Design and Implementation of a Neural Network for Voiced/Unvoiced Classification for a Given System

Kevin Struwe

***Abstract*—** Voiced/unvoiced classification is a task from the field of acoustics to assess the vocal folds' contribution to speech production within a given piece of sound. However, it is a difficult task, commonly approached through means of digital signal processing, which usually delivers subpar results, especially in the transition regions between the two classes. Artificial neural networks deliver results of better quality while being able to be more efficient. This paper provides best practices for the design and the implementation of an artificial neural network approach which is able to achieve better results for this particular problem . It outlines the steps to implement a multi-layer perceptron trained with back-propagation using mini-batch stochastic gradient descent. The implementation was done in Octave/Matlab.

***Keywords*—**Implementation, Linear Predictive Coding, Multilayer Perceptron, Voiced/Unvoiced Classification

I. Introduction

The processing and analysis of speech in general and specifically with voiced/unvoiced (V/UV) classification is a difficult task due to the inherent complexity of speech and the digital processing of it. However, with the current trends in speech recognition through intelligent personal assistants, it becomes more and more important to have good quality systems to execute these tasks. As the term *intelligent* implies, most of these systems work with neural networks with varying degrees of complexity. The literature [1] has already shown an artificial neural network system to perform voiced/unvoiced classification of speech by utilizing coefficients obtained by linear predictive coding (LPC). This paper is an extended version and presents some insights from the development process of the neural network system. It gives more insights into the implementation of the designed system in Octave/Matlab, as these are rarely to be found in any research paper.

The background of the given task is to find a suitable way to do voiced/unvoiced classification in a pre-existing cochlear implant system. To enhance speech recognition, an earlier paper [2] proposed a pitch adaption method in order to set the pitch of given input speech to a fixed value. The actual system already exists and works with linear predictive coding to re-synthesize the speech. However, only speech with an actual pitch need changing. From this situation, this paper explains the development process and the insights in detail.

The first step in developing any system is to take a look at the reference system and its intended deployment-environment.

This work was supported in part by the German Research Foundation (DFG) under Grant GRK 1505.

K. Struwe is with the University of Rostock, Institute of Applied Microelectronics and Computer Engineering, Rostock, 18119, Germany. Email: kevin.struwe@uni-rostock.de.

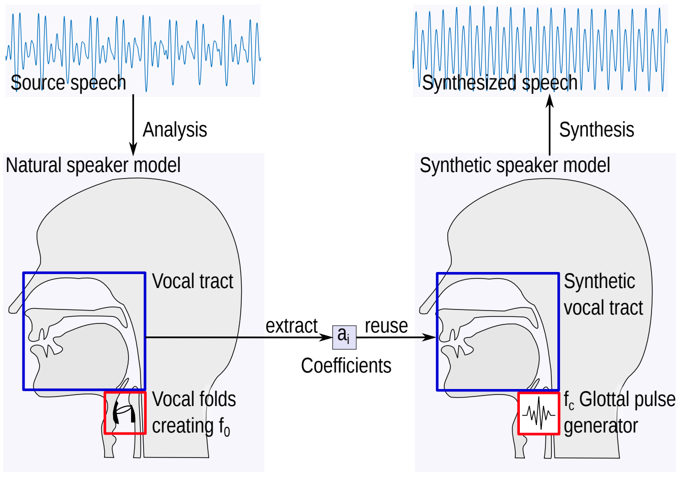

Figure 1. Principle of the linear predictive coding based pitch modification algorithm. This system serves as the environment and reference system for the implementation of the artificial neural network.

For the given scenario, Section II presents the reference system whose functioning principle is shown in Fig. 1. It has already been implemented Octave/Matlab. The reference system delivers the problem that has to be solved. This problem is the lacking functionality in the form of voiced/unvoiced classification, which is examined in Section III-A. Subsequently, information about a possible solution can be extracted from the type of problem, which, in the case of this paper, leads to the conclusion that a neural network suits the purpose just right. Thus, Section III-B explains the basics of how artificial neural networks work. Besides the function of neurons as an opinion amplifier, the general network structure is explained. This structure is then applied to the problem at hand, leading to a number of inputs in the form of linear predictive coding coefficients as explained in Section III-C and an output, which determines the class as concluded from the problem statement. After all preliminary considerations, the actual system's design and implementation is explained in Section IV. To train and test the network, the implementation is based on Octave/Matlab, witch serves as a prototype. However, this prototypical implementation can easily be converted to a better performing language if required.

