Xiang Chen

February 4, 2021

CSC413 P1 Write-up

1.1 GLoVE Parameter Count [0pt]

Pass.

1.2 Expression for gradient $\frac{\partial L}{\partial w_i}$ [1pt]

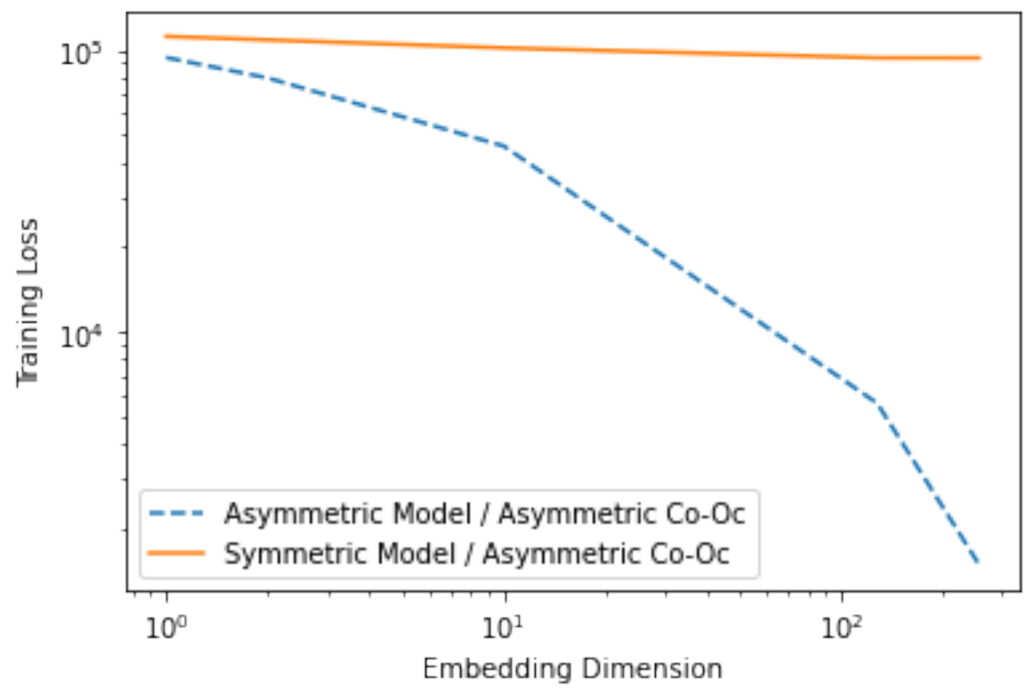
$$\frac{\partial L}{\partial w_i} = \frac{\partial(\sum_{i,j=1}^{V}(\mathbf{w}_i^\top \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij})^2)}{\partial w_i} = 2\sum_{j=1}^{V}(\mathbf{w}_i^\top \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij})\tilde{\mathbf{w}}_j$$

