Ministry of Education and Science of the Russian Federation
Peter the Great St. Petersburg State Polytechnic University
Institute of Computer Science and Control Systems
Control Systems and Technology Department

Final Project Report

Language Identification Using Neural Network Knowledge Bases
Part 2, Program Development

Discipline: Neuro-Informatics and Neuro-Technologies
22 December 2016

Student Group: 13541/8

Christopher W. Blake

Professor

Shkodyrev V.P.

St. Petersburg
2016

Christopher W. Blake

Contents

Christopher W. Blake

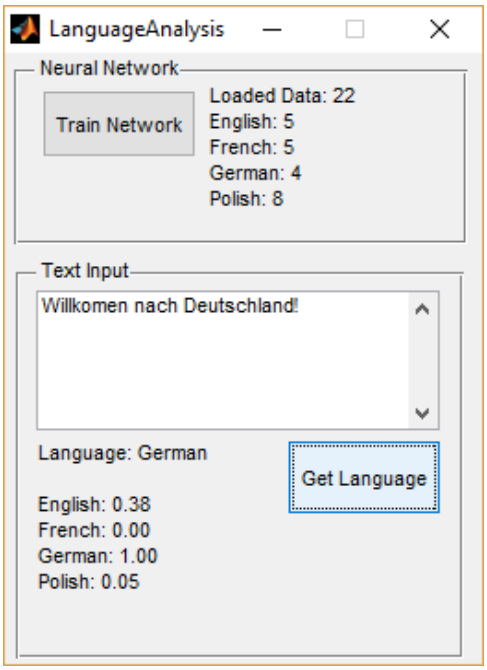
Introduction

In part 1 of this report, neural networks and knowledge bases were reviewed and compared which explain the decisions made for the sample program. Using this information, the most appropriate neural network and knowledge base are developed and utilized in a program for the identification of languages.

This sample program allows the user to import sample text files of various languages and automatically train a neural network. Additional languages and sample data can be added to the program by simply adding text files to the folder. After training of the neural network, the user can then enter any desired text into a textbox and click the "Get Language" button. Pressing this button will use the network and generate a comparison of the text to all available languages, selecting the most similar as the answer.

Program Overview

After researching about various neural networks, a sample program was developed to demonstrate the newly learned principles. This program attempts to analyze a user's text and identify the language.



The chosen software for this implementation is Matlab, because of the built in Neural Network Toolbox. This toolbox provides multiple network types as well as multiple training methods. Hence it is one of the easier systems for software prototyping.

Program Usage

1. Ensure sample documents are stored in the "TrainingData" folder. Each language is a separate folder.
2. Click the **Train Network** button. All documents will be loaded and scanned. The network will then be automatically trained.
3. **Enter text** in any of the trained languages.
4. Click **Get Language**. The text will be analyzed using the trained network.
5. View the **results**. The most likely language will be displayed below the text box. Additionally, the similarity of the text to each language is displayed.

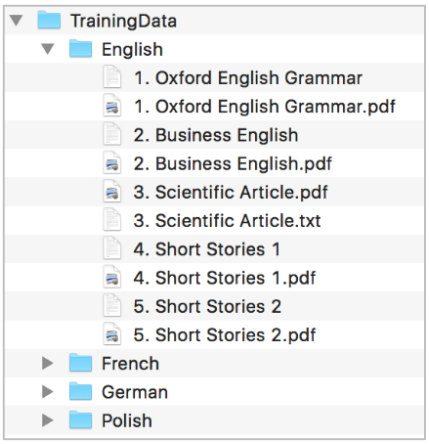
Christopher W. Blake

Development Details

As discussed in part 2 of this project, various knowledge bases, network types, and node configurations were explored. Below is a summary of the decisions that were made.

Knowledge Base and Rules

This program runs on samples of each language. As the nature of language is to be stored in text, the primary storage method of these facts will be in text files. However the language of each text file will be stored in the hierarchy of the folder structure.



Folder Hierarchy

The "TrainingData" folder is the container for all knowledge used by the program. A separate folder is created the name of the language. The language files are placed in this folder. The order and name of these files do not matter.

Language Files

The core data used for the neural network is a series of randomly obtained PDF documents for each desired language to be identified. These documents are manually converted to text files then scanned for the percentage occurrence of each character according to their ASCII code.

