Yildiz Teknik Üniversitesi

BLM2642 Ödev 1

Görüntü Sınıflandırma

Habil Coban habil.coban@std.yildiz.edu.tr

2024/2025 – 1st Semester

İçindekiler

1 Giriş

Bilgisayar Mühendisleri için Diferansiyel Denklemler Dersinin 1. ödevi olan 'Görüntü Sınıflandırma' ödevinin raporudur. Ödevde istenen iki gruplu bir veri kümesini sınıflandırmamızdır. Görüntülerin en az 20x20 piksel olması gerektiği için MNIST verisetindeki 28x28'lik el yazısı rakamlarını kullandım. Yaptığım ödevi bu raporda açıklayacağım.

2 İki Gruplu Sınıflandırma

MNIST veri setinde 28x28 piksellik görseller 1x784 şeklinde satırlarda tutuluyor. Bundan dolayı verimizi işlemeden önce yapmamız gereken bir işlem yok. Verimizi programa okuduktan sonra uygun bir şekilde modeli hazırlayıp eğitimine başlayabiliriz.

Modelimizin girişi(input) 784 piksel değeri ve bir adet Bias dan oluşacak. Başlangıçta rastgele belirlediğimiz ağırlıkları hedef değerlerimize göre güncelleyeceğiz. Aktivasyon fonksiyonu olarak ***tanh()*** kullanmamız gerektiği için 0 rakamını -1 değerine , 1 rakamını 1 değerine eşleştirip modeli buna göre eğiteceğiz. Maliyet(Cost) fonksiyonumuz olarak **MSE** kullanacağız. Cost fonksiyonumuz w lardaki değişime göre optimize edeceğiz.

$$
\begin{aligned}
z^{(L)} &= \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{784} \end{bmatrix} \cdot \begin{bmatrix} x_1 & x_2 & \cdots & x_{784} \end{bmatrix} + b^{(L)} \\
a^{(L)} &= \tanh(z^{(L)}) \\
C_0 &= (a^{(L)} - y)^2
\end{aligned}
$$

Yukarıdaki notasyonları kullanarak fonksiyonumuzun türevini yazalım: [1]

$$\boxed{\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial C_0}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial w^{(L)}}} \tag{1}$$

$$\boxed{\frac{\partial C_0}{\partial w^{(L)}} = 2(a^{(L)} - y) \cdot tanh'(z^{(L)}) \cdot a^{(L-1)}} \tag{2}$$

[1] Cost fonksiyonun türevinde $a^{(L-1)}$ ile gösterilen değer, bir önceki katman yani girdi değerlerimiz oluyor.

Ağırlıklardaki değişimleri (gradyanları) hesapladıktan sonra güncelleme işlemimizi, öğrenme hızına(η) bağlı olarak yapıyoruz. [2] [3] [4]

$$\boxed{w_{i+1} = w_i - \eta \cdot \frac{\partial C_0}{\partial w^{(L)}}} \tag{3}$$

Ağırlıklarımızı bu şekilde güncelleyerek minimum noktaya ulaşabiliriz. Ancak bulduğumuz noktanın global minimum olduğunu bilemiyoruz çünkü elimizdeki fonksiyon convex değil. Daha iyi sonuç veren local minimumlara ulaşmak için kullanılan bazı yöntemler vardır. Bunlardan Gradient Descent(GD) Stochastic Gradient Descent(SGD) ve ADAM algoritmalarını karşılaştıralım.

3 GD, SGD ve ADAM Karşılaştırması

GD, SGD ve ADAM algoritmalarını farklı başlangıç noktaları kullanarak MNIST veri setinin 0 ve 1 rakamları üzerinde eğitip test edelim. Bunların Loss-Time (Hata - Zaman) ve Loss-Iteration(Hata - Adım Sayısı) grafikleri şu şekildedir :

