Machine Learning Lab3: Created by Jibrael Jos,PhD

Topic: Neural Network Explorations

Student Name: Naveen Krishna

Roll No:23122023

Date: 15 March

Submission : 4th April

In [ ]:
```
from __future__ import absolute_import, division, print_function

import tensorflow as tf
from tensorflow.keras import Model, layers
import numpy as np
```

In [ ]:
```
# Cancer dataset parameters.
num_classes = 2 # total classes
num_features = 10 # data features

# Training parameters.
learning_rate = 0.01
training_steps = 5000
display_step = 500

# Network parameters.
n_hidden_1 = 28 # 1st layer number of neurons.
n_hidden_2 = 56 # 2nd layer number of neurons.
```

In [ ]:
```
import pandas

df = pandas.read_csv("cancerAllv3.csv")

print(df)
```

```
     radius  texture  perimeter   area        s        c  concavity  
cp  \
0     17.99    10.38     122.80  1001.0  0.11840  0.27760    0.30010  0.14
710   
1     20.57    17.77     132.90  1326.0  0.08474  0.07864    0.08690  0.07
017   
2     19.69    21.25     130.00  1203.0  0.10960  0.15990    0.19740  0.12
790   
3     11.42    20.38      77.58   386.1  0.14250  0.28390    0.24140  0.10
520   
4     20.29    14.34     135.10  1297.0  0.10030  0.13280    0.19800  0.10
430   
..      ...      ...        ...     ...      ...      ...        ...      
...   
564   21.56    22.39     142.00  1479.0  0.11100  0.11590    0.24390  0.13
890   
565   20.13    28.25     131.20  1261.0  0.09780  0.10340    0.14400  0.09
791   
566   16.60    28.08     108.30   858.1  0.08455  0.10230    0.09251  0.05
302   
567   20.60    29.33     140.10  1265.0  0.11780  0.27700    0.35140  0.15
200   
568    7.76    24.54      47.92   181.0  0.05263  0.04362    0.00000  0.00
000   

        sym       fd  ...  texture2  perimeter2   area2       s2       c2  
\
0    0.2419  0.07871  ...     17.33      184.60  2019.0  0.16220  0.66560   
1    0.1812  0.05667  ...     23.41      158.80  1956.0  0.12380  0.18660   
2    0.2069  0.05999  ...     25.53      152.50  1709.0  0.14440  0.42450   
3    0.2597  0.09744  ...     26.50       98.87   567.7  0.20980  0.86630   
4    0.1809  0.05883  ...     16.67      152.20  1575.0  0.13740  0.20500   
..      ...      ...  ...       ...         ...     ...      ...      ...   
564  0.1726  0.05623  ...     26.40      166.10  2027.0  0.14100  0.21130   
565  0.1752  0.05533  ...     38.25      155.00  1731.0  0.11660  0.19220   
566  0.1590  0.05648  ...     34.12      126.70  1124.0  0.11390  0.30940   
567  0.2397  0.07016  ...     39.42      184.60  1821.0  0.16500  0.86810   
568  0.1587  0.05884  ...     30.37       59.16   268.6  0.08996  0.06444   

     concavity2     cp2    sym2      fd2  diagnosis  
0        0.7119  0.2654  0.4601  0.11890          1  
1        0.2416  0.1860  0.2750  0.08902          1  
2        0.4504  0.2430  0.3613  0.08758          1  
3        0.6869  0.2575  0.6638  0.17300          1  
4        0.4000  0.1625  0.2364  0.07678          1  
..          ...     ...     ...      ...        ...  
564      0.4107  0.2216  0.2060  0.07115          1  
565      0.3215  0.1628  0.2572  0.06637          1  
566      0.3403  0.1418  0.2218  0.07820          1  
567      0.9387  0.2650  0.4087  0.12400          1  
568      0.0000  0.0000  0.2871  0.07039          0  

[569 rows x 31 columns]
```

