ΜΥΕ035- Υπολογιστική Νοημοσύνη
Διδάσκων: Αριστείδης Λύκας

Αναφορά 1ης Εργαστηριακής Άσκησης: Multilayer Perceptron

Ομάδα:
Κονδυλία Βέργου 4325
Παναγιώτης Βουζαλής 2653

Χειμερινό Εξάμηνο 2021

Περιεχόμενα

Δημιουργία MLP

Ο κώδικας για την δημιουργία του dataset μας βρίσκεται στο αρχείο `/datasetgenerator/DatasetGenerator.java`

Η δημιουργία του MLP γίνεται με constructor στο `/main/MLP.java` αξιοποιούνται τα εξής πεδία:

```
    private static int D = 2;           // number of inputs (in H1) - our
inputs are x1, x2
    private static int K = 4;           // number of categories
    private static int H1;          // number of neurons in H1
    private static int H2;          // number of neurons in H2
    private static int H3;          // number of neurons in H3

    private static double trainingThreshold = 0.01; // difference between
two training errors
    private static double worthiness = 75; // generalization ability
at  which the MLP is deemed worthy

    // tanh or relu
    private static TransferFunction transferFunction; //
HyperbolicFunction() or ReluFunction()

    private static int B;               // batch size - 1 is linear
update, N = 4000 in our case is team update
    private static int numHiddenLayers; // number of hidden layers in
network
    private static double learningRate;
```

Όλοι οι νευρώνες των κρυμμένων επιπέδων έχουν συνάρτηση ενεργοποίησης την `transferFunction`, η οποία είναι είτε η υπερβολική εφαπτομένη είτε η relu.

Όλοι οι νευρώνες του επιπέδου εξόδου έχουν συνάρτηση ενεργοποίησης την σιγμοειδής συνάρτηση.

Παρακάτω δίνεται ο constructor για την δημιουργία MLP με 2 κρυμμένα επίπεδα.
Αντίστοιχος είναι ο κώδικας για την δημιουργία με 3 κρυμμένα επίπεδα

```
    public MLP(DatasetGenerator datasets, int hiddenLayers, int h1neurons,
int h2neurons, String activationFunc, Double learnRate, int batchSize) {

        trainingSet = datasets.getTrainingSet();
        testSet = datasets.getTestSet();

        numHiddenLayers = hiddenLayers;
        H1 = h1neurons;
        H2 = h2neurons;
        if (activationFunc.equals("tanh")) { transferFunction = new
HyperbolicFunction(); }
        if (activationFunc.equals("relu")) { transferFunction = new
ReluFunction(); }
        learningRate = learnRate;
        B = batchSize;

        printMLPinfo(hiddenLayers, h1neurons, h2neurons, activationFunc,
learnRate, batchSize);

        initializeHiddenLayers(hiddenLayers);
    }
```

Εκπαίδευση MLP με gradient descent

Η εκπαίδευση του MLP γίνεται m την συνάρτηση `train` που βρίσκεται στο `/main/MLP.java`

Εφαρμόζεται στο trainingSet ο αλγόριθμος gradient descent με mini-batches.
Κάθε batch έχει μέγεθος B, επομένως για:

- B = 1 έχουμε σειριακή ενημέρωση
- B = N έχουμε ομαδική ενημέρωση
- 1<B<N έχουμε mini-batch ενημέρωση

Στο τέλος κάθε εποχής τυπώνεται ο αριθμός της εποχής και το σφάλμα εκπαίδευσης εκείνης της εποχής.

Η εκπαίδευση τελειώνει, όταν η διαφορά του σφάλματος εκπαίδευσης δυο συνεχόμενων εποχών είναι μικρότερη από το κατώφλι 0.01 και ο αλγόριθμος έχει ήδη τρέξει για 700 εποχές

```
public void train() {

        int epoch = 0;
        while(true) {

            currentEpochError = 0;
            int numOfBatches = 0;

            for (int i = 0; i < trainingSet.size(); i += B) {

                numOfBatches ++;
                for(int j = i; j < numOfBatches*B; j++) {

                    Point point = trainingSet.get(i);
                    backPropagate(point.getCoordinates(),
point.createCategoryVector());
                }

                //update weights
                for(Neuron [] layer : layers) {

                    for(Neuron neuron : layer) {

                        neuron.updateWeights(learningRate);
                        neuron.initDelta();
                    }
                }
            }
```

```
double diffEpochError = calculateDiffEpochError(epoch, currentEpochError);

            if (diffEpochError < trainingThreshold && epoch > 700) {

                //System.out.println(".........");
                System.out.printf("Error of epoch %d : %.4f\n", epoch,
currentEpochError);
                break;
            }

            epoch++;
        }
    }
```

Testing MLP

Το testing του MLP γίνεται με την συνάρτηση `test` που βρίσκεται στο `/main/MLP.java`

Εφαρμόζεται στο testingSet η forwardPass και στη συνέχεια ελέγχεται, αν το MLP έβγαλε το σωστό αποτέλεσμα.

```
public void test() {

    for (Point point: testSet) {

        double [] netOutput = forwardPass(point.getCoordinates());
        checkCategory(netOutput, point);
    }
}
```

Forward Pass

Η `forwardPass` συνάρτηση υλοποιείται στο `/main/MLP.java`

Ως παράμετρο περνάμε την είσοδο που εφαρμόζεται και επιστρέφει την έξοδο που υπολογίζει το MLP.

Η `activate` συνάρτηση βρίσκεται στο `/neurons/Neuron.java` και υλοποιεί τον τύπο:

$$o(x) = g(u) \text{ όπου}$$
$$u(x) = \sum_{i=1}^{d} w_1 x_i + \omega_0$$

```
public double[] forwardPass(double inputs[]) {

   double [][] temp_inputs = initTempInputs(inputs);
   int layerNum = 0;

   for (Neuron[] neuronsList: layers) {

      layerNum++;
      int i = 0;
      for (Neuron neuron : neuronsList) {

         neuron.setInputs(temp_inputs[layerNum-1]);      // neuron's input
is set previous levels output
         temp_inputs[layerNum][i] = neuron.activate();   // activate i-th
neuron of layerNum-th layer

         i++;
      }
   }

   return temp_inputs[layerNum];   // last level's output
}
```

Back Propagation

Η `backPropagate` συνάρτηση υλοποιείται στο `/main/MLP.java`

Ως παραμέτρους περνάμε την είσοδο που εφαρμόζεται και την έξοδο που αναμένουμε από το MLP.

Η `calculateDelta` συνάρτηση βρίσκεται στο `/neurons/Neuron.java` και υπολογίζει τις μερικές παραγώγους με τους τύπους:

$$\frac{dE^n}{dw_{ij}^{(h)}} = \delta_i^{(h)} * y_j^{(h-1)} \quad \omega\varsigma\ \pi\rho o\varsigma\ \tau\alpha\ \beta\acute{\alpha}\rho\eta$$

$$\frac{dE^n}{dw_{i0}^{(h)}} = \delta_i^{(h)} \quad \omega\varsigma\ \pi\rho o\varsigma\ \tau\eta\nu\ \pi\acute{o}\lambda\omega\sigma\eta$$

Ανάλογα με το ποιο επίπεδο βρίσκεται ο νευρώνας, το σφάλμα δ υπολογίζεται διαφορετικά

```java
public void backPropagate(double inputs[], double expected[]) {

   double output [] = forwardPass(inputs);
   double diff   [] = calculateDiff(output,expected);
   currentEpochError += calculateEpochError(diff);

   int numLastLayer = layers.size()-1;

   for (int i = numLastLayer; i > -1; i--) {

      Neuron [] layer = layers.get(i);

      int j = 0;  // count neurons in layer
      for (Neuron neuron : layer) {

      // output Layer
      if (i == numLastLayer) {

         neuron.calculateDelta(output[j], diff[j]);
```

Για τους νευρώνες του επιπέδου εξόδου βρίσκουμε το σφάλμα, εφαρμόζοντας τον τύπο

$$\delta_i^{(H+1)} = g'_{H+1}(u_i^{(H+1)}) * (o_i - t_{ni})$$

```
        // hidden Layer
        } else {

            Neuron [] prevLayer = layers.get(i+1);
            double sum = 0;

            for (Neuron prevLayerNeuron: prevLayer) {

               double error = prevLayerNeuron.getError();
               double weight = prevLayerNeuron.getWeight(j);
               sum += weight * error;

            }

            neuron.calculateDelta(neuron.activate(), sum);

            j++;
            }
        }
}
```

Για τους νευρώνες των κρυμμένων επιπέδων βρίσκουμε το σφάλμα, εφαρμόζοντας τον τύπο

$$\delta_i^{(h)} = g_h'(u_i^{(h)}) * \sum_{j=1}^{d_{h+1}} w_{ji}^{(h+1)} \ \delta_j^{(h+1)}$$