File Analysis

The primary input to the neural network is the percentage occurrence of each character per document. Below are the steps used for this analysis.

Steps for character analysis

1. Pick the next available language folder.
2. Open a text file - Read each character into an array.
3. Convert array of characters to ASCII codes.
4. Count the occurrence of each ASCII code.
5. Convert the ASCII count into percentage.

These percentages are then submitted to the neural network, which identifies the likelihood of each language. Note: Because ASCII codes only cover roman characters, languages such as Russian or Chinese cannot be identified. This is a limitation of the rule engine.

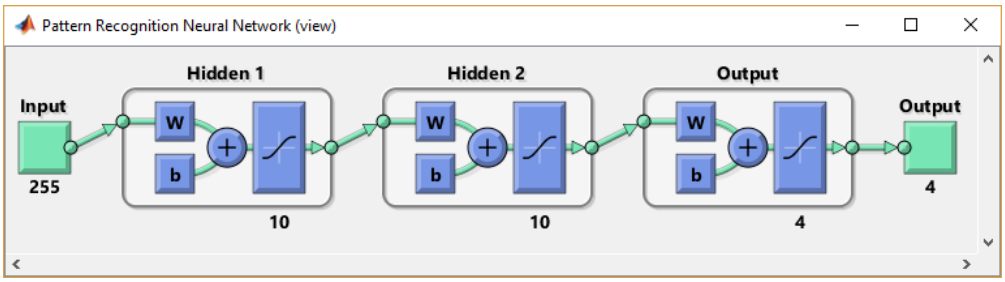
DEC	HEX	Simbolo	DEC	HEX	Simbolo
64	40h	@	96	60h	`
65	41h	A	97	61h	a
66	42h	B	98	62h	b
67	43h	C	99	63h	c
68	44h	D	100	64h	d
69	45h	E	101	65h	e
70	46h	F	102	66h	f
71	47h	G	103	67h	g
72	48h	H	104	68h	h
73	49h	I	105	69h	i
74	4Ah	J	106	6Ah	j
75	4Bh	K	107	6Bh	k
76	4Ch	L	108	6Ch	l
77	4Dh	M	109	6Dh	m
78	4Eh	N	110	6Eh	n
79	4Fh	O	111	6Fh	o
80	50h	P	112	70h	p
81	51h	Q	113	71h	q
82	52h	R	114	72h	r
83	53h	S	115	73h	s
84	54h	T	116	74h	t
129	81h	u	161	A1h	¡
130	82h	é	162	A2h	ó
131	83h	â	163	A3h	ú
132	84h	ä	164	A4h	ñ
133	85h	à	165	A5h	Ñ
134	86h	å	166	A6h	ª
135	87h	ç	167	A7h	º

Network Type

In the matlab package, there exists a modified feedforward network is utilized. The Neural Network Toolbox contains various different network types tuned for different tasks. Because language identification is a classification task, the "patternnet" network type is used. However, the feedforward network is still available as a possibility. This patternet network takes a series of inputs and targets. Using the various input parameters, a vector space out of 0 and 1 is produced. A value of 0 represents that the input is not similar at all to this classification while a value of 1 represents that it is a match.

Node Configuration

The program uses ASCII code identification of each character. Hence, there are 255 inputs. For the hidden layers, various configurations were attempted. However, because of the known "exclusive or" problem by single layer networks, a multi-layer network is utilized. The network in this sample program utilizes 2 hidden layers of 10 nodes each. Finally, the current version of the program identifies 4 different languages, hence there are 4 output nodes. If the number of languages changes, the number of output nodes will also change accordingly.



Semi-Automated Testing

The previously discussed program has extended to include a testing class. This class is used to automatically run various predefined selections of text and report the results. These results include the network training, final outputs, and various plots. Additionally, this script allows the changing of various parameters, shown below. As this can be a large study alone, it will be the focus of the third part of this report.

