On Deep Learning-Based Channel Decoding

Tobias Gruber*, Sebastian Cammerer*, Jakob Hoydis†, and Stephan ten Brink*
* Institute of Telecommunications, Pfaffenwaldring 47, University of Stuttgart, 70659 Stuttgart, Germany
{gruber,cammerer,tenbrink}@inue.uni-stuttgart.de
†Nokia Bell Labs, Route de Villejust, 91620 Nozay, France
jakob.hoydis@nokia-bell-labs.com

***Abstract*—We revisit the idea of using deep neural networks for one-shot decoding of random and structured codes, such as polar codes. Although it is possible to achieve maximum a posteriori (MAP) bit error rate (BER) performance for both code families and for short codeword lengths, we observe that (i) structured codes are easier to learn and (ii) the neural network is able to generalize to codewords that it has never seen during training for structured, but not for random codes. These results provide some evidence that neural networks can learn a form of decoding algorithm, rather than only a simple classifier. We introduce the metric *normalized validation error (NVE)* in order to further investigate the potential and limitations of deep learning-based decoding with respect to performance and complexity.**

I. Introduction

Deep learning-based channel decoding is doomed by the *curse of dimensionality* [1]: for a short code of length $N = 100$ and rate $r = 0.5$, 2^{50} different codewords exist, which are far too many to fully train any neural network (NN) in practice. The only way that a NN can be trained for practical blocklengths is, if it learns some form of decoding algorithm which can infer the full codebook from training on a small fraction of codewords. However, to be able to learn a decoding algorithm, the code itself must have some structure which is based on a simple encoding rule, like in the case of convolutional or algebraic codes. The goal of this paper is to shed some light on the question whether structured codes are easier to "learn" than random codes, and whether a NN can decode codewords that it has never seen during training.

We want to emphasize that this work is based on very short blocklengths, i.e., $N \leq 64$, which enables the comparison with maximum a posteriori (MAP) decoding, but also has an independent interest for practical applications such as the internet of things (IoT). We are currently restricted to short codes because of the exponential training complexity [1]. Thus, the neural network decoding (NND) concept is currently not competitive with state-of-the-art decoding algorithms which have been highly optimized over the last decades and scale to arbitrary blocklengths.

Yet, there may be certain code structures which facilitate the learning process. One of our key finding is that structured codes are indeed easier to learn than random codes, i.e., less training epochs are required. Additionally, our results indicate that NNs may generalize or "interpolate" to the full codebook after having seen only a subset of examples, whenever the code has structure.

A. Related Work

In 1943, McCulloch and Pitts published the idea of a NN that models the architecture of the human brain in order to solve problems [2]. But it took about 45 years until the backpropagation algorithm [3] made useful applications such as handwritten ZIP code recognition possible [4]. One early form of a NN is a Hopfield net [5]. This concept was shown to be similar to maximum likelihood decoding (MLD) of linear block error-correcting codes (ECCs) [6]: an erroneous codeword will converge to the nearest stable state of the Hopfield net which represents the most likely codeword. A naive implementation of MLD means correlating the received vector of modulated symbols with all possible codewords which makes it infeasible for most practible codeword lengths, as the decoding complexity is $\mathcal{O}\left(2^k\right)$ with k denoting the number of information bits in the codeword. The parallel computing capabilities of NNs allow us to solve or, at least, approximate the MLD problem in polynomial time [7]. Moreover, the weights of the NN are precomputed during training and the decoding step itself is then relatively simple.

Due to its low storage capacity, Hopfield nets were soon replaced by feed-forward NNs which can learn an appropriate mapping between noisy input patterns and codewords. No assumption has to be made about the statistics of the channel noise because the NN is able to learn the mapping or to extract the channel statistics during the learning process [8]. Different ideas around the use of NN for decoding emerged in the 90s. While in [8] the output nodes represent the bits of the codeword, it is also possible to use one output node per codeword (*one-hot* coding) [9]. For Hamming coding, another variation is to use only the syndrome as input of the NN in order to find the most likely error pattern [10]. Subsequently, NND for convolutional codes arose in 1996 when Wang and Wicker showed that NND matches the performance of an ideal Viterbi decoder [1]. But they also mentioned a very important drawback of NND: decoding problems have far more possibilities than conventional pattern recognition problems. This limits the NND to short codes. However, the NN decoder for convolutional codes was further improved by using recurrent neural nets [11].