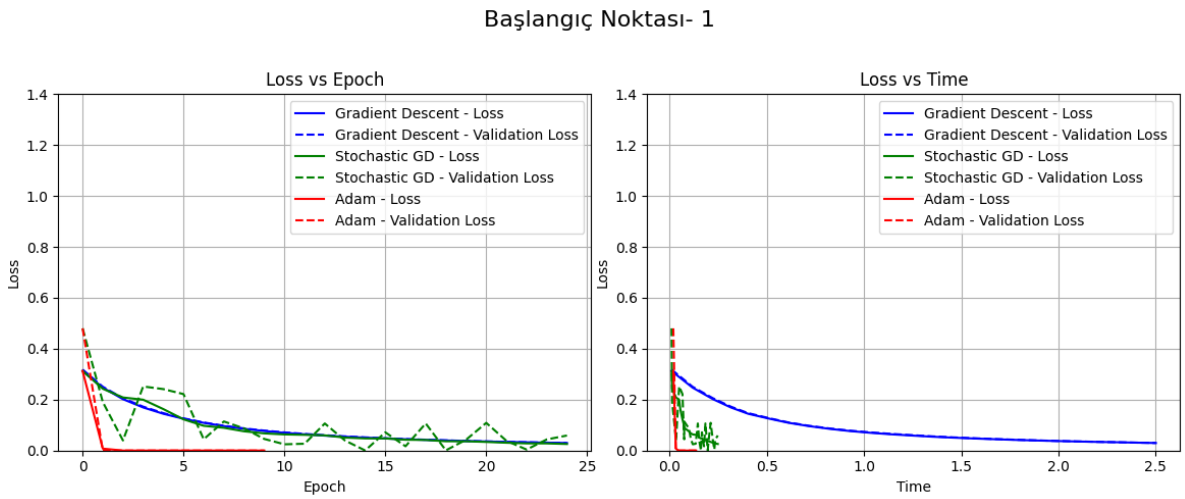

Figure 1: Başlangıç Noktası 1

[2]Not : Eğitimde 8800 tane örnek (4200 tane 1 , 4600 tane 0) kullanılmıştır.
[3]Not : Öğrenme hızı (η) 3 yöntemde de '0.01' dir. SGD ve ADAM da bathsize olarak 30 kullanılmıştır.
[4]Not : Loss, test kümesindeki hatayı göstermektedir.