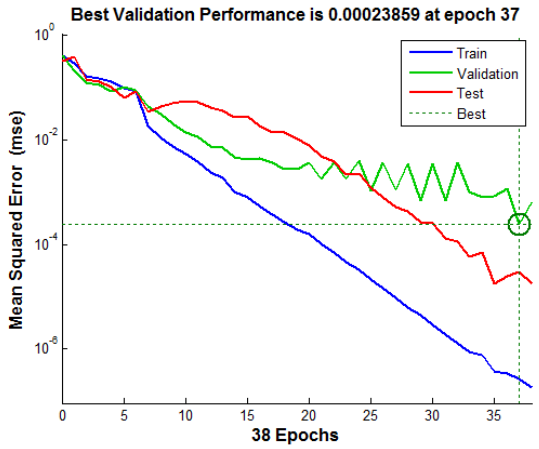
Available Testing Parameters

1. **Training Data Generation Method** (Document Scanning Method)
    a. One training sample per file.
    b. Multiple training samples per file (split by specified character count).
2. **Training Data to Target Generation**
    a. Report target languages as ID values (ie 1,2,3,4,…).
    b. Report target values as a vector of binary values (ie [1 0 0 0], [0 1 0 0], …).
3. **Network Type**
    a. Patternnet
    b. Feedforward
4. **Node Configuration**
    a. Multiple hidden layers.
    b. Number of Nodes per layer.
5. **Training Method**
    a. Backpropagation (using Gradient Descent)
        i. BFGS quasi-Newton backpropagation.
        ii. Conjugate gradient backpropagation with Powell-Beale restarts.
        iii. Conjugate gradient backpropagation with Fletcher-Reeves updates.
        iv. Conjugate gradient backpropagation with Polak-Ribiere updates.
        v. Gradient descent backpropagation.
        vi. Gradient descent with adaptive lr backpropagation.
        vii. Gradient descent with momentum.
        viii. Gradient descent w/momentum & adaptive lr backpropagation.
        ix. One step secant backpropagation
        x. RPROP backpropagation
        xi. Scaled conjugate gradient backpropagation
    b. Supervised
        i. Batch training with weight & bias learning rules.
        ii. Cyclical order weight/bias training.
        iii. Random order weight/bias training.
        iv. Sequential order weight/bias training.
    c. Unsupervised
        i. Batch training with weight & bias learning rules.
        ii. Random order weight/bias training.
6. **Performance Function**
    a. Mean absolute error performance function.
    b. Mean squared error performance function.
    c. Sum absolute error performance function.
    d. Sum squared error performance function.

Results: Network Training

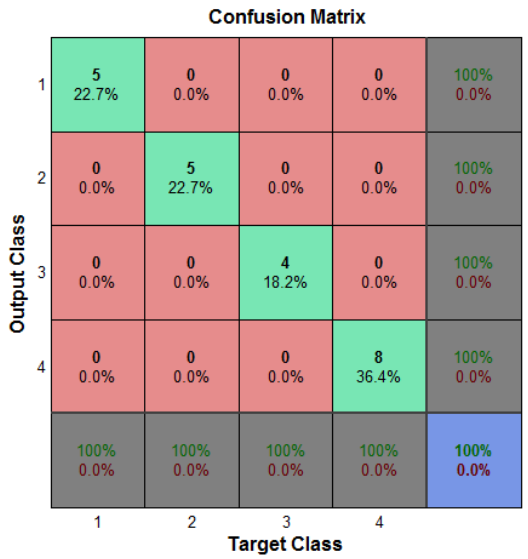
Using the testing class described earlier, one configuration was chosen and testing. This is to show example reports that can be used in part 3 of this report.

The below charts represent the processing results and training result of the neural network.



MSE vs Epochs

It can be seen that during the training, 37 epochs were needed to achieve the required accuracy. However this same accuracy is not properly reflected in validation and testing results.

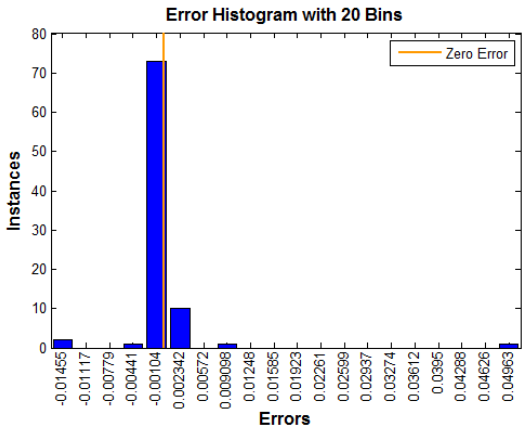


Confusion Matrix

The confusion matrix represents the tendency of the network to confuse one classification with another.

