Neural Networks in MATLAB

This is a good resource on Deep Learning for papers and code:
https://github.com/kjw0612/awesome-deep-vision
Simple Neural Networks in Matlab

• Construct an empty NN object using “patternnet”
  – Specify number of hidden layers, and neurons in each hidden layer

![Neural net with one hidden layer](image)

• Train the network using “train”
  – Pass in the feature vectors and their labeled classes

• Run the network on new data using “net”
Data Format

• Feature vectors are in a $d \times N$ array
  – $d$ is the dimensionality of the vectors
  – $N$ is the number of vectors

• Labeled classes are in a $K \times N$ array
  – $K$ is the number of classes
  – $N$ is the number of vectors

Each column has a 1 corresponding to the class number of each example
clear all
close all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Load Fisher's iris data set. This loads in
% meas(N,4) - feature vectors, each 4 dimensional
% species(N) - class names: 'versicolor', 'virginica', 'setosa'
load fisheriris

N = length(species);

% We will just use 2 feature dimensions, since it is easier to visualize.
% Each column of X is an example.
X = meas(:,1:2)'; % size(X) is (2,N)

% Get indices for examples from each class.
inds1 = find(strcmp(species,'versicolor'));
inds2 = find(strcmp(species, 'virginica'));
inds3 = find(strcmp(species, 'setosa'));

% Assign class numbers.
classNumbers = zeros(1,N);
classNumbers(inds1) = 1;
classNumbers(inds2) = 2;
classNumbers(inds3) = 3;

% Create a "target matrix". Each corresponding
% column of the target matrix will have all zeros except for a one in the
% location of the correct class.
T = zeros(3,N);
T(1,inds1) = 1;
T(2,inds2) = 1;
T(3,inds3) = 1;

figure, hold on;
myColors = [ 'r', 'g', 'b' ];

for i=1:N
    plot(X(1,i),X(2,i), ...
         'Color', myColors(classNumbers(i)), 'Marker', 'o');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Try varying the number of hidden units

Example – Fisher’s iris data (2 of 2)
• Note structure of network

• Look at plots
Special case: the NN classifier with one hidden unit

- The single hidden unit is performing the operation

\[ h(x, \theta, b) = \theta_1 x_1 + \theta_2 x_2 + b \]

- This is the equation of a plane

- The output units are selecting a value of \( h \) for each class
Deep Neural Networks

- A “deep” neural network has multiple hidden layers

- It is hard to train unless you have a good starting guess for the parameters
- We can use “autoencoders” to create a good starting guess
Autoencoder

• We train the network to re-create the input, but using a small number of hidden units
  – This way the network automatically learns important features of the input data

• Each hidden layer is trained as an autoencoder
Deep Network Training

- Train a sequence of shallow autoencoders using unsupervised data
- Train the last layer using supervised data
- Fine-tuning: use backpropagation to fine-tune the entire network using supervised data
Deep Neural Networks in Matlab

- Construct and train autoencoders using "trainAutoencoder"
- Predict the classes of vectors using "predict"
- Call "encode" to get the encoded values of input vectors

Call "trainSoftmaxLayer" to train the final output layer using labeled examples
Example – classify images of digits

• Matlab has a dataset of digit images
  – There are 5000 labeled images, each 28x28 pixels
  – So the input vectors are 784 dimensions
• We’ll use two hidden layers
  – The first will have 100 hidden units

• Note - the Matlab dataset is synthetic
  – However, NIST has real images (digits and letters), available at https://www.nist.gov/srd/nist-special-database-19
  – You can get a Matlab-ready version of the NIST database at https://www.westernsydney.edu.au/bens/home/reproducible_research/emnist
Deep Neural Networks in Matlab (con’t)

- Call “stack” to stack a set of autoencoders to create a deep network

- Call “deepnet” to classify input vectors using the deep network

- Call “train” to fine-tune the whole network
This program is adapted from the Matlab example.
clear all
close all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Get training data. "xTrainImages" is a cell array of length 5000. Each
% element is a 28x28 image. "tTrain" is a 10x5000 array, indicating the
% class of each training example.
[xTrainImages, tTrain] = digittrain_dataset;

