Design and Analysis of Neural-Network-based, Single-User Codes for Multiuser Channels

N. Cameron Matson, Dinesh Rajan, Josehp Camp
Department of Electrical and Computer Engineering, Southern Methodist University, Dallas, TX, USA
Email: {cmatson, rajand, and camp}@smu.edu

***Abstract*—Inspired by its success in other fields, there have been many recent developments in the use of machine learning and neural networks to enable multiuser communication and to design efficient channel codes along with practical decoders. However, there has been little attempt to combine the results of these efforts. In this paper, for the first time, we present a neural network autoencoder architecture to jointly address both problems. The resulting codes designed by our simple and easy-to-train neural network can have arbitrary rates, are comparable to existing state-of-the-art neural network designed codes, and are directly applicable in a multiuser context. We analyze these single-user codes and characterize the design parameters which affect their performance. We then show that these same single-user codes can be used to operate close the maximum sum rate of a K-user Gaussian multiple access channel (MAC) under various SNR scenarios, without the need for retraining or learning a joint code. This improved performance is achieved by introducing a new iterative successive interference cancellation method (SIC) that outperforms traditional onion-peeling.**

***Index Terms*—machine learning, neural networks, autoencoder, channel coding, multiple access channel**

I. Introduction

Supporting multiple users simultaneously is a fundamental challenge in a shared wireless medium. Numerous multiple access schemes (such as TDMA/CDMA/OFDMA and Aloha/CSMA) have been developed at various layers of the protocol stack to solve this problem. In parallel to the development of these multiuser schemes, significant research has been done to develop optimal codes to transmit data from a single user over the wireless channel. Examples of such codes include the low-density parity check (LDPC), Reed-Solomon, Turbo, and Polar code families. The objective of this paper is to develop a family of arbitrary rate, single user codes using ideas from neural networks that also have good performance in a multiuser setting.

In recent years, there has been a flurry of interest in using machine learning techniques to help solve wireless communication problems. Several recent works [1]–[4] have used neural networks to train joint encoders and decoders for multiuser channels, specifically for the interference channel. A modulation scheme for the two-user MIMO interference channel was learned in [1]. An adaptive decoder predicted interference patterns in [2]. The only work to consider more than two users was [3] in which they learned an interference-alignment scheme. In [4], they improved upon the previous works by designing an encoder for the two-user channel that works on long block lengths. In every case, the encoders and decoders for each user were learned jointly, *i.e.*, they were optimized for a joint loss metric such as the average bit error rate (BER) of all users.

In the case of single user codes, neural networks have been used to learn novel modulation constellations [5]–[8], to build good decoders for existing codes [9]–[11], and to design entirely new codes by jointly training encoder and decoder networks [12], [13]. In [12], they trained a pair of neural networks with a Turbo-like architecture, including an interleaver to increase the memory of the code showing that it can approach state-of-the-art performance. They presented the results for codes with rates of $\frac{1}{3}$ and $\frac{1}{2}$, where these rates were dictated by the neural-network *structure* rather than being an arbitrary design parameter. A non-linear code based on the Kronecker operation present in Reed-Muller codes was presented in [13] and shown to outperform Reed-Muller and Polar codes. Due to their similarity to Reed-Muller, these non-linear codes also cannot be designed for arbitrary rates. In both cases [12], [13], arbitrary rates may be achieved via puncturing, but to the best of our knowledge, a systematic study of the performance of punctured neural-network based codes has not been done.

In contrast, our architecture is designed to be simple and easy to train with the purpose of learning arbitrary rate codes. This large family of single-user codes can be directly used on multiuser channels without the need to design or learn a joint code. The contributions of this paper are:

- We propose a simple neural network architecture, which we call NN Code, for creating single-user codes, whose performance are comparable to state-of-the-art codes. The simplicity of the architecture allows codes to be easily trained at arbitrary rates of the form $\frac{k}{n}$. We analyze these codes' performance as a function of the design parameters of these networks, particularly the gap of the rate $\frac{k}{n}$ to capacity, C.
- Using codes generated by NN Code in a multiple access channel, we can operate close to (within $\frac{1}{n}$ of) the corner points of the MAC capacity region, maximizing the sum rate. We show that our codes allow us to operate close to this maximum sum rate regardless of the signal-to-noise ratio (SNR) of each user.
- For the MAC, we introduce an iterative successive interference cancellation (SIC) step along with the same single-user decoders, which can decrease the BER of individual users by up to 60% compared to traditional

onion-peeling decoding. This process enables the addition of new users to the MAC with minimal effect on the performance of existing users.

