

CSC413: Programming Assignment 1

Tianyu Du (1003801647)

2020/02/03 at 16:59:22

1 Linear Embedding GLoVE

Question 1 For each $i \in \{1, 2, \dots, V\}$, $\mathbf{w}_i \in \mathbb{R}^d$ and $b_i \in \mathbb{R}$. For each vocabulary, there are $d + 1$ corresponding trainable parameters. The total number of trainable parameters is therefore

$$V(d + 1) \quad (1.1)$$

Question 2 Define

$$\delta_{ij} := \mathbf{w}_i^T \mathbf{w}_j + b_i + b_j - \log X_{ij} \quad (1.2)$$

By construction, X is symmetric, hence,

$$\delta_{ij} = \delta_{ji} \quad \forall i, j \quad (1.3)$$

Let $\alpha \in \{1, 2, \dots, V\}$,

$$\frac{\partial}{\partial \mathbf{w}_\alpha} L(\{\mathbf{w}_i, b_i\}_{i=1}^V) = \frac{\partial}{\partial \mathbf{w}_\alpha} \sum_{i,j=1}^V (\mathbf{w}_i^T \mathbf{w}_j + b_i + b_j - \log X_{ij})^2 \quad (1.4)$$

$$= \frac{\partial}{\partial \mathbf{w}_\alpha} \left[\sum_{i=j=\alpha} \delta_{ij}^2 + \sum_{i=\alpha} \sum_{j \neq \alpha} \delta_{ij}^2 + \sum_{i \neq \alpha} \sum_{j=\alpha} \delta_{ij}^2 + \sum_{i,j \neq \alpha} \delta_{ij}^2 \right] \quad (1.5)$$

$$= \frac{\partial}{\partial \mathbf{w}_\alpha} \left[\sum_{i=j=\alpha} \delta_{ij}^2 + \sum_{i=\alpha} \sum_{j \neq \alpha} \delta_{ij}^2 + \sum_{i \neq \alpha} \sum_{j=\alpha} \delta_{ij}^2 \right] \quad (1.6)$$

$$= \frac{\partial}{\partial \mathbf{w}_\alpha} \left[\delta_{\alpha\alpha}^2 + \sum_{j \neq \alpha} \delta_{\alpha j}^2 + \sum_{i \neq \alpha} \delta_{i\alpha}^2 \right] \quad (1.7)$$

$$= \frac{\partial}{\partial \mathbf{w}_\alpha} \left[\delta_{\alpha\alpha}^2 + \sum_{j \neq \alpha} \delta_{j\alpha}^2 + \sum_{i \neq \alpha} \delta_{i\alpha}^2 \right] \quad (1.8)$$

$$= \frac{\partial}{\partial \mathbf{w}_\alpha} \left[\delta_{\alpha\alpha}^2 + 2 \sum_{i \neq \alpha} \delta_{i\alpha}^2 \right] \quad (\dagger) \quad (1.9)$$

Because $\frac{\partial \|\mathbf{w}\|_2^2}{\partial \mathbf{w}} = 2\mathbf{w}^T$,

$$\frac{\partial}{\partial \mathbf{w}_\alpha} \delta_{\alpha\alpha}^2 = 2\delta_{\alpha\alpha} 2\mathbf{w}_\alpha^T \quad (1.10)$$

$$= 4\delta_{\alpha\alpha} \mathbf{w}_\alpha^T \quad (1.11)$$

For other $i \neq \alpha$

$$\frac{\partial}{\partial \mathbf{w}_\alpha} \delta_{i\alpha}^2 = 2\delta_{i\alpha} \frac{\partial}{\partial \mathbf{w}_\alpha} \delta_{i\alpha} \quad (1.12)$$

$$= 2\delta_{i\alpha} \mathbf{w}_i^T \quad (1.13)$$

Altogether with (†),

$$\frac{\partial}{\partial \mathbf{w}_\alpha} L(\{\mathbf{w}_i, b_i\}_{i=1}^V) = 4\delta_{\alpha\alpha} \mathbf{w}_\alpha^T + 2 \sum_{i \neq \alpha} 2\delta_{i\alpha} \mathbf{w}_i^T \quad (1.14)$$

$$= 4 \sum_{i=1}^V \delta_{i\alpha} \mathbf{w}_i^T \quad (\dagger\dagger) \quad (1.15)$$

