DOMAIN ADAPTATION FOR AUTOENCODER-BASED END-TO-END COMMUNICATION OVER WIRELESS CHANNELS

Jayaram Raghuram∗1, Yijing Zeng†1, Dolores García Martí2, Somesh Jha1, Suman Banerjee1, Joerg Widmer2, and Rafael Ruiz Ortiz2

1University of Wisconsin - Madison
2IMDEA Networks Institute, Madrid, Spain

ABSTRACT

The problem of domain adaptation conventionally considers the setting where a source domain has plenty of labeled data, and a target domain (with a different data distribution) has plenty of unlabeled data but none or very limited labeled data. In this paper, we address the setting where the target domain has only limited labeled data from a distribution that is expected to change frequently. We first propose a fast and light-weight method for adapting a Gaussian mixture density network (MDN) using only a small set of target domain samples. This method is well-suited for the setting where the distribution of target data changes rapidly (e.g., a wireless channel), making it challenging to collect a large number of samples and retrain. We then apply the proposed MDN adaptation method to the problem of end-of-end learning of a wireless communication autoencoder. A communication autoencoder models the encoder, decoder, and the channel using neural networks, and learns them jointly to minimize the overall decoding error rate. However, the error rate of an autoencoder trained on a particular (source) channel distribution can degrade as the channel distribution changes frequently, not allowing enough time for data collection and retraining of the autoencoder to the target channel distribution. We propose a method for adapting the autoencoder without modifying the encoder and decoder neural networks, and adapting only the MDN model of the channel. The method utilizes feature transformations at the decoder to compensate for changes in the channel distribution, and effectively present to the decoder samples close to the source distribution. Experimental evaluation on simulated datasets and real mmWave wireless channels demonstrate that the proposed methods can quickly adapt the MDN model, and improve or maintain the error rate of the autoencoder under changing channel conditions.

1 Introduction

End-to-end learning of communication systems using autoencoders has been recently shown to be a viable method for designing the next generation of wireless networks [123456]. A point-to-point communication system consists of a transmitter (or encoder), a channel, and a receiver (or decoder). The key idea of end-to-end learning for a communication system is to use an autoencoder architecture to model and learn the transmitter and receiver jointly using neural networks in order to minimize an end-to-end performance metric such as the block error rate (BLER) [1]. The channel can be represented as a stochastic transfer function that transforms its input \( x \in \mathbb{R}^d \) to an output \( y \in \mathbb{R}^d \). It can be regarded as a black-box that is non-linear and non-differentiable due to hardware imperfections (e.g., quantization and amplifiers). Since autoencoders are trained using stochastic gradient descent (SGD)-based optimization, with the gradients calculated using backpropagation [1], it is challenging to work with a black-box channel that is not differentiable. One approach to address this problem is by using a known mathematical model of the channel (e.g., additive Gaussian noise). Use of such models enables the computation of gradients of the loss function with respect to the autoencoder parameters via backpropagation. However, such standard channel models do not capture well the realistic channel effects, as shown in [7]. Alternatively, recent works have proposed to learn the channel using (deep) generative models that approximate \( p(y \mid x) \), the conditional probability density of the channel output given

∗Correspondence to: jayaramr@cs.wisc.edu
†The first two authors have equal contribution.
the channel input, using generative adversarial networks (GANs) \cite{4,5}, mixture density networks (MDNs) \cite{8}, and conditional variational autoencoders (VAEs) \cite{9}. The use of a differentiable generative model of the channel enables SGD-based training of the autoencoder, while also capturing realistic channel effects better than standard models.

Although this end-to-end optimization with real channels learned from data can improve the physical layer design for communication systems, in reality, channels often change, requiring collection of a large number of samples and retraining the channel model and autoencoder frequently. For this reason, adapting the learned conditional probability density of the channel as often as possible using only a small number of samples is required for good communication performance. In this paper, we study the problem of domain adaptation (DA) of autoencoders using an MDN as the channel model. We make the following contributions: i) We propose a light-weight and sample-efficient method for adapting a generative MDN (used for modeling the wireless channel) based on the properties of Gaussian mixtures. ii) Based on the MDN adaptation, we propose two methods to compensate for changes in the class-conditional feature distributions at the decoder (receiver) that maintain or decrease the error rate of the autoencoder, without requiring any retraining.

In contrast to the conventional DA setting, where one has access to a large unlabeled dataset and none or a small labeled dataset from the target domain \cite{10,11}, here we consider DA when the target domain has only a small labeled dataset. This setting applies to the wireless communication channel where the target distribution changes frequently, and we only get to collect a small number of samples at a time from the target distribution. Recent approaches for DA (such as DANN \cite{12}) based on adversarial learning of a shared representation between the source and target domains \cite{13,12,14,15,16,17} have achieved much success on computer vision and natural language processing tasks. The high-level idea is to adversarially learn a shared feature representation for which inputs from the source and target domains are nearly indistinguishable to a domain discriminator DNN, such that a label predictor DNN using this representation and trained using labeled data from only the source domain also generalizes well to the target domain \cite{12}. Adversarial DA methods are not suitable for this problem because of the imbalance in the number of source and target domain unlabeled samples (not enough target domain samples to learn a good domain discriminator). Also, adversarial DA methods are not well-suited for fast and frequent adaptation that is required here. In summary, the problem setting addressed in this work has the following key differences from standard DA:

- The number of target domain samples is much smaller compared to the source domain. All of them are labeled, i.e., no unlabeled samples.
- We have a pre-trained classifier (decoder) from the source domain and do not want to retrain the classifier frequently for two reasons: i) retraining may not be fast enough to keep up with changes in the wireless channel, ii) collecting a reasonably-large number of labeled samples is hard while the channel distribution is changing.
- Unlike adversarial DA methods, we do not train a single classifier that has low error rate on both the domains. We adapt a generative model of the class-conditional feature distribution from the source to the target domain, and design feature transformations at the input of the source domain classifier.
- Our method focuses on low-dimensional input domains such as a communication system, but may not be well suited for high-dimensional input domains such as images.

2 Background

We introduce the notations and definitions used, followed by a brief background on i) end-to-end learning in wireless communication using autoencoders, and ii) MDN-based generative modeling. A detailed background and discussion of related works on these topics is given in Appendix [A].

Notations and Definitions. Vectors and matrices are denoted by boldface symbols. We use uppercase letters for random variables and lowercase for the specific values taken by them. \( \mathbb{R} \) and \( \mathbb{C} \) denote the set of real and complex numbers. We use the concise notation \([n] = \{1, \ldots, n\}\) for any integer \( n \geq 1 \). We define \( 1(c) \) to be an indicator function that takes the value \( 1 \) (0) when predicate \( c \) is true (false). We denote the one-hot-coded vector of all zeros except a 1 at position \( i \) in \( [n] \) by \( 1_{i,m} \in \{0,1\}^m \), and we omit \( m \) when it is clear from the context. The probability density function (pdf) of a multivariate Gaussian with mean vector \( \mu \) and covariance matrix \( \Sigma \) is denoted by \( \mathcal{N}(\cdot | \mu, \Sigma) \). The categorical probability mass function is denoted by \( \text{Cat}(p_1, \ldots, p_k) \). The \( d \times d \) identity matrix is denoted by \( I_d \). The determinant and trace of a matrix \( A \) are denoted by \( \det(A) \) and \( \text{tr}(A) \) respectively. The element-wise (Hadamard) product of two vectors or matrices is denoted by \( \odot \). We use a slightly different notation compared to the convention in machine learning. While a (feature vector, class label) pair is usually denoted by \((x, y)\), here we denote the same by \((y, s)\), where \( y \in \mathbb{R}^d \) is the channel output and \( s \in \mathcal{S} \) is the input message. Also, \( x = E_{\theta_0}(1_s) \) is the encoded representation of a message \( s \), and the channel input. Table [A] in the Appendix provides a quick reference for the notations used in the paper.
2.1 Autoencoder-Based End-to-End Learning

Consider a single-input, single-output (SISO) wireless communication system as shown in Fig. 1 where the transmitter encodes and transmits messages from the set $S = \{1, 2, \cdots, m\}$ to the receiver through $d \geq 2$ discrete uses of the wireless channel. The receiver attempts to accurately decode the transmitted message from the distorted and noisy channel output $y$. We discuss the end-to-end learning of such a system using the concept of autoencoders [1,2].

**Transmitter / Encoder Neural Network.** The transmitter or encoder part of the autoencoder is modeled as a multilayer, feed-forward neural network (NN) that takes as input the one-hot-coded representation $x$, of a message $s \in S$, and produces an encoded symbol vector $x = E_\theta(1_s) \in \mathbb{R}^d$. Here, $\theta$ is the parameter vector (weights and biases) of the encoder NN and $d$ is the encoding dimension. Due to hardware constraints present at the transmitter, a normalization layer is used as the final layer of the encoder network in order to constrain the average power and/or the amplitude of the symbol vectors. The average power constraint is defined as $E[\|x\|^2] = \mathbb{E}_{S}[\|E_\theta(1_s)\|^2] \leq c$, where the expectation is over the prior distribution of the input messages, and $c$ is typically set to 1. The amplitude constraint is defined as $|x_i| \leq 1$, $\forall i \in [d]$. The size of the message set is usually chosen to be a power of 2, i.e., $m = 2^b$ representing $b$ bits of information. Following [1], the communication rate of this system is the number of bits transmitted per channel use, which in this case is $R = b/d$. An autoencoder transmitting $b$ bits over $d$ uses of the channel is referred to as a $(d, b)$ autoencoder. For example, a $(2, 4)$ autoencoder uses a message set of size 16 and an encoding dimension of 2, with a communication rate $R = 2$ bits/channel use.

**Receiver / Decoder Neural Network.** The receiver or decoder component is also a multilayer, feedforward NN that takes as input the one-hot-coded representation $y$, that learns to accurately categorize the received (distorted) symbol vector into one of the $m$ message classes. This is in contrast to conventional autoencoders, where the decoder learns to accurately reconstruct a high-dimensional tensor from its low-dimensional representation learned by the encoder. The mean-squared and median-absolute error are commonly used end-to-end performance metrics for conventional autoencoders. In the case of communication autoencoders, the symbol or block error rate (BLER), defined as $\mathbb{E}_{S, X}[\mathbb{I}(\hat{S}(Y) \neq S)]$, is used as the end-to-end performance metric.

