



Yıldız Teknik Üniversitesi  
Bilgisayar Mühendisliği Bölümü

## Optimizasyon Projesi Raporu

Mehmet Ali Gökçay

22011052

[ali.gokcay@std.yildiz.edu.tr](mailto:ali.gokcay@std.yildiz.edu.tr)

Recep Karakaya

22011068

[recep.karakaya@std.yildiz.edu.tr](mailto:recep.karakaya@std.yildiz.edu.tr)

## Ön Bilgi

Bu projenin amacı, Gradient Descent, Stochastic Gradient Descent ve Adam gibi farklı optimizasyon algoritmalarının, bir görüntü sınıflandırma problemindeki performanslarını analiz etmektir. Analiz, her epoch için model doğruluğu, kayıp değerleri (eğitim ve test), eğitim süreleri ve doğruluk oranları gibi kriterler üzerinden yapılmıştır. Ayrıca, proje kapsamında **bonus** olan 4 sınıfı bir veri kümesi üzerinde sınıflandırma işlemi de gerçekleştirilmiş ve detaylı olarak incelenmiştir. Öncelikle iki sınıfı sınıflandırma istenilen şekilde ele alınmış, ardından dört sınıfı sınıflandırmaya geçilmiştir. İki çıkışlı modelimiz, el yazısıyla yazılmış “0” ve “1” rakamlarını tanıma ve ayırt etme işlemini gerçekleştirir. 4 çıkışlı modelimiz “2” ve “3” rakamlarını da tanır ve ayırt eder.



Videoyu izlemek için buraya tıklayın: <https://youtu.be/MXfbPaEPHFc>

Bütün kod dosyalarını incelemek için buraya tıklayın:

[https://drive.google.com/drive/folders/1kVCZFe-ct\\_ZP2NgHTYtpFXJGotJdLg3m?usp=sharing](https://drive.google.com/drive/folders/1kVCZFe-ct_ZP2NgHTYtpFXJGotJdLg3m?usp=sharing)

## Görüntü Sınıflandırma

Veri Kümesi Hazırlama:

```
void process_images_in_directory(const char *input_dir, const char *output_dir) {
    struct dirent *entry;
    DIR *dp = opendir(input_dir);

    if (dp == NULL) {
        perror("Dizin açılamadı");
        return;
    }

    // Dizin içinde gezmek için
    while ((entry = readdir(dp)) != NULL) {
```

C programlarında dizinleri okuma ve dosya girişlerini listeleye işlevlerini sağlayan bir kütüphane olan dirent.h kütüphanesini kodumuzun neredeyse her yerinde kullandık.

While döngüsü ile dizindeki her bir resmi, `stb_image.h` ve `stb_image_write.h` kütüphanelerini kullanarak yükliyor ve işliyoruz. Resimlerin boyutlarını  $N \times N$  olacak şekilde yeniden boyutlandırıyoruz ve ardından gri tonlara dönüştürüyoruz. Bu işlem, farklı boyutlardaki resimleri modele uygun hale getirirken, renkli resimleri de modelde kullanabilmemizi sağlıyor. Sonrasında, bu gri tonlu görüntülerini normalize edip bir vektöre dönüştürüyoruz. Bu vektörleri, belirlediğimiz dosya adlandırma kurallarına göre bir `.txt` dosyasına kaydediyoruz. Klasör adlarına bağlı olarak dosyalar isimlendirilerek, düzenli train ve test setleri oluşturuluyor.

`process_images.c` kodunu incelemek için buraya tıklayın:

<https://drive.google.com/file/d/1cPMkA0XFjl7A0AwfozVaxcffK8rF6rs/view?usp=sharing>

`initialize_weights` fonksiyonu, ağırlık değerlerini  $(-1, 1)$  aralığında rastgele başlatır.

```
void initialize_weights() {
    int i;
    for (i = 0; i < N * N + 1; i++) {
        //w_first[i] = 0;
        w_first[i] = ((float)rand() / RAND_MAX) * 2.0f - 1.0f;
    }
}
```

## Gradient Descent

```
void gradient_descent(const char *dir_path, const char *dir2_path) {
    struct dirent *entry;
    DIR *dp = opendir(dir_path);
    int epoch,i,current_index;
    float train_losses[EPOCHS], train_accuracies[EPOCHS],train_times[EPOCHS];
    float test_losses[EPOCHS], test_accuracies[EPOCHS];
    clock_t start, end;
    float epoch_time,total_time=0;

    FILE *w_record = fopen("results/w_gd.txt", "w");
    if (w_record == NULL) {
        printf("Dosya açılamadı.\n");
        return;
    }

    float w[N*N+1], learning_rate = 0.001f;
    for (i = 0; i < N * N + 1; i++) {
        //w[i]=0;
        w[i] = w_first[i];
        fprintf(w_record, "%f ", w[i]);
    }
    fprintf(w_record, "\n");

    if (dp == NULL) {
        perror("Dizin açılamadı");
        return;
    }
```

- **dir\_path, dir2\_path:** Eğitim ve test veri dosyalarının bulunduğu dizin yollarını alır.
- **w:** Model ağırlıkları, w\_first ile başlatılır.
- **train\_losses, test\_losses:** Her epoch için eğitim ve test kayıplarını saklar.
- **train\_accuracies, test\_accuracies:** Eğitim ve test doğruluk oranları.
- **w\_record:** Modelin ağırlıklarının her epoch sonunda kaydedileceği dosya.
- **learning\_rate:** Gradyan inişi algoritması için öğrenme hızı.

```

for (epoch = 0; epoch < EPOCHS; epoch++) {
    start = clock();
    float mse = 0.0f, True=0, False=0;
    current_index = 0;

    float gradients[N * N + 1] = {0.0f};
    while ((entry = readdir(dp)) != NULL) {
        const char *filename = entry->d_name;
    }
}

```

Bu kısım, her epoch için eğitim döngüsünü başlatır; train süresini ölçmek için zamanlayıcı başlatılır, mse ve doğruluk sayıçları sıfırlanır, gradyanlar sıfırdan başlatılır, ardından dizindeki her dosya her epoch'ta okunarak işlem yapılır.

```

float x_train[N*N] = {0.00f};

for (i = 0; i < N * N; i++) {
    if (fscanf(input_file, "%f", &x_train[i]) != 1) {
        printf("Dosyadan değer okunamadı: %s\n", file_path);
        fclose(input_file);
        exit(-1);
    }
    fclose(input_file);

    int label;
    if(filename[0] == '0'){
        label=-1;
    }else{
        label=1;
    }
}

```

Bu kısım, eğitim verisini dosyadan okuyarak x\_train dizisine yükler. Ardından, dosya adının ilk karakterine göre etiket (label) belirler: '0' için -1, diğer durumlar için 1.

```

float dot_product = 0.0f;
for(i=0; i<N*N; i++){
    dot_product += w[i] * x_train[i];
}
dot_product += w[N*N];
float z = tanh(dot_product);
float error = z - label;

if(z*label>0){
    True++;
}else{
    False++;
}

mse += error * error;

mse += error * error;

for (i = 0; i < N * N; i++) {
    gradients[i] += (2.0f / N) * error * (1 - z * z) * x_train[i];
}
gradients[N * N] += (2.0f / N) * error * (1 - z * z);

current_index++;
}

//mse = sqrt(mse);
mse /= current_index;

for (i = 0; i < N * N + 1; i++) {
    w[i] -= learning_rate * gradients[i];
    fprintf(w_record, "%f ", w[i]);
}
fprintf(w_record, "\n");

```

Bu kısımda, her eğitim örneği için şu işlemler yapılır:

- **Dot Çarpımı Hesaplama:** Girdi verisi ile ağırlıkların çarpımı hesaplanır ve bias eklenir.
- **Aktivasyon Fonksiyonu:** Hesaplanan dot çarpımına tanh fonksiyonu uygulanarak modelin tahmin sonucu  $z$  elde edilir.
- **Hata Hesaplama:** Tahmin ( $z$ ) ile gerçek etiket (label) arasındaki fark hesaplanır.
- **Doğruluk Hesaplama:** Accuracy hesaplamak için tahminin doğru olup olmadığına bakılır.
- **MSE Güncelleme:** Hata karesi toplanarak mse hesaplanır. Döngüden çıkışınca da data boyutuna bölünür.
- **Gradyan Hesaplama:** Ağırlıkların gradyanları, hata ve mse'nin türevi kullanılarak hesaplanır.
- **Ağırlık Güncelleme:** Gradyanlar öğrenme oranı ile çarpılarak ağırlıklar güncellenir.
- **Ağırlıkların Kaydedilmesi:** Güncellenmiş ağırlıklar projenin B kısmı için dosyaya kaydedilir.

```

train_losses[epoch] = mse;
train_accuracies[epoch] = (True / (True + False)) * 100;

end = clock();
epoch_time = ((float) (end - start)) / CLOCKS_PER_SEC;
total_time += epoch_time;
train_times[epoch] = total_time;

rewinddir(dp);

```

Bu kısımda istenilen grafikleri çizdirebilmek için gerekli olan değişkenler hesaplanır ve dizin işaretçisi sıfırlanarak bir sonraki epoch için hazırlık yapılır.

```

float dot_product = 0.0f;
for(i=0; i<N*N;i++){
    dot_product += w[i] * x_test[i];
}
dot_product += w[N*N];
float z = tanh(dot_product);

float error = z - label;
mse_test += error * error;

if(z>0){
    prediction = 1;
}else{
    prediction = -1;
}

if(label==prediction){
    True++;
}else{
    False++;
}

```

Her epoch'un sonunda test kümesine de aynı işlemler uygulanarak modelin doğruluğu ve kaybı hesaplanır. Test kümesinde yapılan işlemlerin tek farkı, elde edilen z değeri ile ağırlıklar (w) güncellenmez; bunun yerine, z değeri üzerinden tahmin yapılır ve doğru olup olmadığına bakılır.

Bu işlemlerin hepsi epoch sayısı kadar yapılır ve döngüden çıkarılır.

```

FILE *file = fopen("results/results_gd.txt", "w");
for (epoch = 0; epoch < EPOCHS; epoch++) {
    fprintf(file, "%d %f %f %f %f\n", epoch, train_losses[epoch], train_accuracies[epoch], train_times[epoch], test_losses[epoch], test_accuracies[epoch]);
}
fclose(file);

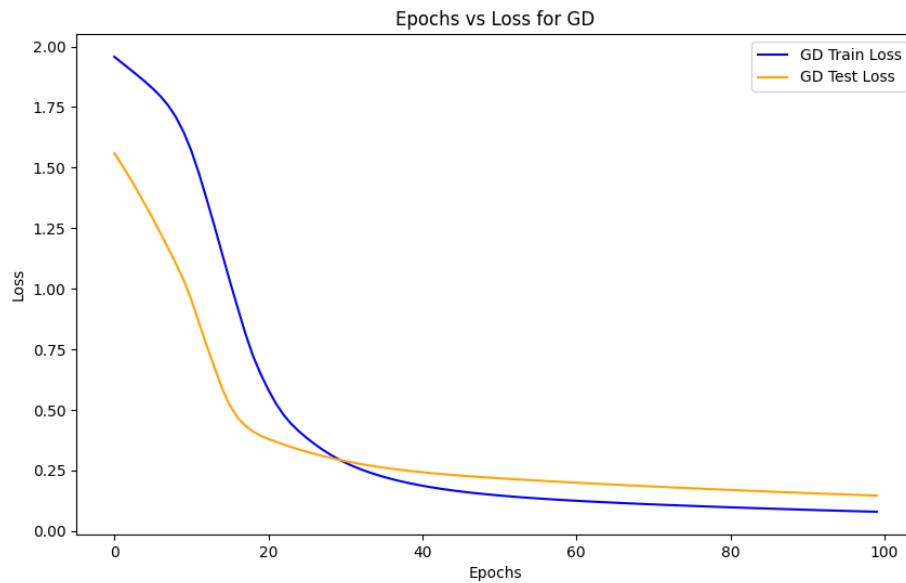
```

Fonksiyonun sonunda da grafikleri çizdirebilmek için gerekli olan değişkenler results\_gd.txt dosyasına kaydedilir.

