7 Training labeled data with Back propagation VS ML algorithm

Aim: To train labeled data using back propagation and compare the test loss, test accuracy, train loss and train accuracy.

Description: I. Updation of weights based on activation function:

→ we have used two different activation functions relu and sigmoid which introduce non-linearity.

→ Here we observe that activation function doesnot directly ~~udp~~ Update weights instead it influences weight updates

(1) RELU: $f(x) = \max\{x, 0\}$

Forward Pass → during forward pass, $f(x)$ is applied to accumulate the output for backpropagation.

Backward Pass → during back propagation, gradient is calculated using its deriviative, Derivative is 1 if ~~+ve~~, and 0, otherwise

weights updation → In this program, SGD is the optimizer which then uses these gradients to update weights for Relu

Ex: In hidden layer 2 consisting of 4 neurons, when relu was applied the weights were updated as follows:

(Using get_weights ( )[0])

o/p: weights:

[-1.8034391e-01 - 1.5058946e-02 .... -1.2755118e-02]

⋮

[4.5094218e-02 -1.4791060e-02 ... 3.3095229e-02] relu deactivate neurons that have given a -ve derivative as o/p by initializing such neurons weights to zero.

② Sigmoid $= F(x) = \frac{1}{1+e^{-x}}$

F.P → during F.P, F(x) is applied to accumulate O/Ps for back-Propagation, in range [0,1]

B.P → during B.P, gradient is calculated using its derivative

$$\frac{\delta o}{\delta v} = o(1-o)$$

weight updation → In this Prg, SGD and Adam were the two optimizers individually used to sigmoid activation function

wts: [[-0.1473886]
[0.03395729]
[-0.6065672]
[0.04672604]]

→ sigmoid squashes values to range (0,1)

II. Accuracy score and loss function formulae:

Loss functions:

(i) Neural Network for Binary Cross Entropy → binary cross entropy

(ii) ANN logistic regression → binary cross entropy

(iii) SVM 1 → hinge loss

(iv) SVM 2 → hinge loss

(v) Non-linear SVC → hinge loss

(vi) logistic regression → binary cross entropy

(vii) Linear Discriminant Analysis with SVC → hinge loss.

(viii) Linear Discriminant Analysis with non-linear SVC → hinge loss

→ log loss = $-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(P_{ij})$

→ hinge loss = $\max\{0, 1-y(\bar{w}\cdot \bar{x})\}$

Accuracy score :- It is the % of correct predictions in given number of predictions Imported from 'sklearn.metrics'

→ accuracy = accuracy_score(true_labels, predicted_labels

→ formula for Accuracy score in terms of TP, TN, FP & F-N

$$\text{Accuracy} = \frac{(TP+TN)}{(TP+TN+FP+FN)}$$

→ Alternatively, the accuracy can be calculated by using the mean squared error (MSE) or the R-square statistical method.

III Confusion matrix :- → It is a tool for evaluating the performance of a neural network model

(Target class)

O/P class	class 0	class 1
class 0	(TP) True Positive	(TN) True Negative
class 1	(FP) False Positive	(FN) False negative

→ It compares the actual labels with the predicted labels and counts how many times they match or mismatch.

(i) TP: Actual value equals predicted value

(ii) TN: Actual value is not equal to predicted value.

(iii) FP: False prediction of -ve class labels to be +ve

Learning models:

a) Linear SVM

b) SVM with RBP Kernel

c) Logistic uegression

d) Linear Discriminant Analysis (with SVC, NL SVC)

e) SVC (model 2, NL-SVC)

Linear Support Vector Machine:

→ It is a supervised machine learning algorithm used for classification or regression.

→ It identifies the best classifies and support vectors

→ It allows us to divide 2 different classes, using a solid line.

loss: hinge loss = $mass(0, 1 - y(\bar{w} \cdot \bar{x}))$

Activation: Linear function; $f(x) = x$

Support Vector Machine with Radial Basis Function Kernel:-

→ It is machine Learning algorithm consisting of a kernel that is used for unsupervised / clustered data points.

→ Such data points cannot be linearly repeated.

→ Therefore, a radial basis function (RBF) is used that can radically seperate clustors or non linearly seperable data

→ It is a unsupervised structure engineering method which is good for noisy data.

Logistic regression:

→ It is a machine learning algorithm used to classify instances in terms of probalities.

