Introduction to Deep Neural Networks (Spring 2021)

Homework #1 (50 Pts, March 24)

Student ID 2019311195

Name 김지유

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:

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

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

```
# ========================= EDIT HERE ========================
y = y.reshape((x.shape[0], 1))
print("x.shape {}, y.shape{}".format(x.shape,y.shape))
print(int(x.shape[0]/batch_size))

for epoch in range(epochs):

    epoch_loss = []
    wd = np.zeros_like(self.W)

    for i in range(x.shape[0]//batch_size):
        start = i*batch_size
        end = start + batch_size
        x_batch = x[start:end]
        y_batch = y[start:end]
        y_predicted = self.forward(x_batch)
        err = (y_predicted-y_batch)

        #배치 손실 구하기
        batch_loss = np.mean(np.square(err))
        epoch_loss.append(batch_loss)

        #가중치 업데이트
        wd = (np.dot(x_batch.T, err)/batch_size)
        self.W = optim.update(self.W, wd, lr)

    #나머지 데이터로 학습하기
```

```
            if int(x.shape[0]%batch_size) != 0:
                start = int(x.shape[0]//batch_size)*batch_size
                x_batch = x[start:]
                y_batch = y[start:]
                y_predicted = self.forward(x_batch)
                err = (y_predicted-y_batch)

                #배치 손실 구하기
                batch_loss = np.mean(np.square(err))
                epoch_loss.append(batch_loss)

                #가중치 업데이트
                last_batch_size = x_batch.shape[0]
                wd = (np.dot(x_batch.T, err)/last_batch_size)
                self.W = optim.update(self.W, wd, lr)

            #에폭 손실 구하기
            final_loss = np.mean(np.array(epoch_loss))

            if epoch%1000 == 0:
                print("cost {}, batch_size {}, epoch {}".format(final_loss, batch_size, epoch))
        # ========================================================
```

(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.

```
# ========================== EDIT HERE ==========================
# DATA
DATA_NAME = 'Concrete'
# ================================================================


# ========================== EDIT HERE ==========================
# Make model & optimizer
model = LinearRegression()


# TRAIN
model.fit(train_x, train_y)


# EVALUATION
test_x, test_y = test_data
predicted_y = model.predict(test_x)
MSE = metric(predicted_y, test_y)
# ================================================================
```

(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	0.1	0.007
	Mini-batch (size=10)	10	0.1	0.007
	Scikit-Learn			0.010

Concrete (# of data: 824)	Full-batch	4500	0.1	136.20
	Mini-batch (size=10)	40000	0.001	135.88
	Scikit-Learn			134.94

Graduate

full-batch:

full-batch인 경우 learning rate가 작을수록 MSE가 커지는 결과를 볼 수 있었다.

mini-batch의 learning rate보다 큰 0.1이 이상적이었다.

learning rate = 0.1인 경우 대략 epoch = 0부터 epoch = 300까지 cost가 감소하다가 그 뒤로는 epoch이 늘어나도 큰 변화를 보이지 않았다.

실제로 learning rate = 0.1인 경우 epoch = 200부터는 아무리 epoch을 키워도 MSE가 변하지 않았다.

따라서 MSE가 낮게 나오는 learning rate와 epoch 조합 중 가장 시간 효율성이 좋은 learning rate = 0.1 epoch = 200을 이상적인 parameter로 뽑았다.

min-batch:

mini-batch의 경우 다양한 learning rate와 epoch의 조합이 가능했다.

learning rate가 0.1인 경우 대략 epoch 범위가 10부터 150까지까지 MSE가 0.007로 낮게 나왔고, epoch가 이 범위에서 멀어질수록 MSE가 높게 나왔다.

learning rate가 0.01인 경우 대략 epoch 범위가 100부터 1300까지 MSE가 0.007로 낮게 나왔고, epoch가 이 범위에서 멀어질수록 MSE가 높게 나왔다.

learning rate가 0.001인 경우 대략 epoch 범위가 100부터 14000까지 MSE가 0.007로 낮게 나왔고, epoch가 이 범위에서 멀어질수록 MSE가 높게 나왔다.

따라서 MSE가 낮게 나오는 learning rate와 epoch 조합 중 가장 시간 효율성이 좋은 learning

rate = 0.1 epoch = 10을 이상적인 parameter로 뽑았다.

