Machine Learning for Language Modelling

Part 3: Neural network language models

Marek Rei

UNIVERSITY OF TARTU

UNIVERSITY OF CAMBRIDGE

Recap

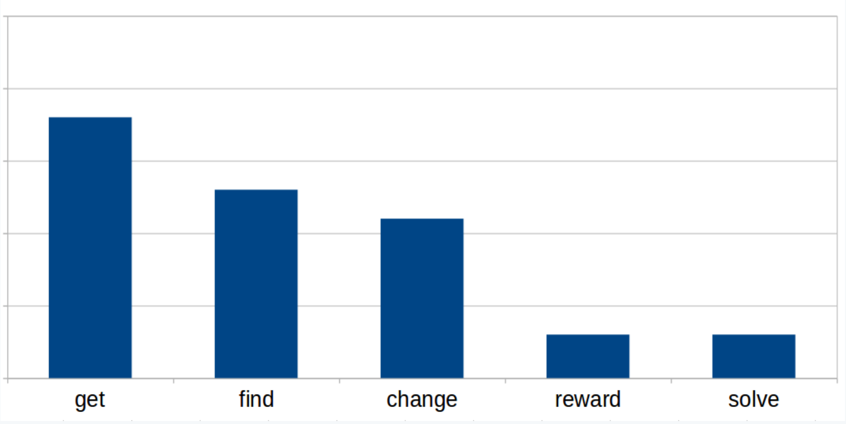
- Language modelling:
  - Calculates the probability of a sentence
  - Calculates the probability of a word in the sentence
- N-gram language modelling

$$P(w_1\ w_2\ \dots\ w_N) = \prod_{i=1}^{N} P(w_i|w_{i-1})$$

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}\ w_i)}{C(w_{i-1})}$$

Recap

- Assigning zero probabilities causes problems
- We use smoothing to distribute some probability mass to unseen n-grams



Recap

- "Stupid" backoff

$$S(w_i|w_{i-2}\ w_{i-1}) = \begin{cases} \frac{C(w_{i-2}\ w_{i-1}\ w_i)}{C(w_{i-2}\ w_{i-1})} & \text{if } C(w_{i-2}\ w_{i-1}\ w_i) > 0 \\ 0.4 \cdot S(w_i|w_{i-1}) & \text{otherwise} \end{cases}$$

- Interpolation

$$\begin{aligned} P_{interp}(w_i|w_{i-2}\ w_{i-1}) = \quad & \lambda_1 P(w_i|w_{i-2}\ w_{i-1}) \\ & +\lambda_2 P(w_i|w_{i-1}) \\ & +\lambda_3 P(w_i) \end{aligned}$$

- Kneser-Ney smoothing

$$P_{KN}(w_i|w_{i-1}) = \frac{max(C(w_{i-1}\ w_i) - D, 0)}{C(w_{i-1})} + \lambda(w_{i-1}) P_{continuation}(w_i)$$

Evaluation: extrinsic

How to evaluate language models?

The best option: evaluate the language model when solving a specific task

- Speech recognition accuracy
- Machine translation accuracy
- Spelling correction accuracy

Compare 2 (or more) models, and see which one is best

Evaluation: extrinsic

Evaluating next word prediction directly

Natural	language	processing
The	world	processing
In	language	understanding
A	resources	sentences
General	resources	text
Natural	enemies	toolkit

Accuracy

1/3 = 0.33

Evaluation: extrinsic

Evaluating next word prediction directly

Natural	language	processing
The	world	processing
In	language	understanding
A	resources	sentences
General	resources	text
Natural	enemies	toolkit

Accuracy

2/3 = 0.67

Evaluation: intrinsic

Extrinsic evaluation can be

- time consuming
- expensive

Instead, can evaluate the task of language modelling directly

Evaluation: intrinsic

Prepare disjoint datasets

Measure performance on the test set, using an evaluation metric.

Evaluation: intrinsic

What makes a good language model?

Language model that prefers **good** sentences to **bad** ones

Language model that prefers sentences that are

- real sentences
- more frequently observed
- grammatical

Perplexity

The most common evaluation measure for language modelling: **perplexity**

$$PP(W) = \sqrt[N]{\frac{1}{\prod_{i=1}^{N} P(w_i|w_{i-1})}}$$

Intuition: The best language model is the one that best predicts an unseen test set.

Might not always predict performance on an actual task.

Perplexity

The best language model is the one that best predicts an unseen test set

Natural language ________

database	0.4
sentences	0.3
and	0.15
understanding	0.1
processing	0.05

processing	0.4
understanding	0.3
sentences	0.15
text	0.1
toolkit	0.05

processing	0.6
information	0.2
query	0.1
sentence	0.09
text	0.01

Perplexity

Perplexity is the probability of the test set, normalised by the number of words

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

Chain rule

$$PP(W) = \sqrt[N]{\frac{1}{\prod_{i=1}^{N} P(w_i | w_1 \ldots w_{i-1})}}$$

Bigrams

$$PP(W) = \sqrt[N]{\frac{1}{\prod_{i=1}^{N} P(w_i | w_{i-1})}}$$

Perplexity example

Text: *natural language processing*

w	p(w | <s>)
processing	0.4
language	0.3
the	0.17
natural	0.13

w	p(w | natural)
processing	0.4
language	0.35
natural	0.2
the	0.05

w	p(w | language)
processing	0.6
language	0.2
the	0.1
natural	0.1

What is the perplexity?

$$PP(W) = \sqrt[N]{\frac{1}{\prod_{i=1}^{N} P(w_i|w_{i-1})}} \qquad PP(\text{natural language processing}) = \sqrt[3]{\frac{1}{0.13 \times 0.35 \times 0.6}} = 3.32$$

Minimising perplexity means maximising the probability of the text

Perplexity example

Let's suppose a sentence consisting of random digits

7 5 0 9 2 3 7 8 5 1 ...

What is the perplexity of this sentence according to a model that assigns P=1/10 to each digit?

$$
\begin{aligned}
PP(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\
&= ((\frac{1}{10})^N)^{-\frac{1}{N}} \\
&= (\frac{1}{10})^{-1} \\
&= 10
\end{aligned}
$$

Perplexity

Trained on 38 million words, tested on 1.5 million words on WSJ text

	Uniform	**Unigram**	**Bigram**	**Trigram**
Perplexity	vocabulary size V	962	170	109

Jurafsky (2012)

Lower perplexity = better language model

Problems with N-grams

Problem 1: They are sparse

There are V^4 possible 4-grams. With $V=10{,}000$ that's 10^{16} 4-grams.

We will only see a tiny fraction of them in our training data.

Problems with N-grams

Problem 2: words are independent

They only map together identical words, but ignore similar or related words.

If

$$P(\text{blue daffodil}) == 0$$

we could use the intuition that "*blue*" is related to "*yellow*" and

$$P(\text{yellow daffodil}) > 0$$

Vector representation

- Let’s represent words (or any objects) as vectors
- Let’s choose them, so that similar words have similar vectors

A vector is just an ordered list of values

[0.0, 1.0, 8.6, 0.0, -1.2, 0.1]

Vector representation

How can we represent words as vectors?

Option 1: each element represents the word.
Also known as “1-hot” or “1-of-V” representation.

	bear	*cat*	*frog*
bear	1	0	0
cat	0	1	0
frog	0	0	1

bear=[1.0, 0.0, 0.0]

cat=[0.0, 1.0, 0.0]

Vector representation

Option 2: each element represents a property, and they are shared between the words.

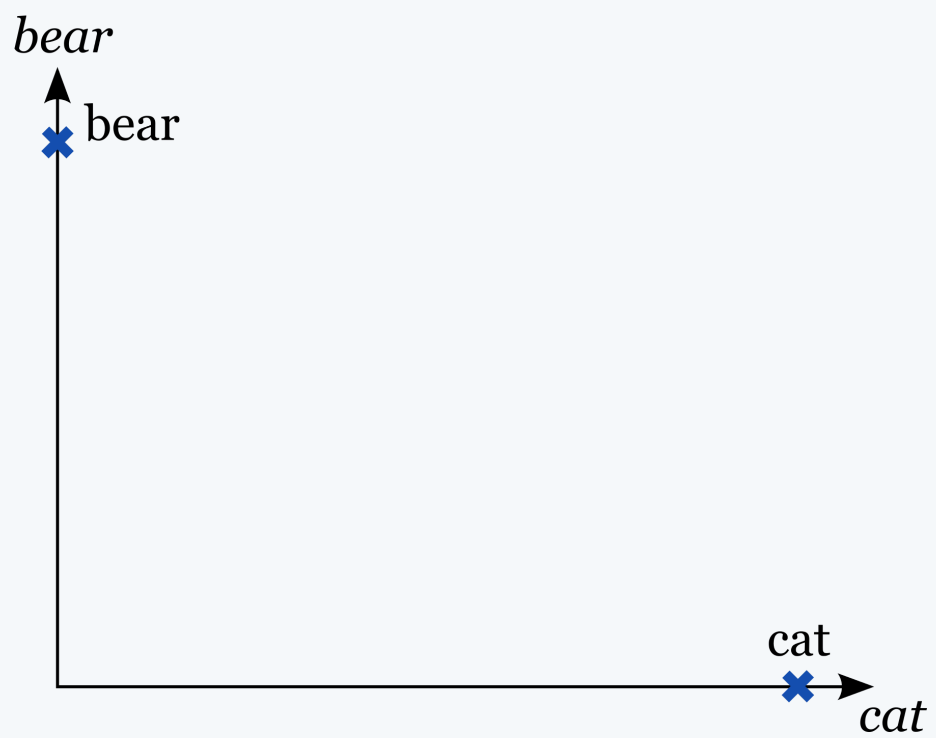
Also known as “distributed” representation.

	furry	*dangerous*	*mammal*
bear	0.9	0.85	1
cat	0.85	0.15	1
frog	0	0.05	0

bear = [0.9, 0.85, 1.0]

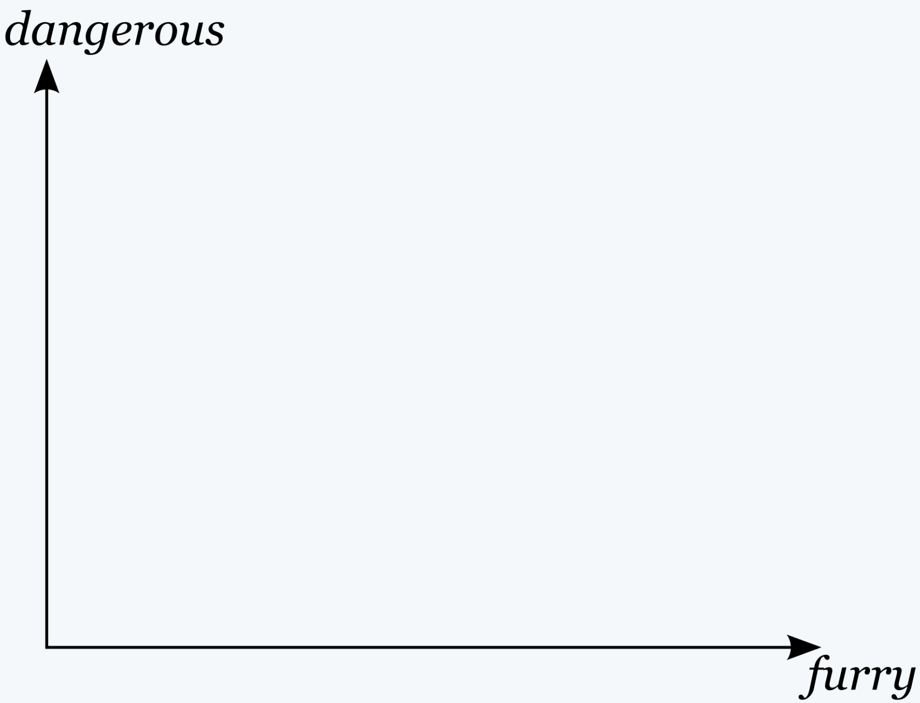
cat = [0.85, 0.15, 1.0]

