This is the Pre-Published Version.

The following publication F. N. Khan, Q. Fan, C. Lu and A. P. T. Lau, "An Optical Communication's Perspective on Machine Learning and Its Applications," in Journal of Lightwave Technology, vol. 37, no. 2, pp. 493-516, 15 Jan.15, 2019 is available at https://doi.org/10.1109/JLT.2019.2897313.

An Optical Communication's Perspective on Machine Learning and its Applications

Faisal Nadeem Khan, Qirui Fan, Chao Lu, and Alan Pak Tao Lau

***Abstract*— Machine Learning (ML) has disrupted a wide range of science and engineering disciplines in recent years. ML applications in optical communications and networking are also gaining more attention, particularly in the areas of nonlinear transmission systems, optical performance monitoring (OPM) and cross-layer network optimizations for software-defined networks (SDNs). However, the extent to which ML techniques can benefit optical communications and networking is not clear and this is partly due to an insufficient understanding of the nature of ML concepts. This review article aims to describe the mathematical foundations of basic ML techniques from communication theory and signal processing perspectives, which in turn will shed light on the types of problems in optical communications and networking that naturally warrant ML use. This will be followed by an overview of ongoing ML research in optical communications and networking with a focus on physical layer issues.**

***Index Terms*—Machine learning, deep learning, artificial intelligence, optical communications, software-defined networks, optical performance monitoring.**

I. Introduction

Artificial intelligence (AI) makes use of computers/machines to perform cognitive tasks, i.e., the ones requiring knowledge, perception, learning, reasoning, understanding and other similar cognitive abilities. An AI system is expected to do three things: (i) store knowledge, (ii) apply the stored knowledge to solve problems, and (iii) acquire new knowledge via experience. The three key components of an AI system include knowledge representation, machine learning (ML), and automated reasoning. ML is a branch of AI which is based on the idea that patterns and trends in a given data set can be learned automatically through algorithms. The learned patterns and structures can then be used to make decisions or predictions on some other data in the system of interest [1].

ML is not a new field as ML-related algorithms exist at least since the 1970s. However, tremendous increase in computational power over the last decade, recent groundbreaking developments in theory and algorithms surrounding ML, and easy access to an overabundance of all types of data worldwide (thanks to three decades of Internet growth) have all contributed to the advent of modern deep learning (DL) technology, a class of advanced ML approaches that displays superior performance in an ever-expanding range of domains. In the near future, ML is expected to power numerous aspects of modern society such as web searches, computer translation, content filtering on social media networks, healthcare, finance, and laws [2].

ML is an interdisciplinary field which shares common threads with the fields of statistics, optimization, information theory, and game theory. Most ML algorithms perform one of the following two types of pattern recognition tasks as shown in

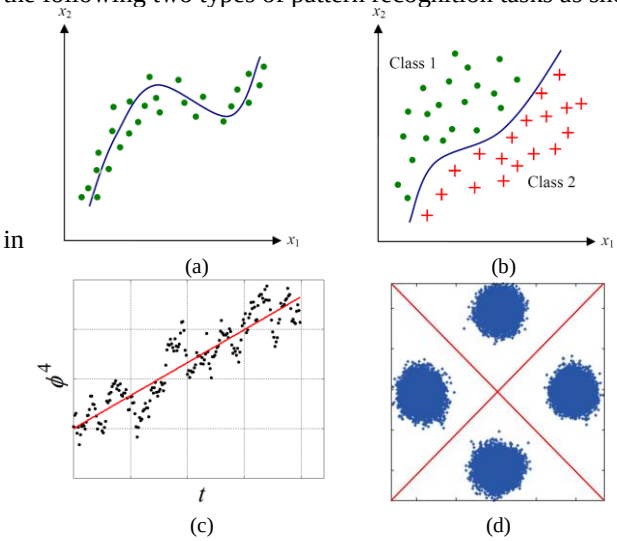
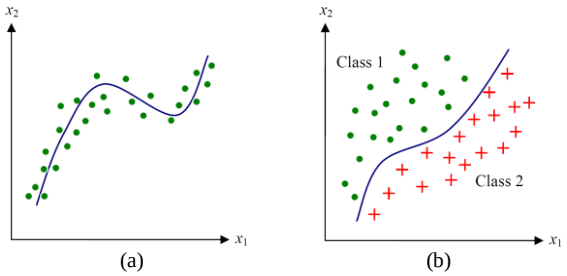


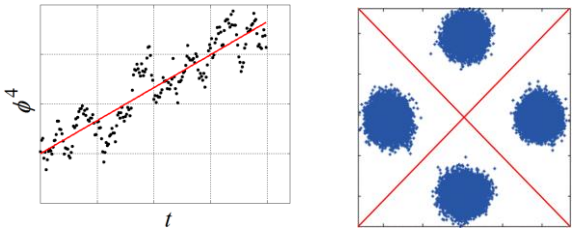
Fig. 1. In the first type, the algorithm tries to find some functional description of given data with the aim of predicting values for new inputs, i.e., *regression problem*. The second type attempts to find suitable decision boundaries to distinguish different data classes, i.e., *classification problem* [3], which is more commonly referred to as *clustering problem* in ML literature. ML techniques are well known for performing exceptionally well in scenarios in which it is too hard to explicitly describe the problem's underlying physics and mathematics.



Manuscript received July, 2018. This work was supported by Hong Kong Government General Research Fund under project number PolyU 152757/16E as well as by National Natural Science Foundation China under project numbers 61435006 and 61401020. *(Corresponding author: Faisal Nadeem Khan.)*

F. N. Khan, Q. Fan, C. Lu, and A. P. T. Lau are with Photonics Research Centre, (Need to separate us and Prof. Lu into EE and EIE)The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong SAR, China. (e-mail: fnadeem.khan@yahoo.com; remi.qr.fan@gmail.com chao.lu@polyu.edu.hk; eeaptlau@polyu.edu.hk).

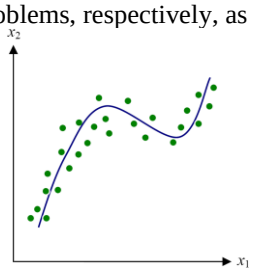
© 2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.



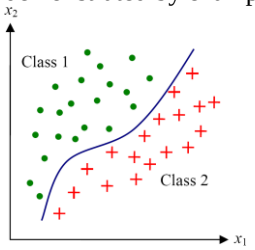
(c) (d)

Fig. 1. Given a data set, ML attempts to solve two main types of problems: (a) functional description of given data and (b) classification of data by deriving appropriate decision boundaries. (c) Laser frequency offset and phase estimation for quadrature phase-shift keying (QPSK) systems by raising the signal phase ϕ to the 4th power and performing regression to estimate the slope and intercept. (d) Decision boundaries for a received QPSK signal distribution.

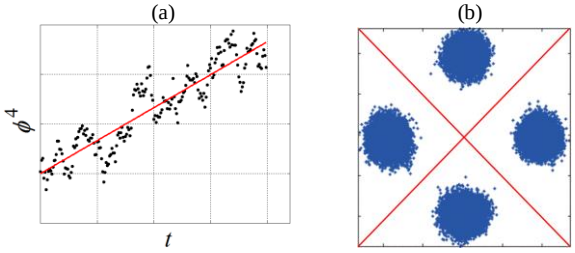
Optical communication researchers are no strangers to regressions and classifications. Over the last decade, coherent detection and digital signal processing (DSP) techniques have been the cornerstone of optical transceivers in fiber-optic communication systems. Advanced modulation formats such as 16 quadrature amplitude modulation (16-QAM) and above together with DSP-based estimation and compensation of various transmission impairments such as laser phase noise have become the key drivers of innovation. In this context, parameter estimation and symbol detection are naturally regression and classification problems, respectively, as



demonstrated by examples in



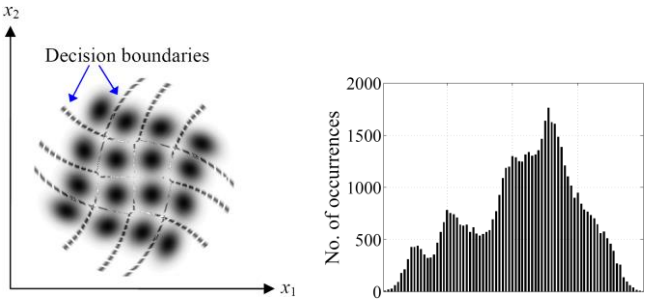
(a)



(b)

(c) (d)

Fig. 1(c) and (d). Currently, most of these parameter estimation and decision rules are derived from probability theory and adequate understanding of the problem's underlying physics. As high-capacity optical transmission links are increasingly being limited by transmission impairments such as fiber nonlinearity, explicit statistical characterizations of inputs/outputs become difficult. An example of 16-QAM multi-span dispersion-free transmissions in the presence of fiber nonlinearity and inline amplifier noise is shown in Fig. 2(a). The maximum likelihood decision boundaries in this case are curved and virtually impossible to derive analytically. Consequently, there has been an increasing amount of research on the application of ML techniques for fiber nonlinearity compensation (NLC). Another related area where ML flourishes is short-reach direct detection systems that are affected by chromatic dispersion (CD), laser chirp and other transceiver components imperfections, which render the overall communication system hard to analyze.



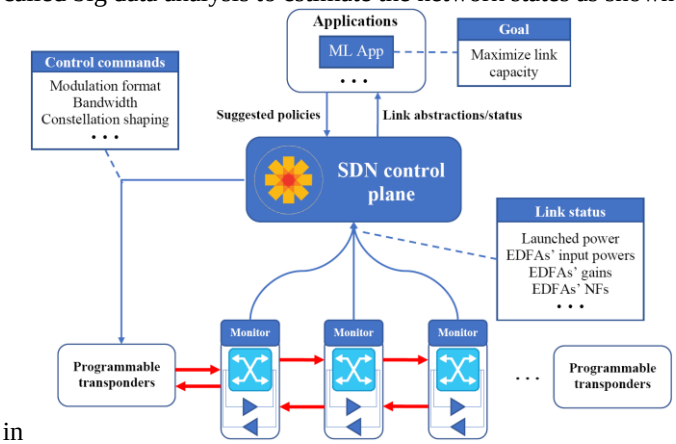
(a) (b)

Fig. 2. (a) Probability distribution and corresponding optimal decision boundaries for received 16-QAM symbols in the presence of fiber nonlinearity are hard to characterize analytically. (b) Probability distribution of received 64-QAM signal amplitudes. The distribution can be used to monitor optical signal-to-noise ratio (OSNR) and identify modulation format. However, this task will be extremely difficult if one relies on analytical modeling.

Optical performance monitoring (OPM) is another area with an increasing amount of ML-related research. OPM is the acquisition of real-time information about different channel impairments ubiquitously across the network to ensure reliable network operation and/or improve network capacity. Often, OPM is cost-limited so that one can only employ simple hardware components and obtain partial signal features to monitor different channel parameters such as OSNR, optical power, CD, etc. [4]. In this case, the mapping between input and output parameters is intractable from underlying physics/mathematics, which in turn warrants ML. An example of OSNR monitoring using received signal amplitudes distribution is shown in Fig. 2(b).

Besides physical layer-related developments, optical network architectures and operations are also undergoing major paradigm shifts under the software-defined networking (SDN) framework and are increasingly becoming complex, transparent and dynamic in nature [5]. One of the key features of SDNs is that they can assemble large amounts of data and perform so-

called big data analysis to estimate the network states as shown in Fig. 3. This in turn can enable (i) adaptive provisioning of resources such as wavelength, modulation format, routing path, etc., according to dynamic traffic patterns and (ii) advance discovery of potential components faults so that preventative maintenance can be performed to avoid major network disruptions. The data accumulated in SDNs can span from physical layer (e.g., OSNR of a certain channel) to network layer (e.g., client-side speed demand) and obviously have no underlying physics to explain their interrelationships. Extracting patterns from such cross-layer parameters naturally demands the use of data-driven algorithms such as ML.



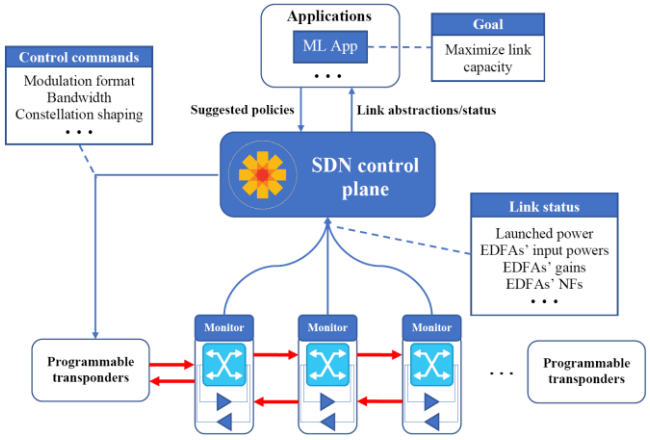


Fig. 3. Dynamic network resources allocation and link capacity maximization via cross-layer optimization in SDNs.

This review paper is intended for the researchers in optical communications with a basic background in probability theory, communication theory and standard DSP techniques used in fiber-optic communications such as matched filters, maximum likelihood/maximum a posteriori (MAP) detection, equalization, adaptive filtering, etc. In this regard, a large class of ML techniques such as Kalman filtering, Bayesian learning, hidden Markov models (HMMs), etc., are actually standard statistical signal processing methods, and hence will not be covered here. We will first introduce artificial neural networks (ANNs) and support vector machines (SVMs) from communication theory and signal processing perspectives. This will be followed by other popular ML techniques like *K*-means clustering, expectation-maximization (EM) algorithm, principal component analysis (PCA), independent component analysis (ICA), as well as more recent DL approaches such as deep neural networks (DNNs), convolutional neural networks (CNNs) and recurrent neural networks (RNNs). The analytical derivations presented in this paper are slightly different from those in standard introductory ML text to better align with the fields of communications and signal processing. We will then provide an overview of applications of ML techniques in various aspects of optical communications and networking.

We emphasize that this is by no means an exhaustive and in-depth discussion on state-of-the-art ML techniques and their respective challenges. Also, the views presented are not the only way to understand the fundamental properties of ML methods. By discussing ML through the language of communications and DSP, we hope to provide a more intuitive understanding of ML, its relation to optical communications and networking, and why/where/how it can play a unique role in specific areas of optical communications and networking.

The rest of the paper is organized as follows. In Section II, we will illustrate the fundamental conditions that warrant the use of a neural network and discuss the technical details of ANNs and SVMs. Section III will describe a range of basic unsupervised ML techniques and briefly discuss reinforcement learning (RL). Section IV will be devoted to more recent ML algorithms. Section V will provide an overview of existing ML applications in optical communications and networking while Section VI will discuss their future role. Links for online resources and codes for standard ML algorithms will be provided in Section VII. Section VIII will conclude the paper. A video presentation of the paper is available at [6].

II. ANNs AND SVMs

What are the conditions that need ML for classification?

Fig. 4 shows three scenarios with 2-dimensional (2D) data $\mathbf{x} = [x_1\ x_2]^{\mathrm{T}}$ and their respective class labels depicted as 'o' and '×' in the figure. In the first case, classifying the data is straightforward: the decision rule is to see whether $\sigma(x_1 - c)$ or $\sigma(x_2 - c)$ is greater or less than 0 where $\sigma(\cdot)$ is the decision function as shown. The second case is slightly more complicated as the decision boundary is a slanted straight line. However, a simple rotation and shifting of the input, i.e., $\mathbf{Wx} + \mathbf{b}$ will map one class of data to below zero and the other class above. Here, the rotation and shifting are described by matrix $\mathbf{W}$ and vector $\mathbf{b}$, respectively. This is followed by the decision function $\sigma(\mathbf{Wx} + \mathbf{b})$. The third case is even more complicated. The region for the 'green' class depends on the outputs of the 'red' and 'blue' decision boundaries. Therefore, one will need to implement an extra decision step to label the 'green' region. The graphical representation of this 'decision of decisions' algorithm is the simplest form of an ANN [7]. The intermediate decision output units are known as hidden neurons and they form the hidden layer.

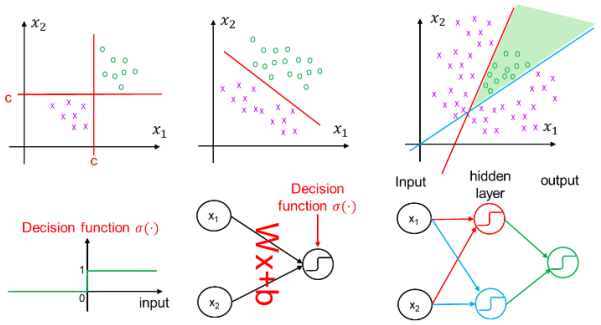


Fig. 4. The complexity of classification problems depends on how the different classes of data are distributed across the variable space.

A. Artificial Neural Networks (ANNs)

Let $\{(\mathbf{x}(1),\mathbf{y}(1)),(\mathbf{x}(2),\mathbf{y}(2)),\ldots(\mathbf{x}(L),\mathbf{y}(L))\}$ be a set of L input-output pairs of M and K dimensional column vectors. ANNs are information processing systems comprising of an input layer, one or more hidden layers, and an output layer. The structure of a single hidden layer ANN with M input, H hidden and K output neurons, respectively, is shown in

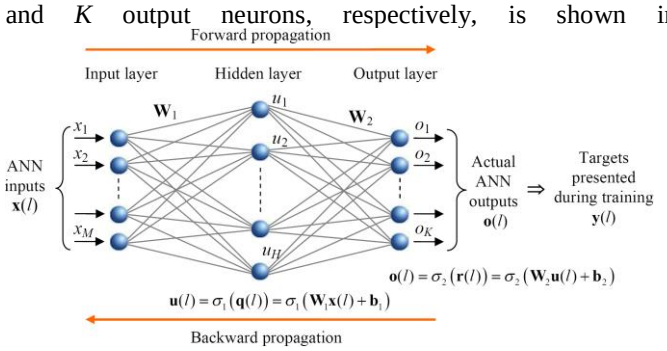


Fig. 5. Neurons in two adjacent layers are interconnected where each connection has a variable weight assigned. Such ANN architecture is the simplest and most commonly-used one [7]. The number of neurons M in the input layer is determined by the dimension of the input data vectors $\mathbf{x}(l)$. The hidden layer enables the modeling of complex relationships between the input and output parameters of an ANN. There are no fixed rules for choosing the optimum number of neurons for a given hidden layer and the optimum number of hidden layers in an ANN. Typically, the selection is made via experimentation, experience and other prior knowledge of the problem. These are known as the *hyperparameters* of an ANN. For regression problems, the dimension K of the vectors $\mathbf{y}(l)$ depends on the actual problem nature. For classification problems, K typically equals to the number of class labels such that if a data point $\mathbf{x}(l)$ belongs to class k, $\mathbf{y}(l) = [0\ 0\cdots 0\ 1\ 0\cdots 0\ 0]^{\mathrm{T}}$ where the '1' is located at the k^{th} position. This is called *one-hot encoding*. The ANN output $\mathbf{o}(l)$ will naturally have the same dimension as $\mathbf{y}(l)$ and the mapping between input $\mathbf{x}(l)$ and $\mathbf{o}(l)$ can be expressed as

$$\begin{aligned}\mathbf{o}(l) &= \sigma_2(\mathbf{r}(l)) \\ &= \sigma_2(\mathbf{W}_2\mathbf{u}(l)+\mathbf{b}_2) \\ &= \sigma_2(\mathbf{W}_2\,\sigma_1(\mathbf{q}(l))+\mathbf{b}_2) \\ &= \sigma_2(\mathbf{W}_2\,\sigma_1(\mathbf{W}_1\,\mathbf{x}(l)+\mathbf{b}_1)+\mathbf{b}_2)\end{aligned} \tag{1}$$

where $\sigma_{1(2)}(\cdot)$ are the *activation functions* for the hidden and output layer neurons, respectively. $\mathbf{W}_1$ and $\mathbf{W}_2$ are matrices containing the weights of connections between the input and hidden layer neurons and between the hidden and output layer neurons, respectively, while $\mathbf{b}_1$ and $\mathbf{b}_2$ are the bias vectors for the hidden and output layer neurons, respectively. For a vector $\mathbf{z} = [z_1\ z_2\ \cdots z_K]$ of length K, $\sigma_1(\cdot)$ is typically an element-wise nonlinear function such as the sigmoid function

$$\sigma_1(\mathbf{z}) = \left[\frac{1}{1+e^{-z_1}}\quad \frac{1}{1+e^{-z_2}}\cdots\quad \frac{1}{1+e^{-z_K}}\right]. \tag{2}$$

As for the output layer neurons, $\sigma_2(\cdot)$ is typically chosen to be a linear function for regression problems. In classification problems, one will normalize the output vector $\mathbf{o}(l)$ using the *softmax* function, i.e.,

$$\mathbf{o}(l) = softmax(\mathbf{W}_2\,\mathbf{u}(l)+\mathbf{b}_2) \tag{3}$$

where

$$softmax(\mathbf{z}) = \frac{1}{\sum_{k=1}^{K} e^{z_k}}\left[e^{z_1}\quad e^{z_2}\quad\cdots\quad e^{z_K}\right]. \tag{4}$$

The softmax operation ensures that the ANN outputs conform to a probability distribution for reasons we will discuss below.

