Introduction to Machine Learning (Spring 2019)

Homework #1 (Due date: April 8)

Student ID 2014310407

Name Lee Jun Hyuk(이준혁)

Instruction: We provide all 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.

(1) [30 pts] Implementation

(a) **[Linear reression]** Implement training and evaluation function in 'models/LinearRegression.py' ('train' and 'eval' respectively).

- **train**

```
# Shuffle the data
len_data = len(x)
perm = np.random.permutation(len_data)

x = x[perm]
y = y[perm]

#Training for epoch times
for epoch in range(epochs):
    batch_loss = 0
    index = 0

    # For each batch
    while index < len_data:
        loss = 0
        grad = []                       # Store the gradient as a list

        # For each elements in a batch
        for i in zip(x[index : min(index + batch_size, len_data)], y[index : min(index + batch_size, 
len_data)]):
            data, label = i
            loss += np.sum(np.power(np.matmul(data, self.W) - label, 2))
            grad.append(np.multiply(np.matmul(data, self.W) - label, data))

        index += batch_size

        # Randomly choose a gradient
        np.random.shuffle(grad)
        grad = grad[0]
        grad = np.asarray(grad)
        grad = grad.reshape(len(grad), 1)

        self.W = optim.update(self.W, grad, lr)
        batch_loss += loss
```

```python
# Final loss is the average of batch loss of final epoch
final_loss = batch_loss / len_data
```

- **eval**

```python
pred = np.matmul(x, self.W)
pred.shape = (pred.shape[0], 1)
```

(b) **[Logistic reression]** Implement training and evaluation function in 'models/LogisticRegression.py' ('train' and 'eval' respectively).

- **train**

```python
# Shuffle the data
len_data = len(x)
perm = np.random.permutation(len_data)

x = x[perm]
y = y[perm]

# Training for epoch times
for epoch in range(epochs):
    batch_loss = 0
    index = 0

    # For each batch
    while index < len_data:
        loss = 0
        grad = []                       # Store the gradient as a list

        # For each elements in a batch
        for i in zip(x[index: min(index + batch_size, len_data)], y[index: min(index + batch_size, 
len_data)]):
            data, label = i
            pred = self._sigmoid(np.matmul(data, self.W))

            # If loss is indeterminate
            if label == 1 - pred:
                loss -= 0

            elif label == 1:
                loss -= np.log(pred)

            else:
                loss -= np.log(1 - pred)

            grad.append(np.multiply(pred - label, data))

        index += batch_size

        # Randomly choose a gradient
        np.random.shuffle(grad)
        grad = grad[0]
```

```
        grad = np.asarray(grad)
        grad = grad.reshape(len(grad), 1)

        self.W = optim.update(self.W, grad, lr)
        batch_loss += loss

# Final loss is the average of batch loss of final epoch
final_loss = batch_loss / len_data
```

- **eval**

```
inner = np.matmul(x, self.W)
sig = self._sigmoid(inner)
pred = []

for i in sig:
    if i >= threshold:
        pred.append(1)
    else:
        pred.append(0)

pred = np.asarray(pred)
pred.shape = (pred.shape[0], 1)
```

(c) **[Optimization]** Implement SGD, Momentum, RMS Prop optimizers in 'optim/Optmizer.py'. Training should be based on the minibatch, not the whole data.

- **SGD init**

```
self.gamma = gamma
self.epsilon = epsilon
```

- **SGD update**

```
updated_weight = w - lr * grad
```

- **Momentum init**

```
self.gamma = gamma
self.epsilon = epsilon
self.velocity = []
```

- **Momentum update**

```
if len(self.velocity) == 0:
    self.velocity = lr * grad

else:
    self.velocity = self.gamma * self.velocity + lr * grad
```

```python
updated_weight = w - self.velocity
```

- **RMSProp init**

```python
self.gamma = gamma
self.epsilon = epsilon
self.G = []
```

