Hacettepe University
Computer Engineering Faculty
Ayça KULA

Practical 1: Learning Distributed Word Representations

CMP 784, Fall 2021

Technical Points

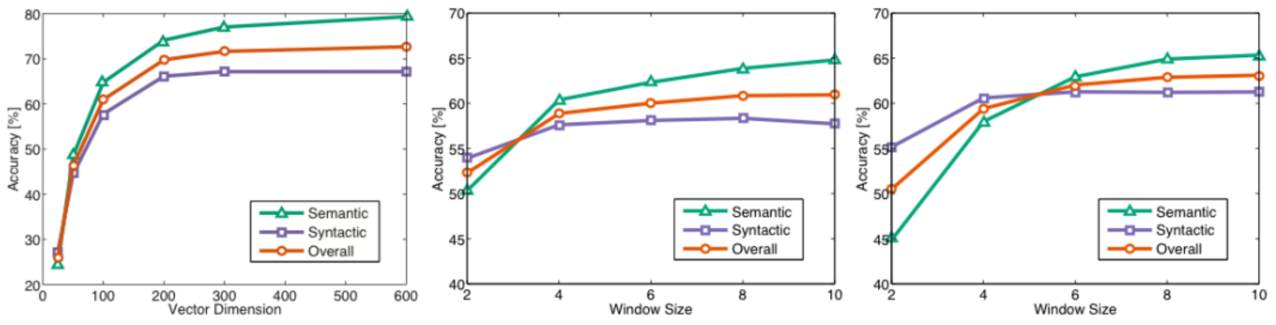
- We need to construct a co-occurrence matrix X, where a cell X_{ij} is a "strength" which represents how often the word i appears in the context of the word j.
- Word instances which appear next to each other see higher X_{ij} increments than word instances which appear with many words in between.
- We build this matrix X_{ij} symmetrically. This means that we treat word co-occurrences where the context word is to the left of the main word exactly the same as co-occurrences where the context word is to the right of the main word.
  How would you generalize the model if our target co-occurence isn't symmetric?

PART 1: Global Vectors for Word Representations (GLoVE)

1. Given the vocabulary size V and embedding dimensionality d, how many trainable parameters does the GLoVE model have?
   This method aims to find a d-dimensional embedding of the words that preserves properties of the co-occurrence matrix by representing the i-th word with a d-dimensional vector. Therefore the trainable parameters are found as:
   Trainable parameters = Vocabulary size * (number of dimensional embeddings)

   - **Corpus size:** The larger corpus sizes the better the performance. The larger the number of examples the bigger the experience is.
   - **Corpus source/type:** Normally the model effects the result. However, we have chosen GLoVE so that we would not change this hyperparameter.
   - **Dimension of word vectors:** Performance is lower for extremely low as well as for extremely high dimensional word vectors: Lower dimensional word vectors are not able to capture the different meanings of the different words in the corpus. This can be viewed as a high bias problem where our model complexity is too low. For instance, let us consider the words ”king”, ”queen”, ”man”, ”woman”. Intuitively, we would need to use two dimensions such as ”gender” and ”leadership” to encode these into 2-bit word vectors. Any lower would fail to capture semantic differences between the four words and any more may capture noise in the corpus that doesn't help in generalization – this is also known as the high variance problem.
   - **Context window size:** A window size of 8 around each center word typically works well for GloVe embeddings
   - **Context symmetry:** The difference may be seen from figure 1.

Figure 1: Accuracies with respect to vector dimension and context window size for GLoVe



(a) Symmetric context (b) Symmetric context (c) Asymmetric context

2. Write the gradient of the loss function with respect to one parameter vector.

$$L\left(\{\mathbf{w}_i, b_i\}_{i=1}^V\right) = \sum_{i,j=1}^V \left(\mathbf{w}_i^\top \mathbf{w}_j + b_i + b_j - \log X_{ij}\right)^2$$

The gradient with respect to the relevant parameters obtained from our cost function.