1.3 Implement the gradient update of GLoVE [1pt]

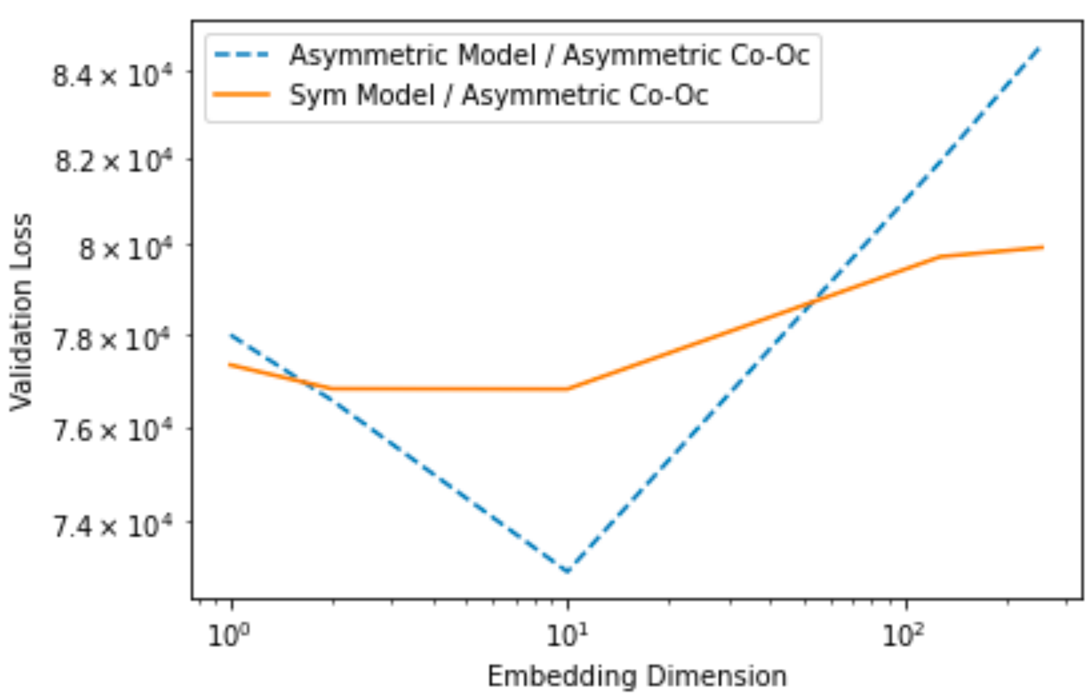
```
def grad_GLoVE(W, W_tilde, b, b_tilde, log_co_occurence):
  "Return the gradient of GLoVE objective w.r.t W and b."
  "INPUT: W - Vxd; W_tilde - Vxd; b - Vx1; b_tilde - Vx1; log_co_occurence: VxV"
  "OUTPUT: grad_W - Vxd; grad_W_tilde - Vxd, grad_b - Vx1, grad_b_tilde - Vx1"
  n,_ = log_co_occurence.shape

  if not W_tilde is None and not b_tilde is None:
  ###########################   YOUR CODE HERE  ##############################
    loss = W @ W_tilde.T + b @ np.ones([1,n]) + np.ones([n,1])@b_tilde.T - log_co_occurence
    grad_W = 2 * np.dot(loss, W_tilde)
    grad_W_tilde = 2 * np.dot(loss.T, W)
    grad_b = 2 * np.dot(loss, np.ones([n,1]))
    grad_b_tilde = 2 * np.dot(loss.T, np.ones([n,1]))
  ############################################################################
  else:
    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_W_tilde = None
    grad_b = 4 * (np.ones([1,n]) @ loss).T
    grad_b_tilde = None

  return grad_W, grad_W_tilde, grad_b, grad_b_tilde
```

1.4 Effects of embedding dimension [0pt]

Pass.



Training Loss
Embedding Dimension
Asymmetric Model / Asymmetric Co-Oc
Symmetric Model / Asymmetric Co-Oc



Validation Loss
Embedding Dimension
Asymmetric Model / Asymmetric Co-Oc
Sym Model / Asymmetric Co-Oc

2.1 Number of parameters in neural network model [1pt]

The total number of trainable parameters in the model is $V \cdot D + N \cdot D \cdot H + H + V \cdot H + V$.

hid_to_output_weights has the largest number of trainable parameters. From the question description, we can know $V \gg H > D > N$, and we only need to compare $N \cdot D \cdot H$ and $V \cdot H$, we know $V \cdot H$ is larger.

2.2 Number of parameters in n-gram mode [1pt]

V^{N+1} parameters in n-gram mode.

2.3 Comparing neural network and n-gram model scaling [0pt]

Pass.

