Assignment-2 Report

Part 0. Guidance

0.1 How to run my code

Task 1

- Use the following command to run the **Least Square model** .

```
python task_1.py --model="Least_Square_model"
```

- Use the following command to run the **Perception model** on tast_1 data set. Learning rate can be set by hand. By the way, to prevent not stopping when meeting a linear inseperable problem, you can limit the number of epochs by **--max_epoch**.

```
python task_1.py \
    --model="Perception_model" \
    --learning_rate=0.02 \
    --max_epoch=10\
```

Task 2

- **First version of model training.** There is a special option **--auto_terminate**, which determines whether to autamatically terminate the training process.

```
#auto_terminate: boolean, if True, the auto termination monitor will work.

#observe_loss_sequence_length: define the length of the recent loss sequence. The
auto monitor determines whether to terminate the training process by
loss sequence. (calculate means/ variance)

# terminate_threshold: if (mean abs)dif-loss < threshold, then terminate.

(1)
#do not automatically terminate. Instead, run all the epochs.
python normal_solve.py \
    --learning_rate=0.5 \
    --learning_rate_decay=0.9 \
    --steps_per_decay=100 \
    --batch_size=32 \
    --max_epoch_num=20 \
    --regularization_rate=1e-4 \
    --auto_terminate=False \


(2)
#automatically terminate judgement
python normal_solve.py \
    --learning_rate=0.5 \
    --learning_rate_decay=0.9 \
    --steps_per_decay=100 \
    --batch_size=32 \
    --max_epoch_num=100 \
    --regularization_rate=1e-4 \
    --auto_terminate=True \
    --observe_loss_sequence_length=10 \
    --terminate_threshold=0.01 \
```

- **Second version of model training.** I divide the original training set into two parts. One is for training, and another one is for validating. Using validation set can help me select good weights.

```
#epoch_num: define the data scale
#validation_proportion: define the propotion of validation set.
#steps_per_validation: define how many steps between two validating processes.

python once_validation.py \
    --learning_rate=0.5 \
    --learning_rate_decay=0.9 \
    --steps_per_decay=100 \
    --batch_size=32 \
    --epoch_num=20 \
    --regularization_rate=1e-4 \
    --validation_proportion=0.125 \
    --steps_per_validation=5 \
```

- **Third version of model training**. In the third version, I try to use cross-validation to select several not bad weights. And then use weight average tech to combine them into a single model.

```
#cross_times: times of cross validation
#ultilize_num: select the top ultilize_num weights on validation set.

python cross_validation_SWA.py \
    --learning_rate=0.5 \
    --learning_rate_decay=0.9 \
    --steps_per_decay=100 \
    --batch_size=32 \
    --epoch_num=20 \
    --regularization_rate=1e-4 \
    --validation_proportion=0.125 \
    --steps_per_validation=5 \
    --cross_times=5 \
    --ultilize_num=4 \
```

Other work

- Run **data_statistics.py** to see the statistics information of task2 dataset.

```
python data_statistics.py
```

- Run the batch-size exploration code

```
python batch_size_explore.py
```

- Run the learning_rate exploration code

```
python learning_rate_explore.py
```