% For speed, let's subsample the data.
K = 6;       % Keep the first K classes
N = 2000;    % Keep only up to N images
ids = find(vec2ind(tTrain)<=K);     % Get ids of images with class <= K
xTrainImages = xTrainImages(ids);   % Keep only those training images
tTrain = tTrain(1:K,ids);

if length(xTrainImages)>N
    xTrainImages = xTrainImages(1:N);   % Keep only first N images
tTrain = tTrain(:,1:N);
end

fprintf('Number of training images:
');
for k=1:K
    fprintf('Class %d:  %d
', k, sum(vec2ind(tTrain)==k));
end

figure;     % Show example images
for i = 1:25
    subplot(5,5,i);
    imshow(xTrainImages{i});
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Get testing data.
[xTestImages, tTest] = digittest_dataset;

ids = find(vec2ind(tTest)<=K);     % Get ids of images with class <= K
xTestImages = xTestImages(ids);   % Keep only those test images
tTest = tTest(1:K,ids);
% Train the first autoencoder.
hiddenSize1 = 100;  % Number of neurons
 tic
autoenc1 = trainAutoencoder(xTrainImages,hiddenSize1, ...  % default is 1000
'MaxEpochs',400, ...  % default is 1000
'L2WeightRegularization',0.004, ...  
'SparsityRegularization',4, ...  
'SparsityProportion',0.15, ...  
'ScaleData', false);
figure, plotWeights(autoenc1);  % Show features

% Reconstruct the data using the trained autoencoder.
xReconstructed = predict(autoenc1,xTrainImages);

figure;  % Show reconstructed images  
for i = 1:25  
    subplot(5,5,i);  
    imshow(xReconstructed{i});  
end

% The 100-dimensional output from the hidden layer of the autoencoder is a  
% compressed version of the input, which summarizes its response to the  
% features visualized above. Train the next autoencoder on a set of these  
% vectors extracted from the training data. First, you must use the encoder  
% from the trained autoencoder to generate the features.
feat1 = encode(autoenc1,xTrainImages);

% Train second autoencoder.
hiddenSize2 = 50;
autoenc2 = trainAutoencoder(feat1,hiddenSize2, ...  
'MaxEpochs',100, ...  
'L2WeightRegularization',0.002, ...  
'SparsityRegularization',4, ...  
'SparsityProportion',0.1, ...  
'ScaleData', false);

% Extract a second set of features by passing the previous set  
% through the encoder from the second autoencoder.
feat2 = encode(autoenc2,feat1);
% Now train the final softmax layer to classify the 50 dimensional feature vectors. Unlike the autoencoders, you train the softmax layer in a supervised fashion using labels for the training data.
softnet = trainSoftmaxLayer(feat2,tTrain,'MaxEpochs',400);

% Stack the autoencoders with the softmax layer to form a deep network.
deepnet = stack(autoenc1,autoenc2,softnet);
view(deepnet)

% Run the deep network on the testing data.
% Need to turn the test images into vectors and put them in a matrix.
xTest = zeros(28*28,length(xTestImages));
for i = 1:length(xTestImages)
    xTest(:,i) = xTestImages{i}(:);
end
y = deepnet(xTest); % Get scores
figure, plotconfusion(tTest,y); % Plot confusion matrix

% The results can be improved by performing backpropagation on the whole multilayer network. This process is called fine tuning.
% Need to turn the training images into vectors and put them in a matrix.
xTrain = zeros(28*28,length(xTrainImages));
for i = 1:length(xTrainImages)
    xTrain(:,i) = xTrainImages{i}(:);
end
% Perform fine tuning
deepnet = train(deepnet,xTrain,tTrain);
toc

% Run network on test data again.
y = deepnet(xTest); % Get scores
figure, plotconfusion(tTest,y); % Plot confusion matrix
Transfer Learning

• Once we have a trained deep network, we can apply it to new tasks, on similar data
• We’ll use the same hidden layers and just train the final output layer
• This will work fine if the data in the new task has similar features
• This saves a lot of time!