Vector representation

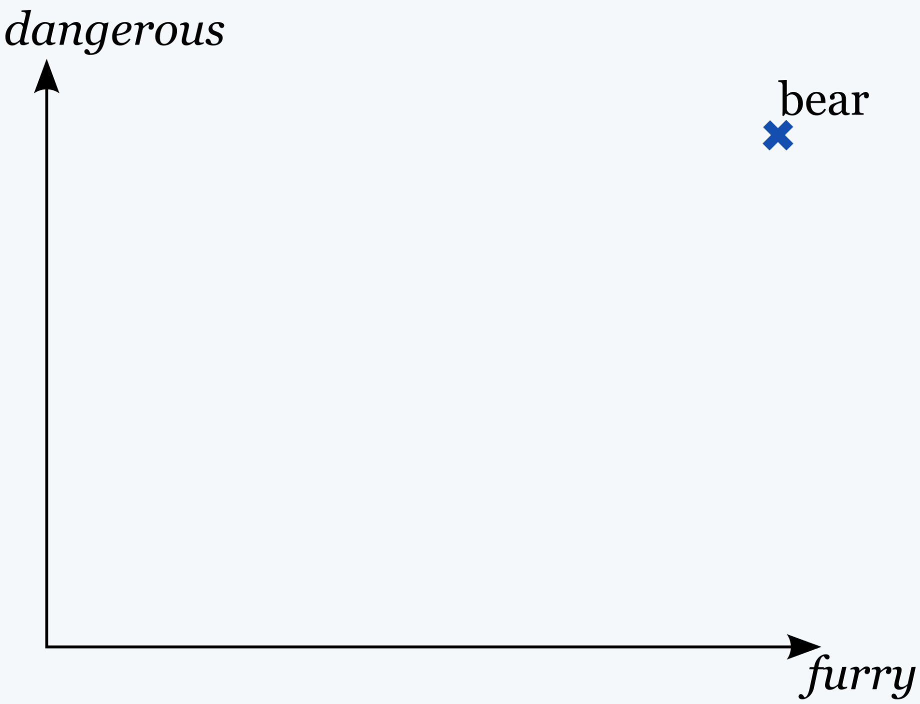


When using 1-hot vectors, we can't fit many and they tell us very little.

Vector representation

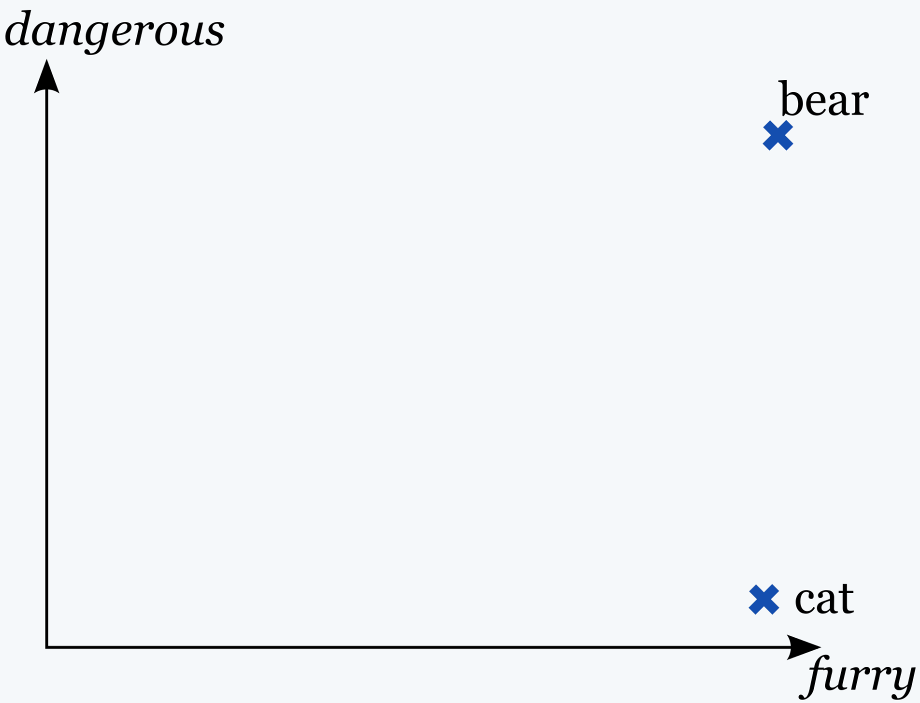


Vector representation



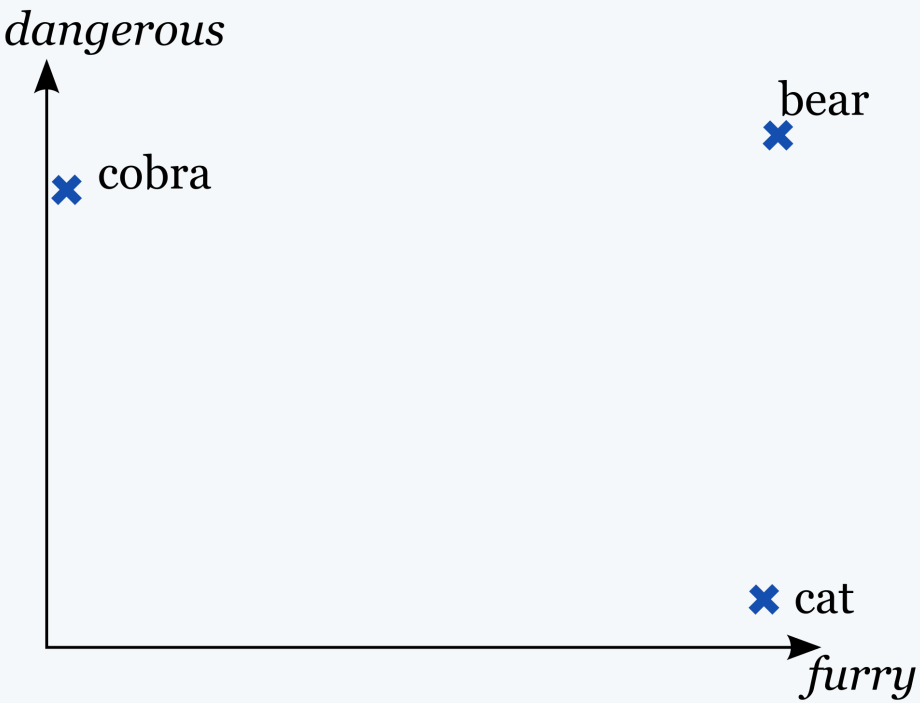
	furry	*dangerous*
bear	0.9	0.85

Vector representation



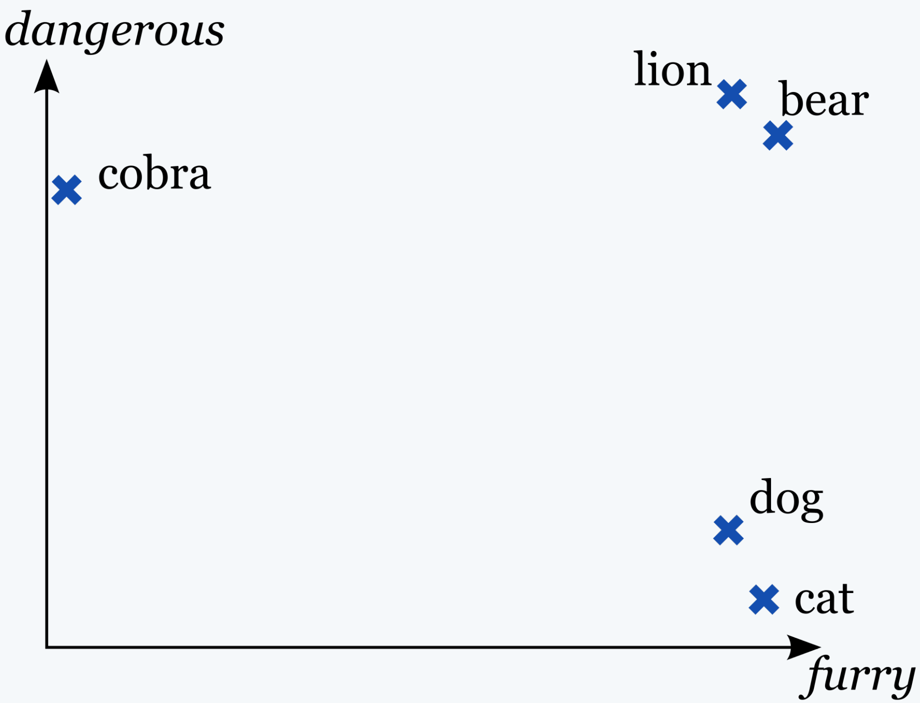
	furry	*dangerous*
bear	0.9	0.85
cat	0.85	0.15

Vector representation



	furry	*dangerous*
bear	0.9	0.85
cat	0.85	0.15
cobra	0.0	0.8

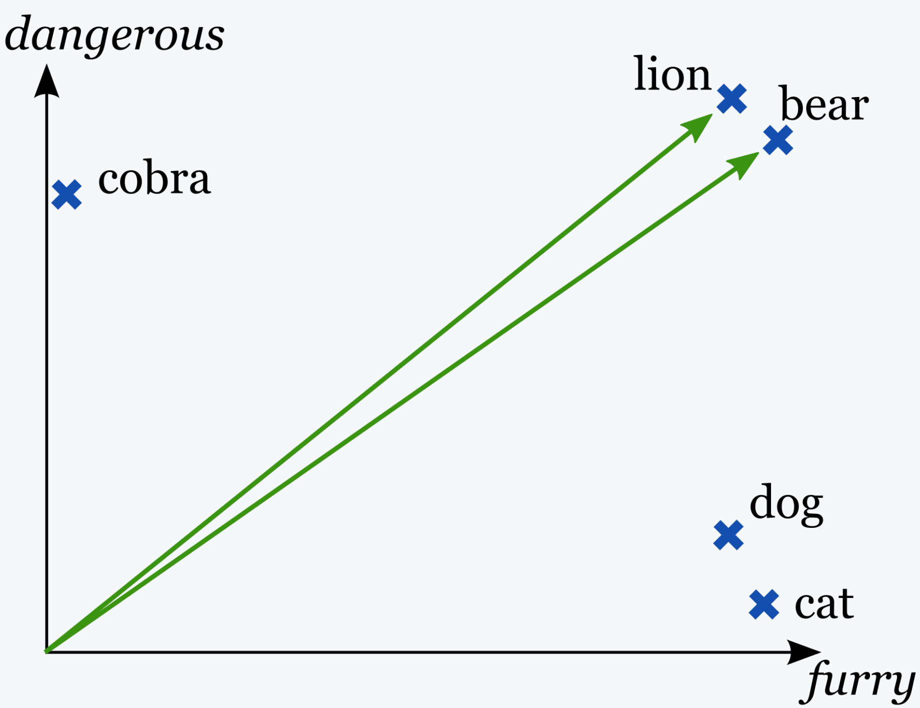
Vector representation



	furry	*dangerous*
bear	0.9	0.85
cat	0.85	0.15
cobra	0.0	0.8
lion	0.85	0.9
dog	0.8	0.15

