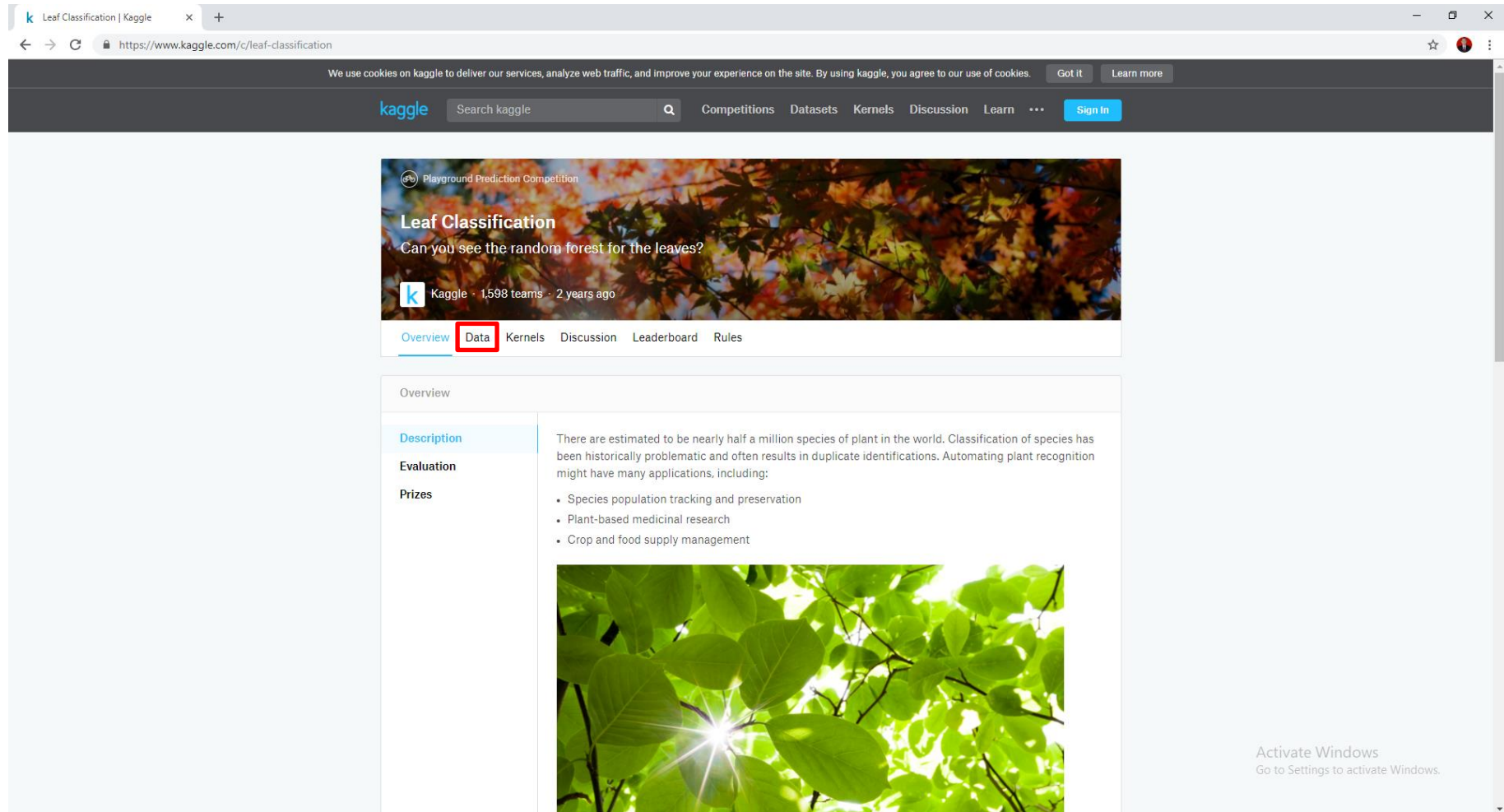


# Leaf Classification

(use `softmax_cross_entropy_with_logits`)

# Training Data

1. Download Dataset from: <https://www.kaggle.com/c/leaf-classification>



The screenshot shows the Kaggle website interface for the 'Leaf Classification' competition. The browser address bar displays the URL <https://www.kaggle.com/c/leaf-classification>. The page header includes the Kaggle logo, a search bar, and navigation links for Competitions, Datasets, Kernels, Discussion, Learn, and a Sign In button. The main content area features a banner for the 'Leaf Classification' competition with the text 'Can you see the random forest for the leaves?' and 'Kaggle · 1,598 teams · 2 years ago'. Below the banner is a tabbed interface with 'Overview', 'Data', 'Kernels', 'Discussion', 'Leaderboard', and 'Rules'. The 'Data' tab is selected and highlighted with a red box. The 'Overview' section is visible, showing a 'Description' tab and a list of prizes. The prizes include 'Species population tracking and preservation', 'Plant-based medicinal research', and 'Crop and food supply management'. A large image of green leaves is displayed at the bottom of the overview section.

