Multilevel MIMO Detection with Deep Learning

Vincent Corlay[†,∗], Joseph J. Boutros[‡], Philippe Ciblat[†], and Loïc Brunel[∗]

[†] Telecom ParisTech, 46 Rue Barrault, 75013 Paris, France, Tel. +33 1 4581 7728, v.corlay@fr.merce.mee.com
[‡] Texas A&M University, Doha, Qatar, [∗]Mitsubishi Electric R&D, Rennes, France

Abstract—A quasi-static flat multiple-antenna channel is considered. We show how real multilevel modulation symbols can be detected via deep neural networks. A multi-plateau sigmoid function is introduced. Then, after showing the DNN architecture for detection, we propose a twin-network neural structure. Batch size and training statistics for efficient learning are investigated. Near-Maximum-Likelihood performance with a relatively reasonable number of parameters is achieved.

I. Introduction

In 2012 Alex Krizhevsky and his team presented a revolutionary deep neural network (DNN) in the ImageNet Large Scale Visual Recognition Challenge [11]. The network largely outperformed all the competitors. This event triggered not only a revolution in the field of computer vision but has also affected many different engineering fields, including the field of digital communications. In our specific area of interest, a lot of new studies were published on machine learning for coding and communication theory since 2016.

In our work, we address the case of multilevel symbol detection on multiple-input multiple-output (MIMO) channels via deep neural networks. There exist many algorithms to perform MIMO detection, whose performance ranges from optimal to highly suboptimal. A first category of decoders includes sphere decoding methods based on lattice points enumeration and radius adaptation. The complexity of sphere decoding is clearly less prohibitive than an exhaustive search and is polynomial in the dimension for small dimensions. Detection based on sphere decoding is quasi-optimal and is very competitive in terms of number of operations for dimensions less than 32 (up to 64 for non-dense MIMO lattices), however it cannot be parallelized because of its sequential nature. Furthermore, the dynamic tree structure of sphere decoding makes it hardware-unfriendly.

In a second category we find linear receivers: the zero-forcing (ZF) detector and the minimum mean squared error (MMSE) detector. Finally, a non-exhaustive list of decoders having performance somewhere between these two categories includes: the decision feedback-equalizer (DFE), the K-best sphere decoder, message passing methods (e.g. belief propagation, approximate message passing, expected propagation) and semidefinite relaxation. While some of these algorithms are near-optimal in specific settings, their performance are largely degraded when these specific conditions are not respected. As a result, the problem of finding hardware-friendly low-complexity methods exhibiting near-optimal performance in most settings remains open.

Neural network based implementation could offer new solutions.

MIMO detection with neural networks has already been investigated by several research groups. In [12] [13], the quadratic form of the MIMO channel is used to build the network. In [15] [9] [8] [6] sub-optimal message passing iterative MIMO decoders are improved with the approach introduced in [7] [10]. The main idea of these studies is to unfold the underlying graph used by an iterative algorithm to get improvement via learning. Simulations show that in most cases learning enhances the performance of the considered algorithm. Nonetheless, these results are almost never compared to optimal detection. It is therefore difficult to assess the real efficiency of such an approach. Additionally, most studies consider binary inputs only. In [13], one-hot encoding is used to address the case of non-binary inputs. Unfortunately, the number of output neurons increases significantly with the spectral efficiency making this solution impractical.

II. Problem settings and network used

In this paper we use row convention for vectors and matrices. We consider a symmetric flat quasi-static MIMO channel with n transmit antennas and n receive antennas. Let G be the $n \times n$ matrix representing the channel coefficients. For simplicity, it is assumed that G has real entries. Any complex matrix of size $n/2$ can be trivially transformed into a real matrix of size n. Let $z \in \mathbb{Z}^n$ be the channel input, i.e., z is the uncoded information sequence. The input message yields the output $y \in \mathbb{R}^n$ via the standard flat MIMO channel equation,

$$y = \underbrace{z \cdot G}_{x} + \eta,$$

where η is a Gaussian vector with i.i.d. $\mathcal{N}(0, \frac{N_0}{2})$ components. The optimal decoder, also called Bayes decoder in the machine learning community, implements the maximum a posteriori (MAP) criterion. A near-optimal neural network detector should implement a function f that approximates the MAP criterion.

$$f(y) \approx \arg \max_{z \in \mathcal{M}} P(z|y).$$

In our settings, the MAP criterion is equivalent to finding the closest possible x, closest in the Euclidean sense, as expressed by the following equation:

