

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

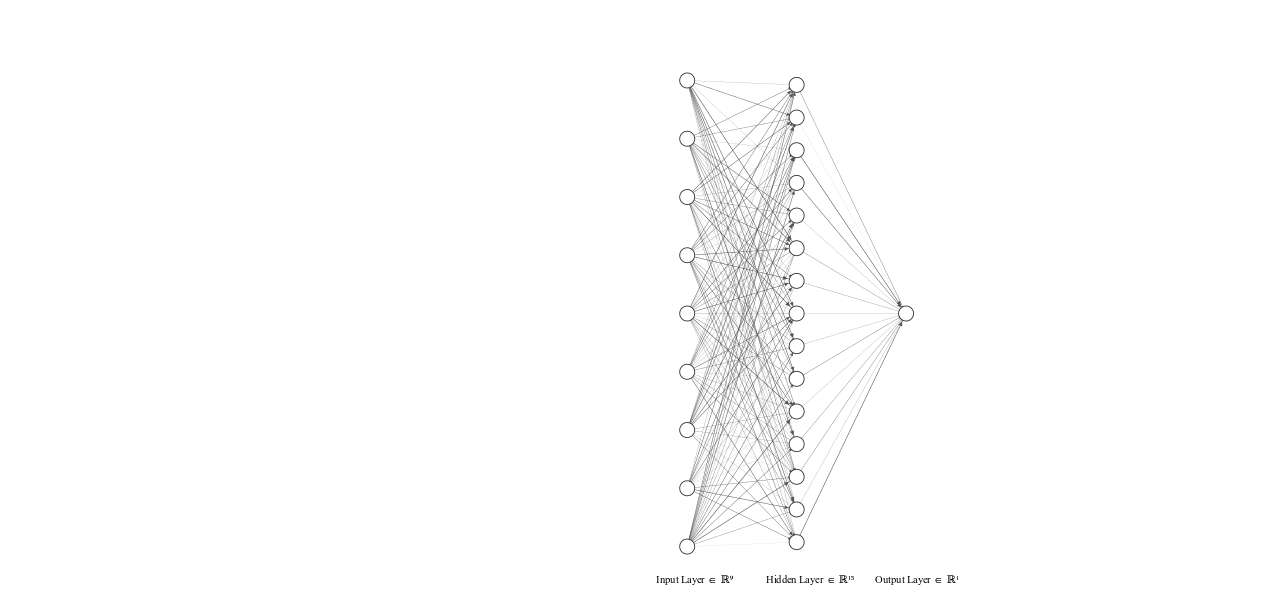
**NOTE:** Python’s *tensorflow* package was used to implement the solution to this problem set.

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.

A picture containing background pattern

Description automatically generated  
*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. 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* | |

--- (1)

--- (2)

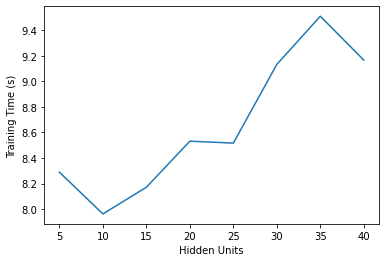
--- (3)  
where *o*=network output at the output layer which is essentially the probability of an ith sample *x*, belonging to the ‘Benign’ class; *xi*=ith sample

--- (4)  
where N=total number of samples;   
yi=true label for ith sample;   
pri=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 increasing 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% | 97.81% | 7.96s |
| 15 | 97.61 ± 3.03% | 98.54% | 8.17s |
| 20 | 96.34 ± 2.94% | 97.08% | 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% | 97.08% | 9.51s |
| 40 | 96.70 ± 2.81% | 97.08% | 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)

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)}