HW1: Regression

Task Description

- COVID-19 daily cases prediction
- Training data: 2699 samples
- Testing data: 1078 samples
- Evaluation metric: Mean Squared Error (MSE)

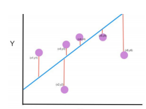
Data

- States (37, encoded to one-hot vectors)
- COVID-like illness (4)
  - cli、ili ...
- Behavior Indicators (8)
  - wearing_mask、travel_outside_state ...
- Mental Health Indicators (3)
  - anxious、depressed ...
- Tested Positive Cases (1)
  - **tested_positive (this is what we want to predict)**

data = 1(row num) + 37 states + 5 days * (4 illness + 8 behaviors + 3 mental + 1 results(positive)) = 118

Evaluation Metric

- Mean Squared Error (MSE)



$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

ground truth

your model (prediction)

#	Team	Members	Score
1	TA_RT		0.85800
	---- boss baseline ----		0.86161
	---- strong baseline ----		1.05728
	---- medium baseline ----		1.49430
	---- simple baseline ----		2.28371

modify 1 选择feature

54 + 70 + 86 + 102 results of 4 days

Submission and Description	Private Score	Public Score	Selected
pred (1).csv Complete (after deadline) · now	1.19474	1.09591	☐

modify 2 change optimizer

源代码用的是SGD，但是因为batch=1，所以只起到了GD的作用，同时SGD的作用是解决卡在local minimum所以根据这张图，change batch size and 加入momentum 作用应该不大

- SGDM

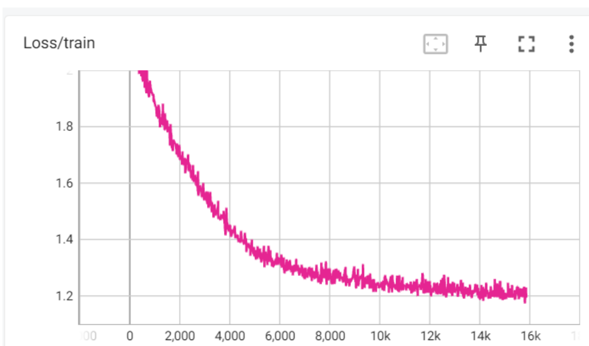
为梯度下降加入一个冲量，每次迭代移动的方向为**梯度的反方向向量**加上**上次移动的方向向量**，向量前面可能会有系数。

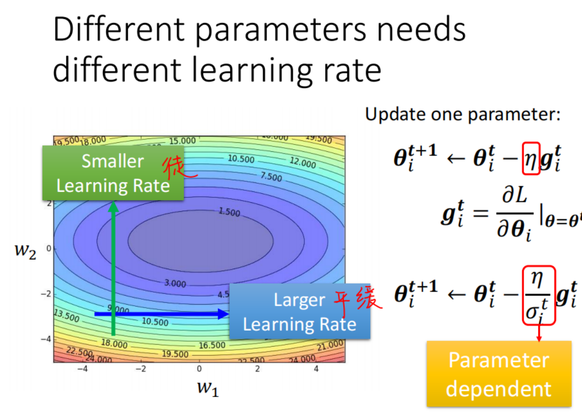
SGDM

$$\theta_t = \theta_{t-1} - \eta m_t$$
$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_{t-1}$$