To train the ANN is to optimize all the parameters $\theta = \{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}_1,\mathbf{b}_2\}$ such that the difference between the actual ANN outputs $\mathbf{o}$ and the target outputs $\mathbf{y}$ is minimized. One commonly-used objective function (also called *loss function* in ML literature) to optimize is the mean square error (MSE)

$$E = \frac{1}{L}\sum_{l=1}^{L} E(l) = \frac{1}{L}\sum_{l=1}^{L}\|\mathbf{o}(l)-\mathbf{y}(l)\|^2 \tag{5}$$

Like most optimization procedures in practice, gradient descent is used instead of full analytical optimization. In this case, the parameter estimates for n+1$^{\text{th}}$ iteration are given by

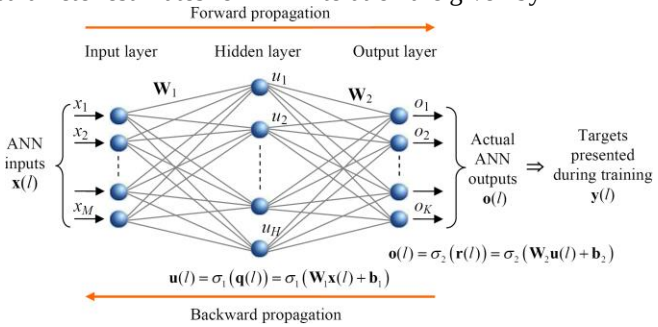


Fig. 5. Structure of a single hidden layer ANN with input vector $\mathbf{x}(l)$, target vector $\mathbf{y}(l)$ and actual output vector $\mathbf{o}(l)$.

$$\theta^{(n+1)} = \theta^{(n)} - \alpha\left.\frac{\partial E}{\partial\theta}\right|_{\theta^{(n)}} \tag{6}$$

where the step size α is known as the *learning rate*. Note that for computational efficiency, one can use a single input-output pair instead of all the L pairs for each iteration in Eq. (6). This is known as stochastic gradient descent (SGD) which is the standard optimization method used in common adaptive DSP such as constant modulus algorithm (CMA) and least mean squares (LMS) algorithm. As a trade-off between computational efficiency and accuracy, one can use a *mini-batch* of data $\{(\mathbf{x}(nP+1),\mathbf{y}(nP+1)),\big((\mathbf{x}(nP+2),\mathbf{y}(nP+2))\big)\ldots(\mathbf{x}(nP+P),\mathbf{y}(nP+P))\}$ of size P for the n^{th} iteration instead. This can reduce the stochastic nature of SGD and

improve accuracy. When all the data set has been used, the update algorithm will have completed one *epoch*. However, it is often the case that one epoch equivalent of updates is not enough for all the parameters to converge to their optimal values. Therefore, one can reuse the data set and the algorithm goes through the 2nd epoch for further parameter updates. There is no fixed rule to determine the number of epochs required for convergence [8].

The update algorithm is comprised of following main steps: (i) *Model initialization*: All the ANN weights and biases are randomly initialized, e.g., by drawing random numbers from a normal distribution with zero mean and unit variance; (ii) *Forward propagation:* In this step, the inputs $\mathbf{x}$ are passed through the network to generate the outputs $\mathbf{o}$ using Eq. (1). The input can be a single data point, a mini-batch or the complete set of L inputs. This step is named so because the computation flow is in the natural forward direction, i.e., starting from the input, passing through the network, and going to the output; (iii) *Backward propagation and weights/biases update:* For simplicity, let us assume SGD using 1 input-output pair $(\mathbf{x}(n), \mathbf{y}(n))$ for the n+1th iteration, sigmoid activation function for the hidden layer neurons and linear activation function for the output layer neurons such that $\mathbf{o}(n) = \mathbf{W}_2\,\mathbf{u}(n) + \mathbf{b}_2$. The parameters $\mathbf{W}_2, \mathbf{b}_2$ will be updated first followed by $\mathbf{W}_1, \mathbf{b}_1$. Since $E(n) = \|\mathbf{o}(n) - \mathbf{y}(n)\|^2$ and $\frac{\partial E(n)}{\partial \mathbf{o}(n)} = 2(\mathbf{o}(n) - \mathbf{y}(n))$, the corresponding update equations are

$$\begin{aligned} \mathbf{W}_2^{(n+1)} &= \mathbf{W}_2^{(n)} - 2\alpha \sum_{k=1}^{K} \frac{\partial o_k(n)}{\partial \mathbf{W}_2} (o_k(n) - y_k(n))^{1} \\ \mathbf{b}_2^{(n+1)} &= \mathbf{b}_2^{(n)} - 2\alpha \frac{\partial \mathbf{o}(n)}{\partial \mathbf{b}_2} (\mathbf{o}(n) - \mathbf{y}(n)) \end{aligned} \tag{7}$$

where $o_k(n)$ and $y_k(n)$ denote the kth element of vectors $\mathbf{o}(n)$ and $\mathbf{y}(n)$, respectively. In this case, $\frac{\partial \mathbf{o}(n)}{\partial \mathbf{b}_2}$ is the Jacobian matrix in which the j^{th} row and mth column is the derivative of the mth element of $\mathbf{o}(n)$ with respect to the jth element of $\mathbf{b}_2$. Also, the j^{th} row and mth column of the matrix $\frac{\partial o_k(n)}{\partial \mathbf{W}_2}$ denotes the derivative of $o_k(n)$ with respect to the j^{th} row and mth column of $\mathbf{W}_2$. Interested readers are referred to [9] for an overview of matrix calculus. Since $\mathbf{o}(n) = \mathbf{W}_2\,\mathbf{u}(n) + \mathbf{b}_2$, $\frac{\partial \mathbf{o}(n)}{\partial \mathbf{b}_2}$ is simply the identity matrix. For $\frac{\partial o_k(n)}{\partial \mathbf{W}_2}$, its k^{th} row is equal to $\mathbf{u}(n)^{\text{T}}$ (where $(\cdot)^{\text{T}}$ denotes transpose) and is zero otherwise. Eq. (7) can be simplified as

$$\begin{aligned} \mathbf{W}_2^{(n+1)} &= \mathbf{W}_2^{(n)} - 2\alpha(\mathbf{o}(n) - \mathbf{y}(n))\mathbf{u}(n)^{\text{T}} \\ \mathbf{b}_2^{(n+1)} &= \mathbf{b}_2^{(n)} - 2\alpha\big(\mathbf{o}(n) - \mathbf{y}(n)\big). \end{aligned} \tag{8}$$

With the updated $\mathbf{W}_2^{(n+1)}$and $\mathbf{b}_2^{(n+1)}$, one can calculate

$$\begin{aligned} \mathbf{W}_1^{(n+1)} &= \mathbf{W}_1^{(n)} - 2\alpha \sum_{k=1}^{K} \frac{\partial o_k(n)}{\partial \mathbf{W}_1} (o_k(n) - y_k(n)) \\ \mathbf{b}_1^{(n+1)} &= \mathbf{b}_1^{(n)} - 2\alpha \frac{\partial \mathbf{o}(n)}{\partial \mathbf{b}_1} \big(\mathbf{o}(n) - \mathbf{y}(n)\big). \end{aligned} \tag{9}$$

[1] One can also express the update of $\mathbf{W}_2$ using 3rd-order tensor notation $\frac{\partial \boldsymbol{o}(n)}{\partial \mathbf{W}_2}$ as supposed to $\sum_k \frac{\partial o_k(n)}{\partial \mathbf{W}_2}$.

Since the derivative of the sigmoid function is given by $\sigma_1'(\mathbf{z}) = \sigma_1(\mathbf{z}) \circ (\mathbf{1} - \sigma_1(\mathbf{z}))$ where $\circ$ denotes element-wise multiplication and $\mathbf{1}$ denotes a column vector of 1's with the same length as $\mathbf{z}$,

$$\begin{aligned} \frac{\partial \mathbf{o}(n)}{\partial \mathbf{b}_1} &= \frac{\partial \mathbf{q}(n)}{\partial \mathbf{b}_1} \frac{\partial \mathbf{u}(n)}{\partial \mathbf{q}(n)} \frac{\partial \mathbf{o}(n)}{\partial \mathbf{u}(n)} \\ &= \text{diag}\big\{\mathbf{u}(n) \circ \big(\mathbf{1} - \mathbf{u}(n)\big)\big\} \cdot \big(\mathbf{W}_2^{(n+1)}\big)^{\text{T}} \end{aligned} \tag{10}$$

where diag$\{\mathbf{z}\}$ denotes a diagonal matrix with diagonal vector $\mathbf{z}$. Next,

$$\begin{aligned} \frac{\partial o_k(n)}{\partial \mathbf{W}_1} &= \sum_j \frac{\partial o_k(n)}{\partial u_j(n)} \frac{\partial u_j(n)}{\partial q_j(n)} \frac{\partial q_j(n)}{\partial \mathbf{W}_1} \\ &= \sum_j w_{2,k,j}^{(n+1)} u_j(n)(1 - u_j(n)) \frac{\partial q_j(n)}{\partial \mathbf{W}_1} \end{aligned} \tag{11}$$

where $w_{2,k,j}^{(n+1)}$ is the k^{th} row and j^{th} column entry of $\mathbf{W}_2^{(n+1)}$. For $\frac{\partial q_j(n)}{\partial \mathbf{W}_1}$, its j^{th} row is $\mathbf{x}(n)^{\text{T}}$ and is zero otherwise. Eq. (11) can be simplified as

$$\frac{\partial o_k(n)}{\partial \mathbf{W}_1} = \Big(\big(\mathbf{w}_{2,k}^{(n+1)}\big)^{\text{T}} \circ \mathbf{u}(n) \circ \big(\mathbf{1} - \mathbf{u}(n)\big)\Big)\mathbf{x}(n)^{\text{T}} \tag{12}$$

where $\mathbf{w}_{2,k}^{(n+1)}$ is the k^{th} row of $\mathbf{W}_2^{(n+1)}$. Since the parameters are updated group by group starting from the output layer back to the input layer, this algorithm is called back-propagation (BP) algorithm (Not to be confused with the digital back-propagation (DBP) algorithm for fiber NLC). The weights and biases are continuously updated until convergence.

For the learning and performance evaluation of an ANN, the data sets are typically divided into three groups: training, validation and testing. The training data set is used to train the ANN. Clearly, a larger training data set is better since the more data an ANN sees, the more likely it is that it has encountered examples of all possible types of input. However, the learning time also increases with the training data size. There is no fixed rule for determining the minimum amount of training data needed since it often depends on the given problem. A rule of thumb typically used is that the size of the training data should be at least 10 times the total number of weights [1]. The purpose of the validation data set is to keep a check on how well the ANN is doing as it learns since during training there is an inherent danger of *over-fitting* (or *over-training*). In this case, instead of finding the underlying general decision boundaries as shown in Fig. 6(a), the ANN tends to perfectly fit the training data (including any noise components of them) as shown in Fig. 6(b). This in turn makes the ANN customized for a few data points and reduces its generalization capability, i.e., its ability to make predictions about new inputs which it has never seen before. The overfitting problem can be avoided by constantly examining ANN's error performance during the course of training against an independent validation data set and enforcing an early termination of the training process if the validation data set gives large errors. Typically, the size of the

validation data set is just a fraction (~ 1/3) of that of training data set. Finally, the testing data set evaluates the performance of the trained ANN. Note that an ANN may also be subjected to *under-fitting* problem which occurs when it is under-trained and thus unable to perform at an acceptable level as shown in Fig. 6(c). Under-fitting can again lead to poor ANN generalization. The reasons for under-fitting include insufficient training time or number of iterations, inappropriate choice of activation functions, and/or insufficient number of hidden neurons used.

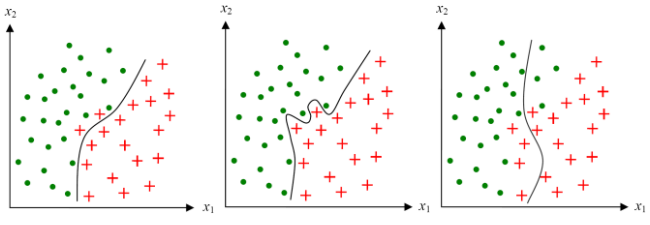


Fig. 6. Example illustrating ANN learning processes with (a) no over-fitting or under-fitting, (b) over-fitting, and (c) under-fitting.

It should be noted that given an adequate number of hidden neurons, proper nonlinearities, and appropriate training, an ANN with one hidden layer has great expressive power and can approximate any continuous function in principle. This is called the *universal approximation theorem* [10]. One can intuitively appreciate this characteristic by considering the classification problem in Fig. 7. Since each hidden neuron can be represented as a straight-line decision boundary, any arbitrary curved boundary can be approximated by a collection of hidden neurons in a single hidden layer ANN. This important property of an ANN enables it to be applied in many diverse applications.

B. Choice of Activation Functions

The choice of activation functions has a significant effect on the training dynamics and final ANN performance. Historically, sigmoid and hyperbolic tangent have been the most commonly-used nonlinear activation functions for hidden layer neurons. However, the rectified linear unit (ReLU) activation function

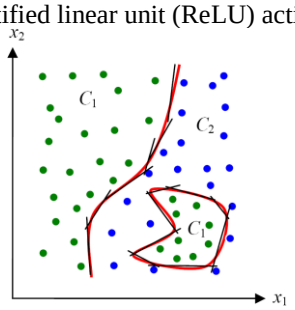
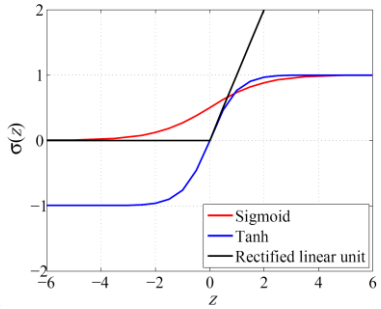


Fig. 7. Decision boundaries for appropriate data classification obtained using an ANN.

has become the default choice among ML community in recent years. The above-mentioned three functions are given by

$$\begin{aligned} \text{Sigmoid: } \sigma(z) &= \frac{1}{1+e^{-z}} \\ \text{Hyperbolic tangent: } \sigma(z) &= \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}} \\ \text{Rectified linear unit: } \sigma(z) &= \max(0,z) \end{aligned} \tag{13}$$



and their plots are shown in Fig. 8. Sigmoid and hyperbolic tangent are both differentiable. However, a major problem with these functions is that their gradients tend to zero as $|z|$ becomes large and thus the activation output gets saturated. In this case, the weights and biases updates for a certain layer will be minimal, which in turn will slow down the weights and biases updates for all the preceding layers. This is known as *vanishing gradient problem* and is particularly an issue when training ANNs with large number of hidden layers. To circumvent this problem, ReLU was proposed since its gradient does not vanish as z increases. Note that although ReLU is not differentiable at $z = 0$, it is not a problem in practice since the probability of having an entry exactly equal to 0 is generally very low. Also, as the ReLU function and its derivative are 0 for $z < 0$, around 50% of hidden neurons' outputs will be 0, i.e., only half of total neurons will be active when the ANN weights and biases are randomly initialized. It has been found that such sparsity of activation not only reduces computational complexity (and thus training time) but also leads to better ANN performance [11]. Note that while using the ReLU activation function, the ANN weights and biases are often initialized using the method proposed by He *et al.* [12]. On the other hand, the Xavier initialization technique [13] is more commonly employed for the hyperbolic tangent activation function. These heuristics-based approaches initialize the weights and biases by drawing random numbers from a truncated normal distribution (instead of a standard normal distribution) with variance which depends on the size of the previous ANN layer.

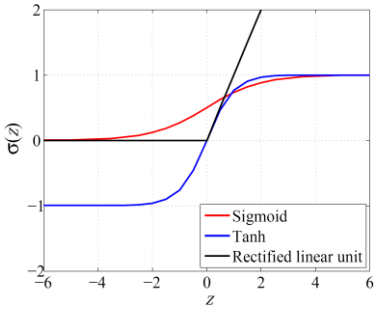


Fig. 8. Common activation functions used in ANNs.

C. Choice of Loss Functions

The choice of loss function E has a considerable effect on the performance of an ANN. The MSE is a common choice in adaptive signal processing and other DSP in telecommunications. For regression problems, MSE works well in general and is also easy to compute. On the other hand, for classification problems, the cross-entropy loss function defined as

$$E = -\frac{1}{L}\sum_{l=1}^{L}\sum_{k=1}^{K} y_k(l)log(o_k(l)) \tag{14}$$

is often used instead of the MSE [10]. The cross-entropy function can be interpreted by viewing the softmax output $\mathbf{o}(l)$ and the class label with one-hot encoding $\mathbf{y}(l)$ as probability distributions. In this case, $\mathbf{y}(l)$ has zero entropy and one can subtract the zero-entropy term from Eq. (14) to obtain

$$\begin{aligned} E &= -\frac{1}{L}\sum_{l=1}^{L}\sum_{k=1}^{K} y_k(l)log\big(o_k(l)\big) + \underbrace{\frac{1}{L}\sum_{l=1}^{L}\sum_{k=1}^{K} y_k(l)log\big(y_k(l)\big)}_{=0} \\ &= \frac{1}{L}\sum_{l=1}^{L}\sum_{k=1}^{K} y_k(l)log\left(\frac{y_k(l)}{o_k(l)}\right) \end{aligned} \tag{15}$$

which is simply the Kullback-Leibler (KL) divergence between the distributions $\mathbf{o}(l)$ and $\mathbf{y}(l)$ averaged over all input-output pairs. Therefore, the cross-entropy is in fact a measure of the similarity between ANN outputs and the class labels. The cross-entropy function also leads to simple gradient updates as the logarithm cancels out the exponential operation inherent in the softmax calculation, thus leading to faster ANN training. The Appendix shows the derivation of BP algorithm for the single hidden layer ANN in

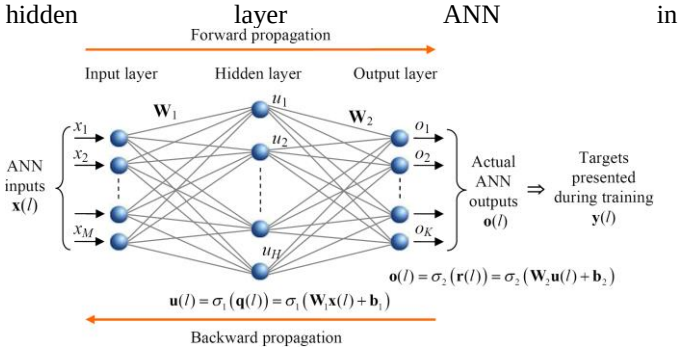


Fig. 5 with cross-entropy loss function and softmax activation function for the output layer neurons.

In many applications, a common approach to prevent over-fitting is to reduce the magnitude of the weights as large weights produce high curvatures which make the decision boundaries overly complicated. This can be achieved by including an extra regularization term in the loss function, i.e.,

$$E' = E + \lambda\|\mathbf{W}\|^2 \tag{16}$$

where $\|\mathbf{W}\|^2$ is the sum of squared element-wise weights. The parameter λ, called regularization coefficient, defines the relative importance of the training error E and the regularization term. The regularization term thus discourages weights from reaching large values and this often results in significant improvement in ANN's generalization ability [14].

