word2vec

LING 570

Fei Xia

(Some contents are from McCormick’s blog)

Many methods for learning word embeddings

- Word2vec (2013)
- GloVe (2014)
- fastText (2016)
- …
- See a list of word embeddings at http://ahogrammer.com/2017/01/20/the-list-of-pretrained-word-embeddings/

Word2vec papers

1) Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In Proceedings of Workshop at ICLR, 2013.

2) Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In Proceedings of NIPS, 2013.

3) Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic Regularities in Continuous Space Word Representations. In Proceedings of NAACL HLT, 2013.

- A good blog:
  http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Many implementations

- In C:
  - https://code.google.com/p/word2vec/ (the original one)
  - https://github.com/dav/word2vec (on patas dropbox/17-18/570/hw11)

- In Java:
  - http://deeplearning4j.org/word2vec.html
  - https://github.com/medallia/Word2VecJava

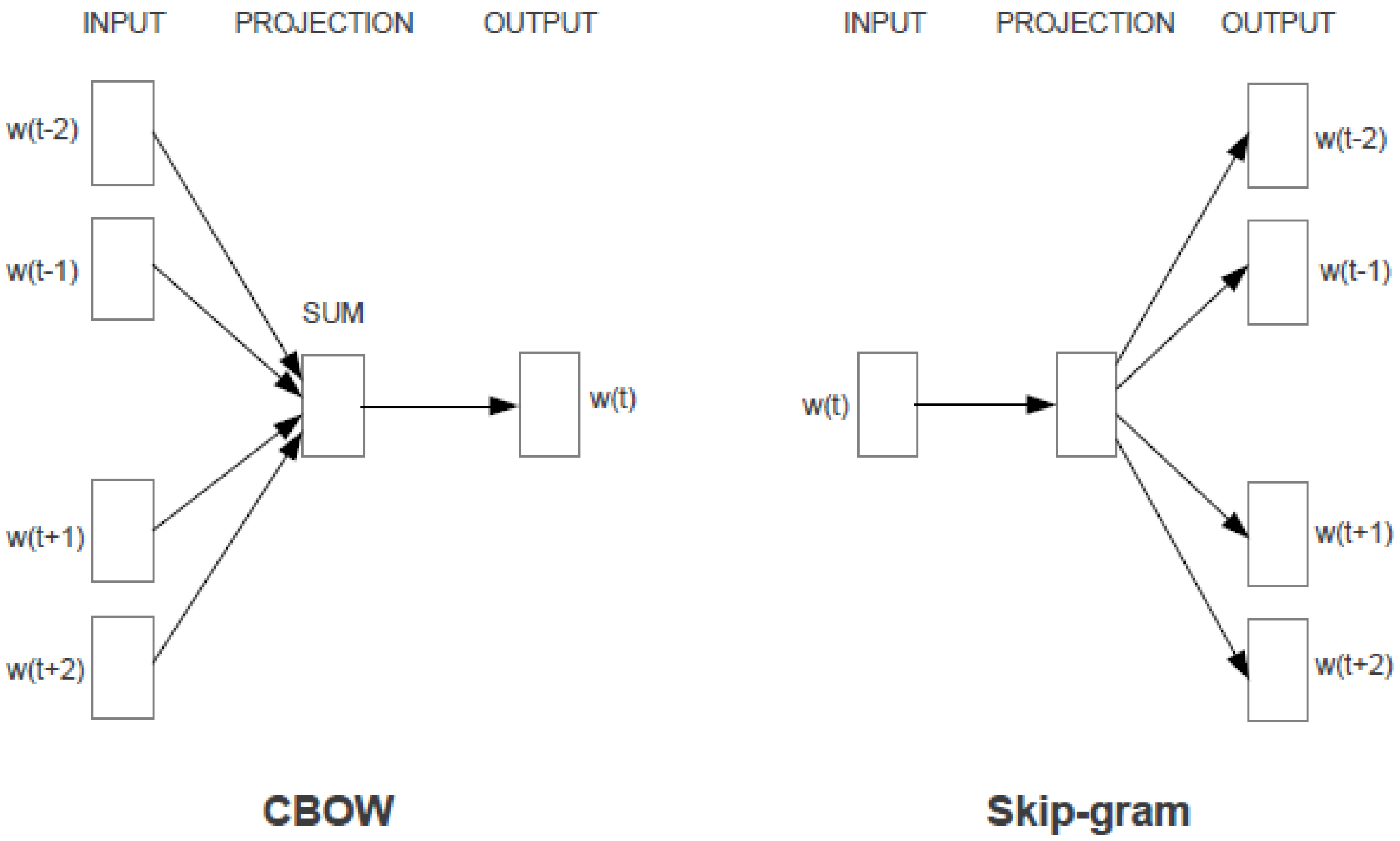
- In python:
  - https://rare-technologies.com/deep-learning-with-word2vec-and-gensim/ (a python library)