Taking the transpose of derivative (††) gives the gradient

$$\nabla_{\mathbf{w}_\alpha} L(\{\mathbf{w}_i, b_i\}_{i=1}^V) = 4 \sum_{i=1}^V \delta_{i\alpha} \mathbf{w}_i \quad (1.16)$$

$$= 4 \sum_{i=1}^V (\mathbf{w}_\alpha^T \mathbf{w}_j + b_\alpha + b_j - \log X_{\alpha j}) \mathbf{w}_i \quad (1.17)$$

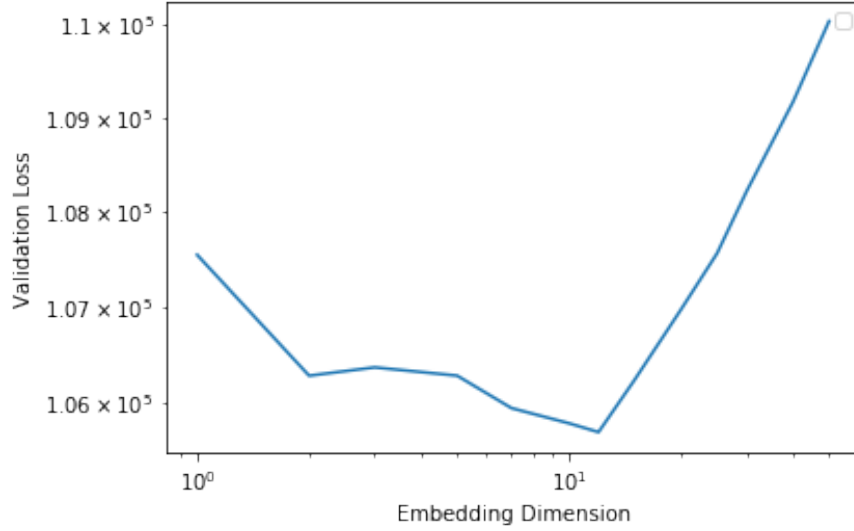
Question 3 Implementation:

```

1 def grad_GLoVe(W, b, log_co_occurrence):
2     "Return the gradient of GloVe objective w.r.t W and b."
3     "INPUT: W - Vxd; b - Vx1; log_co_occurrence: VxV"
4     "OUTPUT: grad_W - Vxd; grad_b - Vx1"
5     V, d = W.shape
6     n, _ = log_co_occurrence.shape
7     ##### YOUR CODE HERE #####
8     delta = (W @ W.T + b @ np.ones([1,n]) + np.ones([n,1]) @ b.T - log_co_occurrence)
9     grad_W = 2 * (delta @ W + delta.T @ W)
10    grad_b = 4 * np.sum(delta, axis=1).reshape(V, 1)
11    #####
12    return grad_W, grad_b
13

```

Question 4 As embedding dimension increases, the performance of model firstly goes up and then down. An embedding dimension of $d = 12$ leads to the optimal performance according to the validation loss. When the embedding dimension is low, increasing the dimension allows the embedding procedure to extract more meaningful information on the concurrence between words. However, when the embedding dimension is too large, the afterward neural network (the one takes embedded results and outputs the prediction) do not have enough complexity (in terms of the number of neurones) to predict next word based on high dimensional features (e.g., the dimension of features would be 150 if we are using 50d embedding).



