02_assignment_stochastic_gradient_descent_robertson

March 23, 2025

Assignment 2: Stochastic Gradient Descent and Momentum

CPSC 381/581: Machine Learning

Yale University

Instructor: Alex Wong

Student: Hailey Robertson

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_assignmen
```

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: N/A
```

IMPORTANT:

- For full credit, your mean classification accuracies for all trained models across all datasets should be no more than 8% worse 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 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!

[11]:
```
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

[12]:
```
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
```

```python
        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
        '''

        # DONE: Implement compute_gradient function

        if loss_func == 'logistic':
            # print('w shape: {}'.format(w.shape)) # (1, 30)
            N = x.shape[1] # number of samples, 341
            z = np.dot(w.T, x)
            # print('y shape: {}'.format(y.shape)) # (1, 341)
            # print('z shape: {}'.format(z.shape)) # (1, 341)
            sigmoid = 1 / (1 + np.exp(-y * z))
            gradient = -np.dot(x, ((y * (1 - sigmoid))).T) / N

            # print('gradient shape: {}'.format(gradient.shape))

            return gradient

        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
        '''

        # DONE: Implement polynomial decay to adjust the initial learning rate
        if self.__eta_decay_factor is None:
            return self.__alpha
        else:
            decay_factor = time_step ** (-self.__eta_decay_factor)
            decayed_alpha = self.__alpha * decay_factor

        return decayed_alpha

    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 assignment
            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
        '''

        # DONE: Implement the optimizer update function
        # For each optimizer type, compute gradients and update weights
        eta = self.__polynomial_decay(time_step) # step size
        gradient = self.__compute_gradients(w, x, y, loss_func)
```

```
        if self.__optimizer_type == 'gradient_descent':
            # print('gradient shape: {}'.format(gradient.shape)) # (30, 1)
            # print('w shape: {}'.format(w.shape)) # (30, 1)
            # print("Before update: w[0]:", w[0])
            w = w - eta * gradient
            # print("After update: w[0]:", w[0])
            return w

        elif self.__optimizer_type == 'momentum_gradient_descent':
            if self.__momentum is None:
                self.__momentum = np.zeros(w.shape)
            self.__momentum = self.__beta * self.__momentum + (1-self.__beta) *␣
↪gradient
            w = w - eta * self.__momentum
            return w

        elif self.__optimizer_type == 'stochastic_gradient_descent':
            indices = np.random.choice(x.shape[1], batch_size, replace=False)
            x_batch = x[:, indices]
            y_batch = y[:, indices]
            gradient = self.__compute_gradients(w, x_batch, y_batch, loss_func)
            w = w - eta * gradient
            return w

        elif self.__optimizer_type == 'momentum_stochastic_gradient_descent':
            if self.__momentum is None:
                self.__momentum = np.zeros(w.shape)
            indices = np.random.choice(x.shape[1], batch_size, replace=False)
            x_batch = x[:, indices]
            y_batch = y[:, indices]
            gradient = self.__compute_gradients(w, x_batch, y_batch, loss_func)
            self.__momentum = self.__beta * self.__momentum + (1-self.__beta) *␣
↪gradient
            w = w - eta * self.__momentum
            return w

        else:
            raise ValueError('Unsupported optimizer type: {}'.format(self.
↪__optimizer_type))
```

Implementation of our logistic regression model for binary classification

[13]:
```
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 of the assignment
        '''

        # DONE: Instantiate optimizer and weights
        self.__optimizer = Optimizer(alpha, eta_decay_factor, beta,␣
↪optimizer_type)
        self.__weights = np.random.randn(x.shape[0], 1) * 0.01
```

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

            # DONE: Compute loss function
            loss = self.__compute_loss(x, y, loss_func)


            if (t % 1000) == 0:
                print('Step={}  Loss={}'.format(t, loss))

            # DONE: Update weights
            self.__weights = self.__optimizer.update(self.__weights, x, y,␣
↪loss_func, batch_size, t)
            # print("loss", loss, "weights", self.__weights)


    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
        '''

        # DONE: Implements the predict function
        # Hint: logistic regression predicts a value between 0 and 1
        z = np.dot(self.__weights.T, x)
        sigmoid = 1 / (1+ np.exp(-z))
        # print(sigmoid.shape)
        predictions = np.where(sigmoid >= 0.5, 1, -1)
        # print(predictions)

        return predictions

    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 of the assignment
        Returns:
            float : loss
        '''

        # DONE: Implements the __compute_loss function
        if loss_func == 'logistic':

            N = x.shape[1] # number of samples, 341
            z = np.dot(self.__weights.T, x)
            loss = np.sum(np.log(1 + np.exp(-y * z))) / N

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

        return loss
```