Kodu açıklama kısmı tamamlandı. Şimdi örnek çıktı ekranı ve istenilen grafiklere geçebiliriz.

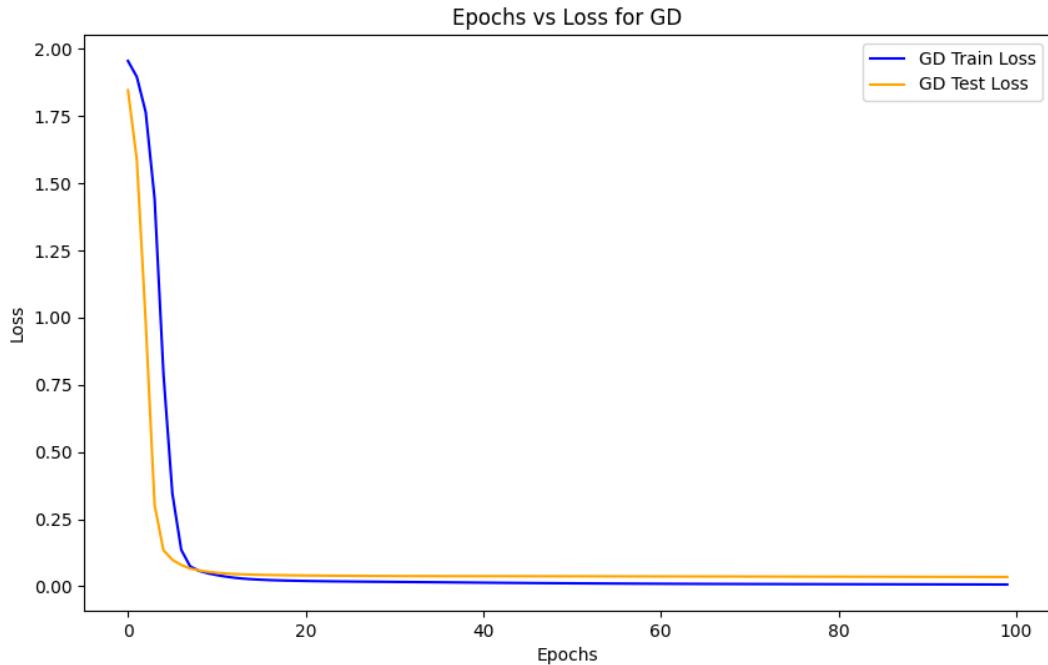
learning\_rate = 0.001 için;

```
Epoch 1: MSE = 1.957551, Accuracy: 47.900002, Time:0.286000, test_mse: 1.559370, test_Accuracy: 55.500000
Epoch 11: MSE = 1.569039, Accuracy: 56.300003, Time:2.571000, test_mse: 0.954239, test_Accuracy: 73.000000
Epoch 21: MSE = 0.582884, Accuracy: 81.800003, Time:4.861001, test_mse: 0.380343, test_Accuracy: 90.500000
Epoch 31: MSE = 0.282237, Accuracy: 91.399994, Time:7.113000, test_mse: 0.288750, test_Accuracy: 92.000000
Epoch 41: MSE = 0.187201, Accuracy: 93.500000, Time:9.406001, test_mse: 0.242841, test_Accuracy: 93.500000
Epoch 51: MSE = 0.146920, Accuracy: 95.300003, Time:11.746002, test_mse: 0.217973, test_Accuracy: 93.500000
Epoch 61: MSE = 0.125064, Accuracy: 96.000000, Time:14.046000, test_mse: 0.199725, test_Accuracy: 94.000000
Epoch 71: MSE = 0.110084, Accuracy: 96.599998, Time:16.318001, test_mse: 0.183934, test_Accuracy: 94.500000
Epoch 81: MSE = 0.098100, Accuracy: 97.000000, Time:18.606001, test_mse: 0.169741, test_Accuracy: 95.500000
Epoch 91: MSE = 0.087816, Accuracy: 97.099998, Time:20.963005, test_mse: 0.156890, test_Accuracy: 95.500000
```



learning\_rate = 0.01 için;

```
Epoch 1: MSE = 1.955740, Accuracy: 50.199997, Time:0.217000, test_mse: 1.846496, test_Accuracy: 51.500000
Epoch 11: MSE = 0.041569, Accuracy: 98.799995, Time:2.563000, test_mse: 0.050338, test_Accuracy: 99.000000
Epoch 21: MSE = 0.019772, Accuracy: 99.400002, Time:4.843000, test_mse: 0.040177, test_Accuracy: 99.000000
Epoch 31: MSE = 0.016238, Accuracy: 99.599998, Time:7.118000, test_mse: 0.038138, test_Accuracy: 99.000000
Epoch 41: MSE = 0.013458, Accuracy: 99.599998, Time:9.429000, test_mse: 0.037639, test_Accuracy: 99.000000
Epoch 51: MSE = 0.011041, Accuracy: 99.699997, Time:11.806000, test_mse: 0.037231, test_Accuracy: 99.000000
Epoch 61: MSE = 0.009346, Accuracy: 99.699997, Time:14.091000, test_mse: 0.036766, test_Accuracy: 99.000000
Epoch 71: MSE = 0.008335, Accuracy: 99.800003, Time:16.356001, test_mse: 0.036236, test_Accuracy: 99.000000
Epoch 81: MSE = 0.007676, Accuracy: 99.800003, Time:18.702000, test_mse: 0.035694, test_Accuracy: 99.000000
Epoch 91: MSE = 0.007076, Accuracy: 99.800003, Time:21.014997, test_mse: 0.035131, test_Accuracy: 99.000000
```



Learning rate'i büyütmek, modelin ağırlıklarını daha büyük adımlarla güncelleseyerek daha hızlı bir şekilde minimuma ulaşmasını sağlar, ancak çok büyük bir öğrenme oranı dengesizliğe yol açabilir o yüzden learning rate'i dikkatli seçmek lazım.

## Stochastic Gradient Descent (SGD)

SGD, bir gradient descent algoritmasıdır ve normal gradient descent'ten tek farkı, her iterasyonda tüm veriyi kullanmak yerine 1 veri noktası ya da mini-batch (örneğin, biz 32 boyutlu bir batch kullandık) kullanmasıdır. Bu nedenle, kodumuzda her iterasyonda tüm veriyi işlemek yerine mini-batch ile işlem yapılacak şekilde yazılmıştır; diğer kısımlar ise neredeyse gradient descent kodu ile aynıdır.

```

for (epoch = 0; epoch < EPOCHS; epoch++) {
    start = clock();
    if (epoch % 30 == 0 && epoch > 0) {
        learning_rate *= 0.80;
        printf("%f \n", learning_rate);
    }
    float mse = 0.0f, True=0, False=0;
    current_index = 0;

    float gradients[N * N + 1] = {0.0f};

    for(batch=0;batch<32;batch++){
        // Dosya yolunu oluştur
        int label;
        char file_path[512];
        if(batch%2==0){
            sprintf(file_path, sizeof(file_path), "%s/%s%d%s", dir_path, "0_",rand() % 500+1,".txt");
            label = -1;
        }else{
            sprintf(file_path, sizeof(file_path), "%s/%s%d%s", dir_path, "1_",rand() % 500+501,".txt");
            label = 1;
        }
    }
}

```

Learning rate'i giderek azaltmak, modelin daha iyi bir şekilde yakınsamasını sağlamak amacıyla, başlangıçta hızlı bir şekilde öğrenirken, ilerleyen adımlarda daha küçük adımlarla daha hassas bir optimizasyon yapılmasını sağlar ve r bölgesinde daha stabil bir yakınsama elde edilmesine yardımcı olur.

Döngünün içinde batch'in yarısını 0, yarısını 1 seçiyor ve aynı zamanda bu değerleri etiketliyoruz.

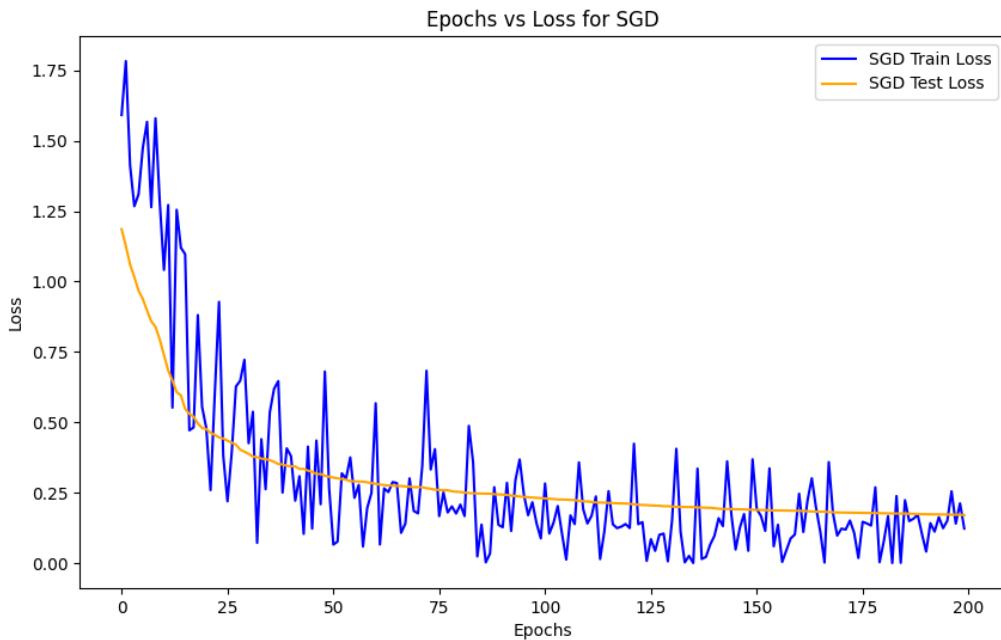
Örnek çıktı ve loss-epochs grafiği;

