A bidirectional LSTM network for pitch estimation of speech sounds

Bachelor thesis of Jona Bosman

Jona Bosman
11297670
Supervisor: Paul Boersma
BA Taalwetenschap, University of Amsterdam
16-07-2019

UNIVERSITEIT VAN AMSTERDAM

SUMMARIES

ENGLISH

My thesis presents the construction of a bidirectional long short-term memory neural network for pitch estimation in human speech. Pitch estimation is a challenging task and state-of-the-art methods have not achieved perfection yet. Dealing with octave errors and the voicing decision is part of the task. Candidates for the pitch are generated by the debiased autocorrelation function (debiased ACF) (Boersma 1993) and the network is trained in choosing the correct candidate on a database consisting of recorded speech and the corresponding laryngograph signal (Atake *et al* 2000), which is treated as a ground truth. After training, the network is evaluated against the algorithm currently implemented in Praat. It makes more errors in choosing the correct candidate for the pitch, but fewer errors in its voicing decision than Praat does. It is also discovered that training in batches rather than online leads to slightly better results in this task. The results are promising, especially on the voicing decision, but more research is needed to be able to make the network better than the debiased ACF in Praat.

DUTCH

In mijn scriptie heb ik een bidirectional long short-term memory neuraal netwerk gemaakt om de toonhoogte in menselijke spraak mee te schatten. Het schatten van de toonhoogte is moeilijk en de beste methodes kunnen het nog niet foutloos. Omgaan met octaafsprongen en de stemhebbendheidsbeslissing hoort bij deze taak. Kandidaten voor de toonhoogte worden gegenereerd met de debiased autocorrelatiefunctie (debiased ACF) (Boersma 1993) en het netwerk wordt getraind op het kiezen van de juiste kandidaat met behulp van een database met opgenomen spraak en het bijbehorende laryngograaf signaal (Atake *et al* 2000), dat gebruikt wordt als ground truth. Na het trainen wordt het netwerk vergeleken met het huidige algoritme in Praat en er blijkt dat het netwerk meer fouten maakt bij het kiezen van de juiste kandidaat, maar minder fouten maakt bij de beslissing over de stem dan Praat. Ook bleek dat het trainen van het netwerk in batches hogere resultaten opleverde dan één voor één. De resultaten zijn veelbelovend, vooral op de stemhebbendheidsbeslissing, maar er is meer onderzoek nodig om het netwerk beter te kunnen maken dan de debiased ACF in Praat.

CONTENTS

Debiased autocorrelation and a bidirectional LSTM network for pitch estimation of speech sounds.

J.B. Bosman

University of Amsterdam, 28-06-2019

UNIVERSITEIT VAN AMSTERDAM

ABSTRACT This article presents the construction of a bidirectional long short-term memory neural network for pitch estimation in human speech. Pitch estimation is a challenging task and state-of-the-art methods have not achieved perfection yet. Dealing with octave errors and the voicing decision is part of the task. Candidates for the pitch are generated by the debiased autocorrelation function (debiased ACF) (Boersma 1993) and the network is trained in choosing the correct candidate on a database consisting of recorded speech and the corresponding laryngograph signal (Atake *et al* 2000), which is treated as a ground truth. After training, the network is evaluated against the algorithm currently implemented in Praat. It makes more errors in choosing the correct candidate for the pitch, but fewer errors in its voicing decision than Praat does. It is also discovered that training in batches rather than online leads to slightly better results in this task. The results are promising, especially on the voicing decision, but more research is needed to be able to make the network better than the debiased ACF in Praat.

The code for this work can be found at https://github.com/JonaBenja/pitch_lstm.

I. INTRODUCTION

THEORETICAL FRAMEWORK

Periodic sounds contain a fundamental frequency (F0), which is defined as the number of periods per second of the sound signal. It can be computed by taking the inverse of the period of the sound wave. The term 'pitch' refers to the perceptual quality of a fundamental frequency. In contrast to pure periodic sounds, speech sounds are not periodic. Therefore a fundamental frequency is not present within speech sounds but humans nevertheless do perceive a pitch in these speech sounds. The task of estimating this pitch is challenged by many difficulties.

Pitch estimation has been approached by many researchers over many years and has been referred to as *pitch estimation techniques*, *pitch extraction techniques, pitch* *determinations* or *fundamental frequency estimations.* One of the difficulties of this task is the potential noise in the sound signal. This noise can interfere with the sound signal and as a result the sound waves will be different, which influences the pitch estimation.

Another problem is about the subharmonics of sounds. A sound of 100 Hz has a period of 10 ms, but a period of 20 ms would also fit the signal. In this way, a pitch tracking algorithm could erroneously propose a frequency of 50 Hz instead of the correct 100 Hz. This problem is referred to as an octave error.

A third challenge is the voicing decision. The pitch tracking methods need to decide whether a frame is voiced or voiceless because only in voiced frames a pitch is perceived.

