Word 2 Vec in C ++ documentation

How it works

Step-1 words are converted into one hot vectors

Eg: the - 0, 0, 0, 0, 1, 0, 0 . . . . .

the size of the one hot vector depends on the number of unique words in the corpus (data set).

Step-2 the one hot vectors are given to a neural network

Step -3

A weight matrix W_0 is initialized with random weights

Step -4

the one hot vector is multiplied with the weighted matix

Step - 5

the output matix is run through a softmax layer and the output is taken

Step - 6

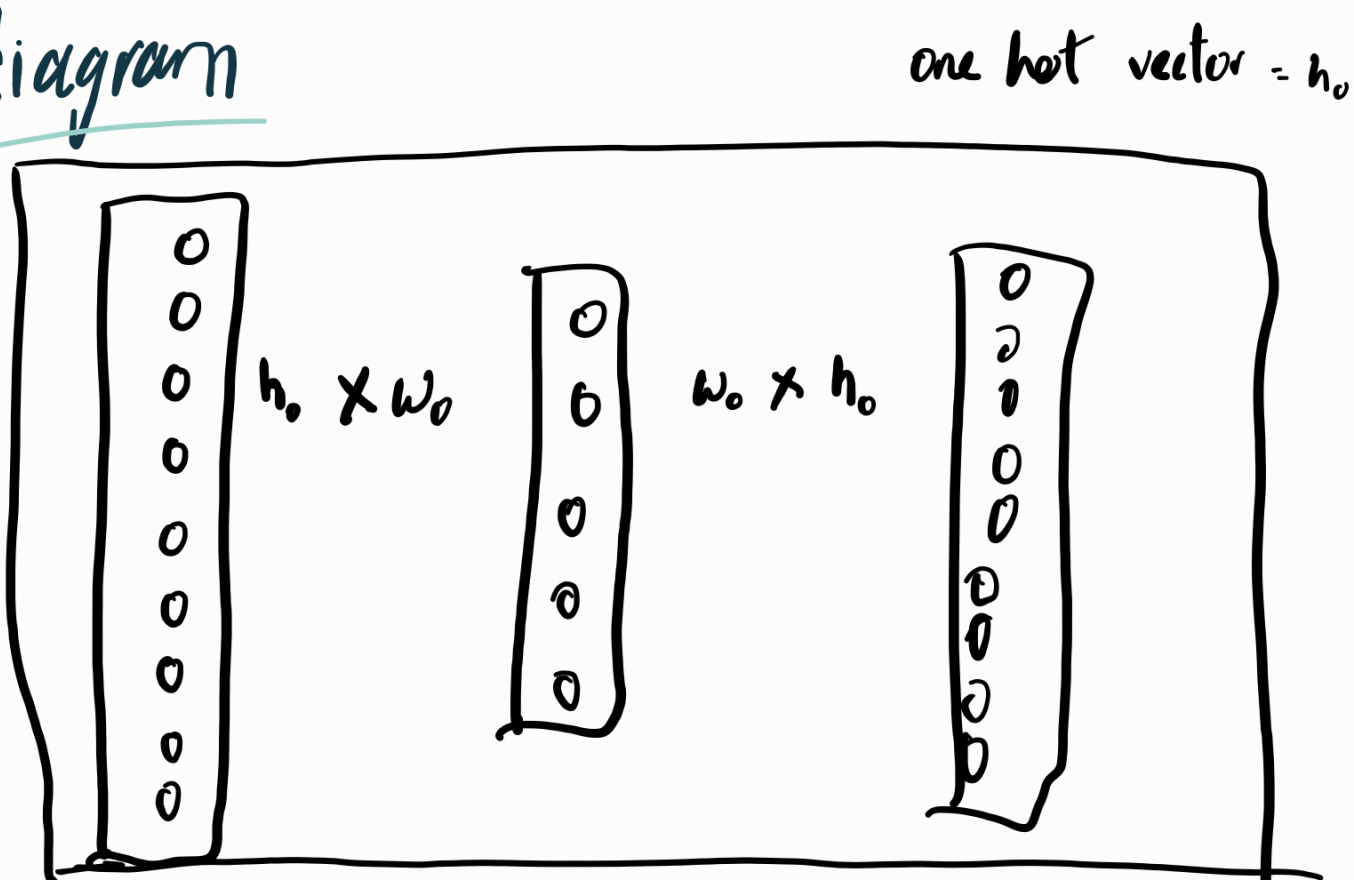
the output is compared to the required out but.

Back propogation is performed and the weight matix is adjusted.

step - 7

the weight matix is the embedding value of the word.

diagram



in detail code explained

The corpus used here is the book song of ice & fire by george R. R Martin (cool55b.txt). The file is in a .txt format.

i) tokenize Text : this function reads the file and splits in into tokens.

```
while (file >> word)
```

splits input by white spaces

Eg: Hello, world is one word

if (ispunct(c))

checks if the character is a punctuation

if punctuation the word in currentToken is added to tokenWords

else

it is converted to lowercase and added to tokenWords

Eg: Hello! How's NLP going? Let's tokenize_text

output: "hello" "hows" "nlp" "going" "lets" "tokenizetext"

why ignore punctuation - how's → hows

else how's → "how", "s"

ii) Build Vocabulary

In this function an unordered map called vocab is initialized.

each word is given an index number

Why use an unordered_map — faster search

best case $O(1)$

worst case $O(n)$

iii) create Context Target Pairs Indexed

CBOW algorithm is being used here.