Neural Networks Project 2022 - Question 1 Plot Report

Kondylia Vergou, AM 4325

Panagiotis Vouzalis, AM 2653

Η συγγραφη του κωδικα εγινε σε περιβαλλον Windows

- Για να λειτουργησουν οι εντολες που αναφερονται παρακατω σε περιβαλλον Linux πρεπει να αντικατασταθει το < ; > με < : >
- Τα dependencies της εφαρμογης μας βρισκονται στο φακελο `/jars`
- Αναλογα με τη διαρυθμιση της πλατφορμας που θα τρεξει η εφαρμογη, μπορει να χρειαστει να προστεθει στο τελος του αρχειου προς εκτελεση το επιθεμα `.java` αλλιως πιθανως θα εμφανιζεται ενα σφαλμα τυπου `error: class not found on application class path: <.java file name>`

Instructions for Windows Platforms

1. Open terminal inside the directory Neural Networks - Question 1
2. Compile with: `javac -cp ".;jars/*" main/*.java`
3. Create a new dataset with: `java -cp ".;jars/*" main/DatasetGenerator` - This is not necessary
4. (*2 hidden layers*) Run java application with: `java -cp ".;jars/*" main/MLPDriver <hidden layers> <h1 neurons> <h2 neurons> <transfer function> <learning rate> <batch size>`
   - example: `java -cp ".;jars/*" main/MLPDriver 2 4 3 tanh 0.05 1`
1. (*3 hidden layers*) Run java application with: `java -cp ".;jars/*" main/MLPDriver <hidden layers> <h1 neurons> <h2 neurons> <h3 neurons> <transfer function> <learning rate> <batch size>`
   - example: `java -cp ".;jars/*" main/MLPDriver 3 4 3 2 tanh 0.05 1`

Instructions for Linux Platforms

1. Open terminal inside the directory Neural Networks - Question 1
2. Compile with: `javac -cp ".:jars/*" main/*.java datasetgenerator/*.java neurons/*.java transferfunctions/*.java`
3. Create a new dataset with: `java -cp ".:jars/*" main/DatasetGenerator` - This is not necessary
4. (*2 hidden layers*) Run java application with: `java -cp ".:jars/*" main/MLPDriver <hidden layers> <h1 neurons> <h2 neurons> <transfer function> <learning rate> <batch size>`
   - example: `java -cp ".:jars/*" main/MLPDriver 2 4 3 tanh 0.05 1`
1. (*3 hidden layers*) Run java application with: `java -cp ".:jars/*" main/MLPDriver <hidden layers> <h1 neurons> <h2 neurons> <h3 neurons> <transfer function> <learning rate> <batch size>`
   - example: `java -cp ".:jars/*" main/MLPDriver 3 4 3 2 tanh 0.05 1`

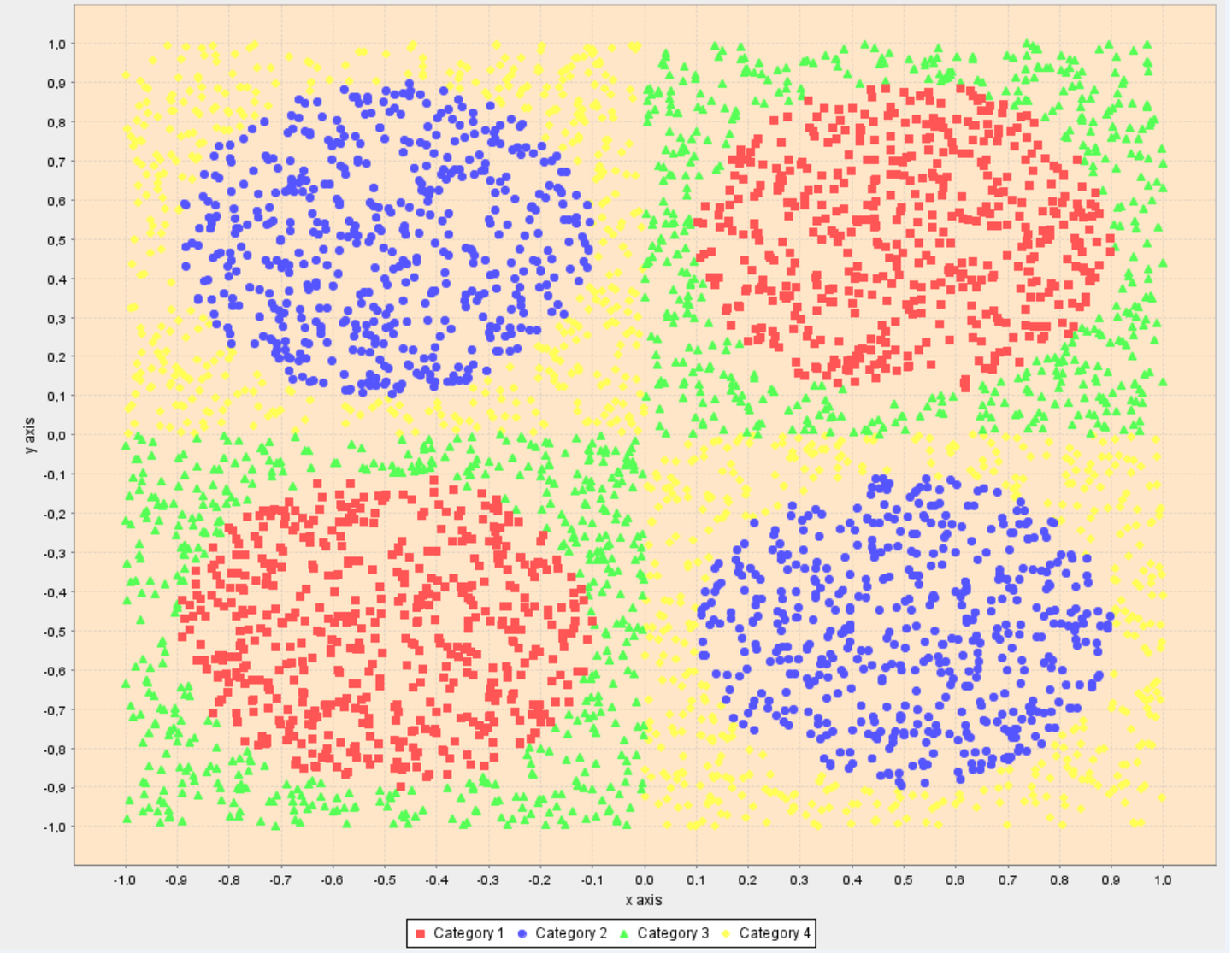
Instructions for Eclipse IDE

1. Run Configurations -> Arguments -> Program Arguments:
2. (*2 hidden layers*) `<hidden layers> <h1 neurons> <h2 neurons> <transfer function> <learning rate> <batch size>` - example: 2 4 3 tanh 0.05 1
3. (*3 hidden layers*) `<hidden layers> <h1 neurons> <h2 neurons> <h3neurons> <transfer function> <learning rate> <batch size>` - example: 3 4 3 2 tanh 0.05 1

Keep the learning rate at 0.05 (Google defaults it at 0.03)

Για τις αναγκες της δευτερης ασκησης δημιουργησαμε το dataset.txt συμφωνα με τις απαιτησεις του ερωτηματος Σ1.
Ο κωδικας για τη δημιουργια του dataset βρισκεται στο αρχειο `/datasetgenerator/DatasetGenerator.java`

MLP Test Set Categorization Plot - Expected



Μελετη προβληματος Σ1

Πως μεταβαλλεται η γενικευτικη ικανοτητα του δικτυου?

Για διαφορους συνδυασμους τιμων των H1, H2, H3?

Χρησιμοποιωντας την tanh ή την relu ως συναρτηση ενεργοποιησης?

Για batch size B=1, B=40, B=400?

Προκυπτει καποιο οφελος απο τη χρηση του H3?