Intuition behind word2vec

- To train a simple neural network with a single hidden layer NN.
  - But we are not going to use the NN for the task we trained it on.
  - All we care are the weights of the hidden layers.
  - Training: use stochastic gradient descent and backpropagation

- Two models:
  - Continuous bag-of-word (CBOW): predict current word using the neighbor words
  - Continuous skip-gram model: predict neighbor words using the current word

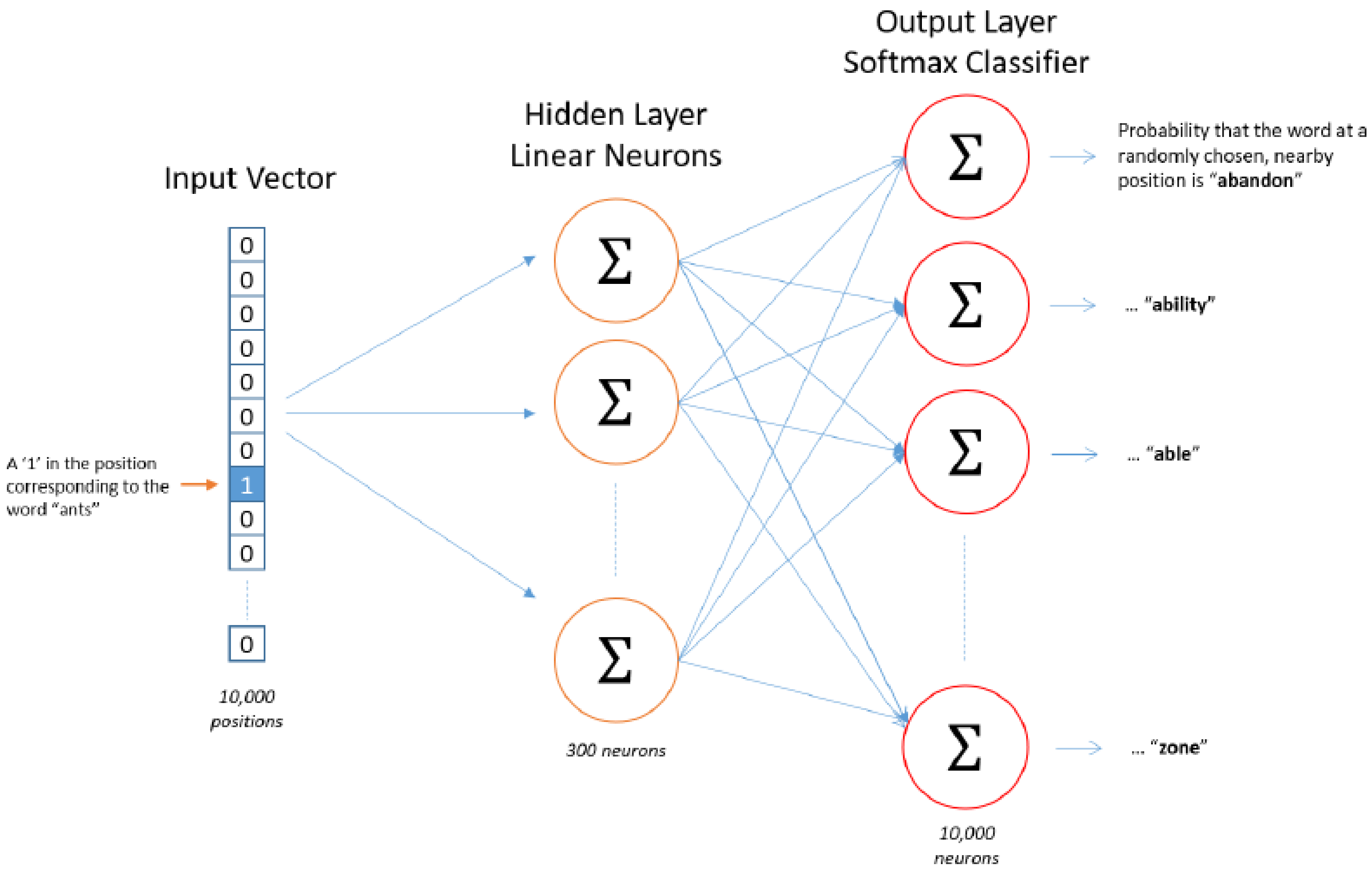
Two word2vec models (Mikolov et al., ICLR 2013)



