MACHINE LEARNING (CSCI 8950): HOMEWORK-4

Accompanying GitHub Repository:
https://github.com/Chintan2108/CSCI-8950-HW/tree/main/HW-4

MARCH 31, 2022

CHINTAN B. MANIYAR
chintanmaniyar@uga.edu

NOTE: Python's *tensorflow* package was used to implement the solution to this problem set.

For this homework, the *WisconsinBreastCancerDetection* dataset was used. It contains 9 predictor variables (all encoded) and one target variable which has two classes namely 'Benign' and 'Malignant', denoted by 2 and 4 respectively. The data was already encoded, however, there were some missing values in one of the predictor variables. Data cleaning was done by dropping the samples (as it was a categorical variable) with any missing values which reduced the sample size from 699 to 683.

Clump_Thickness	Cell_Size_Uni	Cell_Shape_Uni	Marginal_Adhesion	Single_Epithelial_Cell_Size	Bare_Nuclei	Bland_Chromatin	Normal_Nucleoi	Mitoses	Class
5	1	1	1	2	1	3	1	1	2
5	4	4	5	7	10	3	2	1	2
3	1	1	1	2	2	3	1	1	2
6	8	8	1	3	4	3	7	1	2
4	1	1	3	2	1	3	1	1	2

Figure 1: Snapshot of the Wisconsin Breast Cancer Dataset

A 2-layer neural network (Figure 2) was created, with one hidden layer activated by $ReLU$ (1) and the output layer activated by *sigmoid* (2). The hidden layer contained 5 to 40 units and the output layer contained one unit. The output from the network was thresholded to classify a cancer sample into either 'Malignant' or 'Benign' (3). Train-test split ratio was 80:20 which resulted in 564 samples for training and validation and the remaining 137 samples for testing and inference. The training set was further split into 436 samples of pure training and 110 samples of validation. The 137 testing samples were completely unseen by the model.

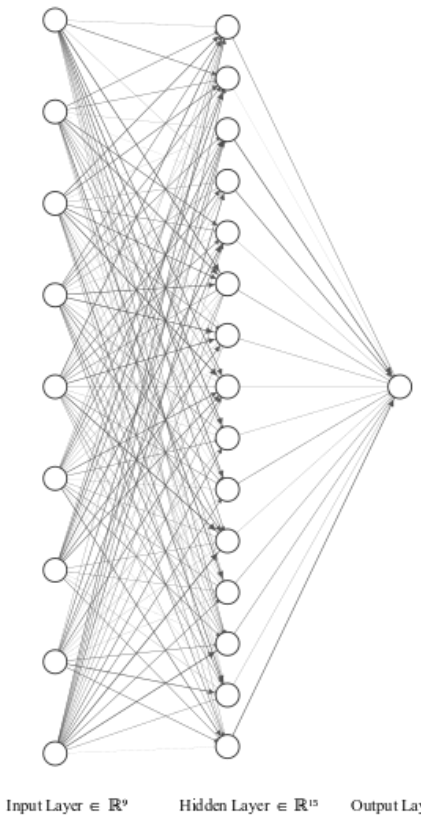
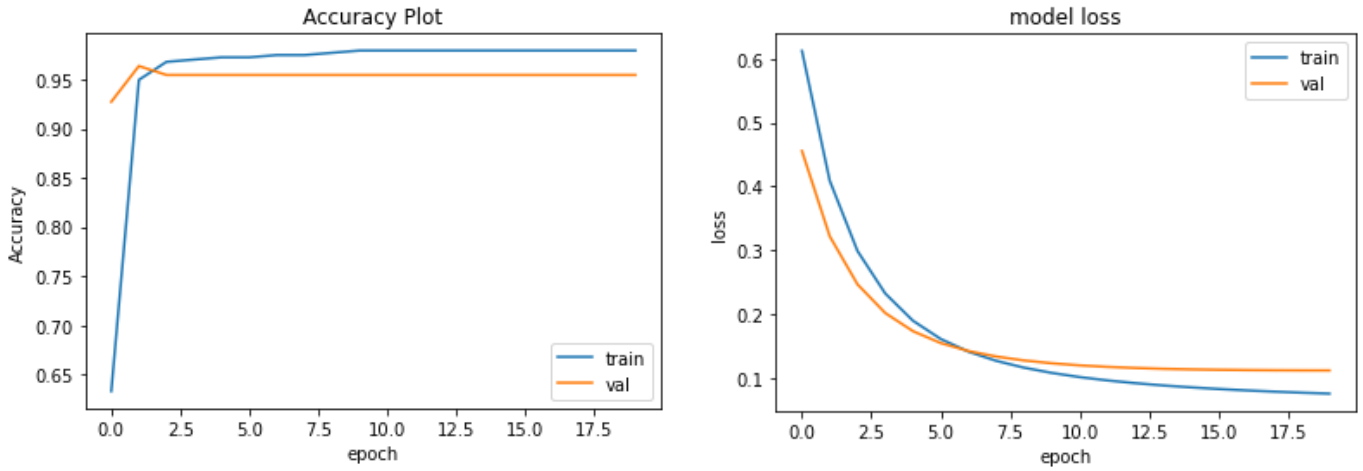