Grafikteki dalgalanmalar, SGD'nin rastgele minibatch seçimi nedeniyle her iterasyonda farklı veri kümelerinden öğrenme yapmasından kaynaklanmaktadır.

```

2- Epoch 1: MSE = 1.591802, Accuracy: 59.375000, Time:0.007000, test_mse: 1.186273, test_Accuracy: 66.000000
2- Epoch 6: MSE = 1.474527, Accuracy: 62.500000, Time:0.042000, test_mse: 0.938303, test_Accuracy: 71.500000
2- Epoch 11: MSE = 1.041315, Accuracy: 71.875000, Time:0.081000, test_mse: 0.739385, test_Accuracy: 76.500000
2- Epoch 16: MSE = 1.097681, Accuracy: 65.625000, Time:0.118000, test_mse: 0.547846, test_Accuracy: 84.000000
2- Epoch 21: MSE = 0.479278, Accuracy: 87.500000, Time:0.156000, test_mse: 0.475593, test_Accuracy: 85.500000
2- Epoch 26: MSE = 0.219428, Accuracy: 93.750000, Time:0.197000, test_mse: 0.434364, test_Accuracy: 86.500000
0.008000
2- Epoch 31: MSE = 0.425644, Accuracy: 87.500000, Time:0.233000, test_mse: 0.388191, test_Accuracy: 87.500000
2- Epoch 36: MSE = 0.535879, Accuracy: 84.375000, Time:0.266000, test_mse: 0.365650, test_Accuracy: 89.000000
2- Epoch 41: MSE = 0.379949, Accuracy: 90.625000, Time:0.302000, test_mse: 0.343881, test_Accuracy: 89.500000
2- Epoch 46: MSE = 0.122344, Accuracy: 93.750000, Time:0.342000, test_mse: 0.324662, test_Accuracy: 90.000000
2- Epoch 51: MSE = 0.065708, Accuracy: 100.000000, Time:0.379000, test_mse: 0.303501, test_Accuracy: 90.000000
2- Epoch 56: MSE = 0.231729, Accuracy: 93.750000, Time:0.418000, test_mse: 0.290502, test_Accuracy: 91.000000
0.006400
2- Epoch 61: MSE = 0.567515, Accuracy: 84.375000, Time:0.457000, test_mse: 0.281524, test_Accuracy: 91.000000
2- Epoch 66: MSE = 0.284178, Accuracy: 90.625000, Time:0.492000, test_mse: 0.274080, test_Accuracy: 90.500000
2- Epoch 71: MSE = 0.176639, Accuracy: 93.750000, Time:0.532000, test_mse: 0.269910, test_Accuracy: 90.500000
2- Epoch 76: MSE = 0.166659, Accuracy: 96.875000, Time:0.566000, test_mse: 0.259468, test_Accuracy: 91.000000
2- Epoch 81: MSE = 0.207645, Accuracy: 93.750000, Time:0.601000, test_mse: 0.252470, test_Accuracy: 92.000000
2- Epoch 86: MSE = 0.136464, Accuracy: 96.875000, Time:0.639000, test_mse: 0.246895, test_Accuracy: 92.000000
0.005120
2- Epoch 91: MSE = 0.127339, Accuracy: 96.875000, Time:0.680000, test_mse: 0.243326, test_Accuracy: 92.000000
2- Epoch 96: MSE = 0.242858, Accuracy: 90.625000, Time:0.720000, test_mse: 0.234950, test_Accuracy: 92.000000
2- Epoch 101: MSE = 0.282720, Accuracy: 90.625000, Time:0.760000, test_mse: 0.229772, test_Accuracy: 92.500000
2- Epoch 106: MSE = 0.012519, Accuracy: 100.000000, Time:0.794000, test_mse: 0.225384, test_Accuracy: 93.500000
2- Epoch 111: MSE = 0.140501, Accuracy: 93.750000, Time:0.829000, test_mse: 0.218585, test_Accuracy: 93.500000
2- Epoch 116: MSE = 0.255551, Accuracy: 93.750000, Time:0.865001, test_mse: 0.213388, test_Accuracy: 93.500000
0.004096
2- Epoch 121: MSE = 0.1244340, Accuracy: 93.750000, Time:0.917001, test_mse: 0.210130, test_Accuracy: 93.500000
2- Epoch 126: MSE = 0.084755, Accuracy: 96.875000, Time:0.954001, test_mse: 0.204757, test_Accuracy: 93.500000
2- Epoch 131: MSE = 0.152602, Accuracy: 93.750000, Time:0.992001, test_mse: 0.200770, test_Accuracy: 93.500000
2- Epoch 136: MSE = 0.000274, Accuracy: 100.000000, Time:1.031001, test_mse: 0.198913, test_Accuracy: 93.500000
2- Epoch 141: MSE = 0.096502, Accuracy: 96.875000, Time:1.070001, test_mse: 0.194936, test_Accuracy: 93.500000
2- Epoch 146: MSE = 0.048213, Accuracy: 96.875000, Time:1.109000, test_mse: 0.191068, test_Accuracy: 93.500000
0.003277
2- Epoch 151: MSE = 0.191605, Accuracy: 93.750000, Time:1.151000, test_mse: 0.188454, test_Accuracy: 93.500000
2- Epoch 156: MSE = 0.135996, Accuracy: 96.875000, Time:1.188000, test_mse: 0.186950, test_Accuracy: 93.500000
2- Epoch 161: MSE = 0.246314, Accuracy: 93.750000, Time:1.224000, test_mse: 0.185603, test_Accuracy: 93.500000
2- Epoch 166: MSE = 0.111441, Accuracy: 96.875000, Time:1.261000, test_mse: 0.182446, test_Accuracy: 93.500000
2- Epoch 171: MSE = 0.122304, Accuracy: 96.875000, Time:1.301000, test_mse: 0.179584, test_Accuracy: 93.500000
2- Epoch 176: MSE = 0.146503, Accuracy: 93.750000, Time:1.340000, test_mse: 0.178211, test_Accuracy: 93.500000
0.002621
2- Epoch 181: MSE = 0.077423, Accuracy: 96.875000, Time:1.380000, test_mse: 0.177027, test_Accuracy: 93.500000
2- Epoch 186: MSE = 0.223420, Accuracy: 93.750000, Time:1.414000, test_mse: 0.175825, test_Accuracy: 93.500000
2- Epoch 191: MSE = 0.040683, Accuracy: 96.875000, Time:1.448000, test_mse: 0.173575, test_Accuracy: 93.500000
2- Epoch 196: MSE = 0.150525, Accuracy: 93.750000, Time:1.486000, test_mse: 0.172610, test_Accuracy: 94.000000

```



## ADAM

**Adam**, geçmiş gradyanları ve gradyanların karelerini kullanarak parametre güncellemeleri yapar. GD'den farkı, her parametre için geçmişteki gradyanları ve karelerini dikkate almasıdır. Bundan dolayı gd kodundan tek farkı ağırlık değerlerini güncellediği kısımdadır.

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$  (Update biased first moment estimate)  
 $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$  (Update biased second raw moment estimate)  
 $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$  (Compute bias-corrected first moment estimate)  
 $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$  (Compute bias-corrected second raw moment estimate)  
 $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$  (Update parameters)

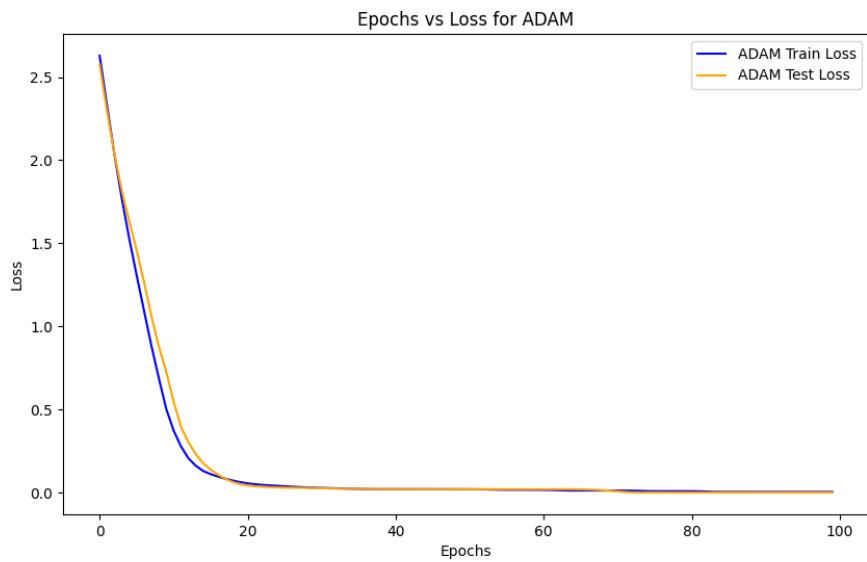
```
for (i = 0; i < N * N + 1; i++) {
    m[i] = beta1 * m[i] + (1 - beta1) * gradients[i];
    v[i] = beta2 * v[i] + (1 - beta2) * (gradients[i] * gradients[i]);

    float m_hat = m[i] / (1 - powf(beta1, epoch + 1));
    float v_hat = v[i] / (1 - powf(beta2, epoch + 1));

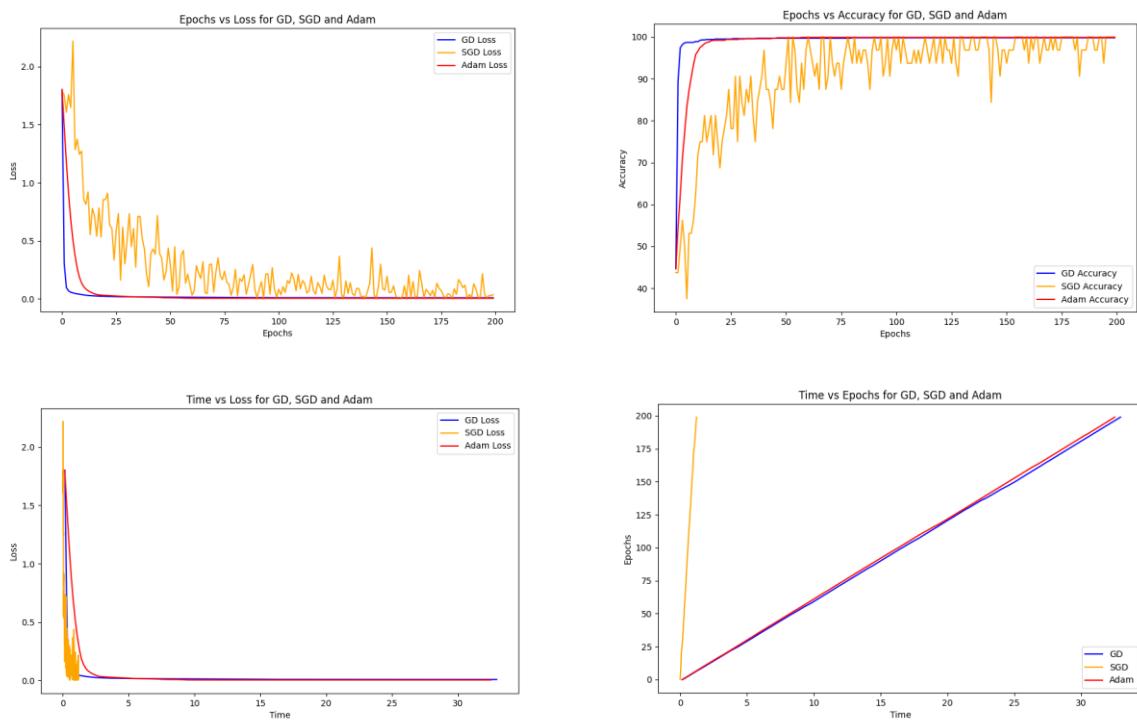
    w[i] -= learning_rate * m_hat / (sqrtf(v_hat) + epsilon);
    fprintf(w_record, "%f ", w[i]);
}
```

Örnek çıktı ve loss-epochs grafiği;

```
3- Epoch 1: MSE = 2.628451, Accuracy: 24.799999, Time:0.227000, test_mse: 2.575202, test_Accuracy: 26.499998
3- Epoch 6: MSE = 1.307838, Accuracy: 62.300003, Time:1.431000, test_mse: 1.457914, test_Accuracy: 58.999996
3- Epoch 11: MSE = 0.372857, Accuracy: 88.300003, Time:2.583000, test_mse: 0.547413, test_Accuracy: 83.000000
3- Epoch 16: MSE = 0.110730, Accuracy: 96.599998, Time:3.699000, test_mse: 0.137678, test_Accuracy: 95.500000
3- Epoch 21: MSE = 0.055208, Accuracy: 98.500000, Time:4.833000, test_mse: 0.042775, test_Accuracy: 99.000000
3- Epoch 26: MSE = 0.037884, Accuracy: 98.900002, Time:5.990999, test_mse: 0.029894, test_Accuracy: 99.000000
3- Epoch 31: MSE = 0.027673, Accuracy: 99.299995, Time:7.105999, test_mse: 0.025911, test_Accuracy: 99.500000
3- Epoch 36: MSE = 0.021629, Accuracy: 99.500000, Time:8.266998, test_mse: 0.021673, test_Accuracy: 99.500000
3- Epoch 41: MSE = 0.020172, Accuracy: 99.500000, Time:9.416998, test_mse: 0.020633, test_Accuracy: 99.500000
3- Epoch 46: MSE = 0.019959, Accuracy: 99.500000, Time:10.564997, test_mse: 0.020381, test_Accuracy: 99.500000
3- Epoch 51: MSE = 0.019547, Accuracy: 99.500000, Time:11.742996, test_mse: 0.020117, test_Accuracy: 99.500000
3- Epoch 56: MSE = 0.016467, Accuracy: 99.599998, Time:12.902995, test_mse: 0.020091, test_Accuracy: 99.500000
3- Epoch 61: MSE = 0.015616, Accuracy: 99.599998, Time:14.074995, test_mse: 0.020009, test_Accuracy: 99.500000
3- Epoch 66: MSE = 0.012018, Accuracy: 99.699997, Time:15.216995, test_mse: 0.019043, test_Accuracy: 99.500000
3- Epoch 71: MSE = 0.011855, Accuracy: 99.699997, Time:16.424995, test_mse: 0.007111, test_Accuracy: 99.500000
3- Epoch 76: MSE = 0.008196, Accuracy: 99.800003, Time:17.607996, test_mse: 0.000558, test_Accuracy: 100.000000
3- Epoch 81: MSE = 0.007840, Accuracy: 99.800003, Time:18.759996, test_mse: 0.000072, test_Accuracy: 100.000000
3- Epoch 86: MSE = 0.004353, Accuracy: 99.900002, Time:19.979998, test_mse: 0.000848, test_Accuracy: 100.000000
3- Epoch 91: MSE = 0.004003, Accuracy: 99.900002, Time:21.223000, test_mse: 0.000323, test_Accuracy: 100.000000
3- Epoch 96: MSE = 0.004005, Accuracy: 99.900002, Time:22.493000, test_mse: 0.000211, test_Accuracy: 100.000000
```

