# Deep Neural Network *a Posteriori* Probability Detector for Two-dimensional Magnetic Recording

Jinlu Shen[1], *Student Member, IEEE*, Ahmed Aboutaleb[1], *Student Member, IEEE*, Krishnamoorthy Sivakumar[1], *Senior Member, IEEE*, Benjamin J. Belzer[1], *Member, IEEE*, Kheong Sann Chan[2], *Member, IEEE*, Ashish James[3]

[1]School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164-2752 USA
[2]Nanjing Institute of Technology, Nanjing, China
[3]Institute for Infocomm Research, A*STAR, Singapore

**In two-dimensional magnetic recording (TDMR) channels, intersymbol interference (within and between tracks) and pattern-dependent media noise are impediments to reaching higher areal density. We propose a novel deep neural network (DNN) based a posteriori probability (APP) detection system with parallel multi-track detection for TDMR channels. The proposed DNN-based APP detector replaces the trellis-based Bahl-Cocke-Jelinek-Raviv (BCJR) or Viterbi algorithm and pattern-dependent noise prediction (PDNP) in a typical TDMR scenario, in that it directly outputs log-likelihood ratios of the coded bits and iteratively exchanges them with a subsequent channel decoder to minimize bit error rate (BER). We investigate three DNN architectures - fully connected DNN, convolutional neural network (CNN), and long short-term memory (LSTM) network. The DNN's complexity is limited by employing linear partial response (PR) equalizer pre-processing. The best performing DNN architecture, CNN, is selected for iterative decoding with a channel decoder. Simulation results on a grain-flipping-probability (GFP) media model show that all three DNN architectures yield significant BER reductions over a recently proposed 2D-PDNP system and a previously proposed local area influence probabilistic (LAIP)-BCJR system. On a GFP model with 18 nm track pitch and 11.4 Teragrains/in$^2$, the CNN detection system achieves an information areal density of 3.08 Terabits/in$^2$, i.e. a 21.72% density gain over a standard BCJR-based 1D-PDNP; the CNN-based system also has 3$\times$ the throughput of 1D-PDNP, yet requires only 1/10th the computer run time.**

***Index Terms*—Two-dimensional magnetic recording, deep learning, convolutional neural network, recurrent neural network, long short-term memory, channel detection, grain-flipping-probability model**

## I. Introduction

DUE to a phenomenon called the "superparamagnetic effect," the hard disk drive (HDD) industry is approaching an areal density limit for conventional one-dimensional magnetic recording (1DMR). Two-dimensional magnetic recording (TDMR) has the potential of achieving an order of magnitude increase in HDD capacity without requiring radical redesign of recording media [1]. HDD recording channels have media noise, which results from magnetic grain-bit interactions and is signal dependent. Typical signal processing for conventional 1DMR includes 1D pattern dependent noise prediction (1D-PDNP, [2], [3]), which assists a trellis based Bahl-Cocke-Jelinek-Raviv (BCJR) or Viterbi algorithm (VA) in detecting coded data bits in presence of intersymbol interference (ISI) and media noise, and exchanges log-likelihood ratios (LLRs) with a channel decoder. Proposed generalizations to 2D-PDNP for two-reader TDMR (e.g., [4]–[6]) suffer a state explosion problem, i.e., trellis state cardinality becomes $4^{(\Delta+I+L)}$, where $\Delta$ is the predictor look-ahead, and $I$ and $L$ are the ISI channel length and predictor order. The complexity grows exponentially with $I+L$, and becomes impractical for more than two readers.

To address this issue, in [7] we proposed a three-track detection system for TDMR, in which a local area influence probabilistic (LAIP) *a priori* detector passes its estimates to a BCJR detector to detect coded bits in presence of ISI, intertrack interference (ITI) and media noise. The LAIP detector was first proposed in [8] and further evaluated in [9]. It estimates media noise in a $3\times3$ local area. Simulation results in [7] show the LAIP-BCJR detector achieves significant detector bit error rate (BER) reductions over 2D-PDNP.

The LAIP detector employs a relatively simple machine learning method, i.e., trained conditional probability mass function (PMF) tables. Deep learning techniques [10] employ much more general network structures and training techniques, and have seen great success in a wide variety of applications. For example, convolutional neural networks (CNNs) [11] work well for spatially correlated data, and are highly competitive in image recognition and computer vision. Long short-term memory (LSTM) networks [12], a type of recurrent neural network (RNN), circumvent the vanishing gradient problem in traditional RNNs, and are well suited for sequential data such as those used in speech recognition and machine translation. Traditional fully connected deep neural networks (FC-DNNs), on the other hand, are "structure agnostic", and learn general relationships between input and output without requiring special assumptions about the input. In TDMR, down-track ISI can span around 10 to 20 bits, whereas the media noise term affecting a given target bit is primarily due to local grain-bit interactions that occur in a 2D neighborhood of the target bit. Thus, it makes sense to consider HDD readings as either sequential data or spatially correlated data, and a DNN could outperform the LAIP in detecting coded bits.

Manuscript received Oct 21, 2019. Corresponding author: B. Belzer (email: belzer@wsu.edu)

Two previous papers [13], [14] employ neural networks (NNs) for equalization of TDMR channels. However, these NNs have only three layers, and are thus not DNNs (which typically have ten or more layers including hidden and output layers), but are more like the first generation NNs introduced in the 1980s. In addition, the NNs in [13], [14] do not appear to interface directly with a channel decoder, and their performance is compared only with that of a 2D linear equalizer, which is known to perform significantly worse than the 2D-BCJR or Viterbi equalizers typically employed in TDMR detectors (e.g. [4]–[6], [15]).

In this paper, we propose a novel DNN based *a posteriori* probability (APP) TDMR detection system, in which a DNN detects data pre-processed by a linear PR equalizer, and iteratively exchanges log-likelihood ratios (LLRs) with a channel decoder. Three types of DNN architecture are investigated: FC-DNN, CNN and LSTM. They are evaluated in terms of detector BER. The best performing DNN architecture, CNN, is selected for iterative channel decoding, and the achieved areal density is compared against both the LAIP-BCJR and a conventional 1D-PDNP system. The proposed system is evaluated on a grain-flipping-probability (GFP) magnetic media model, a realistic model which replicates output from micromagnetic simulations [16] but can be generated several orders of magnitude faster. The GFP model has been validated in previous studies against both spin-stand [17], [18] and HDD [19] signals, and an HDD areal density estimate was made in [20].

A DNN-based media noise predictor was recently proposed in [21], in which a DNN replaces the conventional PDNP by supplying media noise estimates to the BCJR. The BCJR performs ISI equalization, and subtracts the estimated media noise when computing its LLRs. By contrast, in the present paper we eliminate the trellis and replace both the BCJR and PDNP by a single DNN that directly estimates the coded bits. Also, [21] considers only single-track detection for 1DMR, whereas the system proposed in this paper detects three tracks simultaneously and is designed for TDMR.

This work's main contributions are as follows: 1) three novel DNN APP three-track detectors based on FC-DNN, CNN, and LSTM that reduce the detector BER by $30.47\%, 32.87\%$, and $28.27\%$ respectively compared to a state-of-art 2D-PDNP detector; 2) use of a linear minimum mean-squared error (MMSE) filter with partial response (PR) signaling as a preprocessing step to limit the DNN complexity; 3) a method of exploiting spatial correlation in the *a priori* LLRs provided to the CNN by the channel decoder to improve iterative decoding performance; 4) a novel DNN training-per-iteration approach for iterative decoding with a channel decoder. On a GFP model with 18 nm track pitch and 11.4 Teragrains/in$^2$, the CNN detection system achieves an information areal density of 3.08 Terabits/in$^2$ (Tb/in$^2$), i.e. a $21.72\%$ density gain over a standard BCJR-based 1D-PDNP; the CNN-based system also has $3\times$ the throughput of 1D-PDNP, yet requires only 1/10th the computer run time.

This paper is organized as follows. Section II discusses the GFP model data used to train and evaluate our system. Section III provides an overview of the proposed DNN-based APP detection system. Section IV describes the three DNN architectures investigated, i.e., FC-DNN, CNN and LSTM, and section V describes how they are trained. Section VI explains in detail the DNN-based TDMR detection system, and section VII presents Monte Carlo simulation results on the GFP model. Section VIII concludes the paper.