II. Reference system

The application of the proposed V/UV classification is a cochlear implant pitch modification system. Due to problems with the electrode insertion in the inner ear, cochlear implant patients might not be able to achieve acceptable speech recognition scores due to a misrepresentation of (voice-) pitch. To

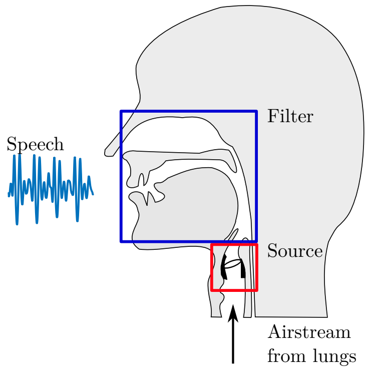

Figure 2. The source-filter model: The airstream from the lungs may or may not excite the vocal folds (larynx), which adds oscillations resulting in a harmonic sound. Subsequently, through the vocal tract, the raw or oscillating airstream is shaped by the filter, whose characteristics are formed by the shape of the vocal tract - resulting in speech.

alleviate this problem, a system for pitch transposition to a beneficial frequency region specific to the patient has been introduced [2].

Since a cochlear implant system has to have a low energy consumption, the algorithms need to be as efficient as possible while serving their purpose. Hence, an linear predictive coding (LPC)-based analysis-synthesis system has been implemented for the pitch modification, as these methods have been determined to be less computationally demanding than other approaches. The approach is to use an LPC analysis stage to determine the coefficients representing vocal tract filter characteristics. A desired synthesis pitch f_c is then used to create a synthetic glottal pulse signal. This signal is used as a harmonic source signal, which is subsequently filtered with the LPC coefficients. This process creates a segment of pitched speech and realizes the source-filter model of speech production (see, also, in section III-C).

Within this reference system the V/UV is one of the most important parts, as it determines which frames need to be modified and which can remain as they are. This is due to the fact that pitch is only present for voiced parts of signal as the vocal folds provide the signal with the harmonic content.

III. Preliminary considerations

A. Voiced Unvoiced Classification

For the detection of the voiced parts of speech one usually speaks of voiced-unvoiced or voiced-unvoiced-silence classification systems. These systems estimate whether a signal portion has periodic content or not. For speech, this means a decision for whether the vocal folds or just the unhindered airsteam from the lungs serve as the source of the sound. This is also reflected in the so-called source-filter model of speech production, which is illustrated in Fig. 2.

The classification difficulty lies in the determination of the threshold between the available features. The thresholds will most likely be different for the various features, some of which are the zero crossing rate (rate of sign change in the waveform representation of the signal), energy or autocorrelation coefficients [3]. It is well known that a low zero crossing rate means that a frame is voiced with a high probability as well as the same goes for a high signal energy. Even from looking at the source signals' spectra it is possible to classify the speech frames. This fact is utilized later on with the linear predictive coding coefficients. However, the exact threshold for when a frame is voiced is hard to pinpoint, as it depends on the signal-to-noise ratio and the actual recording environment as well.

Knowing this, it is not advised to implement yet another bad functioning signal processing approach. Instead, using an artificial neural network to determine the statistics behind the inputs might be a much better idea, since this kind of binary classification problem can be broken down to just a matter of finding the right threshold. Neural networks are able to learn the statistics behind a given input-output dataset and are suitable to find a near-optimal threshold level for many applications, i.e., being a universal approximator (especially feedforward networks [4]). Furthermore, as the V/UV classification problem is binary, this also means that just a single output is enough to decide for a particular class. This makes the neural network easy to design and implement for this application.