NND did not achieve any big breakthrough for neither block nor convolutional codes. Due to the standard training techniques in those times it was not possible to work with NNs employing a large number of neurons and layers, which ren-

978-1-5090-4780-2/17/$31.00 ©2017 IEEE

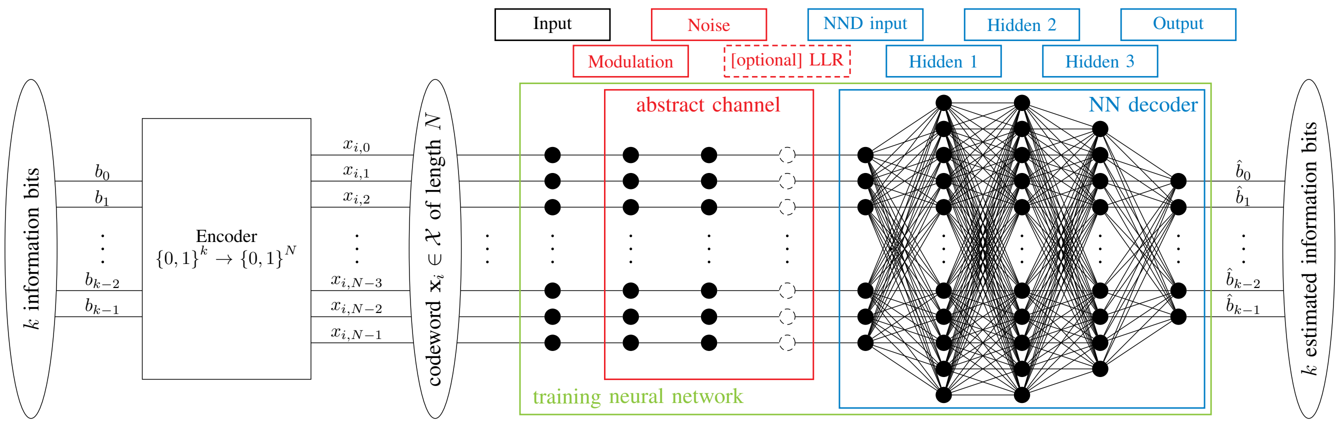


Fig. 1: Deep learning setup for channel coding.

dered them unsuited for longer codewords. Hence, the interest in NNs dwindled, not only for machine learning applications but also for decoding purposes. Some slight improvements were made in the following years, e.g., by using random neural nets [12] or by reducing the number of weights [13].

In 2006, a new training technique, called layer-by-layer unsupervised pre-training followed by gradient descent fine-tuning [14], led to the renaissance of NNs because it made training of NNs with more layers feasible. NNs with many hidden layers are called *deep*. Nowadays, powerful new hardware such as graphical processing units (GPUs) are available to speed up learning as well as inference. In this renaissance of NNs, new NND ideas emerge. Yet, compared to previous work, the NN learning techniques are only used to optimize well known decoding schemes which we denote as introduction of expert knowledge. For instance, in [15], weights are assigned to the Tanner graph of the belief propagation (BP) algorithm and learned by NN techniques in order to improve the BP algorithm. It still seems that the recent advances in the machine learning community have not yet been adapted to the pure idea of *learning to decode*.

II. Deep Learning for Channel Coding

The theory of deep learning is comprehensively described in [16]. Nevertheless, for completeness, we will briefly explain the main ideas and concepts in order to introduce a NN for channel (de-)coding and its terminology. A NN consists of many connected neurons. In such a neuron all of its weighted inputs are added up, a bias is optionally added, and the result is propagated through a nonlinear activation function, e.g., a sigmoid function or a rectified linear unit (ReLU), which are respectively defined as

$$g_{\text{sigmoid}}(z) = \frac{1}{1+e^{-z}}, \qquad g_{\text{relu}}(z) = \max\{0, z\}. \quad (1)$$

If the neurons are arranged in layers without feedback connections we speak of a feedforward NN because information flows through the net from the left to the right without feedback (see Fig. 1). Each layer i with n_i inputs and m_i outputs performs the mapping $\mathbf{f}^{(i)} : \mathbb{R}^{n_i} \to \mathbb{R}^{m_i}$ with the weights and biases of the neurons as parameters. Denoting $\boldsymbol{v}$ as input and $\boldsymbol{w}$ as output of the NN, an input-output mapping is defined by a chain of functions depending on the set of parameters $\boldsymbol{\theta}$ by

$$\boldsymbol{w} = \mathbf{f}(\boldsymbol{v}; \boldsymbol{\theta}) = \mathbf{f}^{(L-1)}\left(\mathbf{f}^{(L-2)}\left(\ldots\left(\mathbf{f}^{(0)}(\boldsymbol{v})\right)\right)\right) \quad (2)$$

where L gives the number of layers and is also called *depth*. It was shown in [17] that such a multilayer NN with $L = 2$ and nonlinear activation functions can theoretically approximate any continuous function on a bounded region arbitrarily closely—if the number of neurons is large enough.

