Math6373. 02/16/2022 . Robert Azencott

HW2 due date = Sunday February 27 midnight

Automatic Classification by MLP : one dataset per group (your choice)

available Datasets at UC Irvine , Kaggle, …pre-validate choice by email to RA

Exclude classical tutorial setups (such as "10 digits recognition" …)

Q0 brief dataset description

# of classes (must be ≥ 6)

Size & practical meaning of each class

# of cases (must be > 6000) ; explain what is a typical case

# of features (must be > 50) ; meaning of each feature

numerical features, # discrete features

Encoding modality for discrete features

HW2 : Q1 : implement 3 layers MLP

Xn = training input vectors   dim Xn = k features

6 classes (or more)   CL1 CL2 … CL6

MLP : Input Xn → hidden layer H → OUTn → softmax→ Pn

dim(OUTn) = 6;   dim(hidden layer H) = h   is   unknown

trueOUTn = one-hot encoding for true class(case n)

Xn → Hn → OUTn → P(n)

P(n)= softmax(OUTn)   = [P1(n), …, P6(n)]

Pj(n) = MLP estimate for probability {case n is in class CLj}

Final MLP decision :

{ case n is in CLj }   if   Pj(n) = max [P1(n), …, P6(n)]

HW2 : Q1 : CrossEntropy loss function

For case n in CL4 for instance ,

MLP+softmax ➔ Pn = [P1(n), …, P6(n)

trueOUTn = [0 0 0 1 0 0] is transformed by softmax into

the true "probability" Qn = [0 0 0 1 0 0] on {1 2 3 4 5 6}

Loss for case n = CrossEntropy(Qn, Pn) = - log[P4(n)]

Loss(n) = CRE(n) = - log[Pj(n)] if case n is in class CLj

average CrossEntropy on training set of size N

trainAVCRE = [CRE(1) + … + CRE(N)] / N

similar definition for testAVCRE

HW2 : Q1 : CrossEntopy loss function

if case n is in class CLj

we have   CRE(n) = - log[Pj(n)]

→ if MLP + SFT yields a bad estimate like Pj(n) = 1/100

then CRE(n) = log (100) = 4.6 = high loss

→ if MLP + SFT yields a very good estimate like Pj(n) = 98%

then CRE(n) = - log (0.98) = 0.02 = very small loss

If average CRE(n) on class CLj = a(j)

→   expect Pj(n) to be of the order of   exp[- a(j)])

HW2 : Q1 : CrossEntropy loss function

Fix optimizer = ADAMS, batch size = N/50 , N= training set size,

Response function = RELU , **# epochs =10**, random initial weights

Fix **h = k or k/2**. Launch a short training of 10 epochs for instance

Monitor AVCRE(m) **epoch per epoch** on training set and test set

Display two monitoring curves versus m= 1... 10 on same graph

Transform the 10 values AVCRE(m) as explained above

in terms of associated estimated probabilities

Display the two transformed monitoring curves. Interpret the results

Fix the weights & thresholds at end of last epoch

Compute then the AVCRE(class CLj) on each class CLj. Interpret results

HW2 : Q2 : test "low value" h_{low} for h = dim(hidden H)

Xn = input vectors   dim Xn = k features   6 classes CL1 … CL6

PCA analysis of set Centered & Rescaled input vectors Xn

Display PEV = percent. explained variance vs #principal components

Compute $\mathbf{h_{low}}$ such that PEV(h_{low}) = 90%

Launch automatic learning for h= h_{low} using

batchsize = 100 , #epochs = 50, loss function = cross-entropy

Plot curves AVCRE(m) on testset & trainset for epochs m= 1… 50

then select the best "m" by following criterion ;

testAVCRE should be as small as possible but inferior to trainAVCRE

HW2 : Q2 : test "low value" h_{low} for h = dim(hidden H)

