```c
typedef struct {
    double data[VECTOR_SIZE];
    double type;
} Vector;
```

```c
void shuffleVectors(Vector* vectors, int size) {
    srand(time(NULL));
    int i,j;
    for (i = size - 1; i > 0; i--) {
        j = rand() % (i + 1);
        Vector temp = vectors[i];
        vectors[i] = vectors[j];
        vectors[j] = temp;
    }
}
```

```c
double matrisCarpim (double* w, Vector* v){
    double result = 0.0;
    int i;
    for(i = 0; i < VECTOR_SIZE; i++){
        result += w[i] * v->data[i];
    }
    return result;
}

double tanhTurev (double a){
    return 1.0 - tanh(a) * tanh(a);
}

void initializeWeight(double* weight, double value){
    int i;
    for(i = 0 ; i<VECTOR_SIZE ; i++){
        weight[i] = value;
    }
}
```

```c
void writeResultToCSV(FILE* file, int epoch, double loss, double time) {
    fprintf(file, "%d,%.6f,%.6f\n", epoch, loss, time);
}
```

N	50	(fotoğrafların kenar boyutları)
VECTOR_SIZE	(N * N + 1)	= 2501
GENERAL_SET_SIZE	202	(101 kedi + 101 köpek)
MAX_ITERATION	500	
EXPECTED_ERROR	0.001	
LEARNING_RATE	0.0001	