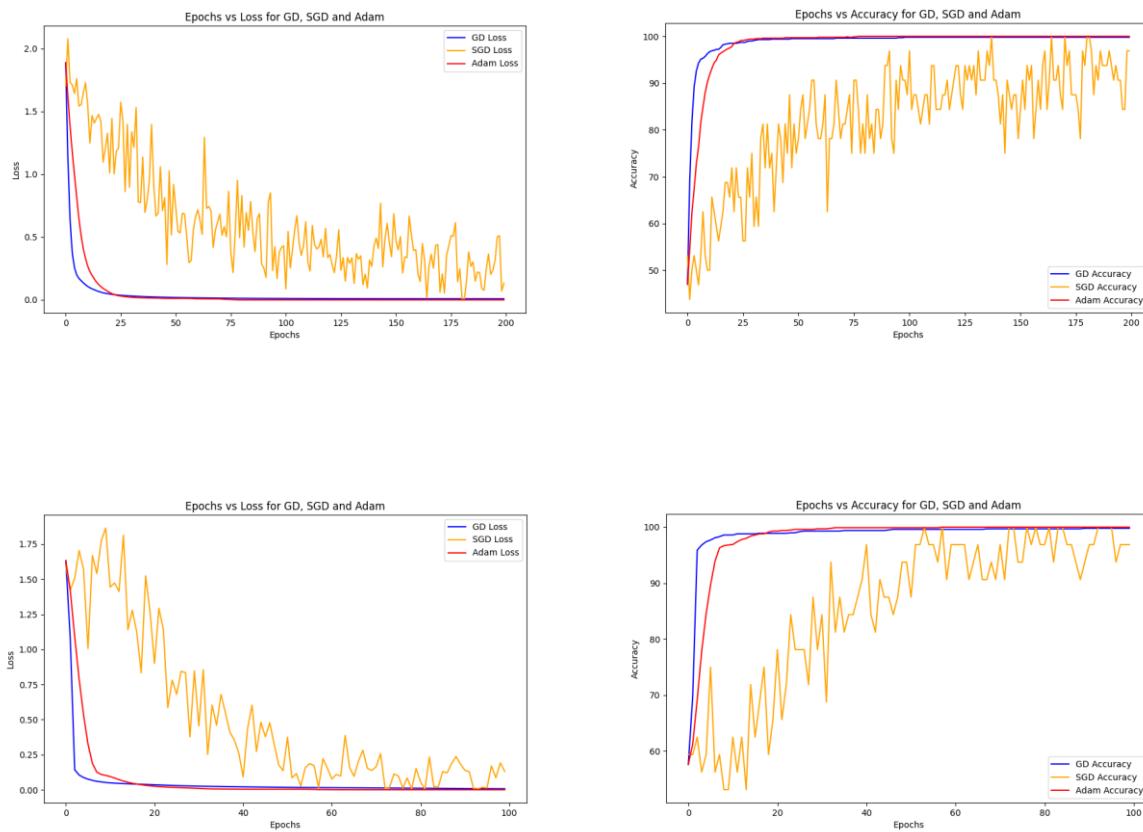


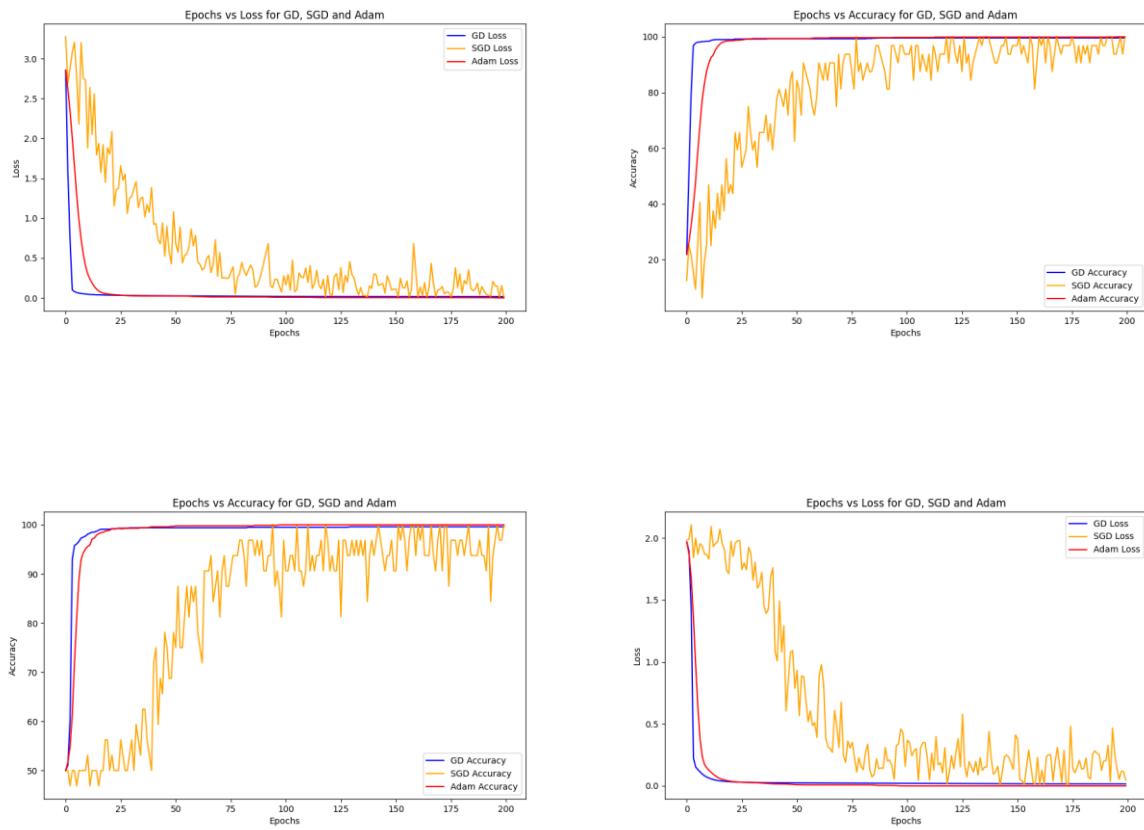
5 farklı ilk w değeri için GD, SGD ve Adam'ı karşılaştırması:



Yorumumuz: SGD, Adam ve GD ile karşılaştırıldığında çok daha hızlıdır, çünkü her iterasyonda tüm veri setini değil, sadece bir minibatch'i kullanır. Ancak, bunun bir dezavantajı vardır: rastgele minibatch seçimi nedeniyle öğrenme sürecinde dalgalanmalar meydana gelir ve optimum noktaya ulaşabilmesi için daha fazla iterasyon gereklidir. Yani, SGD'nin her iterasyonu daha düşük maliyetlidir, ancak istenen sonuca ulaşmak için daha fazla iterasyona ihtiyaç duyar.

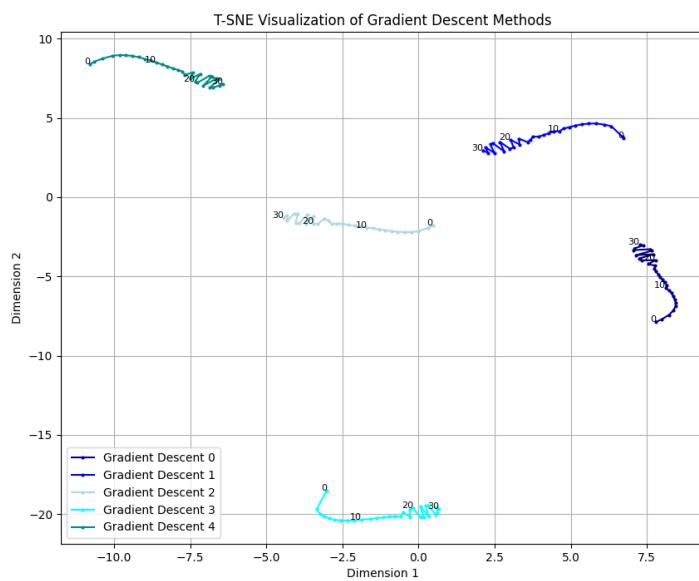
Adam ise, GD'den farklı olarak geçmiş gradyanları dikkate alır, bu sayede daha stabil bir öğrenme sağlar. Bu özellikler göz önünde bulundurularak, elimizdeki problem ve kaynaklara bağlı olarak hangi algoritmanın kullanılacağına karar verilebilir.

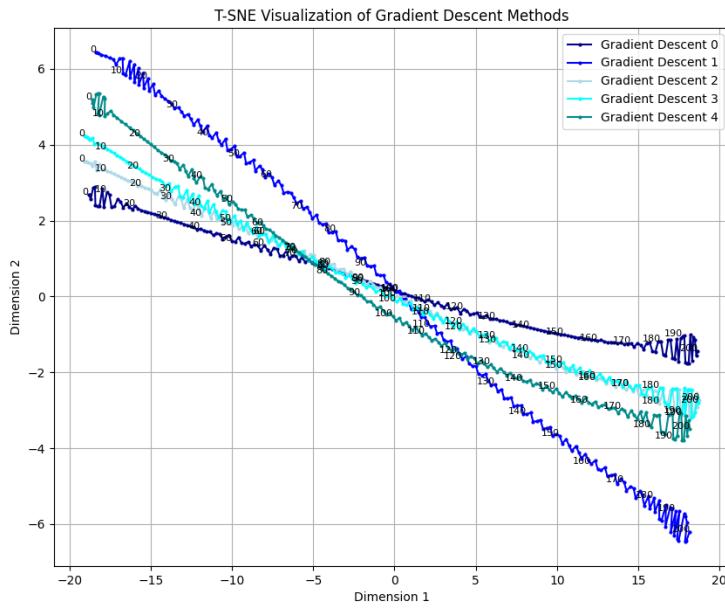




## B: Optimizasyon sürecinin 2 boyutta gösterimi

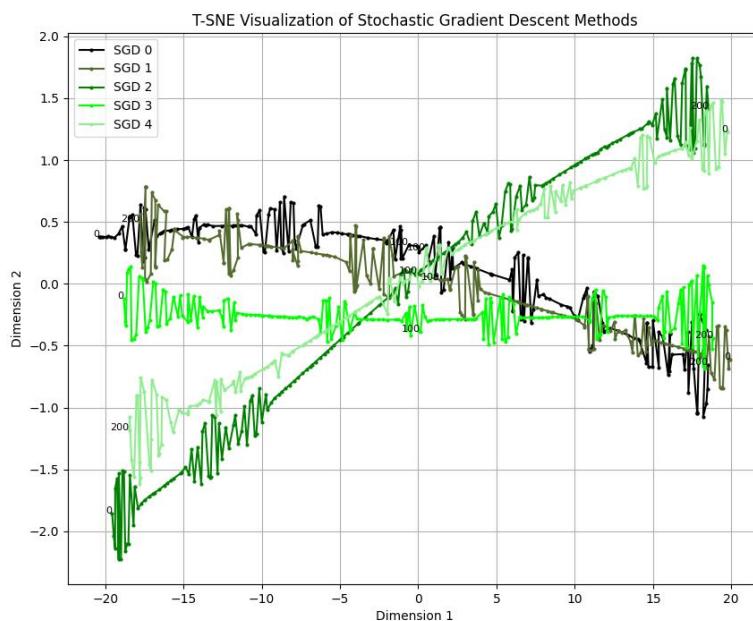
GD için:

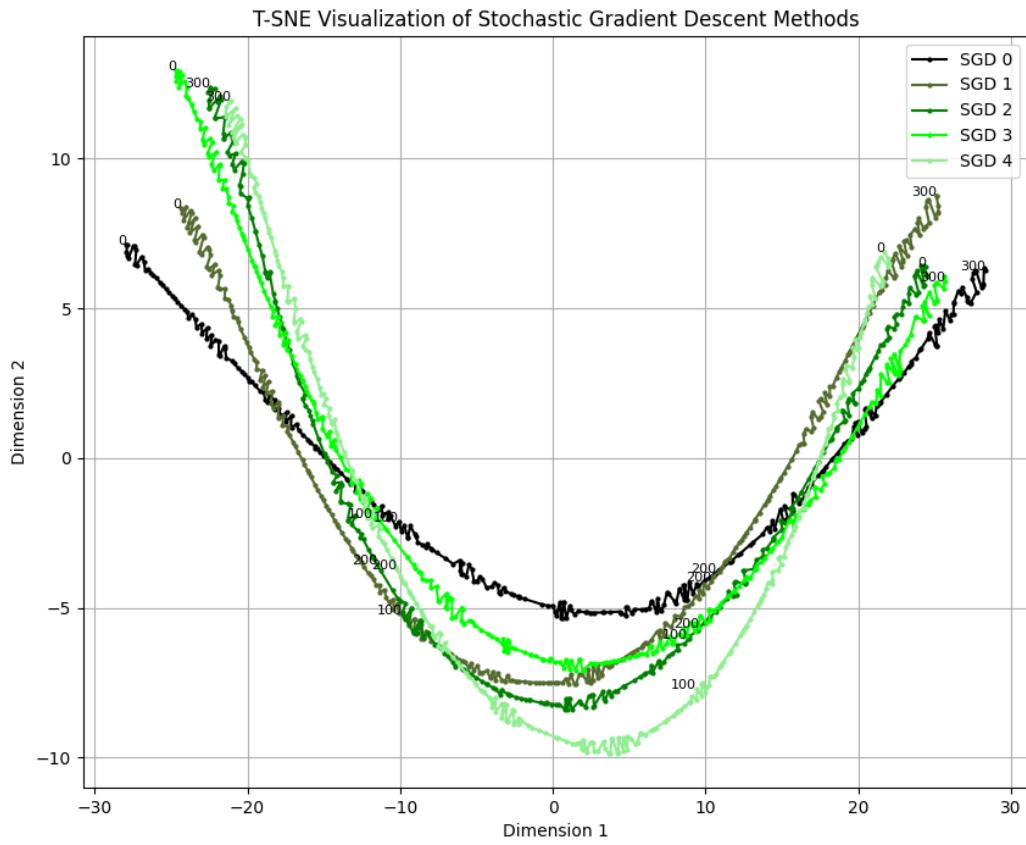




t-SNE sayesinde, yüksek boyutlu optimizasyon uzayındaki ağırlıkları 2 boyuta indirerek açıkça görselleştirdik. Ağırlıklar, başlangıçta farklı yerlerden başlasa da ilerleyen iterasyonlarla birlikte yakınsama gösteriyor. Bu, optimizasyon süreçlerinin sonunda benzer sonuçlara ulaştığını işaret ediyor olabilir.

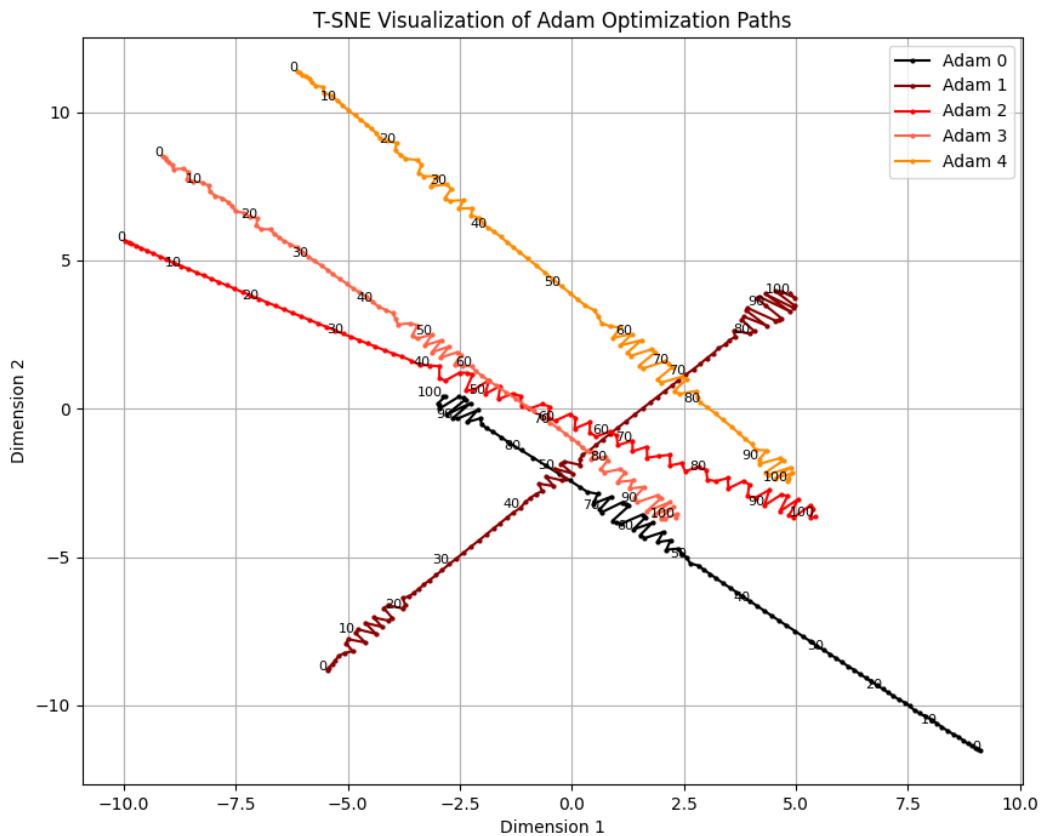
SGD için:





SGD'nin epoch-loss grafiğinde olduğu gibi, bu görsellerde de dalgalanma dikkat çekiyor. Ayrıca iterasyonların sonlarına doğru yolların bir noktada yoğunlaşması, optimizasyon sürecinin durulmaya başladığını ve algoritmanın bir minimum noktaya yaklaştığını gösteriyor olabilir. SGD'nin davranışları, başlangıç noktasına ve rastgele seçilen verilerin tüm veriyi ne kadar yansıtğına fazla duyarlıdır; bu da optimizasyon yollarındaki dalgalanma ve farklılıklara sebep olabilir diye düşünüyoruz.

ADAM için:

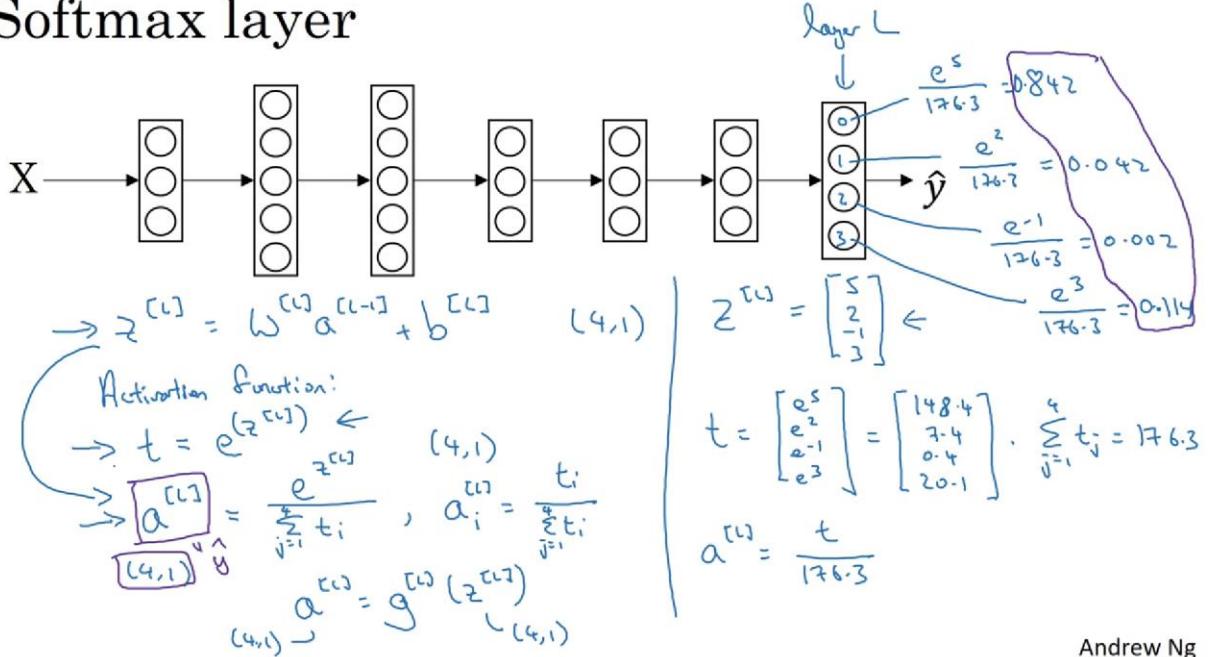


Adam, adaptif yapısı ve geçmişine önem vermesi sayesinde daha düzgün ve az zigzaglı bir optimizasyon yolu izler.

## BONUS 🎉

Yaptığımız araştırmalar sonucunda 4 sınıfı bir veri kümesinde sınıflandırma için en uygun aktivasyon fonksiyonunun softmax olduğunu gördük.

### Softmax layer



Andrew Ng

Kısaltısı softmax, birden fazla sınıf arasında bir olasılık dağılımı oluşturur.

4 sınıfı bir durumda:

- Model her sınıf için bir skor ( $z_1, z_2, z_3, z_4$ ) üretti.
- Bu skorlar  $e^z$  ile pozitif yaptık ve toplamlarını alınırlar:
- Her sınıfın olasılığı, kendi skoru bölü toplam skor:

Sonuç: Toplamı 1 olan olasılıklar (her sınıfın seçilme ihtimali).

```

int softmax(float *z, int num_classes, float *output) {
    float max_z = z[0];
    int i,max=0;
    for (i = 1; i < num_classes; i++) {
        if (z[i] > max_z) max_z = z[i];
    }

    float sum = 0.0f;
    for (i = 0; i < num_classes; i++) {
        output[i] = exp(z[i] - max_z);
        sum += output[i];
    }
    for (i = 0; i < num_classes; i++) {
        output[i] /= sum;
        if(output[i]>output[max]){
            max=i;
        }
    }
    return max;
}

```

Bu kısımda ayrıca çıkış parametrelerini maksimumdan çıkararak normalize ettik. Normalize etmesek de softmax çalışırıdı ancak sayısal kararlılık ve taşıma sorunlarını engellemek amacıyla normalize etmeyi tercih ettik.