**Channel Model.** As discussed in §1 the wireless channel can be represented by a conditional probability density $P(y | x)$ of the channel output given its input. This channel can be equivalently characterized by a stochastic transfer function $y = h(x,z)$ that transforms the encoded symbol vector into the channel output, where $z$ captures the stochastic components of the channel (e.g., random noise, phase offsets). For example, an additive white Gaussian noise (AWGN) channel is represented by $y = h(x,z) = x + z$, with $z \sim N(0, \sigma^2 I_d)$ and $P(y | x) = N(y | x, \sigma^2 I_d)$. For realistic wireless channels, the transfer function and conditional probability density are usually unknown and hard to approximate well with standard mathematical models. Recently, a number of works have applied generative models such as conditional generative adversarial networks (GANs) [4,5], MDNs [8], and conditional variational autoencoders (VAEs) [9] for modeling the wireless channel. To model a wireless channel, generative methods learn a parametric model $P_b(y | x)$ (possibly a neural network) that closely approximates the true conditional density of the channel from a dataset of channel input, output observations. Learning a generative model of the channel comes with important advantages. 1) Once the parameters of the channel model are learned from data, the model can be used to generate any number of representative samples from the channel distribution. 2) A channel model with a differentiable transfer function makes it possible to backpropagate gradients of the autoencoder loss function through the channel and train the autoencoder using stochastic gradient descent (SGD)-based optimization. 3) It allows for continuous adaptation of the generative channel model to variations in the channel conditions, and thereby maintain a low BLER of the autoencoder.
2.2 Generative Channel Model using a Mixture Density Network

In this work, we use an MDN [18, 19] with Gaussian components to model the conditional density of the channel output given its input. MDNs can model complex conditional densities by combining a (feed-forward) neural network with a standard parametric mixture model (e.g., mixture of Gaussians). The MDN learns to predict the parameters of the mixture model \( \phi(x) \) as a function of the channel input \( x \). This can be expressed as \( \phi(x) = M_\theta(x) \), where \( \theta \) is the parameter vector (weights and biases) of the neural network. The parameters of the mixture model defined by the MDN are a concatenation of the parameters from the \( k \) density components, i.e., \( \phi(x) = [\phi_1(x)^T, \cdots, \phi_k(x)^T] \), where \( \phi_i(x) \) is the parameter vector of component \( i \). Focusing on a Gaussian mixture, the channel conditional density modeled by the MDN is given by

\[
P_\theta(y|x) = \sum_{i=1}^{k} P_{\theta_i}(K = i | x) P_{\theta_i}(y | x, K = i) = \sum_{i=1}^{k} \pi_i(x) N(y | \mu_i(x), \sigma_i^2(x)),
\]

where \( \mu_i(x) \in \mathbb{R}^d \) is the mean vector, \( \sigma_i^2(x) \in \mathbb{R}_+^d \) is the variance vector, and \( \pi_i(x) \in [0, 1] \) is the weight (prior probability) of component \( i \). Also, \( K \) is the latent random variable denoting the mixture component of origin. We have assumed that the Gaussian components have a diagonal covariance matrix, with \( \sigma_i^2(x) \) being the diagonal elements\(^3\). The weights of the mixture are parameterized using the softmax function as \( \pi_i(x) = \frac{e^{\alpha_i(x)}}{\sum_{j=1}^{k} e^{\alpha_j(x)}} \), \( \forall i \) in order to satisfy the probability constraint. The MDN simply predicts the un-normalized weights \( \alpha_i(x) \in \mathbb{R} \) (also known as the prior logits). For a Gaussian MDN, the parameter vector of component \( i \) is defined as \( \phi_i(x)^T = [\alpha_i(x), \mu_i(x)^T, \sigma_i^2(x)^T] \), and its output parameter vector \( \phi(x) \) has dimension \( k(2d + 1) \). Details on the conditional log-likelihood (CLL) training objective and the transfer function of the MDN, including a differentiable approximation of the transfer function, are discussed in Appendix B.

3 Fast Adaptation of the MDN Channel Model

In this section, we propose a fast and light-weight method for adapting a Gaussian MDN when the number of target domain samples is much smaller compared to that used for training it. Consider the setting where the channel state (and therefore its conditional distribution) is changing over time due to e.g., environmental factors. Let \( P(y | x) \) denote the (unknown) source channel distribution underlying the dataset \( D_s \) used to train the MDN \( M_\theta(x) \). With a sufficiently large dataset and a suitable choice of \( k \), the Gaussian mixture learned by the MDN \( P_\theta(y | x) \) can closely approximate \( P(y | x) \). Let \( D^{(t)} = \{(x_n^{(t)}, y_n^{(t)}) \), \( n = 1, \cdots, N^{(t)} \} \) denote a small set of iid samples (\( N^{(t)} \ll |D_s| \)) from an unknown target channel distribution \( P^{(t)}(y | x) \), which is potentially different from \( P(y | x) \) but not by a large deviation. Our goal is to adapt the MDN (and therefore the underlying mixture density) using \( D^{(t)} \) such that it closely approximates \( P^{(t)}(y | x) \). Note that the space of inputs to the MDN is the finite set of modulated symbols \( X = \{E_s(1_s), s \in S\} \) (referred to as a constellation), with each symbol corresponding to a unique message \( s \in S \)\(^4\).

**Key Insight.** The proposed method is based on the affine-transformation property of multivariate Gaussians, i.e., one can transform between any two multivariate Gaussians through an affine transformation. Given any two Gaussian mixtures with the same number of components and a one-to-one correspondence between the components, we can find the unique set of affine transformations (one per-component) that transforms one Gaussian mixture to the other. Moreover, the affine transformations are bijective, allowing the mapping to be applied in the inverse direction. This insight allows us to formulate the MDN adaptation as an optimization over the per-component affine transformation parameters, which is a much smaller problem compared to optimizing the weights of all the MDN layers (see Table 3 for a comparison). To reduce the possibility of the adapted MDN finding bad solutions due to the small-sample regime, we include a regularization term based on the Kullback-Leibler divergence (KLD) in the adaptation objective that constrains the distribution shift produced by the affine transformations. The use of a parametric Gaussian mixture, combined with the one-to-one association of the components allows us to derive a closed-form expression for the KLD between the source and target mixture distributions.

\(^3\)The diagonal covariance assumption does not imply conditional independence of \( y \) as long as \( k > 1 \).

\(^4\)The prior probability over the constellation \( \{p(x) \), \( x \in X\} \) is equal to the prior probability over the input messages. This is usually either set to be uniform, or estimated using relative frequencies from a large dataset.
3.1 Transformation Between Gaussian Mixtures

Consider the Gaussian mixtures corresponding to the source and target channel conditional densities
\[ P_{\hat{\theta}}(y \mid x) = \sum_{i=1}^{k} \pi_i(x) N(y \mid \mu_i(x), \sigma_i^2(x)) \quad \text{and} \]
\[ P_{\hat{\theta}}(y \mid x) = \sum_{i=1}^{k} \hat{\pi}_i(x) N(y \mid \hat{\mu}_i(x), \hat{\sigma}_i^2(x)), \]
where \( \hat{\theta} \) is the parameter vector of the adapted MDN. The adapted MDN predicts the parameters of the target Gaussian mixture as \( \hat{\phi}(x) = \hat{M}_{\hat{\theta}}(x) \), where \( \hat{\phi}(x)^T = [\hat{\phi}_1(x)^T, \ldots, \hat{\phi}_k(x)^T] \) is a concatenation of the parameters of the individual components. As before, the parameters of a component \( i \) are defined as \( \hat{\phi}_i(x)^T = [\hat{\alpha}_i(x), \hat{\mu}_i(x)^T, \hat{\sigma}_i^2(x)^T] \).

We focus on the case of diagonal covariances, but the results easily extend to the case of general covariances. We apply the above result to an MDN with a diagonal covariance, as shown in Appendix C, the transformation between any two multivariate Gaussians \( y \sim N(\cdot \mid \mu, \Sigma) \) and \( \tilde{y} \sim N(\cdot \mid \tilde{\mu}, \tilde{\Sigma}) \) can be achieved by the transformation: \( \tilde{y} = C (y - \mu) + A \mu + b \), where the mean vector and covariance matrix of the two Gaussians are related as follows:
\[ \tilde{\mu} = A \mu + b \quad \text{and} \quad \tilde{\Sigma} = C \Sigma C^T. \]

Affine and Inverse-Affine Feature Transformations

Applying the above result to an MDN with \( k \) components, we define the affine feature transformation for component \( i \) mapping from \( y \sim N(\cdot \mid \mu_i(x), \sigma_i^2(x)) \) to \( \tilde{y} \sim N(\cdot \mid \tilde{\mu}_i(x), \tilde{\sigma}_i^2(x)) \) as
\[ \tilde{y} = g_{xi}(y) = C_i (y - \mu_i(x)) + A_i \mu_i(x) + b_i, \ i \in [k]. \]
It is straightforward to also define the inverse-affine transformation from \( \tilde{y} \sim N(\cdot \mid \tilde{\mu}_i(x), \tilde{\sigma}_i^2(x)) \) to \( y \sim N(\cdot \mid \mu_i(x), \sigma_i^2(x)) \) as
\[ y = g_{xi}^{-1} (\tilde{y}) = C_i^{-1} (\tilde{y} - A_i \mu_i(x) - b_i) + \mu_i(x), \ i \in [k]. \]

Note that the feature transformations are conditioned on a given input to the MDN \( x \in \mathcal{X} \) and a given component of the mixture. This idea is illustrated in Fig 3. For the case of diagonal covariances, we constrain \( C_i \) and \( A_i \) to also be diagonal. These feature transformations will be used for defining a validation metric for the MDN adaptation, and also for aligning the target class-conditional distributions of \( y \) with that of the source in §4.

Parameter Transformations

The corresponding transformations between the source and target Gaussian mixture parameters for any component \( i \in [k] \) are given by
\[ \tilde{\mu}_i(x) = A_i \mu_i(x) + b_i, \ \tilde{\sigma}_i^2(x) = C_i^2 \sigma_i^2(x), \ \text{and} \ \tilde{\alpha}_i(x) = \beta_i \alpha_i(x) + \gamma_i, \]
where \( A_i = \text{diag}(a_{i1}, \ldots, a_{id}) \) is a diagonal scale matrix for the means; \( b_i = [b_{i1}, \ldots, b_{id}]^T \) is an offset vector for the means; \( C_i = \text{diag}(c_{i1}, \ldots, c_{id}) \) is a diagonal scale matrix for the variances; \( \beta_i \in \mathbb{R} \) and \( \gamma_i \in \mathbb{R} \) are the scale and offset for the component prior logits. The vector of all adaptation parameters to be optimized is defined as \( \psi^T = [\psi_{i1}^T, \ldots, \psi_{id}^T] \), where \( \psi_{id}^T = [a_{id}, \ldots, a_{id}, b_{id}, \ldots, b_{id}, c_{id}, \ldots, c_{id}, \beta_i, \gamma_i] \). The number of adaptation parameters (dimension of \( \psi \)) is given by \( k (3 d + 2) \). This is typically much smaller than the number of MDN parameters (weights and biases from all layers), even for shallow fully-connected NNs. The overall idea for adapting the MDN is summarized in Fig. 2, where the adaptation layer mapping \( \phi(x) \) to \( \hat{\phi}(x) \) basically implements the parameter transformations in Eq. 7.

