Assignment 2: Stochastic Gradient Descent and Momentum

CPSC 381/581: Machine Learning

Yale University

Instructor: Alex Wong

Prerequisites:

1. Enable Google Colaboratory as an app on your Google Drive account
2. Create a new Google Colab notebook, this will also create a "Colab Notebooks" directory under "MyDrive" i.e.

   ```
   /content/drive/MyDrive/Colab Notebooks
   ```

3. Create the following directory structure in your Google Drive

   ```
   /content/drive/MyDrive/Colab Notebooks/CPSC 381-581: Machine
   Learning/Assignments
   ```

4. Move the 02_assignment_stochastic_gradient_descent.ipynb into

   ```
   /content/drive/MyDrive/Colab Notebooks/CPSC 381-581: Machine
   Learning/Assignments
   ```

so that its absolute path is

```
/content/drive/MyDrive/Colab Notebooks/CPSC 381-581: Machine
Learning/Assignments/02_assignment_stochastic_gradient_descent.ipynb
```

In this assignment, we will optimize a linear function for the logistic regression task using the stochastic gradient descent and its momentum variant. We will test them on several binary classification datasets (breast cancer, digits larger or less than 5, and fir and pine coverage). We will implement a training and validation loop for the binary classification task and test it on the testing split for each dataset.

Submission:

1. Implement all TODOs in the code blocks below.
2. Report your training, validation, and testing scores.

   ```
   Report training, validation, and testing scores here.
   ```

3. List any collaborators.

   ```
   Collaborators: Doe, Jane (Please write names in <Last Name, First
   Name> format)

   Collaboration details: Discussed ... implementation details with Jane
   Doe.
   ```

IMPORTANT:

- For full credit, your mean classification accuracies for all trained models across all datasets should be no more than 8% worse relative to the scores achieved by sci-kit learn's logistic regression model across training, validation and testing splits.
- You may not use batch sizes of more than 10% of the training dataset size for stochastic gradient descent and momentum stochastic gradient descent.
- You will only need to experiment with gradient descent (GD) and momentum gradient descent (momentum GD) on breast cancer and digits (toy) datasets. It will take too long to run them on fir and pine coverage (realistic) dataset to get reasonable numbers. Of course, you may try them on fir and pine coverage :) but they will not count towards your grade.
- Note the run time speed up when comparing GD and momemtum GD with stochastic gradient descent (SGD) and momentum stochastic gradient descent (momentum SGD)! Even though they are faster and observing batches instead of the full dataset at each time step, they can still achieving similar accuracies!

In [ ]:
```
import numpy as np
import sklearn.datasets as skdata
import sklearn.metrics as skmetrics
from sklearn.linear_model import LogisticRegression as LogisticRegressionSciKit
import warnings
import time

warnings.filterwarnings(action='ignore')
np.random.seed = 1
```

