5.Ocean Proximity. One Hidden Layer

1 Ocean Proximity with one-hidden layer neural networks

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))
```

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

A new hiperparameter has to be adjuted: the number of neurons in the hidden layer

```
In [12]: n_epochs = 10000
         learning_rate = 0.1
         batch_size = 200
         n_hidden = 1024
```

1.3 Build the model: a full-connected 9-1024-4 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")
```

A classical neural network topology is defined: X is the input, 200 neurons in the hidden layer with tanh activation function, and 10 outputs with the softmax activation function.

```
In [14]: hidden = tf.layers.dense (X, n_hidden, activation = tf.nn.tanh, name="hidden")
         net_out = tf.layers.dense (hidden, OUTPUTS, name="net_out")
         y = tf.nn.softmax (logits=net_out, name="y")
```

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

```
In [15]: 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 [16]: train_step = tf.train.GradientDescentOptimizer (learning_rate).minimize(mean_log_loss)
```

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

```
In [17]: 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 model MBDG

```
In [18]: 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_train_history.append (accuracy.eval(feed_dict=
                                         {X: x_train[offset:(offset+batch_size)],
                                          t: t_train[offset:(offset+batch_size)]}))
             accuracy_train = accuracy.eval(feed_dict={X: x_train[:NUM_TRAINING_EXAMPLES],
                                                       t: t_train[:NUM_TRAINING_EXAMPLES]})
             accuracy_dev = accuracy.eval(feed_dict={X: x_dev[:NUM_DEV_EXAMPLES],
                                                     t: t_dev[:NUM_DEV_EXAMPLES]})
```

```
            predictions = y.eval(feed_dict={X: x_dev[:NUM_DEV_EXAMPLES]})
            final_correct_preditions = correct_predictions.eval (feed_dict=
                                            {X: x_dev[:NUM_DEV_EXAMPLES],
                                             t: t_dev[:NUM_DEV_EXAMPLES]})
            final_train_mean_log_loss = mean_log_loss.eval (feed_dict=
                                                    {X: x_train[:NUM_TRAINING_EXAMPLES],
                                                      t: t_train[:NUM_TRAINING_EXAMPLES]})
            final_dev_mean_log_loss = mean_log_loss.eval (feed_dict=
                                                    {X: x_dev[:NUM_DEV_EXAMPLES],
                                                      t: t_dev[:NUM_DEV_EXAMPLES]})
```

