BLM2642

Bilgisayar Mühendisliği İçin Diferansiyel Denklemler

Deniz Ege Dereli

22011066

Giriş:

Bu rapor, differansiyel denklemler dersi için hazırladığım proje ödevim ile ilgilidir. Ödevde, verilen bir örneğin önceden belirlenmiş iki kategoriden hangisine ait olduğu konusunda bir tahmin yapılır. Tahmin yapmak için gerekli model, dışarıdan sağlanan veriler ile eğitilir. Bu eğitme işlemi sırasında kullanılan farklı yöntemler karşılaştırılır. Ödevde oluşturulan model, bir perceptron modelidir.

Modelin Eğitim Amacı:

Bu projedeki model, bir cümlenin olumlu mu olumsuz mu olduğunu tahmin etmek için geliştirilmiştir.

Proje Klasörü:

Proje ile ilgili dosyaların hepsi, proje klasöründe bulunmalıdır. Aşağıda, proje klasöründe bulunabilecek dosyalar ve bu dosyaların görevleri kısaca listelenmiştir.

0-preprocess-samples.py -- Örnekleri düzenler

1-generate-dictionary.c -- Örneklerden sözlük üretir

2-generate-hot.c -- Sözlük ve örnekleri kullanark hot vektörleri üretir

3-train.c -- Modeli eğitir

4-test.c -- Modeli test eder

allsamples.txt -- Bütün örnekler (%100)

trainingsamples.txt -- Eğitim için kullanılan örnekler (%80)

testingsamples.txt -- Test için kullanılan örnekler (%20)

dictionary.txt -- Sözlük dosyası

model.txt -- Eğitilen modelin son parametreleri

parameters.log -- Parametrelerin iterasyonlarca değişimi

training.log -- Hatanın iterasyonlarca değişimi ve eğitim süresi

testinghv.txt -- Test örneklerinin hot vektörleri

traininghv.txt -- Eğitim örneklerinin hot vektörleri

testingtrues.txt -- Test örneklerinin gerçek değerleri

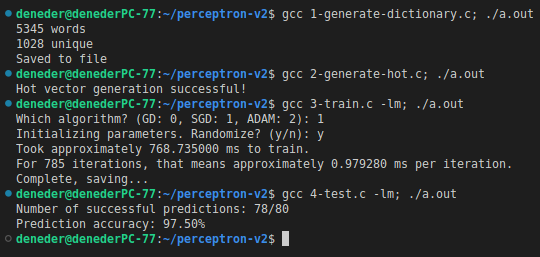
trainingtrues.txt -- Eğitim örneklerinin gerçek değerleri

graph.py -- Belirlenen parametre için grafikler çizer

tsne-graph.py -- Parametrelerin değişimini grafik üzerine çizer

Modelin Eğitimi ve Test Edilmesi:

Modelin eğitilmesi ve test edilmesi için sırasıyla 1, 2, 3, 4 şeklinde numaralandırılmış C kaynak kodlu dosyaların derlenip çalıştırılması ve ekrandaki talimatların takip edilmesi yeterlidir.



Örnek Toplama:

Örnek toplama, bir modeli eğitmenin ilk ve en önemli adımıdır. Örneklerin uygunluğu, modelin doğruluğunu doğrudan etkiler. Eğer örnekler, modelin eğitilmesi için uygun değilse veya yetersizse modelin doğruluğu düşük olacaktır.

Kendi projemde örnekler, üç ayrı dosyada tutuluyor. allsamples.txt sözlük üretiminde kullanılır ve toplanmış tüm örnekleri içerir. Trainingsamples.txt, model eğitilirken (tüm örneklerin %80’ini içerir), Testingsamples.txt ise model test edilirken (tüm örneklerin %20’sini içerir) kullanılır. Bu bakımdan, allsamples.txt’nin diğer iki dosyanın birleşimi olduğunu söyleyebiliriz.

Ben, kendi projem için kullandığım olumlu ve olumsuz cümle örneklerini internetteki çeşitli ücretsiz dil üretim modellerini kullanarak oluşturdum.

Örneklerin İşlenmesi:

Örneklerin işlenmesi, iki adımda gerçekleştirilir. İlk adım, sözlük oluşturmadır. Bu adımda tüm örnekler incelenir, özgün kelimeler kullanılarak bir sözlük dizisi oluşturulur. Bu iş, 1-generate-dictionary.c kaynak kodlu program ile gerçekleştirilebilir. Sonuç, dictionary.txt dosyasına kaydedilir.

İkinci adımda, her bir örnek için bir “hot vector” karşılık oluşturulur. Bu iş, 2-generate-hot.c kaynak kodlu program ile gerçekleştirilir. Hot vektörler, örneklerin sayısal karşılıklarıdır. Öncelikle, uzunluğu sözlük kadar olan, sıfırlardan oluşan bir dizi tanımlanır. Ardından, örnekte bulunan kelimelerin sözlükteki yerleri belirlenir. Hot vektör üzerinde bu konumlara 1 yazılır. Son olarak bu yapılar, daha sonra yeniden kullanılmak için bir dosyaya kaydedilir.

