Yıldız Teknik Üniversitesi

BLM2642 Ödev 1

4 Sınıflı Veri Kümesi Üzerinde Optimizasyon Algoritmalarını Karşılaştırma

İSİM : YAKUP GÜLCAN

ÖĞRENCİ NUMARASI : 23011102

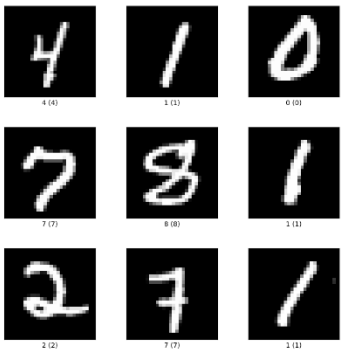
Video Linki :
https://www.youtube.com/watch?v=Re0Bm3HXZkQ

1 - Model Açıklaması

1.1 – Veri Kümesi

Bu projede, MNIST veri kümesinden el yazısı rakam görselleri kullanılmıştır. MNIST, 28x28 piksel boyutunda siyah-beyaz el yazısı rakam görsellerini içeren ve makine öğrenmesi problemlerinde sıkça kullanılan bir veri kümesidir. Bu verisetinden yalnızca 0, 1, 2 ve 3 etiketine sahip görseller seçilerek 4 sınıflı veri kümesi oluşturuldu. Her sınıf için: **2000 eğitim örneği , 500 test örneği** çekilerek toplamda 8000 eğitim örneği ve 2000 test örneği kullanıldı.

Veri seti, TensorFlow kütüphanesi aracılığıyla yüklendi ve görseller, C dilinde işlenebilmesi için .txt formatına dönüştürüldü. Eğitim ve test örneklerinin sayısı, algoritmaların performansını daha iyi gözlemleyebilmek için büyük tutuldu.



1.2 – Model Mimarisi

Dört sınıflı bir sınıflandırma problemi olduğu için model, her sınıf için ayrı bir perceptron ağı gibi düşünülebilir.

- Giriş verisi olarak görsellerin pixel değerleri kullanıldı. Ayrıca bias terimi olması için giriş vektörünün sonuna 1 eklendi. Böylece girişin boyutu 785 oldu.
- 4 sınıflı bir veri kümesine sahip olduğumuz için her sınıf için ayrı bir z değeri hesaplanması gerekir. Giriş verimizi her sınıf için giriş boyutumuzla aynı boyutta weightler ile çarpmamız gerekir. Bu yüzden 4*785 boyutunda weight matrisi kullanıyoruz.

$$z_i = W_i \cdot X$$

Şekil 1

- Bu yolla hesaplanan 4 adet ham z değeri (logit) softmax fonksiyonuna verilerek her sınıfa ait olasılık belirlenmiş olur. Daha sonra en yüksek sınıfa sahip bulunarak modelin örnek için tahmini bulunmuş olur.

1.3 – Aktivasyon Fonksiyonu

Aktivasyon fonksiyonu olarak sınıflandırma problemlerinde sıkça kullanılan softmax kullanıldı. Softmax, ham toplamları (logits) kullanarak olasılık değeri oluşturur.

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{4} e^{z_j}}$$

Şekil 2

Böylece softmax logitsleri kullanarak her sınıfın olasılığını normalize eder ve olasılık değerlerini 0-1 arasında oluşturur.

1.4 – Loss Fonksiyonu

Modelde, dört sınıflı softmax çıktısına uygun olarak **Categorical Cross-Entropy** loss fonksiyonu kullanıldı. Bu loss fonksiyonu, modelin tahminleri y_hat ile gerçek etiket y arasındaki farkı belirtir. Fonksiyonda y_hat olasılığının logaritması alınarak bu olasılık 1den uzaklaştıkça daha büyük bir loss oluşması sağlanır. Ayrıca yanlış etiketler için 1-y_hat kullanılarak modelin yanlış etiketlere verdiği olasılık büyüdükçe loss fonksiyonun artması sağlanır.

$$L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{4} y_{ij} \cdot \log(\hat{y}_{ij})$$

Şekil 3

1.5 – Gradyan Hesaplaması

Gradyanlar, weight matrisinin güncellenmesi için loss fonksiyonunun türevleri olarak hesaplanır. Zincir kuralı sayesinde Loss fonksiyoun türevi alınarak weightler için gradyan hesaplaması yapılır.

$$y_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

Şekil 4

$$L(y,t) = -\sum_{k=1}^{K} t_k \ln y_k$$

Şekil 5

$$\frac{\partial L}{\partial z_k} = y_k - t_k$$

Şekil 6

Bu şekilde matematiksel ifadede görüldüğü gibi loss fonksiyonun z'ye göre türevinde **y – t_hat** değeri hesaplanır. Z değerinin de w_i,j değerine göre türevi hesaplandığında gradyan matrisimizi bulmuş oluruz.

$$\partial L/\partial w_(i,j) = (y_i - \hat{y}_i) * X_(i,j)$$

Şekil 7

Şekil 7'de görüldüğü üzere türev hesabımızı bu şekilde yapabiliriz.

1.6 – ADAM Algoritması

ADAM algoritması RMSprop ve AdaGrad algoritmalarından esinlenerek oluşturulan bir optimizasyon algoritmasıdır. Gradient descent algoritmasındaki gibi her adımda sabit adım büyüklüğü ile ilerlemek yerine mevcut parametrenin daha önceki ağırlıklı ortalaması ve karesel ağırlıklı ortalaması kullanılarak adaptif bir şekilde yakınsamaya çalışır. Bu şekilde türevin 0 olduğu ama minimum olmayan suddle noktalardan hem de yerel minimumlardan kaçınmakta etkilidir.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla W_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla W_t)^2$$

Şekil 8

$$W_t = W_{t-1} - \eta \frac{m_t}{\sqrt{v_t} + \epsilon}$$

Şekil 9

Şekil 8'de hesaplanan mt (first moment estimate) ve vt(second moment estimate) bias correction kullanılar mt_hat ve vt_hat değerleri elde edilir. Daha sonra bu değerler şekil 9'daki ifade de görüldüğü gibi kullanarak weight parametreleri güncellenir.

1.6 – Gradient Descent Algoritması

Gradient Descent (GD), tüm veri kümesi üzerindeki gradyanları hesaplayıp weightleri güncelleyen temel bir optimizasyon yöntemidir. Adım büyüklüğü hesaplanan gradyanın mutlak değeri ve Learning rate ile ilişkilidir. Adam algoritmasındaki gibi momentum değeri kullanmaz, ilkel bir algoritmadır. Veri boyutu arttıkça hesaplama maliyeti de arttığı için günümüzde bu metod mini-batch gradient descent algoritması olarak kullanılmaktadır. Ancak projede tüm veri kümesini kullanacak asıl şekli olarak kullanıldı.

$$W = W - \eta \cdot \nabla W$$

Şekil 10

1.6 – Stochastic Gradient Descent Algoritması

Bu algoritma ise gradient descent algoritmasının tüm veri kümesi üzerinde gradyan hesaplaması yapmak yerine veri kümesindeki her örnek için gradyan hesaplayıp parametrelerin her örnek için güncellenmesini sağlar. Böylece yüksek boyutlu verilerde hesaplama maliyeti düşürülmüş olur. Ancak bu algoritma her örnek için güncelleme yaptığından dolayı kararsız bir algoritmadır ve minimum noktaya yaklaştığında zik zak yapar ve minimum noktadan uzaklaşma ihtimali vardır.

2 – Açıklamalı Program Kodları

2.1 – Veri Setinin Elde Edilmesi

```python
import os
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'
import numpy as np
from tensorflow.keras.datasets import mnist

# Her örnekten kaçar tane alınacağının belirlenmesi
test_count = 500
train_count = 2000

# MNIST datasetinin yüklenmesi
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Train verileri için filtreleme yapılması
train_mask = (y_train == 0) | (y_train == 1) | (y_train == 2) | (y_train == 3)
x_train_filtered = x_train[train_mask]
y_train_filtered = y_train[train_mask]

# Test verileri için filtreleme yapılması
test_mask = (y_test == 0) | (y_test == 1) | (y_test == 2) | (y_test == 3)
x_test_filtered = x_test[test_mask]
y_test_filtered = y_test[test_mask]
```

- Tensorflow aracılığı ile MNIST datasetine yüklendi. Test_count ve train_Count değişkenleri kullanılarak her örnekten kaçar adet test ve train verisi alınacağı belirlendi. Daha Sonrasında yüklenen veri setinden istediğimiz etikete sahip veriler filtrelendi.

```python
# İstenen sayıda örneğin alınması
x_train_final = np.concatenate([x_train_filtered[y_train_filtered == 0][:train_count],
                                x_train_filtered[y_train_filtered == 1][:train_count],
                                x_train_filtered[y_train_filtered == 2][:train_count],
                                x_train_filtered[y_train_filtered == 3][:train_count]])
y_train_final = np.concatenate([y_train_filtered[y_train_filtered == 0][:train_count],
                                y_train_filtered[y_train_filtered == 1][:train_count],
                                y_train_filtered[y_train_filtered == 2][:train_count],
                                y_train_filtered[y_train_filtered == 3][:train_count]])

x_test_final = np.concatenate([x_test_filtered[y_test_filtered == 0][:test_count],
                               x_test_filtered[y_test_filtered == 1][:test_count],
                               x_test_filtered[y_test_filtered == 2][:test_count],
                               x_test_filtered[y_test_filtered == 3][:test_count]])
y_test_final = np.concatenate([y_test_filtered[y_test_filtered == 0][:test_count],
                               y_test_filtered[y_test_filtered == 1][:test_count],
                               y_test_filtered[y_test_filtered == 2][:test_count],
                               y_test_filtered[y_test_filtered == 3][:test_count]])

```

- Numpy aracılığı işe filtrelenen verilerden istediğimiz sayıda örnek birleştirilerek kaydedildi.

```python
# Pixel değerlerinin 0-1 aralığına normalize edilmesi
x_train_final = x_train_final / 255.0
x_test_final = x_test_final / 255.0

# Elde edilen verilerin TXT formatında kaydedilmesi.
with open("4class_train_data.txt", "w") as f:
    for img in x_train_final:
        flattened = img.flatten()
        line = " ".join(map(str, flattened))
        f.write(line + "\n")

with open("4class_train_label.txt", "w") as f:
    for label in y_train_final:
        f.write(str(label) + "\n")


with open("4class_test_data.txt", "w") as f:
    for img in x_test_final:
        flattened = img.flatten()
        line = " ".join(map(str, flattened))
        f.write(line + "\n")

with open("4class_test_label.txt", "w") as f:
    for label in y_test_final:
        f.write(str(label) + "\n")

print("Örnekler başarıyla kaydedildi.")

```

- Kaydedilen değerler önce 0-1 aralığına normalize edildi sonrasında test ve train dataları düzleştirilerek txt dosyalarına yazıldı.