D. Support Vector Machines (SVMs)

In many classification tasks, it often happens that the two data categories are not easily separable with straight lines or planes in the original variable space. SVM is an ML technique that preprocesses the input data $\mathbf{x}(i)$ and transforms it into (sometimes) a higher-dimensional space $\mathbf{v}(i) = \varphi\big(\mathbf{x}(i)\big)$, called *feature space*, where the data belonging to two different classes can be separated easily by a simple straight plane decision boundary or *hyperplane* [15]. An example is shown in

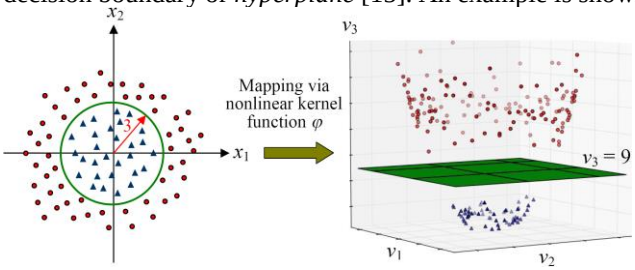


Fig. 9 where one class of data lies within a circle of radius 3 and the other class lies outside. When transformed into the feature

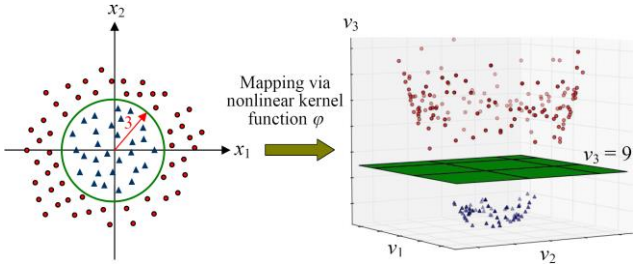
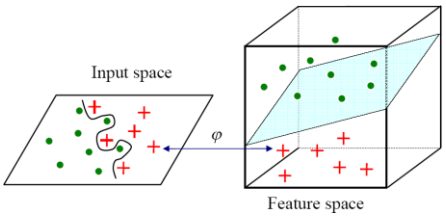


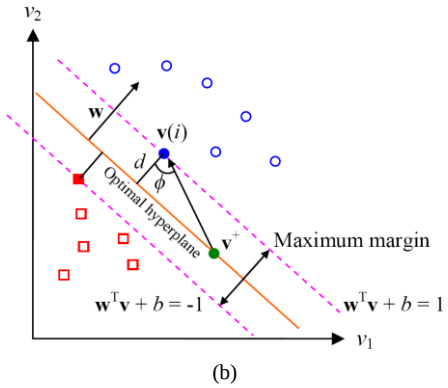
Fig. 9. Example showing how a linearly inseparable problem (in the original 2D data space) can undergo a nonlinear transformation and becomes a linearly separable one in the 3-dimensional (3D) feature space.

space $\mathbf{v} = (v_1, v_2, v_3) = (x_1, x_2, x_1^2 + x_2^2)$, the two data classes can be separated simply by the hyperplane $v_3 = 9$.

Let us first focus on finding the right decision hyperplane after the transformation into feature space as shown in Fig. 10(a). The right hyperplane should have the largest (and also equal) distance from the borderline points of the two data classes. This is graphically illustrated in Fig. 10(b). Had the data points been generated from two probability density functions (PDFs), finding a hyperplane with maximal margin from the borderline points is conceptually analogous to finding a maximum likelihood decision boundary. The borderline points, represented as solid dot and square in Fig. 10(b), are referred to as *support vectors* and are often most informative for the classification task.



(a)



(b)

Fig. 10. (a) Maping from input space to a higher-dimensional feature space using a nonlinear kernel function φ. (b) Separation of two data classes in the feature space through an optimal hyperplane.

More technically, in the feature space, a general hyperplane is defined as $\mathbf{w}^T\mathbf{v} + b = 0$. If it classifies all the data points correctly, all the blue points will lie in the region $\mathbf{w}^T\mathbf{v} + b > 0$ and the red points will lie in the region $\mathbf{w}^T\mathbf{v} + b < 0$. We seek to find a hyperplane $\mathbf{w}^T\mathbf{v} + b = 0$ that maximizes the margin d as shown in

Fig. 10(b). Without loss of generality, let the point $\mathbf{v}(i)$ reside on the hyperplane $\mathbf{w}^T\mathbf{v} + b = 1$ and is closest to the hyperplane $\mathbf{w}^T\mathbf{v} + b = 0$ on which $\mathbf{v}^+$ resides. Since the vectors $\mathbf{v}(i) - \mathbf{v}^+, \mathbf{w}$ and the angle ϕ are related by $cos\phi = \mathbf{w}^T(\mathbf{v}(i) - \mathbf{v}^+)/(\|\mathbf{w}\|\|\mathbf{v}(i) - \mathbf{v}^+\|)$, the margin d is given as

$$\begin{aligned} d &= \|\mathbf{v}(i) - \mathbf{v}^+\| cos\phi \\ &= \|\mathbf{v}(i) - \mathbf{v}^+\| \cdot \frac{\mathbf{w}^T(\mathbf{v}(i) - \mathbf{v}^+)}{\|\mathbf{w}\|\|\mathbf{v}(i) - \mathbf{v}^+\|} \\ &= \frac{\mathbf{w}^T(\mathbf{v}(i) - \mathbf{v}^+)}{\|\mathbf{w}\|} = \frac{\mathbf{w}^T\mathbf{v}(i) - \mathbf{w}^T\mathbf{v}^+}{\|\mathbf{w}\|} \\ &= \frac{\mathbf{w}^T\mathbf{v}(i) + b}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}. \end{aligned} \tag{17}$$

Therefore, we seek to find $\mathbf{w}, b$ that maximize $1/\|\mathbf{w}\|$ subject to the fact that all the data points are classified correctly. To characterize the constraints more mathematically, one can first assign the blue class label to 1 and red class label to -1. In this case, if we have correct decisions for all the data points, the product $y(i)(\mathbf{w}^T\mathbf{v}(i) + b)$ will always be greater than 1 for all i. The optimization problem then becomes

$$\begin{aligned} &\underset{\mathbf{w},b}{\operatorname{argmin}} \frac{1}{\|\mathbf{w}\|} \\ &\text{subject to } y(l)(\mathbf{w}^T\mathbf{v}(l) + b) \geq 1, \quad l = 1, 2, \ldots, L \end{aligned} \tag{18}$$

and thus standard convex programming software packages such as CVXOPT [16] can be used to solve Eq. (18).

Let us come back to the task of choosing the nonlinear function $\varphi(\cdot)$ that maps the original input space $\mathbf{x}$ to feature space $\mathbf{v}$. For SVM, one would instead find a kernel function $K(\mathbf{x}(i), \mathbf{x}(j)) = \varphi(\mathbf{x}(i)) \cdot \varphi(\mathbf{x}(j)) = \mathbf{v}(i)^T\mathbf{v}(j)$ that maps to the inner product. Typical kernel functions include:

- Polynomials: $K(\mathbf{x}(i), \mathbf{x}(j)) = (\mathbf{x}(i)^T\mathbf{x}(j) + a)^b$ for some scalars a, b
- Gaussian radial basis function: $K(\mathbf{x}(i), \mathbf{x}(j)) = \exp(-a\|\mathbf{x}(i) - \mathbf{x}(j)\|^2)$ for some scalar a
- Hyperbolic tangent: $K(\mathbf{x}(i), \mathbf{x}(j)) = \tanh(a\mathbf{x}(i)^T\mathbf{x}(j) + b)$ for some scalars a, b.

The choice of a kernel function is often determined by the designer's knowledge of the problem domain [3]. Note that a larger separation margin typically results in better generalization of the SVM classifier. SVMs often demonstrate better generalization performance than conventional ANNs in various pattern recognition applications. Furthermore, multiple SVMs can be applied to the same data set to realize non-binary classifications such as detecting 16-QAM signals [17][18][19] (to be discussed in more detail in Section V).

It should be noted that ANNs and SVMs can be seen as two complementary approaches for solving classification problems. While an ANN derives curved decision boundaries in the input variable space, the SVM performs nonlinear transformations of the input variables followed by determining a simple decision boundary or hyperplane as shown in

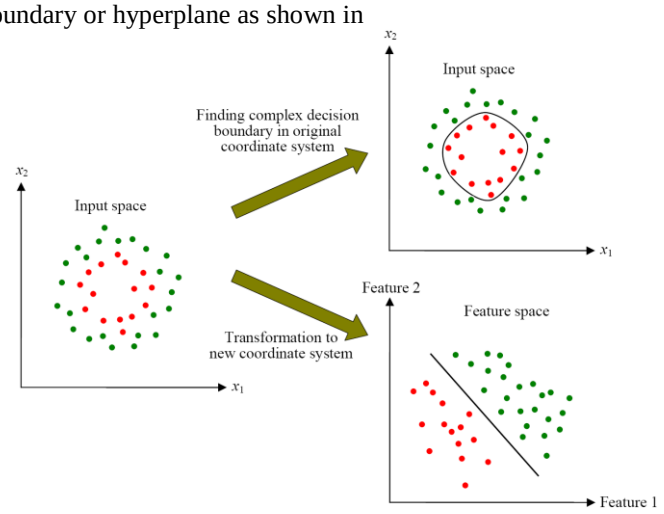


Fig. 11.

III. Unsupervised and Reinforcement Learning

The ANN and SVM are examples of *supervised learning* approach in which the class labels $\mathbf{y}$ of the training data are known. Based on this data, the ML algorithm generalizes to

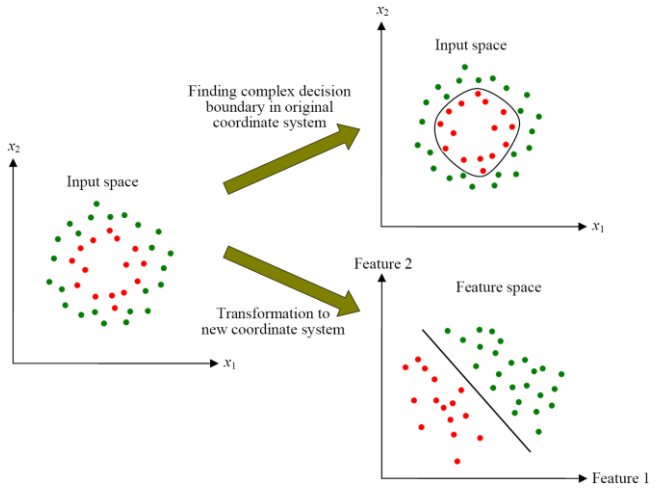


Fig. 11. Example showing how an ANN determines a curved decision boundary in the original input space while an SVM obtains a simple decision boundary in the transformed feature space.

react accurately to new data to the best possible extent. Supervised learning can be considered as a closed-loop feedback system as the error between the ML algorithm's actual outputs and the targets is used as a feedback signal to guide the learning process.

In *unsupervised learning*, the ML algorithm is not provided with correct labels of the training data. Rather, it learns to identify similarities between various inputs with the aim to either categorize together those inputs which have something in common or to determine some better representation/description of the original input data. It is referred to as "unsupervised" because the ML algorithm is not told what the output should be rather it has to come up with it itself [20]. One example of unsupervised learning is data clustering as shown in

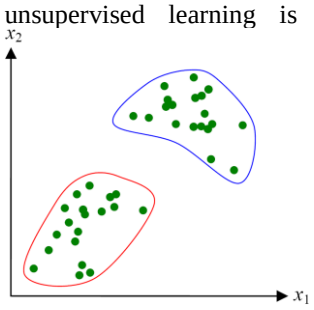


Fig. 12.

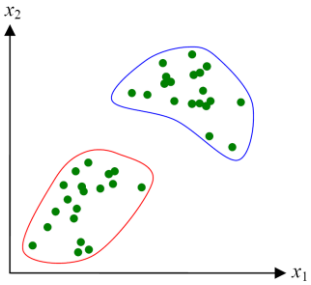


Fig. 12. Data clustering based on unsupervised learning.

Unsupervised learning is becoming more and more important because in many real circumstances it is practically not possible to obtain labeled training data. In such scenarios, an unsupervised learning algorithm can be applied to discover some similarities between different inputs for itself. Unsupervised learning is typically used in tasks such as clustering, vector quantization, dimensionality reduction, and features extraction. It is also often employed as a preprocessing tool for extracting useful (in some particular context) features of the raw data before supervised learning algorithms can be applied. We hereby provide a review of few key unsupervised learning techniques.

A. K-means Clustering

Let $\{\mathbf{x}(1), \mathbf{x}(2), \ldots \mathbf{x}(L)\}$ be the set of data points which is to be split into K clusters $C_1, C_2, \ldots C_K$. *K*-means clustering is an iterative unsupervised learning algorithm which aims to partition L observations into K clusters such that the sum of squared errors for data points within a group is minimized [14]. An example of this algorithm is graphically shown in

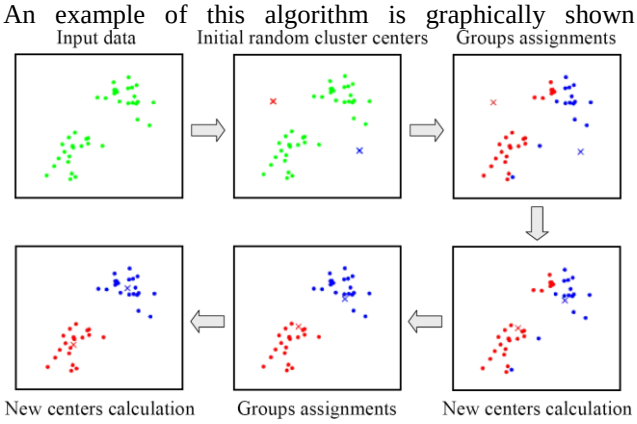


Fig. 13. The algorithm initializes by randomly picking K locations $\boldsymbol{\mu}(j)$, $j = 1, 2, \ldots, K$ as cluster centers. This is followed by two iterative steps. In the first step, each data point $\mathbf{x}(i)$ is assigned to the cluster C_k with the minimum Euclidean distance, i.e.,

$$C_k = \{\mathbf{x}(i): \|\mathbf{x}(i) - \boldsymbol{\mu}(k)\| < \|\mathbf{x}(i) - \boldsymbol{\mu}(j)\| \quad \forall\, j \in \{1, 2, \ldots, K\} \backslash \{k\}\} \tag{19}$$

In the second step, the new center of each cluster C_k is calculated by averaging out the locations of data points that are assigned to cluster C_k, i.e.,

$$\boldsymbol{\mu}(k) = \sum_{\mathbf{x}(i) \in C_k} \mathbf{x}(i) \tag{20}$$

The two steps are repeated iteratively until the cluster centers converge. Several variants of *K*-means algorithm have been proposed over the years to improve its computational efficiency as well as to achieve smaller errors. These include fuzzy *K*-means, hierarchical *K*-means, *K*-means++, *K*-medians, *K*-medoids, etc.

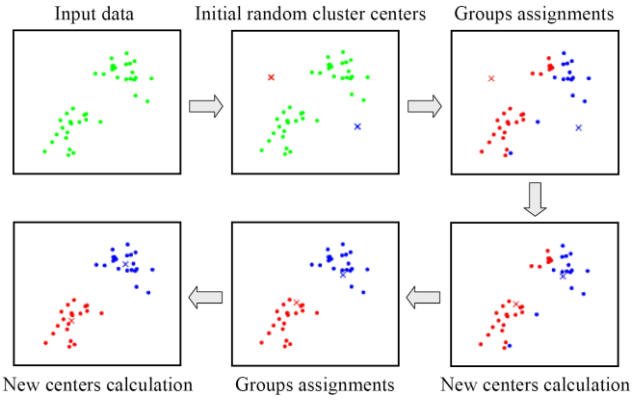


Fig. 13. Example to illustarte initialization and two iterations of *K*-means algorithm. The data points are shown as dots and cluster centers are depicted as crosses.

B. Expectation-Maximization (EM) Algorithm

One drawback of *K*-means algorithm is that it requires the use of hard decision boundaries whereby a data point can only be assigned to one cluster even though it might lie somewhere midway between two or more clusters. The EM algorithm is an improved clustering technique which assigns a probability to

the data point belonging to each cluster rather than forcing it to belong to one particular cluster during each iteration [20]. The algorithm assumes that a given data distribution can be modeled as a superposition of K jointly Gaussian probability distributions with distinct means and covariance matrices $\boldsymbol{\mu}(k), \boldsymbol{\Sigma}(k)$ (also referred to as *Gaussian mixture models*). The EM algorithm is a two-step iterative procedure comprising of expectation (E) and maximization (M) steps [3]. The E step computes the a posteriori probability of the class label given each data point using the current means and covariance matrices of the Gaussians, i.e.,

$$\begin{aligned} p_{ij} &= p\big(C_j|\mathbf{x}(i)\big) \\ &= \frac{p\big(\mathbf{x}(i)|C_j\big)p\big(C_j\big)}{\sum_{k=1}^{K} p(\mathbf{x}(i)|C_k)p(C_k)} \\ &= \frac{\boldsymbol{N}(\mathbf{x}(i)|\boldsymbol{\mu}(k), \boldsymbol{\Sigma}(k))}{\sum_{k=1}^{K} \boldsymbol{N}(\mathbf{x}(i)|\boldsymbol{\mu}(k), \boldsymbol{\Sigma}(k))} \end{aligned} \tag{21}$$

where $\boldsymbol{N}(\mathbf{x}(i)|\boldsymbol{\mu}(k), \boldsymbol{\Sigma}(k))$ is the Gaussian PDF with mean and covariance matrix $\boldsymbol{\mu}(k), \boldsymbol{\Sigma}(k)$. Note that we have inherently assumed equal probability $p(C_j)$ of each class, which is a valid assumption for most communication signals. In scenarios where this assumption is not valid, e.g., the one involving probabilistic constellation shaping (PCS), the actual non-uniform probabilities $p\big(C_j\big)$ of individual symbols shall instead be used in Eq. (21). The M step attempts to update the means and covariance matrices according to the updated soft-labelling of the data points, i.e.,

$$\begin{aligned} \boldsymbol{\mu}(j) &= \frac{\sum_{i=1}^{L} p_{ij}\mathbf{x}(i)}{\sum_{i=1}^{L} p_{ij}} \\ \boldsymbol{\Sigma}(k) &= \sum_{i=1}^{L} p_{ij}(\mathbf{x}(i) - \boldsymbol{\mu}(j))(\mathbf{x}(i) - \boldsymbol{\mu}(j))^{\mathrm{T}} \end{aligned} \tag{22}$$

A graphical illustration of EM algorithm and its convergence process is shown in

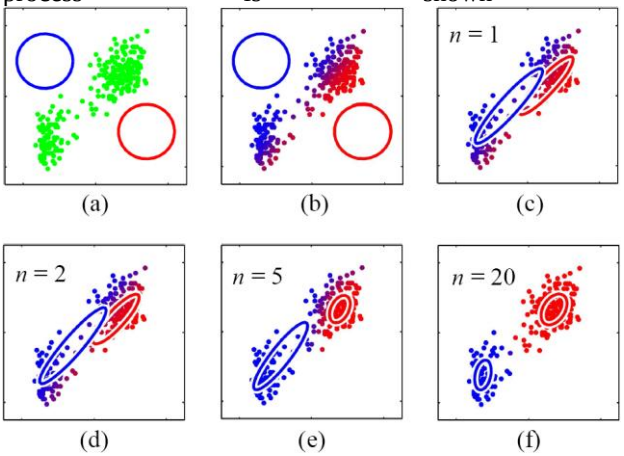


Fig. 14. Fig. 14(a) shows the original data points in green which are to be split into two clusters by applying EM algorithm. The two Gaussian probability distributions are initialized with random means and unit covariance matrices and are depicted using red and blue circles. The results after first E step are shown in Fig. 14(b) where the posterior probabilities in Eq. (21) are expressed by the proportion of red and blue colors for each data point. Fig. 14(c) depicts the results after first M step where the means and covariance matrices of the red and blue Gaussian distributions are updated using Eq. (22), which in turn uses the posterior probabilities computed by Eq. (21). This completes the 1st iteration of the EM algorithm. Fig. 14(d) to (f) show the results after 2, 5 and 20 complete EM iterations, respectively, where the convergence of the algorithm and consequently effective splitting of the data points into two clusters can be clearly observed.