3.2 Divergence Between the Source and Target Distributions

We would like to derive a distributional-divergence metric between the source and target Gaussian mixtures (Eqs. 2 and 3) in order to regularize the adaptation loss. A number of distributional-divergence metrics such as the Kullback-Leibler, Jensen-Shannon [20], and Total Variation [21] divergence are potential candidates, each possessing some unique properties. However, none of these divergences have a closed-form expression for a pair of general...
Gaussian mixtures. Prior works such as [22] have addressed the problem of estimating the KLD between a pair of Gaussian mixtures for the general case where the number of components could be different, with no association between the components. As mentioned earlier, in our problem there exists a one-to-one association between the individual components of the source and target Gaussian mixtures (by definition). This allows us to derive a closed-form expression for the KLD as follows:

\[
\mathcal{D}(P_{\theta_s}, P_{\theta_t}) = \mathbb{E}_{P_{\theta_s}} \left[ \log \frac{P_{\theta_s}(y, K | x)}{P_{\theta_t}(y, K | x)} \right] = \sum_{x \in X} p(x) \sum_{i=1}^{k} \pi_i(x) \log \frac{\pi_i(x)}{\pi_i(x)} + \sum_{x \in X} p(x) \sum_{i=1}^{k} \pi_i(x) D_{KL}(N(\mu_i(x), \sigma_i^2(x)), N(\hat{\mu}_i(x), \hat{\sigma}_i^2(x))) .
\]

The first term in the above expression is the KLD between the component prior probabilities, which can be simplified into a function of the adaptation parameters \([\beta_1, \gamma_1, \cdots, \beta_k, \gamma_k]\). The second term in the above expression involves the KLD between two multivariate Gaussians (a standard result), which can also be simplified into a function of the adaptation parameters. A detailed derivation of this result, and the final expression for the KLD as a function of \(\psi\) are given in Appendix D. To make the dependence on \(\psi\) explicit, the KLD is henceforth denoted by \(\mathcal{D}_{\psi}(P_{\theta_s}, P_{\theta_t})\).

### 3.3 Loss Function for Adaptation

We consider two scenarios for adaptation: 1. Generative adaptation of the MDN in isolation and 2. Discriminative adaptation of the MDN as the channel model for an autoencoder. In the first case, the goal of adaptation is to find a good generative model for the target data distribution, while in the second case, the goal is to improve the classification performance of the autoencoder on the target data distribution. Recall that we are given a small dataset \(\mathcal{D}^{(l)} = \{(x_n^{(l)}, y_n^{(l)})\}, n = 1, \cdots, N_l^{(l)}\) sampled from the target distribution \(P^{(l)}(y | x)\). We formulate the MDN adaptation as a minimization problem with a regularized negative log-likelihood objective, where the regularization term penalizes solutions with a large KL divergence between the source and target Gaussian mixtures.

**Generative Adaptation.** The adaptation objective is the following regularized negative conditional log-likelihood (CLL) of the target dataset:

\[
J_{\text{CLL}}(\psi; \lambda) = -\frac{1}{N_t} \sum_{n=1}^{N_t} \log P_{\psi_n}(y_n^{(t)} | x_n^{(t)}) + \lambda \mathcal{D}_{\psi}(P_{\theta_s}, P_{\theta_t}),
\]

where \(P_{\psi_n}(y_n^{(t)} | x_n^{(t)})\) is given by Eq. [3] and \(\hat{\mu}_i(x), \hat{\sigma}_i^2(x)\) and \(\hat{\gamma}_i(x)\) as a function of \(\psi\) are given by Eq. [7]. The parameters of the original mixture density \(\alpha_i(x), \mu_i(x), \sigma_i^2(x)\), \(\forall i\) are constant terms since they have no dependence on \(\psi\). The regularization constant \(\lambda \geq 0\) controls the KL divergence between the source and target Gaussian mixtures in the optimal solution. Small values of \(\lambda\) weight the CLL term more and allow more exploration in the adaptation; larger values of \(\lambda\) impose a stronger regularization to constrain the space of target distributions.

**Discriminative Adaptation.** With the goal of improving the accuracy of the decoder in recovering the transmitted symbol \(x\) from \(y\), the data-dependent term in the adaptation objective \(J_{\text{CLL}}\) is replaced with the posterior log-likelihood (PLL) as follows:

\[
J_{\text{PLL}}(\psi; \lambda) = -\frac{1}{N_t} \sum_{n=1}^{N_t} \log P_{\psi_n}(x_n^{(t)} | y_n^{(t)}) + \lambda \mathcal{D}_{\psi}(P_{\theta_s}, P_{\theta_t}).
\]

The posterior probability \(P_{\psi_n}(x | y)\) can be expressed in terms of the conditional Gaussian mixtures by applying Bayes’ rule. This is the only difference from the generative adaptation scenario.

We make the following observations about the minimization problem: i) The adaptation objective (in both cases) is a smooth and nonconvex function of \(\psi\); ii) Computing the objective and its gradient w.r.t. \(\psi\) is inexpensive operations since \(N_t^{(l)}\) and the dimension of \(\psi\) are relatively small. Also, this does not require forward and back-propagation steps through the layers of the MDN. For this reason, we use the BFGS quasi-newton method [23] for minimization, instead of SGD-based methods which are more suitable for large-scale learning problems.
3.4 Validation Metric and Selection of $\lambda$

The choice of $\lambda$ in the adaptation objective is crucial as it sets the amount of regularization most suitable for the target domain distribution. We propose a validation metric for selecting $\lambda$ based on the CLL of the inverse-affine-transformed target dataset with respect to the source mixture density. The reasoning is that, if the adaptation finds a solution (i.e., $\psi$) that is a good fit for the target dataset, then the inverse feature transformations based on that solution should produce a transformed target dataset that has a high CLL with respect to the source mixture density. The validation metric is the negative CLL of the inverse-transformed target dataset, given by

$$
\mathcal{L}_{val}(\psi; \mathcal{D}^{(t)}_c) = -\frac{1}{N^{(t)}_{c}} \sum_{n=1}^{N^{(t)}_{c}} \log P_{\hat{\theta}_e}(\mathbf{g}_n^{-1}(x_{n}^{(t)}), y_{n}^{(t)}) | x_{n}^{(t)}).
$$

(11)

Here $i_n^{(t)}$ is the best component assignment for the sample $(x_{n}^{(t)}, y_{n}^{(t)})$, given by

$$
i_n^{(t)} = \arg \max_{i \in [k]} P_{\hat{\theta}_e}(K = i | x_{n}^{(t)}, y_{n}^{(t)}).
$$

(12)

The above equation is simply the maximum-a-posteriori (MAP) rule applied to the component posterior of the target Gaussian mixture defined as

$$
P_{\hat{\theta}_e}(K = i | x, y) = \frac{\hat{\pi}_i(x) N(y | \hat{\mu}_i(x), \hat{\sigma}^2_i(x))}{\sum_{j=1}^{k} \hat{\pi}_j(x) N(y | \hat{\mu}_j(x), \hat{\sigma}^2_j(x))}, \forall i \in [k].
$$

Note that the validation metric (11) is based on the source Gaussian mixture (MDN with parameters $\theta_e$), but the MAP component assignment for each target domain sample Eq. (12) is based on the target Gaussian mixture (MDN with parameters $\hat{\theta}_e$). The adaptation objective is minimized with $\lambda$ varied over a range of values, and in each case the adapted solution $\psi$ is evaluated using the validation metric. The pair of $\lambda$ and $\psi$ resulting in the smallest validation metric is chosen as the final adapted solution.

4 Adaptation of Autoencoder-Based Communication System

In this section, we discuss how the proposed MDN adaptation can be combined with an autoencoder-based communication system to adapt the decoder to changes in the channel conditions. Recall that the decoder is basically a classifier that predicts the most-probable input message from the received channel output $y$. When the decoder operates in a new (target) channel environment, different from the one it was trained on, its classification accuracy can degrade due to the distribution change. Specifically, any change in the channel conditions reflects as changes in the class-conditional density of the decoder’s input, i.e., $\{P(y | s), s \in S\}$ changes. We propose to address this, by designing transformations to the decoder’s input that can compensate for changes in the channel distribution, and effectively present transformed inputs that are close to the source distribution on which the decoder was trained. Our method does not require any change or adaptation to the decoder network $D_{\theta_e}$ itself, but making it fast and suitable for the small-sample-size setting. We next discuss two such input transformation methods for the decoder.

4.1 Adapted Decoder Based on Affine Feature Transformations

Consider the same problem setup as § 3, where we observe a small dataset of samples from the target channel distribution. Suppose we have adapted the MDN channel by optimizing over the parameters $\psi$, we can use the inverse-affine feature transformations (defined in Eq. (6)) to transform the channel output $y$ from a component of the target Gaussian mixture to the same component of the source Gaussian mixture. However, this transformation requires knowledge of both the channel input $x$ and the mixture component $i$, which are not observed (latent) at the decoder. We propose to address this by first determining the most-probable pair of channel input and mixture component for a given $y$ (using the MAP rule), and applying the corresponding inverse-affine feature transformation as follows:

$$
\tilde{y} = \mathbf{g}_{x^*, i^*}^{-1}(y) \quad \text{where} \quad x^*, i^* = \arg \max_{x \in \mathcal{X}, i \in [k]} P_{\hat{\theta}_e}(x, i | y).
$$

(13)

For this generative model, it is easy to see that the class-conditional density is equal to the channel-conditional density, i.e., $P(y | s) = P(y | E_{\theta_e}(1_s)), \forall s$. Hence, by adapting the MDN, we are effectively also adapting the class-conditional density of the decoder’s input.
The joint posterior over the channel input $x$ and mixture component $i$, given the channel output $y$ is based on the adapted (target) Gaussian mixture, given by

$$P_{\theta_0}(x, i | y) = \frac{p(x) \pi_i(x) \mathcal{N}(y | \hat{\mu}_i(x), \hat{\sigma}_i^2(x))}{\sum_{x' \in X} \sum_j p(x') \pi_j(x') \mathcal{N}(y | \hat{\mu}_j(x'), \hat{\sigma}_j^2(x'))}.$$ 

The adapted decoder based on the above affine feature transformation is defined as

$$\hat{D}_{\theta_0}(y) := D_{\theta_0}(g_{x_1}^{-1}(y)) = D_{\theta_0} \circ g_{x_1}^{-1}(y),$$

and illustrated in Fig. 4. Note that the adapted decoder is a function of the parameters $\psi$, even though this is not made explicit in the notation.