Implementation of stochastic gradient descent optimizer for logistic loss

In [ ]:
```
class Optimizer(object):

    def __init__(self, alpha, eta_decay_factor, beta, optimizer_type):
        '''
        Arg(s):
            alpha : float
                initial learning rate
            eta_decay_factor : float
                learning rate decay rate
            beta : float
                momentum discount rate
            optimizer_type : str
                'gradient_descent',
                'momentum_gradient_descent',
                'stochastic_gradient_descent',
                'momentum_stochastic_gradient_descent'
        '''

        self.__alpha = alpha
        self.__eta_decay_factor = eta_decay_factor
        self.__beta = beta
        self.__optimizer_type = optimizer_type
        self.__momentum = None

    def __compute_gradients(self, w, x, y, loss_func='logistic'):
        '''
        Returns the gradient of a loss function

        Arg(s):
            w : numpy[float32]
                d x 1 weight vector
            x : numpy[float32]
```

```
            d x N feature vector
        y : numpy[float32]
            1 x N groundtruth vector
        loss_func : str
            loss by default is 'logistic' only for the purpose of the assignment
    Returns:
        numpy[float32] : d x 1 gradients
    '''

    # TODO: Implement compute_gradient function

    if loss_func == 'logistic':
        N = x.shape[1]
        z = y * (np.dot(w.T,x))
        denom = 1 + np.exp(z)
        num = y * x
        grad = -np.sum(num / denom, axis=1) / N
        return grad.reshape(-1, 1)

    else:
        raise ValueError('Unupported loss function: {}'.format(loss_func))

def __polynomial_decay(self, time_step):
    '''
    Computes the polynomial decay factor t^{-a}

    Arg(s):
        time_step : int
            current step in optimization
    Returns:
        float : polynomial decay to adjust (reduce) initial learning rate
    '''
    if self.__eta_decay_factor is None:
        return 1.0
    if self.__eta_decay_factor <= 0:
        return 1.0
    if time_step <= 0:
        raise ValueError('Time step must be positive')
    return time_step ** (-self.__eta_decay_factor)

def update(self,
           w,
           x,
           y,
           loss_func,
           batch_size,
           time_step):
    '''
    Updates the weight vector based on

    Arg(s):
        w : numpy[float32]
            d x 1 weight vector
        x : numpy[float32]
            d x N feature vector
        y : numpy[float32]
            1 x N groundtruth vector
        loss_func : str
            loss function to use, should be 'logistic' for the purpose of the ass
        batch_size : int
            batch size for stochastic and momentum stochastic gradient descent
        time_step : int
            current step in optimization
    Returns:
        numpy[float32]: d x 1 weights
    '''
```

```
        lr = self.__alpha * self.__polynomial_decay(time_step)

        # TODO: Implement the optimizer update function
        # For each optimizer type, compute gradients and update weights
        grad = self.__compute_gradients(w, x, y, loss_func)
        if self.__optimizer_type == 'gradient_descent':
            # itterate
            return w - lr * grad

        elif self.__optimizer_type == 'momentum_gradient_descent':
            # update momentum
            if self.__momentum is None:
                self.__momentum = np.zeros_like(w)
            self.__momentum = self.__beta * self.__momentum + (1 - self.__beta) * gra
            return w - lr * self.__momentum

        elif self.__optimizer_type == 'stochastic_gradient_descent':
            N = x.shape[1]
            # shuffle data
            indices = np.random.permutation(N)
            x_shuffled = x.T[indices]
            y_shuffled = y.T[indices]
            x_batch = x_shuffled[:batch_size].T
            y_batch = y_shuffled[:batch_size].T
            grad = self.__compute_gradients(w, x_batch, y_batch, loss_func)
            return w - lr * grad

        elif self.__optimizer_type == 'momentum_stochastic_gradient_descent':
            N = x.shape[1]
            # shuffle data
            indices = np.random.permutation(N)
            x_shuffled = x.T[indices]
            y_shuffled = y.T[indices]
            x_batch = x_shuffled[:batch_size].T
            y_batch = y_shuffled[:batch_size].T
            grad = self.__compute_gradients(w, x_batch, y_batch, loss_func)
            if self.__momentum is None:
                self.__momentum = np.zeros_like(w)
            self.__momentum = self.__beta * self.__momentum + (1 - self.__beta) * gra
            return w - lr * self.__momentum

        else:
            raise ValueError('Unsupported optimizer type: {}'.format(self.__optimizer
```