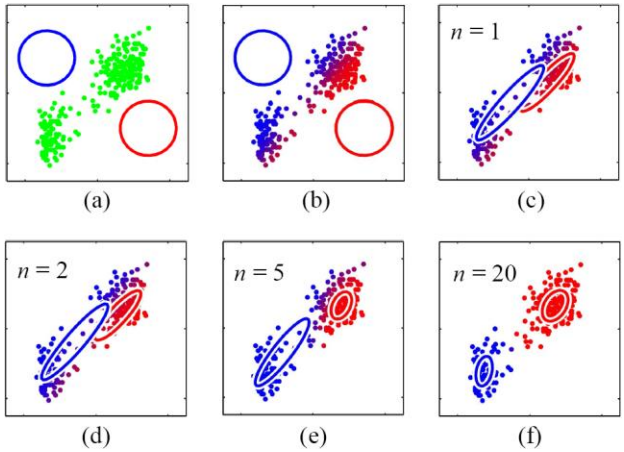


Fig. 14. Example showing the concept of EM algorithm. (a) Original data points and initialization. Results after (b) first E step; (c) first M step; (d) 2 complete EM iterations; (e) 5 complete EM iteratons; and (f) 20 complete EM iterations [3].

C. Principal Component Analysis (PCA)

Principal component analysis is an unsupervised learning technique for features extraction and data representation [21][22]. PCA is often used as a preprocessing tool in many pattern recognition applications for the extraction of limited but most critical data features. The central idea behind PCA is to project the original high-dimensional data onto a lower-dimensional feature space that retains most of the information in the original data as shown in

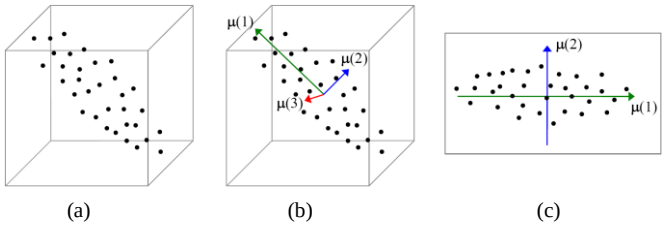


Fig. 15. The reduced dimensionality feature space is spanned by a small (but most significant) set of orthonormal eigenvectors, called principal components (PCs). The first PC points in the direction along which the original data has the greatest variability and each successive PC in turn accounts for as much of the remaining variability as possible. Geometrically, we can think of PCA as a rotation of the axes of the original coordinate system to a new set of orthogonal axes which are ordered based on the amount of variation of the original data they account for, thus achieving dimensionality reduction.

Fig. 15. Example to illustarte the concept of PCA. (a) Data points in the original 3D data space; (b) Three PCs ordered according to the variability in original data; (c) Projection of data points onto a plane defined by the first two PCs while discarding the third one.

More technically, consider a data set $\{\mathbf{x}(1), \mathbf{x}(2), \ldots \mathbf{x}(L)\}$ with L data vectors of M dimensions. We will first compute the mean vector $\bar{\mathbf{x}} = \frac{1}{L}\sum_{i=1}^{L}\mathbf{x}(i)$ and the covariance matrix $\mathbf{\Sigma}$ can then be estimated as

$$\mathbf{\Sigma} \approx \frac{1}{L}\sum_{i=1}^{L}(\mathbf{x}(i)-\bar{\mathbf{x}})(\mathbf{x}(i)-\bar{\mathbf{x}})^{\mathrm{T}} \tag{23}$$

where $\mathbf{\Sigma}$ can have up to M eigenvectors $\boldsymbol{\mu}(i)$ and corresponding eigenvalues λ_i. We then sort the eigenvalues in terms of their magnitude from large to small and choose the first S (where $S << M$) corresponding eigenvectors such that

$$\sum_{i=1}^{S}\lambda_i \Big/ \sum_{i=1}^{M}\lambda_i > R \tag{24}$$

where R is typically above 0.9 [22]. Note that, as compared to the original M-dimensional data space, the chosen eigenvectors span only an S-dimensional subspace that in a way captures most of the data information. One can understand such procedure intuitively by noting that for a covariance matrix, finding the eigenvectors with large eigenvalues corresponds to finding linear combinations or particular directions of the input space that give large variances, which is exactly what we want to capture. A data vector $\mathbf{x}$ can then be approximated as a weighted-sum of the chosen eigenvectors in this subspace, i.e.,

$$\mathbf{x} \approx \sum_{i=1}^{S} w_i \boldsymbol{\mu}(i) \tag{25}$$

where $\boldsymbol{\mu}(i), i = 1, 2, \ldots, S$ are the chosen orthogonal eigenvectors such that

$$\boldsymbol{\mu}^{\mathrm{T}}(m)\boldsymbol{\mu}(l) = \begin{cases} 1 & if \;\; l = m \\ 0 & if \;\; l \neq m \end{cases} \tag{26}$$

Multiplying both sides of Eq. (25) with $\boldsymbol{\mu}^{\mathrm{T}}(k)$ and then using Eq. (26), we get

$$w_k = \boldsymbol{\mu}^{\mathrm{T}}(k)\mathbf{x}, \quad k = 1, 2, \ldots, S \tag{27}$$

The vector $\mathbf{w} = [w_1 \; w_2 \ldots w_S]^{\mathrm{T}}$ of weights describing the contribution of each chosen eigenvector $\boldsymbol{\mu}(k)$ in representing $\mathbf{x}$ can then be considered as a feature vector of $\mathbf{x}$.

D. Independent Component Analysis (ICA)

Another interesting technique for features extraction and data representation is ICA. Unlike PCA which uses orthogonal and uncorrelated components, the components in ICA are instead required to be statistically independent [1]. In other words, ICA seeks those directions in the feature space that are most independent from each other.

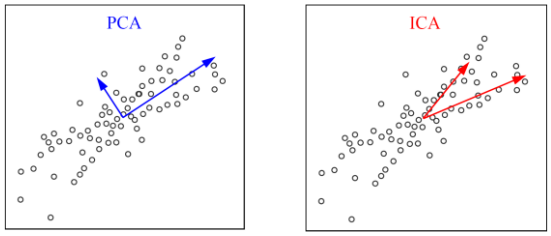


Fig. 16 illustrates the conceptual difference between PCA and ICA. Finding the independent components (ICs) of the observed data can be useful in scenarios where we need to separate mutually independent but unknown source signals from their linear mixtures with no information about the mixing coefficients. An example is the task of polarization demultiplexing at the receiver using DSP. For a data set $\{\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(L)\}$, one seeks to identify a collection of basis vectors $\mathbf{v}(1), \mathbf{v}(2), \ldots \mathbf{v}(S)$ so that $\mathbf{x} \approx \sum_{k=1}^{S} w_k \mathbf{v}(k)$ and the empirical distributions of $w_k, k = 1, 2, \ldots, S$ across all the data $\mathbf{x}$ are statistically independent. This can be achieved by minimizing the mutual information between different w_k.

ICA is used as a preprocessing tool for extracting data features in many pattern recognition applications and is shown to outperform conventional PCA in many cases [23]. This is expected because unlike PCA which is derived from second-order statistics (i.e., covariance matrix) of the input data, ICA takes into account high-order statistics of the data as it considers complete probability distribution.

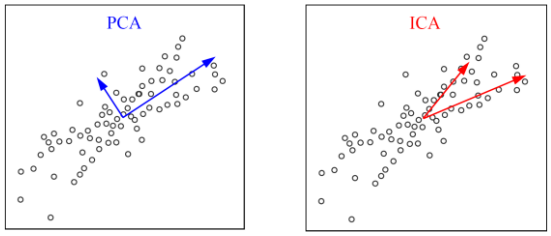


Fig. 16. Example 2D data fitted using (a) PCs bases and (b) ICs bases. As shown, the orthogonal basis vectors in PCA may not be efficient while representing non-orthogonal density distributions. In contrast, ICA does not necessitate orthogonal basis vectors and can thus represent general types of densities more effectively.

We would like to highlight here that the dimensionality of the transformed space in ML techniques can be higher or lower than the original input space depending upon the nature of the problem at hand. If the objective of the transformation is to simply reduce the input data dimensionality (e.g., for decreasing the computational complexity of the learning system) then the dimensionality of the transformed space should be lower than that of original one. On the other hand, a transformation to a higher-dimensional space may be desirable if the data classes can be separated more easily by a classifier in the new space.

E. Reinforcement Learning (RL)

In this learning type, the input of the ML model (called observation) is associated with a reward or reinforcement signal. The output (called action) determines the value of the next observation and hence the reward through the predefined action-observation relationship of a particular problem. The objective here is to learn a sequence of actions that optimizes the final reward. However, unlike supervised learning, the model is not optimized through SGD-like approaches. Rather, the model tries different actions until it finds a set of parameters that lead to better rewards. In RL, the model is rewarded for its good output result and punished for the bad one. In this way, it can learn to choose actions which can maximize the expected reward [24]. Like supervised learning, RL can also be regarded as a closed-loop feedback system since the RL model's actions will influence its later inputs. RL is particularly useful in solving interactive problems in which it is often impossible to attain examples of desired behavior which are not only correct

but are also representative of all the possible situations in which the model may have to act ultimately. In an uncharted territory, an RL model should be able to learn from its own experiences instead of getting trained by an external supervisor with a training data set of labeled examples.

Due to their inherent self-learning and adaptability characteristics, RL algorithms have been considered for various tasks in optical networks including network self-configuration, adaptive resource allocation, etc. In these applications, the actions performed by the RL algorithms may include choosing spectrum or modulation format, rerouting data traffic, etc., while the reward may be the maximization of network throughput, minimization of latency or packet loss rate, etc. (to be discussed in more detail in Section V). Currently, there are limited applications of RL in the physical layer of optical communication systems. This is because in most cases the reward (objective function) can be explicitly expressed as a continuous and differentiable function of the actions. An example is the CMA algorithm where the actions are the filter tap weights and the objective is to produce output signals with a desired amplitude. For such optimization procedures, we simply refer to them as adaptive signal processing instead of RL.

IV. Deep Learning Techniques

A. Deep Learning vs. Conventional Machine Learning

The recent emergence of DL technologies has taken ML research to a whole new level. DL algorithms have demonstrated comparable or better performance than humans in a lot of important tasks including image recognition, speech recognition, natural language processing, information retrieval, etc. [2][25]. Loosely speaking, DL systems consist of multiple layers of nonlinear processing units (thus deeper architectures) and may even contain complex structures such as feedback and memory. DL then refers to learning the parameters of these architectures for performing various pattern recognition tasks.

One way to interpret DL algorithms is that they automatically learn and extract higher-level features of data from lower-level ones as the input propagates through various layers of nonlinear processing units, resulting in a hierarchical representation of data. For example, while performing a complex human face recognition task using a DL-based multilayer ANN (called DNN), the first layer might learn to detect edges, the second layer can learn to recognize more complex shapes such as circles or squares which are built from the edges. The third layer may then recognize even more complex combinations and arrangements of shapes such as the location of two ovals and a triangle in between, which in turn starts to resemble parts of a human face with two eyes and a nose. Such an ability to automatically discover and learn features at increasingly high levels of abstraction empowers DL systems to learn complex relationships between inputs and outputs directly from the data instead of using human-crafted features.

As an example of this notion of hierarchical learning, Fig. 17 shows the use of a DNN as well as a conventional ANN on signal's eye-diagrams to monitor OSNR. In the first approach, the eye-diagrams are directly applied as images at the input of the DNN, as shown in Fig. 17(a), and it is made to automatically learn and discover OSNR-sensitive features without any human intervention. The extracted features are subsequently exploited by DNN for OSNR monitoring. In contrast, with conventional ANNs, prior knowledge in optical communications is utilized in choosing suitable features for the task, e.g., the variances of "1" and "0" levels and eye-opening can be indicative of OSNR. Therefore, these useful features are manually extracted from the eye-diagrams and are then used as inputs to an ANN for the estimation of OSNR as shown in Fig. 17(b). For completeness, Fig. 17(c) shows an analytical and non-ML approach to determine OSNR by finding the powers and noise variances that best fit the noise distributions of "1" and "0" levels knowing that they follow Rician distribution. In this case, a specific

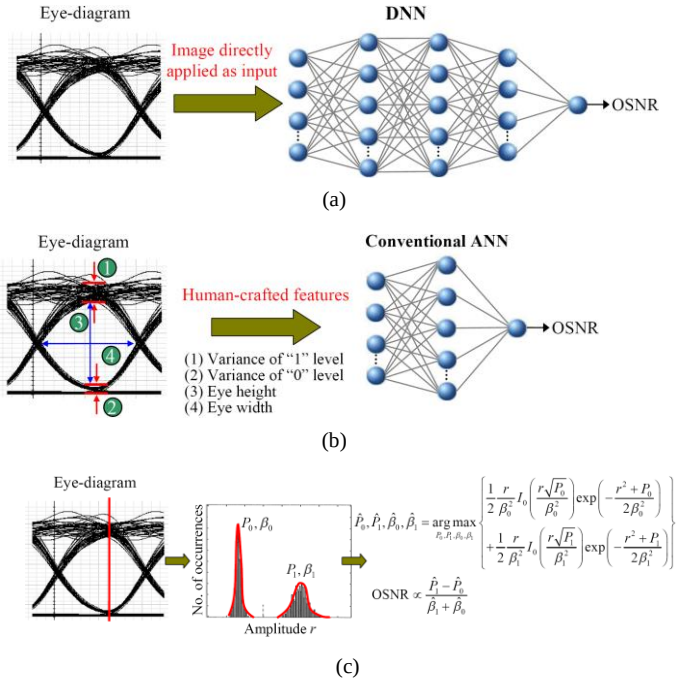


Fig. 17. Example illustrating OSNR monitoring using eye-diagrams' features by applying (a) DNN, (b) conventional ANN, and (c) analytical modeling and parameters fitting.

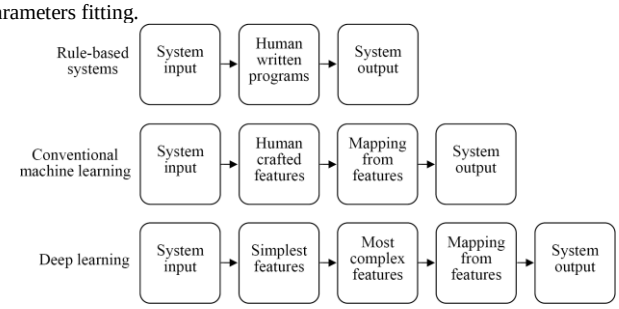


Fig. 18. Conceptual differences between rule-based systems, conventional ML, and DL approaches for pattern recognition.

mathematical formula or computational instruction is pre-coded into the program and there is nothing to learn from the input data.

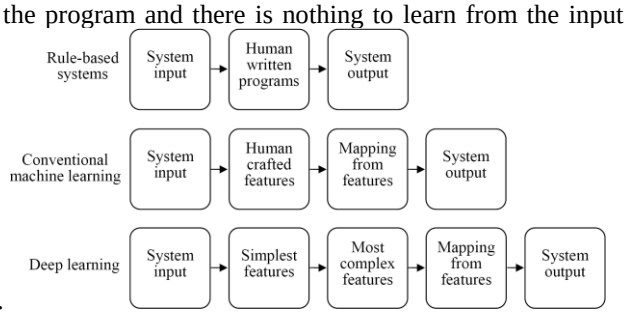


Fig. 18 compares the underpinning philosophies of the three different approaches discussed above. Note that, in principle, there is no hard rule on how many layers are needed for an ML model in a given problem. In practice, it is generally accepted that when more underlying physics/mathematics of the problem is used to identify and extract the suitable data features as inputs, the ML model tends to be simpler.

It should be noted that deep architectures are more efficient or more *expressive* than their shallow counterparts [26]. For example, it has been observed empirically that compared to a shallow neural network, a DNN requires much fewer number of neurons and weights (i.e., around 10 times less connections in speech recognition problems [27]) to achieve the same performance.

A major technical challenge in DL is that the conventional BP algorithm and gradient-based learning methods used for training shallow networks are inherently not effective for training networks with multiple layers due to the *vanishing gradient problem* [2]. In this case, different layers in the network learn at significantly different speeds during the training process, i.e., when the layers close to the output are learning well, the layers close to the input often get stuck. In the worst case, this may completely stop the network from further learning. Several solutions have been proposed to address the vanishing gradient problem in DL systems. These include: (i) choosing specific activation functions such as ReLU [11], as discussed earlier; (ii) pretraining of network one layer at a time in a greedy way and then fine-tuning the entire network through BP algorithm [28]; (iii) using some special architectures such as long short-term memory (LSTM) networks [29]; (iv) applying network optimization approaches which avoid gradients (e.g., global search methods such as genetic algorithm). The choice of a given solution typically depends on the type of DL model being trained and the degree of computational complexity involved.

B. Deep Neural Networks (DNNs)

Unlike shallow ANNs, DNNs contain multiple hidden layers between input and output layers. The structure of a simple three hidden layers DNN is shown in

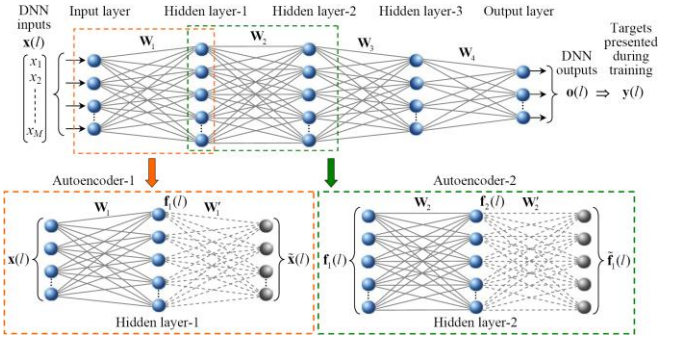


Fig. 19 (top). DNNs can be trained effectively using the BP algorithm. To avoid vanishing gradient problem during training of DNNs, following two approaches are typically adopted. In the first method, the ReLU activation function is simply used for the hidden layers neurons due to its non-saturating nature. In the second approach, a DNN is first pretrained one layer at a time and then the training

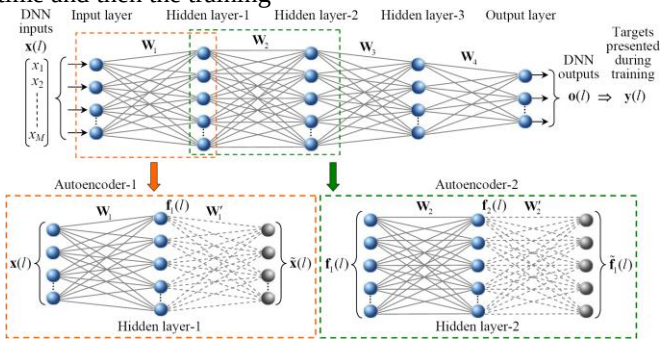


Fig. 19. Schematic diagram of a three hidden layers DNN (top). Two autoencoders used for the pretraining of first two hidden layers of the DNN (bottom). The decoder parts in both autoencoders are shown in grey color with dotted weight lines.

process is fine-tuned using BP algorithm [28]. For pretraining of hidden layers of the DNNs, *autoencoders* are typically employed which are essentially feed-forward neural networks.