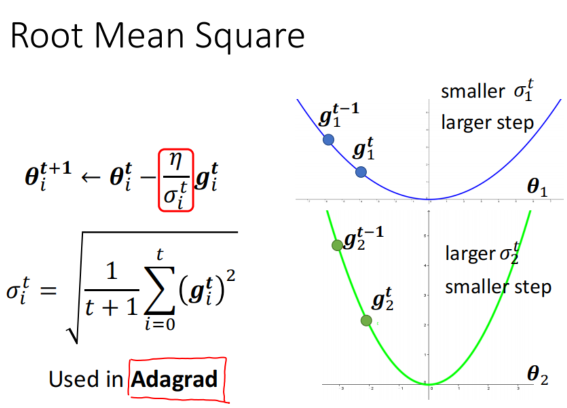
SGDM

但注意到since error surface is rugged, we need to have an adaptive learning rate

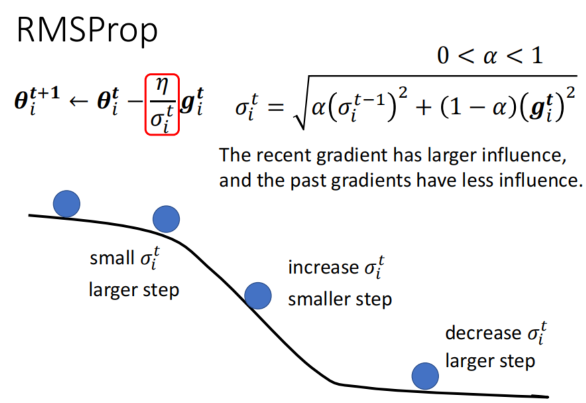




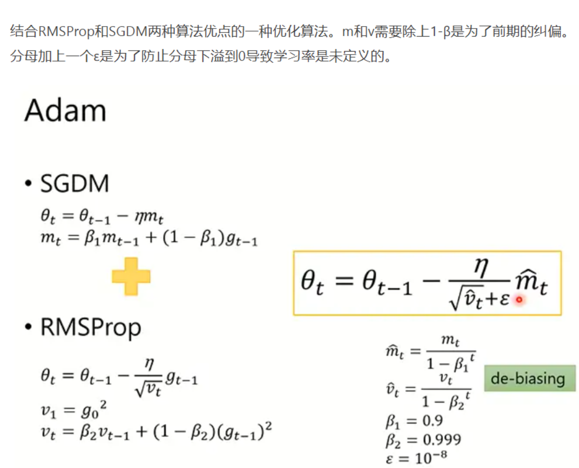
最基础的adaptive learning rate 的算法：Adagrad （use root mean square）



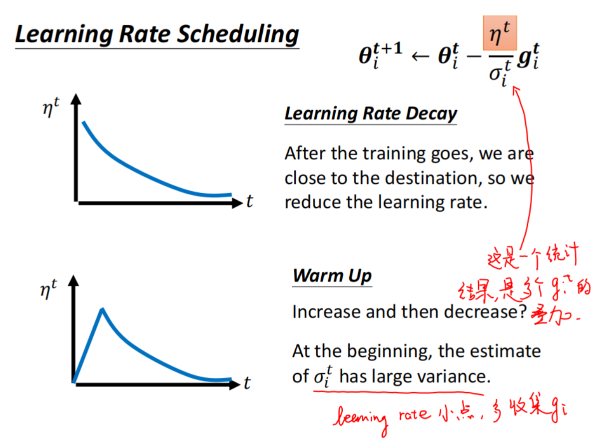
RMSProp 类比Adagrad的一种优化方法，与Adagrad不同的是学习率所除的分母。Adagrad学习率所除的分母会无限累加，导致后期参数更新幅度很小，RMSProp避免了这个问题。然而RMSProp无法解决卡在鞍点的问题。