Eğitim için kullanılan örneklerin hot vektörleri traininghv.txt, test için kullanılan örneklerin hot vektörleri testinghv.txt dosyalarına kaydedilir.

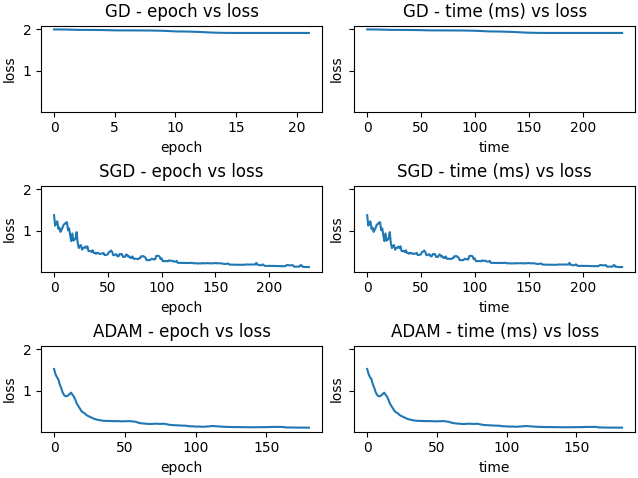
Modelin Eğitimi:

Model, y = tanh(wx) fonksiyonudur. W, parametreler vektörüdür. X, değişkenler yani hot vektördür. Tanh, -1/+1 arasında değer döndürür. Bu bakımdan -1 kategorilerden birini, +1 diğerini ifade eder. Model eğitildikten sonra verdiği değer, -1/+1’den hangisine yakınsa ona yuvarlanır. Bu sayede bir tahmin gerçekleştirilmiş olur.

Modelin eğitilmesi için üç farklı yöntem kullanılabilir: Gradient Descent (GD), Stochastic Gradient Descent (SGD), Adaptive Movement Estimation (ADAM). 5 ayrı başlangıç değeri içinbu yöntemlerin grafikleri, ve bu grafiklerin karşılaştırılması aşağıda verilmiştir.

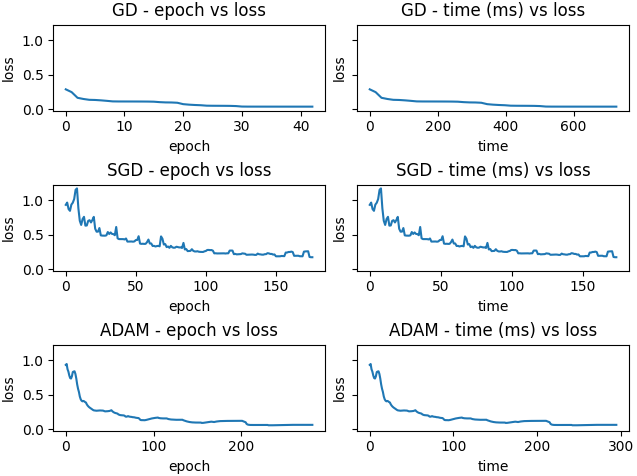
Grafikler, matplotlib kütüphanesi kullanılarak oluşturulmuştur. Yöntemler, maksimum 1000 iterasyon ile sınırlanmıştır. Adım büyüklüğü (EPS) 0.1 olarak belirlenmiştir.

Başlangıç Parametreleri: rastgele

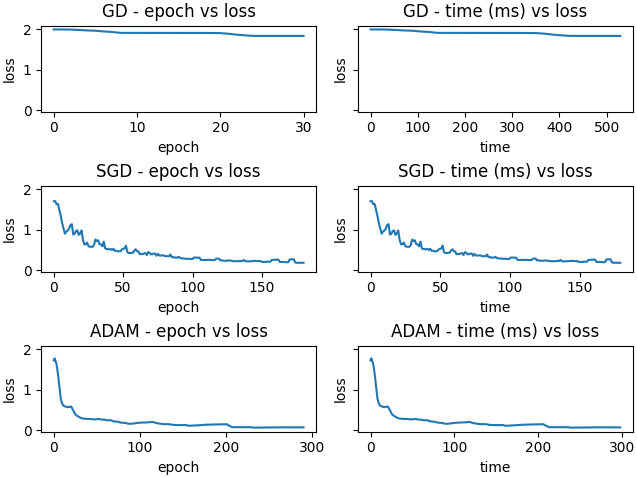
  
Rastgele parametreler ile başlatıldığında, ADAM hem iterasyon sayısı hem de süre bakımından en verimli yöntemdir. GD, yalnızca 22 iterasyon yapmış olmasına rağmen çalışma zamanı en uzun olan yöntemdir.

Yalnızca GD bir lokal minimuma takılmıştır. Bunun nedeni, SGD’nin rastgele bir örnek seçerek bu minimumdan kurtulmuş olması, ADAM’ın ise momentumu ile bu minimumdan çıkabilmiş olmasıdır.