(Στο excel αρχειο 'Problem S1 Study.xlsx' βρισκονται τα αποτελεσματα της ερευνας μας)

Number of h1 inputs (d): locked at 2
Number of categories (K): locked at 4
Training (error) threshold: locked at 0.01
Learning rate: locked at 0.05

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

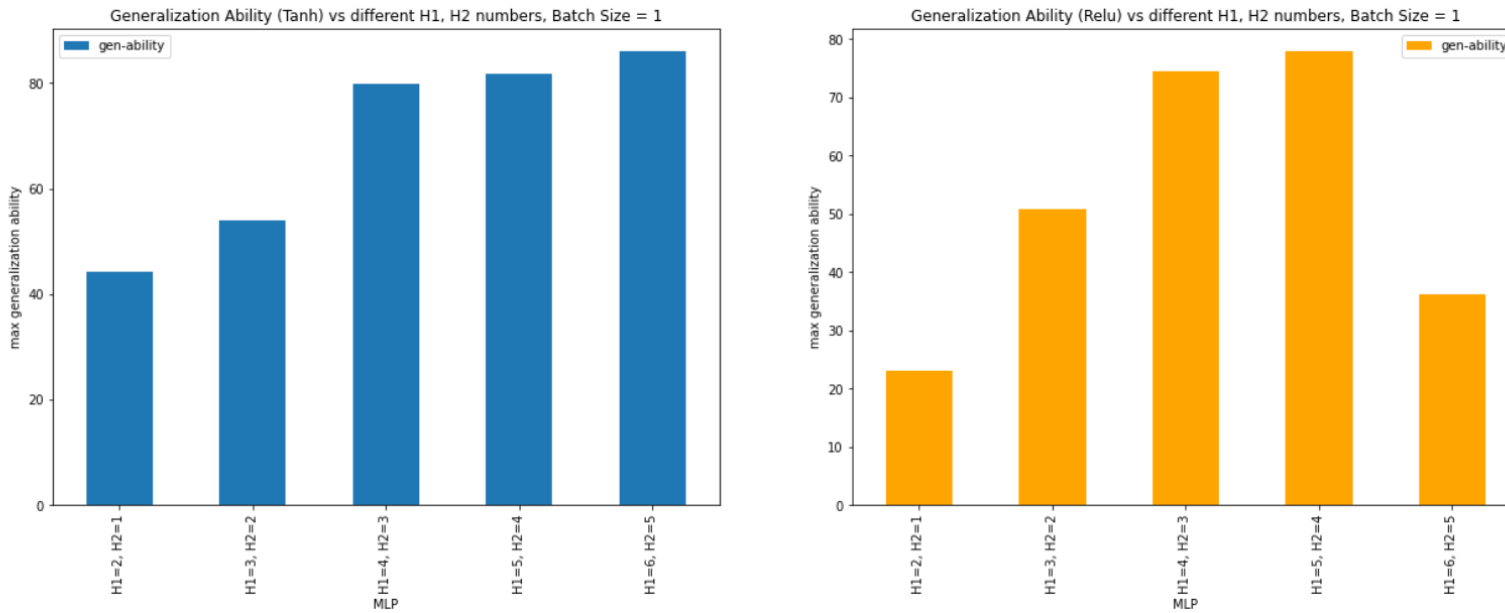
Bar plots για συνδυασμους τιμων H1, H2 για την tanh, relu με batch size = 1

In [2]:
```
fig, ax = plt.subplots(1, 2, figsize = (21, 7))

df1 = pd.DataFrame({'MLP':['H1=2, H2=1', 'H1=3, H2=2', 'H1=4, H2=3', 'H1=5, H2=4', 'H1=6, H2=5'],
                    'gen-ability':[44.12, 53.99, 79.72, 81.62, 86.02]})
```

```
ax[0] = df1.plot.bar(x = 'MLP', y = 'gen-ability', ax = ax[0])
ax[0].title.set_text('Generalization Ability (Tanh) vs different H1, H2 numbers, Batch Size = 1')
ax[0].set_ylabel('max generalization ability')

df2 = pd.DataFrame({'MLP':['H1=2, H2=1', 'H1=3, H2=2', 'H1=4, H2=3', 'H1=5, H2=4', 'H1=6, H2=5'],
                    'gen-ability':[23.08, 50.92, 74.6, 77.96, 36.17]})
df2.plot.bar(x = 'MLP', y = 'gen-ability', color = 'orange', ax = ax[1])
ax[1].title.set_text('Generalization Ability (Relu) vs different H1, H2 numbers, Batch Size = 1')
ax[1].set_ylabel('max generalization ability')
```

Out[2]:

```
Text(0, 0.5, 'max generalization ability')
```



Παρατηρησεις:

- Οσον αφορα την tanh, παρατηρουμε μια αυξηση στη γενικευτικη ικανοτητα οσο αυξανονται οι αριθμοι των νευρωνων σε καθε κρυμμενο επιπεδο, με κορυφωση στις τιμες H1 = 6, H2 = 5.
- Οσον αφορα την relu, παρατηρρουμε μια αυξηση στη γενικευτικη ικανοτητα οσο αυξανονται οι αριθμοι των νευρωνων σε καθε κρυμμενο επιπεδο, με κορυφωση στις τιμες H1 = 5, H2 = 4. Αξιοσημειωτη ειναι η ραγδαια πτωση της γενικευτικης ικανοτητας στις τιμες H1 = 6, H2 = 5.
- Εχοντας τις καλυτερες επιδοσεις των δυο συναρτησεων ενεργοποιησης κατα νου, παρατηρουμε πως οι επιδοσεις των δυο αυτων συναρτησεων ειναι περιπου οι ιδιες γυρω στο 80% γενικευτικης ικανοτητας για batch size = 1.

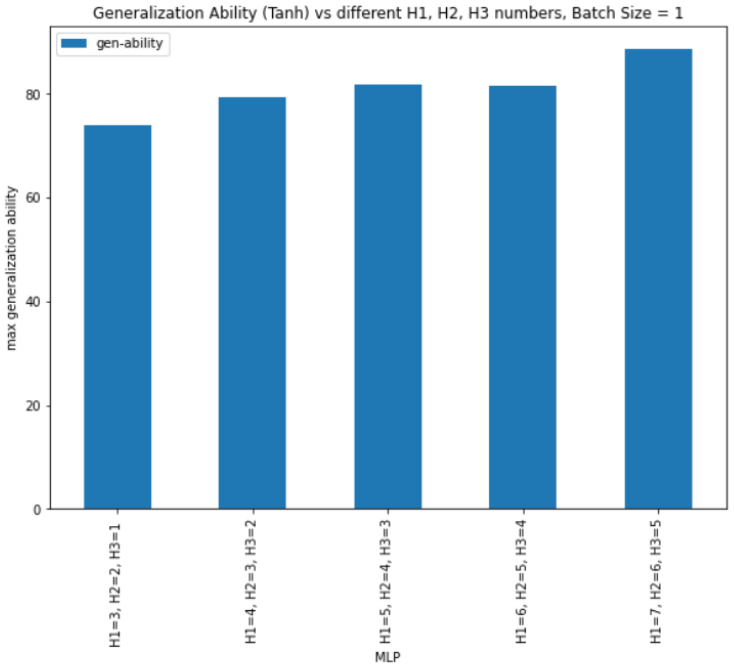
Bar plots για συνδυασμους τιμων H1, H2, H3 για την tanh, relu με batch size = 1

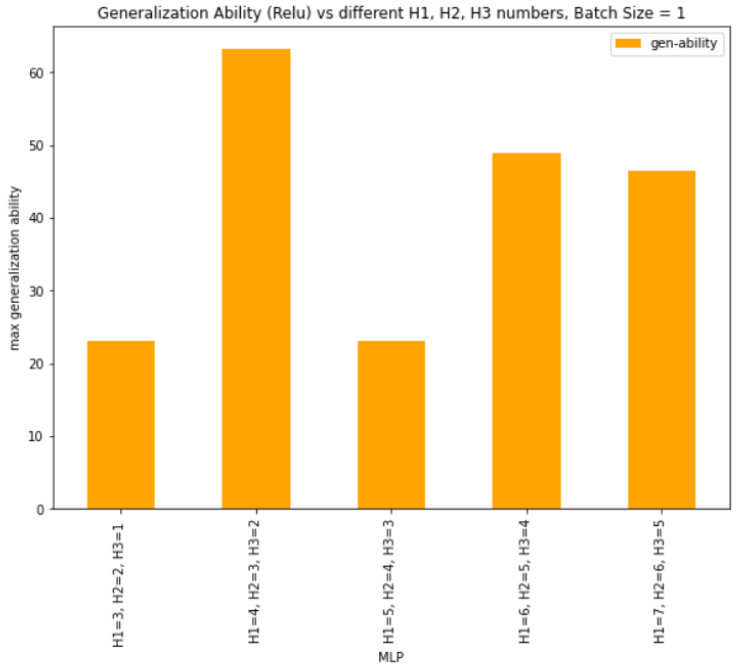
In [3]:

```
fig, ax = plt.subplots(1, 2, figsize = (21, 7))

df1 = pd.DataFrame({'MLP':['H1=3, H2=2, H3=1 ', 'H1=4, H2=3, H3=2', 'H1=5, H2=4, H3=3', 'H1=6, H2=5, H3=4', 'H1
                    'gen-ability':[73.99, 79.36, 81.86, 81.48, 88.67]})
ax[0] = df1.plot.bar(x = 'MLP', y = 'gen-ability', ax = ax[0])
ax[0].title.set_text('Generalization Ability (Tanh) vs different H1, H2, H3 numbers, Batch Size = 1')
ax[0].set_ylabel('max generalization ability')

df2 = pd.DataFrame({'MLP':['H1=3, H2=2, H3=1 ', 'H1=4, H2=3, H3=2', 'H1=5, H2=4, H3=3', 'H1=6, H2=5, H3=4', 'H1
                    'gen-ability':[23.08, 63.19, 23.08, 48.85, 46.4]})
df2.plot.bar(x = 'MLP', y = 'gen-ability', color = 'orange', ax = ax[1])
ax[1].title.set_text('Generalization Ability (Relu) vs different H1, H2, H3 numbers, Batch Size = 1')
ax[1].set_ylabel('max generalization ability')
```

Out[3]:

```
Text(0, 0.5, 'max generalization ability')
```





Παρατηρησεις:

- Οσον αφορα την tanh, η αυξηση της γενικευτικης ικανοτητας οσο αυξανονται οι νευρωνες στα κρυμμενα επιπεδα ειναι πολυ μικρη. Κορυφωση παρατηρουμε για τις τιμες H1 = 7, H2 = 6, H3 = 5.
- Οσον αφορα την relu, η κορυφαια επιδοση της ερχεται στις τιμες H1 = 4, H2 = 3, H3 = 2 και επειτα παρατηρειται μια πτωση οσο αυξανονται οι νευρωνες στα κρυμμενα επιπεδα.
- Εχοντας τις καλυτερες επιδοσεις των δυο συναρτησεων ενεργοποιησης κατα νου, παρατηρουμε πως η επιδοση της tanh ειναι πολυ καλυτερη απο αυτην της relu. (γενικευτικη ικανοτητα 80% vs 60% αντιστοιχα)

Conclusion (batch size = 1):

Για batch size = 1 και MLP 2 κρυμμενων επιπεδων παρατηρουμε πως η tanh εχει περιπου την ιδια επιδοση με τη relu, ενω για MLP 3 κρυμμενων επιπεδων παρατηρουμε πως η tanh εχει πολυ καλυτερη επιδοση απο τη relu.