3.1 Implement gradient with respect to output layer inputs [1pt]

```
###########################   YOUR CODE HERE   ##############################
M_repeat = np.repeat(target_mask, output_activations.shape[1] / target_mask.shape[1], axis=1)
loss_derivative = np.multiply(M_repeat.reshape(M_repeat.shape[0], M_repeat.shape[1]), (output_activations - expanded_target_batch))
return loss_derivative
############################################################################
```

3.2 Implement gradient with respect to parameters [1pt]

```
###########################   YOUR CODE HERE   ##############################
hid_to_output_weights_grad = np.dot(loss_derivative.T, activations.hidden_layer)
output_bias_grad = np.dot(loss_derivative.T, np.ones([loss_derivative.shape[0],]))
embed_to_hid_weights_grad = np.dot(hid_deriv.T, activations.embedding_layer)
hid_bias_grad = np.dot(hid_deriv.T, np.ones([hid_deriv.shape[0],]))
############################################################################
```

3.3 Print the gradients [1pt]

```
loss_derivative[2, 5] 0.0
loss_derivative[2, 121] 0.0
loss_derivative[5, 33] 0.0
loss_derivative[5, 31] 0.0

param_gradient.word_embedding_weights[27, 2] 0.0
param_gradient.word_embedding_weights[43, 3] 0.011596892511489458
param_gradient.word_embedding_weights[22, 4] -0.0222670623817297
param_gradient.word_embedding_weights[2, 5] 0.0

param_gradient.embed_to_hid_weights[10, 2] 0.3793257091930164
param_gradient.embed_to_hid_weights[15, 3] 0.01604516132110917
param_gradient.embed_to_hid_weights[30, 9] -0.4312854367997419
param_gradient.embed_to_hid_weights[35, 21] 0.06679896665436337

param_gradient.hid_bias[10] 0.023428803123345134
param_gradient.hid_bias[20] -0.02437045237887416

param_gradient.output_bias[0] 0.0009701061469027941
param_gradient.output_bias[1] 0.1686894627476322
param_gradient.output_bias[2] 0.0051664774143909235
param_gradient.output_bias[3] 0.15096226471814364
```

3.4 Run model training [0 pt]

Pass.

4.1 t-SNE [1pt]

The output for tsne_plot_representation actually has much noise and we can roughly find some regular patterns behind it. We can see words cluster contains 'should', 'would' , 'could' on the top right corner and I guess they have the same semantic property. In the mid part, the word cluster contains 'department', 'university', 'national' and I guess they share the same semantic attribution.

When we compare the output of tsne_plot_representation and the output of tsne_plot_GLoVe_representation, we can see actually the ouput of tsne_plot_GLoVe_representation cares more about the semantic meaning of each word. In the output of tsne_plot_GLoVe_representation, world cluster contains 'fedural', 'government', 'political', 'nation' in the middle of the graph and it also contains a lot of words with the same semantic meaning. I also find tsne_plot_GLoVe_representation has loose word distribution and tsne_plot_representation has more concentred word distribution.

When we compare the output for plot_2d_GLoVe_representation and tsne_plot_GLoVE_representation, we can find tsne_plot_GLoVE_representation has more concentrarion on semantic meaning (features) and we can see word clusters contain more with sementic meaning (features) and also tsne_plot_GLoVE_representation has a high focus on words with relevance. I also find tsne_plot_GLoVe_representation has loose word distribution and plot_2d_GLoVE_representation has more concentred word distribution.

4.2.1 Specific example [1pt]

GloVe embeddings: she - 1.48167433432594

Concatenation of W_final_asym, W_tilde_final_asym: she - 2.2666140980184095

Averaging asymmetric GLoVE vectors: she - 1.0321156534651934

Neural Netework Word Embeddings: she - 18.303041008736102

Yes, it is 'she'.

After I have run multi-times of program, I find for each time it will generate differnet graphs. I have compared with several differnet graphs and find the regular pattern. I have try many times and finally I can find some graphs which contains all four words. I find the bound shape of the four words actually is not a perfect parallelogram, but they share the same direction and the distance between he and him is roughly the same with she and her. The closest

4.2.2 Finding another Quadruplet [0pt]

Pass.