Performance P improves with experience E

Example = observation = row = vector = instance

Features = Columns = Variables = Fields

Classification: predict which of c classes that an input belong to

Regression: computer program predicts a numeric value for a given input

NLP: sequence of symbols in some language -> convert to a different language

Performance: accuracy = correct/total ; Error Rate = incorrect/total

Precision = TP / TP + FN measure of sensitivity

Recall = TP / TP + FP measure of specificity

Activation function for non-liner

The test and train sets should be disjoint. Why?

Mean squared error

Minimize/reduce the MSE improve the weights

Perceptron is Linear regression model, labels in linear regression is continuous, **activate function** , linear output into non-linear output,

Underfitting occurs when the model is not able obtain a sufficiently low error value on the training set.

Overfitting occurs when the gap between the training error and test error is too large.

Loss function to measure of fitness, minimize this difference; Squared error = 0.5 (y^ - y)^ 2

Sigmoid, special case of softwax

Perceptron, boolean tasks, update the weights whenver the perceptron output is wrong, proved convergence for linerly sepearable classes; No hidden layers

XOR operation

Always remove the labels from the data

1)  Weights - affect the inputs x

2)  Activation Function - converts weighted sums (wixi) into output

3)  Loss/Learning Function– to optimize the w’s to improve prediction

4)  Gradient Descent – method to update weights via loss function

5)  Back Propagation – using deeper weights to update earlier weights

BP: training means updating weights, BP is the method used to determine the updates for the weights

The goal of BP is to optimize the weights

dL/dw11 = x11 \* (h1)(1 – h1) \* w1 \* (y^)(1 – y^) \* (y^- y)

dL/dw21 = x12 \* (h1)(1 – h1) \* w1 \* (y^)(1 – y^) \* (y^- y)

dL/db1 = 1 \* (h1)(1 – h1) \* w1 \* (y^)(1 – y^) \* (y^- y)

dL/dw12 = x11 \* (h2)(1 – h2) \* w2 \* (y^)(1 – y^) \* (y^- y)

dL/dw22= x12 \* (h2)(1 – h2) \* w2 \* (y^)(1 – y^) \* (y^- y)

dL/db2 = 1 \* (h2)(1 – h2) \* w2 \* (y^)(1 – y^) \* (y^- y)

dL/dw1 = h1 \* (y^)(1- y^) \* (y^ - y)

dL/dw2 = h2 \* (y^)(1- y^) \* (y^ - y)

dL/dc = 1 \* (y^)(1 – y^) \* (y – y^)

Call this y^\_error(y^- y)

Call this y^D\_Error (y^)(1 – y^) (y^- y)

When we have more than one input, we need to sum the derivatives.

print("FeedForward:\n")

self.z=(np.dot(X,self.W1\_x))+self.bs. #XisnbycW1iscbyh-->nbyh. Z1=X@W1\_x+bs#shouldbeshapenbyh

self.h=self.Sigmoid(self.z) #activationfunctionshape:nbyh A\_Z1=np.maximum(0,Z1)##ReLU,shapenbyh

self.z2=(np.dot(self.h,self.W2\_h))+self.c #nbyh@hbyo-->nbyo Z2=(A\_Z1@W2\_h)+c##nbyh@hby1

output=self.Sigmoid(self.z2)

print("y^is:\n",output)

print("\n\nThepredictedlabelsare:")

print(MyPerceptron.predict(TwoD\_Test))

##Printtheactualknowntestlabels

print("\nThetruetestlabelsare:")

print(TestLabel)

fromsklearn.preprocessingimportLabelEncoder

Label\_Encoder=LabelEncoder()##instantiateyourencoder

HeartHealthDF["Label"]=Label\_Encoder.fit\_transform(HeartHealthDF["Label"])

fromsklearn.preprocessingimportStandardScaler

MyScaler=StandardScaler()#instantiatethescaler

HeartHealthDF[["Cholesterol","Weight","Height"]]=MyScaler.fit\_transform(HeartHealthDF[["Cholesterol","Weight","Height"]])

fromsklearn.model\_selectionimporttrain\_test\_split

TRAIN\_data,TEST\_data=train\_test\_split(HeartHealthDF,test\_size=0.25)

##Getthelabelfromthetrainingdata

Train\_Label=TRAIN\_data["Label"]##Savethelabel

#print(Train\_Label)

##Dropthelabelfromthetrainingsetnowthatyousavedit

TRAIN\_data=TRAIN\_data.drop("Label",axis=1)

##axis=1meansdropthecolumn.axis=0dropsarow

#print(TRAIN\_data)

##OK!Let'sdothisforthetestingdatanow

Test\_Label=TEST\_data["Label"]##Savethelabel

print(Test\_Label)

##Dropthelabelfromthetrainingsetnowthatyousavedit

TEST\_data=TEST\_data.drop("Label",axis=1)

##axis=1meansdropthecolumn.axis=0dropsarow

print(TEST\_data)

fromsklearn.neural\_networkimportMLPClassifier

##InstantiateyourNNwiththeparametervaluesyouwant

MyNN=MLPClassifier(hidden\_layer\_sizes=(50,80,50),

max\_iter=100,activation='relu',solver='adam',random\_state=1)

##hidden\_layer\_sizesspecifiesthenumberoflayers(3there

##becausewehavethreevaluesinourtuple.)

##Wearealsospecifyingthenumberofnodesinthehiddenlayer.

##########TraintheNNModel

MyNN.fit(TRAIN\_data,Train\_Label)

##Noticethatwegivethemodelthedataand

##thelabelforthedataseperately!

##Nowwecanuseourtestdata(WITHOUTthelabel)

##toseeifourmodelpredictsthelabel.

##So-themodelwillpredictwhatitthinksthelabel

##shouldbe.

##Wehavethelabels,sowecanchecktoseewhichlabels

##themodelpredictedrightandwrong.