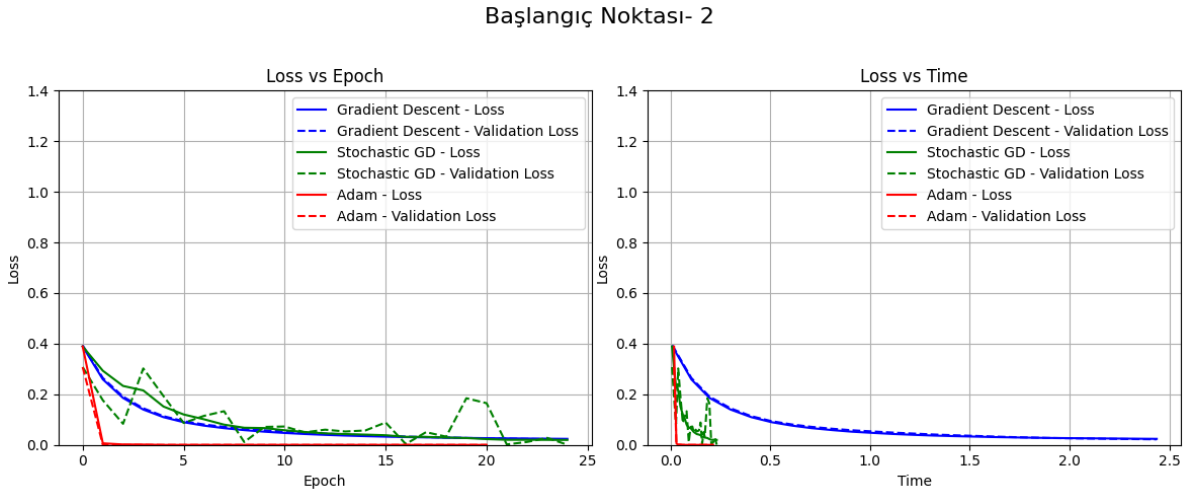

Figure 2: Başlangıç Noktası 2

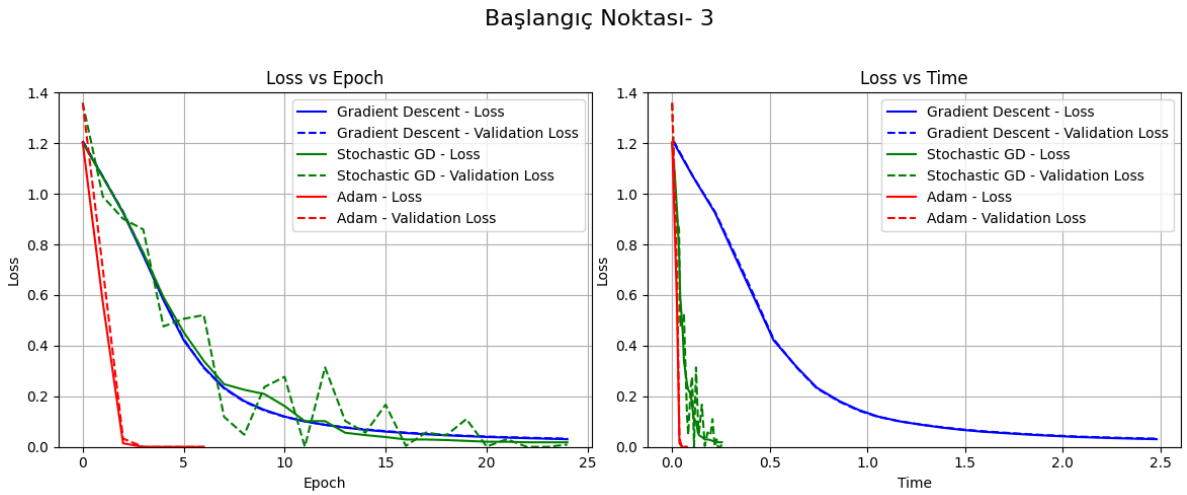

Figure 3: Başlangıç Noktası 3

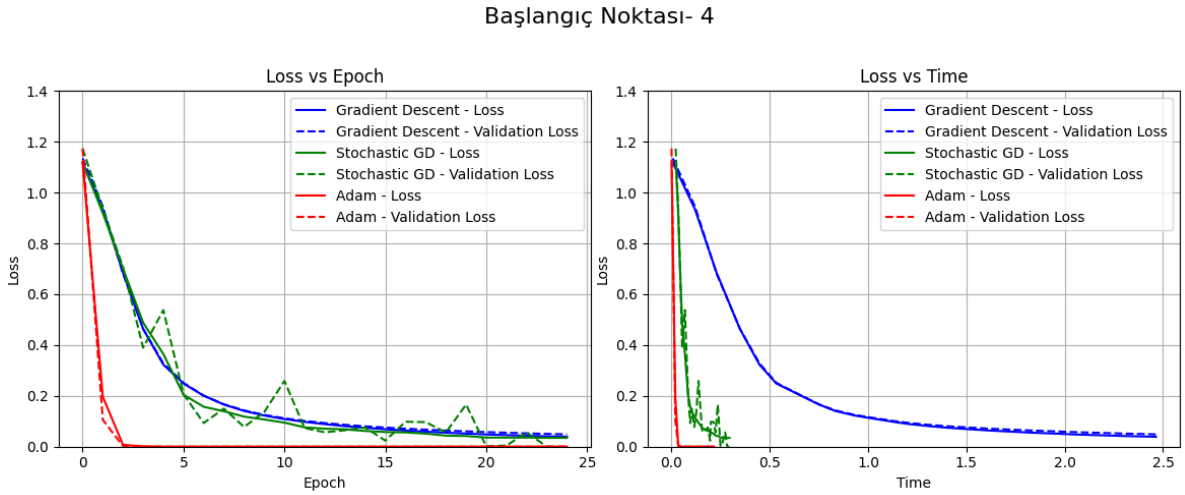

Figure 4: Başlangıç Noktası 4

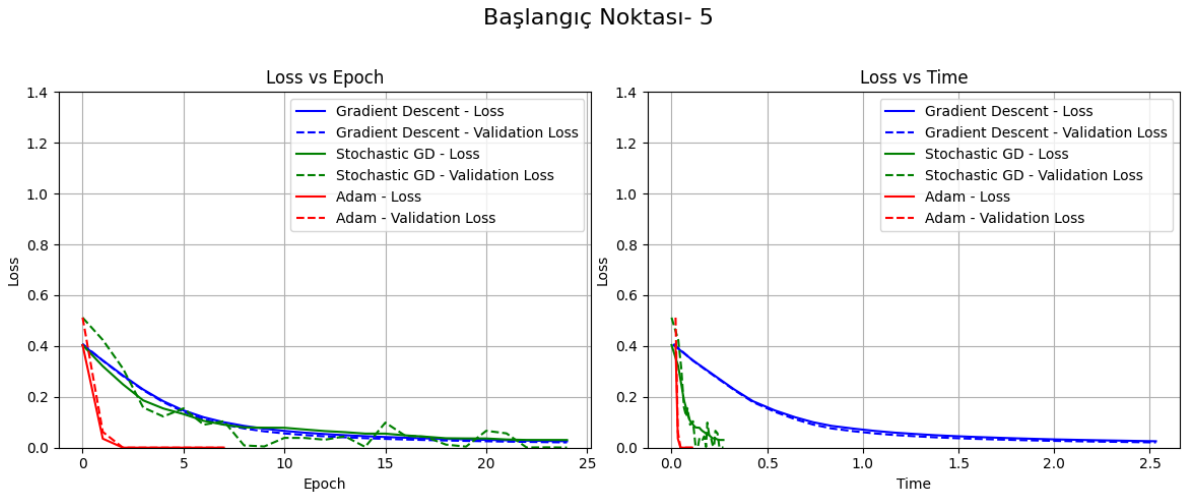

Figure 5: Başlangıç Noktası 5