We also explored a variant of this adapted decoder which uses a soft (probabilistic) assignment of the channel output $y$ to the channel input and mixture component pair $(x, i)$, given by

$$\hat{D}_{\theta_0}(y) = \sum_{x \in X} \sum_{i = 1}^k P_{\theta_0}(x, i | y) D_{\theta_0} \circ g_{x_1}^{-1}(y).$$

From our empirical evaluation, we found the hard MAP assignment based adaptation to have better performance. Hence, our experimental results are based on the adapted decoder [14].

### 4.2 Adapted Decoder Based on MAP Symbol Estimation

In the previous method, an input transformation layer is introduced at the decoder only during adaptation, and not during training of the autoencoder. Alternatively, here we propose an input transformation layer at the decoder that takes the channel output $y$ and produces a best estimate of the encoded symbol $\hat{x}$, which is then given as input to the decoder as shown in Fig. 5. This input transformation layer is included during the autoencoder training as a fixed non-linear transformation that does not have any trainable parameters. Since the decoder is trained to predict using $\hat{x}$ instead of $y$, it is inherently robust to changes in the channel distribution of $y$.

Given a generative model of the channel conditional density using Gaussian mixtures, we can estimate the plug-in Bayes posterior distribution of $x$ given $y$ as

$$P_{\theta_0}(x | y) = \frac{\exp(\log P_{\theta_0}(y | x) + \log p(x))}{\sum_{x' \in X} \exp(\log P_{\theta_0}(y | x') + \log p(x'))}.$$ 

From this, we define the MAP estimate of $x$ given $y$ as

$$\hat{X}_{\text{map}}(y) = \arg\max_{x \in X} P_{\theta_0}(x | y) = \arg\max_{x \in X} \log P_{\theta_0}(y | x) + \log p(x).$$

The adapted decoder based on this input transformation, referred to as the MAP symbol estimation (SE) layer, is defined as

$$\hat{D}_{\theta_0}(y) := D_{\theta_0}(\hat{X}_{\text{map}}(y)) = D_{\theta_0} \circ \hat{X}_{\text{map}}(y),$$

and illustrated in Fig. 5. Whenever the MDN model is adapted to changes in the channel distribution, resulting in a new MDN with parameters $\hat{\theta}_0$, the MAP SE layer is also updated using $\hat{\theta}_0$. This input transformation shields the decoder from changes to the distribution of $y$.

Since the MAP SE layer is also included in the autoencoder during training, the non-differentiable $\arg\max$ function presents an obstacle to training the autoencoder using backpropagation. We address this by using a temperature-scaled softmax approximation to the $\arg\max$, which is differentiable and provides a close approximation for small temperature values. This approximation is used only during training, whereas the exact $\arg\max$ is used during inference. Details on this approximation, and a modified autoencoder training algorithm with temperature annealing are discussed in Appendix E.
5 Experimental Evaluation

We implemented the mixture density network and communication autoencoder models using TensorFlow 2.3 [24] and TensorFlow Probability [25]. We used the BFGS optimizer implementation available in TensorFlow Probability. The code base for our work can be found at https://anonymous.4open.science/r/domain_adaptation-7COD/. All the experiments were run on a Macbook Pro laptop with 16 GB memory and 8 CPU cores. Table 1 summarizes the architecture of the encoder, MDN (channel model), and decoder neural networks. Note that the output layer of the MDN is a concatenation (denoted by ⊕) of three fully-connected layers predicting the means, variances, and mixing prior logit parameters of the Gaussian mixture. We used the following setting in all our experiments. The size of the message set \( m \) was fixed to 16, corresponding to 4 bits. The dimension of the encoding (output of the encoder) \( d \) was set to 2, and the number of mixture components \( k \) was set to 5. The size of the hidden layers \( n_h \) was set to 100.

The generative adaptation objective (9) is used for the experiments in §5.1 where the MDN is adapted in isolation (not as part of the autoencoder). The discriminative adaptation objective (10) is used for the experiments in §5.2 and §5.3 where the MDN is adapted as part of the autoencoder. For the proposed method, the scale and shift components of the adaptation parameters \( \psi \) are initialized to 1s and 0s respectively. This ensures that the target Gaussian mixture is always initialized with the source Gaussian mixture. The regularization constant \( \lambda \) in the adaptation objective was varied over 16 equally-spaced values on the log-scale (base 10) with range \( 10^{-5} \) to 100. The \( \lambda \) value and \( \psi \) corresponding to the smallest validation metric are selected as the final solution (§5.3). We note that minimizing the adaptation objective for different \( \lambda \) values can be efficiently done in parallel over multiple CPU cores.

5.1 MDN adaptation on Simulated Channels

We evaluate the proposed adaptation method for an MDN (§3) on simulated channel variations based on models commonly used for wireless communication. Specifically, we use the following channel models: i) additive white Gaussian noise (AWGN), ii) Ricean fading, and iii) Uniform or flat fading [26]. Details on these channel models and calculation of the their signal-to-noise ratio (SNR) are provided in Appendix F. In each case, the MDN is first trained on a large dataset simulated from a particular type of channel model (e.g., AWGN), referred to as the source channel. The trained MDN is then adapted using a small dataset from a different type of channel model (e.g., Ricean fading), referred to as the target channel. We used a standard constellation corresponding to quadrature amplitude modulation of 16 symbols, referred to as 16-QAM [26], as inputs to the channel. A training set of 25000 samples from the source channel is used to train the MDN. The size of the adaptation dataset from the target channel is varied over a few different values – 5, 10, 20 and 30 samples per symbol, corresponding to target datasets of size 80, 160, 320 and 480 respectively.

Baseline Methods. We evaluate the following two baseline methods for adapting the MDN. 1) A new MDN is initialized using the weights of the MDN trained on the source dataset, and trained using the target dataset. 2) Same as baseline 1, but only the weights of the final layer are optimized (fine-tuned) using the target dataset. The above methods are referred to as transfer and transfer-last-layer respectively. We used the Adam optimization method [27] for 200 epochs, with a batch size of 10 or 0.01 times the target dataset size, whichever is larger.

Evaluation Metric. Since the MDN is generative model, we evaluate the conditional log-likelihood of the learned Gaussian mixture on an unseen test set with 25000 samples from the target channel. We report the relative change in log-likelihood with respect to the original (unadapted) MDN, since the log-likelihood values may not be comparable across datasets. Suppose the log-likelihood of the original MDN is \( \ell_0 \) and that of an adaptation method is \( \ell \), then we calculate \( (\ell - \ell_0) / |\ell_0| \) as the metric. Larger values are better and negative values indicate that adaptation leads to a worse model.

Results and Inference. Table 2 summarizes the results for three (source, target) channel pairs. For each pair, the methods are run on 50 randomly generated training, adaptation, and test datasets. The training dataset is sampled from the source channel, while the adaptation and test datasets are sampled from the target channel. The SNR of the source
Table 2: Relative log-likelihood gain of the MDN adaptation methods on simulated channel variations

and target channels are independently and randomly selected from the range 10 dB to 20 dB for each trial. We observe
that the proposed method has a higher median relative log-likelihood gain for the low sample sizes (80 and 160) and
comparable median for higher sample sizes. Also, the baseline methods often have a 95% confidence interval (CI) that
is very skewed to the left, with a negative 2.5-th percentile. The proposed adaptation is more stable even for the the
smallest sample size, and never has a negative lower CI.

Table 3: Number of parameters being optimized by the MDN adaptation methods.

Table 3 compares the number of parameters being optimized by the proposed and baseline MDN adaptation methods for the architecture in Table 1. The method transfer optimizes all the layer weights of the MDN, which in this case has size 12925. The method transfer-last-layer optimizes only the weights of the final layer, which in this case has size 2525. The number of parameters optimized by the proposed method (i.e., dimension of $\psi$) would be 40, which is a much smaller problem compared to the baseline methods. This makes the proposed method well suited for the small-sample adaptation setting.

5.2 Autoencoder Adaptation on Simulated Channels

Figure 6: Results of affine transformation based adaptation on simulated channels.

We evaluate the proposed decoder adaptation methods on different pairs of simulated source and target channel
distributions. The setup for this experiment for adapting from a source channel A to a target channel B is as follows.
The autoencoder is initially trained using data from the source channel A at an SNR of 14 dB. Details of how the SNR is
related to the distribution parameters of the simulated channels is discussed in Appendix F. The MDN and the decoder
are adapted using a small dataset from the target channel B for different fixed SNRs varied over 8 dB to 20 dB in steps
of 2 dB. For each SNR, the adaptation is repeated over 10 randomly-sampled datasets from the target channel, and the
average block error rate (BLER) values are calculated on a large held-out test dataset (specific to each SNR). The size of training dataset (from channel A) and test dataset (from channel B) are both set to 20,000 samples per symbol, with 16 symbols. The size of the adaptation dataset from the target channel B is varied over 20, 40, and 60 samples per symbol.

The results of this experiment are summarized in Figs. 6 and 7 for three pairs of source and target channels. Figure 6 corresponds to the adaptation method of § 4.1 referred to as Affine, and Figure 7 corresponds to the adaptation method of § 4.2 referred to as MAP SE. The plots show the BLER vs. SNR curve, with average BLER on the y-axis (log-scaled) and SNR on the x-axis. This is commonly used to summarize the error performance of a communication system. The performance of a standard 16-QAM decoder (referred to as 16-QAM) and an autoencoder trained on the source channel without any adaptation (referred to as no_adapt) are included as baselines. For the proposed adaptation methods, the number of samples per symbol from the target channel are indicated as a suffix to the method name. For example, adapt_20 implies that the adaptation used 20 samples per symbol.

Observations and Takeaways.

1. Both the adaptation methods significantly decrease the BLER for the cases AWGN to Uniform fading and Ricean fading to Uniform fading.
2. For the case of AWGN to Ricean fading, the adaptation methods perform at the same level or slightly worse compared to the baselines. We think this is because the distribution of the two domains are not very different.
3. In general, the BLER decreases with increasing size of the target dataset.
4. Between the two adaptation methods, MAP SE performs marginally better than the Affine method.