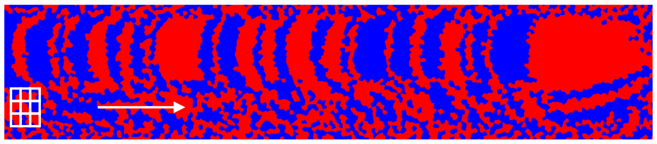
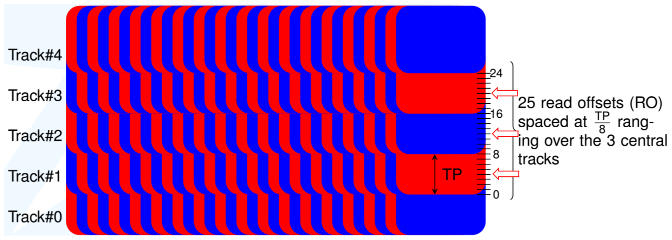


Fig. 1. A capture of the GFP model readback signal with track pitch of 18 nm and bit length of 11 nm. Bit regions are not rectangular but curved stripes due to the shingled write process. The blue and red stripes represent $-1$ and $+1$ coded bits. The $3 \times 3$ white square denotes a $3 \times 3$ convolutional filter applied on the data. The white arrow indicates the direction that the filter moves.


Track#4
Track#3
Track#2
Track#1
Track#0
TP
24
16
8
0
25 read offsets (RO) spaced at $\frac{TP}{8}$ ranging over the 3 central tracks
Fig. 2. Cartoon representation of the writing process and readback sampling position for GFP model data.

## II. The GFP Model

The GFP waveforms are generated based on micro-magnetic simulations. The simulated media has grain density of 11.4 Teragrains per square inch; similar grain densities have been employed in previous papers [22], [23]. The GFP model data used in our system consists of five tracks of coded bits ($\pm1$), denoted as tracks 0 through 4. They are written using shingled writing technology. Fig. 1 shows a capture of the GFP model readback signal. Bit regions are not rectangular, but rather curved stripes due to the relative orientation of the corner write head. The blue and red stripes represent $-1$ and $+1$ coded bits. Track 0 at the bottom is written first. Then track 1 is written, overlapping part of track 0. The writing process repeats until track 4 is written. Track 4 is called the fat track, since it is not followed by any more tracks and thus preserves its original width, i.e., the magnetic write width (MWW). MWW is a characteristic of the write head, and it is equal to 75 nm. In our GFP simulations, bit length (BL) is chosen to be 11 nm, and track pitch (TP), i.e., the distance between adjacent tracks, is set to be 18 nm. The number of grains per coded bit (GPB) is calculated as

$$\begin{aligned}\text{GPB} &= \text{Grain density} \times \text{BL} \times \text{TP}\\ &= 11.4 \text{ Teragrains/in}^2 \times 11\text{nm} \times 18\text{nm}\\ &\quad \times (3.937 \times 10^{-8} \text{ in/nm})^2 = 3.5.\end{aligned}$$

Fig. 2 is a cartoon representation of the writing process and readback sampling position. The input bits in the GFP data are arranged in tracks 0 through 4, and are of size $5 \times N_b$, where $N_b$ is the number of bits per track, or the track length. Their values are randomly distributed and known. Of these

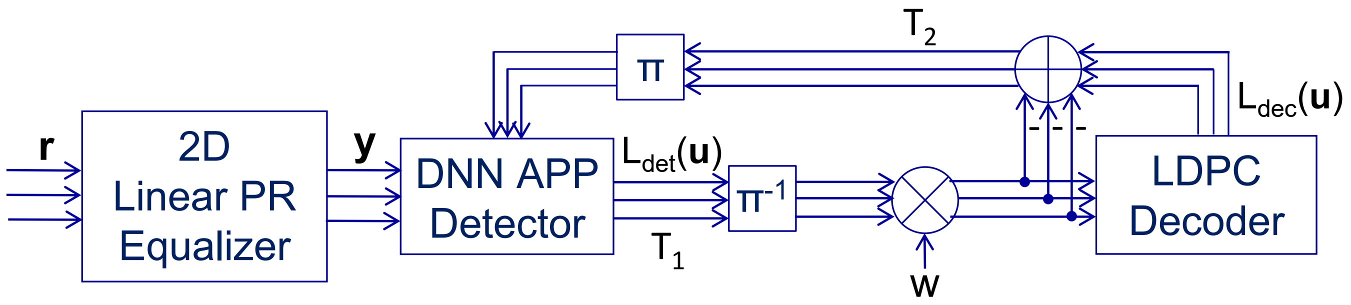


Fig. 3. Block diagram of the DNN-based APP detection system for TDMR.

TABLE I
NORMALIZED $3 \times 3$ CONVOLUTIONAL MASK FOR ESTIMATING 2D-ISI

| GFP data set #1 | GFP data set #2 |
|---|---|
| 0.1163 0.2725 0.0462 | 0.0967 0.1742 0.0598 |
| 0.3087 1.0000 0.5264 | 0.3857 1.0000 0.5596 |
| 0.0959 0.4194 0.3628 | 0.0637 0.2397 0.2757 |

five tracks, only tracks 1, 2, and 3 have readback values, and thus the readback values are of size $3 \times N_b$. There are 25 read offsets spaced at TP/8 ranging over tracks 1, 2, and 3. To sample the data on these three central tracks, we use reader positions at 4, 12 and 20, approximately the center of each track.

We generate two sets of GFP data. They have the same BL and TP, but different reader and writer parameters during the GFP simulation. Each track in GFP data set #1 consists of 41,206 coded bits, and there are 41,207 coded bits in GFP data set #2; this is close to the sector size of 32,768 bits (4K bytes) in a typical HDD. The bits written on the two outer tracks have no readings available, and are considered boundary bits. For GFP data set #1, the three-track readings have raw BERs of roughly 18.54%, 18.53% and 18.33% under a hard decision with threshold 0. For GFP data set #2, the raw BERs are 16.38%, 16.41% and 16.32%. Table I shows the best fit normalized $3 \times 3$ ISI convolutional masks for the two GFP data sets. These ISI masks are estimates of the 2D channel response used to simulate these two data sets, and are estimated using least squares method as in [9]. It can be seen that the simulated channel for these two data sets are significantly different. GFP data set #1 clearly suffers more ITI from both adjacent tracks than GFP data set #2.

The shingled writing process introduces ITI. As the MWW is fixed, a smaller TP results in greater ITI. Compared to typical commercial HDDs with TP = 48nm, the GFP model data with TP = 18 nm used in this paper suffers from rather severe ITI. In our system, a 2D linear PR equalizer is applied on the GFP waveforms prior to the DNN. This pre-processing step partially equalizes the ITI and the down-track ISI, and as a result it helps lower the complexity of the DNN.

## III. DNN Detection System Model and Overview

Fig. 3 shows a block diagram of the proposed DNN-based APP detection system. The system input, consisting of three-track GFP readings, is first pre-processed by a 2D 3-input-3-output linear PR equalizer. The linear equalizer $\mathbf{h}$ (of size $3 \times 15$) is applied on the raw GFP readings $\mathbf{r}$ in order to minimize the MSE between the filtered output $\mathbf{h}*\mathbf{r}$ and desired output $\mathbf{g}*\mathbf{u}$, where $\mathbf{g}$ is a 2D $3 \times 3$ controlled partial response (PR) target response, $\mathbf{u}$ is the block of three-track coded data bits, and $*$ indicates discrete 2D convolution. The linear PR equalizer output $\mathbf{y}$ serves as input to the DNN. The detection system assumes the following discrete time channel model for the readback signal $r_k$:

$$r_k = (\mathbf{h_c} * \mathbf{u})_k + n_{e,k}, \tag{1}$$

where $\mathbf{h_c}$ is the channel response, and $n_{e,k}$ represents AWGN from reader electronics. The channel response $\mathbf{h_c}$ is implicitly time varying and pattern dependent, because the channel is inherently nonlinear [24]. Therefore, pattern dependent media noise arises. The filtered readings $\mathbf{y}$ still retain the effects of pattern dependent media noise, which the linear equalizer $\mathbf{h}$ can not really remove. Unlike a BCJR-PDNP detector that predicts media noise using a linear auto-regressive model and performs trellis-based detection, the DNN-based APP detector directly learns a general model to predict coded bits $\mathbf{u}$ through an offline training process, i.e.

$$P(\hat{u}_k = 1) = \mathcal{T}(\mathbf{y}_k). \tag{2}$$

Here $\mathcal{T}$ is a nonlinear transformation, $\mathbf{y}_k$ is a $3 \times 15$ patch of filtered readings and $\hat{u}_k$ is the DNN's estimate of the center bit of the patch $\mathbf{y}_k$. The binary GFP data bits $\mathbf{u}$ ($\pm 1$) are the target bits that the DNN aims to output; they are available to the DNN as true labels during an initial offline training. As a result, the DNN detection process is essentially an instance of binary classification under supervised learning.