- Run the gradient check code

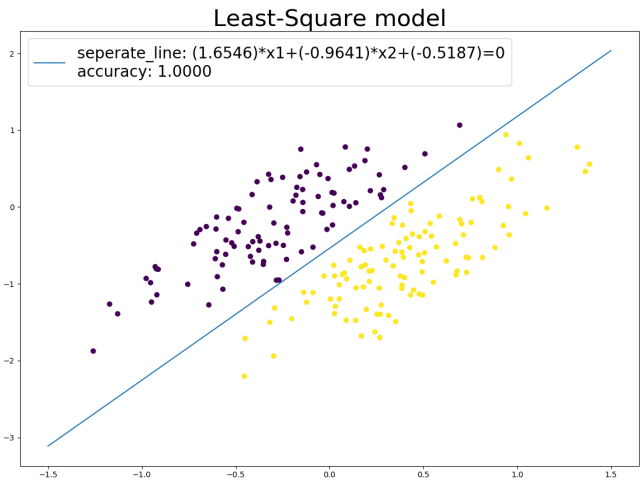
```
python check_gradient.py
```

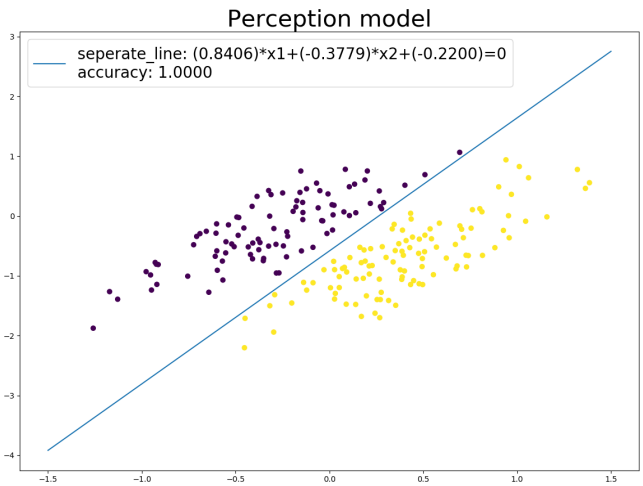
Part 1. Simple Linear Classification

1.0 Description

In this part, we are required to implement two basic linear classification models to distinguish two types of point groups. All the points are in a two dimensional space and $1.1 shows us the experiment results.

1.1 Experiment Results





Model	seperate line function
Least Square model	1.6546 x1 - 0.9641 x2 - 0.5187 = 0
Perception model	0.8406 x1 - 0.3779 x2 - 0.2200 = 0