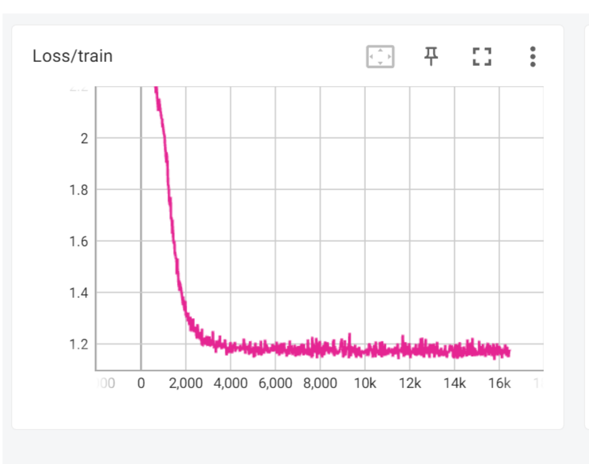


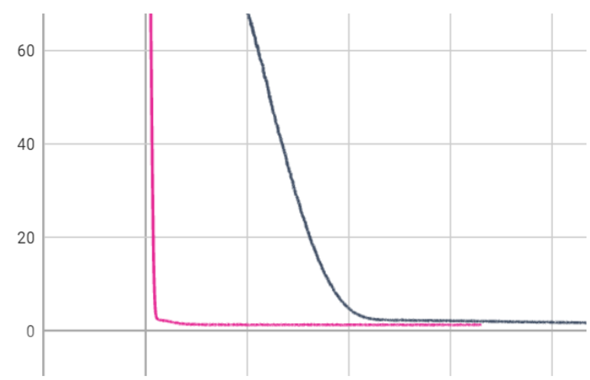
Adam: RMSProp + Momentum



Learning Rate Scheduling







pred2.csv Complete (after deadline) · now	**1.09905**	**1.02771**	☐

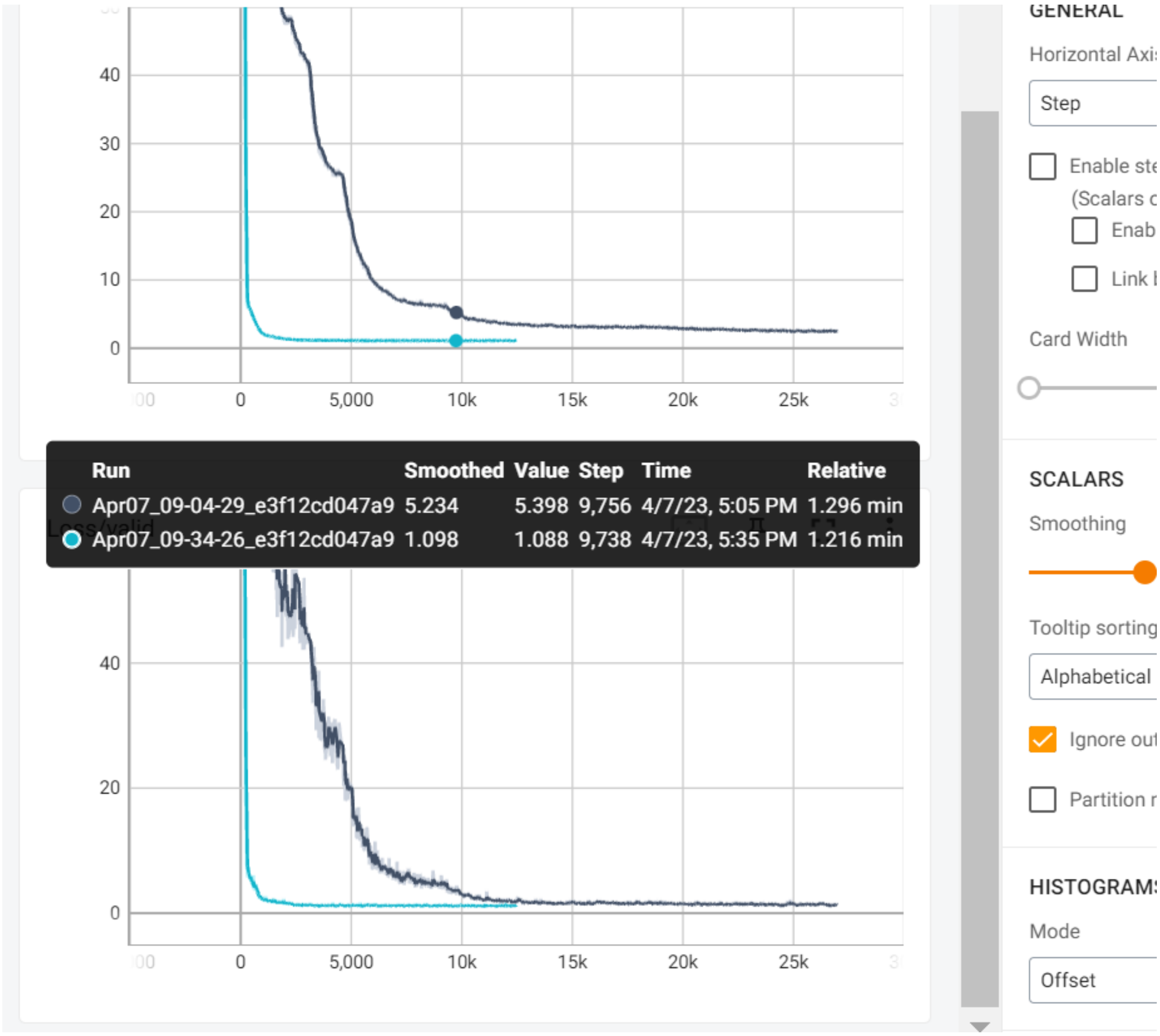
modify3 select index

```
Top 16 Best feature score
[90072.43401367 42336.37370139 26889.70377033 18870.55811361
 11290.79919656 10849.62638725 10420.334481   10365.26105926
 10055.85024148  9859.62690961  9636.4254885   9330.74236337
  9180.16305651  8703.90128488  7857.10815311  7840.26399997]

Top 16 Best feature index
[101  85  69  53 104  88 105  72  89  56  73  40  57  41 103 102]

Top 16 Best feature name
Index(['tested_positive.3', 'tested_positive.2', 'tested_positive.1',
       'tested_positive', 'hh_cmnty_cli.4', 'hh_cmnty_cli.3',
       'nohh_cmnty_cli.4', 'hh_cmnty_cli.2', 'nohh_cmnty_cli.3',
       'hh_cmnty_cli.1', 'nohh_cmnty_cli.2', 'hh_cmnty_cli',
       'nohh_cmnty_cli.1', 'nohh_cmnty_cli', 'ili.4', 'cli.4'],
      dtype='object')
```

```
selector = SelectKBest(score_func=f_regression, k=k)
result = selector.fit(x_data, y_data)
```

