Introduction to Deep Neural Networks (Spring 2021)

Homework #1 (50 Pts, March 24)

Student ID 2017312172

Name 임정한

Instruction: We provide all source codes and datasets in Python. Please write your code to complete two models: *linear regression* and *logistic regression*. Besides, please measure the performance for each model.

NOTE: You should write your source code in the **'EDIT HERE' part** and do not edit other parts. You can check your code by executing the main code ('linear_main.py' for Linear Regression and 'logistic_main.py' for Logistic Regression).

TIP 1: The source code for the perceptron model is provided. Refer to the perceptron model if you are not familiar with the code structure.

TIP 2: You can try to implement the Scikit-learn version first and compare it with the results of your code.

[**Submission format**] When you upload your source code, please compress the following files and upload it with the file name **DNN_HW1_NAME_STUDENTID.zip**. Also, convert this file to pdf and upload it as well.

./linear_sklearn.py

./logistic_sklearn.py

./models/LinearRegression.py

./models/LogisticRegression.py

[20 pts] Linear regression

(1.1) [Implementation] Implement training and evaluation function in 'models/LinearRegression.py' ('train' and 'forward' respectively) using the gradient descent method. Training should be based on minibatch. Given training data (X, Y), the mean squared error (MSE) loss is defined as follows:

$$\boldsymbol{L} = \frac{\boldsymbol{1}}{\boldsymbol{N}} \sum_{(\boldsymbol{x_i}, \boldsymbol{y_i}) \in (\boldsymbol{X}, \boldsymbol{Y})} (\boldsymbol{\hat{y}_i} - \boldsymbol{y_i})^2$$

Answer: Fill your code (only EDIT HERE part) here. You also have to submit your code to i-campus.

```
    for epoch in range(epochs):
        mseloss = 0
        cnt = 0

        tmp = [[x_t, y_t] for x_t, y_t in zip(x, y)]
        random.shuffle(tmp)
        x = [x[0] for x in tmp]
        y = [x[1] for x in tmp]
        x = np.asarray(x)
        y = np.asarray(y)

        for i in range(0, x.shape[0]-batch_size+1, batch_size):

            x_batch = x[i: i + batch_size]
            y_batch = y[i: i + batch_size]
            y_batch = np.asarray(y_batch).reshape(batch_size, 1)

            y_pred = np.matmul(x_batch, self.W)
            m_grad = -np.dot(x_batch.transpose(), (y_batch-y_pred))/batch_size
            mseloss += np.mean(np.square(y_batch-y_pred))
            self.W = optim.update(self.W, m_grad, lr)
            cnt += 1

        mseloss = mseloss/cnt
```

(1.2) [Implementation] Implement the linear regression with scikit-learn library in '/linear_sklearn.py'. The linear regression using scikit-learn library uses an analytic solution. (Use the default hyperparameters.) Please refer to the sample code in the following link: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html.

Answer: Fill your code (only EDIT HERE part) here. You also have to submit your code to i-campus.

```
# TRAIN
```

```
model = LinearRegression().fit(train_x, train_y)


# EVALUATION
test_x, test_y = test_data
pred = model.predict(test_x)
pred = pred.reshape(-1, 1)
MSE = metric(pred, test_y)


print('MSE on Test Data : %.2f ' % MSE)
```

(1.3) [Experiments] For 'Graduate' and 'Concrete' datasets, please tune the number of training epochs and learning rate to minimize MSE. Report your best results for each optimizer. (In the case of 'Full-batch,' it is identical to the case where the mini-batch size is equal to the number of data.) Also, explain your results by comparing different methods.

Answer: Fill in the blank of the table.