2 Network architecture

Question 1 Suppose the embedding layers are already trained, and they are not trainable in this section. The `word_embedding_weights` has shape 16×250 , which maps one-hot-vector representing identities of words to their corresponding embeddings. This weight has been trained in the previous section, therefore, possesses zero trainable parameters. Note that this layer has no bias term. The `embed_to_hid_weights` maps the embedding results of three context words ($3 \times 16 = 48$ d) to the 128d hidden layer. Hence,

$$\mathbf{W}_{\text{embed_to_hid_weights}} \in \mathbb{R}^{128 \times 48} \quad (2.1)$$

$$\mathbf{b}_{\text{embed_to_hid_bias}} \in \mathbb{R}^{128} \quad (2.2)$$

All entries in the weight and bias are trainable.

The layer with `hid_to_output_weights` maps 128d hidden neurones to 250d outputs. Therefore,

$$\mathbf{W}_{\text{hid_to_output_weights}} \in \mathbb{R}^{250 \times 128} \quad (2.3)$$

$$\mathbf{b}_{\text{hid_to_output_bias}} \in \mathbb{R}^{250} \quad (2.4)$$

All entries in the weight and bias are trainable.

Total number of trainable parameters is

$$128 \times 48 + 128 + 250 \times 128 + 250 = 38522 \quad (2.5)$$

The `hid_to_output_weights` part has the largest number of trainable parameters ($128 \times 250 = 32000$).

Question 2 A 4-grams requires 250^3 possible set of context words $\mathbf{w} = (w_1, w_2, w_3)$, and for each \mathbf{w} , there are 250 possible following words. The total number of combinations is

$$250^4 = 3906250000 \quad (2.6)$$

3 Training the Neural Network

Outputs outputs from `print_gradients` methods.

```
1 loss_derivative[2, 5] 0.001112231773782498
2 loss_derivative[2, 121] -0.9991004720395987
3 loss_derivative[5, 33] 0.0001903237803173703
4 loss_derivative[5, 31] -0.7999757709589483
5
6 param_gradient.word_embedding_weights[27, 2] -0.27199539981936866
7 param_gradient.word_embedding_weights[43, 3] 0.8641722267354154
8 param_gradient.word_embedding_weights[22, 4] -0.2546730202374648
9 param_gradient.word_embedding_weights[2, 5] 0.0
10
11 param_gradient.embed_to_hid_weights[10, 2] -0.6526990313918256
12 param_gradient.embed_to_hid_weights[15, 3] -0.13106433000472612
13 param_gradient.embed_to_hid_weights[30, 9] 0.118467746181694
14 param_gradient.embed_to_hid_weights[35, 21] -0.10004526104604386
15
16 param_gradient.hid_bias[10] 0.25376638738156415
17 param_gradient.hid_bias[20] -0.03326739163635379
18
19 param_gradient.output_bias[0] -2.0627596032173052
20 param_gradient.output_bias[1] 0.0390200857392169
21 param_gradient.output_bias[2] -0.7561537928318482
22 param_gradient.output_bias[3] 0.21235172051123635
23
```

Implementations See the submitted Jupyter notebook file for detailed implementations.

```
1 def compute_loss_derivative(self, output_activations, expanded_target_batch):
2     ...
3     ##### YOUR CODE HERE #####
4     return output_activations - expanded_target_batch # batch_size x vocab_size
5     #####
6
7 def back_propagate(self, input_batch, activations, loss_derivative):
8     ...
9     ##### YOUR CODE HERE #####
10    hid_to_output_weights_grad = loss_derivative.T @ activations.hidden_layer
11    output_bias_grad = np.sum(loss_derivative, axis=0)
12    embed_to_hid_weights_grad = hid_deriv.T @ activations.embedding_layer
13    hid_bias_grad = np.sum(hid_deriv, axis=0)
14    #####
15    ...
```