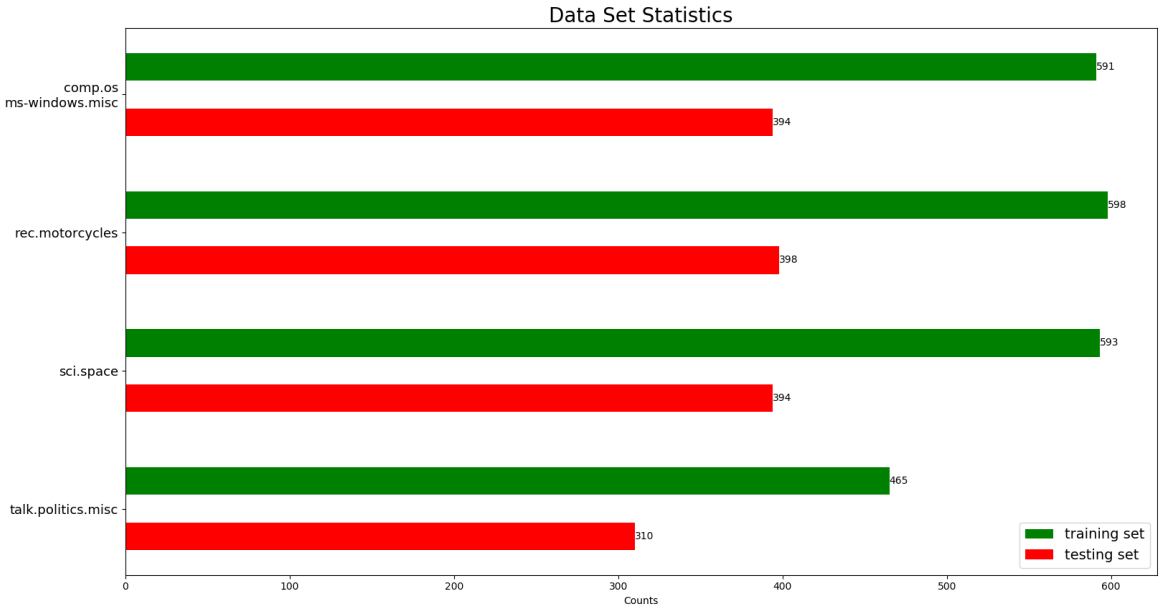
Part 2. Text Classification

2.0 Preview

2.0.1Description

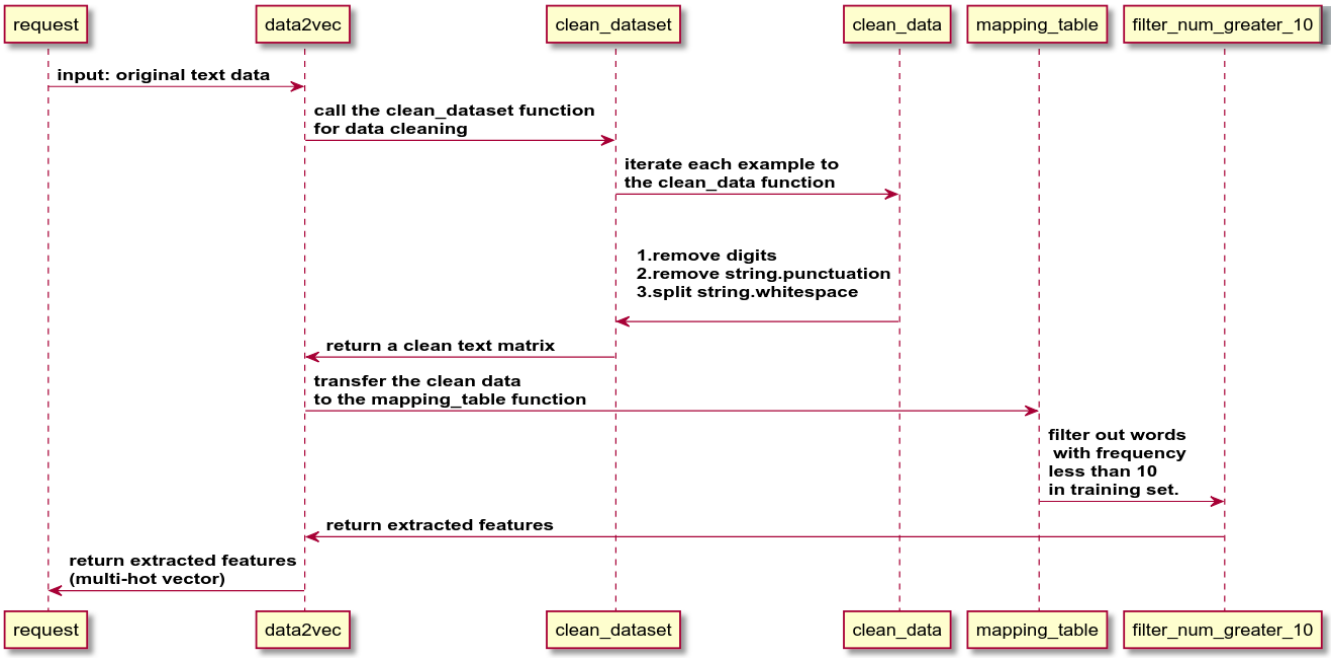
In this part, we are required to do a simple text classification task. The data is from sklearn.dataset.fetch_20newsgroups and we just extract examples belonging to the certain four classes which are **comp.os.ms-windows.misc**, **rec.motorcycles**, **sci.space** and **talk.politics.misc**. I have calculated the quantity distribution information of the data and details are as follows.

2.0.2Dataset Statistics



2.1 Data Preprocess

2.1.1 Pipeline Of Data to Multi-hot vector



2.1.2 From labels to Onehot vector

```
labels = np.eye(class_num)[labels]
```

2.2 optimization theory

2.2.1 Formula deduction

The BP formula will be deducted in this part. I firstly start from considering the situation of setting **batch size = 1** and then extend the conclusions to other situations that you can set any size of batch.

Loss-Function-Abstract

$$L(\vec{x};W) = CrossEntropyLoss(Softmax(W\vec{x}), labels)$$

hints: I put the bias b into W for simplification. And each input feature x needs to append element 1 at the begining.

Softmax Layer

$$Softmax(\vec{a}) = Softmax(\begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_n \end{bmatrix}) = \begin{bmatrix} S_1 \\ S_2 \\ \dots \\ S_n \end{bmatrix}, where S_j = \frac{e^{a_j}}{\Sigma_{i=1}^{n} e^{a_i}}$$

$$\frac{\partial S_i}{\partial a_j} = \begin{cases} \frac{e^{a_i}}{\Sigma_{i=1}^{n} e^{a_i}} + \frac{e^{a_i}}{-1} \frac{e^{a_j}}{(\Sigma_{i=1}^{n} e^{a_i})^2} = S_i(1 - S_i) & i == j \\ \frac{e^{a_i}}{-1} \frac{e^{a_j}}{(\Sigma_{i=1}^{n} e^{a_i})^2} = -S_i S_j & i \neq j \end{cases}$$