Dataset	Batch	# of epochs	Learning rate	MSE
Graduate (# of data: 400)	Full-batch	4000	0.001	0.02
	Mini-batch (size=10)	4000	0.001	0.02
	Scikit-Learn			0.01
Concrete (# of data: 824)	Full-batch	4000	0.001	241.76
	Mini-batch (size=10)	4000	0.001	136.43
	Scikit-Learn			134.94

[30 pts] Logistic Regression

(2.1) [Implementation] Implement training and evaluation function in 'models/ LogisticRegression.py' ('train' and 'forward' respectively) using the gradient descent method. Training should be based on minibatch. Given training data (X, Y), the cross-entropy loss is defined as follows:

$$L = \frac{1}{N} \sum_{(x_i, y_i) \in (X, Y)} Cost(\hat{y_i}, y_i)$$

$$Cost(\hat{y_i}, y_i) = \begin{cases} -log(\hat{y_i}) & if\ y_i = 1 \\ -log(1 - \hat{y_i}) & if\ y_i = 0 \end{cases}$$

Answer: Fill your code (only EDIT HERE part) here. You also have to submit your code to i-campus.

```
for epoch in range(epochs):
    loss = 0
    #DATA SHUFFLING=====================



    tmp = [[x_t, y_t] for x_t, y_t in zip(x, y)]
    random.shuffle(tmp)
    x = [x[0] for x in tmp]
    y = [x[1] for x in tmp]
    x = np.asarray(x)
    y = np.asarray(y)


    #TRAIN==================================


    for i in range(0, x.shape[0]-batch_size+1, batch_size):


        x_batch = x[i: i + batch_size]
        y_batch = y[i: i + batch_size]
        y_batch = np.asarray(y_batch).reshape(batch_size, 1)


        sig_y_pred = self.forward(x_batch)
        diff = sig_y_pred - y_batch
        diff = diff.reshape(-1, 1)
        m_grad = np.matmul(x_batch.transpose(), diff)
        self.W = optim.update(self.W, m_grad, lr)
```

```
            loss += (-y_batch * np.log(sig_y_pred + epsilon) - (1 - y_batch) * np.log

            (1 - sig_y_pred + epsilon)).mean()

        loss /= x.shape[0]
```

(2.2) [Implementation] Implement the logistic regression with scikit-learn library in '/linear_sklearn.py'. Please refer to the sample code in the following link: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

Answer: Fill your code (only EDIT HERE part) here. You also have to submit your code to i-campus.

```
model = LogisticRegression(solver='liblinear', random_state=0).fit(train_x, train_y)

# EVALUATION

test_x, test_y = test_data

pred = model.predict(test_x)

pred = pred.reshape(-1, 1)

ACC = metric(pred, test_y)
```

(2.3) [Experiments] For 'Titanic' and 'Digit' datasets, please tune the number of training epochs and learning rate to maximize the accuracy. Report your best results for each training method. (In the case of 'Full-batch,' it is identical to the case where the mini-batch size is equal to the number of data.) Also, explain your results by comparing different methods.

Answer: Fill in the blank of the table.