The remainder of this paper is structured as follows: In Section II, we present the architecture for the neural network and describe the training methodology and metrics. In Section III, we describe our proposed SIC algorithm for the MAC. The results for both single- and multiuser channels are shown in Section IV, and we conclude in Section V.

II. Simple Arbitrary Rate Neural Network Codes

In this section, we present the neural network architectures used in this work and describe the network training methodology and performance metrics.

A. Single Rate Encoder/Decoder Architectures

The base neural network architecture of NN Code is shown in Fig. 1 with details in Table I. The architecture is based on autoencoder networks [9], [14] which consist of: (*i*) an encoder network, which transforms the input into a lower-dimensional representation, and (*ii*) a decoder network, which reconstructs the original input. Autoencoders have been used in computer vision and language processing tasks because of their ability to reduce dimensionality. The main difference between these models and autoencoders used for communication problems is the introduction of a noisy channel between the encoder and decoder. We consider the additive white Gaussian noise (AWGN) channel.

In our work, the encoder and decoder consist of multiple layers of 1-D convolution layers followed by a single fully connected layer. At the encoder, the input is a string of bits of length kL. The input is reshaped to be $L \times k$ and then passed through multiple convolutional layers. The convolution operates over the "L" dimension, so L must be at least the size of the convolution filter. We find that performance increases by increasing L up to a point before plateauing. Each 1-D convolutional layer is followed by a ReLu activation function. The shape is preserved throughout each convolutional layer, *i.e.*, there is no spatial down/upsampling. The final layer is fully-connected and has n nodes. Thus, the output has shape $L \times n$. The data is then passed through a learned power normalization layer, which ensures that the output signal has unit power. This is implemented via a Batch Normalization [15] layer without the scaling and centering, a technique borrowed from [12].

The architecture of the decoder is nearly identical to that of the encoder. The differences are the input shape, which matches the output of the encoder, and the final fully connected layer which has k nodes and is followed by a logistic sigmoid activation function. The logistic sigmoid, described by the function $f(x) = \frac{1}{1+e^{-x}}$, takes a real value x as input and produces a value between 0 and 1, which is commonly used to represent a "bit probability". Thus, the decoder network takes as input an nL-length, real-valued signal and outputs the probability of each of the original kL bits being a 1.

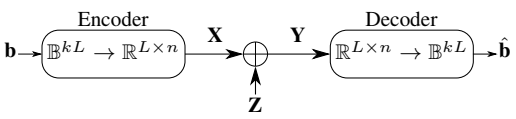


Fig. 1. Channel code autoencoder architecture.

TABLE I
NN Code Encoder Architecture

	Layer Name		Output Shape
Encoder	Input		$kL \times 1$
	Reshape		$L \times k$
	M_E layers	1-D conv. (kernel w_E, F_E filters) ReLu	$L \times k$
	Fully Connected Unit Power Normalization		$L \times n$
Decoder	Input		$L \times n$
	M_D layers	1-D conv. (kernel w_D, F_D filters) ReLu	$L \times n$
	Fully Connected Sigmoid		$L \times k$
	Reshape		$kL \times 1$

The rate, R, of the NN Code is the ratio of the lengths of the input and outputs at the encoder: $R = \frac{kL}{nL} = \frac{k}{n}$. By adjusting the values of k and n independently, we can create arbitrary rate single-user codes. The free variable L allows to scale the block length of the code without affecting the underlying rate.