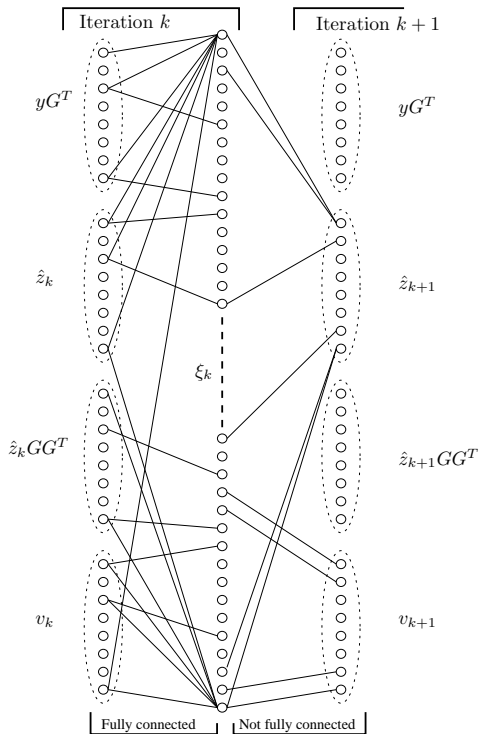$$\hat{z} = \arg \min_{z \in \mathcal{M}} ||y - \underbrace{z.G}_{x}||.$$

Fig. 1. DNN architecture, two layers per iteration.

Fig. 2. $\sigma_c(t)$ for 5-level integer symbols.

Neurons in regular DNN include a non-linear activation function, such as the sigmoid function and the rectified linear unit [4]. In the sequel, the standard sigmoid function $\sigma(t) = 1/(1 + e^{-t})$ is employed.

A. Architecture

In [12], the architecture of the network is inspired from the projected gradient descent:

$$\hat{z}_{i+1} = \prod\left(\hat{z}_i - \eta \cdot \frac{\partial ||y - z.G||^2}{\partial z}|_{z=\hat{z}_i}\right) = \prod\left(\hat{z}_i - 2\eta \cdot yG^T + \eta \cdot \hat{z}_i GG^T\right),$$

where $\prod$ is a projection operator. Our neural network embraces the same paradigm. It takes the form of an iterative algorithm where an estimate of the output is available after each iteration. It is illustrated in Figure 1. A generic iteration has two layers, as shown in the figure, where the network structure is derived from the following matrix equations:

$$\xi_k = \sigma_c\left(W_{1k}^1 \hat{z}_k + W_{1k}^2 yG^T + W_{1k}^3 \hat{z}_k GG^T + W_{1k}^4 v_k + \text{bi}_{1k}\right),$$
$$\hat{z}_{k+1} = \sigma_c\left(W_{2k}\xi_k + \text{bias}_{2k}\right), \quad v_{k+1} = W_{3k}z_k + \text{bias}_{3k}.$$

In the expression of ξ_k, we can clearly recognize the terms used by the gradient descent, weighted by W_1^i instead of η (the two other terms are a hidden variable and a bias term commonly used in neural networks). The intuition behind this expression is that the network will learn specific learning rates η for each iteration and each component. The operation performed between the ξ layer and the next layer can be interpreted as the projection operator $\prod$. The activation function used σ_c is described in the next section.

In [12], the matter of how $\hat{z}_0$ should be initialized for the first iteration of the neural network is not discussed. We
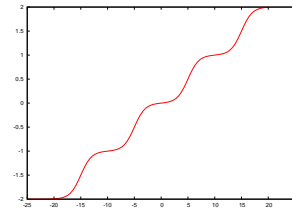
address and take advantage of this question in the section on the twin-network.

B. The multilevel activation function

The default approach to address a multi-class problem with neural networks is to use the so-called "one-hot encoding". Namely, if the network should classify data between more than two categories, say M categories, it will have M output neurons where legal combinations of values are only the M combinations with a single neuron equal to 1 and all the others equal to 0. Unfortunately, this approach implies a large amount of output neurons. In the network of Figure 1, if each component of the input message z can take M levels, using one-hot encoding means having $n \times M$ output neurons (the neurons labeled z_{k+1} in Figure 1) instead of n in the binary case. This implies a greater complexity as well as longer training.