In order to find the optimal weights of the NN, a training set of known input-output mappings is required and a specific loss function has to be defined. By the use of gradient descent optimization methods and the backpropagation algorithm [3], weights of the NN can be found which minimize the loss function over the training set. The goal of training is to enable the NN to find the correct outputs for unseen inputs. This is called generalization. In order to quantify the generalization ability, the loss can be determined for a data set that has not been used for training, the so-called validation set.

In this work, we want to use a NN for decoding of noisy codewords. At the transmitter, k information bits are encoded into a codeword of length N. The coded bits are modulated and transmitted over a noisy channel. At the receiver, a noisy version of the codeword is received and the task of the decoder is to recover the corresponding information bits. In comparison to iterative decoding, the NN finds its estimate by passing each layer only once. As this principle enables low-latency implementations, we term it *one-shot* decoding.

Obtaining labeled training data is usually a very hard and expensive task for the field of machine learning. But using NN for channel coding is special because we deal with man-made signals. Therefore, we are able to generate as many training samples as we like. Moreover, the desired NN output, also denoted as label, is obtained for free because if noisy codewords are generated, the transmitted information bits are obviously known. For the sake of simplicity, binary phase shift keying (BPSK) modulation and an additive white Gaussian noise (AWGN) channel is used. Other channels can be adopted straightforwardly, and it is this flexibility that may be a particular advantage of NN-based decoding.

In order to keep the training set small it is possible to extend the NN with additional layers for modulating and adding noise (see Fig. 1). These additional layers have no trainable parameters, i.e., they perform a certain action such as adding noise and propagate this value only to the node of the next layer with the same index. Instead of creating, and thus storing, many noisy versions of the same codeword, working on the noiseless codeword is sufficient. Thus, the training set $\mathcal{X}$ consists of all possible codewords $\mathbf{x}_i \in \mathbb{F}_2^N$ with $\mathbb{F}_2 \in \{0,1\}$ (the labels being the corresponding information bits) and is given by $\mathcal{X} = \{\mathbf{x}_0, \ldots, \mathbf{x}_{2^k-1}\}$.

As recommended in [16], each hidden layer employs a ReLU activation function because it is nonlinear and at the same time very close to linear which helps during optimization. Since the output layer represents the information bits, a sigmoid function forces the output neurons to be in between zero and one, which can be interpreted as the probability that a "1" was transmitted. If the probability is close to the bit of the label, the loss should be incremented only slightly whereas large errors should result in a very large loss. Examples for such loss functions are the mean squared error (MSE) and the binary cross-entropy (BCE), defined respectively as

$$\mathrm{L}_{\mathrm{MSE}} = \frac{1}{k}\sum_i \left(b_i - \hat{b}_i\right)^2 \tag{3}$$

$$\mathrm{L}_{\mathrm{BCE}} = -\frac{1}{k}\sum_i \left[b_i \ln\left(\hat{b}_i\right) + (1-b_i)\ln\left(1-\hat{b}_i\right)\right] \tag{4}$$

where $b_i \in \{0,1\}$ is the ith target information bit (label) and $\hat{b}_i \in [0,1]$ the NN soft estimate.

There are some alternatives for this setup. First, log-likelihood ratio (LLR) values could be used instead of channel values. For BPSK modulation over an AWGN channel, these are obtained by

$$\mathrm{LLR}(y) = \ln\frac{P(x=0|y)}{P(x=1|y)} = \frac{2}{\sigma^2}y \tag{5}$$

where σ^2 is the noise power and y the received channel value. This processing step can be also implemented as an additional layer without any trainable parameters. Note, that the noise variance must be known in this case and provided as an additional input to the NN.[1] Representing the information bits in the output layer as a *one-hot*-coded vector of length 2^k is another variant. However, we refrain from this idea since it does not scale to large values of k. Freely available open-source machine learning libraries, such as Theano[2], help to implement and train complex NN models on fast concurrent GPU architectures. We use Keras[3] as a convenient high-level abstraction front-end for Theano. It allows to quickly deploy NNs from a very abstract point of view in the Python programming language that hides away a lot of the underlying complexity. As we support reproducible research, we have made parts of the source code of this paper available.[4]

[1]Inspired by the idea of spatial transformer networks [18], one could alternatively use a second NN to estimate σ^2 from the input and provide this estimate as an additional parameter to the LLR layer.