B. Artificial Neural Networks

Artificial neural networks (ANN) are the technological abstraction of natural neural networks. They model the ability to represent and categorize multiple classes from a problem set based on their inherent statistics. However, these statistics are not obvious. So the actual strength of neural networks is to determine them through learning and error correction.

An artificial neural network is a collection of so called *neurons* that are interconnected with each other. The neurons are organized in layers similar to Fig. 3. This particular kind of neural network is commonly referred to as *multilayer perceptron* [5], for which three types of neurons exist:

1) input - no incoming connections,
2) hidden - incoming and outgoing connections,
3) output - no outgoing connections.

Each connection between the neurons has a corresponding weight to determine its influence on the input. Besides, each neuron has an activation function $S(x)$, which determines the output (or activation) of the neuron depending on the weight and the inputs of all its incoming connections. An example activation function is the logistic function, which is illustrated in Fig. 4 and is calculated according to the equation

$$S(x) = \frac{1}{1+e^{-x}}. \tag{1}$$

The input neurons contain the features to classify and are arguably no real neurons, since their activation functions have no inputs and their outputs are defined by the feature set. Subsequent hidden layers (more than one are possible) further process the information for the final display at the output neurons.

The execution of an ANN means the calculation of a *forward* calculation. This calculation basically means the evaluation of each activation function with its respective inputs and weights and assigning the corresponding output values accordingly. This is done for each layer. For further details, see, also [6].

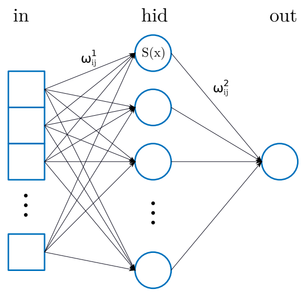

Figure 3. Common structure of a multilayer perceptron with the 3 layers *input*, *hidden* and *output*.

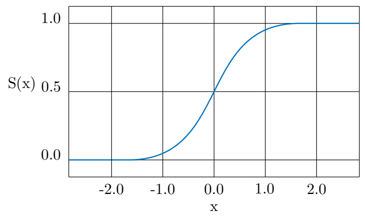

Figure 4. The sigmoidal activation function (also known as logistic function).

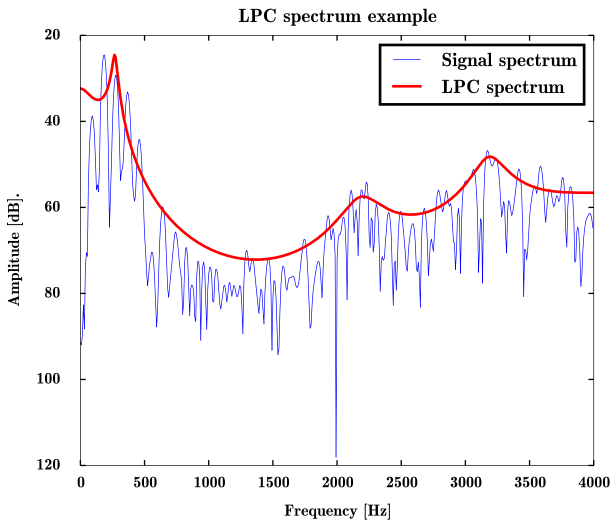

Figure 5. Example of a spectrum obtained through linear predictive coding. The blue graph is the original spectrum, obtained with a size 1024 fast fourier transform. The red graph has been obtained using the LPC-coefficients for this frame.

However, the most important part of using an ANN is the training procedure. One of the most successful training approaches is the back-propagation algorithm. It is an efficient error correcting algorithm devised in 1986 [7]. After a forward calculation, the obtained results are compared to control-results and the weights are corrected according to the slope of the error function (backward-pass). The weight correction works with some variant of gradient descent, which is able to effectively minimize the difference between the calculated network output and a desired control output by estimating the degree to which a weight should be changed in any particular direction in the error topology. For that to work best, it is required to have many training examples to be able to calculate an accurate error gradient later on. However, this minimization can only be local and it is not possible to know whether the algorithm found the optimum. The specifics of this training approach are explained in great detail in [8].