$$\mathcal{L}(\{w_i, b_i\}_{i=1}^V) = \sum_{i,j=1}^V \overset{\nearrow F}{(w_i^T w_j + b_i + b_j - \log X_{ij})}^2 \qquad x \in \{1, 2, \dots, V\}$$

$$\frac{\partial \mathcal{L}(\{w_i, b_i\}_{i=1}^V)}{\partial w_x} = \frac{\partial}{\partial w_i}\left[\sum_{i=j=x} F_{ij}^2 + \sum_{i=x}\sum_{j\neq x} F_{ij}^2 + \sum_{i\neq x}\sum_{j=x} F_{ij}^2 + \sum_{i\neq x}\sum_{j\neq x} F_{ij}^2\right]$$

$$= \frac{\partial}{\partial w_x}\left[F_{xx}^2 + \underbrace{\sum_{j\neq x} F_{xj}^2 + \sum_{i\neq x} F_{ix}^2}_{} + \cancel{F_{ij}^2} \rightarrow 0\right] = \frac{\partial}{\partial w_x}\left[F_{xx}^2 + 2\sum_{i\neq x} F_{ix}^2\right]$$

$F_{xj} = F_{ix}$ since they are symmetric

$$= \underbrace{\frac{\partial}{\partial w_x} F_{xx}^2}_{} + 2\frac{\partial}{\partial w_x}\left[\sum_{i\neq x} F_{ix}^2\right]$$

Inside F_{xx} there is $\rightarrow w_i w_j$

$$\frac{\partial L}{\partial w_x} = 2(F_{xx})\,(\cancel{2 w_x^T}) + (2)\sum_{i\neq x} 2 F_{ix} w_i^T$$

$$\boxed{\frac{\partial L}{\partial w_i} = 4\sum_{i=1}^V (w_i^T w_j + b_i + b_j - \log X_{ij})\, w_i^T}$$

$$\boxed{\frac{\partial L}{\partial b_i}} = \frac{\partial}{\partial b_x}\left[F_{xx}^2 + 2\sum_{i\neq x} F_{ix}^2\right] = \underbrace{\frac{\partial F_{xx}^2}{\partial b_x}} + 2\frac{\partial}{\partial b_x}\left[\sum_{i=x} F_{ix}^2\right]$$

$$= 2(F_{xx})\cancel{(2)} + 2\sum_{i=x}(2)F_{ix}$$

$$\boxed{\frac{\partial L}{\partial b_i} = 4\sum_{r=1}^{V}(w_i^T w_j + b_i + b_j - \log X_{ij})}$$

3. Implement the gradient update of GLoVE in language model.ipynb.

```
def grad_GLoVE(W,  b, log_co_occurence):
  "Return the gradient of GLoVE objective w.r.t W and b."
  "INPUT: W - Vxd; b - Vx1; log_co_occurence: VxV"
  "OUTPUT: grad_W - Vxd; grad_b - Vx1"
  n,_ = log_co_occurence.shape

  # loss = (W @ W.T + b @ np.ones([1,n]) + np.ones([n,1])@b.T - 0.5*(log_co_occurence +
     log_co_occurence.T))
  # grad_W = 4 *(W.T @ loss).T
  # grad_b = 4 * (np.ones([1,n]) @ loss).T
  ###########################   YOUR CODE HERE   ##############################
  # V, d = W.shape

  delta = (W @ W.T + b @ np.ones([1,n]) + np.ones([n,1])@b.T - log_co_occurence)
  grad_W = 2 * (delta @ W + delta.T @ W)
  grad_b = 4 * np.sum(delta, axis=1).reshape(W.shape[0], 1)
  ############################################################################
  return grad_W, grad_b
```

4. Train the model with varying dimensionality d. Which d leads to optimal validation performance? Why does/doesn't larger d always lead to better validation error?

Learning rate, epoch and initial variance has been changed to find the optimal embedding dimension value. The validation loss is small for learning rate of 0.01. However, as seen in Figure 2, difference between validation and train losses increase and this situation leads to underfit model.

Epoch	Learning Rate	Validation Loss	Embedding dimension
500	0.01	*1.035 x 10^5*	30
500	0.05	*1.06 x 10^5*	10
500	0.1	*1.06 x 10^5*	10
500	0.2	*1.06 x 10^5*	12
650	0.1	*1.06 x 10^5*	10
650	0.01	*1.045 x 10^5*	30

Table 1: Validation loss with respect to learning rate.