##Wewilluseaconfusionmatrixforthis

Test\_Prediction=MyNN.predict(TEST\_data)

#print(Test\_Prediction)

#print(Test\_Label)

#ConfusionMatrix

fromsklearn.metricsimportconfusion\_matrix

fromsklearn.metricsimportaccuracy\_score,precision\_score,recall\_score,f1\_score

#Comparingthepredictionsagainsttheactualobservations

MyConfusionMatrix=confusion\_matrix(Test\_Prediction,Test\_Label)

print("\n\n",MyConfusionMatrix)

#Printingtheaccuracy

print("TheaccuracyofthesklearnMLPClassifieris:")

print(accuracy\_score(Test\_Prediction,Test\_Label))

self.w\_=np.zeros(1+X.shape[1])##

importnumpyasnp

importmatplotlib.pyplotasplt

classneural\_nets(object):

@staticmethod

defSigmoid(value,deriva=False):

ifderiva:

returnvalue\*(1-value)

return1/(1+np.exp(-value))

def\_\_init\_\_(self,X,y,bs,c,W1\_x,W2\_h):

self.LR=0.01

self.X=X

self.y=y

self.bs=bs

self.c=c

self.W1\_x=W1\_x

self.W2\_h=W2\_h

self.z=None

self.h=None

self.z2=None

self.GA=False

deftrain(self,X,y):

output=self.FeedForward(X)

self.BackProp(X,y,output)

returnoutput

defFeedForward(self,X):

print("FeedForward:\n")

self.z=(np.dot(X,self.W1\_x))+self.bs#XisnbycW1iscbyh-->nbyh

print("Z1is:\n",self.z)

self.h=self.Sigmoid(self.z)#activationfunctionshape:nbyh

print("His:\n",self.h)

self.z2=(np.dot(self.h,self.W2\_h))+self.c#nbyh@hbyo-->nbyo

print("Z2is:\n",self.z2)

output=self.Sigmoid(self.z2)

print("y^is:\n",output)

returnoutput

defBackProp(self,X,y,output):

print("\nBackProp:\n")

#Y^-Y

self.output\_error=output-y

#print("Y^-Y\n",self.output\_error)

#print("SIGY^\n",self.Sigmoid(output,deriva=True))

#(Y^-Y)(Y^)(1-Y^)

self.output\_delta=self.output\_error\*self.Sigmoid(output,deriva=True)

#print("D\_Error(Y^)(1-Y^)(Y^-Y)is:\n",self.output\_delta)

#(Y^-Y)(Y^)(1-Y^)(W2)

self.D\_Error\_W2=self.output\_delta.dot(self.W2\_h.T)#D\_ErrortimesW2

#print("W2is\n",self.W2)

#print("D\_ErrortimesW2\n",self.D\_Error\_W2)

#(H)(1-H)(Y^-Y)(Y^)(1-Y^)(W2)

self.H\_D\_Error\_W2=self.D\_Error\_W2\*self.Sigmoid(self.h,deriva=True)

#Notethat\*willmultiplyrespectivevaluestogetherineachmatrix

#print("DerivativesigHis:\n",self.Sigmoid(self.h,deriva=True))

#print("self.H\_D\_Error\_W2is\n",self.H\_D\_Error\_W2)

#------UPDATEweightsandbiases------------------

#XT(H)(1-H)(Y^-Y)(Y^)(1-Y^)(W2)

self.X\_H\_D\_Error\_W2=X.T.dot(self.H\_D\_Error\_W2)#thisisdW1

#(H)T(Y^-Y)(Y^)(1-Y^)

self.h\_output\_delta=self.h.T.dot(self.output\_delta)#thisisdW2

#print("thegradient:\n",self.X\_H\_D\_Error\_W2)

#print("thegradientaverage:\n",self.X\_H\_D\_Error\_W2/self.n)

ifself.GA=="True":

print("Usingaveragegradient........\n")

#self.W1\_x=self.W1\_x-self.LR\*(self.X\_H\_D\_Error\_W2/3)

#self.W2\_h=self.W2\_h-self.LR\*(self.h\_output\_delta/3)##averagethegradients

##print("NewW1:\n",self.W1)

else:

#print("Usingsumgradient........\n")

self.W1\_x=self.W1\_x-self.LR\*self.X\_H\_D\_Error\_W2#cbyhfirstset(input->hidden)weights

self.W2\_h=self.W2\_h-self.LR\*self.h\_output\_delta#adjustingsecondset(hidden->output)weights

print("NewW1:\n",self.W1\_x)

print("NewW2:\n",self.W2\_h)

#print("Thebbiasesbeforetheupdateare:\n",self.bs)

self.bs=self.bs-self.LR\*self.H\_D\_Error\_W2

#print("TheH\_D\_Error\_W2is...\n",self.H\_D\_Error\_W2)

print("Updatedbsare:\n",self.bs)

self.c=self.c-self.LR\*self.output\_delta

print("Updatedc'sare:\n",self.c)

#print("TheW1is:\n",self.W1\_x)

print("TheW1gradientis:\n",self.X\_H\_D\_Error\_W2)

#print("TheW1gradientaverageis:\n",self.X\_H\_D\_Error\_W2/self.n)

print("TheW2gradientis:\n",self.h\_output\_delta)

#print("TheW2gradientaverageis:\n",self.h\_output\_delta/self.n)

print("Thebiasesbgradientis:\n",self.H\_D\_Error\_W2)

print("Thebiascgradientis:\n",self.output\_delta)

#Setup

X=np.array([[1,2,3],[-1,-2,-3],[3,4,-2]])

y=np.array([[1],[0],[0]])

W1\_x=np.array([[1,-1],[2,-2],[3,-3]])

bs=np.array([[1,2]])

W2\_h=np.array([[4],[5]])

c=3

#Initial

NN=neural\_nets(X,y,bs,c,W1\_x,W2\_h)

TotalLoss=[]

AverageLoss=[]

Epochs=1000

foriinrange(Epochs):

print("Iteration",i+1)

output=NN.train(X,y)

print("TotalLoss:",.5\*(np.sum(np.square(output-y))))

TotalLoss.append(.5\*(np.sum(np.square(output-y))))

print("AverageLoss:",.5\*(np.mean(np.square((output-y)))))