C. Linear Predictive Coding

Knowing to use an ANN for the given problem, and also already knowing what the output should look like, the next step is to determine the input features. Fortunately, the linear predictive coding (LPC) coefficients used in the reference system suit this purpose just fine.

Linear predictive coding estimates the vocal characteristics of a speaker based on a piece of sound [9]. It is widely used for speech analysis and speech recognition systems, since it is fairly easy to compute and delivers reliable results.

LPC is based on the source-filter model of speech production. The principle of this model has been shown in Fig. 2. In a more abstract way to describe speech production, the human vocal tract is seen as a buzzer with an attached tube that has specific filtering properties. The buzzer represents the vocal folds and constitute the *source*. The tube is seen as having different diameters at different points, thus amplifying and attenuating certain frequencies as a *filter*.

From an implementation perspective, LPC tries to predict a future sample $x(j)$ of a signal portion $x(j-i)$ by using the last p samples with $i := [1..p]$. To achieve this goal, LPC employs p coefficients. The number of coefficients p is called the LPC *order*.

The prediction of the next sample $x(j)$ using the coefficients basically means solving the equation

$$x(j) = \sum_{i=1}^{p} a_i x(j-i) + e(j), \tag{2}$$

where a_i are the LPC-coefficients and $x(j)$ is the sample to predict. In other words, $e(j)$ represents the source of the model as residual which has to be minimized, and a_i represents the filter coefficients to tune. To obtain the coefficients for an entire frame, a matrix of samples is created and solved for the coefficients. This process results in the minimization of the residual (that is, the sum of squared distances between predicted and actual samples). For further details, the interested reader is referred to the literature [10].

The coefficients can be used to perfectly recreate the source frame again when using the residual as the source signal. Other than that, it can also be used to create a synthetic signal when a synthetic glottal pulse is used. This is possible, because the LPC-coefficients constitute the parameters to a FIR-filter (finite impulse response), which represents the vocal tract characteristics.

From a signal processing viewpoint, these characteristics are also the envelope of the signal spectrum (i.e., low-pass filtered spectrum), as can be seen in Fig. 5. This means in return that the filter characteristics also contain information about the spectrum contour, which can be used as input for the neural network.

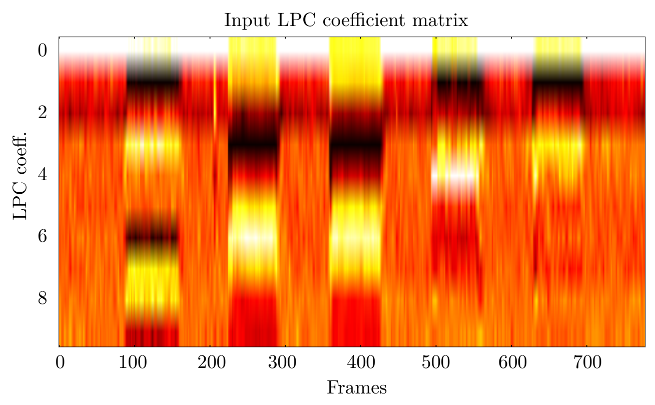

Figure 6. Plot of the LPC coefficients of an audio file. The audio contained the vowels **a,e,i,o,u**. The plot shows an example of the variances between the individual vowels and the ground noise.

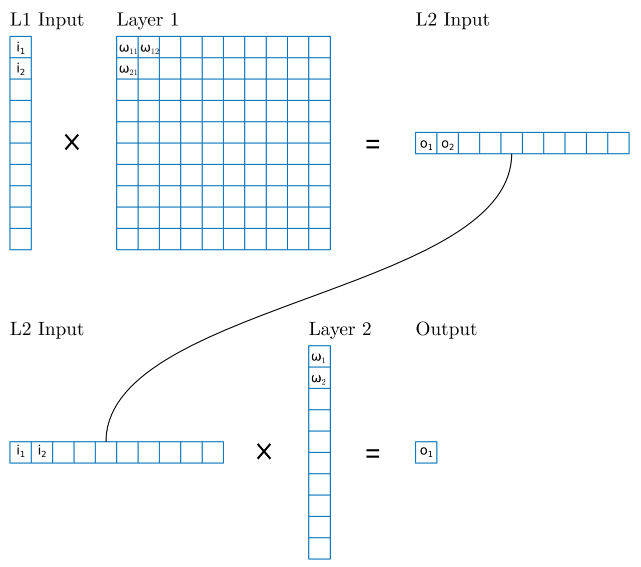

Figure 7. Implemented structure of the voiced/unvoiced classification network.