B. Training Methodology and Metrics

During each feed-forward step, we generate a random binary vector, $\mathbf{b}$, kL-bits long and feed it through the encoder, which generates a continuous-valued output signal, $\mathbf{X} \in \mathbb{R}^{L\times n}$, with an average unit power constraint, $\frac{1}{nL}||\mathbf{X}||_F^2 = 1$, where $||\cdot||_F$ is the Frobenius norm. The encoder output then passes through an AWGN channel layer, which adds random noise $\mathbf{Z} \sim \mathcal{N}(\mathbf{0}, N_0\mathbf{I})$. These corrupted symbols then pass through the decoder, which outputs a probability vector $\hat{\mathbf{b}} = p(\mathbf{b})$. During training, vector $\hat{\mathbf{b}}$ along with the true bit values, $\mathbf{b}$, are used to compute the binary cross-entropy loss, $\sum_i^{kL} b_i \log(\hat{b_i}) + (1-b_i)\log(1-\hat{b_i})$, which is used to update the weights of the encoder and decoder networks via gradient descent.

We follow many of the training insights from [12], specifically training the encoder and decoder separately and using a large batch size. However, unlike [12] in which they train the encoder and decoder at different SNRs, since the coding rate of our networks is a function of the SNR, we train both the encoder and decoder at the test SNR. We also found no benefit to training the decoder more than the encoder. Hence, they are trained for an equal number of steps in each epoch. The simplicity of the NN architecture makes them easy to train. In this paper, we present the results of many networks with a wide range of rates. On average, it takes less than an

hour to train these networks on an NVIDIA P100 GPU. A summary of the network parameters can be found in Table II.

TABLE II
TRAINING PARAMETERS OF THE NETWORKS USED IN THIS PAPER.

Loss	Binary Cross-Entropy
Encoder/Decoder params	3 layers, kernel size 3, 128 filters
Batch Size	500
Optimizer, init. learning rate	Adam, 0.0001
Enc./Dec. Train steps per epoch	500
Block Length L	50
Number of epochs	< 100
(Early stopping criteria)	10 non-decreasing epochs

The BER can be calculated from the predicted bit probabilities as BER $= \frac{1}{kL}\sum_i^{kL}\left[\mathbb{1}(\hat{b}_i > 0.5) \neq b_i\right]$, where $\mathbb{1}$ is the indicator function. One of the primary objectives of this architecture is to operate at the highest theoretical rate allowed by the channel SNR. Thus, in addition to the BER, we also quantify the "goodput", which is computed as $\frac{k}{n}(1 - \text{BER})$. This goodput metric captures the trade-off between pushing more bits over the channel at the cost of more errors.

All of the results presented in Section IV are from networks created and trained in the aforementioned way. Unless otherwise specified, the codes in use are designed, trained, and tested at the maximum feasible rate, subject to the integer constraints on k and n, and capacity of the channel. Specifically, we first designate a value of n and an effective SNR value. The effective SNR is simply $\frac{P}{N_0}$ in the single user scenario and represents the signal-to-interference-plus-noise ratio (SINR) in the multiuser case as defined in Section III. Then, we calculate the maximum value of k such that the rate $R = \frac{k}{n} < C(\text{SNR})$, which implies $k = \lfloor nC(\text{SNR}) \rfloor$. Here, $C(x)$ is defined as $0.5\log_2(1 + x)$ and is the capacity of the single user real Gaussian channel[1]. All of the rates and goodputs are measured in bits per channel use (bpcu).

III. Multiple Access Channels and Successive Interference Cancellation Model

In this section, we first present the system model for the Gaussian multiple access channel. Then, we describe the traditional onion peeling method for achieving capacity, and finally present a modified, iterative successive interference cancellation (SIC) scheme to improve the performance when using our neural network codes.

A. Multiple Access Channels

Consider a multiple access channel with K users. Let X_i be the signal transmitted by user $i \in \{1...K\}$ at each time instant. The time index is not explicitly shown for simplicity. The average transmit power for user i is denoted by P_i. The received signal, Y, in this MAC is given by:

$$Y = \sqrt{h_1}X_1 + \sqrt{h_2}X_2 + \cdots + \sqrt{h_K}X_K + Z \qquad (1)$$

[1]For convenience, we consider real-valued channels.

At the receiver, the signal of each user, i, is scaled by a channel with magnitude h_i^2. The $Z \sim \mathcal{N}(0, N_0)$ is additive noise with power N_0.