Figure 2: Network Structure with 9 input units, 15 hidden units and 1 output unit

The *binaryCrossEntropy* loss (4) was used to train the network and performed 10-fold cross validation for the seen dataset. Gradient descent was used to optimize the network training, and errors were back-propagated. Using validation set and callback techniques for early stopping, each training exercise was performed for 30 epochs. Figure 3 shows the average metrics of accuracy and loss for both training and validation datasets.



(a) *(b)*

Figure 3: (a) Accuracy variation in 30 epochs of training on training and validation sets, (b) Loss variation in 30 epochs on training and validation sets showing model convergence

$$f(x) = max(0, x) \text{ --- (1)}$$

$$g(x) = \frac{1}{1+e^{-x}} \text{ --- (2)}$$

$$x_i = \begin{cases} Benign, & o < 0.5 \\ Malignant, & o \geq 0.5 \end{cases} \text{ --- (3)}$$

where o=network output at the output layer which is essentially the probability of an ith sample x, belonging to the 'Benign' class; x_i=ith sample

$$BCE_{Loss} = \frac{1}{N}\sum_{i=1}^{N} -(y_i) \times \log(pr_i) + (1 - y_i) \times \log(1 - pr_i) \text{ --- (4)}$$

where N=total number of samples;

y_i=true label for ith sample;

pr_i=predicted probability of ith sample

Moreover, experiments were conducted on the network structure by steadily increasing the number of hidden units in the network, starting from 5 units to 40 units. Each network structure was trained independently to determine the best number of hidden units. Table 1 shows the metrics for different number of hidden units. Validation accuracy is shown as the average validation accuracy of 10 models with their respective standard deviations in the 10-fold cross validations, while the test accuracy on unseen dataset is shown for the best model out of the 10 models. Training time shows the longest training time taken by a model in the 10-fold cross validation. While one would

expect training time to strictly increase with increasing number of hidden units, it was interesting to see breaks in that trend (Figure 4).

Table 1: Model metrics for different hidden units in the neural network structure

Hidden Units	**Average Validation Accuracy**	**Best Test Accuracy**	**Training Time**
5	96.88 ± 3.08%	95.62%	8.29s
10	97.07 ± 2.73%	96.81%	7.96s
15	97.61 ± 3.03%	**97.54%**	8.17s
20	96.34 ± 2.94%	96.08%	8.53s
25	96.52 ± 2.89%	96.35%	8.52s
30	97.44 ± 2.73%	96.35%	9.13
35	96.89 ± 3.06%	96.08%	9.51s
40	96.70 ± 2.81%	96.08%	9.17s

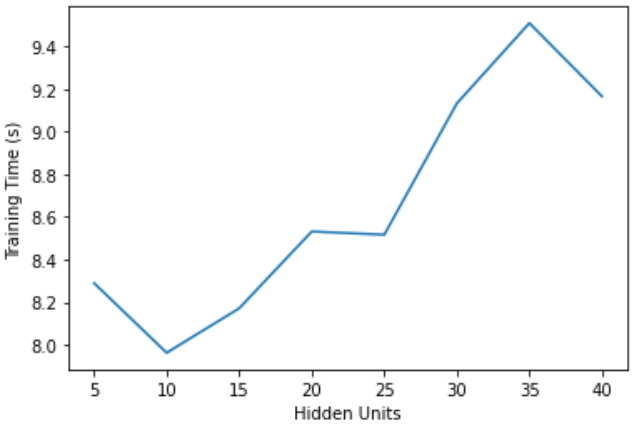


Figure 4: Training times for the model with different number of hidden units in the neural network