It can be seen in this matrix that for the given inputs and outputs, there is a perfect correlation.

In this example each language has been given an ID number:
1=English,2=French,3=German,4=Polish



Error Histogram

error = outputs - targets

The following histogram shows a comparison of the calculated outputs in comparison to the original targets.

Christopher W. Blake

Results: Language Identification

A series of sentences is used for each language. Both the combined text and each individual sentence is analyzed. The similarity results (0 = dissimilar, 1 = similar) are shown for each input text, and the greatest similarity values is highlighted. It can be seen that in each case the trained network is usually able to predict the correct language. However the network sometimes makes mistakes. Hence, more adjustment to the network is required.

English

Input Text	English	French	German	Polish
Hello my friend. Welcome to Germany. Good Day. This is an example sentence in English. Here there are many options in order to find a good life.	0.9988	0.001	0.7144	0.000
Hello my friend. Welcome to Germany.	0.9999	0.001	0.9910	0.000
Good Day. This is an example sentence in English	0.0056	0.0027	0.9994	0.0158
Here there are many options in order to find a good life.	0.9985	0.0001	0.3249	0.0000

French

Input Text	English	French	German	Polish
Bonjour, mon ami. Bienvenue en France. Bonne journée. Cest une phrase en anglais. Ici, il ya beaucoup doptions pour trouver une bonne vie.	0.001	0.9881	0.0034	0.0102
Bonjour, mon ami. Bienvenue en France.	0.004	0.0380	1.000	0.0025
Bonne journée. Cest une phrase en anglais.	0.0001	0.9974	0.0005	0.0052
Ici, il ya beaucoup doptions pour trouver une bonne vie.	0.0155	0.9671	0.001	0.0016

German

Input Text	English	French	German	Polish
Hallo mein Freund. Wilkommen nach Deutschland. Guten Tag. Das ist ein Beispiel Satz auf Deutsch. Hier gibt es viele Optionen, um ein gutes Leben zu finden!	0.7795	0.002	0.9985	0.001
Hallo mein Freund. Wilkommen nach Deutschland.	0.0094	0.0015	0.9994	0.0002
Guten Tag. Das ist ein Beispiel Satz auf Deutsch.	0.0002	0.0184	0.8469	0.0004
Hier gibt es viele Optionen, um ein gutes Leben zu finden!	0.9972	0.0001	0.3101	0.0000

Polish

Input Text	English	French	German	Polish
Czesc moj przyjacielu. Witamy w Polsce. Dzien dobry. To jest przyklad zdania po polsku. Jest tu duzo mozliwosci, by miec tu piekne zycie!	0.0007	0.0101	0.0001	0.9973
Czesc moj przyjacielu. Witamy w Polsce	0.0003	0.0074	0.0001	0.9887
Dzien dobry. To jest przyklad zdania po polsku.	0.0084	0.0000	0.7432	0.9836
Jest tu duzo mozliwosci, by miec tu piekne zycie!	0.0392	0.9053	0.0001	0.0527

Conclusion

Having learned the fundamentals of Neural Networks and Knowledge Bases in part 1 of this report, a sample program was developed using the Neural Network Toolbox in Matlab. Additionally, a test class was created to allow for further improvement in a semi-automated manor. These programs work together to build the best program possible, which will be explored in part 3 of this report. This is done by utilizing knowledge about Data Collection, Input/Target Preparation, Network Type, Node Configuration, Training Method, and Performance Function. By utilizing this knowledge, it makes the user-version program the most likely to work properly.

The testing class progam is able to successfully identify text from the languages of English, French, German, and Polish in most cases. However, different configurations of the network or additional data samples are likely necessary because all samples failed 1 out of 4 times.