AverageLoss.append(.5\*(np.mean(np.square((output-y)))))

#Plot

fig1=plt.figure()

ax=plt.axes()

x=np.linspace(0,1000,Epochs)

ax.plot(x,TotalLoss)

plt.show()

fig2=plt.figure()

ax=plt.axes()

x=np.linspace(0,1000,Epochs)

ax.plot(x,AverageLoss)

plt.show()

Optional 1: one vector at a time, update weight and biase, and loss, and did again; Time and space

Optional 2: Sum it all, do it in batches, control size of batch

Boostrap

Df = pd.read\_csv

X = np.array(df[:, 1,2,3])

Y = np.arrary(df[:, 0]).T

# for b and c

Np.mean(, axis = 0)

The shape of b and c is not wrong, just sum

Sometimes works, sometime don’t, due to global vs local minimum,

Epochs, take all dataset to …

*import*nltk

nltk.download('wordnet')

*from*nltk.corpus*import*wordnet*as*wn

print("Numberofwords(lemmas)inEnglishWordNet:",len(list(wn.words())))

Global vs local minimum

A local minimum of a function is a point where the function value is smaller than at nearby points, but possibly greater than at a distant point. A global minimum is a point where the function value is smaller than at all other feasible points.

Text

Description automatically generated

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  | Loss function: categorical cross entropy  Y is hot-encoded  Activation function is softmax  The normalized exponential function converts a vector of k real numbers into a probability distribution of k probabilities that all sum to 1, which means give a probability of each output      Jacobian is often interchangeably used to refer to both the Jacobian matrix or its determinant.    The Jacobian matrix collects all first-order partial derivatives of a multivariate function that can be used for backpropagation.    Categorical cross-entropy  1) The categorical cross-entropy loss is exclusively used in multi-class classification tasks.    2) Each sample (input vector) belongs to (represents) exactly ONE class (or category)    3) The label (Y) is one-hot encoded and so is a vector of size C (C is for category)/  Text  Description automatically generated with low confidence    temp = y ## Actual labels  one\_hot\_labels = np.zeros((n, NumberOfLabels))  for i in range(n):  one\_hot\_labels[i, temp[i]-1] = 1  y = one\_hot\_labels  print(y)    Softmax:    1) Also called softargmax.    2) Normalized exponential function.    3) Converts a vector of k real numbers into a probability distribution of k probabilities that all sum to 1.    4) Can also be used in multinomial logistic regression.    5) Often used as the last activation function of a neural network    6) It is a “smooth” approximation to the argmax    1) Cross-entropy (from the field of information theory) calculates the difference between two probability distributions.    2) Specifically, and different from KL divergence, cross-entropy calculates the total entropy between the distributions.  H(x) = -SUM (over all x in X) P(x) \* log(P(x)  the cross-entropy between P and Q is:  H(P, Q) = – sum (x in X) P(x) \* log(Q(x))    from math import log2  import numpy as np    def cross\_entropy(p, q):  return -sum([p[i]\*log2(q[i]) for i in range(len(p))])    Y = [1, 0, 0]  Yhat = [0.80, 0.15, 0.05]    CrossEntropy\_Y\_Yhat = cross\_entropy(Y, Yhat)  print(CrossEntropy\_Y\_Yhat)    Yhat = [1, 0.0000000000001, 0.000000000001]    Information gain is a measure quantifies how much a given “tree node split” unmixes the labels at a node.    Mathematically it is measure of the difference between impurity values before splitting the data at a node and the weighted average of the impurity after the split.    STEPS:    1) One-Hot Encode the Label (Y) of the Dataset    2) Use Softmax to update all predicted outcomes (Y^) so that each is a probability distribution that sums to 1.    3) Use Cross Entropy to measure the “error/loss” between Y and Y^. If the error/loss is “0”, then the prediction is perfect.    4) When using a batch of inputs at a time – we will need to AVERAGE…    Cross-Entropy = 0.00: Perfect probabilities.    Cross-Entropy < 0.02: Great probabilities.    Cross-Entropy < 0.05: On the right track.    Cross-Entropy < 0.20: Fine.    Cross-Entropy > 0.30: Not great.    Cross-Entropy > 1.00: Terrible.    Cross-Entropy > 2.00 Something is broken.    **Part 1:**  Suppose you have a NN such that X is n by 4, H (the one hidden layer) has 3 hidden units, and there is one output, Y^. The hidden layer activation function is the Sigmoid. The output activation function is the Sigmoid. The Loss function is SUM 1/2 (yhat - y)^2.  Write out the complete derivative set for **dL/dw11** for n = 3 X rows. For example,  [ [x11, x12, x13, x14]  [x21, x22, x23, x24],  [x31, x32, x33, x34] ]  Show the chain of derivatives, the derivative values, and other details that illustrate your understanding of the concepts. We are not using “numbers” here – so there will be no “numeric” values.  **Answer:**  We have X = [ [x11, x12, x13, x14]. Y = [[y1],[y2],[y3]] W1 = [[w11,w12,w13],  [x21, x22, x23, x24], [w21,w22,w23],  [x31, x32, x33, x34] ] [w31,w32,w33],  [w41,w42,w43]]  W2 = [w1, w2, w3]      Z1 = X @ W1 + b  [**z11** = x11**w11** + x12w21 + x13w31 + x14w41 + b1 z12 z13  **z21**= x21**w11** + x22w21 + x23w31 + x24w41 + b1 z22 z23  **z31** = x31**w11** + x32w21+x33w31+x34w41+b1 z32 z33] shape n by h    H1 = [**h11** = sig(**z11**) h12 = Sig(z12) h13 = Sig(z13)  **h21** = sig(**z21**) h22 = Sig(z22) h23 = Sig(z23)  **h31** = sig(**z31**) h32 = Sig(z32) h33 = Sig(z33)    Z2 = [ z(2)1= **h11**w1 + h12w2 + h13w3 + c Y^ = [ y^1 = sig(z(2)1)  z(2)2 = **h21**w1 + h22w2 + h23w3 + c y^2 = sig(z(2)2)  z(2)3 = **h31**w1 + h32w2 + h33w3 + c] y^3 = sig(z(2)3)]      dL/dw11 = dz11/dw11 \* dh11/dz11 \* dz(2)1/dh11 \* dy^1/dz(2)1\* dL/dy^1 +  dL/dw11 = dz21/dw11 \* dh21/dz21 \* dz(2)2/dh21 \* dy^2/dz(2)2\* dL/dy^2 +  dL/dw11 = dz31/dw11 \* dh31/dz31 \* dz(2)3/dh31 \* dy^3/dz(2)3\* dL/dy^3    x11 \* (h11)(1 – h11) \* w1 \* (y^1)(1 – y^1) \* (y^1– y1) +  x21 \* (h21)(1 – h21) \* w1 \* (y^2)(1 – y^2) \* (y^2– y2) +  x31 \* (h31)(1 – h31) \* w1 \* (y^3)(1 – y^3) \* (y^3– y3)    **Part 2**:  Suppose you have the following NN details: X is n by 4, H (the number of hidden units in the one hidden layer) is 2, and each output, y^ has  3 values  (y^1, y^2, and y^3). The activation function for H is the Sigmoid and the activation function for the output layer is Softmax. The labels, Y, are one-hot-encoded. The Loss function is Categorical Crossentropy.  **Answer these questions:**   1. Suppose Y is    [[3]   [1]   [2]   [3]   [3] ]   What would Y look like once it was one-hot-encoded?  **Answer**:  [[0,0,1]  [1,0,0]  [0,1,0]  [0,0,1]  [0,0,1]]     1. Suppose the NN produces this output for one of the inputs:   [  -.2,   1.4,   .89]  What would this output be once Softmax is applied?  Why is this softmax result a probability distribution?  **Answer**:  The softmax formula: e^x / Se^x\_i  [0.112, 0.555, 0.333]  The normalized exponential function converts a vector of k real numbers into a probability distribution of k probabilities that all sum to 1, which means give a probability of each output     1. Suppose you read only one row of X ([ [x11, x12, x13, x14] ]) into this network. Show what the following would all be:   X = [ [x11, x12, x13, x14] ]  Y =  [0, 1, 0] (Hint: one hot encoded –you can pretend that the category is a 2. )  W1 = [[w11, w12]  [w21, w22]  [w31, w32]  [w41, w42]]  Z1 = [z11 = x11w11 + x12w21 + x13w31 + x14w41 + b1 z12 = x11w12 + x12w22 + x13w32 + x14w42 + b2]  H = [h1 = sig(z11) h2 = sig(z12)]  Z2 = [z2 = h1w1 + h2w2 + c]  Y^ = [softmax(z2)] (such that y^ is the final output after softmax activation)     1. Write out the chain of derivatives for dL/dw11 as well as what the derivatives would resolve into.    Hint: When using one-hot-encoding, softmax, and categorical cross entropy, as you are using here, the last two derivatives (**dy^/dz2 and dL/dy^ )** can be expressed together as dL/dz and can be resolved as one value (as we proved in class).      dL/dw11 = dz11/dw11 \* dh11/dz11 \* dz(2)1/dh11 \* dy^1/dz(2)1\* dL/dy^1  = dz11/dw11 \* dh11/dz11 \* dz(2)1/dh11 \* dL/dz  = dz11/dw11 \* dh11/dz11 \* dz(2)1/dh11 \* (y^1 – y1)  = x11 \* (h11)(1-h11) \* w1 \* (y^1 – y1) | |  |  |
|  |  |  | RNNs  Recurrence relations  Transfer Learning  Attention  Memory    Sequential data: audio, text, stock  Create and maintain a relationship(dependency) between past and current data/info flow  Propagate this information forward through time  This is done via a recurrence relation    Y\_t = f(x\_t, h\_t-1)    RNNs have a state h\_t, that is updated at each time step as a sequence is processed  The ht is parameterized using a weight matrix W which will be learned, the set of weights is the SAME across all time step through the sequence    Graphical user interface, text  Description automatically generated    Ht = tanh(W(h->h)h\_t-1 + W(x->h)X\_t)  Diagram  Description automatically generated  RNNs processing sequential data of variable length  RNNs share parameters (same weight)  Sequence of vectors, one hot encode a vocabulary  Unfolded recurrent structure: regardless of the sequence length, the learned model always has the same input size. And it is possible to use the same transition function with the same parameters at every time step    The attention to focus on more important aspects  Transfer learning can bringing past knowledge into the system  Runtime is O(T) because can not parallel due to sequential  Memory cost is O(T) all states must be stored until they are reused during the backward pass      RNNs Back Prop Through Time BPTT: apply the back propagation algorithm to the unrolled graph    dEt/dU is the same as dEt/dW    Vanishing gradients: multiply many small number together    Activation functions: relu, tanh, sigmoid  Parameter Initialization: initialize weights to identity matrix, and biases to zero    LSTM long short term memory: contain computational blocks that control information flow, separate cell state  Information is added or removed through structures called gates.  Forget, store, update, output  Backpropagation through time with uninterrupted gradient flow    Tensorflow/Keras: RNN, GRU, LSTM    Sentiment  Word mapping, max/min length, padding sequence  Batch size, epochs |
|  |  |  |  |  |
|  | #UsetheMNISTdataset,andTF/KerastobuildaNNwithtwohiddenlayers.UseSoftmax/CatCrossentropyforthe  #outputactivationandLossfunction.Forthemiddlelayers,useReLUforoneandSigmoidfortheother.Thegoal  #forthispartistosimplyassurethatthefunctionsandmethodsworkforyou.Youcanlocatethesolutiononline  #ontheTensorflowsite.      importtensorflowastf    mnist=tf.keras.datasets.mnist  Data\_=tf.keras.datasets.fashion\_mnist    (x\_train,y\_train),(x\_test,y\_test)=Data\_.load\_data()  x\_train,x\_test=x\_train/255.0,x\_test/255.0    model=tf.keras.models.Sequential([  tf.keras.layers.Flatten(input\_shape=(28,28)),  tf.keras.layers.Dense(128,activation='relu'),  tf.keras.layers.Dense(64,activation='sigmoid'),  tf.keras.layers.Dropout(0.2),  tf.keras.layers.Dense(10,activation='softmax')  ])    model.compile(optimizer='adam',  loss='sparse\_categorical\_crossentropy',  metrics=['accuracy'])    model.fit(x\_train,y\_train,epochs=5)  model.evaluate(x\_test,y\_test)  #  #Epoch1/5  #1875/1875[==============================]-4s2ms/step-loss:0.5756-accuracy:0.8007  #Epoch2/5  #1875/1875[==============================]-3s2ms/step-loss:0.3982-accuracy:0.8566  #Epoch3/5  #1875/1875[==============================]-3s2ms/step-loss:0.3600-accuracy:0.8689  #Epoch4/5  #1875/1875[==============================]-3s2ms/step-loss:0.3339-accuracy:0.8788  #Epoch5/5  #1875/1875[==============================]-3s2ms/step-loss:0.3131-accuracy:0.8844  #313/313[==============================]-1s1ms/step-loss:0.3450-accuracy:0.8756 | |  |  |
|  |  |  |  |  |

