Chapter IR:III

III. Retrieval Models

- ❑ Overview of Retrieval Models
- ❑ Boolean Retrieval
- ❑ Vector Space Model
- ❑ Binary Independence Model
- ❑ Okapi BM25
- ❑ Divergence From Randomness
- ❑ Latent Semantic Indexing
- ❑ Explicit Semantic Analysis
- ❑ Language Models
- ❑ Combining Evidence
- ❑ Learning to Rank

Language Models

Background

Language models in general include methods to represent the syntactical structures of languages to study them, and to solve natural language processing tasks.

A key goal of modeling a language is to solve the membership problem:
Given a string and a language, decide whether the string belongs to the language.

Two complementary approaches have been pursued:

- Formal languages
  Theoretical approach with an explicit grammar specification and applications in comparably small, controlled languages (e.g., query languages, programming languages).
- Statistical language models
  Probabilistic approach where grammar is captured only implicitly by analyzing large text collections. Can be applied in less controlled situations.

Important applications of statistical language models:

- Part-of-speech tagging
- Machine translation
- Speech and handwriting recognition
- Information retrieval

Language Models
Basics: Grammar

- Alphabet Σ.

  An alphabet Σ is a non-empty set of signs or symbols.

- Word w.

  A word w is a finite sequence of symbols from Σ. The length of a word $|w|$ is the number of symbols it is made of.

  ε denotes the empty word; it is the only word of length 0.
  Σ^* denotes the set of all words over Σ.

- Language L.

  A language L is a set of words over an alphabet Σ.

- Grammar G.

  A grammar G is a calculus to define a language—and a set of rules by which words can be derived. The language corresponding to G contains all words that can be generated using its rules.

Language Models

Basics: Grammar

- Alphabet Σ.

  An alphabet Σ is a non-empty set of signs or symbols.

- Word w.

  A word w is a finite sequence of symbols from Σ. The length of a word $|w|$ is the number of symbols it is made of.

  ε denotes the empty word; it is the only word of length 0.
  Σ^* denotes the set of all words over Σ.

- Language L.

  A language L is a set of words over an alphabet Σ.

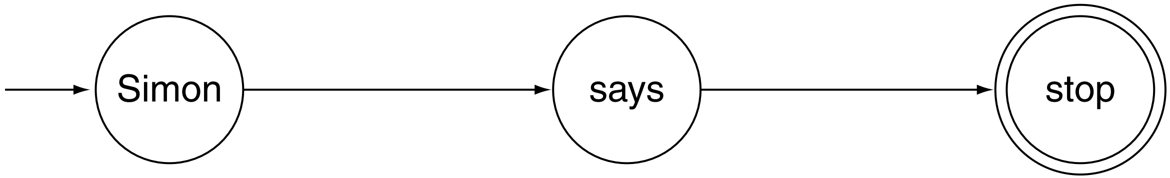
- Grammar G.

  A grammar G is a calculus to define a language—and a set of rules by which words can be derived. The language corresponding to G contains all words that can be generated using its rules.

Language Models

Example: Deterministic Language Model

Grammar G_1 as deterministic finite automaton:



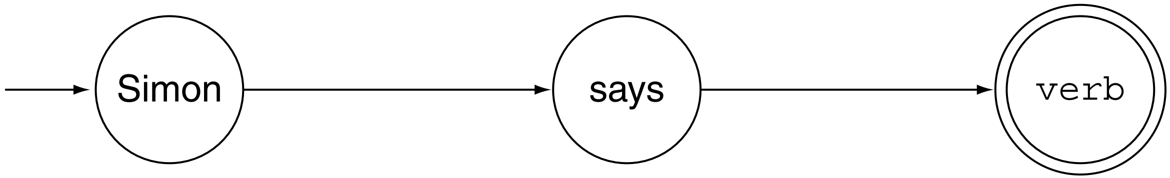
Generated language:

- $L(G_1) = \{\texttt{Simon says stop}\}$
- How to allow for other "Simon says" sentences?

Language Models

Example: Deterministic Language Model

Grammar G_2 as deterministic finite automaton:



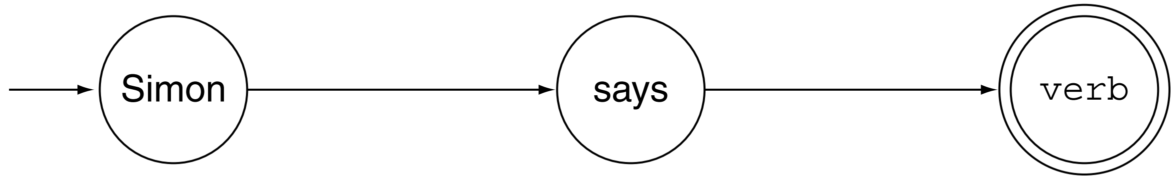
Generated language:

- Let $\texttt{verb} = \{\texttt{jump}, \texttt{run}, ...\}$ denote the set of all verbs.
- $L(G_2)$ contains `Simon says` sentences, e.g.:
  `Simon says jump, Simon says run,` ...
- $|L(G_2)| = |\texttt{verb}|$

Language Models

Example: Deterministic Language Model

Grammar G_2 as deterministic finite automaton:



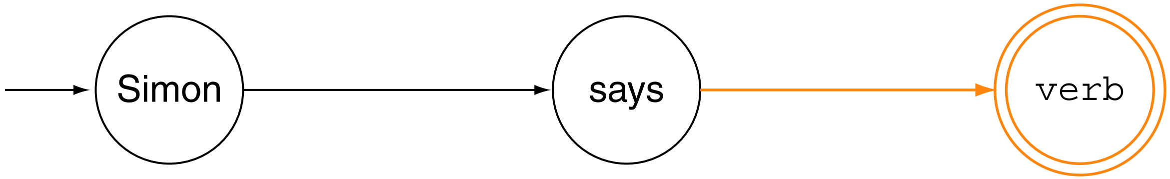
Generated language:

- Let $\texttt{verb} = \{\texttt{jump}, \texttt{run}, ...\}$ denote the set of all verbs.
- $L(G_2)$ contains `Simon says` sentences, e.g.:
  `Simon says jump`, `Simon says run`, ...
- $|L(G_2)| = |\texttt{verb}|$
- Is the sentence `Simon says science` member of $L(G_2)$?