Language	Training Data Sets	Example Input Texts	Number Correctly Identified	Number Incorrectly Identified
English	5	4	3	1
French	5	4	3	1
German	4	4	3	1
Polish	8	4	3	1

References

1. Kasabov, N., & Song, Q. (march 1999). Dynamic Evolving Fuzzy Neural Networks with 'm-out-of-n' Activation Nodes for On-line Adaptive Systems. The Information Science Discussion Paper Series, 99(04). Retrieved December 18, 2016, from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.5.8527&rep=rep1&type=pdf
2. Mehrotra, D (1997). Elements of Artificial Neural Networks.Boston: MIT Press.
3. Principe, J. C., Euliano, N. R., & Lefebvre, W. C. (2000). Neural and adaptive systems: fundamentals through simulation. New York: Wiley.
4. Schmidhuber, J. (1990). A local learning algorithm for dynamic feedforward and recurrent networks. München: Inst. für Informatik.
5. Skapura, D. M. (1996). Building neural networks. New York, NY: ACM Press.
6. Artificial Neural Network. (n.d.). Retrieved December 18, 2016, from https://en.wikipedia.org/wiki/Artificial_neural_network
7. Neural network models (supervised). (n.d.). Retrieved December 19, 2016, from http://scikit-learn.org/stable/modules/neural_networks_supervised.html
8. Neural Network Toolbox. (n.d.). Retrieved December 18, 2016, from https://www.mathworks.com/products/neural-network.html
9. Ontology (n.d.). Retrieved December 20, 2016, from https://en.wikipedia.org/wiki/Ontology
10. Types of artificial neural networks. (n.d.). Retrieved December 19, 2016, from https://en.wikipedia.org/wiki/Types_of_artificial_neural_networks

Appendix 1 – Matlab Code

Class – KnowledgeClass

```
classdef KnowledgeClass
    methods (Static)

        %Counts all ASCII characters in a text file
        function charInfo = ScanDocument_old(fileAddress)
            this = KnowledgeClass;

            %Load text from file
            theText = fileread(fileAddress);

            %Scan document text
            charInfo = this.ScanText(theText);
        end

        %Scans a string for character percentages
        function charInfo = ScanText(theText)
            %Convert to ascii codes
            %slCharacterEncoding('UTF-8')
            %slCharacterEncoding('Windows-1252')
            theText = uint16(theText);

            %Count each character by ascii code
            charInfo(255) = 0;
            for c = 65:255
               charInfo(c) = length(strfind(theText, c));
            end

            %Adjust for percentage
            theSum = sum(charInfo);
            for c = 1:255
               charInfo(c) = charInfo(c) /theSum * 100;
            end

            %Display in one column
            %charInfo = charInfo';
        end

        %Gets information for each file
        function dataSets = ScanDocuments(folder)
            this = KnowledgeClass;

            %Get Languages
            files = dir(folder);
            files = files(3:length(files)); % ignore '.' and '..'
            dirFlags = [files.isdir];
            languageFolders = files(dirFlags);

            %Go through each folder
            dataSetCounter = 0;
            for langID = 1:length(languageFolders)
                %Get the language
                theLanguage = languageFolders(langID).name;

                %Scan the language folder for text files
                theFiles = dir ([folder '\' theLanguage '\*.txt']);

                %Scan each file in the folder
                for t = 1:length(theFiles);
                    dataSetCounter = dataSetCounter + 1;

                    %Get the filename
                    theFileName = theFiles(t).name;

                    %Store the language
                    dataSets(dataSetCounter).language = theLanguage;
                    dataSets(dataSetCounter).languageID = langID;

                    %Store the file name
                    dataSets(dataSetCounter).fileName = theFileName;

                    %Read the text file
                    fileAddress = [folder '\' theLanguage '\' theFileName];
                    theText = fileread(fileAddress);;

                    %Store the file character details
                    dataSets(dataSetCounter).charInfo = this.ScanText(theText);
                end
            end
        end
        function dataSets = ScanDocumentsByPiece(folder, pieceLength)
            this = KnowledgeClass;

            %Get Languages
            files = dir(folder);
            files = files(3:length(files)); % ignore '.' and '..'
            dirFlags = [files.isdir];
            languageFolders = files(dirFlags);

            %Go through each folder
            dataSetCounter = 0;
```

```
            for langID = 1:length(languageFolders)
                %Get the language
                theLanguage = languageFolders(langID).name;

                %Scan the language folder for text files
                theFiles = dir ([folder '\' theLanguage '\*.txt']);

                %Scan each file in the folder
                for t = 1:length(theFiles);


                    %Get the filename
                    theFileName = theFiles(t).name;

                    %Read the text file
                    fileAddress = [folder '\' theLanguage '\' theFileName];
                    theText = fileread(fileAddress);

                    %Split the file up into pieces and scan each piece
                    interval = pieceLength;
                    for i=[1:interval:length(theText)-interval]
                        dataSetCounter = dataSetCounter + 1;

                        %Store the language
                        dataSets(dataSetCounter).language = theLanguage;
                        dataSets(dataSetCounter).languageID = langID;

                        %Store the file name
                        dataSets(dataSetCounter).fileName = theFileName;

                        %Take a piece of the text file to be a dataset
                        textPiece = theText([i:i+interval]);

                        %Store the file character details
                        dataSets(dataSetCounter).charInfo = this.ScanText(textPiece);
                    end
                end
            end
        end

        %Reformats the dataset into the correct structure for
        %a neural network
        function [inputs, targets] = getTrainingData(dataSets)
            this = KnowledgeClass;
            %Get first line
            inputs = dataSets(1).charInfo;
            targets = dataSets(1).languageID;
            for i = 2: length(dataSets)
                inputs = [inputs; dataSets(i).charInfo];
                targets = [targets; dataSets(i).languageID];
            end

            %Transpose to columns for NN training
            inputs = inputs';
            targets = targets';

        end
        function [inputs, targets] = getTrainingDataNodal(dataSets)
            this = KnowledgeClass;
            %Use original method
            [inputs, targets] = this.getTrainingData(dataSets);

            %Convert from numbers to nodes
            countIDs = max(targets);
            for i=1:1:length(targets)
               target(i,countIDs) = 0;
               target(i, targets(i)) = 1;
            end
            targets = target';
        end

        %Converts a nodal result to a language ID
        %Example [0 0 1 0] ==> 3
        function languageID = getLanguageID(nodalResult)
            languageID = find(nodalResult==max(nodalResult));
        end
    end

end
```