Distributed vectors group similar words/objects together

Vector representation

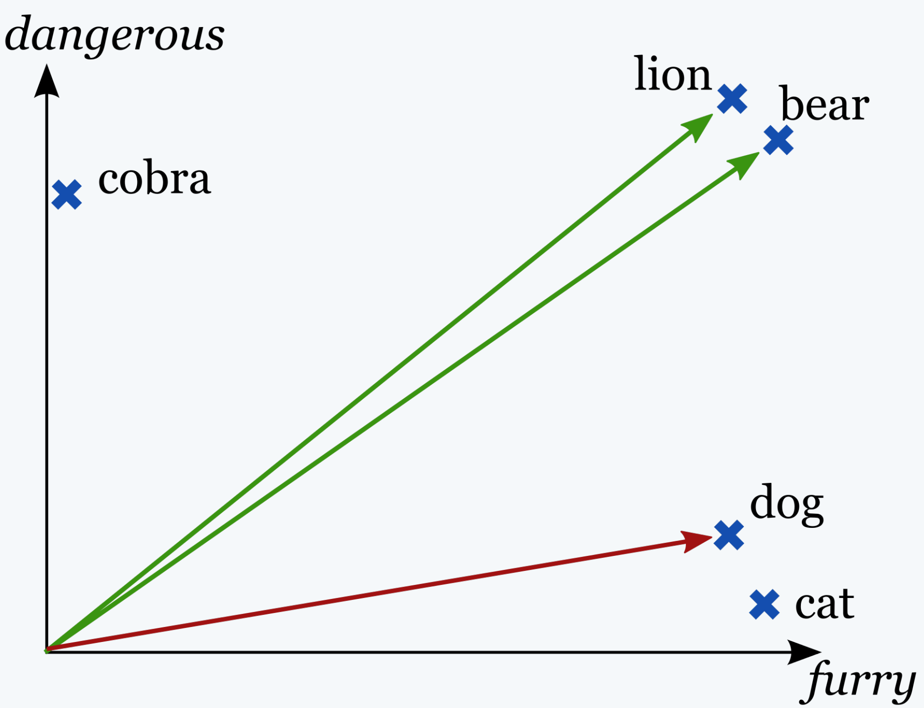


$$cos(a,b) = \frac{\sum_i a_i \cdot b_i}{\sqrt{\sum_i a_i^2} \cdot \sqrt{\sum_i b_i^2}}$$

cos(lion, bear) = 0.998

Can use cosine to calculate similarity between two words

Vector representation

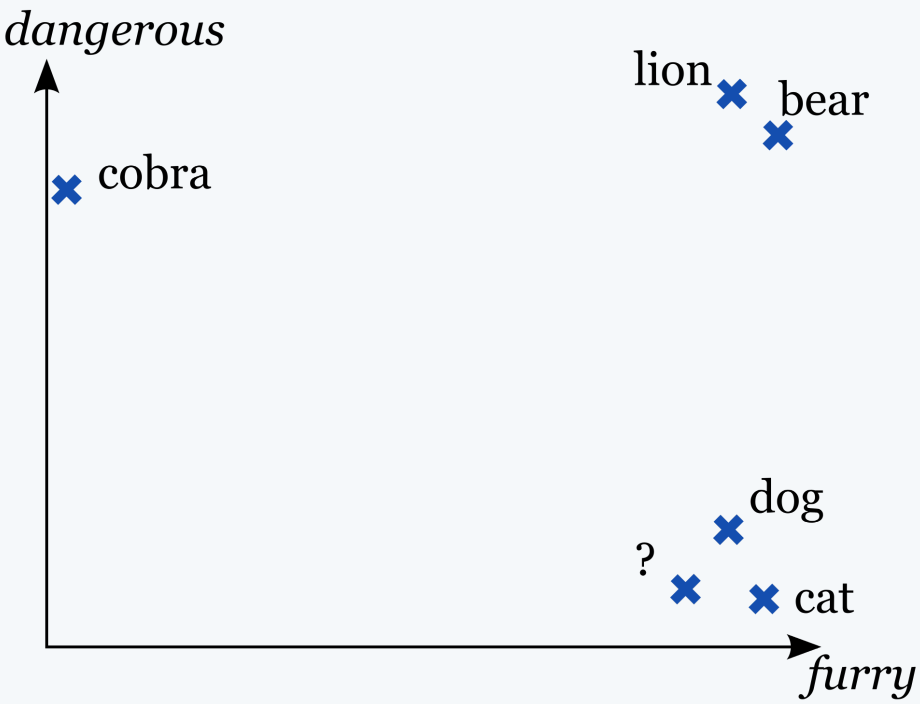


$$cos(a,b) = \frac{\sum_i a_i \cdot b_i}{\sqrt{\sum_i a_i^2} \cdot \sqrt{\sum_i b_i^2}}$$

cos(lion, bear) = 0.998
cos(lion, dog) = 0.809
cos(cobra, dog) = 0.727

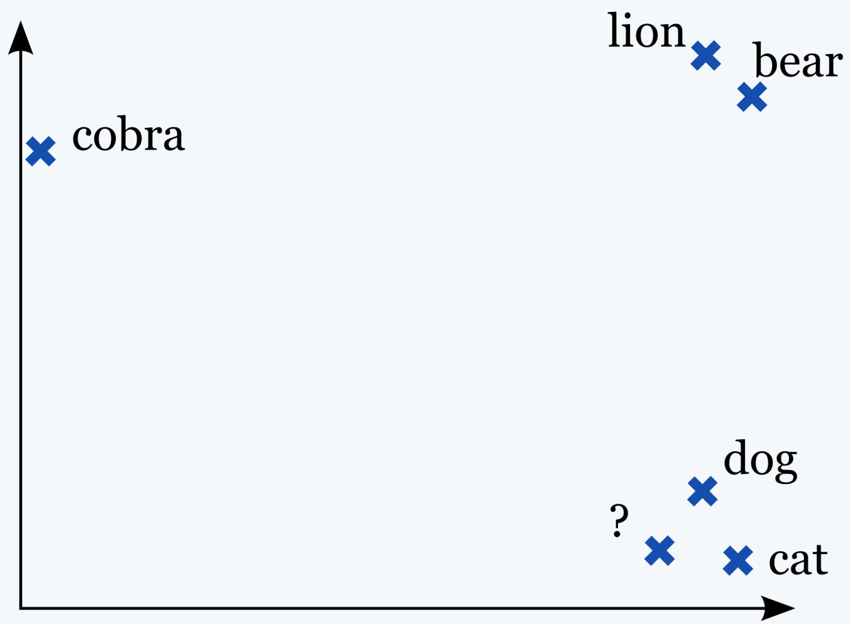
Can use cosine to calculate similarity between two words

Vector representation



We can infer some information, based only on the vector of the word

Vector representation



We don't even need to know the labels on the vector elements

Vector representation

The vectors are usually not 2 or 3-dimensional.
More often 100-1000 dimensions.

bear -0.089383 -0.375981 -0.337130 0.025117 -0.232542 -0.224786 0.148717 -0.154768 -0.260046 -0.156737 -0.085468 0.180366 -0.076509 0.173228 0.231817 0.314453 -0.253200 0.170015 -0.111660 0.377551 -0.025207 -0.097520 -0.020041 0.117727 0.105745 -0.352382 0.010241 0.114237 -0.315126 0.196771 -0.116824 -0.091064 -0.291241 -0.098721 0.297539 0.213323 -0.158814 -0.157823 0.152232 0.259710 0.335267 0.195840 -0.118898 0.169420 -0.201631 0.157561 0.351295 0.033166 0.003641 -0.046121 0.084251 0.021727 -0.065358 -0.083110 -0.265997 0.027450 0.372135 0.040659 0.202577 -0.109373 0.183473 -0.380250 0.048979 0.071580 0.152277 0.298003 0.017217 0.072242 0.541714 -0.110148 0.266429 0.270824 0.046859 0.150756 -0.137924 -0.099963 -0.097112 -0.110336 -0.018136 -0.032682 0.182723 0.260882 -0.146807 0.502611 0.034849 -0.092219 -0.103714 -0.034353 0.112178 0.065348 0.161681 0.006538 0.364870 0.153239 -0.366863 -0.149125 0.413624 -0.229378 -0.396910 -0.023116

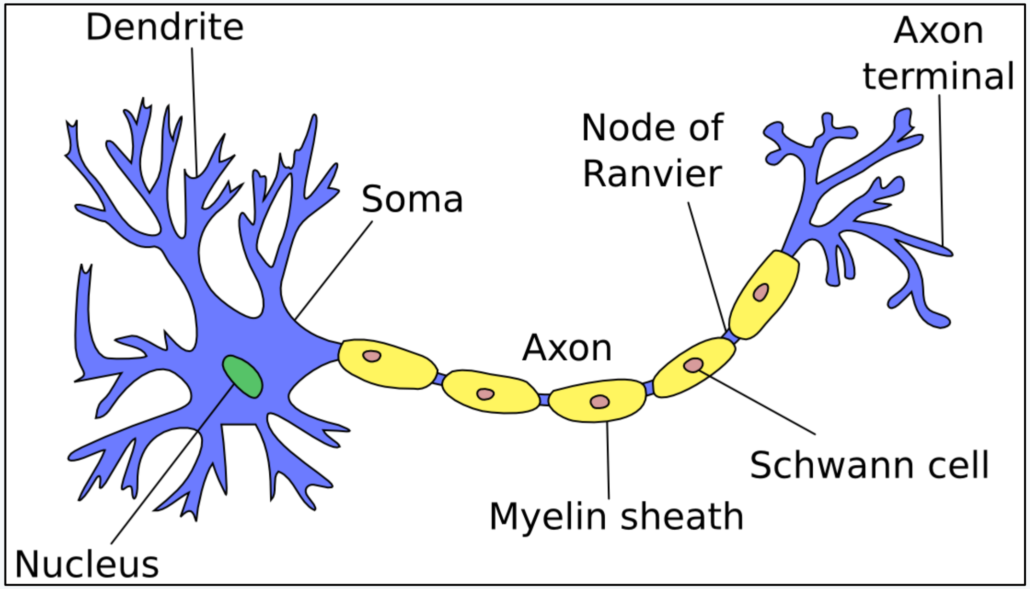
Idea

- Let’s build a neural network language model
- … that represents each word as a vector
- … and similar words have similar vectors