The fake task for skip-gram

- Task: Given a specific word w1 (aka the input word) in the middle of a sentence, pick a nearby word at random, what’s the probability that w2 is chosen?
  - The input is a word
  - The output is a probability distribution
  - “nearby”: window size

- Training data: a large corpus of text (e.g., 100B words)



Input Vector
Hidden Layer
Linear Neurons
Output Layer
Softmax Classifier
0
0
0
0
0
0
0
1
0
0
0
A '1' in the position corresponding to the word "ants"
10,000 positions
300 neurons
10,000 neurons
Probability that the word at a randomly chosen, nearby position is "abandon"
… "ability"
… "able"
… "zone"

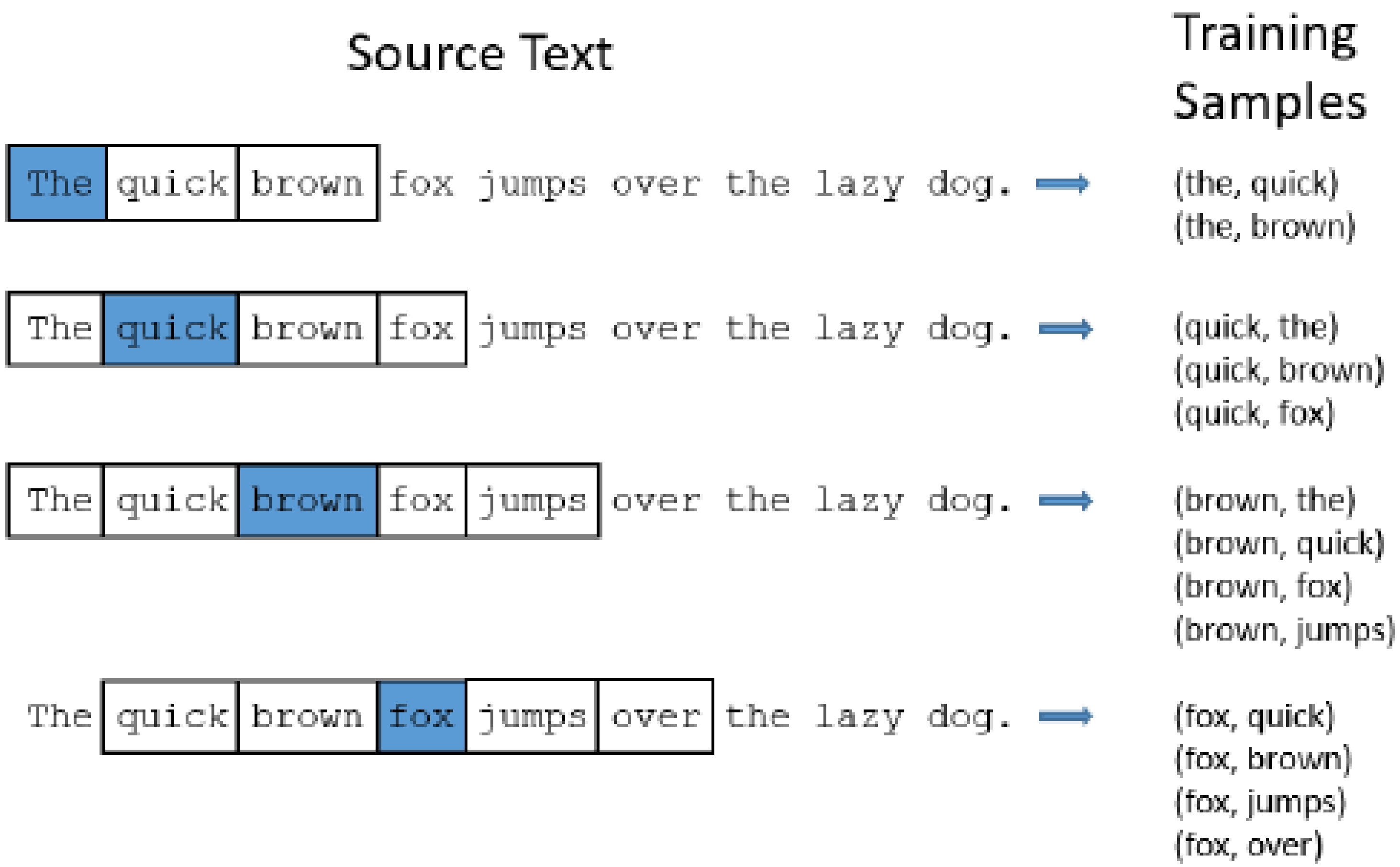
The NN

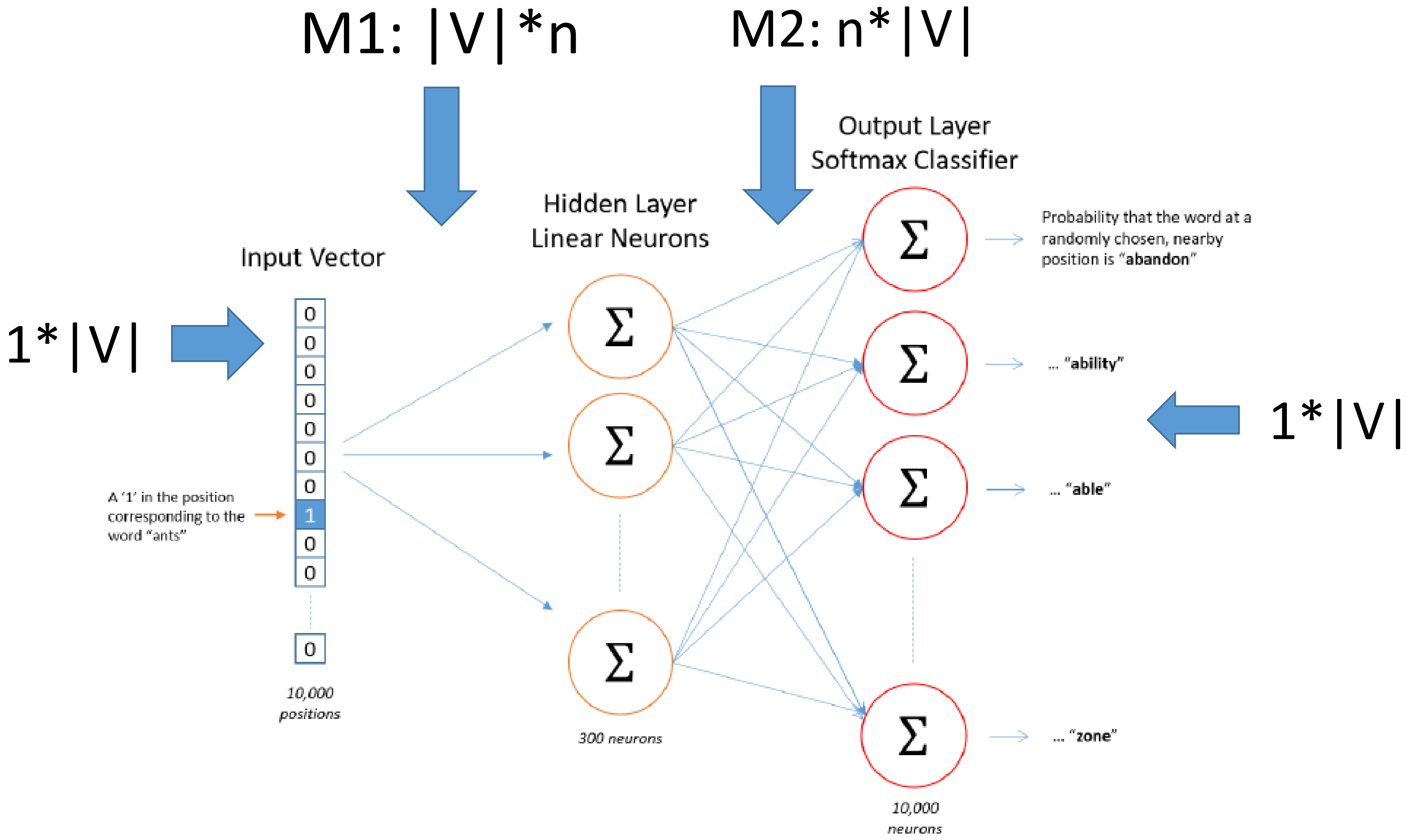
- Input layer:
  - |V| neurons
  - Each input word is represented as a one-hot vector: one dimension is 1, the rest are all zeros.