Başlangıç Parametreleri: 0.001

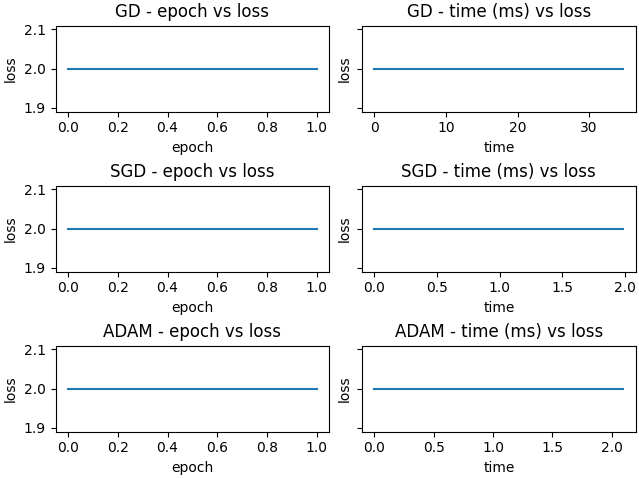
  
Burada da yine süre bakımından en kötü yöntemin GD olduğu görülmektedir. SGD, kısmen yüksek bir hata oranıyla da olsa ortalama 170ms’de yakınsamıştır. Bu bakımdan, en hızlı yakınsayan yöntem olmuştur.

Başlangıç Parametreleri: 0.1

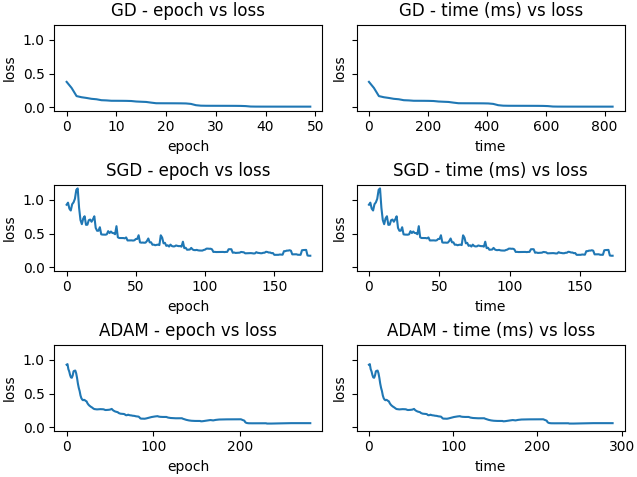


GD, yine bir lokal minimuma takılmıştır.

Başlangıç Parametreleri: 0.5

  
Parametre değerlerinin çok yüksek olmasından dolayı türevler 0 çıkmış, bu nedenle parametrelerde değişiklik olmamış ve yöntemler ilk adımda yakınsamıştır.

Başlangıç Parametreleri: 0



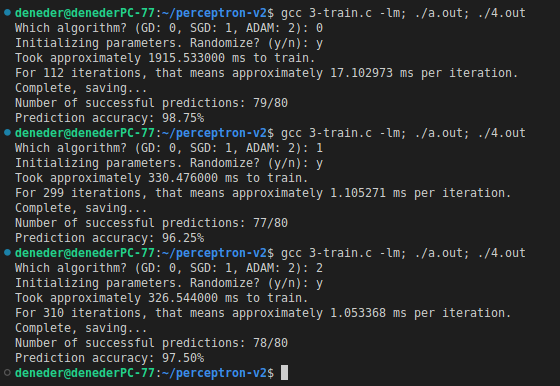
Yine hata ve süre bakımından en verimli yöntem ADAM’dır. GD, adım sayısı olarak en düşük olsa da süre bakımından en verimsiz yöntemdir.

GD, bu örnekte her iterasyonu ADAM’dan yaklaşık 16 kat daha uzun sürede tamamlar. SGD’nin bir adım süresi, ADAM’ın bir adım süresiyle yaklaşık olarak aynıdır. Bu bakımdan da ADAM’ın en verimli yöntem olduğu söylenebilir.

Modelin Test Edilmesi:

Model, en başta ayrılmış olan %20’lik örnekler kullanılarak test edilir. Modelin yaptığı tahminler ve bu tahminlerin doğruluğu incelenir. Bu bilgiler ışığında modelin başarılı olup olmadığına karar verilir. Model yeterince başarılı değilse başarı düzeyinin nasıl arttırılabileceği düşünülür.

İterasyon limiti 1000, EPS 0.05, parametreler rastgele ayarlanmıştır. Modelin üç farklı yöntemle eğitilmesi ve eğitim sonuçları aşağıda verilmiştir.