Gradient Descent

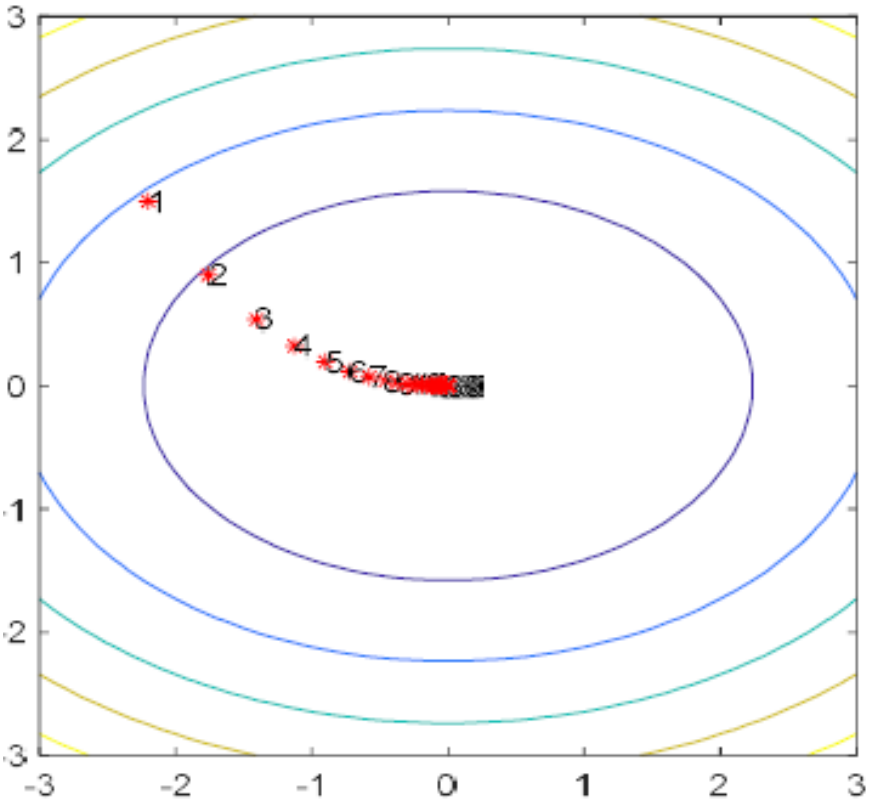
$\hat{Y} = \tanh(w \cdot x)$

$\partial \tanh(a) / \partial a = 1 - (\tanh(a))^2$

$\nabla w = (\hat{y} - y) \cdot (1 - (\hat{y})^2) \cdot x$

n = training set size

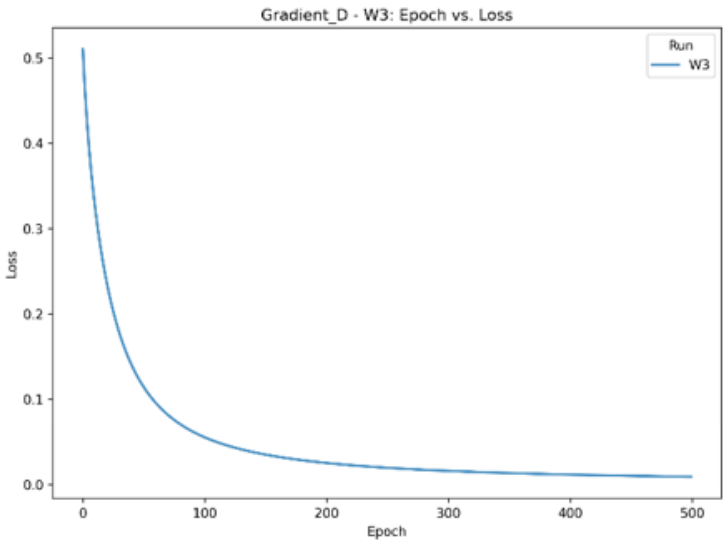
$$x_{t+1} = x_t - \eta_t \nabla f(x_t) = x_t - \eta_t \sum_{i=1}^{n} \nabla f_i(x_t)$$

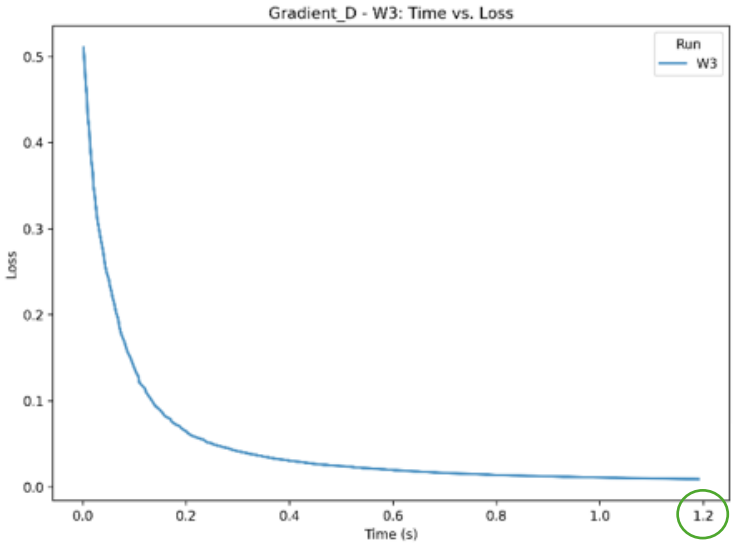


```c
double gradientDescent(Vector* vec, double* weights, int setSize, FILE* file){
    clock_t start = clock();
    double loss=1, error, totalError=0;
    double temp, predictedY, gradient;
    int i, j, k, step=0;
    double zaman=0;

    while(loss > EXPECTED_ERROR && step <MAX_ITERATION)
    {
        totalError = 0;
        for(i=0;i<setSize;i++){
            temp = matrisCarpim(weights, &vec[i]);
            predictedY = tanh(temp);
            error = predictedY - vec[i].type; //        yeliz(1)

            totalError += (error*error)/2;

            gradient = error * tanhTurev(temp) ;

            for(j = 0; j<VECTOR_SIZE ; j++){
                weights[j] -= LEARNING_RATE * gradient * vec[i].data[j];
            }
        }
        loss = totalError / setSize;  // Ortalama loss
        zaman = (double)(clock() - start) / CLOCKS_PER_SEC;
        writeResultToCSV(file, step, loss, zaman);
        step++;

        if(step%100 == 0&& DEV == 1)    //DEBUG
            printf("Step: %d, Loss: %lf, Zaman: %lf\n", step, loss, zaman);
    }

    return loss;
}
```