THE PITCH DETECTION METHOD IN PRAAT

A lot of different methods for pitch detection have been proposed (Gerhard 2003). They mainly operate in either the time or frequency domain (Kedem 1986), (Piszczalski & Galler 1979), (Dorken & Nawab 1994), (Flanagan 1965) and (Noll 1967). Other known pitch tracking algorithms are TEMPO (Kawahara*,* Katayose, de Cheveigne & Patterson 1999) and RAPT (Talkin 1995). Regarding the aim of this article (retrieving better results than the algorithm currently implemented in Praat on estimating the pitch of speech sounds in a database), the algorithm in Praat will be described briefly.

The method used in Praat consists of two parts: generating the set of candidates for the possible pitch estimate for each frame in a spoken sentence and choosing the best candidate out of this set. The autocorrelation method is used for the first part.

Autocorrelation is the cross-correlation of a sound with itself. The autocorrelation function is computed from a windowed sound signal and for each local peak, a candidate with a corresponding strength for this windowed sound signal can be computed. Boersma (1993) improved the traditional autocorrelation method (Sondhi 1968) by dividing the autocorrelation function of the sound by the autocorrelation of the window. This results in an autocorrelation signal that is not biased due to windowing and could therefore be called the *debiased autocorrelation function* (debiased ACF). The formula for the autocorrelation and the improvements are shown in formulas 1-6. First, a piece of the speech signal is windowed (1) with a window function (2):

$$a(t) = (x(t_{mid} - \frac{1}{2}T + t) - \mu_x)w_t \tag{1}$$

$$w(t) = \frac{(exp(-12(t/T - \frac{1}{2})^2) - e^{-12})}{(1 - e^{-12})} \tag{2}$$

A piece of the sound $x(t)$ with a duration of the window length, T, is taken from around the middle of the sound (t_{mid}). t is the time in the sound signal and ranges from 0 to the window length T. The signal mean (μ_x) is subtracted from this. Then, the autocorrelation for the windowed sound is computed (3) as a function of the lag time of the shifted signal,

$$r_a(\tau) = r_a(-\tau) = \frac{\int_0^{T-\tau} a(t)a(t+\tau)dt}{\int_0^T a^2(t)dt} \tag{3}$$

after which the autocorrelation for the window itself is computed (4).

$$r_w(\tau) = \frac{\int_0^{T-\tau} w(t)w(t+\tau)dt}{\int_0^T w^2(t)dt} \tag{4}$$

The autocorrelation of both the sound and the window are functions of the time lag τ. To retrieve the final autocorrelation for a windowed signal, the autocorrelation of the signal is divided by the autocorrelation of the window (5)

$$r_x(\tau) \approx \frac{r_a(\tau)}{r_w(\tau)} \tag{5}$$

The strengths of the candidates are computed with the formulas 6 and 7. The strength of the voiceless candidate is computed with formula 6, the strengths of the voiced candidates are computed with formula 7.

$$R = voicingThreshold - \max(0, 2 - \frac{(localpeak)/(globalpeak)}{silenceThreshold/(1+voicingThreshold)}) \quad (6)$$

$$R(\tau_{max}) = r(\tau_{max}) - octaveCost * \log_2(minimumPitch * \tau_{max}) \quad (7)$$

The parameters *silenceThreshold*, v*oicingThreshold*, *octaveCost* and *minimumPitch* are used in Praat to compute the best candidate for each frame. *Localpeak* refers to each local peak in the autocorrelation function and *globalpeak* refers to the global peak in the autocorrelation function. τ_{max} refers to the voiced candidates that correspond to the lag times that have the highest local strength. *Minimumpitch* is the pitch floor which is a parameter that can be set when computing the autocorrelation function. When setting this parameter, no candidates lower than the minimum pitch will be proposed. In Praat, the default value for the pitch floor is 75 Hz.

After the generation of the candidates, the best candidate has to be selected for each frame. In Praat, the dynamic programming technique Viterbi is used (van Alphen & van Bergem 1989) and (Forney 1973). At first it seems that the best candidate for the pitch would be the candidate with the highest strength, but within one sound, this could lead to a discontinuous pitch. The strongest candidate in a frame could be an octave lower or higher than the strongest candidate in the previous and following frames. This phenomenon is called an *octave error.* An attempt to suppress the occurrence of octave errors is to include an *octave jump cost* in the algorithm to compute the pitch contour. This *octave jump cost* is based on the candidates of neighbouring frames. In this way, an octave jump is disfavoured and is less likely to occur in the pitch contour.

Another problem is the voicing decision. Only voiced frames have a pitch. Therefore, to compute the right pitch in a frame, the algorithm has to decide whether the frame is voiced or not. Voiced frames have a stronger periodicity than voiceless frames. The so-called *voicing decision* is the hardest on edges of voiced and unvoiced parts. The algorithm sets a voicing threshold and if all the candidate strengths do not exceed this threshold, the frame is marked voiceless. Like the *octave jump cost*, the algorithm uses a *voiced/unvoiced cost* as well, to disfavour voice changes.