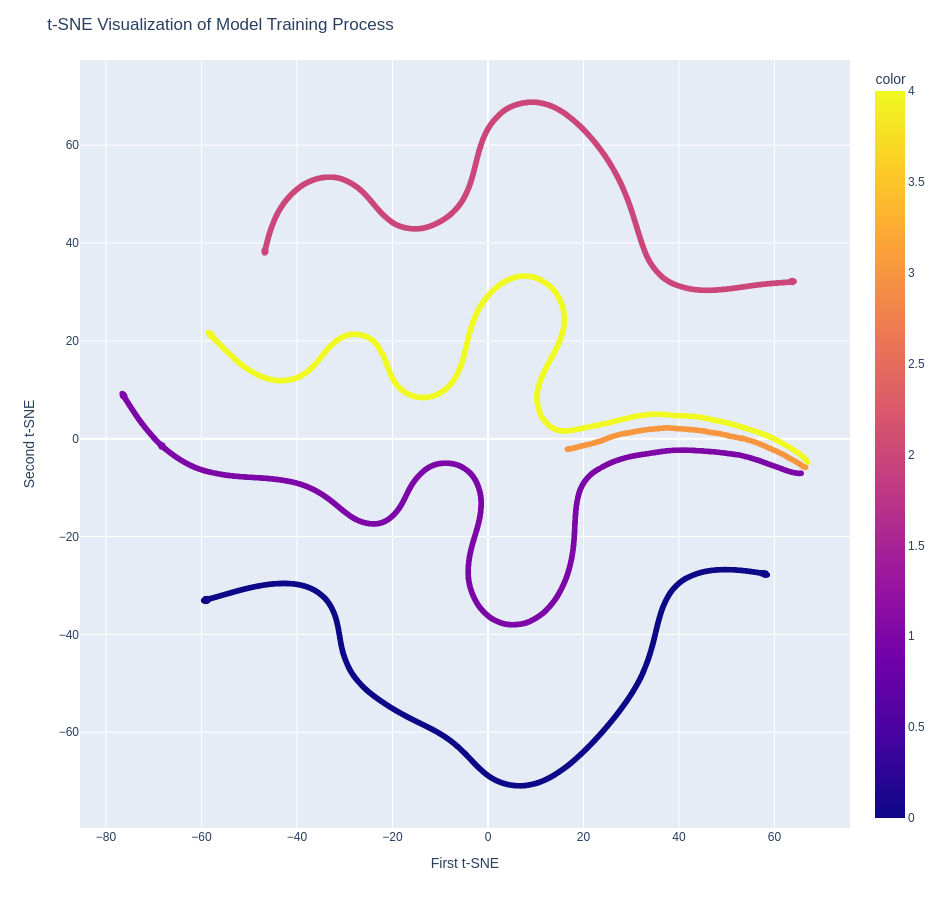


Optimizasyon Sürecinin 2-Boyutta Gösterimi:

Aşağıda, modelin 5 farklı parametre değerinden başlayarak SGD ile eğitiminin 2-boyutta gösterimi verilmiştir.

Her bir iterasyonda parametreler vektörünün değişimi, parameters.log dosyasına kaydedilir. Ancak parametreler vektörü, sözlükteki kelime sayısı (1000+) kadar boyuttan oluştuğu için xy-koordinat sisteminde doğrudan gösterilemez. Optimizasyon sürecinin iki boyutta gösterilebilmesi için de parametreler vektörünün iki boyuta indirgenmesi gerekir. Bu iş için de t-SNE ve PCA yöntemleri kullanılır.

T-SNE, 50 boyut ve üstü uzayları doğrudan indirgemekte çok iyi performans gösteremeyebilir. Bu nedenle, parametreler vektörünü önce PCA kullanarak 50 boyuta, ardından t-SNE ile 2 boyuta indirdim.



Numaralar (renkler) ve parametrelerin başlangıç değerleri:

0 – 0.01 (lacivert)

1 – 0.001 (mor)

2 - -0.03 (pembe)

3 – 0 (turuncu)

4 - -0.0009 (sarı)

Yukarıda da görüldüğü üzere 1, 3, 4 aynı lokal minimum noktaya yakınsamaktadır. Bu bakımdan 3’ün şanslı olduğu söylenebilir, çünkü minimuma yakın bir noktada başlamıştır. 0 ve 2 ise ayrı lokal minimumlara yakınsamıştır.

Burada iterasyon sınırı 1500, adım büyüklüğü (EPS) 0.01 belirlenmiştir. Grafik, Python ile birlikte plotly, scikit-learn, numpy kütüphaneleri kullanılarak oluşturulmuştur.

Kodlar:

1-generate-dictionary.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_WORD 300

int main()

{

int word\_in\_dict, unique\_words = 0, sample\_word\_i = 0, sample\_word\_count = 1;

FILE \*sPtr, \*dPtr;

char \*\*dict, ch, sample\_word[MAX\_WORD];

sPtr = fopen("allsamples.txt", "r");

dPtr = fopen("dictionary.txt", "w");

if (sPtr == NULL || dPtr == NULL)

{

puts("!! Unable to access 'samples.txt' or 'dictionary.txt'\n");

return -1;

}

// Count number of words in samples

while ((ch = fgetc(sPtr)) != EOF)

{

if (ch == ' ' || ch == '\n') sample\_word\_count++;

}

printf("%d words\n", sample\_word\_count);

// Allocate memory for dictionary

dict = calloc(sample\_word\_count, sizeof(char \*));

for (int i = 0; i < sample\_word\_count; i++)