Softmax + CrossEntropy

$$L(S) = -\Sigma_{i=1}^{n} target_i * \log S_i$$

$$\frac{\partial L(S)}{\partial a_j} = \Sigma_{i=1}^{n} \frac{\partial L(S)}{\partial S_i} \frac{\partial S_i}{\partial a_j} = -\Sigma_{i==j} \frac{target_i}{S_i} S_i(1 - S_i) - \Sigma_{i \neq j} \frac{target_i}{S_i}(-S_i S_j) = S_j - target_i$$

$$\frac{\partial L(S)}{\partial \vec{a}} = \vec{S} - \overrightarrow{target}$$

Using- ChainRule

$$\frac{\partial L(\vec{x})}{\partial W} = \frac{\partial L(\vec{a})}{\partial \vec{a}} \frac{\partial (a(\overrightarrow{x}))}{\partial W} = \vec{x}(\vec{S} - \overrightarrow{target})^T = [\, x_{11} \quad x_{12} .. \quad x_{1n} \,]^T \left[\overrightarrow{S_1} - \overrightarrow{target_1} \right]^T$$

Extend-to-batch

$$\frac{\partial L(X)}{\partial W} = \frac{1}{N} [\, \overrightarrow{x_1}, \dots, \overrightarrow{x_N} \,] \, (S - target)^T = \frac{1}{N} \begin{bmatrix} [x_{11} & x_{12} .. & x_{1n}] \\ [x_{21} & x_{22} .. & x_{2n}] \\ \dots & \dots & \dots \\ [x_{N1} & x_{N2} .. & x_{Nn}] \end{bmatrix}^T \left[\overrightarrow{S_1} - \overrightarrow{target_1} \quad .. \quad \overrightarrow{S_N} - \overrightarrow{target_N} \right]^T$$

- **Update**

$$W^n = W^{n-1} - \frac{\partial L(X)}{\partial W} - 2\lambda \|W\|$$

2.2.2 Implementation by numpy without loop op

In $2.3.1, the BP update formula is deducted in matrics form, so there is no need to use loop operation here. All the operations are matrics-wide, which is very easy to be implemented by numpy. Souce code can be found in models.py.

2.2.3 Should the bias term be regularized?

My answer is No. There is no need to regularize the bias term.

- In logistic model, the bias has no attribution to the model complexity, so there is no need to add the bias into regularization terms.

  In book "The Elements of Statistical Learning", there is a chapter mention this point.

  > Panalization of the intercept would make the procedure depend on the origin chosen for Y; that is, adding a constant c to each of the targets yi would not simply result in a shift of the predictions by the same amount c.

2.2.4 How do I check my gradient calculation is correct

Using numerical solution method to calculate the gradient and compare with the gradient result calculated by formula. Source code can be seen in file: check_gradient.py.

$$\|judge:formula(\frac{\partial L}{\partial W_{ij}})-(\frac{\Delta L}{\Delta W_{ij}})\|<\sigma$$

