BRSU

Neural Networks
Assignment 7

Bastian Lang

November 22, 2015

1 OUTLINE

- Output Representation and Decision Rule
- Computer Experiment
  - Bayesian Decision Boundary
  - Experimental Determination of Optimal Multilayer Perceptron
    - Optimal Number of Hidden Neurons
    - Optimal Learning and Momentum Constants
    - Evaluation of Optimal Network Design
- Feature Detection
  - Relation to Fisher's Linear Discriminent
- Back-Propagation and Differentation
  - Jacobian Matrix
- Hessian Matrix
- Generalization
  - Sufficient Training Set Size for a Valid Generalization
- Approximation of Functions
  - Universal Approximation Theorem
  - Bounds on Approximation Errors
  - Curse of Dimensionality
  - Practical Considerations
- Cross-Validation
  - Model Selection
  - Early Stopping Method of Training
  - Variants of Cross-Validation
- Network Pruning Techniques
  - Complexity-Regularization
  - Weight Decay
  - Weight Elimination
  - Approximate Smoother
  - Hessian-based Network Pruning
  - Computing the inverse Hessian matrix

- Virtues and Limitations of Back-Propagation Learning
  - Connectionism
  - Feature Detection
  - Function Approximation
  - Computational Efficiency
  - Sensitivity Analysis
  - Robustness
  - Convergence
  - Local Minima
  - Scaling
- Accelerated Convergence of Back-Propagation Learning
  - 4 Heuristics
- Supervised Learning viewed as an Optimization Problem
  - Conjugate-Gradient Method
  - Example
  - Summary of the Nonlinear Conjugate Gradient Algorithm
  - Quasi-Newton Methods
  - Comparison of Quasi-Newton Methods with Conjugate-Gradient Methods
- Convolutional Networks
- Summary and Discussion

2 PCA & ICA

2.1 OUTPUT

2.2 CODE

```
# -*- coding: utf-8 -*-
"""
Created on Sat Nov 21 12:41:57 2015

@author: bastian
"""

from matplotlib.mlab import PCA as mlabPCA
import matplotlib.pyplot as plt
```

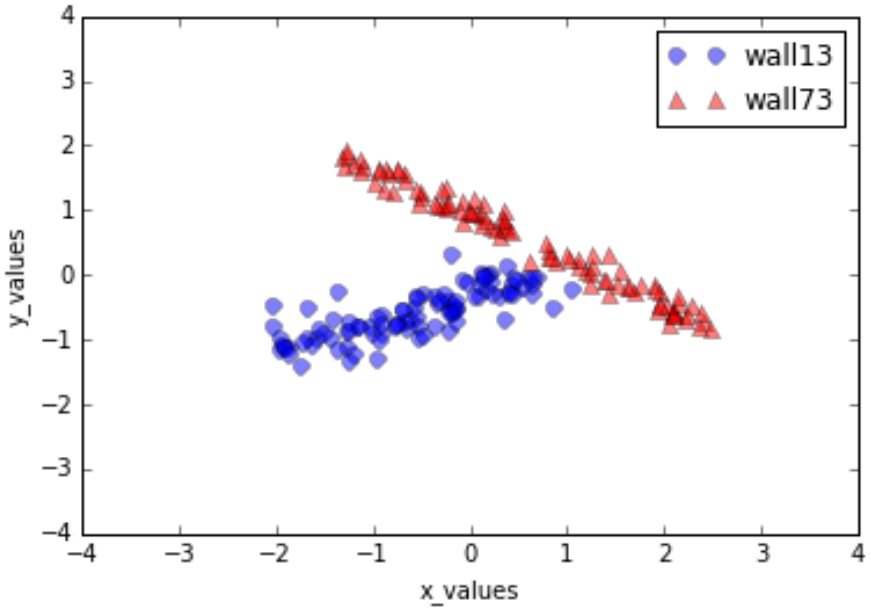

Figure 2.1: Both datasets in new coordinate system after performing PCA.

```
import numpy as np
from sklearn.decomposition import FastICA as ICA
from sklearn.cluster import KMeans
import neurolab as nl


def cluster_data(data,class_label):

    result = KMeans(n_clusters=2, random_state=170).fit_predict(data)


    plt.scatter(data[:,0],data[:,1], c=result)

    plt.xlabel('x_values')
    plt.ylabel('y_values')
    plt.xlim([-4,4])
    plt.ylim([-4,4])
    plt.legend()
    plt.title('Transformed samples versus original data')

    plt.show()
```