{

dict[i] = calloc(MAX\_WORD, sizeof(char));

dict[i][0] = '\0';

}

// puts("Memory allocated");

// Find unique words

rewind(sPtr);

do {

ch = fgetc(sPtr);

if (ch == ' ' || ch == '\n' || ch == EOF)

{

sample\_word[sample\_word\_i] = '\0';

sample\_word\_i = 0;

word\_in\_dict = 0;

for (int i = 0; i < sample\_word\_count; i++)

{

if (strcmp(sample\_word, dict[i]) == 0)

{

word\_in\_dict = 1;

break;

}

}

if (word\_in\_dict == 0)

{

for (int i = 0; i < sample\_word\_count; i++)

{

if (dict[i][0] == '\0')

{

strcpy(dict[i], sample\_word);

break;

}

}

unique\_words++;

}

}

else sample\_word[sample\_word\_i++] = ch;

// printf("%s\n", sample\_word);

}

while (ch != EOF);

printf("%d unique\n", unique\_words);

// for (int i = 0; i < unique\_words; i++) printf("%s\n", dict[i]);

// Save dictionary array to file

fprintf(dPtr, "%d\n", unique\_words);

for (int i = 0; i < unique\_words; i++) fprintf(dPtr, "%s\n", dict[i]);

puts("Saved to file");

for (int i = 0; i < sample\_word\_count; i++) free(dict[i]);

free(dict);

fclose(sPtr);

fclose(dPtr);

return 0;

}

2-generate-hot.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_WORD 300

int main()

{

int word\_in\_dict, unique\_words = 0, sample\_word\_i = 0, sample\_word\_count = 1;

FILE \*sPtr, \*dPtr;

char \*\*dict, ch, sample\_word[MAX\_WORD];

sPtr = fopen("allsamples.txt", "r");

dPtr = fopen("dictionary.txt", "w");

if (sPtr == NULL || dPtr == NULL)

{

puts("!! Unable to access 'samples.txt' or 'dictionary.txt'\n");

return -1;

}

// Count number of words in samples

while ((ch = fgetc(sPtr)) != EOF)

{

if (ch == ' ' || ch == '\n') sample\_word\_count++;

}

printf("%d words\n", sample\_word\_count);

// Allocate memory for dictionary

dict = calloc(sample\_word\_count, sizeof(char \*));

for (int i = 0; i < sample\_word\_count; i++)

{

dict[i] = calloc(MAX\_WORD, sizeof(char));

dict[i][0] = '\0';

}

// puts("Memory allocated");

// Find unique words

rewind(sPtr);

do {

ch = fgetc(sPtr);

if (ch == ' ' || ch == '\n' || ch == EOF)

{

sample\_word[sample\_word\_i] = '\0';

sample\_word\_i = 0;

word\_in\_dict = 0;

for (int i = 0; i < sample\_word\_count; i++)

{

if (strcmp(sample\_word, dict[i]) == 0)

{

word\_in\_dict = 1;

break;

}

}

if (word\_in\_dict == 0)

{

for (int i = 0; i < sample\_word\_count; i++)

{

if (dict[i][0] == '\0')

{

strcpy(dict[i], sample\_word);

break;

}

}

unique\_words++;

}

}

else sample\_word[sample\_word\_i++] = ch;

// printf("%s\n", sample\_word);

}

while (ch != EOF);

printf("%d unique\n", unique\_words);

// for (int i = 0; i < unique\_words; i++) printf("%s\n", dict[i]);

// Save dictionary array to file

fprintf(dPtr, "%d\n", unique\_words);

for (int i = 0; i < unique\_words; i++) fprintf(dPtr, "%s\n", dict[i]);

puts("Saved to file");

for (int i = 0; i < sample\_word\_count; i++) free(dict[i]);

free(dict);

fclose(sPtr);

fclose(dPtr);

return 0;

}

3-train.c

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

#define EPS 0.01

#define CONVERGENCE\_LIMIT 0.000001

#define STEP\_LIMIT 1500

#define SGD\_MEAN\_NUM 5

double multiplication\_of\_wx(int \*singleHotVector, double \*parameters);

double \*partial\_derivative\_of\_mean\_square\_error(int \*singleHotVector, double \*parameters, int y\_true);

void sum\_of\_partial\_derivative\_mse(int \*\*hotVectors, double \*parameters, double \*pdmse\_sum, int \*y\_true);

void gradient\_descent(int \*\*hotVectors, double \*parameters, int \*y\_true);

double total\_mean\_square\_error(int \*\*hotVectors, double \*parameters, int \*y\_true);

void stochastic\_gradient\_descent(int \*\*hotVectors, double \*parameters, int \*y\_true);

double \*stochastic\_partial\_derivative\_mse(int \*\*hotVectors, double \*parameters, int \*y\_true);

int adaptive\_movement\_estimation\_adam(int \*\*hotVectors, double \*parameters, int \*y\_true, FILE \*logPtr, FILE \*pLogPtr);

void write\_log(FILE \*logPtr, int step, double total\_mse);

void write\_parameters\_log(FILE \*pLogPtr, int step, double \*parameters);

static int N = 0; // Number of training samples / hot vectors

static int D = 0; // Number of parameters / unique words / dictionary size

int main()