For this channel, the capacity region is given by the K-tuple of rates $(R_1, R_2, \ldots R_K)$ that satisfy:

$$\sum R_i < C\left(\frac{\sum h_i P_i}{N_0}\right),\ i \subseteq \{1 \ldots K\} \qquad (2)$$

Any rate-tuple $(R_1, \ldots, R_K)$ that satisfies these constraints is achievable, *i.e.*, there exists codes (with potentially asymptotically large block sizes) that allow for transmitting at these rates and while ensuring an arbitrarily small probability of error.

B. Traditional SIC

One known method for achieving the capacity of the MAC is the use of what's called an onion peeling decoder. In this decoding process, a first user is decoded while treating the remaining users as noise. Once the first user is decoded, their bits are re-encoded and subtracted from the original signal. This procedure continues with each successive user, cancelling out each one from the original signal. As long as each user transmits at a rate below the single user capacity of their effective channel, then this scheme achieves capacity.

In practice, the SIC proceeds as follows. Without loss of generality, we order the users $1 \ldots K$ in the reverse order in which they are decoded, *i.e.*, user K is decoded first, and user 1 is decoded last. The rate for user i is then computed as $C(\frac{h_i P_i}{N_0 + \sum_{j<i} h_j P_j})$, where the argument to the capacity function is the SINR for user i. With this transmission rate, the signal of user K can be decoded (using the single user decoder) since it is operating at a rate below the capacity of the channel which treats the signal of all the other users as noise. After successful decoding, the re-encoded signal of user K can be subtracted from the received signal Y to obtain the residual signal $Y - \hat{X}_K$. This decoding process continues with the decoding of the other users.

C. Iterative SIC

In theory, since each ordered user is using a code below the capacity of their effective channel, the receiver should be able to decode each signal in order without errors. In practice, our decoders are not perfect, meaning there is always some error in decoding. Also, the codes themselves are not perfectly Gaussian, meaning the interference is not Gaussian. Hence, we may be inaccurately approximating each user's capacity as that of a single-user Gaussian channel, given their effective SINR. The resulting error in decoding leads to an imperfect estimate of the signal, *i.e.*, $\hat{X}_i \neq X_i$. Thus, there is residual error when decoding subsequent users. To overcome this limitation, we propose a modification to the SIC scheme in which we iterate multiple times through each user's decoder. The modified procedure is described in Algorithm 1.

At the receiver, we first initialize all of the estimated signals, $\hat{X}_i$ to 0 (step 1). Then, we process the received signal for T iterations (choosing T is discussed in Section IV-B), each time removing the interfering signal estimates (step 4), estimating

the user's bits (step 5), and re-estimating that user's transmitted signal (step 6). Note that in the first iteration, and the first user we have $\sum_{j\neq i}\hat{X}_j = 0 \implies \hat{Y}_i = Y$; in other words, since the estimates are initialized to 0, stopping after the first iteration is equivalent to traditional SIC.

Algorithm 1 Generalized Iterative SIC

```
Initialize X̂_i = 0, ∀i
for t ← 1, T do
    for all users i do
        Ŷ_i = Y − Σ_{j≠i} X̂_j
        b̂_i = D_i(Ŷ_i)
        X̂_i = E_i(b̂_i)
    end for
end for
```

IV. Numerical Results

We trained our NN Code architecture with different values of k, n, and SNR on single-user channels and then utilized those codes in conjunction with SIC on MACs. In this section, we first present the performance and characterize the behavior of the NN Code for single-user channels. Then we demonstrate the effectiveness of using these codes in multiuser channels.

A. Single User Results

1) Arbitrary Rate Goodput Performance: The primary benefit of this architecture is its ability to operate at any rate. The rate of the code is a function of the variables k and n rather than that of the architecture of the neural network itself. Thus, different rate codes can be learned for different SNR regimes. The goodput metric helps capture the dual goals of sending more bits over the channel with the fewest number of errors. Fig. 2 shows the goodput for different NN Codes operating "at capacity" for the given SNR as described in Section II-B as well as the that of the two TruboAE codes from [12].

For $n = 2$, the goodput of both our code and TurboAE is flat for all valid SNRs. The rate is the same ($\frac{1}{2}$) for both codes, regardless of the SNR. Thus, any difference in the goodput is due to differences in BER. For $n = 3$, the situation is similar at low SNR, but when the SNR increases sufficiently, the NN Code can increase its rate from $\frac{1}{3}$ to $\frac{2}{3}$ to take advantage of the increased capacity of the channel by adjusting k.