2.2 – Model Kodlarının Oluşturulması

```
#define TRAIN_ROWS 800      // Eğitim kümesindeki örnek sayısı
#define INPUT_SIZE 785          // Modelin giriş büyüklüğü /28*28+1
#define EPSILON 1e-8            // log(0) ve 0a bölmeden kaçınmak için eps değeri
#define NUM_CLASSES 4           // Verideki sınıf sayısı
#define LEARNING_RATE 0.1       // GD ve SGD için learning rate
#define ADAM_LEARNING_RATE 0.01 // Adam için learning rate
#define EPOCHS 100              // Epochs sayısı, 1 epoch = veri kümesindeki tüm örneklerin kullanılması
#define EPOCH_PERIOD 10         // epoch bilgisinin hangi aralıklarla bastırılacağını belirlemek için değişken
#define TEST_COUNT 200          // Test kümesindeki örnek sayısı
#define ADAM_BETA1 0.9          // Adam için hyperparameterlar
#define ADAM_BETA2 0.999

// Kaydedilecek dosyaların isimleri
#define ADAM_LOG "adam_logs_m.csv"
#define ADAM_WEIGHTS "adam_weights_m.csv"
#define GD_LOG "gd_logs_m.csv"
#define GD_WEIGHTS "gd_weights_m.csv"
#define SGD_LOG "sgd_logs_m.csv"
#define SGD_WEIGHTS "sgd_weights_ml.csv"
```

- Kodda kullanılacak sabitler ve modele ait verilerin kaydedileceği dosyaların isimlerinin belirlenmesi.

```
/*
Oluşturacağımız model ile ilgili verileri struct yapısında tutarak kod karmaşıklığının önüne geçiyoruz.
Model ile ilgili ve fonksiyonlarda sıkça beraber kullanılan değerleri Model structumuzda tutuyoruz.
*/

typedef struct {
    double **train_data;
    int *train_labels;
    double **weights;
    double *logits;                 // softmax öncesi z değerlerini tutan pointer
    double *probabilities;          // z çıktılarının softmaxe verilerek her sınıf için tahminin belirlenmesi
    double **gradients;
    double **initial_weights;       // modelleri çalıştırmadan önce elde ettiğimiz random weightsleri kaydediyoruz.
    double **test_data;
    int *test_labels;
    double **logs;                  // modellerin her epochtaki loss, accuracy ve zaman metriklerini tutmak için pointer
    /*
    Allocate Memory For Logs

    train_loss
    train_accuracy
    test_loss
    test_accuracy
    time_elapsed
    */

    // Adam parametreleri
    double **adam_m;  // First moment estimate
    double **adam_v;  // Second moment estimate
    int adam_t;       // Adam Timestep
} Model;
```

- Kodda fonksiyonlarla iletişimin daha rahat ve kompleks olmadan yapılması, verilerin daha düzenli tutulabilmesi için Model structunun oluşturulması.

2.3 – Verilerin Yüklenmesi ve Bellek Tahsisleri

```c
// train ve test verilerinin dosyadan okunması ve modele ait pointerler için bellek tahsisi yapılması.
void read_train_data(const char *data_file, const char *label_file, Model *model) {
    FILE *data_fp, *label_fp,*test_label_file,*test_data_file; // dosyalardan okuma yapmamız için gereken file pointerlar
    int i, j, row;

    // modelimizdeki tüm pointerleri NULL olarak başlatıyoruz
    model->train_data = NULL;
    model->train_labels = NULL;
    model->weights = NULL;
    model->logits = NULL;
    model->probabilities = NULL;
    model->gradients = NULL;
    model->adam_m = NULL;
    model->adam_v = NULL;
    model->initial_weights = NULL;
    model->test_data = NULL;
    model->test_labels = NULL;
    model->logs = NULL;

    // Memory allocation
    model->train_data = (double **)malloc(TRAIN_ROWS * sizeof(double *));
    model->train_labels = (int *)malloc(TRAIN_ROWS * sizeof(int));
    model->weights = (double **)malloc(NUM_CLASSES * sizeof(double *));
    model->logits = (double *)malloc(NUM_CLASSES * sizeof(double));
    model->probabilities = (double *)malloc(NUM_CLASSES * sizeof(double));
    model->gradients = (double **)malloc(NUM_CLASSES * sizeof(double *));
    model->initial_weights = (double **)malloc(NUM_CLASSES * sizeof(double *));
    model->test_labels = (int *) malloc(TEST_COUNT * sizeof(int));
    model->logs = (double **) malloc(5 * sizeof(double *));

    // logs
    for(i=0;i<5;i++){
        model->logs[i] = (double *)malloc(EPOCHS * sizeof(double));
    }

    // test_data
    model->test_data = (double **)malloc(TEST_COUNT * sizeof(double *));
    for (i = 0; i < TEST_COUNT; i++) {
        model->test_data[i] = (double *)malloc(INPUT_SIZE * sizeof(double));
    }

    // Dosyadan okuma
    test_label_file = fopen("4class_test_label.txt", "r");
    if(test_label_file == NULL) {
        fprintf(stderr, "Error opening label file\n");
        exit(1);
    }

```

- Model structumuzdaki pointerlara bellek tahsisini ve train ve test verilerinin dosyalardan okunma işlemini gerçekleştiriyoruz.

```c
    // parametrelerin rastgele olarak başlatılması
    srand(time(NULL));
    for (i = 0; i < NUM_CLASSES; i++) {
        for (j = 0; j < INPUT_SIZE; j++) {
            model->weights[i][j] = (double) rand()/RAND_MAX*0.1 - 0.05 ;
            model->initial_weights[i][j] = model->weights[i][j];
        }
    }
```

- Weightlerin -0.05 ile +0.05 aralığında rastgele başlatılması.

2.4 – Genel Fonksiyonlar

```c
// veri örnekleri için tahmin edilen değeri hesaplayan fonksiyon.
int predict(Model *model, double *input) {
    int cls, feature;
    int predicted_class = 0;

    // logits yani z değerlerini hesaplıyoruz
    for(cls = 0; cls < NUM_CLASSES; cls++) {
        model->logits[cls] = 0.0;
        for(feature = 0; feature < INPUT_SIZE; feature++) {
            model->logits[cls] += input[feature] * model->weights[cls][feature];
        }
    }

    // Softmax
    softmax(model);

    // en yüksek olasılığa sahip sınıf etiketini buluyoruz
    for(cls = 1; cls < NUM_CLASSES; cls++) {
        if(model->probabilities[cls] > model->probabilities[predicted_class]) {
            predicted_class = cls;
        }
    }

    return predicted_class;
}
```

- Predict fonksiyonu ile verilen girdi için logit değerleri hesaplanır ve daha sonrasında hesaplanan değerler için aşağıda görülen softmax fonksiyonu çağrılır ve bu olasılıklar olasılıklara dönüştürülür. Daha sonra en yüksek olasığıa sahip sınıfın indisi döndürülür.

```c
// forward passte elde ettiğimiz z değerlerini olasılık değerlerine dönüştürmek için softmax fonksiyonu
void softmax(Model *model) {
    int i;
    double max_z = model->logits[0];

    // Overflow'u önlemek için max değer bulma
    for (i = 1; i < NUM_CLASSES; i++) {
        if (model->logits[i] > max_z) {
            max_z = model->logits[i];
        }
    }

    double logits_expsum = 0.0;
    for (i = 0; i < NUM_CLASSES; i++) {
        model->probabilities[i] = exp(model->logits[i] - max_z);
        logits_expsum += model->probabilities[i];
    }

    // Normalize etme
    for (i = 0; i < NUM_CLASSES; i++) {
        model->probabilities[i] /= logits_expsum;
    }
}
```

- Softmax fonksiyonu öncesinde çağrılan predict fonksiyonundan hesaplanan logit değerlerini kullanarak olasılık değerleri oluşturur. Tüm bunların öncesinde overflow hatasını engellemek için en yüksek değere sahip logit diğer logitlerden çıkarılarak normalizasyon uygulanır.

```c

// Categorical cross entropy kullanarak tek bir veri örneği için hata hesaplaması
double compute_loss(double *predictions, int true_label) {
    return -log(predictions[true_label] + EPSILON);
}

```

- Compute loss ile tahmin edilecek veriye ait etiketin sınıf olasılığının logaritması alınarak hata hesaplaması yapılır.

```c
// adam ve gradient descent algoritmasında her epochta gradyanları hesaplayan fonksiyon
void compute_gradients(Model *model) {
    int cls, row, feature;

    // gradyanları önce 0 ile dolduruyoruz.
    for (cls = 0; cls < NUM_CLASSES; cls++) {
        for (feature = 0; feature < INPUT_SIZE; feature++) {
            model->gradients[cls][feature] = 0.0;
        }
    }

    // Eğitim setindeki her veri örneği için gradyanları hesaplıyoruz
    for (row = 0; row < TRAIN_ROWS; row++) {

        // predict fonksiyonu kullanarak bulunduğumuz satıra ait olasılıkları elde ediyoruz
        predict(model, model->train_data[row]);

        // gradyanları hesaplayıp topluyoruz NUM_CLASSES*INPUT_SIZE kadar parametre
        // burada hem modelimizin gerçek etiketten farkını hem de yanlış etiket için verdiği olasılıkları kullanıyoruz.
        // Bu sayede daha kararlı bir öğrenme süreci elde ediyoruz.
        for (cls = 0; cls < NUM_CLASSES; cls++) {
            // gradyan hesabında kullanılan y-y_hat değerini kullanıyoruz.
            double error = model->probabilities[cls] - (cls == model->train_labels[row] ? 1.0 : 0.0);

            // g(w[cls][feature]) = (y[row]-y_hat[row])*x[row][feature] yaparak hesaplama yapıyoruz.
            for (feature = 0; feature < INPUT_SIZE; feature++) {
                model->gradients[cls][feature] += error * model->train_data[row][feature];
            }
        }
    }

    // Gradyanları normalize ediyoruz
    for (cls = 0; cls < NUM_CLASSES; cls++) {
        for (feature = 0; feature < INPUT_SIZE; feature++) {
            model->gradients[cls][feature] /= TRAIN_ROWS;
        }
    }
}
```

- Compute_gradients fonksiyonu ile model açıklamaları kısmında çıkarılan formüle uygun olarak gradyan hesaplamaları yapılır. Öncelikle her weighte ait gradyan tüm örnekler için hesaplanır daha sonra bu kümülatif toplam veri sayısına bölünerek normalize edilir.

```c
// gradyanları kullanarak weightleri güncelleyecek gradient descent algoritması wt+1 = wt - e*dL/dw
void gradient_descent(Model *model) {
    int cls, feature;
    // Iterate through all classes
    for (cls = 0; cls < NUM_CLASSES; cls++) {
        // Iterate through all features (including bias)
        for (feature = 0; feature < INPUT_SIZE; feature++) {
            // Update weights using the precomputed gradients
            model->weights[cls][feature] -= LEARNING_RATE * model->gradients[cls][feature];
        }
    }
}

```

- gradient_descent fonksiyonu ile hesaplanan gradyanlara uygun olarak weightler güncellenir.

2.5 – Algoritmalar

```c
// modeli gradient descent ile eğitmemiz için fonksiyon
void train_model_gd(Model *model) {
    int epoch, row, cls, feature;
    clock_t start, end; // algoritmanın çalışma süresini kaydetmek için değişkenler
    double elapsed_time;
    start = clock(); // döngü başlamadan önceki zamanı ölçüyoruz

    for (epoch = 0; epoch < EPOCHS; epoch++) {

        save_weights_log(model, GD_WEIGHTS, epoch);         // T-sne ile görselleştirme için her epochta mevcut weightleri kaydediyoruz


        /*
        Modelimizin performansı için her epochta model metriklerini kaydediyoruz.
        train ve test kümesi için loss ve doğruluk hesaplıyoruz.
        */
        model->logs[0][epoch] = compute_train_loss(model);
        model->logs[1][epoch] = calculate_accuracy_train(model);
        model->logs[2][epoch] = compute_test_loss(model);
        model->logs[3][epoch] = calculate_accuracy_test(model);

        // modelimizin eğitim sürecini ekrana bastırma
        if (epoch % EPOCH_PERIOD == 0 || epoch == 0) {
            printf("Epoch %d, Loss: %f\n", epoch, model->logs[0][epoch]);
        }

        //Gradyanları hesapla ve güncelle
        compute_gradients(model);
        gradient_descent(model);

        // her epochta geçen sürenin kaydını tutuyoruz.
        end = clock();
        elapsed_time = (double)(end - start) / CLOCKS_PER_SEC;
        model->logs[4][epoch] = elapsed_time;

    }
    // modelimiz çalışmasını bitirdikten sonra elde ettiğimiz verileri csv formatında kaydediyoruz
    save_logs(model,GD_LOG);
}
```