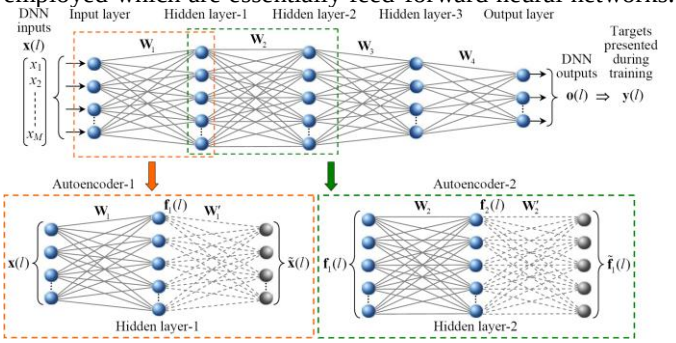


Fig. 19 (bottom) shows two simple autoencoders used for the unsupervised pretraining of first two hidden layers of the DNN. First, hidden layer-1 of the DNN is pretrained in isolation using autoencoder-1 as shown in the figure. The first part of autoencoder-1 (called encoder) maps input vectors **x** to a hidden representation $\mathbf{f}_1$ while the second part (called decoder) reverses this mapping in order to synthesize the initial inputs **x**. Once autoencoder-1 learns these mappings successfully, hidden layer-1 is considered to be pretrained. The original input vectors **x** are then passed through the encoder of autoencoder-1 and the corresponding representations $\mathbf{f}_1$ (also called feature vectors) at the output of pretrained hidden layer-1 are obtained. Next,

vectors $\mathbf{f}_1$ are utilized as inputs for the unsupervised pretraining of hidden layer-2 using autoencoder-2, as depicted in the figure. This procedure is repeated for the pretraining of hidden layer-3 and the corresponding feature vectors $\mathbf{f}_3$ are then used for the supervised pretraining of final output layer by setting the desired outputs $\mathbf{y}$ as targets. After isolated pretraining of hidden and output layers, the complete DNN is trained (i.e., fine-tuned) using BP algorithm with $\mathbf{x}$ and $\mathbf{y}$ as inputs and targets, respectively. By adopting this autoencoders-based hierarchical learning approach, the vanishing gradient problem can be successfully bypassed in DNNs.

C. Convolutional Neural Networks (CNNs)

CNNs are a type of neural network primarily used for pattern recognition within images though they have also been applied in a variety of other areas such as speech recognition, natural language processing, video analysis, etc. The structure of a typical CNN is shown in Fig. 20(a) comprising of a few alternating *convolutional* and *pooling* layers followed by an ANN-like structure towards the end of the network. The convolutional layer consists of neurons whose outputs only depend on the neighboring pixels of the input as opposed to fully-connected (FC) layers in typical ANNs as shown in Fig. 20(b). That is why it is called *local network*, *local connected network* or *local receptive field* in ML literature. The weights are also shared across the neurons in the same layer, i.e., each neuron undergoes the same computation $\mathbf{w}^{\mathrm{T}}(\cdot)+b$ but the input is a different part of the original image. This is followed by a decision-like nonlinear activation function and the output

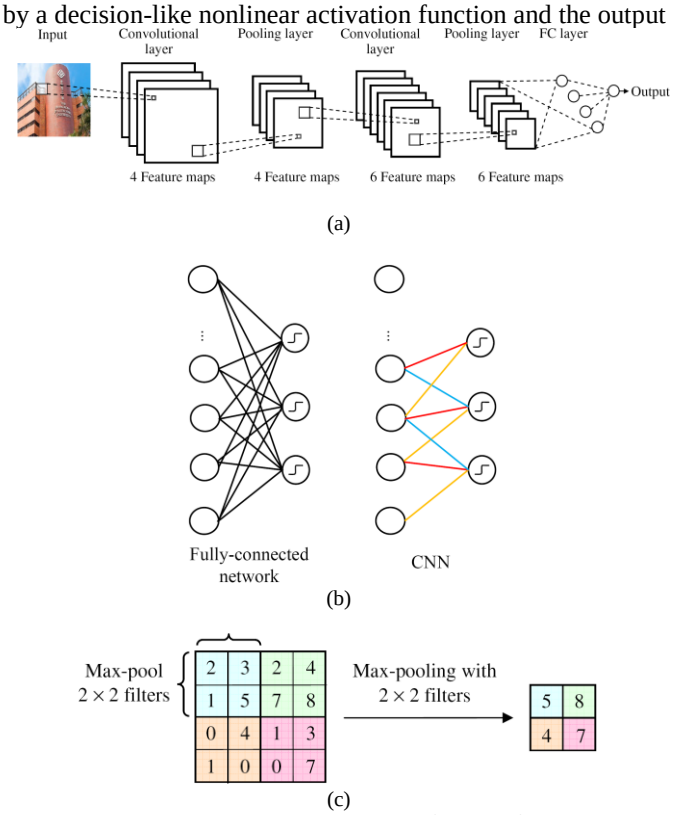


Fig. 20. (a) A simple CNN architecture comprising of two sets of convolutional and pooling layers followed by an FC layer on top. (b) In a CNN, a node in the next layer is connected to a small subset of nodes in the previous layer. The weights (indicated by colors of the edges) are also shared among the nodes. (c) Nonlinear down-sampling of feature maps via a max-pooling layer.

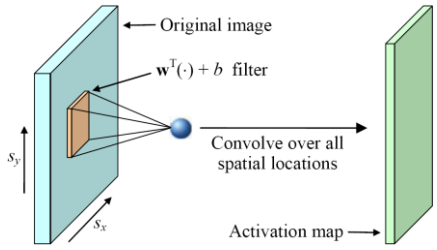


Fig. 21. Convolution followed by an activation function in a CNN. Viewing the $\mathbf{w}^{\mathrm{T}}(\cdot)+b$ operation as *cross-correlating* a 2D function $g(s_x,s_y)$ with the input image, the overall feature map indicates which location in the original image best resembles $g(s_x,s_y)$.

is called a *feature map* or *activation map*. For the same input image/layer, one can build multiple feature maps, where the features are learned via a training process. A parameter called *stride* defines how many pixels we slide the $\mathbf{w}^{\mathrm{T}}(\cdot)+b$ filter across the input image horizontally/vertically per output. The stride value determines the size of a feature map. Next, a *max-pooling* or *sub-sampling* layer operates over the feature maps by picking the largest value out of 4 neighboring neurons as shown in Fig. 20(c). Max-pooling is essentially nonlinear down-sampling with the objective to retain the largest identified features while reduce the dimensionality of the feature maps.

The $\mathbf{w}^{\mathrm{T}}(\cdot)+b$ operation essentially multiplies part of the input image with a 2D function $g(s_x,s_y)$ and sums the results as shown in Fig. 21. The sliding of $g(s_x,s_y)$ over all spatial locations is the same as *convolving* the input image with $g(-s_x,-s_y)$ (hence the name convolutional neural networks). Alternatively, one can also view the $\mathbf{w}^{\mathrm{T}}(\cdot)+b$ operation as *cross-correlating* $g(s_x,s_y)$ with the input image. Therefore, a high value will result if that part of the input image resembles $g(s_x,s_y)$. Together with the decision-like nonlinear activation function, the overall feature map indicates which location in the original image best resembles $g(s_x,s_y)$, which essentially tries to identify and locate a certain feature in the input image. With this insight, the interleaving convolutional and sub-sampling layers can be intuitively understood as identifying higher-level and more complex features of the input image.

The training of a CNN is performed using a modified BP algorithm which updates convolutional filters' weights and also takes the sub-sampling layers into account. Since a lot of weights are supposedly identical as the network is essentially performing the convolution operation, one will update those weights using the average of the corresponding gradients.

D. Recurrent Neural Networks (RNNs)

In our discussion up to this point, different input-output pairs $(\mathbf{x}(i),\mathbf{y}(i))$ and $(\mathbf{x}(j),\mathbf{y}(j))$ in a data set are assumed to have no relation with each other. However, in a lot of real-world applications such as speech recognition, handwriting recognition, stock market performance prediction, inter-symbol interference (ISI) cancellation in communications, etc., the sequential data has important spatial/temporal dependence to be learned. An RNN is a type of neural network that performs

pattern recognition for data sets with memory. RNNs have feedback connections, as shown in Fig. 22, and thus enable the information to be temporarily memorized in the networks [30]. This property allows RNNs to analyze sequential data by making use of their inherent memory.

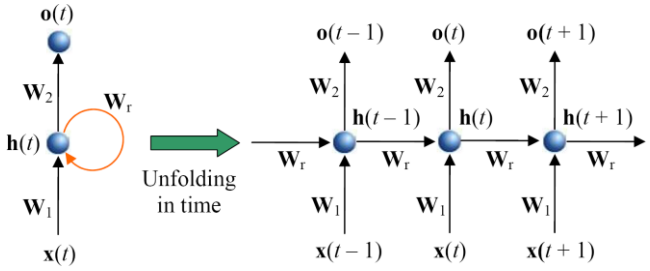


Fig. 22. Schematic diagram of an RNN and the unfolding in time.

Consider an RNN as shown in Fig. 22 with an input $\mathbf{x}(t)$, an output $\mathbf{o}(t)$ and a hidden state $\mathbf{h}(t)$ representing the memory of the network, where the subscript t denotes time. The model parameters $\mathbf{W}_1$, $\mathbf{W}_2$ and $\mathbf{W}_r$ are input, output and recurrent weight matrices, respectively. An RNN can be unfolded in time into a multilayer network [31], as shown in Fig. 22. Note that unlike a feed-forward ANN which employs different parameters for each layer, the same parameters $\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_r$ are shared across all steps which reflects the fact that essentially same task is being performed at each step but with different inputs. This significantly reduces the number of parameters to be learned. The hidden state $\mathbf{h}(t)$ and output $\mathbf{o}(t)$ at time step t can be computed as

$$\mathbf{h}(t) = \sigma_1(\mathbf{W}_1\,\mathbf{x}(t) + \mathbf{W}_r\,\mathbf{h}(t-1) + \mathbf{b}_1) \tag{28}$$

$$\mathbf{o}(t) = \sigma_2(\mathbf{W}_2\,\mathbf{h}(t) + \mathbf{b}_2) \tag{29}$$

where $\mathbf{b}_1$ and $\mathbf{b}_2$ are the bias vectors while $\sigma_1(\cdot)$ and $\sigma_2(\cdot)$ are the activation functions for the hidden and output layer neurons, respectively. Given a data set $\{(\mathbf{x}(1),\mathbf{y}(1)),(\mathbf{x}(2),\mathbf{y}(2)),\ldots(\mathbf{x}(L),\mathbf{y}(L))\}$ of input-output pairs, the RNN is first unfolded in time to represent it as a multilayer network and then BP algorithm is applied on this graph, as shown in Fig. 23, to compute all the necessary matrix derivatives $\left\{\frac{\partial E}{\partial \mathbf{W}_1}, \frac{\partial E}{\partial \mathbf{W}_2}, \frac{\partial E}{\partial \mathbf{W}_r}, \frac{\partial E}{\partial \mathbf{b}_1}, \frac{\partial E}{\partial \mathbf{b}_2}\right\}$. The loss function can be cross-entropy or MSE. The matrix derivative $\frac{\partial E}{\partial \mathbf{W}_r}$ is a bit more complicated to calculate since $\mathbf{W}_r$ is shared across all hidden layers. In this case,

$$\frac{\partial E}{\partial \mathbf{W}_r} = \sum_{t=1}^{L} \frac{\partial E(t)}{\partial \mathbf{W}_r} = \sum_{t=1}^{L} \frac{\partial E(t)}{\partial \mathbf{h}(t)} \frac{\partial \mathbf{h}(t)}{\partial \mathbf{W}_r} = \sum_{t=1}^{L} \frac{\partial E(t)}{\partial \mathbf{h}(t)} \sum_{l=1}^{t} \frac{\partial \mathbf{h}(t)}{\partial \mathbf{h}(l)} \frac{\partial \mathbf{h}(l)}{\partial \mathbf{W}_r} \tag{30}$$

where most of the derivatives in Eq. (30) can be easily computed using Eqs. (28) and (29). The Jacobian $\frac{\partial \mathbf{h}(t)}{\partial \mathbf{h}(l)}$ is further decomposed into $\frac{\partial \mathbf{h}(t)}{\partial \mathbf{h}(t-1)} \frac{\partial \mathbf{h}(t-1)}{\partial \mathbf{h}(t-2)} \cdots \frac{\partial \mathbf{h}(l+1)}{\partial \mathbf{h}(l)}$ so that efficient updates naturally involve the flow of matrix derivatives from the last data point $(\mathbf{x}(L),\mathbf{y}(L))$ back to the first $(\mathbf{x}(1),\mathbf{y}(1))$. This algorithm is called back-propagation through time (BPTT) [32].

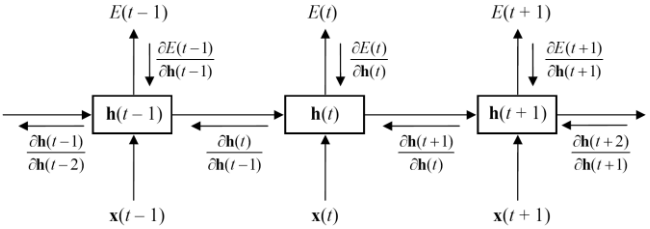


Fig. 23. Flow of gradient signals in an RNN.

In the special case when the nonlinear activation function is absent, the RNN structure resembles a linear multiple-input multiple-output (MIMO) channel with memory 1 in communication systems. Optimizing the RNN parameters will thus be equivalent to estimating the channel memory given input and output signal waveforms followed by maximum likelihood sequence detection (MLSD) of additional received signals. Consequently, an RNN may be used as a suitable tool for channel characterization and data detection in *nonlinear* channels with memory such as long-haul transmission links with fiber Kerr nonlinearity or direct detection systems with CD, chirp or other component nonlinearities. Network traffic prediction may be another area where RNNs can play a useful role.

One major limitation of conventional RNNs in many practical applications is that they are not able to learn long-term dependencies in data (i.e., dependencies between events that are far apart) due to the so called *exploding* and *vanishing gradient problems* encountered during their training. To overcome this issue, a special type of RNN architecture called long short-term memory (LSTM) network is designed which can model and learn temporal sequences and their long-range dependencies more accurately through better storing and accessing of information [29]. An LSTM network makes decision on whether to forget/delete or store the information based on the importance which it assigns to the information. The assigning of importance takes place through weights which are determined via a learning process. Simply put, an LSTM network learns over time which information is important and which is not. This allows LSTM network's short-term memory to last for longer periods of time as compared to conventional RNNs which in turn leads to improved sequence learning performance.

V. Applications of ML Techniques in Optical Communications and Networking

Fig. 24 shows some significant research works related to the use of ML techniques in fiber-optic communications. A brief discussion on these works is given below.

A. Optical Performance Monitoring (OPM)

Optical communication networks are becoming increasingly complex, transparent and dynamic. Reliable operation and efficient management of these complex fiber-optic networks require incessant and real-time information of various channel impairments ubiquitously across the network, also known as OPM [33]. OPM is widely regarded as a key enabling technology for SDNs. Through OPM, SDNs can become aware of the real-time network conditions and subsequently adjust different transceiver/network elements parameters such as

launched powers, data rates, modulation formats, spectrum assignments, etc., for optimized transmission performance [4]. Unfortunately, conventional OPM techniques have shown limited success in simultaneous and independent monitoring of multiple transmission impairments since the effects of different impairments are often difficult to separate analytically. Another crucial OPM requirement is low complexity since the OPM devices need to be deployed ubiquitously across optical networks. ML techniques are proposed as an enabler for realizing low complexity (and hence low cost) multi-impairment monitoring in optical networks and have already shown tremendous potential.

Most existing ML-based OPM techniques adopt a supervised learning approach utilizing training data sets of labeled examples during the offline learning process of selected ML models. The training data may, e.g., consist of signal representations like eye-diagrams, asynchronous delay-tap plots (ADTPs), amplitude histograms (AHs), etc., and their corresponding known impairments values such as CD, differential group delay (DGD), OSNR, etc., serving as data labels, as shown in Fig. 25. During the training phase, the inputs to an ML model are the impairments-indicative feature vectors **x** of eye-diagrams/ADTPs/AHs while their corresponding labels **y** are used as the targets as shown in Fig. 26(a). The ML model then learns the mapping between input features and the labels. Note that in case of eye-diagrams, the features can be parameters like eye-closure, *Q*-factor, root-mean-square jitter, crossing amplitude, etc. [34]. On the other hand, for AHs/ADTPs, the empirical one-dimensional (1D)/2D histograms can be treated as features [35][36]. Once the offline training process is completed, the ML model can be used for real-time monitoring (as the computations involved are relatively simple) in deployed networks as shown in Fig. 26(b).

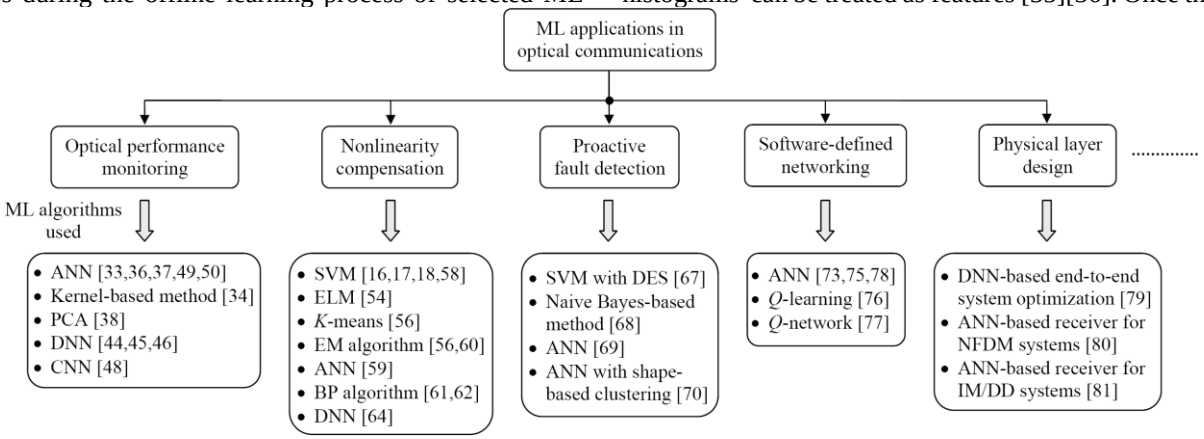


Fig. 24. Some key applications of ML in fiber-optic communications.

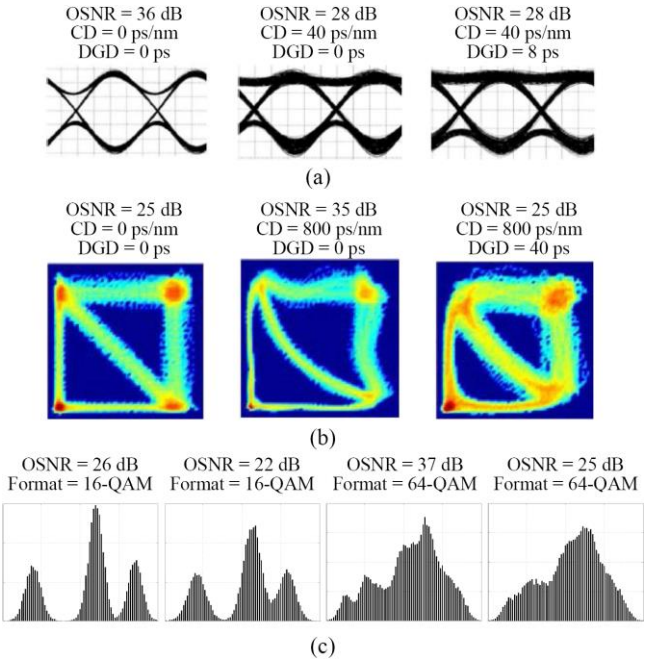


Fig. 25. Impairments-dependent patterns reflected by (a) eye-diagrams [34], (b) ADTPs [35] and (c) AHs [36], and their corresponding known impairments values which serve as data labels during the training process.

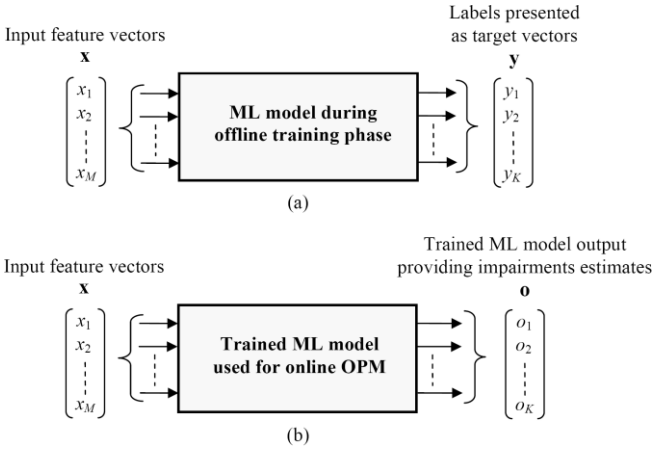
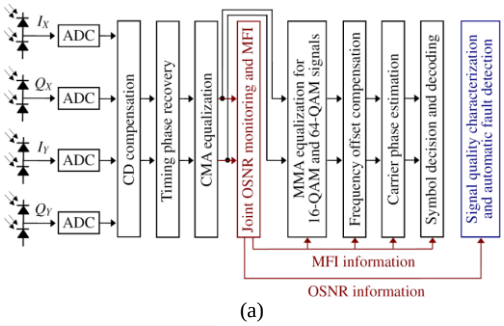


Fig. 26. (a) ML model during the offline training phase with feature vectors **x** as inputs and the labels **y** as targets. (b) Trained ML model used for online OPM with feature vectors **x** as inputs and the impairments estimates **o** as outputs.