UserInterface

```
function varargout = UserInterface(varargin)
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @UserInterface_OpeningFcn, ...
                   'gui_OutputFcn',  @UserInterface_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
function UserInterface_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
function varargout = UserInterface_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
% ABOVE HERE --- DO NOT EDIT ---

% BELOW HERE --- YOU CAN EDIT ---
function buttonTrain_Callback(hObject, eventdata, handles)
% --- Executes on button press in buttonTrain.
kc = KnowledgeClass;
%Create same randomization seed point.
setdemorandstream(491218383)
%Load data
dataSets = kc.ScanDocuments('KnowledgeBase');
countDataSets = num2str(length(dataSets));
[inputs, targets] = kc.getTrainingDataNodal(dataSets);
handles.Languages = unique({dataSets(:).language})';
%Identify unique languages
x = [dataSets(:).languageID];
LanguagesIDs = unique(x);
countLanguages = num2str(histc(x(:), LanguagesIDs));
%Count how many samples of each language are present
for i=1:length(handles.Languages)
   infoLanguages(i) = strcat(char(handles.Languages(i)), {': '}, countLanguages(i));
end
%Update info box;
set(handles.textboxLoadedData, 'String', [['Loaded Data: ', countDataSets]; infoLanguages']);

%Create the network
net = patternnet([10, 10]);
net.trainParam.showWindow=0; %prevent training window from popping up

%Train
[net, tr] = train(net,inputs, targets);

%Update handles
handles.LanguageNet = net;
set(handles.textboxInput, 'Enable','on');
guidata(hObject,handles)

%%%%%%%%%%%%%%%%%%%
%%%%% BUTTONS %%%%%
function textboxInput_Callback(hObject, eventdata, handles)
    checkLanguage(hObject, eventdata, handles);
function buttonCheckLanguage_Callback(hObject, eventdata, handles)
    checkLanguage(hObject, eventdata, handles);
function checkLanguage(hObject, eventdata, handles)
kc = KnowledgeClass;

%Get submitted string
inputString = get(handles.textboxInput, 'String');
combinedString = '';
for r=1:size(inputString)
    combinedString = [combinedString, inputString(r,:)];
end
inputString = combinedString;

%Analyze string
inputCharInfo = kc.ScanText(inputString)';
output = handles.LanguageNet(inputCharInfo);

%Compare number to languages
languageID = kc.getLanguageID(output);
language = char(handles.Languages(languageID));

%Show result and weight
text = {};
for i=1:length(output)
    line = [char(handles.Languages(i)), ': ', num2str(output(i), '%0.2f')];
    text = [text(:);line];
end
set(handles.textboxResult, 'String', [['Language: ', language]; {''}; text(:)])
```