- train_model_gd fonksiyonu ile model gradient descent algoritması ile eğitilir. Her epochta modele ait mevcut weight değerleri ve modelin performans metrikleri kaydedilir. Daha önce açıklanan compute_gradients fonksiyonu ile gradyanlar hesaplanır ve gradient_descent fonksiyonunda da gradyanlara göre weightler güncellenir.

```c
// modelin sgd ile eğitilmesi için fonksiyon
void train_model_sgd(Model *model) {
    int epoch, cls, feature, row, iteration;
    clock_t start, end;
    double elapsed_time;
    start = clock();
    int rand_index;

    // random fonksiyonunun düzgün calısması için srand kullanıyoruz
    srand(time(NULL));

    for (epoch = 0; epoch < EPOCHS; epoch++) {
        save_weights_log(model, SGD_WEIGHTS, epoch); // T-Sne için weight kaydı

        // modele ait metriklerin her epochta tutulması
        model->logs[0][epoch] = compute_train_loss(model);
        model->logs[1][epoch] = calculate_accuracy_train(model);
        model->logs[2][epoch] = compute_test_loss(model);
        model->logs[3][epoch] = calculate_accuracy_test(model);

        // modelimize ait anlık loss verilerini ekrana bastırma
        if (epoch % EPOCH_PERIOD == 0 || epoch == 0) {
            printf("Rand_index: %d, Epoch: %d, Loss: %f \n", rand_index, epoch, model->logs[0][epoch]);
        }

        // her epochta tüm veri kümesi üzerinden geçebilmek için iterasyon kullanıyoruz
        for(iteration=0; iteration < TRAIN_ROWS; iteration++){

            // veri kümesinden rastgele örnek seciyoruz
            rand_index = rand() % TRAIN_ROWS;

            // forward pass ile modele ait olasılıkları buluyoruz
            predict(model,model->train_data[rand_index]);

            // her örnek için weightleri güncelliyoruz
            for (cls = 0; cls < NUM_CLASSES; cls++) {
                double error = model->probabilities[cls] - (cls == model->train_labels[rand_index] ? 1.0 : 0.0);

                for (feature = 0; feature < INPUT_SIZE; feature++) {
                    model->weights[cls][feature] -= LEARNING_RATE * error * model->train_data[rand_index][feature];
                }
            }
        }
        end = clock();
        elapsed_time = (double)(end - start) / CLOCKS_PER_SEC;
        model->logs[4][epoch] = elapsed_time;

    }
    // modele ait metriklerin kaydedilmesi
    save_logs(model,SGD_LOG);
}
```

- train_model_sgd fonksiyonu ile model stochastic gradient descent algoritması ile eğitilir. Öncelikle modele ait weightler ve metrikler her epochta kaydedilir. Daha sonrasında her epochta eğitim kümesindeki örnek sayısı kadar güncelleme yapılarak tüm veri seti üzerinden geçilmiş olur. Bu algoritma ile her örnek için gradyan hesaplanır ve ilgili weight anında değiştirilmiş olur. Her ne kadar daha sık güncelleme hızlı yakınsamayı sağlasa da daha sonrasında modelin unstabil bir şekilde davranmasına neden olur.

```c
// modelin adam algoritması ile eğitilmesi
void train_model_adam(Model *model) {
    int epoch, row, cls, feature;
    clock_t start, end;
    double elapsed_time;
    start = clock();

    // Adam parametrelerini başlatıyoruz
    if (model->adam_m == NULL) {
        initialize_adam_params(model);
    }

    for (epoch = 0; epoch < EPOCHS; epoch++) {
        model->adam_t++;

        save_weights_log(model, ADAM_WEIGHTS, epoch); //T-sne için weightlerin kaydedilmesi

        // metriklerin tutulması
        model->logs[0][epoch] = compute_train_loss(model);
        model->logs[1][epoch] = calculate_accuracy_train(model);
        model->logs[2][epoch] = compute_test_loss(model);
        model->logs[3][epoch] = calculate_accuracy_test(model);

        // belli periyotla verileri ekrana bastırma
        if (epoch % EPOCH_PERIOD == 0 || epoch == 0) {
            printf("Epoch %d, Loss: %f\n", epoch, model->logs[0][epoch]);
        }


        // mevcut gradyanları hesapla
        compute_gradients(model);

        // ADAM güncellemelerini uygulama
        for (cls = 0; cls < NUM_CLASSES; cls++) {
            for (feature = 0; feature < INPUT_SIZE; feature++) {

                // first moment estimate
                model->adam_m[cls][feature] = ADAM_BETA1 * model->adam_m[cls][feature] +
                                              (1 - ADAM_BETA1) * model->gradients[cls][feature];

                // second moment estimate
                model->adam_v[cls][feature] = ADAM_BETA2 * model->adam_v[cls][feature] +
                                              (1 - ADAM_BETA2) * (model->gradients[cls][feature] * model->gradients[cls][feature]);

                // Bias correction
                double m_hat = model->adam_m[cls][feature] / (1 - pow(ADAM_BETA1, model->adam_t));
                double v_hat = model->adam_v[cls][feature] / (1 - pow(ADAM_BETA2, model->adam_t));

                // Update weights
                model->weights[cls][feature] -= ADAM_LEARNING_RATE * m_hat / (sqrt(v_hat) + EPSILON);
            }
        }

        // epochta geçen süreyi kaydetme
        end = clock();
        elapsed_time = (double)(end - start) / CLOCKS_PER_SEC;
        model->logs[4][epoch] = elapsed_time;

    }
    // modele ait metriklerin csv olarak kaydedilmesi
    save_logs(model,ADAM_LOG);
}
```

- train_model_adam fonksiyonu ile model adam algoritması ile eğitilir. Modele ait weightler ve metrikler her epochta kaydedilir. Algoritma çalışmaya başlamadan önce adam spesifik parametrelerin daha önce başlatılıp başlatılmadığını kontrol eder. Daha sonrasında her epochta compute_gradients ile gradyanlar hesaplanır. Daha sonra bu gradyanlar kullanılarak mt ve vt değerleri olarak bilinen first moment ve second moment değerleri hesaplanır. Bunlara bias correction uygulandıktan sonra modele ait weightler güncellenir. Böylece her weight için daha önceki türevlerinin ağırlıklı ortalaması kullanılarak hızlandırılma veya yavaşlatma uygulanır. Bu şekilde model hem daha hızlı hem de daha stabil bir şekilde converge eder.

2.6 – Yan Fonksiyonlar

```c
// modelin test kümesi üzerindeki doğruluğunu hesaplayan fonksiyon
double calculate_accuracy_test(Model *model){
    int i;
    int num_correct = 0;
    for(i = 0; i < TEST_COUNT; i++) {
        int prediction = predict(model, model->test_data[i]);
        if(prediction == model->test_labels[i]) {
            num_correct++;
        }
    }
    float accuracy = (double) num_correct / TEST_COUNT * 100.0;
    return accuracy;
}

// modelin eğitim kümesi üzerindeki doğruluğunu hesaplayan fonksiyon
double calculate_accuracy_train(Model *model){
    int num_correct = 0,i;
    for(i = 0; i < TRAIN_ROWS; i++) {
        int prediction = predict(model, model->train_data[i]);
        if(prediction == model->train_labels[i]) {
            num_correct++;
        }
    }
    double accuracy = (double) num_correct / TRAIN_ROWS * 100.0;
    return accuracy;

}

// modelin eğitim kümesi lossunu hesaplayan fonksiyon
double compute_train_loss(Model *model){
    int row;
    double actual_total_loss = 0.0;
    for (row = 0; row < TRAIN_ROWS; row++) {
        predict(model,model->train_data[row]);
        actual_total_loss += compute_loss(model->probabilities, model->train_labels[row]);
    }

    actual_total_loss = (double) actual_total_loss / TRAIN_ROWS;

    return actual_total_loss ;
}

// modelin test kümesi lossunu hesaplayan fonksiyon
double compute_test_loss(Model *model){
    int row;
    double actual_total_loss = 0.0;
    for (row = 0; row < TEST_COUNT; row++) {
        predict(model,model->test_data[row]);
        actual_total_loss += compute_loss(model->probabilities, model->test_labels[row]);
    }

    actual_total_loss = (double) actual_total_loss / TEST_COUNT;

    return actual_total_loss;
}
```

- Her epochta modele ait train_loss, test_loss, train_Accuracy ve test_accuracy değerlerinin hesaplanmasını sağlayan fonksiyonlar.

```c
// modele ait metrikleri csv olarak kaydeden fonksyion
void save_logs(Model *model, char *label){
    FILE *loss_file = fopen(label, "w");
    int i;
    fprintf(loss_file, "Train_Loss,Train_Accuracy,Test_Loss,Test_Accuracy,Time_Elapsed\n");


    for(i=0; i<EPOCHS; i++){

        fprintf(loss_file, "%lf,%lf,%lf,%lf,%lf\n",
                model->logs[0][i], model->logs[1][i], model->logs[2][i], model->logs[3][i], model->logs[4][i]);

    }

    fclose(loss_file);
}

// modelin her epochtaki weightlerini csv olarak kaydeden fonksiyon
void save_weights_log(Model *model, const char *filename, int epoch) {
    FILE* fp_weight = fopen(filename,"a");
    int i, j;
    if(epoch==0){
        fprintf(fp_weight,"epoch,", epoch);
        for(i=0; i < NUM_CLASSES; i++){
            for(j=0; j<INPUT_SIZE;j++){
                fprintf(fp_weight,"w%d%d,",i,j);
            }
        }
        fprintf(fp_weight, "\n");
    }

    fprintf(fp_weight,"%d,", epoch);
    for(i = 0; i < NUM_CLASSES; i++){
        for(j = 0; j < INPUT_SIZE; j++){
            fprintf(fp_weight,"%lf,", model->weights[i][j]);
        }
    }
    if(epoch != EPOCHS-1)
        fprintf(fp_weight, "\n");  // Her sınıf için yeni satır
    fclose(fp_weight);
}
```

- Modele ait metriklerin ve weightlerin dosyaya kaydedilip daha sonrsında görselleştirme için kullanılmasını sağlayan fonksiyonlar.

2.7 – Görselleştirme Kodları

```python
# CSV dosyalarını okuma
gd_data = pd.read_csv("gd_logs_m.csv")
sgd_data = pd.read_csv("sgd_logs_m.csv")
adam_data = pd.read_csv("adam_logs_m.csv")

# Epoch sayısını bulma
epochs = range(1, len(gd_data)+ 1)

# Zamanı sıfırdan başlatma
gd_data["Time_Elapsed"] -= gd_data["Time_Elapsed"].iloc[0]
sgd_data["Time_Elapsed"] -= sgd_data["Time_Elapsed"].iloc[0]
adam_data["Time_Elapsed"] -= adam_data["Time_Elapsed"].iloc[0]
```

- Modele ait metriklerin kaydedildiği dosyalardan pandas aracılığı ile csv dosyasından okunması. Oluşan dataframedeki satır sayısını bularak epoch sayısını hesaplanması. Loss ve Accuracy zaman grafikleri için zamanın 0dan başlatılması için düzeltme yapılması.