Language Models

Example: Deterministic Language Model

Grammar G_2 as deterministic finite automaton:



Generated language:

- Let $\texttt{verb} = \{\texttt{jump}, \texttt{run}, ...\}$ denote the set of all verbs.
- $L(G_2)$ contains `Simon says` sentences, e.g.:
  `Simon says jump`, `Simon says run`, ...
- $|L(G_2)| = |\texttt{verb}|$
- Is the sentence `Simon says science` member of $L(G_2)$?

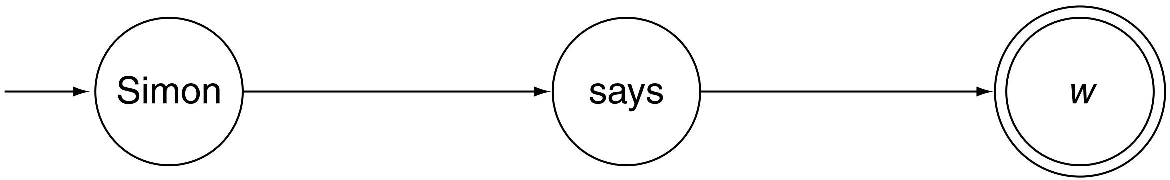
  I'm gonna have to science the shit out of this.
  Mark Watney in The Martian

➜ Allowing every word would still result in exceedingly unlikely sentences.

Language Models

Example: Statistical Language Model

Grammar G_3 as probabilistic automaton:



$w \in T$	$P(w)$
`jump`	0.05
`run`	0.03
⋮	⋮
`science`	0.002
⋮	⋮

where w is a random variable over a vocabulary T.

Generated language:

- $L(G_3)$ contains every three-word sentence starting with `Simon says` followed by a word w from T with probability $P(w) > \tau$ where τ is a threshold.
- Put another way, G_3 maps every sentence s that can be formed over its vocabulary Σ to a probability $P(s)$ so that

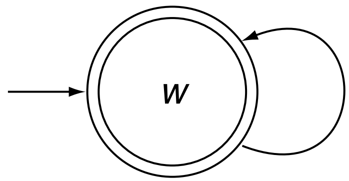
$$\sum_{s \in \Sigma^*} P(s) = 1$$

In general, probabilistic automata can be used to generate arbitrary documents.

Language Models

Example: Statistical Language Model

Grammar G_4 as probabilistic automaton:



$w \in T$	$P(w)$
⊥	0.2
the	0.2
a	0.1
that	0.04
says	0.03

$w \in T$	$P(w)$
likes	0.02
Simon	0.01
Mark	0.01
science	0.002
⋮	⋮

where w is a random variable over a vocabulary T.

Generated language:

- $\perp$ denotes the probability that the automaton stops.
- $L(G_4)$ contains all sentences that can be formed over the vocabulary T, assigning a membership probability to each one, e.g.:

  $s =$ `Simon says that Mark likes science` $\perp$

  $P(s) = 0.01 \cdot 0.03 \cdot 0.04 \cdot 0.01 \cdot 0.02 \cdot 0.01 \cdot 0.2 = 0.0000000000048 = 4.8 \cdot 10^{-12}$

- Suppose every document were generated by its own language model $\mathbf{d}$.

➜ Given a query q, $P(\mathbf{d}_1 \mid q) > P(\mathbf{d}_2 \mid q)$ may indicate that d_1 is more relevant to q than d_2.

Language Models

Retrieval Model $\mathcal{R} = \langle \mathbf{D}, \mathbf{Q}, \rho \rangle$ [Generic Model] [Boolean] [VSM] [BIM] [BM25] [LSI] [ESA] [LM]

Document representations $\mathbf{D}$.

- $T = \{t_1, \ldots, t_m\}$ is the set of m index terms (stemmed words).
- $p(t \mid d)$ is the probability of generating t given d.
- $\mathbf{d} = \{(t_1, p(t_1 \mid d), \ldots, (t_m, p(t_m \mid d))\}$ is a probability distribution over T.

Query representations $\mathbf{Q}$.

- $\mathbf{q} = (t_1, \ldots, t_{|q|})$, where $t_i \in T$, is a sequence of index terms.

Relevance function ρ.

- $\rho(d, q) = P(\mathbf{d} \mid \mathbf{q})$, the query likelihood model.
- R^+ is a set of documents relevant to q obtained via relevance feedback.
- $\mathbf{R}^+ = \{(t_1, p(t_1 \mid R^+), \ldots, (t_m, p(t_m \mid R^+))\}$ is a probability distribution over T.
- $\rho(d, q) = \varphi_{KL}(\mathbf{d}, \mathbf{R}^+)$, the relevance model.

Language Models

Retrieval Model $\mathcal{R} = \langle \mathbf{D}, \mathbf{Q}, \rho \rangle$ [Generic Model] [Boolean] [VSM] [BIM] [BM25] [LSI] [ESA] [LM]

Document representations $\mathbf{D}$.

- $T = \{t_1, \ldots, t_m\}$ is the set of m index terms (stemmed words).
- $p(t \mid d)$ is the probability of generating t given d.
- $\mathbf{d} = \{(t_1, p(t_1 \mid d), \ldots, (t_m, p(t_m \mid d))\}$ is a probability distribution over T.

Query representations $\mathbf{Q}$.

- $\mathbf{q} = (t_1, \ldots, t_{|q|})$, where $t_i \in T$, is a sequence of index terms.

Relevance function ρ.

- $\rho(d, q) = P(\mathbf{d} \mid \mathbf{q})$, the query likelihood model.
- R^+ is a set of documents relevant to q obtained via relevance feedback.
- $\mathbf{R}^+ = \{(t_1, p(t_1 \mid R^+), \ldots, (t_m, p(t_m \mid R^+))\}$ is a probability distribution over T.
- $\rho(d, q) = \varphi_{KL}(\mathbf{d}, \mathbf{R}^+)$, the relevance model.

Language Models

Relevance Function ρ: Derivation

Let $\mathbf{d}$ denote a language model for document d, and $\mathbf{q}$ the sequence of query terms from query q. Then the query likelihood model is derived as follows:

$$P(\mathbf{d} \mid \mathbf{q}) = \frac{P(\mathbf{q} \mid \mathbf{d}) \cdot P(\mathbf{d})}{P(\mathbf{q})} \qquad (1)$$

(1) Application of Bayes' rule.

Language Models

Relevance Function ρ: Derivation

Let $\mathbf{d}$ denote a language model for document d, and $\mathbf{q}$ the sequence of query terms from query q. Then the query likelihood model is derived as follows:

$$P(\mathbf{d} \mid \mathbf{q}) = \frac{P(\mathbf{q} \mid \mathbf{d}) \cdot P(\mathbf{d})}{P(\mathbf{q})} \quad (1)$$

$$\stackrel{\text{rank}}{=} P(\mathbf{q} \mid \mathbf{d}) \cdot P(\mathbf{d}) \quad (2)$$

(1) Application of Bayes' rule.