Concrete

full-batch:

full-batch인 경우 learning rate가 클수록 적은 epoch으로 낮은 MSE에 빨리 도달하였다.

learning rate가 0.1인 경우 대략 epoch = 4500에서 MSE = 136.20이 나왔고, epoch =4500에서 멀어질수록 MSE가 증가하였다.

learning rate가 0.01인 경우 대략 epoch = 4000에서 MSE = 158.85이 나왔고, epoch가 증가할수록 MSE는 감소하였지만 epoch = 20000에서도 MSE = 138.13이 나올 정도로 좋지 않았다.

learning rate가 0.001인 경우에도 learning rate가 0.01인 경우와 마찬가지로 epoch가 증가할수록 MSE는 감소하였지만 epoch = 20000에서도 MSE = 182.72이 나올 정도로 좋지 않았다.

min-batch:

mini-batch인 경우 learning rate가 작을수록 적은 epoch으로 낮은 MSE에 빨리 도달하였고, epoch이 커질수록 MSE가 줄어들다 특정값에 수렴했다.

learning rate가 0.1인 경우 epoch가 커질수록 MSE가 줄어들다 epoch = 4000부터 MSE가 140.22에 머물렀다.

learning rate가 0.01인 경우 대략 epoch가 커질수록 MSE가 줄어들다 epoch = 30000부터 MSE가 136.00에 머물렀다.

learning rate가 0.001인 경우에도 epoch가 커질수록 MSE가 지속적으로 줄어들었다.

epoch = 40000에서는 MSE가 135.88, epoch = 90000인 경우 MSE가 135.16이 나왔다.

[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:

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

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

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

```
        # ======================== EDIT HERE ========================

        for epoch in range(epochs):


            epoch_loss = []

            wd = np.zeros_like(self.W)


            for i in range(x.shape[0]//batch_size):

                start = i*batch_size

                end = start + batch_size

                x_batch = x[start:end]   #x_batch.shape = (10,7)

                y_batch = y[start:end].reshape(-1,1) #y_batch.shape = (10,1)

                y_predicted = self.forward(x_batch) #y_predicted.shape = (10,1)

                err = (y_predicted - y_batch) #err.shape = (10,1)


                #배치 손실 구하기

                loss_arr = np.where(y_batch==1, -np.log(y_predicted+epsilon), -np.log(1-
y_predicted+epsilon)) #loss_arr.shape = (10,)

                batch_loss = np.mean(loss_arr)

                epoch_loss.append(batch_loss)


                #가중치 업데이트

                wd = (np.dot(x_batch.T, err) / batch_size) #wd.shpae = (7,1)

                self.W = optim.update(self.W, wd, lr)


            #나머지 데이터로 학습하기
```

```
                if int(x.shape[0]%batch_size) != 0:
                    start = int(x.shape[0]//batch_size)*batch_size
                    x_batch = x[start:]
                    y_batch = y[start:].reshape(-1,1)
                    y_predicted = self.forward(x_batch)
                    err = (y_predicted - y_batch)

                    #배치 손실 구하기
                    loss_arr = np.where(y_batch==1, -np.log(y_predicted+epsilon), -np.log(1-
y_predicted+epsilon))
                    batch_loss = np.mean(loss_arr)
                    epoch_loss.append(batch_loss)

                    #가중치 업데이트
                    last_batch_size = x_batch.shape[0]
                    wd = (np.dot(x_batch.T, err) / last_batch_size)
                    self.W = optim.update(self.W, wd, lr)

                #에폭 손실 구하기
                loss = np.mean(np.array(epoch_loss))

                if epoch%100==0:
                    print("cost {}, batch_size {}, epoch {}".format(loss, batch_size, epoch))
            # ========================================================

            # ========================= EDIT HERE ======================
            dot_ = np.dot(x,self.W) #dot_.shpae = (10,1)
```

```
        sigmoid = self._sigmoid(dot_) #sigmoid.shape = (10,1)
        y_predicted = np.where(sigmoid>=threshold, 1, 0)
        # ============================================================


        # ========================= EDIT HERE ========================
        sigmoid = 1/(1+np.exp(-x))
        # ============================================================
```