→ It is used to learn the parameters of a model by using maximum livelihood estimation and uses Bayes rules for probabilities.

Loss function: $\mathcal{L} = -\log(|y_i/2 - 0.5 + \hat{y_i}|)$ or $\hat{y_i} = 1/(1 + \exp(-\bar{w}\cdot\bar{x}))$

activation function: (Sigmoid) $f(x) = 1/(1+e^{-x})$

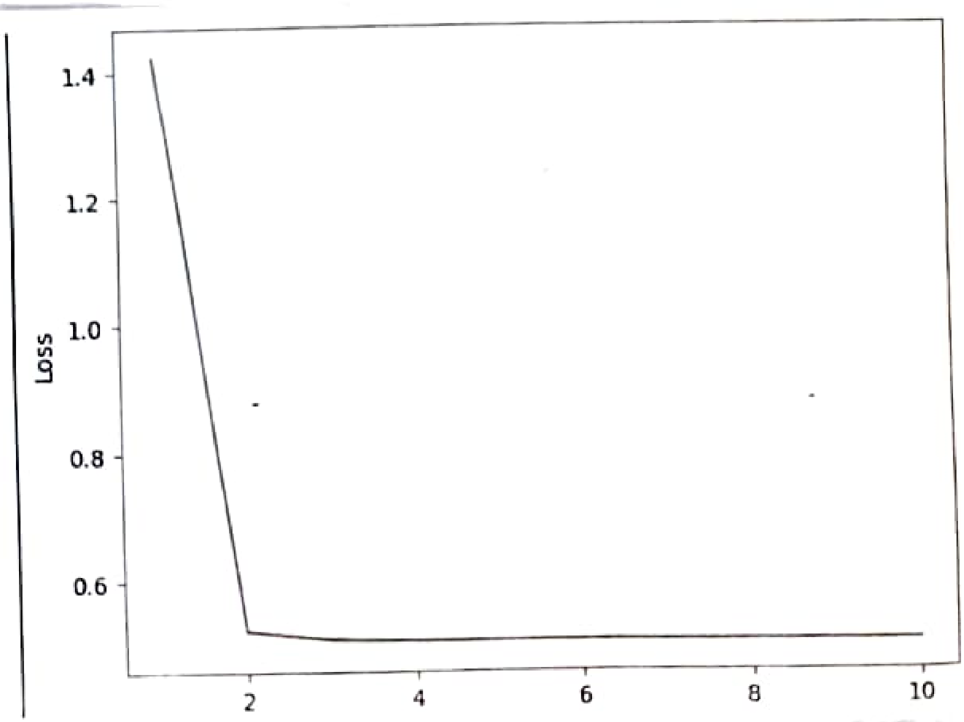
Linear Discriminant Analysis:-

→ It is a supervised learning algorithm used for classification tasks in machine Learning.

→ It is used to find a linear Combination of features that seperate classes.

Loss: Squared error loss or even hinge loss is used

$$l = \max(0, 1 - y(\bar{w}\cdot\bar{x}))$$



ANN model with GD

```
import Keras
from Keras.model import sequential
from tensorflow import Dense
from tensorflow Keras import layers, optimizers
model = sequential()
model.add(Dense(Units=4, Kernel-initializer='uniform', activati
    ='relu', input_dim=8))
model.add(Dense(Units=4, Kernal-initializer='uniform', activa
    ='relu'))
model.add(Dense[units=1, Kernel-initializer='uniform', activat
    ='sigmoid'))
Learning_rate=0.01
optimizer= optimizers.SGD(Learning_rate = learning_rate]
model.compile(optimizer= optimizer, loss='binary_crossentropy', metr
    =['accuracy'])
model.summary()
history= model.fit(x_train, y_train, batch_size=2, epochs=10)
Plt.plot(history.history['loss'])
plt.title('EPOCH VS LOSS')
plt.ylabel('loss')
plt.xlabel('epoch')
for layers in model.layers:
        if hasattr(Layer, 'weights'):
            Print(layer.get_weight()[0])
from sklearn.metrices import Confusion matrix
```

```
cm= confusion_matrix (y_test, np.round(y_pred))
 print(cm)

Loss, accuracy = model.evaluate(
 print("test loss:", loss)
 print("test accuracy:", accuracy)
 loss, accuracy = model
 print ("Train loss:", loss)
 print ("Train accuracy:", accuracy)
```

Output: MODEL SUMMARY
model: "sequential"