5.3 Autoencoder Adaptation on Real FPGA Traces

We evaluate the performance of the adaptation methods on real over-the-air wireless experiments. We use a recent high-performance mmWave testbed [28], featuring a high-end FPGA board with 2 GHz of bandwidth per channel and 60 GHz SIVERS antennas [29]. This platform allows to transmit the custom constellations generated by the encoder and to store the data to be either trained by the MDN, or extracted for further performance analysis. We train the MDN with a standard 16-QAM constellation with 96000 samples. We evaluate the performance of our adaptation for 20, 35 and 50 samples per symbol. We introduced an IQ imbalance-based distortion to the constellation and gradually increase the level of imbalance to the system. The BLER of the proposed adaptation methods and the baseline methods (16-QAM and no adaptation) is shown as a function of the IQ imbalance in Fig. 8. The proposed methods (both Affine and MAP SE) show an order of magnitude decrease in BLER compared to the baseline methods when the IQ imbalance is over 25%.

6 Conclusions

In this paper we proposed a fast and light-weight method for adapting a Gaussian MDN to a target domain with a very limited number of adaptation samples. The method is based on finding the optimal set of component-conditional affine transformations that transform the source Gaussian mixture to the target Gaussian mixture. This is formulated as the minimization of a conditional (or posterior) log-likelihood, regularized by the KL-divergence between the two

\[ \text{KL}(\mu_2, \mu_1) \]

M-QAM is short for M-ary quadrature amplitude modulation with an M-symbol constellation. This is a standard technique for modulation and demodulation (decoding), which does not adapt based on the channel conditions.

IQ imbalance is a common issue in radio frequency communications that introduces distortions to the final constellation.
mixture distributions. We applied the MDN adaptation to an autoencoder-based end-to-end communication system, specifically by transforming the inputs to the decoder such that their class-conditional distributions are close to that of the source domain. This allows for fast adaptation of both the MDN channel and the autoencoder without the need for expensive data collection and retraining. We demonstrated the effectiveness of the proposed methods through extensive experiments on both simulated wireless channels and a real mmWave FPGA testbed.

**Limitations & Future Work**

The proposed adaptation for a Gaussian MDN is primarily targeted for low-dimensional problems such as the wireless channel. It can be challenging to apply on high-dimensional input domains with structure. Extensions of the proposed work to deep generative models based on normalizing flows [30, 31, 32] is an interesting direction, which would be more suitable for high-dimensional inputs. In this work, we do not adapt the encoder network, i.e., the autoencoder constellation is not adapted to changes in the channel distribution. Adapting the encoder, decoder, and channel networks jointly would allow for more flexibility, but would likely be slower and require more data from the target distribution.

**References**

[1] Timothy J. O’Shea and Jakob Hoydis. An introduction to deep learning for the physical layer. *IEEE Transactions on Cognitive Communications and Networking*, 3(4):563–575, 2017.

[2] Sebastian Dörner, Sebastian Cammerer, Jakob Hoydis, and Stephan ten Brink. Deep learning based communication over the air. *IEEE Journal of Selected Topics in Signal Processing*, 12(1):132–143, 2018.

[3] Fayçal Ait Aoudia and Jakob Hoydis. Model-free training of end-to-end communication systems. *IEEE Journal on Selected Areas in Communication*, 37(11):2503–2516, 2019.

[4] Timothy J. O’Shea, Tamoghna Roy, and Nathan West. Approximating the void: Learning stochastic channel models from observation with variational generative adversarial networks. In *International Conference on Computing, Networking and Communications (ICNC)*, pages 681–686. IEEE, 2019.

[5] Hao Ye, Geoffrey Ye Li, Biing-Hwang Fred Juang, and Kathiravetpillai Sivanesan. Channel agnostic end-to-end learning based communication systems with conditional GAN. In *IEEE Globecom Workshops (GC Wkshps)*, pages 1–5. IEEE, 2018.

[6] Tianqi Wang, Chao-Kai Wen, Hanqing Wang, Feifei Gao, Tao Jiang, and Shi Jin. Deep learning for wireless physical layer: Opportunities and challenges. *CoRR*, abs/1710.05312, 2017.

[7] Fayçal Ait Aoudia and Jakob Hoydis. End-to-end learning of communications systems without a channel model. In 52nd Asilomar Conference on Signals, Systems, and Computers (ACSSC), pages 298–303. IEEE, 2018.

[8] Dolores García Martí, Joan Palacios Beltrán, Jesús Omar Lacruz, and Joerg Widmer. A mixture density channel model for deep learning-based wireless physical layer design. In *Proceedings of the 23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 53–62, 2020.
[9] William Xia, Sundeep Rangan, Marco Mezzavilla, Angel Lozano, Giovanni Geraci, Vasilli Semkin, and Giuseppe Loianno. Millimeter wave channel modeling via generative neural networks. In IEEE Globecom Workshops (GC Wkshps), pages 1–6. IEEE, 2020.

[10] Jing Jiang. A literature survey on domain adaptation of statistical classifiers. Technical report, UIUC, 2008. http://www.mysmu.edu/faculty/jingjiang/papers/da_survey.pdf

[11] Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of representations for domain adaptation. In Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, pages 137–144. MIT Press, 2006.

[12] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor S. Lempitsky. Domain-adversarial training of neural networks. Journal of Machine Learning Research, 17:59:1–59:35, 2016.

[13] Yaroslav Ganin and Victor S. Lempitsky. Unsupervised domain adaptation by backpropagation. In Proceedings of the 32nd International Conference on Machine Learning (ICML), volume 37 of JMLR Workshop and Conference Proceedings, pages 1180–1189. JMLR.org, 2015.

[14] Mingsheng Long, Zhanjie Cao, Jiamin Wang, and Michael I. Jordan. Conditional adversarial domain adaptation. In Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems, pages 1647–1657, 2018.

[15] Kuniaki Saito, Kohei Watanabe, Yoshihata Ushiku, and Tatsuya Harada. Maximum classifier discrepancy for unsupervised domain adaptation. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 3723–3732. IEEE Computer Society, 2018.

[16] Han Zhao, Remi Tachet des Combes, Kun Zhang, and Geoffrey J. Gordon. On learning invariant representations for domain adaptation. In Proceedings of the 36th International Conference on Machine Learning (ICML), volume 97 of Proceedings of Machine Learning Research, pages 7523–7532. PMLR, 2019.

[17] Fredrik D. Johansson, David A. Sontag, and Rajesh Ranganath. Support and invertibility in domain-invariant representations. In The 22nd International Conference on Artificial Intelligence and Statistics (AISTATS), volume 89 of Proceedings of Machine Learning Research, pages 527–536. PMLR, 2019.

[18] Christopher M. Bishop. Mixture density networks. Technical report, Aston University, 1994. http://publications.aston.ac.uk/id/eprint/373/.

[19] Christopher M. Bishop. Pattern recognition and machine learning, 5th Edition, chapter 5, pages 272–277. Information Science and Statistics. Springer, 2007.

[20] Jianhua Lin. Divergence measures based on the Shannon entropy. IEEE Transactions on Information Theory, 37(1):145–151, 1991.

[21] Sergio Verdú. Total Variation distance and the distribution of relative information. In Information Theory and Applications Workshop (ITA), pages 1–3. IEEE, 2014.

[22] John R. Hershey and Peder A. Olsen. Approximating the Kullback-Leibler divergence between Gaussian mixture models. In Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), pages 317–320. IEEE, 2007.

[23] Jorge Nocedal and Stephen Wright. Numerical Optimization, chapter 5–7. Springer Science & Business Media, 2006.

[24] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[25] TensorFlow Probability. https://www.tensorflow.org/probability.

[26] Andrea Goldsmith. Wireless Communications. Cambridge University Press, 2005.

[27] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In 3rd International Conference on Learning Representations (ICLR), Conference Track Proceedings, 2015.

[28] Jesus O. Lacruz, Rafael Ruiz, and Joerg Widmer. A real-time experimentation platform for sub-6 ghz and millimeter-wave MIMO systems. In ACM MobiSys’21, 2021.
[29] SIVERSIMA. EVK06002 Development Kit, 2020. https://www.siversima.com/product/evk-06002-00/.

[30] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. In 5th International Conference on Learning Representations (ICLR), Conference Track Proceedings. OpenReview.net, 2017.

[31] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In Advances in Neural Information Processing Systems (NeurIPS), pages 10236–10245, 2018.

[32] Lilian Weng. Flow-based deep generative models. lilianweng.github.io/lil-log, 2018.

[33] John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman. Learning bounds for domain adaptation. In Advances in Neural Information Processing Systems 20. Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, pages 129–136. Curran Associates, Inc., 2007.

[34] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. Machine Learning, 79(1-2):151–175, 2010.

[35] Masashi Sugiyama, Matthias Krauledat, and Klaus-Robert Müller. Covariate shift adaptation by importance weighted cross validation. Journal of Machine Learning Research, 8:985–1005, 2007.

[36] Masashi Sugiyama and Motoaki Kawanabe. Machine Learning in Non-stationary Environments: Introduction to Covariate Shift Adaptation. MIT Press, 2012.

[37] Marco Saerens, Patrice Latinne, and Christine Decaestecker. Adjusting the outputs of a classifier to new a priori probabilities: A simple procedure. Neural Computation, 14(1):21–41, 2002.

[38] Matthinus Christoffel Du Plessis and Masashi Sugiyama. Semi-supervised learning of class balance under class-prior change by distribution matching. Neural Networks, 50:110–119, 2014.

[39] Hao Ye, Geoffrey Ye Li, and Biing-Hwang Juang. Power of deep learning for channel estimation and signal detection in OFDM systems. IEEE Wireless Communication Letters, 7(1):114–117, 2018.

[40] Ruirui Li, Jyun-Yu Jiang, Xian Wu, Hongda Mao, Chu-Cheng Hsieh, and Wei Wang. Bridging mixture density networks with meta-learning for automatic speaker identification. In ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 3522–3526. IEEE, 2020.

[41] Axel Brando Guillaumes. Mixture Density Networks For Distribution and Uncertainty Estimation. PhD thesis, Universitat Politècnica de Catalunya. Facultat d’Informàtica de Barcelona, 2017.

[42] Heiga Zen and Andrew Senior. Deep mixture density networks for acoustic modeling in statistical parametric speech synthesis. In 2014 IEEE international conference on acoustics, speech and signal processing (ICASSP), pages 3844–3848. IEEE, 2014.

[43] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-Softmax. In 5th International Conference on Learning Representations (ICLR), Conference Track Proceedings. OpenReview.net, 2017.