(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.

```
# ======================== EDIT HERE ========================
# DATA
DATA_NAME = 'Titanic'


# HYPERPARAMETERS
num_epochs = 300
# ============================================================


# ======================== EDIT HERE ========================
# Make model & optimizer
model = LogisticRegression(max_iter=num_epochs, random_state=0)


# TRAIN
model.fit(X=train_x, y=train_y)


# EVALUATION
```

```
test_x, test_y = test_data

predicted_y = model.predict(test_x)

predicted_y = predicted_y.reshape(-1,1)

ACC = metric(predicted_y, 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 (# of data: 779)	Full-batch	1000	0.1	0.815
	Mini-batch (size=10)	400	0.0001	0.84
	Scikit-Learn			0.83
Digit (# of data: 11501)	Full-batch	500	0.1	0.984
	Mini-batch (size=10)	500	0.001	0.992
	Scikit-Learn			0.99

Titanic

Full-batch

Full-batch 인 경우 learning rate 를 0.1 로 해도, 0.01 로 해도 epoch 에 따라 accuracy 가 규칙적으로 변하지 않고 불규칙적으로 변했다.

따라서 실험 결과 중 accuracy 가 제일 높게 나오면서도 시간 효율성이 제일 좋은 learning rate = 0.1, epoch = 1000 으로 했다.

Mini-batch

Mini-batch 의 경우 epoch 에 따른 accuracy 변화가 full-batch 에 비해 규칙적이였고 accuracy 도 더 높았다.

Learning rate 가 0.1 인 경우 epoch 이 3000 일 때 accuracy 가 0.843 으로 가장 높았다.

Learning rate 가 0.001 인 경우 epoch 이 400 일 때 accuracy 가 0.843 으로 가장 높았다.

따라서 실험 결과 중 accuracy 가 제일 높게 나오면서도 시간 효율성이 제일 좋은 learning rate = 0.001, epoch = 400 으로 했다.

Digit

Full-batch

full-batch 인 경우 learning rate 가 낮으면 epoch 을 증가시켜도 cost 가 줄어들지 않으면서 학습이 되지 않는 모습을 보였다.

learning rate 를 상대적으로 큰 0.1 로 두고, epoch 을 500 으로 두는 것이 가장 적절했다.