Testing

```
% Final Project:
%   Neural Network Language Detection
% Written By:
%   Christopher Blake
% Date:
%   20 December, 2016
%clear
%clc
%clear inputs;
%clear targets;
close all;
showTrainingWindow = 0;

% Access to the KnowledgeBase functions
kc = KnowledgeClass;

% Create same randomization seed point.
setdemorandstream(491218383)

%% Part 1 - Load Sample Data and convert into inputs and targets

    % Load sample data from 'KnowledgeBase' folder
    if (exist('inputs', 'var'))
    else
        dataSets = kc.ScanDocuments('KnowledgeBase');              %Load by entire document
        %dataSets = kc.ScanDocumentsByPiece('KnowledgeBase', 2000);   %Load by piece (number of characters)
        [inputs, targets] = kc.getTrainingDataNodal(dataSets);
    end

%% Part 2 - Create the Network

    % Pick hidden layers
    hiddenLayerSize = [10, 10];
    net = patternnet(hiddenLayerSize);

%% Part 3 - Decide how to use the training data

    % Data for training, validation, and testing
    % For a list of all data division functions type: help nndivide
    net.divideFcn = 'dividerand';  % Divide data randomly
    net.divideMode = 'sample';  % Divide up every sample
    net.divideParam.trainRatio = 70/100;
    net.divideParam.valRatio = 15/100;
    net.divideParam.testRatio = 15/100;

%% Part 4 - Choose Training Method

    %net.MaxEpoch = 1000;

    % For a list of all training functions type: help nntrain

    % Backpropagation training functions that use gradient derivatives
    %net.trainFcn = 'trainbfg';%  - BFGS quasi-Newton backpropagation.
    %net.trainFcn = 'traincgb';%  - Conjugate gradient backpropagation with Powell-Beale restarts.
    %net.trainFcn = 'traincgf';%  - Conjugate gradient backpropagation with Fletcher-Reeves updates.
    %net.trainFcn = 'traincgp';%  - Conjugate gradient backpropagation with Polak-Ribiere updates.
    %net.trainFcn = 'traingd';%   - Gradient descent backpropagation.
    %net.trainFcn = 'traingda';%  - Gradient descent with adaptive lr backpropagation.
    %net.trainFcn = 'traingdm';%  - Gradient descent with momentum.
    %net.trainFcn = 'traingdx';%  - Gradient descent w/momentum & adaptive lr backpropagation.
    %net.trainFcn = 'trainoss';%  - One step secant backpropagation.
    %net.trainFcn = 'trainrp';%   - RPROP backpropagation.
    net.trainFcn = 'trainscg';%  - Default: Scaled conjugate gradient backpropagation.

    %  Supervised weight/bias training functions
    %net.trainFcn = 'trainb';%    - Batch training with weight & bias learning rules.
    %net.trainFcn = 'trainc';%    - Cyclical order weight/bias training.
    %net.trainFcn = 'trainr';%    - Random order weight/bias training.
    %net.trainFcn = 'trains';%    - Sequential order weight/bias training.

    %  Unsupervised weight/bias training functions
    %net.trainFcn = 'trainbu';%   - Unsupervised batch training with weight & bias learning rules.
    %net.trainFcn = 'trainru';%   - Unsupervised random order weight/bias training.

%% Part 5 - Choose a Performance and Plot Functions

    % For a list of all performance functions type: help nnperformance
    %net.performFcn = 'mae';  % - Mean absolute error performance function.
    net.performFcn = 'mse';  % - Default: Mean squared error performance function.
    %net.performFcn = 'sae';  % - Sum absolute error performance function.
    %net.performFcn = 'sse';  % - Sum squared error performance function.

    % For a list of all plot functions type: help nnplot
    net.plotFcns = {'plotperform','plottrainstate','ploterrhist','plotregression', 'plotfit'};

