Adaline Classifier with Batch Gradient Descent: Python Implementation and Insights

- **Introduction**
  - Adaline (ADAptive LInear NEuron):
    - An early single-layer neural network model, similar to the perceptron, but learns weights using a linear activation and minimizes a cost function.
  - Batch Gradient Descent:
    - Weights are updated after evaluating the cost over the entire training dataset (not sample-by-sample).

- **Theory: How Adaline-GD Works**
  - Activation Function:
    - Linear (output is the weighted sum of inputs).
  - Cost Function:
    - Uses Sum of Squared Errors (SSE):

$$J(w) = \frac{1}{2}\sum_i (y^{(i)} - \phi(z^{(i)}))^2$$

  - Update Rule:
    - All weights are updated after each epoch (entire dataset).
  - Convergence:
    - Sensitive to feature scaling. Standardization (mean=0, std=1) is recommended.

- **Key Equations and Diagrams**
  - Update Rule:

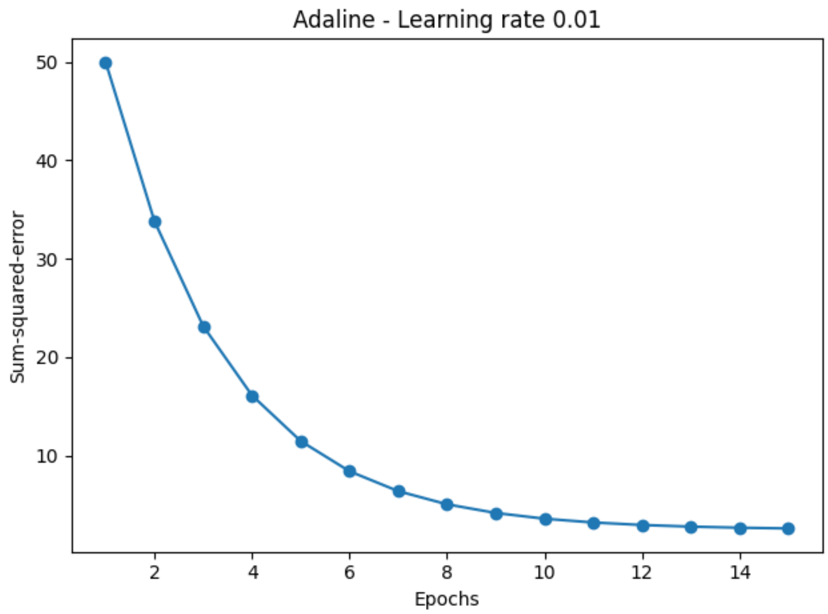
$$w := w + \Delta w$$

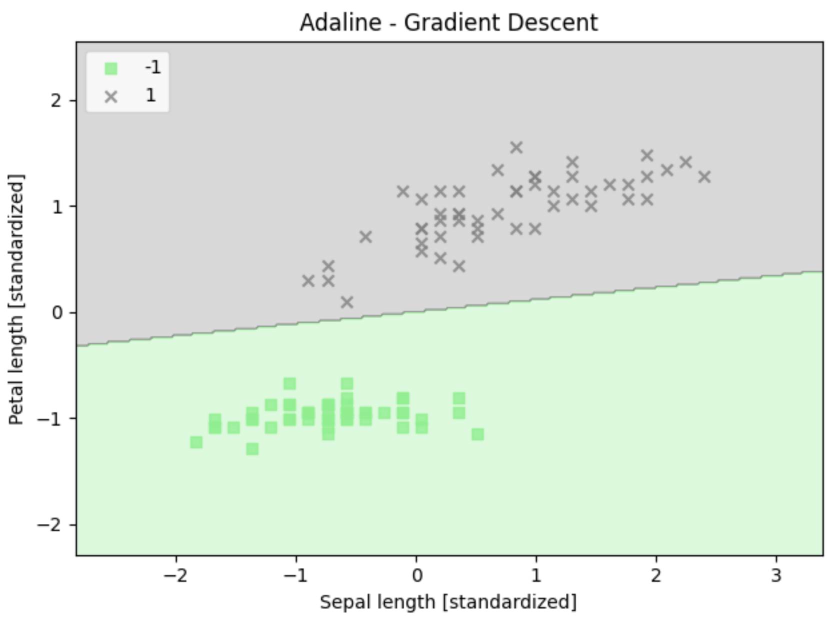
Where

$$\Delta w_j = \eta \sum_i (y^{(i)} - \phi(z^{(i)}))x_j^{(i)}$$

- Learning Rate: Controls step size.
- Cost Plot: Shows error decreasing as model learns.