Leaf Classification  
Can you see the random forest for the leaves?  
Kaggle · 1,598 teams · 2 years ago

Overview Data Kernels Discussion Leaderboard Rules

Overview

Description

Evaluation

Prizes

- Species population tracking and preservation
- Plant-based medicinal research
- Crop and food supply management

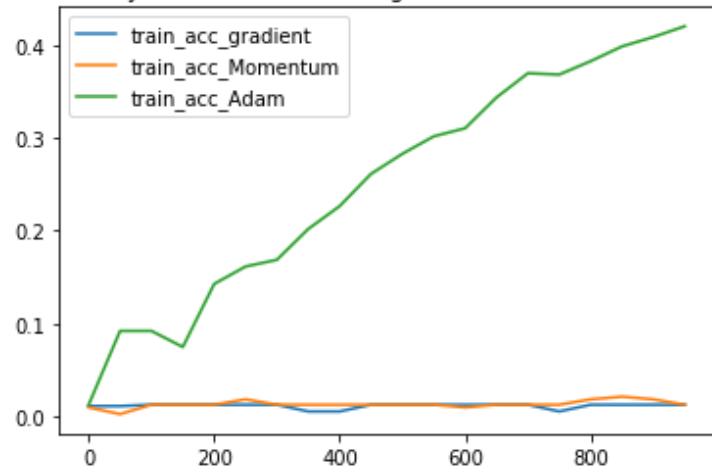
Activate Windows  
Go to Settings to activate Windows.

# Description

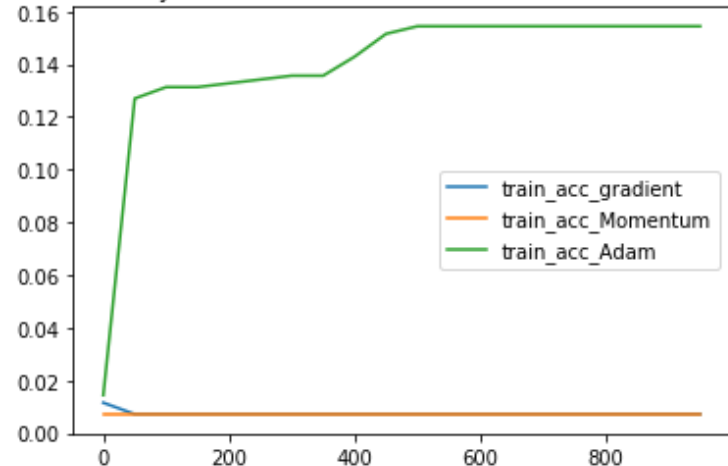
- This homework tried to make leaf classification with dataset from <https://www.kaggle.com/c/leaf-classification> using **MLP (Multilayer Perceptron)**. The dataset consists approximately 1,584 images and contains binary leaf images and extracted features, including shape, margin & texture, to accurately identify 99 species of plants.
- This work used “softmax regression and cross entropy error” to make prediction of leaf classification with some parameters tuning:
  - ✓ Compare the performance of different optimizers (3 optimizers)
  - ✓ Compare the performance of sigmoid, ReLU, ELU
  - ✓ Compare the performance of normalizing the input data or not
- You can use the following code to normalize the input features
  - ✓ `from sklearn.preprocessing import StandardScaler`
  - ✓ `scaler = StandardScaler()`
  - ✓ `feature_matrix = scaler.fit_transform(feature_matrix)`

# Simulation Result (Accuracy Performance)

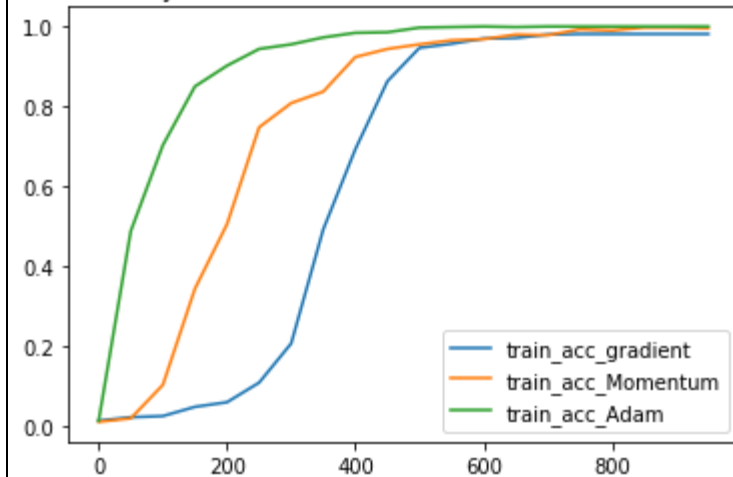
Accuracy Performance With Sigmoid and Without Normalize