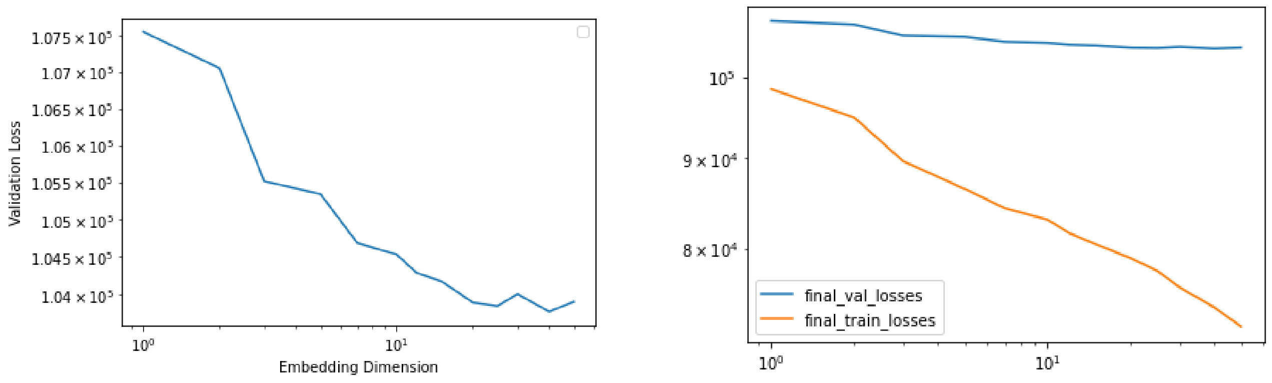


Figure 2: Validation and training loss plot for learning rate of 0.01

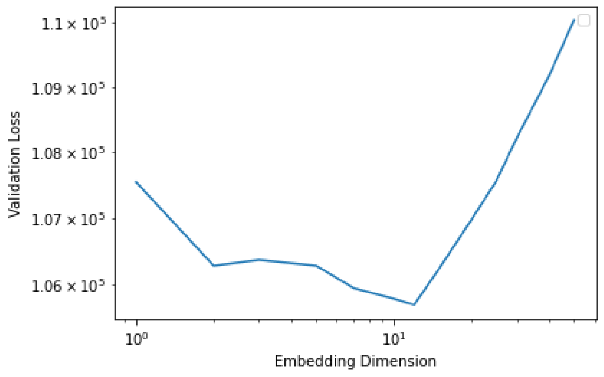


Figure 3: Validation loss for learning rate of 0.2.

In figure 3, it can be seen that the performance of model increases until d= 12, after that value its performance decreases. Therefore embedding dimension of 12 leads to optimal validation performance.In high embedding dimensions, the complexity is high and the neural network model cannot predict the new words.On the other hand, if embedding dimension is low, the model can extract more information for the concurrence matrix.

PART 2: Network architecture

As above, assume we have 250 words in the dictionary and use the previous 3 words as inputs. Suppose we use a 16-dimensional word embedding and a hidden layer with 128 units. The trainable parameters of the model consist of 3 weight matrices and 2 sets of biases. What is the total number of trainable parameters in the model? Which part of the model has the largest number of trainable parameters?

At third part of the code, there is a class called ”Params” which is representing the trainable parameters of the model.

- **word embedding weights**, a matrix of size $N_V \times D = 250 \times 16$, where N_V is the number of words in the vocabulary and D is the embedding dimension. 250×16 = trainable parameters.
- **embed to hid weights** a matrix of size $N_H \times contextLength \times D$, where N_H is the number of hidden units. The first D columns represent connections from the embedding of the first context word, the next D columns for the second context word, and so on.Thus, $128 \times 3 \times 16$ trainable parameters.
- **hid to output weights**, a matrix of size $N_V \times N_H$.Thus, 250×128 trainable parameters.

- **hid bias**, a vector of length N_H. Thus, 128 trainable parameters.
- **output bias**, a vector of length N_V.Thus, 250 trainable parameters.

Total number of trainable parameters: $250 \times 16 + 128 \times 3 \times 16 + 250 \times 128 + 128 + 250 = 42522$

”hid to output weights” part has the largest number of trainable parameters which is 250×128 and also we add the ”hid bias” which is 128 trainable parameters.

