ECE 1513 Introduction to Machine Learning

Assignment 3 Part II

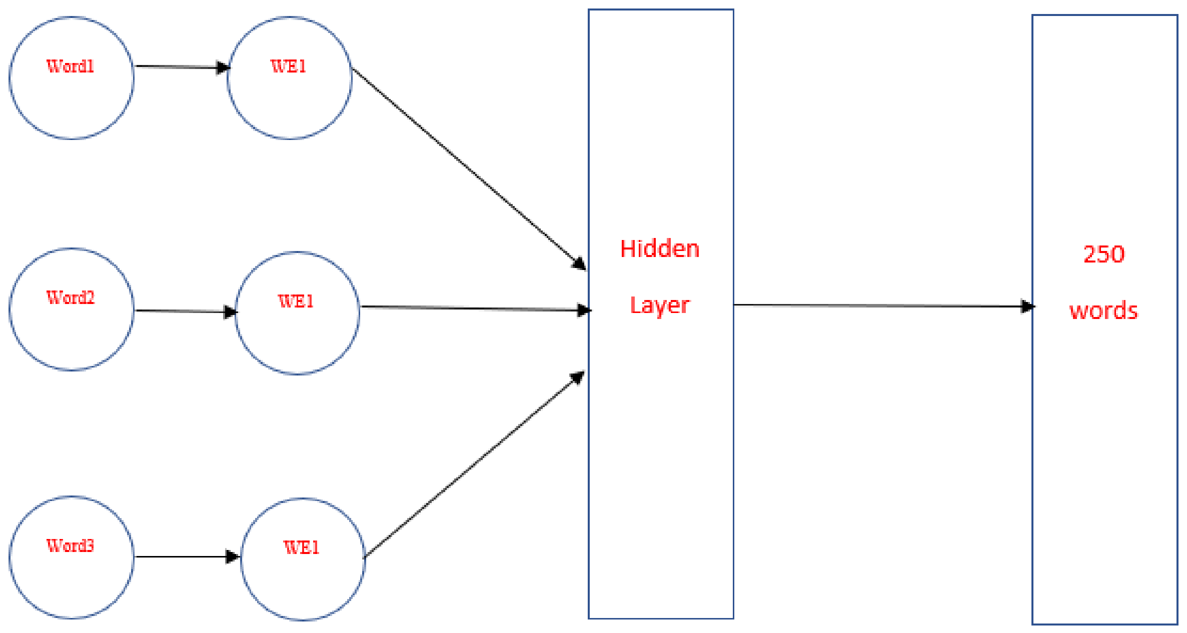
Zihao Jiao (1002213428)

Date: Feb26th, 2020

Question A

Network graph

(Input) (Word Embedding layer) (Hidden layer) (output)



(Logistic) (SoftMax)

Equations (Vectorized) :

$\boldsymbol{X \cdot W_{EM}^{T} = Embedding}$ Dimensions: $\mathbf{(1,3) \cdot (3,16)^{T}}$

$\boldsymbol{(Embedding \cdot W_{hid} + b_{hid}) = Hidden}$ Dimensions: $\mathbf{(1,16) \cdot (128,16)^{T} + (1,128)}$

$\boldsymbol{Logistic(Hidden) = H_{out}}$

$\boldsymbol{H_{out} \cdot W_{hid-output} + b_{output} = output_{in}}$ Dimensions: $\mathbf{(1,128) \cdot (250,128)^{T} + (1,250)}$

$\boldsymbol{output_{out} = softmax(output_{in})}$

Total number of training parameters:

Word_Embedding_weights: 3 × 16 = 48

Embedding_hid_weights: 128 × 16 = 2048

Hid_bias: 128

Hid_output_weights: 128 × 256 = 32000

Output_bias: 250

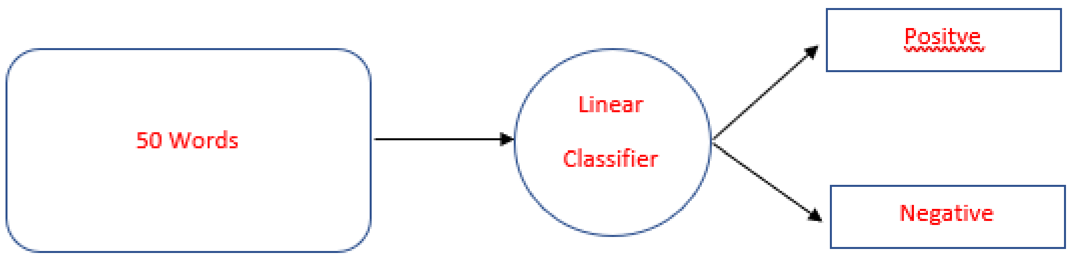
Thus,

Total number of trainable parameters : 48+2048+128+32000+250 = **34474**

The largest number of trainable parameters is **‘***Hid_output_weights’* 32000.

Question B

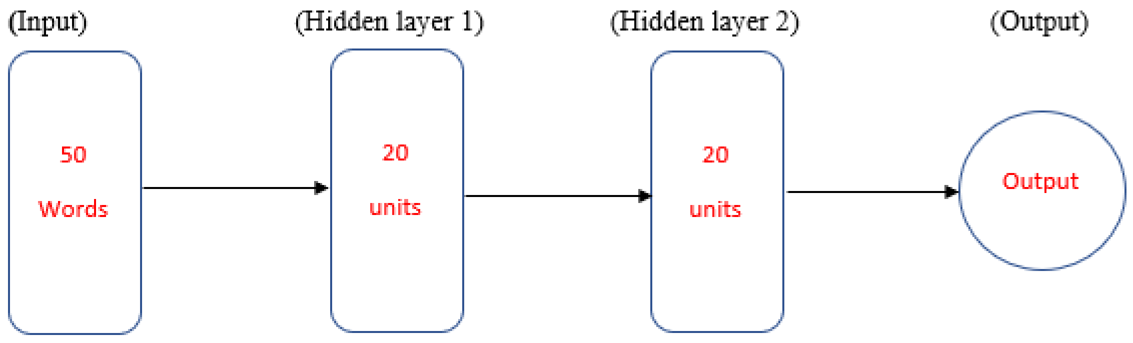
Linear Model



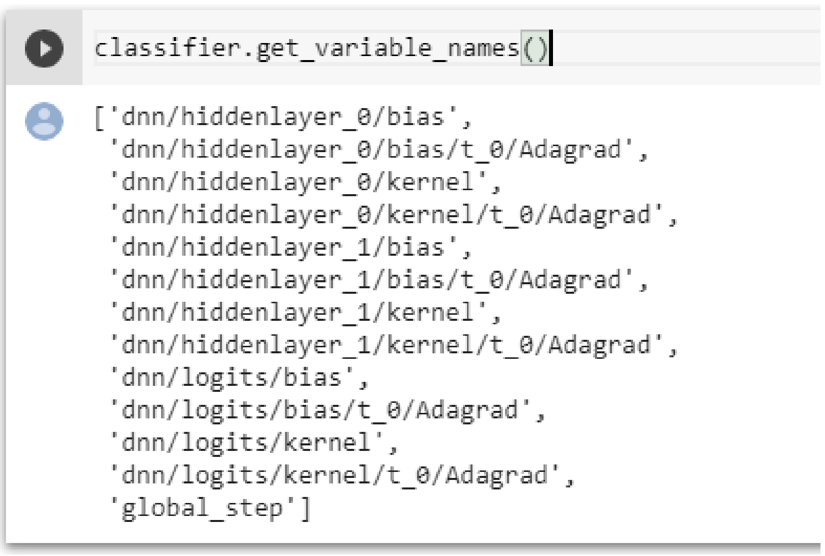
Equations (vectorized) :