Backpropagation tells us which parameters to change in a neural network

The goal in a neural network or any optimization problem is to lower and minimize whatever loss function we define.

Mean squared error loss MSE

Tune the weight and bias on one neuron

Instantaneous rate of change (derivative)

Gradient descent

Backprop on a non-linear problem (multiple neurons with activation function)

Forward propagation resulting in an output and loss

Backward propagation using the chain rule to compute the gradient

Descend the loss by performing gradient descent

Through a simple backpropagation algorithm learning becomes possible

Neural network

It is based very loosely on how we think the human brain works

A collection of neurons are created and connected together, allow them to send messages to each other

Next, the network is asked to solve a problem, which it attempts to do over and over, each time strengthening the connections that lead to success and diminishing those that lead to failure.

#Importlibrary

importnumpyasnp

importpandasaspd

importmatplotlib.pyplotasplt

#(Satisfied)Overview:Youwillbewritingcodeforaonehidden-layer,multiple-outputNNusingPython.

#YouwillnotuseanyNNpackages

#(Satisfied)Yourcodeacceptslabeleddataofsizenbyc.Thefirstcolumnwillbethelabel.

#Thelabelwillbenumericandcanbebetween2throughk.

#Forexample,ifkis3,therearethreecategories,1,2,and3.

#Forsimplicity,youcanassumethatk<5.

#(Satisfied)Itisrequiredthatyourcodespecificallyreadsadatafilecalled“TheData.csv”

#(Satisfied)ThecodewilluseEITHERtheSigmoidorReLUforthehiddenlayerandtheUSERwilldecide.

#(Satisfied)ThecodewillusetheSoftmaxactivationfortheoutputlayer,

#alongwiththeone-hot-encodingfortheY,andthecategoricalcrossentropyfortheLoss.

#(Satisfied)TheUSERwillchoosehowmanyepochstorun.

#(Satisfied)Thecodeshouldshow(asoutput)thetotalloss,thelossasitgoes,

#andavisualizationthatshowshowthelossisdecreasing.

#Thecodeshouldshowthefinalprediction(aftersoftmaxandthenargmaxareapplied).

#Forexample,ifYis[13123]thenthepredictedresultsmightbe

#(aftersoftmaxandthenargmax)[13223].

#Createaconfusionmatrixtocompare.

#(Satisfied)Youmaychoosetousesumsormeansorablendinyourcode–that’suptoyou.

#(PartialSatisfied)EnableyourcodetotraininbatchesandhavetheUSERchoosethenumberofbatches.

#Iwilllimitthischoicetobetween1batch(theentiredatasetatonce)to3batches.

#Thisway,thesizeofthedatasetwillnotbeanissue.Rememberthattraininginbatcheswillusesums(ormeans).

#(Satisfied)ExtraCredit(5points)Yourcodeshouldsplitthedatasetintotestingandtrainingdatasets.

#ItwilltraintheNNandthentesttheaccuracyusingthetestingset.

#Createaconfusionmatrixandnotethefinalaccuracy.

#Askforuserinput

epoch=eval(input("Howmanyepochstorun(Suggestion:startwith100/500/1000):\n"))

activation\_func=eval(input("Chooseyouractivationfunctionforhiddenlayer\n"

"Enter0forSigmoidorEnter1forReLU:\n"))

batches=eval(input("Howmanybatchestorun(Enter1,2,or3):\n"))

#readdataset

df=pd.read\_csv("TheData.csv")

#Setupvariables

InputColumns=len(df.columns)-1

NumberOfLabels=len(df.iloc[:,0].unique())

LR=.01

LRB=.01

defsplit\_dataset(sets,labels):

"""

Splitthedatasetrandomlyinto80%trainingand20%developmentset

"""

indices=list(range(len(sets)))

num\_training\_indices=int(0.8\*len(sets))

np.random.shuffle(indices)

train\_indices=indices[:num\_training\_indices]

dev\_indices=indices[num\_training\_indices:]

#splittheactualdata

train\_set,train\_labels=sets.iloc[train\_indices],labels.iloc[train\_indices]

dev\_set,dev\_labels=sets.iloc[dev\_indices],labels.iloc[dev\_indices]

returntrain\_set,train\_labels,dev\_set,dev\_labels

#Splitthedataset

train\_set,train\_labels,dev\_set,dev\_labels=split\_dataset(df.iloc[:,1:],df.iloc[:,0])

dev\_labels=np.array(dev\_labels).T

y\_dev\_actual=dev\_labels

dev\_labels=np.array([dev\_labels]).T

defmini\_batch\_gradient\_descent(train\_set,train\_labels,batches):

indices=list(range(len(train\_set)))

num\_training\_indices=int(len(train\_set)/batches)

np.random.shuffle(indices)

train\_indices=indices[:num\_training\_indices]

result\_set,result\_labels=train\_set.iloc[train\_indices],train\_labels.iloc[train\_indices]

returnresult\_set,result\_labels

train\_set,train\_labels=mini\_batch\_gradient\_descent(train\_set,train\_labels,batches)

n=len(train\_set)

train\_labels=np.array(train\_labels).T

y\_train\_actual=train\_labels

train\_labels=np.array([train\_labels]).T

#Creatingonehotlabelsfory------------------

defone\_hot\_labels(y):

temp=y

one\_hot\_labels=np.zeros((n,NumberOfLabels))

print(one\_hot\_labels)

foriinrange(n):

one\_hot\_labels[i,temp[i]-1]=1

y=one\_hot\_labels

returny

train\_labels=one\_hot\_labels(train\_labels)