Implementation of our logistic regression model for binary classification

In [ ]:

```
class LogisticRegression(object):

    def __init__(self):
        # Define private variables
        self.__weights = None
        self.__optimizer = None

    def fit(self,
            x,
            y,
            T,
            alpha,
            eta_decay_factor,
            beta,
            batch_size,
            optimizer_type,
            loss_func='logistic'):
        '''
        Fits the model to x and y by updating the weight vector
```

```
        using gradient descent

        Arg(s):
            x : numpy[float32]
                d x N feature vector
            y : numpy[float32]
                1 x N groundtruth vector
            T : int
                number of iterations to train
            alpha : float
                learning rate
            eta_decay_factor : float
                learning rate decay rate
            beta : float
                momentum discount rate
            batch_size : int
                number of examples per batch
            optimizer_type : str
                'gradient_descent',
                'momentum_gradient_descent',
                'stochastic_gradient_descent',
                'momentum_stochastic_gradient_descent'
            loss_func : str
                loss function to use, by default is 'logistic' only for the purpose o
        '''

        # TODO: Instantiate optimizer and weights
        self.__optimizer = Optimizer(alpha, eta_decay_factor, beta, optimizer_type)

        self.__weights = np.zeros((x.shape[0], 1), dtype=np.float32)

        for t in range(1, T + 1):

            # TODO: Compute loss function
            loss = self.__compute_loss(x, y, loss_func)
            if (t % 10000) == 0:
                print('Step={}  Loss={}'.format(t, loss))

            # TODO: Update weights
            self.__weights = self.__optimizer.update(self.__weights, x, y, loss_func,


    def predict(self, x):
        '''
        Predicts the label for each feature vector x

        Arg(s):
            x : numpy[float32]
                d x N feature vector
        Returns:
            numpy[float32] : 1 x N vector
        '''

        # TODO: Implements the predict function
        z = np.dot(self.__weights.T, x)
        sig = 1 / (1 + np.exp(-z))
        return np.where(sig >= 0.5, 1, -1)


    def __compute_loss(self, x, y, loss_func):
        '''
        Computes the logistic loss

        Arg(s):
            x : numpy[float32]
                d x N feature vector
```

```
            y : numpy[float32]
                1 x N groundtruth vector
            loss_func : str
                loss function to use, by default is 'logistic' only for the purpose o
        Returns:
            float : loss
        '''

        # TODO: Implements the __compute_loss function

        if loss_func == 'logistic':
            z = np.dot(self.__weights.T, x)
            loss = np.mean(np.log(1 + np.exp(-y * z)))

        else:
            raise ValueError('Unsupported loss function: {}'.format(loss_func))

        return loss
```