PART 3: Training the Neural Network

You will need to complete the implementation of two additional methods which are needed for training:

- **compute loss derivative**

```
def compute_loss_derivative(self, output_activations, expanded_target_batch):
        """Compute the derivative of the cross-entropy loss function with respect to the inputs
        to the output units. In particular, the output layer computes the softmax

            y_i = e^{z_i} / \sum_j e^{z_j}.

        This function should return a batch_size x vocab_size matrix, where the (i, j) entry
        is dC / dz_j computed for the ith training case, where C is the loss function

            C = -sum(t_i log y_i).

        The arguments are as follows:

            output_activations - the activations of the output layer, i.e. the y_i's.
           expanded_target_batch - each row is the indicator vector for a target word,
                i.e. the (i, j) entry is 1 if the i'th word is j, and 0 otherwise."""

    ###########################   YOUR CODE HERE   ##############################
        # Reference: Neural Networks and Deep Learning by Michael Nielsen
        # activation - y
        # --> y refers to desired output
        return output_activations - expanded_target_batch
    ############################################################################
```

- **backpropagate**

```
 def back_propagate(self, input_batch, activations, loss_derivative):
        """Compute the gradient of the loss function with respect to the trainable parameters
        of the model. The arguments are as follows:

            input_batch - the indices of the context words
            activations - an Activations class representing the output of Model.compute_activations
            loss_derivative - the matrix of derivatives computed by compute_loss_derivative

        Part of this function is already completed, but you need to fill in the derivative
        computations for hid_to_output_weights_grad, output_bias_grad, embed_to_hid_weights_grad,
        and hid_bias_grad. See the documentation for the Params class for a description of what
        these matrices represent."""

        # The matrix with values dC / dz_j, where dz_j is the input to the jth hidden unit,
        # i.e. y_j = 1 / (1 + e^{-z_j})
        hid_deriv = np.dot(loss_derivative, self.params.hid_to_output_weights) \
                    * activations.hidden_layer * (1. - activations.hidden_layer)

        ###########################   YOUR CODE HERE   ##############################
        # https://github.com/mkisantal/matlab-mnist/blob/master/nn_train.m
        #  % output layer
        #  hid_to_output_weights_grad =  hidden_layer_state * error_deriv'; --> hidden_layer_state is output of forward_prop
        #  output_bias_grad           = sum(error_deriv, 2);

        hid_to_output_weights_grad = loss_derivative.T @ activations.hidden_layer
        output_bias_grad = np.sum(loss_derivative, axis=0)
        embed_to_hid_weights_grad =  hid_deriv.T @ activations.embedding_layer
        hid_bias_grad = np.sum(hid_deriv, axis=0)  # axis = 1, keepdims = True
        ############################################################################

        # The matrix of derivatives for the embedding layer
        embed_deriv = np.dot(hid_deriv, self.params.embed_to_hid_weights)

        # Embedding layer
        word_embedding_weights_grad = np.zeros((self.vocab_size, self.embedding_dim))
        for w in range(self.context_len):
            word_embedding_weights_grad += np.dot(self.indicator_matrix(input_batch[:, w]).T,
                                                  embed_deriv[:, w * self.embedding_dim:(w + 1) * self.embedding_dim])

        return Params(word_embedding_weights_grad, embed_to_hid_weights_grad, hid_to_output_weights_grad,
                    hid_bias_grad, output_bias_grad)
```

The output of print gradient is:

```
loss_derivative[2, 5] 0.001112231773782498
loss_derivative[2, 121] -0.9991004720395987
loss_derivative[5, 33] 0.0001903237803173703
loss_derivative[5, 31] -0.7999757709589483

param_gradient.word_embedding_weights[27, 2] -0.27199539981936866
param_gradient.word_embedding_weights[43, 3] 0.8641722267354154
param_gradient.word_embedding_weights[22, 4] -0.2546730202374652
param_gradient.word_embedding_weights[2, 5] 0.0

param_gradient.embed_to_hid_weights[10, 2] -0.6526990313918256
param_gradient.embed_to_hid_weights[15, 3] -0.13106433000472612
param_gradient.embed_to_hid_weights[30, 9] 0.11846774618169402
param_gradient.embed_to_hid_weights[35, 21] -0.10004526104604389
param_gradient.hid_bias[10] 0.2537663873815642
param_gradient.hid_bias[20] -0.03326739163635368

param_gradient.output_bias[0] -2.0627596032173052
param_gradient.output_bias[1] 0.0390200857392169
param_gradient.output_bias[2] -0.7561537928318482
param_gradient.output_bias[3] 0.21235172051123635
```