#NNwithsoftmax,hotcoding,crossentropy

classNeuralNetwork(object):

def\_\_init\_\_(self):

self.InputNumColumns=InputColumns#columns

self.OutputSize=NumberOfLabels#Categories

self.HiddenUnits=3#onelayerwithhunits

self.n=n#numberoftrainingexamples,n

self.activation\_func=activation\_func

print("InitializeNN:\n")

#RandomW1

self.W1=np.random.randn(self.InputNumColumns,self.HiddenUnits)#cbyh

print("INITW1is\n",self.W1)

#RandomW2

self.W2=np.random.randn(self.HiddenUnits,self.OutputSize)#hbyo

print("INITW2is:\n",self.W2)

#Randombiasforhiddenlayer

self.b=np.random.randn(1,self.HiddenUnits)

print("INITb'sare:\n",self.b)

#Randombiasforlastlayer

self.c=np.random.randn(1,self.OutputSize)

print("INITcis\n",self.c)

defFeedForward(self,X):

print("FeedForward\n\n")

self.z=(np.dot(X,self.W1))+self.b

#XisnbycW1iscbyh-->nbyh

print("Z1is:\n",self.z)

ifself.activation\_func:

self.h=self.ReLU(self.z)#activationfunctionshape:nbyh

print("His:\n",self.h)

else:

self.h=self.Sigmoid(self.z)#activationfunctionshape:nbyh

print("His:\n",self.h)

self.z2=(np.dot(self.h,self.W2))+self.c#nbyh@hbyo-->nbyo

print("Z2is:\n",self.z2)

#UsingSoftmaxfortheoutputactivation

output=self.Softmax(self.z2)

returnoutput

defSigmoid(self,s,deriv=False):

ifnotderiv:

return1/(1+np.exp(-s))

returns\*(1-s)

defSoftmax(self,M):

expM=np.exp(M)

SM=expM/np.sum(expM,axis=1)[:,None]

returnSM

defReLU(self,re,deriv=False):

ifderiv:

return1.\*(re>0)

returnnp.maximum(0,re)

defBackProp(self,X,y,output):

print("\n\nBackProp\n")

self.LR=LR

self.LRB=LRB

#Y^-Y

self.output\_error=output-y

print("Y^-Y\n",self.output\_error)

#NOTETOREADER........................

#Here-weDONOTmultiplybyderivativeofSigfory^b/cweareusing

#crossentropyandsoftmaxforthelossandlastactivation

#REMOVED#self.output\_delta=self.output\_error\*self.Sigmoid(output,deriv=True)

#Sotheabovelineiscommentedout...............

self.output\_delta=self.output\_error

#(Y^-Y)(W2)

self.D\_Error\_W2=self.output\_delta.dot(self.W2.T)#D\_ErrortimesW2

#(H)(1-H)(Y^-Y)(Y^)(1-Y^)(W2)

#WestillusetheSigmoidonH

ifself.activation\_func:

self.H\_D\_Error\_W2=self.D\_Error\_W2\*self.ReLU(self.h,deriv=True)

else:

self.H\_D\_Error\_W2=self.D\_Error\_W2\*self.Sigmoid(self.h,deriv=True)

#Notethat\*willmultiplyrespectivevaluestogetherineachmatrix

#print("DerivativesigHis:\n",self.Sigmoid(self.h,deriv=True))

#print("self.H\_D\_Error\_W2is\n",self.H\_D\_Error\_W2)

########------UPDATEweightsandbiases------------------

#XT(H)(1-H)(Y^-Y)(Y^)(1-Y^)(W2)

self.X\_H\_D\_Error\_W2=X.T.dot(self.H\_D\_Error\_W2)#thisisdW1

#(H)T(Y^-Y)-

self.h\_output\_delta=self.h.T.dot(self.output\_delta)#thisisfordW2

print("Usingsumgradient........\n")

self.W1=self.W1-self.LR\*(self.X\_H\_D\_Error\_W2)#cbyhadjustingfirstset(input->hidden)weights

self.W2=self.W2-self.LR\*(self.h\_output\_delta)

print("Thesumofthebupdateis\n",np.sum(self.H\_D\_Error\_W2,axis=0))

print("Thebbiasesbeforetheupdateare:\n",self.b)

self.b=self.b-self.LRB\*np.sum(self.H\_D\_Error\_W2,axis=0)

print("Updatedbsare:\n",self.b)

self.c=self.c-self.LR\*np.sum(self.output\_delta,axis=0)

print("TheW1is:\n",self.W1)

print("TheW1gradientis:\n",self.X\_H\_D\_Error\_W2)

print("TheW2gradientis:\n",self.h\_output\_delta)

print("Thebiasesbgradientis:\n",np.sum(self.H\_D\_Error\_W2,axis=0))

print("Thebiascgradientis:\n",np.sum(self.output\_delta,axis=0))

################################

defTrainNetwork(self,X,y):

output=self.FeedForward(X)

print("OutputinTNN\n",output)

self.BackProp(X,y,output)

returnoutput

defpredict(self,X):

output=self.FeedForward(X)

returnnp.argmax(output,axis=1)+1

MyNN=NeuralNetwork()

TotalLoss=[]

foriinrange(epoch):

print("\nRUN:\n",i)

output=MyNN.TrainNetwork(train\_set,train\_labels)

print("Theoutputis:\n",output)