[2]https://github.com/Theano/Theano

[3]https://github.com/fchollet/keras

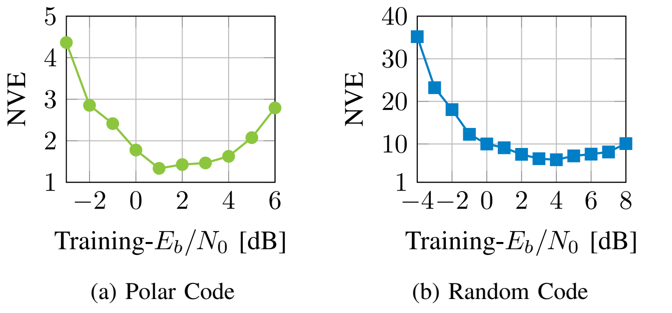


Fig. 2: NVE versus training-E_b/N_0 for 16 bit-length codes for a 128-64-32 NN trained with $M_{\mathrm{ep}} = 2^{16}$ training epochs.

III. Learn to Decode

In the sequel, we will consider two different code families: random codes and structured codes, namely polar codes [19]. Both have codeword length $N = 16$ and code rate $r = 0.5$. While random codes are generated by randomly picking codewords from the codeword space with a Hamming distance larger than two, the generator matrix of polar codes of block size $N = 2^n$ is given by

$$\mathbf{G}_N = \mathbf{F}^{\otimes n}, \qquad \mathbf{F} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{6}$$

where $\mathbf{F}^{\otimes n}$ denotes the nth Kronecker power of $\mathbf{F}$. The codewords are now obtained by $\mathbf{x} = \mathbf{u}\mathbf{G}_N$, where $\mathbf{u}$ contains k information bits and $N-k$ frozen positions, for details we refer to [19]. This way, polar codes are inherently structured.

A. Design parameters of NND

Our starting point is a NN as described before (see Fig. 1). We introduce the notation 128-64-32 which describes the design of the NN decoder employing three hidden layers with 128, 64, and 32 nodes, respectively. However, there are other design parameters with a non-negligible performance impact:

1) What is the best training signal-to-noise-ratio (SNR)?
2) How many training samples are necessary?
3) Is it easier to learn from LLR channel output values rather than from the direct channel output?
4) What is an appropriate loss function?
5) How many layers and nodes should the NN employ?
6) Which type of regularization[5] should be used?

The area of research dealing with the optimization of these parameters is called *hyperparameter optimization* [20]. In this work, we do not further consider this optimization and restrict ourselves to a fixed set of hyperparameters which we have found to achieve good results. Our focus is on the differences between random and structured codes.

[4]https://github.com/gruberto/DL-ChannelDecoding

[5]Regularization is any method that trades-off a larger training error against a smaller validation error. An overview of such techniques is provided in [16, Ch. 7]. We do not use any regularization techniques in this work, but leave it as an interesting future investigation.

Since the performance of NND depends not only on the SNR of the validation data set (for which the bit error rate (BER) is computed) but also on the SNR of the training data set[6], we define below a new performance metric, the *normalized validation error (NVE)*. Denote by ρ_t and ρ_v the SNR (measured as E_b/N_0) of the training and validation data sets, respectively, and let $\text{BER}_\text{NND}(\rho_\text{t}, \rho_\text{v})$ be the BER achived by a NN trained at ρ_t on data with ρ_v. Similarly, let $\text{BER}_\text{MAP}(\rho_\text{v})$ be the BER of MAP decoding at SNR ρ_v. For a set of S different validation data sets with SNRs $\rho_{\text{v},1}, \dots, \rho_{\text{v},S}$, the NVE is defined as

$$\text{NVE}(\rho_\text{t}) = \frac{1}{S}\sum_{s=1}^{S} \frac{\text{BER}_\text{NND}(\rho_\text{t}, \rho_{\text{v},s})}{\text{BER}_\text{MAP}(\rho_{\text{v},s})}. \quad (7)$$

The NVE measures how good a NND, trained at a particular SNR, is compared to MAP decoding over a range of different SNRs. Obviously, for NVE $= 1$, the NN achieves MAP performance, but is generally greater. In the sequel, we compute the NVE over $S = 20$ different SNR points from 0 dB to 5 dB with a validation set size of 20000 examples for each SNR.

