Line H. Clemmensen & Sneha Das
DTU Compute

Line H. Clemmensen & Sneha Das , DTU Compute

Exercises 02582
Module 10
Spring 2025

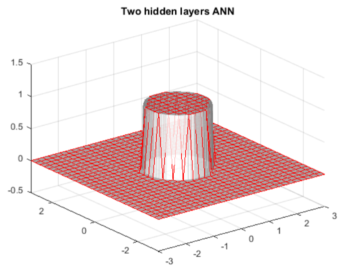
April 10, 2025

Topics: ANNs, Autoencoders

Exercises (Coding hints at the end of this document):

1 Fitting a regression neural network to a cylinder:

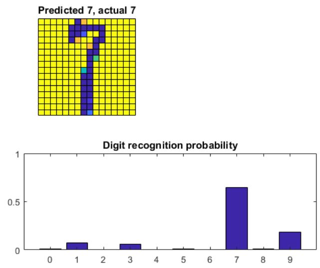
- Use the function `ANNcylinder.m` or correspondingly in R or Python, and experiment with different **two layer networks**.
- What is the minimum number of nodes in a two layer networks that outperforms one layer with 100 nodes?



2 Fit a neural network to the Zip data set (classification of digits) and try using regularizations to avoid overfitting:

- Try ANN classification with `ANNearlystoppingZip.m` or correspondingly in R or Python, and experiment with different models.

- Try regularized ANN with `ANNregularizationZip.m` or correspondingly in R or Python, and different penalty on the weights.

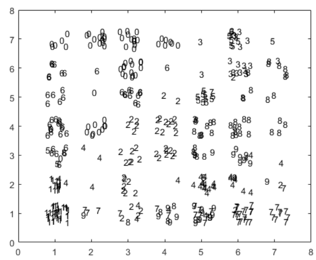


3 Try compressing the zip data with an autoencoder. You can use `AutoencoderZip.m` in Matlab. The weights in the encoder represents the pattern used in each latent variable.

- What is the cost function for the autoencoder?
- How many layers do you need in the encoder and decoder, respectively?
- How many nodes (dimensions) do you need in your latent representation to get a decent reconstruction?
- What should you do in order to get a low dimensional linear latent representation of your data using NN autoencoders?

4 Try the SOM clustering method in the Zip data:

- Try clustering of zip data using Self Organizing Maps. You can use `SOMzip.m`
- Use two dimensional grids with different number of nodes.
- Can you interpret the result?



Line H. Clemmensen & Sneha Das
DTU Compute

Exercises 02582
Module 10
Spring 2025

April 10, 2025

Topics: ANNs, Autoencoders

Resources for this exercise:

Listing 1: Resources in Matlab

```
load zipdata % load data

net = fitnet(); % Create a Fitting Network
net.divideParam.trainRatio = % Set up Division of Data
net.divideParam.valRatio   = % Set up Division of Data
net.divideParam.testRatio  = % Set up Division of Data
net.performParam.regularization =  %  Define L2 norm
   regularization
net.trainParam.epochs = % Set maximum number of training
   epochs
[net,tr] = train(); % Train the Network
view(net) % View the Network
outputs = net(); % Test the Network

autoenc = trainAutoencoder(Xtrain,hiddenSize,... %Train
   autoencoder
autoenc = stack(autoenc1,autoenc2); %Stack autoencoders
Xpred = predict(autoenc,Xtrain); % Calculate reconstructed
   data
Xcomp   = encode(autoenc,Xtest); % Encode test data
```

```
Xdecode = decode(autoenc,Xcomp); % Decode test data
w_encoder = autoenc.EncoderWeights; %Extract encoder weights
w_decoder = autoenc.DecoderWeights; %Extract decoder weights
view(autoenc) % View the Network

net = selforgmap(); % Create Self Organizing Network
[net,tr] = train(net,X'); % Train Self Organizing Network
```

Listing 2: Resources in R

```
mainANNcylinder.R # Exercise 1
ANNearlystoppingZip.R # Exercise 2
ANNregularizationZip.R # Exercise 2
Autoencoderzip.R # Exercise 3
SOM.R # Exercise 4
zipdata.mat # zipdata
library(neuralnet) # Training of neural nets
library(ANN2) # Autoencoders
library(kohonen) # Self-organizing maps (SOM)

readMat("zipdata.mat") # Read zipdata
neuralnet(X,hid,...) # Train neural networks
autoencoder(X,hid,...) # Train autoencoder
som(X,grid,...) # Train self-organizing map
```

Listing 3: Resources in Python

```
zipdata.mat # zip data
import numpy as np # numpy
from mpl_toolkits.mplot3d import Axes3D # for 3d plots in
    matplotlib
train_dataset = Data.TensorDataset(X_train, y_train) # to
    create a dataset for dataloader in PyTorch
pip install minisom or conda install minisom # to install
    SOM

import torch # import pytorch for neural networks
torch.nn.Linear(in_feature, out_features) # linear layer in
    ann
torch.nn.ReLU() # relu function
torch.tensor(X_train, dtype=torch.float) # convert to torch
    tensor with type float
torch.nn.MSELoss()  # MSE loss function
torch.nn.CrossEntropyLoss # CrossEntropy loss function
```

```
earlyStopping = EarlyStopping(min_delta=-2, patience=5) #
   for creating earlystopping class
torch.optim.SGD(aenet.parameters(), lr=learning_rate,
   weight_decay=weight_decay) # weight_decay is
   regularization

from minisom import MiniSom  # import SOM
som = MiniSom(shape[0], shape[1], features) # create SOM
   class with shape (x,y) and number of features in the data
som.pca_weights_init(X) # Initializes the weights to span
   the first two principal components.
som.train_random(X, num_iteration, verbose=True) #Trains the
    SOM picking samples at random from data.
```

End of exercise