MaxValueIndex=np.argmax(output,axis=1)

print('Predictiony^is\n',MaxValueIndex+1)

df\_confusion=pd.crosstab(y\_train\_actual,MaxValueIndex+1)

print("Theconfusionmatrixbetweenactuallabelandpredictlabel(trainingset):\n")

print("Rowisactual,Colispredicted:")

print(df\_confusion)

#UsingCategoricalCrossEntropy...........

loss=np.mean(-train\_labels\*np.log(output))#Weneedytoplacethe"1"intherightplace

print("Thecurrentaveragelossis\n",loss)

TotalLoss.append(loss)

predicted=MyNN.predict(dev\_set)

print("Theactuallabel:")

print(y\_dev\_actual)

print("Thepredictlabel:")

print(predicted)

df\_confusion=pd.crosstab(y\_dev\_actual,predicted)

print("Theconfusionmatrixbetweenactuallabelandpredictlabel(devset):\n")

print("Rowisactual,Colispredicted:")

print(df\_confusion)

defaccuracy(y\_true,y\_pred,normalize=True):

accuracy=[]

foriinrange(len(y\_pred)):

ify\_pred[i]==y\_true[i]:

accuracy.append(1)

else:

accuracy.append(0)

ifnormalize==True:

returnnp.mean(accuracy)

ifnormalize==False:

returnsum(accuracy)

print("Theaccuracyscoreis:")

print(accuracy(y\_dev\_actual,predicted,True))

#Plottotalloss

fig1=plt.figure()

ax=plt.axes()

x=np.linspace(0,10,epoch)

ax.plot(x,TotalLoss)

plt.show()

importpandasaspd

fromtensorflowimportkeras

fromsklearn.preprocessingimportLabelEncoder

fromsklearn.preprocessingimportStandardScaler

fromsklearn.model\_selectionimporttrain\_test\_split

importnumpyasnp

####################################################################################

#Example1

#Orangesvs.Grapefruit:diameter,weight,andcolordata

####################################################################################

#Thisisa2-labeldataset(0and1)

url="https://drive.google.com/file/d/1Pf7wp0PhskFbbNZaAkNgDRbBOcwu47PV/view?usp=sharing"

url='https://drive.google.com/uc?id='+url.split('/')[-2]

data=pd.read\_csv(url)

print("Checkfirst10rowsofdataset\n",data.head(10))

#splittargetandfeatures

x=data.iloc[:,1:]

y=pd.DataFrame([1ifeach=="orange"else0foreachindata['name']],columns=["target"])

#FeatureScaling,normalizedata

sc=StandardScaler()

x=sc.fit\_transform(x)

print('Thenormalizedfeatures:',x)

#Thenletscreatex\_train,y\_train,x\_test,y\_testarrays

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=42)

#MODEL1-forthetwo-labeldataset

model1=keras.Sequential([

keras.layers.Dense(3,input\_shape=(5,),activation="relu"),#firsthiddenlayer

keras.layers.Dense(2,activation="relu"),#Thisisthesecondhiddenlayerwithtwounits

keras.layers.Dense(1,activation="sigmoid")])#weareusing0or1here,sowehaveoutputsizeof1.

model1.compile(

#optimizer='SGD',

optimizer="Adam",

loss=keras.losses.MeanSquaredError(),

#loss="categorical\_crossentropy",

metrics=["accuracy"])

model1.fit(x\_train,y\_train,epochs=50)

Validation\_Loss,Validation\_Accuracy=model1.evaluate(x\_test,y\_test)

print('Thevalidationlossandaccuracy:',Validation\_Loss,Validation\_Accuracy)

#93.1%sgd+meansquareerror

#49.3%sgd+categorical\_crossentropy

#50.1%adam+categorical\_crossentropy

#94.4%adam+meansquareerror

#3unitsinfirstlayer2unitsinthesecondlayeroutputforsigmoid94.4

#3unitsinfirstlayerthenoutputforsigmoid93.3%

#3unitsinfirstlayer2unitsinthesecondlayeroutputforsigmoid94.4

#3unitsinfirstlayer3unitsinthesecondlayeroutputforsigmoid93.4

#2unitsinfirstlayer3unitsinthesecondlayeroutputforsigmoid93.1

#2unitsinfirstlayer2unitsinthesecondlayeroutputforsigmoid93.3

####################################################################################

#Example2

#academicsuccessw/one-hotencoding,labelswillbe0,1,2(dropout,graduate,enrolled)

####################################################################################

#Thisisa3-labeldataset(0,1,2)

url="https://drive.google.com/file/d/1qFziUtrehFwy4vHa67Q3taJs\_zQ6EMuQ/view?usp=sharing"

url='https://drive.google.com/uc?id='+url.split('/')[-2]

data=pd.read\_csv(url)

print("Checkfirst10rowsofdataset\n",data.head(10))

#splittargetandfeatures

x=data.iloc[:,:-1]

le=LabelEncoder().fit(data.iloc[:,-1])

y=le.transform(data.iloc[:,-1])

#FeatureScaling

sc=StandardScaler()

x=sc.fit\_transform(x)

#onehotlabels

temp=y#thetrainingsetlabels

one\_hot\_labels=np.zeros((len(y),3))

foriinrange(len(y)):

one\_hot\_labels[i,temp[i]-1]=1

y=one\_hot\_labels

print("Theone-hotfortraininglabels\n",y)

#Thenletscreatex\_train,y\_train,x\_test,y\_testarrays

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=42)

#MODEL2-forthethree-labeldataset

model2=keras.Sequential([

keras.layers.Dense(18,input\_shape=(36,),activation="relu"),#Hiddenlayer1

keras.layers.Dense(12,activation='sigmoid'),#Hiddenlayer2

keras.layers.Dense(8,activation='sigmoid'),#Hiddenlayer3

keras.layers.Dense(6,activation="relu"),#Hiddenlayer4

keras.layers.Dense(3,activation='softmax')#outputlayer

])

model2.compile(

#optimizer='SGD',

optimizer="Adam",

#loss=keras.losses.MeanSquaredError(),

loss="categorical\_crossentropy",

metrics=["accuracy"])

model2.fit(x\_train,y\_train,epochs=100)