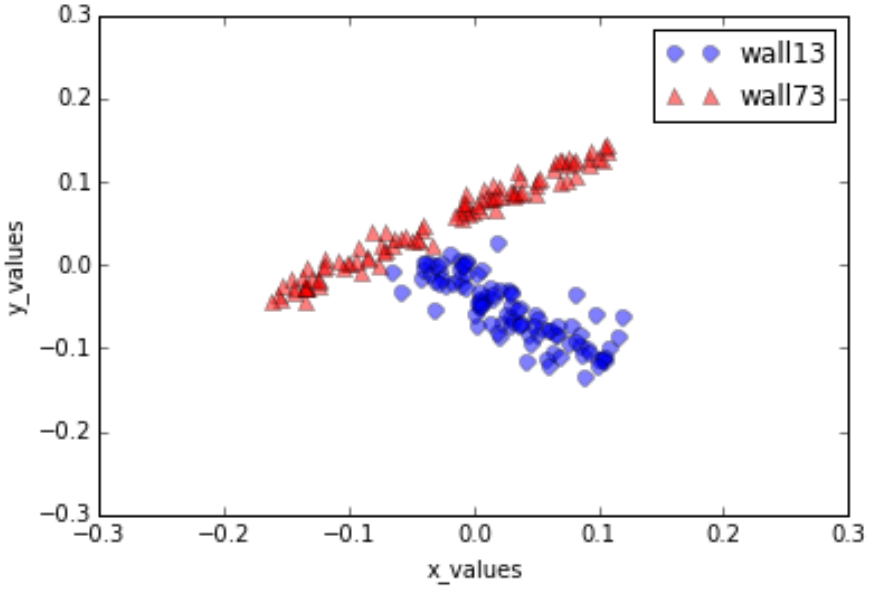

Figure 2.2: Both datasets in new coordinate system after performing ICA.

```
def do_ica(data, class_label):

    # ica
    ica = ICA()
    result = ica.fit(data).transform(data)



    plt.plot(result[:,0],result[:,1],
             'o', markersize=7, color='blue', alpha=0.5, label=class_label)

    plt.xlabel('x_values')
    plt.ylabel('y_values')
    plt.xlim([-0.5,0.5])
    plt.ylim([-0.5,0.5])
    plt.legend()
    plt.title('Transformed samples versus original data')

    plt.show()


def do_pca(data, class_label):
```

```
    mlab_pca = mlabPCA(wall13_data)


    print('PC axes in terms of the measurement axes scaled by the standard deviations:\n

    # pca
    plt.plot(mlab_pca.Y[:,0],mlab_pca.Y[:,1],
             'o', markersize=7, color='blue', alpha=0.5, label=class_label)
    # original
    plt.plot(wall13_data[:,0], wall13_data[:,1],'^', markersize=7, color='red', alpha=0.

    plt.xlabel('x_values')
    plt.ylabel('y_values')
    plt.xlim([-4,40])
    plt.ylim([-4,10])
    plt.legend()
    plt.title('Transformed samples versus original data')

    plt.show()

    return mlab_pca.Y



def split_pca(combined_data, label_1, label_2):

    mlab_pca = mlabPCA(combined_data)

    print('PC axes in terms of the measurement axes scaled by the standard deviations:\n

    plt.plot(mlab_pca.Y[0:100,0],mlab_pca.Y[0:100,1],
             'o', markersize=7, color='blue', alpha=0.5, label=label_1)
    plt.plot(mlab_pca.Y[100:200,0], mlab_pca.Y[100:200,1],
             '^', markersize=7, color='red', alpha=0.5, label=label_2)

    plt.xlabel('x_values')
    plt.ylabel('y_values')
    plt.xlim([-4,4])
    plt.ylim([-4,4])
    plt.legend()
    #plt.title('Transformed samples with class labels from matplotlib.mlab.PCA()')

    plt.show()
```

```
    return mlab_pca.Y


def split_ica(combined_data, label_1, label_2):

    ica = ICA()
    result = ica.fit(combined_data).transform(combined_data)


    plt.plot(result[0:100,0],result[0:100,1],
             'o', markersize=7, color='blue', alpha=0.5, label=label_1)
    plt.plot(result[100:200,0], result[100:200,1],
             '^', markersize=7, color='red', alpha=0.5, label=label_2)

    plt.xlabel('x_values')
    plt.ylabel('y_values')
    plt.xlim([-0.3,0.3])
    plt.ylim([-0.3,0.3])
    plt.legend()
    #plt.title('Transformed samples with class labels from matplotlib.mlab.PCA()')

    plt.show()

    return result

wall13_data = np.genfromtxt('wall13.csv', delimiter=',')
do_pca(wall13_data, 'wall13')
#o_ica(wall13_data, 'wall13')


wall73_data = np.genfromtxt('wall73.csv', delimiter=',')
do_pca(wall73_data, 'wall73')
do_ica(wall73_data, 'wall73')

mixed_data = np.concatenate((wall13_data, wall73_data), axis=0)
Y = split_pca(mixed_data, 'wall13', 'wall73')
split_ica(mixed_data, 'wall13', 'wall73')
#cluster_data(Y, 'mixed')
```