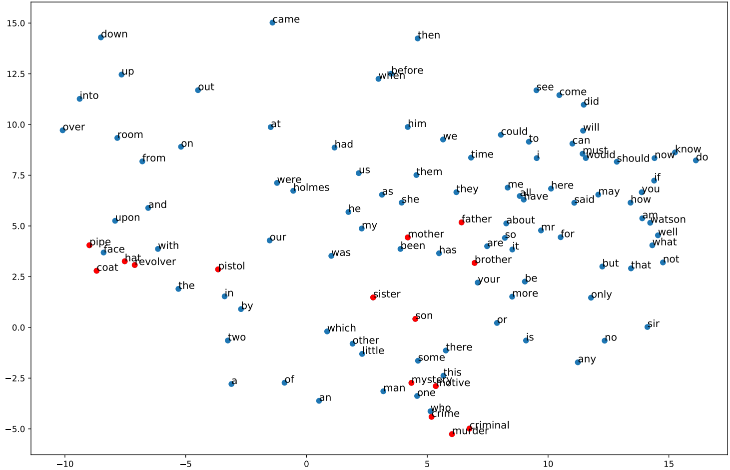


Figure: The embeddings for 110 common words from Sherlock Holmes, normalised and projected into 2D with principal component analysis

Conclusions

- The model did well at grouping words with similar meanings together
- e.g. **you** and **yourself**, **brother** and **son**, and **crime** and **murder**
- This worked best for more common words - embeddings for rarer words were still quite random
- More training data would be required to unlock the full potential of word embeddings - state-of-the-art word vectors are trained on datasets with over a billion words

Future Work

- 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

Future Work

- 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

Future Work

- 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
- It is also common to use a set of **pre-trained** word embeddings out of the box if the dataset is small or computational power is limited

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

Any Questions?