4 Analysis

Question 1 The predicted probabilities can be found below. For the first example, the model predicts 'government of united own', which does not really make sense. But for the second and third trails, the model produces sensible outputs.

```
1 >>> trained_model.predict_next_word("government", "of", "united")
2 government of united own Prob: 0.06786
3 government of united states Prob: 0.06067
```

```

4 government of united life Prob: 0.05791
5 government of united money Prob: 0.05206
6 government of united . Prob: 0.05050
7 government of united end Prob: 0.04033
8 government of united time Prob: 0.03520
9 government of united house Prob: 0.03320
10 government of united say Prob: 0.02317
11 government of united team Prob: 0.02231
12
13 >>> trained_model.predict_next_word("city", "of", "new")
14 city of new york Prob: 0.98987
15 city of new . Prob: 0.00139
16 city of new ? Prob: 0.00080
17 city of new , Prob: 0.00078
18 city of new days Prob: 0.00059
19 city of new children Prob: 0.00058
20 city of new times Prob: 0.00055
21 city of new years Prob: 0.00033
22 city of new music Prob: 0.00033
23 city of new people Prob: 0.00030
24
25 >>> trained_model.predict_next_word("life", "in", "the")
26 life in the world Prob: 0.15756
27 life in the first Prob: 0.13498
28 life in the end Prob: 0.05317
29 life in the united Prob: 0.04379
30 life in the street Prob: 0.04040
31 life in the game Prob: 0.03705
32 life in the country Prob: 0.03588
33 life in the school Prob: 0.02934
34 life in the place Prob: 0.02903
35 life in the city Prob: 0.02711
36

```

`find_occurrences("life", "in", "the")` returns

```

1 The tri-gram "life in the" was followed by the following words in the training set:
2     big (7 times)
3     united (2 times)
4     world (1 time)
5     department (1 time)
6

```

the prediction `life in the country` was not in the dataset, but the model still assigns it with a positive possibility.

Question 2 On the output from `tsne_plot_representation(trained_model)` we can see there is a cluster of *prepositions* on the top-right area (around (10, 13)), which consists of words like 'against', 'thought', and 'of', etc. Another cluster appears near (-18, 0), which includes *modal verbs* such as 'should', 'would', and 'might'.

Comparing the graph from `tsne_plot_GLoVe_representation(W_final, b_final)` and the previous graph, we can observe a shift of the preposition cluster, which shifted from (10, 13) to (3, -3). The presented distribution from `plot_2d_GLoVe_representation(W_final_2d, b_final_2d)` (the third graph) is more clustered compared with the previous two graphs. In previous graphs, vocabularies are distributed more or less evenly, but in the third graph, most vocabularies are clustered around the top right corner.

For the graph from `tsne_plot_GLoVe_representation(W_final_2d, b_final_2d)` forms a circular and radial distribution. Vocabularies are distributed around (5, 0), but no words can be found near the centroid

(5, 0).

Question 3 The distance between **new** and **york** is 3.90. The following method computes distances between all pairs of words:

```
1 dist = np.zeros([vocab_size, vocab_size])
2 for i, word_i in enumerate(data["vocab"]):
3     for j, word_j in enumerate(data["vocab"]):
4         dist[i, j] = trained_model.word_distance(word_i, word_j)
5
```

It turns out that the distance between **new** and **york** was on the 56 percentile among all pairs of words.

Further, **new** is not in the top ten nearest words of **york**. And, **york** is not in the top ten nearest words of **new** neither. Therefore, **new** and **york** are not closed.

Even though the pair **new york** appears frequently (in everyday conversation), but it is possible that the term **new** is used more often as an adjective than a city name.

Question 4 **government** is closer to **university** according to the trained model. It could be that **government** is more frequently used as a social institution than a component in the political system. Therefore, the model believes the similarity between **government** and **university** to be higher according to the provided dataset.

```
1 ('government', 'political') 1.2808505981043723
2 ('government', 'university') 1.1354211512227212
3
```