CBOW stands for Continuous Bag Of Words

How CBOW works -

Sentence - My name is Eshvar Baliji

Window size = 2

target word = is

context words - My, name, Eshvar, Balaji

target word = balaji

context words - is, Eshvar

This is what is happening in this code.

Here the context words & the target word are stored as context_target_pairs

Why CBOW and not Skip-Gram?

CBOW shows better results than Skip-Gram on small datasets.

CBOW - predicts the target word

Skip-Gram - predicts the context words given the target word.

iv) initialized Weights

srand(time(0)) - random number generator

here W_1 & W_2 = [— vocab size —] × embedding dim

random values are assigned to w_1 & w_2

v) softmax

convert all the logits (raw scores) into probability distributions b/w 0 to 1.

First max score in the vector is found.

exp_score[i] = exp(score(i) - max_score)

here score - max_score

then exponent is done

then the value is added to sum_exp

then val is divided by sum_exp
↳ is exp_score

vi) forward Pass

W1 = vocab_size x embedding_dim

W2 = embedding_dim x vocab_size

One hot vector x W_1

then

W2 x one hot vector

then

logits are applied to the softmax to get probability

vii) trainModel

epochs are initialized here

learning_rate is initialized here

here the model trained.

ie the embedding matrix is

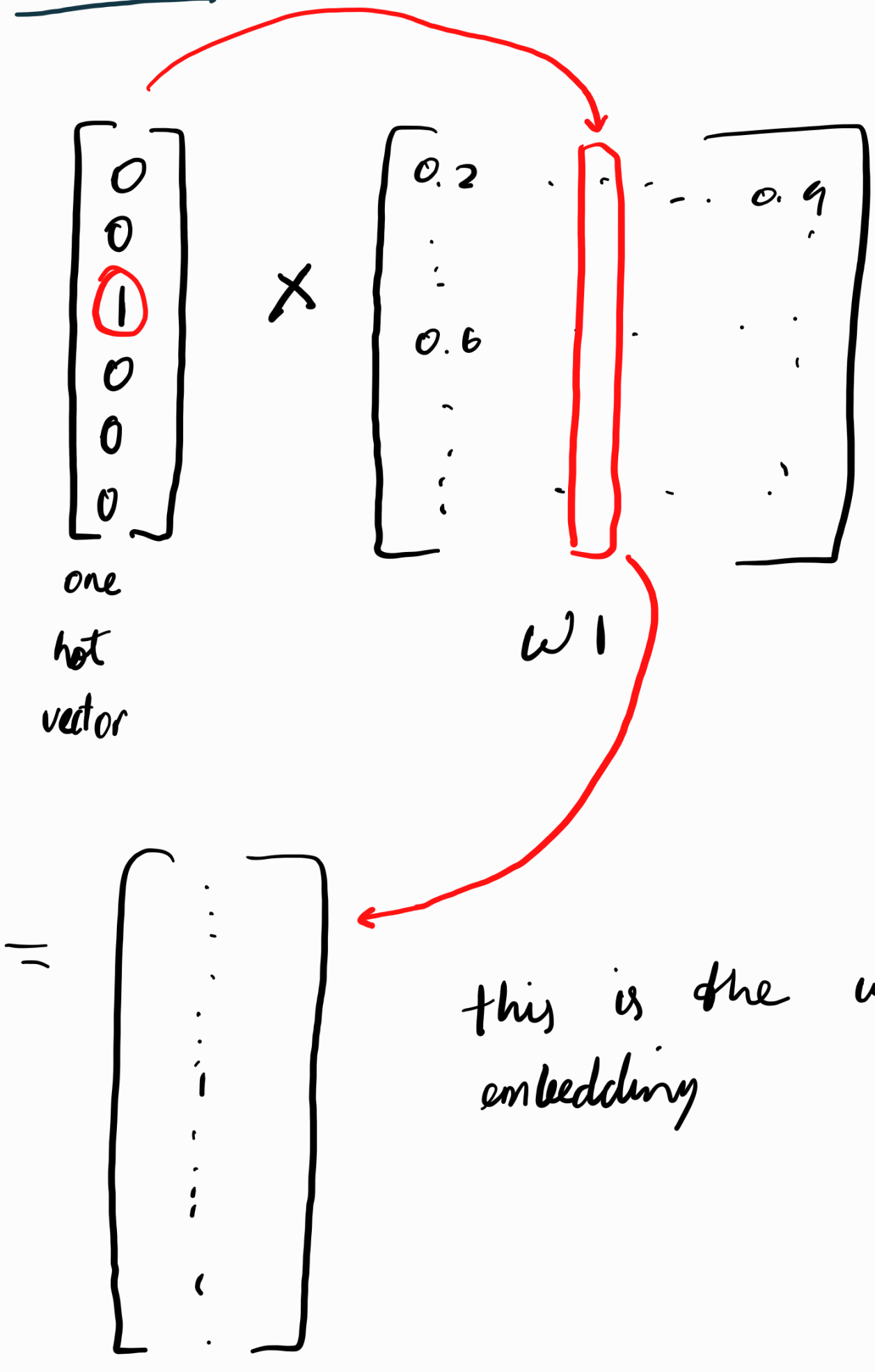
built per word.

viii) getWordEmbedding

this function gets embedding vector

for a word from w1

Math



Word2Vec : It is a techinque in NLP designed to convert words into dense numerical vectors.

These embeddings capture semantic & contextual relationship b/w words.

semantic relationship : The word king & queen is placed close to each other

contextual relationship : "cat" & "dog" both placed near "pet".

or - king - man + woman ≃ queen

embedding improve performance in tasks like:

Machine translation (understanding word meaning)

Sentiment analysis (capturing context)

Information retrival (matching related terms)