$Y = W^T \cdot X + b$ Dimensions: $(1,50)^T \times (50,1) + (1,)$

DNN model



By running this command, we can see there are three weights(kernel) and three biases.



```
classifier.get_variable_names()
```

```
['dnn/hiddenlayer_0/bias',
 'dnn/hiddenlayer_0/bias/t_0/Adagrad',
 'dnn/hiddenlayer_0/kernel',
 'dnn/hiddenlayer_0/kernel/t_0/Adagrad',
 'dnn/hiddenlayer_1/bias',
 'dnn/hiddenlayer_1/bias/t_0/Adagrad',
 'dnn/hiddenlayer_1/kernel',
 'dnn/hiddenlayer_1/kernel/t_0/Adagrad',
 'dnn/logits/bias',
 'dnn/logits/bias/t_0/Adagrad',
 'dnn/logits/kernel',
 'dnn/logits/kernel/t_0/Adagrad',
 'global_step']
```

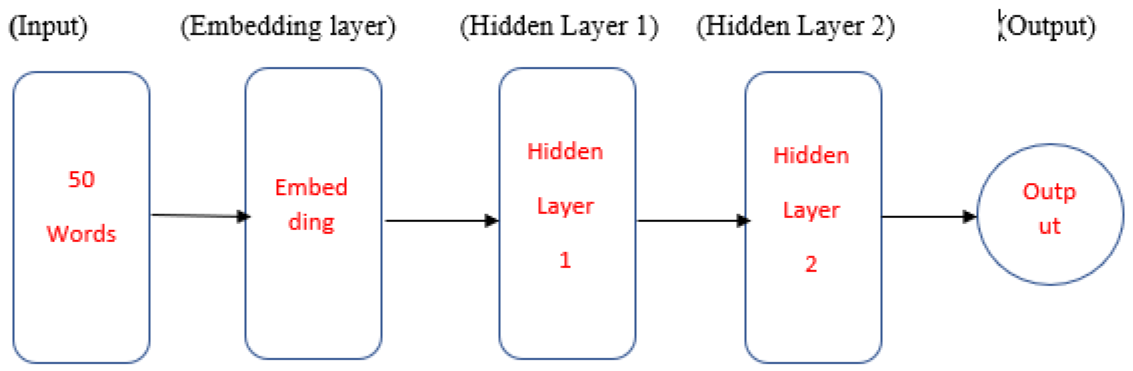
Equations (vectorized) :

Hidden_1 = $W_{input\ to\ hid1}^{T} \cdot X + b1$ Dimensions: $(50,20)^{T} \cdot (50,1) + (20,1)$

Hidden_2 = $W_{hid1\ to\ hid2}^{T} \cdot Hidden\_1 + b2$ Dimensions: $(20,20)^{T} \cdot (20,1) + (20,1)$

Output = $W_{hid2\ to\ output}^{T} \cdot Hidden\_2 + b3$ Dimensions: : $(1,20)^{T} \cdot (20,1) + (1,)$

DNN + Embedding Model



Equations (vectorized) :

Embedding = $W_{emb}^{T} \cdot X$ Dimensions: $(2,50) \cdot (50,1)$

Hidden_1 = $W_{input\ to\ hid1}^{T} \cdot Embedding + b1$ Dimensions: $(20,2) \cdot (2,1) + (20,1)$

Hidden_2 = $W_{hid1\ to\ hid2}^{T} \cdot Hidden\_1 + b2$ Dimensions: $(20,20) \cdot (20,1) + (20,1)$

Output = $W_{hid2\ to\ output}^{T} \cdot Hidden\_2 + b3$ Dimensions: $(1,20) \cdot (20,1) + (1,)$

Task 1: Use a Linear Model with Sparse Inputs and an Explicit Vocabulary

We are using 50 informative terms as our dataset, then create a linear classifier. Detailed structural of model can refer previous question. Then use 1000 iterations with 0.1 learning rate to train the model.

The performance of linear classifier is good, it got accuracy 0.788 for training set and 0.785 for testing set.

Task 2: Use a Deep Neural Network (DNN) Model

For task 2 we changed our linear model to a DNN model. It has two hidden layers, each one with 20 hidden units. The model has three weights three biases, the output from DNN is a single value.

The performance of DNN model better than linear model, it achieves 0.88 accuracy on training set and 0.8 for testing set. Basically, all evaluation metrics are better than linear model.