Layer(type)	Output Shape	Param #
dense(Dense)	(None, 4)	36
dense_1(Dense)	(None, 4)	20
dense_2(Dense)	(None, 1)	5

Total params : 61
Trainable params: 61
Non-trainable params: 0

```
Epoch 1/10
400/400 [==========] Loss: 0.6282 accuracy: 0.7985
Epoch 10/10
400/400 [==========] Loss: 0.5011 accuracy: 0.7997
```

Predicted:

```
[[0.20175475]
 :
 [0.20175475]]
[0 0 0 0 0 0 1 0 0 0]
```

Layer Name: dense-~~as~~
weights:
[[-1.0014366e-02 ...... 2.9499233e-02]
:
[-4.6992740e-01 .... -1.7125118e+00]]
Layer Name: dense_1
weights
[[-3.7107181e-02 ..... -2.0850260e+00]
:
[2.5169026e-02 .... -1.4069778e+00]
Layer Name: dense_2
weights:
[[0.07775325]
:
[2.5431979]]
Confusion matrix:
[[156 0]
[44 0]]
7/7[=========] loss: 05279 accuracy: 0.7997
train loss: 0.5207919294499.
test accuracy. 0.7799997 1339
25/25[======] loss: 0.5008 accuracy: 0.7997
train loss: 0.50075632333
train accuracy: 0.7997496724

SVC(1)

code: from sklearn.metrices import accuracy-score
from sklearn. preprocessing import standard scaler
from sklearn. pipeline import pipeline
from sklearn. metrices import hinge-loss
ML techniques 'SVC(1)
from sklearn svm import linear SVC
clf = linear SVC(c=1, loss = "hinge")
clf = fit (x-train, y. train)
Loss, accuracy = model.evaluate (x. train, y-train)
Print(" Train loss:", loss)
Print(" Train accuracy:" accuracy)
y-pred = clf. predict (x-test)
accuracy = accuracy-score(y-test, y-pred)
Print(" Test accuracy:", accuracy)
avg-hinge-loss = hinge_loss (y-test, decision-scores)
Print(" Test loss:", avg_hinge_loss)

Output. 23/25 [========] loss: 0.4963 accuracy: 0.8030
25/25 [========] loss: 0.5007 accuracy: 0.7997

Train loss: 0.5007498860389192
Train accuracy: 0.7997496724128723
Test accuracy : 0.275
Test loss: 0.3536984461120856

SVC(2)

Code: from sklearn. svm. import lean SVC
clf = Pipeline([("scaler", standard scalar()), ("linear_svc", linear svc
(C=1, loss "hinge"))])
clf. fit(x-train, y-train)

y-pred= clf.predict(x-test)
accuracy = accuracy-score(y-test, y-pred)
Print("Test accuracy:", accuracy)
desicion-scores = clf.descision-function(x-test)
avg-hinge-loss = hinge-loss(y-test, descision-scores)
Print("Test loss:", avg-hinge-loss)
loss, accuracy= model.evaluate(x-train, y-train)
Print("Train loss:", loss)
Print("Train accuracy", accuracy)

Output: Test accuracy :0.78
Test Loss: 0.4400000000008543
25/25[= = = = = = = = =] loss: 0.5008 accuracy: ...
Train loss: 0.500771939754486l
Train accuracy: 0.79974967241 28723

non-linear SVC

<u>Code</u>:- from sklearn.svm import svc
clf= Pipeline([("scaler", standard.scalar()), ("svm.clf", ... (Kernel
= "rbf", gamma=5, C=1)),])
clf.fit(x-train, y-train)
y-pred= clf.predict(x-test)
accuracy= accuracy score(y-test, y-pred)
desicion-scores = clf.decision-function(x-test)
avg-hing-loss = hinge-loss(y-test, desicion-scores)
loss, accuracy= model.evaluate(x-train, y-train)

Output: Test accuracy: 0.78
Test loss: 0.572415188538 4846
25/25[= = = = = = = = =] loss 0.5008 accuracy: 0.7997
Train loss: 0.500771939734 4861
Train accuracy: 0.7997496724128723

Code: # logistic regression
```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(random_state = 42)
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
y_train_pred = clf.predict(x_train)
train accuracy = accuracy_score(y_train, y_train_pred)
loss = model.evaluate(x_train, y_train)
Loss = model.evaluate(x_train, y_test)
```
Output: Train Accuracy: 0.7909887359198 98
Test Accuracy: 0.765
Train Loss: 0.79974967241 18733
Test loss: 0.79974967213 897705