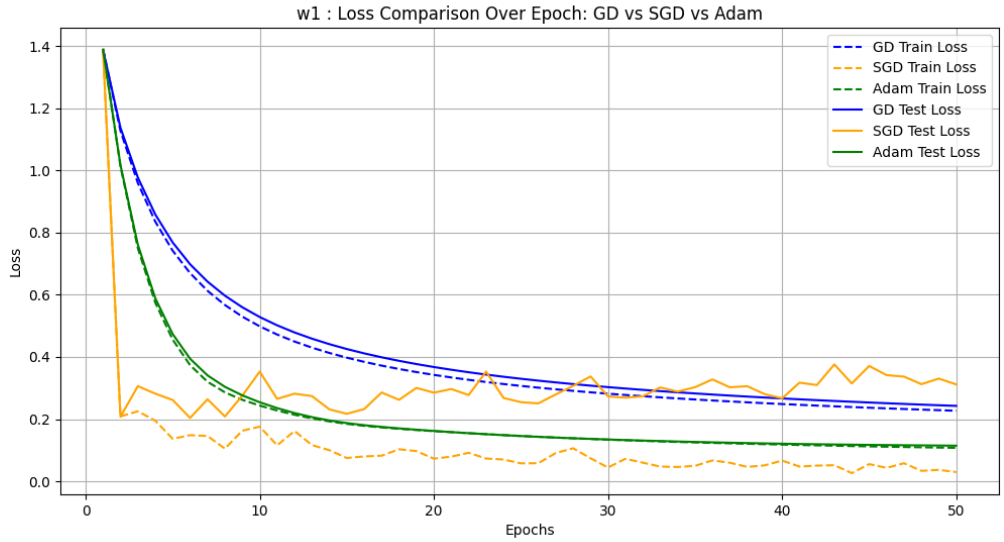
```python
# Grafik oluşturma
plt.figure(figsize=(12, 6))

# Train Accuracy grafikleri
plt.plot(epochs, gd_data["Train_Accuracy"], label="GD Train Accuracy", linestyle="--", color="blue")
plt.plot(epochs, sgd_data["Train_Accuracy"], label="SGD Train Accuracy", linestyle="--", color="orange")
plt.plot(epochs, adam_data["Train_Accuracy"], label="Adam Train Accuracy", linestyle="--", color="green")

# Test Accuracy grafikleri
plt.plot(epochs, gd_data["Test_Accuracy"], label="GD Test Accuracyss", color="blue")
plt.plot(epochs, sgd_data["Test_Accuracy"], label="SGD Test Accuracy", color="orange")
plt.plot(epochs, adam_data["Test_Accuracy"], label="Adam Test Accuracy", color="green")

# Grafik ayarları
plt.title("Accuracy Comparison Over Epoch: GD vs SGD vs Adam")
plt.xlabel("Epochs")
plt.ylabel("Accuracy(%)")
plt.legend()
plt.grid(True)

# Grafiği kaydetme ve gösterme
plt.savefig("Accuracy_Epoch_full_m.png")
plt.show()
```

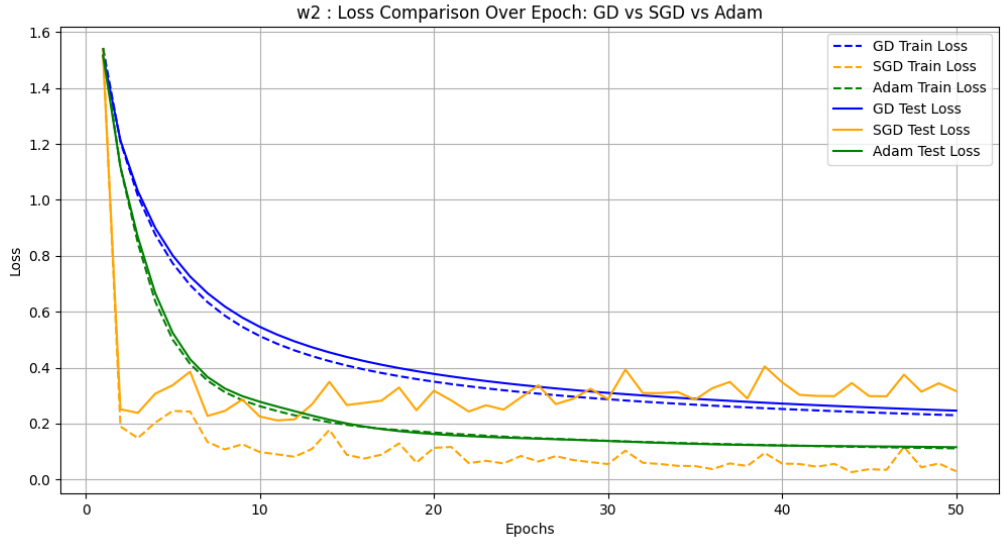
- Okunan csv dosyasındaki değerlere göre matplotlib aracılığı ile grafiklerin oluşturulması.

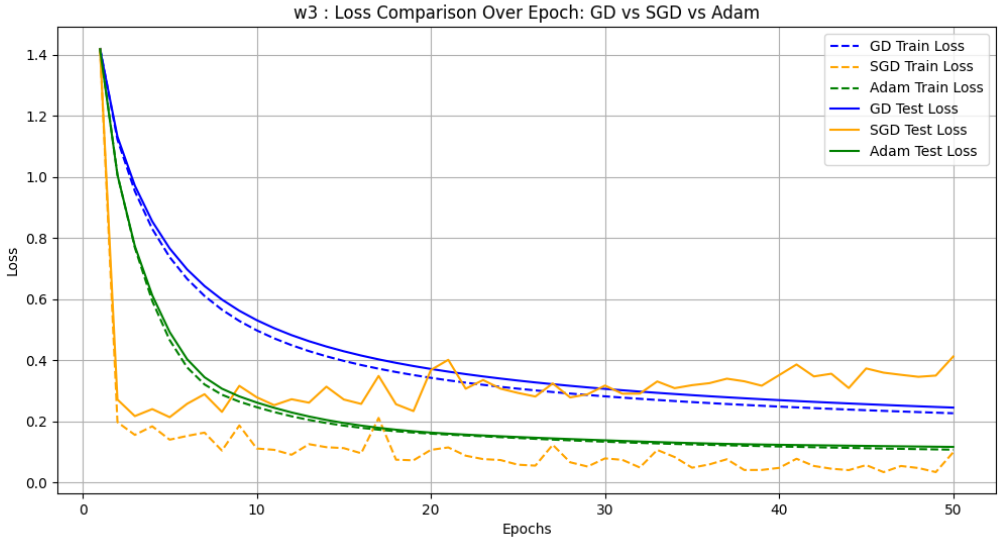
3 – Grafiklerin Yorumlanması

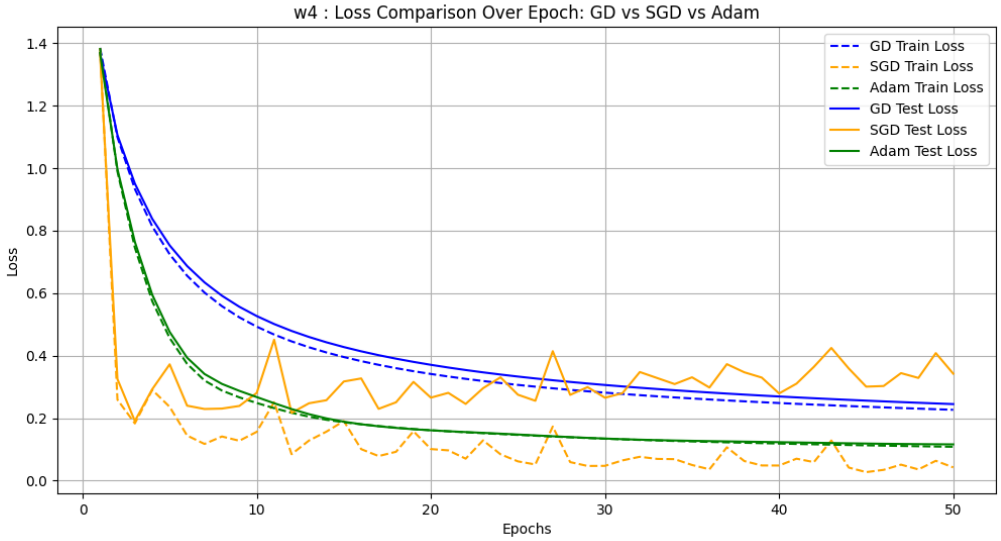
- 5 farklı initial weight vektör değeri için açıklanan kodlar çalıştırıldı ve 4 metriğe göre topam 4*5 farklı grafik elde edildi. Burada bu grafiklere bakılarak gerekli bilgiler çıkarılacaktır. Ardından her algoritmanın t sen grafikleri yorumlanacaktır.

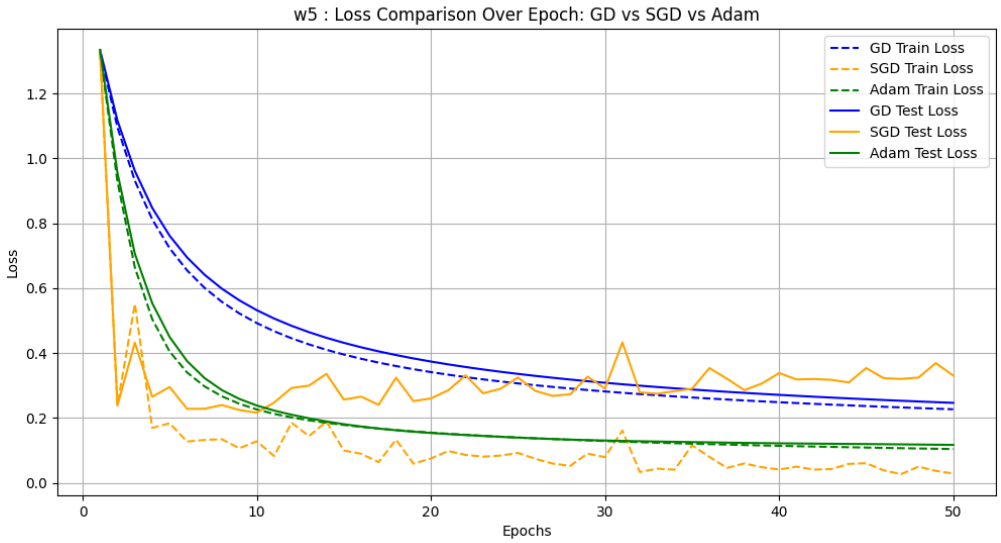
3.1 – LOSS-Epoch Grafikleri





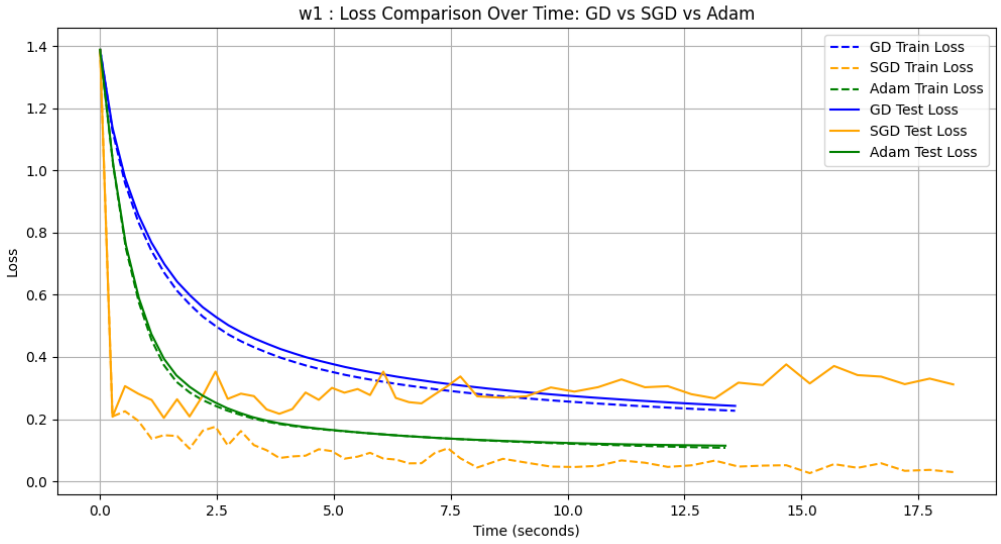

w3 : Loss Comparison Over Epoch: GD vs SGD vs Adam
GD Train Loss
SGD Train Loss
Adam Train Loss
GD Test Loss
SGD Test Loss
Adam Test Loss
Loss
Epochs