To address this issue we introduce a novel activation function: we adapt the non-linearity in the output neurons to take into account non-binary symbols. Our customized sigmoid function shall be defined as a sum of standard sigmoids,

$$\sigma_c(t) = \sum_{i=1}^{M} \sigma(t - \tau_i) + A,$$

where τ_i are sigmoid shifts and A is an overall translation. As an example, for $z \in \{-2, -1, 0, 1, 2\}^n$ ($M = 5$), the customized sigmoid is taken to be $\sigma_c(t) = \sigma(t + 15) + \sigma(t + 5) + \sigma(t - 5) + \sigma(t - 15) + \sigma(t - 25) - 2$, as depicted on Figure 2.

C. The twin-network

To further improve our system, we considered the paradigm of a random forest [14]: "divide and conquer". With a random forest, many decision trees are trained on a random subset of the training data with a randomly picked subset of dimensions. One decision tree alone tends to highly overfit. But the random forest, based on the aggregation of the trees and a majority decision rule, has very good and consistent results. The important idea is to introduce some randomness between the trees. The concept of a random forest is analogous to extreme pruning successfully utilized by the cryptography community for sphere decoding [3]. They built trees having low success rate and repeated the operation many times with different bases of the lattice. They observed that complexity decreases much faster than the performance deterioration. This successful concept was also known in Ordered Statistics Decoding two decades ago. Therefore, in case of sub-optimality of the network, a solution can be to duplicate the network
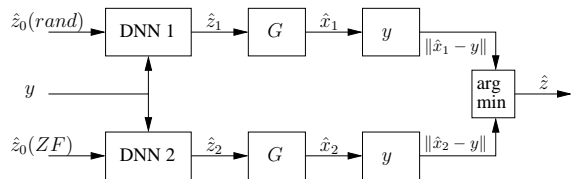
Fig. 3. Block representation of the neural system.

and introduce randomness instead of increasing the number of parameters in the DNN. An easy way to introduce randomness is to initialize neural networks with distinct $\hat{z}_0$ obtained via different manner. An instance of such system is illustrated in Figure 3. The first DNN is initialized with a random $\hat{z}_0$, while the second DNN receives an initial $\hat{z}_0$ obtained by ZF.

III. TRAINING STATISTICS

Only a limited amount of studies discuss what training statistics should be used for efficient training of a neural-based decoder. In [5], they introduce the notion of Normalized Validation Error (NVE) to investigate which SNR is most suited for efficient training. They empirically observed that a SNR neither too high nor too low is the most efficient. In most papers, authors mix noisy data obtained at different SNRs to perform training, in hope that the network is efficient at all those SNRs. To the best of our knowledge, in all papers on neural networks for decoding, the input message z associated to a noisy received signal y is used as label for the training.

Regardless of the noise, the label that should be used for a given y is what would have been decoded by the optimal decoder, not the transmitted sequence. Consider for instance a simple BPSK. If the noise moves a point (e.g. +1) further than the decoding threshold (e.g. -0.2), one should not tell the neural network to try to recover the original point (here +1): it should decode the point associated to the region the received y belongs to (here -1).

Let us call $\mathcal{C}$ a given constellation/code/lattice that we want to train to decode and c_i an element of $\mathcal{C}$. Leaving apart the notion of SNR, the optimal decoder (which we could also call the Voronoi classifier) performs the following operation: given a y (anywhere) in the space of $\mathcal{C}$, it finds the c_i associated to the decoding (Voronoi) region where y is located. Moreover, if we want the network to learn the entire structure of $\mathcal{C}$, the training sample should be composed of points sampled randomly in its space. Equivalently, one can randomly choose elements of $\mathcal{C}$ (with equiprobability) and add uniformly distributed noise.

Nevertheless, to get quasi-maximum-likelihood decoding (MLD) performance on the Gaussian channel, the network doesn't need to learn the entire structure of $\mathcal{C}$ but rather the most relevant decision boundaries *around the* c_i. Indeed, some regions along the boundaries are so far from c_i such that the Gaussian noise almost never sends c_i to those regions. Therefore, a quasi-MLD network can potentially make many simplifications compared to a perfect MLD network and thus reduce its complexity. These simplifications can be learned

by training the network with Gaussian noise.

Unfortunately, getting MLD label can be very costly (especially compared to using the input message z): any sample should be decoded with the optimal decoder and potentially stored. Hence, if we were to use z as label for the training due to limited resources, what SNR should be used on the Gaussian channel? In light of the above discussion, we would want both to learn the necessary structure of the code to get quasi-MLD performance (i.e. the SNR should not be too high) but the "noise" in the label (i.e. messages that are wrongly labeled w.r.t. the optimal decoder) should not be too high either. Empirically, we observed that the SNR corresponding to an error probability of 10^{-2} is a good trade-off (only one sample out of 100 is mis-labeled but the SNR is low enough to properly explore $\mathcal{C}$).