PART 4: Analysis

1. Pick three words from the vocabulary that go well together (for example, 'government of united', 'city of new', 'life in the', 'he is the' etc.). Use the model to predict the next word. Does the model give sensible predictions? Try to find an example where it makes a plausible prediction even though the 4-gram wasn't present in the dataset (raw sentences.txt). To help you out, the function find occurrences lists the words that appear after a given 3-gram in the training set.

```
trained_model.predict_next_word("government", "of", "united")
government of united . Prob: 0.14276
government of united life Prob: 0.10426
government of united own Prob: 0.09081
government of united states Prob: 0.05131
government of united york Prob: 0.04767
government of united ? Prob: 0.03922
government of united home Prob: 0.02585
government of united family Prob: 0.02461
government of united all Prob: 0.02378
government of united children Prob: 0.02108

trained_model.predict_next_word("city", "of", "new")
city of new york Prob: 0.99444
city of new . Prob: 0.00150
city of new home Prob: 0.00031
city of new world Prob: 0.00026
city of new , Prob: 0.00025
city of new life Prob: 0.00025
city of new place Prob: 0.00024
city of new to Prob: 0.00017
city of new year Prob: 0.00015
city of new ? Prob: 0.00013

trained_model.predict_next_word("life", "in", "the")
life in the world Prob: 0.17904
life in the united Prob: 0.11468
life in the game Prob: 0.05330
life in the country Prob: 0.04961
life in the house Prob: 0.04396
life in the city Prob: 0.03701
life in the next Prob: 0.03046
life in the business Prob: 0.02983
life in the street Prob: 0.02856
life in the first Prob: 0.02802

trained_model.predict_next_word("he", "is", "the")
he is the best Prob: 0.30298
he is the same Prob: 0.09604
he is the way Prob: 0.06930
he is the only Prob: 0.04856
he is the man Prob: 0.04572
he is the first Prob: 0.04371
he is the right Prob: 0.02787
he is the president Prob: 0.02374
he is the law Prob: 0.01761
he is the other Prob: 0.01701

find_occurrences("he","is","the")
The tri-gram "he is the" was followed by the following words in the training set:
    president (4 times)
    man (4 times)
    only (4 times)
    one (4 times)
    best (2 times)
    next (1 time)
    group (1 time)
    same (1 time)
    city (1 time)
    government (1 time)
    first (1 time)
    public (1 time)
    director (1 time)
    show (1 time)
    second (1 time)

find_occurrences("life", "in", "the")
The tri-gram "life in the" was followed by the following words in the training set:
    big (7 times)
    united (2 times)
    world (1 time)
    department (1 time)
```

2. Plot the 2-dimensional visualization using the method tsne plot representation. Look at the plot and find a few clusters of related words. What do the words in each cluster have in common? Plot the 2-dimensional visualization using the method tsne plot GLoVE representation for a 256 dimensional embedding. How do the t-SNE embeddings for both models compare? Plot the 2-dimensional visualization using the method plot 2d GLoVE representation. How does this compare to the t-SNE embeddings? (You don't need to include the plots with your submission.)

   One of the cluster for the ”tsne plot representation(trained model)” function is ”you, we , they” words centered around (-5,20). Another cluster that can be seen from figure 4 is ”will, should, would, might” centered around (20,10).

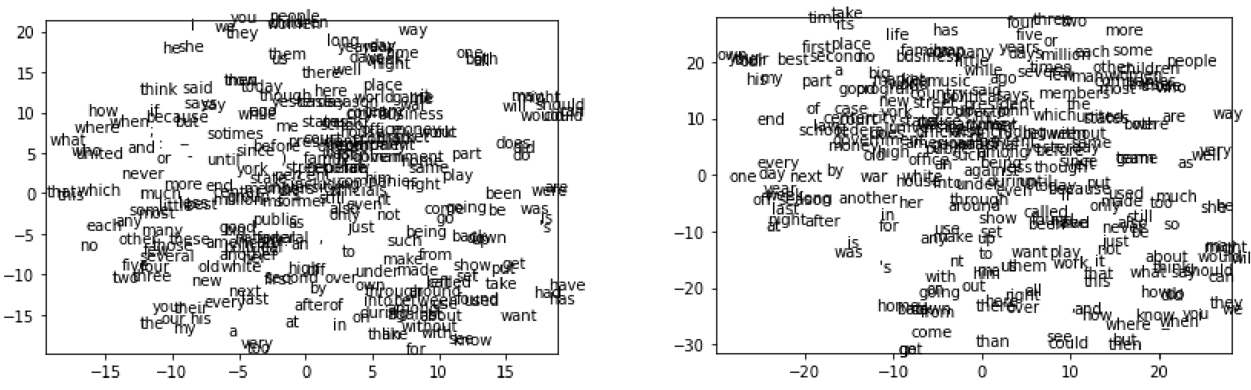


Figure 4: Left figure corresponds to ”tsne plot representation(trained model)” result and right figure ”tsne plot GLoVE representation(W final, b final)”

