7.Ocean Proximity. Deep ANN Final Test

1 Ocean Proximity with deep artificial neural networks: Final test

Let's suppose that a full-connected 9-150-75-25-10-4 deep neural network with the hyperparameters described in section 1.2 Hyperparameters is the best model we can find to solve the Ocean Proximity classification problem. Finally, we must check this model aginst the final test set.

In [1]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from tqdm import tqdm
```

In [2]:
```
%run 1.ReadingData.py
```

```
Name of the label file: OceanProximityOneHotEncodedClasses.csv
x_train: (16342, 9)
t_train: (16342, 4)
x_dev: (2043, 9)
t_dev: (2043, 4)
x_test: (2043, 9)
t_test: (2043, 4)
```

1.1 Initialization

In [3]:
```
INPUTS = x_train.shape[1]
OUTPUTS = t_train.shape[1]
NUM_TRAINING_EXAMPLES = int(round(x_train.shape[0]/1))
NUM_DEV_EXAMPLES = int (round (x_dev.shape[0]/1))
NUM_TEST_EXAMPLES = int (round (x_test.shape[0]/1))
```

Some data is displayed to test correctness:

In [4]:
```
INPUTS
```

Out[4]: 9

In [5]:
```
OUTPUTS
```

```
Out[5]: 4

In [6]: NUM_TRAINING_EXAMPLES

Out[6]: 16342

In [7]: NUM_DEV_EXAMPLES

Out[7]: 2043

In [8]: x_train[:5]

Out[8]: array([[ 0.42031873, -0.66206164, -0.64705882, -0.69739051, -0.58752328,
                -0.82056672, -0.61914159, -0.69639039, -0.60742018],
               [ 0.43027888, -0.98087141, -0.01960784, -0.91784933, -0.91371819,
                -0.84629614, -0.91810557, -0.58127474, -0.78350192],
               [ 0.26294821, -0.72582359, -0.1372549 , -0.94485986, -0.91713222,
                -0.95392248, -0.91810557, -0.72952097, -0.15628802],
               [-0.44621514, -0.05632306, -0.49019608, -0.73401495, -0.74674115,
                -0.85251829, -0.73754317, -0.3834154 ,  0.09195838],
               [-0.39243028,  0.16471838, -0.41176471, -0.86189532, -0.80757294,
                -0.81277502, -0.78885052, -0.7176039 , -0.62350258]])

In [9]: t_train[:5]

Out[9]: array([[0., 1., 0., 0.],
               [0., 0., 0., 1.],
               [1., 0., 0., 0.],
               [0., 1., 0., 0.],
               [0., 1., 0., 0.]])

In [10]: x_dev[:5]

Out[10]: array([[-0.07171315, -0.10733262, -0.1372549 , -0.89343303, -0.88081937,
                 -0.94910171, -0.86712712, -0.58443332, -0.56041006],
                [-0.4123506 , -0.18384697,  0.49019608, -0.88371738, -0.83612663,
                 -0.91894392, -0.86548265, -0.60979849, -0.27587515],
                [-0.61952191,  0.11583422,  1.        , -0.9123048 , -0.88112973,
                 -0.96575016, -0.88324289, -0.56120605,  0.99999588],
                [ 0.45418327, -0.9957492 , -0.17647059, -0.88961799, -0.82557418,
                 -0.88531069, -0.82798882, -0.79089944, -0.48742067],
                [ 0.15338645, -0.64930925,  0.33333333, -0.96032352, -0.95561763,
                 -0.97634463, -0.95428383, -0.31657494, -0.23133925]])

In [11]: t_dev[:5]

Out[11]: array([[0., 1., 0., 0.],
                [1., 0., 0., 0.],
                [0., 0., 1., 0.],
                [0., 0., 0., 1.],
                [1., 0., 0., 0.]])
```

1.2 Hyperparameters

The number of hidden layers and neurons per layer must be adjusted.

```
In [12]: n_epochs = 20000
         learning_rate = 0.1
         batch_size = 200
         n_neurons_per_layer = [150,75,25,10]
```

1.3 Build the model: a 9-150-75-25-10-4 deep neural network architecture

```
In [13]: X = tf.placeholder (dtype=tf.float32, shape=(None,INPUTS),name="X")
         t = tf.placeholder (dtype=tf.float32, shape=(None,OUTPUTS), name="t")
```

The deep neural network topology is defined: a full-connected 8-100-50-25-10-10 architecture. The ReLU activation function is chosen for the hidden layers and linear logits with softmax for the ouput layer.