'YIN' PITCH DETECTION METHOD

The autocorrelation correlation is also used by de Cheveigne and Kawahara (2003). They use a basic definition of the autocorrelation function (8) and improve it in steps. The algorithm is called YIN. The last step involves choosing the best local estimate for each frame, which corresponds to choosing the best candidate in the debiased ACF in Praat.

$$r_t(\tau) = \sum_{j=t+1}^{t+W} x_j x_{j+\tau} \tag{8}$$

where τ is the time lag, W is the window size, t is the time index and x is the sound.

de Cheveigne & Kawahara evaluate YIN over a database, using the electroglottograph (EGG) signal recorded alongside as a ground truth to measure the accuracy. YIN was evaluated against other pitch tracking methods, among which the debiased ACF in Praat. Only the voiced frames of the database were used in this evaluation and a gross error was noted when the proposed frequency deviated 20% from the frequency of the ground truth. According to the paper, the debiased ACF in Praat had an error percentage of 2.7% and YIN had an error percentage of 0.30% when tested against the EGG-ground truth, that was computed with the TEMPO algorithm. When the experiment was run again with the EGG-ground truth computed with the system in Praat, the gross error percentage of de debiased ACF was 2.33%. It should be noted that the command Praat was called with was: "*To Pitch (ac): 0.01, 40, 15, "no", 0.0, 0.0, 0.01, 0.0, 0.0, 800*". The default setting for this command in Praat is: "*To Pitch (ac): 0.01, 75, 15, "no", 0.03,* *0.45, 0.01, 0.35, 0.14, 600*". The *octave-jump cost* (0.35 in the default setting) and the *voiced/unvoiced cost* (0.14 in the default setting) were thus put to zero in the YIN-method of de Cheveigne and Kawahara, but the debiased ACF method in Praat needs to use these parameters to achieve good results. A pitch floor of 40 Hz and a pitch ceiling of 800 Hz were used. When the experiment is rerun with the default values for the *octave-jump cost* and *voiced/unvoiced cost* parameters but with the pitch range of 40-800 Hz , the debiased ACF in Praat reaches a gross error percentage of 0.92%. In the paper of de Cheveigne and Kawahara, the sounds in the database were downsampled to 16kHz. The pitch range chosen by de Cheveigne and Kawahara is outside of the usual pitch range of Praat, which is 75-600 Hz. Lastly, the parameter *very accurate* in Praat was set to "no". This all could have prevented the debiased ACF in Praat from performing optimally.

NEURAL NETWORKS

Artificial neural networks have been proposed in the past, but they have gotten a lot of new attention since 2012, when a network created by Krizhevsky, Sutskever and Hinton (2012) won the first place of the annual ImageNet contest. The network performed better than the state-of-the-art methods at that time. Since then, neural networks have been used more and their results have gotten better, especially in the field of Natural Language Processing. Neural networks performed excellent at for example speech recognition, (Hinton *et al* 2012) and machine translation (Wu *et al* 2016).

Neural networks are inspired by the architecture of neurons and connections in the brain. They consist of a number of layers made up of so-called 'neurons' or 'nodes'. These neurons can be activated when their activation threshold is exceeded. By adjusting these activations and thresholds, the network can create a model that predicts at what moments neurons are activated. The first layer of neurons represents the input one can give the network, for example a digital sound sample. The last layer of the network represents the output, for example certain words the speech sound could mean. When the network is provided with a number of speech sounds together with the correct word, it can adjust the model to fit this data and create a model that can classify many sounds as their correct word. This is an example of supervised learning. The more quality data it gets, the better the network becomes.

Neural networks can achieve a much higher accuracy when so-called ‘hidden’ layers are added. These ‘deep’ networks can abstract the input more and extract features that ‘shallow’ networks cannot.

A neural network can be used for pitch estimation. In 1989, a neural network was developed by Barnard, Cole, Vea and Alleva (1989) to classify speech sounds of the TIMIT database as “having a pitch” or “having no pitch”. The input for the network was information about the peaks in a windowed speech signal, combined with information about the local peaks in the same window . The training set for this network consisted of 35.000 training samples. The best result achieved was a 2.0% error rate. Although this article was published long before the neural network revolution described in the section above, its results are promising and show that temporal information is essential in achieving good results in speech research.

RNN NETWORKS

A way of incorporating temporal information (such as sequences of pitch candidates) into neural networks is by using Recurrent Neural Networks (RNN’s). RNN’s receive input samples in a sequence. In generating the output for a given input, these networks use the output information of the preceding input. The use of RNN’s is demonstrated by Ba, Mnih & Kavukcuoglu (2014), who created an RNN network that could recognise house numbers on Google Street View. Gregor, Danihelka, Graves, Rezende and Wierstra (2015) were able to let an RNN generate images of digits. However, the usefulness of RNN’s is challenged by the exploding or vanishing gradient problem, described by Bengio, Simard and Frasconi (1994). The core of this problem is that when the distance between the remembered output and current input are large, the weights and biases of the network get exponentially small or large and will at some point during the training process prevent the network from learning. This problem can be solved by using a special kind of RNN's, namely the Long Short Term Memory network.