Validation\_Loss,Validation\_Accuracy=model2.evaluate(x\_test,y\_test)

print('Thevalidationlossandaccuracy:',Validation\_Loss,Validation\_Accuracy)

#5layer,relu,relu,sigmoid,sigmoid,softmax;10,5,3,3,373.2%

#5layer,relu,relu,sigmoid,sigmoid,softmax;18,12,8,6,374.5%

#4layer,relu,relu,sigmoid,softmax;18,12,6,373.5%

#4layer,relu,relu,sigmoid,softmax;10,5,3,373.3%

#5layer,sigmoid,sigmoid,relu,relu,softmax;18,12,8,6,375.2%

#59.4%sgd+meansquareerror

#74.9%sgd+categorical\_crossentropy

#75.7%adam+categorical\_crossentropy

#75.2%adam+meansquareerror

CNN:

For convolution we must first rotate 180 degrees

Cross-correlation and convolution are both operations applied to images. Cross-correlation means sliding a kernel (filter) across an image. Convolution means sliding a flipped kernel across an image.

1) Let’s first think about images.

2) If an image is b & w, it may have a “depth” of one pixel.

3) If an image is RGB, it will have a “depth” of three pixels…

What is Image Classification?

Image classification is the process of segmenting images into different categories based on their features.

A feature could be the edges in an image, the pixel intensity, the change in pixel values, and many more.

An image consists of the smallest indivisible segments called pixels.

Every pixel has a strength often known as the pixel intensity.

Digital images usually comes with three color channels - Red-Green-Blue (RGB)

Why RGB? - Because it has been seen that a combination of these three can produce all possible color pallets.

A screenshot of a computer

Description automatically generated with low confidence

Text, calendar

Description automatically generated

Text, schematic

Description automatically generated

Keras with CPU vs GPU

|  |  |
| --- | --- |
| Notwbook seetings GPU | TPU |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | #-\*-coding:utf-8-\*-  """  CreatedonWedOct1909:45:052022  ForthispartoftheAssignment,yourgoalistodiginabitdeeper.Therefore,youwilltakeseveralsteps.    Step1:UseTHISCODELinkstoanexternalsite.togetabasicmultilayer2DconvolutionalNNworkingforacolor  dataset.Everythingyouneedisinthecode.    Step2:Onceyougetthecodeworking,createaPowerPointTutorialthatincludesallthecodeandexplanations  forwhateachlineofcodeisdoing.Includewhysizesandshapesareasnotedinthecode.Includevisualexamples  asoftenaspossible.PretendthatitisyourjobtoEXPLAINandILLUSTRATEeachsteptoapersonwhoisnewtothe  topic.    Forexample-eachslideinyourPowerPointSetmighthave1-5(with5asthemax)linesofcodethatworktogether  orindividuallytoperformatask.Youwillincludeandexplainthecode.Whenpossible,offerinsightful  illustrations.IncludeOUTPUTonslidesasyougo.ReallyassurethataviewercanSEEandunderstandwhatisgoingon.    Thereare1000sofwaystodothis:)Becreativeandclear.  @author:profa  """  #ImageProcessingPython  #https://note.nkmk.me/en/python-numpy-image-processing/    importtensorflowastf  importmatplotlib.pyplotasplt    (train\_images,train\_labels),(test\_images,test\_labels)=tf.keras.datasets.cifar10.load\_data()    print(type(train\_images))  print(train\_images.shape)#50000rows,32by32,depth3  plt.imshow(train\_images[2])  plt.show()    print(train\_images[0,:,:,0])  print(train\_images[0,:,:,0].shape)    #Normalizepixelvaluestobebetween0and1  train\_images,test\_images=train\_images/255.0,test\_images/255.0  class\_names=['airplane','automobile','bird','cat','deer',  'dog','frog','horse','ship','truck']    plt.figure(figsize=(10,10))  foriinrange(25):  plt.subplot(5,5,i+1)  plt.xticks([])  plt.yticks([])  plt.grid(False)  plt.imshow(train\_images[i])  #TheCIFARlabelshappentobearrays,  #whichiswhyyouneedtheextraindex  plt.xlabel(class\_names[train\_labels[i][0]])  plt.show()    model=tf.keras.models.Sequential()  #https://www.tensorflow.org/api\_docs/python/tf/keras/layers/Conv2D  model.add(tf.keras.layers.Conv2D(32,(3,3),activation='relu',input\_shape=(32,32,3)))  #https://www.tensorflow.org/api\_docs/python/tf/keras/layers/MaxPool2D  model.add(tf.keras.layers.MaxPooling2D((2,2)))  model.add(tf.keras.layers.Conv2D(64,(3,3),activation='relu'))  model.add(tf.keras.layers.MaxPooling2D((2,2)))  model.add(tf.keras.layers.Conv2D(64,(3,3),activation='relu'))    model.summary()    model.add(tf.keras.layers.Flatten())  model.add(tf.keras.layers.Dense(64,activation='relu'))  model.add(tf.keras.layers.Dense(10))    model.summary()    model.compile(optimizer='adam',  loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),  metrics=['accuracy'])    history=model.fit(train\_images,train\_labels,epochs=10,  validation\_data=(test\_images,test\_labels))    plt.plot(history.history['accuracy'],label='accuracy')  plt.plot(history.history['val\_accuracy'],label='val\_accuracy')  plt.xlabel('Epoch')  plt.ylabel('Accuracy')  plt.ylim([0,1])  plt.legend(loc='lowerright')  plt.show()    test\_loss,test\_acc=model.evaluate(test\_images,test\_labels,verbose=2)    print(test\_acc)      Tensorflow    1 filter is a collection of kernel\  Show me the image of maxpooling, conv2d |  |  |
|  |  | Canvas  In class  Perception  1hiddne laye  CNN  RNN  Filling 1 foel  Muliple trchoiue answer choose all the true    Dl/dw11 = dl/dy^ \* dy^/dz = (y^ - y)    No decrive for CNN    1 vector and mutiple vector    8 questions |
|  |  |  |