CHALMERS
UNIVERSITY OF TECHNOLOGY

End-to-end Autoencoder Learning for fiber-optic communication systems

Master's thesis in Master Programme of Communication Engineering

SHEN LI

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Master's thesis 2018:NN

End-to-end Autoencoder Learning for Fiber-optic Communication Systems

Shen Li

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Master Programme of Communication Engineering
Chalmers University of Technology
Gothenburg, Sweden 2018

End-to-end Autoencoder Learning for Fiber-optic Communication Systems
Shen Li

Supervisor: Henk Wymeersch, Christian Häger
Examiner: Henk Wymeersch

Master's Thesis 2018:NN
Department of Electrical Engineering
Master Programme of Communication Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 79 039 8286

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2018

End-to-end Autoencoder Learning forFiber-optic Communication Systems
Shen Li
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Modulation formats that can achieve high spectral efficiency like multi-level quadrature amplitude modulation (M-QAM) are employed by fiber-optic communication systems to increase data rates. However, M-QAM has more compacted constellations, which means it's more sensitive to noise and higher input power is needed. Kerr effect becomes one of the main difficulties as it constrains the optimal input power. Autoencoders (AEs) in machine learning field is a promising tool to jointly optimize the transmitter and the receiver in a single process to not only realize constellation shaping but also design an optimal receiver.

In this thesis, we develop an AE for a simplified memoryless fiber channel model. The AE can approach maximum likelihood (ML) performance and leads to optimized constellations that are more robust against nonlinear signal-noise interaction (NLSNI) than conventional quadrature amplitude modulation (QAM) formats. Moreover, it is shown that the AE approach can be used to establish tight lower bounds on the channel capacity by computing achievable information rates (AIRs).

Keywords: fiber-optic communication, autoencoder, neural networks, nonlinear phase noise, achievable information rate.

Acknowledgements

I would like to thank Prof. Henk Wymeersch and Dr. Chistian Häger first who proposed this thesis topic and gives me this valuable opportunity to do this master thesis project in which I really learned a lot.

I would like to sincerely express my gratitude to my two supervisors Henk Wymeersch and Christian Häger for their continuous guidance, patience and encouragement not only on the academic research, but also on my life. I also appreciate the help of Nil Garcia for a lot of inspiring discussions about my thesis work. Thank you all again a lot for the work and guidance on the writing of the conference paper as well as the thesis report. Moreover, I am genuine thankful to Chalmers C3SE to provide the PC-cluster which saves a lot of my time for running simulations.

In a word, I sincerely thank all the people who were involved during my one-year master thesis. This thesis experience would be unforgettable for me as I really has harvested a lot of new knowledge and gain confidence and inspiration of life with the help of them all.

Shen Li, Gothenburg, June 2018

Contents

List of Figures **xi**

1 Introduction **1**
1.1 Limits of nonlinear fiber-optical communication 1
1.2 Conventional approaches 3
1.3 Machine learning approaches 3
1.4 Goal of the Thesis 3
1.5 Ethics in machine learning 4

2 Theory **5**
2.1 Nonlinear phase noise in fiber-optic channels 5
2.2 Maximum likelihood detection for nonlinear fiber-optic channels . . . 5
2.3 Neural networks basics 6
2.3.1 Basic structure and components of neural networks 7
2.3.2 Autoencoder 15
2.3.3 TensorFlow 17

3 Proposed autoencoder structure **19**
3.1 Introduction 19
3.2 Proposed AE structure 19
3.3 AE constellations and detectors 20
3.4 AE for bounding the capacity 20

4 Performance Analysis **23**
4.1 Simulation scenario 23
4.2 Selection of number of layers and activation functions 23
4.3 Symbol Error Rate 25
4.4 Learned constellations 27
4.5 Achievable Information Rate 29

5 Conclusion **31**

Bibliography **33**

List of Figures

1.1 Received 16 QAM constellation with nonlinear phase noise 2
1.2 SER as a function of input power in the presence of nonlinear phase noise for 16 QAM assuming that the fiber length is 100 km, the nonlinear parameter is 1.27, the total noise power is −21.3 dBm and the number of iterations to simulate the model is 50. 2