(2) Rank-preserving omission of $P(\mathbf{q})$; it does not depend on $\mathbf{d}$.

Language Models

Relevance Function ρ: Derivation

Let $\mathbf{d}$ denote a language model for document d, and $\mathbf{q}$ the sequence of query terms from query q. Then the query likelihood model is derived as follows:

$$P(\mathbf{d} \mid \mathbf{q}) = \frac{P(\mathbf{q} \mid \mathbf{d}) \cdot P(\mathbf{d})}{P(\mathbf{q})} \quad (1)$$

$$\overset{\text{rank}}{=} P(\mathbf{q} \mid \mathbf{d}) \cdot P(\mathbf{d}) \quad (2)$$

$$= P(\mathbf{q} \mid \mathbf{d}) \quad (3)$$

(1) Application of Bayes' rule.

(2) Rank-preserving omission of $P(\mathbf{q})$; it does not depend on $\mathbf{d}$.

(3) Assume $P(\mathbf{d})$ is uniform for all $d \in D$, thereby canceling its influence.
This assumption is not required; as a prior, $P(\mathbf{d})$ can be used as a weight of relative importance of d (e.g., PageRank, quality, etc.).

Language Models

Relevance Function ρ: Derivation

Given a language model $\mathbf{d}$ of document d and a sequence $\mathbf{q}$ of the terms in query q, compute the probability that $\mathbf{q}$ has been generated by $\mathbf{d}$.

$$P(\mathbf{q} \mid \mathbf{d}) \quad = \quad P(t_1, \ldots, t_{|q|} \mid \mathbf{d}) \tag{4}$$

(4) Inflating $\mathbf{q}$.

Language Models

Relevance Function ρ: Derivation

Given a language model $\mathbf{d}$ of document d and a sequence $\mathbf{q}$ of the terms in query q, compute the probability that $\mathbf{q}$ has been generated by $\mathbf{d}$.

$$P(\mathbf{q} \mid \mathbf{d}) = P(t_1, \ldots, t_{|q|} \mid \mathbf{d}) \tag{4}$$

$$= \prod_{i=1}^{|q|} P(t_i \mid \mathbf{d}) \tag{5}$$

(4) Inflating $\mathbf{q}$.

(5) Assuming independence between terms.

Language Models

Relevance Function ρ: Derivation

Given a language model $\mathbf{d}$ of document d and a sequence $\mathbf{q}$ of the terms in query q, compute the probability that $\mathbf{q}$ has been generated by $\mathbf{d}$.

$$P(\mathbf{q} \mid \mathbf{d}) = P(t_1, \ldots, t_{|q|} \mid \mathbf{d}) \quad (4)$$

$$\overset{\text{rank}}{=} \sum_{i=1}^{|q|} \log P(t_i \mid \mathbf{d}) \quad (5)$$

$$= \prod_{t \in \mathbf{q}} P(t \mid \mathbf{d})^{\mathit{tf}(t,q)} \quad (6)$$

(4) Inflating $\mathbf{q}$.

(5) Assuming independence between terms.
Rank-preserving logarithmization to handle small probabilities.

(6) Combine duplicate occurrences of term t in query q.
This corresponds to the multinomial distribution, albeit omitting its factor $|d|/\prod_{t \in q} \mathit{tf}(t,q)$, which counts the permutations of q's terms but is constant for q.

Language Models

Relevance Function ρ: Estimation

Let t denote a term from the set of index terms T of document collection D. The construction of a language model $\mathbf{d}$ to represent document d is done as follows.

$$P(t \mid \mathbf{d}) = \frac{\mathit{tf}(t,d)}{|d|}, \qquad \text{where} \quad \sum_{t \in T} P(t \mid \mathbf{d}) = 1 \tag{7}$$

(7) Maximum likelihood estimation of t's probability under the assumed language model $\mathbf{d}$ for document d's topic, given the observed sample d.
Problem: $P(t \mid \mathbf{d}) = 0$ for $t \notin d$, causing $P(\mathbf{q} \mid \mathbf{d}) = 0$ if $t \in q$.

Language Models

Relevance Function ρ: Estimation

Let t denote a term from the set of index terms T of document collection D. The construction of a language model $\mathbf{d}$ to represent document d is done as follows.

$$P(t \mid \mathbf{d}) = \frac{\mathit{tf}(t,d)}{|d|}, \qquad \text{where} \quad \sum_{t \in T} P(t \mid \mathbf{d}) = 1 \qquad (7)$$

$$P(t \mid \mathbf{D}) = \frac{\sum_{d \in D} \mathit{tf}(t,d)}{\sum_{d \in D} |d|}, \qquad \text{where} \quad \sum_{t \in T} P(t \mid \mathbf{D}) = 1 \qquad (8)$$

(7) Maximum likelihood estimation of t's probability under the assumed language model $\mathbf{d}$ for document d's topic, given the observed sample d.
Problem: $P(t \mid \mathbf{d}) = 0$ for $t \notin d$, causing $P(\mathbf{q} \mid \mathbf{d}) = 0$ if $t \in q$.

(8) Maximum likelihood estimation of t's probability in a language model $\mathbf{D}$ for D.

Language Models

Relevance Function ρ: Estimation

Let t denote a term from the set of index terms T of document collection D. The construction of a language model $\mathbf{d}$ to represent document d is done as follows.

$$P(t \mid \mathbf{d}) = \frac{\mathit{tf}(t,d)}{|d|}, \quad \text{where} \quad \sum_{t \in T} P(t \mid \mathbf{d}) = 1 \tag{7}$$

$$P(t \mid \mathbf{D}) = \frac{\sum_{d \in D} \mathit{tf}(t,d)}{\sum_{d \in D} |d|}, \quad \text{where} \quad \sum_{t \in T} P(t \mid \mathbf{D}) = 1 \tag{8}$$

$$P(t \mid \mathbf{d})' = (1-\lambda) \cdot P(t \mid \mathbf{d}) \; + \; \lambda \cdot P(t \mid \mathbf{D}) \tag{9}$$

(7) Maximum likelihood estimation of t's probability under the assumed language model $\mathbf{d}$ for document d's topic, given the observed sample d.
Problem: $P(t \mid \mathbf{d}) = 0$ for $t \notin d$, causing $P(\mathbf{q} \mid \mathbf{d}) = 0$ if $t \in q$.

