Sydney Larson
Kevin Shin

ECE30 Project

P1: Loss

Code:

```
Loss:
ADD X19, XZR, XZR // reset i counter
LDUR X1, [SP, #8] // load in original dataset address
STUR LR, [SP, #32] // store BL Loss + 4 address
LDURS S7, [X9, #0] // reset loss
   LossLoopIntro:
   LDURS S20, [X1, #0] // dataset[i][0] // first half of register
   LDURS S21, [X1, #4] // dataset[i][1] // second half of register
   // check to see if we are using SOCRdata or RawSOCRdata
   FSUBS S22, S20, S21 // x - y
   FCMPS S22, S15 // x - y > -2
   B.LT TSA_Loss // if RawSOCRdata, branch for normalization

   LossLoopOutro:
   FMULS S20, S20, S0 // m*dataset[i][0]
   FADDS S20, S20, S1 // m*dataset[i][0] + c
   FSUBS S20, S21, S20 // dataset[i][1] - m*dataset[i][0] + C
   FMULS S20, S20, S20 // powered
   FADDS S7, S7, S20 // lossSum += powered
   ADDI X19, X19, #1 // i++
   ADDI X1, X1, #8 // dataset[i][0] < X1 holds the array address so if I add by increments of 8,
that increases by 1 register.
   SUBS XZR X0, X19 // arraysize - i
   B.NE LossLoopIntro

         Post-LossLoopIntro:
         FMULS S7, S7, S19   // loss -= inverseN
         LDUR LR, [SP, #32] // load in BL Loss + 4 address
         BR LR  // return with S7
   //loss = 1/n * sum((yi-Y_pred)^2)
   //return the loss in S7
```

P2: Train

Code:

```
train:
    SUBI SP, SP, #40
    STUR LR, [SP, #0] // store LR for main function
    STUR X1, [SP, #8] // store dataset address
    ADD X19, XZR, XZR // reset incrememter for train loop
    LDURS S0, [X9, #0] // m = 0
    LDURS S1, [X9, #0] // c = 0

    TrainLoop:
    LDURS S10 [X9 #0] // D_m = 0 and reset
    LDURS S11 [X9 #0] // D_c = 0 and reset
    LDUR X1, [SP, #8] // load original dataset address
    ADD X10, XZR, XZR // reset incrementer for nested train loop

            NestedTrainLoopIntro:
            LDURS S20, [X1, #0] // dataset[j][0] // x
            LDURS S21, [X1, #4] // dataset[j][1] // y
             // check to see if we are using SOCRdata or RawSOCRdata
            FSUBS S14, S20, S21 // x - y
            FCMPS S14, S15 // x - y > -2
            B.LT TSA_Train // if RawSOCRdata, branch for normalization

            NestedTrainLoopOutro:
            FMULS S13, S20, S0 // M*dataset[j][0]
            FADDS S13, S13, S1 // M*dataset[j][0] + c
            FSUBS S13, S21, S13 // dataset[j][1] - M*dataset[j][0] + c
            FADDS S11, S11, S13 // D_c += dataset[j][1] - M*dataset[j][0] + C
            FMULS S12, S20, S13 // dataset[j][0] * dataset[j][1] - M*dataset[j][0] + c
            FADDS S10, S10, S12 // D_m += dataset[j][0] * dataset[j][1] - M*dataset[j][0] + c
            ADDI X1, X1, #8 // add 1 register ( needs to be saved)
            ADDI X10, X10, #1 // j++
            SUBS XZR, X0, X10 // dataset - j
            B.NE NestedTrainLoopIntro

    FMULS S10, S10, S3 // D_m *= -2/dataset.size()
    FMULS S11, S11, S3 // D_c *= -2/dataset.size()
    FMULS S16, S2, S10 // lr * D_m
    FMULS S17, S2, S11 // lr * D_c
    FSUBS S0, S0, S16 // M - lr * D_m
    FSUBS S1, S1, S17 // C - lr * D_c
    STUR X19, [SP, #16]
    BL Loss
```

```
    LDUR X19, [SP, #16]

    LDUR LR, [SP, #0]
    FCMPS S7, S8 // loss - absolute
    B.LT pre-stop
   LDURS S18, [SP, #24] // load in previousLoss into S18
    FSUBS S18, S7, S18 // loss - prevLoss
    FCMPS S18, S9 // epsilon
    B.LT pre-stop
    STURS S7, [SP, #24] // store prevLoss

    ADDI X19, X19, #1 // i++
    SUBS XZR X2, X19 // check if i < numEpochs
    B.NE TrainLoop      // if yes, loop

    ADDI SP, SP, #40 // de-allocate stack
    BR LR
```

without epsilon/threshold on epoch = 1000
SOCRdata
0.686338663101 = m
7.54992157681044773198664188385009765625E-9 = c
0.528939187526702880859375 = loss