An advantage of NN Code is its the ability to increase n to allow the desired, fractional rate change with SNR. The goodput curve for $n = 30$ demonstrates that, with sufficiently large n, the goodput increases smoothly with SNR, which minimizes the difference between the goodput and capacity. Any family of codes, found via neural networks or otherwise, that allows for arbitrary rates will be able to adjust the rate with SNR. However, ***good* codes of arbitrary rates may not be available** either through explicit construction or a method such as puncturing, whereas the flexibility and simplicity of NN Code makes it trivial to train codes of any rate.

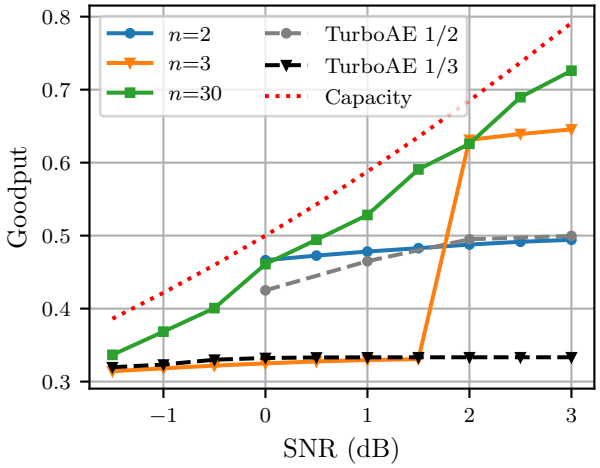


Fig. 2. Goodput performance of arbitrary rate NN Codes vs. fixed rate TurboAE [12]. The arbitrary rate allows NN Code to scale with SNR.

2) Characterization of Neural Network Codes: The BER and goodput of the codes presented in this paper and the codes they produce are determined by three parameters: n, k, and the channel SNR.

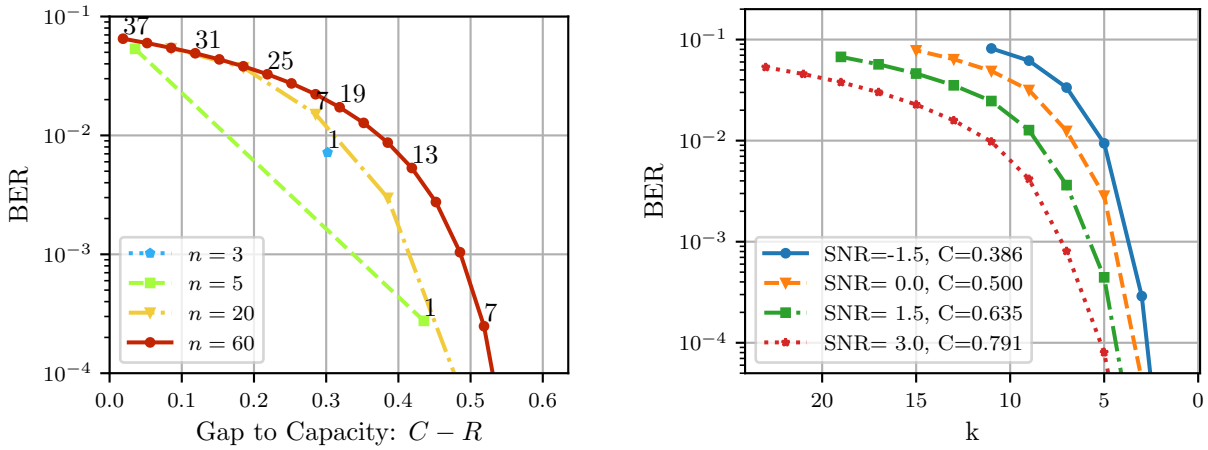
The parameter n determines the granularity of the rates that the NN Code can produce. As discussed above, a large value of n is what enables the NN Code to scale the rate of the network with the SNR of the channel, allowing codes arbitrarily close to the capacity. The trade-off is that n also determines the size of the neural network. A larger n results in a larger network, which is harder to train. In general, for equal rates, a network with a larger n will have a higher BER. Large n also allows us to generate arbitrarily *low* rate codes, an important requirement for use in the multiple access channel when the interference can cause very low effective SNRs.