- **RMSProp update**

```python
if len(self.G) == 0:
    self.G = np.power(grad, 2)

else:
    self.G = self.gamma * self.G + (1 - self.gamma) * np.power(grad, 2)

eps = np.asarray([self.epsilon] *len(self.G))
eps = eps.reshape(len(grad), 1)

updated_weight = np.sqrt(self.G  + eps)
updated_weight = np.divide(updated_weight, lr)
updated_weight = np.reciprocal(updated_weight)
updated_weight = np.multiply(updated_weight, grad)
updated_weight = w - updated_weight
```
-

(2) **[30 pts]** Experimental results

(a) **[Linear Regression]** For 'Graduate' and 'Concrete' dataset, adjust the number of training epochs and learning rate to minimize RMSE. Report your best results for each optimizer.
(Batch size = 10, epsilon = 0.01, gamma = 0.9)

Dataset	**Optimizer**	**# of epochs**	**Learning rate**	**MSE**
Graduate	SGD	300	0.01	0.0794
	Momentum	200	0.004	0.0792
	RMSProp	200	0.004	0.0793

Concrete	SGD	240	0.025	11.43
	Momentum	240	0.008	11.41
	RMSProp	240	0.05	11.79

(b) **[Logistic Regression]** For 'Titanic' and 'Digit' dataset, adjust the number of training epochs and learning rate to maximize accuracy. Report your best results for each optimizer.
(Batch size = 10, epsilon = 0.01, gamma = 0.9)

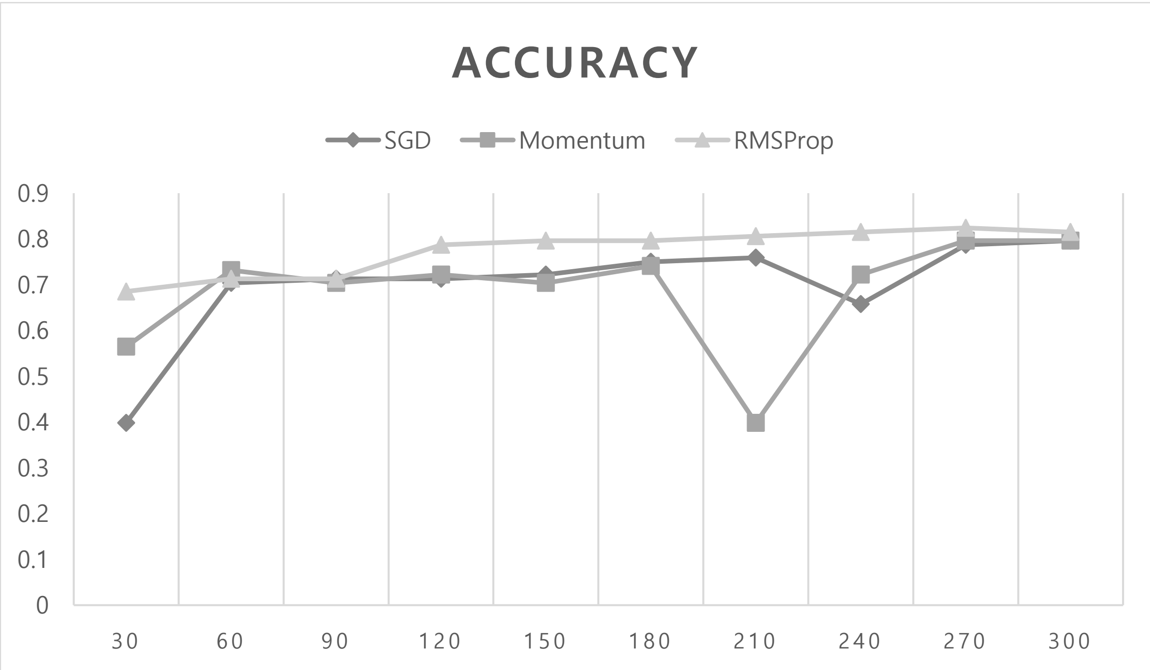
Answer: Fill the blank in the table.