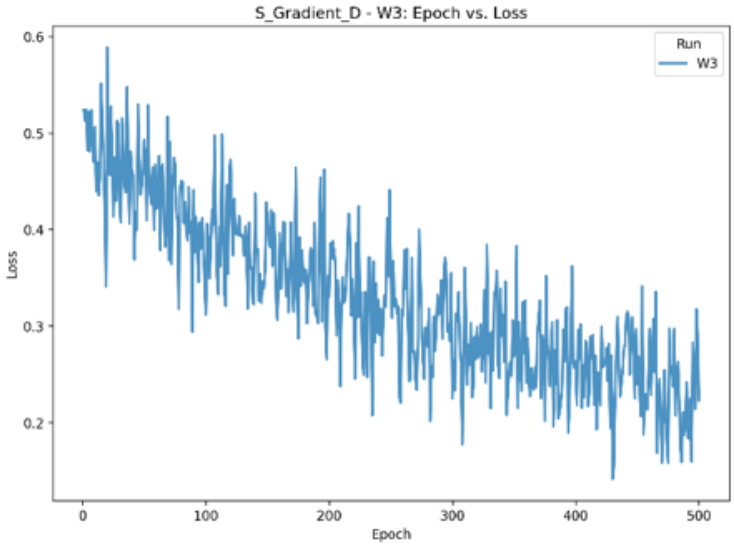
3. Başlangıç w değeri = 0.00

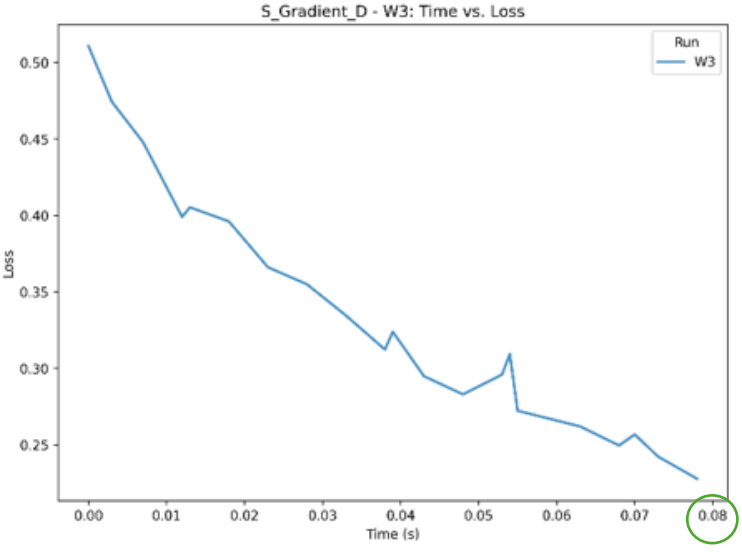
Graddes



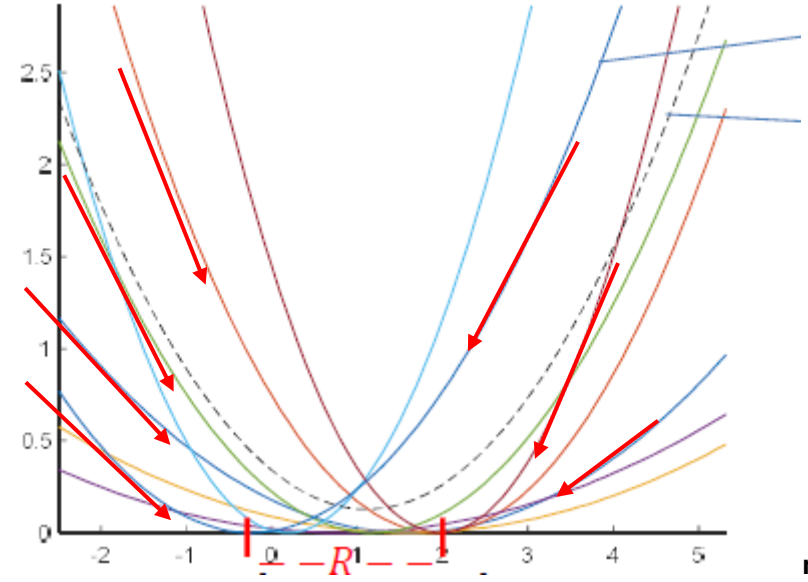


Sgd

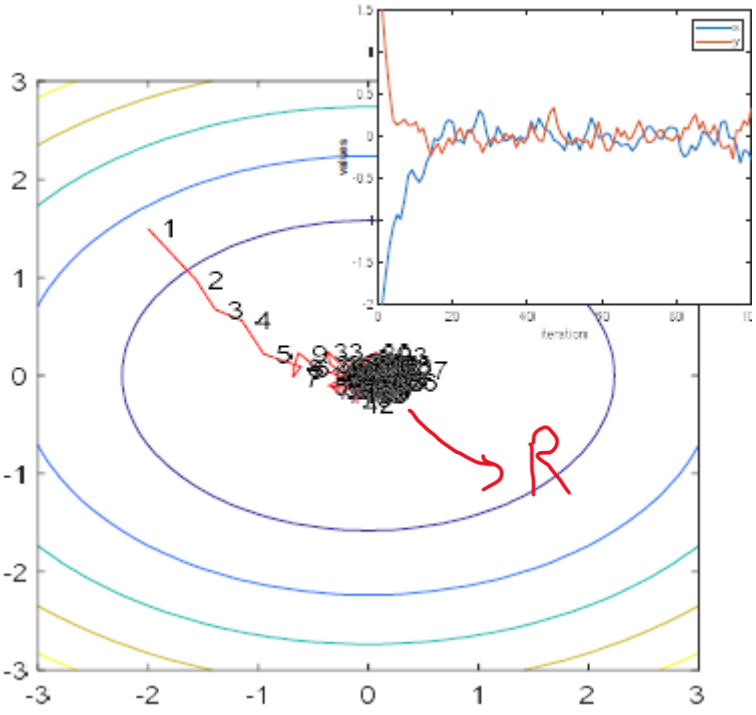




