A NEURAL NETWORK ALTERNATIVE TO NON-NEGATIVE AUDIO MODELS

Paris Smaragdis[♯,♭] *Shrikant Venkataramani*[♯]

[♯]University of Illinois at Urbana-Champaign
[♭]Adobe Research

ABSTRACT

We present a neural network that can act as an equivalent to a Non-Negative Matrix Factorization (NMF), and further show how it can be used to perform supervised source separation. Due to the extensibility of this approach we show how we can achieve better source separation performance as compared to NMF-based methods, and propose a variety of derivative architectures that can be used for further improvements.

1. INTRODUCTION

During the last decade, we have seen an increasing use of Non-Negative Models for audio source separation [1, 2]. In this paper we describe an alternative computational approach for such models that makes use of neural networks. The reason for this approach is to take advantage of the multiple conveniences of neural network models that allow us to design non-negative model variants that are overcomplete, multi-layered, arbitrarily non-linear, have temporal structure, can address non-linear mixtures, etc. Additionally, this approach allows us to effortlessly implement new architectures due to the wealth of automatic differentiation tools available for this purpose. As we will show in this paper, using a neural network approach also allows us to obtain significantly improved results.

In this paper we will address two of the main issues that need to be resolved to implement a non-negative model using a neural network; the calculation of a non-negative basis representation from an audio signal, and the calculation of a non-negative latent state from an audio signal. Using these two steps we can easily replicate most of the existing literature in non-negative models. In the remainder of this paper we will introduce a process for these two calculations and then show how they compare with a traditional non-negative audio model in separation tasks.

2. NON-NEGATIVE AUTOENCODERS

2.1. A non-negative autoencoder architecture

The well-known K-rank Non-Negative Matrix Factorization (NMF) model as introduced in [3] is defined as:

$$\mathbf{X} \approx \mathbf{W} \cdot \mathbf{H} \tag{1}$$

where $\mathbf{X} \in \mathbb{R}_{\geq 0}^{M \times N}$ is a non-negative input matrix to approximate, and $\mathbf{W} \in \mathbb{R}_{\geq 0}^{M \times K}$, $\mathbf{H} \in \mathbb{R}_{\geq 0}^{K \times N}$ are the two non-negative factors, commonly referred to as bases and activations respectively. The set $\mathbb{R}_{\geq 0}^{M \times N}$ is that of real, non-negative matrices of size $M \times N$. This factorization has been the core element for many source separation methods in the last few years [1, 2].

Let us now reinterpret the NMF model as a linear autoencoder. The obvious formulation is:

$$\begin{aligned} &\text{1}^{\text{st}}\text{ layer:} \quad \mathbf{H} = \mathbf{W}^{\ddagger} \cdot \mathbf{X} \\ &\text{2}^{\text{nd}}\text{ layer:} \quad \hat{\mathbf{X}} = \mathbf{W} \cdot \mathbf{H} \end{aligned} \tag{2}$$

in which we enforce the constraint that $\mathbf{W}, \mathbf{H} \geq 0$. The non-negative matrices $\mathbf{W}$ and $\mathbf{H}$ would correspond to their namesakes in the NMF model, whereas the matrix $\mathbf{W}^{\ddagger}$ would be some form of a pseudoinverse of $\mathbf{W}$ that produces a non-negative $\mathbf{H}$. The output $\hat{\mathbf{X}}$ is the model's approximation to the input $\mathbf{X}$. In autoencoder terminology, the first layer weights $\mathbf{W}^{\ddagger}$ are referred to as the *encoder* (which produces a code representing the input), and the upper layer weights $\mathbf{W}$ are referred to as the *decoder* (which uses the code to reconstruct the input). Although this representation would be functionally equivalent to NMF, it would not exhibit any specific advantage and is more complicated and burdensome to implement. Instead we use a slightly different formulation that, as we will show later on, has more interpretative power and is more in line with common neural network designs. Consider the Non-Negative Autoencoder (NAE) model:

$$\begin{aligned} &\text{1}^{\text{st}}\text{ layer:} \quad \mathbf{H} = g\left(\mathbf{W}^{\ddagger} \cdot \mathbf{X}\right) \\ &\text{2}^{\text{nd}}\text{ layer:} \quad \hat{\mathbf{X}} = g\left(\mathbf{W} \cdot \mathbf{H}\right) \end{aligned} \tag{3}$$

where $g : \mathbb{R}^{M \times N} \mapsto \mathbb{R}_{\geq 0}^{M \times N}$, i.e. an element-wise function that produces non-negative outputs. Well-known examples of such functions in the neural network literature include the rectified linear unit: $g(x) = \max(x, 0)$, the softplus: $g(x) = \log(1 + e^x)$ or even the absolute value function $g(x) = |x|$. By applying such an activation function we ensure that our latent representation $\mathbf{H}$ and that the approximation $\hat{\mathbf{X}}$ are both non-negative. There is no guarantee that the matrices $\mathbf{W}^{\ddagger}$ and $\mathbf{W}$ will be non-negative, but that is not a necessary constraint as long as the output and latent state are.

There are of course many ways to estimate the two weight matrices $\mathbf{W}^{\ddagger}$ and $\mathbf{W}$, but for the remainder of this paper we will use the following approach. The entire input $\mathbf{X}$ will be used as a single batch and the parameter updating will be estimated using the RProp algorithm [4]. For the activation function we will use the softplus function [5]. We will use the cost function from [3]:

$$D(\mathbf{X}, \hat{\mathbf{X}}) = \sum_{i,j} \left(\mathbf{X}_{i,j} \left[\log(\mathbf{X}_{i,j}) - \log(\hat{\mathbf{X}}_{i,j}) \right] - \mathbf{X}_{i,j} + \hat{\mathbf{X}}_{i,j} \right) \tag{4}$$

where the subscripts i, j acts as indices on the matrix they are applied on.

Partial funding for this work was provided by the National Science Foundation under award number 1453104

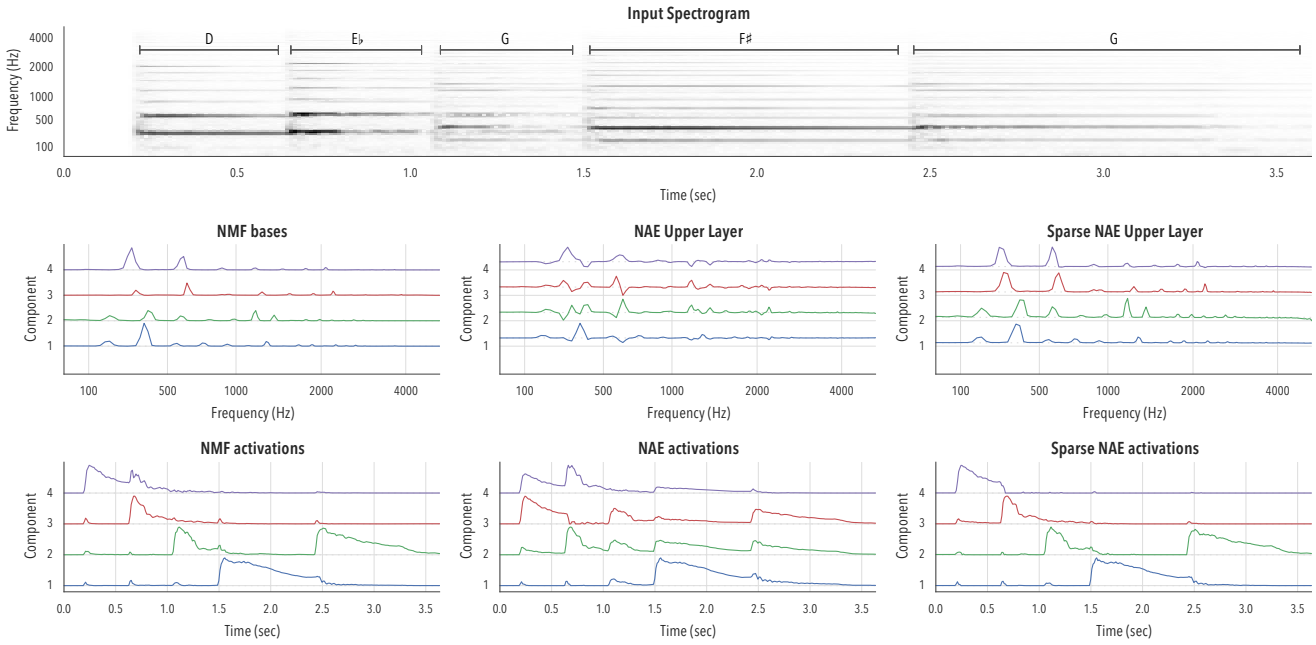

Fig. 1. A comparison between NMF and NAEs for component discovery in spectrograms. The input spectrogram is shown in top plot, and consists of five notes, as labelled. The left plots show the coding discovered by NMF. The NMF bases and activations correctly identify the spectral shape and activation of the four pitches. The middle plots show the results for an NAE, with the upper layer's matrix rows shown at the top plot and the first layer outputs shown at the bottom plot. Although they approximate the input well, they are not as perceptually meaningful. At the right we see the results from a sparse NAE. Asking for a sparse first layer output results in an encoding that's equivalent to NMF.

2.2. Learning a non-negative model

In the context of audio processing, NMF is often used to decompose the magnitude of time-frequency distributions (e.g. the input matrix $\mathbf{X}$ is often a magnitude spectrogram). To illustrate the differences of this model with NMF when using such inputs, consider the following example from [6] shown in figure 1. This is a snippet of a piano performance, with the note sequence $\{\text{D}, \text{E}^\flat, \text{G}, \text{F}^\sharp, \text{G}\}$, which is four distinct pitches (each having a different spectrum), with the pitch G repeated twice making this a five-note sequence. This structure can be clearly seen in the spectrogram in figure 1.

As shown in [6], we can extract that information by modeling the magnitude spectrogram using equation 1 with a rank of 4. We set $\mathbf{X}$ as the input spectrogram, and upon convergence we obtain $\mathbf{W}$ and $\mathbf{H}$, which will respectively hold the model's four *bases* and their corresponding four *activations*. Upon visual examination we see that the columns of $\mathbf{W}$ resemble the spectra of the four notes in the input, whereas the rows of $\mathbf{H}$ activate that spectra at the right points in time to construct the input. These are shown in the middle and bottom left plots in figure 1.

We now turn to the problem of estimating a similar encoding using a neural network. Using the model in equation 3 we obtain the matrices $\mathbf{W}$ and $\mathbf{H}$. These essentially represent an NMF model, albeit with $\mathbf{W}$ potentially having negative values, and having the non-negativity of the output being enforced by the nonlinearity $g(\cdot)$ (which is set to softplus in this case). This model learns a good representation, but it isn't as intuitive as the NMF model. We see that the spectral bases take on negative values which will result in some cross-cancellation being used for the approximation, thereby obfuscating the component-wise additive structure of the model.

One way to resolve the problem of basis cross-cancellations is to use regularization. We see that in this case multiple bases are activated simultaneously, forcing each unique spectrum in the input to be represented by multiple bases at a time. This is a very redundant coding of the input resulting in an unnecessarily busy activation pattern. By adding a sparsity regularizer on $\mathbf{H}$ we can obtain a more efficient coding of the input and minimize activation redundancy. We do so by extending the cost function in equation 4 by:

$$\mathcal{L} = D(\mathbf{X}, \hat{\mathbf{X}}) + \lambda ||\mathbf{H}||_1 \tag{5}$$

We repeat the above experiment using this new cost function and report the result in the right plots of figure 1. As is clearly evident, this model learns a representation which is qualitatively equivalent to NMF (in fact it is slightly more efficient due to the regularization).

2.3. Learning a latent representation given a model

Having learned a model for a sound, we now turn to the problem of extracting the activations $\mathbf{H}$ for an input sound if the bases $\mathbf{W}$ are already known. This is a crucial process for non-negative audio models since it allows us to explain new signals given already learned models. In the case of NMF this is a very straightforward operation; the estimation is the same as learning the full model but we keep the matrix $\mathbf{W}$ fixed. In the case of the NAE model this operation isn't as obvious. Ideally we would expect to pass a new input through the first layer of a trained NAE and obtain an estimate of $\mathbf{H}$ for that sound, but this is not a reliable estimator when using mixtures of sounds. Fortunately the solution to this problem isn't

complicated and is a reinterpretation of the neural network training process.

Consider the model in equation 3. For the task at hand we will be given an input spectrogram $\mathbf{X}$ and a learned model $\mathbf{W}$ and we would have to estimate an $\mathbf{H}$ such that $\hat{\mathbf{X}} \approx \mathbf{X}$. This essentially becomes a single-layer non-linear network:

$$\hat{\mathbf{X}} = g(\mathbf{W} \cdot \mathbf{H}) \tag{6}$$

In usual neural network problems we would be given a target $\mathbf{X}$ with corresponding inputs $\mathbf{H}$ and would be expected to learn the model weights $\mathbf{W}$. What we have to solve in this case is a complementary problem, where we are given the targets and weights but we need to estimate the inputs. This is of course simply the original problem with the dot product operands swapped, and we can easily solve it using simple gradient backpropagation[1] (or any other variants of neural network learning).

2.4. Extensions

Since we now make use of a neural network framework we can easily implement extensions of this model. The most obvious case would be the one of a multilayered (or deep) network, as opposed to the *shallow* model presented above. In this case we implement the NAE as:

$$\begin{aligned} \mathbf{Y}_0 &= \mathbf{X} \\ \mathbf{Y}_i &= g(\mathbf{W}_i \cdot \mathbf{Y}_{i-1}), i = 1, 2, ..., 2L \\ \mathbf{H} &= \mathbf{Y}_L \\ \hat{\mathbf{X}} &= \mathbf{Y}_{2L} \end{aligned} \tag{7}$$

where we use $2L$ layers overall, and we ensure that the layer sizes are symmetric about the middle, i.e. that if $\mathbf{W}_i \in \mathbb{R}^{M \times N}$ then $\mathbf{W}_{2L+1-i} \in \mathbb{R}^{N \times M}$. The output of the L'th layer $\mathbf{H}$ will be the latent representation. This model effectively uses the first L layers as an encoder to produce a latent representation, and then uses the upper L layers as a decoder and produces an approximation of the output $\hat{\mathbf{X}}$. Just as before, we minimize the previously used cost function between the network's output $\hat{\mathbf{X}}$ and input $\mathbf{X}$, and train using the same methods as before. This kind of model will allow us to use more complex representations of the input with a richer dictionary, which would be impossible to simulate with NMF.

Additionally, we can use more exotic layer types, and implement each layer using a recurrent neural network (RNN) and its variants [7, 8] to make use of temporal context, or use convolutional layers [9] to make use of time/frequency context, or any of the many flavors of neural network layers that are available today. In this paper we will limit our discussion to the two models explicitly described above, but it is very easy to implement any other layer type for additional modeling power.

3. SUPERVISED SEPARATION

We now turn our attention to the problem of supervised separation [10]. In this setting, NMF is often used to learn an a priori model of the types of sounds, and then once presented with a new mixture containing such sounds, the learned models are used to decompose that mixture into the contribution of each source. We therefore have two steps or processing, one being the training of the source models, and the other being the fitting of these models on a mixture sound. We have addressed both of these problems in the previous sections, for the source separation problem we need to add a couple more details discussed below.

The first step for this source separation process is to learn models of the sounds we expect to encounter. We can simply do that independently for each sound type using the methodology shown in section 2.2. The only information that we need to retain would be the decoders of the learned NAEs, which will be used to compose an approximation of the input mixture.

Once the decoders are obtained, the next step is to use them simultaneously to explain a mixture containing sounds relating to them. To do that we will combine them using the following setup:

$$\begin{aligned} \hat{\mathbf{X}}_1 &= g(\mathbf{W}_1 \cdot \mathbf{H}_1) \\ \hat{\mathbf{X}}_2 &= g(\mathbf{W}_2 \cdot \mathbf{H}_2) \\ \hat{\mathbf{X}} &= \hat{\mathbf{X}}_1 + \hat{\mathbf{X}}_2 \end{aligned} \tag{8}$$

where $\mathbf{W}_i$ are the already obtained decoders, one for each sound class. We use each decoder to approximate an output $\hat{\mathbf{X}}_i$ and then we sum these outputs to produce an approximation of the input mixture $\mathbf{X}$. The only parameters we can adjust to achieve this approximation are $\mathbf{H}_i$, the latent representations of the two models. Conceptually this problem isn't much different than the problem in section 2.3 and is easy to solve using standard methods. One optional change we can make at this point is to add one more regularizer to discourage models being active simultaneously in a redundant way. We do so by setting the cost function to:

$$\mathcal{L} = D(\mathbf{X}, \hat{\mathbf{X}}) + \lambda \sum_k ||\mathbf{H}_k||_1 \tag{9}$$

This regularizer usually results in a modest improvement, but is by no means necessary.

In the case of a multilayer NAE (or any other layer type), the above equations need to be extended to include the entire decoder of the pretrained models. To explain a mixture we would only need to estimate just the inputs to the first layer of these decoders, and then sum their outputs.

4. EXPERIMENTS

We now present some experiments separating speech mixtures to compare the NAE approach to a traditional NMF method. We composed 32 0dB mixtures of two random TIMIT speakers [11], and for each speaker's 10 sentences we use 9 to train a speaker model and 1 to use in the mixture. For preprocessing we take the magnitude spectrogram of the mixture using a 512pt DFT, applying a square-root Hann window, and a hop size of 25%. The magnitude spectra of the training data and the mixture are being used as inputs to an NMF or NAE estimator. To reconstruct the extracted sources from $\hat{\mathbf{X}}_i$ we use:

$$s_i(t) = \mathrm{STFT}^{-1}\left(\frac{\hat{\mathbf{X}}_i}{\sum_j \hat{\mathbf{X}}_j} \odot \mathbf{X} \odot e^{\mathrm{i}\mathbf{\Phi}}\right) \tag{10}$$

where $\mathbf{X}_i$ is the estimated magnitude spectrogram of the i'th source, and $\mathbf{\Phi}$ is a matrix containing the phase of the input mixture. The operator $\odot$ denotes element-wise multiplication, and $\mathrm{STFT}^{-1}(\cdot)$ is the inverse spectrogram, which produces a waveform from a complex-valued spectrogram. We run two experiments to measure the performance of this approach. We measured the success of separation using the median BSS_EVAL metrics [12] and STOI index [13].

[1] by transposition: $\hat{\mathbf{X}}^\top = g((\mathbf{W} \cdot \mathbf{H})^\top) = g(\mathbf{H}^\top \cdot \mathbf{W}^\top)$ this becomes the same as a generic training problem where we can estimate $\mathbf{H}^\top$ by pretending it is a weight matrix