Training, validating and testing logistic regression for binary classification

[14]:
```
def compare_scores(score_train, score_val, score_test, sk_score_train,␣
↪sk_score_val, sk_score_test):
    """
    Compare the performance of our logistic regression model with scikit-learn␣
↪model.
    If the performance is more than 8% worse for any dataset (train/validation/
↪test),
    print 'Rerun' because I don't want to assess this by hand.
    """
    threshold = 0.08

    if score_train < sk_score_train * (1 - threshold):
        print(f"\nTraining score is more than 8% worse. Ours: {score_train:.
↪4f}, Scikit: {sk_score_train:.4f}. Rerun.")

    if score_val < sk_score_val * (1 - threshold):
        print(f"\nValidation score is more than 8% worse. Ours: {score_val:.
↪4f}, Scikit: {sk_score_val:.4f}. Rerun.")

    if score_test < sk_score_test * (1 - threshold):
        print(f"\nTest score is more than 8% worse. Ours: {score_test:.4f},␣
↪Scikit: {sk_score_test:.4f}. Rerun.")

    if (score_train >= sk_score_train * (1 - threshold)) and \
       (score_val >= sk_score_val * (1 - threshold)) and \
       (score_test >= sk_score_test * (1 - threshold)):
        print("\nPerformance is acceptable.\n")
```