LSTM NETWORKS

LSTM networks were first constructed by Hochreiter and Schmidhuber (1997). They consist of multiple so-called LSTM cells. These cells represent the number of steps in the sequence that can be remembered by the layer. Each LSTM cell consists of three gates and a cell state: a forget gate, which decides how much information is remembered from the previous cell state; an input gate, which decides how much of the new information is added to the cell state and an output gate, which decides how much of the cell state is outputted and passed on to the next LSTM cell. In this way, the LSTM layer remains a constant information flow. This is unlike RNN cells that take all the information from their own input and the previous output, add them together and passes them on to the next cell. This can lead to a vanishing or exploding gradient. LSTM layers avoid this problem by the constant flow of information and this makes them very useful for sequence training. LSTM networks have shown to be successful in Sak, Senior and Beaufays (2014).

A disadvantage of the first LSTM layers can be that the information flow is only transferred in one direction: the direction of the given sequence. The current input will then only get information based on the previous input and not on the following input. This can be a disadvantage for some research topics, for example phoneme recognition, where the following phoneme can influence the realisation of the current phoneme.

A great solution for this problem is the use of bidirectional LSTM layers (Shuster & Paliwal 1997). A bidirectional LSTM layer (BLSTM) receives the sequence in the given order, but also reverses it and adds together the outputs of both directions. In this way, future input is treated as past input and the network can use both information sources. BLSTM layers can outperform regular LSTM layers in certain contexts, like in Graves and Schmidhuber (2005) and Graves, Jaitly and Mohammed (2013a). Graves, Jaitly and Mohammed (2013b) discovered that the depth (multiple layers stacked on each other) is more important than the number of cells in each separate layer.

The architecture of a BLSTM layer suits the problem of pitch determination well. It is a sequential problem, since all frames in a sound follow each other, and frames further in the sequence can influence frames earlier in the sequence.

QUESTION AND HYPOTHESIS

This article presents the construction of a bidirectional Long Short Term Memory network for pitch estimation. The candidates for the right pitch are generated by the debiased autocorrelation as currently implemented in Praat (Boersma & Weenink 2019). A bidirectional LSTM network is developed to choose the optimal patch of these candidates to create the best pitch contour. The question that will be investigated in this paper is: does a bidirectional LSTM network select the correct pitch better than the debiased ACF in Praat with regards to the challenges of voicing decisions and octave errors?

II. METHOD

PROCEDURE

Three networks were created in Python version 2.7.10, using Keras version 2.2.4 with a Tensorflow backend (version 1.13.1) on a Macbook Pro 2016 with operating system Mojave 10.14.5. The main goal for the network was choosing the right candidate out of a list of pitch candidates and their strengths. A subtask of this was the voicing decision. The pitch candidates and strengths for each frame were provided as input for the networks and the output was one of the provided candidates per frame.

CORPUS AND GROUND TRUTH

For the training of the networks, a database created by Atake *et al* (2000) was used. This database contains utterances of 14 male and 14 female speakers who each spoke 30 sentences, with a total of 840 sentences and a total duration of 36 minutes. These utterances were recorded along with an electroglottograph (EGG). The EGG was placed on the throat of the participants to measure the opening and closing of the vocal cords. This movement of the vocal cords can be used to retrieve a signal that is more periodical than regular speech recordings, since the signal is not distorted by the mouth, tongue and other body parts used during speaking. Speech is voiced when the vocal cords are closed and unvoiced when the vocal cords are open, so the voice of an utterance can be read from the EGG signal. The higher degree of periodicity makes the EGG signal a sufficient candidate to retrieve a ground truth from, for the right candidate for the pitch of a frame as well as the voice of a frame.

Each sound signal in the database consists of 3 channels: channel 1 contains the speech signal, channel 2 contains the EGG signal and channel 3 contains a voicing mask for the utterance. This voicing mask was retrieved from the EGG signal.

For training the network and measuring its accuracy, a ground truth is needed. The ground truth that was used in this research was computed from the EGG signal using the current algorithm in Praat. For every sentence in the corpus and for every frame in each sentence, the pitch value of the EGG signal that was proposed by Praat was chosen as the ground truth value. For voicing decisions, the voicing mask was used. It was noticed that in some frames the voicing decision of the EGG signal was different from the voicing decision of the voicing mask. This occurred once in roughly half of the sentences, so around 400 times in total. It can probably be explained by the fact that the candidates for the pitch were computed using Praat and the voicing mask was created using another method. To be consistent, the voicing decision of the voicing mask was used in determining the ground truth values. For some frames that were labelled as voiced by the voicing mask, the EGG signal did not have any candidates, and thus were seen as voiceless by the EGG signal. These frames were seen as voiceless as well in the ground truth, since it would otherwise be impossible to get a voiced ground truth value for these frames. The ground truth values were used as training labels for the network and to measure the accuracy of both the current Praat algorithm and the trained neural network.