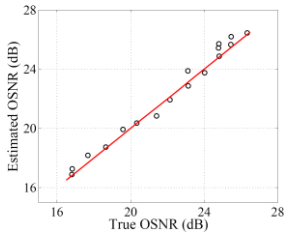
ML algorithms have been applied successfully for cost-effective multi-impairment monitoring in optical networks. Wu *et al.* [34] exploited impairments-sensitive features of eye-diagrams using an ANN for simultaneous monitoring of OSNR, CD and DGD. Similarly, Anderson *et al.* [35] demonstrated joint estimation of CD and DGD by applying kernel-based ridge regression on ADTPs. In [37], we showed that the raw empirical moments of asynchronously sampled signal

amplitudes are sensitive to OSNR, CD and DGD. The first five empirical moments of received signal amplitudes are thus used as input features to an ANN for multi-impairment monitoring. Unlike [34][35], which can only determine the magnitude of the CD, this technique enables monitoring of both magnitude and sign of accumulated CD. In [38], the low-frequency part of the received signal's RF spectrum is used as input to an ANN for OSNR monitoring in the presence of large inline uncompensated CD. Apart from supervised learning, unsupervised ML techniques have also been employed for OPM. In [39], PCA and statistical distance measurement based pattern recognition is applied on ADTPs for joint OSNR, CD and DGD monitoring as well as identification of bit-rate and modulation format of the received signal.

The emergence of SDNs imposes new requirements on OPM devices deployed at the intermediate network nodes. As a lot of OSNR/CD/polarization-mode dispersion (PMD) monitoring techniques are modulation format dependent, the OPM devices are desired to have modulation format identification (MFI) capabilities in order to select the most suitable monitoring technique. Although the modulation format information of a signal can be obtained from upper-layer protocols in principle, it is practically not available for OPM task at the intermediate network nodes because the OPM units are often stand-alone devices and can only afford limited complexity [4]. Note that MFI may also be beneficial for digital coherent receivers in elastic optical networks (EONs) since it can enable fast switching between format-dependent carrier recovery modules as conventional supervisory channels may not be able to provide modulation format information that quickly [40]. Reported ML-based MFI techniques in the literature include *K*-means algorithm [41], ANNs [36][42], variational Bayesian expectation-maximization (VBEM) [43], and DNN [44] based methods.



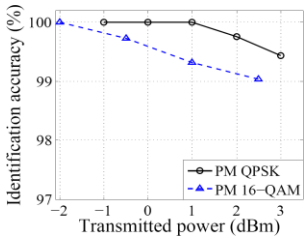
(a)



(b)

		Identified Modulation Format		
		PM QPSK	PM 16-QAM	PM 64-QAM
Actual Modulation Format	PM QPSK	19 100%	0 0%	0 0%
	PM 16-QAM	0 0%	17 100%	0 0%
	PM 64-QAM	0 0%	0 0%	21 100%

(c)



(d)

Fig. 27. (a) Receiver DSP configuration with the DNN-based OSNR monitoring and MFI stage shown in red color. (b) True versus estimated OSNRs for 112 Gb/s PM 16-QAM signals. (c) MFI accuracies (in number of instances and percentange of correct identifications) for different modulation formats in the absence of fiber nonlinear effects [45]. (d) Effect of fiber nonlinearity on the MFI accuracies [44].

Recently, DL algorithms have also been applied for OPM. In [45], we demonstrated joint OSNR monitoring and MFI in digital coherent receivers, as shown in Fig. 27(a), using DNNs in combination with AHs depicted in Fig. 25(c). The DNNs automatically extracted OSNR and modulation format sensitive features of AHs and exploited them for the joint estimation of these parameters. The OSNR monitoring results for one signal type are shown in Fig. 27(b) and it is clear from the figure that OSNR estimates are quite accurate. The confusion table/matrix in Fig. 27(c) summarizes MFI results (in the absence of fiber nonlinear effects) for 57 test cases used for evaluation. The upper element in each cell of this table represents the number of instances of correct/incorrect identifications for a given actual modulation format while the bottom element shows percentage of correct/incorrect identifications for a given actual modulation format. It is evident from the table that no errors are encountered in the identification of all three modulation formats under consideration. The performance of this technique in the presence of fiber nonlinearity is shown in Fig. 27(d) and it is clear from the figure that identification accuracies decrease slightly in this case [44]. However, they still remain higher than 99%, thus showing the resilience of this technique against fiber nonlinear effects. Since this technique uses DL algorithms inside standard digital coherent receiver, it avoids extra hardware costs. Similarly, Tanimura *et al.* [46] applied DNN on asynchronously sampled raw data for OSNR monitoring in a coherent receiver. Using a deep 5-layers architecture and a large training data set of 400,000 samples, the DNN is shown to learn and extract useful OSNR-sensitive features of incoming signals without involving any manual feature engineering. An extension of this method is presented in [47] where the DNN-based monitor is enhanced using the dropout technique [48] at the inference time (unlike typical approach of using dropout during training) so that multiple "thinned" DNNs with slightly different configurations provide multiple OSNR estimates. This in turn enables them to compute confidence intervals of the OSNR estimates as an auxiliary output. In [49], raw eye-diagrams are treated as images (comprising of various pixels) and are processed using a CNN for automatic extraction of features which are then used for joint OSNR monitoring and MFI. This technique exhibits better performance than conventional ML approaches.

Open issues: While ML-based OPM has received significant attention over the last few years, there are certain issues which still need to be addressed. For example, accurate OSNR monitoring in long-haul transmission systems in the presence of fiber nonlinearity is still a challenging task as nonlinear distortions are incorrectly treated as noise by most OSNR monitoring methods. Developing ML-based techniques to estimate actual OSNR irrespective of other transmission impairments, channel power, and wavelength-division multiplexing (WDM) effects is highly desirable in future optical networks. Recently, there have been some initial attempts in this regard exploiting amplitude noise covariance of

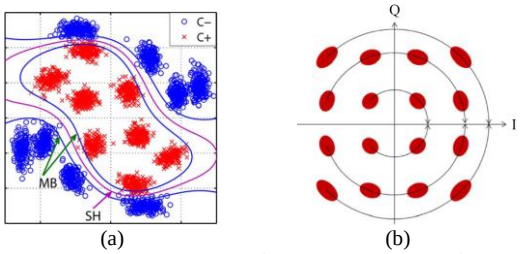
received symbols [50], and features of nonlinear phase noise along with time correlation properties of fiber nonlinearities [51]. Another open issue is the development of ML-based monitoring techniques using only low-bandwidth components as this can reduce the computational complexity and cost of OPM devices installed at the intermediate network nodes. Alternatively, instead of physical deployment of OPM units across the network, capturing of various physical layer data (such as launched powers, parameters of various optical amplifiers and fibers, etc.) via network management and using the ML algorithms to uncover complex relationships between these parameters and the actual link OSNRs can also be investigated.

B. Fiber Nonlinearity Compensation (NLC)

The optical fiber communication channel is nonlinear due to the Kerr effect and thus classical linear systems equalization/detection theory is suboptimal in this context. Signal propagation in optical fibers in the presence of Kerr nonlinearity together with distributed copropagating amplified spontaneous emission (ASE) noise $n(t,z)$ can be described by the stochastic nonlinear Schrödinger equation (NLSE)

$$\frac{\partial}{\partial z}u(t,z)+j\frac{\beta_2}{2}\frac{\partial^2}{\partial t^2}u(t,z)=j\gamma|u(t,z)|^2u(t,z)+n(t,z) \quad (31)$$

where $u(t,z)$ is the electric field while β_2 and γ are group velocity dispersion (GVD) parameter and fiber nonlinear coefficient, respectively. Although the NLSE can be numerically evaluated using the split-step Fourier method (SSFM) to simulate the waveforms evolution during transmission, the interplay between signal, noise, nonlinearity and dispersion complicates the analysis. This is also essentially the limiting factor of the DBP technique [52]. At present, stochastic characteristics of nonlinearity-induced noise depend in a complex manner on dispersion, modulation format, transmission distance, amplifier and type of optical fiber [53]. As an example, two received signal distributions after linear compensation of various transceiver and transmission impairments are shown in Fig. 28 for long-haul systems with and without inline dispersion compensation. From Fig. 28(a), it is obvious that the decision boundaries are nonlinear, which naturally calls for the use of ML techniques. In contrast, the nonlinear noise is more Gaussian-like in dispersion-unmanaged transmissions [54], as shown in Fig. 28(b). However, the noise is correlated in time and hard to model analytically.



(a) (b)

Fig. 28. Received signal distributions after linear equalization for long-haul (a) dispersion-managed [17] and (b) dispersion-unmanaged [53] transmissions.

More specifically, consider a transmission link with inline distributed erbium-doped fiber amplifiers (EDFAs). Let $\mathbf{y}=[y_1\ y_2\ \ldots]$ be a symbol sequence with overall transmitted signal $q(t)=\sum_j y_j s(t-jT)$ where $s(t)$ is the pulse shape and T is the symbol period. The received signal is given by

$$x(t)=f_{\mathrm{NLSE}}(q(t),n(t,z)) \quad (32)$$

where $f_{\mathrm{NLSE}}(\cdot)$ is the input-output mapping characterized by the NLSE. It should be noted that electronic shot noise and quantization noise also act as additional noise sources but are omitted here as ASE noise and its interaction with Kerr nonlinearity and dispersion are the dominant noise sources in long-haul transmissions. Other WDM effects are also omitted here for simplicity. The received signal is sampled to obtain $\mathbf{x}=[x_1\ x_2\ \ldots]$. While using ML for NLC, one seeks to develop a neural network or generally a mapping function $g(\cdot)$ so that the output vector

$$\mathbf{o}=g(\mathbf{x}) \quad (33)$$

of ML model is as close as possible to $\mathbf{y}$.

There are a few classes of approaches to learn the best mapping $g(\cdot)$. One direction is to completely ignore the intricate interactions between nonlinearity, CD and noise in NLSE and treat the nonlinear fiber as a black-box system. To this end, in [55], we proposed the use of an ANN after CD compensation in a digital coherent receiver and applied a training technique called extreme learning machine (ELM) that avoids SGD-like iterative weights and biases updates [56]. Fig. 29 shows the simulated *Q*-factor for 27.59 GBd/s return-to-zero (RZ) QPSK transmissions over 2000 km standard single-mode fiber (SSMF). The proposed ELM-based approach provides comparable performance to DBP but is computationally much simpler.

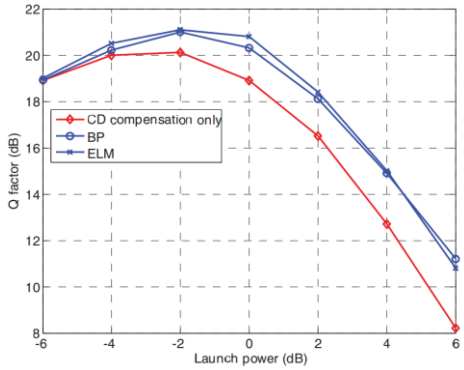


Fig. 29. *Q*-factor for 27.59 GBd/s RZ-QPSK signals after transmission over 2000 km SSMF [55].

For dispersion-managed systems, Zibar *et al.* [57] investigated the use of EM and *K*-means algorithms to derive nonlinear decision boundaries for optimized transmission performance. By applying the EM algorithm after regular DSP, the distribution of the combined fiber nonlinear distortions, laser phase noise and transceiver imperfections can be estimated. EM algorithm assumes that each cluster is Gaussian distributed with different mean and covariance matrix (the outer clusters in Fig. 28(a) are expected to have larger variances) so that the received signal distribution is a Gaussian mixture model. EM algorithm optimizes the mean and covariance matrix parameters iteratively to best fit the observed signal distribution in a maximum likelihood sense. The converged

mixture model is then used for symbols detection. In contrast, the *K*-means algorithm provides less improvement because it assumes that the clusters share the same covariance matrix [58]. Nevertheless, for dispersion-unmanaged systems, both EM and *K*-means algorithms provide minimal performance gain as optimal decision boundaries in this case are nearly straight lines as shown in [57]. Li *et al*. [17] applied SVMs for fiber NLC. However, since basic SVM is only a binary classifier, classifying *M*-QAM signals would require $\log_2 M$ binary SVMs in this case. Other SVM-related works with different complexities and performance concerns are reported in [18][19][59] with a 0.5−2 dB gain in *Q*-factor compared to linear equalization methods. In [60], the use of an ANN per sub-carrier to equalize nonlinear distortions in coherent optical orthogonal frequency-division multiplexing (CO-OFDM) systems is studied.

Another class of ML techniques incorporates limited amount of underlying physics to develop better fiber NLC methods. Pan *et al*. [61] noted that the received phase distortions of a 16-QAM signal are correlated across symbols due to nonlinearity in addition to the effect of slowly varying laser phase noise. They proposed to interleave the soft forward error correction (FEC) decoding with a modified EM algorithm. In particular, the EM algorithm takes the output of the soft-decision low-density parity-check (LDPC) decoder as input and provides phase noise estimates $\hat{\phi}_k$ which are then used to compensate the symbols and fed back to the LDPC decoder, thus forming an adaptive feedback equalizer. An additional regularization term is added inside the EM algorithm to prevent large phase jumps.

Finally, ML approaches can also augment well-developed analytical models and signal processing strategies for NLC. Häger *et al.* [62] consider the standard DBP algorithm

$$\mathbf{W}^{-1}\,\sigma^{-1}(\mathbf{W}^{-1}\,\sigma^{-1}(\dots)) \tag{34}$$

where $\mathbf{W}$ is a matrix (representing linear CD operation) and $\sigma(z) = e^{j\gamma|z|^2}$ is the nonlinear phase rotation. However, when viewed from an ML perspective, this sequence of interleaved linear and nonlinear operations resembles a standard neural network structure and hence, all the parameters in $\mathbf{W}$ as well as the nonlinear function parameters can be learned. A dedicated complex-valued DNN-like model following time-domain DBP procedures but generalizing the filter of the linear step and the scaling factor of the nonlinear phase rotation is developed. The loss function is differentiable with respect to the real and imaginary parts of the parameters and thus can be learned using standard BP techniques. The learned DBP algorithm performs similar to conventional DBP but is computationally simpler. For high baud rate systems, a sub-band division solution to reduce the required linear filter size was recently proposed [63]. In another approach, perturbation analysis of fiber nonlinearity is used to analyze the received signal $x(t) = x_{\text{lin}}(t) + \Delta x(t)$, where $x_{\text{lin}}(t)$ is the received signal if the system is linear while the intra-channel four-wave mixing (IFWM) perturbations $\Delta x(t)$ are given by

$$\Delta x(t) = \sum_{m,n} P^{3/2} C_{m,n} x_{\text{lin}}(t - mT) x_{\text{lin}}^*(t - (m+n)T) x_{\text{lin}}(t - nT) \tag{35}$$

where T is the symbol period, P is the signal power and $C_{m,n}$ is determined by the fiber parameters [64]. From an ML perspective, the $x_{\text{lin}}(t - mT)x_{\text{lin}}^*(t - (m+n)T)x_{\text{lin}}(t - nT)$ triplets and $x(t)$ can serve as inputs to an ANN to estimate $C_{m,n}$ and $\Delta x(t)$, which can then be used to pre-distort the transmitted signal and obtain better results. The proposed technique is demonstrated in an 11000 km subsea cable transmission and it outperforms transmitter side perturbation-based pre-distortion methods by 0.3 dB in both single-channel and WDM systems [65].

Open issues: Table I shows some key techniques using ML for fiber NLC. Most of these works incorporate ML as an extra DSP module placed either at transmitter or receiver. While effective to a certain extent, it is not clear what is the best sequence of conventional signal processing and ML blocks in such a hybrid DSP configuration. One factor driving the choice of sequence is the dynamic effects such as carrier frequency offset, laser phase noise, PMD, etc., that are hard to be captured in the learning process of an ML algorithm. In this case, one can perform ML-based NLC after linear compensations so as to avoid tackling these time-varying dynamics in ML. In the other extreme, RNN structures can embrace all the time-varying dynamics in principle but it may be an overkill since we do know their underlying physics and it should be exploited in the overall DSP design. Also, in case of hybrid configurations, the accuracy of conventional DSP algorithms such as CMA or carrier phase estimation (CPE) plays a major role in the quality of the data sets which ML fundamentally relies on. Therefore, there are strong dependencies between ML and conventional

Reference, Year	ML algorithm used	Data rate	Transmission link	Modulation format	Polarization multiplexing	WDM	Experimental demonstration
[55], 2011	ANN	27.59 GBd/s	2000 km SSMF	RZ-QPSK	No	No	No
[57], 2012	EM	14 GBd/s	< 800 km SSMF/DCF	16-QAM	Yes	No	Yes
[60], 2015	ANN	40 Gb/s	200 km SSMF	16-QAM CO-OFDM	No	No	Yes
[62], 2018	Learned DBP	20 GBd/s	3200 km SSMF	16-QAM	No	No	No
[65], 2018	Learned PPD using DNN	4 × 12.25 GBd/s	11000 km Trans-Pacific cable	DSM-PS-8/16/64-QAM	Yes	Yes	Yes

TABLE I Some key ML-based fiber NLC techniques. PPD: pre/post-distortion, DCF: dispersion-compensating fiber, DSM: digital subcarrier modulation, PS: probabilistic shaped

DSP blocks and the right balance is still an open area of research. Finally, to the best of our knowledge, an ML-based single-channel processing technique that outperforms DBP in practical WDM settings has yet to be developed.

Numerous studies are conducted to also address the computational complexity issues of conventional and ML techniques for fiber NLC. For conventional NLC algorithms, we direct the readers to the survey paper [66]. On the other hand, the computational complexity of ML algorithms for NLC varies significantly with the architecture and the training process used which make comparison with the conventional techniques difficult. Generally, the training processes are too complex to be performed online as they require a lot of iterations and potentially massive training data. For the inference phase (i.e., using the trained model for real-time data detection), most ML algorithms proposed involve relatively simple computations, leading to the perception that ML techniques are generally simple to implement since offline training processes are typically not counted towards the computational complexity. However, in reality, the training does take up a lot of computational resources and time which should not be completely disregarded while evaluating the complexity of ML approaches for NLC.

C. Proactive Fault Detection

Reliable network operations are essential for the carriers to provide service guarantees, called service-level agreements (SLAs), to their customers regarding system's availability and promised quality levels. Violation of these guarantees may result in severe penalties. It is thus highly desirable to have an early warning and proactive protection mechanism incorporated into the network. This can empower network operators to know when the network components are beginning to deteriorate and preventive measures can then be taken to avoid serious disruptions [33].

Conventional fault detection and management tools in optical networks adopt a rigid approach where some fixed threshold limits are set by the system engineers and alarms are triggered to alert malfunctions if those limits are surpassed. Such traditional network protection approaches have the following main drawbacks: (i) These methods protect a network in a passive manner, i.e., they are unable to forecast the risks and tend to reduce the damages only after a failure occurs. This approach may result in the loss of immense amounts of data during network recovery process once a failure happens. (ii) The inability to accurately forecast the faults leads to ultraconservative network designs involving large operating margins and protection switching paths which in turn result in an underutilization of the system resources. (iii) They are unable to determine the root cause of faults. (iv) Apart from hard failures (i.e., the ones causing major signal disruptions), several kinds of soft failures (i.e., the ones degrading system performance slowly and slightly) may also occur in optical networks which cannot be easily detected using conventional methods.