2.1 A simple artificial neural network 7
2.2 The structure of a neuron 8
2.3 Activation functions 8
2.4 Illustration of gradient descent on a level set 11
2.5 Illustration of stochastic gradient descent on a level set 11
2.6 Backpropagation: First, the input $a^{[0]}$ is fed into the neural network and forward-propagated until the loss function $L(\hat{y}, y)$ at the end. Then the gradients of each layers' weights and biases $dW^{[l]}$ and $db^{[l]}$ are calculated from the end to the front. Finally, all the weights and biases are updated according to the gradients. 14
2.7 Illustration of the structure of an AE 16
2.8 General structure of a DAE 17
2.9 Data flow graph of an one-layer neural network in TensorFlow 18

3.1 A simpliest communication system 19
3.2 Autoencoder structure assuming 2 hidden layers in both the transmitter and receiver neural network 20

4.1 Data flow graph of the AE for $M = 16$ in Tensorflow 24
4.2 SER as a function of number of training iterations for appropriate network patameters (left) and insufficient parameters (right) for $M = 16$. 25
4.3 SER as a function of P_{in} for $M = 16$. 26
4.4 ML decision boundaries for the AE constellation at $P_{\text{in}} = 0$ dBm (left) and learned AE decision regions (right). 27
4.5 Learned 16-point constellation for AWGN channel 27
4.6 Learned 16-point constellations for the nonlinear fiber channel under different P_{in} 28
4.7 Learned 256-point constellations for the nonlinear fiber channel under different P_{in} 29

4.8 Comparison of the AIR of the AE to various information-theoretic capacity bounds and 16-QAM. 30

1

Introduction

1.1 Limits of nonlinear fiber-optical communication

Nowadays, the amount of information is increasing rapidly. In order to increase the transmission speed and information capacity, optical fiber communication has become the most important way of wired communication and provides for over 99% of global data traffic because of its high transmission bandwidth, large capacity, low transmission loss and long transmission distance. Fiber transmission rates can be increased by using high order constellations, like multi-level quadrature amplitude modulation (M-QAM). However, these constellations are more compact and require higher input power [1] and thus more susceptible to nonlinear impairments such as nonlinear signal-noise interaction (NLSNI). Due to the Kerr effect [2], the fiber channel will cause a rotation of the transmitted constellation symbols and the rotation degree is proportional to the symbols' energy. Thus, the nonlinear Kerr effect constrains the optimal input power of signals and becomes one of the main difficulties of fiber-optical communication.

To visualize the effect on symbols, a 16-QAM constellation pattern after going through a channel with Kerr effect and amplified spontaneous emission (ASE) noise is shown in Figure 1.1. Symbols with high energy rotates more than those with lower energy. Figure 1.2 shows the symbol error rate (SER) as a function of input power for the linear additive white Gaussian noise (AWGN) channel and the nonlinear fiber channel only considering the Kerr effect and ASE noise. Both systems use the Euclidean distance detection. From this figure, the SER of nonlinear fiber channel first decreases as the input power increases like the linear channel because the signal-to-noise ratio (SNR) is higher and then goes up again as the nonlinear effect becomes the main distortion.

Due to NLSNI, the challenge is that the optimal constellation and the optimal receiver for a fiber communication system are unknown. The achievable transmission rates are themselves upper-bounded by the channel capacity, which is also unknown for optical channels with NLSNI, even for simplified nondispersive scenarios, though upper [3] and lower [3–5] capacity bounds have been established for the simplified non-dispersive channel.