(8) Maximum likelihood estimation of t's probability in a language model $\mathbf{D}$ for D.

(9) Jelinek-Mercer smoothing: linear interpolation of language models $\mathbf{d}$ and $\mathbf{D}$.

Language Models

Relevance Function ρ: Estimation

Taking into account the length of a document d yields an alternative smoothing method.

$$P(t \mid \mathbf{d})' \quad = \quad (1-\lambda) \cdot P(t \mid \mathbf{d}) \; + \; \lambda \cdot P(t \mid \mathbf{D}) \tag{9}$$

(9) Jelinek-Mercer smoothing: linear interpolation of language models $\mathbf{d}$ and $\mathbf{D}$.

Language Models

Relevance Function ρ: Estimation

Taking into account the length of a document d yields an alternative smoothing method.

$$P(t \mid \mathbf{d})' = (1-\lambda) \cdot P(t \mid \mathbf{d}) + \lambda \cdot P(t \mid \mathbf{D}) \quad (9)$$

$$\lambda = \frac{\alpha}{|d| + \alpha} \quad (10)$$

(9) Jelinek-Mercer smoothing: linear interpolation of language models $\mathbf{d}$ and $\mathbf{D}$.

(10) Dirichlet smoothing: adjust λ with respect to the length of document d. The longer a document d, the more trustworthy its language model $\mathbf{d}$ becomes.

Language Models

Relevance Function ρ: Estimation

Taking into account the length of a document d yields an alternative smoothing method.

$$P(t \mid \mathbf{d})' = (1-\lambda) \cdot P(t \mid \mathbf{d}) \; + \; \lambda \cdot P(t \mid \mathbf{D}) \tag{9}$$

$$\lambda = \frac{\alpha}{|d| + \alpha} \tag{10}$$

$$P(t \mid \mathbf{d})'' = \frac{\textit{tf}(t,d) + \alpha \cdot P(t \mid \mathbf{D})}{|d| + \alpha} \tag{11}$$

(9) Jelinek-Mercer smoothing: linear interpolation of language models $\mathbf{d}$ and $\mathbf{D}$.

(10) Dirichlet smoothing: adjust λ with respect to the length of document d. The longer a document d, the more trustworthy its language model $\mathbf{d}$ becomes.

(11) Substitution of λ in $P(t \mid \mathbf{d})'$.

Language Models

Relevance Function ρ: Example

Let $q = \texttt{president lincoln}$ and let $d_1 \in D$ be a document from a collection D.

Assumptions:

- *tf*$(\texttt{president}, d_1) = 15$ and $\sum_{d \in D}$ *tf*$(\texttt{president}, d) = 160,000$
- *tf*$(\texttt{lincoln}, d_1) = 25$ and $\sum_{d \in D}$ *tf*$(\texttt{lincoln}, d) = 2,400$
- $|d_1| = 1,800$ and $|D| = 500,000$ at $|d|_{\text{avg}} = 2,000$, yielding 10^9 terms.
- $\alpha = |d|_{\text{avg}} = 2,000$

$$
\begin{aligned}
\rho(\mathbf{d}_1, \mathbf{q}) &= \log \frac{15 + 2000 \cdot (1.6 \cdot 10^5 / 10^9)}{1800 + 2000} + \log \frac{25 + 2000 \cdot (2400/10^9)}{1800 + 2000} \\
&= \log(15.32/3800) + \log(25.005/3800) \\
&= -5.51 + -5.02 \\
&= -10.53
\end{aligned}
$$

Logarithmization yields negative relevance scores; recall that only the ranking among documents is important, not the scores themselves.

Language Models

Relevance Function ρ: Example

Let $q = \texttt{president lincoln}$ and let $d_1 \in D$ be a document from a collection D.

Assumptions:

- $\mathit{tf}(\texttt{president}, d_1) = 15$ and $\sum_{d \in D} \mathit{tf}(\texttt{president}, d) = 160,000$
- $\mathit{tf}(\texttt{lincoln}, d_1) = 25$ and $\sum_{d \in D} \mathit{tf}(\texttt{lincoln}, d) = 2,400$
- $|d_1| = 1,800$ and $|D| = 500,000$ at $|d|_{\text{avg}} = 2,000$, yielding 10^9 terms.
- $\alpha = |d|_{\text{avg}} = 2,000$

D	president	lincoln	LM	#	BM25	#
d_1	15	25	-10.53	1	20.66	1
d_2	15	1	-13.75	3	12.74	4
d_3	15	0	-19.05	5	5.00	5
d_4	1	25	-12.99	2	18.20	2
d_5	0	25	-14.40	4	15.66	3

Language Models

Relevance Function ρ: Summary

$$\rho(\mathbf{d},\mathbf{q}) \quad = \quad P(\mathbf{d} \mid \mathbf{q}) \quad \propto \quad P(\mathbf{d}) \cdot \prod_{i=1}^{|q|} \frac{\textit{tf}(t_i,d) + \alpha \cdot \frac{\sum_{d\in D} \textit{tf}(t_i,d)}{\sum_{d \in D} |d|}}{|d| + \alpha}$$

Assumptions:

1. The user has a mental model of the desired document and generates the query from that model.
2. The equation represents a probability estimate that the document the user had in mind was in fact this one.
3. Independence of word occurrence in documents.
4. Terms not in query q are equally likely to occur in relevant and irrelevant documents.
5. The prior $P(\mathbf{d})$ may be chosen uniform for all documents, or to boost more important documents.

Language Models

Discussion

Advantages:

- Mathematically precise, conceptually simple, computationally tractable, and intuitively appealing
- Competitive retrieval performance

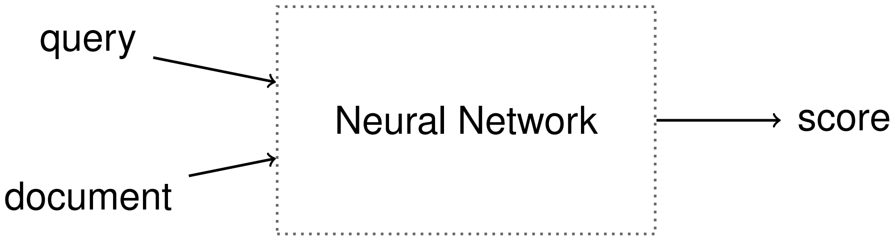
Disadvantages:

- Requires extensive tuning
- Assumption of equivalence between document and information need representation is unrealistic
- Difficult to represent the fact that a query is just one of many possible queries to describe a particular need

Word Embeddings

Overview

Goal:

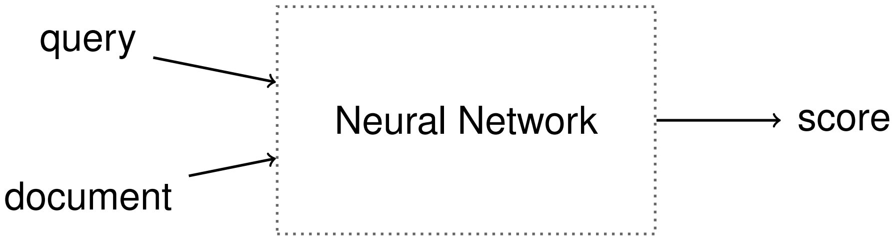


Problem: How do we represent text so we can feed it to the neural network?