[44] Jean Paul Linnartz. Ricean fading. http://www.wirelesscommunication.nl/reference/chaptr03/ricepdf/rice.htm 2001.
Appendix

### A.1 Loss Function and Training of the Autoencoder

Expanding on the brief background provided in § 2.1, here we provide a formal discussion of the end-to-end training of the autoencoder. First, let us define the input-output mapping of the autoencoder as $f_{\theta}(1_s) = D_{\theta_{e}}(h_{\theta_{e}}(E_{\theta_{e}}(1_s), z)) = (D_{\theta_{e}} \circ h_{\theta_{e}} \circ E_{\theta_{e}})(1_s)$, where $\theta^{T} = [\theta_{e}^{T}, \theta_{c}^{T}, \theta_{d}^{T}]$ is the combined vector of parameters from the encoder, channel, and decoder. Given an input message $s \in S$, the autoencoder maps the one-hot-coded representation of $s$ into an output probability vector over the message set. Note that, while the encoder and decoder neural networks are deterministic, a forward pass through the autoencoder is stochastic due to the channel transfer function $h_{\theta_{e}}$. The learning objective of the autoencoder is to accurately recover the input message at the decoder with a high probability. The cross-entropy (CE) loss, which is commonly used for training classifiers, is also suitable for end-to-end training of the autoencoder. For an input $s$ with encoded representation $x = E_{\theta_{e}}(1_s)$, channel output $y = h_{\theta_{e}}(x, z)$, and decoded output $D_{\theta_{d}}(y) = [P_{\theta_{d}}(1 | y), \cdots, P_{\theta_{d}}(m | y)]^{T}$, the CE loss is given by

$$
\ell_{\text{CE}}(1_s, f_{\theta}(1_s)) = -1^{T} \log f_{\theta}(1_s) = -1^{T} \log D_{\theta_{d}}(h_{\theta_{e}}(E_{\theta_{e}}(1_s), z)) = -\log P_{\theta_{d}}(s \mid h_{\theta_{e}}(E_{\theta_{e}}(1_s), z)),
$$

which is always non-negative and takes the minimum value 0 when the correct message is decoded with probability 1. The autoencoder aims to minimize the following expected CE loss over the input message set and the channel output:

$$
E[\ell_{\text{CE}}(1_S, f_{\theta}(1_S))] = -\sum_{s=1}^{m} p(s) \int_{\mathbb{R}^{d}} P_{\theta_{d}}(y \mid E_{\theta_{e}}(1_s)) \log P_{\theta_{d}}(s \mid y) \, dy.
$$

Here $p(s), \forall s \in S$ is the prior probability of the input messages, which is usually assumed to be uniform in the absence of prior knowledge. In practice, the autoencoder minimizes an empirical estimate of the expected CE loss.
loss function by generating a large set of samples from the channel conditional density given each message. Let \( Y^{(s)} = \{ y_n^{(s)} = h_{\theta_e}(E_{\theta_e}(1_s), z_n), n = 1, \cdots, N \} \) denote a set of independent and identically distributed (iid) samples from \( P_{\theta_e}(y | E_{\theta_e}(1_s)) \), the channel conditional density given message \( s \). Also, let \( Y = \cup_{s \in S} Y^{(s)} \) denote the combined set of samples. The empirical expectation of the autocoder CE loss (19) is then given by

\[
L_{\text{auto}}(\theta ; Y) = - \sum_{s=1}^{m} p(s) \frac{1}{N} \sum_{n=1}^{N} \log P_{\theta_e}(s | h_{\theta_e}(E_{\theta_e}(1_s), z_n)).
\] (20)

It is clear from the above equation that the channel transfer function \( h_{\theta_e} \) should be differentiable in order to be able to backpropagate gradients through the channel to the encoder network. The transfer function defining sample generation for a Gaussian MDN channel is discussed in Appendix B.

The training algorithm for jointly learning the autoencoder and channel model (based on [8]) is given in Algorithm 1. It is an alternating (cyclic) optimization of the channel parameters and the autoencoder (encoder and decoder) parameters. The reason this type of alternating optimization is required is because the empirical expectation of the CE loss Eq. (20) is valid only when the channel conditional density (i.e., \( \theta_c \)) is fixed. The training algorithm can be summarized as follows. First, the channel model is trained for \( N_{cc} \) epochs using data sampled from the channel with an initial encoder constellation (e.g., M-QAM). With the channel model parameters fixed, the parameters of the encoder and decoder networks are optimized for one epoch of mini-batch SGD updates (using any adaptive learning rate algorithm e.g., Adam [27]). Since the channel model is no longer optimal for the updated encoder constellation, it is retrained for \( N_{ce} \) epochs using data sampled from the channel with the updated constellation. This alternate training of the encoder/decoder and the channel networks is repeated for \( N_{ae} \) epochs or until convergence.

### Algorithm 1 End-to-end training of the autoencoder with a generative channel model

1. **Inputs**: Message size \( m \); Encoding dimension \( d \); Initial constellation \( \{ E_0(1_s), \forall s \in S \} \); Number of optimization epochs for the autoencoder \( N_{ae} \) and channel \( N_{ce} \).
2. **Output**: Trained network parameters \( \theta_e, \theta_c, \theta_d \).
3. Initialize the encoder, channel, and decoder network parameters.
4. Sample training data \( D_{ch}^{(0)} \) from the channel using the initial constellation.
5. Train the channel model for \( N_{ce} \) epochs to minimize \( L_{ch}(\theta_c ; D_{ch}^{(0)}) \).
6. **for** epoch \( t = 1, \cdots, N_{ae} \);  
   7. Freeze the channel model parameters \( \theta_c \).  
   8. Perform a round of mini-batch SGD updates of \( \theta_e \) and \( \theta_d \) with respect to \( L_{\text{auto}}(\theta ; Y) \).
   9. Sample training data \( D_{ch}^{(t)} \) from the channel with the updated constellation \( \{ E_{\theta_e}(1_s), \forall s \in S \} \).
10. Train the channel model for \( N_{ce} \) epochs to minimize \( L_{ch}(\theta_c ; D_{ch}^{(t)}) \).
11. **Return** \( \theta_e, \theta_c, \theta_d \).

Finally, we observe some interesting nuances of the communication autoencoder learning task that is not common to other domains such as images. 1) The size of the input space is finite, equal to the number of distinct messages \( m \). Because of the stochastic nature of the channel transfer function, the same input message results in a different \( y \), even when \( x \) is held constant. 2) There is theoretically no limit on the number of samples that can be generated for training and validating the autoencoder. These two factors make the autoencoder learning less susceptible to overfitting, that is a common pitfall with neural network training.

### A.2 A Primer on Domain Adaptation

We provide a brief review of domain adaptation (DA) and discuss the key differences of our problem setting from that of standard DA. In the traditional learning setting, training and test data are assumed to be sampled independently from the same distribution \( P(x, y) \), where \( x \) and \( y \) are the input vector and target respectively. In many real world settings, it can be hard or impractical to collect a large labeled dataset \( D_t^y \) for a target domain where the machine learning model (e.g., a DNN classifier) is to be deployed. On the other hand, it is common to have access to a large unlabeled dataset \( D_s^u \) from the target domain, and a large labeled dataset \( D_s^y \) from a different but related source domain. Both \( D_s^u \) and \( D_s^y \) are much larger than \( D_t^y \), and in most cases there is no labeled data from the target domain (referred to

---

9 The notion used in this section is different from the rest of the paper, but consistent with the statistical learning literature.

10 One could have multiple source domains in practice; we consider the single source domain setting.
as unsupervised DA). For the target domain, the unlabeled dataset (and labeled dataset if any) are sampled from an unknown target distribution, i.e., \( x \in \mathcal{P}_t \sim P_t(x) \) and \( (x, y) \in \mathcal{D}_t \sim P_t(x, y) \). For the source domain, the labeled dataset is sampled from an unknown source distribution, i.e., \( (x, y) \in \mathcal{D}_s \sim P_s(x, y) \). The goal of DA is to leverage the available labeled and unlabeled datasets from the two domains to learn a predictor, denoted by the parametric function \( \hat{y} = \hat{f}_\theta(x) \), such that the following risk function w.r.t. the target distribution is minimized:

\[
R_t[\hat{f}_\theta] = \mathbb{E}_{(x, y) \sim P_t}[\ell(\hat{f}_\theta(x), y)] = \sum_{y} \int_{x} P_t(x, y) \ell(\hat{f}_\theta(x), y) \, dx,
\]

where \( \ell(\hat{y}, y) \) is a loss function that penalizes the prediction \( \hat{y} \) for deviating from the true value \( y \) (e.g., cross-entropy or hinge loss). In a similar way, we can define the risk function w.r.t. the source distribution \( R_s[\hat{f}_\theta] \). A number of seminal works in DA theory \cite{11, 33, 34} have studied this learning setting and provide bounds on \( R_t[\hat{f}_\theta] \) in terms of \( R_s[\hat{f}_\theta] \) and the divergence between source and target domain distributions. Motivated by this foundational theory, a number of recent works \cite{13, 12, 14, 15, 16, 17} have proposed using DNNs for adversarially learning a shared representation across the source and target domains such that a predictor using this representation and trained using labeled data from only the source domain also generalizes well to the target domain. An influential work in this line of DA is the domain adversarial neural network (DANN) proposed by \cite{13} and later by \cite{12}. The key idea behind the DANN approach is to adversarially train a label predictor NN and a domain discriminator NN in order to learn a feature representation for which i) the source and target inputs are nearly indistinguishable to the domain discriminator, and ii) the label predictor has good generalization performance on the source domain inputs.

**Special Cases of DA.** While the general DA problem addresses the scenario where \( P_s(x, y) \) and \( P_t(x, y) \) are different, certain special cases of DA have also been explored. One such special case is **covariate shift** \cite{15, 30}, where only the marginal distribution of the inputs changes (i.e., \( P_t(x) \neq P_s(x) \)), but the conditional distribution of the target given the input does not change (i.e., \( P_t(y | x) \approx P_s(y | x) \)). Another special case is the so-called **label shift** or class-prior mismatch \cite{37, 38}, where only the marginal distribution of the label changes (i.e., \( P_t(y) \neq P_s(y) \)), but the conditional distribution of the input given the target does not change (i.e., \( P_t(x | y) \approx P_s(x | y) \)). Prior works have proposed targeted theory and methods for these special cases of DA.