The second variable that determines the final BER of the NN Code is the difference between the rate of the NN Code and the capacity of the channel (*i.e.*, the gap to capacity). For a fixed value of n, we increase the gap to capacity by reducing the rate (i.e., reducing k). By increasing the gap to capacity, we reduce the BER. This relationship is present in all channel codes: the code adds redundancy, and the amount of redundancy depends on the desired error rate for a given channel. Using the same code on a channel with a higher SNR (*i.e.*, a channel with larger capacity) lowers the error rate.

We can see the effect of the gap to capacity on BER clearly in Fig. 3a, which shows the BER versus the gap to capacity for different values of n at a fixed SNR of 1.5 dB. Some of the points are marked with the corresponding value of k. This effect is consistent across SNR as well. Fig. 3b shows the BER versus k for the NN Code with $n = 30$ at four different SNRs, showing that with a large n and small k, we can create low rate codes that provide enough gap to achieve arbitrarily small errors rates.

B. MAC Results

We now present the results of using our NN Code in a SIC scheme to approach the capacity of the MAC. For each user in



(a) (b)

Fig. 3. (a) BER of NN Codes at different gaps to capacity over a 1.5 dB channel. The gap is controlled via selection of k and n for a given channel SNR. Each line is a different n and some sample values of k are included in the figure. (b) BER vs. k of different NN Codes with $n = 30$ at different SNRs.

the MAC, the codes are designed according to Section III, and each of the users operate with single-user codes "at capacity", as described in Section II-B.

1) Iterative SIC: First, we demonstrate the effectiveness of the modified SIC algorithm to reduce BER. Fig. 4 shows the reduction in BER versus the number of iterations on both a two- and three-user MAC. In this result, each user's SNR is 1.5 dB. In the two-user MAC, the user 2 BER improves by roughly half after 50 iterations; while in the three-user MAC, the user 3 BER decreases by 65% and the user 2 BER by 45% from the initial value after one iteration (traditional SIC). Across all SNR scenarios tested, in which each user's SNR $\in \{-1.5, 0, 1.5, 3\}$, the average BER improvement for users 1 and 2 in the 2-MAC was 13% and 45%, respectively, and $20\%, 40\%, 60\%$ for users 1 through 3 in the 3-MAC.

The first-decoded user benefits the most in the iterative scheme versus traditional SIC. Without iterations, the receiver decodes the user K with the full interference of the other users. During the following iterations, the receiver has reliable estimates of the other users and can cancel them prior to decoding user K. The effect is smaller for the other users since they are initially decoded with some interference cancellation. After a few iterations, the improvement in BER tapers off as the receiver's estimate of each user's signal converges.

2) SIC MAC Results: We now present the results of the SIC scheme with our NN Codes by comparing the final goodputs after 50 SIC iterations with the capacity region of the two-user MAC. Fig. 5a shows the two-user capacity region for a symmetric MAC with $\text{SNR}_1 = \text{SNR}_2 = 1.5\text{dB}$, while Fig. 5b shows an asymmetric scenario of $(\text{SNR}_1, \text{SNR}_2) = (-1.5, 3)$. In each figure, there are several quantities marked. Three points marked with a black "*" on the boundary of the pentagon achieve the maximum sum rate. Points "B" and "D" can be achieved via SIC by decoding user 1 or user 2

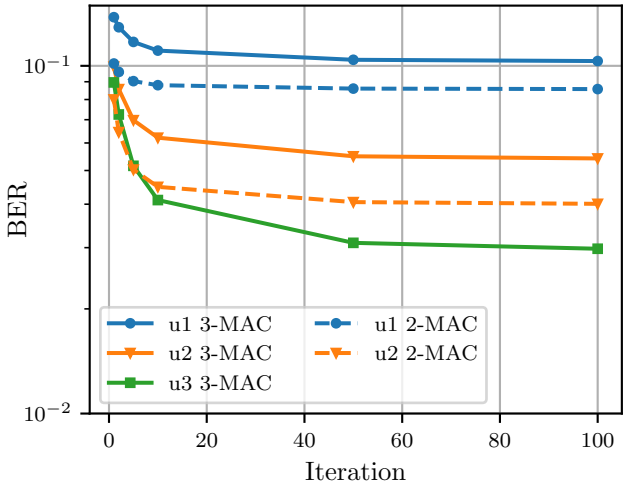


Fig. 4. Performing multiple iterations of SIC improves BER for all users.