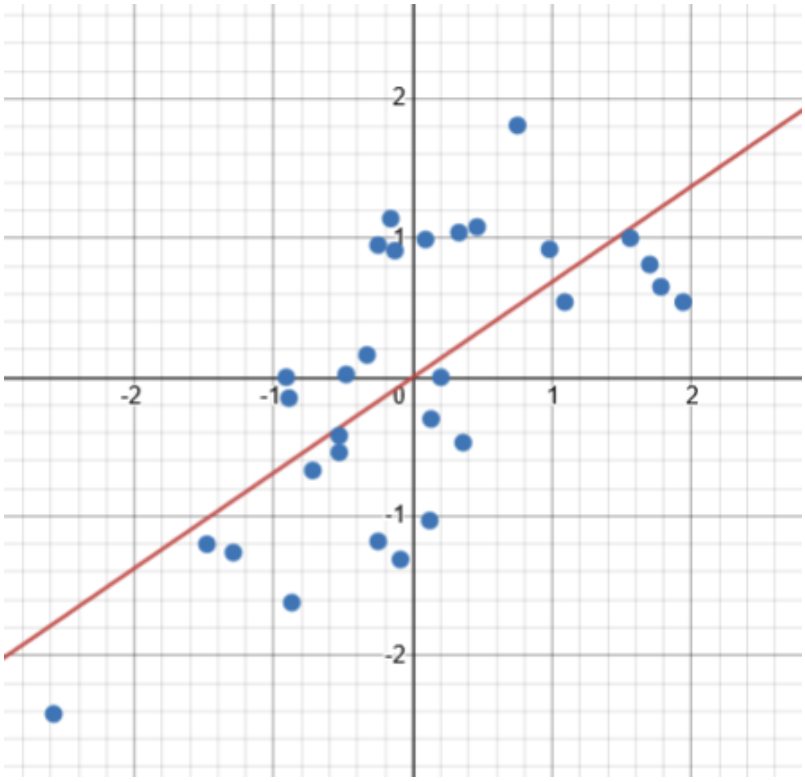
RawSOCRdata
0.68633878231 = m
0.000001509984713266021572053432464599609375 = c
0.52893906831741333007812 5 = loss

after epsilon/threshold on epoch = 2
SOCRdata
0.461439341307 = m
4.55033897283 * 10^-37 = C
0.579518795013 = LOSS

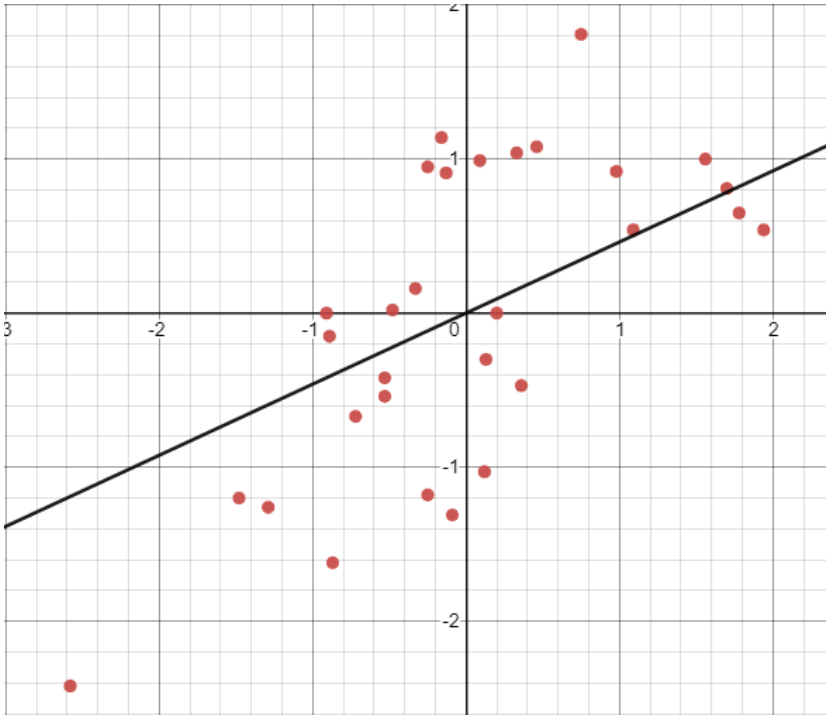
RawSOCRdata on epoch = 2
0.461439341307 = M
3.89417039059 * 10^-7 = C
0.579518795013 = LOSS