Farklı noktalarda başlasalar bile genel olarak benzer grafikler ortaya çıkıyor. Bir örneği daha yakından inceleyelim ve yorumlayalım.

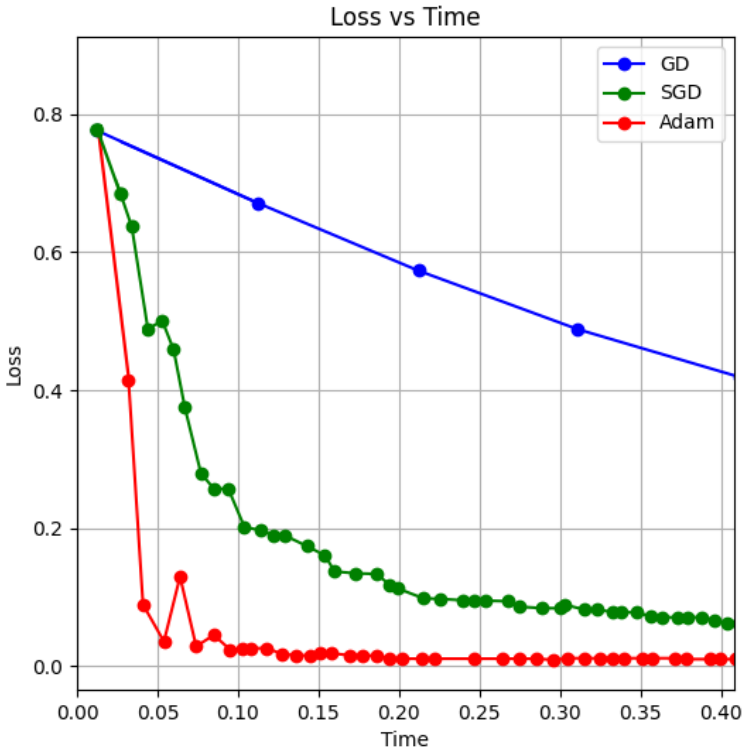

Figure 6: Detaylı Örnek

- Minumum noktaya ulaşma süresi : ADAM $<$ SGD $<$ GD
- Her bir iterasyonun süresi : SGD $<$ ADAM $<$ GD
- SGD ve ADAM stochastic metodlar olduğu için daha gürültülü bir şekilde ilerliyor.

