Word Embeddings using Continuous Bag of Words (CBOW) model

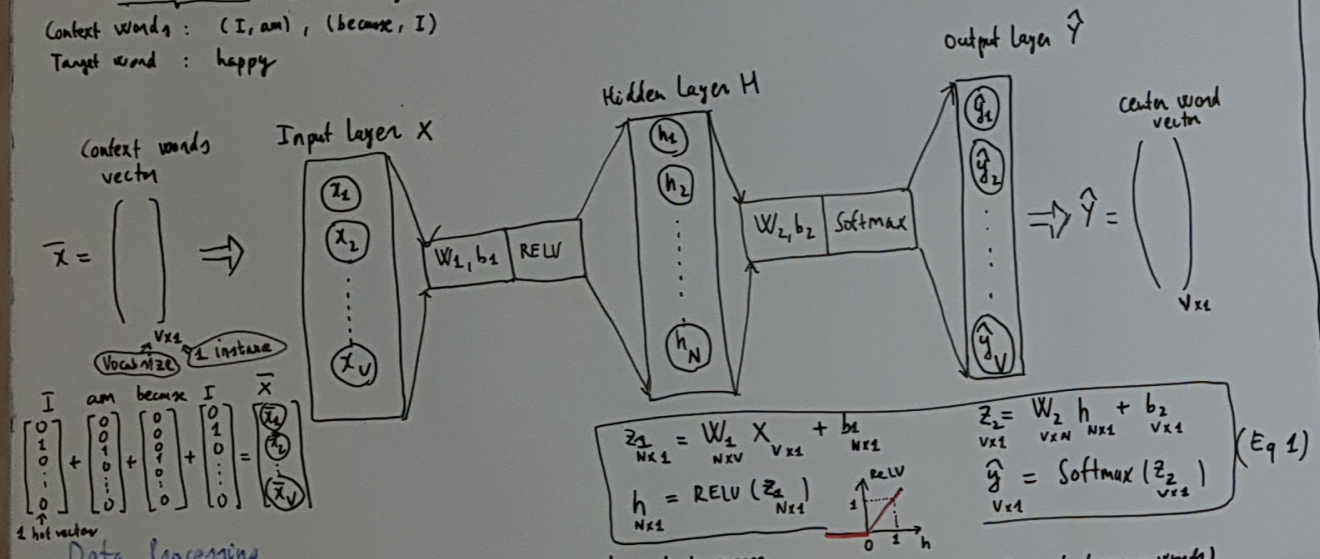
I. CBOW model: try to predict a center word given a few context words

Ex. "I am happy because I am learning"

↳ Context half-size = 2 and target is to predict the word "happy"

Context words: (I, am), (because, I)

Target word: happy



- Data Processing

Doc "shakespeare.txt" → punctuations → tokenize → change to lowercase.

↳ data = [... a list of words ...] with size ≈ 67,000

↳ unique words ≡ size of vocab ≈ 5700

↳ word counts = {'the': 1500, '.': 9000, ...}

↳ word2ind = {..., 'king': 2745, ..., 'kindness': 2743, ...}

ind2word = {..., 2745: 'king', ..., '2743': 'kindness', ...}

V: size of vocab (unique words)

N: size of hidden layer (≡ size of word embedding vector) (it is a hyper parameters)

II. Training the Model

(1) def initialize_model(N, V, random_seed = 1)

W_1, W_2, b_1, b_2 in range [0, 1]

Python: numpy.random.rand(size_of_matrix).

(2) def softmax(z): $z = \begin{bmatrix} [\,\vdots\,]_{V\times 1} & [\,\vdots\,]_{V\times 1} & \cdots & [\,\vdots\,]_{V\times 1} \end{bmatrix}_{V\times m}$

$\hat{y} = \frac{e^{z_i}}{\sum_{i=0}^{V-1} e^{z_i}} \Rightarrow \frac{np.exp(z)}{np.sum(np.exp(z), axis=0)}$

(number of instances for batch size)

(3) def forward_prop(X, W_1, W_2, b_1, b_2):

see Eq. 1.