Task 3: Use an Embedding with a DNN Model

As for task 3, we implement previous DNN model using an embedding column. An embedding column takes sparse data as input and returns a lower-dimensional dense vector as output.

The embedding column projects data into two dimensions, thus, the weight of embedding layer is (50,2). Then, like previous DNN model, we have two hidden layers of 20 hidden units each. We train the model with leaning rate 0.1.

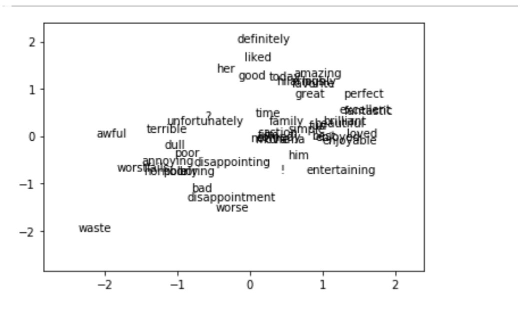
Our performances are not improved after adding embedding layer, the accuracy is fall to 0.787 and 0.782 for training set and testing set respectively.

Task 4: Convince yourself there's actually an embedding in there

Because from previous task, we added a embedding column to DNN model, we cannot tell what is actually different compare to DNN. In this task, we print the dimension of embedding layer out, the shape of it is (50,2) meaning that our 50 informative words are embedding to a two-dimensional lower place.