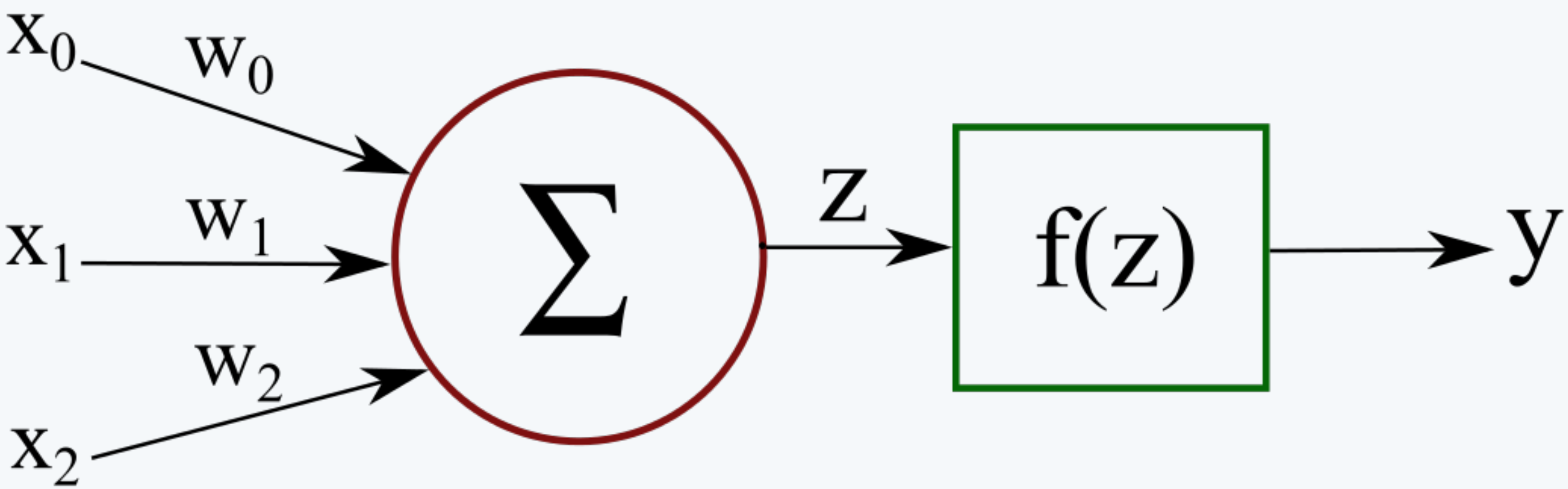
- Similar contexts will predict similar words
- Optimise the vectors together with the model, so we end up with vectors that perform well for language modelling (aka representation learning)

Neuron

- A neuron is a very basic classifier
- It takes a number of input signals (like a feature vector) and outputs a single value (a prediction).

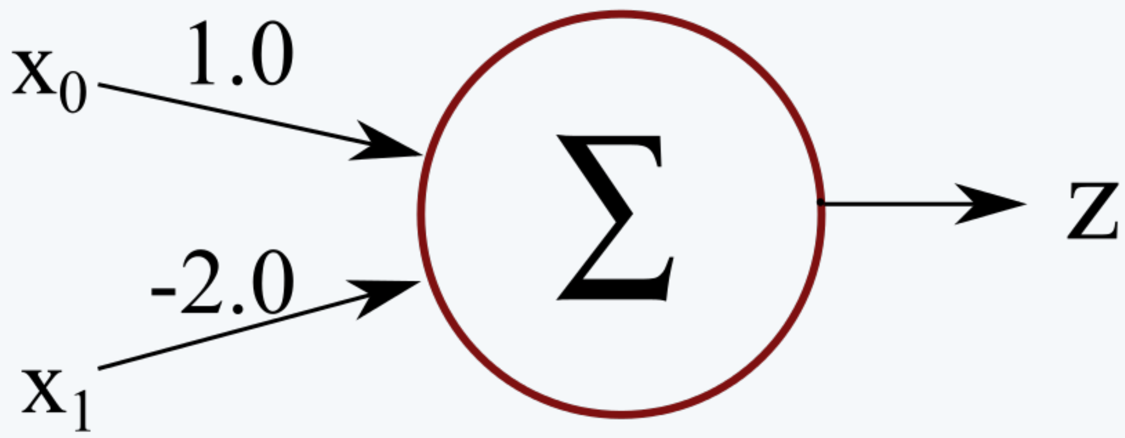


Artificial neuron



Input: $[x_0, x_1, x_2]$
Output: y

Artificial neuron

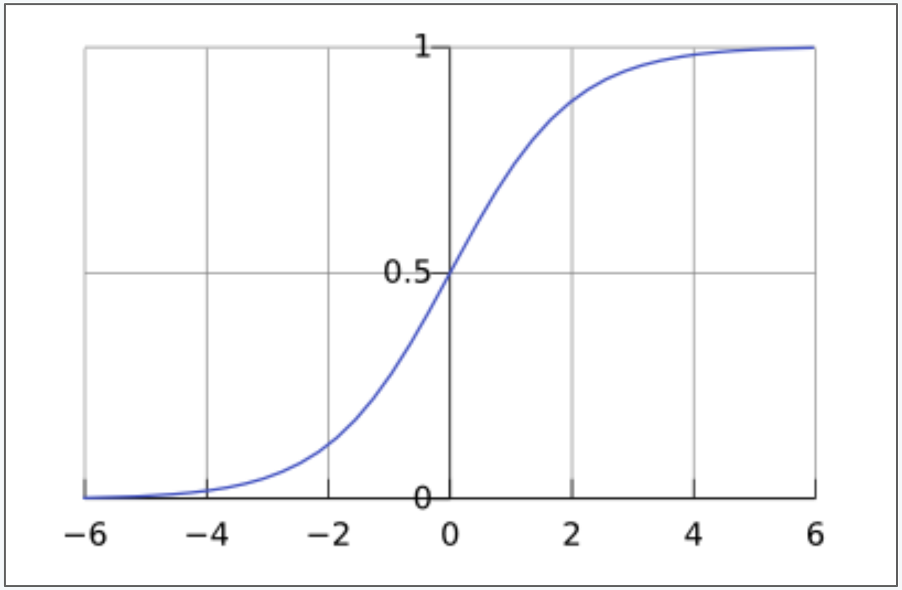


$$x = bear = [0.9, 0.85]$$

$$z = x_0 \cdot w_0 + x_1 \cdot w_1$$

$$z = 0.9 \cdot 1.0 + 0.85 \cdot (-2.0) = -0.8$$

Sigmoid function



- Takes in any value
- Squeezes it into a range between 0 and 1
- Also known as the logistic function

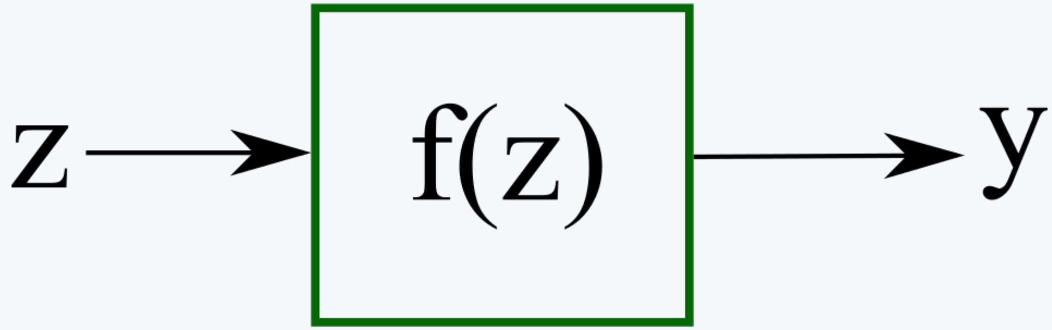
- A non-linear activation function allows us to solve non-linear problems

$$y = \frac{1}{1 + e^{-z}}$$

$$z \in (-\infty, \infty)$$

$$y \in (0, 1)$$

Artificial neuron

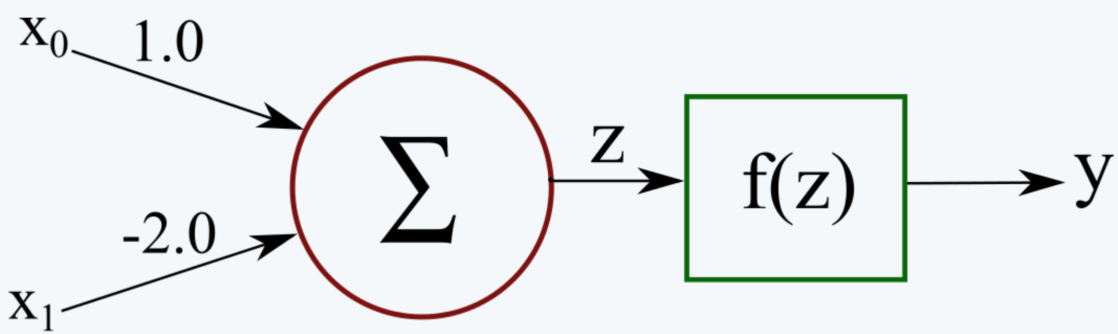


$$z = -0.8$$

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

$$y = \frac{1}{1+exp(-(-0.8))} = 0.31$$

Artificial neuron



$$z = \sum_i x_i w_i$$

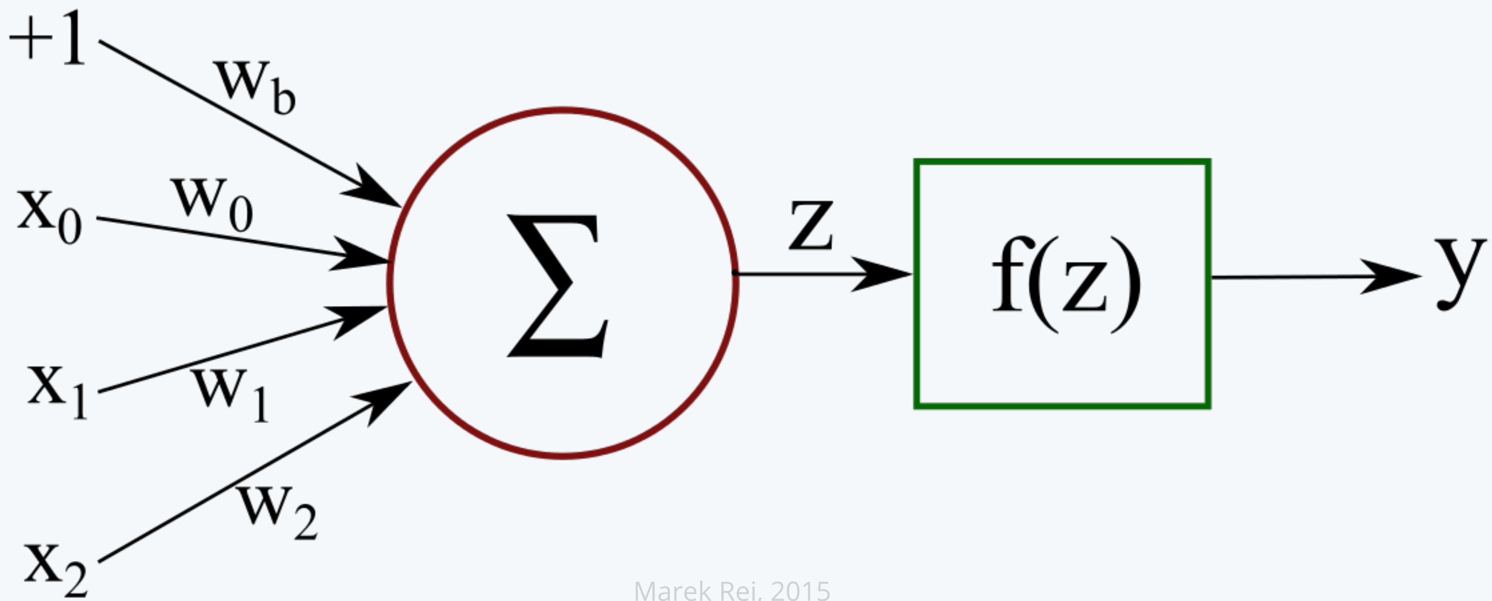
$$y = \frac{1}{1 + exp(-z)}$$

	x0	*x1*	*z*	*y*
bear	0.9	0.85	-0.8	0.31
cat	0.85	0.15	0.55	0.63
cobra	0.0	0.8	-1.6	0.17
lion	0.85	0.9	-0.95	0.28
dog	0.8	0.15	0.5	0.62