Stochastic Gradient Descent



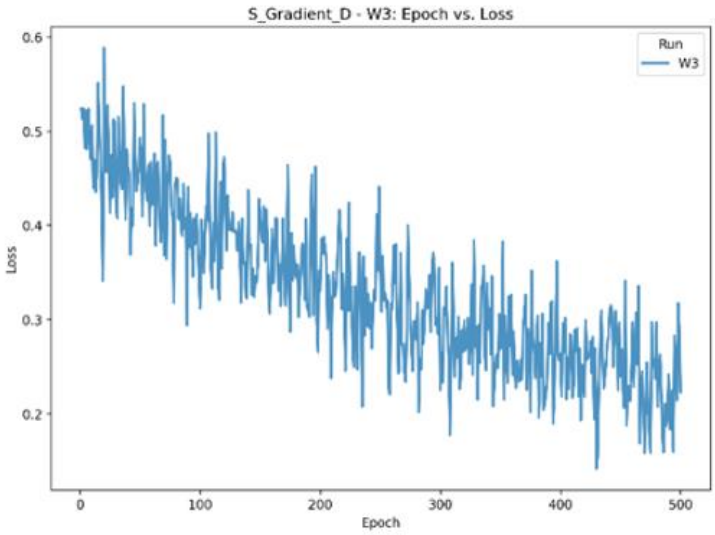
mini batch = 10

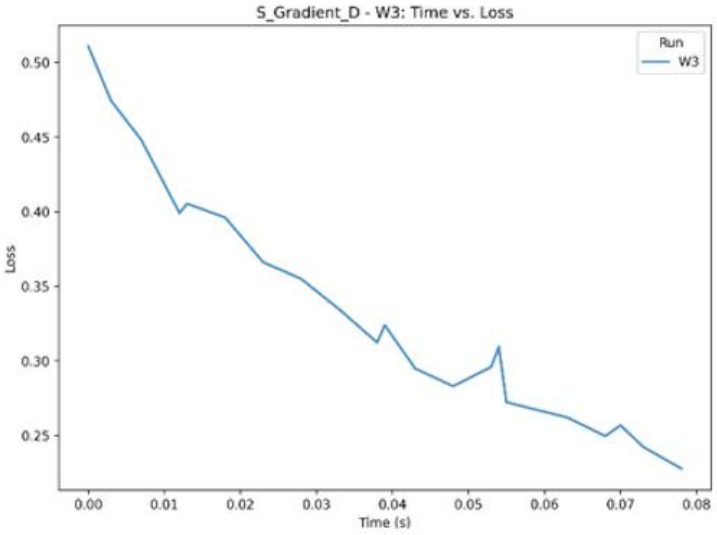



```
if (fmod(zaman, 0.005f) == 0.0f) {
    eps *= 0.9f;
}
```