Δεν προκυπτει καποιο οφελος απο τη χρηση του 3ου κρυμμενου επιπεδου καθως η γενικευτικη ικανοτητα και του 2-layer MLP και του 3-layer MLP ειναι περιπου 80%.

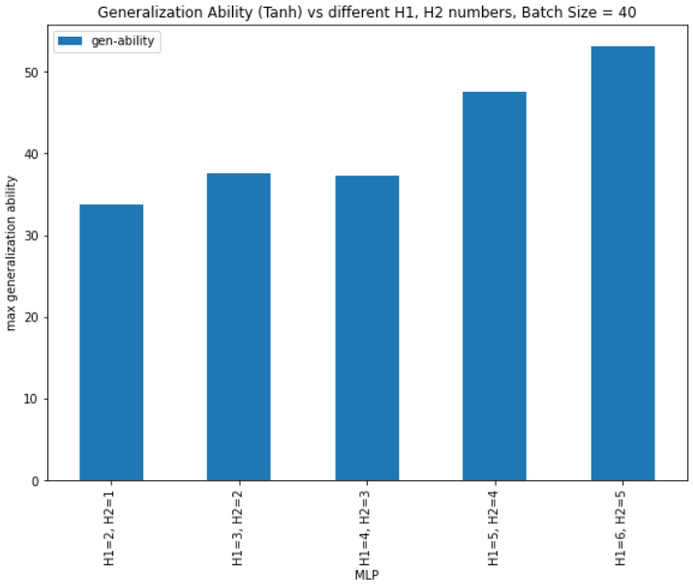
Bar plots για συνδυασμους τιμων H1, H2 για την tanh, relu με batch size = 40

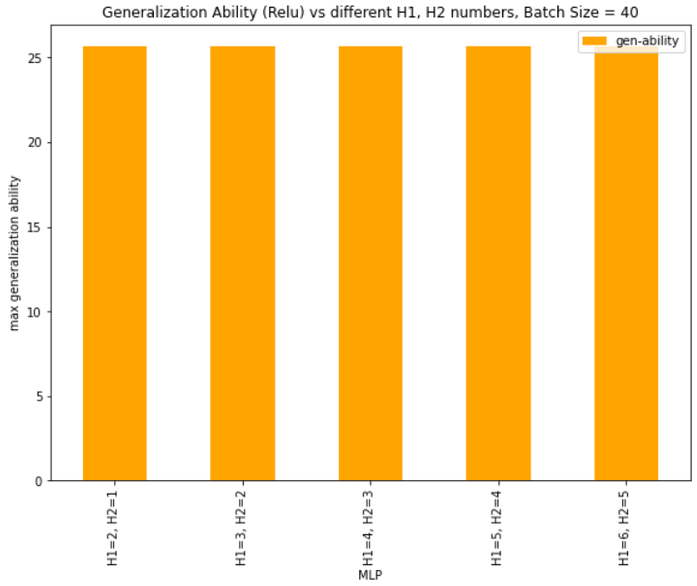
In [4]:

```
fig, ax = plt.subplots(1, 2, figsize = (21, 7))

df1 = pd.DataFrame({'MLP':['H1=2, H2=1', 'H1=3, H2=2', 'H1=4, H2=3', 'H1=5, H2=4', 'H1=6, H2=5'],
                    'gen-ability':[33.76, 37.5, 37.31, 47.46, 53.11]})
ax[0] = df1.plot.bar(x = 'MLP', y = 'gen-ability', ax = ax[0])
ax[0].title.set_text('Generalization Ability (Tanh) vs different H1, H2 numbers, Batch Size = 40')
ax[0].set_ylabel('max generalization ability')

df2 = pd.DataFrame({'MLP':['H1=2, H2=1', 'H1=3, H2=2', 'H1=4, H2=3', 'H1=5, H2=4', 'H1=6, H2=5'],
                    'gen-ability':[25.67, 25.67, 25.67, 25.67, 25.67]})
df2.plot.bar(x = 'MLP', y = 'gen-ability', color = 'orange', ax = ax[1])
ax[1].title.set_text('Generalization Ability (Relu) vs different H1, H2 numbers, Batch Size = 40')
ax[1].set_ylabel('max generalization ability')
```

Out[4]:

```
Text(0, 0.5, 'max generalization ability')
```





Παρατηρησεις:

- Οσον αφορα την tanh, παρατηρειται μια αυξηση στη γενικευτικη ικανοτητα οσο αυξανονται οι νευρωνες στα κρυμμενα επιπεδα, με κορυφωση στις τιμες H1 = 6, H2 = 5.
- Οσον αφορα τη relu, η γενικευτικη ικανοτητα παραμενει η ιδια οσο αυξανονται οι οι νευρωνες στα κρυμμενα επιπεδα.
- Εχοντας κατα νου τις καλυτερες επιδοσεις για τις δυο συναρτησεις, παρατηρουμε πως για batch size = 40, η tanh ειναι κατα πολυ καλυτερη της relu. (γενικευτικη ικανοτητα >50& vs 25% αντιστοιχα)

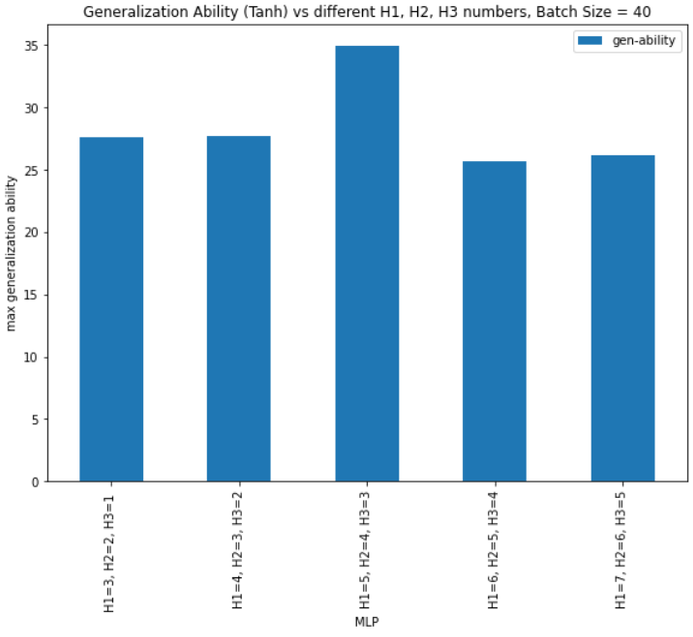
Bar plots για συνδυασμους τιμων H1, H2, H3 για την tanh, relu με batch size = 40