Training, validating and testing logistic regression for binary classification

In [ ]:

```
# Load breast cancer, digits, and tree coverage datasets
datasets = [
    skdata.load_breast_cancer(),
    skdata.load_digits(),
    skdata.fetch_covtype()
]
dataset_names = [
    'breast cancer',
    'digits greater or less than 5',
    'fir and pine coverage',
]

# Loss functions to minimize
dataset_optimizer_types = [
    # For breast cancer dataset
    [
        'gradient_descent',
        'momentum_gradient_descent',
        'stochastic_gradient_descent',
        'momentum_stochastic_gradient_descent'
    # For digits greater than or less than 5 dataset
    ], [
        'gradient_descent',
        'momentum_gradient_descent',
        'stochastic_gradient_descent',
        'momentum_stochastic_gradient_descent'
    # For fir and pine coverage dataset
    ], [
        'stochastic_gradient_descent',
        'momentum_stochastic_gradient_descent'
    ]
]

# TODO: Select hyperparameters
dataset_alphas = [
    # For breast cancer dataset
    [1e-4, 1e-4, 1e-4, 1e-4],
    # For digits greater than or less than 5 dataset
    [5e-3, 5e-3, 1e-2, 1e-2],
    # For fir and pine coverage dataset
    [1e-3, 1e-3]
]

dataset_eta_decay_factors = [
```

```
    # For breast cancer dataset
    [None, None, 0.5, 0.5],
    # For digits greater than or less than 5 dataset
    [None, None, 0.5, 0.5],
    # For fir and pine coverage dataset
    [0.95, 0.95]
]

dataset_betas = [
    # For breast cancer dataset
    [None, 0.9, None, 0.9],
    # For digits greater than or less than 5 dataset
    [None, 0.9, None, 0.9],
    # For fir and pine coverage dataset
    [None, 0.95]
]

cancer_batch = int(datasets[0].data.shape[0] * 0.6 * 0.1)
digits_batch = int(datasets[1].data.shape[0] * 0.6 * 0.1)
tree_batch = int(datasets[2].data.shape[0] * 0.6 * 0.005)
dataset_batch_sizes = [
    # For breast cancer dataset
    [None, None, cancer_batch, cancer_batch],
    # For digits greater than or less than 5 dataset
    [None, None, digits_batch, digits_batch],
    # For fir and pine coverage dataset
    [tree_batch, tree_batch]
]

dataset_Ts = [
    # For breast cancer dataset
    [10000, 10000, 200000, 200000],
    # For digits greater than or less than 5 dataset
    [20000, 20000, 30000, 30000],
    # For fir and pine coverage dataset
    [5000, 5000]
]

# Zip up all dataset options
dataset_options = zip(
    datasets,
    dataset_names,
    dataset_optimizer_types,
    dataset_alphas,
    dataset_eta_decay_factors,
    dataset_betas,
    dataset_batch_sizes,
    dataset_Ts)


for options in dataset_options:

    # Unpack dataset options
    dataset, \
        dataset_name, \
        optimizer_types, \
        alphas, \
        eta_decay_factors, \
        betas, \
        batch_sizes, \
        Ts = options

    '''
    Create the training, validation and testing splits
    '''
    x = dataset.data
```

```
y = dataset.target

if dataset_name == 'digits greater or less than 5':
    y[y < 5] = 1
    y[y >= 5] = 0
elif dataset_name == 'fir and pine coverage':

    idx_fir_or_pine = np.where(np.logical_or(y == 1, y == 2))[0]

    x = x[idx_fir_or_pine, :]
    y = y[idx_fir_or_pine]

    # Pine class: 0; Fir class: 1
    y[y == 2] = 0

print('Preprocessing the {} dataset ({} samples, {} feature dimensions)'.format(d

# Shuffle the dataset based on sample indices
shuffled_indices = np.random.permutation(x.shape[0])

# Choose the first 60% as training set, next 20% as validation and the rest as te
train_split_idx = int(0.60 * x.shape[0])
val_split_idx = int(0.80 * x.shape[0])

train_indices = shuffled_indices[0:train_split_idx]
val_indices = shuffled_indices[train_split_idx:val_split_idx]
test_indices = shuffled_indices[val_split_idx:]

# Select the examples from x and y to construct our training, validation, testing
x_train, y_train = x[train_indices, :], y[train_indices]
x_val, y_val = x[val_indices, :], y[val_indices]
x_test, y_test = x[test_indices, :], y[test_indices]

'''
Trains and tests logistic regression model from scikit-learn
'''
model_scikit = LogisticRegressionSciKit(penalty=None, fit_intercept=False)

# TODO: Train scikit-learn logistic regression model
model_scikit.fit(x_train, y_train)

print('***** Results on the {} dataset using scikit-learn logistic regression mod

# TODO: Score model using mean accuracy on training set
predictions_train = model_scikit.predict(x_train)
score_train = skmetrics.accuracy_score(y_train, predictions_train)
print('Training set mean accuracy: {:.4f}'.format(score_train))

# TODO: Score model using mean accuracy on validation set
predictions_val = model_scikit.predict(x_val)
score_val = skmetrics.accuracy_score(y_val, predictions_val)
print('Validation set mean accuracy: {:.4f}'.format(score_val))