The DNN consists of several layers. Interconnections between the layers are defined by functions. Parameters that specify these functions in the DNN include weights, biases, offsets and scales. Among these parameters, some are specified prior to training; they are called hyperparameters. The rest of the parameters are learnable through training; the goal of training is to optimize the learnables so that they provide an accurate description of the input-output relationship between the bottom (input) layer and the top (output) layer of the DNN. In our simulations, the goal is to arrive at optimized learnables that yield the lowest detector BER, or highest code rate and areal density.

Since the equalizer output target $\mathbf{g} * \mathbf{u}$ is multi-level for each binary target bit, the DNN learns through training how to detect the binary target bit from the equalizer output. This replaces the typical trellis processing done in the BCJR or VA with more general DNN processing.

Each patch of filtered readings that the DNN uses to estimate one coded bit $u$ is considered as one example, and it corresponds to one label. In our system, one example consists of three tracks of filtered readings of length 15 (same length as the linear PR equalizer), and the label is the true value of the coded bit $u$ in the center of the $3\times15$ example patch. After one example is formed, we move down-track by one bit and form the next $3\times15$ example. The total number of examples per track $N$ is thus $N=N_b-14$.

An objective function measures the error between the true label $u$ and its DNN estimate $\hat{u}$. We choose the cross entropy loss (described in section IV-A) as the objective function. The DNN's goal is to minimize the objective function. This is done through iterative gradient-based optimization during the training. At each training step, the DNN computes the gradient with respect to each of the learnables over the training data set, and updates them in the direction of descending gradient. The optimized learnables, along with the DNN structure, are stored at the end of the training. In real-time detection, the stored values are pre-loaded into the network and used to make predictions on a previously unseen test data set. Both training and test data in our system are generated using the same GFP model parameters. This is justified by the fact that the read and write head as well as recording media are fixed in a specific set of HDDs.

Given a block of filtered waveform inputs, the DNN outputs APPs of each coded bit being $\pm1$. These APPs are fed into a soft-input-soft-output channel decoder following the DNN. The channel decoder is an irregular repeat accumulate (IRA) low density parity check (LDPC) decoder [25]. Because the GFP data is randomly distributed, coset decoding is employed by the IRA decoder. The three tracks are processed independently by the IRA decoder, assuming each track contains a separate codeword. The DNN detector and the IRA decoder exchange LLRs; the LLR magnitudes are capped at thresholds $T_1$ and $T_2$ at the outputs of the detector and the decoder. A multiplicative weight $w$ is applied to the LLRs passed by the DNN to the IRA decoder after thresholding in order to slow the convergence of the system and thereby avoid local minima in the channel decoder's BER.

We investigate three DNN architectures, i.e., FC-DNN, CNN and LSTM. For each DNN architecture, three networks with the same structure are trained in parallel to estimate tracks 1, 2 and 3. Details of the DNN architectures are described in section IV. We evaluate the performance of the three architectures on the test data set based on DNN output BERs. The best-performing DNN architecture is selected for iterative decoding with the channel decoder. We explore iterative decoding with two decoding passes. In the second decoding pass, the channel decoder passes its soft estimates of the coded bits back to the DNN, and a separate DNN is trained for the second iteration. Details of the iterative decoding are described in section VI.

## IV. DNN ARCHITECTURES

In this section, we describe three DNN architectures implemented for TDMR APP detection: FC-DNN, CNN and LSTM. Because the FC-DNN and the LSTM network do not yield the lowest detector BERs, they are not included in subsequent turbo detection with the channel decoder. The CNN is selected for iterative decoding, and the different CNN structures in both decoding passes are described below in section IV-B.

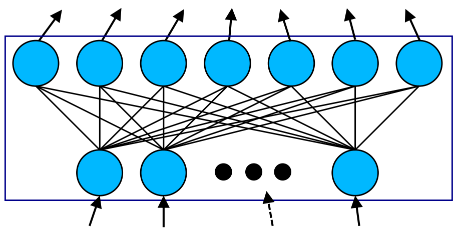


Fig. 4. Interconnections between two fully connected layers. Each node in one layer is connected to every node in an immediately adjacent layer.

In each of the three types of DNN, we group the layers and categorize them into three functional stages: input stage, hidden stage and output stage. The input stage typically consists of an input layer. To account for ITI and the correlation introduced by the linear PR equalizer, the network input consists of three-track filtered readings of size $3\times15$, shaped in certain dimensions that are tailored to each of the three network architectures. The true label for each example is the bit value at the center of the $3\times15$ patch. When estimating tracks 1 and 3, we use boundary track bits (tracks 0 and 4) to form the three-track external input.

The hidden stages are the main stages performing the function of the network. The function of the hidden stages depends on the DNN architecture. For instance, as shown in Fig. 5, the hidden stages in the CNN apply convolutional filter banks to the input. The output stage generates the probabilities that each bit belongs to each of the two classes, and computes the network loss. It is made up of several layers that work jointly, and is identical in all the three architectures.

### A. *Fully connected neural network*

We first describe a traditional FC-DNN for APP detection. It consists of an input stage, four stacks of fully connected stages (fully connected stages #1 through 4), and an output stage, for a total of 12 layers.

In fully connected networks, all the nodes in each fully connected layer are connected to every node in the immediately previous layer. These connections are specified by weights, which are optimized during network training and can be zero. Fig. 4 illustrates such connection between two fully connected layers. Because layers are fully connected, to form each training example, we vectorize a $3\times15$ window of filtered readings into a $45\times1$ column vector. The ensemble of all the column vectors comprises the input stage.

In each fully connected stage, a fully connected layer with a number of hidden nodes is followed by a ReLU layer for network activation. Each node in the fully connected layer applies the affine function $y=w\cdot x+b$ to its input $x$, where $w,b$ are trainable parameters. The weights and biases between each fully connected layers are the learnables to train and optimize in the FC-DNN. The ReLU layer utilizes the ReLU function $f(x)=\max(0,x)$, a popular nonlinear activation function that generates sparsity, is easy to compute,

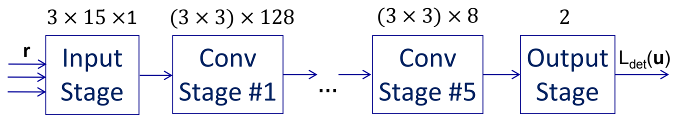


Fig. 5. Functional block diagram of the CNN for the first decoding pass. The CNN is structured in a series of stages, including an input stage, several stacks of convolutional stages, and an output stage. In the first decoding pass, the size of each input example at the input stage is $[3 \times 15 \times 1]$, and there are five convolutional stages, i.e., convolutional stages #1 through 5, with 128, 64, 32, 16 and 8 convolutional filter banks respectively.

and combats the vanishing gradient problem that occurs with the sigmoid activation function [10].

The number of hidden nodes in each hidden layer of the four fully connected stages is $128 - 64 - 32 - 8$. We find in our experiments that a decreasing number of nodes gives best results. This is probably because we have the most information available at the beginning of the network. We also find that given the fixed input size $(45 \times 1)$, further increase in the depth of the DNN does not improve the network performance for a relatively short training time (roughly 30 min). Potential benefits could result from a deeper network for a larger input size and enough training time.

The output stage is made up of a fully connected layer with two hidden nodes, a softmax layer and a classification layer. The fully connected layer combines all the learned features to make a classification. Its output size is equal to the number of classes of the data set, $K$, and $K = 2$. Following it is the softmax layer, which applies the following softmax function to the output $x_k$ of the fully connected layer:

$$p_k = \exp(x_k)/(\sum_{j=1}^{K} \exp(x_j)), k = 1, 2. \tag{3}$$

This softmax function produces a probability distribution $p_k$ over the $K$ output classes. This is considered as the soft information formed by the DNN detector, where $p_1$ is the probability that the bit is 0, and $p_2$ is the probability that the bit is 1. The last layer in the output stage is the classification layer, which computes the cross entropy loss as

$$J = \sum_{i=1}^{N_{mb}} \sum_{k=1}^{K} 1(\hat{u}(i) = k) \times \ln(p_{ik}), \tag{4}$$

where $N_{mb}$ denotes the number of training examples in the mini batch that we compute cross entropy over, $1(\cdot)$ is the indicator function that is turned on when the expression inside the parentheses is true, and $p_{ik}$ is the probability from the softmax layer that the $i^{th}$ example belongs to class $k$. The cross-entropy loss is the most common loss function for binary classification problems. Minimizing this objective function yields accurate and reliable classification [26].