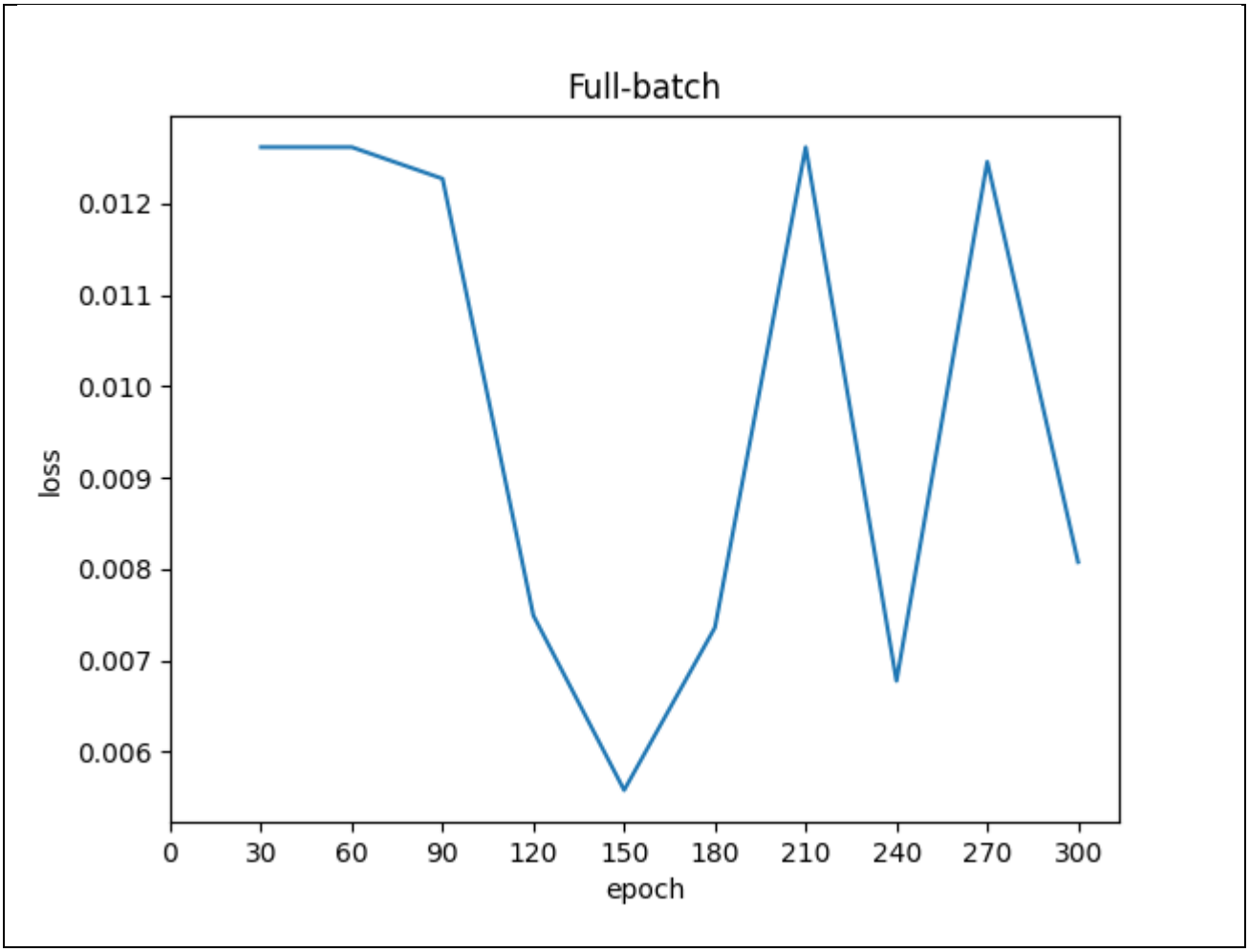
Dataset	Batch	# of epochs	Learning rate	Accuracy
Titanic	Full-batch	300	0.0005	0.667