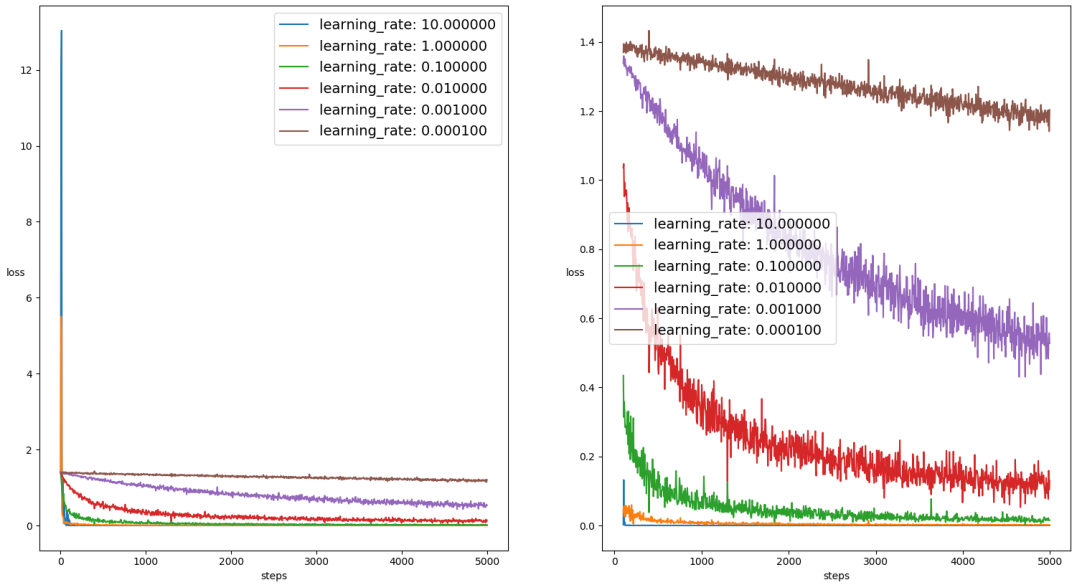
2.3 Training Details

2.3.1 How to determine the learning rate

I have tried different kinds of learning rate. By comparing the convergence trend of different loss curves, I can decide which learning rate is better.

In my experiments, I fix the batch size at 32, regularization_rate at 1e-4. Learning rate decay op isn't added and all the cases will run through 5000 steps.

- The following two subplots are nearly the same plots. The only difference between them is in subplot 2, the first 20 steps are cut off.



If we set a too small learning rate, the convergence of the loss curve may be too slow. On the contrary, too large learning rate may cause the model not converge.

According to the plots, it is obvious to know which orders of magnitude are suitable. I think learning rates between 0.1 and 1 may be proper.

Also learning rate decay skill can be added to help.

2.3.2 When to terminate the training procedure

Here I give out three Strategies to find out when to terminate the training procedure.

Brief view

- method 1: Observe the recent 20 training loss and analyce its mean difference. (Also variance can be considered.)

$$Mean\_dif = \Sigma_{n=0}^{9} |loss_{2n+1} - loss_{2n+2}|$$

  When Mean_dif < threshold, we can terminate the training procedure.

- method 2: Separate a validation set and use it to select a good weight. **Once we find the performance on validation set becomes worse and worse, we can terminate the training procedure.**
- method 3: Improved version of method 2. Using cross validation and weight average techs.

Implementation discussion

method 2: In method 2, I seperate a part of my training set as my validation set. While training, the validating process is also starts, and I will record the best weight in terms of the validation set. When the performance starts to be worse, I may stop the training procedure and choose the recorded best weight directly.

method 3: Similar with method 2. Method 3 will repeat method 2 for some times. For example, method 3 will repeat the method_2 for 8 times. After each repetition, the dataset will be disrupted again. Finally I will select the top 4 weights (sorted by accuracy on validation set) and apply weight average tech to gain a new, robust model.

Analyses:

Method 1 is a good way to terminate the training procedure. **It focus more on the behavioral trend of the loss and always can terminate the training procedure earlier, while Method 2 and method 3 focus more on model generalization ability.** The cons of Method 2 is that it reduced the number of training data. Method 3 makes up for this by using weight average op, but at the same time it sacrifices speed and needs more running time.