{

clock\_t start, end;

FILE \*hotVectorsPtr, \*truePtr, \*modelPtr, \*logPtr, \*pLogPtr;

double elapsed\_time;

int \*\*hotVectors, \*y\_true, i, j;

double \*parameters, \*\*sgd\_parameters, parameter\_init;

double total\_mse = 1, total\_mse\_old = 0;

char ch;

int mode, step = 0;

// srand(time(NULL));

printf("Which algorithm? (GD: 0, SGD: 1, ADAM: 2): ");

scanf("%d", &mode);

if (mode < 0 || mode > 2) {

puts("!! Undefined algorithm");

return -1;

}

// Open hot vectors file

hotVectorsPtr = fopen("traininghv.txt", "r");

if (hotVectorsPtr == NULL)

{

puts("!! Unable to open file 'traininghv.txt'");

return -1;

}

// Open true values file

truePtr = fopen("trainingtrues.txt", "r");

if (truePtr == NULL)

{

puts("!! Unable to open file 'trues.txt'");

return -1;

}

// Open model file

modelPtr = fopen("model.txt", "w");

if (modelPtr == NULL)

{

puts("!! Unable to open file 'model.txt'");

return -1;

}

// Open log file

logPtr = fopen("training.log", "w");

if (logPtr == NULL)

{

puts("!! Unable to open file 'training.log'");

return -1;

}

// Open parameters log file

pLogPtr = fopen("parameters.log", "w");

if (logPtr == NULL)

{

puts("!! Unable to open file 'parameters.log'");

return -1;

}

// Get number of hot vectors (N) and number of unique words (D)

do {

ch = fgetc(hotVectorsPtr);

if (ch == '\n') N += 1;

if (ch == '0' || ch == '1') D += 1;

}

while (ch != EOF);

D /= N;

rewind(hotVectorsPtr);

// printf("N: %3d, D: %3d\n", N, D);

// Read hot vectors (NxD)

hotVectors = calloc(N, sizeof(int \*));

for (int i = 0; i < N; i++) hotVectors[i] = calloc(D, sizeof(int));

i = j = 0;

while ((ch = getc(hotVectorsPtr)) != EOF)

{

// printf("%d %d %d\n", i, j, ch);

if (ch == '\n')

{

i++;

j = 0;

}

if (ch == '1' || ch == '0')

{

hotVectors[i][j] = ch - '0';

j++;

}

}

fclose(hotVectorsPtr);

// Read true y values

y\_true = calloc(N, sizeof(int));

i = 0;

while ((ch = getc(truePtr)) != EOF)

{

if (ch == '+')

{

y\_true[i] = +1;

i++;

}

if (ch == '-')

{

y\_true[i] = -1;

i++;

}

}

fclose(truePtr);

// Initialize parameters

parameters = calloc(D, sizeof(double));

printf("Initializing parameters. Randomize? (y/n): ");

getc(stdin);

ch = getc(stdin);

if (ch == 'y' || ch == 'Y') {

for (int i = 0; i < D; i++) parameters[i] = (double) (rand() % 1000) / 10000;

} else {

printf("Enter a real number: ");

scanf("%lf", &parameter\_init);

for (int i = 0; i < D; i++) parameters[i] = parameter\_init;

}

// Record the starting clock

start = clock();

// Gradient Descent

if (mode == 0)

{

// Repeat until converges or step number exceeds the limit

while (fabs(total\_mse - total\_mse\_old) > CONVERGENCE\_LIMIT && step < STEP\_LIMIT)

{

gradient\_descent(hotVectors, parameters, y\_true);

total\_mse\_old = total\_mse;

total\_mse = total\_mean\_square\_error(hotVectors, parameters, y\_true);

write\_log(logPtr, step, total\_mse);

write\_parameters\_log(pLogPtr, step, parameters);

// for (int j = 0; j < D; j++) printf("% lf ", parameters[j]);

// printf("-> %lf\n", fabs(total\_mse - total\_mse\_old));

step++;

}

}

// Stochastic Gradient Descent

if (mode == 1)

{

sgd\_parameters = calloc(SGD\_MEAN\_NUM, sizeof(double \*));

for (int i = 0; i < SGD\_MEAN\_NUM; i++)

sgd\_parameters[i] = calloc(D, sizeof(double));

// Repeat until converges or step number exceeds the limit

while (fabs(total\_mse - total\_mse\_old) > CONVERGENCE\_LIMIT && step < STEP\_LIMIT)

{

stochastic\_gradient\_descent(hotVectors, parameters, y\_true);

// Instead use mean of last K parameters

// Updating SGD parameters array...

for (int i = 0; i < D; i++) sgd\_parameters[step % SGD\_MEAN\_NUM][i] = parameters[i];

total\_mse\_old = total\_mse;

total\_mse = total\_mean\_square\_error(hotVectors, parameters, y\_true);

write\_log(logPtr, step, total\_mse);

write\_parameters\_log(pLogPtr, step, parameters);