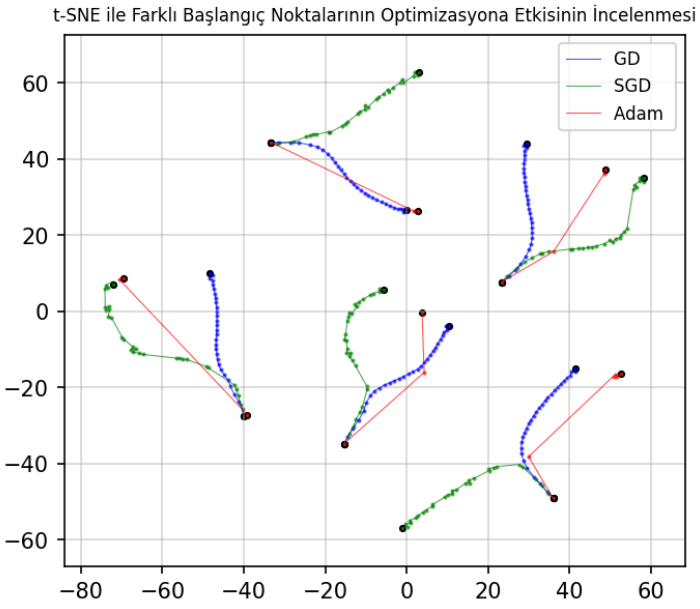

Figure 7: t-SNE Algoritması ile Görselleştirme 1

Yukarıda 5 farklı noktadan başlatılan modellerin ulaştıkları noktaları t-SNE sayesinde 2 boyutta görüyoruz.

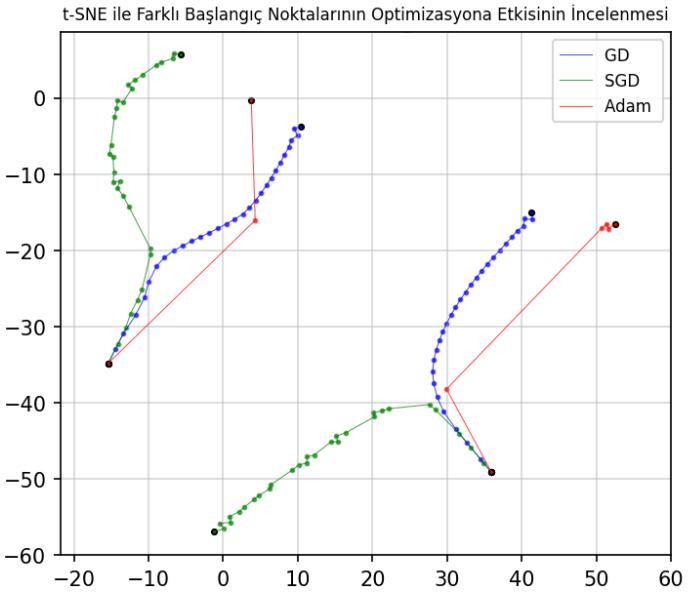

Figure 8: t-SNE Algoritması ile Görselleştirme 2

Grafiği incelediğimizde, aynı noktadan başlamalarına rağmen algoritmalarının farklı lokal minimumlara ulaşabildiğini gözlemlemekteyiz. Ayrıca, ADAM ve SGD loss grafiklerinde olduğu gibi zikzaklı bir ilerleme göstermektedir.

4 Kodlar ve 10 Gruplu Sınıflandırma

Projemin devamında, çoklu sınıflandırma işlemlerini gerçekleştirebilmek amacıyla modüler bir Sinir Ağı (Neural Network) yapmayı hedefledim. Bu doğrultuda, farklı programlama dillerinde geliştirilmiş sinir ağı örneklerini inceledim ve C dilinde struct yapılarından faydalanarak temel bir sinir ağı modeli yazdım.
[5]

```c
typedef struct
{
    double* weights; // Neurons Connections
    double* delta_weights; // Difference in weights
    double* firstMoment; // for ADAM
    double* secondMoment; // for ADAM
    double output_val;
    double gradient;

    int weightCount;
    int m_myIndex;
} Neuron;

typedef struct
{
    int neuronNum;
    Neuron** neurons;

} Layer;

typedef struct
{
    int* topology;
    int layerNum;
    Layer** layers;
}Net;

//---------------------- Neuron Codes ----------------------

Neuron* newNuron(int weightCount, int myIndex);
void freeNuron(Neuron* neuron);
void feedForwardNeuron(Neuron* neuron, Layer* prevLayer);
void updateWeightsNeuron(Neuron* neuron, Layer* nextLayer, double learningRate, int
    batchSize);
void updateWeightsNeuronADAM(Neuron* neuron, Layer* nextLayer, double learningRate, int
    batchSize, double beta1, double beta2);

// ---------------------- Layer Codes ----------------------

Layer* newLayer(int neuronNum, int nextNeuronNum);
void freeLayer(Layer* layer);
void feedForwardLayer(Layer* prevLayer, Layer* currLayer);

// ---------------------- Net Codes ----------------------

Net* newNet(int* topology, int layerNum);
void freeNet(Net* net);
void feedForwardNet(Net* net, double* inputVals, int inputSize);
void updateWeightsNet(Net* net, double learningRate, int batchSize);
void updateWeightsNetADAM(Net* net, double learningRate, int batchSize, double beta1,
    double beta2);
```

[5]Not: Onlu sınıflandırma işlemi için ikili sınıflandırmadan farklı olarak **Sigmoid** (σ) ve **Cross Entropy Loss** kullandım.