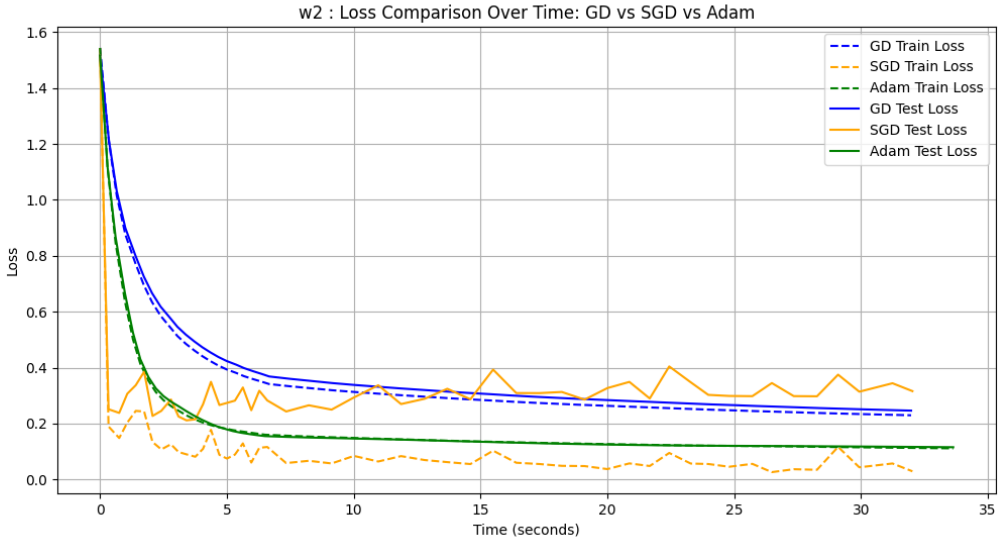

w4 : Loss Comparison Over Epoch: GD vs SGD vs Adam
GD Train Loss
SGD Train Loss
Adam Train Loss
GD Test Loss
SGD Test Loss
Adam Test Loss
Loss
Epochs

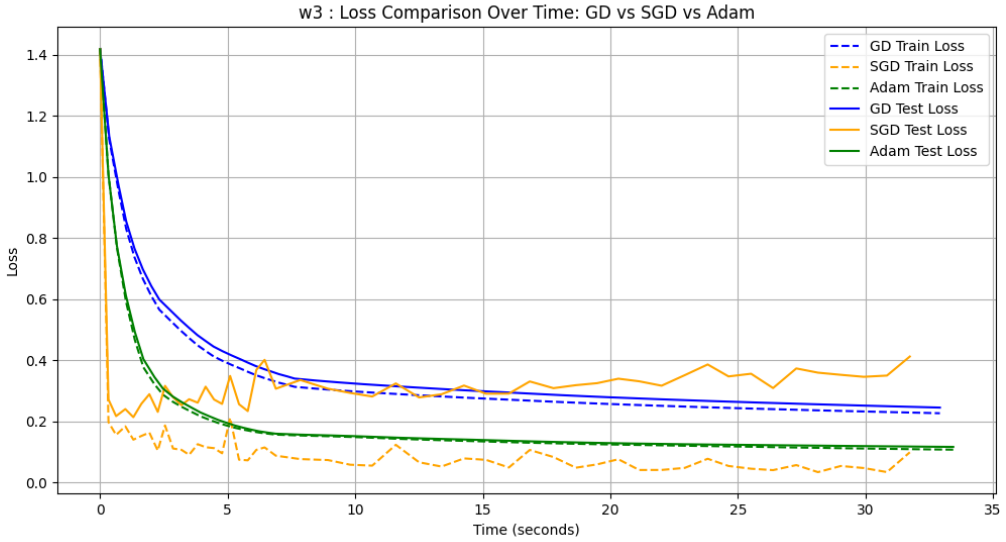

w5 : Loss Comparison Over Epoch: GD vs SGD vs Adam
GD Train Loss
SGD Train Loss
Adam Train Loss
GD Test Loss
SGD Test Loss
Adam Test Loss
Loss
Epochs

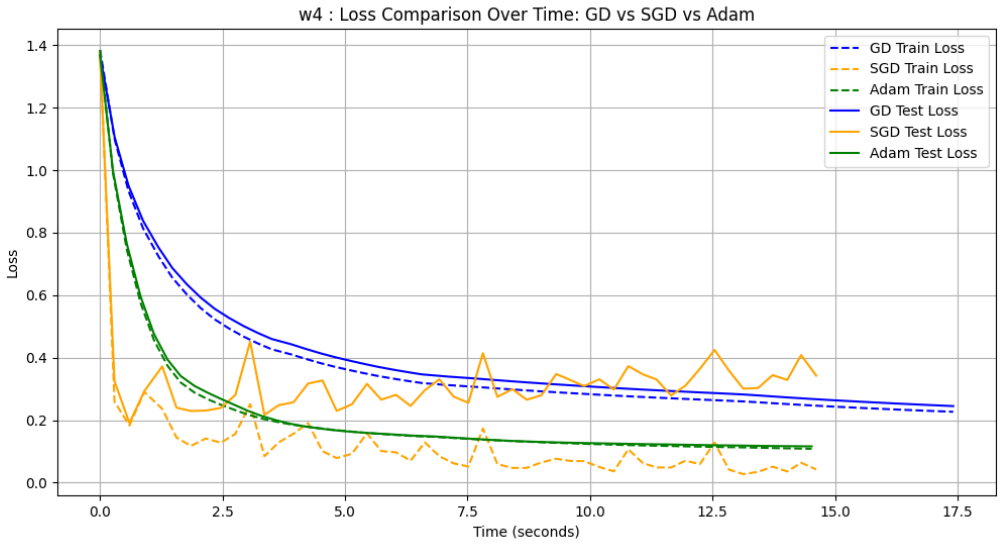
- Görüldüğü üzere 5 farklı initial weight vektörü için 5 farklı grafik elde edildi.
- Bu grafiklerde açıkça görüldüğü üzere SGD algoritması her epochta çokça güncelleme yaptığından dolayı hızlıca lossunu düşürebilmektedir ancak daha sonra loss belirli bir aralığa geldiğinde bu bölgede salınım yapmakta ve bazı grafiklerde bu aralıktan uzaklaşmaya başlamaktadır. Özellikle yüksek veri örneğine sahip olduğumuz için SGD algoritması veri örneği kadar güncelleme almaktadır bu yüzden bu denli kararsız davranmaktadır. Learning rate düşürülmesi ile daha stabil bir yakınsama elde edilebilir.
- Gradient descent ve adam algoritması ise neredeyse her initial weight için benzer davranışı göstermektedir. İki algoritma da yumuşak bir şekilde lossunu düşürmektedir. Adam algoritması adaptaif momentum özelliklerini kullandığı için Gradient Descentin sabit adım sayısına üstünlük kurmaktadır ve loss'u daha iyi bir şekilde düşürebilmektedir. Yine örnek veri sayısı, learning rate ve adam parametreleri değiştirildiğinde farklı yaklaşımlar elde edilebilir. Örneğin adam learning rate 0.001 yerine 0.01 yapıldığında zikzak çizmeye başlamaktadır.

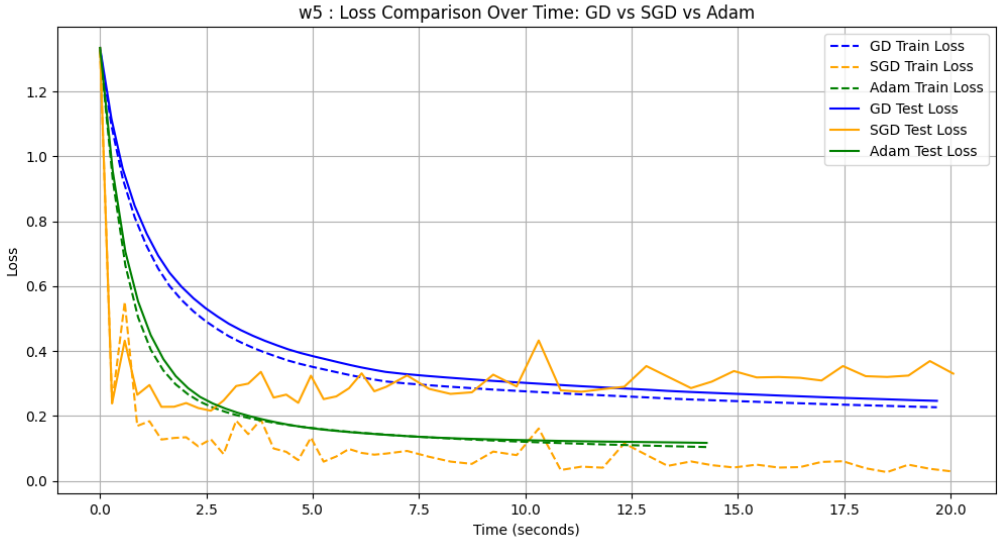
3.2 – LOSS-Time Grafikler




w2 : Loss Comparison Over Time: GD vs SGD vs Adam
GD Train Loss
SGD Train Loss
Adam Train Loss
GD Test Loss
SGD Test Loss
Adam Test Loss
Loss
Time (seconds)

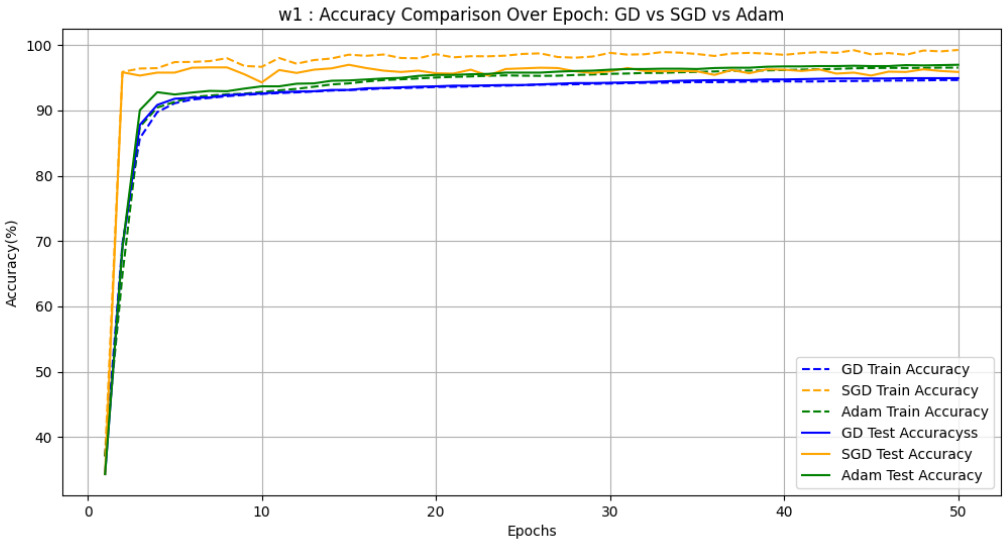

w3 : Loss Comparison Over Time: GD vs SGD vs Adam
GD Train Loss
SGD Train Loss
Adam Train Loss
GD Test Loss
SGD Test Loss
Adam Test Loss
Loss
Time (seconds)