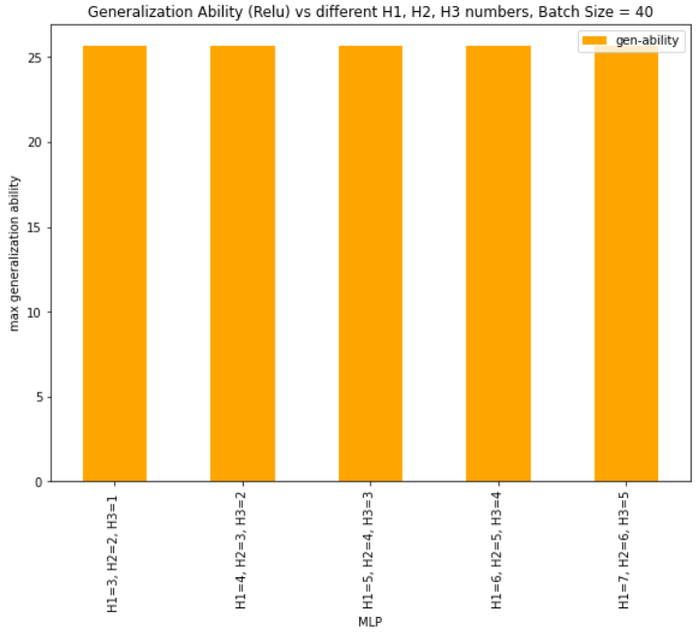
In [5]:

```
fig, ax = plt.subplots(1, 2, figsize = (21, 7))

df1 = pd.DataFrame({'MLP':['H1=3, H2=2, H3=1 ', 'H1=4, H2=3, H3=2', 'H1=5, H2=4, H3=3', 'H1=6, H2=5, H3=4', 'H1
                    'gen-ability':[27.63, 27.71, 34.96, 25.67, 26.14]})
ax[0] = df1.plot.bar(x = 'MLP', y = 'gen-ability', ax = ax[0])
ax[0].title.set_text('Generalization Ability (Tanh) vs different H1, H2, H3 numbers, Batch Size = 40')
ax[0].set_ylabel('max generalization ability')

df2 = pd.DataFrame({'MLP':['H1=3, H2=2, H3=1 ', 'H1=4, H2=3, H3=2', 'H1=5, H2=4, H3=3', 'H1=6, H2=5, H3=4', 'H1
                    'gen-ability':[25.67, 25.67, 25.67, 25.67, 25.67]})
df2.plot.bar(x = 'MLP', y = 'gen-ability', color = 'orange', ax = ax[1])
ax[1].title.set_text('Generalization Ability (Relu) vs different H1, H2, H3 numbers, Batch Size = 40')
ax[1].set_ylabel('max generalization ability')
```

Out[5]:

```
Text(0, 0.5, 'max generalization ability')
```





Παρατηρησεις:

- Οσον αφορα την tanh δεν παρατηρειται καποια αυξηση στη γενικευτικη ικανοτητα οσο αυξανονται οι κρυμμενοι νευρωνες, εκτος απο την κορυφωση που παρατηρειται στις τιμες H1 = 5, H2 = 4, H3 = 3.
- Οσον αφορα τη relu, η γενικευτικη ικανοτητα παραμενει η ιδια οσο αυξανονται οι οι νευρωνες στα κρυμμενα επιπεδα.
- Εχοντας κατα νου τις καλυτερες επιδοσεις για τις δυο συναρτησεις, παρατηρουμε πως για batch size = 40, η tanh ειναι κατα πολυ καλυτερη της relu. (γενικευτικη ικανοτητα 35& vs 25% αντιστοιχα)

Conclusion (batch size = 40):

Για batch size = 40 και MLP 2 κρυμμενων επιπεδων παρατηρουμε πως η tanh εχει καλυτερη επιδοση απο τη relu, ενω για MLP 3 κρυμμενων επιπεδων παρατηρουμε παλι πως η tanh εχει καλυτερη επιδοση απο τη relu.

Δεν προκυπτει καποιο οφελος απο τη χρηση του 3ου κρυμμενου επιπεδου καθως η γενικευτικη ικανοτητα του 2-layer MLP ειναι μεγαλυτερη απο αυτη του 3-layer MLP. (γενικευτικη ικανοτητα >50% vs 35% αντιστοιχα)

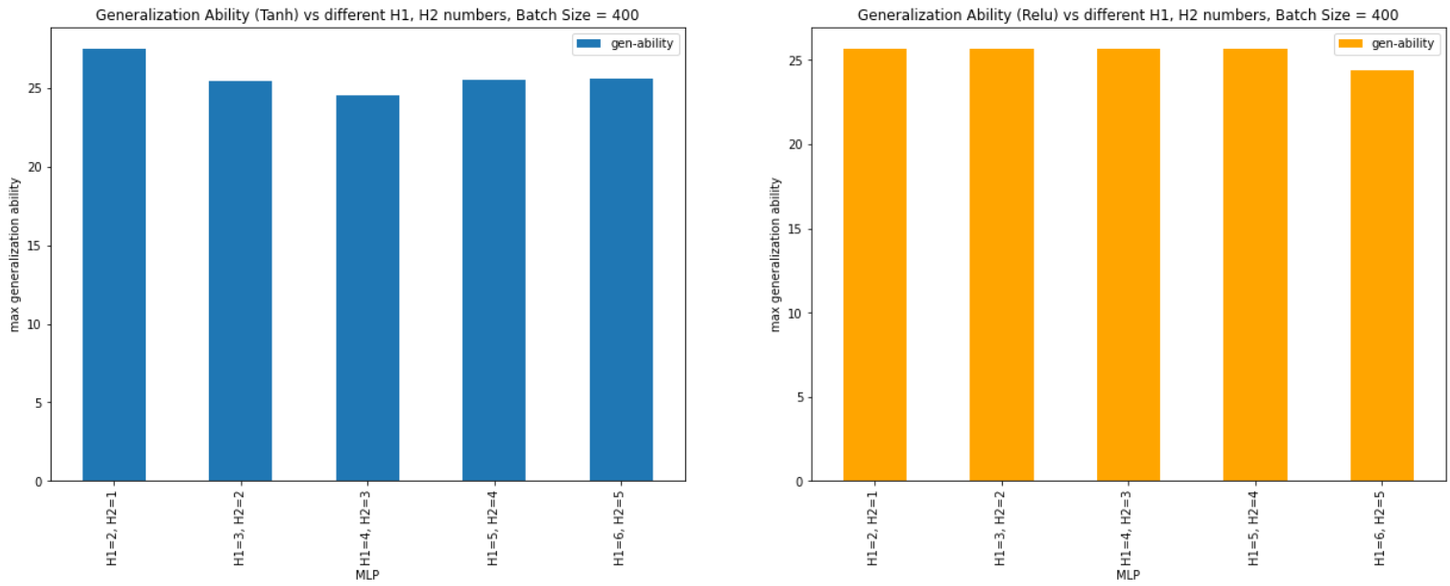
Bar plots για συνδυασμους τιμων H1, H2 για την tanh, relu με batch size = 400

In [6]:

```
fig, ax = plt.subplots(1, 2, figsize = (21, 7))

df1 = pd.DataFrame({'MLP':['H1=2, H2=1', 'H1=3, H2=2', 'H1=4, H2=3', 'H1=5, H2=4', 'H1=6, H2=5'],
                    'gen-ability':[27.51, 25.45, 24.53, 25.56, 25.64]})
ax[0] = df1.plot.bar(x = 'MLP', y = 'gen-ability', ax = ax[0])
ax[0].title.set_text('Generalization Ability (Tanh) vs different H1, H2 numbers, Batch Size = 400')
ax[0].set_ylabel('max generalization ability')

df2 = pd.DataFrame({'MLP':['H1=2, H2=1', 'H1=3, H2=2', 'H1=4, H2=3', 'H1=5, H2=4', 'H1=6, H2=5'],
                    'gen-ability':[25.67, 25.67, 25.67, 25.67, 24.36]})
df2.plot.bar(x = 'MLP', y = 'gen-ability', color = 'orange', ax = ax[1])
ax[1].title.set_text('Generalization Ability (Relu) vs different H1, H2 numbers, Batch Size = 400')
ax[1].set_ylabel('max generalization ability')
```

Out[6]:

```
Text(0, 0.5, 'max generalization ability')
```



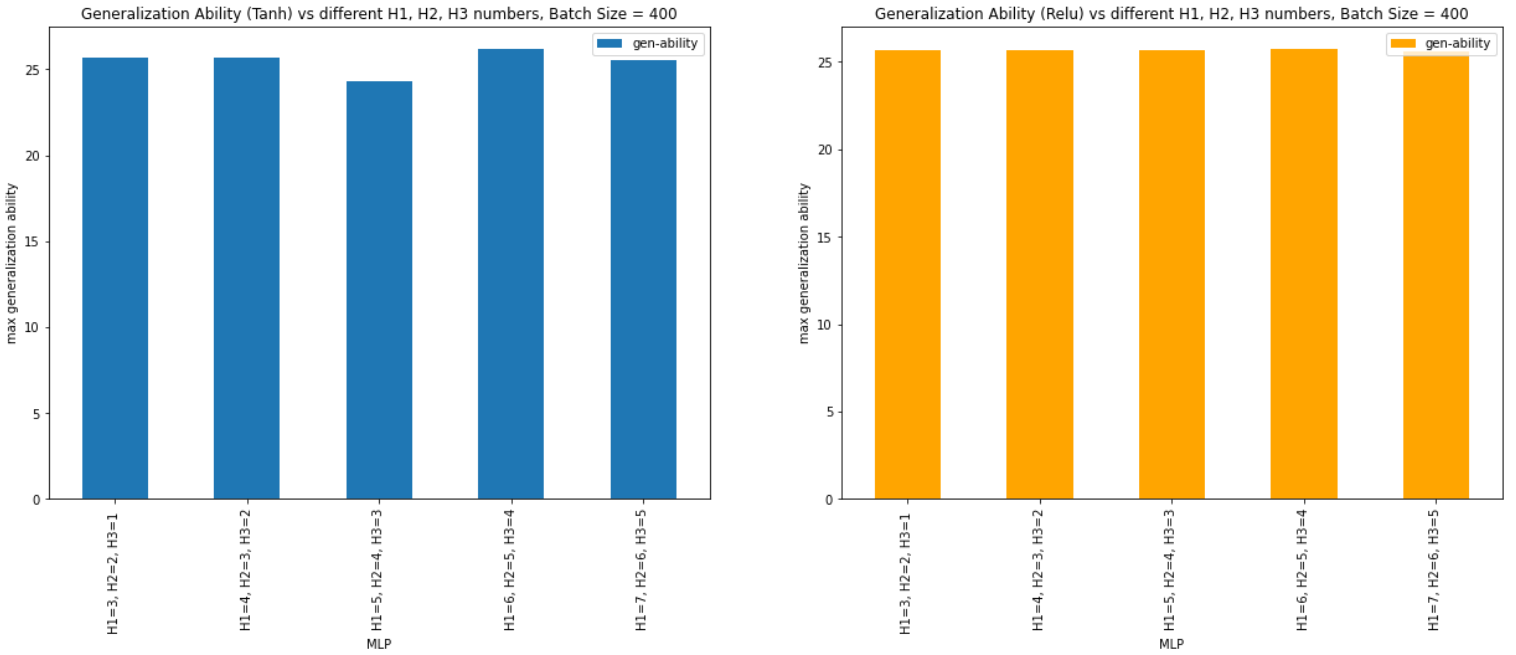
Παρατηρησεις:

- Οσον αφορα την tanh δεν παρατηρειται καποια αυξηση στη γενικευτικη ικανοτητα οσο αυξανονται οι κρυμμενοι νευρωνες. Η κορυφωση παρατηρειται στις τιμες H1 = 2, H2 = 1.
- Οσον αφορα τη relu, η γενικευτικη ικανοτητα παραμενει η ιδια οσο αυξανονται οι οι νευρωνες στα κρυμμενα επιπεδα.
- Εχοντας κατα νου τις καλυτερες επιδοσεις για τις δυο συναρτησεις, παρατηρουμε πως για batch size = 400, η tanh ειναι το ιδιο ικανη με τη relu. (γενικευτικη ικανοτητα γυρω στο 25% και για τις δυο)

Bar plots για συνδυασμους τιμων H1, H2, H3 για την tanh, relu με batch size = 400

In [7]:
```
fig, ax = plt.subplots(1, 2, figsize = (21, 7))

df1 = pd.DataFrame({'MLP':['H1=3, H2=2, H3=1 ', 'H1=4, H2=3, H3=2', 'H1=5, H2=4, H3=3', 'H1=6, H2=5, H3=4', 'H1
                    'gen-ability':[25.67, 25.67, 24.32, 26.2, 25.56]})
ax[0] = df1.plot.bar(x = 'MLP', y = 'gen-ability', ax = ax[0])
ax[0].title.set_text('Generalization Ability (Tanh) vs different H1, H2, H3 numbers, Batch Size = 400')
ax[0].set_ylabel('max generalization ability')

df2 = pd.DataFrame({'MLP':['H1=3, H2=2, H3=1 ', 'H1=4, H2=3, H3=2', 'H1=5, H2=4, H3=3', 'H1=6, H2=5, H3=4', 'H1
                    'gen-ability':[25.67, 25.67, 25.67, 25.74, 25.56]})
df2.plot.bar(x = 'MLP', y = 'gen-ability', color = 'orange', ax = ax[1])
ax[1].title.set_text('Generalization Ability (Relu) vs different H1, H2, H3 numbers, Batch Size = 400')
ax[1].set_ylabel('max generalization ability')
```

Out[7]:
```
Text(0, 0.5, 'max generalization ability')
```



Παρατηρησεις:

- Οσον αφορα την tanh δεν παρατηρειται καποια αυξηση στη γενικευτικη ικανοτητα οσο αυξανονται οι κρυμμενοι νευρωνες.
- Οσον αφορα τη relu, η γενικευτικη ικανοτητα παραμενει η ιδια οσο αυξανονται οι οι νευρωνες στα κρυμμενα επιπεδα.
- Εχοντας κατα νου τις καλυτερες επιδοσεις για τις δυο συναρτησεις, παρατηρουμε πως για batch size = 400, η tanh ειναι το ιδιο ικανη με τη relu. (γενικευτικη ικανοτητα γυρω στο 25% και για τις δυο)

Conclusion (batch size = 400):

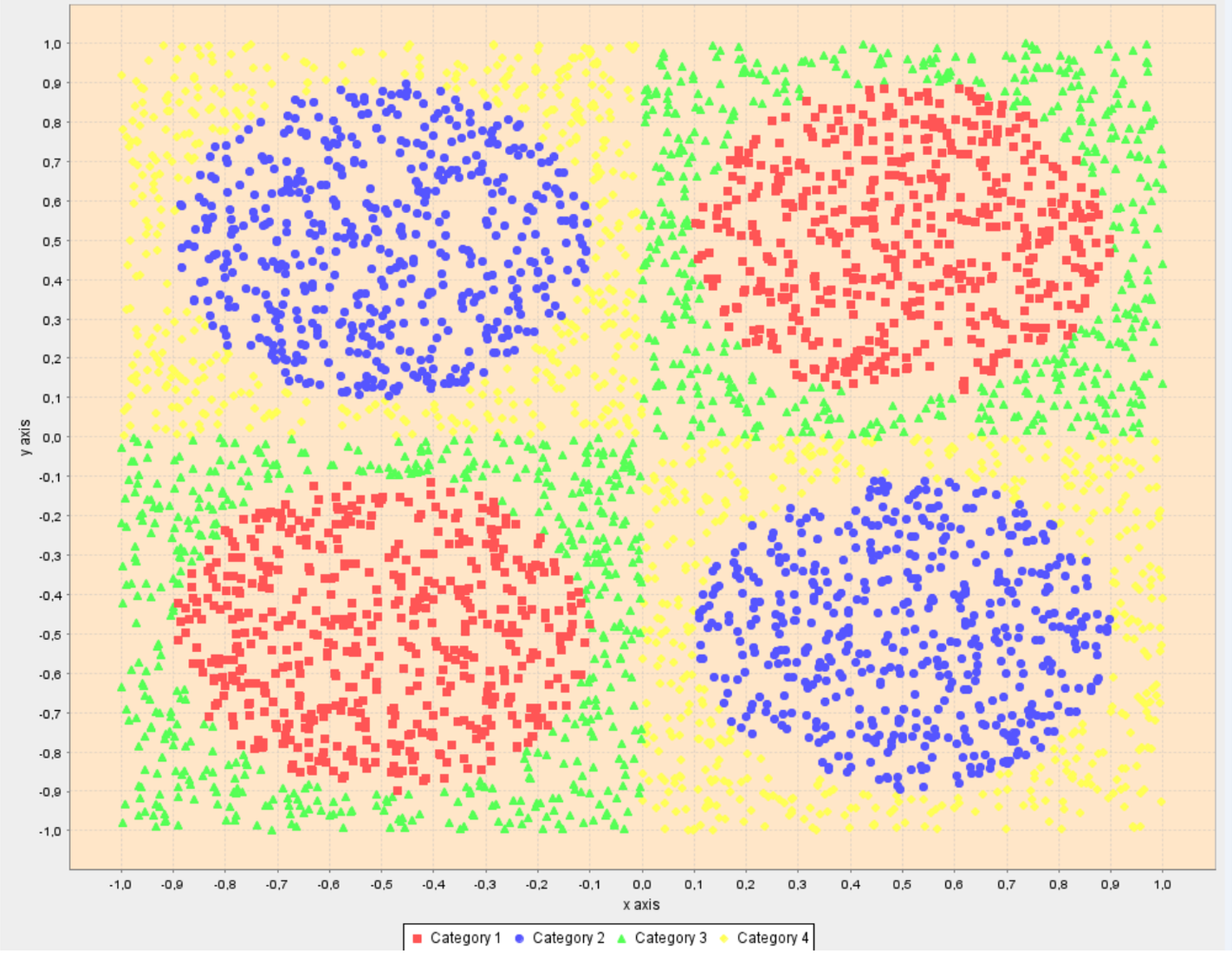
Για batch size = 400 και MLP 2 κρυμμενων επιπεδων παρατηρουμε πως η tanh εχει την ιδια επιδοση με τη relu, ενω για MLP 3 κρυμμενων επιπεδων παρατηρουμε παλι πως η tanh εχει την ιδια επιδοση με τη relu.