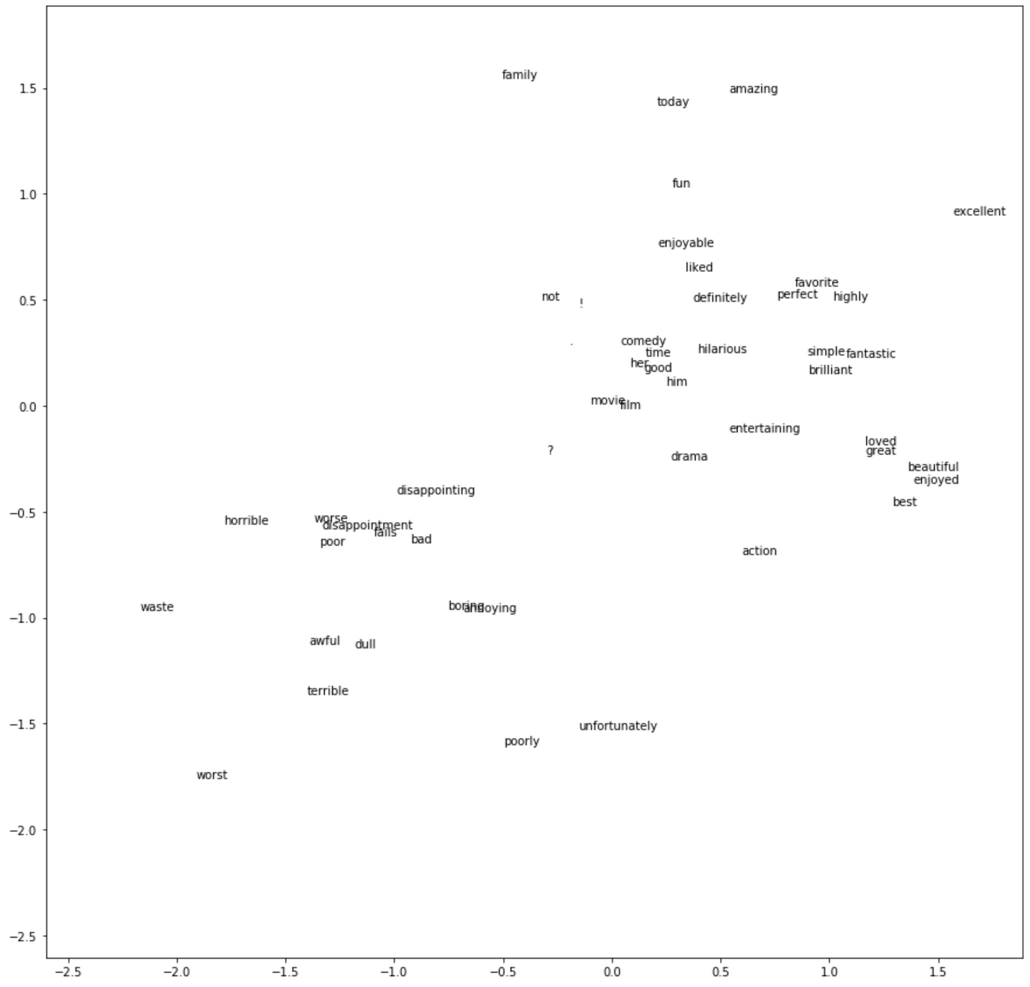
Task 5: Examine the Embedding

1. Run the following code to see the embedding we trained in Task 3. Do things end up where you'd expect?



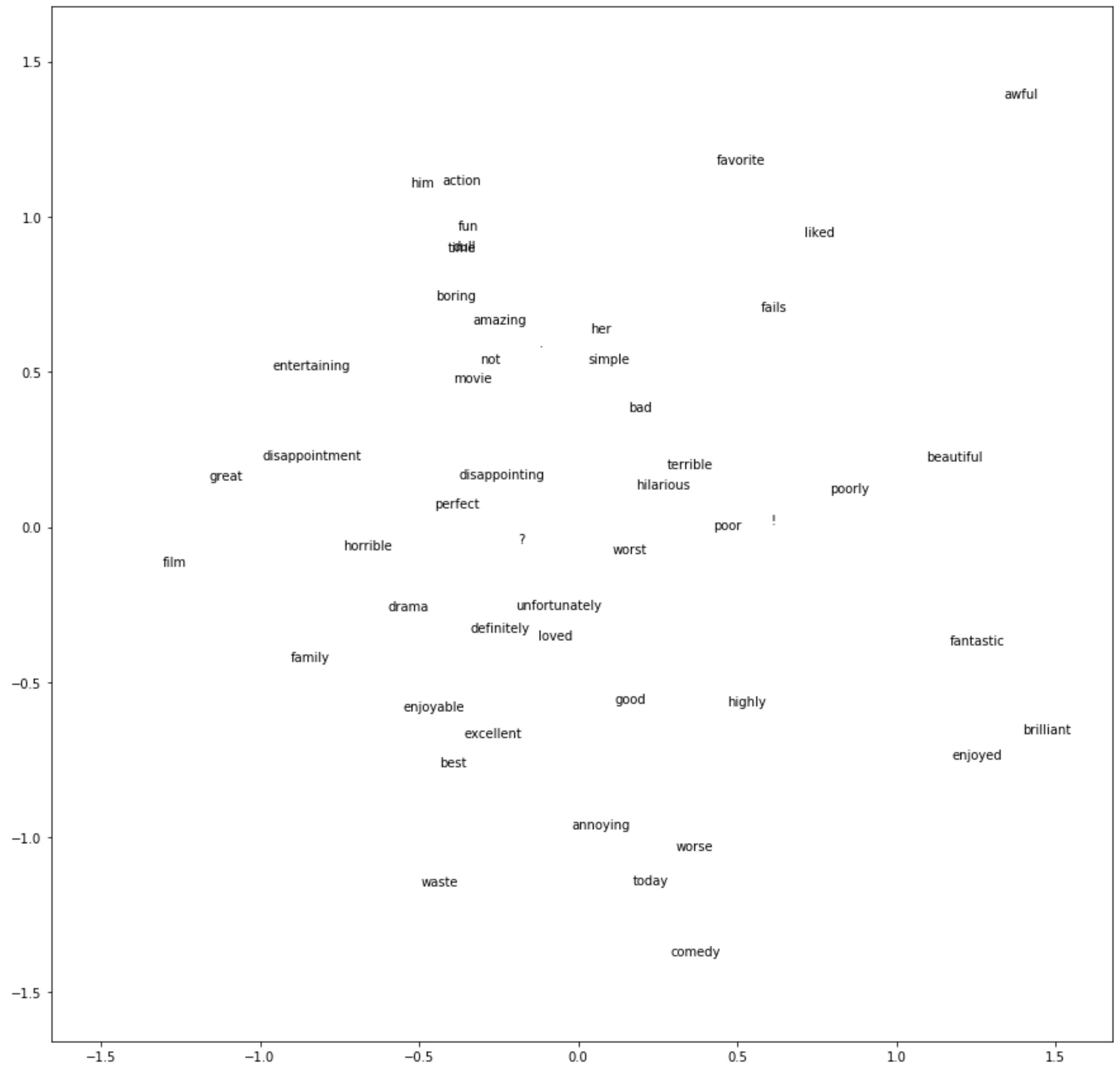
Not really like I would expect, this graph shows the words distribution without embedding layer implemented. The visualization not really good because some words are narrowed together, and also hard to see where the decision boundary is.