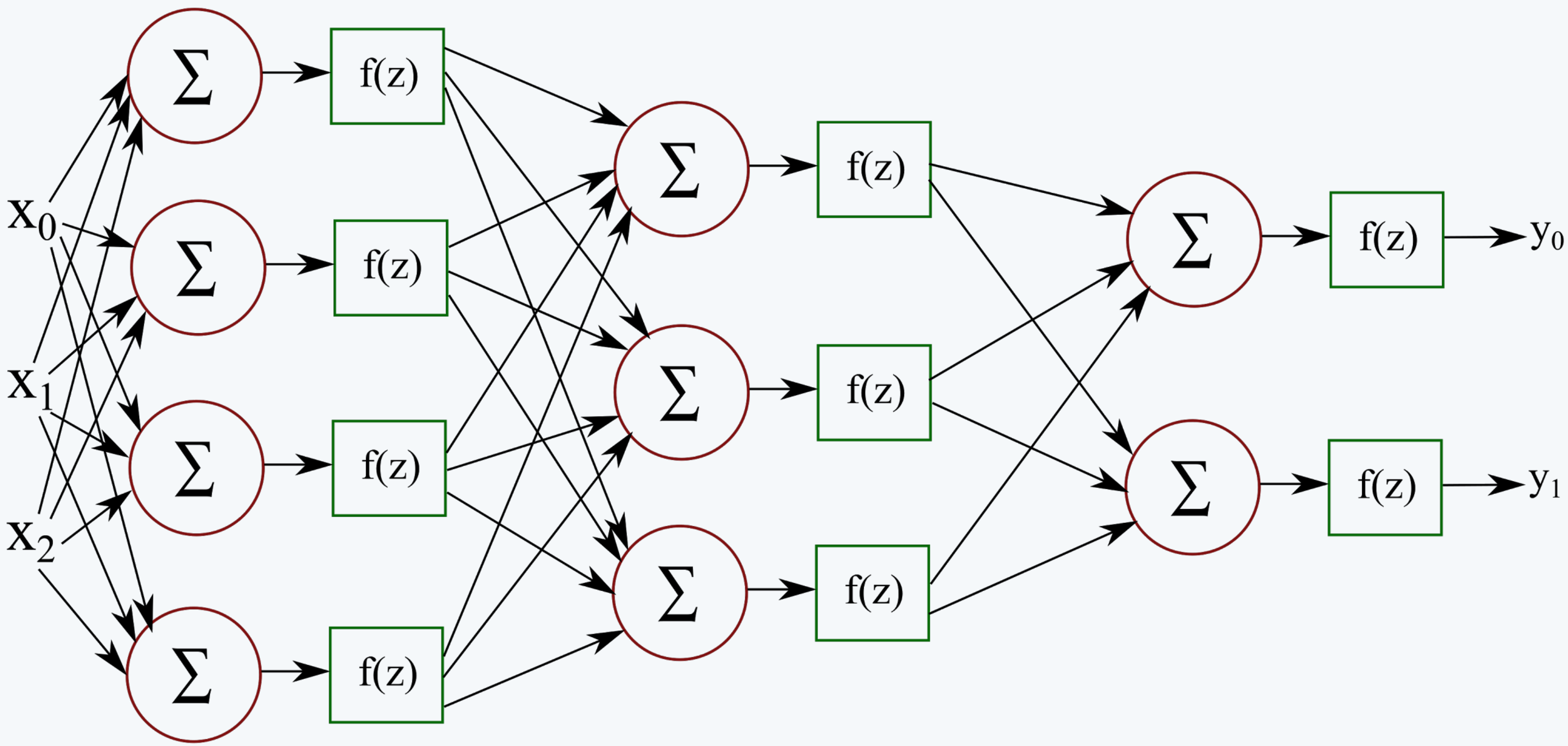
Artificial neuron

It is common for a neuron to have a separate bias input.

But when we do representation learning, we don't really need it.

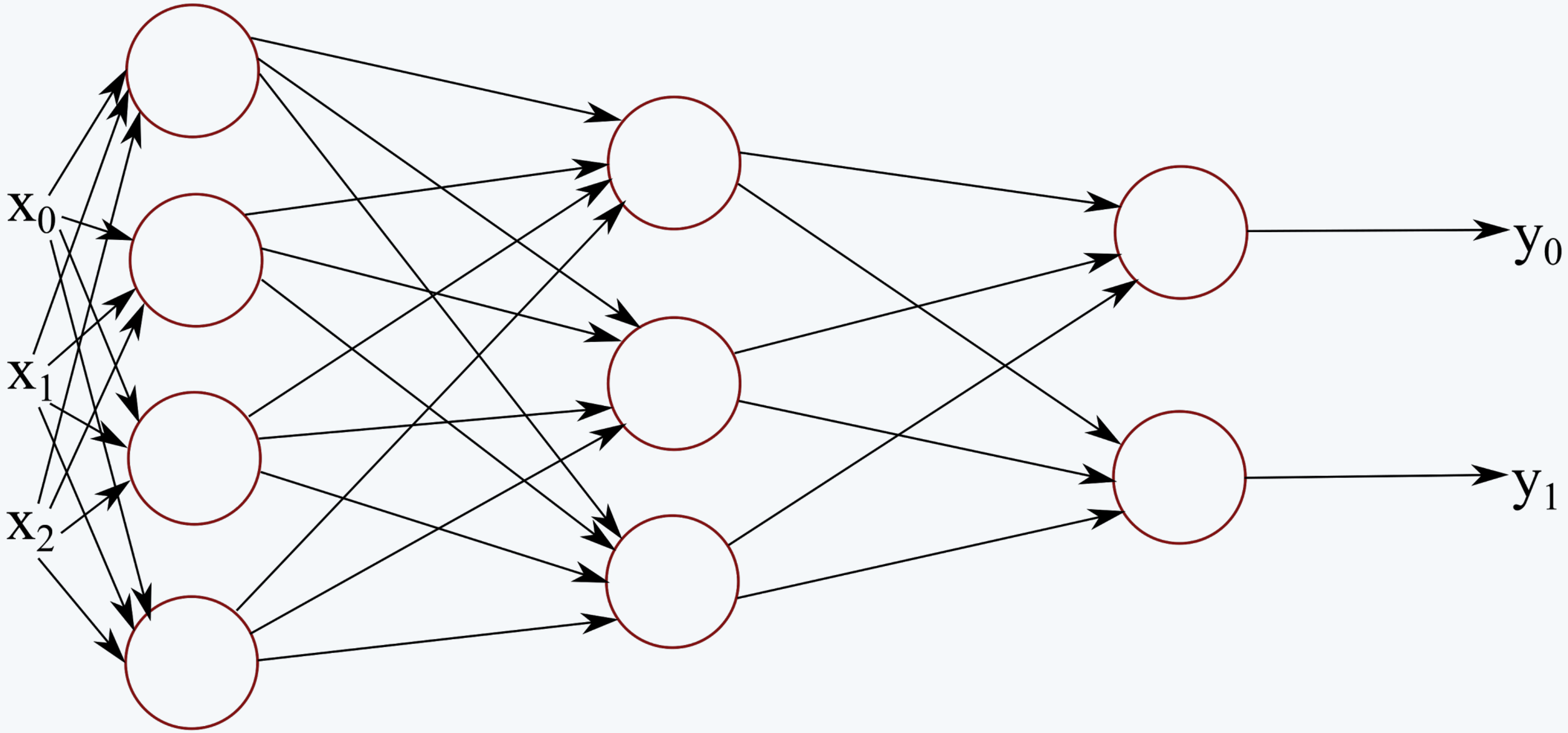


Neural network



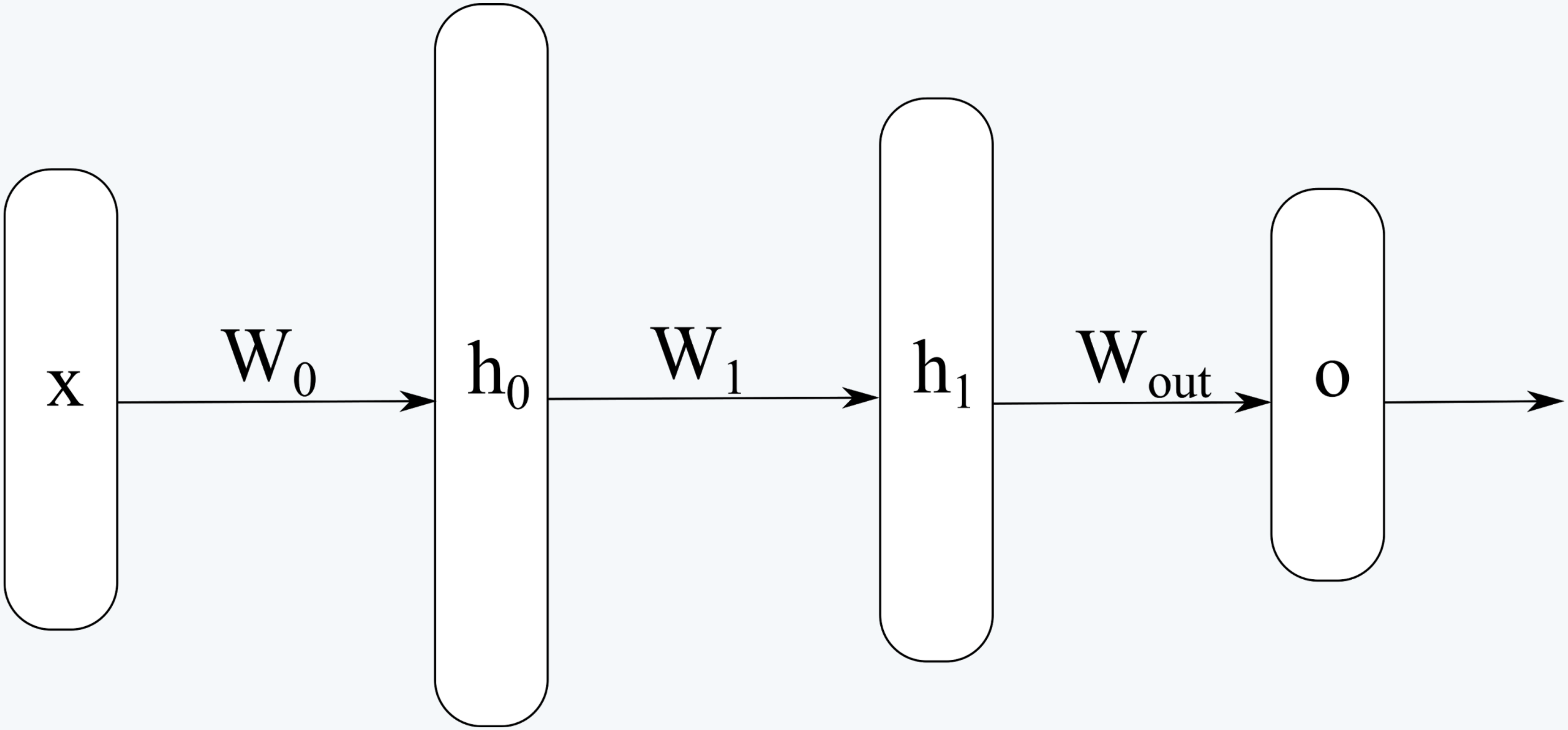
Many neurons connected together

Neural network



Usually, the neuron is shown as a single unit

Neural network



Or a whole layer of neurons is represented as a block

Matrix operations

- Vectors are matrices with a single column
- Elements indexed by row and column

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \qquad W = \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \\ w_{30} & w_{31} & w_{32} \\ w_{40} & w_{41} & w_{42} \end{bmatrix}$$