To **select a stopping epoch mSTOP** : first evaluate visually

StabTrain = epoch of stabilization for trainAVCRE(m)

improvement of trainAVCRE(m) should be small for m > StabTrain

MinTest = epoch when testAVCRE(m) reaches a global minimum

and then starts roughly increasing for most m > MinTest

SafeZone = all epochs m such that testAVCRE(m) > trainAVCRE(m)

this is the zone of **no overfit**

SafeMinTest = epoch m* in SafeZone where testAVCRE(m) reaches

its minimum on SafeZone & starts increasing on SafeZone

m* is often a good mSTOP, but compare it to StabTrain

HW2 : Q2 : test "low value" h_{low} for h = dim(hidden H)

Fix m= mSTOP after preceding analysis

Fix the corresponding MLP classifier denoted $\mathbf{MLP_{low}}$

For each case #n, this classifier outputs probabilities P1(n) … P6(n)

Apply the decision rule

Predicted class (case n) = class CLj when Pj(n) = max[P1(n) … P6(n)]

Run all cases through this MLP_{low} classifier

Compute the 6x6 confusion matrix (in % of accurate classification)

interpret the confusion matrix on test set and trainset

HW2 : Q3 : evaluate h_{high} for h= dim(hidden layer)

Separately for each class CLj , of sizeN(j)

Launch PCA analysis for all the N(j) inputs Xn belonging to class CLj

Display curve PEVj = % explained variance vs # principal components

Compute hj such that PEVj(hj) = 90%

Define $\mathbf{h_{high}}$ **= h1 + h2 + … + h6**

Implement the approach of Q2 for this new value of h

Then select the best of the two sizes $\mathbf{h_{high}}$ **,** $\mathbf{h_{low}}$

Q4 : start DEEP LEARNING by AutoEncoder construction

Xn = input vector   dim Xn = k   **h= dim H=** **h_{high}**

MLP : INP → H → OUT → softmax→ P(n) = [P1(n), … , P6(n)]

Goal: Improve this MLP= MLP_{high}

Xn → vector Zn (read on Hidden layer H); dim Zn = h

Zn is computed using the weights and thresholds of MLP_{high}

Construct an auto encoder L1 → L2 → L3 to encode /decode Zn

Zn on L1 => Kn on L2 = encoding Zn   => Z'n on L3   = decoding Kn

Z'n should be close to Zn;   dim (hidden layer L2) = h2

First step: Compute h2 by PCA analysis of the new inputs Zn

h2= # principal components to get 95% of explained variance

Q4 : AutoEncoder construction

Construct an auto encoder L1 → L2 → L3 to encode /decode Zn

Xn → vector Zn (read on Hidden layer H); dim Zn = h

Zn on L1 => Kn on L2 = encoding Zn => Z'n on L3 = decoding Kn

dim (hidden layer L2) = h2 ; dimL1 = dimL3 = **h = h_{high}**

Z'n should be close to Zn → **Loss function = MSE(Zn,Z'n)**

Train auto encoder using batches of size 100

Monitor training by display of the two curves

TrainMSE(m) and TestMSE(m) versus m (epoch per epoch)

Plot also the rescaled TrainRMSE(m) and TestRMSE(m)

Select the best m and fix the corresponding weights/thresholds

Q5 : MLP with 3 hidden layers

Keep only weights /thresholds for first half L1 → L2 of autoencoder

Zn computed from Xn using the weights and thresholds of MLP_{high}

Zn has become a typical input for L1

The weights /thresholds of autocoder L1→ L2 transform Zn into

a new input vector Kn on L2

Zn on layer L1 → Kn on layer L2 ; L2 has size h2 = dim(Kn)

Train new short classifier with only 1 hidden layer H3 and inputs Kn

Kn on L2 → H3 → OUT →softmax → new probability vector P(n)

select dimH3 by PCA analysis of the set of all Kn

Use cross-entropy as loss function

Q6 : MLP_{long} with three hidden layers

Using Q3, Q4 and Q5 we now have a long MLP = MLP_{long}

inputXn → H = L1 → L2 → H3 → OUT → softmax → 6 probabilities

Xn => Zn => Kn => Un => On => softmax => P(n)

weights /thresholds for Xn =>Zn were obtained in Q3

weights /thresholds for Zn =>Kn were obtained in Q4

weights/thresholds for Kn => Un => On => P(n) were obtained in Q5

Compare performances between all constructed MLP classifiers

MLP_{low} MLP_{high} MLP_{long}