We train our NN decoder in so-called "epochs". In each epoch, the gradient of the loss function is calculated over the entire training set $\mathcal{X}$ using *Adam*', a method for stochastic gradient descent optimization [22]. Since the noise layer in our architecture generates a new noise realization each time it is used, the NN decoder will never see the same input twice. For this reason, although the training set has a limited size of 2^k codewords, we can train on an essentially unlimited training set by simply increasing the number of epochs M_ep. However, this makes it impossible to distinguish whether the NN is improved by a larger amount of training samples or more optimization iterations.

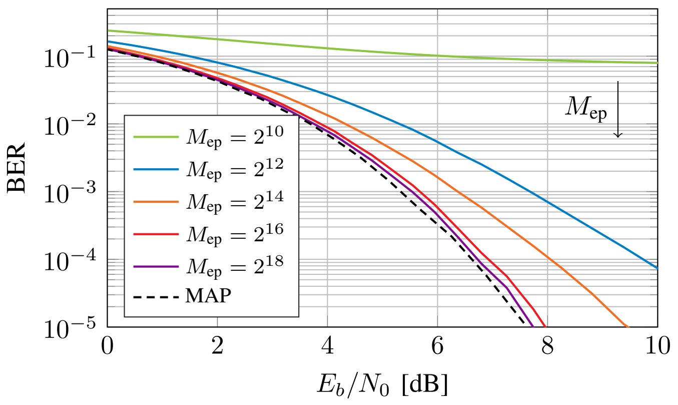
Starting with a NN decoder architecture of 128-64-32 and $M_\text{ep} = 2^{22}$ learning epochs, we train the NN with datasets of different training SNRs and evaluate the resulting NVE. The result is shown in Fig. 2, from which it can be seen that there is an "optimal" training E_b/N_0. An explanation for the occurrence of an optimum can be explained by the two cases:

1) $E_b/N_0 \to \infty$; train without noise, the NN is not trained to handle noise.
2) $E_b/N_0 \to 0$; train only with noise, the NN can not learn the code structure.

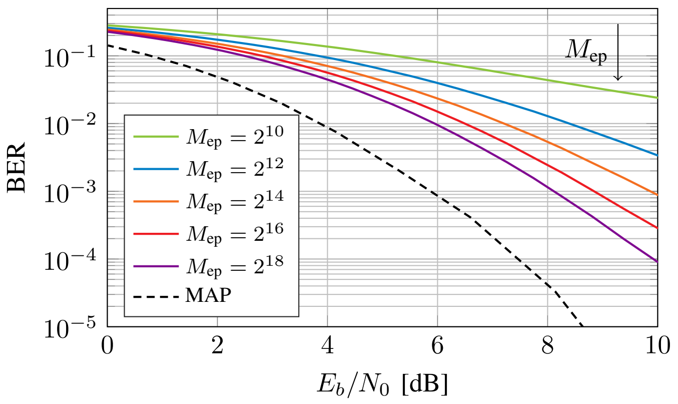
This clearly indicates an optimum somewhere in between these two cases. From now on, a training E_b/N_0 of 1 dB and 4 dB is chosen for polar and random codes, respectively.

Fig. 3 shows the BER achieved by a very small NN of dimensions 128-64-32 as a function of the number of training epochs ranging from $M_\text{ep} = 2^{10}, \dots, 2^{18}$. For BER simulations, we use 1 million codewords per SNR point. For both code families, the larger the number of training epochs, the closer is the gap between MAP and NND performance.

[6]It would also be possible to have a training data set which contains a mix of different SNR values, but we have not investigated this option here. Recently, the authors in [21] observed that starting at a high training SNR and then gradually reducing the SNR works well.



(a) Polar Code



(b) Random Code

Fig. 3: Influence of the number of epochs M_ep on the BER of a 128-64-32 NN for 16 bit-length codes with code rate $r = 0.5$.

However, for polar codes, close to MAP performance is already achieved for $M_\text{ep} = 2^{18}$ epochs, while we may need a larger NN or more training epochs for random codes.

In Fig. 4, we illustrate the influence of direct channel values versus channel LLR values as decoder input in combination with two loss functions, MSE and BCE. The NVE for all combinations is plotted as a function of the number of training epochs. Such a curve is also called "learning curve" since it shows the process of learning. Although it is ususally recommended to normalize the NN inputs to have zero mean and unit variance, we train the NN without any normalization which seems to be sufficient for our setup. For a few training epochs, the LLR input improves the learning process; however, this advantage disappears for a larger M_ep. The same holds for BCE against MSE. For polar codes with LLR values and BCE the learning appears not to converge for the applied number of epochs. In summary, for training the NN with a large number of training epochs it does not matter if LLR or channel values are used as inputs and which loss function is employed. Moreover, normalization is not required.