Kod kısmına geçecek olursak, ikili sınıflandırma için yazdığımız kod ile dörtlü sınıflandırma kodu arasında çok büyük farklar yoktur. Çünkü dörtlü sınıflandırma kodunu, aslında ikili sınıflandırma kodunun üzerine inşa etmeye çalıştık.

```

float w_first[N * N + 1][NUM_CLASSES];

void initialize_weights() {
    int i,c;
    for (i = 0; i < N * N + 1; i++) {
        for (c = 0; c < NUM_CLASSES; c++) {
            w_first[i][c] = ((float)rand() / RAND_MAX) * 2.0f - 1.0f;
        }
    }
}

```

En temel fark, ikili sınıflandırma kodunda ağırlıklar ( $w$ ),  $N \times N + 1$  boyutunda bir vektörken, dörtlü sınıflandırma kodunda ağırlıklar  $[N \times N + 1] \times [NUM\_CLASSES]$  boyutunda bir matristir. Yani, dörtlü sınıflandırmada her sınıf için ayrı bir ağırlık vektörü bulunur.

## Gradient Descent V2

```
float z[NUM_CLASSES] = {0.0f};
for (c = 0; c < NUM_CLASSES; c++) {
    for (i = 0; i < N * N; i++) {
        z[c] += w[i][c] * x_train[i];
    }
    z[c] += w[N * N][c]; // Bias ekleme
}

float softmax_output[NUM_CLASSES];
int prediction = softmax(z, NUM_CLASSES, softmax_output);

if(prediction == label){
    True++;
}else{
    False++;
}

mse -= log(softmax_output[label]); // eğer 1 olursa hata 0 olur

for (c = 0; c < NUM_CLASSES; c++) {
    float error = softmax_output[c] - (c == label ? 1.0f : 0.0f);
    for (i = 0; i < N * N; i++) {
        gradients[i][c] += error * x_train[i];
    }
    gradients[N * N][c] += error; // Bias için gradyan
}

current_index++;
}

for (c = 0; c < NUM_CLASSES; c++) {
    for (i = 0; i < N * N + 1; i++) {
        //w[i][c] -= Learning_rate * gradients[i][c] / current_index;
        w[i][c] -= learning_rate * gradients[i][c];
        fprintf(w_record, "%f ", w[i][c]);
    }
}
fprintf(w_record, "\n");
```

Hesapladığımız z değerlerini softmax fonksiyonuna sokarak her bir sınıf için olasılıkları elde ediyoruz. Bu olasılıkları kullanarak modelimizin tahminini belirliyoruz. Modelin tahmini, en yüksek olasılığa sahip sınıfın etiketidir.

Hata fonksiyonu olarak kullandığımız MSE'yi, doğru sınıfın softmax çıktısının logaritmasını alarak hesaplıyoruz. Eğer doğru sınıfın olasılığı 1'e yakınsa, logaritma sıfır olur ve hata sıfır olur. Ancak doğru sınıfın olasılığı sıfıra yakınsa, logaritma negatif bir değer alır ve hata büyür.

Ağırlık güncellemelerini, her bir sınıf için gradyanları hesaplayarak yapıyoruz. Gradyanları öğrenme oranı ile çarparak ağırlıkları güncelliyoruz ve her epoch sonunda bu güncellenmiş ağırlıkları bir dosyaya kaydediyoruz.

```
int label = filename[0] - '0';

float z[NUM_CLASSES] = {0.0f};
for (c = 0; c < NUM_CLASSES; c++) {
    for (i = 0; i < N * N; i++) {
        z[c] += w[i][c] * x_test[i];
    }
    z[c] += w[N * N][c];
}

float probabilities[NUM_CLASSES];
int prediction = softmax(z, NUM_CLASSES, probabilities);

if (label == prediction) {
    True++;
} else {
    False++;
}

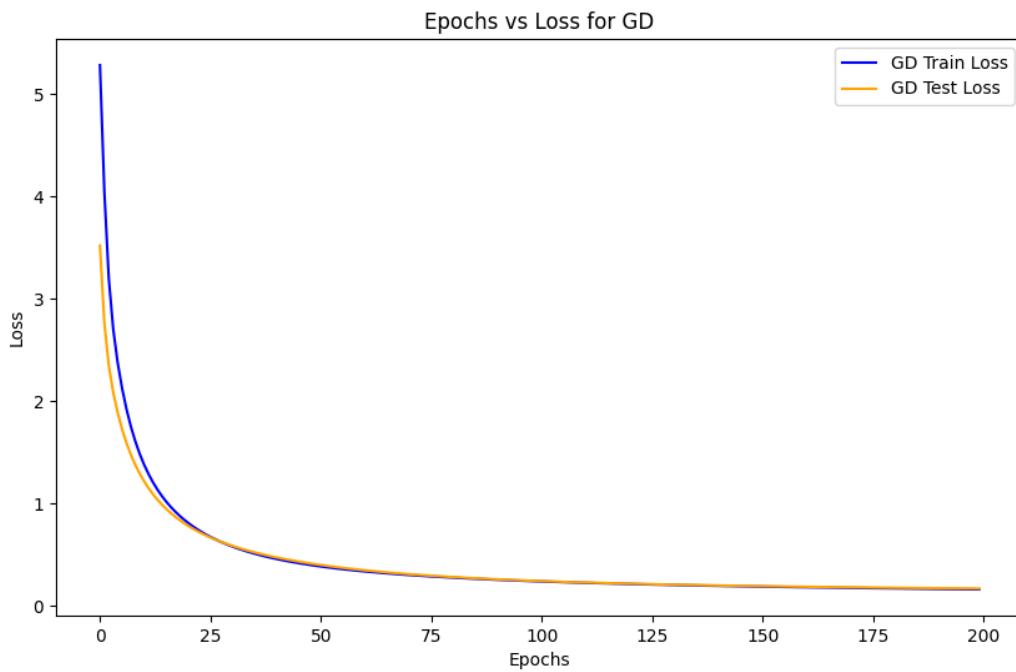
mse_test -= log(probabilities[label]);
```

Test aşamasında, train kısmında yaptığımız işlemlerin aynısı yapılır, ancak burada ağırlık matrisimiz güncellenmez.

Örnek Çıktı:

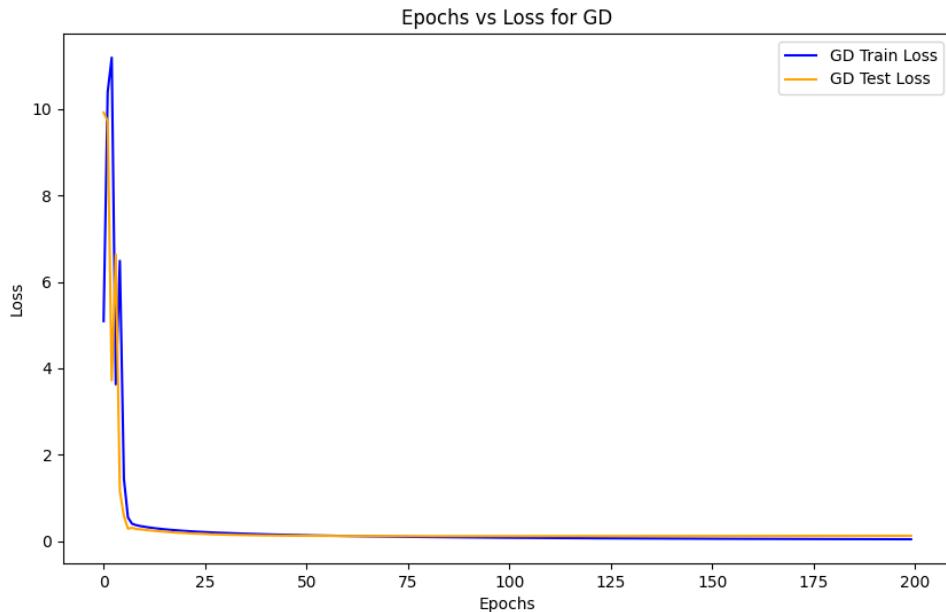
learning\_rate = 0.0001 için;

```
Epoch 1: MSE = 5.281985, Accuracy: 36.950001, Time:0.365000, test_mse: 3.518713, test_Accuracy: 41.000000
Epoch 6: MSE = 2.127694, Accuracy: 55.449997, Time:2.157000, test_mse: 1.727102, test_Accuracy: 59.500004
Epoch 11: MSE = 1.379195, Accuracy: 68.250000, Time:3.878000, test_mse: 1.216958, test_Accuracy: 69.750000
Epoch 16: MSE = 1.018611, Accuracy: 75.550003, Time:5.615000, test_mse: 0.947445, test_Accuracy: 75.000000
Epoch 21: MSE = 0.809789, Accuracy: 79.250000, Time:7.361000, test_mse: 0.780909, test_Accuracy: 77.750000
Epoch 26: MSE = 0.674159, Accuracy: 82.200005, Time:9.205000, test_mse: 0.668107, test_Accuracy: 81.000000
Epoch 31: MSE = 0.579412, Accuracy: 84.699997, Time:11.228001, test_mse: 0.586070, test_Accuracy: 83.000000
Epoch 36: MSE = 0.510271, Accuracy: 86.199997, Time:13.195002, test_mse: 0.523348, test_Accuracy: 84.500000
Epoch 41: MSE = 0.458112, Accuracy: 87.199997, Time:15.052002, test_mse: 0.473679, test_Accuracy: 85.750000
Epoch 46: MSE = 0.417540, Accuracy: 88.450005, Time:17.036001, test_mse: 0.433331, test_Accuracy: 86.000000
Epoch 51: MSE = 0.385124, Accuracy: 89.250000, Time:18.999001, test_mse: 0.399938, test_Accuracy: 86.500000
Epoch 56: MSE = 0.358635, Accuracy: 89.850006, Time:20.852003, test_mse: 0.371916, test_Accuracy: 87.000000
Epoch 61: MSE = 0.336572, Accuracy: 90.200005, Time:22.832003, test_mse: 0.348146, test_Accuracy: 87.500000
Epoch 66: MSE = 0.317895, Accuracy: 90.449997, Time:24.758001, test_mse: 0.327802, test_Accuracy: 88.500000
Epoch 71: MSE = 0.301866, Accuracy: 91.000000, Time:27.394001, test_mse: 0.310255, test_Accuracy: 88.750000
Epoch 76: MSE = 0.287947, Accuracy: 91.199997, Time:29.792002, test_mse: 0.295019, test_Accuracy: 89.250000
Epoch 81: MSE = 0.275743, Accuracy: 91.599998, Time:32.292000, test_mse: 0.281708, test_Accuracy: 89.500000
Epoch 86: MSE = 0.264951, Accuracy: 91.750000, Time:34.905094, test_mse: 0.270014, test_Accuracy: 89.750000
Epoch 91: MSE = 0.255336, Accuracy: 92.049995, Time:37.278996, test_mse: 0.259688, test_Accuracy: 91.000000
Epoch 96: MSE = 0.246712, Accuracy: 92.199997, Time:39.383999, test_mse: 0.250522, test_Accuracy: 91.750000
Epoch 101: MSE = 0.238931, Accuracy: 92.549995, Time:41.295002, test_mse: 0.242347, test_Accuracy: 92.000000
Epoch 106: MSE = 0.231869, Accuracy: 92.750000, Time:43.182003, test_mse: 0.235021, test_Accuracy: 92.750000
Epoch 111: MSE = 0.225426, Accuracy: 93.049995, Time:45.092003, test_mse: 0.228426, test_Accuracy: 93.000000
Epoch 116: MSE = 0.219518, Accuracy: 93.349998, Time:47.000000, test_mse: 0.222461, test_Accuracy: 93.500000
Epoch 121: MSE = 0.214076, Accuracy: 93.449997, Time:48.869999, test_mse: 0.217044, test_Accuracy: 93.750000
Epoch 126: MSE = 0.209041, Accuracy: 93.650002, Time:50.790001, test_mse: 0.212105, test_Accuracy: 93.750000
Epoch 131: MSE = 0.204363, Accuracy: 93.750000, Time:52.677002, test_mse: 0.207584, test_Accuracy: 93.750000
Epoch 136: MSE = 0.200002, Accuracy: 93.849998, Time:54.605000, test_mse: 0.203430, test_Accuracy: 93.750000
Epoch 141: MSE = 0.195921, Accuracy: 93.849998, Time:56.561001, test_mse: 0.199602, test_Accuracy: 94.000000
Epoch 146: MSE = 0.192092, Accuracy: 93.949997, Time:58.521004, test_mse: 0.196061, test_Accuracy: 94.250000
Epoch 151: MSE = 0.188487, Accuracy: 94.099998, Time:60.469002, test_mse: 0.192777, test_Accuracy: 94.500000
Epoch 156: MSE = 0.185086, Accuracy: 94.099998, Time:62.338001, test_mse: 0.189722, test_Accuracy: 94.500000
Epoch 161: MSE = 0.181868, Accuracy: 94.250000, Time:64.271004, test_mse: 0.186874, test_Accuracy: 95.000000
Epoch 166: MSE = 0.178817, Accuracy: 94.400002, Time:66.139992, test_mse: 0.184211, test_Accuracy: 95.000000
Epoch 171: MSE = 0.175919, Accuracy: 94.500000, Time:68.103996, test_mse: 0.181716, test_Accuracy: 95.000000
Epoch 176: MSE = 0.173160, Accuracy: 94.450005, Time:70.044991, test_mse: 0.179373, test_Accuracy: 95.000000
Epoch 181: MSE = 0.170530, Accuracy: 94.550003, Time:71.854988, test_mse: 0.177168, test_Accuracy: 95.000000
Epoch 186: MSE = 0.168018, Accuracy: 94.599998, Time:73.779991, test_mse: 0.175090, test_Accuracy: 95.000000
Epoch 191: MSE = 0.165616, Accuracy: 94.650002, Time:75.768992, test_mse: 0.173128, test_Accuracy: 95.000000
Epoch 196: MSE = 0.163314, Accuracy: 94.900002, Time:77.641991, test_mse: 0.171272, test_Accuracy: 95.250000
```