Word Embeddings

Overview

Goal:



Problem: How do we represent text so we can feed it to the neural network?

Solution: Turn words into numbers.

Word Embeddings

Representing Words

```
apples are great
```

Word Embeddings

Representing Words

`apples are great`

Assign each word a random value.

- apples ➜ 6.3
- are ➜ -3.5
- great ➜ 4.2

Word Embeddings

Representing Words

```
apples are great

apples are awesome
```

Assign each word a random value.

- apples ➜ 6.3
- are ➜ -3.5
- great ➜ 4.2
- awesome ➜ -32.1

Word Embeddings
Representing Words

`apples are great`

`apples are awesome`

Assign each word a random value.

- apples ➜ 6.3
- are ➜ -3.5
- great ➜ 4.2
- awesome ➜ -32.1

Problems:

- `great` and `awesome` mean similar things and used in similar ways.
- They are likely to have very different values.
- Bad for neural networks, requiring more complexity and training.

Word Embeddings

Developing a Better Representation

How can we let similar words have similar values?

➜ Learning how to use one word helps use the other at the same time.

Word Embeddings

Developing a Better Representation

How can we let similar words have similar values?

➜ Learning how to use one word helps use the other at the same time.

Words can be used in many contexts, pluralised, and so on.

➜ Assign each word multiple values for different contexts.

Word Embeddings

Developing a Better Representation

How can we let similar words have similar values?

➜ Learning how to use one word helps use the other at the same time.

Words can be used in many contexts, pluralised, and so on.

➜ Assign each word multiple values for different contexts.

How to decide which words are similar? How to learn multiple values?

➜ Neural network + clever training.

Word Embeddings

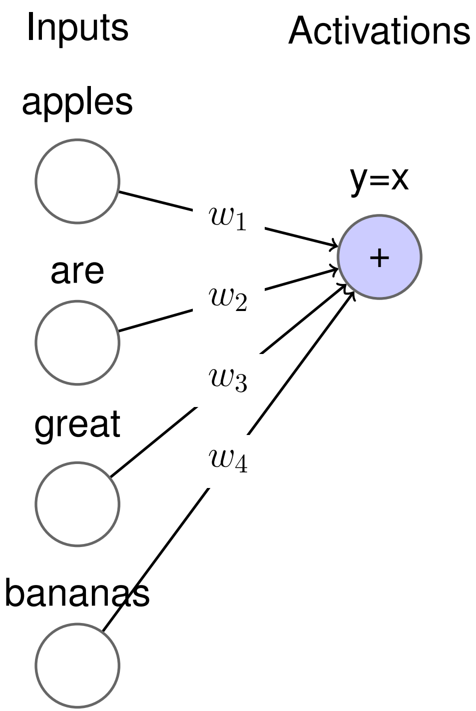
Training a Neural Network

Training data: `apples are great, bananas are great.`

Word Embeddings

Training a Neural Network

Training data: `apples are great`, `bananas are great`.

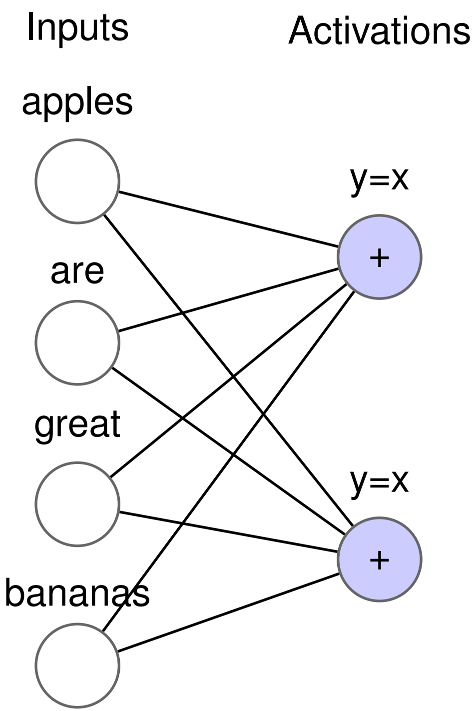


- Four unique inputs, each corresponding to a word.
- Linear activation function does nothing, just a place to do addition.
- Weights randomly initialised and optimised with backpropagation.