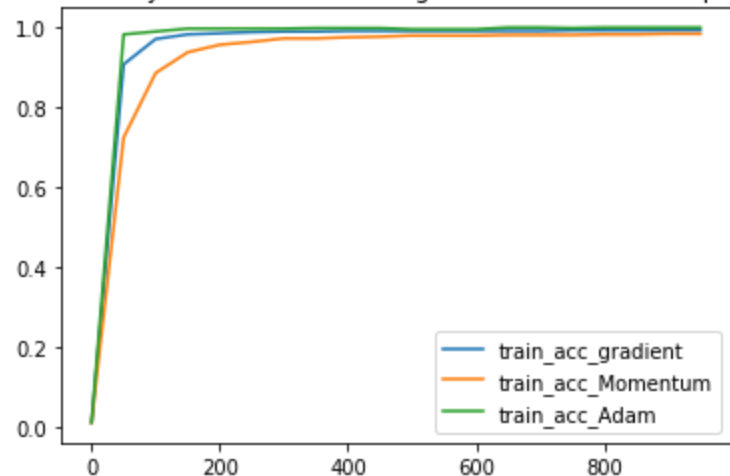
Accuracy Performance With RELU and Without Normalize



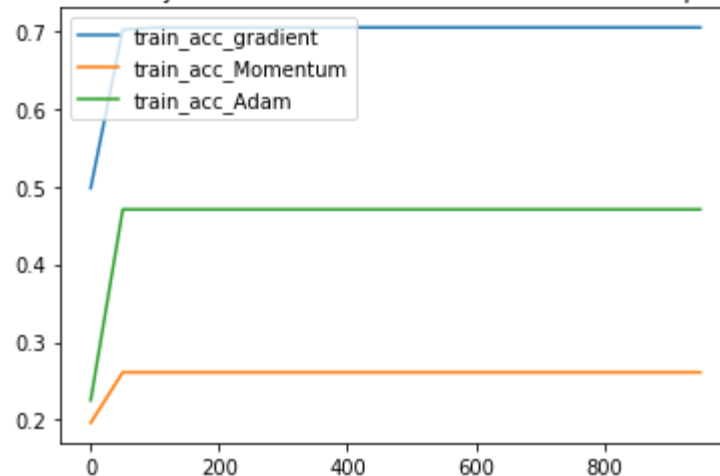
Accuracy Performance With ELU and Without Normalize



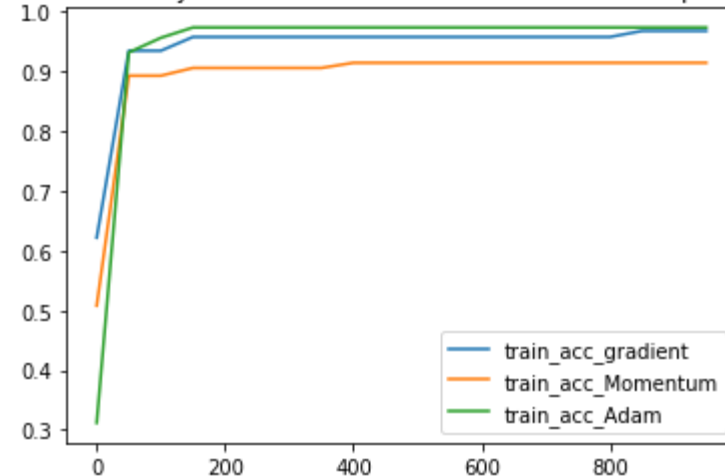
Accuracy Performance With Sigmoid and Normalize Input



Accuracy Performance With RELU and Normalize Input



Accuracy Performance With ELU and Normalize Input



# Simulation Result (Accuracy Performance)

- Un - Normalize Input

Optimizers	Function	Number of Epochs	Highest Accuracy
Gradient Descent	Sigmoid	950	0.012987013
Momentum	Sigmoid	950	0.012987013
Adam	Sigmoid	950	0.4199134
Gradient Descent	RELU	950	0.007215007
Momentum	RELU	950	0.007215007
Adam	RELU	950	0.15440115
Gradient Descent	ELU	950	0.981241
Momentum	ELU	950	0.995671
Adam	ELU	600	1.0