learning\_rate = 0.001 için; (learning rate arttığı için kararsız ama hızlı ilerliyor)

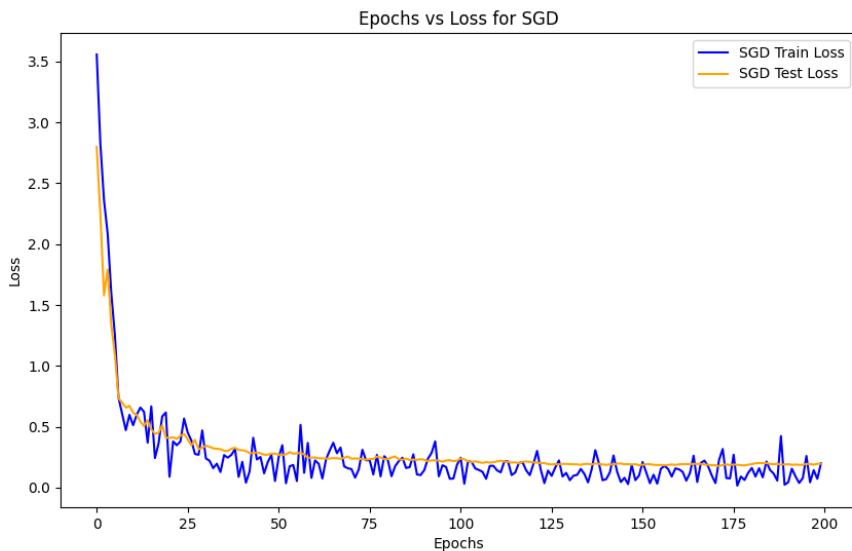
```
Epoch 1: MSE = 5.090679, Accuracy: 38.950001, Time:0.364000, test_mse: 9.916633, test_Accuracy: 32.750000
Epoch 6: MSE = 1.444581, Accuracy: 80.400002, Time:2.184000, test_mse: 0.584667, test_Accuracy: 90.500000
Epoch 11: MSE = 0.334973, Accuracy: 93.750000, Time:3.953000, test_mse: 0.263238, test_Accuracy: 93.250000
Epoch 16: MSE = 0.275855, Accuracy: 94.800003, Time:5.767000, test_mse: 0.219027, test_Accuracy: 93.500000
Epoch 21: MSE = 0.237810, Accuracy: 95.250000, Time:7.578000, test_mse: 0.187804, test_Accuracy: 93.750000
Epoch 26: MSE = 0.218563, Accuracy: 95.650002, Time:9.584000, test_mse: 0.166071, test_Accuracy: 94.750000
Epoch 31: MSE = 0.189707, Accuracy: 96.200005, Time:11.545000, test_mse: 0.151034, test_Accuracy: 95.750000
Epoch 36: MSE = 0.173055, Accuracy: 96.500000, Time:13.557000, test_mse: 0.140715, test_Accuracy: 95.750000
Epoch 41: MSE = 0.159300, Accuracy: 96.599998, Time:15.644000, test_mse: 0.133635, test_Accuracy: 95.500000
Epoch 46: MSE = 0.147599, Accuracy: 96.750000, Time:17.600000, test_mse: 0.128700, test_Accuracy: 95.750000
Epoch 51: MSE = 0.137455, Accuracy: 97.000000, Time:19.626001, test_mse: 0.125180, test_Accuracy: 96.250000
Epoch 56: MSE = 0.128568, Accuracy: 97.099998, Time:21.688002, test_mse: 0.122630, test_Accuracy: 96.250000
Epoch 61: MSE = 0.120722, Accuracy: 97.150002, Time:23.779001, test_mse: 0.120798, test_Accuracy: 96.250000
Epoch 66: MSE = 0.113735, Accuracy: 97.349998, Time:25.826002, test_mse: 0.119531, test_Accuracy: 96.500000
Epoch 71: MSE = 0.107461, Accuracy: 97.399994, Time:27.811005, test_mse: 0.118709, test_Accuracy: 96.250000
Epoch 76: MSE = 0.101788, Accuracy: 97.500000, Time:29.893003, test_mse: 0.118228, test_Accuracy: 96.250000
Epoch 81: MSE = 0.096635, Accuracy: 97.599998, Time:31.870005, test_mse: 0.118006, test_Accuracy: 96.250000
Epoch 86: MSE = 0.091944, Accuracy: 97.799995, Time:33.958008, test_mse: 0.117978, test_Accuracy: 96.250000
Epoch 91: MSE = 0.087669, Accuracy: 97.849998, Time:35.933010, test_mse: 0.118093, test_Accuracy: 96.250000
Epoch 96: MSE = 0.083778, Accuracy: 97.899994, Time:37.921005, test_mse: 0.118311, test_Accuracy: 96.250000
Epoch 101: MSE = 0.080238, Accuracy: 97.949997, Time:39.865002, test_mse: 0.118595, test_Accuracy: 96.250000
Epoch 106: MSE = 0.077021, Accuracy: 98.000000, Time:41.815994, test_mse: 0.118923, test_Accuracy: 96.250000
Epoch 111: MSE = 0.074092, Accuracy: 98.099998, Time:43.826992, test_mse: 0.119278, test_Accuracy: 96.500000
Epoch 116: MSE = 0.071419, Accuracy: 98.099998, Time:45.799995, test_mse: 0.119649, test_Accuracy: 96.500000
Epoch 121: MSE = 0.068968, Accuracy: 98.150002, Time:47.926994, test_mse: 0.120030, test_Accuracy: 96.500000
Epoch 126: MSE = 0.066712, Accuracy: 98.199997, Time:50.102997, test_mse: 0.120414, test_Accuracy: 96.500000
Epoch 131: MSE = 0.064623, Accuracy: 98.250000, Time:52.121994, test_mse: 0.120799, test_Accuracy: 96.500000
Epoch 136: MSE = 0.062682, Accuracy: 98.349998, Time:54.282993, test_mse: 0.121181, test_Accuracy: 96.250000
Epoch 141: MSE = 0.060869, Accuracy: 98.400002, Time:56.311993, test_mse: 0.121560, test_Accuracy: 96.000000
Epoch 146: MSE = 0.059172, Accuracy: 98.549995, Time:58.373993, test_mse: 0.121935, test_Accuracy: 96.000000
Epoch 151: MSE = 0.057576, Accuracy: 98.549995, Time:60.413994, test_mse: 0.122305, test_Accuracy: 95.750000
Epoch 156: MSE = 0.056073, Accuracy: 98.549995, Time:62.561993, test_mse: 0.122672, test_Accuracy: 95.750000
Epoch 161: MSE = 0.054653, Accuracy: 98.650002, Time:64.526993, test_mse: 0.123035, test_Accuracy: 96.000000
Epoch 166: MSE = 0.053310, Accuracy: 98.650002, Time:66.511993, test_mse: 0.123396, test_Accuracy: 96.000000
Epoch 171: MSE = 0.052038, Accuracy: 98.750000, Time:68.509987, test_mse: 0.123754, test_Accuracy: 96.000000
Epoch 176: MSE = 0.050830, Accuracy: 98.799995, Time:70.537979, test_mse: 0.124110, test_Accuracy: 96.000000
Epoch 181: MSE = 0.049682, Accuracy: 98.799995, Time:72.552979, test_mse: 0.124464, test_Accuracy: 96.000000
Epoch 186: MSE = 0.048590, Accuracy: 98.849998, Time:74.637985, test_mse: 0.124817, test_Accuracy: 95.750000
Epoch 191: MSE = 0.047550, Accuracy: 98.900002, Time:76.788986, test_mse: 0.125168, test_Accuracy: 95.750000
Epoch 196: MSE = 0.046557, Accuracy: 98.949997, Time:78.789986, test_mse: 0.125516, test_Accuracy: 95.750000
```