[15]:

```
# 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'
    ]
]

# DONE: Select hyperparameters

# Step size (always used)
dataset_alphas = [
    # For breast cancer dataset
    [1e-6, 1e-6, 0.001, 0.01],
    # For digits greater than or less than 5 dataset
    [1e-4, 1e-4, 0.01, 0.1],
    # For fir and pine coverage dataset
    [0.001, 0.01]
]

# How much the learning rate should decrease over time (only for SGD)
dataset_eta_decay_factors = [
```

```
    # For breast cancer dataset
    [None, None, 0.99, 0.99],
    # For digits greater than or less than 5 dataset
    [None, None, 0.95, 0.95],
    # For fir and pine coverage dataset
    [0.99, 0.99]
]

# Momentum (only for momentum-based ..duh)
# None for no momentum
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, 0.9]
]

# Samples (only for SGD)
dataset_batch_sizes = [
    # For breast cancer dataset
    # # N = 569
    [None, None, 24, 24],
    # For digits greater than or less than 5 dataset
    # # N = 1797
    [None, None, 32, 32],
    # For fir and pine coverage dataset
    # # N = 495141
    [1000, 1000]
]


# Iterations
dataset_Ts = [
    # For breast cancer dataset
    [10000, 10000, 10000, 10000],
    # For digits greater than or less than 5 dataset
    [10000, 10000, 10000, 10000],
    # For fir and pine coverage dataset
    [3000, 3000]
]

# 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(dataset_name, x.shape[0], x.shape[1]))

    # 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 testing
```

```
    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 sets
    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)

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


    print('***** Results on the {} dataset using scikit-learn logistic␣
↪regression model *****'.format(dataset_name))

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

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

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

    '''
    Trains, validates, and tests our logistic regression model for binary␣
↪classification
    '''
    # 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)

    # DONE: Set the ground truth to the appropriate classes (integers)␣
↪according to lecture
    # -1 and +1
    y_train = np.where(y_train == 0, -1, 1)
    y_val = np.where(y_val == 0, -1, 1)
    y_test = np.where(y_test == 0, -1, 1)

    model_options = zip(optimizer_types, alphas, eta_decay_factors, betas,␣
↪batch_sizes, Ts)

    for optimizer_type, alpha, eta_decay_factor, beta, batch_size, T in␣
↪model_options:

        # DONE: Initialize our logistic regression model
        model_ours = LogisticRegression()

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

        time_start = time.time()

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

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

        # DONE: Score model using mean accuracy on training set
```

```python
        predictions_train = model_ours.predict(x_train)
        score_train = np.mean(predictions_train == y_train)
        # print("Predictions", predictions_train)
        # print("Ground truth", y_train)
        print('Training set mean accuracy: {:.4f}'.format(score_train))

        # DONE: Score model using mean accuracy on training set
        predictions_val = model_ours.predict(x_val)
        score_val = np.mean(predictions_val == y_val)
        print('Validation set mean accuracy: {:.4f}'.format(score_val))

        # DONE: Score model using mean accuracy on training set
        predictions_test = model_ours.predict(x_test)
        score_test = np.mean(predictions_test == y_test)
        print('Testing set mean accuracy: {:.4f}'.format(score_test))

        compare_scores(
            score_train, score_val, score_test,
            sk_score_train, sk_score_val, sk_score_test
        )

    print('')
```

```
Preprocessing the breast cancer dataset (569 samples, 30 feature dimensions)
***** Results on the breast cancer dataset using scikit-learn logistic
regression model *****
Training set mean accuracy: 0.9531
Validation set mean accuracy: 0.9649
Testing set mean accuracy: 0.9737
***** Results of our logistic regression model trained on breast cancer dataset
*****
         optimizer_type=gradient_descent
         alpha=1e-06
         eta_decay_factor=None
         beta=None
         batch_size=None
         T=10000
Step=1000  Loss=0.4270500224685998
Step=2000  Loss=0.364788504772O5127
Step=3000  Loss=0.33337323820259707
Step=4000  Loss=0.31303327228726313
Step=5000  Loss=0.29829359870313954
Step=6000  Loss=0.2869407988965641
Step=7000  Loss=0.27785147000222793
Step=8000  Loss=0.2703727118147251
Step=9000  Loss=0.2640914146974563
Step=10000  Loss=0.2587304473641291
Total training time: 0.271522 seconds
```

```
Training set mean accuracy: 0.9179
Validation set mean accuracy: 0.9211
Testing set mean accuracy: 0.9035

Performance is acceptable.

***** Results of our logistic regression model trained on breast cancer dataset
*****
         optimizer_type=momentum_gradient_descent
         alpha=1e-06
         eta_decay_factor=None
         beta=0.9
         batch_size=None
         T=10000
Step=1000  Loss=0.41872181102313405
Step=2000  Loss=0.35935853700887027
Step=3000  Loss=0.32884081942389604
Step=4000  Loss=0.30888403977348616
Step=5000  Loss=0.29436364226160494
Step=6000  Loss=0.2831676593069645
Step=7000  Loss=0.27420674441552617
Step=8000  Loss=0.2668408571383118
Step=9000  Loss=0.2606622277298364
Step=10000  Loss=0.2553961402822704
Total training time: 0.286938 seconds
Training set mean accuracy: 0.9238
Validation set mean accuracy: 0.9298
Testing set mean accuracy: 0.9123

Performance is acceptable.

***** Results of our logistic regression model trained on breast cancer dataset
*****
         optimizer_type=stochastic_gradient_descent
         alpha=0.001
         eta_decay_factor=0.99
         beta=None
         batch_size=24
         T=10000
Step=1000  Loss=0.35844075283384147