```
In [14]: hidden_layers = []
         hidden_layers.append(tf.layers.dense (X, n_neurons_per_layer[0],
                                              activation = tf.nn.relu))
         for layer in range(1,len(n_neurons_per_layer)):
             hidden_layers.append(tf.layers.dense (hidden_layers[layer-1],
                          n_neurons_per_layer[layer], activation = tf.nn.relu))
         net_out = tf.layers.dense (hidden_layers[len(n_neurons_per_layer)-1], OUTPUTS)
         y = tf.nn.softmax (logits=net_out, name="y")
```

```
In [15]: for layer in range(len(n_neurons_per_layer)): print (hidden_layers[layer])
```

```
Tensor("dense/Relu:0", shape=(?, 150), dtype=float32)
Tensor("dense_1/Relu:0", shape=(?, 75), dtype=float32)
Tensor("dense_2/Relu:0", shape=(?, 25), dtype=float32)
Tensor("dense_3/Relu:0", shape=(?, 10), dtype=float32)
```

The $log - loss$, $cross - entropy$ (the sun of log-loss is a loss) and and *cost* (the mean of cross-entropy) functions:

```
In [16]: cross_entropy = tf.nn.softmax_cross_entropy_with_logits_v2 (labels=t, logits=net_out)
         mean_log_loss = tf.reduce_mean (cross_entropy, name="cost")
```

The training algorithm: gradient descent method with a softmax function at the outputs:

```
In [17]: train_step = tf.train.GradientDescentOptimizer (learning_rate).minimize(mean_log_loss)
```

Model evaluation: accuracy. The percentage of correctly classified instances.

```
In [18]: correct_predictions = tf.equal(tf.argmax(y,1),tf.argmax(t,1))
         accuracy = tf.reduce_mean(tf.cast(correct_predictions,tf.float32))
```

1.4 Execute the 9-150-75-25-10-4 deep neural network with M-BGD

In [19]:
```
init = tf.global_variables_initializer()
accuracy_train_history = []
with tf.Session() as sess:
    sess.run(init)
    for epoch in tqdm(range(n_epochs)):
        offset = (epoch * batch_size) % (NUM_TRAINING_EXAMPLES - batch_size)
        sess.run (train_step, feed_dict={X: x_train[offset:(offset+batch_size)],
                                         t: t_train[offset:(offset+batch_size)]})

    accuracy_test = accuracy.eval(feed_dict={X: x_test[:NUM_TEST_EXAMPLES],
                                              t: t_test[:NUM_TEST_EXAMPLES]})
    test_predictions = y.eval(feed_dict={X: x_test[:NUM_TEST_EXAMPLES]})

    test_correct_preditions = correct_predictions.eval (feed_dict=
                                         {X: x_test[:NUM_TEST_EXAMPLES],
                                          t: t_test[:NUM_TEST_EXAMPLES]})
```

```
100%|| 20000/20000 [03:21<00:00, 99.50it/s]
```

In [20]:
```
"Accuracy for the TEST set: " + str(accuracy_test)
```

Out[20]:
```
'Accuracy for the TEST set: 0.9388155'
```

In [21]:
```
test_predictions
```

Out[21]:
```
array([[8.1986576e-01, 6.4289913e-04, 3.3940219e-06, 1.7948791e-01],
       [9.9886191e-01, 1.0785614e-03, 4.7880158e-06, 5.4819564e-05],
       [1.3193237e-02, 7.9329489e-05, 9.8672664e-01, 7.3480385e-07],
       ...,
       [9.9853885e-01, 4.4980676e-05, 5.8038779e-08, 1.4161899e-03],
       [9.9814260e-01, 9.1308633e-05, 6.3880510e-04, 1.1272870e-03],
       [1.0934104e-01, 8.9064920e-01, 9.0073836e-06, 7.2069679e-07]],
      dtype=float32)
```

In [22]:
```
test_rounded_predictions=np.round(test_predictions)
indices = np.argmax(test_predictions,1)
for row, index in zip(test_rounded_predictions, indices): row[index]=1
test_rounded_predictions[:10]
```

Out[22]:
```
array([[1., 0., 0., 0.],
       [1., 0., 0., 0.],
       [0., 0., 1., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [1., 0., 0., 0.],
```

```
                [1., 0., 0., 0.],
                [1., 0., 0., 0.],
                [0., 0., 0., 1.],
                [0., 0., 0., 1.]], dtype=float32)
```

```
In [23]: t_test[:10] #target classes
```

```
Out[23]: array([[1., 0., 0., 0.],
                [1., 0., 0., 0.],
                [0., 0., 1., 0.],
                [0., 1., 0., 0.],
                [0., 0., 1., 0.],
                [1., 0., 0., 0.],
                [1., 0., 0., 0.],
                [1., 0., 0., 0.],
                [0., 0., 0., 1.],
                [1., 0., 0., 0.]])
```

```
In [24]: test_correct_preditions[:10]
```

```
Out[24]: array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
                False])
```

Since this is the final neural model, the accuracy is calculated against the final test set, achieving a 94% of accuracy.