$$5 \times 1 \qquad\qquad 5 \times 3$$

Matrix operations

Multiplying by a constant - each element is multiplied individually

$$c \cdot \begin{vmatrix} x_{00} & x_{01} \\ x_{10} & x_{11} \\ x_{20} & x_{21} \end{vmatrix} = \begin{vmatrix} c \cdot x_{00} & c \cdot x_{01} \\ c \cdot x_{10} & c \cdot x_{11} \\ c \cdot x_{20} & c \cdot x_{21} \end{vmatrix}$$

$$2 \cdot \begin{vmatrix} 1.5 & 0.5 \\ 2 & 0 \\ -1 & 1 \end{vmatrix} = \begin{vmatrix} 3 & 1 \\ 4 & 0 \\ -2 & 2 \end{vmatrix}$$

$$c \in \mathbf{R}$$

Matrix operations

Adding matrices - the corresponding elements are added together

$$\begin{vmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \\ a_{20} & a_{21} \end{vmatrix} + \begin{vmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \\ b_{20} & b_{21} \end{vmatrix} = \begin{vmatrix} a_{00}+b_{00} & a_{01}+b_{01} \\ a_{10}+b_{10} & a_{11}+b_{11} \\ a_{20}+b_{20} & a_{21}+b_{21} \end{vmatrix}$$

$$3 \times 2 \qquad\qquad 3 \times 2 \qquad\qquad 3 \times 2$$

$$\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} + \begin{vmatrix} 2 & 1 \\ -1 & 0 \end{vmatrix} = \begin{vmatrix} 3 & 3 \\ 2 & 4 \end{vmatrix}$$

Matrix operations

Matrix multiplication - multiply and add elements in corresponding row and column

$$\begin{vmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{vmatrix} \times \begin{vmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \\ b_{20} & b_{21} \end{vmatrix} = \begin{vmatrix} \sum_k a_{0k}b_{k0} & \sum_k a_{0k}b_{k1} \\ \sum_k a_{1k}b_{k0} & \sum_k a_{1k}b_{k1} \end{vmatrix}$$

2×3 $\quad 3 \times 2$ $\quad 2 \times 2$

$$\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} \times \begin{vmatrix} 1 & 2 \\ 1 & 2 \end{vmatrix} = \begin{vmatrix} 3 & 6 \\ 7 & 14 \end{vmatrix}$$

Matrix operations

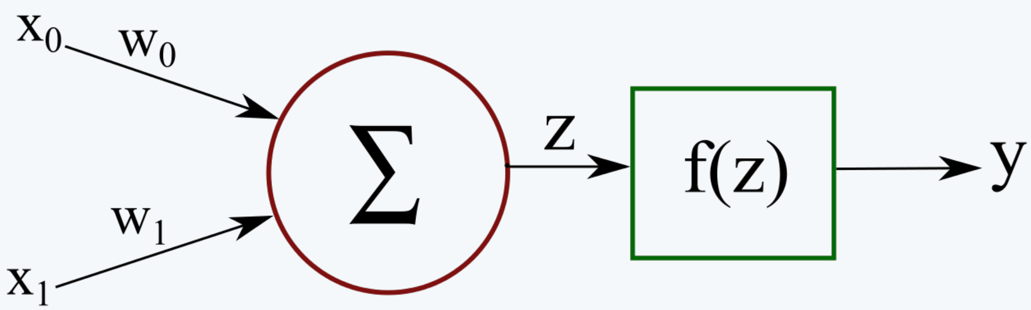
Matrix transpose - rows become columns, columns become rows

$$A = \begin{vmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{vmatrix} \qquad A^T = \begin{vmatrix} a_{00} & a_{10} \\ a_{01} & a_{11} \\ a_{02} & a_{12} \end{vmatrix}$$

$$2 \times 3 \qquad\qquad 3 \times 2$$

$$a = \begin{vmatrix} 1 \\ 2 \\ 3 \end{vmatrix} \qquad a^T = \begin{vmatrix} 1 & 2 & 3 \end{vmatrix}$$

Neuron activation with vectors



$$z = \sum_i x_i w_i$$

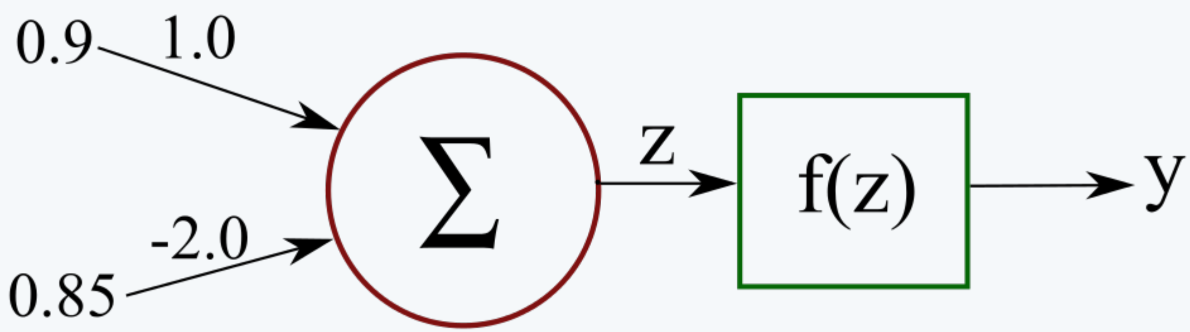
$$y = \frac{1}{1 + exp(-z)}$$

$$x = \begin{vmatrix} x_0 \\ x_1 \end{vmatrix} \qquad W = \begin{vmatrix} w_0 & w_1 \end{vmatrix}$$

$$z = W \cdot x \qquad f(z) = \frac{1}{1 + e^{-z}}$$

$$y = f(z) = f(W \cdot x)$$

Neuron activation with vectors

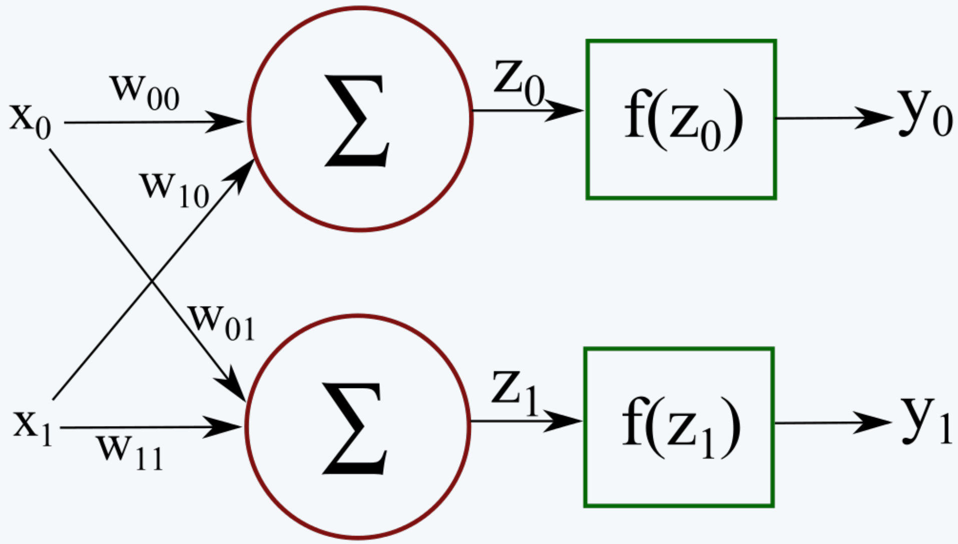


$$x = \begin{vmatrix} 0.9 \\ 0.85 \end{vmatrix} \qquad W = \begin{vmatrix} 1.0 & -2.0 \end{vmatrix}$$

$$z = W \cdot x = \begin{vmatrix} 1.0 & -2.0 \end{vmatrix} \cdot \begin{vmatrix} 0.9 \\ 0.85 \end{vmatrix} = 0.9 + (-1.7) = -0.8$$

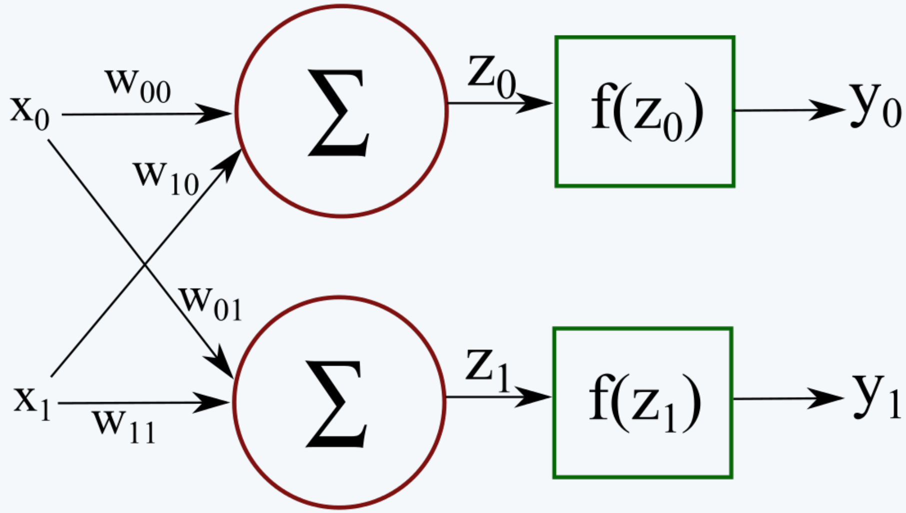
$$y = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-(-0.8)}} = 0.31$$

Feedforward activation



- The same process applies when activating multiple neurons
- Now the weights are in a matrix as opposed to a vector
- Activation $f(z)$ is applied to each neuron separately