Experiment results of these three methods can be seen in part $2.5.2

2.4 Gradient Decent Analysis

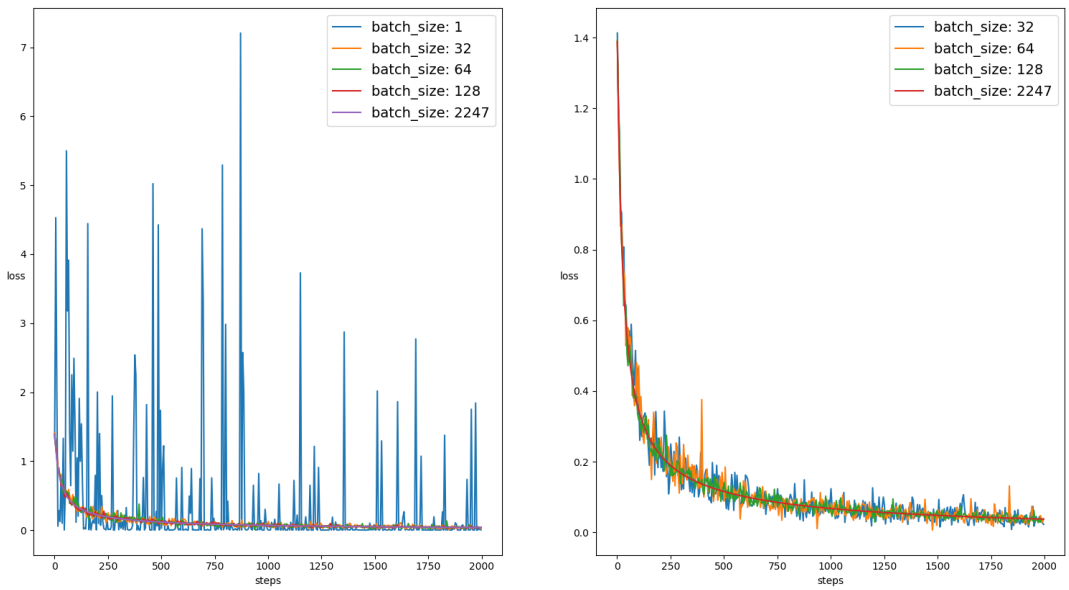
2.4.1 what do you observe by doing different ways of gradient descent?

I fix the learning rate at 0.2 and will decay it every 200 steps. Plots are as follows. The right plot excludes the solution of SGD since it brings too much noise in, affecting us to observe other curves.

The horizontal axis represents the number of steps and the vertical axis represents the error.

As we can see, the downward trend of the error curve is very similar. But the larger batch size we set, the smoother loss curve we will gain. On the contrary, small batch size setting will bring some noise into update process.

Also larger batch size needs more calculations in each update step, which will slow the training process.



2.4.2 can you tell what are the pros and cons of each of the three different gradient update strategies?

Gradient Decent Method	pros	cons
SGD	faster	more noise; more instability when updating
F-BGD	more correct gradient direction	more calculations, much slower, needs much memery
mini-BGD	middle	middle

SGD has faster running speed. But its gradient instability increases due to the calculation of individual data at each iteration. While **F-BGD** can calculate more accuracy gradients than those of SGD. But it will be slower.

So we need to make a trade-off between velocity and gradient accuracy. **Mini-BGD** just plays this role. It will have not bad speed, and can calculate not bad gradients.

By the way, if we have GPU resources and consider the parallel computing, we may choose the batch_size that can full the GPU memery.

2.5 experiment results and other work

2.5.1 experiment results

hints: max training steps are unified as about 1400. So the epoch number will be ajusted when batch_size changes.

Type	options	batch size	l_rate	decay rate	epoch num	test accur(top 1)	(top 2)
normal-train	auto_terminate=False	1	0.1	No Decay	3	0.923797	0.975267
		32	0.5	0.9	20	0.932487	0.976604
		64			40	0.929813	0.978610
		2247			1400	0.933155	0.977273
	auto_terminate=True observe_steps = 10 threshold=0.001(when batch size=1) threshold=0.01(when batch size = others)	1 threshold=0.001	0.1	No Decay	3	0.925134	0.979278
		32 threshold=0.01	0.5	0.9	20	0.933155	0.979278
		64 threshold=0.01			40	0.933155	0.975267
		2247 threshold=0.001			1400	0.927139	0.972594
once-validation	validation_proportion=0.125	1	0.1	No Decay	3	0.917112	0.975267
		32	0.5	0.9	20	0.933155	0.976604
		64			40	0.924465	0.975267
		2247			1400	0.922460	0.975936
cross-validation	v_proportion=0.125 cross_times=6 utilize_num=4	32	0.5	0.9	20	0.930481	0.975267
		64			40	0.931150	0.976604
		2247			1400	0.929813	0.978610

2.5.2. more work

- **learning rate decay** I use learning rate decay trick to make the learning rate be more flexible.

- **weight average tech** This technique is used in cross_validation training process in order to ultilize all the training data information and kick off some not good weight-model, since the initialization of weights influence.
- **random select** To guarantee the randomness of the data, I shuffle the whole training set each epoch.