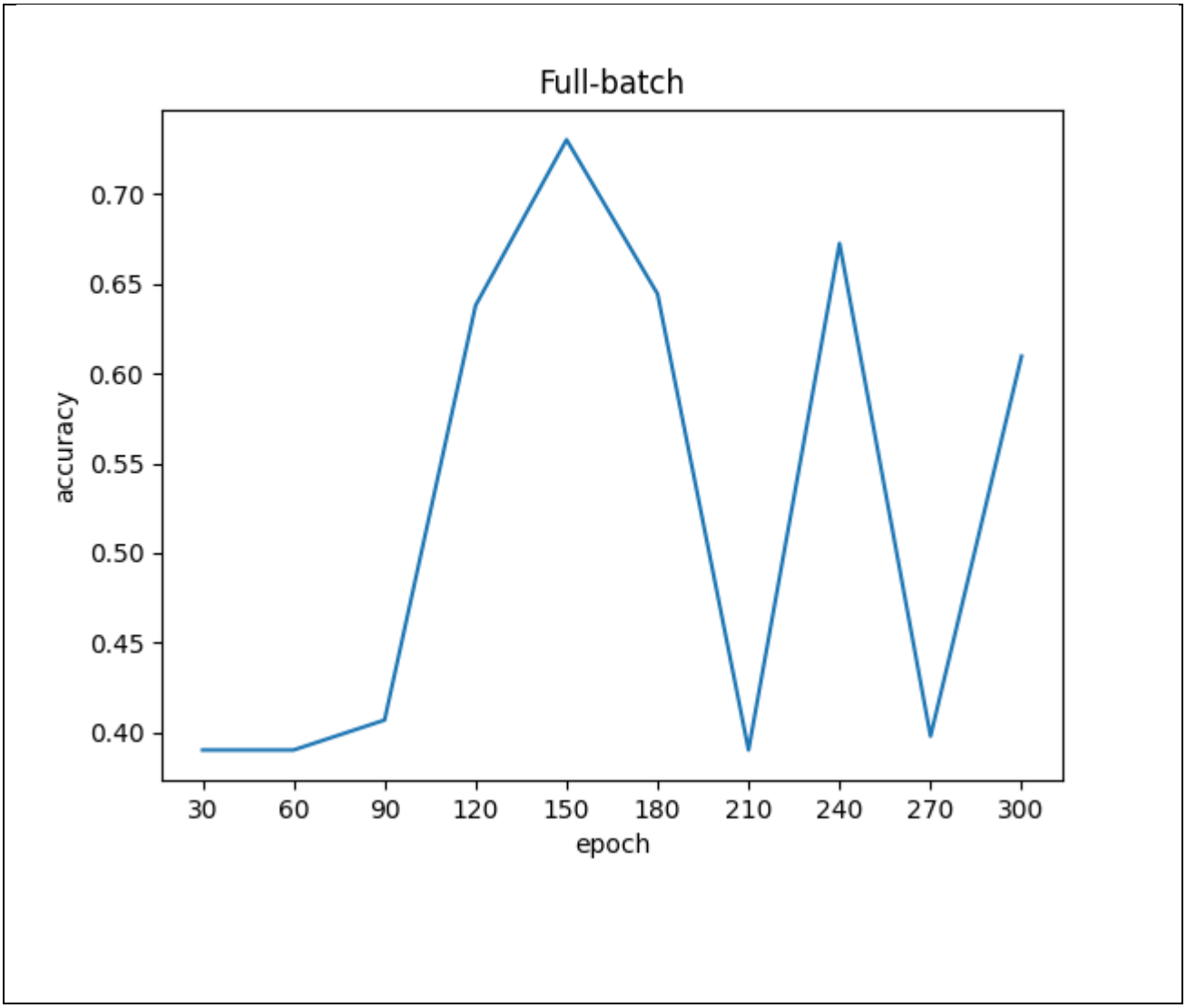
(# of data: 779)	Mini-batch (size=10)	300	0.0005	0.824
	Scikit-Learn			0.83
Digit (# of data: 11501)	Full-batch	300	0.0005	0.994
	Mini-batch (size=10)	300	0.0005	0.984
	Scikit-Learn			0.99