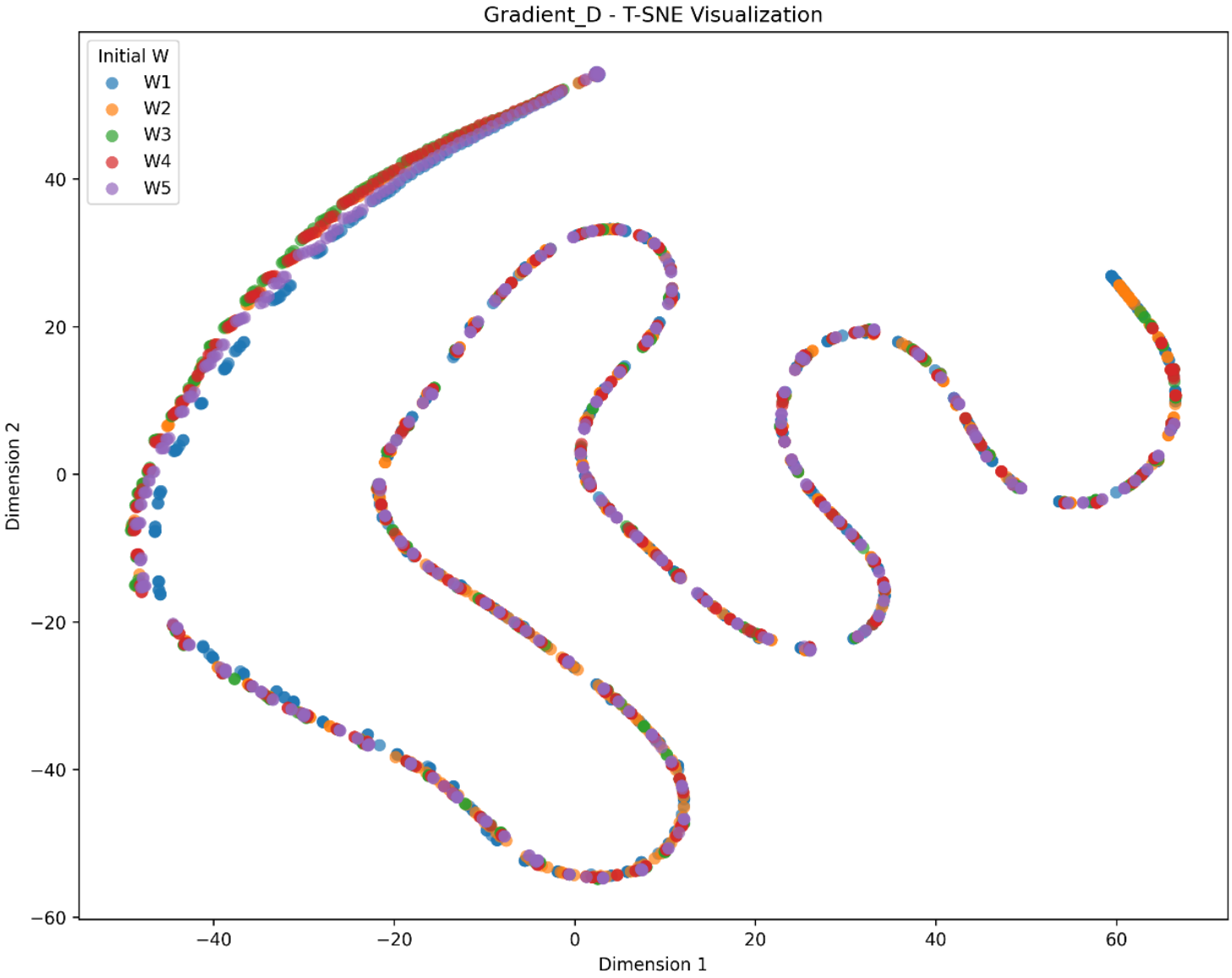
```c
double stochasticGradientDescent(Vector* vec, double* weights, int setSize, FILE* file){
    clock_t start = clock();
    double loss=1, error, totalError=0;
    double temp, predictedY, gradient;
    int miniBatch = 10;
    int i, j, k, step=0;
    float zaman=0 , eps = LEARNING_RATE;

    while(loss > EXPECTED_ERROR && step <MAX_ITERATION)
    {
        totalError = 0;
        for(i=0;i<miniBatch;i++){ //mini-batch ile gradyanti dengeliyoruz 10 tane kullandik
            k = rand() % TRAINING_SET_SIZE*0.8f;
            temp = matrisCarpim(weights, &vec[k]);
            predictedY = tanh(temp);
            error = predictedY - vec[k].type;

            totalError += (error*error)/2;

            gradient = error * tanhTurev(temp) ;

            for(j = 0; j<VECTOR_SIZE ; j++){
                weights[j] -= eps * gradient * vec[k].data[j];
            }

        }
        loss = totalError / miniBatch;
        step++;
        zaman = (double)(clock() - start) / CLOCKS_PER_SEC;
        writeResultToCSV(file, step, loss, zaman);

        //SGD de sona dogru titresimi azaltmak icin LEARNING_RATE azaltiyoruz
        if (fmod(zaman, 0.005f) == 0.0f) {
            eps *= 0.9f;
        }

        if(step%100 == 0 && DEV == 1)   //DEBUG
            printf("Step: %d, Loss: %lf, Zaman: %lf\n", step, loss, zaman);

    }
    return loss;
}
```

Sgd


S_Gradient_D - W3: Epoch vs. Loss
Run
W3
Loss
Epoch
0.6
0.5
0.4
0.3
0.2
0
100
200
300
400
500


S_Gradient_D - W3: Time vs. Loss
Run
W3
Loss
Time (s)
0.50
0.45
0.40
0.35
0.30
0.25
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08

Gradient descent



Stochastic Gradient descent