## Stochastic Gradient Descent V2 (SGD)

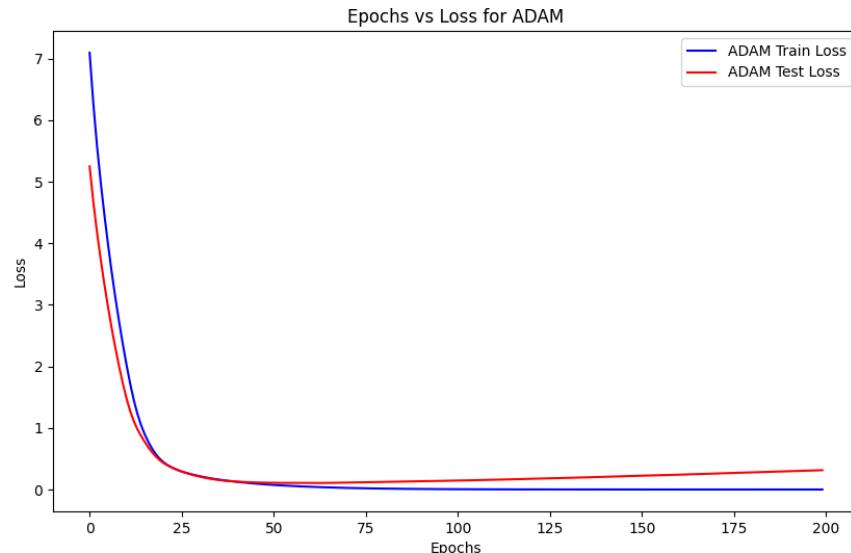
SGD V2 ve ADAM V2 için kodda, GD'de yaptığımız değişikliklerin aynısını uyguladık. Bu nedenle kodu tekrar açıklamayacağız aksi halde rapor çok uzayacak. Kodu detaylı incelemek isterseniz, drive linkinden erişebilirsiniz.

```
Epoch 1: MSE = 3.558223, Accuracy: 45.312500, Time:0.013000, test_mse: 2.798246, test_Accuracy: 44.750000
Epoch 6: MSE = 1.249891, Accuracy: 70.312500, Time:0.063000, test_mse: 1.092151, test_Accuracy: 72.500000
Epoch 11: MSE = 0.512831, Accuracy: 85.937500, Time:0.125000, test_mse: 0.615326, test_Accuracy: 84.500000
Epoch 16: MSE = 0.667909, Accuracy: 87.500000, Time:0.212000, test_mse: 0.478328, test_Accuracy: 86.500000
Epoch 21: MSE = 0.089584, Accuracy: 95.312500, Time:0.274000, test_mse: 0.409155, test_Accuracy: 88.250000
Epoch 26: MSE = 0.453151, Accuracy: 85.937500, Time:0.358000, test_mse: 0.398432, test_Accuracy: 88.250000
0.008000
Epoch 31: MSE = 0.240330, Accuracy: 93.750000, Time:0.433000, test_mse: 0.344923, test_Accuracy: 88.250000
Epoch 36: MSE = 0.267395, Accuracy: 95.312500, Time:0.481000, test_mse: 0.303133, test_Accuracy: 90.750000
Epoch 41: MSE = 0.212133, Accuracy: 92.187500, Time:0.557000, test_mse: 0.308082, test_Accuracy: 90.250000
Epoch 46: MSE = 0.252240, Accuracy: 93.750000, Time:0.634000, test_mse: 0.282108, test_Accuracy: 90.750000
Epoch 51: MSE = 0.251034, Accuracy: 95.312500, Time:0.661000, test_mse: 0.271457, test_Accuracy: 92.250000
Epoch 56: MSE = 0.054075, Accuracy: 96.875000, Time:0.723000, test_mse: 0.282142, test_Accuracy: 90.750000
0.006400
Epoch 61: MSE = 0.224399, Accuracy: 93.750000, Time:0.786000, test_mse: 0.248210, test_Accuracy: 92.000000
Epoch 66: MSE = 0.368937, Accuracy: 90.625000, Time:0.846000, test_mse: 0.243046, test_Accuracy: 92.250000
Epoch 71: MSE = 0.152633, Accuracy: 93.750000, Time:0.914000, test_mse: 0.237526, test_Accuracy: 92.250000
Epoch 76: MSE = 0.225844, Accuracy: 95.312500, Time:0.976000, test_mse: 0.234563, test_Accuracy: 93.000000
Epoch 81: MSE = 0.224996, Accuracy: 93.750000, Time:1.039000, test_mse: 0.234077, test_Accuracy: 93.000000
Epoch 86: MSE = 0.161438, Accuracy: 93.750000, Time:1.101000, test_mse: 0.239471, test_Accuracy: 91.750000
0.005120
Epoch 91: MSE = 0.143683, Accuracy: 95.312500, Time:1.161000, test_mse: 0.229555, test_Accuracy: 92.750000
Epoch 96: MSE = 0.184024, Accuracy: 95.312500, Time:1.223000, test_mse: 0.213037, test_Accuracy: 93.750000
Epoch 101: MSE = 0.246326, Accuracy: 92.187500, Time:1.285000, test_mse: 0.239784, test_Accuracy: 91.000000
Epoch 106: MSE = 0.145248, Accuracy: 93.750000, Time:1.346000, test_mse: 0.208944, test_Accuracy: 93.250000
Epoch 111: MSE = 0.141634, Accuracy: 93.750000, Time:1.406000, test_mse: 0.215152, test_Accuracy: 93.000000
Epoch 116: MSE = 0.124547, Accuracy: 93.750000, Time:1.468000, test_mse: 0.208378, test_Accuracy: 93.250000
0.00496
Epoch 121: MSE = 0.195634, Accuracy: 93.750000, Time:1.528000, test_mse: 0.207460, test_Accuracy: 93.000000
Epoch 126: MSE = 0.097163, Accuracy: 96.875000, Time:1.594000, test_mse: 0.190637, test_Accuracy: 94.250000
Epoch 131: MSE = 0.061021, Accuracy: 100.000000, Time:1.667000, test_mse: 0.192911, test_Accuracy: 94.250000
Epoch 136: MSE = 0.043299, Accuracy: 100.000000, Time:1.755000, test_mse: 0.195667, test_Accuracy: 93.750000
Epoch 141: MSE = 0.069837, Accuracy: 96.875000, Time:1.821000, test_mse: 0.186954, test_Accuracy: 94.500000
Epoch 146: MSE = 0.080757, Accuracy: 98.437500, Time:1.888000, test_mse: 0.192540, test_Accuracy: 93.500000
0.003277
Epoch 151: MSE = 0.210793, Accuracy: 92.187500, Time:1.977000, test_mse: 0.188625, test_Accuracy: 94.000000
Epoch 156: MSE = 0.154812, Accuracy: 95.312500, Time:2.043000, test_mse: 0.186114, test_Accuracy: 94.250000
Epoch 161: MSE = 0.150534, Accuracy: 95.312500, Time:2.086000, test_mse: 0.191635, test_Accuracy: 93.500000
Epoch 166: MSE = 0.046210, Accuracy: 98.437500, Time:2.171001, test_mse: 0.194707, test_Accuracy: 93.000000
Epoch 171: MSE = 0.037414, Accuracy: 100.000000, Time:2.254001, test_mse: 0.184665, test_Accuracy: 93.750000
Epoch 176: MSE = 0.270292, Accuracy: 95.312500, Time:2.329000, test_mse: 0.189603, test_Accuracy: 93.750000
0.002621
Epoch 181: MSE = 0.162076, Accuracy: 92.187500, Time:2.400001, test_mse: 0.193397, test_Accuracy: 93.000000
Epoch 186: MSE = 0.145117, Accuracy: 95.312500, Time:2.467001, test_mse: 0.194489, test_Accuracy: 93.000000
Epoch 191: MSE = 0.044048, Accuracy: 98.437500, Time:2.537000, test_mse: 0.193064, test_Accuracy: 93.250000
Epoch 196: MSE = 0.260169, Accuracy: 95.312500, Time:2.622000, test_mse: 0.195037, test_Accuracy: 93.000000
```



## ADAM V2

```
3- Epoch 1: MSE = 7.094832, Accuracy: 12.050000, Time:1.083000, test_mse: 5.208463, test_Accuracy: 15.500000
3- Epoch 6: MSE = 3.979593, Accuracy: 20.500000, Time:2.829000, test_mse: 2.961923, test_Accuracy: 26.000000
3- Epoch 11: MSE = 2.031104, Accuracy: 47.150002, Time:4.701000, test_mse: 1.482601, test_Accuracy: 52.749996
3- Epoch 16: MSE = 0.891091, Accuracy: 69.150002, Time:6.549000, test_mse: 0.770071, test_Accuracy: 72.000000
3- Epoch 21: MSE = 0.446475, Accuracy: 84.899994, Time:8.441999, test_mse: 0.433659, test_Accuracy: 85.750000
3- Epoch 26: MSE = 0.293569, Accuracy: 91.049995, Time:10.401000, test_mse: 0.291562, test_Accuracy: 90.000000
3- Epoch 31: MSE = 0.214566, Accuracy: 93.250000, Time:12.349999, test_mse: 0.208006, test_Accuracy: 91.750000
3- Epoch 36: MSE = 0.161911, Accuracy: 94.849998, Time:14.266999, test_mse: 0.154551, test_Accuracy: 95.000000
3- Epoch 41: MSE = 0.124324, Accuracy: 96.200005, Time:16.184999, test_mse: 0.128812, test_Accuracy: 95.750000
3- Epoch 46: MSE = 0.095912, Accuracy: 97.299995, Time:18.205000, test_mse: 0.119192, test_Accuracy: 96.750000
3- Epoch 51: MSE = 0.074115, Accuracy: 98.199997, Time:20.244999, test_mse: 0.108758, test_Accuracy: 96.500000
3- Epoch 56: MSE = 0.057166, Accuracy: 98.650002, Time:22.124996, test_mse: 0.106790, test_Accuracy: 96.500000
3- Epoch 61: MSE = 0.044072, Accuracy: 99.150002, Time:24.112993, test_mse: 0.105412, test_Accuracy: 96.500000
3- Epoch 66: MSE = 0.033882, Accuracy: 99.349998, Time:26.047993, test_mse: 0.107442, test_Accuracy: 96.750000
3- Epoch 71: MSE = 0.025867, Accuracy: 99.650002, Time:27.934996, test_mse: 0.112255, test_Accuracy: 97.000000
3- Epoch 76: MSE = 0.019599, Accuracy: 99.699997, Time:29.854994, test_mse: 0.117048, test_Accuracy: 97.250000
3- Epoch 81: MSE = 0.014923, Accuracy: 99.750000, Time:31.766994, test_mse: 0.122069, test_Accuracy: 97.000000
3- Epoch 86: MSE = 0.011347, Accuracy: 99.949997, Time:33.706001, test_mse: 0.126610, test_Accuracy: 97.000000
3- Epoch 91: MSE = 0.008526, Accuracy: 99.949997, Time:35.623001, test_mse: 0.132214, test_Accuracy: 97.250000
3- Epoch 96: MSE = 0.006370, Accuracy: 100.000000, Time:37.596996, test_mse: 0.138290, test_Accuracy: 97.250000
3- Epoch 101: MSE = 0.004768, Accuracy: 100.000000, Time:39.568993, test_mse: 0.144861, test_Accuracy: 97.000000
3- Epoch 106: MSE = 0.003611, Accuracy: 100.000000, Time:41.624992, test_mse: 0.151930, test_Accuracy: 96.750000
3- Epoch 111: MSE = 0.002747, Accuracy: 100.000000, Time:43.565994, test_mse: 0.159055, test_Accuracy: 96.750000
3- Epoch 116: MSE = 0.002096, Accuracy: 100.000000, Time:45.527996, test_mse: 0.166535, test_Accuracy: 96.500000
3- Epoch 121: MSE = 0.001603, Accuracy: 100.000000, Time:47.492996, test_mse: 0.174568, test_Accuracy: 96.250000
3- Epoch 126: MSE = 0.001239, Accuracy: 100.000000, Time:49.354000, test_mse: 0.182438, test_Accuracy: 96.250000
3- Epoch 131: MSE = 0.000967, Accuracy: 100.000000, Time:51.230000, test_mse: 0.189742, test_Accuracy: 96.250000
3- Epoch 136: MSE = 0.000752, Accuracy: 100.000000, Time:53.138996, test_mse: 0.197752, test_Accuracy: 96.000000
3- Epoch 141: MSE = 0.000584, Accuracy: 100.000000, Time:55.877991, test_mse: 0.206320, test_Accuracy: 96.000000
3- Epoch 146: MSE = 0.000452, Accuracy: 100.000000, Time:57.880990, test_mse: 0.215579, test_Accuracy: 96.000000
3- Epoch 151: MSE = 0.000352, Accuracy: 100.000000, Time:59.029887, test_mse: 0.224822, test_Accuracy: 95.750000
3- Epoch 156: MSE = 0.000278, Accuracy: 100.000000, Time:60.907986, test_mse: 0.232708, test_Accuracy: 96.000000
3- Epoch 161: MSE = 0.000220, Accuracy: 100.000000, Time:62.846985, test_mse: 0.240890, test_Accuracy: 96.000000
3- Epoch 166: MSE = 0.000172, Accuracy: 100.000000, Time:64.727982, test_mse: 0.250011, test_Accuracy: 96.000000
3- Epoch 171: MSE = 0.000134, Accuracy: 100.000000, Time:66.658981, test_mse: 0.259662, test_Accuracy: 95.750000
3- Epoch 176: MSE = 0.000104, Accuracy: 100.000000, Time:68.467979, test_mse: 0.270040, test_Accuracy: 95.750000
3- Epoch 181: MSE = 0.000082, Accuracy: 100.000000, Time:70.420982, test_mse: 0.279451, test_Accuracy: 95.750000
3- Epoch 186: MSE = 0.000065, Accuracy: 100.000000, Time:72.333984, test_mse: 0.287210, test_Accuracy: 96.000000
3- Epoch 191: MSE = 0.000052, Accuracy: 100.000000, Time:74.267982, test_mse: 0.296564, test_Accuracy: 96.250000
3- Epoch 196: MSE = 0.000041, Accuracy: 100.000000, Time:76.205978, test_mse: 0.306436, test_Accuracy: 96.000000
```



Test loss'un sonlara doğru artması overfitting olduğunu söyleyebilir.