// for (int j = 0; j < D; j++) printf("% lf ", parameters[j]);

// printf("-> %lf\n", fabs(total\_mse - total\_mse\_old));

step++;

}

// Mean of last K iteration

for (int i = 0; i < D; i++) {

for (int j = 0; j < SGD\_MEAN\_NUM-1; j++)

parameters[i] += sgd\_parameters[j][i];

parameters[i] /= SGD\_MEAN\_NUM;

}

for (int i = 0; i < SGD\_MEAN\_NUM; i++) free(sgd\_parameters[i]);

free(sgd\_parameters);

}

// ADAM

if (mode == 2)

step = adaptive\_movement\_estimation\_adam(hotVectors, parameters, y\_true, logPtr, pLogPtr);

// Record the ending clock

end = clock();

// Converting clock time to ms

elapsed\_time = ((double)(end - start) / CLOCKS\_PER\_SEC) \* 1000.0;

printf("Took approximately %lf ms to train.\n", elapsed\_time);

printf("For %d iterations, that means approximately %lf ms per iteration.\n", step, elapsed\_time / step);

// Save trained parameters and free memory

puts("Complete, saving...");

if (step == STEP\_LIMIT) puts("WARNING: Model did not converge.");

for (int i = 0; i < D; i++) fprintf(modelPtr, "%lf\n", parameters[i]);

free(parameters);

// Save time to log file

fprintf(logPtr, "%lf %lf", elapsed\_time, step / elapsed\_time);

for (int i = 0; i < N; i++) free(hotVectors[i]);

free(hotVectors);

free(y\_true);

fclose(modelPtr);

fclose(logPtr);

fclose(pLogPtr);

return 0;

}

void gradient\_descent(int \*\*hotVectors, double \*parameters, int \*y\_true)

{

double \*pdmse\_sum = calloc(D, sizeof(double));

sum\_of\_partial\_derivative\_mse(hotVectors, parameters, pdmse\_sum, y\_true);

for (int i = 0; i < D; i++)

parameters[i] -= EPS \* pdmse\_sum[i];

free(pdmse\_sum);

}

double \*partial\_derivative\_of\_mean\_square\_error(int \*singleHotVector, double \*parameters, int single\_y\_true)

{

double wx, \*pdmse = calloc(D, sizeof(double));

wx = multiplication\_of\_wx(singleHotVector, parameters);

// printf("%lf\n", wx);

for (int i = 0; i < D; i++)

pdmse[i] = (single\_y\_true - tanh(wx)) \* singleHotVector[i] \* (-1 / (cosh(wx) \* cosh(wx)));

return pdmse;

}

void sum\_of\_partial\_derivative\_mse(int \*\*hotVectors, double \*parameters, double \*pdmse\_sum, int \*y\_true)

{

double \*pdmse;

for (int i = 0; i < N; i++)

{

pdmse = partial\_derivative\_of\_mean\_square\_error(hotVectors[i], parameters, y\_true[i]);

for (int j = 0; j < D; j++) pdmse\_sum[j] += pdmse[j];

free(pdmse);

}

}

double multiplication\_of\_wx(int \*singleHotVector, double \*parameters)

{

double m = 0;

for (int i = 0; i < D; i++) m += singleHotVector[i] \* parameters[i];

return m;

}

double total\_mean\_square\_error(int \*\*hotVectors, double \*parameters, int \*y\_true)

{

double r, mse\_sum = 0;

for (int i = 0; i < N; i++)

{

r = (y\_true[i] - tanh(multiplication\_of\_wx(hotVectors[i], parameters)));

mse\_sum += r \* r;

}

return (1.0 / N) \* mse\_sum;

}

void stochastic\_gradient\_descent(int \*\*hotVectors, double \*parameters, int \*y\_true)

{

double \*approx\_pdmse;

approx\_pdmse = stochastic\_partial\_derivative\_mse(hotVectors, parameters, y\_true);

for (int i = 0; i < D; i++)

parameters[i] -= EPS \* approx\_pdmse[i];

free(approx\_pdmse);

}

double \*stochastic\_partial\_derivative\_mse(int \*\*hotVectors, double \*parameters, int \*y\_true)

{

int random\_i = rand() % N;

return partial\_derivative\_of\_mean\_square\_error(hotVectors[random\_i], parameters, y\_true[random\_i]);

}

int adaptive\_movement\_estimation\_adam(int \*\*hotVectors, double \*parameters, int \*y\_true, FILE \*logPtr, FILE \*pLogPtr)

{

double total\_mse = 1, total\_mse\_old = 0;

double \*m, \*v, \*approx\_gradient, \*m\_hat, \*v\_hat;

double alpha = EPS, beta1 = 0.9, beta2 = 0.999, e = 0.000001;

int t = 0;

m = calloc(D, sizeof(double));

v = calloc(D, sizeof(double));

m\_hat = calloc(D, sizeof(double));

v\_hat = calloc(D, sizeof(double));

while (fabs(total\_mse - total\_mse\_old) > CONVERGENCE\_LIMIT && t < STEP\_LIMIT)