Feedforward activation



$$x = \begin{vmatrix} x_0 \\ x_1 \end{vmatrix}$$

$$W = \begin{vmatrix} w_{00} & w_{10} \\ w_{01} & w_{11} \end{vmatrix}$$

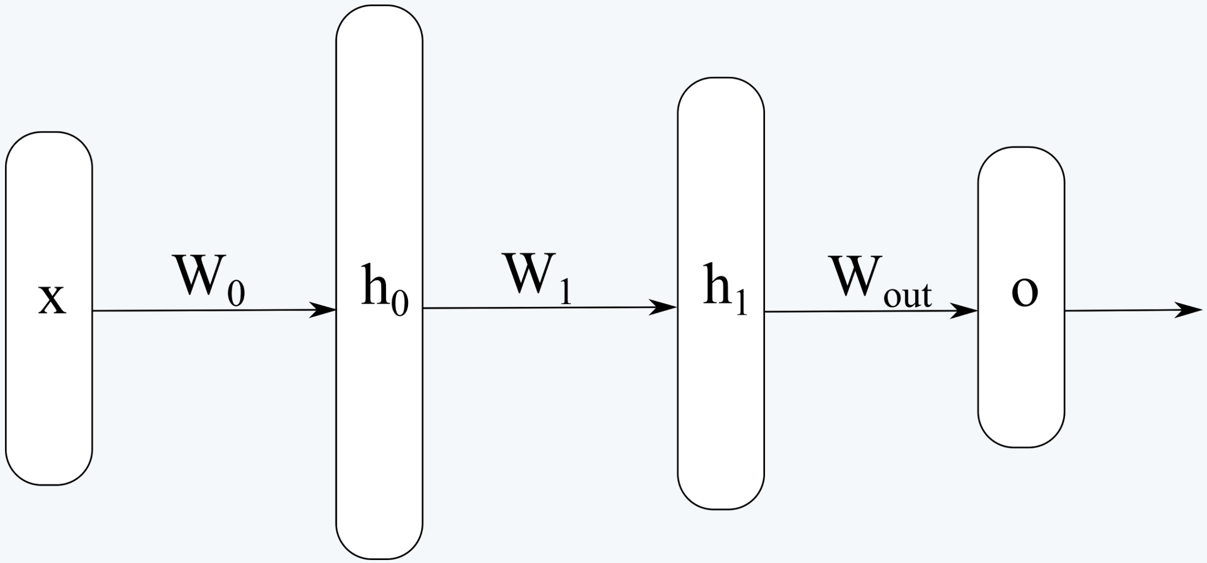
$$z = W \cdot x = \begin{vmatrix} w_{00} & w_{10} \\ w_{01} & w_{11} \end{vmatrix} \cdot \begin{vmatrix} x_0 \\ x_1 \end{vmatrix}$$

$$= \begin{vmatrix} w_{00} \cdot x_0 + w_{10} \cdot x_1 \\ w_{01} \cdot x_0 + w_{11} \cdot x_1 \end{vmatrix} = \begin{vmatrix} z_0 \\ z_1 \end{vmatrix}$$

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

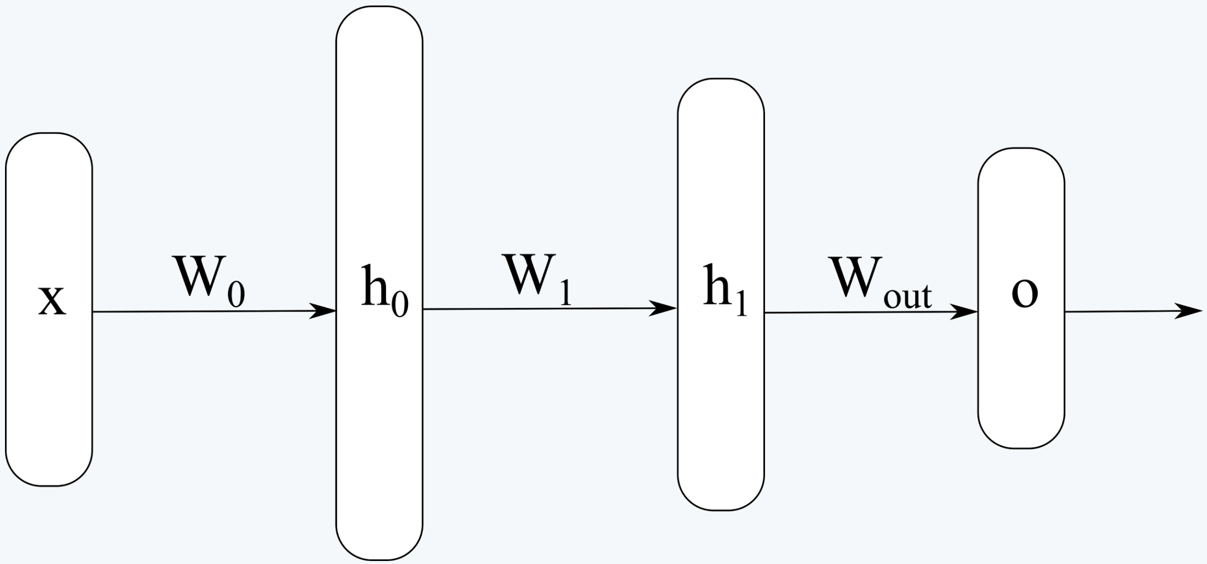
$$y = f(z) = f(W \cdot x) = \begin{vmatrix} f(z_0) \\ f(z_1) \end{vmatrix} = \begin{vmatrix} y_0 \\ y_1 \end{vmatrix}$$

Feedforward activation



1. Take vector from the previous layer
2. Multiply it with the weight matrix
3. Apply the activation function
4. Repeat

Feedforward activation



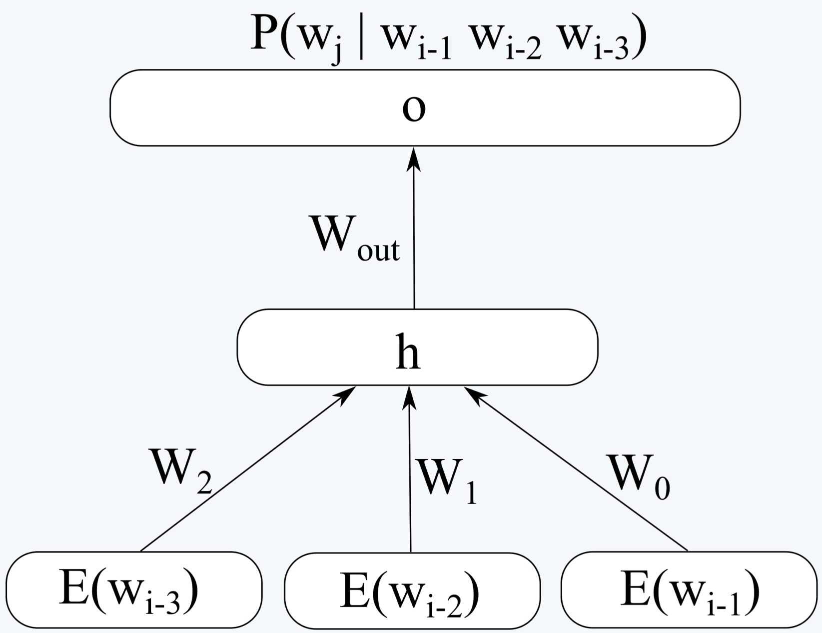
$$h_0 = f(W_0 \cdot x)$$

$$h_1 = f(W_1 \cdot x)$$

$$o = f(W_{out} \cdot h_1)$$

$$o = f(W_{out} \cdot f(W_1 \cdot f(W_0 \cdot x)))$$

Neural network language model



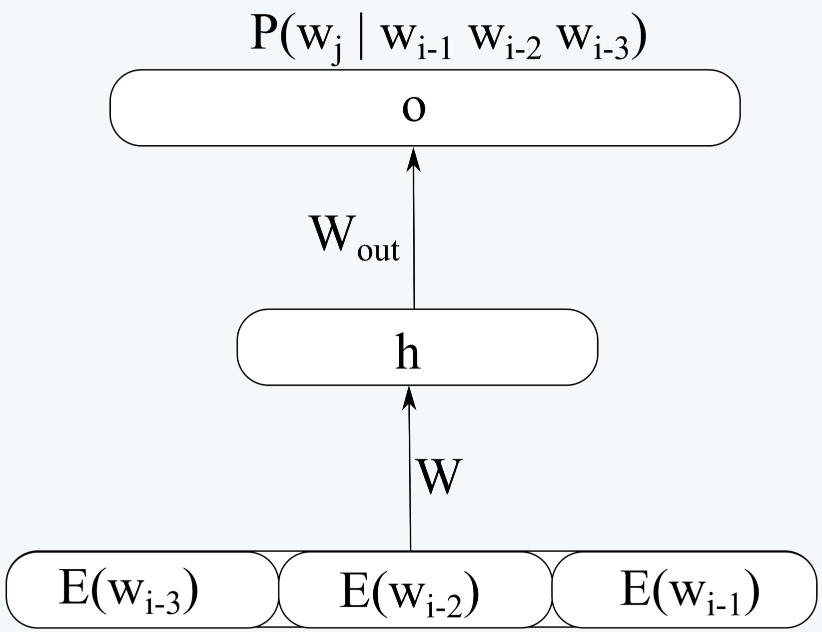
Input: vector representations of previous words

$E(w_{i-3}), E(w_{i-2}), E(w_{i-1})$

Output: The conditional probability of w_i being the next word

$P(w_i \mid w_{i-1}\ w_{i-2}\ w_{i-3})$

Neural network language model



We can also think of the input as a concatenation of the context vectors

The hidden layer h is calculated as in previous examples

How do we calculate $P(w_i \mid w_{i-1}\ w_{i-2}\ w_{i-3})$?

Softmax

- Takes a vector of values and squashes them into the range (0,1), so that they add up to 1
- We can use this as a probability distribution

$$softmax(z)_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

Softmax

$$softmax(z)_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

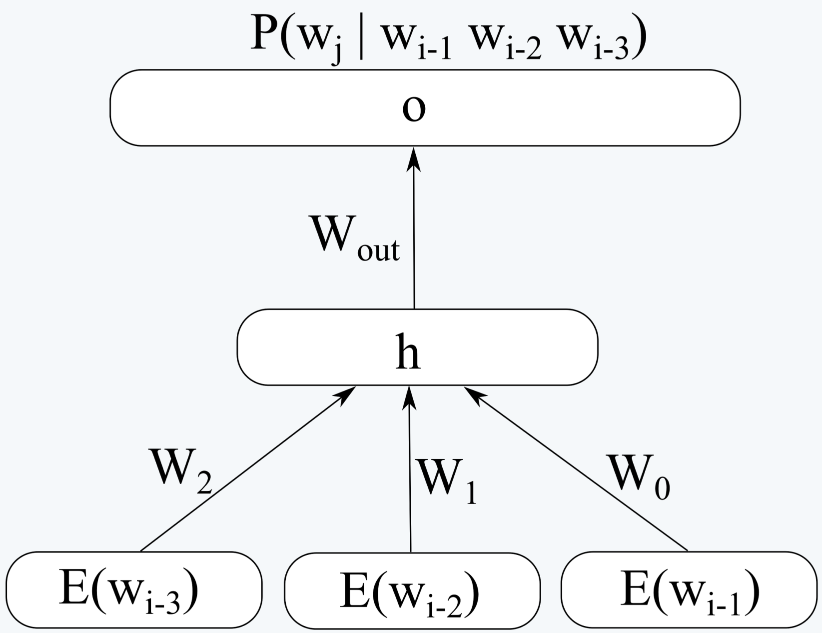
	0	1	2	3	SUM
z	2.0	5.0	-4.0	0.0	3
exp(z)	7.389	148.413	0.018	1.0	156.82
softmax(z)	0.047	0.946	0.000	0.006	~1.0

Softmax

$$softmax(z)_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

	0	1	2	3	SUM
z	-5.0	-4.5	-4.0	-6.0	-19.5
exp(z)	0.007	0.011	0.018	0.002	0.038
softmax(z)	0.184	0.289	0.474	0.053	1.0