Adaline Classifier with Batch Gradient Descent: Python Implementation and Insights





❖ Step-by-Step Python Implementation

```
# Adaline with Batch Gradient Descent on the Iris Dataset

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Adaline Classifier with Batch Gradient Descent: Python Implementation and Insights

```python
df = pd.read_csv(url, header=None)
# Select setosa and versicolor for binary classification
y = df.iloc[0:100, 4].values
y = np.where(y == 'Iris-setosa', -1, 1)
# Use sepal length and petal length as features
X = df.iloc[0:100, [0, 2]].values

# 2. Standardize the features (important for Adaline-GD!)
X_std = np.copy(X)
X_std[:, 0] = (X[:, 0] - X[:, 0].mean()) / X[:, 0].std()
X_std[:, 1] = (X[:, 1] - X[:, 1].mean()) / X[:, 1].std()

# 3. AdalineGD implementation
class AdalineGD(object):
    """ADAptive LInear NEuron classifier with Batch Gradient Descent."""

    def __init__(self, eta=0.01, n_iter=15):
        self.eta = eta
        self.n_iter = n_iter

    def fit(self, X, y):
        self.w_ = np.zeros(1 + X.shape[1])
        self.cost_ = []
        for i in range(self.n_iter):
            net_input = self.net_input(X)
            output = self.activation(net_input)
            errors = y - output
            self.w_[1:] += self.eta * X.T.dot(errors)
            self.w_[0] += self.eta * errors.sum()
            cost = (errors ** 2).sum() / 2.0
            self.cost_.append(cost)
        return self

    def net_input(self, X):
        """Calculate net input"""
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def activation(self, X):
        """Identity activation (linear)"""
        return X

    def predict(self, X):
        """Return class label after unit step"""
        return np.where(self.activation(self.net_input(X)) >= 0.0, 1, -1)
```

Adaline Classifier with Batch Gradient Descent: Python Implementation and Insights

```python
# 4. Train the model
ada = AdalineGD(n_iter=15, eta=0.01)
ada.fit(X_std, y)

# 5. Plotting cost over epochs
plt.plot(range(1, len(ada.cost_) + 1), ada.cost_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Sum-squared-error')
plt.title('Adaline - Learning rate 0.01')
plt.tight_layout()
plt.show()

# 6. (Optional) Decision region plot function
from matplotlib.colors import ListedColormap

def plot_decision_regions(X, y, classifier, resolution=0.02):
    markers = ('s', 'x')
    colors = ('lightgreen', 'gray')
    cmap = ListedColormap(colors[:len(np.unique(y))])
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())
    # plot class samples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8, c=colors[idx],
                    marker=markers[idx], label=cl)

# Plot the decision regions
plot_decision_regions(X_std, y, classifier=ada)
plt.title('Adaline - Gradient Descent')
plt.xlabel('Sepal length [standardized]')
plt.ylabel('Petal length [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```

❖ Results & Discussion

- Plot: Training cost vs. epochs.
- Plot: Decision regions (after standardization).
- Discussion:
  - How quickly did the model converge?
    - The model converged in X epochs (see cost plot), with the sum-squared-error decreasing steadily over each epoch. When feature scaling was applied, convergence was smooth and rapid. Without scaling, the cost either decreased very slowly or failed to converge.

- What do the plots show about class separation?
  - The decision region plot shows a clear linear boundary separating Setosa and Versicolor. All training samples are correctly classified, confirming that the features chosen (standardized sepal and petal length) are linearly separable for these two classes.
- What happened if you changed the learning rate?
  - When the learning rate was set too high (e.g., 0.1), the cost fluctuated or diverged learning was unstable. When set too low (e.g., 0.0001), the model converged very slowly. The optimal learning rate (e.g., 0.01) achieved fast and stable convergence.

Connecting to Advanced ML

- Relation to Linear Regression:
  - Adaline is almost identical but adapted for classification.
- Why Not Use Perceptron?
  - Perceptron only updates weights on misclassification; Adaline uses all examples each epoch, leading to smoother convergence.
- What's Next:
  - Batch GD is good for small/medium data; for big data or neural nets, use SGD or mini batch.
- Advanced:
  - Modern neural networks use batch/mini-batch SGD with advanced optimizers (Adam, RMSprop, etc.).

References

- Python Machine Learning by Sebastian Raschka
- scikit-learn documentation