1. Explanation of your code implementations for each required task.

a. data.py

i. Use **self.data_index** as the current center word index. When it is less than skip_window or larger than **len(data)-skip_window**, move it to the next executable valid position.

ii. Outer loop: keep track of current batch size, continue until it reaches **batch_size**.

iii. Inner loop: loop within **±skip_window** until it reaches **num_skips**.

iv. At the end of the outer loop, add the samples drawn from the inner loop to the final samples only if it does not exceed the **batch_size** limit.

b. model.py: negative likelihood

i. The calculation is based on the formula: $$-\log \prod_{j=0, j\neq m}^{2m} \frac{\exp(u_{c-m+j}^T v_c)}{\sum_{k=1}^{|V|} \exp(u_k^T v_c)}$$

ii. **mul_1**: multiply the two input embeddings and sum over its second dimension, which gives their **dot product** result $u_{c-m+j}^T v_c$.

iii. **mul_2**: calculate the **matrix multiplication** of center embedding(batch_size * embed_size) with the **transpose** of context embedding(embed_size * batch_size), which is the **dot product** of center word to **every** context word: $u_k^T v_c$

iv. Thus the final loss is given by: **-∑log [exp(mul_1) / sum(exp(mul_2))]**

c. model.py: negative sampling

i. The calculation is based on the formula:

$$J = -\sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} - \sum_{(w,c)\in \tilde{D}} \log\left(\frac{1}{1+\exp(u_w^T v_c)}\right)$$

ii. **weights**: adjust **count** by the power of ¾ according to the paper.

iii. Sample negative samples using **torch.multinomial**, which returns a tensor where each row contains num_samples **indices** sampled from the multinomial probability distribution located in the corresponding row of tensor input, i.e. the **weights**.

iv. **neg_mul**: dot product of negative embeddings

v. **mul**: dot product of positive embeddings

vi. Thus the final loss is given by:

-∑log sigmoid(mul) - ∑log sigmoid(-neg_mul)

d. word_analogy.py

i. **test_diffs**, **candidate_diffs**: **normalized diff** vector of test_embs and candidate_embs.
ii. Calculate the similarity of **each candidate** by **summing** over its dot product to **every test word**, then the **most similar** word is given by the **largest cosine value**.

2. Hyper-parameters explored w/ a description of what each parameter controls and how varying each of them is likely to affect the training. You need to experiment with at least three hyperparameters. We suggest: skip_window, batch_size, embedding_size and max_num_steps.

a. skip_window=1, batch_size=64, embedding_size=128, max_num_steps = 200001

nll	Avg loss from 8.290 to **7.428**
neg	Avg loss from 1.389 to **0.944**

b. skip_window=2, batch_size=64, embedding_size=128, max_num_steps = 200001

nll	Avg loss from 8.303 to **8.024**
neg	Avg loss from 1.387 to **1.124**

c. (**best**) skip_window=1, batch_size=64, embedding_size=256, max_num_steps = 200001

nll	Avg loss from 8.297 to **7.417**
neg	Avg loss from 1.385 to **0.939**

d. skip_window=1, batch_size=64, embedding_size=128, max_num_steps = 100001

nll	Avg loss from 8.290 to **7.61**
neg	Avg loss from 1.389 to **1.073**

e. skip_window=1, batch_size=128, embedding_size=128, max_num_steps = 100001

nll	final Avg loss **8.854**
neg	Avg loss from **1.30**

From the results above, it can be briefly concluded:

1. Compare with **a** and **b**, larger **skip_window** leads to higher loss.
2. Compare with **a** and **c**, larger **embedding** results in a slightly lower loss.
3. Compare with **a** and **d**, with smaller **max_num_steps**, program runtime can be cut down by half, but the loss will be a little bit larger in the end.
4. Compare with d and e, with larger **batch_size**, the loss is higher.

3. Results (the ones you get by running the evaluation perl script - no need to submit the predictions for the DEV data) on the analogy task for five different configurations of hyperparameters, along with the best configuration (on the word pairs dev.txt). Explain these results. Are there any trends you observe? For example, what does increasing dimensions of vectors do? You should say what you learned from exploring these configurations. You are free to consult any resources to support your arguments.

a. skip_window=1, batch_size=64, embedding_size=128, max_num_steps = 200001

nll	Overall Accuracy 32.8%
neg	Overall Accuracy 31.8%

b. skip_window=2, batch_size=64, embedding_size=128, max_num_steps = 200001

nll	Overall Accuracy 34.0%
neg	Overall Accuracy 32.1%

c. skip_window=1, batch_size=64, embedding_size=256, max_num_steps = 200001

nll	Overall Accuracy 32.8%
neg	Overall Accuracy 33.1%

d. skip_window=1, batch_size=64, embedding_size=128, max_num_steps = 100001

nll	Overall Accuracy 33.2%
neg	Overall Accuracy 32.3%

e. skip_window=1, batch_size=128, embedding_size=128, max_num_steps = 100001

nll	Overall Accuracy 34.4%
neg	Overall Accuracy 33.5%

From the results above, it can be briefly concluded:

1. Compare with **a** and **b**, larger **skip_window** leads to slightly higher accuracy.
2. Compare with **a** and **c**, with larger **embedding**, accuracy is higher on negative sampling.
3. Compare with **a** and **d**, with different **max_num_steps**, the results are not making much difference.
4. Compare with **d** and **e**, with larger **batch_size**, the accuracy is much higher.

My speculation is, with larger **batch_size**, according to the objective function, the probability of many other unrelated words given a center word can be minimized, thus the accuracy will be higher.

4. Summary of the comparison in bias scores (for the already given data, not your custom task) obtained for the best model as obtained with loss function of NLL and negative sampling. Justification for the bias score along with the attributes and targets used must be mentioned.

a. **best nll**: skip_window=1, batch_size=64, embedding_size=256, max_num_steps = 200001

   Final Bias Scores
   {
   "EuropeanAmerican_AfricanAmerican_Pleasant_Unpleasant": "-0.63753045",
   "Flowers_Insects_Pleasant_Unpleasant": "-0.2391101",
   "Male_Female_Career_Family": "-0.2912987",
   "Math_Arts_Male_Female": "1.4251659",
   "MusicalInstruments_Weapons_Pleasant_Unpleasant": "-0.06084507",
   "Science_Arts_Male_Female": "0.5422834"
   }

b. **best neg**: skip_window=1, batch_size=64, embedding_size=256, max_num_steps = 200001

   Final Bias Scores
   {
   "EuropeanAmerican_AfricanAmerican_Pleasant_Unpleasant": "-0.014409131",
   "Flowers_Insects_Pleasant_Unpleasant": "0.15473033",
   "Male_Female_Career_Family": "-0.6455905",
   "Math_Arts_Male_Female": "0.95586777",
   "MusicalInstruments_Weapons_Pleasant_Unpleasant": "-0.3347172",
   "Science_Arts_Male_Female": "0.5234563"
   }