In order to answer the question how large the NN should be, we trained NNs with different sizes and structures. From Fig. 5, we can conclude that, for both polar and random codes, it is possible to achieve MAP performance. Moreover, and somewhat surprisingly, the larger the net, the less training

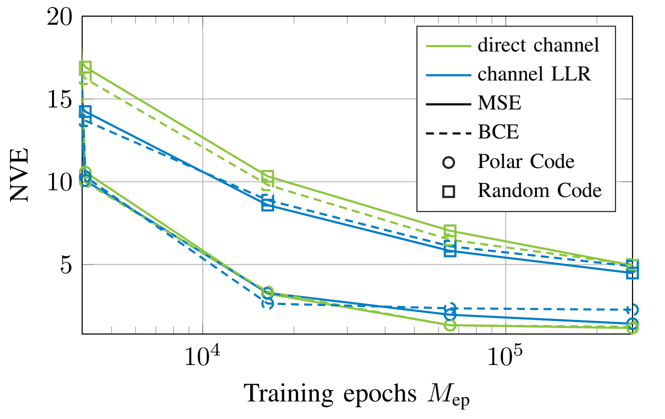


Fig. 4: Learning curve for 16 bit-length codes with code rate $r = 0.5$ for a 128-64-32 NN.

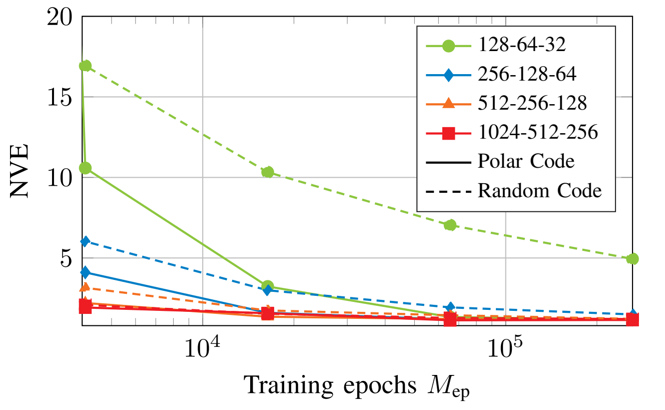


Fig. 5: Learning curve for different NN sizes for 16 bit-length codes with code rate $r = 0.5$.

epochs are necessary. In general, the larger the number of layers and neurons, the larger is the expressive power or *capacity* of the NN [16]. Contrary to what is common in classic machine learning tasks, increasing the network size does not lead to overfitting since the network never sees the same input twice.

B. Scalability

Up to now, we have only considered 16 bit-length codes which are of little practical importance. Therefore, the scalability of the NN decoder is investigated in Fig. 6. One can see that the length N is not crucial to learn a code by deep learning techniques. What matters, however, is the number of information bits k that determines the number of different classes (2^k) which the NN has to distinguish. For this reason, the NVE increases exponentially for larger values of k for a NN of fixed size and fixed number of training epochs. If a NN decoder is supposed to scale, it must be able to generalize from a few training examples. In other words, rather than learning to classify 2^k different codewords, the NN decoder should learn a decoding algorithm which provides the correct output for any possible codeword. In the next section, we investigate whether structure allows for some form of generalization.

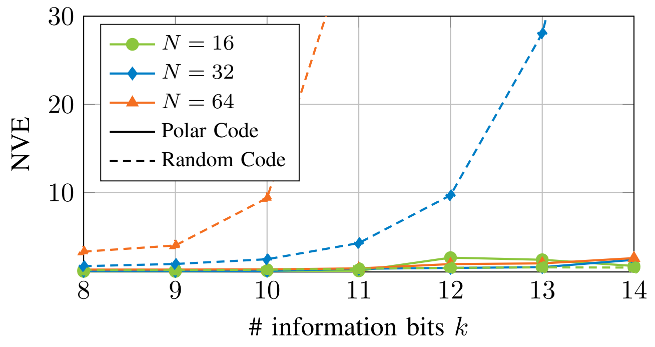


Fig. 6: Scalability shown by NVE for a 1024-512-256 NN for 16/32/64 bit-length codes with different code rates and $M_{\text{ep}} = 2^{16}$ training epochs.