2. Re-train the model by rerunning the code in Task 3, and then run the embedding visualization below again. What stays the same? What changes?



The x-axis and y-axis scaled down, we can actually see there are two classes, left down class is negative sentiment, right up class is positive sentiment. Words with extreme sentiment expression like ‘excellent’, ‘amazing’ or ‘worst’, ‘terrible’ are far way from the decision boundary, because the model confident thinking they are either belong positive class or negative class. The majority words classes stay same, which means that embedding layer can project words into another dimension that would make classification more clearly.

3. Finally, re-train the model again using only 10 steps (which will yield a terrible model). Run the embedding visualization below again. What do you see now, and why?

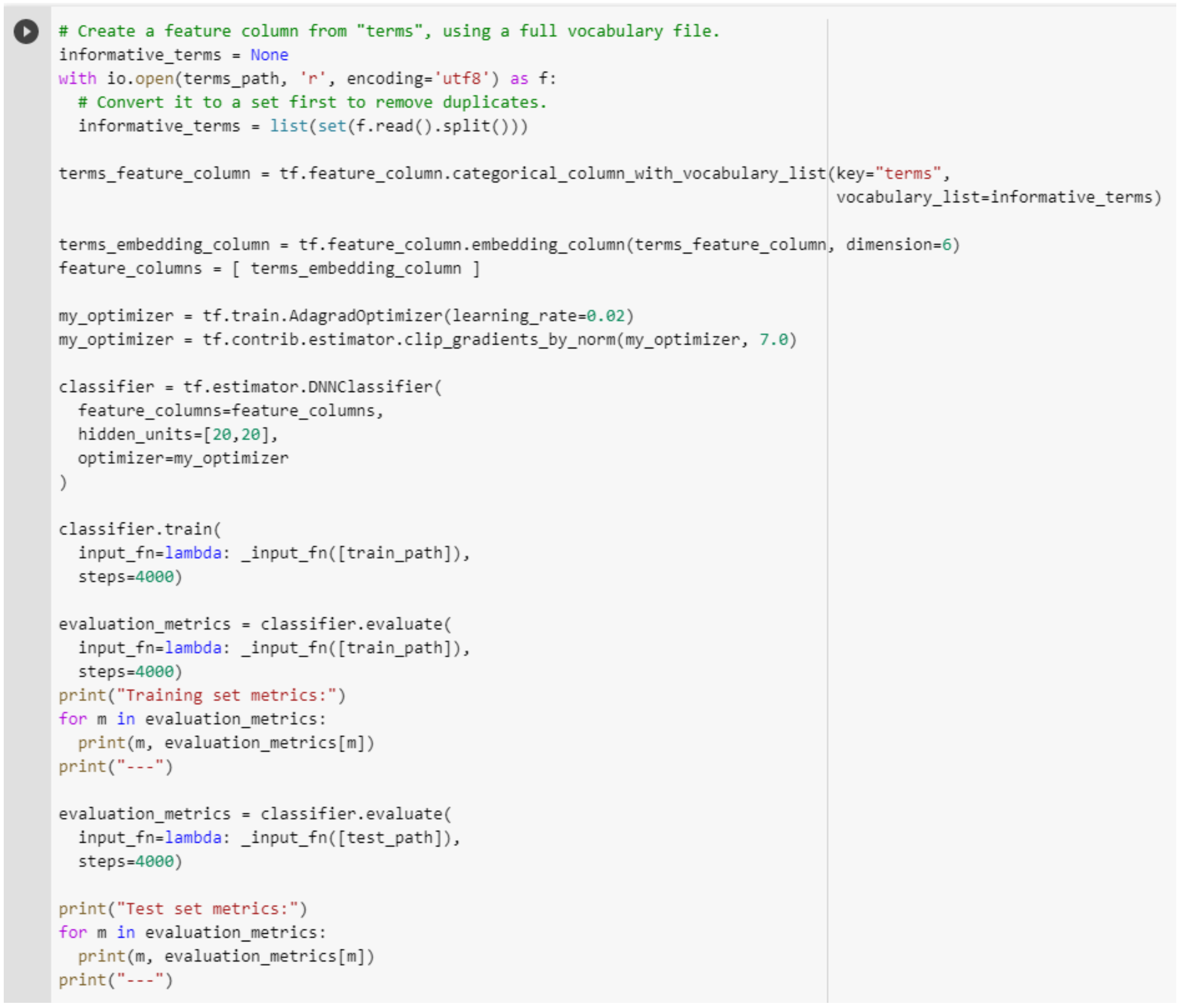


The visualization results look really bad, the model with only 10 steps training leads wrong and unclear prediction. All the words for positive and negative classes mixed together, no decision boundary, no visualization of two classes detected.

Task 6: Try to improve the model's performance

- Using the full vocabulary file with all 30,716 terms.
- Changing the embedding dimension to 6
- Set the learning rate is 0.02
- Set the optimizer is 7.0
- Set two hidden layers each has 20 hidden units
- Set 4000 steps

The performance after changing these I got 0.87232 accuracy for training set, and 0.8468 accuracy for testing set. Increased a lot compare to baseline evaluation result.
The code and performance results are below.



```python
# Create a feature column from "terms", using a full vocabulary file.
informative_terms = None
with io.open(terms_path, 'r', encoding='utf8') as f:
  # Convert it to a set first to remove duplicates.
  informative_terms = list(set(f.read().split()))

terms_feature_column = tf.feature_column.categorical_column_with_vocabulary_list(key="terms",
                                                                                 vocabulary_list=informative_terms)

terms_embedding_column = tf.feature_column.embedding_column(terms_feature_column, dimension=6)
feature_columns = [ terms_embedding_column ]

my_optimizer = tf.train.AdagradOptimizer(learning_rate=0.02)
my_optimizer = tf.contrib.estimator.clip_gradients_by_norm(my_optimizer, 7.0)

classifier = tf.estimator.DNNClassifier(
  feature_columns=feature_columns,
  hidden_units=[20,20],
  optimizer=my_optimizer
)

classifier.train(
  input_fn=lambda: _input_fn([train_path]),
  steps=4000)

evaluation_metrics = classifier.evaluate(
  input_fn=lambda: _input_fn([train_path]),
  steps=4000)
print("Training set metrics:")
for m in evaluation_metrics:
  print(m, evaluation_metrics[m])
print("---")

evaluation_metrics = classifier.evaluate(
  input_fn=lambda: _input_fn([test_path]),
  steps=4000)

print("Test set metrics:")
for m in evaluation_metrics:
  print(m, evaluation_metrics[m])
print("---")
```

```
Training set metrics:
accuracy 0.8458
accuracy_baseline 0.5
auc 0.919583
auc_precision_recall 0.91868013
average_loss 0.36325192
label/mean 0.5
loss 9.081298
precision 0.83257675
prediction/mean 0.52073646
recall 0.86568
global_step 4000
---
Test set metrics:
accuracy 0.82152
accuracy_baseline 0.5
auc 0.9035165
auc_precision_recall 0.9002309
average_loss 0.39701253
label/mean 0.5
loss 9.925314
precision 0.81169534
prediction/mean 0.5175757
recall 0.83728
global_step 4000
---
```