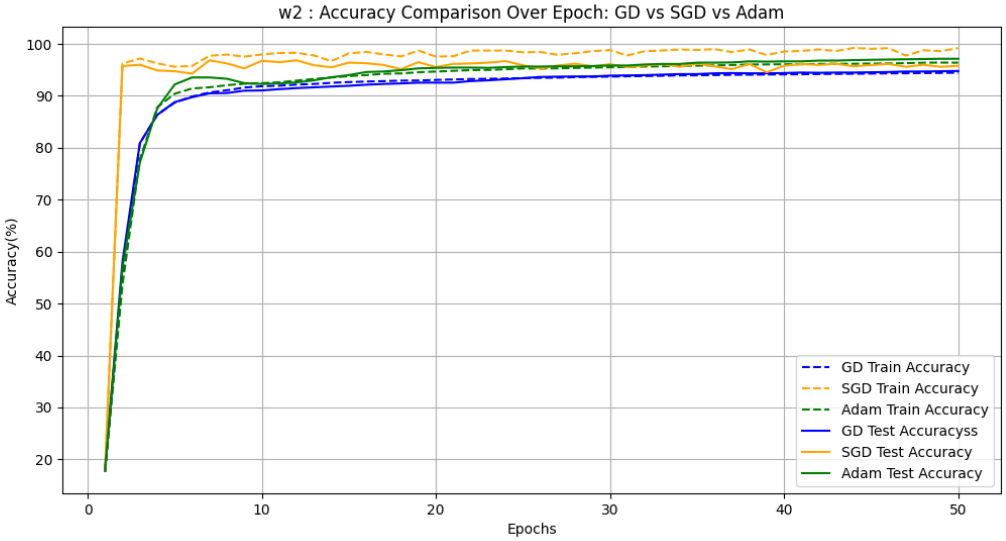


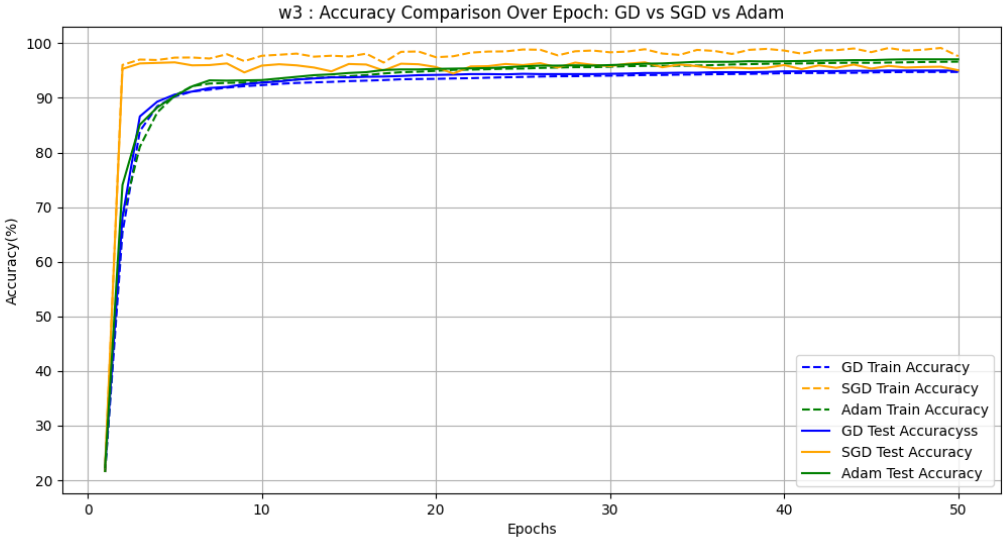


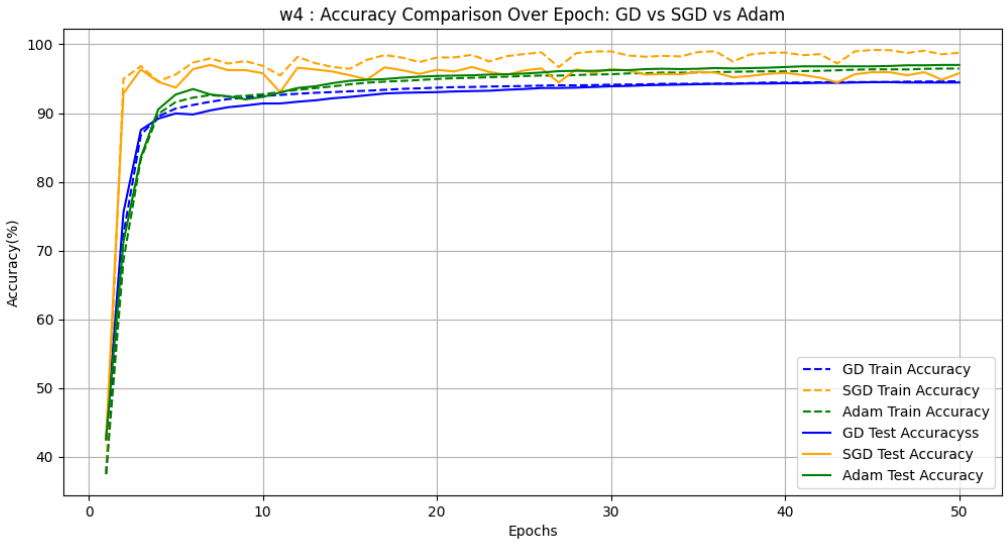
- Loss – Time grafiklerine bakarak algoritmalar hakkında loss-epoch grafiklerinde yapılan çıkarımlar yapılabilir.
- Ancak bu grafiklerde görüldüğü üzere bazı algoritmalar aynı epoch sayısına sahip olmalarına rağmen farklı süreler çalışmasını bitirdiği görülebiliyor. Yapılan train verisinin karıştırılma işlemindeki rastgelelik ve sgd algoritmasının her adımda farklı bir örnek seçerek ilerlemesi başlangıç değerleri ve karıştıma işleminin algoritmanın yakınsama hızını etkilediği görülebilmektedir.

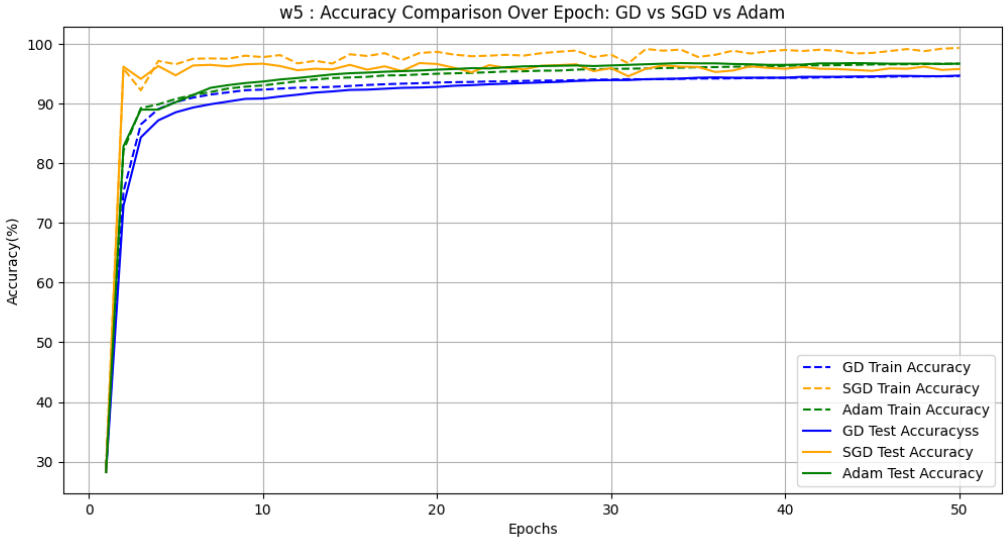
3.3 – Accuracy-Epoch Grafikleri


w1 : Accuracy Comparison Over Epoch: GD vs SGD vs Adam
Accuracy(%)
Epochs
GD Train Accuracy
SGD Train Accuracy
Adam Train Accuracy
GD Test Accuracyss
SGD Test Accuracy
Adam Test Accuracy


w2 : Accuracy Comparison Over Epoch: GD vs SGD vs Adam
Accuracy(%)
Epochs
GD Train Accuracy
SGD Train Accuracy
Adam Train Accuracy
GD Test Accuracyss
SGD Test Accuracy
Adam Test Accuracy


w3 : Accuracy Comparison Over Epoch: GD vs SGD vs Adam
Accuracy(%)
Epochs
GD Train Accuracy
SGD Train Accuracy
Adam Train Accuracy
GD Test Accuracyss
SGD Test Accuracy
Adam Test Accuracy