```
100%|| 10000/10000 [02:56<00:00, 57.48it/s]
```

In [19]:
```
"Final accuracy in training: " + str(accuracy_train)
```

Out[19]:
```
'Final accuracy in training: 0.86690736'
```

In [20]:
```
"Maximum accuracy in training: " + str(np.max(accuracy_train_history))
```

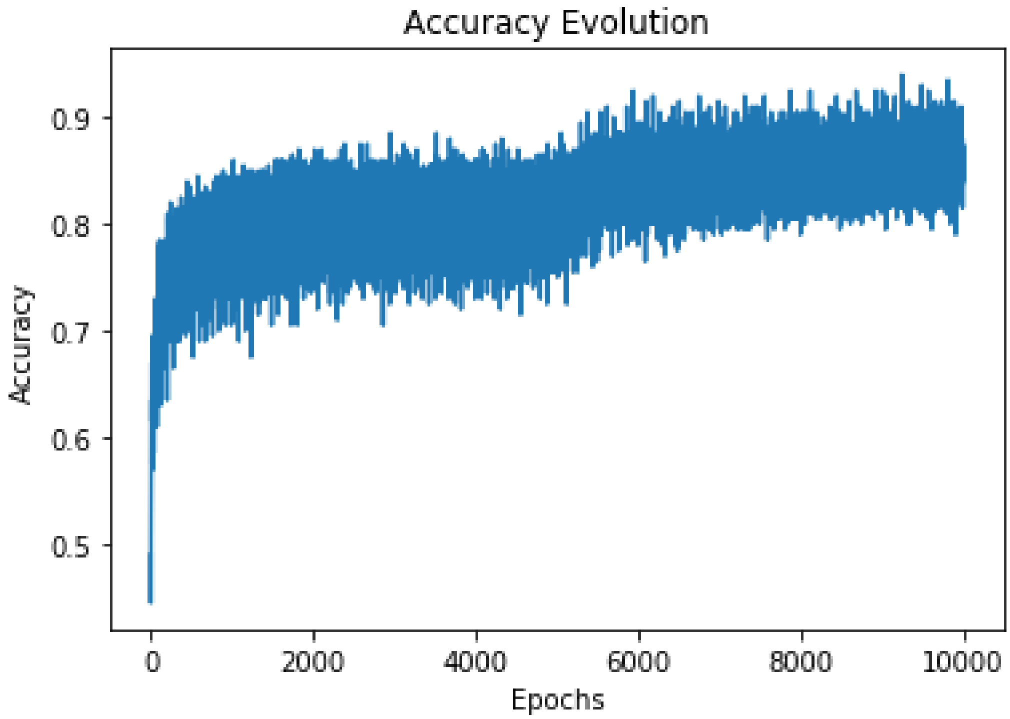
Out[20]:
```
'Maximum accuracy in training: 0.94'
```

In [21]:
```
"Accuracy for the development set: " + str(accuracy_dev)
```

Out[21]:
```
'Accuracy for the development set: 0.86882037'
```

In [22]:
```
plt.title ("Accuracy Evolution")
plt.xlabel ("Epochs")
plt.ylabel ("Accuracy")
plt.plot (range(n_epochs),accuracy_train_history)
```

Out[22]:
```
[<matplotlib.lines.Line2D at 0xb28ebe7f0>]
```



Note that accuracy *zig-zags* due to mini-batch *zig-zagging* errors, yet the trend is positive.

In [23]:
```
predictions
```

Out[23]:
```
array([[2.8358586e-06, 9.9999607e-01, 1.0803618e-06, 1.3063440e-12],
       [4.7512901e-01, 1.7212782e-04, 1.1902060e-01, 4.0567827e-01],
       [2.9780855e-02, 2.0806327e-04, 6.3425231e-01, 3.3575881e-01],
       ...,
       [6.2122315e-01, 4.9600946e-03, 1.8369439e-01, 1.9012237e-01],
       [4.3160281e-01, 2.6565066e-02, 5.3385204e-01, 7.9800533e-03],
       [9.3746078e-01, 4.3941315e-02, 3.4436624e-04, 1.8253641e-02]],
      dtype=float32)
```

In [24]:
```
rounded_predictions=np.round(predictions)
indices = np.argmax(predictions,1)
for row, index in zip(rounded_predictions, indices): row[index]=1
rounded_predictions[:10]
```

Out[24]:
```
array([[0., 1., 0., 0.],
       [1., 0., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.],
       [1., 0., 0., 0.],
       [1., 0., 0., 0.],
       [1., 0., 0., 0.],
```

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

```
In [25]: t_dev[:10]
```

```
Out[25]: array([[0., 1., 0., 0.],
                [1., 0., 0., 0.],
                [0., 0., 1., 0.],
                [0., 0., 0., 1.],
                [1., 0., 0., 0.],
                [1., 0., 0., 0.],
                [1., 0., 0., 0.],
                [1., 0., 0., 0.],
                [0., 1., 0., 0.],
                [0., 1., 0., 0.]])
```

```
In [26]: final_correct_preditions[:10]
```

```
Out[26]: array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
                 True])
```

```
In [27]: final_train_mean_log_loss
```

```
Out[27]: 0.33502594
```

```
In [28]: final_dev_mean_log_loss
```

```
Out[28]: 0.33095166
```

Train accuracy is now 87% and dev accuracy has raised to 87%. This means that the neural model is not overfitted. Time spent is about 3 minutes, even though mini-batch gradient descent has been applied. Back propagation makes intensive vectorial computation. Thus, the need of a GPU.