The SelectKBest class is a univariate feature selection method that selects the k highest scoring features based on a specified scoring function. In this case, the scoring function used is f_regression, which is a statistical test for **linear regression that measures the linear relationship between each feature and the target variable.**

The fit() method ca**lculates the scores of each feature based on the scoring functio**n and selects the top k features with the highest scores.

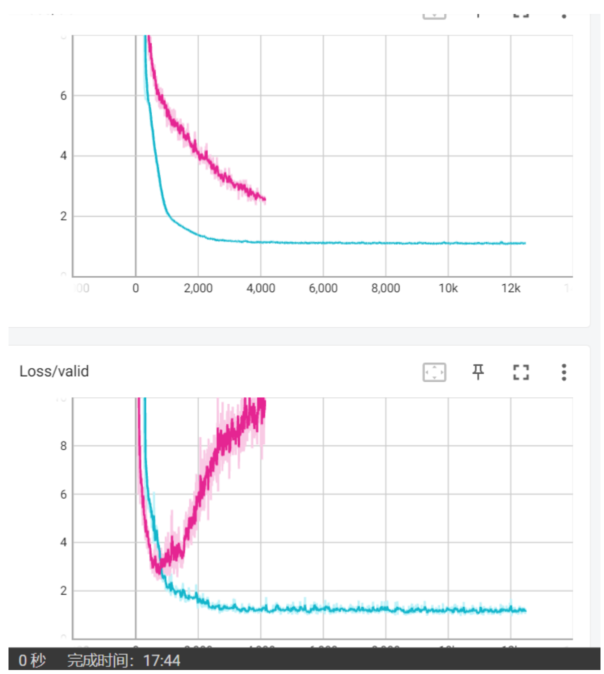


Submission and Description	Private Score	Public Score	Selected
pred_lala.csv Complete (after deadline) · now	**0.987**	**0.97838**	

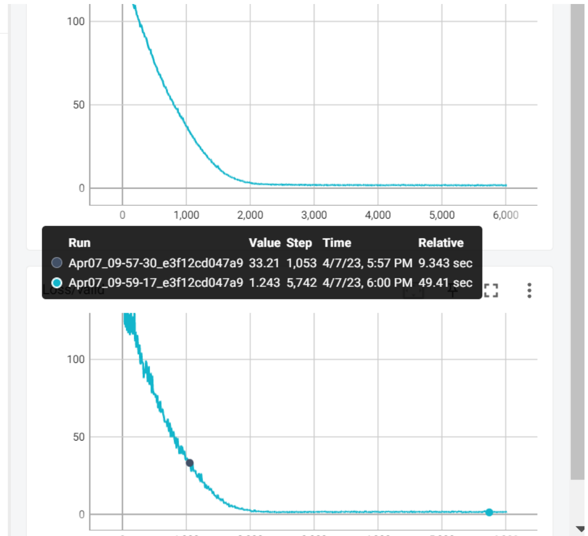
modify4 change network

```python
# TODO: modify model's structure,
self.layers = nn.Sequential(
        nn.Linear(input_dim, 16),
        nn.ReLU(),
        nn.Linear(16, 8),
        nn.ReLU(),
        nn.Linear(8, 1)
)
```

```python
self.layers = nn.Sequential(
        nn.Linear(input_dim, 64),
        nn.ReLU(),
        nn.BatchNorm1d(64),
        nn.Dropout(0.2),

        nn.Linear(64, 32),
        nn.ReLU(),
        nn.BatchNorm1d(32),
        nn.Dropout(0.2),

        nn.Linear(32, 8),
        nn.ReLU(),
        nn.Linear(8, 1)
)
```



```
    nn.Linear(input_dim, 32)
    nn.ReLU(),
    nn.BatchNorm1d(32),
    nn.Dropout(0.3),

    nn.Linear(32, 16),
    nn.ReLU(),
    nn.BatchNorm1d(16),
    nn.Dropout(0.1),

    nn.Linear(16, 8),
    nn.ReLU(),
    nn.BatchNorm1d(8),

    nn.Linear(8, 1)
```