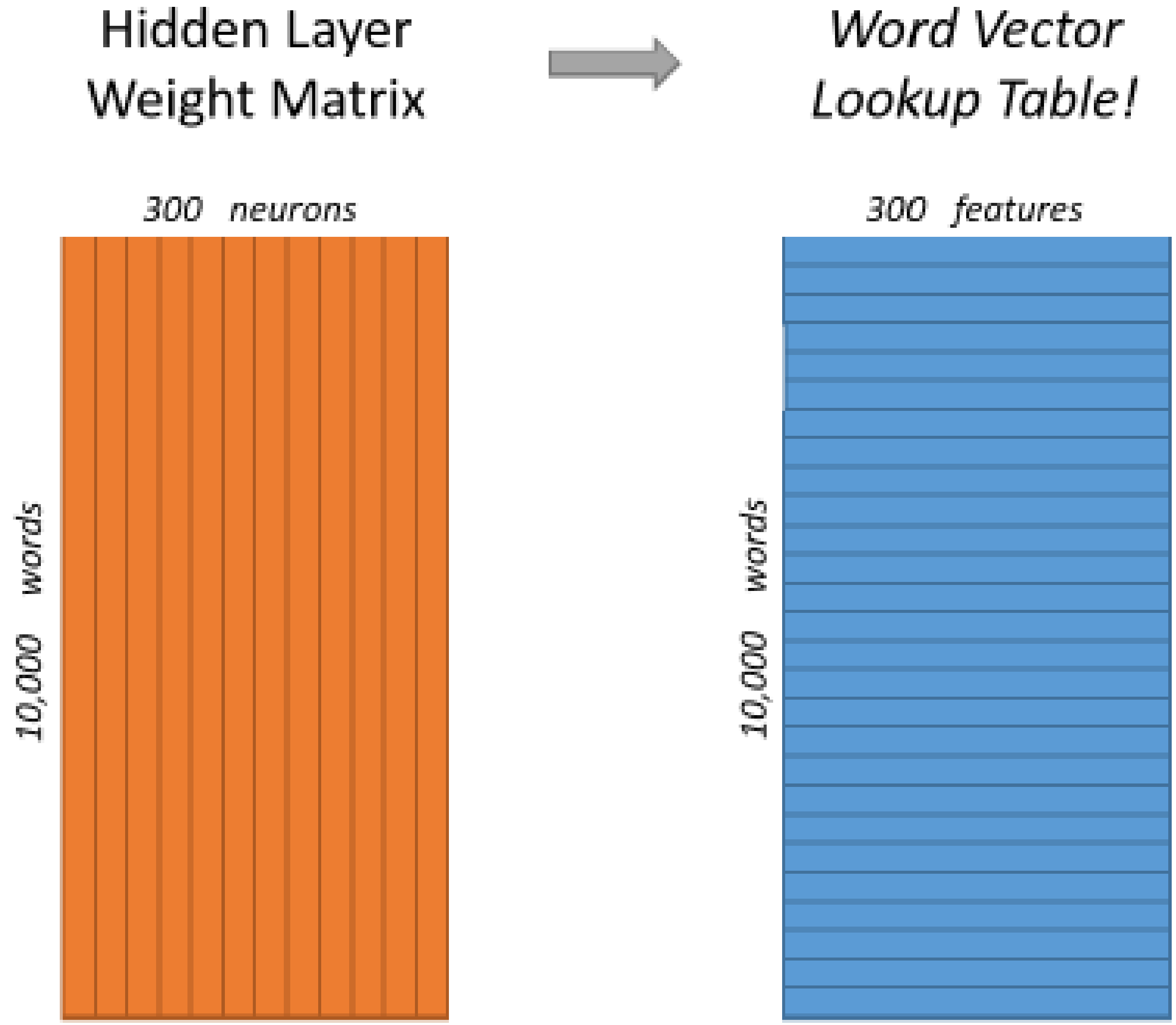
- Hidden layer:
  - # of neurons = # of dimensions in word embeddings
  - Use linear neuron ("no activation function"): i.e., $y = z$

- Output layer:
  - |V| neurons: the output of each neuron is [0,1]
  - Output neurons use softmax: so the output vector is a probability distribution