IV. SIMULATION RESULTS

In this section we present neural networks performance observed under several settings. For each of these settings the results we report are the best complexity-performance trade-off we obtained, i.e. we decreased the network size as much as possible while keeping near MLD performance.

For the first set of simulations, depicted in Figure 4, the settings are the following. We take $n = 8$ and $M = 5$ levels on each z_i. The MIMO channel is a static channel randomly sampled from an i.i.d. Gaussian matrix. The considered matrix instance has condition number 17 and Hermite constant -4.7dB (as a real lattice), i.e., this is a bad channel realization and an interesting challenge to our DNN. Additionally, we used the multilevel activation function. The training is done in a regular way with the Adam optimizer and a small batch size (≈ 200). The multilevel MIMO detector used for these simulations has $1.25n$ iterations, ξ is of size $7n$ and v of size n. Hence, the twin-DNN has $2 \times 1.25n \times 42n^2 \approx 100n^3$ parameters (which is about 10 times smaller than 5^8).

We observe that the twin-network DNN performance is close to the MLD performance and clearly outperforms the single DNN (we show only the curve for the randomly initialized single DNN because it matches the one initialized with the ZF point). This means that, under a different initialization, the two single DNNs are almost never wrong at the same time (except for the cases that cannot be recovered by the optimal decoder). Hence, this approach can be beneficial to improve a sub-optimal neural network.

The second set of simulations was performed under the same settings as the one described above, but the batch size is increased to $\approx 3e^4$ to train the network. Moreover, the size of the ξ layer is decreased to $4n$. In Figure 5, we show a significant improvement of performance for the single DNN case: within just three iterations ($<< 1.25n$) and with a decreased network size we manage to get near-MLD performance (even though the number of parameters in the network is decreased to $3 \times 24n^2$). We don't believe that
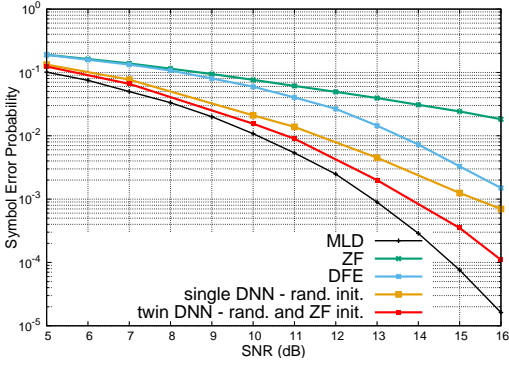
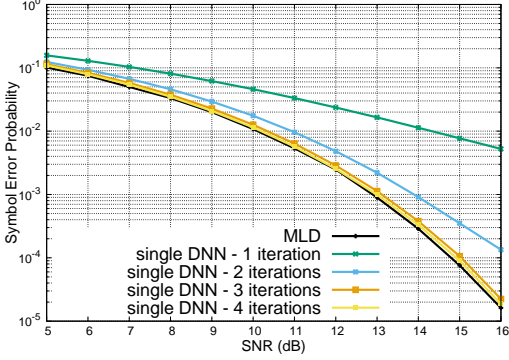Fig. 4. First simulations, with **small batch size** training.

Fig. 6. DNN with soft outputs.

Fig. 5. Second simulations, with **large batch size** ($\approx 3e^4$).

Fig. 7. DNN for the $T55$ MIMO channel.

the improvement is caused by a larger amount of data used to train the network: Firstly, in the small-batch simulations we let the networks learn for a large enough amount of time. Secondly, the convergence to quasi-MLD performance with a large batch size is very fast. We rather believe that a non-noisy gradient is better suited for efficient learning in our settings.

In this work, we also aim at comparing the performance of multilevel activation functions and one-hot encoding. Note that one-hot encoding associated to the soft-max activation function yields soft outputs. Hence, we modify the network used in Figure 5 by replacing each M-level output neuron (i.e. the neurons labeled z_{k+1} in Figure 1) by M neurons to get soft outputs. Moreover, we used 10 iterations. The result obtained is depicted in Figure 6. We observe that we don't manage to get quasi-optimal performance as in Figure 5. Additionally, the training phase of this network took significantly more time than the previous one and required much more fine tuning of hyper-parameters. To summarize, this network is more complex and harder to train.