$z_1 = np.dot(W_1, X) + b_1$

$z_1[z_1 < 0] = 0$ ← it is h in (Eq. 1)

$h = z_1$ (for clean only)

$z_2 = np.dot(W_2, h) + b_2$

return z_2, h

(4) def compute_cost(y, $\hat{y}$, batch_size):

logprobs = np.multiply(np.log($\hat{y}$), y) + np.multiply(np.dot(1 − $\hat{y}$), 1 − y)

$cost = -\frac{1}{batch\_size}$ np.sum(logprobs)

cost = np.squeeze(cost)

return cost.

(5) def back_prop(X, $\hat{y}$, y, h, W_1, W_2, b_1, b_2, batch_size); (≡ m)

$\frac{\partial J}{\partial W_1} = RELU(W_2^T(\hat{y} - y))\, X^T / m$

$\frac{\partial J}{\partial W_2} = (\hat{y} - y)\, h^T / m$

$\frac{\partial J}{\partial b_1} = \frac{1}{m} RELU(W_2^T(\hat{y} - y)) 1_m^T = np.sum(RELU(W_2^T(\hat{y} - y)), axis=1, keepdims=True)/m$

$\frac{\partial J}{\partial b_2} = \frac{1}{m}(\hat{y} - y) 1_m^T = np.sum(\hat{y} - y, axis=1, keepdims=True)/m$

return $\frac{\partial J}{\partial W_1}$, $\frac{\partial J}{\partial W_2}$, $\frac{\partial J}{\partial b_1}$, $\frac{\partial J}{\partial b_2}$

(6) def gradient_descent(data, word2Ind, N, V, num_iters, α = 0.03)

doc (x.txt)

W_1, W_2, b_1, b_2 ← from (1)

batch_size = 128

iters = 0

C = 2 ← context half size

for x, y in get_batches(data, word2Ind, V, C, batch_size):

z_1, h ← from (3)

$\hat{y}$ ← from (2)

cost ← from (4) ← print (cost) to check convergence

$\frac{\partial J}{\partial W_1}, \frac{\partial J}{\partial W_2}, \frac{\partial J}{\partial b_1}, \frac{\partial J}{\partial b_2}$ ← from (5)

$W_1 -= \alpha * \frac{\partial J}{\partial W_1}$

$W_2 -= \alpha * \frac{\partial J}{\partial W_2}$

$b_1 -= \alpha * \frac{\partial J}{\partial b_1}$

$b_2 -= \alpha * \frac{\partial J}{\partial b_2}$

} update weights and biases

iters += 1

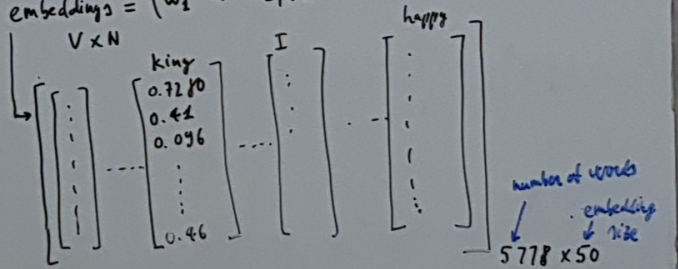
if iters > num_iters: break } Finish training condition

if iters % 100 == 0: α *= 0.66 } Reduce learning rate

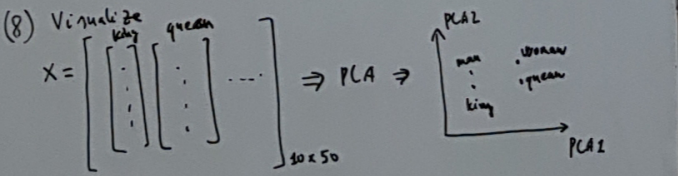
return W_1, W_2, b_1, b_2

(7) Get word embedding after Training Neural Network

embeddings = $(W_1^T + W_2)/2$



(8) Visualize



(9) Predict center word

I want ? go to school

X

⇒ Feed it to Neural Network (drawn above) ⇒ $\hat{y}$ (5a x 1) ⇒ Cosine Similarity with embedding words ⇒ The most similar is "to" ⇒ get "to"