GENERATION OF THE CANDIDATES

The pitch candidates were computed using the debiased ACF, as currently implemented in Praat. These candidates were generated from the speech signal of the corpus, with the default settings in Praat, that is, with the following command: “*To Pitch (ac): 0.0, 75.0, 15, "yes", 0.03, 0.45, 0.01, 0.35, 0.14, 600.0”.*

For every frame in every sentence of the corpus, a number ranging from 1 to 15 candidates were generated, together with the strengths of these candidates. Frames for which less than 15 candidates were computed, were padded with zeros (0) to make all frames have the same length. Every candidate list contained one voiceless candidate, but this candidate was removed from the input, since the values for the frequency (0) and strength (0) would be the same for every frame and therefore not contribute to the network. Every frame thus consisted of 28 elements: 14 frequency candidates and the corresponding 14 strengths. The parts of the speech signal that were silent were encoded as voiceless. The voicing mask that was included in the database was not used during the generation of the candidates.

MEASURING ACCURACY

The percentage of errors was used as a measure of the accuracy of the network. All wrong decisions of the voice of a frame and all chosen pitch values that deviated more than 20% from the ground truth values were considered errors. This measure was computed over a test set, which contained 20% of the database. The first half of the database were sentences spoken by females and the second half were sentences spoken by males. To prevent a bias for one gender, the test set contained the first 10% and the last 10% of the database, resulting in an equal amount per gender. For the deep network, 20% percent of the total frames in de database was taken as the test set in the same way as described above. For the LSTM network, this was slightly different. All frames in a sentence form a sequence that is fed to the LSTM network. The complete sequence has to be presented to make the network learn from it. It cannot be the case that a part of the sequence is in the training set and another part is in the test set. Therefore it was decided to use 20% of the total number of sentences instead of the number of frames.

NETWORK 1: DEEP NEURAL NETWORK

INPUT

The data contained 212.113 frames, of which 20% (42.422 frames) was used as test data. The training data consisted of the rest of the data (169.691 frames).

OUTPUT

The output of the network was a vector with a prediction for each of the candidates. An extra element for the voiceless candidate was added, making the output vector consist of 15 elements.

ARCHITECTURE

The first network had an input layer of 28 neurons, two deep layers of respectively 28 and 1000 neurons and an output layer of 15 neurons. Both deep layers were activated with a rectified linear activation (9). The input for the first hidden layer of 28 nodes was a matrix of 28 (number of inputs) by 28 (number of nodes in layer). Each element of the input was multiplied by the weights of these nodes ($\mathbf{W}_{\mathbf{d1}}$ a matrix of 28 by 28 weights) and a bias was added ($\mathbf{b}_{\mathbf{d1}}$, a vector of 28 biases). Each element of the vector of activations $\mathbf{z}$, consisting of 28 elements, was activated by the ReLU activation. The input for the second hidden layer of 1000 nodes was a matrix of 28 (output of first layer) by 1000 (number of nodes in layer). The input was multiplied by the weights of this layer ($\mathbf{W}_{\mathbf{d2}}$, a matrix of 1000 by 28 weights) and a bias was added ($\mathbf{b}_{\mathbf{d2}}$, a vector of 1000 biases). The activation of every node was given as input to the ReLU-activation.

$$ReLU(z_i) = \max(z_i, 0) \tag{9}$$

where i ranges from 0 to the number of nodes in the layer that is activated. The output layer was activated with a softmax activation (10), that resulted in a prediction for each output neuron.

$$\hat{p}_i = \frac{e^{\hat{y}_i}}{\sum_{j=1}^{C} e^{\hat{y}_j}} \tag{10}$$

where $\hat{p}_i$ is an output neuron, of which the number ranges from 1 to C. C is the number of classes, which equals the number of output neurons and is 15 in the networks constructed in this paper. $\hat{y}_i$is the activation of the output neuron. This activation, taken to the power of e, is divided by the sum of the activations of all output neurons. The loss function was a categorical cross entropy (11) due to the one-hot encoding of the models output.

$$loss = -\log(\hat{p}_{correct}) \tag{11}$$

Where $\hat{p}_{correct}$ is the softmax prediction of the model for the correct class.The network was optimized with the Adam optimizer, a variant of the Stochastic Gradient Descent optimizer, implemented as described in Kingma and Ba (2014) (see their article for a full description of the algorithm).

<u>TRAINING</u>

The network was trained for 100 epochs. In 1 epoch, all the inputs in the training data are fed to the network in random order and the network adjusts its weight and bias parameters according to the information of the loss function and optimizer. So after 100 epochs, that network has seen all of the input data 100 times. The training data were fed to the network in a batch size of 256, which means that after every 256 inputs, the network updates its weights and biases, with the information of all 256 inputs. This decreases the computational load since updates are done less often than if they are done after every single input. After a series of training sessions, the highest accuracy for the LSTM network was achieved with a batch size of 256, which is why this batch size was chosen.