# TODO: Score model using mean accuracy on testing set
predictions_test = model_scikit.predict(x_test)
score_test = skmetrics.accuracy_score(y_test, predictions_test)
print('Testing set mean accuracy: {:.4f}'.format(score_test))

'''
Trains, validates, and tests our logistic regression model for binary classificat
'''
# Take the transpose of the dataset to match the dimensions discussed in lecture
# i.e., (N x d) to (d x N)
x_train = np.transpose(x_train, axes=(1, 0))
x_val = np.transpose(x_val, axes=(1, 0))
x_test = np.transpose(x_test, axes=(1, 0))
```

```
    y_train = np.expand_dims(y_train, axis=0)
    y_val = np.expand_dims(y_val, axis=0)
    y_test = np.expand_dims(y_test, axis=0)

    # TODO: Set the ground truth to the appropriate classes (integers) according to l
    # convert 0's to -1's
    y_train = np.where(y_train == 0, -1, y_train)
    y_val = np.where(y_val == 0, -1, y_val)
    y_test = np.where(y_test == 0, -1, y_test)

    model_options = zip(optimizer_types, alphas, eta_decay_factors, betas, batch_size

    for optimizer_type, alpha, eta_decay_factor, beta, batch_size, T in model_options
        # skip gradient descent and momentum gradient descent for fir and pine covera
        if dataset_name == 'fir and pine coverage' and optimizer_type in ['gradient_d
            continue
        # TODO: Initialize our logistic regression model
        model_ours = LogisticRegression()

        print('***** Results of our logistic regression model trained on {} dataset *
        print('\t optimizer_type={} \n\t alpha={} \n\t eta_decay_factor={} \n\t beta=
            optimizer_type, alpha, eta_decay_factor, beta, batch_size, T))

        time_start = time.time()

        # TODO: Train model on training set
        model_ours.fit(x_train, y_train, T, alpha, eta_decay_factor, beta, batch_size

        time_elapsed = time.time() - time_start
        print('Total training time: {:3f} seconds'.format(time_elapsed))

        # TODO: Score model using mean accuracy on training set
        predictions_train = model_ours.predict(x_train)
        score_train = skmetrics.accuracy_score(y_train.flatten(), predictions_train.f
        print('Training set mean accuracy: {:.4f}'.format(score_train))

        # TODO: Score model using mean accuracy on validation set
        predictions_val = model_ours.predict(x_val)
        score_val = skmetrics.accuracy_score(y_val.flatten(), predictions_val.flatten
        print('Validation set mean accuracy: {:.4f}'.format(score_val))

        # TODO: Score model using mean accuracy on testing set
        predictions_test = model_ours.predict(x_test)
        score_test = skmetrics.accuracy_score(y_test.flatten(), predictions_test.flat
        print('Testing set mean accuracy: {:.4f}'.format(score_test))

    print('')
```

```
Preprocessing the breast cancer dataset (569 samples, 30 feature dimensions)
***** Results on the breast cancer dataset using scikit-learn logistic regression mode
l *****
Training set mean accuracy: 0.9355
Validation set mean accuracy: 0.9211
Testing set mean accuracy: 0.9298
***** Results of our logistic regression model trained on breast cancer dataset *****
        optimizer_type=gradient_descent
        alpha=0.0001
        eta_decay_factor=None
        beta=None
        batch_size=None
        T=10000
Step=10000  Loss=0.2903085142409005
Total training time: 0.335429 seconds
Training set mean accuracy: 0.9208
Validation set mean accuracy: 0.9298
Testing set mean accuracy: 0.9035
***** Results of our logistic regression model trained on breast cancer dataset *****
        optimizer_type=momentum_gradient_descent
        alpha=0.0001
        eta_decay_factor=None
        beta=0.9
        batch_size=None
        T=10000
Step=10000  Loss=0.17278983605962947
Total training time: 0.432598 seconds
Training set mean accuracy: 0.9238
Validation set mean accuracy: 0.9386
Testing set mean accuracy: 0.9035
***** Results of our logistic regression model trained on breast cancer dataset *****
        optimizer_type=stochastic_gradient_descent
        alpha=0.0001
        eta_decay_factor=0.5
        beta=None
        batch_size=34
        T=200000
Step=10000  Loss=0.22985778842168397
Step=20000  Loss=0.21970266949342426
Step=30000  Loss=0.2149852310350894
Step=40000  Loss=0.21291256691550983
Step=50000  Loss=0.2104380884088789
Step=60000  Loss=0.20909242008499165
Step=70000  Loss=0.2078529830929266
Step=80000  Loss=0.20692956163007586
Step=90000  Loss=0.20639198308356183
Step=100000  Loss=0.20575036467596977
Step=110000  Loss=0.20498062738607115
Step=120000  Loss=0.20441586196438125
Step=130000  Loss=0.2040471111532782
Step=140000  Loss=0.20368842748325203
Step=150000  Loss=0.20329260182301903
Step=160000  Loss=0.2027611263460274
Step=170000  Loss=0.2024368610439995