Δεν προκυπτει καποιο οφελος απο τη χρηση του 3ου κρυμμενου επιπεδου καθως η γενικευτικη ικανοτητα του 2-layer MLP ειναι η ιδια με αυτη του 3-layer MLP. (γενικευτικη ικανοτητα 25% και για τα δυο MLP)

Plot παραδειγματων του test set του δικτυου με την καλυτερη γενικευτικη ικανοτητα

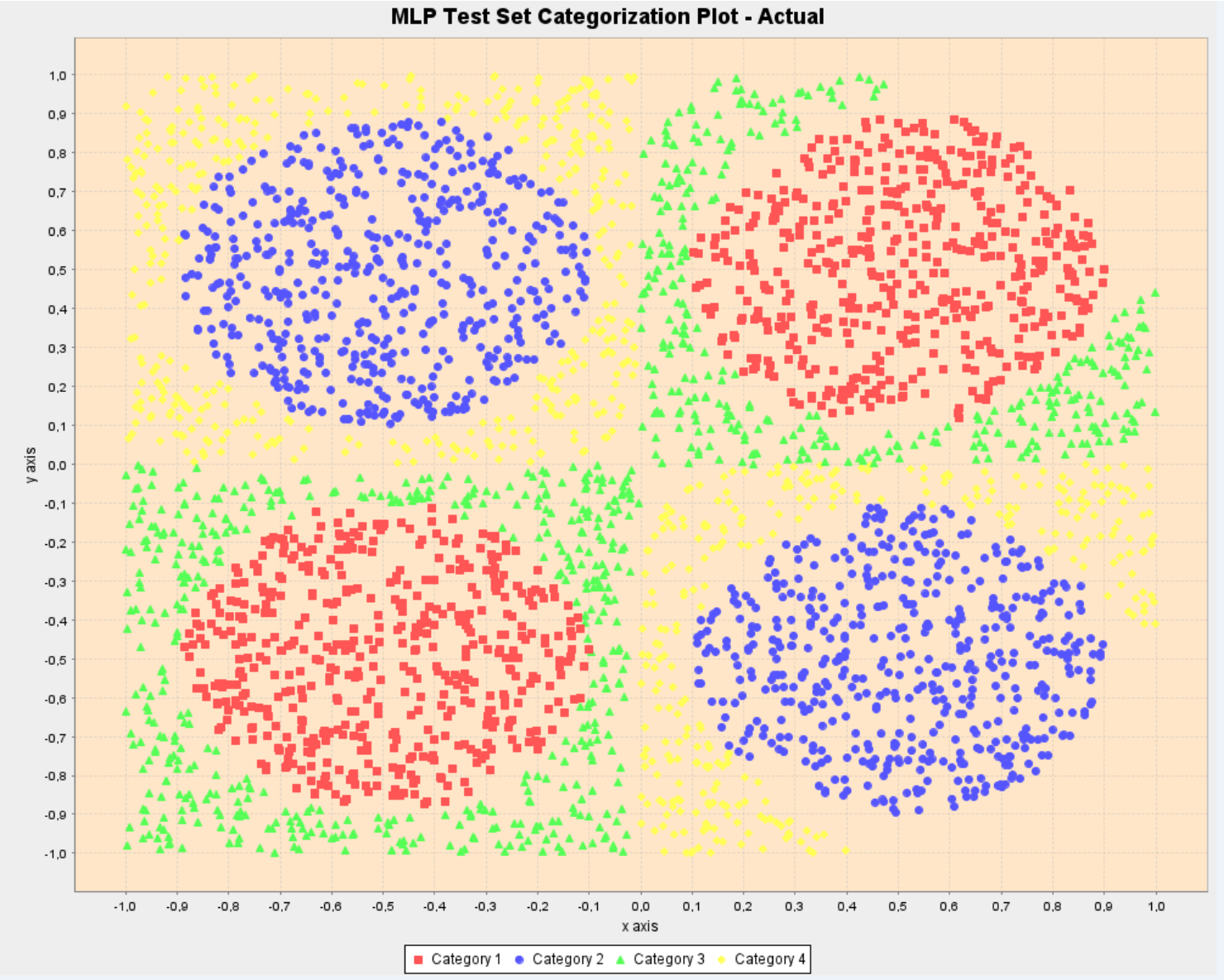
Plot του dataset S1 με τις 4 κατηγοριες:

MLP Test Set Categorization Plot - Expected

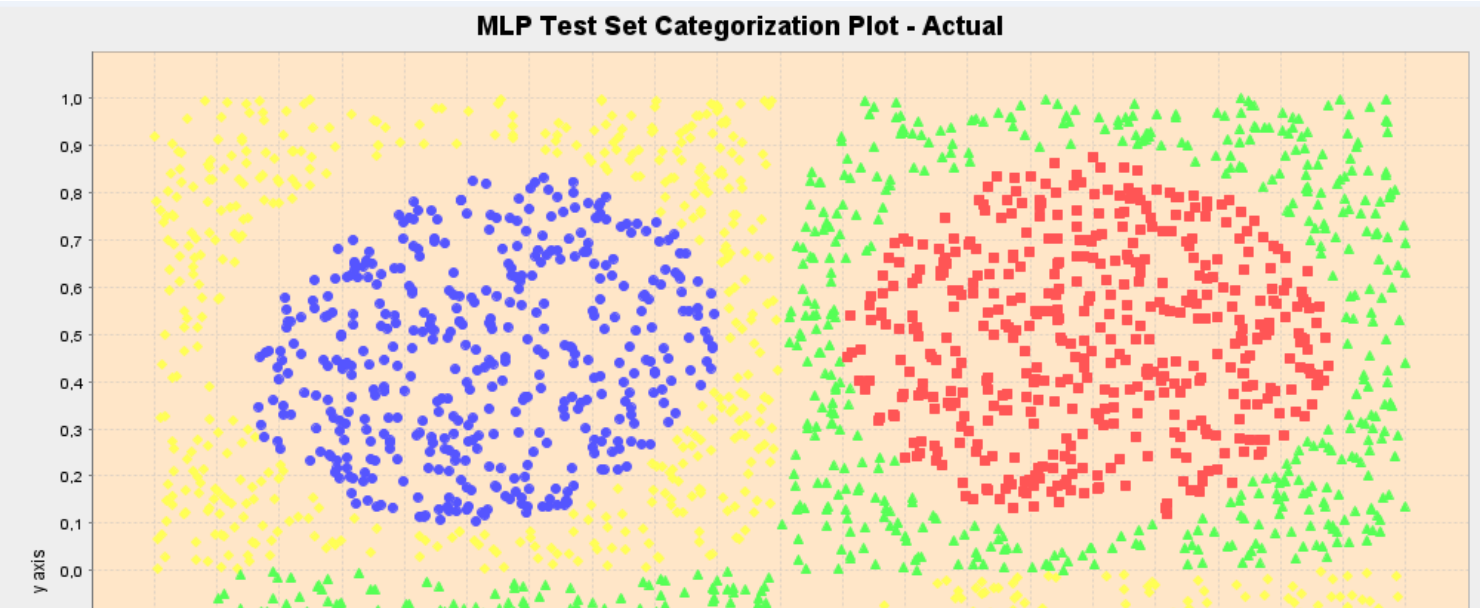

y axis
x axis
Category 1
Category 2
Category 3
Category 4

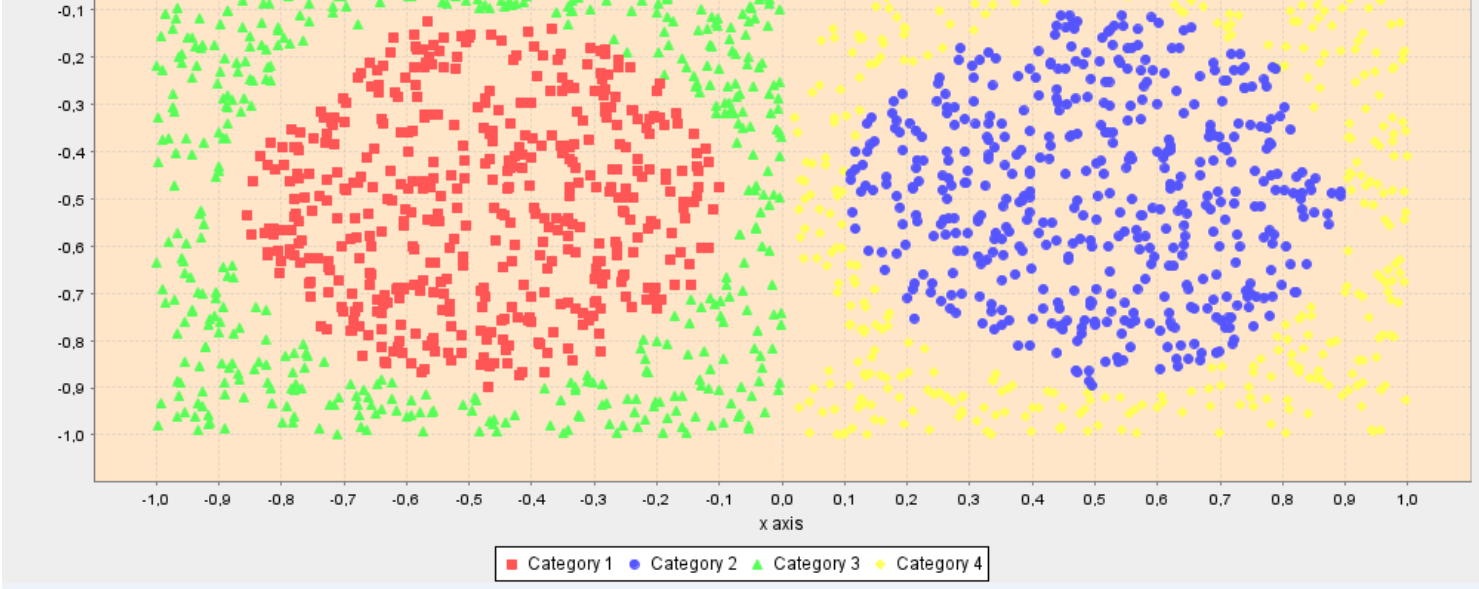
Plot των καλυτερων δικτυων που μελετησαμε:

2-layer MLP, H1 = 6, H2 = 5, Batch Size = 1, mean generalization ability = 86.02%

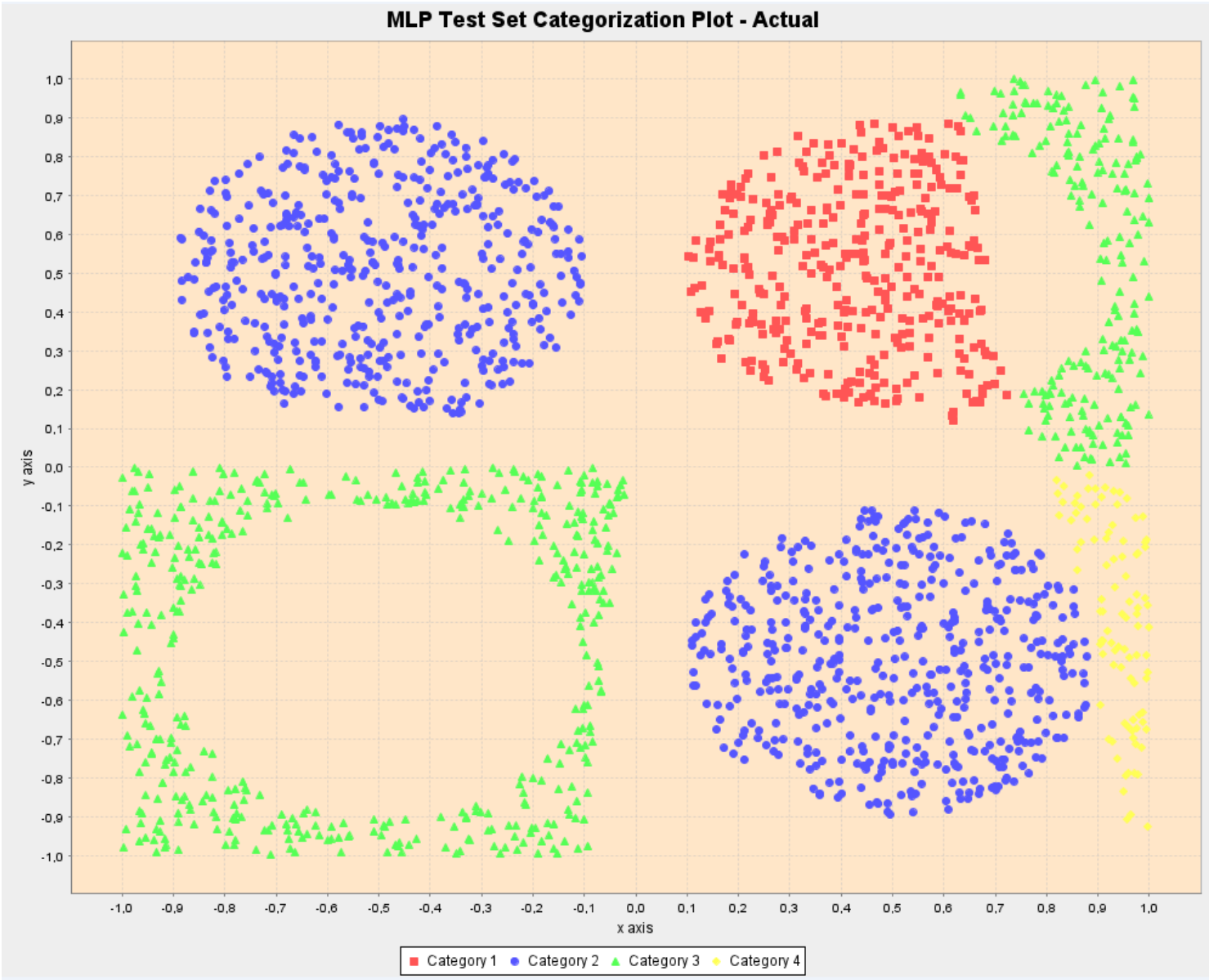


3-layer MLP, H1 = 7, H2 = 6, H3 = 5, Batch Size = 1, mean generalization ability = 88.67%

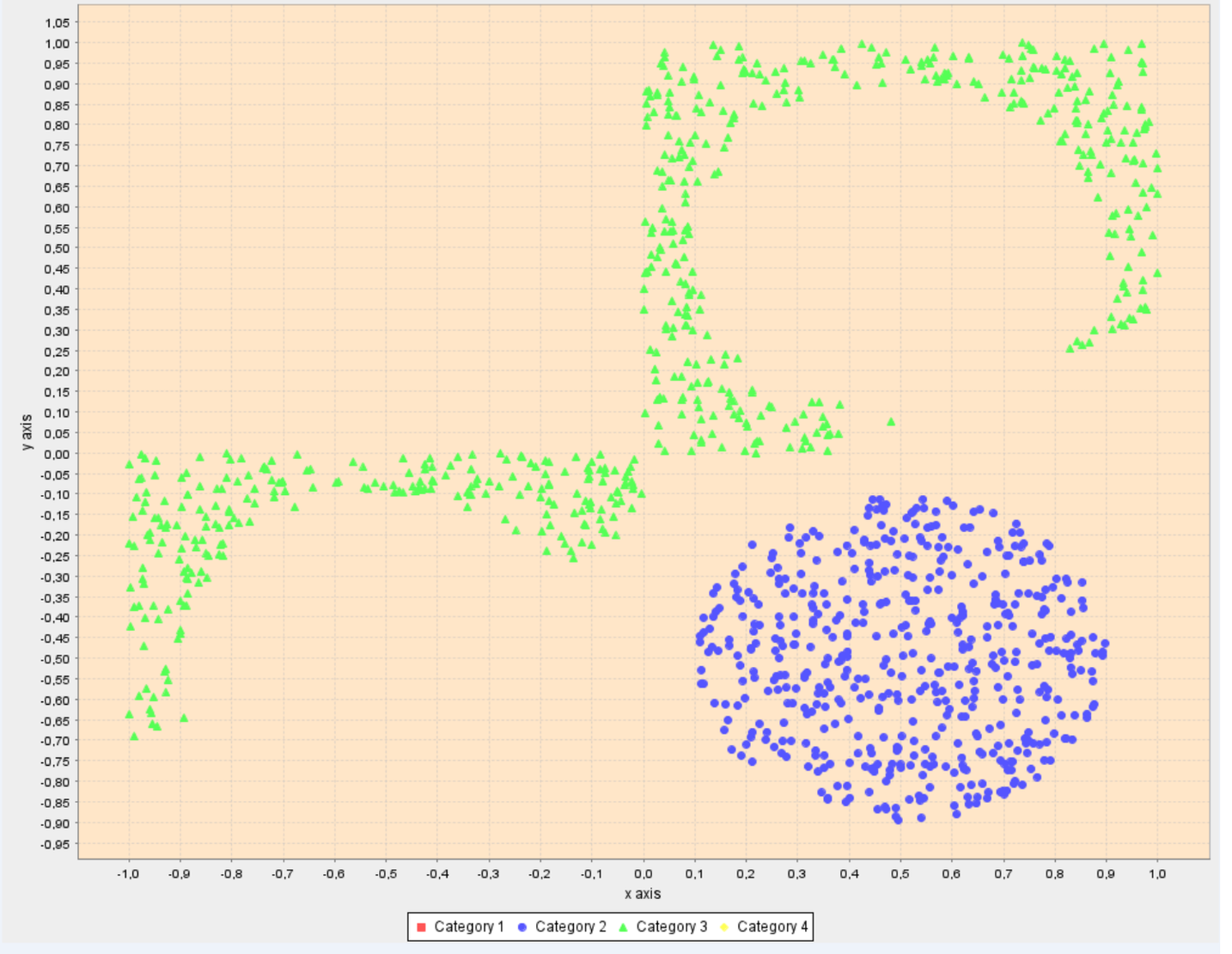




2-layer MLP, H1 = 6, H2 = 5, Batch Size = 40, mean generalization ability = 53.11%



3-layer MLP, H1 = 5, H2 = 4, H3 = 3, Batch Size = 40, mean generalization ability = 34.96%


y axis
x axis
Category 1
Category 2
Category 3
Category 4