ML-enabled proactive fault management has recently been conceived as a powerful means to assure reliable network operation [67]. Instead of using traditional fixed pre-engineered solutions, this new mechanism relies on dynamic data-driven operations, leveraging immense amounts of operational data retrieved through network monitors (e.g., using simple network management protocol (SNMP)). The data repository may include network components' parameters such as optical power levels at different network nodes, EDFAs' gains, current drawn and power consumption of various devices, shelf temperature, temperatures of various critical devices, etc. ML-based fault prediction tools are able to learn historical fault patterns in networks and uncover hidden correlations between various entities and events through effective data analytics. Such unique and powerful capabilities are extremely beneficial in realizing proactive fault discovery and preventive maintenance mechanisms in optical networks. Fig. 30 illustrates various fault management tasks powered by the ML-based data analytics in optical networks including proactive fault detection, fault classification, fault localization, fault identification, and fault recovery.

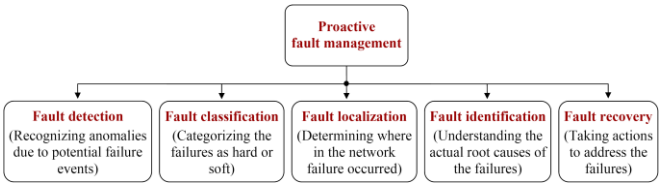


Fig. 30. Fault management tasks enabled by ML-based approaches.

Recently, a few ML-based techniques have been developed for advanced failure prediction in networks. Wang *et al.* [68] demonstrated an ML-based network equipment failure prediction method in software-defined metropolitan area networks (SDMANs) using a combination of double exponential smoothing (DES) and an SVM. Their approach involves constant monitoring of various physical parameters of the boards used in WDM nodes. The set of parameters includes the boards' power consumption, laser bias current, laser

Fault Label	Description
I	Point abnormalities due to random flash events and may lead to abrupt device damage
II	Local abnormalities indicating potential flaws with potential long-term impact on service performance
III	Steady abnormalities due to preceding system configuration changes, and may lead to damage and/or consistent performance loss
IV	Ramp abnormalities representing gradual system and/or service distortion possibilities

(a)

		Abnormality Detection Rate [%]	Proactive Reaction Time [hours]	Detected / True Faults
Condition-based	Label I	100	0	1
Data-driven		100	0	1
Condition-based	Label II	0	0	0
Data-driven		~57	>100*	0.57
Condition-based	Label III	~45	0	0.45
Data-driven		100	10	1
Condition-based	Label IV	25	0	.25
Data-driven		~93	96	1.25

(b)

Fig. 31. (a) Fault types typically encountered in commercial fiber-optic networks. (b) Comparison of fault detection rates and proactive reaction

times of data-driven and condition-based methods for the fault types given in (a) [70].

temperature offset, and environment temperature. However, to realize proactive failure detection, DES, which is basically a time-series prediction algorithm, is used to predict the future values of these parameters. Next, an SVM-based classifier is used to learn the relationship between forecasted values of boards' parameters and the occurrence of failure events. This method is shown to predict boards' failures with an average accuracy of 95%.

Similarly, in [69], proactive detection of fiber damages is demonstrated using ML-based pattern recognition. In their work, the state-of-polarization (SOP) rotation speed is constantly monitored in a digital coherent receiver and if it exceeds a certain predefined limit, the system considers it as an indication of some fiber stress event (leading to certain fiber damages) and a flag is raised. Next, Stokes parameters' traces are recorded which are shown to exhibit unique patterns for various mechanical stress events on the fiber such as bending, shaking, etc. These patterns are exploited using a naive Bayes classifier for their recognition. This technique is shown to predict various fiber stress events (and thus fiber breaks before their actual occurrence) with 95% accuracy.

In [70], a cognitive fault detection architecture is proposed for intelligent network assurance. In their work, an ANN is used to learn historical fault patterns in networks for proactive fault detection. The ANN is trained to learn how the monitored optical power levels evolve over time under normal or abnormal network operation (i.e., recognize power level abnormalities due to occurrence of certain faults). The trained ANN is then shown to detect significant network faults with better detection accuracies and proactive reaction times as compared to conventional threshold-based fault detection approaches, as shown in Fig. 31. An extension of this method is presented in [71] which makes use of an ANN and shape-based clustering algorithm to not only proactively detect and localize faults but also determine their likely root causes. The two-stage fault detection and diagnosis framework proposed in their work involves monitoring optical power levels across various network nodes as well as nodes' local features such as temperature, amplifier gain, current draw profiles, etc. In the first stage, an ANN is applied to detect faults by identifying optical power level abnormalities across various network nodes. The faulty node is then localized using network topology information. In the second stage, the faulty node's local features (which are also interdependent) are further analyzed using a clustering technique to identify potential root causes.

Open issues: Realization of ML-based proactive fault management in optical networks is still at its nascent stage. While few techniques for detecting and localizing hard failures have been proposed and deployed, the development of effective automated solutions for soft failures is still a relatively unexplored area. Furthermore, most of the existing works focus on the detection/localization of faults while the development of mechanisms which can uncover actual root causes of these faults as well as facilitate efficient fault recovery process is an open area for research. Another major problem faced while implementing ML-based fault detection/prevention is the unavailability of extensive data sets corresponding to different faulty operational conditions. This is mainly due to the fact that current network operators adopt ultraconservative designs with large operating margins in order to reduce the fault occurrence probability in their networks. This, however, limits the chances to collect sufficient examples of various network failure scenarios. In this context, the development of ML algorithms which could predict network faults accurately despite using minimal training data sets is an interesting area for research.

D. Software-Defined Networking (SDN)

Software-defined networking approach centralizes network management by decoupling the data and control planes. SDN technologies enable the network infrastructure to be centrally controlled/configured in an intelligent way by using various software applications. Data-driven ML techniques naturally fit in SDNs where abundant data can be captured by accessing the monitors spanning the whole network. Many studies have demonstrated the applications of ML in solving particular problems in SDNs such as network traffic prediction, fault detection, quality-of-transmission (QoT) estimation, etc. We refer the readers to two recent survey papers [72][73] for comprehensive reviews on these topics. In contrast, systematic integration of those ML applications into an SDN framework for cross-layer optimization is less reported, which is what we will focus on here. Morales *et al.* [74] performed ANN-based data analytics for robust and adaptive network traffic modeling. Based on the predicted traffic volume and direction, the virtual network topology (VNT) is adaptively reconfigured to ensure that the required grade of service is supported. Compared to static VNT design approaches, this predictive method decreases the required number of transponders to be installed at the routers by 8–42% as shown in Fig. 32, thus reducing energy consumption and costs. Similarly, Alvizu *et al.* [76] used ML to predict tidal traffic variations in a software-defined mobile

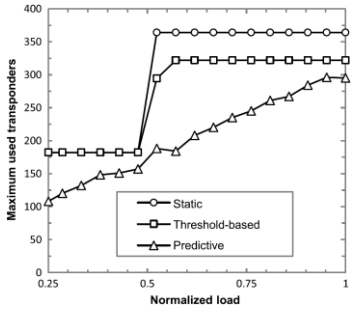


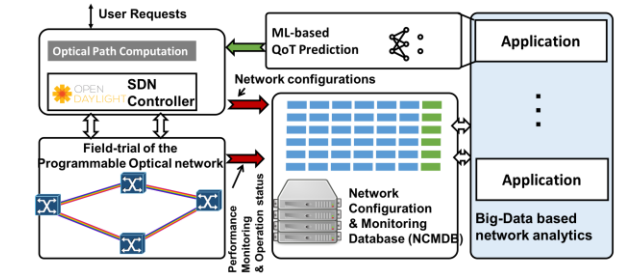
Fig. 32. Maximum used transponders versus load [75].

metro-core network. In their work, ANNs are employed to forecast traffic at different locations of an optical network and the predicted traffic demands are then exploited to optimize the online routing and wavelength assignments using a combination of analytical derivations and heuristics. Energy savings of ~31% are observed as compared to traditional static methods used in mobile metro-core networks.

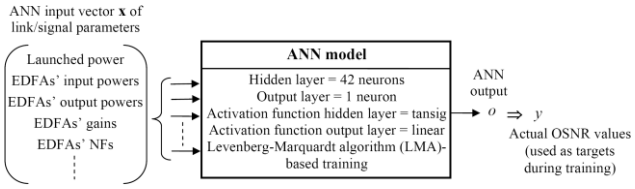
In [77], an RL technique called *Q*-learning is used to solve the path and wavelength selection problem in optical burst switched (OBS) networks. Initially, for each burst to be transmitted between a given source-destination pair, the algorithm picks a path and wavelength from the given sets of paths and wavelengths, respectively, as action and then a

reward which depends on the success or failure of that burst transmission is determined. In this way, the algorithm learns over time how to select optimal paths and wavelengths which can minimize burst loss probability (BLP) for each source-destination pair. It has been shown that the *Q*-learning algorithm reduces BLP significantly as compared to other adaptive schemes proposed in the literature. Similarly, Chen *et al.* [78] applied an RL algorithm, called *Q*-network, for joint routing, modulation and spectrum assignment (RMSA) provisioning in SDNs. In their work, the *Q*-network self-learns the best RMSA policies under different network states and time-varying demands based on the feedback obtained from the network for the RMSA actions taken in those conditions. Compared to shortest-path (SP) routing and the first-fit (FF) spectrum assignment approach, 4 times reduction in request blocking probability is reported using this method.

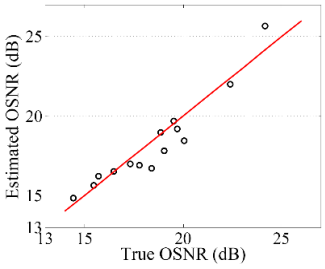
In [79], we demonstrated an ML-assisted optical network planning framework for SDNs. In this work, the network configuration as well as the real-time information about different link/signal parameters such as launched power, EDFAs' input and output powers, EDFAs' gains, EDFAs' noise figures (NFs), etc., is stored in a network configuration and monitoring database (NCMDB) as shown in Fig. 33(a). Next, an ANN is trained using this information where vectors **x** comprising of above-mentioned link/signal parameters are applied at the input of the ANN while actual known OSNR values *y* corresponding to those links are used as targets, as depicted in Fig. 33(b). The ANN is then made to learn the relationship between these two sets of data by optimizing its various parameters. After training, the ANN is able to predict the performance (in terms of OSNR) of various unestablished lightpaths in the network, as shown in Fig. 33(c), for optimum network planning. We demonstrated that the ML-based performance prediction mechanism can be used to increase the



(a)



(b)



(c)

Fig. 33. (a) Schematic diagram of ML-assisted optical network planning framework for SDNs. (b) ANN model with link/signal parameters as inputs and estimated OSNRs as outputs. (c) True versus estimated OSNRs using the ANN model [79].

transmission capacity in an SDN framework by adaptively configuring a probabilistic shaping-based spectral efficiency tunable transmitter.

Open issues: We already observe some benefits of ML-assisted network state prediction and decision-making in SDNs. However, some practical concerns need to be addressed when applying ML in SDNs. Firstly, real networks still require worst-case performance guarantees, which in turn necessitates a full understanding of the robustness of the chosen ML algorithms. Secondly, network characteristics can vary significantly in different network scenarios. An ML model trained using one particular data set may not be able to generalize to all network scenarios and thus the scalability of such a method becomes questionable.

A number of concerns also need to be addressed to realize more active use of RL in SDNs. Firstly, it must be shown that RL algorithms are scalable to handle large and more complex networks. Secondly, the RL algorithms must demonstrate fast convergence in real network conditions so as to limit the impact of non-optimal actions taken during the early learning phase of these algorithms.

The interpretability of ML methods is another issue as it is not desirable in practice to adopt an algorithm without really understanding how and why it works. Consequently, much needs to be done in understanding the fundamental properties of ML algorithms and how to properly incorporate them into SDN framework.

E. Physical Layer Design

Machine learning techniques offer the opportunity to optimize the design of individual physical components as well as complete end-to-end fiber-optic communication systems. Recently, we have seen some noticeable research works in this regard with quite encouraging results.

In [80], a complete optical communication system including transmitter, receiver and nonlinear channel is modeled as an end-to-end fully-connected DNN. This approach enables the optimization of transceivers in a single end-to-end process where the transmitter learns waveform representations that are robust to channel impairments while the receiver learns to equalize channel distortions. The results for 42 Gb/s intensity modulation/direct detection (IM/DD) systems show that the DL-based optimization outperforms the solutions based on two- and four-level pulse amplitude modulation (PAM2/PAM4) and conventional receiver equalization, for a range of transmission distances. Jones *et al.* [81] proposed an ANN-based receiver for

nonlinear frequency-division multiplexing (NFDM) optical communication systems. Unlike standard nonlinear Fourier transform (NFT) based receivers which are vulnerable to losses and noise in NFDM systems, the ANN-based receiver tackles these impairments by learning the distortion characteristics of previously-transmitted pulses and applying them for inference for future decisions. The results demonstrate improved bit-error rate (BER) performance as compared to conventional NFT-based receivers for practical link configurations. In [82], an ANN is used in receiver DSP for mitigating linear and nonlinear impairments in IM/DD systems. The ANN infers linear and nonlinear channel responses simultaneously which are then exploited for increasing the demodulation reliability beyond the capability of linear equalization techniques. Using an ANN along with standard feed-forward equalization (FFE), up to 10 times BER improvement over FFE-only configurations is demonstrated for 84 GBd/s PAM4 transmission over 1.5 km SSMF.

VI. Future Role of ML in Optical Communications

The emergence of SDNs with their inherent programmability and access to enormous amount of network-related monitored data provides unprecedented opportunities for the application of ML methods in these networks. The vision of future intelligent optical networks integrates the programmability/automation functionalities of SDNs with data-analytics capabilities of ML technologies to realize self-aware, self-managing and self-healing network infrastructures. Over the past few years, we have seen an increasing amount of research on the application of ML techniques in various aspects of optical communications and networking. As ML is gradually becoming a common knowledge to the photonics community, we can envisage some potential significant developments in optical networks in the near future ushered in by the application of emerging ML technologies.

Looking to the future, we can foresee a vital role played by ML-based mechanisms across several diverse functional areas in optical networks, e.g., network planning and performance prediction, network maintenance and fault prevention, network resources allocation and management, etc. ML can also aid cross-layer optimization in future optical networks requiring big data analytics since it can inherently learn and uncover hidden patterns and unknown correlations in big data which can be extremely beneficial in solving complex network optimization problems. The ultimate objective of ML-driven next-generation optical networks will be to provide infrastructures which can monitor themselves, diagnose and resolve their problems, and provide intelligent and efficient services to the end users.

VII. Online Resources for ML Algorithms

Standard ML algorithms' codes and examples are readily available online and one seldom needs to write their own codes from the very beginning. There are several off-the-shelf powerful frameworks available under open-source licenses such as TensorFlow, Pytorch, Caffe, etc. Matlab, which is widely used in optical communications researches is not the most popular programming language among the ML community. Instead, Python is the preferred language for ML research partly because it is freely available, multi-platform, relatively easy to use/read, and has a huge number of libraries/modules available for a wide variety of tasks. We hereby report some useful resources including example Python codes using TensorFlow library to help interested readers get started with applying simple ML algorithms to their problems. More intuitive understanding of ANNs can be found at this visual playground [83]. The Python codes for most of the standard neural network architectures discussed in this paper can be found in these Github repositories [84][85] with examples. For non-standard model design, TensorFlow also provides low-level programming interfaces for more custom and complex operations based on its symbolic building blocks, which are documented in detail in [86].

VIII. Conclusions

In this paper, we discussed how the rich body of ML techniques can be applied as a unique and powerful set of signal processing tools in fiber-optic communication systems. As optical networks become faster, more dynamic and more software-defined, we will see an increasing number of applications of ML and big data analytics in future networks to solve certain critical problems that cannot be easily tackled using conventional approaches. A basic knowledge and skills in ML will thus become necessary and beneficial for researchers in the field of optical communications and networking.

Appendix

For cross-entropy loss function defined in Eq. (14), the derivative with respect to the output is given by

$$\frac{\partial E(n)}{\partial o_j(n)} = -\frac{y_j(n)}{o_j(n)}. \tag{36}$$

With softmax activation function for the output neurons,

$$\begin{aligned}\frac{\partial o_j(n)}{\partial r_k(n)} &= \frac{\left(\sum_{m=1}^{K} e^{r_m(n)}\right)e^{r_j(n)}\delta_{j,k} - e^{r_j(n)}\cdot e^{r_k(n)}}{\left(\sum_{m=1}^{K} e^{r_m(n)}\right)^2} \\ &= \frac{\left(\sum_{m=1}^{K} e^{r_m(n)}\right)e^{r_j(n)}\delta_{j,k} - e^{r_j(n)}\cdot e^{r_k(n)}}{\left(\sum_{k=1}^{K} e^{r_k(n)}\right)^2} \\ &= o_j(n)\delta_{j,k} - o_j(n)o_k(n)\end{aligned} \tag{37}$$

where $\delta_{j,k} = 1$ when $j = k$ and 0 otherwise. Consequently,

$$\begin{aligned}\frac{\partial E(n)}{\partial r_k(n)} &= \sum_{j=1}^{K} \frac{\partial E(n)}{\partial o_j(n)}\frac{\partial o_j(n)}{\partial r_k(n)} \\ &= \sum_{j=1}^{K} -\frac{y_j(n)}{o_j(n)}\Big(o_j(n)\delta_{j,k} - o_j(n)o_k(n)\Big) \\ &= \sum_{j=1}^{K} -y_j(n)\Big(\delta_{j,k} - o_k(n)\Big) = o_k(n) - y_k(n)\end{aligned} \tag{38}$$

as $\sum_{j=1}^{K} y_j(n) = 1$. Therefore,

$$\frac{\partial E(n)}{\partial \mathbf{r}(n)} = \mathbf{o}(n) - \mathbf{y}(n).$$

Now, since $\frac{\partial \mathbf{r}(n)}{\partial \mathbf{b}_2}$, $\frac{\partial \mathbf{r}(n)}{\partial \mathbf{b}_1}$, $\frac{\partial r_k(n)}{\partial \mathbf{W}_2}$, $\frac{\partial r_k(n)}{\partial \mathbf{W}_1}$ are the same as $\frac{\partial \mathbf{o}(n)}{\partial \mathbf{b}_2}$, $\frac{\partial \mathbf{o}(n)}{\partial \mathbf{b}_1}$, $\frac{\partial o_k(n)}{\partial \mathbf{W}_2}$, $\frac{\partial o_k(n)}{\partial \mathbf{W}_1}$ for MSE loss function and linear activation function for the output neurons (as $\mathbf{o}(n) = \mathbf{r}(n)$ for that case), it follows that the update equations Eq. (8) to Eq. (12) also hold for the ANNs with cross-entropy loss function and softmax activation function for the output neurons.

ACKNOWLEDGMENT

The authors would like to thank anonymous reviewers for their valuable comments and suggestions.

REFERENCES

[1] S. Marsland, *Machine Learning: An Algorithmic Perspective*, 2nd ed. Boca Raton, USA: CRC Press, 2015.

[2] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, Aug. 2013.

[3] C. M. Bishop, *Pattern recognition and Machine Learning*. New York, USA: Springer, 2006.

[4] Z. Dong, F. N. Khan, Q. Sui, K. Zhong, C. Lu, and A. P. T. Lau, "Optical performance monitoring: A review of current and future technologies," *J. Lightwave Technol.*, vol. 34, no. 2, pp. 525–543, Jan. 2016.

[5] A. S. Thyagaturu, A. Mercian, M. P. McGarry, M. Reisslein, and W. Kellerer, "Software defined optical networks (SDONs): A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2738–2786, Oct.–Dec. 2016.

[6] https://www.youtube.com/channel/UCLZL8KsCzOODkDKBOI3S2lw

[7] R. A. Dunne, *A Statistical Approach to Neural Networks for Pattern Recognition*. Hoboken, USA: John Wiley & Sons, 2007.

[8] I. Kaastra, and M. Boyd, "Designing a neural network for forecasting financial and economic time series," *Neurocomputing*, vol. 10, no. 3, pp. 215–236, Apr. 1996.

[9] C. D. Meyer, *Matrix Analysis and Applied Linear Algebra*. Philadelphia, USA, Society for Industrial and Applied Mathematics, 2000.

[10] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. New York, USA: John Wiley & Sons, 2007.