P3: Visualization

SOCRdata on epoch = 1000



RawSOCRdata on epoch = 2 with epsilon and lossThreshold



P4: Early stop of training using absolute values

Code, added to the end of train before incrementing i:

```
FCMPS S7, S8 // loss < absolute
B.LT pre-stop
```

P5: Early stop of training using difference in losses

Code, added to the end of train before incrementing i:

```
LDURS S18, [SP, #24] // load in previousLoss
FSUBS S18, S7, S18 // loss - prevLoss
FCMPS S18, S9 // check if loss - prevLoss < epsilon
B.LT pre-stop
STURS S7, [SP, #24] // store current loss
```

P6: Normalization of dataset

We get NaN values when running the algorithm with the RawSOCRData file. Since we are implementing a normalization function, it stands to reason that the values in this file are too big to go through the algorithm without ending up becoming greater than 32 bits.

We added a line to the data file to have a float constant of -2. We used this value so that you can run both the SOCRdata file and the RawSOCRdata file on the same file

condition: -2.0

Code:
TSA_Loss:
BL Normalization
B LossLoopOutro

TSA_Train:
BL Normalization
B NestedTrainLoopOutro

Normalization:
FSUBS S20, S20, S23 // dataset[j][0] - mean
FDIVS S20, S20, S24 // (dataset[j][0] - mean) /SD
FSUBS S21, S21, S25 // dataset[j][1] - mean
FDIVS S21, S21, S26 // (dataset[j][1] - mean) /SD
BR LR

For both train and loss functions

Use for reference

SOCRdata on epoch = 1000 without epsilon or lossThreshold

000000003F2FB3E4	0000000003201B4E8	000000003DCCCCCD	00000000BD888889	0000000000000000	0000000000000000	0000000000000000	000000003F07688F
000000003F19999A	000000003C23D70A	00000000B48DDDDE	0000000080000000	000000003E881421	00000000BE37F5B0	00000000BE919C34	00000000C0000000
00000000B2E2FC97	0000000080000000	0000000000000000	000000003D088889	000000003D04312D	00000000BF98F739	00000000BE919C34	00000000A722AAAB
000000003F800000	0000000026533333	000000003F800000	0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000

SOCRdata on epoch = 2 with epsilon and lossThreshold

000000003E7D0310	0000000031A3D70B	000000003DCCCCCD	00000000BD888889	0000000000000000	0000000000000000	0000000000000000	000000003F38CD7E
000000003F19999A	000000003C23D70A	00000000BF8C8FEC	00000000B2CCCCCE	000000003FBBD8C4	00000000BF7DF170	00000000BE919C34	00000000C0000000
00000000BDE0E647	00000000B123D70B	0000000000000000	000000003D088889	000000003F302590	00000000BF98F739	00000000BE919C34	00000000A722AAAB
000000003F800000	0000000026533333	000000003F800000	0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000

RawSOCRdata on epoch = 10000 without epsilon or lossThreshold

000000003F2FB3E6	0000000035CAAAAE	000000003DCCCCCD	00000000BD888889	0000000000000000	0000000000000000	0000000000000000	000000003F07688D
000000003F19999A	000000003C23D70A	00000000B4577778	0000000080000000	000000003E881455	00000000BE37F5F0	00000000C241CCCC	00000000C0000000
00000000B2AC5F93	0000000080000000	0000000000000000	000000003D088889	000000003D043189	00000000BF98F73A	00000000C241CCCC	00000000A722AAAB
000000003FE3509E	0000000430173E2	0000000041505CD6	0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000

RawSOCRdata on epoch = 2 with epsilon and lossThreshold

000000003E7D0310	0000000033681B4F	000000003DCCCCCD	00000000BD888889	0000000000000000	0000000000000000	0000000000000000	000000003F38CD7E
000000003F19999A	000000003C23D70A	00000000BF8C8FED	00000000B519999A	000000003FBBD8CA	00000000BF7DF170	00000000C241CCCC	00000000C0000000
00000000BDE0E648	00000000B375C290	0000000000000000	000000003D088889	000000003F302590	00000000BF98F73A	00000000C241CCCC	00000000A722AAAB
000000003FE3509E	0000000430173E2	0000000041505CD6	0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000