Sinir ağı küçük bileşenlerin bir araya gelmesiyle oluşuyor. En temel birim olan **nöron**, kendisinden sonraki katmanla olan ağırlıkları tutuyor. Her bir **katman** ise nöron listesini barındırıyor. Katmanların bir araya gelmesiyle bir **sinir ağını** tamamlamış oluyoruz.

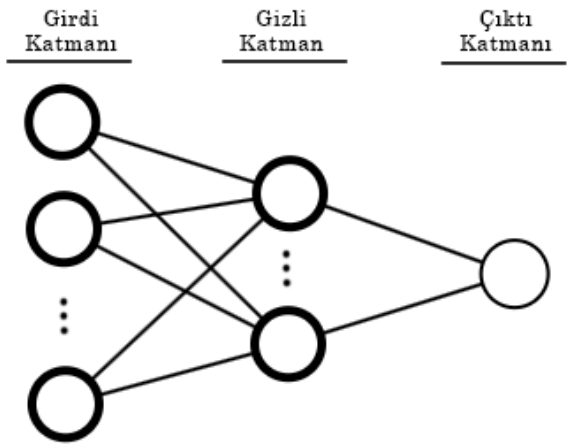

Figure 9: Örnek Sinir Ağı Modeli

Sinir ağında gradyanları hesaplayan, **backpropagation** yapan ve bazı hesaplamalar için gerekli olan fonksiyonlar aşağıdadır.

```
// ---------------------- Calculatinons ----------------------
double getRandom();

double activation(double x);

double derivActivation(double x);

double calculateErr(Net* net, double* targetVals);

void   softmaxNet(Net* net);

void   resetDeltaWeights(Net* net);

void   printNumber(double* values, int lenght);

void   testSample(Net* net, Data* data, int testIdx);

void   testResults(Net* net, Data* data, int splitIdx, int* correct, int* wrong);

void   generateRandomElements(int start, int end, int k, int* result);

// ---------------------- Net Calculatinons ----------------------
double sumDOW(Neuron* neuron, Layer* nextLayer);

void   calculateHiddenGrad(Layer* currentLayer, Layer* nextLayer);

void   calculateOutputGrad(double* targetVals, Layer* outputLayer);

void   backPropagation(Net* net, double* targetVals, int targetSize);
```

Gradient Descent algoritmasının kodu:

```
void trainGD(Net* net, Data* data, int epoch, double learningRate) {

int epochIdx, dataIdx, batchSize, i;
int dataCount = data->rowCount;
int splitIdx = (2 * dataCount) / 10; // 80-20 ratio. First 20 percent is for testing the model
batchSize = dataCount - splitIdx;
double loss = 0;
double valError = 1; // Must be greater than stop_error

//Train
epochIdx = 0;
    while (epochIdx < epoch && valError > STOP_ERROR)
    {
        epochIdx++;
        valError = 0;
        loss = 0;

        //Calculate gradient for each train sample and accumulate them
        for (dataIdx = splitIdx; dataIdx < dataCount; dataIdx++)
        {
            feedForwardNet(net, data->inputVals[dataIdx], data->colCount);
            softmaxNet(net);
            backPropagation(net, data->targetVals[dataIdx], data->numOfClasses);
            loss += calculateErr(net, data->targetVals[dataIdx]);
        }

        // Update weight based on average gradients
        updateWeightsNet(net, learningRate, batchSize);

        //Validation Error calculation
        for (dataIdx = 0; dataIdx < splitIdx; dataIdx++)
        {
            feedForwardNet(net, data->inputVals[dataIdx], data->colCount);
            softmaxNet(net);
            valError += calculateErr(net, data->targetVals[dataIdx]);
        }
        loss = loss / batchSize;
        valError = valError / splitIdx;
        printf("Loss :%lf ,", loss);
        printf(" Validation Error:%lf\n", valError);

    }

return;
}
```