Step=180000  Loss=0.20211503096782915
Step=190000  Loss=0.2018237293075803
Step=200000  Loss=0.2018742238815235
Total training time: 11.478329 seconds
Training set mean accuracy: 0.9179
Validation set mean accuracy: 0.9298
Testing set mean accuracy: 0.9035
***** Results of our logistic regression model trained on breast cancer dataset *****
        optimizer_type=momentum_stochastic_gradient_descent
        alpha=0.0001
        eta_decay_factor=0.5
        beta=0.9
```

```
        batch_size=34
        T=200000
Step=10000  Loss=0.2316590852262425
Step=20000  Loss=0.22210054271895394
Step=30000  Loss=0.2154737449312446
Step=40000  Loss=0.21262425271792776
Step=50000  Loss=0.21069140755947424
Step=60000  Loss=0.20928839292245793
Step=70000  Loss=0.2080740752623165
Step=80000  Loss=0.20732062854591785
Step=90000  Loss=0.20630164459250916
Step=100000  Loss=0.20559591798263913
Step=110000  Loss=0.20504384661485878
Step=120000  Loss=0.20447110138894187
Step=130000  Loss=0.20421179019210978
Step=140000  Loss=0.20381593881350601
Step=150000  Loss=0.2034087243301398
Step=160000  Loss=0.20280588497403093
Step=170000  Loss=0.20247183844488323
Step=180000  Loss=0.20224258177479543
Step=190000  Loss=0.20187267948699736
Step=200000  Loss=0.20159556646471627
Total training time: 11.532451 seconds
Training set mean accuracy: 0.9208
Validation set mean accuracy: 0.9298
Testing set mean accuracy: 0.9035

Preprocessing the digits greater or less than 5 dataset (1797 samples, 64 feature dime
nsions)
***** Results on the digits greater or less than 5 dataset using scikit-learn logistic
regression model *****
Training set mean accuracy: 0.9035
Validation set mean accuracy: 0.8663
Testing set mean accuracy: 0.9000
***** Results of our logistic regression model trained on digits greater or less than
5 dataset *****
        optimizer_type=gradient_descent
        alpha=0.005
        eta_decay_factor=None
        beta=None
        batch_size=None
        T=20000
Step=10000  Loss=0.244387531417709
Step=20000  Loss=0.24277320961101692
Total training time: 2.710289 seconds
Training set mean accuracy: 0.8998
Validation set mean accuracy: 0.8691
Testing set mean accuracy: 0.9028
***** Results of our logistic regression model trained on digits greater or less than
5 dataset *****
        optimizer_type=momentum_gradient_descent
        alpha=0.005
        eta_decay_factor=None
        beta=0.9
        batch_size=None
        T=20000
Step=10000  Loss=0.2443871966884419
Step=20000  Loss=0.242772922813918
Total training time: 2.714349 seconds
Training set mean accuracy: 0.8998
Validation set mean accuracy: 0.8691
Testing set mean accuracy: 0.9028
***** Results of our logistic regression model trained on digits greater or less than
5 dataset *****
        optimizer_type=stochastic_gradient_descent
        alpha=0.01
```

```
        eta_decay_factor=0.5
        beta=None
        batch_size=107
        T=30000
Step=10000  Loss=0.261813193599257
Step=20000  Loss=0.2564011237287529
Step=30000  Loss=0.2541253538344087
Total training time: 5.674910 seconds
Training set mean accuracy: 0.8998
Validation set mean accuracy: 0.8747
Testing set mean accuracy: 0.9167
***** Results of our logistic regression model trained on digits greater or less than
5 dataset *****
        optimizer_type=momentum_stochastic_gradient_descent
        alpha=0.01
        eta_decay_factor=0.5
        beta=0.9
        batch_size=107
        T=30000
Step=10000  Loss=0.26232996825333815
Step=20000  Loss=0.25661812806594037
Step=30000  Loss=0.25429252355498544
Total training time: 6.029009 seconds
Training set mean accuracy: 0.9007
Validation set mean accuracy: 0.8774
Testing set mean accuracy: 0.9139

Preprocessing the fir and pine coverage dataset (495141 samples, 54 feature dimension
s)
***** Results on the fir and pine coverage dataset using scikit-learn logistic regress
ion model *****
Training set mean accuracy: 0.7432
Validation set mean accuracy: 0.7447
Testing set mean accuracy: 0.7433
***** Results of our logistic regression model trained on fir and pine coverage datase
t *****
        optimizer_type=stochastic_gradient_descent
        alpha=0.001
        eta_decay_factor=0.95
        beta=None
        batch_size=1743
        T=5000
Total training time: 455.674244 seconds
Training set mean accuracy: 0.7169
Validation set mean accuracy: 0.7179
Testing set mean accuracy: 0.7182
***** Results of our logistic regression model trained on fir and pine coverage datase
t *****
        optimizer_type=momentum_stochastic_gradient_descent
        alpha=0.001
        eta_decay_factor=0.95
        beta=0.95
        batch_size=1743
        T=5000
Total training time: 449.982237 seconds
Training set mean accuracy: 0.7126
Validation set mean accuracy: 0.7149
Testing set mean accuracy: 0.7143
```