NETWORK 2: BIDIRECTIONAL LSTM

INPUT

The data contained 840 sentences (212.113 frames), of which 20% (168 sentences, 43.404 frames) was used as test data. The training data consisted of the rest of the data (672 sounds, 168.709) frames). Each sentence, a sequence of multiple frames, was seen as one input. The whole sequences were given to the LSTM network, but the LSTM cells receive one frame at the time.

OUTPUT

The output of the network was a vector with a prediction for each of the candidates. An extra element for the voiceless candidate was added, making the output vector consist of 15 elements.

ARCHITECTURE

The second network that was constructed was a bidirectional LSTM network with an input layer, a masking layer, two bidirectional LSTM layers and an output layer. Both LSTM layers consisted of 96 LSTM cells in parallel. After a series of training sessions, the highest accuracies for the LSTM networks were achieved with LSTM layers consisting of 96 cells, which is why this number of cells was chosen. Each LSTM layer was built according to the following formulas (14-18) using both sigmoid (12) and hyperbolic tangent activation (13). Each gate and the cell state in the LSTM cells has its own weights ($\mathbf{W_f}$, $\mathbf{W_l}$, $\mathbf{W_o}$ and $\mathbf{W_o}$ for the input and $\mathbf{U_f}$, $\mathbf{U_l}$, $\mathbf{U_o}$ and $\mathbf{U_c}$ for the cell state) and biases ($\mathbf{b_f}$, $\mathbf{b_l}$, $\mathbf{b_o}$ and $\mathbf{b_c}$). The input is fed to the LSTM cells one frame at a time. The number k is an index from 1 to the number of steps in the sequence. At every step k, the information in each LSTM cell is updated. The cell state $\mathbf{c}(k)$ is a vector with length 96 that contains the cell state of all 96 LSTM cells at step k. The hidden state $\mathbf{h}(k)$ is a vector of length 96 that contains the output of all 96 LSTM cells at step k. The input vector $\mathbf{z}$ consists of 28 elements (28 candidates per frame). The weights matrices for the input have a dimension of 96 (the number of cells) by 28 (number of inputs) and the weight matrices for the cell state have a dimension of 96 (the number of cells) by 96 (the number of cells). The bias vectors have a length of 96 (the number of cells).

Sigmoid activation:

$$\sigma(v_j) = \frac{1}{1 - e^{-v_j}} \tag{12}$$

Hyperbolic tangent activation:

$$\tanh(v_j) = \frac{e^{v_j} - e^{-v_j}}{e^{v_j} + e^{-v_j}} \tag{13}$$

where v_j is a single element of the vector that is fed to the activation function.

Forget gate:

$$\mathbf{f}(k) = \sigma(\mathbf{W_f z}(k) + \mathbf{b_f} + \mathbf{U_f h}(k-1)) \tag{14}$$

After the multiplication of the inputs with their corresponding weights and the hidden state with its corresponding weights, the two resulting vectors are summed elementwise, along with the biases of this gate. The output of the forget gate is thus a vector of length 96, consisting of a sigmoid number for each element in the cell state. The forget gate determines how much of the cell state will be forgotten. The sigmoid function in this formula is applied elementwise.

Input gate:

$$\mathbf{l}(k) = \sigma(\mathbf{W_l z}(k) + \mathbf{b_f} + \mathbf{U_f h}(k-1)) \tag{15}$$

The input gate determines how much of the current input will be added to the cell state. The computation of this vector is done in the same as in the forget gate. The output of the input gate is also a vector of length 96. The sigmoid function in this formula is applied elementwise.

Output gate:

$$\mathbf{o}(k) = \sigma(\mathbf{W_o z}(k) + \mathbf{b_o} + \mathbf{U_o h(k-1)}) \tag{16}$$

The output gate determines how much of the current cell state will be the output for the current LSTM cell *k*. The computation of this vector is done in the same as in the forget gate and input gate. The output of the output gate is also a vector of length 96. The sigmoid function in this formula is applied elementwise.

Cell state update:

$$\mathbf{c}(k) = \mathbf{f}(k) \odot \mathbf{c}(k-1) + \mathbf{i}(k) \odot \tanh(\mathbf{W_c z}(k) + \mathbf{b_c} + \mathbf{U_c h}(k-1)) \tag{17}$$

where the output of the forget gate (a vector with 96 sigmoid numbers) is multiplied elementwise with the previous cell state. The current cell state is updated and scaled elementwise with a tanh function. Then the output of the input gate (a vector with 96 sigmoid numbers) is multiplied elementwise with the output of the tanh function. The result of this multiplication is summed elementwise with the result of the multiplication of the output of the forget gate with the previous cell state. The tanh function in this formula is applied elementwise.