first, respectively. Point "C" can be achieved via time-sharing between the two rate pairs. Each figure also shows (gray pentagon) the actual code rate tuple (R_1, R_2), where $R_i = \frac{k_i}{n_i}$ for the NN Code with $n = 30$. This operating point is very close to the boundary because of the relatively large n. Three green square markers represent the goodput of each user operating at their respective single-user rate and the goodput resulting between an even time-sharing between the two. Note that the middle green marker corresponds to a TDMA scheme. Finally, the red dots indicate the final goodput of the modified SIC algorithm after 50 iterations.

The SIC scheme is clearly better than TDMA. Even if the two users operate with zero errors at the individual single-user capacities, their sum goodput is lower than that achieved by the SIC scheme. Also, to maximize the sum goodput is it is critical that the NN Codes operate at the highest rate possible, which is only possible with large values of n and the freedom to choose k, which is easily realized by NN Code.

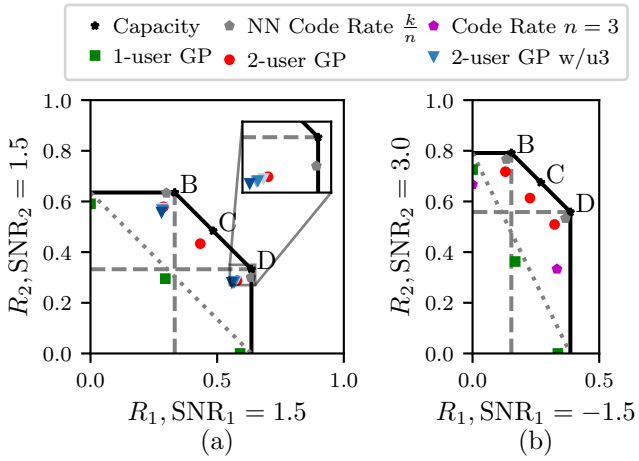


Fig. 5. Two 2-User MAC capacity regions for different SNR regimes (a) symmetric: $\mathrm{SNR}_1 = \mathrm{SNR}_2 = 1.5$dB, (b) asymmetric: $\mathrm{SNR}_{1,2} = \{-1.5, 3.0\}$.

Points "B" and "D" both maximize the sum rate of users 1 and 2. In Fig. 5a, since each user has the same SNR, there is no difference in operating at point "B" versus "D". In other words, the sum rate is the same whichever user is decoded first. This symmetry is not necessarily the case when the single-user SNRs differ, as in Fig. 5b. In this scenario, the goodput at points "B" and "D" are 0.897 and 0.882, respectively. This minor difference is due to the constraint that the actual rate R needs to be a ratio of two integers (k, n), while the capacity is, in general, an irrational number. While the sum rates of each operational point is the same, 0.900 in this case, the individual rates are different depending on which user is decoded first, *i.e.*, $(R_1, R_2) = (0.133, 0.767)$ at "B" and $(0.367, 0.533)$ at "D". Since the single-user performance depends on both the rate and the effective SNR of the channel, the individual and thus sum goodputs are different.

The small difference in the goodputs between operating near "B" vs. "D" is because n is sufficiently large, which results in a consistent gap to capacity for any SNR. With a smaller n, the difference between "B" and "D" can be significant. For example, Fig. 5b also shows (magenta pentagon) the actual code rates for $n = 3$. Not only are the operating points significantly further away from their corner points, but there is no user 2 rate sufficiently low to operate at point "B."