Training examples



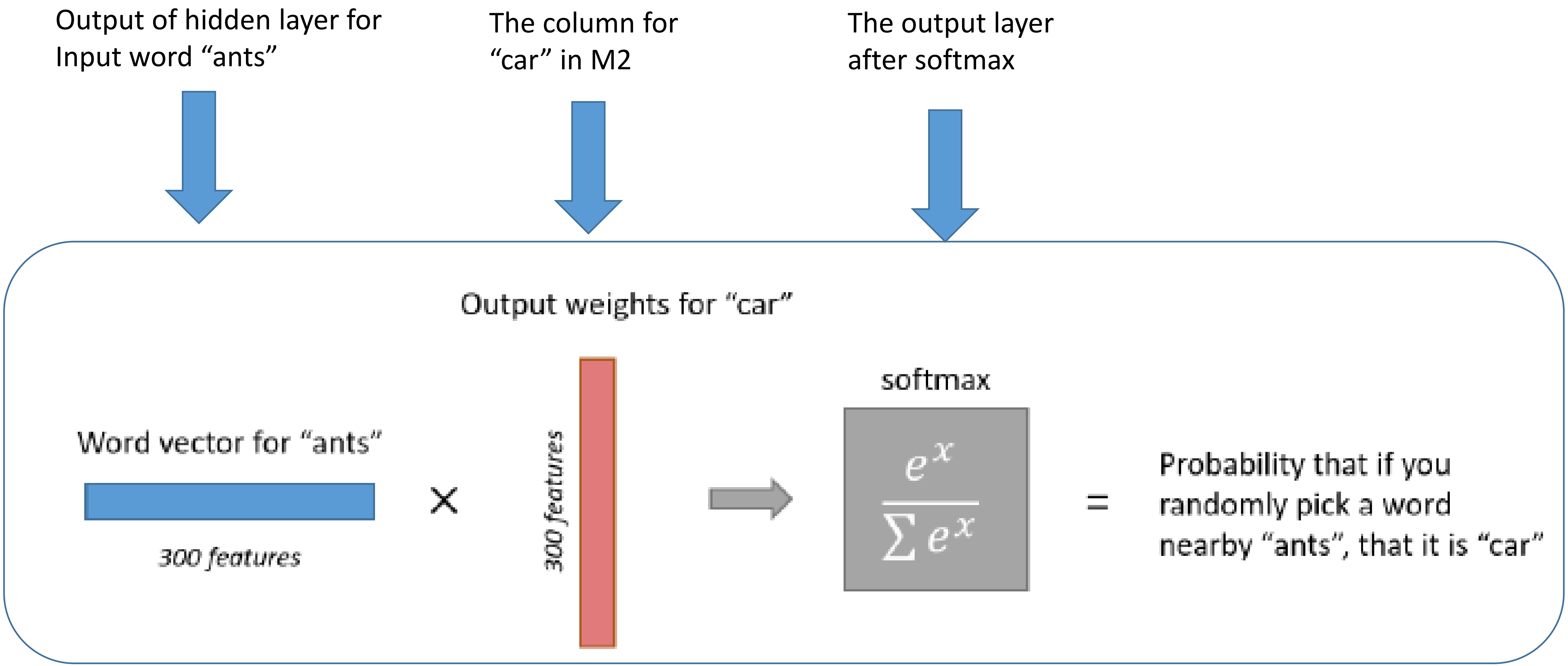

M1: |V|*n
M2: n*|V|
Output Layer
Softmax Classifier
Hidden Layer
Linear Neurons
Input Vector
1*|V|
Probability that the word at a randomly chosen, nearby position is "abandon"
... "ability"
... "able"
... "zone"
1*|V|
A '1' in the position corresponding to the word "ants"
0
0
0
0
0
0
0
1
0
0
0
10,000 positions
300 neurons
10,000 neurons


Hidden Layer
Weight Matrix
Word Vector
Lookup Table!
300 neurons
10,000 words
300 features
10,000 words