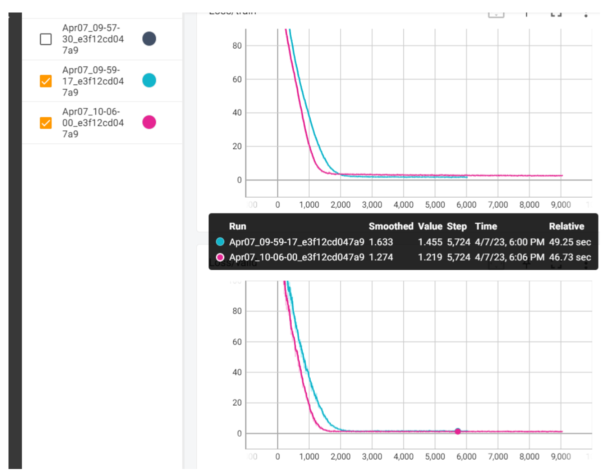


pred_modelchange.csv

Complete (after deadline) · now

14.66525 **6.02546**

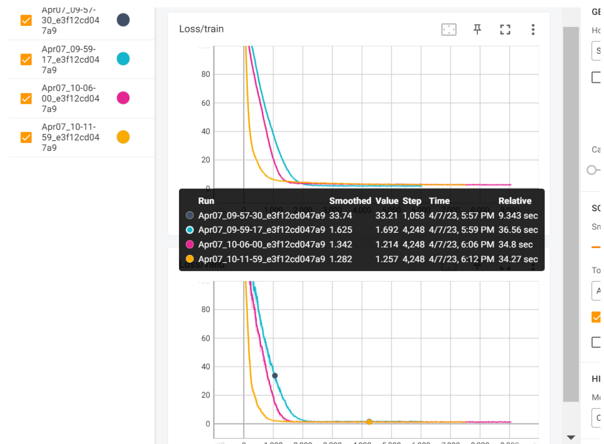
```
self.layers = nn.Sequential(
        nn.Linear(input_dim, 32),
        nn.ReLU(),
        nn.BatchNorm1d(32),
        nn.Dropout(0.3),

        nn.Linear(32, 16),
        nn.ReLU(),
        nn.BatchNorm1d(16),
        nn.Dropout(0.1),

        nn.Linear(16, 1)
)
```



pred_modelchange2.csv
Complete (after deadline) · now

1.32578 **1.96952**

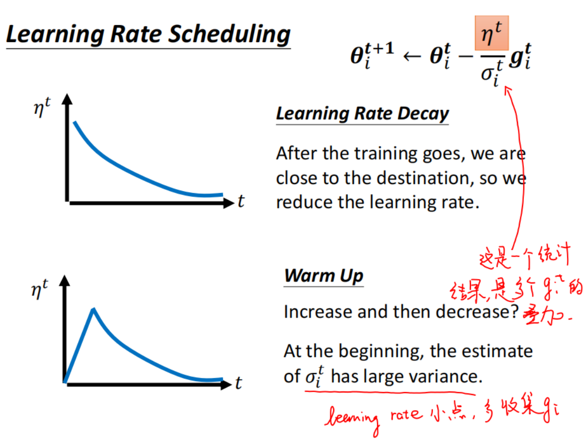
```
# TODO: modify model's structure, be awa
self.layers = nn.Sequential(
        nn.Linear(input_dim, 64),
        nn.LeakyReLU(0.2),
        nn.BatchNorm1d(64),
        nn.Dropout(0.2),

        nn.Linear(64, 16),
        nn.LeakyReLU(0.2),
        #nn.BatchNorm1d(10),
        nn.Dropout(0.1),

        nn.Linear(16, 1)
)
# self.layers = nn.Sequential(
#         nn.Linear(input_dim,16),
```



pred_modelchange3.csv
Complete (after deadline) · now

1.01822 **0.97619**

modify5 change learning rate



To schedule the combination of warmup and cosine annealing for your optimizer, you can use the PyTorch `CosineAnnealingWarmRestarts` scheduler. This scheduler allows you to set up warmup and cosine annealing schedules in a single step.

pred_modelchange4.csv

Complete (after deadline) · now

0.93288 **0.87783**