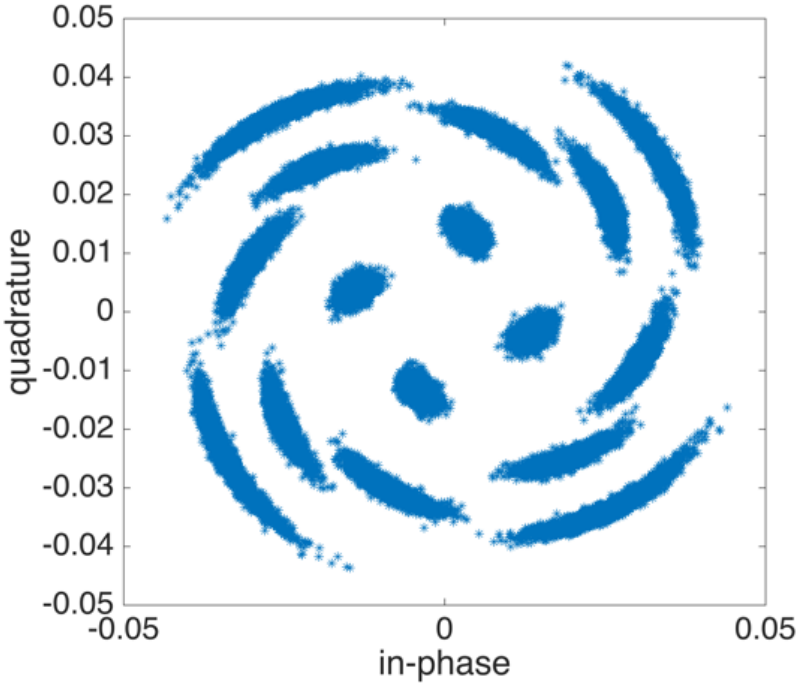

Figure 1.1: Received 16 QAM constellation with nonlinear phase noise

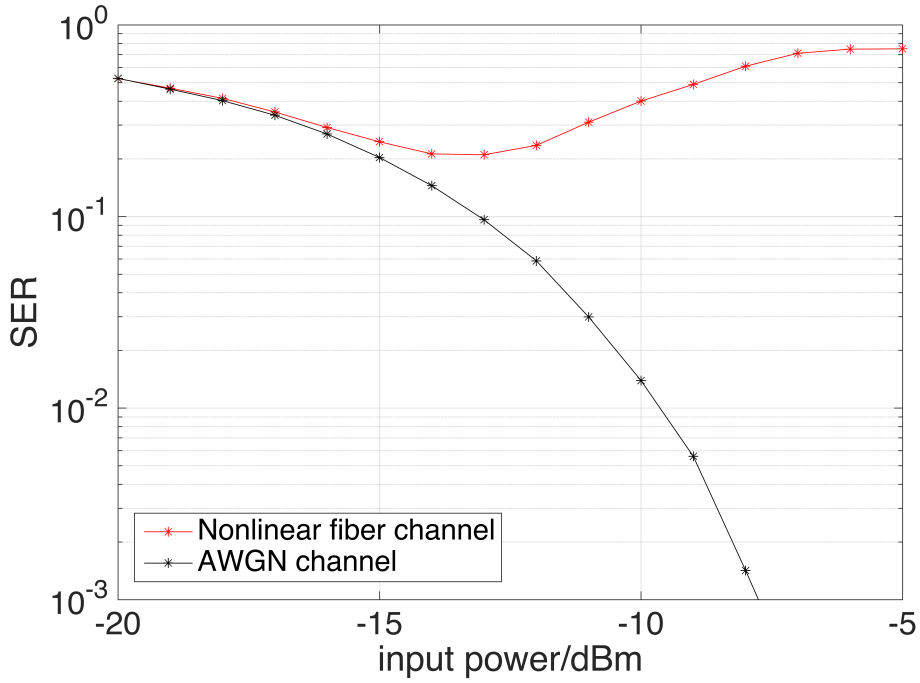

Figure 1.2: SER as a function of input power in the presence of nonlinear phase noise for 16 QAM assuming that the fiber length is 100 km, the nonlinear parameter is 1.27, the total noise power is -21.3 dBm and the number of iterations to simulate the model is 50.