Output:

$$\mathbf{h}(k) = \mathbf{o}(k) \odot \tanh(\mathbf{c}(k)) \tag{18}$$

The current cell state is scaled elementwise with a tanh function and multiplied elementwise with the output of output gate to get the new hidden state and thus the output of the LSTM layer at the current step *k*. The tanh function in this formula is applied elementwise.

The output layer consisted of 15 nodes for the 15 possible candidates and was activated with a softmax activation. The loss function used was a categorical cross entropy and the optimizer that was used was the Adam optimizer.

The length of all the sounds in de database was not equal; the number of frames in each sound was variable. An LSTM network needs to have inputs of the same length to be trained in batches so this problem needed to be solved before the network could be trained. There were two possible solutions: padding each sound to an equal number of frames or training the network with each batch consisting of one single sound. This means that the weights and biases of the network are updated after every single input. Training the network with a batch size of 1 is usually referred to as 'online training'. It is not clear whether one of these methods performs better than the other, (Wilson & Martinez 2004) and (Principe, Euliano & Lefebvre 2002). Usually, padding is done by adding zeros to the sequence until it reaches the desired length. In this case, the pad value was a list of 28 zeros, representing a frame. Each sound was complemented with these lists until it had a length of 447 frames. By adding a masking layer to the network, the pad frames are ignored during the training of the network and the padding will not trouble the training.

To find the best results possible, the LSTM network was trained in batches and online to be able to compare the two solutions.

TRAINING

The network was trained on the padded data for 115 epochs with a batch size of 256. Using a batch size of 1, the network was trained as well for 115 epochs.

III. RESULTS

After training the networks, they were all evaluated on the test set. The debiased ACF in Praat was evaluated on the same test set.

The accuracies of both methods were measured by comparing every frequency that was chosen from the speech signal with the frequency that was chosen by the EGG signal. Every difference between these frequencies that was larger than 20% was labeled as an error. Every wrong decision about the voice also counted as an error.

The percentage of errors in all frames in the test set was measured to compare the methods.

METHOD	**FREQUENCY ERRORS**	**TOO HIGH / TOO LOW ERRORS**	**VOICING ERRORS**	**VOICED/VOICELESS ERRORS**	**TOTAL ERRORS**
Debiased ACF	0.16%	0.07% / 0.09%	11.31%	9.91% / 1.40%	11.47%
Deep network (batch = 256)	0.37%	0.25% / 0.12%	7.3%	4.2% / 3.1%	7.68%
BLSTM network (batch = 256)	0.58%	0.26% / 0.32%	5.5%	2.53% / 2.97%	6.08%
BLSTM network (batch = 1)	0.84%	0.41% / 0.43%	8.63%	3.11% / 5.52%	9.48%

Table 1: results of the three networks that were trained and the debiased ACF in Praat. All methods were evaluated over a test set with 20% of the database. The column “Too high/too low errors” gives the percentages of the frequency errors that were too high and too low. The column “Voiced/voiceless errors” gives the percentages of the voicing errors that were predicted as voiced but were voiceless in the ground truth and vice versa.

IV. CONCLUSION

A bidirectional Long Short-term Memory network was developed to choose the right pitch for each frame of a speech sound. The network makes fewer errors about the voice of a frame, but more errors about the estimated pitch value than the method currently implemented in Praat. In a comparison between training in batches and online training on the data used in this article, it was discovered that training in batches yielded slightly better results.

V. DISCUSSION

The neural network that was constructed has a lower error percentage on the test set than the current debiased ACF in Praat. However, recall that this percentage consists of two parts: the errors about the voice of a frame and the errors in frequency. Table 1 shows that the LSTM network made more frequency errors but made less voicing errors than the debiased ACF in Praat. The network alone is thus not the best method for pitch estimation, but it could play a role in the voicing decision, maybe replacing the parameters about the voice in the debiased ACF in Praat.

The most striking result is that the deep network made fewer frequency errors than the LSTM network. This suggests that the LSTM network does not benefit from the memory information, which may be due to the fact that the output for each frame (which was the information that was added to the cell state in the LSTM cell) was a place indicating which candidate was chosen. This place did not correspond directly to the frequency, since the same frequency (or roughly the same) could be the first candidate in one frame but the fourth candidate in another. This problem could be solved by using an output that is the same for all inputs and does not depend on the candidates in the input. As suggested by Paul Boersma, an output layer of 100 nodes each representing a frequency in between 1 and 1000 at a 10 Hz interval could be used. The frequency information will then be stored in the cell state and contribute more to the network than the place information.

The deep network had two deep layers, one existing of 1000 nodes. This large number of nodes enables the deep network to make more abstractions of the input data than a single LSTM layer can. With these abstractions, the network could determine more precisely which features are most important in getting good results. This might explain the fact that the deep network made less frequency errors than the LSTM network. Nevertheless, it was expected that the LSTM network would benefit more from the memory information than the deep network would benefit from the large number of nodes in the hidden layer.

The LSTM network overfitted the training data at the end of the training. This occurred earlier in the online training. Overfitting can reduce performance on a test set. Regularization is shown to reduce overfitting (Zaremba, Sutskever & Vinyals 2014) and (Merity, Keskar & Socher 2017), but it did not improve performance on the test set in this study. Using a few regularization methods, the network even achieved lower results on the test set. More research could be done to find the best way of applying regularization to the networks constructed in this study. The use of more data could help the network have a lower chance on overfitting and achieve better results.

Lastly, since not all frames had 15 candidates, some frames were padded to 15 with zeros, which did not contribute to the network. It could be the case that these pads have prevented the networks from performing optimally.

REFERENCES

Atake, Y., Irino, T., Kawahara, H., Lu, J., Nakamura, S., & Shikano, K. (2000). Robust fundamental frequency estimation using instantaneous frequencies of harmonic components. *6th International Conference on Spoken Language Processing.*

Alphen, P. van, & Bergem, D. van (1989). Markov models and their application in speech recognition. *Proceedings of the Institute of Phonetic Sciences of the University of Amsterdam, 13*, 1-26.

Ba, J., Mnih, V., & Kavukcuoglu, K. (2014). Multiple object recognition with visual attention In *Proc. International Conference on Learning Representations,* arXiv:1412.7755.

Barnard, E., Cole, R. A., Vea, M. P., & Alleva, F. A. (1991). Pitch detection with a neural-net classifier. *IEEE Transactions on Signal Processing, 39*, 298-307.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*, 157-166.

Boersma, P. (1993). Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound. *Proceedings of the Institute of Phonetic Sciences, 17,* 97-110.

Cheveigné, A. de, & Kawahara, H. (2002). YIN, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America, 111***,** 1917-1930.

Chollet, F. (2015). keras, *GitHub*, https://github.com/fchollet/keras

Dorken, E., & Nawab, S. H. (1994). Improved musical pitch tracking using principal decomposition analysis. *Proceedings of ICASSP'94, IEEE International Conference on Acoustics, Speech and Signal Processing*, *2*, 217-220.

Flanagan, J. (1965) Speech Analysis, Synthesis and Perception. New York: Springer-Verlag.

Forney, G. D. (1973). The Viterbi Algorithm. *Proceedings of The IEEE, 61***,** 268-278.

Gerhard, D. (2003). Pitch extraction and fundamental frequency: History and current techniques Regina: Department of Computer Science, University of Regina.

Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, *18*, 602-610.

Graves, A., Jaitly, N., & Mohamed, A. R. (2013a). Speech recognition with Recurrent Neural Networks. *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, 6645-6649.

Graves, A., Jaitly, N., & Mohamed, A. R. (2013b). Hybrid speech recognition with deep bidirectional LSTM. *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, 273-278.

Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., & Wierstra, D. (2015). Draw: A recurrent neural network for image generation. *arXiv:1502.04623*.

Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A. R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. &
Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, *29*, 82-97.

Hochreiter, S. and Schmidhuber, J., (1997). Long short-term memory. *Neural Computation*, *9*, 1735–1780.

Kawahara, H., Katayose, H., de Cheveigné, A. D., & Patterson, R. D. (1999). Fixed point analysis of frequency to instantaneous frequency mapping for accurate estimation of F0 and periodicity. In *Sixth European Conference on Speech Communication and Technology*.

Kedem, B. (1986). Spectral analysis and discrimination by zero-crossings. *Proceedings of the IEEE*, *74*, 1477-1493

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv:1412.6980*

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural Networks. *Advances in Neural Information Processing Systems*, 1097-1105.

Merity, S., Keskar, N. S., & Socher, R. (2017). Regularizing and optimizing LSTM language models. *arXiv:1708.02182.*

Noll, A.M. (1967). Cepstrum Pitch Determination. *The Journal of the Acoustical Society of America 41*, 293-309.

Piszczalski, M., & Galler, B. A. (1979). Predicting musical pitch from component frequency ratios. *The Journal of the Acoustical Society of America*, *66*, 710-720.

Principe, J. C., Euliano, N. R., & Lefebvre, W. C. (2000). Neural and adaptive systems. New York: Wiley

Sak, H., Senior, A., & Beaufays, F. (2014). Long short-term memory recurrent neural network architectures for large scale acoustic modeling. *Fifteenth Annual Conference of the International Speech Communication Association*.

Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal*

Processing, *45*, 2673-2681.

Sondhi, M. (1968). New methods of pitch extraction. *IEEE Transactions on Audio and Electroacoustics*, *16*, 262-266.

Talkin, D. (1995). A Robust Algorithm for Pitch Tracking (RAPT), *Speech Coding and Synthesis*, W. B. Kleijn and K. K. Palatal (editors), 497-518.

Wilson, D. R., & Martinez, T. R. (2003). The general inefficiency of batch training for gradient descent learning. *Neural Networks*, *16*, 1429-1451.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, Ł., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M. and Dean, J., Stevens, K. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv:1609.08144*.

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent neural network regularization. *arXiv:1409.2329*.