Stochastic Gradient Descent algoritmasının kodu:

```
void trainSGD(Net* net, Data* data, int epoch, int batchSize, double learningRate) {

    int epochIdx, dataIdx, shuffledIdx, trainSize, i;
    int dataCount = data->rowCount;
    int splitIdx = (2 * dataCount) / 10; // 80-20 ratio. First 20 percent is for testing the model
    trainSize = dataCount - splitIdx;
    double loss = 0;
    double valError = 1;//Must be greater than stop_error at first

    if (batchSize >= trainSize || batchSize == 0) {
        printf("Batch size is cant be equal to or greater than train size");
        return;
    }


    //Train
```

```
    epochIdx = 0;
    while (epochIdx < epoch && valError > STOP_ERROR)
    {
        valError = 0;
        loss = 0;
        epochIdx++;

        // Shuffle indices and select random elements
        int* indices = (int*)malloc(trainSize * sizeof(int));
        generateRandomElements(splitIdx, dataCount - 1, batchSize, indices);

        //Calculate gradients from bacth samples and accumulate them
        for (dataIdx = 0; dataIdx < batchSize; dataIdx++)
        {
            shuffledIdx = indices[dataIdx];
            feedForwardNet(net, data->inputVals[shuffledIdx], data->colCount);
            softmaxNet(net);
            backPropagation(net, data->targetVals[shuffledIdx], data->numOfClasses);
            loss += calculateErr(net, data->targetVals[shuffledIdx]);
        }

        // Update weight based on average gradients
        updateWeightsNet(net, learningRate, batchSize);
        free(indices); // free random indices

        //Validation Error calculation
        for (dataIdx = 0; dataIdx < splitIdx; dataIdx++)
        {
            feedForwardNet(net, data->inputVals[dataIdx], data->colCount);
            softmaxNet(net);
            valError += calculateErr(net, data->targetVals[dataIdx]);
        }
        loss = loss / batchSize;
        valError = valError / splitIdx;
        printf("Loss :%lf ,", loss);
        printf(" Validation Error:%lf\n", valError);

    }

}
```

ADAM algoritmasının kodu:

```
void trainADAM(Net* net, Data* data, int epoch, int batchSize, double learningRate) {

    int epochIdx, dataIdx, shuffledIdx, trainSize, i;
    int dataCount = data->rowCount;
    int splitIdx = (2 * dataCount) / 10; // 80-20 ratio. First 20 percent is for testing
    the model
    trainSize = dataCount - splitIdx;
    double loss = 0;
    double valError = 1; // Must be greater than stop_error

    if (batchSize >= trainSize || batchSize == 0) {
        printf("Batch size is cant be equal to or greater than train size");
        return;
    }


    //Train
    epochIdx = 0;
    while (epochIdx < epoch && valError > STOP_ERROR)
    {
        valError = 0;
        loss = 0;
        epochIdx++;

        // Shuffle indices and select random elements
```

```
25          int* indices = (int*)malloc(trainSize * sizeof(int));
26          generateRandomElements(splitIdx, dataCount - 1, batchSize, indices);
27
28
29          for (dataIdx = 0; dataIdx < batchSize; dataIdx++)
30          {
31              shuffledIdx = indices[dataIdx];
32              feedForwardNet(net, data->inputVals[shuffledIdx], data->colCount);
33              softmaxNet(net);
34              backPropagation(net, data->targetVals[shuffledIdx], data->numOfClasses);
35              loss += calculateErr(net, data->targetVals[shuffledIdx]);
36          }
37
38          // Update weight based on average gradients
39          // ADAM's update different than GD and SGD
40          updateWeightsNetADAM(net, learningRate, batchSize, 0.9, 0.99);
41          free(indices); // free random indices
42
43          //Validation Error calculation
44          for (dataIdx = 0; dataIdx < splitIdx; dataIdx++)
45          {
46              feedForwardNet(net, data->inputVals[dataIdx], data->colCount);
47              softmaxNet(net);
48              valError += calculateErr(net, data->targetVals[dataIdx]);
49          }
50          loss = loss / batchSize;
51          valError = valError / splitIdx;
52          printf("Loss :%lf ,", loss);
53          printf(" Validation Error:%lf \n", valError);
54
55      }
56  }
```