3) Scalability: In Fig. 5a, we have also plotted (blue triangles) the two-user goodput pair of users 1 and 2 from the three-user MAC when user 3 is at different SNRs. The inset shows a zoomed-in view of the area around point "D". Since SNR_1 and SNR_2 are symmetrical, the sum goodput is the same at both points "B" and "D". Without user 3, the sum goodput is 0.896. With user 3, the sum goodput of users 1 and 2 decreases slightly as SNR_3 increases. At the lowest SNR, the reduction in the two-user goodput was less than 1%. The maximum reduction is only 3%. This demonstrates the ability of the SIC scheme with the NN Code to scale to more than two users, all while using single-user codes. Scaling to even larger numbers will be studied in the future.

V. Conclusion

In this paper, we have have presented a simple neural network architecture that allows the creation of arbitrary rate single-user channel codes. The flexibility of the design makes it possible to scale the code rate along with the capacity of the channel, and their simplicity makes them easy to train. We have shown the usefulness of these NN single-user codes in operating close to the corner points of the MAC capacity region, which is only possible when arbitrary rate codes are available. Our iterative SIC scheme markedly reduces the BER of decoding each user, which makes it possible to add additional users without significantly impacting the goodputs of existing users. Future work will be to incorporate more sophisticated architectures to reduce the single-user BER, scaling the MAC to more users, and using arbitrary rate codes to investigate other multiuser channels.

References

[1] T. Erpek, T. J. O'Shea, and T. C. Clancy, "Learning a Physical Layer Scheme for the MIMO Interference Channel," in *2018 IEEE International Conference on Communications (ICC)*, May 2018.

[2] D. Wu, M. Nekovee, and Y. Wang, "Deep Learning-Based Autoencoder for m-User Wireless Interference Channel Physical Layer Design," *IEEE Access*, vol. 8, 2020.

[3] R. K. Mishra, K. Chahine, H. Kim, S. Jafar, and S. Vishwanath, "Distributed Interference Alignment for K-user Interference Channels via Deep Learning," in *2021 IEEE International Symposium on Information Theory (ISIT)*, Jul. 2021.

[4] K. Chahine, N. Ye, and H. Kim, "DeepIC: Coding for Interference Channels via Deep Learning," Aug. 2021, arXiv: 2108.06028.

[5] S. Dörner, S. Cammerer, J. Hoydis, and S. t. Brink, "Deep Learning Based Communication Over the Air," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, Feb. 2018.

[6] T. J. O'Shea, T. Roy, N. West, and B. C. Hilburn, "Physical Layer Communications System Design Over-the-Air Using Adversarial Networks," in *2018 26th European Signal Processing Conference (EUSIPCO)*, Sep. 2018.

[7] Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh, and P. Viswanath, "Joint Channel Coding and Modulation via Deep Learning," in *2020 IEEE SPAWC*, May 2020.

[8] B. Zhu, J. Wang, L. He, and J. Song, "Joint Transceiver Optimization for Wireless Communication PHY Using Neural Network," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, Jun. 2019.

[9] H. Kim, Y. Jiang, R. Rana, S. Kannan, S. Oh, and P. Viswanath, "Communication Algorithms via Deep Learning," May 2018, arXiv: 1805.09317.

[10] Y. Jiang, S. Kannan, H. Kim, S. Oh, H. Asnani, and P. Viswanath, "DEEPTURBO: Deep Turbo Decoder," in *2019 IEEE SPAWC*, Jul. 2019.

[11] M. V. Jamali, X. Liu, A. V. Makkuva, H. Mahdavifar, S. Oh, and P. Viswanath, "Reed-Muller Subcodes: Machine Learning-Aided Design of Efficient Soft Recursive Decoding," in *2021 IEEE International Symposium on Information Theory (ISIT)*, Jul. 2021.

[12] Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh, and P. Viswanath, "Turbo Autoencoder: Deep learning based channel codes for point-to-point communication channels," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[13] A. V. Makkuva, X. Liu, M. V. Jamali, H. Mahdavifar, S. Oh, and P. Viswanath, "KO codes: inventing nonlinear encoding and decoding for reliable wireless communication via deep-learning," in *Proceedings of the 38th International Conference on Machine Learning*, Jul. 2021.

[14] T. O'Shea and J. Hoydis, "An Introduction to Deep Learning for the Physical Layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, Dec. 2017.

[15] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proceedings of the 32nd International Conference on Machine Learning*, Jun. 2015.