Optimization of RNN-Based Speech Activity Detection

Gregory Gelly and Jean-Luc Gauvain, *Member, IEEE*

***Abstract*—Speech activity detection (SAD) is an essential component of automatic speech recognition systems impacting the overall system performance. This paper investigates an optimization process for recurrent neural network (RNN) based SAD. This process optimizes all system parameters including those used for feature extraction, the NN weights, and the back-end parameters. Three cost functions are considered for SAD optimization: the frame error rate, the NIST detection cost function, and the word error rate of a downstream speech recognizer. Different types of RNN models and optimization methods are investigated. Three types of RNNs are compared: a basic RNN, long short-term memory (LSTM) network with peepholes, and a coordinated-gate LSTM (CG-LSTM) network introduced by Gelly and Gauvain. Well suited for non-differentiable optimization problems, quantum-behaved particle swarm optimization is used to optimize feature extraction and posterior smoothing, as well as for the initial training of the neural networks. Experimental SAD results are reported on the NIST 2015 SAD evaluation data as well as REPERE and AMI meeting corpora. Speech recognition results are reported on the OpenKWS'13 test data. For all tasks and conditions, the proposed optimization method significantly improves the SAD performance and among all the tested SAD methods the CG-LSTM model gives the best results.**

***Index Terms*—Speech activity detection, recurrent neural networks, long short-term memory, particle swarm optimization.**

I. Introduction

SPEECH activity detection (SAD) is a crucial task in any speech processing system. Although there has been a long history of research on this subject, interest has renewed over the last few years specifically for challenging conditions covering a variety of acoustic environments. This can be seen in recent research projects and challenges (e.g., DARPA RATS program, NIST OpenSAD, CHIME challenge).

Numerous SAD methods and models have been proposed exploiting the spectro-temporal properties of speech and noise to effectively separate speech from non-speech [2], [3]. Some of these methods are energy-based [4], [5], others use autocorrelation coefficients [6], [7], or features that characterize the degree of non-stationarity of the signal on windows of 200 to 300 ms [8], [9].

Neural Network (NN) based methods have been proposed to estimate the probability of speech using low-level acoustic features [10]–[13] from a given audio segment (10 to 30 ms). NN have also been used for a fusion and decision making operating on higher level features [14], [15]. Among these NN-based methods, Recurrent Neural Networks (RNN) and especially Long Short-Term Memory (LSTM) networks have several properties that make them an attractive choice for SAD [16]–[18]. Contrary to a Multi-Layer Perceptron (MLP), a recurrent network does not work on a fixed-size window but with an unlimited temporal context. Even though RNNs were notoriously difficult to train due to the vanishing gradient problem, recent developments [19]–[21] have popularized training techniques to cope with this issue.

We compare three types of RNNs for SAD: a basic RNN, an LSTM network with peepholes and forget units [22], and a coordinated-gate LSTM (CG-LSTM) network that we introduced in [1] which provides more flexibility than the latter LSTM cell. We also compare the three RNN-based SAD methods to a standard MLP-based method and two feature-based SAD methods. The first feature-based method uses the noise robust *CrossCorr* score defined in [7] as a measure of the periodicity of the autocorrelation function linearly combined with the value of the maximum peak of the autocorrelation function as proposed in [7], and the second method uses the Long-Term Signal Variability (LTSV) proposed in [9].

A schematic representation of the three types of SAD systems considered in this work is given in Fig. 1. In all cases, the signal is processed at a 10 ms frame rate to compute the associated set of features: the *CrossCorr* score [7], a short term power spectrum for the LTSV method, and an MFCC vector [23] for the three types of NNs. The raw scalar output of each SAD model is post-processed using the same decision and smoothing back-end. The final SAD output is a set of speech segments defined by their start and end points.

We investigate a global optimization process for RNN-based SAD in order to optimize all system parameters including the feature extraction parameters, the NN weights, and the back-end parameters. Three cost functions are considered for SAD optimization: the frame error rate (FER), the NIST detection cost function (DCF), and the word error rate (WER) which is particularly suitable when the SAD system is used in the front-end of a speech recognizer.

Manuscript received March 5, 2017; revised September 17, 2017; accepted October 16, 2017. Date of publication November 8, 2017; date of current version January 25, 2018. This work was supported in part by the French National Agency for Research as part of the Speech and Language Technologies for Security Applications Project under Grant ANR-14-CE28-0021 and in part by the Intelligence Advanced Research Projects Activity via Department of Defense U.S. Army Research Laboratory Contract W911NF-12-C-0013. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Mohamed Afify. *(Corresponding author: Gregory Gelly.)*

The authors are with LIMSI-CNRS, Orsay 91405, France (e-mail: gelly@limsi.fr; gauvain@limsi.fr).

Digital Object Identifier 10.1109/TASLP.2017.2769220

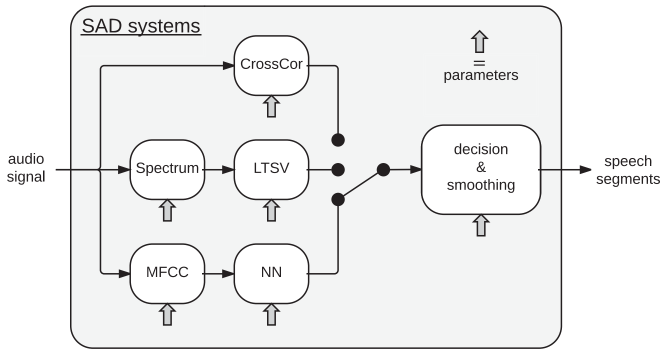


Fig. 1. Schematic representation of the SAD systems: The audio signal is processed by the appropriate front-end (*CrossCorr* score, short term spectrum, and MFCC). The SAD model produces a scalar output post-processed by the decision and smoothing component. The thick arrows represent the trainable/tunable parameters.

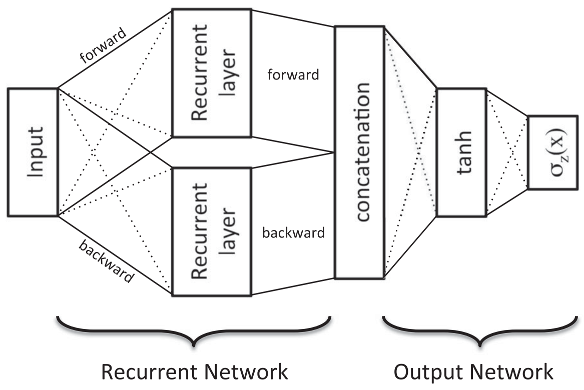


Fig. 2. Bidirectional RNN: two different RNNs process the sequence in opposite directions. The output network takes as input the concatenation of the outputs of the forward and backward layers.

Given this objective, there is a need to optimize non-differentiable model components such as the *CrossCorr* score, the LTSV, and the smoothing back-end, while employing a complex loss function (WER). This led us to investigate quantum-behaved particle swarm optimization (QPSO) which we compare to, and also combine with, the usual gradient descent (GD) training method for NN-based SADs.

In order to validate our method across languages and data types, the experimental work was carried out on broadcast data coming from the REPERE evaluation,[1] the AMI meeting corpus[2] and telephone data from two NIST evaluations: OpenSAD'15[3] and OpenKWS'13.[4]

The remainder of this paper is organized as follows. Section II provides details about the explored neural network architectures including the coordinated-gate LSTM [1]. Section III describes the hybrid QPSO-GD training along with the considered loss functions. The experimental setup is presented in Section IV and results are given in Section V. Finally conclusions are drawn in Section VI. Details of the back-propagation algorithm used to train the CG-LSTM network and which have not been previously published are provided in the Appendix.

II. Network Architecture

This section overviews the architectures of the neural networks used in our experiments. These include a basic RNN as introduced by Elman [24] and RNNs based on two flavors of LSTM cells: the LSTM with peepholes as introduced by Gers *et al.* [22], and the coordinated-gate LSTM (CG-LSTM) introduced in [1].

One shortcoming of unidirectional RNNs is that they are only able to make use of the left context. For SAD purposes there is no reason not to exploit the right context as well. Bidirectional recurrent neural networks (*BiRNN*) were proposed by Schuster and Paliwal [25] to do just that: two distinct RNNs process the sequence both forward and backward, and then the output of both networks are combined and fed into the feedforward output layers that we call the output network (cf Fig. 2). In the literature (e.g., [26]–[28]) bidirectional networks are reported to always outperform unidirectional ones when dealing with frame-wise classification, so only bidirectional RNNs were used in this study.[5]

For all RNNs discussed in this paper, the output network has only one hidden layer with *tanh* activation functions and a logistic activation function for the single output of the final layer. It therefore produces sequences of real values between 0 and 1 that can be interpreted as the probability that the current audio frame contains speech.

In order to have a fair comparison between the various types of neural networks, all the NN-based models that we tested use MFCC features as input (with first and second order derivatives), and all neural networks are configured to have the same number of parameters (6k). The MFCC extraction parameters are optimized jointly with the weights of the NN and with the parameters of the post-processing component.

A. Basic RNN

Given an input sequence $(\boldsymbol{x}(1), ..., \boldsymbol{x}(t_f))$, a basic recurrent layer computes the output vector sequence $(\boldsymbol{z}(1), ..., \boldsymbol{z}(t_f))$ by iterating the (1) for $t = 1 \rightarrow t_f$:

$$\boldsymbol{z}(t) = \sigma(\boldsymbol{W} \cdot \boldsymbol{x}(t) + \boldsymbol{V} \cdot \boldsymbol{z}(t-1) + \boldsymbol{b}) \qquad (1)$$

with $\boldsymbol{z}(0) = \boldsymbol{0}$ and where the *weight matrix* $\boldsymbol{W}$ and the *bias vectors* $\boldsymbol{b}$ define the affine transformation of the input vectors, $\boldsymbol{V}$ is the *interconnection matrix* between the current internal state of the recurrent layer and its output at the previous step, and σ is the *transfer function* of the layer. The latter is typically chosen among bounded non-linear functions such as the hyperbolic tangent applied element-wise in our case.

[1]http://www.defi-repere.fr/
[2]http://groups.inf.ed.ac.uk/ami/corpus
[3]www.nist.gov/itl/iad/mig/nist-open-speech-activity-detection-evaluation
[4]www.nist.gov/itl/iad/mig/open-keyword-search-evaluation

[5]For on-line SAD, one should use unidirectional RNNs. As shown in Table IV the error rates with the unidirectional RNN is a little higher than that of the bidirectional RNN. Alternatively, if a delay on the output is acceptable, a bidirectional RNN can be used with small (e.g., 3 s) overlapping windows of the signal.

For the output network, given an input vector $\boldsymbol{x}(t)$ which is the concatenation of the outputs of both forward and backward recurrent layers for timestep t, a *fully connected feed-forward network* with one hidden layer associates an output vector $\boldsymbol{z}(t)$ computed as follows:

$$\boldsymbol{z}(t) = \sigma_z(\boldsymbol{W}_z \cdot \tanh(\boldsymbol{W}_h \cdot \boldsymbol{x}(t) + \boldsymbol{b}_h) + \boldsymbol{b}_z) \tag{2}$$

where σ_z is the logistic function:

$$\sigma_z(x) = \frac{1}{1 + \mathrm{e}^{-x}}\ , \quad x \in \mathbb{R} \tag{3}$$

B. LSTM RNN

Long Short-Term Memory cells have recently been recognized as the leading approach to modeling sequential structure. They were introduced to overcome some of the shortcomings of basic RNNs [29] and were popularized by Graves [27], [28]. Like Graves, we use the LSTM variant with peepholes introduced by Gers *et al.* [22].

The use of such a LSTM layer instead of the basic RNN layer modifies the computation of $\boldsymbol{z}(t)$ as follows:

$$\boldsymbol{a}_i(t) = \boldsymbol{W}_i \cdot \boldsymbol{x}(t) + \boldsymbol{V}_i \cdot \boldsymbol{z}(t-1) + \boldsymbol{u}_i \odot \boldsymbol{c}(t-1) + \boldsymbol{b}_i \tag{4}$$

$$\boldsymbol{a}_f(t) = \boldsymbol{W}_f \cdot \boldsymbol{x}(t) + \boldsymbol{V}_f \cdot \boldsymbol{z}(t-1) + \boldsymbol{u}_f \odot \boldsymbol{c}(t-1) + \boldsymbol{b}_f \tag{5}$$

$$\boldsymbol{i}(t) = \sigma_i(\boldsymbol{a}_i(t)) \quad ; \quad \boldsymbol{f}(t) = \sigma_f(\boldsymbol{a}_f(t)) \tag{6}$$

$$\boldsymbol{a}_c(t) = \boldsymbol{W}_c \cdot \boldsymbol{x}(t) + \boldsymbol{V}_c \cdot \boldsymbol{z}(t-1) + \boldsymbol{b}_c \tag{7}$$

$$\boldsymbol{c}(t) = \boldsymbol{f}(t) \odot \boldsymbol{c}(t-1) + \boldsymbol{i}(t) \odot \sigma_c(\boldsymbol{a}_c(t)) \tag{8}$$

$$\boldsymbol{a}_o(t) = \boldsymbol{W}_o \cdot \boldsymbol{x}(t) + \boldsymbol{V}_o \cdot \boldsymbol{z}(t-1) + \boldsymbol{u}_o \odot \boldsymbol{c}(t) + \boldsymbol{b}_o \tag{9}$$

$$\boldsymbol{o}(t) = \sigma_o(\boldsymbol{a}_o(t)) \tag{10}$$

$$\boldsymbol{z}(t) = \boldsymbol{o}(t) \odot \sigma_z(\boldsymbol{c}(t))\, r \tag{11}$$

where $\odot$ is the element-wise multiplication, $\boldsymbol{i}(t)$, $\boldsymbol{f}(t)$, $\boldsymbol{c}(t)$ and $\boldsymbol{o}(t)$ are respectively the activation vectors of:

- the *input gates* that controls the amount of information that enters the LSTM layer,
- the *forget gates* that controls the amount of information that is remembered from the previous step,
- the *cells* which are the internal state of the LSTM layer,
- the *output gates* that controls the amount of information that comes out of the LSTM layer.

All these activation vectors have the same size as the LSTM layer, as do the vectors $\boldsymbol{u}_i$, $\boldsymbol{u}_f$, and $\boldsymbol{u}_o$ in (4), (5) and (9).

C. Coordinated-Gate LSTM RNN

For the CG-LSTM cell that we introduced in [1] and in which direct links are added between the three gates of a cell, (6) and (10) are modified into (13), (15) and (17):

$$\tilde{\boldsymbol{a}}_i(t) = \boldsymbol{v}_i \odot \boldsymbol{i}(t-1) + \boldsymbol{w}_i \odot \boldsymbol{f}(t-1) + \boldsymbol{y}_i \odot \boldsymbol{o}(t-1) \tag{12}$$

$$\boldsymbol{i}(t) = \sigma_i(\boldsymbol{a}_i(t) + \tilde{\boldsymbol{a}}_i(t)) \tag{13}$$

$$\tilde{\boldsymbol{a}}_f(t) = \boldsymbol{v}_f \odot \boldsymbol{i}(t-1) + \boldsymbol{w}_f \odot \boldsymbol{f}(t-1) + \boldsymbol{y}_f \odot \boldsymbol{o}(t-1) \tag{14}$$

$$\boldsymbol{f}(t) = \sigma_f(\boldsymbol{a}_f(t) + \tilde{\boldsymbol{a}}_f(t)) \tag{15}$$

$$\tilde{\boldsymbol{a}}_o(t) = \boldsymbol{v}_o \odot \boldsymbol{i}(t) + \boldsymbol{w}_o \odot \boldsymbol{f}(t) + \boldsymbol{y}_o \odot \boldsymbol{o}(t-1) \tag{16}$$

$$\boldsymbol{o}(t) = \sigma_o(\boldsymbol{a}_o(t) + \tilde{\boldsymbol{a}}_o(t)) \tag{17}$$

where we use nine new peepholes vectors $\boldsymbol{v}_{\{i,f,o\}}$, $\boldsymbol{w}_{\{i,f,o\}}$ and $\boldsymbol{y}_{\{i,f,o\}}$ and element-wise multiplications $\odot$ so that a gate can have access to the gates of the same cell.

With these new links the three gates of a cell can interact more efficiently and improve the cell behavior (cf. results in Section V). The back-propagation algorithm for the CG-LSTM cell is given in the appendix.

III. SAD Optimization

This section details the optimization process designed to train any SAD system whether it aims to minimize the frame error rate or the word error rate of an ASR system.

This optimization process is based on the heuristic QPSO algorithm which is augmented by a mini-batch gradient descent algorithm when at least part of the SAD model is differentiable, which is the case for all the NN-based SADs.

The choice of QPSO as the optimization algorithm was driven by the need to cope with non-differentiable optimization problems, and also by the fact that QPSO, which is well suited for difficult optimization tasks, has been shown to be more effective than genetic algorithms.

The raw output of all SAD systems (NN-based or not) is post-processed by the same decision and smoothing back-end (cf. Fig. 1), the parameters of which are optimized for each SAD method. The 6 smoothing parameters are:

- the onset and offset thresholds;
- the duration of padding before and after each speech segment;
- the threshold for short speech segment deletion;
- the threshold for short silence deletion.

There are different tunable parameters for the different feature extraction methods. The tunable parameters for each of the three front-ends are listed in Table I.

For the NN-based SADs, QPSO is first used to jointly optimize the MFCC front-end, the neural network weights and the smoothing back-end parameters, where these three sets of parameters are seen by QPSO as a single large parameter vector. GD training is then used to improve the neural network weights. Finally, QPSO is optionally used to refine the back-end parameters to take into account the changes in the NN weights. The impact of this alternating strategy is shown in Section V-B, Table IV.

TABLE I
TUNABLE PARAMETERS FOR THE THREE FRONT-ENDS: *CrossCor*, LTSV, AND MFCC

6	*CrossCor* front-end
1	pre-emphasis coefficient
1	analysis window size
2	autocorrelation lag range
1	level of white noise added to the signal
1	relative weight between autocorrelation peak and cross-correlation coefficient
9	LTSV front-end
1	pre-emphasis coefficient
1	level of white noise added to the signal
5	FFT parameters: window type and size, frame step, minimum and maximum frequencies
2	analysis span and step in number of frames
8	MFCC front-end
2	window type and size
2	minimum and maximum frequencies
1	number of filters
1	number of DCT coefficients
2	Δ and $\Delta\Delta$ cepstrum context sizes

The number of parameters is given in the first column.

For the *CrossCorr* and LTSV SAD methods, only QPSO is used to optimize the model parameters jointly with the decision and smoothing back-end parameters.

More details of the optimization process are given in the following subsections.

A. QPSO

The benefits of heuristic optimization techniques such as genetic algorithms for minimizing complex loss functions was shown in [30]. Since then, similar but more efficient methods such as quantum-behaved particle swarm optimization (QPSO) were developed ([31]–[33]). QPSO is a variant of the PSO algorithm which was motivated by the social behavior of bird flocks and was first introduced by Kennedy and Eberhart as a population-based optimization technique ([34], [35]. Although the PSO algorithm is comparable in performance to genetic algorithms, QPSO was shown to be a more powerful tool than both of them when performing difficult optimization tasks (cf. [32]).

As for many other heuristic optimization schemes, the main idea behind QPSO is to encode the whole set of M parameters to be optimized into a single vector of dimension M, starting with random positions in the search space $\boldsymbol{X}_j$, $j \in [\![1, N]\!]$ for N particles. Then, QPSO produces at each iteration of the optimization process new positions based on the performance of the current particles' positions, the memory of the best positions $\boldsymbol{P}_j$, $j \in [\![1, N]\!]$ seen so far by each particle, and the global best position $\boldsymbol{G}$. The pseudo-code for QPSO is given in Algorithm 1.

B. Gradient Descent

We found that using a second optimization technique specific to neural networks is beneficial to locally improve the solution found by QPSO. As a reminder this is only applicable to the NN components (which are differentiable). Back-propagation as detailed in appendix B is combined with the *SMORMS3* mini-batch gradient descent algorithm as proposed in [36] (cf. Algorithm 2 for the pseudo-code). *SMORMS3* was chosen as it yielded better results than RPROP [37], RMSPROP [38], Adam [39] or the Sum of Functions Optimizer [40].

Algorithm 1: QPSO.

```
for $j = 1..N$ do
    Randomly initialize $\boldsymbol{X}_j$ and $\boldsymbol{P}_j$ for particle $j$
    Evaluate the losses $L(\boldsymbol{X}_j)$ and $L(\boldsymbol{P}_j)$
end for
Find the global best position $\boldsymbol{G}$ among $\boldsymbol{X}_j$ and $\boldsymbol{P}_j$ for $j \in [\![1, N]\!]$
while stopping criterion = false do
    for $j = 1..N$ do
        for $i = 1..M$ do
            Randomly select $\phi$, $u$ and $\alpha$ from a uniform distribution on $]0, 1]$
            $y = \phi \cdot P_{i,j} + (1 - \phi) \cdot \boldsymbol{G}_i$
            if $\alpha > 0.5$ then
                $X_{i,j} = y + |X_{i,j} - P_{i,j}| \ln(1/u)$
            else
                $X_{i,j} = y - |X_{i,j} - P_{i,j}| \ln(1/u)$
            end if
        end for
        Evaluate the loss $L(\boldsymbol{X_j})$ for the new $\boldsymbol{X}_j$
        Update $\boldsymbol{G}$ and $\boldsymbol{P}_j$ if necessary (e.g., $L(\boldsymbol{X_j}) < L(\boldsymbol{G})$)
    end for
end while
```

Algorithm 2: SMORMS3.

```
Initialize $\boldsymbol{p}$ the parameters of the neural network with the best candidate $\boldsymbol{G}$ obtained at the end of QPSO.
Initialize the internal vectors (same size as $\boldsymbol{p}$):
$\boldsymbol{m} = \boldsymbol{1}$, $\boldsymbol{a} = \boldsymbol{0}$ and $\boldsymbol{a}_{sq} = \boldsymbol{0}$
while stopping criterion = false do
    Compute the gradient $\nabla L(\boldsymbol{p})$ of the loss function $L$ w.r.t. $\boldsymbol{p}$ using back-propagation as described in appendix B.
    Update the internal vectors:
        $\boldsymbol{r} = 1/(\boldsymbol{m} + 1)$
        $\boldsymbol{a} = (1 - \boldsymbol{r}) \cdot \boldsymbol{a} + \boldsymbol{r} \cdot \nabla L(\boldsymbol{p})$
        $\boldsymbol{a}_{sq} = (1 - \boldsymbol{r}) \cdot \boldsymbol{a}_{sq} + \boldsymbol{r} \cdot (\nabla L(\boldsymbol{p}))^2$
        $\boldsymbol{m} = 1 + \boldsymbol{m} \cdot (1 - \boldsymbol{a}^2/(\boldsymbol{a}_{sq} + \epsilon))$
    Update the NN parameters:
        $\boldsymbol{p} = \boldsymbol{p} - \nabla L(\boldsymbol{p}) \cdot \min(0.001, \boldsymbol{a}^2/(\boldsymbol{a}_{sq} + \epsilon))/(\sqrt{\boldsymbol{a}_{sq}} + \epsilon)$
end while
(products, divisions and min are performed element-wise)
```

C. Mini-Batch Selection

For both the QPSO and GD algorithms, a small number of audio segments (about 200 of 60 seconds each) are randomly selected for a mini-batch training at each iteration of the optimization loop. We also keep track of the 100 audio segments that lead to the highest error rates (worst cases) seen so far and add them to our mini-batch at each training step. The effect of keeping the worst cases is shown in Table IV (compare the second and last rows).

D. Loss Functions

In this section the two loss functions used in training the SAD systems are defined. L_{FER} is the weighted frame error rate for which only basic annotations are needed (time codes of speech segments) and is straight-forward to implement. L_{WER} is a loss function specifically designed to optimize SAD models for ASR systems that was introduced in [1].

1) Weighted Frame Error Rate (L_{FER}): The weighted frame error rate with respect to manual annotations is defined as:

$$L_{\text{FER}} = \alpha \sum_{t_s \in S} (1 - z(t_s)) + (1 - \alpha) \sum_{t_n \in N} z(t_n) \tag{18}$$

where S is the set of speech frames, N is the set of non-speech frames, α is a weight reflecting the relative importance of the two types of errors (errors on speech frames vs non-speech frames), and $z(t)$ is the binary output of the SAD system at time t. $z(t) = 0$ if the current frame is recognized as a non-speech frame and $z(t) = 1$ if the current frame is recognized as a speech frame.

Gradient descent training requires a differentiable loss function in order to be able to determine the gradient with respect to the parameters. Therefore, a weighted version of the maximum likelihood loss function of a binary classifier is used to mimic the behavior of L_{FER}:

$$L'_{\text{FER}} = -\alpha \sum_{t_s \in S} \ln(1 - z(t_s)) - (1 - \alpha) \sum_{t_n \in N} \ln(z(t_n)) \tag{19}$$

where $\forall t, z(t) \in [0, 1]$ is the real valued output of the neural network.

2) WER-Like Metric (L_{WER}): Optimizing an SAD algorithm to minimize the WER of an ASR system would be best achieved by computing the actual WER for all the SAD settings. Unfortunately, the computational load is too high for this to be a realistic solution.

Instead a loss function L_{WER} was designed that makes use of the ASR output and takes into account the fact that all words, independently of their length, have the same weight.

Let $\mathcal{C}, S, D$ and $\mathcal{I}$ be respectively the sets of correct word occurrences, substitutions, deletions and insertions in the ASR output with respect to the human reference for the whole audio file. For each word occurrence $w \in \mathcal{C} \cup \mathcal{S} \cup \mathcal{I}$, we denote F_w the set of audio frames corresponding to w in the ASR output. The sets $\mathcal{S}'$, $\mathcal{D}'$ and $\mathcal{I}'$ are defined as follows:

- a word $w \in \mathcal{S}'$ if and only if $w \in \mathcal{S}$ and all frames in F_w are classified as speech by the SAD system.
- a word $w \in \mathcal{D}'$ if and only if either $w \in \mathcal{D}$ or if $w \in \mathcal{C} \cup \mathcal{S}$ and at least one frame in F_w is classified as non-speech by the SAD system.
- a word $w \in \mathcal{I}'$ if and only if $w \in \mathcal{I}$ and at least one frame in F_w is classified as speech by the SAD system.

The following two ratios are also introduced:

- τ_d^w is the ratio between the duration of the word w not detected as speech and the whole duration of the word.
- τ_i^w is the ratio between the duration of the word w detected as speech and the whole duration of the word.

Finally the L_{WER} loss function is defined as

$$L_{\text{WER}} = \frac{|\mathcal{S}'| + |\mathcal{D}'| + |\mathcal{I}'| + \tau_i + \tau_d}{N_r} \tag{20}$$

$$with \quad \tau_i = \sum_{w \in pW_I} \tau_i^w \quad and \quad \tau_d = \sum_{w \in W_C \cup W_S} \tau_d^w \tag{21}$$

where N_r is the number of words in the reference transcription. The two terms τ_i and τ_d were introduced to smooth the discontinuities of the metric. Doing so the optimization algorithm is less prone to being trapped on a plateau of the loss function.

For GD training, we need a differentiable loss function to be able to compute the gradient with respect to the parameters. Thus, we designed a differentiable equivalent to (20):

$$\begin{aligned} L'_{\text{WER}} = &- \sum_{w \in W_C \cup W_S} \left(\frac{1}{\delta_w} \sum_{t_w \in F_w} \ln(z(t_w)) \right) \\ &- \sum_{w \in W_I} \left(\frac{1}{\delta_w} \sum_{t_w \in F_w} \ln(1 - z(t_w)) \right) \end{aligned} \tag{22}$$

where $\forall t, z(t) \in [0; 1]$ is the output of the neural network for the frame t_w of the word w and δ_w is the number of frames in the word w.

E. Overall Optimization Process

The optimization process for the CrossCorr and LTSV SAD methods is performed in one step. This step consists of using QPSO to jointly optimize the feature extraction parameters and the parameters of the decision and smoothing back-end. To do so requires using the adequate loss function for the evaluation metric (i.e., L_{FER} to minimize the FER and L_{WER} to minimize the WER).

For the NN-based SADs, the optimization process is composed of three steps:

1) QPSO is first used to jointly optimize the MFCC front-end, the NN weights and the parameters of the decision and smoothing back-end. This first step is denoted *QPSO* in the table of results.
2) GD training is used to refine the NN weights, the other parameters being kept at the values estimated with QPSO in step 1. This second step is denoted *GD*.
3) QPSO is then used to refine the parameters of the decision and smoothing back-end. This step is denoted *back-end QPSO* in the following.

In steps 1 and 3 the loss functions L_{FER} and L_{WER} are used. Their differentiable versions, L'_{FER} and L'_{WER} are used in step 2.

For all of the results presented in this paper, the durations of the 3 optimization steps (QPSO, GD, back-end QPSO) are identical by design, i.e., the number of iterations in each step was set to require roughly the same computation time for each of the 3 steps.

IV. Experimental Conditions

This section describes the four sets of data used in the experiments reported here: the NIST OpenSAD'15 evaluation data, the REPERE and AMI Meeting corpora, and the NIST OpenKWS'13 evaluation data for speech recognition purposes.

A. *OpenSAD'15 Data and Conditions*

The goal of the NIST open Speech Activity Detection 2015 (OpenSAD'15) [41] evaluation was to assess the state-of-the-art in identifying speech activity regions in signals from distorted, degraded, weak, and/or noisy communication channels. Most of the evaluation data came from sequestered data from the DARPA Robust Automatic Transcription of Speech (RATS) program. The data is challenging for SAD as it is highly heterogeneous including a variety of transmitter/receiver radio-link channels, noisy conditions and long segments with only little speech. The speech data was originally collected over public telephone networks (landline or cell) and was retransmitted/received over 7 HF/VHF/UHF channels called A, B, C, E, F and G. Two of these channels (A and C) were present only in the evaluation data. This corpus contains conversations in five languages: English, Levantine Arabic, Farsi, Pashto and Urdu.

The evaluation metric is the detection cost function (DCF) defined as follows:

$$DCF = 0.75 \cdot P_{\text{miss}} + 0.25 \cdot P_{fa} \tag{23}$$

which is a weighted combination of misses and false alarms.

The behavior of the DCF is close to the behavior of the weighted frame error rate loss function L_{FER} defined in (18) with $\alpha = 0.75$. The difference resides in the denominators used for computing P_{miss} and P_{fa} (i.e., the total duration of speech and the total duration of non-speech). Since the data used in the OpenSAD evaluation originally comes from telephone conversations, these two denominators are approximately the same. Collars are used to compute the NIST DCF which has a smoothing effect on the DCF values.

It is important to mention that scoring of the evaluation data could only be carried out by NIST as the annotations of the evaluation data have not been distributed, therefore the contrastive results are reported on the development data set.

B. *REPERE Data*

As explained in [42] and [43], the main task of the REPERE evaluation was the multimedia recognition of persons in TV shows. The REPERE corpus is comprised of news shows from two French TV channels: LCP and BFMTV. The video files contain very different show types such as debates, interviews, reports in the field. This corpus includes audio tracks with a great variety of acoustic environments which can be challenging for SAD systems. These characteristics make the REPERE corpus a good choice to evaluate the robustness of the various SAD methods for broadcast conditions. The FER is used to evaluate the SAD results on this data.

C. *AMI Data*

The European-funded AMI project collected a set of recorded meetings that is now available as a public resource. The AMI Meeting Corpus consists of 100 hours of meetings recorded in three rooms with different acoustic properties. While the language of the meetings is English, most of the speakers are non-native. This meeting scenario is very different for the other audio data types and is challenging for SAD. As for the REPERE data, the FER metric is used to evaluate the SAD results.

D. *OpenKWS'13 Data*

Data from the IARPA Babel program were used to assess the SAD systems optimized for ASR. Experiments were carried out using data in four languages: Vietnamese, Pashto, Turkish and Tagalog. The Babel datasets[6] are approximately gender-balanced and contain a diversity of styles, speakers and environments. The training set includes 160 hours of audio for approximately 90 hours of speech per language. The evaluation set contains 30 hours of audio for approximately 17 hours of speech for each language. Orthographic transcriptions to train and evaluate ASR systems are provided for this data. These transcriptions are used to evaluate all of the SAD methods that are optimized to minimize the WER of the downstream ASR system.

State-of-the-art ASR systems were trained on only the distributed Babel data. All systems used hybrid HMM-MLP acoustic models and 4-gram language models with vocabularies sizes of 10 K words similar to the systems described in [44].

V. Results

This section provides the results using the studied SAD methods for the four different corpora. All models were trained using the optimization framework introduced in Section III. Contrastive results are given in Tables II and III with and without QPSO optimization. Without QPSO optimization, the setup proposed in [7] and [9] are used for the non-NN methods, CrossCorr and LTSV respectively. The LIMSI standard MFCC front-end and gradient-descent alone is used for NN training.

A. *FER Optimization*

Six SAD systems trained using the L_{FER} or L'_{FER} loss functions are compared on all four datasets. The NIST DCF values for OpenSAD'15 and the FER for the other three corpora are given in Table II. For the REPERE and AMI data, the SAD metric to minimize is the FER and we use the L_{FER} loss function with $\alpha = 0.5$ to train the SAD systems.

The BiRNN architecture depicted in Fig. 2 with only one recurrent layer for each direction is used for all RNN-based

[6]Vietnamese IARPA-babel107b-v0.7, Pashto IARPA-babel104b-v0.4b, Turkish IARPA-babel105b-v0.5, Tagalog IARPA-babel106-v0.2g

TABLE II
RESULTS ON THE FOUR DATASETS FOR 6 SAD SYSTEMS TRAINED USING THE L_{FER} LOSS FUNCTION

Metric	FER			DCF
Cost function	FER (L_{FER}; $\alpha = 0.5$)			L_{FER} ($\alpha = 0.75$)
Corpus	REPERE	AMI	OpenKWS'13	OpenSAD'15
CrossCorr	17.94 (29.85)	8.34 (45.38)	7.60 (19.52)	12.0 (13.6)
LTSV	16.67 (19.13)	8.29 (54.94)	7.85 (19.02)	10.1 (13.7)
MLP	15.62 (17.20)	6.84 (10.23)	6.29 (12.15)	5.3 (6.7)
Basic RNN	15.45 (16.13)	6.55 (6.93)	6.24 (8.33)	4.1 (4.5)
LSTM	13.37 (15.25)	6.20 (6.53)	5.83 (8.07)	3.4 (3.7)
CG-LSTM	12.66 (14.83)	5.93 (6.40)	5.76 (7.84)	3.0 (3.2)

Results are given in terms of DCF for the OpenSAD'15 data and in terms of FER for the 3 others. Numbers in parentheses are without any QPSO optimization.

TABLE III
WERS ON THE OPENKWS EVALUATION DATA FOR FOUR LANGUAGES AFTER OPTIMIZING EACH SAD MODEL USING THE L_{WER} LOSS FUNCTION

Metric	WER				
Cost function	FER (L_{FER}; $\alpha = 0.5$)	L_{WER}			
Corpus	Vietnamese		Pashto	Turkish	Tagalog
CrossCorr	56.7 (58.8)	56.1 (58.8)	63.9 (64.9)	60.6 (62.9)	57.1 (60.0)
LTSV	57.1 (58.5)	55.6 (58.5)	64.0 (64.5)	60.6 (61.2)	56.7 (56.9)
MLP	56.6 (61.5)	55.4 (61.1)	63.5 (64.2)	60.3 (62.4)	56.2 (59.4)
Basic RNN	56.5 (56.8)	55.2 (56.5)	63.5 (63.8)	60.2 (60.7)	56.3 (57.7)
LSTM	56.5 (56.4)	54.8 (56.0)	63.2 (63.7)	60.2 (60.5)	56.2 (57.5)
CG-LSTM	56.4 (56.2)	54.6 (55.7)	63.2 (63.6)	60.0 (60.5)	56.0 (57.4)

Contrastive results based on the FER and the L_{FER} loss function are given for Vietnamese. Numbers in parentheses are without any QPSO optimization.

SADs. For the comparison to be fair, all the NN models have 6 K weights. The size of the recurrent layer is 35 for the basic RNN and 13 for the LSTM networks. The output network of the RNNs has one hidden layer with 16 neurons. For the MLP-based SAD, only the output network with a hidden layer of size 164 is used.[7]

As shown in Table II, the NN-based SAD systems are seen to consistently outperform the non NN-based SAD models and the RNN-based systems outperform the MLP-based one. The best performance is obtained with the LSTM networks, the CG-LSTM network improving the DCF by 12% relative over the original LSTM network on the OpenSAD'15 data and reducing the FER by 4% relative on average on the three other corpora.

It should be noted that network size is quite small leading to fast decoding. For the CG-LSTM network, decoding is done in 0.001xRT on a standard desktop CPU.

B. WER Optimization

The impact of the different SAD systems on the ASR performance is measured by the actual WER on the OpenKWS evaluation data. The four NN-based systems have the same number of parameters (6 K weights). The WERs on the evaluation data are given in Table III. These results were obtained after optimizing each SAD system using the loss function L_{WER} (and L'_{WER}) defined in (19) for each of the four languages (Vietnamese, Pashto, Turkish and Tagalog).

The impact of using a task specific loss function can be seen by comparing the two sets of results for the Vietnamese language. Using the L_{WER} loss function, designed to mimic the WER, leads to an average relative gain of 2.9% in WER for all SAD models compared to using the FER.

The differences in WER performance across the various SAD models is much less important than the differences in DCF for the OpenSAD task and in FER on the REPERE and AMI corpora, however the overall ranking remains the same. As can be expected the WER of a downstream ASR system is less impacted by SAD than the DCF or the FER. To take the extreme case, if the entire audio signal is wrongly hypothesized as speech, there will generally be only a small impact on the WER as the ASR can ignore noisy segments, whereas the FER and the DCF will be directly correlated with the real amount of non-speech data.

Overall the relative differences in results for the tested conditions are comparable across the 4 languages, exhibiting the language independence of the optimization method.

The CG-LSTM network obtains the lowest WER for all languages. This network reduces the WER by 1.8% relative compared to the best non-neural SAD system using LTSV. A more detailed analysis shows that this gain mainly comes from reducing the number of insertions (e.g., 5.0% to 3.6% for Vietnamese), the CG-LSTM model being able to discard signal (even speech) that causes ASR insertions.

[7] While a more sophisticated non-recurrent architecture using more contextual information could improve the SAD performance, the goal here was to compare both recurrent and non-recurrent simple architectures on the exact same input sequence.

TABLE IV
DCFs ON THE OPENSAD DATA, FERs AND WERs ON THE OPENKWS DATA (VIETAMESE ONLY) WITH DIFFERENT STRATEGIES TO TRAIN BIDIRECTIONAL (BIDIR) LSTM NETWORKS

Optimization strategy				Bidir	CG	OpenSAD'15	OpenKWS'13	
QPSO	*GD*	*back-end QPSO*	WC addition			DCF	FER	WER
✓			✓	✓	✓	6.42	9.96	56.6
✓	✓	✓		✓	✓	3.22	7.13	56.4
	✓		✓	✓	✓	3.19	7.84	55.7
	✓	✓	✓	✓	✓	3.17	6.65	55.4
✓	✓		✓		✓	3.15	6.12	55.4
✓	✓		✓	✓	✓	3.04	5.83	55.0
✓	✓	✓	✓	✓		3.38	5.83	54.8
✓	✓	✓	✓	✓	✓	3.01	5.76	54.6

The column 'WC addition' indicates whether or not training includes the worst case (WC) segments in the mini-batch.

Since the performance difference between the best two systems (LSTM and CG-LSTM) is quite small (0.2), we used NIST's sc_stats tool to perform a two-tailed significance test with the null hypothesis that there is no performance difference between the two systems. With the "Matched Pair Sentence Segment (Word Error)" test a significant difference between the two systems was found at the level of $p = 0.05$ (i.e., the NIST standard condition to compare ASR systems) on the Vietnamese and Turkish data.

C. Importance of the Optimization Strategy

To evaluate the relative importance of the QPSO and the gradient descent algorithms, different combinations of these two optimization schemes have been tested for training the CG-LSTM network. When we do not perform the *QPSO* step, the SAD system uses standard MFCC extraction parameters which are not optimized to improve the performance. Similarly, when we do not perform the *back-end QPSO* step, the decision thresholds are set to 0.5 and the boundaries of the speech segments are determined without any smoothing. Table IV reports the resulting DCFs on the OpenSAD development data and the WERs on the OpenKWS Vietnamese evaluation data set. For a fair comparison, all these results were obtained with identical overall training times.

These results show that even though GD alone leads to good results, using the 3-step approach achieves the best performance. In particular, combining the two optimization techniques through the *QPSO* and *GD* steps lowers the FER by 23% relative, the WER by 1.5% relative and the DCF by more than 5% relative compared to the best optimization method used alone. The gains brought by the *back-end QPSO* step are small[8] but consistent. Adding the worst cases in the mini-batch training as described in Section III-C Table IV (2nd and last row) leads to a 19.2% relative reduction in FER and a 3.2% relative gain in WER and improves the DCF by close to 7% relative. The comparison of uni- and bi-directional RNNs is given in lines 5 and 6. The differences are small but significant.

[8]We performed a two-tailed significance test with the null hypothesis that there is no performance difference between the systems with and without smoothing and found a significant difference between the two systems at the level of $p < 0.001$.

VI. CONCLUSIONS

In this paper, we proposed an optimization method for neural-network based speech activity detection, with the goal of optimizing all system parameters, including the feature extraction parameters, the NN weights, and the back-end parameters. Three metrics are considered: the frame error rate, the NIST detection cost function, and the WER of a downstream speech recognizer. Being well suited for non-differentiable optimization problems, QPSO training is used to tune the MFCC front-end and the smoothing back-end, and for the initial training of the neural network. Gradient descent training is then used to refine the neural network weights. We defined two loss functions, L_{FER} and L_{WER}, approximating the FER, the NIST DCF and the WER, along with their differentiable versions (L'_{FER} and L'_{WER}) for use with GD training. Three types of RNNs were investigated: a basic RNN, an LSTM network with peepholes, and a coordinated-gate LSTM network.

Experimental results were reported using the DCF, FER and WER metrics on four different corpora covering a range of data types and languages: the REPERE evaluation data, the AMI meeting corpus, the NIST OpenSAD'15 evaluation data and the multilingual conversational corpus from the NIST OpenKWS'13 evaluation. The proposed optimization method outperforms gradient-descent training alone. Using appropriate loss functions allows optimization of either the frame error rate or the word error rate of a downstream ASR system. For all conditions the CG-LSTM network outperforms the original LSTM network, a basic RNN as well as an MLP-based and two other baseline SAD systems.

APPENDIX
CG-LSTM BACK-PROPAGATION

This appendix details the back-propagation algorithm for a bidirectional CG-LSTM network.

The goal of the back-propagation is to efficiently compute the first derivatives of the loss function L that we want to minimize with respect to the weights and biases of the neural network. To do so, we first compute the gradient of L w.r.t. the output $\boldsymbol{z}(t)$ of the neural network for $t = t_f \rightarrow 1$ and then back-propagate these gradients through all the layers from the last to the first using the derivation chain rule.

First, we detail the back-propagation through the output network. Then, we fully describe the back-propagation of the gradients in a CG-LSTM layer.

A. Output Network

For an MLP such as the output network of our bidirectional RNN architecture, we rewrite (2) as follows:

$$\boldsymbol{a}_h(t) = \boldsymbol{W}_h \cdot \boldsymbol{x}(t) + \boldsymbol{b}_h \tag{24}$$

$$\boldsymbol{h}(t) = \tanh(\boldsymbol{a}_h(t)) \tag{25}$$

and

$$\boldsymbol{a}_z(t) = \boldsymbol{W}_z \cdot \boldsymbol{h}(t) + \boldsymbol{b}_z \tag{26}$$

$$\boldsymbol{z}(t) = \sigma_z(\boldsymbol{a}_z(t)) \tag{27}$$

Then, we first back-propagate the gradient through the activation function of the output layer defined in (27). To do that, we need to compute $\boldsymbol{J}_{\sigma_z}(\boldsymbol{a}_z(t))$ the Jacobian matrix of σ_z at the point $\boldsymbol{a}_z(t)$ and then the chain rule for partial derivatives gives:

$$\frac{\partial L}{\partial \boldsymbol{a}_z}(t) = \boldsymbol{J}_{\sigma_z}(\boldsymbol{a}_z(t)) \cdot \frac{\partial L}{\partial \boldsymbol{z}}(t) \tag{28}$$

In our case where σ_z is the logistic function (cf. (3)) applied element-wise to $\boldsymbol{a}_z(t)$, $\boldsymbol{J}_{\sigma_z}(\boldsymbol{a}_z(t))$ is diagonal and (28) can be rewritten in a simpler and a more computationally efficient way:

$$\frac{\partial L}{\partial \boldsymbol{a}_z}(t) = \boldsymbol{z}(t) \odot (1 - \boldsymbol{z}(t)) \odot \frac{\partial L}{\partial \boldsymbol{z}}(t) \tag{29}$$

Then, using again the chain rule we can back-propagate through (26) and (25):

$$\frac{\partial L}{\partial \boldsymbol{h}}(t) = \boldsymbol{W}_z^T \cdot \frac{\partial L}{\partial \boldsymbol{a}_z}(t) \tag{30}$$

$$\frac{\partial L}{\partial \boldsymbol{a}_h}(t) = \boldsymbol{J}_{\sigma_h}(\boldsymbol{a}_h(t)) \cdot \frac{\partial L}{\partial \boldsymbol{h}}(t) \tag{31}$$

where $\boldsymbol{W}_z^T$ is the transpose of $\boldsymbol{W}_z$.

Since σ_h is in our case the hyperbolic tangent applied element-wise we can rewrite (31) similarly to what we did to obtain (29):

$$\frac{\partial L}{\partial \boldsymbol{a}_h}(t) = (\boldsymbol{1} - \boldsymbol{h}(t) \odot \boldsymbol{h}(t)) \odot \frac{\partial L}{\partial \boldsymbol{h}}(t) \tag{32}$$

Now, we have everything needed to compute the first derivatives of L w.r.t. the weight matrices and the bias vectors by summing the independent contributions of each time-step to the gradients:

$$\frac{\partial L}{\partial \boldsymbol{W}_z} = \sum_{t=1}^{t_f} \frac{\partial L}{\partial \boldsymbol{a}_z}(t) \cdot \boldsymbol{h}(t)^T \quad ; \quad \frac{\partial L}{\partial \boldsymbol{b}_z} = \sum_{t=1}^{t_f} \frac{\partial L}{\partial \boldsymbol{a}_z}(t) \tag{33}$$

and

$$\frac{\partial L}{\partial \boldsymbol{W}_h} = \sum_{t=1}^{t_f} \frac{\partial L}{\partial \boldsymbol{a}_h}(t) \cdot \boldsymbol{x}(t)^T \quad ; \quad \frac{\partial L}{\partial \boldsymbol{b}_h} = \sum_{t=1}^{t_f} \frac{\partial L}{\partial \boldsymbol{a}_h}(t) \tag{34}$$

Since we need to back-propagate the gradient further into the recurrent layers, we have to compute the derivatives of L w.r.t. the input sequence $(\boldsymbol{x}(1), ..., \boldsymbol{x}(t_f))$ of the output network:

$$\frac{\partial L}{\partial \boldsymbol{x}}(t) = \boldsymbol{W}_h^T \cdot \frac{\partial L}{\partial \boldsymbol{a}_h}(t) \tag{35}$$

Then, these gradients are given as input to the back-propagation algorithm for the CG-LSTM layers.

B. CG-LSTM

For a CG-LSTM layer, we use a slightly different back-propagation algorithm which is called back-propagation through time (BPTT [45]). As for standard back-propagation, BPTT consists in repeatedly applying the chain rule, but for a recurrent layer we also need to take into account the back-propagation of the gradient through the recurrent links. Hence, we iterate over time from the last step to the first ($t : t_f \to 1$) back-propagating through (11) to (4) for each time-step. For the CG-LSTM model, one also needs to take into account the new links between the gates described by (12) to (17).

Thus, first we back-propagate the gradient through (11) and we have:

$$\frac{\partial L}{\partial \boldsymbol{o}}(t) = \left(\frac{\partial L}{\partial \boldsymbol{z}}(t) + \boldsymbol{\epsilon}(t+1)\right) \odot \sigma_z(\boldsymbol{c}(t)) + \boldsymbol{\beta}_o(t+1) \tag{36}$$

where $\boldsymbol{\epsilon}(t+1)$ is define by (54) and corresponds to the contribution of the recurrent links to the gradient with $\boldsymbol{\epsilon}(t_f+1) = \boldsymbol{0}$ since no contribution to the gradient can come from after the end of the sequence. And with

$$\boldsymbol{\beta}_o(t) = \boldsymbol{y}_i \odot \frac{\partial L}{\partial \boldsymbol{a}_i}(t) + \boldsymbol{y}_f \odot \frac{\partial L}{\partial \boldsymbol{a}_f}(t) + \boldsymbol{y}_o \odot \frac{\partial L}{\partial \boldsymbol{a}_o}(t) \tag{37}$$

and $\boldsymbol{\beta}_o(t_f+1) = \boldsymbol{0}$.

Then, back-propagating the gradient through (10), we can compute:

$$\frac{\partial L}{\partial \boldsymbol{a}_o}(t) = \boldsymbol{J}_{\sigma_o}(\boldsymbol{a}_o(t)) \cdot \frac{\partial L}{\partial \boldsymbol{o}}(t) \tag{38}$$

We introduce the following notations to describe the 2 contributions of the output gates to the gradients of the current LSTM layer when back-propagating the gradient through (9):

$$\boldsymbol{\epsilon}_o(t) = \boldsymbol{V}_o^T \cdot \frac{\partial L}{\partial \boldsymbol{a}_o}(t) \tag{39}$$

$$\boldsymbol{\gamma}_o(t) = \boldsymbol{u}_o \odot \frac{\partial L}{\partial \boldsymbol{a}_o}(t) \tag{40}$$

Then, we can compute:

$$\frac{\partial L}{\partial \boldsymbol{c}}(t) = \boldsymbol{J}_{\sigma_z}(\boldsymbol{c}(t)) \cdot \left(\left(\frac{\partial L}{\partial \boldsymbol{z}}(t) + \boldsymbol{\epsilon}(t+1)\right) \odot \boldsymbol{o}(t)\right) + \boldsymbol{\gamma}(t) \tag{41}$$

where

$$\boldsymbol{\gamma}(t) = \boldsymbol{\gamma}_o(t) + \boldsymbol{\gamma}_c(t+1) + \boldsymbol{\gamma}_f(t+1) + \boldsymbol{\gamma}_i(t+1) \tag{42}$$

with $\boldsymbol{\gamma}(t_f) = \boldsymbol{\gamma}_o(t_f)$ since no contribution to the gradient can come from after the end of the sequence.

Then, back-propagating through (8) using the chain rule, we get:

$$\frac{\partial L}{\partial \boldsymbol{a}_c}(t) = \boldsymbol{J}_{\sigma_c}(\boldsymbol{a}_c(t)) \cdot \left(\frac{\partial L}{\partial \boldsymbol{c}}(t) \odot \boldsymbol{i}(t)\right) \tag{43}$$

and

$$\boldsymbol{\gamma}_c(t) = \frac{\partial L}{\partial \boldsymbol{c}}(t) \odot \boldsymbol{f}(t) \tag{44}$$

and also the gradients w.r.t. the activation vectors of the input and forget gates:

$$\frac{\partial L}{\partial \boldsymbol{f}}(t) = \frac{\partial L}{\partial \boldsymbol{c}}(t) \odot \boldsymbol{c}(t-1) + \boldsymbol{\beta}_f(t+1) \tag{45}$$

$$\frac{\partial L}{\partial \boldsymbol{i}}(t) = \frac{\partial L}{\partial \boldsymbol{c}}(t) \odot \sigma_c(\boldsymbol{a}_c(t)) + \boldsymbol{\beta}_i(t+1) \tag{46}$$

where

$$\boldsymbol{\beta}_f(t) = \boldsymbol{w}_i \odot \frac{\partial L}{\partial \boldsymbol{a}_i}(t) + \boldsymbol{w}_f \odot \frac{\partial L}{\partial \boldsymbol{a}_f}(t) + \boldsymbol{w}_o \odot \frac{\partial L}{\partial \boldsymbol{a}_o}(t-1) \tag{47}$$

$$\boldsymbol{\beta}_i(t) = \boldsymbol{v}_i \odot \frac{\partial L}{\partial \boldsymbol{a}_i}(t) + \boldsymbol{v}_f \odot \frac{\partial L}{\partial \boldsymbol{a}_f}(t) + \boldsymbol{v}_o \odot \frac{\partial L}{\partial \boldsymbol{a}_o}(t-1) \tag{48}$$

Similarly to what we did for the output gates, we can back-propagate the gradient through both equations in (6):

$$\frac{\partial L}{\partial \boldsymbol{a}_f}(t) = \boldsymbol{J}_{\sigma_f}(\boldsymbol{a}_f(t)) \cdot \frac{\partial L}{\partial \boldsymbol{f}}(t) \tag{49}$$

$$\frac{\partial L}{\partial \boldsymbol{a}_i}(t) = \boldsymbol{J}_{\sigma_i}(\boldsymbol{a}_i(t)) \cdot \frac{\partial L}{\partial \boldsymbol{i}}(t) \tag{50}$$

Now, we introduce the following notations to describe the contributions of the input and forget gates to the gradients of the rest of the LSTM layer when back-propagating the gradient through (4) and (5):

$$\boldsymbol{\epsilon}_f(t) = \boldsymbol{V}_f^T \cdot \frac{\partial L}{\partial \boldsymbol{a}_f}(t) \quad ; \quad \boldsymbol{\epsilon}_i(t) = \boldsymbol{V}_i^T \cdot \frac{\partial L}{\partial \boldsymbol{a}_i}(t) \tag{51}$$

$$\boldsymbol{\gamma}_f(t) = \boldsymbol{u}_f \odot \frac{\partial L}{\partial \boldsymbol{a}_f}(t) \quad ; \quad \boldsymbol{\gamma}_i(t) = \boldsymbol{u}_i \odot \frac{\partial L}{\partial \boldsymbol{a}_i}(t) \tag{52}$$

Finally, we back-propagate the gradient through (7) :

$$\boldsymbol{\epsilon}_c(t) = \boldsymbol{V}_c^T \cdot \frac{\partial L}{\partial \boldsymbol{a}_c}(t) \tag{53}$$

We can now determine the contribution of the recurrent links in the (36) and (41) for the next computation (i.e., $t-1$):

$$\boldsymbol{\epsilon}(t) = \boldsymbol{\epsilon}_o(t) + \boldsymbol{\epsilon}_s(t) + \boldsymbol{\epsilon}_f(t) + \boldsymbol{\epsilon}_i(t) \tag{54}$$

When we reach $t = 1$, we can compute the first derivatives of L w.r.t. the weight matrices and the bias vectors by summing the contribution of all time steps. Thus, for $g \in \{i, f, c, o\}$, we have:

$$\frac{\partial L}{\partial \boldsymbol{W}_g} = \sum_{t=1}^{t_f} \frac{\partial L}{\partial \boldsymbol{a}_g}(t) \cdot \boldsymbol{x}(t)^T \tag{55}$$

$$\frac{\partial L}{\partial \boldsymbol{V}_g} = \sum_{t=2}^{t_f} \frac{\partial L}{\partial \boldsymbol{a}_g}(t) \cdot \boldsymbol{z}(t-1)^T \tag{56}$$

$$\frac{\partial L}{\partial \boldsymbol{b}_g} = \sum_{t=1}^{t_f} \frac{\partial L}{\partial \boldsymbol{a}_g}(t) \tag{57}$$

as well as for the input, forget and output gates ($g \in \{i, f, o\}$):

$$\frac{\partial L}{\partial \boldsymbol{u}_g} = \sum_{t=2}^{t_f} \frac{\partial L}{\partial \boldsymbol{a}_g}(t) \cdot \boldsymbol{c}(t-1)^T \tag{58}$$

Finally, we can also compute the first derivatives of L w.r.t. the nine new links between the gates for $g \in \{i, f\}$:

$$\frac{\partial L}{\partial \boldsymbol{v}_g} = \sum_{t=2}^{t_f} \frac{\partial L}{\partial \boldsymbol{a}_g}(t) \odot \boldsymbol{i}(t-1) \tag{59}$$

$$\frac{\partial L}{\partial \boldsymbol{w}_g} = \sum_{t=2}^{t_f} \frac{\partial L}{\partial \boldsymbol{a}_g}(t) \odot \boldsymbol{f}(t-1) \tag{60}$$

and

$$\frac{\partial L}{\partial \boldsymbol{v}_o} = \sum_{t=1}^{t_f} \frac{\partial L}{\partial \boldsymbol{a}_o}(t) \odot \boldsymbol{i}(t) \tag{61}$$

$$\frac{\partial L}{\partial \boldsymbol{w}_o} = \sum_{t=1}^{t_f} \frac{\partial L}{\partial \boldsymbol{a}_o}(t) \odot \boldsymbol{f}(t) \tag{62}$$

and for $g \in \{i, f, o\}$:

$$\frac{\partial L}{\partial \boldsymbol{y}_g} = \sum_{t=2}^{t_f} \frac{\partial L}{\partial \boldsymbol{a}_g}(t) \odot \boldsymbol{o}(t-1) \tag{63}$$

Acknowledgment

The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoD/ARL, or the U.S. Government.

References

[1] G. Gelly and J.-L. Gauvain, “Minimum word error training of RNN-based voice activity detection,” in *Proc. Interspeech*, 2015, pp. 2650–2654.
[2] S. Van Gerven and F. Xie, “A comparative study of speech detection methods,” in *Proc. Eurospeech*, vol. 97, 1997.
[3] P. C. Khoa, “Noise robust voice activity detection,” Ph.D. dissertation, School Comput. Eng., Nanyang Technol. Univ., Singapore, 2012.
[4] L. Lamel, L. Rabiner, A. E. Rosenberg, and J. G. Wilpon, “An improved endpoint detector for isolated word recognition,” *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-29, no. 4, pp. 777–785, Aug. 1981.
[5] G. Evangelopoulos and P. Maragos, “Speech event detection using multiband modulation energy,” in *Proc. Interspeech*, 2005, pp. 685–688.

[6] T. Kristjansson, S. Deligne, and P. Olsen, “Voicing features for robust speech detection,” *Entropy*, vol. 2, no. 2.5, pp. 369–372, 2005.

[7] H. Ghaemmaghami, B. J. Baker, R. J. Vogt, and S. Sridharan, “Noise robust voice activity detection using features extracted from the time-domain autocorrelation function,” in *Proc. Interspeech*, 2010.

[8] J. Ramırez, J. C. Segura, C. Benıtez, A. De La Torre, and A. Rubio, “Efficient voice activity detection algorithms using long-term speech information,” *Speech Commun.*, vol. 42, no. 3, pp. 271–287, 2004.

[9] P. K. Ghosh, A. Tsiartas, and S. Narayanan, “Robust voice activity detection using long-term signal variability,” *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 19, no. 3, pp. 600–613, Mar. 2011.

[10] T. Ng *et al.*, “Developing a speech activity detection system for the DARPA RATS program,” in *Proc. Interspeech*, 2012, pp. 1969–1972.

[11] I. Shafran and R. Rose, “Robust speech detection and segmentation for real-time ASR applications,” in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, vol. 1, 2003, pp. I-432–I-435.

[12] G. Saon, S. Thomas, H. Soltau, S. Ganapathy, and B. Kingsbury, “The IBM speech activity detection system for the DARPA RATS program,” in *Proc. INTERSPEECH*, 2013, pp. 3497–3501.

[13] X.-L. Zhang and J. Wu, “Deep belief networks based voice activity detection,” *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 21, no. 4, pp. 697–710, Apr. 2013.

[14] T. Drugman, Y. Stylianou, Y. Kida, and M. Akamine, “Voice activity detection: Merging source and filter-based information,” *IEEE Signal Process. Lett.*, vol. 23, no. 2, pp. 252–256, Feb. 2016.

[15] N. Ryant, M. Liberman, and J. Yuan, “Speech activity detection on youtube using deep neural networks,” in *Proc. Interspeech*, 2013, pp. 728–731.

[16] T. Hughes and K. Mierle, “Recurrent neural networks for voice activity detection,” in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2013, pp. 7378–7382.

[17] S. Tong, H. Gu, and K. Yu, “A comparative study of robustness of deep learning approaches for VAD,” in *Proc. 2016 IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2016, pp. 5695–5699.

[18] F. Eyben, F. Weninger, S. Squartini, and B. Schuller, “Real-life voice activity detection with LSTM recurrent neural networks and an application to Hollywood movies,” in *Proc. 2013 IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2013, pp. 483–487.

[19] J. Martens and I. Sutskever, “Learning recurrent neural networks with hessian-free optimization,” in *Proc. 28th Int. Conf. Mach. Learn.*, 2011, pp. 1033–1040.

[20] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, “Advances in optimizing recurrent networks,” in *Proc. 2013 IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2013, pp. 8624–8628.

[21] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” in *Proc. Int. Conf. Mach. Learn.*, vol. 28, 2013, pp. 1310–1318.

[22] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, “Learning precise timing with LSTM recurrent networks,” *J. Mach. Learn. Res.*, vol. 3, no. Aug, pp. 115–143, 2002.

[23] S. Davis and P. Mermelstein, “Comparison of parametric representations of monosyllabic word recognition in continuously spoken sentences,” *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-28, no. 4, pp. 357–366, Aug. 1980.

[24] J. L. Elman, “Finding structure in time,” *Cogn. Sci.*, vol. 14, no. 2, pp. 179–211, 1990.

[25] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.

[26] P. Baldi, S. Brunak, P. Frasconi, G. Soda, and G. Pollastri, “Exploiting the past and the future in protein secondary structure prediction,” *Bioinformatics*, vol. 15, no. 11, pp. 937–946, 1999.

[27] A. Graves, *Supervised Sequence Labelling With Recurrent Neural Networks*, vol. 385. Berlin, Germany: Springer, 2012.

[28] A. Graves, A.-R. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in *Proc. 2013 IEEE Acoust., Speech, Signal Process.*, 2013, pp. 6645–6649.

[29] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[30] G. Gelly and P. Vernis, “Neural networks as a guidance solution for soft-landing and aerocapture,” in *Proc. AIAA Guid. Navig. Control Conf.* Chicago, IL, USA, 2009, pp. 5664–5685.

[31] J. Sun, B. Feng, and W. Xu, “Particle swarm optimization with particles having quantum behavior,” in *Proc. Congr. Evol. Comput.*, 2004, pp. 325–331.

[32] J. Sun, X. Wu, V. Palade, W. Fang, C.-H. Lai, and W. Xu, “Convergence analysis and improvements of quantum-behaved particle swarm optimization,” *Inf. Sci.*, vol. 193, pp. 81–103, 2012.

[33] X. Fu, W. Liu, B. Zhang, and H. Deng, “Quantum behaved particle swarm optimization with neighborhood search for numerical optimization,” *Math. Probl. Eng.*, vol. 2013, 2013, Art. no. 469723.

[34] R. C. Eberhart and J. Kennedy, “A new optimizer using particle swarm theory,” in *Proc. 6th Int. Symp. Micro Mach. Hum. Sci.*, New York, NY, USA, vol. 1, 1995, pp. 39–43.

[35] M. Clerc and J. Kennedy, “The particle swarm-explosion, stability, and convergence in a multidimensional complex space,” *IEEE Trans. Evol. Comput.*, vol. 6, no. 1, pp. 58–73, Feb. 2002.

[36] S. Funk, “RMSPROP loses to SMORMS3,” 2015. [Online]. Available: http://sifter.org/ simon/journal/20150420.html

[37] M. Riedmiller and H. Braun, “A direct adaptive method for faster backpropagation learning: The RPROP algorithm,” in *Proc. IEEE Int. Conf. Neural Netw.*, 1993, pp. 586–591.

[38] T. Tieleman and G. Hinton, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude,” *COURSERA: Neural Netw. Mach. Learn.*, vol. 4, pp. 26–30, 2012.

[39] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” arXiv: 1412.6980, 2014.

[40] J. Sohl-Dickstein, B. Poole, and S. Ganguli, “An adaptive low dimensional quasi-Newton sum of functions optimizer,” arXiv: 1311.2115, 2013. [Online]. Available: http://arxiv.org/abs/1311.2115

[41] G. Sanders, “NIST open speech-activity-detection evaluation,” Nat. Inst. Std. Technol., Gaithersburg, MD, USA, 2015.

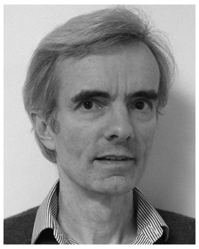
[42] J. Kahn, O. Galibert, L. Quintard, M. Carré, A. Giraudel, and P. Joly, “A presentation of the REPERE challenge,” in *Proc. 2012 10th Int. Workshop Content-Based Multimedia Indexing*, 2012, pp. 1–6.

[43] P. Gay, “Segmentation et identification audiovisuelle de personnes dans des journaux télévisés,” , Université du Maine, Le Mans, France, 2015.

[44] V.-B. Le *et al.*, “Developing STT and KWS systems using limited language resources,” in *Proc. Interspeech*, 2014, pp. 2484–2488.

[45] P. J. Werbos, “Backpropagation through time: What it does and how to do it,” *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990.

Gregory Gelly received the Master’s degree from Ecole Polytechnique, Palaiseau, France, in 2002, and the Master’s degree from ISAE-Supaero, Toulouse, France, in 2003. Since 2014, he has been working toward the Ph.D. degree at the Spoken Language Processing Group, LIMSI-CNRS, Orsay, France, investigating optimization techniques for speech activity detection and language identification based on recurrent neural networks. He was an Aerospace Engineer with Airbus Defense and Space, where he developed innovative guidance algorithms for various spacecrafts. In 2010, he joined the “Car Systems Engineering” team with Airbus Defense and Space to develop optimization methods for a hybrid powered F1 race cars in partnership with Renault Sport F1. In 2012, he was the Head of the Speech Processing Team, French Ministry of Defense, developing speech processing systems based on recurrent neural networks.



Jean-Luc Gauvain received the Doctorate degree in electronics and the HDR (habilitation to direct research) degree in computer science from Paris-Sud University, Orsay, France, in 1982 and 1994, respectively. He is currently a Senior Research Scientist with the CNRS and the Head of the Spoken Language Processing Group, LIMSI, Orsay, France. Since 1983, he has been a permanent CNRS Researcher with LIMSI. From June 1990 to November 1991, he was a Visiting Researcher with AT&T Bell Laboratories. His current research focuses on speech technology including speech recognition, audio indexing, language identification, and speaker recognition. He was the author or coauthor of more than 300 papers in this field and was the recipient of a CNRS silver medal in 2007. He was the Co-Editor-in-Chief for the *Speech Communication* journal for a three-year mandate (2006–2008). He has been actively involved in many speech-related European and U.S. research programs, and was the Scientific Coordinator for the Quaero Research Program from 2008 to 2013. He is an ISCA Fellow.