Again, from the reference system, it is thus possible to infer that the number of inputs should be equal to the order of the LPC analysis.

IV. System Design and Implementation

A. Structure

Since the reference system already uses LPC coefficients for signal modification, these coefficients have been deemed suitable upon evaluation. Recorded coefficient plots have shown the required distinct characteristics for the parts to differentiate. An example plot is shown in Fig. 6

10 LPC coeffients are used as input to the neural network, which are obtained by the order 10 LPC analysis used in the reference system. They have been determined by first segmenting the entire signal into signal frames of 50 ms duration with 10 ms hop length (i.e., 40 ms overlap). Subsequently, the frames were pre-emphasized and windowed, and only then the actual linear predictive analysis was performed. Usually LPC analysis does not need overlapping frames, but further processing requires a certain signal-frame length to not have the calculation time increase too much.

The size of the hidden layer largely depends on the complexity and ambiguousness of the input dataset. If the two classes are very distinct individually, one or two hidden nodes might suffice for a good (binary) classification result. If, however, the two classes are very noisy and are inherently ambiguous as it is the case for the given problem, many more neurons might be required. This topic is covered in a mathematical way in [11], where generic problem classes are considered. Nevertheless, for the given problem, it is suitable to start off testing with a small number of neurons (2-4) and gradually increase the number if the training efforts are fruitless.

Consequently, through experimenting, the network structure consists of a simple multilayer perceptron with 10 input neurons, 10 neurons in the hidden layer and one output. Also, the network utilizes the logistic activation function (sigmoid). Initial testing has shown that two outputs are not necessary as they behave strictly complementary after training.

```
while(mse >= eps)
  % fetch data samples
  l1inp, ctr = get_minibatch(data_in, data_out, Nm)

  % forward step
  l2inp = 1./(1 + exp(-(l1inp * l1w)))
  l2o = 1./(1 + exp(-(l2inp * l2w)))

  % calculate deltas
  l2d = (ctr - l2o).*(l2o.*(1 - l2o))
  l1d = (l2d * l2w') .* (l2inp .* (1 - l2inp))

  % update weights
  l2w += eta * l2inp' * l2d
  l1w += eta * l1inp' * l1d

  mse = mean((ctr-l2o)^2)
end
```

Figure 8. Program code for the neural network training procedure. It uses a back-propagation training approach with mini-batch gradient descent and a learning rate modifier.

B. Implementation

The implementation follows the developed structure. The basic principle of implementing an ANN in the form of a multilayer perceptron is to implement the different layers in the form of matrices. An implemented version of the network is shown in Fig. 7 and is explained further as the section advances. In addition, Fig. 8 shows the Octave/Matlab code used to create and train the network.

This structure means for the input data implementation, that 10 neurons, and thus, a size 10 vector is used for the 10 LPC coefficients. However, this is viable just for the actual forward implementation when used in action. During the training of the neural network, it is possible and required to use entire sets of inputs to represent the variation of the input data and the actual control results appropriately. At this point, the implementation deviates from the structure (at least conceptually).

As the entire dataset takes a significant amount of calculation time to complete for an entire set (batch learning) and a

```
function [outp] = nnfwd(l1inp, l1w, l2w)
  l2inp = 1./(1 + exp(-(l1inp * l1w)));
  outp = 1./(1 + exp(-(l2inp * l2w)));
end
```

Figure 9. The forward function of the neural network implemented in Octave/Matlab.

single set for each iteration (online learning) does not account for the variation of the data, so-called *mini-batches* are used for learning and thus, training. Mini-batches are parts of the complete dataset, and might be scrambled beforehand. This kind of data selection ensures a more appropriate error gradient calculation while having low computational costs. In the code, line 3 invokes the creation of the mini-batches out of the input and output datasets. The size of the mini-batch is indicated through the `Nm` argument. It is important to note that it is required to have input as well as output data acquired from an evaluated source to be able to actually train the network.