In [ ]:

```
features=['radius','texture','perimeter','area','s','c','concavity','cp',

import numpy as np
X = np.array(df)
y = X[:,30]
X = X[:,0:9]
```

```
print(X)
print(y)
```

```
[[1.799e+01 1.038e+01 1.228e+02 ... 3.001e-01 1.471e-01 2.419e-01]
 [2.057e+01 1.777e+01 1.329e+02 ... 8.690e-02 7.017e-02 1.812e-01]
 [1.969e+01 2.125e+01 1.300e+02 ... 1.974e-01 1.279e-01 2.069e-01]
 ...
 [1.660e+01 2.808e+01 1.083e+02 ... 9.251e-02 5.302e-02 1.590e-01]
 [2.060e+01 2.933e+01 1.401e+02 ... 3.514e-01 1.520e-01 2.397e-01]
 [7.760e+00 2.454e+01 4.792e+01 ... 0.000e+00 0.000e+00 1.587e-01]]
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 0. 0. 0. 0. 0. 1. 1. 0. 1. 1. 0. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 0. 1. 0.
 1. 1. 0. 1. 0. 1. 1. 0. 0. 0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 1. 0. 0. 1. 1.
 0. 0. 0. 1. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1.
 0. 1. 1. 0. 0. 0. 1. 1. 0. 1. 0. 1. 1. 0. 1. 1. 0. 0. 1. 0. 0. 1. 0. 0.
 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 1.
 1. 0. 0. 1. 1. 0. 0. 0. 0. 1. 0. 0. 1. 1. 1. 0. 1. 0. 1. 0. 0. 0. 1. 0.
 0. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 0. 1. 0. 0. 1. 0. 1. 1. 1. 1.
 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0. 1. 1. 0. 1.
 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 1. 0. 1. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 1.
 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 1. 1. 0. 1. 1. 1. 0. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
 1. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.
 1. 1. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0. 1.
 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 0.]
```

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y ,
                                                    random_state=104,
                                                    test_size=0.25,
                                                    shuffle=True)
```

In [ ]:

```
# x_train, x_test = x_train / 255., x_test / 255.
X_train=tf.keras.utils.normalize(X_train, axis=-1, order=2)
X_test=tf.keras.utils.normalize(X_test, axis=-1, order=2)
print(X_train)
print(X_test)
```

```
[[2.61529582e-02 2.63015545e-02 1.67383178e-01 ... 8.46362372e-05
  7.85437867e-05 4.15857509e-04]
 [2.94403288e-02 3.33261284e-02 1.86572349e-01 ... 3.33531131e-05
  3.69420807e-05 3.91008583e-04]
 [3.18381269e-02 6.13886719e-02 2.02842712e-01 ... 1.57467114e-05
  3.49718009e-05 5.61240998e-04]
 ...
 [2.96343386e-02 2.66434656e-02 1.88699395e-01 ... 6.94486213e-05
  4.65917657e-05 3.78935386e-04]
 [2.77436915e-02 4.45105311e-02 1.76787627e-01 ... 6.36416157e-05
  4.99145197e-05 4.42451567e-04]
 [2.50847504e-02 3.23667866e-02 1.59714095e-01 ... 7.17015595e-05
  5.62149648e-05 3.12087270e-04]]
[[1.39825003e-02 1.11100014e-02 9.07477796e-02 ... 7.52259805e-05
  5.45195142e-05 1.10584768e-04]
 [2.28115566e-02 3.92305915e-02 1.51240125e-01 ... 2.57352680e-04
  1.51570487e-04 3.71823417e-04]
 [2.12454202e-02 2.39496349e-02 1.43115363e-01 ... 3.39482955e-04
  1.72237676e-04 3.32549071e-04]
 ...
 [2.54446993e-02 3.33275669e-02 1.64662129e-01 ... 7.74317124e-05
  5.97700976e-05 4.23080490e-04]
 [2.54669830e-02 3.86668285e-02 1.66103125e-01 ... 2.08845482e-04
  8.90330593e-05 3.10835072e-04]
 [2.78670784e-02 3.61159271e-02 1.78692802e-01 ... 1.04695065e-04
  7.08530144e-05 4.55500943e-04]]
```

In [ ]:

```python
    # Create TF Model.
class NeuralNet(Model):
        # Set layers.
        def __init__(self):
            super(NeuralNet, self).__init__()
            # First fully-connected hidden layer.
            self.fc1 = layers.Dense(n_hidden_1, activation=tf.nn.relu)
            # First fully-connected hidden layer.
            self.fc2 = layers.Dense(n_hidden_2, activation=tf.nn.relu)
            # Second fully-connecter hidden layer.
            self.out = layers.Dense(num_classes)

        # Set forward pass.
        def call(self, x, is_training=False):
            x = self.fc1(x)
            x = self.fc2(x)
            x = self.out(x)
            if not is_training:
                # tf cross entropy expect logits without softmax, so only
                # apply softmax when not training.
                x = tf.nn.softmax(x)
            return x

    # Build neural network model.
neural_net = NeuralNet()
```

In [ ]:

```python
# Cross-Entropy Loss.
# Note that this will apply 'softmax' to the logits.
def cross_entropy_loss(x, y):

    # Convert labels to int 64 for tf cross-entropy function.
    y = tf.cast(y, tf.int64)
```

```
    # Apply softmax to logits and compute cross-entropy.
    loss = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y, logit
    # Average loss across the batch.
    return tf.reduce_mean(loss)

# Accuracy metric.
def accuracy(y_pred, y_true):
    # Predicted class is the index of highest score in prediction vector
    correct_prediction = tf.equal(tf.argmax(y_pred, 1), tf.cast(y_true, t
    return tf.reduce_mean(tf.cast(correct_prediction, tf.float32), axis=-

# Stochastic gradient descent optimizer.
# optimizer = tf.optimizers.SGD(learning_rate)
optimizer = tf.optimizers.Adam(learning_rate)
#optimizer = tf.optimizers.RMSprop(learning_rate)
#optimizer = tf.optimizers.Adagrad(learning_rate)
```

In [ ]:
```
# Optimization process.
def run_optimization(x, y):
    # Wrap computation inside a GradientTape for automatic differentiatio
    with tf.GradientTape() as g:
        # Forward pass.
        pred = neural_net(x, is_training=True)
        # Compute loss.
        loss = cross_entropy_loss(pred, y)

    # Variables to update, i.e. trainable variables.
    trainable_variables = neural_net.trainable_variables

    # Compute gradients.
    gradients = g.gradient(loss, trainable_variables)

    # Update W and b following gradients.
    optimizer.apply_gradients(zip(gradients, trainable_variables))
```

In [ ]:
```
for step in range(training_steps):
        run_optimization(X_train, y_train)


        if(step%display_step==0):
            pred = neural_net(X_train, is_training=True)
            loss = cross_entropy_loss(pred, y_train)
            acc = accuracy(pred, y_train)
            print("loss: %f, accuracy: %f" % ( loss, acc))
```

```
loss: 0.657261, accuracy: 0.633803
loss: 0.249210, accuracy: 0.896714
loss: 0.168855, accuracy: 0.929577
loss: 0.147256, accuracy: 0.938967
loss: 0.146559, accuracy: 0.931925
loss: 0.135507, accuracy: 0.953052
loss: 0.131888, accuracy: 0.946009
loss: 0.129692, accuracy: 0.950704
loss: 0.129194, accuracy: 0.950704
loss: 0.132066, accuracy: 0.943662
```

In [ ]:
```
# Test model on validation set.
pred = neural_net(X_test, is_training=False)
print("Test Accuracy: %f" % accuracy(pred, y_test))
```

```
Test Accuracy: 0.937063
```

In [ ]:

```

```