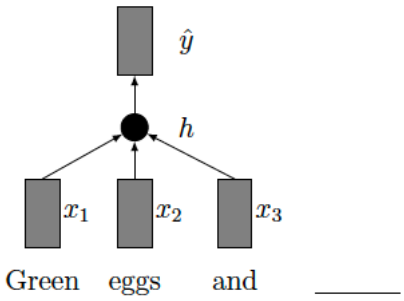
CLASS Exercises: WORD2VEC

Exercise 1

A feed-forward neural network language model (LM) is an alternative architecture for training word vectors. This architecture focuses on predicting a word given the N previous words. This is done by concatenating the word vectors of N previous words and use them as input of a single hidden layer of size H with a non-linearity (e.g. tanh). Finally, a softmax layer is used to make a prediction of the current word. The size of the vocabulary is V. The model is trained using a cross entropy loss for the current word.

Let the word vectors of the N previous words be $x_1; x_2; \ldots x_N$, each a column vector of dimension D, and let y be the one-hot vector for the current word. The network is specified by the equations that follow these lines:



$$x = \begin{bmatrix} x_1 \\ x_2 \\ \ldots \\ x_N \end{bmatrix}$$

$$h = \tanh(Wx + b)$$

$$\hat{y} = softmax(Uh + d)$$

$$J = CE(y, \hat{y})$$

$$CE = -\sum_i y_i \log(\hat{y}_i)$$

The dimensions of our parameters and variables are $x \in \mathbb{R}^{(N \cdot D)}, W \in \mathbb{R}^{Hx(N \cdot D)}, b \in \mathbb{R}^H, h \in \mathbb{R}^H, U \in \mathbb{R}^{V\,x\,H}, d \in \mathbb{R}^V, \hat{y} \in \mathbb{R}^V$

1a. Mention 2 important differences between this feed-forward neural network LM and the CBOW model. Explain how these differences might affect the word vectors obtained.

1b. Compute the complexity of forward propagation in a feed-forward LM for a single training example. Propose at least one way to change the model that would reduce this complexity.

SOLUTION

1a. The CBOW is trained to predict a center word given a context window that extends on both sides, while word vectors learned by NNLM do not capture the context to the right of the word.

The CBOW model simply uses the sum of context words, while the NNLM model combines context words non-linearly. Thus the NNLM can learn to treat “not good to" differently from “good to not", etc.

1b The forward propagation complexity for an NNLM is N x D for concatenating the word vectors, N x D x H to compute h and H x V to compute $\hat{y}$ from h: in total, O(NDH + HV). Typically, V >> ND, so the latter term dominates the forward propagation computation.

The complexity can be reduced by using negative sampling to compute the softmax or using the hierarchical softmax.

Exercise 2.

2a. We know that dense word vectors like the ones obtained with word2vec or GloVe have many advantages over using sparse one-hot word vectors. Name a few.

2b. Also name at least 2 disadvantages of sparse vectors that it are not solved in dense vectors. Which of the following is NOT an advantage dense vectors have over sparse vectors?

SOLUTION

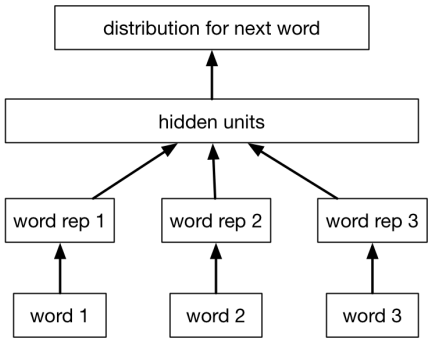
2a Models using dense word vectors generalize better to rare words than those using sparse vectors.
Dense word vectors encode similarity between words while sparse vectors do not.
Dense word vectors are easier to include as features in machine learning systems than sparse vectors.

2b. Just like sparse representations, word2vec or GloVe do not have representations for unseen words and hence do not help in generalization.

Also, there is only one representation per word, so polysemy is not solved.

Exercise 3

Given the following neural architecture. What is it learning? Can you explain which exact NLP task is training?



SOLUTION

The architecture is predicting the 4th word given the 3 previous words. This is an architecture for the task of language modeling, predicting the probability of sequences of words.

Exercise 4

We have each used the Word2Vec algorithm to obtain word embeddings for the same vocabulary of words V.

In particular, developer A has got `context' vectors $u_w^A$ and `center' vectors v_w^A for every w in V, and developer B has got `context' vectors $u_w^B$ and `center' vectors v_w^B for every w in V .

For every pair of words w, w'in V, the inner product is the same in both models: $(u_w^A)^T v_{w'}^A = (u_w^B)^T v_{w'}^B$. Does it mean that, for every word w in V, $v_w^A = v_w^B$? Discuss your response.

SOLUTION

No. Word2Vec model only optimizes for the inner product between word vectors for words in the same context.
One can rotate all word vectors by the same amount and the inner product will still be the same. Alternatively one can scale the set of context vectors by a factor of k and the set of center vectors by a factor of 1=k. Such transformations preserves inner product, but the set of vectors could be di_erent.
Note that degenerate solutions (all zero vectors etc.) are discouraged.