Finally, we perform a last simulation on the $T55$ MIMO channel used in [12]. The associated matrix is ill-conditioned, which makes it challenging for linear detectors but not necessarily for the sphere decoder. We take $n = 16$, $M = 5$ levels with the multilevel activation function on output neurons. We observe in Figure 7 that this situation is well handled by our neural network.
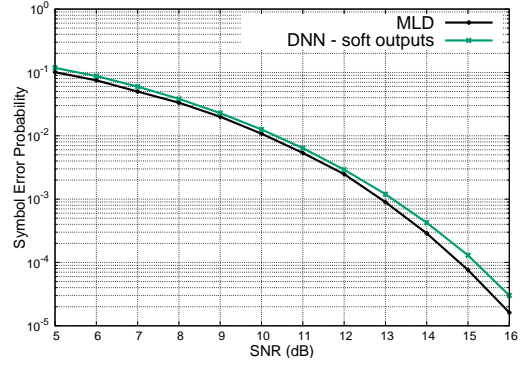
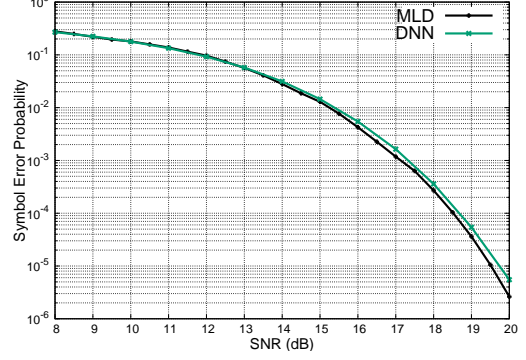The complexity of the different models presented in this section is summarized in Figure 8. We plot the number of parameters (number of edges) of the network as a function of the cardinality of the constellation (obtained as M^n). We also write in blue the complexity of the network used in [12] for the T55 MIMO matrix. We believe that the number of parameters indicated in blue could be diminished without degrading the performance if a larger batch size is used in training.

In light of these results, we may conclude that deep learning, with the proposed approach, is competitive for a large range of MIMO channels. However, deep learning in some extremal situations is difficult to set up, namely for specific channels where the function to be approximated is very challenging. For instance, if the MIMO channel is the generator matrix of a dense lattice (e.g. E_8, BW_{16}, Λ_{24} [1]), the function to
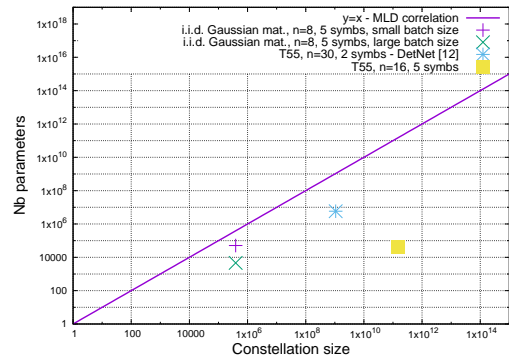

Fig. 8. Complexity analysis of the considered models.

learn is more complex (see next section) and even a neural network with a large number of iterations and an increased size for each layer fails to achieve near-MLD performance, as shown in Figure 9. Fortunately these extremal communication channels are rarely encountered.
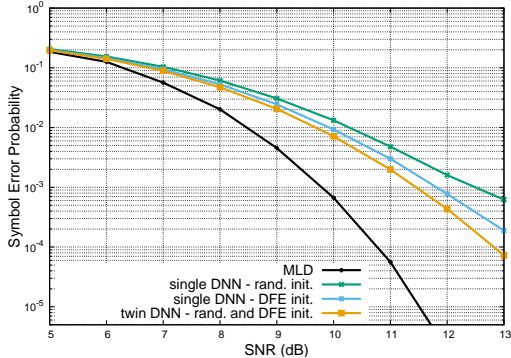

Fig. 9.   MIMO channel is taken to be the generator of BW_{16}

V. CONNECTION WITH (INFINITE) LATTICE DECODING

Lattice modeling of the MIMO channel is not always successful because of the finite number of levels which induces a finite constellation: the MLD point in the lattice can be out of the finite MIMO constellation. With the regular sphere decoder, it is possible to bound the number of states that each component of z can take and overcome this issue. However, if complexity reduction techniques are used as preprocessing, such as basis reduction, then this issue is difficult to avoid. Similarly, the hyperplane logical decoder (HLD) introduced in [2], a neural network based lattice decoder, cannot be used (i.e. leads to disappointing performance) for MIMO detection because it can detect messages which are not in the finite constellation.