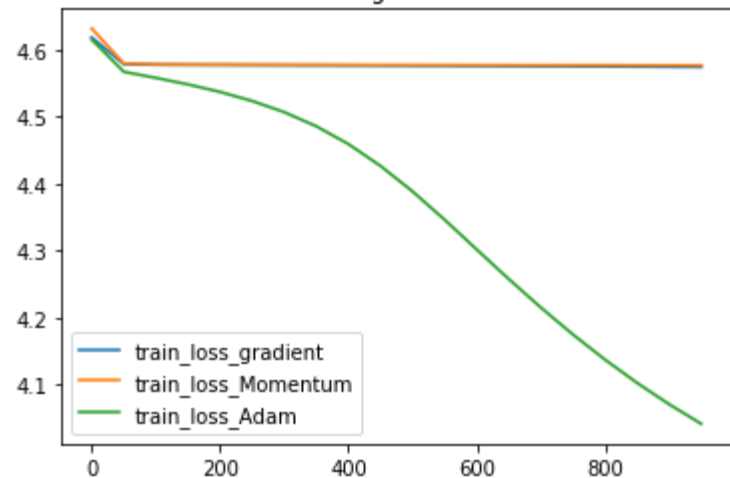
# Simulation Result (Accuracy Performance)

- **Normalize Input**

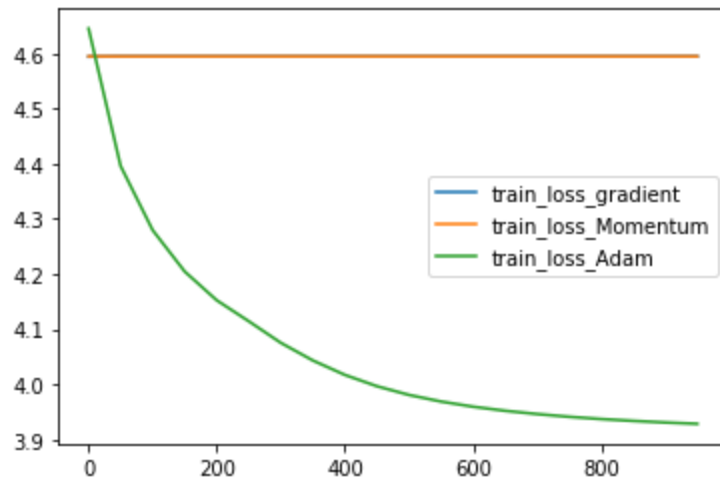
Optimizers	Function	Number of Epochs	Highest Accuracy
Gradient Descent	Sigmoid	950	0.992785
Momentum	Sigmoid	950	0.984127
Adam	Sigmoid	650	1.0
Gradient Descent	RELU	950	0.7041847
Momentum	RELU	950	0.26118326
Adam	RELU	950	0.47041848
Gradient Descent	ELU	950	0.96825397
Momentum	ELU	950	0.91486293
Adam	ELU	950	0.97402596

# Simulation Result (Loss Performance)

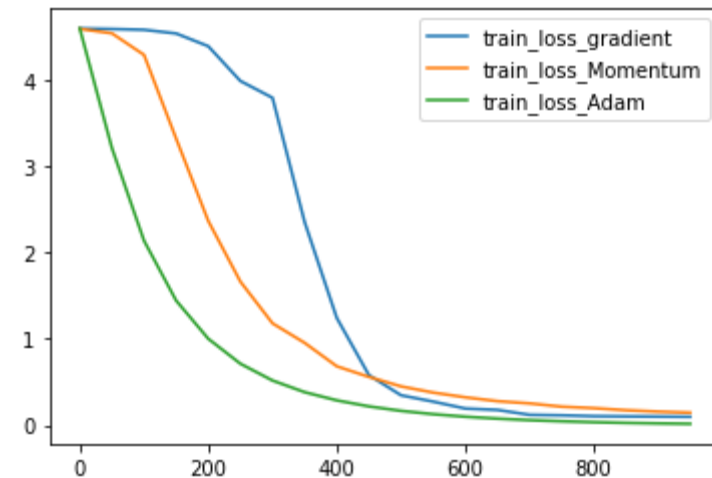
Loss Performance With Sigmoid and Without Normalize