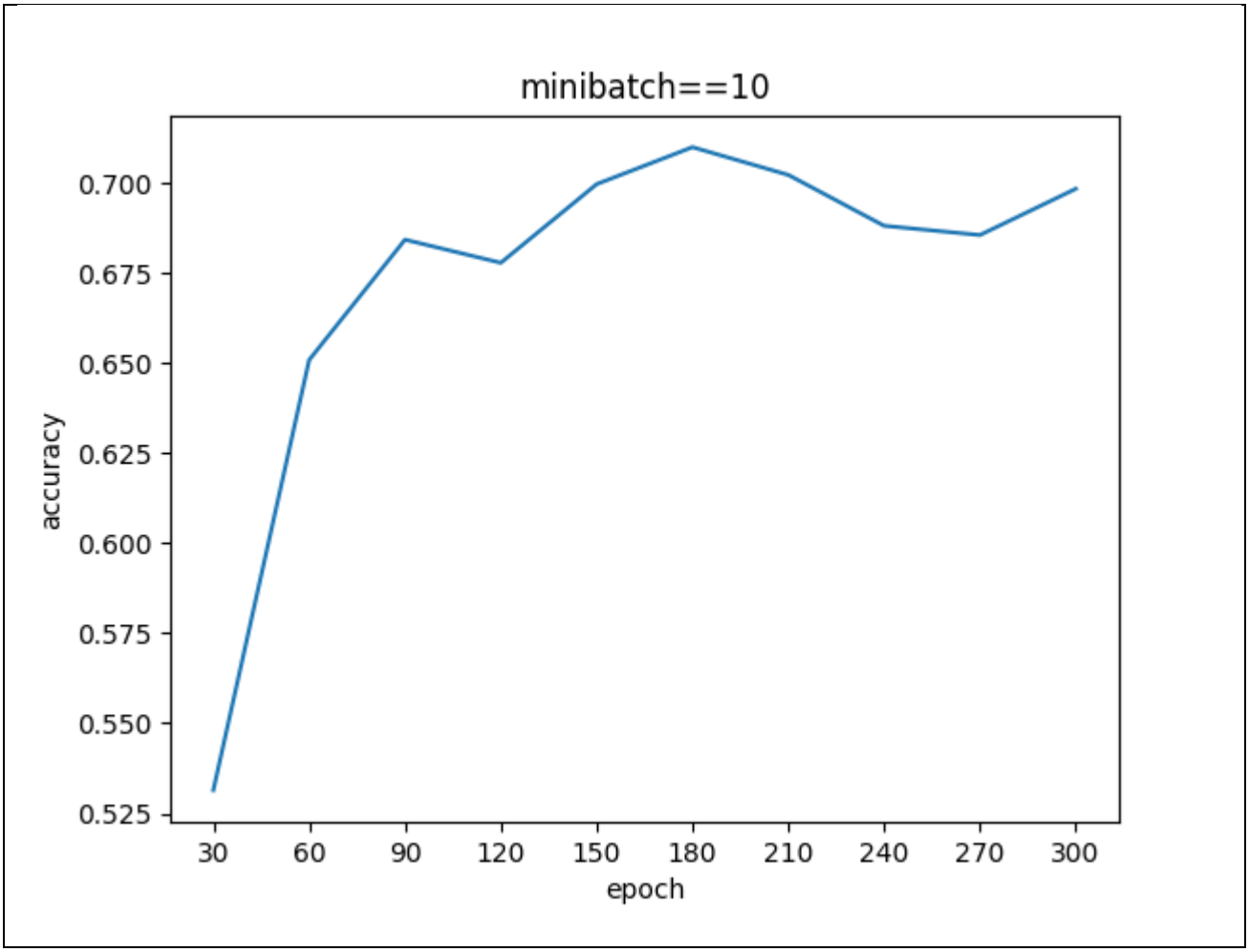
(2.4) [Experiments] For the 'Titanic' dataset, execute the logistic regression with full-batch training and mini-batch training. Given the following parameters, draw two plots each:1) a plot whose x-axis and y-axis are **epochs** and **accuracy**, and 2) a plot whose x-axis and y-axis are **epochs** and **loss**. Use 'matplotlib' for plotting the graph.