The actual work the network performs is done in lines 6 and 7. This is the forward step, which also incorporates the layer matrices `l1w` and `l2w`. Basically, these are the core of the neural network - they are modified during training and determine the actual outputs. Each of those two lines execute all neurons for the particular layer by feeding the result of the multiplication of the weights and the inputs into their activation function. Also, the output of the first layer is used as the input of the second. If more layers are used, this scheme is continued. This is how the hierarchical structure of the networks is realized programmatically.

The actual modification and the training of the neural network happens in the lines 10 to 15. These lines implement the following formula:

$$\omega^{n+1} = \omega^{n} - \eta \left(\frac{\partial E}{\partial \omega^{n}} \right) \tag{3}$$

which constitutes the gradient descent with a learning rate η to adjust the weight of the gradient, i.e., to which amount of the error gradient the weight is changed. It becomes obvious that to obtain the gradients for the errors one has to calculate the derivative the activation function. Now it comes in handy to have chosen the activation to be the logistic function, as its derivative can be easily determined to be

$$\frac{d}{dx} S(x) = S(x)(1 - S(x)). \tag{4}$$

This means, it is possible to input the calculated values directly to calculate the error gradient. Back-propagation itself then works exactly like in the forward calculation through feeding the delta values back into the lower layer. Subsequently, the weights are being adjusted accordingly.

After training, just the layer matrices `l1w` and `l2w` are required to classify any suitable input into the learned categories. This allows for a drastically reduced neural network calculation, which is just a forward calculation at this point, which is shown exemplarily in Fig. 9.

C. Hyper-parameters and data shape

The range and shape of the in- and output data as well as the selection of the hyper-parameters is very important. A significant amount of research has gone into the shape and range of the input values. Usually neural networks make use of floating point numbers, which inherently allows for a wide range of values. However, the normalization of input data plays a major role in how well a neural network is trainable and thus, how much time it takes for it to converge. Sola and Sevilla [12] concluded that in any case the order of magnitude of the different input values should match. In the given network design, the inputs have been treated even further to the degree that they fit into the range from -1.0 to 1.0. This allows the weights of the network to have values that are comparable for later analysis if desired.

The output parameters range from 0.0 to 1.0, with 1.0 resembling the voiced class. This is a fairly straightforward design decision, coherent with the model of a binary classification system.

The hyper parameter selection is basically an educated guess based on the pertinent literature. Whether learning rates are used or omitted [13] or efficient mini-batches can be used for even faster calculations [14]; There is no definitive solution to the ideal values to be selected.

D. Training

The training itself was performed with back-propagation using mini-batch gradient descent with a batch size of 20. The training set consisted of 2.300 frames recorded from two speakers (1 male, 1 female) recorded with a sampling rate of 8.000 Hz. Subsequently, the LPC-coefficients were calculated according to the outlined method and scaled to lie within the interval of $[-1, 1]$. The control output values were hand-annotated using spectrograms for reference. To cover a variety of possible coefficient constellations, the training audio consisted of a set of phonemes instead of full sentences.

The network was fitted to an overall mean square error $E_{MSE} < 0.01$ and with an initial learning rate of $\eta = 0.05$. It was very prone to overfitting, which could easily be seen when training with a lower mean square error (e.g., $E_{MSE} < 0.001$). Fitting to a lower error lead to the evaluation error rates drastically increasing.

The classification error after training was about 3.17 % using all 2.300 frames.

V. Concluding remarks

This paper described a practical method for designing and implementing a neural network for the purpose of voiced/unvoiced classification. Although the stated methods can be used as a practical guide, it is always required to adjust the given tools to the problem, not the other way around. The resulting trained neural network is usable in the same problem domain (V/UV classification), but also in different systems, given the input data is equivalent.