## *B. Convolutional neural networks*

We now describe a convolutional neural network for APP detection. Fig. 5 shows the structure of the CNN for the first decoding pass. It can be categorized into an input stage, five stacks of convolutional stages and an output stage. The input stage consists of an image input layer. Each convolutional stage consists of three layers, to be described below. The

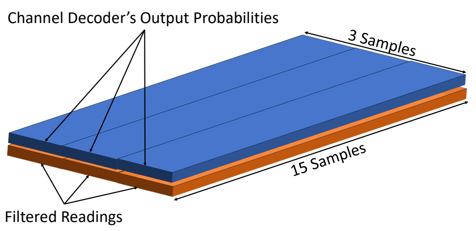


Fig. 6. Image input in one training example in second decoding pass. It is a $[3 \times 15 \times 2]$ array of two $[3 \times 15 \times 1]$ images. The first image, shown in orange at the bottom, is the filtered waveforms, and the second image, shown in blue on the top, is the *a priori* information from the IRA decoder.

output stage is identical to that of the FC-DNN, and consists of three layers, thereby giving a total of $1 + (5 \times 3) + 3 = 19$ layers. External input to the network is required to be in the form of a multidimensional array in order to be considered as an image. In the first decoding pass, the size of the image input layer is $[3 \times 15 \times 1]$ and the overall size of the entire input is $[3 \times 15 \times 1 \times N]$. In the second decoding pass, *a priori* information from the channel decoder becomes available, i.e., for each coded bit there exists an extrinsic soft estimate from the channel decoder. Therefore, we form the input layer size as $[3 \times 15 \times 2]$, and store the channel decoder's estimates as a second image stacked on top of the first $[3 \times 15 \times 1]$ image of filtered readings. This is illustrated in Fig. 6.

Following the input stage are several stacks of convolutional stages. There are five of them in the first decoding pass, and six of them in the second decoding pass. Fig. 7 shows the structure of a convolutional stage. It consists of a 2D convolutional layer, a batch normalization layer, and a rectified linear unit (ReLU) layer. The 2D convolutional layer applies several sliding 2D convolutional filter banks, or kernels, of size $3 \times 3$ to its layer input, as shown by the white $3 \times 3$ square and arrow in Fig. 1. Zero padding is applied to both horizontal and vertical input borders so that the output of the convolutional layer has the same size as its input. This filtering process can be regarded as autonomous feature extraction; each convolutional filter bank corresponds to one feature map. Applying convolutional filters to generate features greatly reduces the efforts for manual feature selection in traditional machine learning, and also provides translation invariance and parameter sharing. For our system, the network learns through this feature extraction process to account for the signal-dependent media noise due to the underlying grain model.

In the first decoding pass, we choose the number of filter banks at convolutional stages #1 through 5 to be $128 - 64 - 32 - 16 - 8$. In the second decoding pass, an extra convolutional stage #6 with 4 filter banks is added after convolutional stage #5. Because the network input in our system only consists of three rows, the rationale for decreasing the number by a factor of 2 at each stage is to enable the CNN to extract more salient features from the input stage, and then gradually abstract only the part that is relevant to the output. The per stage filter bank numbers are experimentally found to work well for our system in terms of classification accuracy, while keeping the overall network complexity reasonable. The fact that the second decoding pass is able to leverage more convolutional stages is probably due to the doubled input size and thus more

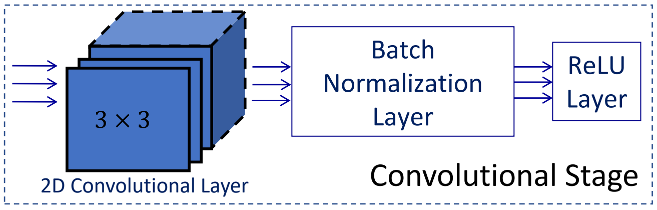


Fig. 7. The structure of a convolutional stage. It consists of a 2D convolutional layer, a batch normalization layer, and a rectified linear unit (ReLU) layer. Each convolutional layer includes a certain number of $3 \times 3$ 2D convolutional filter banks, or kernels. Each filter bank is applied to its input in a sliding block manner.

information.

A batch normalization layer follows the convolutional layer to normalize the convolutional layer's output across a mini-batch, i.e., $\hat{x}_i = (x_i - \mu_B)/\sqrt{\sigma_B^2 + \epsilon}$, $y_i = \gamma \hat{x}_i + \beta$, where $\mu_B$ denotes the mini-batch mean, $\sigma_B$ denotes the mini-batch standard deviation, $\epsilon$ is a small denominator offset for numerical stability, and $\beta$, $\gamma$ are learnable offset and scale factors that are optimized during the training. This normalization process speeds up the training, reduces the network's sensitivity to initialization and increases the network's stability. As in a fully connected network, the last layer component in a convolutional stage is the ReLU layer, which activates the network. Neither the batch normalization layer nor the ReLU layer change the size of its input. The output size of each convolutional stage is $3 \times 15 \times N_f$, where $N_f$ denotes the number of filter banks at that convolutional stage. Based on our choice of $N_f$'s, the dimensionality of the output at each convolutional stage is rather low. Thus, no max pooling layer is employed for downsampling. The last functional stage, i.e., the output stage, of the CNN is identical to that of the FC-DNN, described in section IV-A. Overall, the learnables of the CNN include the coefficients of the filter banks in the convolutional layers, the parameters in the batch normalization layers, and the weights and biases in the fully connected layers in the output stage.

## C. *Long short-term memory*

We describe another DNN APP detector, based on LSTM. The LSTM consists of an input stage, seven stacks of LSTM stages (LSTM stages #1 through 7), and an output stage, with a total of 11 layers. The input stage for LSTM consists of a sequence input layer, with sequence size $N_s = 45$. Thus the input to LSTM is the same as the input to the FC-DNN. The input is interpreted as $N_s$ time steps in a sequence. Each subsequent LSTM stage consists of a bidirectional LSTM (BLSTM) layer, which is a hybrid version of LSTM and bidirectional RNN (BRNN). Bidirectional dependencies can be learned because the network has access to the complete sequence at each time step. Each BLSTM layer is made up of $N_s$ repeating LSTM block cells, corresponding to the $N_s$ time steps. At each BLSTM layer, information passes through each LSTM block cell sequentially. Fig. 8 shows the data flow inside a typical LSTM block cell at time step $t$, i.e., the $t^{\text{th}}$ LSTM block cell.

LSTM layers derive inter-time step relationships mainly through three type of gates—input gate, denoted as $i$ in Fig. 8, forget gate, denoted as $f$, and output gate, denoted as $o$. At

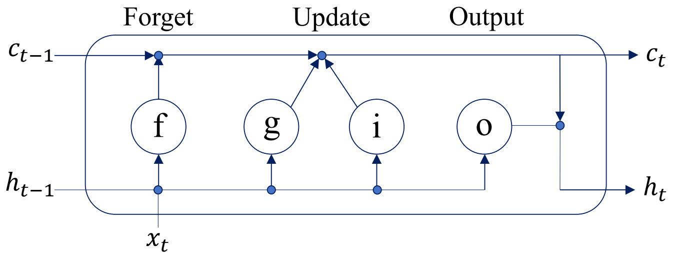


Fig. 8. Data flow inside the LSTM block cell at time step $t$. The LSTM block cell consists of a forget gate $f$, an input gate $i$, a cell candidate $g$ and an output gate $o$. The cell state and hidden state at time step $t$ are denoted as $c_t, h_t$ respectively. The output $h_t$ of the LSTM block cell at time step $t$ becomes input to the LSTM block cell at time step $t+1$.

a particular time step, the input gate allows the new input to pass through, the forget gate discards irrelevant information and the output gate allows the output to be updated. These three gates control the states at each time step, including the hidden state $h_t$, and the cell state $c_t$. The hidden state $h_t$ in the right corner of Fig. 8 contains the output of the LSTM block cell at time step $t$. It stores information from previous time steps, which can be longer than the sequence length 45, and thus the name long short-term memory. The dimension of the hidden state $h_t$ is chosen as 100 for all of the seven BLSTM layers in the seven LSTM stages. The cell state $c_t$ is where memory from previous time steps is taken into consideration. At each time step, information is either added to (through input gate $i$) or removed from (through forget gate $f$) $c_t$. The hyperbolic tangent function (tanh) is used as state activation function to update $c_t$ and $h_t$, and the sigmoid function $\sigma(x) = 1/(1 + \exp(-x))$ is used as the gate activation function to update the gates. LSTM stages #1 through 6 output the complete sequence (of length 45); LSTM stage #7 outputs the last time step of the sequence. The output stage of the LSTM is identical to that of the FC-DNN and the CNN, described in section IV-A. The learnables of the LSTM include the weights in the BLSTM layers, and the weights and biases in the fully connected layers in the output stage.