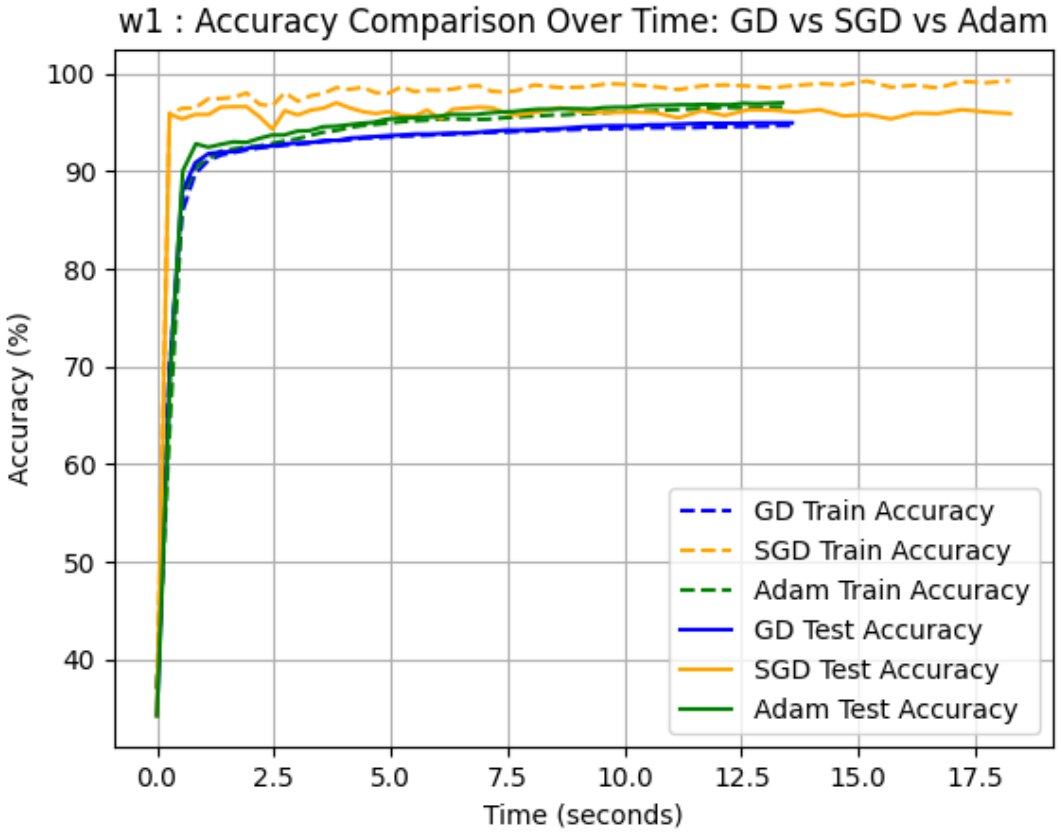


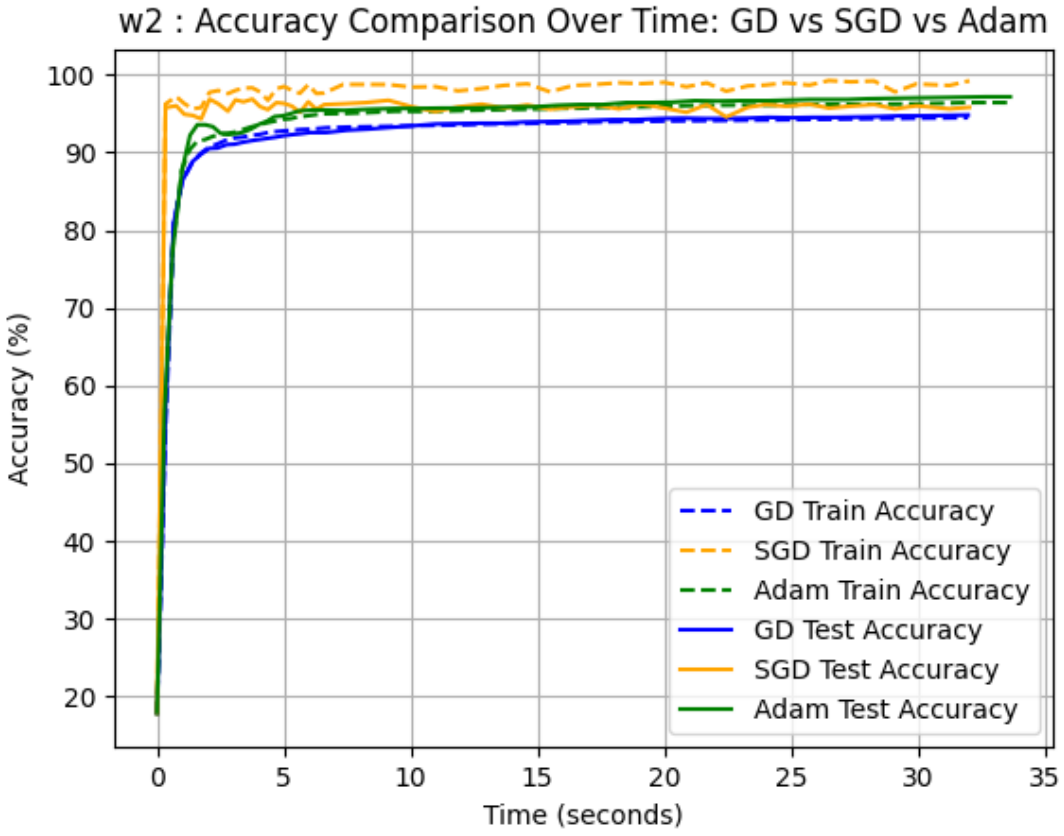


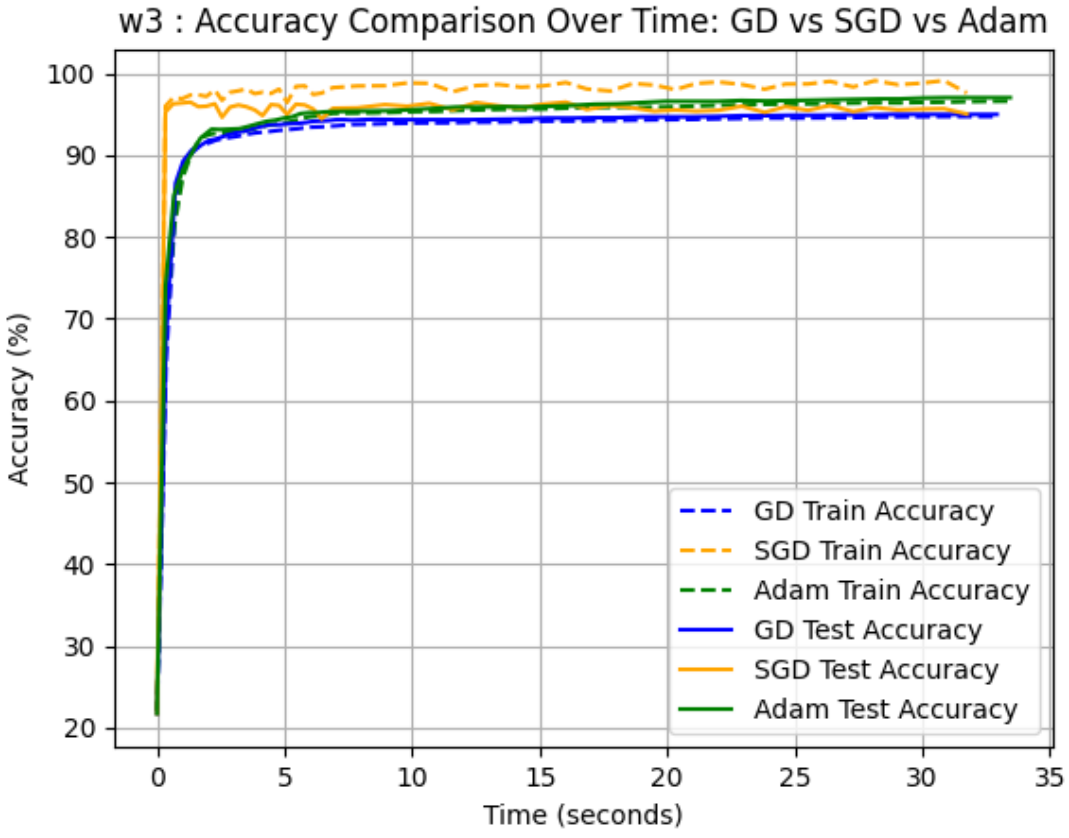
- Accuracy time grafiklerine bakıldığında her 3 algoritmanında 0-10 epoch arasında hem test hem de train kümesi üzerinde %90 üzerinde doğruluğa çıktığı görüşmektedir.
- SGD algoritması yine her epochta veri örneği kadar güncelleme yaptığı için bir yerden sonra eğitim kümesine overfit olmaktadır, ilk 5 epochtan sonra test doğruluğunun da salınım yaptığını ve epoch sayısı arttıkça test doğruluğu liderliğini Adam algoritmasını kaptırdığı görülmektedir.
- Adam algoritması da ilk epochlardan sonra ufak bir salınım yaptıktan sonra hem eğitim hem de test kümesi üzerindeki doğruluğunu arttırmaya devam ettirmektedir ve belirli bir epoch sayısından sonra SGD algoritmasından daha iyi bir doğruluk değerine ulaşmaktadır.
- Yine veri setinin büyüklüğünden dolayı belirli bir eşik değerden sonra model ne kadar eğitilirse eğitilsin performansını arttıramamaktadır. Daha küçük veri setleri ile deneme yaptığımda SGD algoritmasının test kümesi üzerinde %100 doğruluk oranlarına ulaştığını gördüm ve adam algoritmasının da belirli bir süre sonra aynı davranışını gösterdiğini gözlemledim.

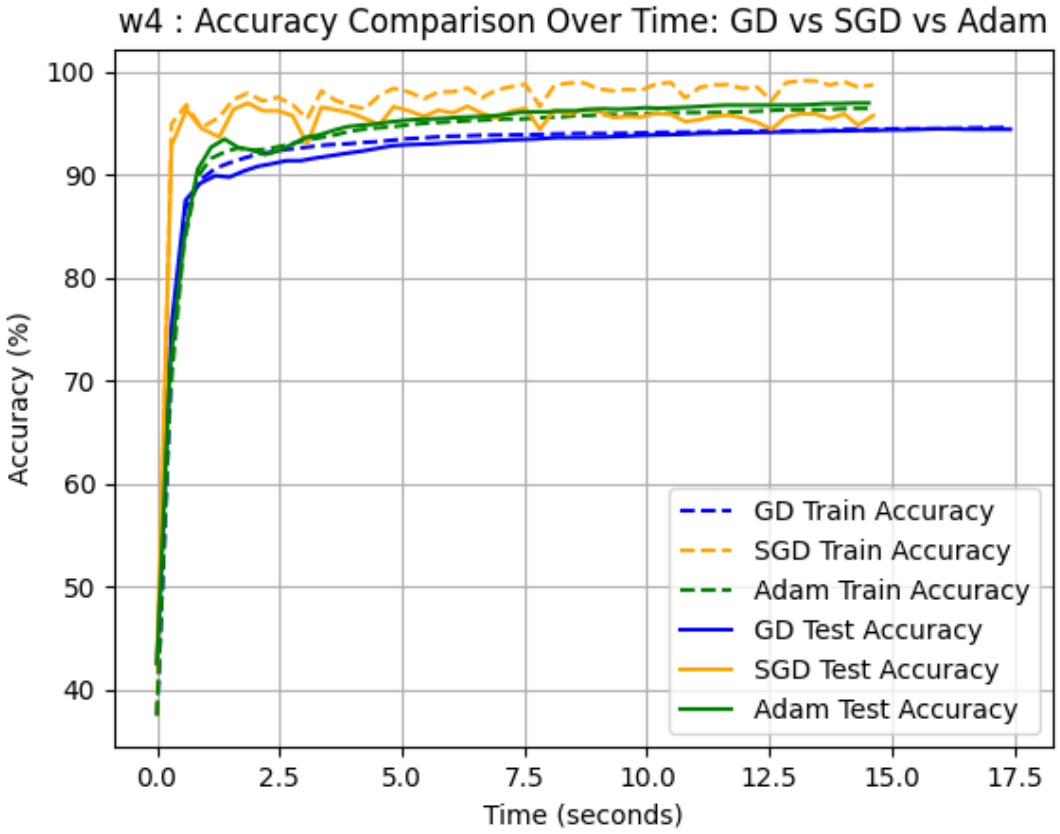
- Yine bu grafiklere bakarak Adam algoritmasının Gradient descent algoritması üzerindeki üstünlüğü görebilmekteyiz.

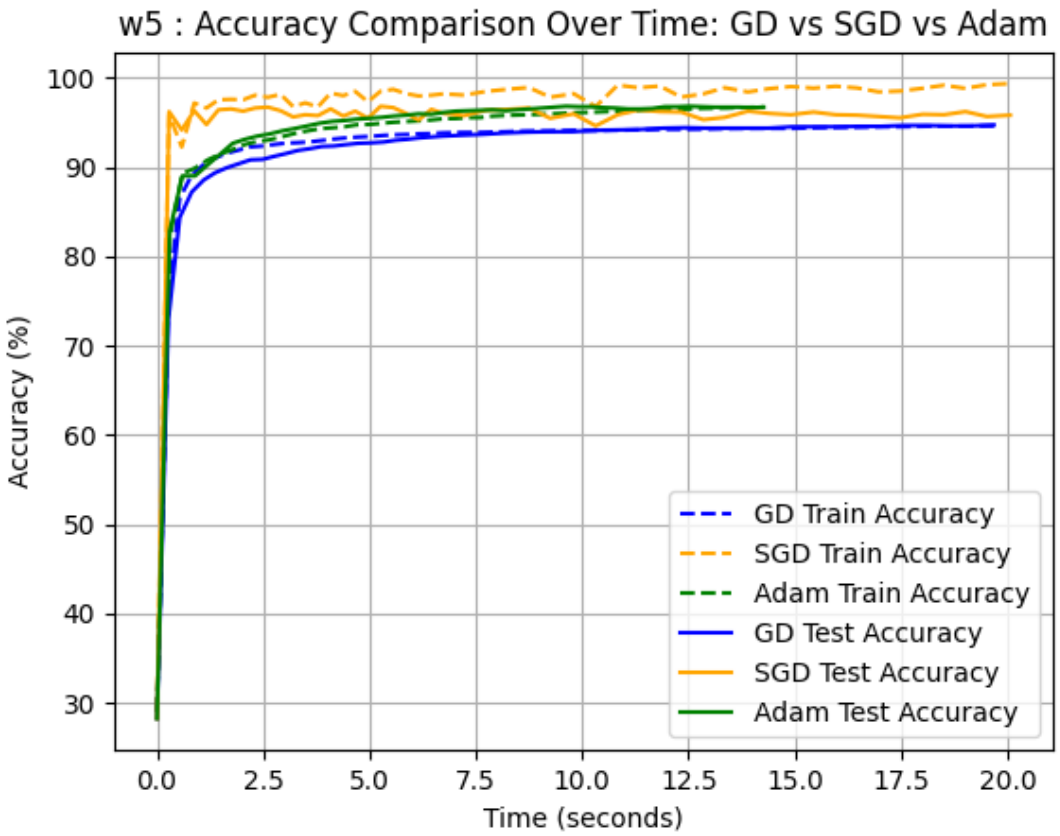
3.4 – Accuracy-Time Grafikleri




w2 : Accuracy Comparison Over Time: GD vs SGD vs Adam
Accuracy (%)
Time (seconds)
GD Train Accuracy
SGD Train Accuracy
Adam Train Accuracy
GD Test Accuracy
SGD Test Accuracy
Adam Test Accuracy