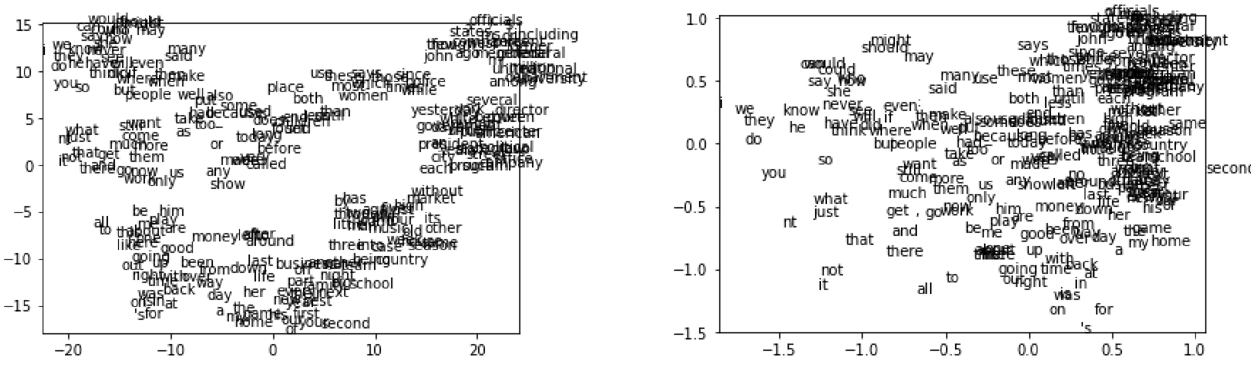


Figure 5: Left figure corresponds to ”tsne plot GLoVE representation(W final 2d, b final 2d)” result and right figure ”plot 2d GLoVE representation(W final 2d, b final 2d)”

For ”tsne plot GLoVE representation(W final, b final)” function the ”you, we , they” cluster has been shifted to (30,-20).
For the function ”tsne plot GLoVE representation(W final 2d, b final 2d)”, there is a significant difference for clusters because they are prominently/clearly separated to clusters.
However, for ”plot 2d GLoVE representation(W final 2d, b final 2d)” has more clusters compared to other ones.”you, we , they” cluster is at (-1,0)

3. Are the words ’new’ and ’york’ close together in the learned representation? Why or why not?
As you can see from the results below the words ”new” and ”york” are not so close.They have a distance of 3.513. When we displayed the nearest words for both ”new” and ”york”, they are not in each others list. This means that, they are not close to each other.

```
trained_model.word_distance("new","york")
3.5128424215354244

trained_model.display_nearest_words("york")
state: 0.8639999532807732
school: 0.9541780760985669
ms.: 1.0210745454591996
university: 1.0259370928182088
general: 1.0352507065577967
```

```
national: 1.0471328107313858
city: 1.0525436094968286
family: 1.0567788510562528
public: 1.060115142428818
music: 1.061184555927523

trained_model.display_nearest_words("new")
old: 2.503635370671453
white: 2.605746058498431
second: 2.746535133660538
high: 2.9552780663748788
several: 2.97239933950015
first: 2.9906972068496067
four: 2.996789284777312
political: 3.0079243415044 67
five: 3.025665704793815
next: 3.068724106184249
```

4. Which pair of words is closer together in the learned representation: ('government', 'political'), or ('government', 'university')? Why do you think this is?

   As seen below, "government" and "university" is closer to each other. Government refers to an administrative unit and it is related with the words management/administration. Therefore, the model could find these words more similar.

```
trained_model.word_distance("government","political")
1.2192079764254984

trained_model.word_distance("government","university")
1.0626503444832562
```

References

https://github.com/mkisantal/matlab-mnist/blob/master/nn$_t$*rain.m*

Neural Networks and Deep Learning by Michael Nielsen

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representatio. Empirical Methods in Natural Language Processing (EMNLP), 2014.

https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/