1.2 Conventional approaches

Conventional techniques to deal with NLSNI include improved detector designs [1, 6, 7] and optimized modulation formats [7–9] (constellation shaping). In order to avoid complexity and focus on the Kerr nonlinearity, simplified memoryless and dispersionless models are often studied. In [7], the expression of the maximum likelihood (ML) detection boundaries are derived and 4-point constellation formats different from 4-QAM are studied, yielding better SER performance than 4-QAM. A closed-form ML-based detector for such a channel model was developed in [1] and the SER performance was compared with those of some other common sub-optimal detectors, showing a better trade-off of complexity and performance. In [8], a "joint optimization" approach was used to optimize amplitude-phase shift keying (APSK) constellations under a improved version of the two-stage (TS) detector [7] including the amplitude direction, a phase rotation and the phase direction. This constellation shaping approach used brute-force gird search to first list all the possible cases in which how many constellation rings there are and how the points distributed in those rings and then determines the optimal radii of all the rings for each case. The limitation of these methods is that they only focus on the design of either the detector or the transmitter (constellation shaping), but cannot jointly optimize both of them at the same time.

1.3 Machine learning approaches

A different approach for constellation or detector design is to rely on machine learning and deep learning, including [10–13, 15–17]. Recently, autoencoders (AEs) have emerged as a promising tool for end-to-end design and have been shown to lead to good performance for wireless [10, 11], noncoherent optical [16], as well as visible light communication [17]. A new way was first proposed in [10] to regard communication system design as a single optimization process using deep neural networks and learns interesting constellations for the linear AWGN channel. In [13], deep neural network is used to unroll the split-step Fourier method (SSFM) of digital backpropagation (DBP) [14]. The result shows that the learned DBP significantly reduces the complexity of conventional DBP. In [16], end-to-end AE learning for optical communication systems was first implemented by experiment. The AE learns the samples of the transmitted wave form as well as a good receiver resulting in better performance than some conventional modulations and receivers.

1.4 Goal of the Thesis

Different from conventional sub-optimal approaches, AE can jointly optimize the transmitter and the receiver in a single process. In this thesis, we apply an AE to a simplified memory-less fiber channel model similar to [1, 3–8]. As the channel model we used is simplified, the channel probability density function (PDF) is known analytically [4–6], which allows us to get access to an ML decoder. With this benchmark, the goal is to find a good constellation format and a near-ML receiver

for this channel and then good performance will follow. The performance is evaluated by comparing the SERs and achievable information rates (AIRs) [18–21] of the systems using conventional constellation formats like M-QAM and an ML decoder. Moreover, we use AE to find good input distributions and auxiliary channels distribution to create a lower bound of channel capacity by calculating the achievable information rate (AIR).

1.5 Ethics in machine learning

Continuous progress and widespread use of machine learning brings huge advantages to human society. However, there are still some underlying ethical issues we cannot neglect.

First, machine learning usually needs a lot of training data to learn algorithms, which leads to privacy concern. The training data can be some personal information which is not open source and public. For example, a search engine algorithm may use internet users' sensitive information including searching habits, searching content, et cetera, to learn and recommand users' content of interest. Users' personal data may be stolen by other services and used illegally.

Second, algorithms can be biased and discriminational [22]. Algorithms sometimes are actually predicting future trends according to previous data. The algorithm model and the input of the algorithm determine the result of prediction. If the training dataset is discriminational, incomplete, or even incorrect. The trained algorithm using this kind of training data will also be biased. On the other hand, the design of algorithms itself may contain discrimination of developers. No matter where the discrimination is from, algorithms may retain and magnify the discrimination. For example, if discriminational algorithms are used for crime risk assessment or credit assessment, unfair results will cause a loss to people.

Third, safety and responsibility problem is another issue. For example, if a traffic accident happens to a self-driving car, who should take the responsibility? How the safety of artificial intelligence products can be guaranteed?

In this thesis, machine learning is used in fiber-optic communication systems to find a good set of transceiver and the above-mentioned ethical problems will not happen in our work.

2
Theory

In this chapter, we first briefly introduce the nonlinear phase noise in fiber-optic channels. The basic theory behind deep learning such as the components of a neural network, different optimizer algorithms and backpropagation is discussed. We also introduce the concept of AEs in machine learning field and the software library that we use in this thesis work.