{

t += 1;

approx\_gradient = stochastic\_partial\_derivative\_mse(hotVectors, parameters, y\_true);

for (int i = 0; i < D; i++)

{

m[i] = (beta1 \* m[i]) + ((1 - beta1) \* approx\_gradient[i]);

v[i] = (beta2 \* v[i]) + ((1 - beta2) \* approx\_gradient[i] \* approx\_gradient[i]);

m\_hat[i] = m[i] / (1 - pow(beta1, t));

v\_hat[i] = v[i] / (1 - pow(beta2, t));

parameters[i] -= alpha \* m\_hat[i] / (sqrt(v\_hat[i]) + e);

}

total\_mse\_old = total\_mse;

total\_mse = total\_mean\_square\_error(hotVectors, parameters, y\_true);

write\_log(logPtr, t, total\_mse);

write\_parameters\_log(pLogPtr, t, parameters);

// for (int j = 0; j < D; j++) printf("% lf ", parameters[j]);

// printf("%4d -> %lf\n", t, total\_mse);

}

return t;

}

void write\_log(FILE \*logPtr, int step, double total\_mse)

{

fprintf(logPtr, "%d %lf\n", step, total\_mse);

}

void write\_parameters\_log(FILE \*pLogPtr, int step, double \*parameters)

{

fprintf(pLogPtr, "%d", step);

for (int i = 0; i < D; i++) fprintf(pLogPtr, " %lf", parameters[i]);

fputc('\n', pLogPtr);

}

4-test.c

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int getD();

int getN();

double multiplication\_of\_wx(int \*singleHotVector, double \*parameters);

int main()

{

int D, N, \*\*hotVectors, \*y\_true, number\_of\_successful = 0;

FILE \*hvPtr, \*modelPtr, \*testTruesPtr;

double \*parameters, y\_hat;

char ch;

D = getD();

N = getN();

// printf("%d, %d\n", D, N);

hvPtr = fopen("testinghv.txt", "r");

modelPtr = fopen("model.txt", "r");

testTruesPtr = fopen("testingtrues.txt", "r");

if (hvPtr == NULL || modelPtr == NULL || testTruesPtr == NULL)

{

puts("!! Unable to open files");

return -1;

}

// Read parameters

parameters = calloc(D, sizeof(double));

for (int i = 0; i < D; i++) {

fscanf(modelPtr, "%lf", &parameters[i]);

// printf("%3d: % lf\n", i, parameters[i]);

}

fclose(modelPtr);

// Read hot vectors

hotVectors = calloc(N, sizeof(int \*));

for (int i = 0; i < N; i++)

{

hotVectors[i] = calloc(D, sizeof(int));

for (int j = 0; j < D; j++) fscanf(hvPtr, "%d", &hotVectors[i][j]);

}

// for (int i = 0; i < N; i++) {

// for (int j = 0; j < D; j++) printf("%d ", hotVectors[i][j]);

// puts("");

// }

fclose(hvPtr);

// Read true "y" values

y\_true = calloc(N, sizeof(int));

for (int i = 0; i < N; i++)

{

ch = fgetc(testTruesPtr);

fgetc(testTruesPtr);

if (ch == '+') y\_true[i] = +1;

else y\_true[i] = -1;

}

fclose(testTruesPtr);

// Test the model

for (int i = 0; i < N; i++)

{

y\_hat = tanh(multiplication\_of\_wx(hotVectors[i], parameters));

// printf("% lf, %d\n", y\_hat, y\_true[i]);

y\_hat = y\_hat > 0 ? 1 : -1;

if (y\_hat == y\_true[i]) number\_of\_successful++;

}

printf("Number of successful predictions: %d/%d\n", number\_of\_successful, N);

printf("Prediction accuracy: %.2lf%%\n", 100 \* ((double)number\_of\_successful / N));

for (int i = 0; i < N; i++) free(hotVectors[i]);

free(hotVectors);

free(y\_true);

return 0;

}

// Get number of unique words (Length of parameters)

int getD()

{

FILE \*dPtr = fopen("dictionary.txt", "r");

int D;

if (dPtr == NULL) {

puts("!! Unable to read 'dictionary.txt'");

return -1;

}

fscanf(dPtr, "%d", &D);

fclose(dPtr);

return D;

}

// Get number of test samples (hot vectors)

int getN()

{

FILE \*hvPtr = fopen("testinghv.txt", "r");

int N = 0;

char ch;

if (hvPtr == NULL) {

puts("!! Unable to read 'testinghv.txt'");

return -1;

}

do {

ch = fgetc(hvPtr);

if (ch == '\n') N++;

}

while(ch != EOF);

fclose(hvPtr);

return N;

}

double multiplication\_of\_wx(int \*singleHotVector, double \*parameters)

{

int D = getD();

double m = 0;

for (int i = 0; i < D; i++) m += singleHotVector[i] \* parameters[i];

return m;

}