[11] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. AISTATS*, Fort Lauderdale, FL, USA, 2011, vol. 15, pp. 315–323.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. ICCV*, Santiago, Chile, 2015, pp. 1026–1034.

[13] X. Glorot, and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. AISTATS*, Chia Laguna Resort, Sardinia, Italy, 2010, pp. 249–256.

[14] A. Webb, *Statistical Pattern Recognition*, 2nd ed. Chicester, UK: John Wiley & Sons, 2002.

[15] A. Statnikov, C. F. Aliferis, D. P. Hardin, and I. Guyon, *A Gentle Introduction to Support Vector Machines in Biomedicine*. Singapore: World Scientific, 2011.

[16] M. S. Andersen, J. Dahl, and L. Vandenberghe, *CVXOPT: Python software for convex optimization*. [Online]. Available: https://cvxopt.org

[17] M. Li, S. Yu, J. Yang, Z. Chen, Y. Han, and W. Gu, "Nonparameter nonlinear phase noise mitigation by using M-ary support vector machine for coherent optical systems," *IEEE Photonics J.*, vol. 5, no. 6, pp. 7800312–7800312, Dec. 2013.

[18] D. Wang, M. Zhang, Z. Li, Y. Cui, J. Liu, Y. Yang, and H. Wang, "Nonlinear decision boundary created by a machine learning-based classifier to mitigate nonlinear phase noise," in *Proc. ECOC*, Valencia, Spain, 2015, Paper P.3.16.

[19] T. Nguyen, S. Mhatli, E. Giacoumidis, L. V. Compernolle, M. Wuilpart, and P. Mégret, "Fiber nonlinearity equalizer based on support vector classification for coherent optical OFDM," *IEEE Photonics J.*, vol. 8, no. 2, pp. 1–9, Apr. 2016.

[20] M. Kirk, *Thoughtful Machine Learning with Python*. Sebastopol, USA: O'Reilly Media, 2017.

[21] I. T. Jolliffe, *Principal Component Analysis*, 2nd ed. New York, USA: Springer, 2002.

[22] J. E. Jackson, *A User's Guide to Principal Components*. Hoboken, USA: John Wiley & Sons, 2003.

[23] L. J. Cao, K. S. Chua, W. K. Chong, H. P. Lee, and Q. M. Gu, "A comparison of PCA, KPCA and ICA for dimensionality reduction in support vector machine," *Neurocomputing*, vol. 55, no. 1–2, pp. 321–336, Sep. 2003.

[24] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, USA: MIT Press, 2018.

[25] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, Nov. 2009.

[26] Y. Bengio and O. Delalleau, "On the expressive power of deep architectures," in *Algorithmic Learning Theory*, J. Kivinen, C. Szepesvári, E. Ukkonen, and T. Zeugmann, Eds. Heidelberg, Germany: Springer, 2011, pp. 18–36.

[27] L. J. Ba, and R. Caurana, "Do deep nets really need to be deep?," in *Proc. NIPS*, Montreal, Canada, 2014, pp. 2654–2662.

[28] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training deep neural networks," *J. Machine Learning Research*, vol. 10, pp. 1–40, Jan. 2009.

[29] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Massachusetts, USA: MIT Press, 2016.

[30] D. P. Mandic and J. Chambers, *Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability*. Chicester, UK: John Wiley & Sons, 2001.

[31] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," in *Proc. ICLR*, Banff, Canada, 2014.

[32] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proc. ICML*, Atlanta, GA, USA, 2013, pp. 1310–1318.

[33] F. N. Khan, Z. Dong, C. Lu, and A. P. T. Lau, "Optical performance monitoring for fiber-optic communication networks," in *Enabling Technologies for High Spectral-Efficiency Coherent Optical Communication Networks*, X. Zhou and C. Xie, Eds. Hoboken, USA: John Wiley & Sons, 2016, Ch. 14.

[34] X. Wu, J. A. Jargon, R. A. Skoog, L. Paraschis, and A. E. Willner, "Applications of artificial neural networks in optical performance monitoring," *J. Lightwave Technol.*, vol. 27, no. 16, pp. 3580–3589, Aug. 2009.

[35] T. B. Anderson, A. Kowalczyk, K. Clarke, S. D. Dods, D. Hewitt, and J. C. Li, "Multi impairment monitoring for optical networks," *J. Lightwave Technol.*, vol. 27, no. 16, pp. 3729–3736, Aug. 2009.

[36] F. N. Khan, Y. Zhou, A. P. T. Lau, and C. Lu, "Modulation format identification in heterogeneous fiber-optic networks using artificial neural networks," *Optics Express*, vol. 20, no. 11, pp. 12422–12431, May 2012.

[37] F. N. Khan, T. S. R. Shen, Y. Zhou, A. P. T. Lau, and C. Lu, "Optical performance monitoring using artificial neural networks trained with empirical moments of asynchronously sampled signal amplitudes," *IEEE Photon. Technol. Lett.*, vol. 24, no. 12, pp. 982–984, Jun. 2012.

[38] T. S. R. Shen, Q. Sui, and A. P. T. Lau, "OSNR monitoring for PM-QPSK systems with large inline chromatic dispersion using artificial neural network technique," *IEEE Photon. Technol. Lett.*, vol. 24, no. 17, pp. 1564–1567, Sep. 2012.

[39] M. C. Tan, F. N. Khan, W. H. Al-Arashi, Y. Zhou, and A. P. T. Lau, "Simultaneous optical performance monitoring and modulation format/bit-rate identification using principal component analysis," *J. Optical Commun. and Networking*, vol. 6, no. 5, pp. 441–448, May 2014.

[40] P. Isautier, K. Mehta, A. J. Stark, and S. E. Ralph, "Robust architecture for autonomous coherent optical receivers", *J. Optical Commun. and Networking*, vol. 7, no. 9, pp. 864–874, Sep. 2015.

[41] N. G. Gonzalez, D. Zibar, and I. T. Monroy, "Cognitive digital receiver for burst mode phase modulated radio over fiber links," in *Proc. ECOC*, Torino, Italy, 2010, Paper P6.11.

[42] F. N. Khan, Y. Zhou, Q. Sui, and A. P. T. Lau, "Non-data-aided joint bit-rate and modulation format identification for next-generation heterogeneous optical networks," *Optical Fiber Technology*, vol. 20, no. 2, pp. 68–74, Mar. 2014.

[43] R. Borkowski, D. Zibar, A. Caballero, V. Arlunno, and I. T. Monroy, "Stokes space-based optical modulation format recognition in digital coherent receivers," *IEEE Photon. Technol. Lett.*, vol. 25, no. 21, pp. 2129–2132, Nov. 2013.

[44] F. N. Khan, K. Zhong, W. H. Al-Arashi, C. Yu, C. Lu, and A. P. T. Lau, "Modulation format identification in coherent receivers using deep machine learning," *IEEE Photon. Technol. Lett.*, vol. 28, no. 17, pp. 1886–1889, Sep. 2016.

[45] F. N. Khan, K. Zhong, X. Zhou, W. H. Al-Arashi, C. Yu, C. Lu, and A. P. T. Lau, "Joint OSNR monitoring and modulation format identification in digital coherent receivers using deep neural networks," *Optics Express*, vol. 25, no. 15, pp. 17767–17776, Jul. 2017.

[46] T. Tanimura, T. Hoshida, J. C. Rasmussen, M. Suzuki, and H. Morikawa, "OSNR monitoring by deep neural networks trained with asynchronously sampled data," in *Proc. OECC*, Niigata, Japan, 2016, Paper TuB3-5.

[47] T. Tanimura, T. Kato, S. Watanabe, and T. Hoshida, "Deep neural network based optical monitor providing self-confidence as auxiliary output," in *Proc. ECOC*, Rome, Italy, 2018, Paper We1D.5.

[48] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, Jun. 2014.

[49] D. Wang, M. Zhang, Z. Li, J. Li, M. Fu, Y. Cui, and X. Chen, "Modulation format recognition and OSNR estimation using CNN-based deep learning," *IEEE Photon. Technol. Lett.*, vol. 29, no. 19, pp. 1667–1670, Oct. 2017.

[50] A. S. Kashi, Q. Zhuge, J. C. Cartledge, A. Borowiec, D. Charlton, C. Laperle, and M. O'Sullivan, "Fiber nonlinear noise-to-signal ratio monitoring using artificial neural networks," in *Proc. ECOC*, Gothenburg, Sweden, 2017, Paper M.2.F.2.

[51] F. J. V. Caballero, D. J. Ives, C. Laperle, D. Charlton, Q. Zhuge, M. O'Sullivan, and S. J. Savory, "Machine learning based linear and nonlinear noise estimation," *J. Optical Commun. and Networking*, vol. 10, no. 10, pp. D42–D51, Oct. 2018.

[52] E. Ip, "Nonlinear compensation using backpropagation for polarization-multiplexed transmission," *J. Lightwave Technol.*, vol. 28, no. 6, pp. 939–951, Mar. 2010.

[53] P. Poggiolini, and Y. Jiang, "Recent advances in the modeling of the impact of nonlinear fiber propagation effects on uncompensated coherent transmission systems," *J. Lightwave Technol.*, vol. 35, no. 3, pp. 458–480, Feb. 2017.

[54] A. Carena, G. Bosco, V. Curri, Y. Jiang, P. Poggiolini, and F. Forghieri, "EGN model of non-linear fiber propagation," *Optics Express*, vol. 22, no. 13, pp. 16335–16362, Jun. 2014.

[55] T. S. R. Shen, and A. P. T. Lau, "Fiber nonlinearity compensation using extreme learning machine for DSP-based coherent communication systems," in *Proc. OECC*, Kaohsiung, Taiwan, 2011, pp. 816–817.

[56] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1–3, pp. 489–501, Dec. 2006.

[57] D. Zibar, O. Winther, N. Franceschi, R. Borkowski, A. Caballero, V. Arlunno, M. N. Schmidt, N. G. Gonzales, B. Mao, Y. Ye, K. J. Larsen, and I. T. Monroy, "Nonlinear impairment compensation using expectation maximization for dispersion managed and unmanaged PDM 16-QAM transmission," *Optics Express*, vol. 20, no. 26, pp. B181–B196, Dec. 2012.

[58] E. Alpaydin, *Introduction to Machine Learning*, 2nd ed. Cambridge, USA: MIT Press, 2010.

[59] E. Giacoumidis, S. Mhatli, M. F. C. Stephens, A. Tsokanos, J. Wei, M. E. McCarthy, N. J. Doran, and A. D. Ellis, "Reduction of nonlinear intersubcarrier intermixing in coherent optical OFDM by a fast Newton-based support vector machine nonlinear equalizer," *J. Lightwave Technol.*, vol. 35, no. 12, pp. 2391–2397, Jun. 2017.

[60] E. Giacoumidis, S. T. Le, M. Ghanbarisabagh, M. McCarthy, I. Aldaya, S. Mhatli, M. A. Jarajreh, P. A. Haigh, N. J. Doran, A. D. Ellis, and B. J. Eggleton, "Fiber nonlinearity-induced penalty reduction in CO-OFDM by ANN-based nonlinear equalization," *Optics Lett.*, vol. 40, no. 21, pp. 5113–5116, Nov. 2015.

[61] C. Pan, H. Bülow, W. Idler, L. Schmalen, and F. R. Kschischang, "Optical nonlinear phase noise compensation for 9 × 32-Gbaud PolDM-16QAM transmission using a code-aided expectation-maximization algorithm," *J. Lightwave Technol.*, vol. 33, no. 17, pp. 3679–3686, Sep. 2015.

[62] C. Häger, and H. D. Pfister, "Nonlinear interference mitigation via deep neural networks," in *Proc. OFC*, San Diego, CA, USA, 2018, Paper W3A.4.

[63] C. Häger, and H. D. Pfister. "Wideband time-domain digital backpropagation via subband processing and deep learning," in *Proc. ECOC*, Rome, Italy, 2018, Paper Tu4F.4.

[64] Z. Tao, L. Dou, W. Yan, L. Li, T. Hoshida, and J. C. Rasmussen, "Multiplier-free intrachannel nonlinearity compensating algorithm operating at symbol rate," *J. Lightwave Technol.*, vol. 29, no. 17, pp. 2570–2576, Sep. 2011.

[65] V. Kamalov, L. Jovanovski, V. Vusirikala, S. Zhang, F. Yaman, K. Nakamura, T. Inoue, E. Mateo, and Y. Inada, "Evolution from 8QAM live traffic to PS 64-QAM with neural-network based nonlinearity compensation on 11000 km open subsea cable," in *Proc. OFC*, San Diego, CA, USA, 2018, Paper Th4D.5.

[66] R. Dar, and P. J. Winzer, "Nonlinear interference mitigation: Methods and potential gain," *J. Lightwave Technol.*, vol. 35, no. 4, pp. 903–930, Feb. 2017.

[67] F. N. Khan, C. Lu, and A. P. T. Lau, "Optical performance monitoring in fiber-optic networks enabled by machine learning techniques," in *Proc. OFC*, San Diego, CA, USA, 2018, Paper M2F.3.

[68] Z. Wang, M. Zhang, D. Wang, C. Song, M. Liu, J. Li, L. Lou, and Z. Liu, "Failure prediction using machine learning and time series in optical network," *Optics Express*, vol. 25, no. 16, pp. 18553–18565, Aug. 2017.

[69] F. Boitier, V. Lemaire, J. Pesic, L. Chavarria, P. Layec, S. Bigo, and E. Dutisseuil, "Proactive fiber damage detection in real-time coherent receiver," in *Proc. ECOC*, Gothenburg, Sweden, 2017, Paper Th.2.F.1.

[70] D. Rafique, T. Szyrkowiec, H. Griesser, A. Autenrieth, and J.-P. Elbers, "Cognitive assurance architecture for optical network fault management," *J. Lightwave Technol.*, vol. 36, no. 7, pp. 1443–1450, Apr. 2018.

[71] D. Rafique, T. Szyrkowiec, A. Autenrieth, and J.-P. Elbers, "Analytics-driven fault discovery and diagnosis for cognitive root cause analysis," in *Proc. OFC*, San Diego, CA, USA, 2018, Paper W4F.6.

[72] J. Mata, I. de Miguel, R. J. Durán, N. Merayo, S. K. Singh, A. Jukan, and M. Chamania, "Artificial intelligence (AI) methods in optical networks: A comprehensive survey," *Optical Switching and Networking*, vol. 28, pp. 43–57, Apr. 2018.

[73] F. Musumeci, C. Rottondi, A. Nag, I. Macaluso, D. Zibar, M. Ruffini, and M. Tornatore, "An overview on application of machine learning techniques in optical networks," *IEEE Commun. Surveys Tuts.*, to be published. DOI: 10.1109/COMST.2018.2880039.

[74] F. Morales, M. Ruiz, L. Gifre, L. M. Contreras, V. López, and L. Velasco, "Virtual network topology adaptability based on data analytics for traffic prediction," *J. Optical Commun. and Networking*, vol. 9, no. 1, pp. A35–A45, Jan. 2017.

[75] D. Rafique, and L. Velasco, "Machine learning for network automation: overview, architecture, and applications", *J. Opt. Commun. Networking*, vol. 10, no. 10, pp. D126–D143, Oct. 2018.

[76] R. Alvizu, S. Troia, G. Maier, and A. Pattavina, "Matheuristic with machine-learning-based prediction for software-defined mobile metro-core networks," *J. Optical Commun. and Networking*, vol. 9, no. 9, pp. D19–D30, Sep. 2017.

[77] Y. V. Kiran, T. Venkatesh, and C. S. Murthy, "A reinforcement learning framework for path selection and wavelength selection in optical burst switched networks," *IEEE J. Selected Areas in Commun.*, vol. 25, no. 9, pp. 18–26, Dec. 2007.

[78] X. Chen, J. Guo, Z. Zhu, R. Proietti, A. Castro, and S. J. B. Yoo, "Deep-RMSA: A deep-reinforcement-learning routing, modulation and spectrum assignment agent for elastic optical networks," in *Proc. OFC*, San Diego, CA, USA, 2018, Paper W4F.2.

[79] S. Yan *et al.*, "Field trial of machine-learning-assisted and SDN-based optical network planning with network-scale monitoring database," in *Proc. ECOC*, Gothenburg, Sweden, 2017, Paper Th.PDP.B.4.

[80] B. Karanov, M. Chagnon, F. Thouin, T. A. Eriksson, H. Bulow, D. Lavery, P. Bayvel, and L. Schmalen, "End-to-end deep learning of optical fiber communications," *J. Lightwave Technol.*, vol. 36, no. 20, pp. 4843–4855, Oct. 2018.

[81] R. T. Jones, S. Gaiarin, M. P. Yankov, and D. Zibar, "Time-domain neural network receiver for nonlinear frequency division multiplexed systems," *IEEE Photon. Technol. Lett.*, vol. 30, no. 12, pp. 1079–1082, Jun. 2018.

[82] J. Estaran, R. R.-Müller, M. A. Mestre, F. Jorge, H. Mardoyan, A. Konczykowska, J.-Y. Dupuy, and S. Bigo, "Artificial neural networks for linear and non-linear impairment mitigation in high-baudrate IM/DD systems," in *Proc. ECOC*, Düsseldorf, Germany, 2016, Paper M.2.B.2.

[83] D. Smilkov, and S. Carter, *TensorFlow — A neural network playground*. [Online]. Available: http://playground.tensorflow.org

[84] A. Damien, *GitHub repository — TensorFlow tutorial and examples for beginners with latest APIs.* [Online]. Available: https://github.com/aymericdamien/TensorFlow-Examples

[85] M. Zhou, *GitHub repository — TensorFlow tutorial from basic to hard*. [Online]. Available: https://github.com/MorvanZhou/Tensorflow-Tutorial

[86] TensorFlow, *Guide for programming with the low-level TensorFlow*

APIs. [Online]. Available: https://www.tensorflow.org/programmers_guide/low_level_intro

Faisal Nadeem Khan was born in Jhang, Pakistan. He received B.Sc. degree in electrical engineering from University of Engineering and Technology, Taxila, Pakistan, and M.Sc. degree in communications technology from University of Ulm, Ulm, Germany. He received Ph.D. degree in electronic and information engineering from The Hong Kong Polytechnic University, Hong Kong. From 2012 to 2015, he was a Senior Lecturer at the School of Electrical and Electronic Engineering, Universiti Sains Malaysia. He is currently working at the Photonics Research Centre, The Hong Kong Polytechnic University. His research interests include machine learning and signal processing techniques for high-speed fiber-optic communication systems. He has authored or coauthored more than 50 research papers in prestigious international journals and conferences as well as written one book chapter. He has been an invited speaker at various prestigious international conferences including Optical Fiber Communication (OFC) 2018, and Signal Processing in Photonic Communications (SPPCom) 2017, among others.

Qirui Fan was born in Zhejiang, China, in 1992. He received the B.Eng. and M.Eng. degrees in electrical engineering from Hunan University, Changsha, China, in 2014 and 2017, respectively. Currently, he is a Ph.D. student at the Department of Electrical Engineering, The Hong Kong Polytechnic University, Hong Kong. His research interests include machine learning techniques for fiber nonlinearity compensation.

Chao Lu received the B.Eng. degree in electronic engineering from Tsinghua University, Beijing, China, in 1985, and the M.Sc. and Ph.D. degrees from the University of Manchester, Manchester, U.K., in 1987 and 1990, respectively. In 1991, he joined, as a Lecturer, the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, where he has been an Associate Professor since January 1999. From June 2002 to December 2005, he was seconded to the Institute for Infocomm Research, Agency for Science, Technology and Research, Singapore, as a Program Director and Department Manager, helping to establish a research group in the area of optical communication and fiber devices. Since April 2006, he has been a Professor with the Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hong Kong. His research interests include optical communication systems and networks, fiber devices for optical communication, and sensor systems.

Alan Pak Tao Lau received his B.A.Sc. degree in Engineering Science (Electrical option) and M.A.Sc. degree in Electrical and Computer Engineering from University of Toronto, Canada, in 2003 and 2004, respectively. He obtained his Ph.D. degree in Electrical Engineering from Stanford University, USA, in 2008. He joined The Hong Kong Polytechnic University in 2008 as an Assistant Professor. He is now a Professor and his current research interests include long-haul and short-reach coherent optical communication systems, optical performance monitoring and machine learning applications in optical communications and networks. He collaborates extensively with industry in various aspects of optical communications.