Based on these experiments and their performances (Table 1), a neural network with 15 hidden units is suggested for this problem, as it achieves the highest test accuracy on an unseen dataset.

References

TensorFlow documentation (https://www.tensorflow.org/api_docs)

APPENDIX-A: Codebase

```
#!/usr/bin/env python
# coding: utf-8

# **Name: Chintan B. Maniyar**
# <br>**MACHINE LEARNING - CSCI 8950**
# <br>**Homework-4: Back-propogation Neural Network** <br>
# <hr>

# importing packages
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Input, Dropout
from matplotlib import pyplot as plt
from sklearn.model_selection import StratifiedKFold
from time import time

# read data
features = ['Clump_Thickness', 'Cell_Size_Uni', 'Cell_Shape_Uni', 'Marginal_Adhesion',
 'Single_Epithelial_Cell_Size', 'Bare_Nuclei', 'Bland_Chromatin', 'Normal_Nucleoi', 'Mitoses', 'Class']
bcd = pd.read_csv('./breast_cancer/breast-cancer-wisconsin.data', names=features)
bcd.head()

bcd.describe()

# specifying missing values as NaNs
bcd['Bare_Nuclei'] = bcd['Bare_Nuclei'].replace('?', np.NaN)

# dropping tuples with null values
bcd = bcd.dropna()
bcd.describe()

X_df = bcd.iloc[:,0:-1]
y_df = bcd.iloc[:,-1].map({2:0, 4:1})

X = X_df.to_numpy()
X = StandardScaler().fit_transform(X)
y = y_df.to_numpy()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

X_train.shape, X_test.shape, y_train.shape, y_test.shape

def buildModel(features_nums, hidden_units):
    input = Input(shape=(features_nums), name='input_layer')
    x = Dense(hidden_units, activation='relu', name='hidden_layer')(input)
```

```
    output = Dense(1, activation='sigmoid', name='output_layer')(x)

    return Model(inputs=input, outputs=output)

def createModel(input_nums, hidden_nums):
    model = buildModel(input_nums, hidden_nums)

    # adding optimizer and loss function
    model.compile(loss='binary_crossentropy',optimizer='sgd', metrics=['accuracy'])
    return model

model.summary()

model.fit(X_train, y_train, epochs=10, validation_split=0.2)

history = model.fit(X_train, y_train, epochs=10, batch_size=10, validation_split=0.2, verbose=0)

history.history.keys()

model.evaluate(X_test, y_test)

# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Accuracy Plot')
plt.ylabel('Accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

# fix random seed for reproducibility
seed = 7
np.random.seed(seed)

X = X_train
Y = y_train

hidden_units = [5,10,5,20,25,30,35,40]
results = {}

for num_hidden_units in hidden_units:
    print('Hidden Units: %d' % num_hidden_units)

    # define 10-fold cross validation
    kfold = StratifiedKFold(n_splits=10, shuffle=True,random_state=seed)
    cvscores = []
    times = []
    for train, test in kfold.split(X, Y):
```

```
        model = createModel(X_train.shape[1], num_hidden_units)


        start = time()
        model.fit(X[train], Y[train], epochs=10, batch_size=10, verbose=0)
        end = time()

        scores = model.evaluate(X[test], Y[test], verbose=0)
        print("%s: {%.2f}" % (model.metrics_names[1], scores[1]*100))
        cvscores.append(scores[1] * 100)

    print("{%.2f} (+/- {%.2f})" % (np.mean(cvscores), np.std(cvscores)))
    print('*****************************************************************************')
results[num_hidden_units] = {'cvscores': cvscores, 'times': times, 'cvmean':np.mean(cvscores), 'cvstd':np.std(cvscores)}
```