%% Part 6 - Train the network
    % Prevent training window from popping up
    net.trainParam.showWindow=showTrainingWindow;

    % Train
    [net, tr] = train(net,inputs, targets);

%% Part 7 - Collect Comparison Data From the Network
    %Comparison of inputs, targets, and outputs
```

```
    outputs = net(inputs);
    errors = gsubtract(targets,outputs);
    performance = perform(net,targets,outputs);

    %Example English Sentences
    e = (net(kc.ScanText('Hello my freind. Welcome to Germany. Good Day. This is an example sentence in English.
Here there are many options in order to find a good life.')'));
    e1 = (net(kc.ScanText('Hello my freind. Welcome to Germany.')'));
    e2 = (net(kc.ScanText('Good Day. This is an example sentence in English.')'));
    e3 = (net(kc.ScanText('Here there are many options in order to find a good life.')'));
    englishResultIDs = [kc.getLanguageID(e),kc.getLanguageID(e1),kc.getLanguageID(e2),kc.getLanguageID(e3)];
    englishSimilarity = [e,e1,e2,e3];

    %Example French Sentences
    f = (net(kc.ScanText('Bonjour, mon ami. Bienvenue en France. Bonne journée. Cest une phrase en anglais. Ici, il
ya beaucoup doptions pour trouver une bonne vie.')'));
    f1 = (net(kc.ScanText('Bonjour, mon ami. Bienvenue en France.')'));
    f2 = (net(kc.ScanText('Bonne journée. Cest une phrase en anglais.')'));
    f3 = (net(kc.ScanText('Ici, il ya beaucoup doptions pour trouver une bonne vie.')'));
    frenchResultIDs = [kc.getLanguageID(f),kc.getLanguageID(f1),kc.getLanguageID(f2),kc.getLanguageID(f3)];
    frenchSimilarity = [f,f1,f2,f3];

    %Example German Sentences
    g = (net(kc.ScanText('Hallo mein Freund. Wilkommen in Deutschland. Guten Tag. Das ist ein Beispiel Satz auf
Deutsch. Hier gibt es viele Optionen, um ein gutes Leben zu finden!')'));
    g1 = (net(kc.ScanText('Hallo mein Freund. Wilkommen in Deutschland.')'));
    g2 = (net(kc.ScanText('Guten Tag. Das ist ein Beispiel Satz auf Deutsch')'));
    g3 = (net(kc.ScanText('Hier gibt es viele Optionen, um ein gutes Leben zu finden!')'));
    germanResultIDs = [kc.getLanguageID(g),kc.getLanguageID(g1),kc.getLanguageID(g2),kc.getLanguageID(g3)];
    germanSimilarity = [g,g1,g2,g3];

    %Example Polish Sentences
    p = (net(kc.ScanText('Czesc moj przyjacielu. Witamy w Polsce. Dzien dobry. To jest przyklad zdania po polsku.
Jest tu duzo mozliwosci, by miec tu piekne zycie!')'));
    p1 = (net(kc.ScanText('Czesc moj przyjacielu. Witamy w Polsce')'));
    p2 = (net(kc.ScanText('Dzien dobry. To jest przyklad zdania po polsku.')'));
    p3 = (net(kc.ScanText('Jest tu duzo mozliwosci, by miec tu piekne zycie!')'));
    polishResultIDs = [kc.getLanguageID(p),kc.getLanguageID(p1),kc.getLanguageID(p2),kc.getLanguageID(p3)];
    polishSimilarity = [p,p1,p2,p3];

%% Part 8 - Show Comparison Results

    %All training result details
    %tr

    % Show the node structure
    %view(net)

    % Results of preselected text
    englishResultIDs        %Which ID it predicted. English=1, French=2, German=3, Polish=4
    %englishSimilarity       %How similar it is to each language. Row 1 is English, Row 2 is French, ...

    frenchResultIDs
    %frenchSimilarity

    germanResultIDs
    %germanSimilarity

    polishResultIDs
    %polishSimilarity

    %Epoch info
    best_epoch = tr.best_epoch
    epochs = length(tr.epoch)

    %Plots
    % figure, plotperform(tr)
    % figure, plottrainstate(tr)
    %plotconfusion(targets,outputs)
    % figure, plotroc(targets,outputs)
    % figure, ploterrhist(errors)
```