## V. DNN TRAINING

For each of the three architectures, three DNNs with the same structure are trained independently to estimate each of the three tracks. Weights are initialized to be zero-mean Gaussian with a standard deviation of 0.01. Biases and offsets are initialized to zero. Scales are initialized to one. We employ the adaptive moment estimation (Adam) optimizer [27]. As its name suggests, Adam computes individual adaptive learning rates for different parameters from estimates of the first and second moments $m_l$ and $v_l$ of the gradient at the $l$th step. Compared to standard stochastic gradient descent (SGD), Adam is known to help escape local saddle points and accelerate the training in the relevant direction. In our experiments, Adam achieves a high accuracy much faster than SGD. The set of gradient update rules for each individual parameter for Adam is summarized as

$$\begin{aligned} m_l &= \beta_1 m_{l-1} + (1-\beta_1)\nabla E[\theta_l] \\ v_l &= \beta_2 v_{l-1} + (1-\beta_2)(\nabla E[\theta_l])^2 \\ \theta_{l+1} &= \theta_l - \frac{\alpha m_l}{\sqrt{v_l}+\epsilon}, \end{aligned}$$

where $\theta_l$ denotes the parameter value at step $l$, $\beta_1$, $\beta_2$ are the gradient decay factor and the squared gradient decay factor, $\alpha$ is the learning rate, and $\epsilon$ is a small offset. Values for hyperparameters $\beta_1, \beta_2, \alpha, \epsilon$ are specified prior to training; in our experiments for all DNN architectures, we set $\beta_1 = 0.95$, $\beta_2 = 0.99$ and $\epsilon = 10^{-8}$. For learning rate $\alpha$, we make it piece-wise decaying with an initial value of 0.05. In CNN and FC-DNN, $\alpha$ drops by a factor of 0.75 every epoch, i.e., one pass of the entire training data through the network. In LSTM, for track 2, $\alpha$ drops by a factor of 0.75 every epoch, whereas for tracks 1 and 3 it drops by the same factor every two epochs. Such learning rate scheduling is experimentally found to yield best results.

In our experiments, we have $N \approx 3\times10^6$ training examples. It is computationally inefficient to pass all of them to the DNN at once and compute the gradient. Therefore, we divide the training data into mini batches of size $N_{mb} = 10^4$, and pass one mini batch to the DNN at a time for computing the gradient. One such pass of a mini batch is called an iteration. The total number of iterations in an epoch thus equals $N/10^4$, which is roughly 290 in our simulation. The choice of mini batch size is experimentally found to work well with our system, based on a trade-off between computational efficiency and training accuracy (low BER). We train the DNN over a certain number of epochs, $N_e$. $N_e$ is roughly the smallest number of epochs we find that is required for the network to converge. For the FC-DNN and CNN, $N_e = 15$ for all three tracks. For LSTM, $N_e = 10$.

We also pass to the network a validation data set, which is used for evaluating the network as the training moves forward. Such evaluation, or validation, gives us an idea of how the trained network generalizes to previously unseen data. Both the training data and validation data are randomly shuffled prior to the beginning of every epoch in order to reduce the effect of noise and generalize the learning.

For training a CNN for use with iterative decoding, we explore the possibility of early stopping based on ongoing validation results. Because the network input contains noise, both the training and validation accuracy fluctuate when evaluated, say every 50 iterations. Based on the level of fluctuation we observed, when training the CNN for the second decoding pass, we perform validation every epoch and stop training when validation loss does not decrease after three consecutive evaluations, i.e., after three epochs. We observe in our experiments that the CNN training for tracks 1, 2 and 3 in the second decoding pass auto-stopped after 11, 3 and 2 epochs. In other words, track 1 requires the most number of epochs to converge, whereas track 3 requires the least. This probably results from the shingled writing process, described in section II. As shown in Fig. 1, both tracks 1 and 3 are next to a boundary track. Track 3 abuts the fat track, track 4, whereas track 1 adjoins a fellow narrow track, track 0. In our system, the coded bits on tracks 0 and 4 are known boundary bits and are passed to the DNN. Because track 4 is the fat track and has a higher SNR, it may have provided better help to the DNN in estimating track 3 than track 0 could have helped in estimating track 1.

## VI. TDMR TURBO DETECTION SYSTEM

Fig. 3 shows the three-input-three-output system block diagram. In the linear PR equalizer, the PR mask and the filter coefficients are co-designed using the monic constraint according to the method described in [28]; the upper and lower track outputs are produced using bits on the two boundary tracks. For the two GFP data sets, the two designed PR masks are

$$\mathbf{g} = \begin{bmatrix} 0.0028 & 0.1623 & 0.1417 \\ 0.2795 & 1.0000 & 0.2903 \\ 0.2347 & 0.2684 & 0.0780 \end{bmatrix}$$

for GFP data set #1, and

$$\mathbf{g} = \begin{bmatrix} 0.0080 & 0.0780 & 0.1097 \\ 0.2635 & 1.0000 & 0.2768 \\ 0.1965 & 0.1275 & 0.0267 \end{bmatrix}$$

for GFP data set #2. This pre-processing step equalizes cross-track ITI and down-track ISI to a window size approximately the same as the PR equalizer, i.e., $3 \times 15$, which helps reduce the complexity of the subsequent DNN. The $3\times3$ PR target for the equalizer also enables the CNN to better capture the local features or correlations in the equalized readings, because the size of the filters used in all convolutional layers are $3 \times 3$. The larger coefficients on the first and third row of the PR mask for GFP data set # 1 imply higher ITI in GFP data set #1 than GFP data set #2.

Since the CNN yields the lowest BER among the three DNN architectures (per Table II in section VII), we choose the CNN detector for iterative (turbo) decoding with the IRA decoder. The CNN detector receives the filtered waveforms from the 2D linear PR equalizer and outputs soft estimates for all three tracks. These soft estimates are converted to LLRs, de-interleaved per track, and then fed into the channel decoder. This simulates the scenario that HDD data are first encoded and then interleaved. The purpose of interleaving and de-interleaving is to break correlation among the errors introduced by the detector, so that the errors appear random to the channel decoder. Accordingly, the channel decoder's output LLRs are first interleaved and then sent to the CNN detector as *a priori* information for use in the next decoding pass.

For the second decoding pass, we train and optimize a separate CNN tailored to the *a priori* information from the channel decoder. This piece of *a priori* information is shaped as part of the image input in a similar manner as the filtered waveforms. The input in the second iteration is a $[3 \times 15 \times 2]$ array of two $[3 \times 15 \times 1]$ images. The first image is the filtered waveforms, and the second image is the *a priori* information from the IRA decoder. In our system, what is passed to the CNN from the IRA decoder is actually a linearly shifted version of the IRA's estimated probabilities of each coded bit. We experimentally find that passing probabilities yields better performance than passing LLRs. This is most likely because the log function in LLRs is nonlinear, and as a result introduces distortion. Because probabilities are between 0 and 1, we subtract 0.5 from them to make them zero mean before passing to the CNN. The CNN in the second iteration further lowers the BER, and passes its output LLRs to the IRA decoder again. The IRA decoder produces the final decoded codeword at the end of the second iteration.

LLRs from the CNN to the channel decoder are multiplied by a weight factor $w < 1$. This reduces the CNN output LLR magnitudes, which otherwise tend to overestimate the CNN bit reliabilities. In addition, the CNN's processing (including the non-linear ReLU function) makes it difficult to express the CNN's output LLRs as a sum of extrinsic and *a-priori* terms. Therefore, subtraction of the CNN's input LLRs (provided by the channel decoder) from its output LLRs in order to form extrinsic information to pass to the channel decoder is not possible. We find experimentally that $w = 0.5$ results in lowest decoded BER. Thresholds are also applied on LLRs during turbo iteration to avoid numerical issues.

## VII. Simulation Results

This section presents Monte Carlo simulation results of the DNN-based APP detection system. The system is tested on two GFP data sets, both with TP = 18 nm, BL = 11 nm, and GPB = 3.5. The binary input bits in each block of these data sets are of size $5 \times N_b$, and the waveforms are $3 \times N_b$, where the track length $N_b = 41,207$ for GFP data set #1, and $N_b = 41,206$ for GFP data set #2. GFP data set #1 corresponds to the special sets of 512 training patterns that were used to train the conditional PMFs for the LAIP [9]. GFP data set #2 is identical to one tested in [29], where simulation results in Fig. 16 of that paper show that an information density of 2.4 Tb/in$^2$ (corresponding to 0.2105 U/G) can be achieved with this data set by employing a 3-input/1-output 2D linear PR equalizer with a 1D BCJR and IRA decoder. In both GFP data sets, the boundary bits on the outer two input tracks are known. A total of a hundred blocks of input and waveform data are available. We assign seventy blocks as the training data set, ten blocks as the validation data set, and twenty blocks as the test data set. Both the training data set and the validation data set are available to the DNN during training. Simulation results are reported only for the test data set.