Mini-batch

Mini-batch 의 경우 learning rate 가 0.1 또는 0.01 이면 cost 가 너무 빠르게 줄어들며 epoch 을 작게 해도 accuracy 가 0.99 를 넘지 못했다.

Learning rate 가 0.0001 의 경우 cost 가 너무 느리게 줄어들며 epoch 을 크게 해도 0.99 를 넘지 못했다.

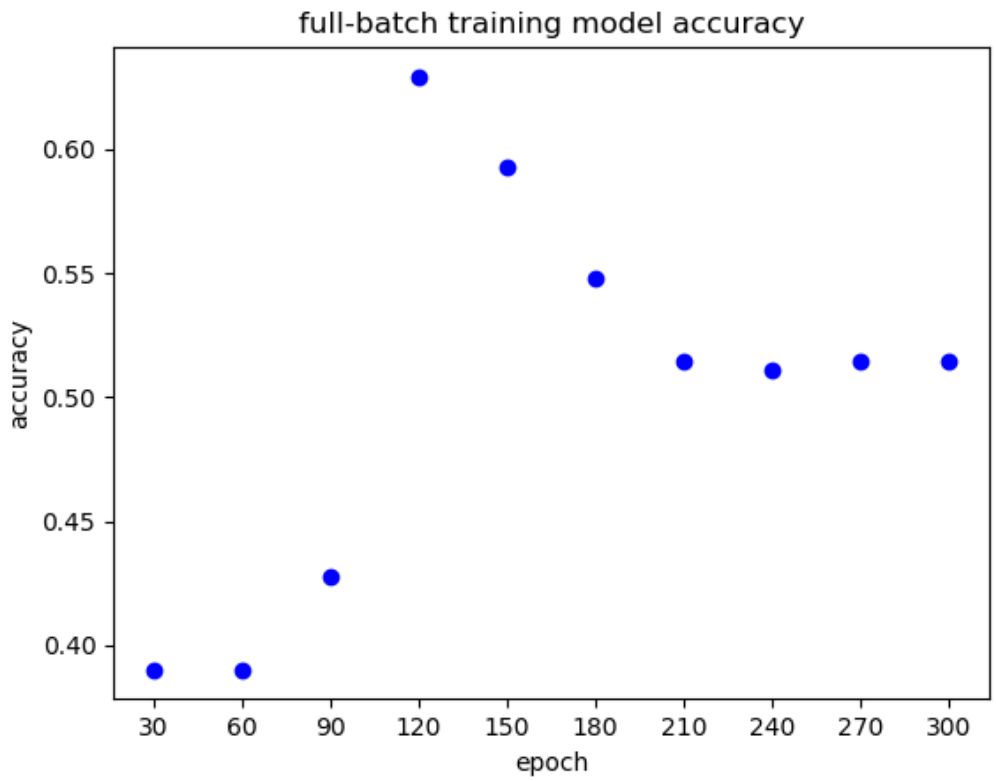
Learning rate 가 0.001 인 경우 cost 가 적당히 줄어들면서 epoch 이 500 일 때 가장 높은 accuracy 인 0.99 를 보여줬다.

(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.

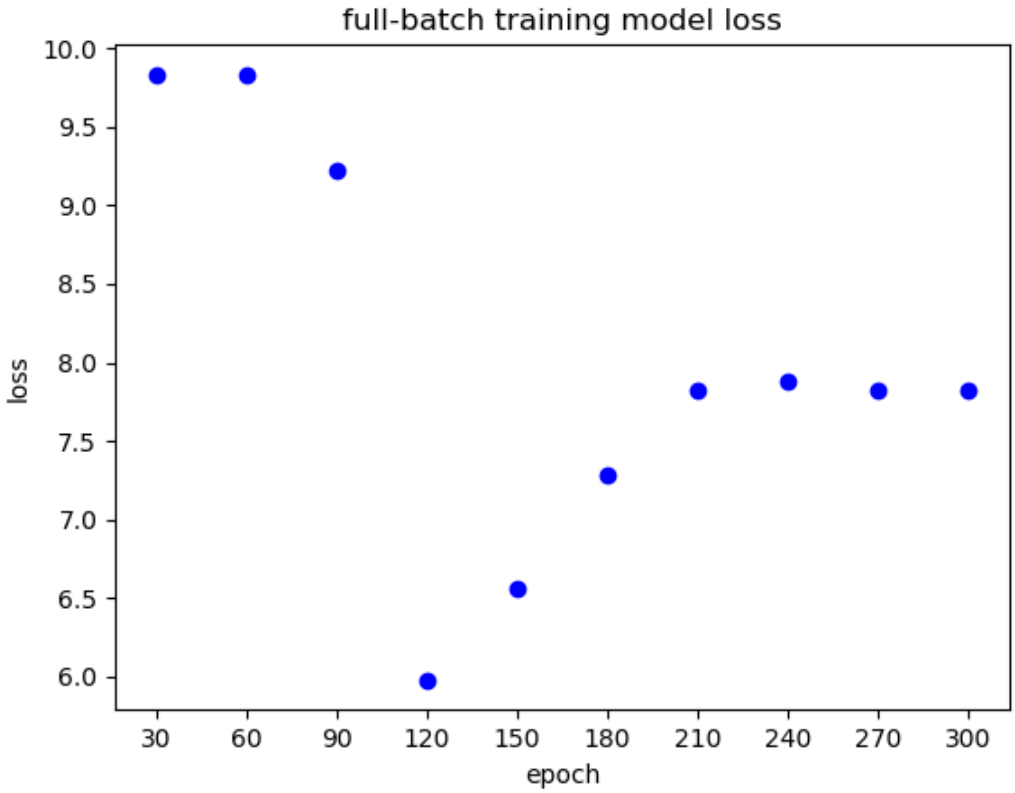
Parameter Settings	
Batch size	10
Learning rate	0.0005

Epsilon	0.01
Gamma	0.9
# of Epochs	30, 60, 90, …, 300

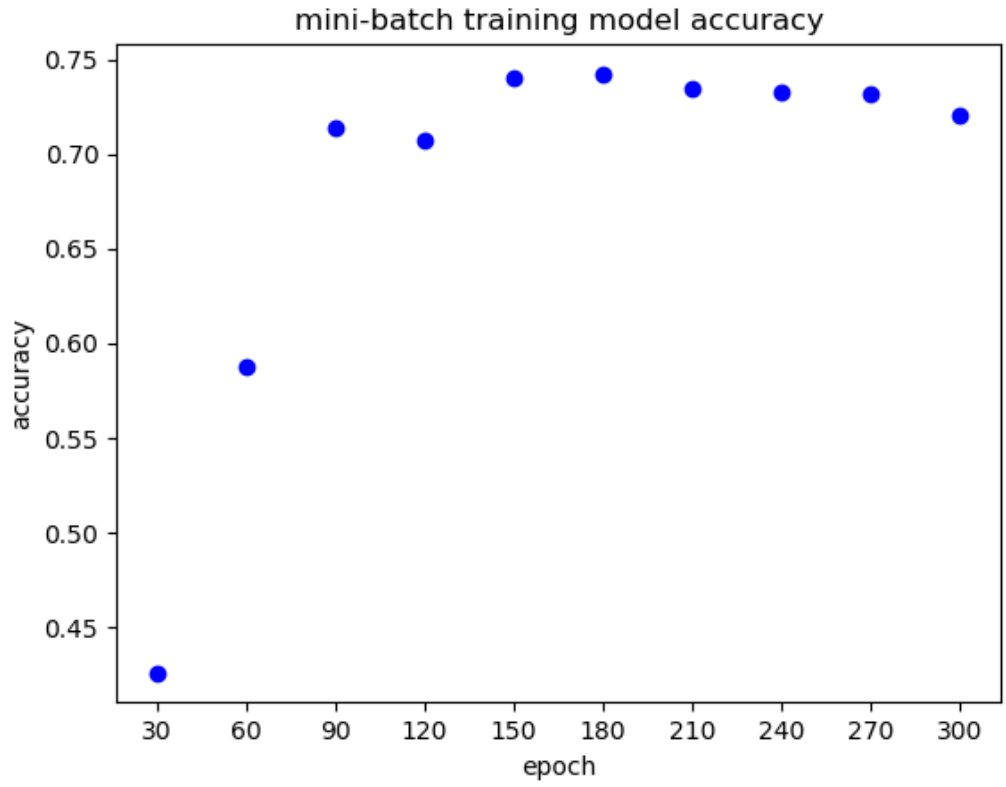
Answer: Draw the figure in the blank.



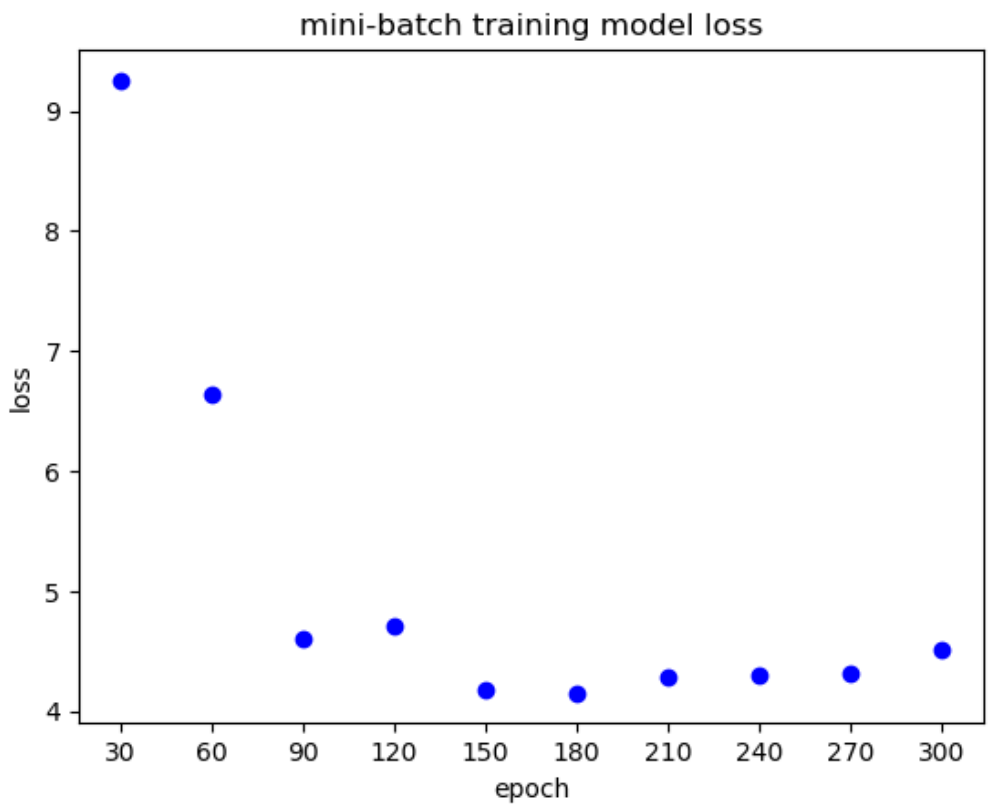
<full-batch training plot 1>



<full-batch training plot 2>



<mini-batch training plot 1>



<mini-batch training plot 2>