Word Embeddings

Training a Neural Network

Training data: `apples are great, bananas are great.`

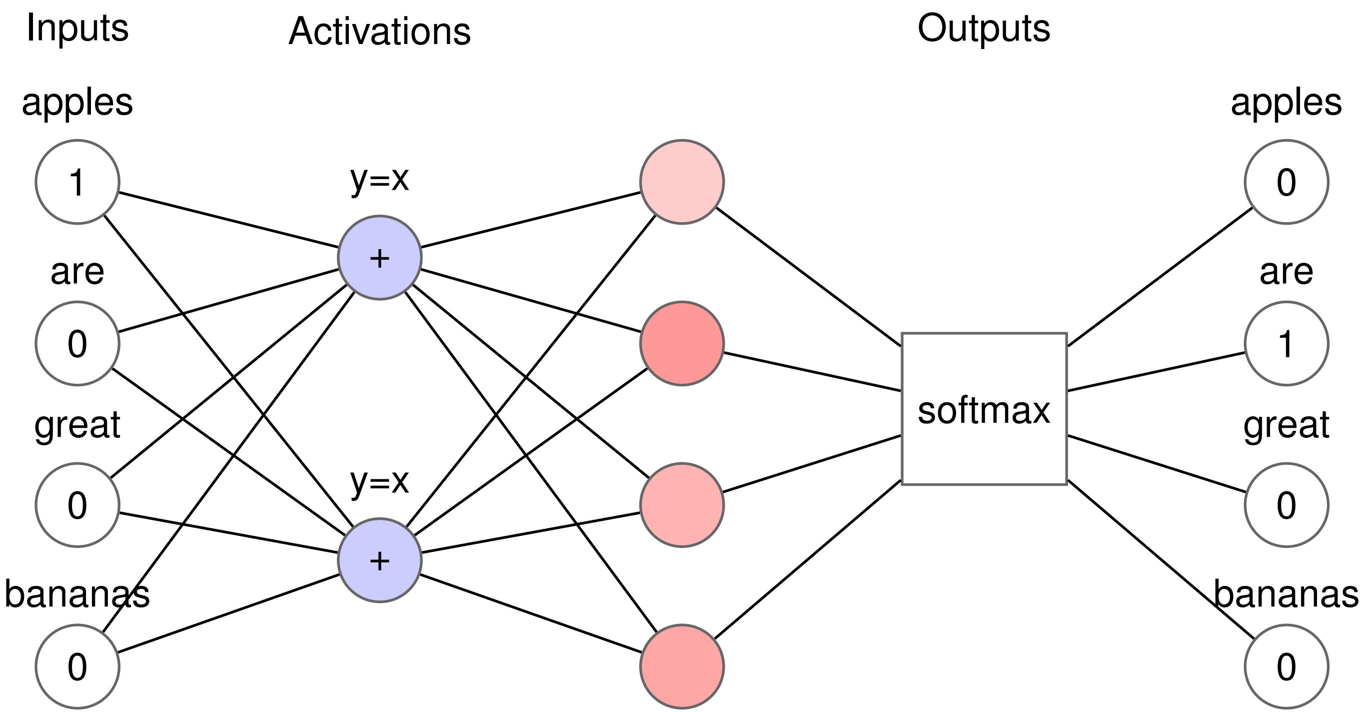


- To represent words with multiple values, add additional activation functions.
- Each activation function is associated with another weight for each word.

Word Embeddings

Training a Neural Network

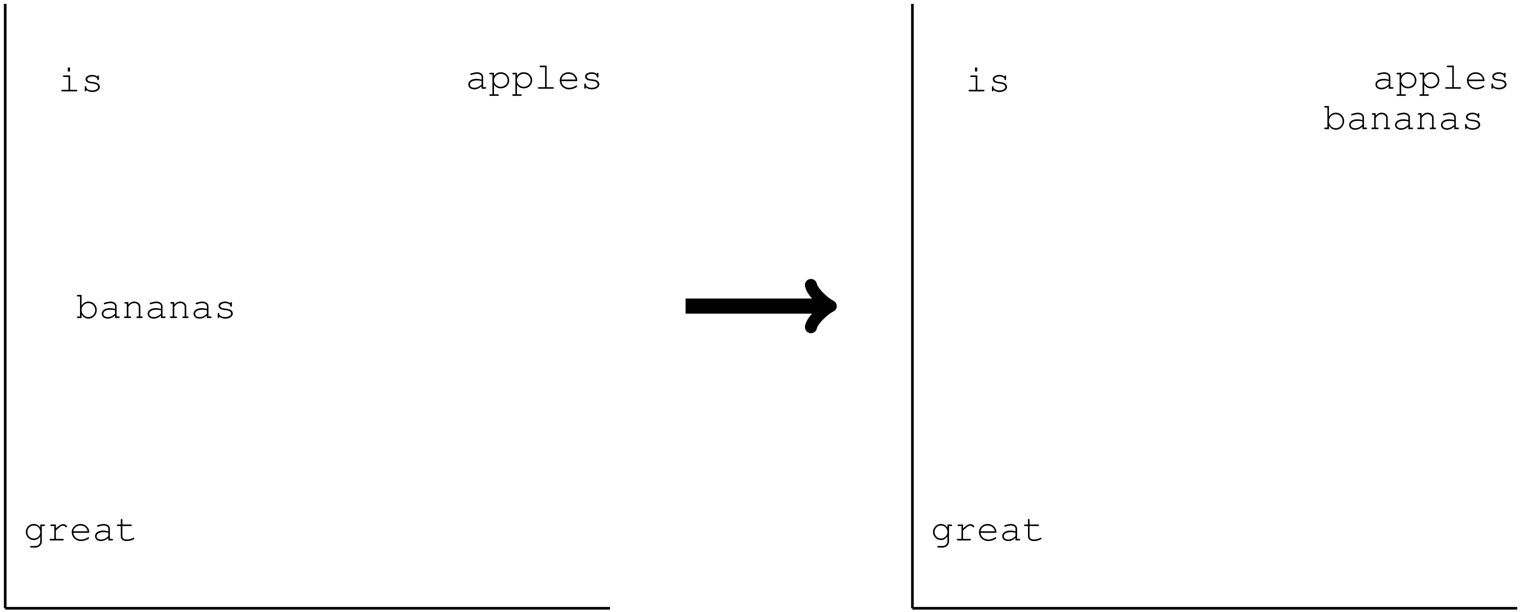
Training data: `apples are great, bananas are great.`



- Use input word to predict next word in phrase ➜ `apples`
- We want the largest output value after softmax to be the target word.
- Cross entropy loss with backpropagation to optimise weights.

Word Embeddings

Visualising Word Embeddings



- Weights going into activation layer are the values associated with each word.
- When words appear in similar contexts, values (weights) become similar.
- All the weights for a given word are called the **word embedding**.

Word Embeddings

Summary

Word embeddings let us represent text as values for machine learning problems.

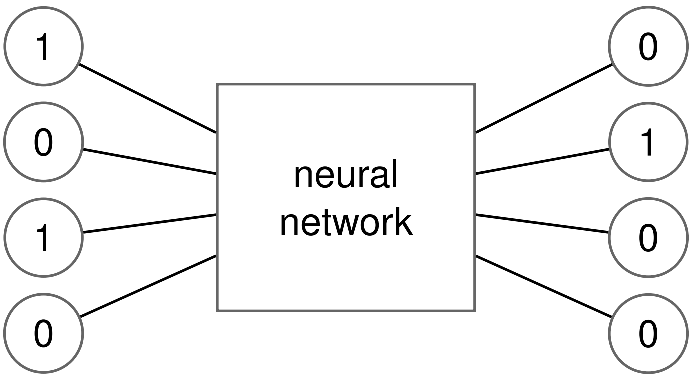
- Rather than using random values, use a neural network to learn values.
- Use context of words in training dataset to optimise weights for embeddings.
- Similar words get similar embeddings, which helps with training.

Problem: Just predicting the next word doesn't provide much context.