Step=2000  Loss=0.33944414095967484
Step=3000  Loss=0.3291189775275483
Step=4000  Loss=0.3210265725550344
Step=5000  Loss=0.3159328870095344
Step=6000  Loss=0.3114689010270899
Step=7000  Loss=0.30789855591945703
Step=8000  Loss=0.3048316818599375
Step=9000  Loss=0.30236655211556535
```

```
Step=10000  Loss=0.2998336482716568
Total training time: 0.485271 seconds
Training set mean accuracy: 0.9120
Validation set mean accuracy: 0.9123
Testing set mean accuracy: 0.9035

Performance is acceptable.

***** Results of our logistic regression model trained on breast cancer dataset
*****
         optimizer_type=momentum_stochastic_gradient_descent
         alpha=0.01
         eta_decay_factor=0.99
         beta=0.9
         batch_size=24
         T=10000
Step=1000  Loss=0.5771194328230421
Step=2000  Loss=0.5311636065713808
Step=3000  Loss=0.5100023180942997
Step=4000  Loss=0.49107259383223995
Step=5000  Loss=0.4790405515433979
Step=6000  Loss=0.47106448042941645
Step=7000  Loss=0.462635166864956
Step=8000  Loss=0.4574672677518454
Step=9000  Loss=0.45192069558004394
Step=10000  Loss=0.45084615783322285
Total training time: 0.507263 seconds
Training set mean accuracy: 0.9150
Validation set mean accuracy: 0.9211
Testing set mean accuracy: 0.9298

Performance is acceptable.


Preprocessing the digits greater or less than 5 dataset (1797 samples, 64
feature dimensions)
***** Results on the digits greater or less than 5 dataset using scikit-learn
logistic regression model *****
Training set mean accuracy: 0.9156
Validation set mean accuracy: 0.9025
Testing set mean accuracy: 0.8861
***** Results of our logistic regression model trained on digits greater or less
than 5 dataset *****
         optimizer_type=gradient_descent
         alpha=0.0001
         eta_decay_factor=None
         beta=None
         batch_size=None
```

```
        T=10000
Step=1000  Loss=0.4281522490299727
Step=2000  Loss=0.37205103141668566
Step=3000  Loss=0.34492371452292153
Step=4000  Loss=0.3281520777210791
Step=5000  Loss=0.3164679248980921
Step=6000  Loss=0.3077450612352203
Step=7000  Loss=0.30093628951788
Step=8000  Loss=0.2954550770228764
Step=9000  Loss=0.29094187269077343
Step=10000  Loss=0.2871610619588243
Total training time: 0.544074 seconds
Training set mean accuracy: 0.8998
Validation set mean accuracy: 0.9109
Testing set mean accuracy: 0.8833

Performance is acceptable.

***** Results of our logistic regression model trained on digits greater or less
than 5 dataset *****
        optimizer_type=momentum_gradient_descent
        alpha=0.0001
        eta_decay_factor=None
        beta=0.9
        batch_size=None
        T=10000
Step=1000  Loss=0.43702451358572564
Step=2000  Loss=0.3745567724902582
Step=3000  Loss=0.34592803346712225
Step=4000  Loss=0.32862740201924195
Step=5000  Loss=0.3167151784655617
Step=6000  Loss=0.30788059471122886
Step=7000  Loss=0.3010117679133514
Step=8000  Loss=0.2954958828339183
Step=9000  Loss=0.290961579330691
Step=10000  Loss=0.28716751033122145
Total training time: 0.561097 seconds
Training set mean accuracy: 0.8989
Validation set mean accuracy: 0.9136
Testing set mean accuracy: 0.8833

Performance is acceptable.

***** 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=0.95
```

```
        beta=None
        batch_size=32
        T=10000
Step=1000  Loss=0.43149918261962306
Step=2000  Loss=0.4218202712766845
Step=3000  Loss=0.41655529131456714
Step=4000  Loss=0.41308527014643365
Step=5000  Loss=0.4104401581730333
Step=6000  Loss=0.4083406886041121
Step=7000  Loss=0.4065811947018845
Step=8000  Loss=0.4051253344221095
Step=9000  Loss=0.40383349072613606
Step=10000  Loss=0.40270246290162337
Total training time: 0.889526 seconds
Training set mean accuracy: 0.8692
Validation set mean accuracy: 0.8858
Testing set mean accuracy: 0.8667

Performance is acceptable.

***** Results of our logistic regression model trained on digits greater or less
than 5 dataset *****
        optimizer_type=momentum_stochastic_gradient_descent
        alpha=0.1
        eta_decay_factor=0.95
        beta=0.9
        batch_size=32
        T=10000
Step=1000  Loss=0.29918906216155605
Step=2000  Loss=0.29374963641814233
Step=3000  Loss=0.291401847229874
Step=4000  Loss=0.28942154658319824
Step=5000  Loss=0.2881304060865366
Step=6000  Loss=0.28722889678808117
Step=7000  Loss=0.2862749165553029
Step=8000  Loss=0.28569153567648764
Step=9000  Loss=0.2849962753418624
Step=10000  Loss=0.2845142622856062
Total training time: 0.899369 seconds
Training set mean accuracy: 0.9017
Validation set mean accuracy: 0.9136
Testing set mean accuracy: 0.8861

Performance is acceptable.


Preprocessing the fir and pine coverage dataset (495141 samples, 54 feature
dimensions)
```

```
***** Results on the fir and pine coverage dataset using scikit-learn logistic
regression model *****
Training set mean accuracy: 0.7564
Validation set mean accuracy: 0.7572
Testing set mean accuracy: 0.7571
***** Results of our logistic regression model trained on fir and pine coverage
dataset *****
         optimizer_type=stochastic_gradient_descent
         alpha=0.001
         eta_decay_factor=0.99
         beta=None
         batch_size=1000
         T=3000
Step=1000  Loss=1.619466284067763
Step=2000  Loss=0.6060110305533307
Step=3000  Loss=0.5872086299295305
Total training time: 164.024928 seconds
Training set mean accuracy: 0.7109
Validation set mean accuracy: 0.7107
Testing set mean accuracy: 0.7119

Performance is acceptable.

***** Results of our logistic regression model trained on fir and pine coverage
dataset *****
         optimizer_type=momentum_stochastic_gradient_descent
         alpha=0.01
         eta_decay_factor=0.99
         beta=0.9
         batch_size=1000
         T=3000
Step=1000  Loss=0.9585612130481179
Step=2000  Loss=0.8596895839366846
Step=3000  Loss=0.801505433007527
Total training time: 163.885973 seconds
Training set mean accuracy: 0.7180
Validation set mean accuracy: 0.7189
Testing set mean accuracy: 0.7197

Performance is acceptable.
```

[ ]:

[ ]: