2020 IEEE INTERNATIONAL WORKSHOP ON MACHINE LEARNING FOR SIGNAL PROCESSING, SEPT. 21–24, 2020, ESPOO, FINLAND

ROBUST CLASSIFICATION USING HIDDEN MARKOV MODELS AND MIXTURES OF NORMALIZING FLOWS

Anubhab Ghosh, Antoine Honoré, Dong Liu, Gustav Eje Henter, Saikat Chatterjee

School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Sweden

ABSTRACT

We test the robustness of a maximum-likelihood (ML) based classifier where sequential data as observation is corrupted by noise. The hypothesis is that a generative model, that combines the state transitions of a hidden Markov model (HMM) and the neural network based probability distributions for the hidden states of the HMM, can provide a robust classification performance. The combined model is called normalizing-flow mixture model based HMM (NMM-HMM). It can be trained using a combination of expectation-maximization (EM) and backpropagation. We verify the improved robustness of NMM-HMM classifiers in an application to speech recognition.

Index Terms— Speech recognition, generative models, hidden Markov models, neural networks.

1. INTRODUCTION

Neural networks have received much attention in the last decade for their success in regression and pattern classification. Discriminative systems based on neural networks have often been found to outperform many classical classification methods. The systems are data-driven and model-free. The power of neural networks is in non-linear signal transformations through multiple layers. An educated guess is that the non-linear transformations produce appropriate features for classification. While explored as an active research area, neural networks and deep architectures have been criticised for low explainability.We will provide a clear example where neural networks have lack explainability.

Discriminative classification setups optimize a cost function that directly uses a discrimination measure between classes to achieve a high classification performance. It is used to answer pre-fixed queries. The number of classes and class labels are fixed in the optimization and prediction as well. The optimization is known as 'discriminative training'.

Neural network based discriminative classification methods are known to suffer from robustness issues. It is found that a small, often imperceptible amount of noise, in the input signal / features can lead to wrong classifications. Small perturbations known as 'adversarial perturbations', can be engineered to confuse a neural network based discriminative classifier [1], [2]. The reason for this lack of robustness is not fully understood. This is a clear example of where we lack explanations.

We perceive that data-driven, model-free, discriminative classification systems are hard to analyze. Lack of analysis with mathematical tractability hinders finding the reasons for failures or methods to mitigate the issue.

Model-based systems can be amenable for scrutiny and tractable mathematical analysis, which is a basic precursor of explanations. We propose to combine the analytical tractability of model-based systems and the non-linear transformation advantage of neural networks. The combination will provide a better classification performance than a set of pure model-based systems. The combined models will provide robustness. Robustness can be experimentally verified using real-world data and various types of noises. In this article our main objective is to provide such an experimental verification.

In pursuit of our objective, we use speech recognition as a real application scenario. Hidden Markov models (HMMs) are time-tested in speech recognition [3]. HMMs can model sequential data and have been used in a variety of applications including handwriting recognition, activity recognition, genetic signal processing, transport forecasting, etc. For speech recognition, probability distributions of states in an HMM are classically modelled using Gaussian mixture models (GMMs). The GMM-HMM combination is purely model-driven. GMM-HMMs are generative models, meaning they can explain how sequential data is generated. It can be trained using time-tested probabilistic methods, such as expectation-maximization (EM), variational Bayes (VB) and Markov-Chain-Monte-Carlo (MCMC). After training, GMM-HMM models can be used directly for ML based classification.

ML based classification using generative models addresses Bayes minimum risk criterion. It can easily accommodate a growing number of classes. Adversarial perturbation is not relevant in the case where ML-based classification is performed using generative models. However, robustness for various types of noises remains an important issue.

Recent advances have improved the ability of the neural network based models to describe probability distribu-

978-1-7281-6662-9/20/$31.00

tions. EM was used for training mixture models based on 'normalizing-flow' in [4]. Normalizing-flow is a generative model [5], [6]. Normalizing-flow mixture models (NMMs) are also generative models and may handle multiple modes and manifolds in a data distribution better than GMMs. In [7], NMMs were used for modelling the state distributions in an HMM. In this article, we call this model as NMM-HMM in contrast to GMM-HMM. NMM-HMM parameters can be learned using EM and backpropagation. Our main technical contribution in this article is to show than the NMM-HMM is more robust than GMM-HMM for speech recognition at various noisy conditions in ML-based classification. We verify this in an extensive experimental study of robust phone recognition on the TIMIT database using the Kaldi and PyTorch toolkits.

1.1. Relevant literature survey for speech recognition

Phonemes can be thought of as one of the fundamental units of speech sounds, that indicate the pronunciation of a word. A spoken utterance, which is essentially a sequence of words, is composed of phonemes. The task of phone recognition involves creating a system that can be given input as a human spoken utterance, and the system outputs a sequence of phones that indicate the pronunciation of the actual spoken text. Speech phone recognition is popular in the research community because it is a fundamental task in developing a speech recognition system [8]. It is also inherently free from the limitations of vocabulary. The phone recognition accuracy continues to improve [9].

Neural networks are recently much used for speech recognition, mainly in discriminative training setups. A list of neural network based methods and their performances for phone recognition can be found in the Github site [9]. An earlier attempt is to use restricted Boltzmann machines (RBMs) to form a deep belief network (DBN) that served as the acoustic model for HMM [10]. The DBN used a softmax output layer and was trained discriminatively using backpropagation, achieving a phone recognition accuracy of around 77% on the TIMIT dataset.

Dynamical system models have also been used for sequence-to-sequence classification. The dynamical neural networks are recurrent neural networks (RNNs), long-short-term-memory networks (LSTMs), and their gated recurrent unit based simplifications. Attention mechanisms can further improve performance. Example works are [11], [12]. Almost all these example works use discriminative training. All these methods used a softmax function to represent the phone probability estimation. The best phone recognition accuracy result so far known using discriminative training is 85.1% [13]. Finally, the authors in [14] proposed a hybrid HMM based on PyTorch-Kaldi toolkit, whose phone recognition accuracy is mentioned as 86.2%.

2. NMM-HMM

Our interest in this article is ML-based classification. Let the number of classes be C. We denote the generative model for c'th class as $\mathbf{H}_c$. We write a data sequence $\underline{\mathbf{x}} = [\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_T]^T$ (where the superscript T denotes transpose), where each $\mathbf{x}_t \in \mathbb{R}^D$ is the feature vector extracting from the signal at time t and T denotes the sequence length. Then the ML-classification problem is

$$c^\star = \arg\max_c p(\underline{\mathbf{x}}|\mathbf{H}_c) \tag{1}$$

where $p()$ denotes likelihood. Our interest is to use NMM-HMM as $\mathbf{H}_c$ instead of GMM-HMM. The NMM-HMM system has been described in detail in [7]. They called their model 'Gen-HMM'. We rename Gen-HMM as NMM-HMM (Normalizing-flow Mixture Model based HMM) since we compare with GMM-HMM. We restate some essential theory in the remainder of the section about NMM-HMM for consistency and completeness.

2.1. Central principle behind NMM-HMM

The framework used for implementing the normalizing flow-based model is a Hidden Markov Model (denoted by $\mathbf{H}$). An HMM is basically composed of the following quantities: the set of hidden states of the HMM denoted by S, the initial probability vector for the states of the HMM denoted by $\mathbf{q}$, the state-transition probability matrix denoted by $\mathbf{A}$, and the output probability distribution for each state s denoted by $p\left(\mathbf{x}|s;\mathbf{\Psi}_s\right)$ and parameterised by a set of parameters $\mathbf{\Psi}_s$.The probabilistic model of the NMM-HMM for each hidden state is a weighted mixture of K density functions that is defined as:

$$p\left(\mathbf{x}|s;\mathbf{\Psi}_s\right) = \sum_{k=1}^{K} \pi_{s,k} p\left(\mathbf{x}|s;\phi_{s,k}\right) \tag{2}$$

In (2), the weights $\pi_{s,k}$ denote the probability of drawing a given component k from a categorical distribution with $\pi_{s,k} = p\left(k|s;\mathbf{H}\right)$ and they satisfy $\sum_{k=1}^{K} \pi_{s,k} = 1$ for each s. The feature vector $\mathbf{x}$ is considered to be generated from a flow-based *generator* function $\mathbf{g}_{s,k} : \mathbb{R}^D \rightarrow \mathbb{R}^D$, such that $\mathbf{x} = \mathbf{g}_{s,k}\left(\mathbf{z}\right)$, where $\mathbf{z}$ is a D-dimensional latent variable that is drawn from a known prior distribution. As we are considering normalizing flow models, $p\left(\mathbf{z}\right)$ is assumed to be a standard multivariate normal distribution. Assuming the function $\mathbf{g}_{s,k}$ is invertible, and the corresponding *normalizing* function is $\mathbf{f}_{s,k}$ (or equivalently $\mathbf{g}_{s,k}^{-1}$), s.t. $\mathbf{z} = \mathbf{f}_{s,k}\left(\mathbf{x}\right)$, we have:

$$p\left(\mathbf{x}|s;\mathbf{\Phi}_{s,k}\right) = p_{s,k}\left(\mathbf{f}_{s,k}\left(\mathbf{x}\right)\right) \left|\det\left(\frac{\partial \mathbf{f}_{s,k}\left(\mathbf{x}\right)}{\partial \mathbf{x}}\right)\right| \tag{3}$$

This equation shows that the density computation is exact and tractable. In practice, a logarithm is applied on both sides of

(3) to transform it into a sum, and the log-likelihood of the signal is calculated using the log-probability of the prior and the log-determinant of the Jacobian.

2.2. Learning problem formulation in NMM-HMM

The NMM-HMM model is intended to be used for modeling a sequential signal such as $\underline{\mathbf{x}}$, having an empirical distribution of the dataset as $\hat{d}\left(\underline{\mathbf{x}}\right)$ and the maximum likelihood maximization problem that is required to solve is:

$$\arg\max_{\mathbf{H}} \frac{1}{R}\sum_{r=1}^{R}\log\left(p\left(\underline{\mathbf{x}}^{r};\mathbf{H}\right)\right) \tag{4}$$

where r denotes the index of the sequential signal in the training data, R denotes the total number of sequential input signals under consideration and H denotes the HMM model under consideration from the possible hypothesis set of models. This problem can be solved using the well known Expectation-Maximization framework. The Expectation step ("E-step") involves the calculation of the posterior probability distribution of hidden sequences $\underline{\mathbf{s}}$ and $\underline{\mathbf{k}}$ to obtain an expected value of the likelihood of the data sequence $\underline{\mathbf{x}}$ under the current model parameters $\mathbf{H^{old}}$.". This is shown as follows:

$$L\left(\mathbf{H};\mathbf{H}^{old}\right)=\mathbf{E}_{p(\underline{\mathbf{s}},\underline{\mathbf{k}}|\underline{\mathbf{x}};\mathbf{H}^{old})}\log\left(p\left(\underline{\mathbf{x}},\underline{\mathbf{s}},\underline{\mathbf{k}};\mathbf{H}\right)\right) \tag{5}$$

The second step consists of the Maximisation step ("M-step") consists of finding the model $\mathbf{H}$ that maximizes the expected log-likelihood computed in (5). This can be decomposed into three separate maximization problems as in (6):

$$\begin{aligned}&\max_{\mathbf{H}} L\left(\mathbf{H};\mathbf{H}^{old}\right)\\&=\max_{\mathbf{q}} L\left(\mathbf{q};\mathbf{H}^{old}\right)+\max_{\mathbf{A}} L\left(\mathbf{A};\mathbf{H}^{old}\right)+\max_{\mathbf{\Psi}} L\left(\mathbf{\Psi};\mathbf{H}^{old}\right)\end{aligned} \tag{6}$$

where,

$$L\left(\mathbf{q};\mathbf{H}^{old}\right)=\mathbf{E}_{p(\underline{\mathbf{s}}|\underline{\mathbf{x}};\mathbf{H}^{old})}\log\left(p\left(s_1;\mathbf{H}\right)\right) \tag{7}$$

$$L\left(\mathbf{A};\mathbf{H}^{old}\right)=\mathbf{E}_{p(\underline{\mathbf{s}}|\underline{\mathbf{x}};\mathbf{H}^{old})}\sum_{t=2}^{T}\log\left(p\left(s_t|s_{t-1};\mathbf{H}\right)\right) \tag{8}$$

$$L\left(\mathbf{\Psi};\mathbf{H}^{old}\right)=\mathbf{E}_{p(\underline{\mathbf{s}},\underline{\mathbf{k}}|\underline{\mathbf{x}};\mathbf{H}^{old})}\log\left(p\left(\underline{\mathbf{x}},\underline{\mathbf{k}}|\underline{\mathbf{s}};\mathbf{H}\right)\right) \tag{9}$$

The maximisation problems in (7), (8) can be solved using standard EM forward-backward algorithm to compute the posterior distribution effectively, explained in [15]. For solving the maximisation problem of the last equation 9, we need to maximise the likelihood with respect to the mixture of weights and the set of parameters of the flow model. This would require the computation of the log-determinant for the flow model that is explained in the next section.

2.3. Implementation of individual generator models

Each generator function is realised as a flow model. It maps a given observation from the feature space to a latent space of the same dimension. Every mapping from layer to layer should be bijective (i.e. one-to-one and invertible) and log-determinant in the inverse direction should be easy to compute. The signal flow for a flow model having L layers can be illustrated as follows:

$$\mathbf{z}=\mathbf{h_0}\underset{\mathbf{f}_{s,k}^{[1]}}{\overset{\mathbf{g}_{s,k}^{[1]}}{\rightleftharpoons}}\mathbf{h_1}\underset{\mathbf{f}_{s,k}^{[2]}}{\overset{\mathbf{g}_{s,k}^{[2]}}{\rightleftharpoons}}\mathbf{h_2}\underset{\mathbf{f}_{s,k}^{[3]}}{\overset{\mathbf{g}_{s,k}^{[3]}}{\rightleftharpoons}}\mathbf{h_3}\ldots\underset{\mathbf{f}_{s,k}^{[L]}}{\overset{\mathbf{g}_{s,k}^{[L]}}{\rightleftharpoons}}\mathbf{h_L}=\mathbf{x} \tag{10}$$

where $\mathbf{f}_{s,k}^{[l]}$ denotes the l^{th} layer network of $\mathbf{f}_{s,k}$, and each such $\mathbf{f}_{s,k}^{[l]}$ is invertible. Flow models have been first proposed in [5]. Different flow model architectures may have different kinds of *coupling* between two successive layers in the network or some other bijective mapping. To illustrate a small section of the mapping from the data space to the latent space, let us consider the input feature at the l^{th} layer denoted by $\mathbf{h}_l$. This feature is mapped to $\mathbf{h}_{l-1}$ using the function $\mathbf{f}_{s,k}$. At every layer the D-dimensional input feature is split into two parts. Let us assume the features are $\left[\mathbf{h}_{l,1:d},\mathbf{h}_{l,d+1:D}\right]^T$ (where d denotes the number of components in the first sub part). The relation is as follows:

$$\begin{aligned}\mathbf{h}_{l,1:d}&=\mathbf{h}_{l-1,1:d}\\\mathbf{h}_{l,d+1:D}&=\mathbf{h}_{l-1,d+1:D}\odot\exp\left(\mathbf{s}\left(\mathbf{h}_{l-1,1:d}\right)\right)+\mathbf{t}\left(\mathbf{h}_{l-1,1:d}\right)\end{aligned} \tag{11}$$

The inverse function ($\mathbf{f}:X\rightarrow Z$) (from data space to latent space, which is used in Jacobian computation during training) is defined as:

$$\begin{aligned}\mathbf{h}_{l-1,1:d}&=\mathbf{h}_{l,1:d}\\\mathbf{h}_{l-1,d+1:D}&=\left(\mathbf{h}_{l,d+1:D}-\mathbf{t}\left(\mathbf{h}_{l,1:d}\right)\right)\odot\exp\left(-\mathbf{s}\left(\mathbf{h}_{l,1:d}\right)\right)\end{aligned} \tag{12}$$

(where $\odot$ denotes element-wise multiplications, $\mathbf{s}:\mathbb{R}^d\rightarrow\mathbb{R}^{D-d}$, $\mathbf{t}:\mathbb{R}^d\rightarrow\mathbb{R}^{D-d}$, with $\mathbf{s},\mathbf{t}$ being shallow feed-forward Neural Nets that differ only in the activation function for the last layer, which is a hyperbolic tangent (**tanh**) activation function for the scale function ($\mathbf{s}$) (modeling logarithm of standard deviation) and an identity activation for the translation function ($\mathbf{t}$)[16]. For the flow model, the inverse mapping shown in (12) is computed for every layer and the determinant of the Jacobian matrix is computed as the product of the determinants of the Jacobian matrices at every layer:

$$\det\left(\nabla\mathbf{f}_{s,k}\right)=\prod_{l=1}^{L}\det\left(\nabla\mathbf{f}_{s,k}^{[l]}\right) \tag{13}$$

where each $\det\left(\nabla \mathbf{f}_{s,k}^{[l]}\right)$ is computed as:

$$\begin{aligned}\det\left(\nabla \mathbf{f}_{s,k}^{[l]}\right) &= \det\left(\begin{bmatrix} \mathbf{I}_{1:d} & \mathbf{0} \\ \frac{\partial \mathbf{h}_{l-1,d+1:D}}{\partial \mathbf{h}_{l-1,1:d}} & diag\left(-\mathbf{s}\left(\mathbf{h}_{l,1:d}\right)\right) \end{bmatrix}\right) \\ &= \det\left(diag\left(-\mathbf{s}\left(\mathbf{h}_{l,1:d}\right)\right)\right)\end{aligned} \tag{14}$$

In (14), $\mathbf{I}_{1:d}$ denotes an identity matrix and $diag(\dots)$ denotes a diagonal matrix with the elements of the vector in the main diagonal. It should be kept in mind and an alternate ordering between the two parts of the signal $[\mathbf{h}_{l,1:d}, \mathbf{h}_{l,d+1:D}]^T$, described in the affine coupling layer in (11), is required to avoid the problem of partial identity mapping [16] in the model. For solution of (9), the problem can be broken down into two parts: learning the set of mixture of weights: $\mathbf{\Pi} = \{\pi_{\mathbf{s}} | s \in S\}$, and the learning the set of flow model parameters: $\mathbf{\Phi} = \{\phi_{\mathbf{s}} | s \in S\}$. This is shown as:

$$\max_{\mathbf{\Psi}} L\left(\mathbf{\Psi}; \mathbf{H}^{old}\right) = \max_{\mathbf{\Pi}} L\left(\mathbf{\Pi}; \mathbf{H}^{old}\right) + \max_{\mathbf{\Phi}} L\left(\mathbf{\Phi}; \mathbf{H}^{old}\right) \tag{15}$$

The problem of learning the mixture of weights can be solved using a simple lagrangian formulation while problem of learning the flow model parameters can be solved by using the results of the change of variable (3) and the log-determinant derived in (14) as:

$$\begin{aligned}L\left(\mathbf{\Phi}; \mathbf{H}^{old}\right) &= \mathbf{E}_{p(\underline{\mathbf{s}}, \underline{\mathbf{k}} | \underline{\mathbf{x}}; \mathbf{H}^{old})} \log\left(p\left(\underline{\mathbf{x}} | \underline{\mathbf{s}}, \underline{\mathbf{k}}; \mathbf{H}\right)\right) \\ &= \mathbf{E}_{p(\underline{\mathbf{s}}, \underline{\mathbf{k}} | \underline{\mathbf{x}}; \mathbf{H}^{old})}[\log\left(p_{\underline{\mathbf{s}}, \underline{\mathbf{k}}}\left(\mathbf{f}_{s,k}\left(\underline{\mathbf{x}}\right)\right)\right) \\ &\qquad + \log\left(\left|\det\left(\nabla \mathbf{f}_{s,k}\right)\right|\right)]\end{aligned} \tag{16}$$

The first term on the right hand side of (16) is basically an expectation computed over the log-probability of latent data derived using the normalizing function $\mathbf{f}_{s,k} : X \rightarrow Z$, and the second term is the result of the log-determinant of the Jacobian that is computed using (13) and (14). The essential steps of the learning procedure are described in psuedocode in Algorithm 1.

Algorithm 1: Learning Algorithm of NMM-HMM

Input: Dataset $\underline{\mathbf{x}}$, initial model parameters and the empirical distribution of the dataset $\hat{d}\left(\underline{\mathbf{x}}\right)$
Result: Optimized model parameters: $\mathbf{q}, \mathbf{A}, \mathbf{\Pi}, \mathbf{\Phi}$
Set learning rate (η), no. of mini-batches, max. epochs for flows (N_{max})
Initialization of $\mathbf{H^{old}}$, $\mathbf{H}$: where, $\mathbf{H} = \{\mathbf{q}, \mathbf{A}, S, p\left(\mathbf{x}|s; \mathbf{\Psi}_s\right)\}$, and
set $\mathbf{H^{old}} \leftarrow \mathbf{H}$
while $\mathbf{H}$ *has not converged* **do**
  for *num_epochs* $\leq N_{max}$ **do**
    Input a mini-batch from the dataset as $\{\underline{\mathbf{x}}^r\}_{r=1}^{R_b}$, with batch-size R_b
    Compute the posterior $p\left(\underline{\mathbf{s}}^r, \underline{\mathbf{k}}^r | \underline{\mathbf{x}}^r; \mathbf{H}^{old}\right)$ and the loss function $L\left(\mathbf{\Phi}; \mathbf{H}^{old}\right)$ shown in (16)
    Compute $\partial\mathbf{\Phi}$ by optimising $L\left(\mathbf{\Phi}; \mathbf{H}^{old}\right)$
    Update $\mathbf{\Phi} \leftarrow \mathbf{\Phi} + \eta \cdot \partial\mathbf{\Phi}$
  end
  $\mathbf{q} \leftarrow \arg\max_{\mathbf{q}} L\left(\mathbf{q}; \mathbf{H}^{old}\right)$
  $\mathbf{A} \leftarrow \arg\max_{\mathbf{A}} L\left(\mathbf{A}; \mathbf{H}^{old}\right)$
  $\mathbf{\Pi} \leftarrow \arg\max_{\mathbf{\Pi}} L\left(\mathbf{\Pi}; \mathbf{H}^{old}\right)$
  Update $\mathbf{H^{old}} \leftarrow \mathbf{H}$
end

3. EXPERIMENTS AND RESULTS

This section describes the experimental setup for comparing the performance of the NMM-HMM model and the conventional GMM-HMM model, on the task of phone recognition.

3.1. Dataset and feature extraction

The experiments were carried out using the TIMIT dataset [8], which has utterances labelled at the phoneme level. The speech signal in the dataset has been sampled at 16 kHz, and the dataset consists of 6300 phoneme-level speech utterances that have been split into two sets - A training set consisting of 4620 utterances and a testing set consisting of 1680 utterances. In the original TIMIT dataset there are 61 phones available for classification. There also exists a smaller, 'folded' set of 39 phones mapped from the larger set of phones [8]. The experiments were performed for classification on the folded set of 39 phones. Four varieties of additive noises from from the NOISEX-92 database (also sampled at the same frequency of 16 kHz) were considered and snippets were taken from random offsets in the noise database and were added to the clean test data at different Signal to Noise ratio levels (with respect to the clean test signal) to generate noisy test sets. The features used were Mel frequency Cepstral Coefficients (MFCCs) that were extracted on a per frame basis (25 ms window frames, with a 10ms time shift between successive windows). This resulted in a vector of 13 MFCCs, along with column-wise concatenation of velocity (Δ coefficients) and acceleration coefficients ($\Delta\Delta$ coefficients) into a 39-dimensional feature vector. These feature vectors were subsequently processed, split into training and test sets and arranged on a per-class basis i.e. for each of the 39 phonemes present. This helped to set up the conditions for generative training of the models.

3.2. Model training

The NMM-HMM models for each of the 39 classes of phones were trained using generative training. Training was carried out using Adam [17] optimization, with data processed on a batch-wise basis. For each HMM model, the number of states used was clipped between 3 and 5, with the exact num-

Table 1. Recognition accuracy (in %) for GMM-HMM at varying number of mixture components

Model-Type	No. of components (K)			
	K=3	K=10	K=15	K=20
GMM-HMM	66.7	70.8	71.9	72.8

Table 2. Recognition accuracy (in %) for NMM-HMM at varying number of mixture components

Model-Type	No. of components (K)	
	K=1	K=3
NMM-HMM	76.7	77.6

ber of states used depending upon the mean sequence length of the incoming signal. For the GMM-HMM model, diagonal covariance matrices were considered for modeling each component Gaussian in the model, and the no. of mixture components were to be decided based on the basis of model performance. The state-transition matrix $\mathbf{A}$ for each NMM-HMM model was initialised as an upper triangular matrix. For each NMM-HMM model, a *flow block* was used to refer to a pair of consecutive coupling layers that have been described in (11),(12). It was necessary to ensure that the signal in the l^{th} layer would be alternated by the $(l+1)^{th}$ layer, so that there is a mixing of the signals between consecutive coupling layers and no identity mapping occurs. Each NMM-HMM model consisted of *four* such flow blocks in the implementation, which was found to work well. The evaluation was done using a full forward-backward algorithm, with the metric used as *accuracy* $(100 - PER\%)$ computed on the test set, as a percentage of the number of correct predicted phonemes among the total set of phonemes.

3.3. Results

3.3.1. Clean training and testing

The performance of the NMM-HMM model was compared with the baseline GMM-HMM model for training and testing on clean data. For the purpose of comparison with the baseline model, the GMM-HMM model was trained and tested on the clean data, for varying number of mixture components, i.e. $K = \{3, 10, 15, 20\}$, and the model with the best performance was chosen for comparison with the NMM-HMM model. The results are shown in Table 1. It was found that accuracy increased with more number of mixture components. Based on the results, the model with $K = 20$ components was chosen for comparison. Next we simulated NMM-HMM with varying number of mixture components. Results are shown in Table 2. It was found that $K = 3$ components was best suited for the NMM-HMM model. There was some increase in accuracy by increasing number of components from $K = \{1, 3\}$, but with more number of components $(K > 3)$, the training time increased significantly without much change in test accuracy. Upon comparing Tables 1 and 2, and find that the NMM-HMM model with $K = 3$ mixture components outperformed the baseline GMM-HMM model with $K = 20$ components. The performance improvement is 4.8%.

We finally mention that the NMM-HMM based ML-classification provides a similar classification accuracy for TIMIT phone recognition in comparison with discriminative training based DBN that had 77% accuracy [10].

3.3.2. Clean training and noisy testing - robustness test

The next experiment is for checking the robustness of the NMM-HMM relative to GMM-HMM. We train using clean data and test using noisy data. We used four types of noise: white, pink, babble and high frequency channel (labelled as 'hfchannel') at different signal-to-noise-ratio (SNR) levels. The performance of a robust system is expected to deteriorate relatively slowly as the noise power increases (SNR decreases).

Table 3 shows the results. The performance drop is counted with respect to the clean data performance (as seen in Table 2 of the previous subsection 3.3.1, and shown in parenthesis in Table 3). We find that the performance drop is gradual with decrease in SNR for white, babble and hfchannel noises. While the performance drop is gradual for NMM-HMM in case of pink noise at SNR = 25, 20 and 15 dB, there is a drastic drop for NMM-HMM when the pink noise is at 10 dB SNR. Overall the NMM-HMM shows a significantly greater noise robustness compared to GMM-HMM.

3.3.3. Noisy training and noisy testing - robustness test

In this set of experiments, the models were trained using a joint dataset comprised of clean data and data corrupted with white noise at 10dB SNR. Test results are shown in Table 4. We again find that NMM-HMM is more robust than GMM-HMM.

4. CONCLUSION

In this work, we demonstrated that it is possible to use neural networks for improving maximum-likelihood based classification performance and robustness against noise. In addition the methods are able to use time-tested signal processing based features as MFCCs and machine learning techniques as expectation-maximization for training the models. In the future we can consider use of the input features such as logarithm of the power spectrum.

5. REFERENCES

[1] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai, "One pixel attack for fooling deep neural net-

Table 3. Test accuracy (in %) for clean and various noise conditions. We compare GMM-HMM and NMM-HMM for folded 39-phone classification. We use the notations GMM and NMM to represent GMM-HMM and NMM-HMM, respectively. The performance drop is shown in parenthesis with respect to the clean train and clean test scenario as in Tables 1 and 2.