Input one-hot vector $\qquad$ M1 $\qquad$ Output of the hidden layer

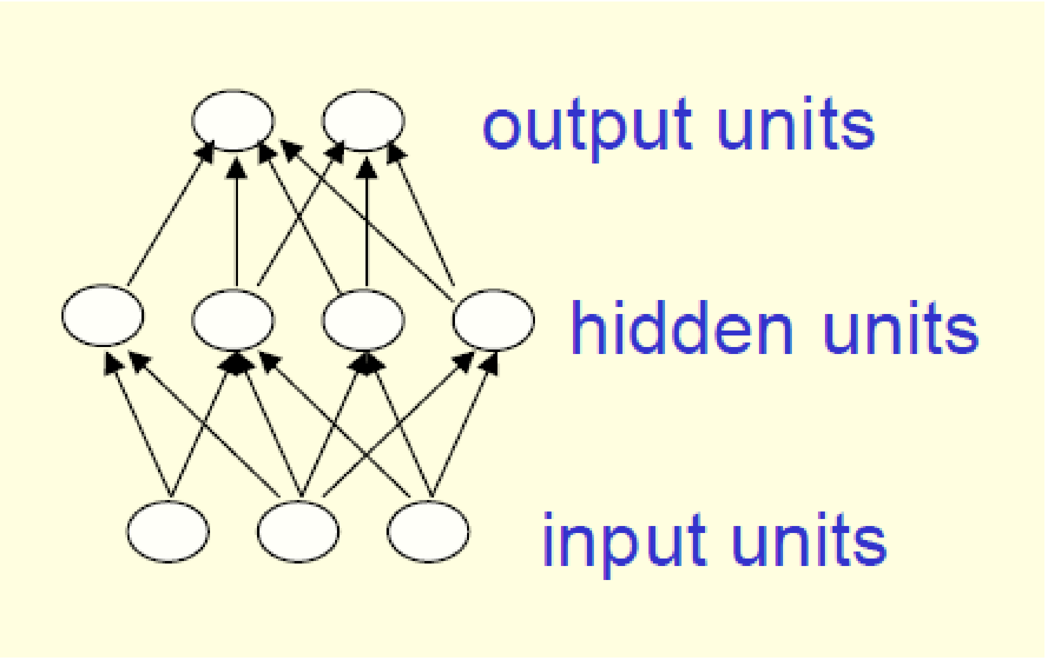
$$\begin{bmatrix}0 & 0 & 0 & \boxed{1} & 0\end{bmatrix} \times \begin{bmatrix}17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ \boxed{10 \quad 12 \quad 19} \\ 11 & 18 & 25\end{bmatrix} = \begin{bmatrix}10 & 12 & 19\end{bmatrix}$$

The output of the hidden layer is just the word vector (or word embedding) of the input word.

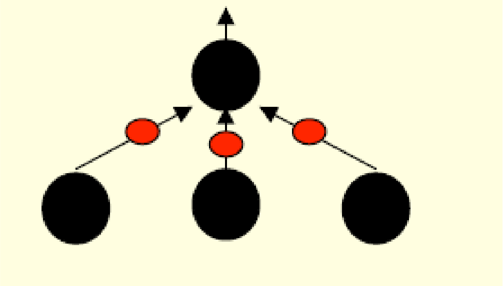

Output of hidden layer for
Input word “ants”
The column for
“car” in M2
The output layer
after softmax
Output weights for “car”
Word vector for “ants”
300 features
×
300 features
softmax
$\frac{e^x}{\sum e^x}$
=
Probability that if you
randomly pick a word
nearby “ants”, that it is “car”

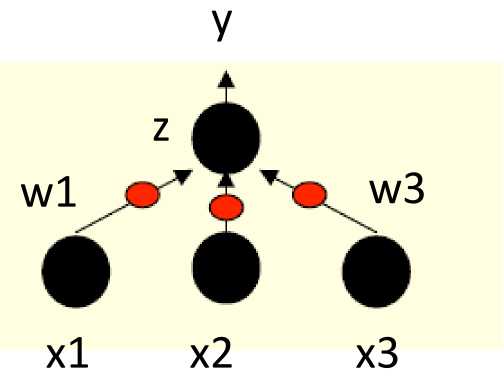
Feed-forward neural network

- This is the simplest type of NN:
  - The first layer is the input and the last layer is the output
  - If there is more than one hidden layer, we call them deep NN
- Training: learn the weights on the arcs, using back propagation.