2.1 Nonlinear phase noise in fiber-optic channels

The propagation of signals in a fiber with ideal distributed amplification is modeled by the nonlinear Schrödinger equation (NLSE) [23].

$$\frac{\partial \mathbf{x}(z,t)}{\partial z} = i\gamma \left\|\mathbf{x}(z,t)\right\|^2 \mathbf{x}(z,t) - i\frac{\beta_2}{2}\frac{\partial^2 \mathbf{x}(z,t)}{\partial t^2} - \mathbf{n}(z,t) \tag{2.1}$$

where $\mathbf{x}(z,t) \triangleq [\mathbf{x}_x \ \mathbf{x}_y]^\intercal$ is the transmitted signal, z and t are time and distance coordinates, γ is the nonlinearity parameter, β_2 is the group velocity dispersion coefficient, $\mathbf{n}(z,t)$ is Gaussian noise. The first term on the right side of the equation represents the Kerr effect in fiber-optic system causing a phase shift which is proportional to the signal's power. This nonlinear phase noise is one of the most important distortion in fiber-optic communication systems. The second term on the right side of the equation represents dispersion. Since the likelihood function of the channel in Eq. 2.1 is unknown, we consider a simplified nondispersive memoryless channel which is obtained by neglecting β_2 in Eq. 2.1. The resulting per-sample model is defined by the recursion [3]

$$x_{k+1} = x_k e^{jL\gamma|x_k|^2/K} + n_{k+1}, \quad 0 \le k < K, \tag{2.2}$$

where $x_0 = x$ is the (complex-valued) channel input, $y = x_K$ is the channel output, $n_{k+1} \sim \mathcal{CN}(0, P_N/K)$, L is the total link length, P_N is the noise power, and γ is the nonlinearity parameter. The model assumes ideal distributed amplification and $K \to \infty$.

2.2 Maximum likelihood detection for nonlinear fiber-optic channels

ML detection, which is widely used for linear channels, requires the PDF of the received signal, but for nonlinear fiber-optic channels, the PDF of the received signal

is hard to compute. However, for the simplified channel model in (2.2), the channel PDF is known analytically. In [7], ML decision boundaries are derived for a similar channel model only considering nonlinear phase noise and using distributed amplification.

Let $x_i \in \Omega^2$ be one transmitted symbol from constellation pattern Ω^2 with input power P and phase θ_o and y be the received signal with received amplitude r and phase θ. σ^2 denotes the variance of the ASE noise and r denotes the received electric field amplitude divided by σ. The conditional PDF $p(y|x_i)$ is the joint PDF of r and θ [6, 7]:

$$p(y|x_i) = f_{P,\theta_o}(r,\theta) = \frac{f_R(r,P)}{2\pi} + \frac{1}{\pi}\sum_{m=1}^{\infty} \mathbf{Re}\left\{C_m(r)e^{jm(\theta-\theta_o)}\right\} \tag{2.3}$$

where

$$f_R(r,P) = 2re^{-(r^2+P/\sigma^2)}I_0(2r\sqrt{P/\sigma^2}) \tag{2.4}$$

is the Rice PDF of r and $C_m(r)$ is the Fourier coefficient:

$$C_m(r) = \frac{r\sec\sqrt{jmx}}{s_m}e^{\sqrt{P/\sigma^2}\sqrt{jmx}\tan\sqrt{jmx}}e^{-\frac{r^2+\alpha_m^2}{2s_m}}I_m(\frac{\alpha_m r}{s_m}) \tag{2.5}$$

where $x = \frac{\gamma PL}{\sqrt{P/\sigma^2+1/2}}$, $\alpha_m = (P/\sigma^2)^{1/4}\sec\sqrt{jmx}$, $s_m = \frac{\tan\sqrt{jmx}}{2\sqrt{jmx}}$ and $I_m(\cdot)$ is the mth-order modified Bessel function of the first kind.

With the known fiber channel PDF, the ML detector is defined as:

$$\hat{x}_i = \underset{x_i\in\Omega^2}{\arg\max}\ p(y|x_i) \tag{2.6}$$

After receiving a signal y, the ML detector find the x_i that yields the largest conditional PDF $p(y|x_i)$, thus realizing ML detection.