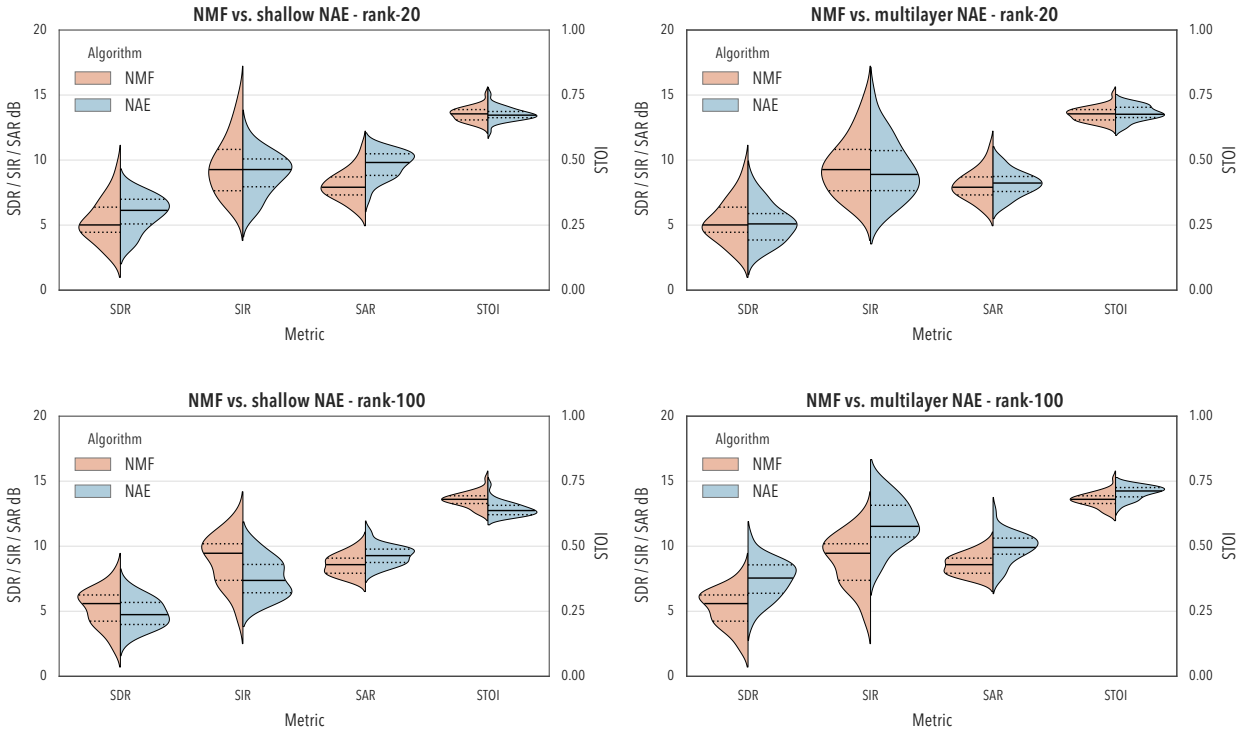

Fig. 2. Comparison of source separation performance on speech/speech mixtures between NMF and NAEs. The left-facing (pink) distributions are of NMF separation results, whereas the right-facing (blue) distributions are of NAE results. The thick solid line in each distribution shows the median value over all experiments, and the dashed lines delimit the corresponding interquartile range. The top plots compare the results between rank-20 NMF with a 20 unit NAE (left), and a rank-20 NMF with a four-layer NAE ($L = 2$) with 20 units in each layer (right). The bottom plots show the same type of comparison for models with rank 100.

The first experiment compared the ability of the NAE model to resolve mixtures when using various layer sizes. The results of are shown in figure 3. We see that for a shallow NAE (equation 3) performance peaks around 20 components (roughly the same behavior as with NMF separators). For a multilayer NAE (equation 8, $L = 2$ and all layers being the same size), we see that performance increases as we add more components, and peaks at 100.

We also compared a basic NMF separator with a shallow and a multilayer NAE ($L = 2$) of the same size (figure 2). In general, we see that the shallow NAE performs roughly equivalently to NMF separators, albeit with worsening performance when using higher rank decompositions (which is expected since as shown before, shallow NAE performance degrades at large ranks). For the multilayered NAE, we see that it matches NMF performance with a rank of 20, but performs significantly better for a rank of 100 (again this is expected from the results in figure 3). Note that for the large NAE the interquartile range of the results is above the interquartile range of NMF, implying a consistently better performance.

5. CONCLUSIONS

In this paper we presented an alternative approach to applying non-negative models for source separation. We show that this approach results in significantly improved performance, and that it lends itself to a wealth of model extensions that would be difficult to implement using the traditional NMF methodology.