### A. Detector-only BER Comparison

We first evaluate the three DNN architectures implemented in the first decoding pass in terms of detector-only BER comparison. GFP data set #1 is used for this evaluation. Fig. 9 shows a portion of the training learning curves of the three DNN architectures on track 2. Accuracy is equal to $1-\text{BER}$, which means the percentage of bits that are classified correctly. More than 90% accuracy is achieved within two epochs. During training, the FC-DNN classifies the fastest among the three, followed by LSTM. The CNN initially makes the slowest classification but eventually achieves the highest accuracy. It is possible that the initial convergence of the CNN can be improved by fine tuning the hyperparameters in the optimizer used in the training process. The learning curves for tracks 1 and 3 are similar.

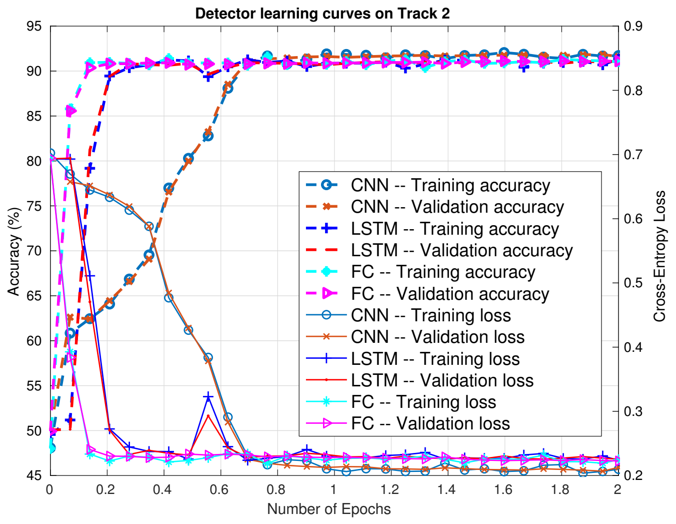


Fig. 9. The training learning curves of FC-DNN, CNN and LSTM on track 2. More than 90% accuracy is achieved within two epochs. During training, the FC-DNN classifies the fastest among the three, followed by LSTM. The CNN initially makes the slowest classification, but eventually achieves the highest accuracy. The learning curves for tracks 1 and 3 are similar.

TABLE II
Detector BER comparison

| TDMR Detectors | BER Track 1 | BER Track 2 | BER Track 3 | BER average |
|---|---|---|---|---|
| None | 18.54% | 18.53% | 18.33% | 18.47% |
| FC DNN | 8.30% | 8.21% | 7.56% | 8.03% |
| CNN | 7.99% | 7.87% | 7.38% | 7.75% |
| LSTM | 8.41% | 8.62% | 7.80% | 8.28% |
| 2D-PDNP | 10.60% | 12.48% | N/A | 11.54% |
| LAIP-BCJR | 8.99% | 7.69% | 8.88% | 8.52% |

Table II summarizes the simulation results for detector BER. The first row in the table shows the raw BERs on tracks 1, 2 and 3, which are obtained by applying a threshold of 0 on the readback signals before the equalizer. The second, third, and fourth rows show that the FC-DNN, CNN and LSTM achieve average BERs (over all three tracks) of 8.03%, 7.75%, and 8.28% respectively, thereby achieving BER reductions of 56.55%, 58.05% and 55.18% over the raw BER. This suggests that the TDMR detection problem is more suitably considered as an image processing problem rather than a time series problem.

In the fifth row in Table II, we report the BER of a state-of-art 2D-PR 2D-BCJR/2D-PDNP two-track detection system, where the 2D-PDNP two-track uses the following 2D autoregressive model to predict media noise [6]:

$$\mathbf{n}_k = \sum_{i=0}^{N_p} \mathbf{P}_i(\mathbf{A}_k)\mathbf{n}_{k-i} + \mathbf{\Lambda}(\mathbf{A}_k)\mathbf{w}_k. \tag{5}$$

Details of our implementation are described in [7]. The 2D-PDNP parameters are trained over forty blocks of GFP data and tested on the remaining sixty blocks. We observed experimentally that more than forty training blocks showed no improvement. One possible reason is that the relatively low-complexity linear model in (5) is unable to learn from more training data. Compared to the BCJR/2D-PDNP, the CNN detector achieves detector BER reductions of 24.62% and 36.94% on tracks 1 and 2, or an average BER reduction of 32.87%. The FC-DNN and the LSTM detector gives average BER reductions of 30.47% and 28.27% separately. All of the three DNN detectors detect three tracks and thus achieve a factor of $1.5\times$ throughput gain.

In the sixth row in Table II, we show the detector BER of a recently proposed 2D-PR LAIP-BCJR three-track detector as described in [7]. The LAIP detector considers a $3 \times 3$ bit cell (denoted as A, B, …, H, and U) and assumes an additive model for read back value of center bit U:

$$y_U = \alpha_U + \alpha_{\text{total}} = \alpha_U + \sum_{i \in \{A,B,\ldots,H\}} \alpha_i, \tag{6}$$

where $\alpha_\beta$ denotes the *local area influence* (LAI) on target bit U due to bit $\beta$. Details of the LAIP detector are described in [7]. The LAIP detector is trained on 512 blocks of special training patterns, simulated using the same GFP model as GFP data set #1 and with the same track length. This special training data set is required inherently by the LAIP detector due to the flip-and-subtraction estimation method for LAIs, whereas the DNN can be trained directly from the regular GFP data set or from actual HDD data if available. Compared to the LAIP-BCJR detector, the proposed CNN, FC-DNN and LSTM detector achieves average detector BER reductions of 9.07%, 5.83%, and 2.85%.

## B. Areal Density Comparison

We implement iterative decoding between the CNN detector and the IRA decoder. The LLR thresholds $T_1$ and $T_2$ at the outputs of the CNN detector and IRA decoder are $T_1 = 10.0$, and $T_2 = 5.0$. We use systematic IRA codes. The number of internal IRA decoder iterations is 100 in the first decoding pass, and 200 in the second decoding pass. For GFP data set #1, two decoding passes are done. For GFP data set #2, one decoding pass is done. Results are reported in terms of both user bits per grain (U/G), where U/G = achieved-code-rate/GPB, and areal density, where Areal-density = U/G · Grain-density. Here "achieved-code-rate" is the highest code rate after puncturing that achieves a final decoded BER of $10^{-5}$. We use a puncturing scheme that accurately simulates puncturing bits written to a HDD; this scheme is described in detail in [9].

Table III summarizes the simulation results of the DNN detection system with channel decoding. The raw channel BER is equal to that of the track detected for single-track detection, i.e., track 2, whereas it is the average BER over the three tracks detected for three-track detection, i.e., tracks 1, 2 and 3. When we get zero error counts, we also provide in parenthesis a conservative BER upper bound estimate with a 95% confidence level. This BER upper bound is computed as $3/N_{\text{tcb}}$, where $N_{\text{tcb}}$ is the total number of transmitted coded bits [30]. For GFP data set #1, with two passes of channel decoding, i.e., one loop between the CNN and the IRA, the DNN system achieves an average code rate of 0.6833 over the three detected tracks, which corresponds to 2.232 Terabits per square inch ($\text{Tb/in}^2$), or equivalently 0.1957 U/G. The base rate of the IRA code used to produce this result is 0.6506. For GFP data set #2, with one single pass of channel decoding, the DNN system gives an average code rate of 0.9433, which corresponds to 3.081 $\text{Tb/in}^2$, or 0.2702 U/G. To our knowledge, this is the highest density ever reported on GFP model data with grain density of 11.4 $\text{Tg/in}^2$. The base rate of the IRA code used to produce this result is 0.7507. Further density gains are likely to result from a higher base rate code requiring very little puncturing.

For density comparison, the BCJR/2D-PDNP detector in the fifth row of Table II is interfaced with an IRA decoder, and the average density result on the two tracks under a single decoding pass on GFP data set #2 is shown in the sixth row of Table III. We also implement a standard 1D-PR 1D-BCJR/PDNP single-track detection system [24], based on the following 1D autoregressive model:

$$n_k(\mathbf{u}_k) = \sum_{i=1}^{L} a_i(\mathbf{u}_k) n_{k-i}(\mathbf{u}_k) + \sigma(\mathbf{u}_k) w_k. \tag{7}$$

Comparing the fourth, sixth and last rows of Table III, 1D-PDNP gives 13.4% areal density gain over 2D-PDNP on GFP data set #2, whereas CNN gives 38.11% density gain, all under one decoding pass. The fact that 1D-PDNP has better performance than 2D-PDNP is at least partially due to the fact that 1D-PDNP considers a pattern of seven downtrack bits, whereas 2D-PDNP only considers three bits on each of the two tracks (as in [6] where the 2D-PDNP was proposed) in order to maintain a reasonable trellis state cardinality. We note, however, that 2D-PDNP doubles the data throughput compared to 1D-PDNP. Furthermore, the proposed CNN system with one decoding pass (last row of Table III) achieves a 21.72% density gain over the 1D-PDNP turbo system with two decoding passes (fifth row of Table III). Note that we allow up to 20 iterations between the 1D-PDNP and the IRA decoder, but the 1D-PDNP system is unable to take advantage of more turbo loops at a higher code rate. In contrast, we only trained CNNs per iteration for up to two decoding passes, and it is likely higher density can be achieved with more decoding passes. As for GFP data set #1, the proposed CNN system (third row of Table III) achieves a 5.12% areal density gain and three times throughput gain over 1D-PDNP (first row of Table III), both with two decoding passes.