2.3 Neural networks basics

Artificial Neural Network (ANN) was inspired by biological neural networks that exist in animal neural systems and brains. ANN has a long history of evolution, and it is widely used in nowadays research. In 1943, Warren McCulloch and Walter Pitts proposed a neuron model that outputs either 1 or 0 based on a threshold value, and successfully modeled NOT/OR/AND logic functions [24]. In 1958, Rosenblatt proposed the concept of perceptron whose output depends on the linear combination of inputs and weights. By adjusting the weights, it can solve linearly separable classification problem [25]. Geoffrey Hinton solved non-linear classification problem by introducing sigmoid function into a perceptron and using backpropagation for training [26], which is the prototype of more advanced ANN model, such as Recurrent Neural Network (RNN), Convolution Neural Network (CNN) , sparse auto-encoding, etc.

Universal approximation theorem [27] shows that an ANN with single hidden layer has ability to approximate continuous functions on compact subsets $\mathbb{R}^N$, and people found that ANNs with multiple hidden layers always have better performance than ANNs with single hidden layer. A deep learning model or a Deep Neural Network (DNN) is simply an ANN with multiple hidden layers, and these is no clear definition about the number of hidden layers that a DNN has. Compared with conventional algorithmic approaches, DNN can be trained and adjust itself automatically to solve the problem at hand, avoiding developing extremely complicated algorithms. Structures, activation functions and training rules of DNN may be modified according to different applications, and we can find the information of interest that are deeply hiding inside of the training data. For example, RNN is widely used in Natural Language Processing (NLP) [28], CNN is heavily used in image processing such as pattern recognition and image identification [29], sparse auto-encoding can be used for representation learning [30], etc.

2.3.1 Basic structure and components of neural networks

2.3.1.1 Neural network structure

ANNs are mathematical models or computing models simulating the structures and functions of biological neural networks [31] . An ANN usually comprises an input layer, an output layer and one or more hidden layers. A simple ANN with 3 hidden layers is illustrated in Figure 2.1. Each layer consists of many computing units called neurons. Figure 2.2 shows the structure of a single neuron.

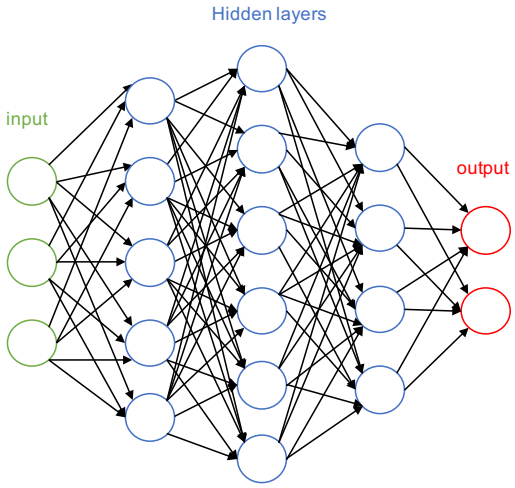

Figure 2.1: A simple artificial neural network

The neuron takes inputs from the previous layer and generates an output according to $y = f(\mathbf{w}^\intercal \mathbf{x} + b)$, where $\mathbf{w}$ is a vector of weights, $\mathbf{x} \triangleq [x_1\ x_2\ ...\ x_n]$, $b \in \mathbb{R}$ is a bias, and $f(\cdot)$ is an activation function. The role of nonlinear activation functions is to make the network nonlinear so that the neural network can approximate arbitrary complex functions. If we don't use them, the neural network will be always linear and there is no point to use many layers to simulate complex computing processes. Table 2.1 lists some widely used activation functions.