Learn the weights of linear neuron





bias

i th input

$$y = b + \sum_i x_i w_i$$

output

index over input connections

weight on

i th input

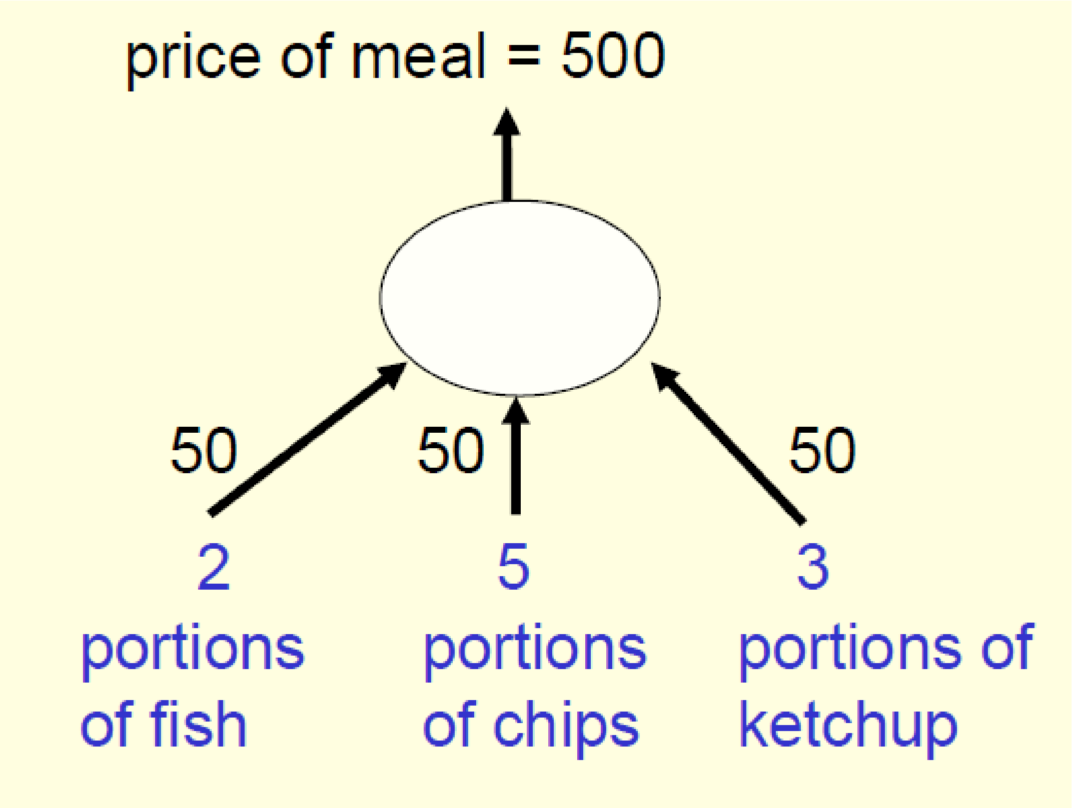
weight vector

$$y = \sum_i w_i x_i = \mathbf{w}^T \mathbf{x}$$

neuron's estimate of the desired output

input vector

The toy example



- Residual error = 350
- The "delta-rule" for learning is:
  $$\Delta w_i = \varepsilon \, x_i \, (t - y)$$
- With a learning rate ε of 1/35, the weight changes are +20, +50, +30
- This gives new weights of 70, 100, 80.
  - Notice that the weight for chips got worse!

➔ Repeat until the residual error is small enough

Time complexity for training

Time complexity is O(E*T*Q):

- E is the number of the training epochs
- T is the number of word tokens in the training set
- Q: determined by the model architecture (e.g., sizes of the two matrices)

Summary

- There are many ways to learn word embeddings.

- Word2vec is one of the earliest and most well-known method:
  - Creating “fake” tasks and use the weights from the models
  - There are two models: CBOW and Skip-gram
  - For both, use feed-forward NNs with linear neurons (to make learning faster)

- There are many implementations:
  - Speed is a big issue.
  - Many tricks to make training faster: e.g., https://rare-technologies.com/word2vec-in-python-part-two-optimizing/

Additional slides

Tricks for Skip-Gram training (Mikolov et al., 2013-NIPS)

- Tricks:
  - Dealing with phrases
  - Negative sampling
  - Subsampling frequent words
  - Hierarchical softmax

- The first three tricks are explained at the blog

http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/

Frequent words in the training data

In the sentence: “The fox jumps over the fence”

There are two “problems” with common words like “the”:

1. When looking at word pairs, (“fox”, “the”) doesn’t tell us much about the meaning of “fox”. “the” appears in the context of pretty much every word.

2. We will have many more samples of (“the”, …) than we need to learn a good vector for “the”.

Subsampling frequent words

- For each word we encounter in our training text, there is a chance that we will effectively delete it from the text. The probability that we cut the word is related to the word’s frequency.

- If we have a window size of 10, and we remove a specific instance of “the” from our text:
  - As we train on the remaining words, “the” will not appear in any of their context windows.
  - We’ll have 10 fewer training samples where “the” is the input word.