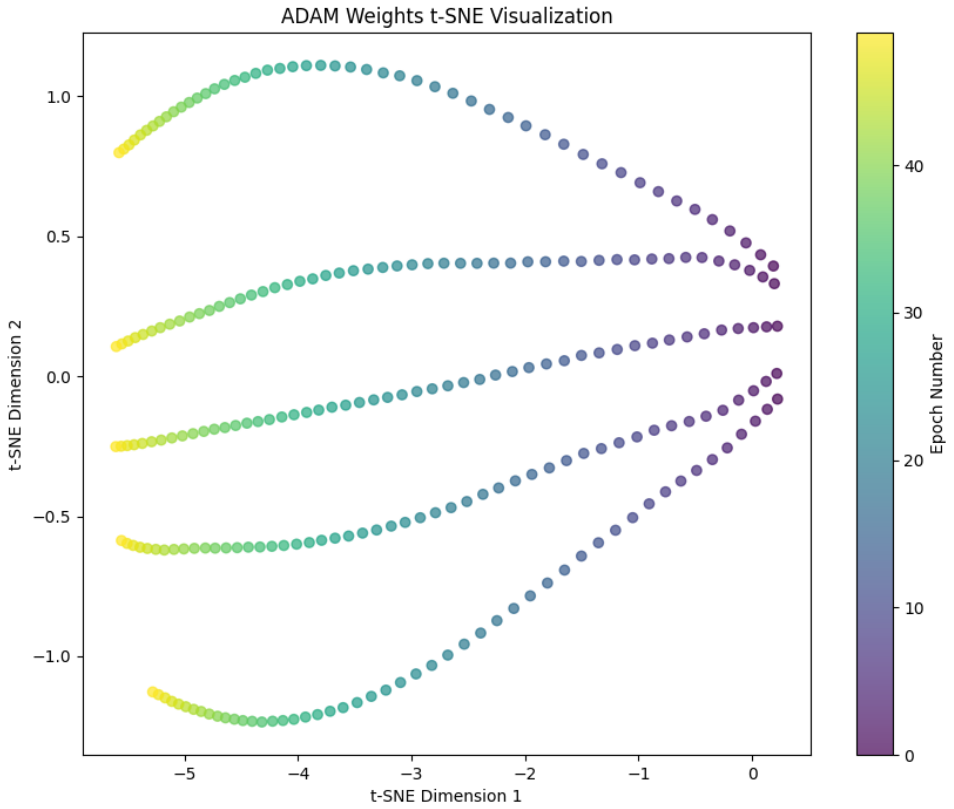

w3 : Accuracy Comparison Over Time: GD vs SGD vs Adam
Accuracy (%)
Time (seconds)
GD Train Accuracy
SGD Train Accuracy
Adam Train Accuracy
GD Test Accuracy
SGD Test Accuracy
Adam Test Accuracy


w4 : Accuracy Comparison Over Time: GD vs SGD vs Adam
Accuracy (%)
Time (seconds)
GD Train Accuracy
SGD Train Accuracy
Adam Train Accuracy
GD Test Accuracy
SGD Test Accuracy
Adam Test Accuracy


w5 : Accuracy Comparison Over Time: GD vs SGD vs Adam
Accuracy (%)
Time (seconds)
GD Train Accuracy
SGD Train Accuracy
Adam Train Accuracy
GD Test Accuracy
SGD Test Accuracy
Adam Test Accuracy

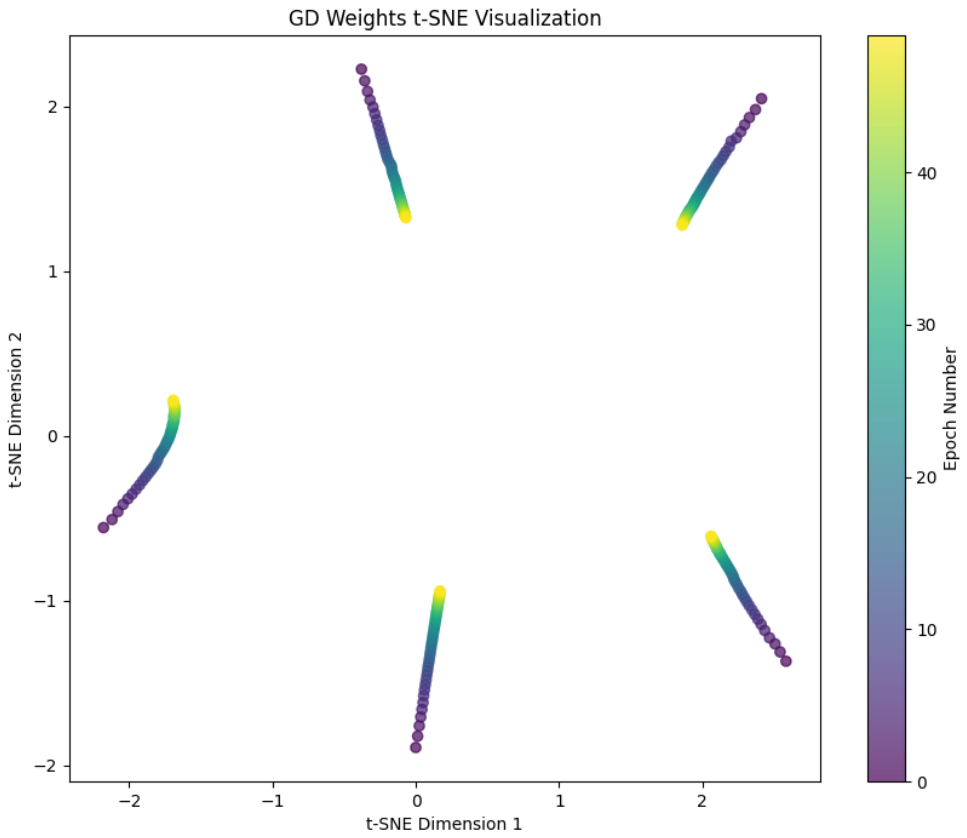
- Accuracy zaman grafiklerine baktığımızda da SGD algoritmasının çok hızlı bir şekilde yüksek doğruluk oranlarına ulaştığını görebiliyoruz. Ancak süre arttıkça adam algoritmasının SGD algoritmasının salınım yapmasından faydalanarak öne geçtiği görülebiliyor.
- Sonuçta böyle bir veri kümesi üzerinde ve örnek sayısına bakılarak 3 algoritmanın da iyi bir iş çıkardığı gözlemlenebilmektedir. Ancak SGD algoritması her ne kadar hızlı bir şekilde belirli bir sonuca ulaşsa da belirli bir yerden sonra sadece salınım yapmaktadır ve yakınsamak yerine zik zak çizmektedir. Ancak adam ve gd algoritmalarının çalıştırılma süresi arttıkça yavaş bir şekilde de olsa doğruluk oranlarını arttırabildikleri görülmektedir. Bu iki algoritma arasında da adam algoritmasının üstünlüğü göze çarpmaktadır.

3.5 – Adam T-SNE Grafiği



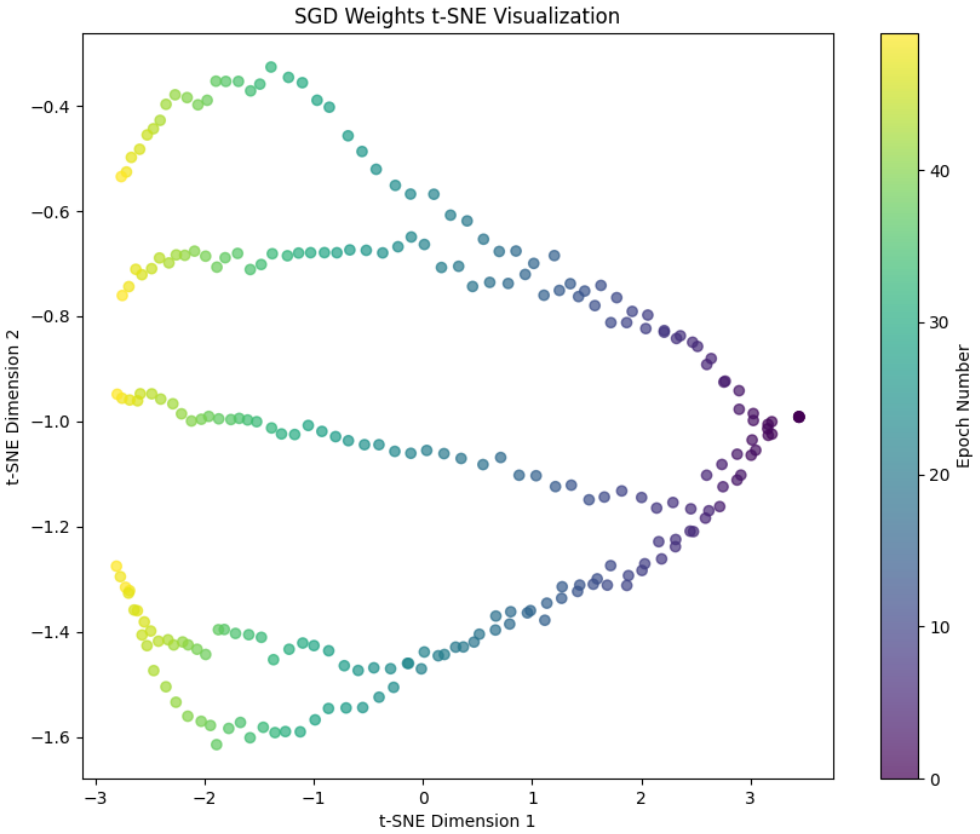
- Bu T-SNE grafiğine ve epoch sayısına göre görselleştirmeye bakılarak öncelikle başlangıç weightslerinin belirli bir yerde toplandığı daha sonra model eğitimi başladıkça başlangıç değerlerine göre 5 farklı yörünge oluştuğu görülmektedir. Yörüngelerin yönelimine bakarak adam algoritması ile eğitim sürecinde parametrelerin benzer süreçleri takip ettikleri ancak başlangıç değerlerinin modelin farklı bir yoldan öğrenmeye götürdüü görülebilmektedir.
- Ayrıca yörüngelere bakıldığına noktalar arasındaki mesafelerin önce giderek açıldığı daha sonra bu noktaların tekrar birbirine yaklaştığı görülmektedir. Bu da adam

3.6 – Gradient Descent T-SNE Grafiği



- Bu grafiğe bakıldığında ise Gradient Descent algoritması ile model eğitimi sürecinde parametrelerin ortak bir minimuma doğru ilerlediği görülmektedir. Yörüngeler üzerindeki renk geçişlerine de bakarak learning ratenin arttırılması gerektiği çıkarımı yapılabilmektedir. Çünkü noktalar birbirine yapışık olarak görülmektedir, adam grafiğinde görüldüğü gibi learning rate'nin arttırılması bu noktalar arası mesafenin açılacağını ve böylece daha hızlı bir yakınsama olabileceğini gösteriyor.

3.7 – Stochastic Gradient Descent T-SNE Grafiği



- SGD T-SEN grafiğine baktığımızda ise yine aynı şekilde parametrelerin belirli bi aralıkta başlayarak takip ettiği yörüngeye göre önce uzaklaştığı daha sonra ortak bir minimuma doğru yöneldiği anlaşılabilmektedir.
- Grafikteki noktaların dağılımına bakarak da SGD içindeki rastgele örnek seçiminin modelin ilerleyiş yörüngesini etkilediği görülmektedir.
- Ayrıca noktalar arası mesafelerin değişimi ve dağılımına bakılarak Adam ve GD grafiklerindekinden daha sert hareketlenmeler olduğu görülebilmektedir. Bu da her epochta eğitim kümesininin tamamının görülmesi ve her örnekte güncelleme yapılması ile açıklanabilir.

4 – Sonuç

Sonuç olarak bakıldığında Adam algoritmasının adaptif momentum özelliği özellikle dikkat çekmektedir ve kendisini ön plana çıkarmaktadır. Ayrıca modeller için hiperparametre ayarlarının doğru yapılmasının önemi de anlaşılmaktadır çünkü T-SEN grafikleri ve metrik grafiklerine bakıldığında özellikle Learning Rate değiştirilerek kayda değer değişiklikler gözlenebilir. Ayrıca 4 sınıflı bir veri seti yerine daha yüksek sayıda sınıfa sahip bir kümede eğitim yapma, eğitim kümesininin büyüklüğü de metrikleri kayda değer şekilde etkilemektedir.

5 – Kaynakça

- https://towardsdatascience.com/derivative-of-the-softmax-function-and-the-categorical-cross-entropy-loss-ffceefc081d1
- https://jmlb.github.io/ml/2017/12/26/Calculate_Gradient_Softmax/
- https://jamhuri.medium.com/understanding-the-adam-optimization-algorithm-a-deep-dive-into-the-formulas-3ac5fc5b7cd3#:~:text=The%20Adam%20algorithm%20is%20a,to%20the%20choice%20of%20hyperparameters.
- https://www.geeksforgeeks.org/adam-optimizer/
- https://math.stackexchange.com/questions/3993037/computing-the-gradient-of-cross-entropy-loss
- https://www.quora.com/What-is-the-difference-between-epochs-batches-and-iterations-when-it-comes-to-deep-learning-neural-network-training-What-is-the-best-way-to-count-them-all-together-if-at-all
- https://www.mosismath.com/AI/BackPropagation.html