Although not all means of optimization for neural networks have been introduced and applied, this paper provides a useful

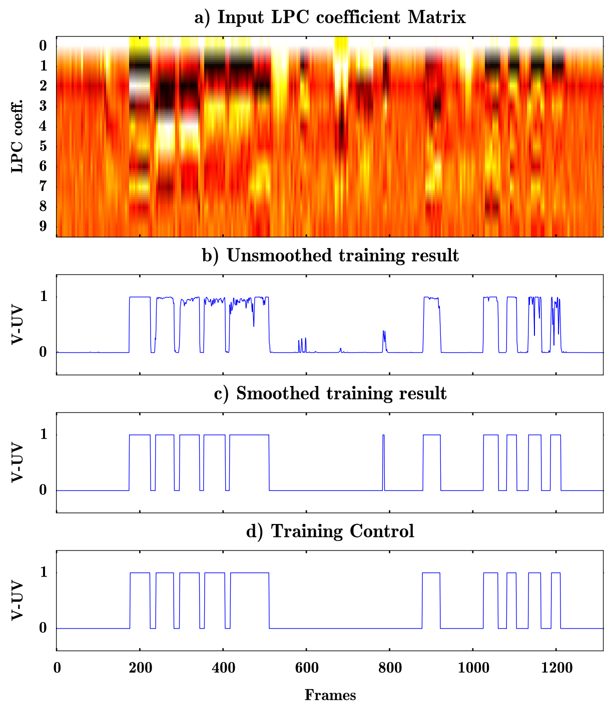

Figure 10. This figure shows the result plots for part of the training dataset. a) displays the 10 LPC coefficients per frame over all available training frames. b) shows the unsmoothed output of the neural network. c) shows the decision based on the output for a threshold of 0.25, smoothed with a size 5 median filter. d) shows the control data for the training set.

implementation guide. The usage of bias neurons has been tested but has not yielded a significant improvement against the presented method. There are always a plethora of ways to approach a problem and here just one of them has been explained.

Acknowledgment

The author thanks the welisa graduate school for its support. Furthermore, thanks are due to prof. Alice E. Smith from Auburn University for her technical advice on neural networks and feature selection. Special thanks are due to prof. Ralf Salomon for editorial assistance and fruitful discussions.

References

[1] K. Struwe, "Voiced-unvoiced classification of speech using a neural network trained with lpc coefficients," in *Control, Artificial Intelligence, Robotics & Optimization (ICCAIRO), 2017 International Conference on*, 2017.

[2] ——, "Apt: Enhanced speech comprehension through adaptive pitch transposition in cochlear implants," in *eHealth 360°*. Springer, 2017, pp. 224–228.

[3] R. Bachu, S. Kopparthi, B. Adapa, and B. Barkana, "Separation of voiced and unvoiced using zero crossing rate and energy of the speech signal," in *American Society for Engineering Education (ASEE) Zone Conference Proceedings*, 2008, pp. 1–7.

[4] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[5] F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," DTIC Document, Tech. Rep., 1961.

[6] L. M. Belue, "Multilayer perceptrons for classification," DTIC Document, Tech. Rep., 1992.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[8] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[9] A. Bundy and L. Wallen, "Linear predictive coding," in *Catalogue of Artificial Intelligence Tools*. Springer, 1984, pp. 61–61.

[10] D. O'Shaughnessy, "Linear predictive coding," *IEEE potentials*, vol. 7, no. 1, pp. 29–32, 1988.

[11] S.-C. Huang and Y.-F. Huang, "Bounds on the number of hidden neurons in multilayer perceptrons," *IEEE transactions on neural networks*, vol. 2, no. 1, pp. 47–55, 1991.

[12] J. Sola and J. Sevilla, "Importance of input data normalization for the application of neural networks to complex industrial problems," *IEEE Transactions on Nuclear Science*, vol. 44, no. 3, pp. 1464–1468, 1997.

[13] T. Schaul, S. Zhang, and Y. LeCun, "No more pesky learning rates." *ICML (3)*, vol. 28, pp. 343–351, 2013.

[14] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient mini-batch training for stochastic optimization," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 661–670.