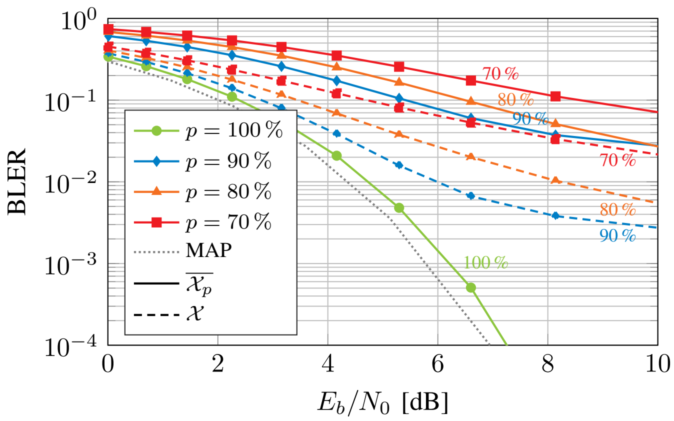
IV. Capability of Generalization

As Fig. 2–6 show, NNDs for polar codes always perform better than random codes for a fixed NN design and number of training epochs. This provides a first indication that structured codes, such as polar codes, are easier to learn than random codes. In order to confirm this hypothesis, we train the NN based on a subset $\mathcal{X}_p$ which covers only p % of the entire set of valid codewords. Then, the NN decoder is evaluated with the set $\overline{\mathcal{X}_p}$ that covers the remaining $100 - p\,\%$ of $\mathcal{X}$. As a benchmark, we evaluate the NN decoder also for the set of all codewords $\mathcal{X}$. Instead of BER as in Fig. 3, we now use the block error rate (BLER) for evaluation (see Fig. 7). This way, we only consider whether an entire codeword is correctly detected or not, exluding side-effects of similarities between codewords which might lead to partially correct decoding. While for polar codes the NN is able to decode codewords that were not seen during training, the NN cannot decode any unseen codeword for random codes. Fig. 8 emphasizes this observation by showing the single-word BLER for the codewords $\mathbf{x}_i \in \overline{\mathcal{X}_{80}}$ which were not used for training. Obviously, the NN fails for almost every unseen random codeword which is plausible. But for a structured code, such as a polar codes, the NN is able to generalize even for unseen codewords. Unfortunately, the NN architecture considered here is not able to achieve MAP performance if it is not trained on the entire codebook. However, finding a network architecture that generalizes best is topic of our current investigations.

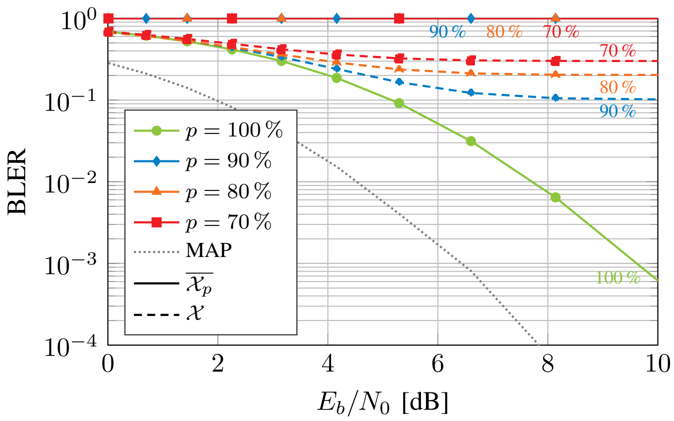
In summary, we can distinguish two forms of generalization. First, as described in Section III, the NN can generalize from input channel values with a certain training SNR to input channel values with arbitrary SNR. Second, the NN is able to generalize from a subset $\mathcal{X}_p$ of codewords to an unseen subset $\overline{\mathcal{X}_p}$. However, we observed that for larger NNs the capability of the second form of generalization vanishes.

V. Outlook and Conclusion

For small block lengths, we achieved to decode random codes as well as polar codes with MAP performance. But learning is limited through exponential complexity as the number of information bits in the codewords increases. The



(a) 16 bit-length Polar Code ($r = 0.5$)



(b) 16 bit-length Random Code ($r = 0.5$)

Fig. 7: BLER for a 128-64-32 NN trained on $\mathcal{X}_p$ with $M_{\text{ep}} = 2^{18}$ learning epochs. Solid and dashed lines show the performance on $\overline{\mathcal{X}_p}$ on $\mathcal{X}$, respectively.

very surprising result is that the NN is able to generalize for structured codes, which gives hope that decoding algorithms can be learned. State-of-the-art polar decoding currently suffers from high decoding complexity, a lack of possible parallelization and, thus, critical decoding latency. NND inherently describes a highly parallelizable structure, enabling one-shot decoding. This renders deep learning-based decoding a promising alternative channel decoding approach as it avoids sequential algorithms. Future investigations will be based on the exploration of regularization techniques as well as recurrent and memory-augmented neural networks, which are known to be Turing complete [23] and have recently shown remarkable performance in algorithm learning.