In this section, we present a new strategy to avoid this issue while using a lattice-based approach. Namely, we show how the detection can be performed in the fundamental parallelotope $\mathcal{P}$ (see [2]) and still detect only possible messages belonging to the finite alphabet. This leads to both:

- A better understanding of the hardness of the problem that the neural network should solve.
- A new strategy for lattice-based multilevel MIMO detection with neural networks.

We present the approach in four steps. Consider that the first component of z is to be detected.

- Step 1: Go in the fundamental parallelotope $\mathcal{P}$ and consider only the $n-1$ last coordinates.

$$y'^{[n-1]} = y^{[n-1]} \bmod \mathcal{P}^{[n-1]} = y^{[n-1]} - (tG)^{[n-1]},$$

where $t = \lfloor xG^{-1} \rfloor$.

- Step 2: Compute the decision boundary function (in pink on Figure 10): $u' = g(y'^{[n-1]})$.
- Step 3: Go back to the original location.

$$u = u' + \Sigma_i t_i \frac{b_i \cdot e_n}{||e_n||^2},$$
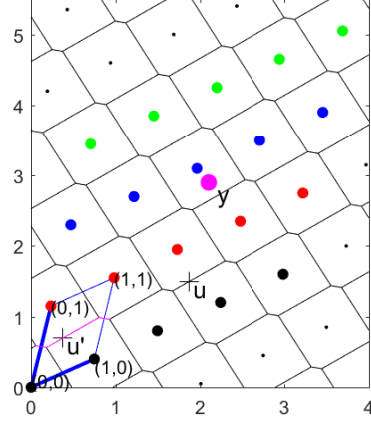

Fig. 10.   Example of lattice-based MIMO detection in $\mathcal{P}$.

where $\{b_i\}$ is the lattice basis and $\{e_i\}$ defines the coordinate system.

- Step 4: Apply the multilevel sigmoid function on $u - y_n$ with delays equal to:

$$\tau = \frac{e_n \cdot b_n}{||e_n||^2}.$$

The main operational cost of this algorithm is due to the decision boundary function. It is closely related to the Boolean equation of the HLD and can be computed with a DNN.

REFERENCES

[1] J. Conway and N. Sloane. *Sphere packings, lattices and groups*. Springer-Verlag, New York, 3rd edition, 1999.
[2] V. Corlay, J.J. Boutros, P. Ciblat, and L. Brunel, "Neural Lattice Decoders," arXiv preprint arXiv:1807.00592, July 2018.
[3] N. Gama, P.Q. Nguyen, and O. Regev, "Lattice Enumeration Using Extreme Pruning," *EUROCRYPT 2010*, vol 6110, Springer, 2010.
[4] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016.
[5] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning-based channel decoding," *Conference on Information Sciences and Systems*, March 2017.
[6] H. He, C. Weny, S. Jin, G. Ye Liz, "A Model-Driven Deep Learning Network for MIMO Detection," arXiv preprint arXiv:1809.09336, Sept. 2018.
[7] J.R. Hershey, J. Le Roux, and F.Weninger, "Deep Unfolding: Model-Based Inspiration of Novel Deep Architectures," arXiv preprint arXiv:1409.2574, Nov. 2014.
[8] M. Imanishi, S. Takabe, T. Wadayama, "Deep Learning-aided iterative detector for massive overloaded MIMO channels," arXiv preprint arXiv:1806.10827, June 2018.
[9] X. Liu and Y. Li, "Deep MIMO Detection Based on Belief Propagation," *IEEE Information Theory Workshop (ITW)*, Guangzhou, China, Nov. 2018.
[10] E. Nachmani, Y. Be'ery and D. Burshtein, "Learning to decode linear codes using deep learning," *54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Monticello, Illinois, pp. 341-346, Sept. 2016.
[11] NIPS'12 Proceedings of the *25th International Conference on Neural Information Processing Systems*, vol. 1, pp. 1097-1105, Dec. 2012.
[12] N. Samuel, T. Diskin, and A. Wiesel, "Deep MIMO detection," arXiv preprint arXiv:1706.01151, June 2017.
[13] N. Samuel, T. Diskin, and A. Wiese, "Learning to Detect," arXiv preprint arXiv:1805.07631, May 2018.
[14] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.
[15] X. Tan, W. Xu, Y. Be'ery, Z. Zhang, X. You, and C. Zhang, "Improving Massive MIMO Belief Propagation Detector with Deep Neural Network," arXiv preprint arXiv:1804.01002, Apr. 2018.