Word Embeddings
word2vec

Continuous Bag of Words (CBOW)

➜ Increase context by using surrounding words to predict what occurs in the middle.

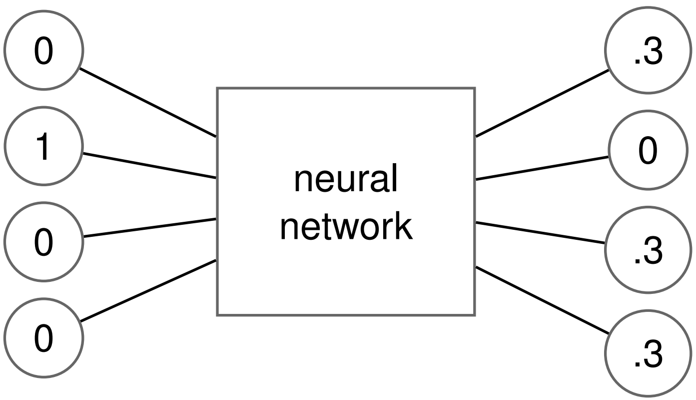


Word Embeddings

word2vec

Skip gram

➜ Increase context by using word in the middle to predict surrounding words.



Word Embeddings

Efficiently Training word2vec

- In practice, there are hundreds of activation functions.
- And significantly more training data (e.g., all of Wikipedia).
- Vocabulary (input size) is much larger, typically 3,000,000 words and phrases.

Total weights to optimise:

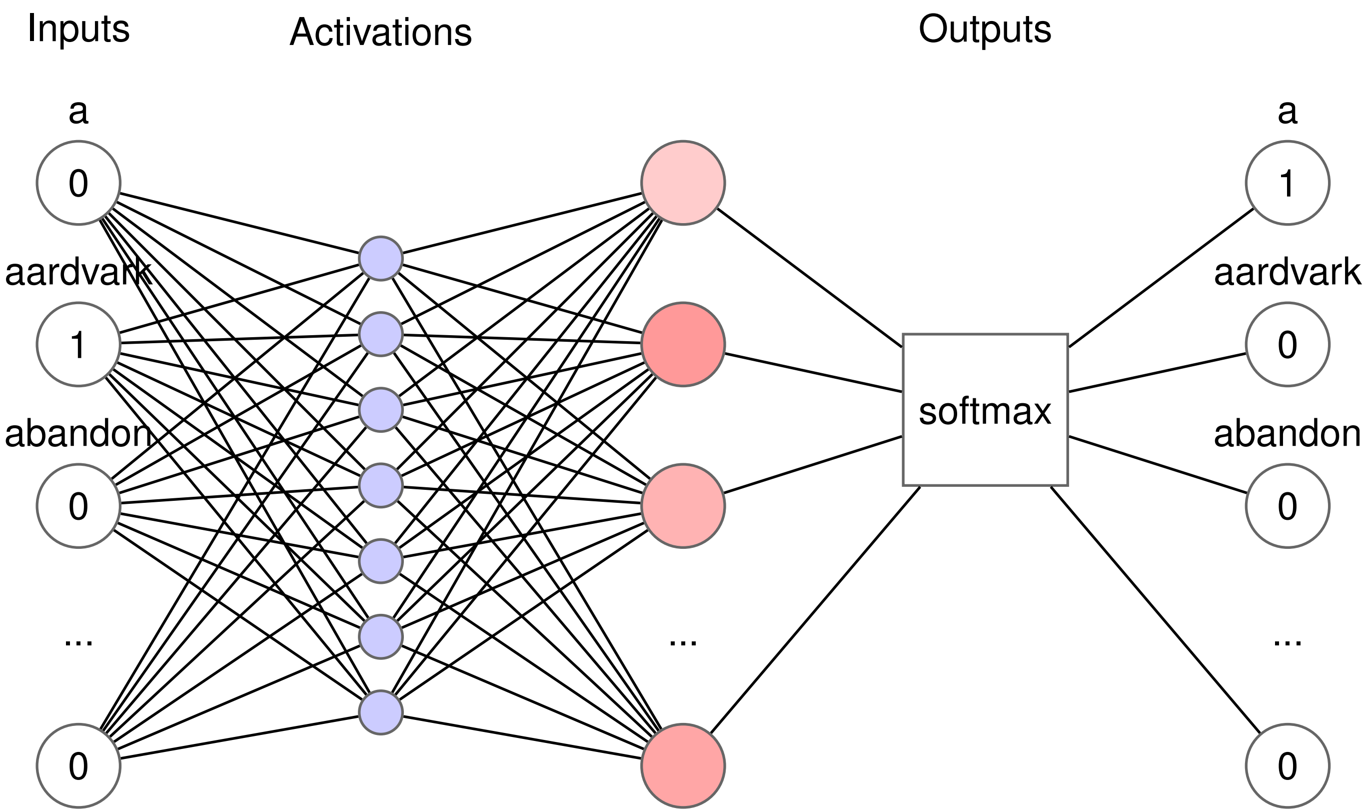
$$3,000,000 \cdot 100 \cdot 2 = 600,000,000$$

3M words, 100 activations (times 2 for input+output).

Solution: negative sampling.