We next compare the density of the CNN system with that of the LAIP-BCJR system in [7]. One global pass, i.e., two IRA decoding passes are done. The number of internal IRA decoder iterations is 200 and 100 for the two IRAs following the LAIP and the BCJR respectively. To make it a fair comparison, no inner loops between the LAIP and the first IRA decoder or between the BCJR and the second IRA decoder are done. Because the LAIP is trained using the same model as GFP data set #1, it is not evaluated on GFP data set #2. The CNN system in the third row of Table III gives a 4.32% areal density gain over the LAIP-BCJR system (second row in Table III).

TABLE III
AREAL DENSITY COMPARISON

| TDMR Detectors | GFP Model | Raw Channel BER | Number of tracks detected | Areal Density $(\mathrm{Tb/in}^2)$ | User Bits per Grain | Code Rate | Decoded BER | Decoded FER |
|---|---|---|---|---|---|---|---|---|
| 1D-PDNP 2 passes | GFP #1 | 0.1853 | 1 | 2.123 | 0.1862 | 0.6500 | 0 (1.9583e-6) | 0 (0.0500) |
| LAIP-BCJR 2 passes | GFP #1 | 0.1847 | 3 | 2.139 | 0.1876 | 0.6550 | 0 (3.7300e-7) | 0 (0.0100) |
| CNN 2 passes | GFP #1 | 0.1847 | 3 | 2.232 | 0.1957 | 0.6833 | 0 (1.8650e-6) | 0 (0.0500) |
| 1D-PDNP 1 pass | GFP #2 | 0.1641 | 1 | 2.482 | 0.2177 | 0.7600 | 0 (1.2115e-6) | 0 (0.0375) |
| 1D-PDNP 2 passes | GFP #2 | 0.1641 | 1 | 2.531 | 0.2220 | 0.7750 | 0 (1.2115e-6) | 0 (0.0375) |
| 2D-PDNP 1 pass | GFP #2 | 0.1641 | 2 | 2.230 | 0.1957 | 0.6830 | 0 (6.8377e-06) | 0 (0.0250) |
| CNN 1 pass | GFP #2 | 0.1637 | 3 | 3.081 | 0.2702 | 0.9433 | 0 (1.6153e-6) | 0 (0.0500) |

TABLE IV
COMPUTATIONAL COMPLEXITY COMPARISON

| Method | mul/div | add/sub | exp/log |
|---|---|---|---|
| 1D-PDNP | 137,985 | 106,746 | 257 |
| 2D-PDNP | 86,657 | 54,392 | 257 |
| LAIP-BCJR | 291,560 | 189,279 | 257 |
| CNN | 113,761 | 102,062 | 1 |

### C. Storage, Latency and Complexity Comparison

In regards to offline storage overhead, the conditional PMF tables stored by the LAIP require about 780 MB of storage [9]. The three FC-DNNs require 0.18 MB for storing $49,518$ learnables, and the LSTM requires 17 MB for storing $4,686,006$ learnables for GFP data set #1. The three CNNs in each decoding pass together require approximately 1.2 MB for storing around 0.3 million learnables. The 1D-PDNP requires 0.035 MB for storing $2,560$ parameters. All the variables above are stored as double-precision floating-point values. The storage requirement for the LAIP-BCJR system is thus $325\times$ that of the two-pass CNN system.

As for online detection time, in the 1D-PDNP system, the latency due to one run of the 1D-PDNP detector is roughly $771.9$ microseconds ($\mu$s) per bit. In the LAIP-BCJR system, the latency caused by one run of LAIP detector is approximately $766.4$ $\mu\mathrm{s/bit}$, and $281.3$ $\mu\mathrm{s/bit}$ for one run of the 2D-BCJR. For the one global shot LAIP-BCJR system in the second row of Table III, the total latency caused by the LAIP and 2D-BCJR detectors is thus $1047.7$ $\mu\mathrm{s/bit}$. In the DNN detection system, the three tracks can be detected in parallel. The latency is around $12.8$ $\mu\mathrm{s/bit}$ for the FC-DNN, and $94.2$ $\mu\mathrm{s/bit}$ for LSTM. The CNN latency ranges from $64.2$ $\mu\mathrm{s/bit}$ to $79.5$ $\mu\mathrm{s/bit}$ per decoding pass. Thus the CNN latency for one decoding pass in the last row of Table III is approximately $1/10$ the latency of one pass of the 1D-PDNP in the fourth row of Table III. The above results are estimated when all systems are running on the same CPU. When running on a GPU, the CNN latency is reduced to the range between $12.1$ $\mu\mathrm{s/bit}$ and $14.3$ $\mu\mathrm{s/bit}$, and the latency of LSTM becomes $43.2$ $\mu\mathrm{s/bit}$. We note that the capability to use GPU-enabled hardware for acceleration of training and real-time operation is an inherent advantage of DNNs.

In addition to the run time measurement, computational complexity comparison (per bit) between the four detectors is given in Table IV. The 1D-PDNP looks at $1\times 7$ bit patterns on track 2, and has 128 states. The 2D-PDNP considers $2\times 3$ bit patterns on tracks 1 and 2, and has 64 states. 2D-PDNP with 256 states was investigated in [7], and it gives 0.7% BER reduction on track 2 as well as 0.2% BER increase on track 1, compared to the 64 state version. However, the complexity of the BCJR algorithm grows as the square of the number of states (and same statement holds for Viterbi algorithm), and 256-state 2D-PDNP would require more than 1 million multiplications. The minimal performance improvement of 256 states does not justify the increase in complexity. Thereby we present only results of 64 states here. For the LAIP-BCJR detector, more than 90% of the complexity is due to the 2D-BCJR, which is a straightforward BCJR implementation according to [31]. The number of states in the BCJR algorithm in turn grows exponentially with the number of tracks being detected. This explains why the LAIP-BCJR that detects three tracks has the highest complexity. The LAIP detector is implemented in MATLAB, which might explain the large running time of the LAIP. The CNN is implemented using the deep learning toolbox in MATLAB. Table IV shows that the CNN requires fewer operations than both the 1D-PDNP and the LAIP-BCJR. This supports the measurement that CNN has the least latency. We note that exponentiations/logarithms are required in our implementation of BCJR, but could be avoided if an approximation such as MAX-Log-MAP algorithm [32] is used.

### D. Boundary Conditions

The results in Tables II and III assume perfect knowledge of the bits on the top and bottom boundary tracks. In a more realistic scenario, bits on both boundary tracks would be unknown, but readback values would still be available. By making hard decisions on the boundary track readings, we can (roughly) estimate the boundary track bits. For GFP data set #1, this would mean a raw BER of $18.5\%$ for both boundary rows. In the 2D LAIP-BCJR system we describe in [9], we estimate the unknown boundary tracks by pre-processing them with a simple 1D BCJR detector with four trellis states. Simulations in [9] with GFP data set #1 show that this scheme reduces the boundary track BER to 7.6%. In [9], we therefore introduce random errors at a 7.6% BER into the boundary track data bits, and simulate the LAIP-BCJR detector with these estimated boundary conditions; the simulations show that using the estimated boundaries reduces the LAIP-BCJR detector's achieved areal density by about 7.0% compared to the perfect boundary case. We anticipate that the CNN APP detector would suffer a similar density penalty if boundary estimation were used; for GFP data set #1, this density penalty would drop the achieved areal density of the LAIP-BCJR and CNN APP detectors slightly below that of 1D-PDNP, which does not use adjacent track information. Further reduction of the boundary track BER can be achieved by using a single channel decoder pass on the boundary tracks, which would improve the CNN APP detector's achieved density compared to 1D-PDNP. It also important to note that, for GFP data set #2, the lower raw BER of 16.4% would result in lower boundary bit BERs after simple four state BCJR detection, thus reducing the areal density penalty due to unknown boundaries to $< 7\%$, and that, even assuming a 7% density loss, the CNN APP detector with one channel decoding pass would still achieve a 13.2% density gain over the 1D-PDNP detector (with 2 decoding passes) on GFP data set #2.