Kodu tamamladıktan sonra artık istediğimiz topolojiyi girerek Sinir Ağı oluşturabiliriz ve 10 sınıfta test edebiliriz. Farklı topolojileri test etmek için MNIST veri setindeki 60.000 örneği 80-20 oranında bölerek modeli eğittim. Eğitirken ADAM algoritmasını kullandım. Birden farklı gizli katman kombinasyonu denedim. Şu ana kadar en iyi çalışan topoloji 784-128-32-10 şeklinde oldu. Test setinde **%96.8 başarı** oranına kadar çıkabildim.

```
---- Test Results ----
Wrong    :265
Correct  :8135
Accuracy :96.845
```

Figure 10: Test Verisinde %96.8 Doğruluk

(a) Örnek Test 1 **(b)** Örnek Test 2

Figure 11

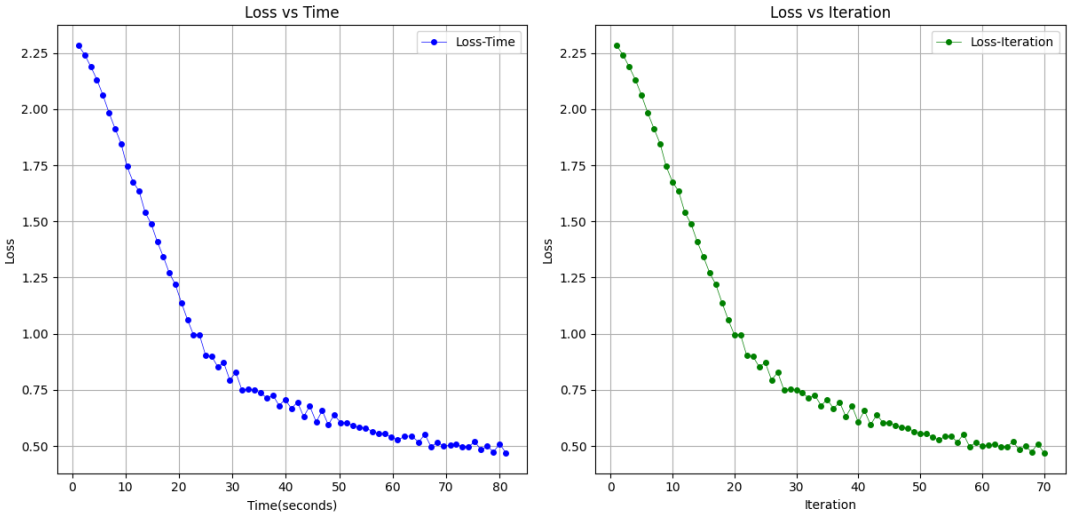

Figure 12: 10 Sınıflı Sinir ağının in Eğitim loss ları

5 Sonuç

Bu proje, görüntü sınıflandırma problemleri üzerine çalışırken yalnızca algoritmaların matematiksel temellerini değil, aynı zamanda Sinir Ağı modellerinin temellerini derinleme-

sine anlamamı sağladı. *Gradient Descent, Stochastic Gradient Descent ve ADAM* algoritmalarını karşılaştırırken, her bir algoritmanın farklı veri dağılımlarında ve başlangıç koşullarında nasıl davrandığını gözlemleme fırsatı buldum. Bu süreç, optimizasyon tekniklerinin avantajları ve dezavantajları konusunda pratik bir anlayış kazandırdı. Ayrıca, farklı topoloji ve parametreleri denemek, model tasarımında deneme-yanılma yönteminin önemini gösterdi.

Sonuç olarak, bu proje hem teorik bilgiyi pratiğe dökme hem de kodlama yeteneklerini geliştirme açısından oldukça öğretici bir deneyim oldu.

6 Kaynakça

- 3Blue1Brown. (2017, November 3). *Backpropagation calculus | DL4* [Video]. Youtube
- Tsoding Daily. (2023, May 16). *Making a new deep learning framework (ML in C EP.02)* [Video]. Youtube
- Nicolai Nielsen. (2022, June 20). *Coding a Neural Network from Scratch in C: No Libraries Required* [Video]. Youtube
- Mark Kraay. (2022, January 16). *neural network from scratch in C* [Video]. Youtube
- far1din. (2023, January 23). *Convolutional Neural Networks from Scratch | In Depth* [Video]. Youtube
- Futurology — An Optimistic Future. (2020, December 19). *Convolutional neural networks explained (CNN visualized)* [Video]. Youtube