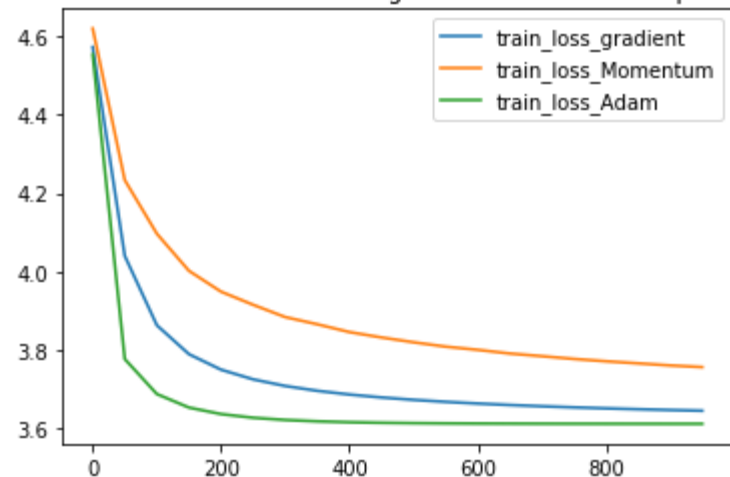
Loss Performance With RELU and Without Normalize



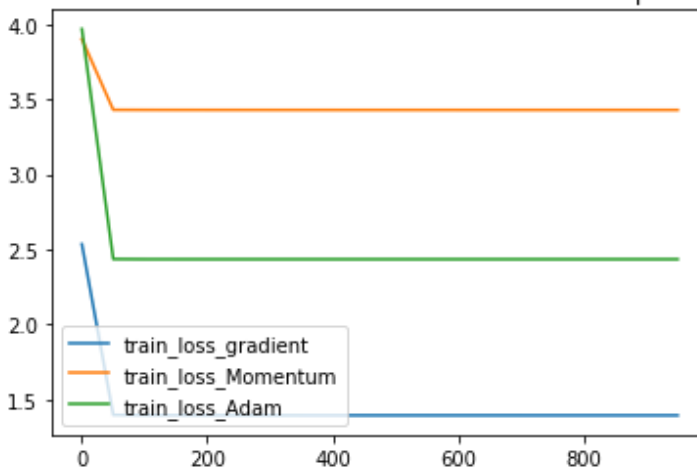
Loss Performance With ELU and Without Normalize



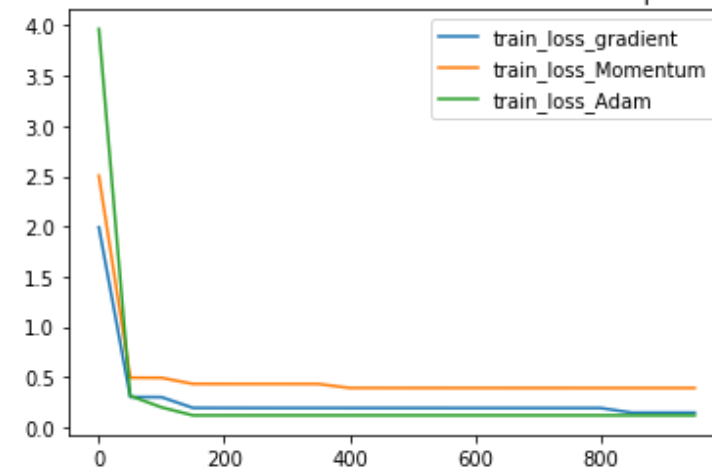
Loss Performance With Sigmoid and Normalize Input



Loss Performance With RELU and Normalize Input



Loss Performance With ELU and Normalize Input



# Simulation Result (Loss Performance)

- Un - Normalize Input

Optimizers	Function	Number of Epochs	Lowest Loss
Gradient Descent	Sigmoid	950	4.574041
Momentum	Sigmoid	950	4.5766363
Adam	Sigmoid	950	4.0416007
Gradient Descent	RELU	950	4.59512
Momentum	RELU	950	4.59512
Adam	RELU	950	3.928196
Gradient Descent	ELU	950	0.09765818
Momentum	ELU	950	0.14541991
Adam	ELU	950	0.016129047



# Simulation Result (Loss Performance)

- Normalize Input

Optimizers	Function	Number of Epochs	Lowest Loss
Gradient Descent	Sigmoid	950	3.645684
Momentum	Sigmoid	950	3.7568676
Adam	Sigmoid	950	3.6123497
Gradient Descent	RELU	950	1.3925711
Momentum	RELU	950	3.4282568
Adam	RELU	950	2.433491
Gradient Descent	ELU	950	0.14606574
Momentum	ELU	950	0.39170784
Adam	ELU	950	0.119353764