Neural network language model

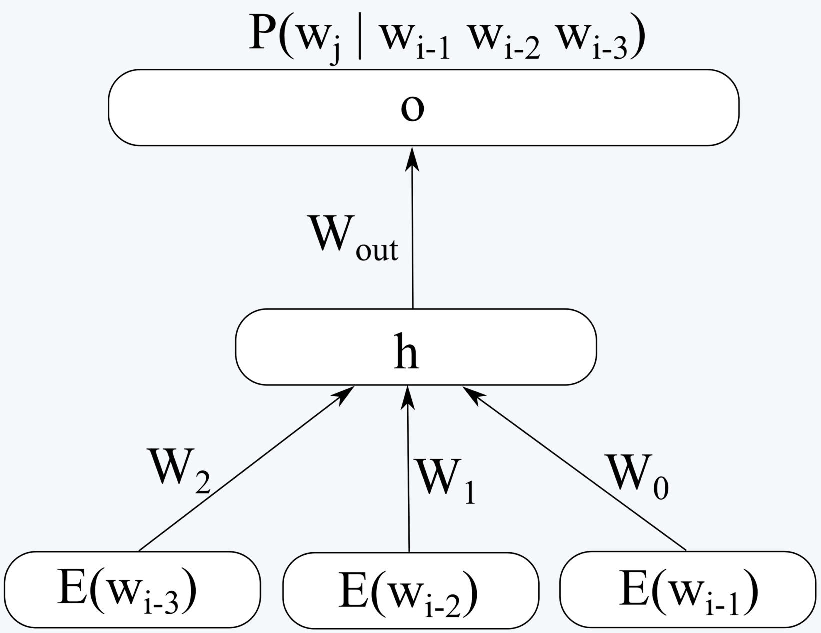


Our output vector o has an element for each possible word w_j

We take a softmax over that vector

The result is used as $P(w_i \mid w_{i-1}\ w_{i-2}\ w_{i-3})$

Neural network language model



1. Multiply input vectors with weights

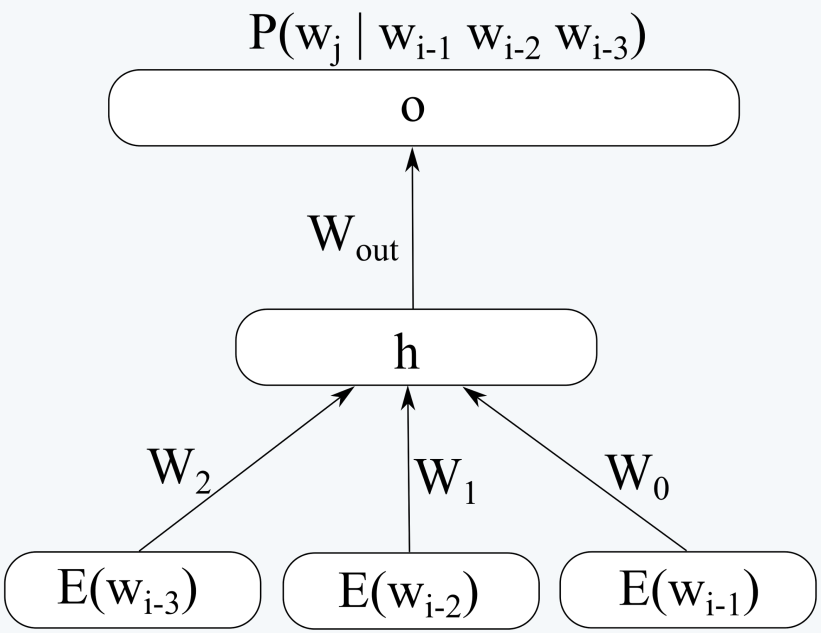
$$
\begin{aligned}
z = &W_2 \cdot E(w_{i-3}) \\
&+ W_1 \cdot E(w_{i-2}) \\
&+ W_0 \cdot E(w_{i-1})
\end{aligned}
$$

2. Apply the activation function

$$h = f(z)$$

Bengio et al. (2003)

Neural network language model



3. Multiply hidden vector with output weights

$$s = W_{out} \cdot h$$

4. Apply softmax to the output vector

$$o = softmax(s)$$

Now the j-th element in the output vector, o_j, contains the probability of w_j being the next word.

$$0 \leq j < V$$

NNLM example

Word embedding (encoding) matrix E

V = 4, M = 3

Each word is represented as a 3-dimensional column vector

$$\begin{array}{cccc} \textbf{Bob} & \textbf{often} & \textbf{goes} & \textbf{swimming} \\ \end{array}$$
$$\left|\begin{array}{cccc} -0.5 & -0.2 & 0.3 & 0 \\ 0.1 & 0.5 & -0.1 & -0.4 \\ 0.4 & -0.3 & 0.6 & 0.2 \end{array}\right|$$

NNLM example

The weight matrices going from input to the hidden layer

They are position-dependent

$$W_0 = \begin{vmatrix} 0.2 & -0.1 & 0.4 \\ -0.2 & 0.3 & 0.5 \\ 0.1 & 0 & -0.3 \end{vmatrix}$$

$$W_1 = \begin{vmatrix} 0 & -0.2 & 0.2 \\ 0.1 & 0.3 & -0.1 \\ -0.3 & 0.4 & 0.5 \end{vmatrix}$$

$$W_2 = \begin{vmatrix} -0.1 & 0.1 & -0.4 \\ 0.3 & 0 & 0.4 \\ -0.2 & 0.2 & -0.3 \end{vmatrix}$$

NNLM example

Output (decoding) matrix, W_{out}

Each word is represented as a 3-dimensional row vector

Bob	-0.4	-0.6	0.1
often	0.5	-0.2	-0.5
goes	-0.1	0	0.4
swimming	0.6	0.2	-0.3

NNLM example

1. Multiply input vectors with weights

$$
\begin{aligned}
z = & W_2 \cdot E(w_{i-3}) \\
& + W_1 \cdot E(w_{i-2}) \\
& + W_0 \cdot E(w_{i-1})
\end{aligned}
$$

$W_2E(w_{i-3})$	$W_1E(w_{i-2})$	$W_0E(w_{i-1})$	z
-0.1	-0.16	0.31	0.05
0.01	0.16	0.21	0.38
0	0.11	-0.15	-0.04

NNLM example

2. Apply the activation function

$$h = f(z)$$

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

$$\begin{array}{c} \mathbf{h} \\ \begin{vmatrix} 0.512 \\ 0.594 \\ 0.49 \end{vmatrix} \end{array}$$

NNLM example

3. Multiply hidden vector with output weights

$$s = W_{out} \cdot h$$

$$\begin{array}{c} \mathbf{s} \\ \begin{vmatrix} -0.512 \\ -0.108 \\ 0.145 \\ 0.279 \end{vmatrix} \end{array}$$

NNLM example

4. Apply softmax to the output vector

$$o = softmax(s)$$

	o
Bob	0.151
often	0.226
goes	0.291
swimming	0.333

P(Bob | Bob often goes) = 0.151

P(swimming | Bob often goes) = 0.333

References

Pattern Recognition and Machine Learning
Christopher Bishop (2007)

Machine Learning: A Probabilistic Perspective

Kevin Murphy (2012)

Machine Learning

Andrew Ng (2012)

https://www.coursera.org/course/ml

Using Neural Networks for Modelling and Representing Natural Languages

Tomas Mikolov (2014)

http://www.coling-2014.org/COLING%202014%20Tutorial-fix%20-%20Tomas%20Mikolov.pdf

Deep Learning for Natural Language Processing (without Magic)

Richard Socher, Christopher Manning (2013)

http://nlp.stanford.edu/courses/NAACL2013/

Extra materials

Entropy

The expectation of a discrete random variable X with probability $p(x)$

$$E(X) = \sum_{x} p(x) x$$

The expected value of a function $f(x)$ of a discrete random variable X with probability $p(x)$

$$E(f(x)) = \sum_{x} p(x) f(x)$$

Entropy

The entropy of a random variable X is the expected negative log probability

$$H(X) = -\sum_{x} p(x) log_2(p(x))$$

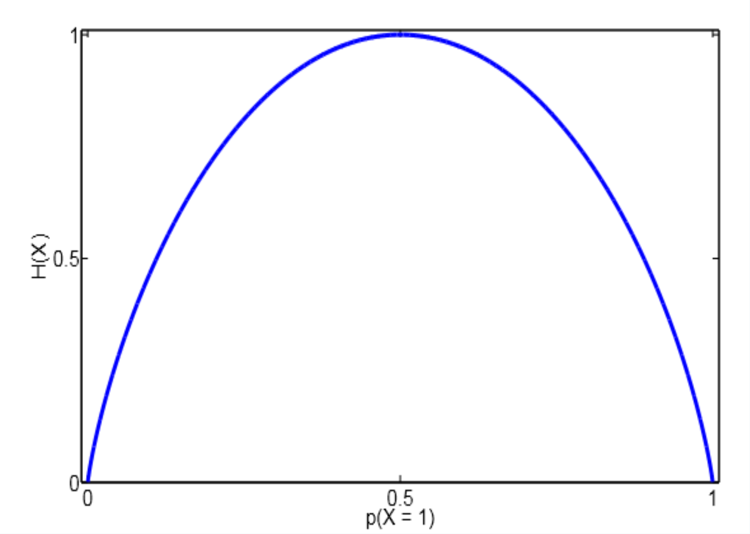
Entropy is a measure of uncertainty.

Entropy is also a lower bound on the average number of bits required to encode a message.

Entropy of a coin toss

A coin toss comes out heads (X=1) with probability **p**, and tails (X=0) with probability **1−p**.

$$H(X) = -\sum_x p(x) log_2(p(x))$$



1) p = 0.5

$$\begin{aligned} H(X) &= -(0.5 log_2 0.5 + 0.5 log_2 0.5) \\ &= -(-0.5 - 0.5) = 1 \end{aligned}$$

2) p = 1.0

$$H(X) = -(0 log_2 0 + 1 log_2 1)$$

$$= -(0 + 0) = 0$$

Cross entropy

The cross-entropy of a (true) distribution p^* and a (model) distribution p is defined as:

$$H(p^*, p) = -\sum_x p^*(x) log_2 p(x)$$

$H(p^*,p)$ indicates the avg. number of bits required to encode messages sampled from p* with a coding scheme based on p.

Cross entropy

We can approximate H(p*,p) with the normalised log probability of a single very long sequence sampled from p.

$$H(p^*, p) = \lim_{n \leftarrow \infty} -\frac{1}{n} log_2 p(w_1 \ldots w_n)$$

$$\approx -\frac{1}{N} log_2 p(w_1 \ldots w_N)$$

Perplexity and entropy

$$PP(w_1 \ldots w_N) = 2^{H(w_1 \ldots w_N)}$$

$$= 2^{-\frac{1}{N} log_2 p(w_1 \ldots w_N)}$$

$$= p(w_1 \ldots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{p(w_1 \ldots w_N)}}$$

Perplexity example

Text: *natural language processing*

w	p(w | <s>)
processing	0.4
language	0.3
the	0.17
natural	0.13

w	p(w | natural)
processing	0.4
language	0.35
natural	0.2
the	0.05

w	p(w | language)
processing	0.6
language	0.2
the	0.1
natural	0.1

What is the perplexity?

$$PP(W) = \sqrt[N]{\frac{1}{\prod_{i=1}^{N} P(w_i|w_{i-1})}} \qquad PP(\text{natural language processing}) = \sqrt[3]{\frac{1}{0.13 \times 0.35 \times 0.6}} = 3.32$$

And entropy?

$$PP(w_1 \ldots w_N) = 2^{H(w_1 \ldots w_N)} \qquad H(\text{natural language processing}) = log_2(3.32) = 1.73$$