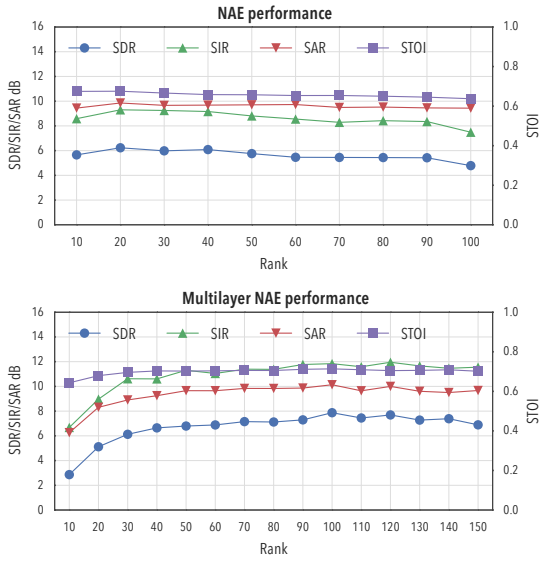

Fig. 3. Performance on speech/speech mixtures of shallow (top) and multilayer (bottom) NAEs with varying number of components. For speech/noise mixtures the results are generally 2 to 4 dB higher.

6. REFERENCES

[1] Paris Smaragdis, Cedric Fevotte, Gautham J Mysore, Nasser Mohammadiha, and Matthew Hoffman, "Static and dynamic source separation using nonnegative factorizations: A unified view," *IEEE Signal Processing Magazine*, vol. 31, no. 3, pp. 66–75, 2014.

[2] Tuomas Virtanen, Jort Florent Gemmeke, Bhiksha Raj, and Paris Smaragdis, "Compositional models for audio processing: Uncovering the structure of sound mixtures," *IEEE Signal Processing Magazine*, vol. 32, no. 2, pp. 125–144, 2015.

[3] Daniel D Lee and H Sebastian Seung, "Algorithms for non-negative matrix factorization," in *Advances in neural information processing systems*, 2001, pp. 556–562.

[4] Martin Riedmiller and Heinrich Braun, "A direct adaptive method for faster backpropagation learning: The rprop algorithm," in *Neural Networks, 1993., IEEE International Conference On*. IEEE, 1993, pp. 586–591.

[5] Xavier Glorot, Antoine Bordes, and Yoshua Bengio, "Deep sparse rectifier neural networks.," in *Aistats*, 2011, vol. 15, p. 275.

[6] Paris Smaragdis and Judith C Brown, "Non-negative matrix factorization for polyphonic music transcription," in *Applications of Signal Processing to Audio and Acoustics, 2003 IEEE Workshop on.* IEEE, 2003, pp. 177–180.

[7] Felix A Gers and E Schmidhuber, "Lstm recurrent networks learn simple context-free and context-sensitive languages," *IEEE Transactions on Neural Networks*, vol. 12, no. 6, pp. 1333–1340, 2001.

[8] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[9] Yann LeCun and Yoshua Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, pp. 1995, 1995.

[10] Paris Smaragdis, Bhiksha Raj, and Madhusudana Shashanka, "Supervised and semi-supervised separation of sounds from single-channel mixtures," in *International Conference on Independent Component Analysis and Signal Separation*. Springer, 2007, pp. 414–421.

[11] John S Garofolo, Lori F Lamel, William M Fisher, Jonathan G Fiscus, David S Pallett, Nancy L Dahlgren, and Victor Zue, "Timit acoustic-phonetic continuous speech corpus," *Linguistic data consortium, Philadelphia*, vol. 33, 1993.

[12] C Févotte, R Gribonval, and E Vincent, "Bss eval toolbox user guide. irisa, rennes," Tech. Rep., France, Tech. Rep. 1706, 2005.

[13] Cees H Taal, Richard C Hendriks, Richard Heusdens, and Jesper Jensen, "An algorithm for intelligibility prediction of time–frequency weighted noisy speech," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 7, pp. 2125–2136, 2011.