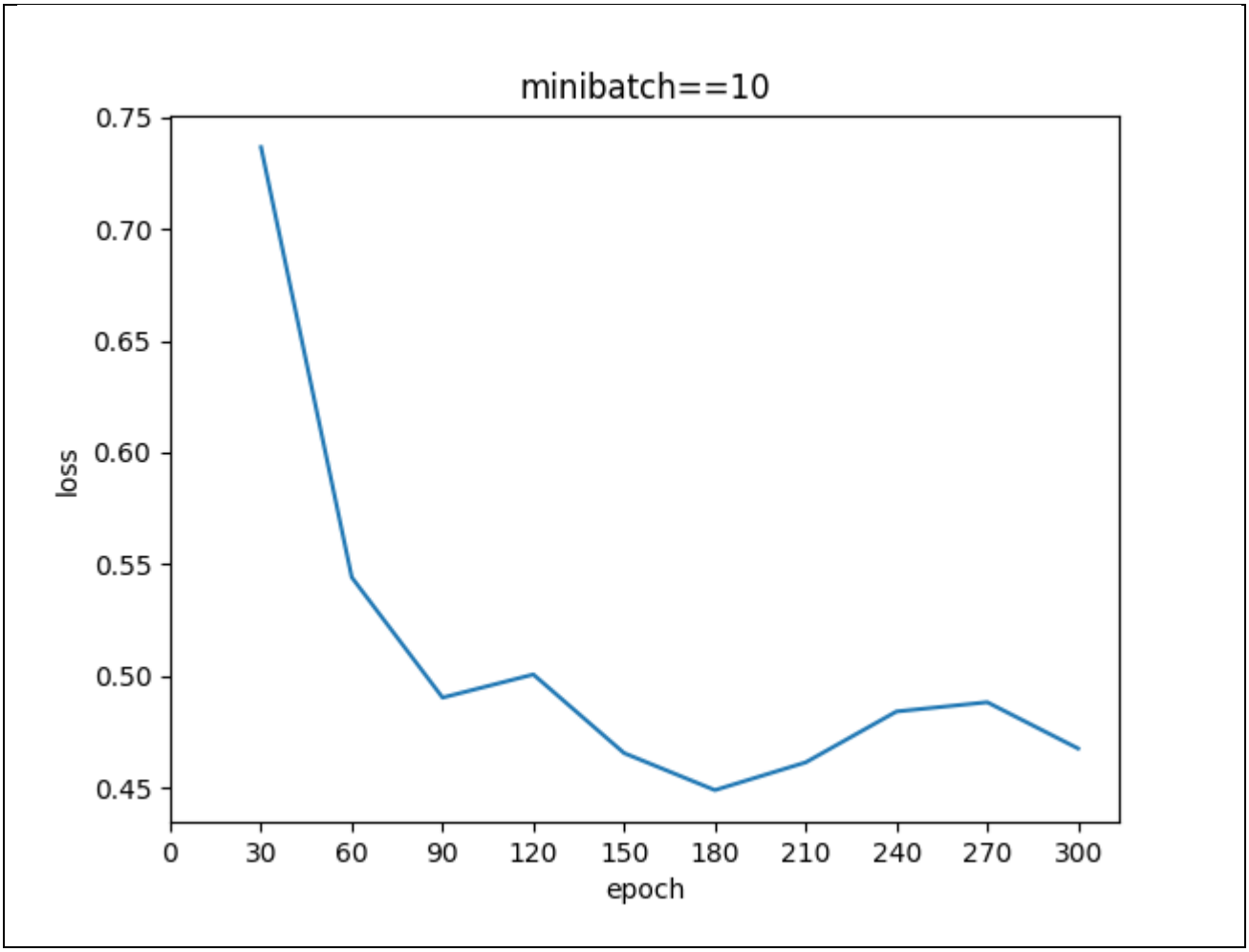
Parameter Settings	
Batch size	10
Learning rate	0.0005
Epsilon	0.01
Gamma	0.9
# of Epochs	30, 60, 90, …, 300

Answer: Draw the figure in the blank.


Full-batch
0.012
0.011
0.010
0.009
0.008
0.007
0.006
loss
0
30
60
90
120
150
180
210
240
270
300
epoch


Full-batch
accuracy
0.70
0.65
0.60
0.55
0.50
0.45
0.40
30
60
90
120
150
180
210
240
270
300
epoch


minibatch==10
0.700
0.675
0.650
0.625
0.600
0.575
0.550
0.525
accuracy
30
60
90
120
150
180
210
240
270
300
epoch


minibatch==10
0.75
0.70
0.65
0.60
0.55
0.50
0.45
loss
0
30
60
90
120
150
180
210
240
270
300
epoch