Code: # linear discriminant Analysis with SVC
```
from sklearn.dicriminant_analysis import LinearDiscriminantAnalysis
from sklearn.svm import SVC
clf = LinearDiscriminantAnalysis()
clf = fit(x_train, y_train)
y_pred = clf.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
loss = model.evaluate(x_test, y_test)
loss, accuracy = model.evaluate(x_train, y_train)
```

Output:- Test Accuracy: 0.825
Test loss: 0.77999997713897705
Train loss: 0.500749836035 9192
Train accuracy: 0.799749674241 28722

Code: # LDA with NL-SVC

```
from sklearn.discriminant_analysis import linear
        Discriminant Analysis
from sklearn.svm import SV
lda = linear Discriminant Analysis()
lda.fit [x-train, y-train)
x-train-lda = lda.transform (x-train)
x-test-lda = lda.tranform(x-test)
clf = pipeline[("scalar", standard scalar()), ('svm-clf",
                                  SVC(kernel
                                  "rbf", gamma=
                                  c=1))])
clf.fit (xtrain-lda, y-train)
y-pred = clf.predict(x-test-lda)
accuracy = accuracy-score(y-test, y-pred)
decision-scores = clf.decision-function(x-test-lda)
avg-hinge-loss = hinge-loss(y-test, decision-scores)
Loss, accuracy = model.evaluate(x-train, y-train)
```

Output: Test Accuracy: 0.335
Test loss: 0.35369844611120886
Train Loss: 0.5007719397544861
Train Accuracy: 0.7997496724128723

Code:

```
from sklear.linear-model import logistic Regression
model = squential()
model.add[Dense(units=1, Kernel-initializer='uniform
```

```
                activation='sigmoid', input-dim=8))
model.compile(optimizer=('adam', loss= binary-crossentropy
              ; metrics= ["accuracy"])

y-pred= model.predict(x-test)
model.summary
history= model.fit(x-train, y-train, batchsize=2, epochs=
                                                        10)

for layer in model.layers:
        if hasattr(layer, 'weights'):
      Print(layer.get-weights()[0])
from sklearn.metrices import confusion-matrix
cm = confussion-matrix(y-test, np.round(y-pred))
Print(cm)
```

Output :- Model: "sequential"

Layer(type)	Output Shape	Param #
dense-25(Dense)	(None, 1)	9

Total params: 9

Trainable Params: 9

Non-Trainable Params : 0

Epoch 1/10

400/400 [= = = = = = = = = =] loss: 185.917 accuracy: 0.6846

:

Epoch 10/10

400/400 [= = = = = = = = = = = =] loss: 29.7861 accuracy: 0.6834

Layer*name : dense

Weights:

[[-2.5827095e-02]
 [1.9932108e-01]
 :
 [6.7915698e-04]]

Confusion matrix:

[[156 0]
 [44 0]]

test loss: 70.51509857177734

test accuracy: 0.27000001072883606

train loss: 72.27491760253906

train accuracy: 0.2215269088894279 48.

Result / observations:-

	Test loss	Test accuracy	Train Loss	Train accuracy
ANN with SGD	0.52	0.78	0.50	0.799
SVC(1)	0.35	0.65	0.50	0.799
SVC(2)	0.44	0.78	0.50	0.799
NL-SVC	0.57	0.78	0.50	0.799
logistic Regression	0.57	0.77	0.50	0.790
LDA	0.52	0.78	0.50	0.799
LDA(NL-SVC)	0.35	0.83	0.50	0.799
ANN to logistic Regression	0.70	0.27	0.72	0.22

(i) Maximum Accuracy for testing:
-> ANN with SGD, SVC(2), NL-SVC, LDA (with NL-SVC) provide highest accuracy.
-> LDA with Non linear SVC gives max accuracy of 83%.

(ii) Maximum Accuracy for training
-> except logistic regression all models gives ~80% accuracy (≃ 79.9%)

(iii) Minimum loss for testing and Training
-> SVC model 1 gives us a minimum loss of 44% and test accuracy of 78% which is close to train accuracy of 79%

-> Overall, LDA with non-linear SVC is the best performer in all ML algorithms and minimum test loss of 35%.
• logistic regression performed the worst with losses ranging upto 70%.

24/8/23