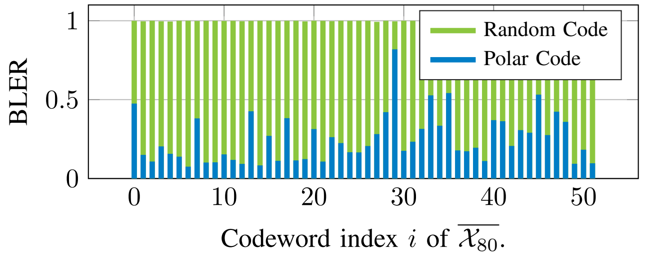


Fig. 8: Single-word BLER for $\mathbf{x}_i \in \overline{\mathcal{X}_{80}}$ at $E_b/N_0 = 4.16\,\text{dB}$ and $M_{\text{ep}} = 2^{18}$ learning epochs.

References

[1] X.-A. Wang and S. B. Wicker, "An artificial neural net Viterbi decoder," *IEEE Trans. Commun.*, vol. 44, no. 2, pp. 165–171, Feb. 1996.

[2] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1." Cambridge, MA, USA: MIT Press, 1986, pp. 318–362.

[4] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec. 1989.

[5] J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Nat. Acad. Sci.*, vol. 79, pp. 2554–2558, 1982.

[6] J. Bruck and M. Blaum, "Neural networks, error-correcting codes, and polynomials over the binary n-cube," *IEEE Trans. Inform. Theory*, vol. 35, no. 5, pp. 976–987, Sept. 1989.

[7] G. Zeng, D. Hush, and N. Ahmed, "An application of neural net in decoding error-correcting codes," in *IEEE Int. Symp. on Circuits and Systems*, vol. 2, May 1989, pp. 782–785.

[8] W. R. Caid and R. W. Means, "Neural network error correcting decoders for block and convolutional codes," in *Proc. IEEE Globecom Conf.*, vol. 2, Dec. 1990, pp. 1028–1031.

[9] A. D. Stefano, O. Mirabella, G. D. Cataldo, and G. Palumbo, "On the use of neural networks for Hamming coding," in *IEEE Int. Symp. on Circuits and Systems*, vol. 3, June 1991, pp. 1601–1604.

[10] L. G. Tallini and P. Cull, "Neural nets for decoding error-correcting codes," in *Proc. IEEE Tech. Applicat. Conf. and Workshops Northcon95*, Oct. 1995, pp. 89–.

[11] A. Hamalainen and J. Henriksson, "A recurrent neural decoder for convolutional codes," in *Proc. IEEE Int. Conf. on Commun. (ICC)*, vol. 2, 1999, pp. 1305–1309.

[12] H. Abdelbaki, E. Gelenbe, and S. E. El-Khamy, "Random neural network decoder for error correcting codes," in *Int. Joint Conf. on Neural Networks*, vol. 5, 1999, pp. 3241–3245.

[13] J.-L. Wu, Y.-H. Tseng, and Y.-M. Huang, "Neural network decoders for linear block codes," *Int. Journ. of Computational Engineering Science*, vol. 3, no. 3, pp. 235–255, 2002.

[14] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, July 2006.

[15] E. Nachmani, Y. Be'ery, and D. Burshtein, "Learning to decode linear codes using deep learning," *CoRR*, 2016. [Online]. Available: http://arxiv.org/abs/1607.04793

[16] I. Goodfellow, Y. Bengio, and A. Courville, "Deep Learning," 2016, book in preparation for MIT Press. [Online]. Available: http://www.deeplearningbook.org

[17] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[18] M. Jaderberg, K. Simonyan, A. Zisserman *et al.*, "Spatial transformer networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2017–2025.

[19] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inform. Theory*, vol. 55, no. 7, pp. 3051 –3073, 2009.

[20] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems 24*. Curran Associates, Inc., 2011, pp. 2546–2554.

[21] D. George and E. A. Huerta, "Deep Neural Networks to Enable Real-time Multimessenger Astrophysics," *ArXiv e-prints*, Dec. 2016.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[23] H. T. Siegelmann and E. D. Sontag, "On the computational power of neural nets," in *Proc. of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 440–449.