Dataset	**Optimizer**	**# of epochs**	**Learning rate**	**Acc.**
Titanic	SGD	300	0.0005	0.796
	Momentum	300	0.0003	0.806
	RMSprop	300	0.0005	0.815
Digit	SGD	60	0.005	0.994
	Momentum	60	0.001	0.994
	RMSprop	60	0.001	0.994

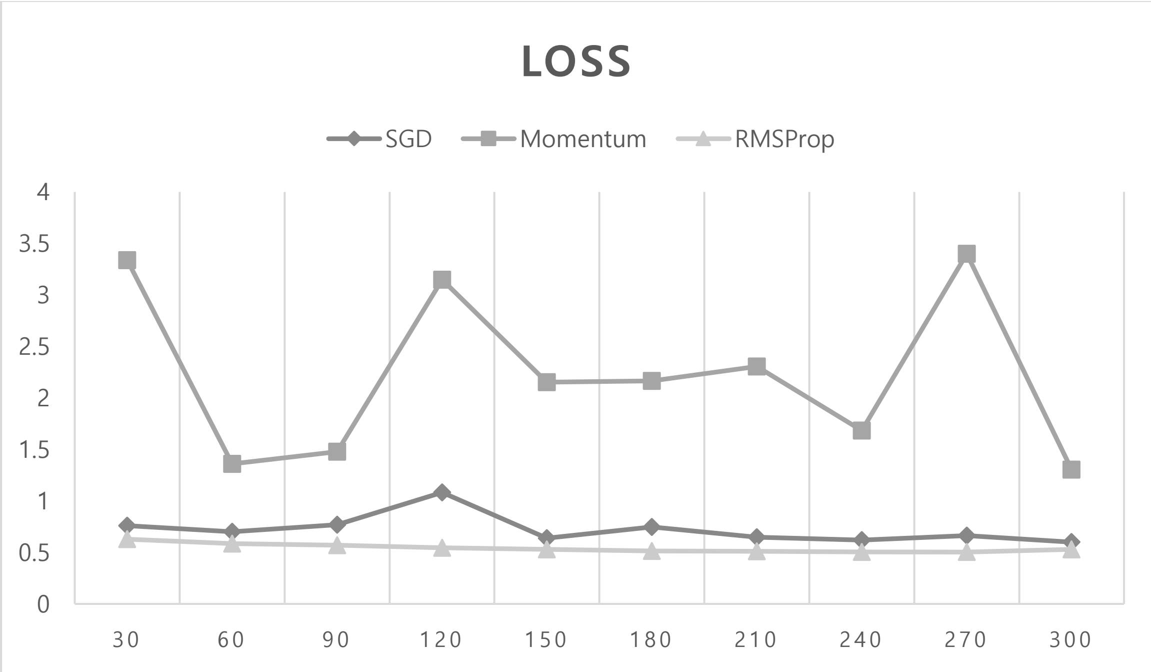
(c) **[Optimization]** For 'Titanic' dataset, execute the logistic regression with three optimization methods. Given the following parameter settings, draw two plots : a plot whose x-axis and y-axis are epochs and accuracy, and a plot whose x-axis and y-axis are epochs and cross-entropy loss. Explain which optimization method shows the best accuracy.

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

Answer: draw the plot and explain the result, especially about the correlation with loss and accuracy according to different optimization methods.



RMSProp가 가장 좋은 성능을 보이고 있음을 확인할 수 있다. SGD는 RMSProp보다 안 좋은데, 특히 epoch 이 작을 때 성능이 안 좋은 것을 알 수 있다. 이는 SGD의 느린 수렴성을 설명한다. Momentum은 SGD보다는 좋아 보이나, epoch이 210일 때 성능이 급감할을 보인다. 이는 Momentum이 특정 상황에서 안정적이지 않다는 것처럼 보인다.



역시 loss 측면에서도 RMSProp가 가장 좋은 성능을 보이고 있음을 확인할 수 있다. SGD는 RMSProp보다 안 좋지만, 큰 차이를 보이지 않는다. 눈에 띄는 점은 Momentum인데, loss가 잘 줄어들지 않고 진동하는 것을 확인할 수 있다. Learning rate가 momentum 방식에 대해서는 크다고 생각해 볼 수 있다. 실제로 hyperparameter tuning을 해 본 결과, learning rate를 0.0003정도로 더 줄였을 때 더 학습이 잘 됨을 확인했다.