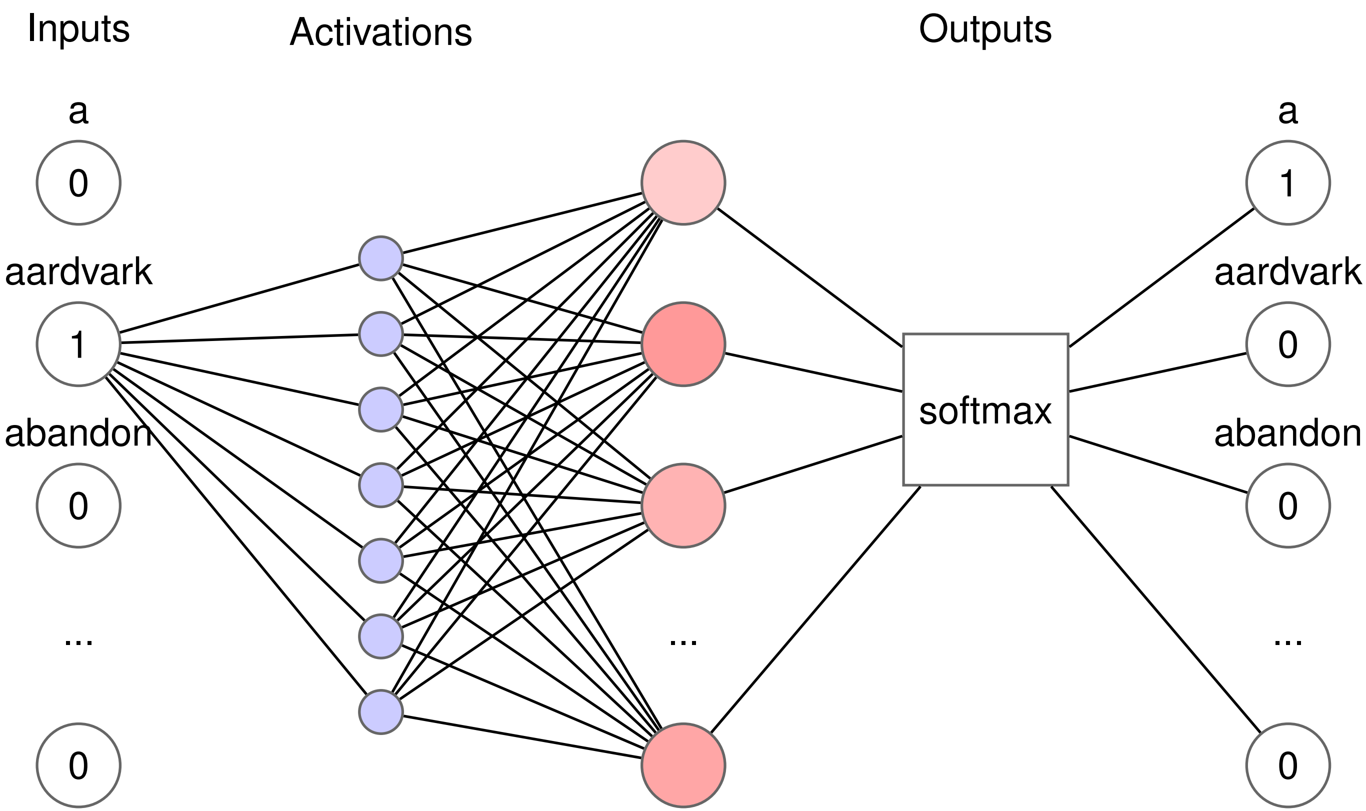
Word Embeddings

Efficiently Training word2vec



Word Embeddings

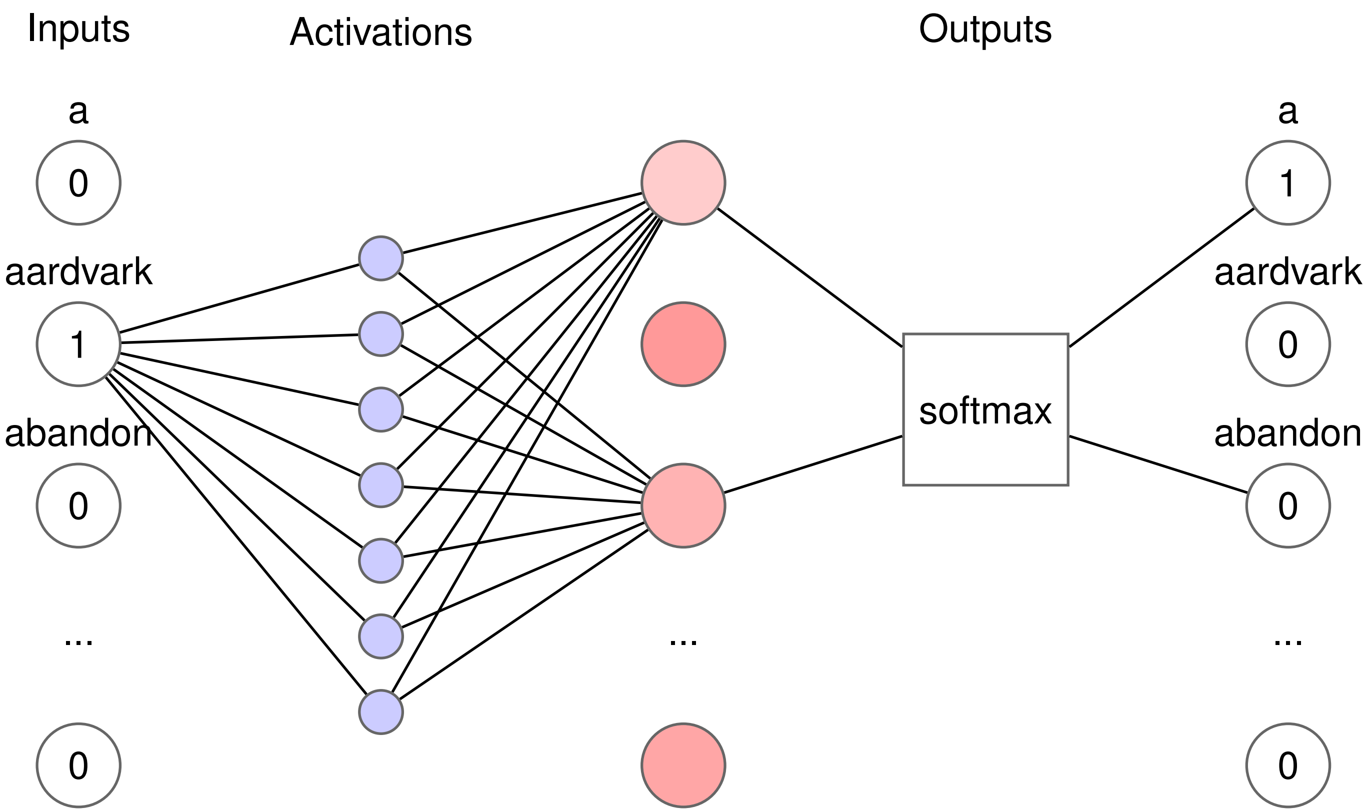
Efficiently Training word2vec



- Drop weights that do not contribute to prediction.
- Still left with over 300,000,000 weights to optimise.

Word Embeddings

Efficiently Training word2vec



- Randomly select subset of words will be ‘negative’ samples.
- `a` is still our target word, but now `abandon` is a negative sample.
- Now only need to optimise approximately 300 weights per step.

Question 1: How would you design a ranking function with word embeddings alone?

Question 2: How could you represent queries and documents with embeddings?

Question 3: How would you train a neural ranking model if you had query and document embeddings?

Relevant papers:
- https://dl.acm.org/doi/pdf/10.1145/2838931.2838936
- https://cs.stanford.edu/~quocle/paragraph_vector.pdf