Performance for clean data training and testing as a reference: GMM: 72.8 and NMM: 77.6								
Type of Noise	SNR levels for different kinds of noises							
	25dB		20dB		15dB		10dB	
	GMM	NMM	GMM	NMM	GMM	NMM	GMM	NMM
white	55.6 (17.2)	67.1 (10.5)	46.8 (26.0)	60.0 (17.6)	36.8 (36.0)	49.4 (28.2)	27.9 (44.9)	37.7 (39.9)
pink	59.9 (12.9)	69.3 (8.3)	51.9 (20.9)	61.7 (15.9)	42.3 (30.5)	48.6 (29)	32.2 (40.6)	33.7 (43.9)
babble	65.7 (7.1)	70.7 (6.9)	59.3 (13.5)	65.8 (11.8)	49.3 (23.5)	56.2 (21.4)	37.4 (35.4)	42.3 (35.3)
hfchannel	62.3 (10.5)	67.9 (9.7)	54.4 (18.4)	63.4 (14.2)	44.1 (28.7)	55.8 (21.8)	33.3 (39.5)	44.9 (32.7)

Table 4. Test accuracy (in %) using noisy training. The performance drop is shown in parenthesis with respect to the clean train and clear test scenario.

Model-Type	Clean	white noise	
		15dB	10dB
GMM-HMM	72.0	55.9 (16.1)	53.7 (18.3)
NMM-HMM	76.8	69.2 (7.6)	65.7 (11.1)

works," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, pp. 828–841, 2019.

[2] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard, "DeepFool: a simple and accurate method to fool deep neural networks," in *Proc. of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2574–2582.

[3] Mark Gales, Steve Young, et al., "The application of hidden Markov models in speech recognition," *Foundations and Trends in Signal Processing*, vol. 1, no. 3, pp. 195–304, 2008.

[4] Dong Liu, Antoine Honoré, Saikat Chatterjee, and Lars K. Rasmussen, "Neural network based explicit mixture models and expectation-maximization based learning," in *International Joint Conference on Neural Networks (IJCNN)*, 2020.

[5] Laurent Dinh, David Krueger, and Yoshua Bengio, "NICE: Non-linear independent components estimation," *Proc. of ICLR 2015 – Workshop Track*, vol. 1, no. 2, pp. 1–13, 2015.

[6] Ivan Kobyzev, Simon Prince, and Marcus A Brubaker, "Normalizing flows: Introduction and ideas," *arXiv preprint arXiv:1908.09257*, 2019.

[7] Dong Liu, Antoine Honoré, Saikat Chatterjee, and Lars K. Rasmussen, "Powering hidden markov model by neural network based generative models," in *European Conference on Artificial Intelligence (ECAI)*, 2020.

[8] Carla Lopes and Fernando Perdigao, "Phone recognition on the TIMIT database," *Speech Technologies/Book*, vol. 1, pp. 285–302, 2011.

[9] Gabriel Synneave, "wer_we_are," GitHub Repository, https://github.com/syhw/wer_are_we, 2015.

[10] Abdel Rahman Mohamed, George Dahl, and Geoffrey Hinton, "Deep belief networks for phone recognition," in *NIPS workshop on deep learning for speech recognition and related applications*, 2009, vol. 1, p. 39.

[11] Alex Graves, Abdel Rahman Mohamed, and Geoffrey Hinton, "Speech recognition with deep recurrent neural networks," *Proc. of ICASSP*, , no. 3, pp. 6645–6649, 2013.

[12] Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio, "Attention-based models for speech recognition," *Advances in Neural Information Processing Systems*, vol. 2015-January, pp. 577–585, 2015.

[13] Mirco Ravanelli, Philemon Brakel, Maurizio Omologo, and Yoshua Bengio, "Light Gated Recurrent Units for Speech Recognition," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 2, pp. 92–102, 2018.

[14] Mirco Ravanelli, Titouan Parcollet, and Yoshua Bengio, "The PyTorch-Kaldi Speech Recognition Toolkit," *Proc. of ICASSP*, vol. 2019-May, pp. 6465–6469, 2019.

[15] Christopher M Bishop, *Pattern recognition and machine learning*, springer, 2006.

[16] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio, "Density estimation using real NVP," *Proc. of ICLR 2017 - Conference Track*, 2019.

[17] Diederik P Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.