**A.3 Related Works**

**ML-based Communication Systems.** The study of end-to-end learning for communications was first introduced in \cite{1}, where the authors use the autoencoder model to learn an encoding and decoding of a point-to-point system with a channel model. The autoencoder training approaches can be broadly divided in two categories: i) **model-aware**, in which the channel model is analytically defined, e.g. \cite{1, 2}, and ii) **model-free**, in which the channel model is learned from the data, e.g. \cite{3}. The problem with model-aware approaches is that, when tested on real hardware implementations, the use of inaccurate channel models results in performance loss compared to standard schemes \cite{2}. The first works to learn the channel from data are \cite{4} and \cite{39}. These works use a GAN to generate samples from the stochastic channel distribution. However, the results seem to only work for simple channels, and no guaranties are presented about the statistical validity of the result. Aoudia et al. \cite{3} present a different model-free approach for autoencoder learning where the channel does not require a model and is allowed to be non-differentiable. Recently, \cite{8} proposed to use a mixture density network to learn the channel since Gaussian mixtures are known to be universal approximators that can accurately capture the channel distribution, given sufficient parametric complexity. Garcia et al. \cite{8} also propose to adapt the MDN model to changing channel conditions by training the MDN with a small set of samples from a new distribution. This is the same approach taken by the baseline method transfer in our experiments.

**Mixture Density Networks.** MDNs were first introduce by \cite{18}, providing a new framework for modeling complex conditional densities using neural networks. Bishop \cite{18} applied MDNs to the problem of robot inverse kinematics. Other applications of MDNs include speaker identification, redshift estimation, uncertainty estimation, handwriting and speech synthesis \cite{40, 41, 42}. However, to the best of our knowledge, the problem of adapting an MDN has only been studied by \cite{40}. Li et al. \cite{40} address the problem of new speaker identification using MDNs in the setting of small number of labeled samples from new speakers. They propose a gradient-based meta-learning algorithm for MDNs that learns to transfer knowledge from the existing set of speakers to new speakers in the small-sample setting. Although the setting of small number of target domain samples is similar to our problem, we do not deal with the problem of transferring knowledge from multiple sources (speakers in this case) to target domain.

**B MDN Training and Sample Generation**

In this section, we supplement the brief background on MDNs provided in \cite{2, 2} with details about the MDN training, sampling function, and how we make the sampling function differentiable to enable backpropagation-based training of
the autoencoder. Given a dataset of input-output pairs sampled from the channel \( \mathcal{D}_c = \{ (x_n, y_n), n = 1, \ldots, N_c \} \), the MDN is trained to minimize the negative conditional log-likelihood (CLL) of the data given by

\[
L_{ch}(\theta_c ; \mathcal{D}_c) = - \frac{1}{N_c} \sum_{n=1}^{N_c} \log P_{\theta_c}(y_n | x_n).
\]

With a large \( N_c \), the MDN can learn a sufficiently-complex parametric density model of the channel. The negative CLL objective can be interpreted as the sample estimate of the Kullback-Leibler divergence between the true (unknown) conditional density \( P(y | x) \) and the conditional density modeled by the MDN \( P_{\theta_c}(y | x) \). Therefore, minimizing the negative CLL finds the MDN parameters \( \theta_c \) that lead to a close approximation of the true conditional density. Standard SGD-based optimization methods such as Adam [27] can be applied to find the MDN parameters \( \theta_c \) that (locally) minimize the negative CLL.

After the MDN is trained, new simulated samples from the channel distribution can be generated from the Gaussian mixture using the following stochastic sampling method:

1. Randomly select a channel input \( x \) from the categorical prior distribution \( \{ p(x), x \in \mathcal{X} \} \).
2. Randomly select a component \( K = i \) according to the mixture weights \( \{ \pi_1(x), \ldots, \pi_k(x) \} \).
3. Randomly sample \( z \) from the standard \( d \)-dimensional Gaussian density \( z \sim N(\cdot | 0, I_d) \).
4. Generate the channel output as \( y = \sigma_i^2(x) \odot z + \mu_i(x) \).

Recall that \( \odot \) refers to the element-wise product of two vectors. The channel transfer or sampling function for a Gaussian MDN can thus be expressed as

\[
y = h_{\theta_c}(x, z) = \sum_{i=1}^{k} \mathbb{I}(K = i) (\sigma_i^2(x) \odot z + \mu_i(x)),
\]

where \( K \sim \text{Cat}(\pi_1(x), \ldots, \pi_k(x)) \) and \( z \sim N(\cdot | 0, I_d) \). Note that this transfer function is not differentiable w.r.t parameters \( \pi_i(x) \) and the MDN weights predicting it, because of the indicator function. As such, it is not directly suitable for SGD (backpropagation) based end-to-end training of the autoencoder. We next propose a differentiable approximation of the MDN transfer function based on the Gumbel softmax reparametrization [43], which is used in our autoencoder implementation.

### B.1 Differentiable MDN Transfer Function

Consider the transfer function of the MDN in Eq. (22). We would like to replace sampling from the categorical mixture prior \( \text{Cat}(\pi_1(x), \ldots, \pi_k(x)) \) with a differentiable function that closely approximates it. We apply the Gumbel-Softmax reparametrization [43] which solves this exact problem. First, recall that the component prior probabilities can be expressed in terms of the prior logits as:

\[
\pi_i(x) = \frac{e^{o_i(x)}}{\sum_{j=1}^{k} e^{o_j(x)}}, \quad \forall i \in [k].
\]

Consider \( k \) iid standard Gumbel random variables \( G_1, \ldots, G_k \overset{\text{iid}}{\sim} \text{Gumbel}(0, 1) \). It can be shown that, for any \( x \in \mathcal{X} \), the random variable

\[
S(x) = \arg\max_{i \in [k]} G_i + o_i(x)
\]

follows the categorical distribution \( \text{Cat}(\pi_1(x), \ldots, \pi_k(x)) \). This standard result is known as the Gumbel-max transformation. While Eq. (23) can be directly used inside the indicator function in Eq. (22), the argmax will still result in the transfer function being non-differentiable. Therefore, we use the following temperature-scaled softmax function as a smooth approximation of the argmax

\[
\tilde{S}_i(x; \tau) = \exp[(G_i + o_i(x)) / \tau] / \sum_{j=1}^{k} \exp[(G_j + o_j(x)) / \tau], \quad \forall i \in [k],
\]

where \( \tau > 0 \) is a temperature constant. For small values of \( \tau \), the temperature-scaled softmax will closely approximate the argmax, and the vector \( [\tilde{S}_1(x; \tau), \ldots, \tilde{S}_k(x; \tau)] \) will closely approximate the one-hot vector \( [\mathbb{I}(S(x) = 1), \ldots, \mathbb{I}(S(x) = k)] \).

Applying this Gumbel softmax reparametrization in Eq. (22), we define a modified differentiable transfer function for the Gaussian MDN as

\[
y = \tilde{h}_{\theta_c}(x, z) = \sum_{i=1}^{k} \tilde{S}_i(x; \tau) (\sigma_i^2(x) \odot z + \mu_i(x)).
\]
With this transfer function, it is straightforward to compute gradients with respect to the prior logits \( \alpha_i(x), \forall i \). Another neat outcome of this approach is that the stochastic components (Gumbel random variables \( G_i \)) are fully decoupled from the deterministic parameters \( \alpha_i(x) \) in the gradient calculations with respect to \( \hat{S}_i(x; \tau) \). In our experiments, we used this Gumbel-softmax based smooth transfer function while training the autoencoder, but during prediction (inference), we use the exact argmax based transfer function. We found \( \tau = 0.01 \) to be a good choice for all the experiments.

## C Transformation Between Multivariate Gaussians

We provide additional details on the feature and parameter transformations between two general multivariate Gaussians. This result was applied in §3 to formulate the MDN adaptation. Consider first the standard transformation from \( y \sim N(\cdot | \mu, \Sigma) \) to \( \hat{y} \sim N(\cdot | \hat{\mu}, \hat{\Sigma}) \) given by:

- Apply a whitening transformation \( z = D^{-1/2} U^T (y - \mu) \) such that \( z \sim N(\cdot | 0, I) \).
- Transform \( z \) into the new Gaussian density using \( \hat{y} = \hat{U} \hat{D}^{1/2} z + \hat{\mu} \).

We have denoted the eigen-decomposition of the covariance matrices by \( \Sigma = UDU^T \) and \( \hat{\Sigma} = \hat{U}\hat{D}\hat{U}^T \), where \( U \) and \( \hat{U} \) are the orthogonal eigenvector matrices, and \( D \) and \( \hat{D} \) are the diagonal eigenvalue matrices. Combining the two steps, the overall transformation from \( y \) to \( \hat{y} \) is given by

\[
\hat{y} = \hat{U} \hat{D}^{1/2} D^{-1/2} U^T (y - \mu) + \hat{\mu}.
\]

Suppose we define the matrix \( C = \hat{U} \hat{D}^{1/2} D^{-1/2} U^T \), then it is easily verified that the covariance matrices are related by \( \hat{\Sigma} = C \Sigma C^T \). In general, the mean vector and covariance matrix of any two Gaussians can be related by the following parameter transformations:

\[
\hat{\mu} = A \mu + b \quad \text{and} \quad \hat{\Sigma} = C \Sigma C^T,
\]

with parameters \( A \in \mathbb{R}^{d \times d}, b \in \mathbb{R}^d, \) and \( C \in \mathbb{R}^{d \times d} \). Substituting the above parameter transformations into the feature transformation, we get

\[
\hat{y} = C (y - \mu) + A \mu + b.
\]

## D Divergence Between the Source and Target Gaussian Mixtures

Referring to §3.2, we provide a detailed derivation of the KLD between the source and target Gaussian mixtures under the assumption of one-to-one association between the components.

\[
D(P_{\theta_s}, P_{\theta_c}) = \mathbb{E}_{P_{\theta_s}} \left[ \log \frac{P_{\theta_s}(y, K | x)}{P_{\theta_c}(y, K | x)} \right]
\]

\[
= \sum_{x \in \mathcal{X}} p(x) \sum_{i=1}^k \int_{\mathbb{R}^d} P_{\theta_s}(y, K = i | x) \log \frac{P_{\theta_s}(y, K = i | x)}{P_{\theta_c}(y, K = i | x)} dy
\]

\[
= \sum_{x \in \mathcal{X}} p(x) \sum_{i=1}^k \int_{\mathbb{R}^d} P_{\theta_s}(K = i | x) P_{\theta_s}(y | x, K = i) \log \frac{P_{\theta_s}(K = i | x) P_{\theta_s}(y | x, K = i)}{P_{\theta_c}(K = i | x) P_{\theta_c}(y | x, K = i)} dy
\]

\[
= \sum_{x \in \mathcal{X}} p(x) \sum_{i=1}^k \pi_i(x) \int_{\mathbb{R}^d} N(y | \mu_i(x), \sigma_i^2(x)) \left( \log \frac{\pi_i(x)}{\pi_i(x)} + \log \frac{N(y | \mu_i(x), \sigma_i^2(x))}{N(y | \hat{\mu}_i(x), \hat{\sigma}_i^2(x))} \right) dy
\]

\[
= \sum_{x \in \mathcal{X}} p(x) \sum_{i=1}^k \pi_i(x) \frac{\pi_i(x)}{\pi_i(x)} + \sum_{x \in \mathcal{X}} p(x) \sum_{i=1}^k \pi_i(x) D_{KL}(N(\cdot | \mu_i(x), \sigma_i^2(x)), N(\cdot | \hat{\mu}_i(x), \hat{\sigma}_i^2(x))).
\]

(26)
As discussed in §4.2, the presence of the non-differentiable argmax where 
\[
\argmax_{z} \quad \text{given } \theta.
\]
Applying this result to the KLD term in Eq. (26), which has diagonal covariances, we get
\[
D_{KL}\left(N(\cdot | \mu, \Sigma), N(\cdot | \hat{\mu}, \hat{\Sigma})\right) = \frac{1}{2} \log \frac{\det(\Sigma)}{\det(\hat{\Sigma})} + \frac{1}{2} \text{tr}(\hat{\Sigma}^{-1} \Sigma) + \frac{1}{2} (\hat{\mu} - \mu)^T \hat{\Sigma}^{-1} (\hat{\mu} - \mu) - \frac{d}{2}.
\]
Applying this result to the KLD term in Eq. (26), which has diagonal covariances, we get
\[
D_{KL}\left(N(\cdot | \mu_i(x), \sigma_i^2(x)), N(\cdot | \hat{\mu}_i(x), \hat{\sigma}_i^2(x))\right) = \frac{1}{2} \sum_{j=1}^{d} \left[ \log e_{ij}^2 + \frac{1}{c_{ij}} + \frac{1}{c_{ij}^2} \sigma_{ij}^2(x) (a_{ij} \mu_{ij}(x) + b_{ij} - \mu_{ij}(x))^2 \right] - \frac{d}{2},
\]
where \(a_{ij}, b_{ij}\) are the normalization terms in the softmax function.
Substituting Eqs. (27) and (28) into the last step of Eq. (26) gives the KLD between the source and target distributions as a function of the adaptation parameters \(\psi\).

E. MAP Symbol Estimation Autoencoder

In this section, we discuss some details of the adapted decoder with MAP symbol estimation that were not addressed in §4.2. The MAP SE method for adapting the decoder introduced the following transformation layer prior to the decoder in order to estimate the channel input \(x\) from the channel output \(y\):
\[
\hat{X}_{\text{map}}(y) = \arg\max_{x \in \mathcal{X}} P_{\theta}(x | y) = \arg\max_{x \in \mathcal{X}} \log P_{\theta}(y | x) + \log p(x).
\]
As discussed in §4.2, the presence of the non-differentiable argmax poses a problem for backpropagation-based training of the autoencoder. We propose to address this using a temperature-scaled softmax approximation to the argmax, similar to the method discussed in Appendix B.1.

Consider the posterior distribution of the channel input given the channel output
\[
P_{\theta}(x | y) = \frac{p(x) P_{\theta}(y | x)}{\sum_{x' \in \mathcal{X}} p(x') P_{\theta}(y | x')} = \frac{\exp(q_{\theta}(x, y))}{\sum_{x' \in \mathcal{X}} \exp(q_{\theta}(x', y))},
\]
where \(q_{\theta}(x, y) = \log P_{\theta}(y | x) + \log p(x)\) is defined for convenience. Let us introduce a temperature constant \(\tau > 0\) in the softmax function, and define the temperature-scaled posterior distribution
\[
P_{\theta}^{(\tau)}(x | y) = \frac{\exp(q_{\theta}(x, y) / \tau)}{\sum_{x' \in \mathcal{X}} \exp(q_{\theta}(x', y) / \tau)}.
\]
For large \(\tau\), the above posterior approaches a uniform distribution. For small \(\tau\) it approaches a distribution with probability 1 for \(x\) corresponding to the maximum exponent, and 0s elsewhere. Based on this observation, we define the following smooth approximation of the MAP SE layer
\[
\hat{X}_{\text{soft}}(y) = \sum_{x \in \mathcal{X}} P_{\theta}^{(\tau)}(x | y) x
\]
This can be interpreted as the conditional expectation of \(x\) given \(y\) with respect to the temperature-scaled posterior distribution \(P_{\theta}^{(\tau)}\). We can show that this smooth MAP estimate approaches the true MAP estimate in the limit as \(\tau\) approaches 0, i.e.,
\[
\lim_{\tau \to 0} \sum_{x \in \mathcal{X}} P_{\theta}^{(\tau)}(x | y) x = \arg\max_{x \in \mathcal{X}} P_{\theta}(x | y).
\]
Training Based on Temperature Annealing.

The MAP symbol estimation autoencoder uses the smooth MAP estimate (30) during training, and the exact MAP estimate during inference. In order to have good convergence and to prevent the training from getting stuck at poor solutions, we do not fix the temperature $\tau$ to a small value throughout. Instead, we decrease $\tau$ according to a temperature annealing schedule during training. Specifically, $\tau$ is initialized to a reasonably large value (e.g., $\tau_0 = 1$), and it is decreased by an exponential factor $\eta \in (0, 1)$ at the end of every $r \geq 1$ epochs. The solution (autoencoder parameters) at the end of $r$ epochs for the current temperature is used to initialize the training at the next lower temperature. This process is continued until a small final temperature $\tau_f$ is reached. In our experiments, we set the constants related to temperature annealing as follows: $\tau_0 = 1$, $\tau_f = 0.05$, $r = 10$, $\eta = 0.7169$. This choice of $\eta$ ensures that there are 10 temperature steps including the initial and final values.

F Simulated Channel Variation Models

We provide details of the mathematical models used to create simulated channel variations in our experiments. These models are frequently used in the study of wireless channels [26].

F.1 Uniform Fading Model

The channel output $y \in \mathbb{R}^d$ for this model as a function of the channel input (modulated symbol vector) $x \in \mathbb{R}^d$ is given by

$$ y = A x + n, $$

where $A \sim \text{Unif}(0, a)$ is a uniformly-distributed scale factor, and $n \sim \mathcal{N}(\cdot | 0, \sigma_n^2 I_d)$ is an additive Gaussian noise vector. Both $A$ and $n$ are assumed to be independent of each other and $x$. The average power in the signal component of $y$ is given by

$$ \bar{p}_{\text{avg}} := E[||A x||^2] = E[A^2] E[||x||^2] = \frac{a^2}{3} p_{\text{avg}}, $$

where $p_{\text{avg}}$ denotes the average power in the channel input $x$. The noise power in this case is given by $E[||n||^2] = \sigma_n^2$. The signal-to-noise ratio (SNR) for this model is therefore given by

$$ \frac{E_b}{N_0} = \frac{E[||A x||^2]}{2 R E[||n||^2]} = \frac{a^2 p_{\text{avg}}}{6 R \sigma_n^2}, $$

where $R$ is the communication rate of the system in bits/channel use. We select the fading factor $a$ such that the channel output has a target SNR value using the following equation:

$$ a = \sqrt{\frac{6 R \sigma_n^2 (E_b/N_0)}{p_{\text{avg}}}}. \quad (31) $$

F.2 Ricean and Rayleigh Fading Models

The channel output for the Ricean fading model is given by

$$ y = A x + n, $$

where $A$ is a diagonal matrix with the diagonal elements $a_1, \ldots, a_d \overset{\text{iid}}{\sim} \text{Rice}(\cdot | \nu, \sigma_n^2)$ following a Rice distribution, and $n \sim \mathcal{N}(\cdot | 0, \sigma_n^2 I_d)$ is an additive Gaussian noise vector. It is assumed that $n$ and $A$ are independent of each other and of $x$. Note that Rayleigh fading is a special case of Ricean fading when the parameter $\nu = 0$. For this model, the average power in the signal component of $y$ is given by

$$ \bar{p}_{\text{avg}} := E[||A x||^2] = \sum_{i=1}^d E[a_i^2 x_i^2] = \sum_{i=1}^d E[a_i^2] E[x_i^2] = (2 \sigma_n^2 + \nu^2) E[||x||^2] = (2 \sigma_n^2 + \nu^2) p_{\text{avg}}, $$

where $p_{\text{avg}}$ denotes the average power in the channel input $x$. We used the fact that the second moment of the Rice distribution is given by $E[a_i^2] = 2 \sigma_n^2 + \nu^2$. It is useful to consider the derived parameters $K = \nu^2 / 2 \sigma_n^2$ which
corresponds to the ratio of power along the line-of-sight (LoS) path to the power along the remaining paths, and \(\Omega = 2\sigma_a^2 + \nu^2\) which corresponds to the total power received along all the paths. The SNR for this model is given by

\[
\frac{E_b}{N_0} = \frac{\mathbb{E}[\|A\|_2^2]}{2R\mathbb{E}[\|n\|_2^2]} = \frac{(2\sigma_a^2 + \nu^2)p_{\text{avg}}}{2R\sigma_0^2}.
\]

For a given input average power and target SNR, the parameters of the Rice distribution can be set using the equation

\[
2\sigma_a^2 + \nu^2 = \frac{2R\sigma_0^2}{p_{\text{avg}}} \frac{(E_b/N_0)}{p_{\text{avg}}},
\]

To create channel variations of different SNR, we fix the variance \(\sigma_a^2\) and vary the power of the LoS component \(\nu^2\). Suppose the smallest SNR value considered is \(S_{\text{min}}\), we set \(\sigma_a^2\) using

\[
2\sigma_a^2 = \frac{2R\sigma_0^2 S_{\text{min}}}{p_{\text{avg}}},
\]

and set \(\nu\) to achieve a target SNR \(E_b/N_0\) using

\[
\nu^2 = \frac{2R\sigma_0^2 (E_b/N_0 - S_{\text{min}})}{p_{\text{avg}}}.\]

For this choice of parameters, the power ratio of LoS to non-LoS components is given by

\[
K = \frac{E_b / N_0}{S_{\text{min}}} - 1.
\]

The \(K\)-factor for Rician fading in indoor channel environments with an unobstructed line-of-sight is typically in the range 4 dB to 12 dB [44]. Rayleigh fading is obtained for \(K = 0\) (or \(\nu = 0\)).

Finally, note that the vector \(x\) is composed of one or more pairs of in-phase and quadrature (IQ) components of the encoded signal (dimension can be expressed as \(d = 2m\)). Since each IQ component is transmitted as a single RF signal, the Ricean amplitude scale is kept the same for successive pairs of IQ components in \(x\). In other words, the amplitude scales are chosen to be \(a_1, a_1, \cdots, a_m, a_m\). This does not change any of the above results.