## VIII. Conclusion

This paper describes the design of a DNN-based APP detection system for parallel multi-track turbo TDMR detection. We investigate three DNN architectures, all of which achieve significant BER reductions compared to state-of-art multi-track detectors. The CNN is found to be the best performing DNN architecture, and we select it for iterative decoding with a channel decoder. The CNN APP detection system achieves significant areal density gains over the standard 1D-PDNP and state-of-the-art LAIP-BCJR detection systems with same or fewer decoding passes, and with $\leq 1/10$ the per bit latency. More turbo loops between the CNN detector and the channel decoder should result in further density gains.

## Acknowledgment

This work is supported by the United States National Science Foundation (NSF) under grant CCF-1817083. The authors would like to thank the anonymous reviewers for their constructive comments and suggestions.

## References

[1] R. Wood, M. Williams, A. Kavcic, and J. Miles, "The feasibility of magnetic recording at 10 terabits per square inch on conventional media," *IEEE Trans. Magnetics*, vol. 45, no. 2, pp. 917–923, Feb. 2009.

[2] A. Kavcic and J. M. Moura, "The Viterbi algorithm and Markov noise memory," *IEEE Trans. Inform. Theory*, vol. 46, no. 1, pp. 291–301, Jan. 2000.

[3] J. Moon and J. Park, "Pattern-dependent noise prediction in signal-dependent noise," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 4, pp. 730–743, Apr 2001.

[4] J. Yao, E. Hwang, B. V. K. V. Kumar, and G. Mathew, "Two-track joint detection for two-dimensional magnetic recording (TDMR)," in *2015 IEEE Int. Conf. on Commun. (ICC)*, June 2015, pp. 418–424.

[5] Y. Wang and B. V. K. V. Kumar, "Multi-track joint detection for shingled magnetic recording on bit patterned media with 2-d sectors," *IEEE Trans. Magnetics*, vol. 52, no. 7, pp. 1–7, July 2016.

[6] S. Shi and J. R. Barry, "Multitrack detection with 2D pattern-dependent noise prediction," in *2018 IEEE Int. Conf. on Commun. (ICC)*, May 2018, pp. 1–6.

[7] J. Shen, X. Sun, K. Sivakumar, B. J. Belzer, K. S. Chan, and A. James, "TDMR detection system with local area influence probabilistic a priori detector," in *2019 IEEE Int. Conf. on Commun. (ICC)*, May 2019, pp. 1–7.

[8] X. Sun, K. Sivakumar, B. Belzer, and R. Wood, "High density turbo TDMR detection with local area influence probabilistic model," *IEEE Trans. Magnetics*, vol. 53, no. 2, pp. 1–8, Feb. 2017, article no. 9400208.

[9] X. Sun, J. Shen, B. J. Belzer, K. Sivakumar, A. James, K. S. Chan, and R. Wood, "ISI/ITI turbo equalizer for TDMR using trained local area influence probabilistic model," *IEEE Trans. Magnetics*, vol. 55, no. 4, pp. 1–15, April 2019.

[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[11] Y. Lecun and Y. Bengio, "Convolutional networks for images, speech, and time-series," in *The handbook of brain theory and neural networks*, M. Arbib, Ed. Cambridge, MA, USA: MIT Press, 1995.

[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[13] M. Yamashita, H. Osawa, Y. Okamoto, Y. Nakamura, Y. Suzuki, K. Miura, and H. Muraoka, "Read/write channel modeling and two-dimensional neural network equalization for two-dimensional magnetic recording," *IEEE Trans. Magnetics*, vol. 47, no. 10, pp. 3558–3561, Oct 2011.

[14] M. Yamashita, Y. Okamoto, Y. Nakamura, H. Osawa, K. Miura, S. J. Greaves, H. Aoi, Y. Kanai, and H. Muraoka, "Modeling of writing process for two-dimensional magnetic recording and performance evaluation of two-dimensional neural network equalizer," *IEEE Trans. Magnetics*, vol. 48, no. 11, pp. 4586–4589, Nov 2012.

[15] Y. Wang and B. V. K. V. Kumar, "Micromagnetics-based analysis of multi-track detection with simplified 2-d write precompensation on shingled magnetic recording," *IEEE Trans. Magnetics*, vol. 52, no. 9, pp. 1–11, Sept 2016.

[16] K. S. Chan, R. Radhakrishnan, K. Eason, M. R. Elidrissi, J. J. Miles, B. Vasic, and A. R. Krishnan, "Channel models and detectors for two-dimensional magnetic recording," *IEEE Trans. Magnetics*, vol. 46, no. 3, pp. 804–811, March 2010.

[17] K. S. Chan, E. M. Rachid, K. Eason, R. Radhakrishnan, and K. K. Teo, "Comparison of one- and two-dimensional detectors on simulated and spin-stand readback waveforms," *Journal of Magnetism and Magnetic Materials*, vol. 324, no. 3, pp. 336 – 343, 2012, selected papers of the 9th Perpendicular Magnetic Recording Conference(PMRC 2010). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304885310009170

[18] K. Keng Teo, M. R. Elidrissi, K. S. Chan, and Y. Kanai, "Analysis and design of shingled magnetic recording systems," *Journal of Applied Physics*, vol. 111, no. 07B716, 2012. [Online]. Available: http://aip.scitation.org/doi/pdf/10.1063/1.3679383

[19] K. S. Chan, K. K. Teo, M. Y. Lin, H. C. Kee, M. R. Elidrissi, Q. Li, B. Ko, Q. Choo, and M. N. Maung, "Comparison of signals from micromagnetic simulations, GFP model, and an HDD readback," *IEEE Trans. Magnetics*, vol. 51, no. 11, pp. 1–4, Nov 2015.

[20] M. R. Elidrissi, K. S. Chan, K. K. Teo, K. Eason, E. Hwang, B. V. K. V. Kumar, and Z. Qin, "Modeling of 2-D magnetic recording and a comparison of data detection schemes," *IEEE Trans. Magnetics*, vol. 47, no. 10, pp. 3685–3690, Oct 2011.

[21] A. Sayyafan, B. J. Belzer, K. Sivakumar, J. Shen, K. S. Chan, and A. James, "Deep neural network based media noise predictors for use in high-density magnetic recording turbo-detectors," *IEEE Trans. Magnetics*, vol. 55, no. 12, pp. 1–6, Dec. 2019.

[22] T. Chong, S. Piramanayagam, and R. Sbiaa, "Perspectives for 10 terabits/in$^2$ magnetic recording," *Journal of Nanoscience and Nanotechnology*, vol. 11, pp. 2704–9, 03 2011.

[23] K. S. Chan and M. R. Elidrissi, "A system level study of two-dimensional magnetic recording (TDMR)," *IEEE Trans. Magnetics*, vol. 49, no. 6, pp. 2812–2817, June 2013.

[24] E. Kurtas, J. Park, X. Yang, W. Radich, and A. Kavi, "Detection methods for data-dependent noise in storage channels," in *Coding and Signal Processing for Magnetic Recording Systems*, B. Vasi and E. Kurtas, Eds. CRC Press, 2004, ch. 33.

[25] H. Jin, A. Khandekar, and R. McEliece, "Irregular repeat-accumulate codes," in *Proc. 2nd Int. Symp. on Turbo Codes and Rel. Top.*, Brest, France, Sept. 2000, pp. 1–8.

[26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference for Learning Representations*, San Diego, CA, 2015. [Online]. Available: https://arxiv.org/abs/1412.6980

[28] C. K. Matcha and S. G. Srinivasa, "Generalized partial response equalization and data-dependent noise predictive signal detection over media models for TDMR," *IEEE Trans. Magnetics*, vol. 51, pp. 1–15, Oct. 2015, article no. 3101215.

[29] K. S. Chan, A. James, S. Shafi'ee, S. Rahardja, J. Shen, K. Sivakumar, and B. J. Belzer, "User areal density optimization for conventional and 2-D detectors/decoders," *IEEE Trans. Magnetics*, vol. 54, no. 2, pp. 1–12, Feb. 2018, article no. 3100412.

[30] F. Scholz, "Confidence bounds and intervals for parameters relating to the binomial, negative binomial, Poisson and hypergeometric distributions with applications to rare events," *ConfidenceBounds.pdf*, 2008. [Online]. Available: http://faculty.washington.edu/fscholz/DATAFILES498B2008/

[31] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. 20, pp. 284–287, March 1974.

[32] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal map decoding algorithms operating in the log domain," in *Proceedings IEEE International Conference on Communications ICC '95*, vol. 2, June 1995, pp. 1009–1013 vol.2.