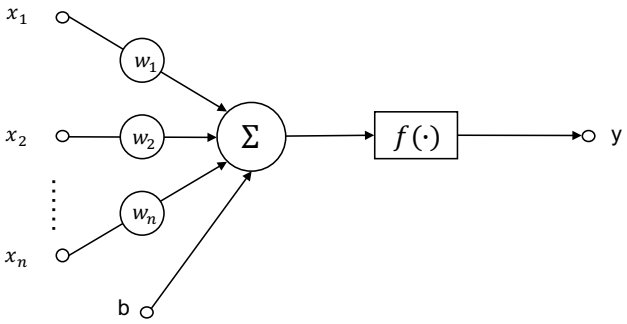

Figure 2.2: The structure of a neuron

Table 2.1: List of some common-used activation functions

Name	$f(x)$	Range
Identity	x	$(-\infty, \infty)$
Sigmoid	$\frac{1}{1+e^{-x}}$	$(0, 1)$
TanH	$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$(-1, 1)$
ReLU	$\max(x, 0)$	$[0, \infty)$
Softmax	$\frac{e^{x_i}}{\sum_i e^{x_i}}$	$(0, 1]$

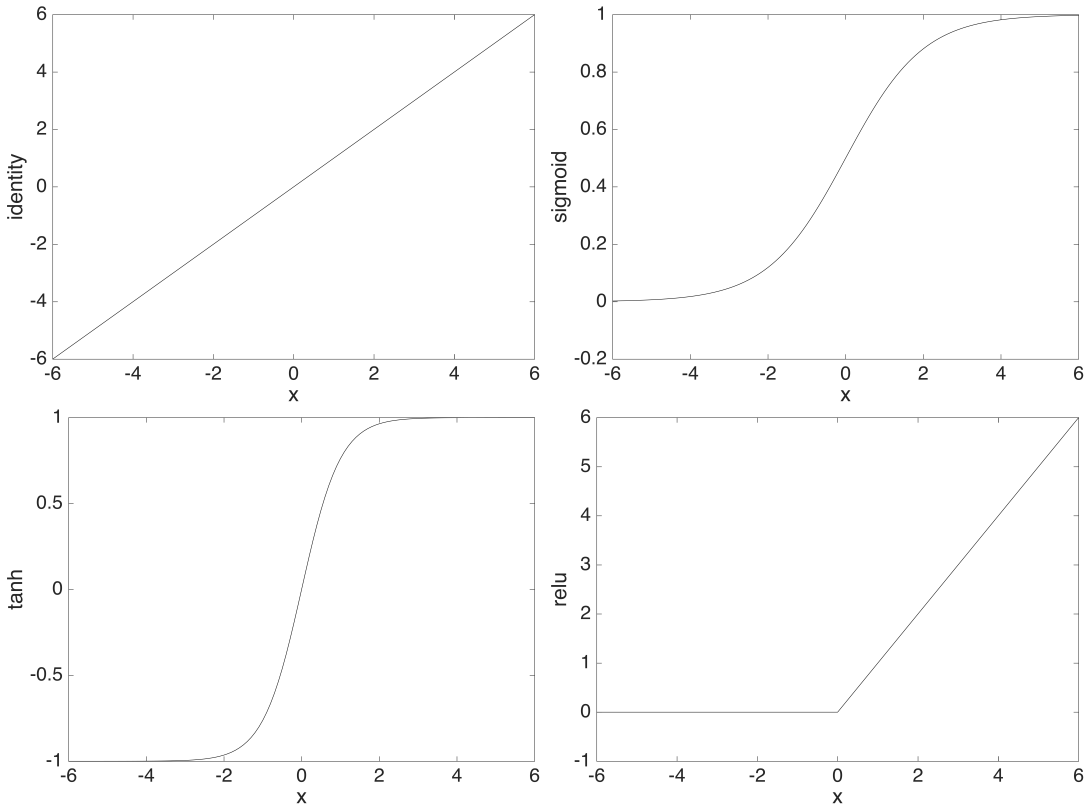

Figure 2.3: Activation functions

Different activation functions have different features. For instance, as shown in Figure 2.3, sigmoid function will saturate (the gradient of sigmoid function is close to zero) when the input is very large, which results in slow convergence rate because the gradient of the activation function is needed to update parameters in backpropagation (details in 2.3.1.5). TanH function also has this saturated gradient problem but TanH is better than sigmoid because its output is zero-centered. However, the computation complexity of both sigmoid and TanH are high as they have to do exponential computation. ReLU function is much simpler and computationally more efficient than sigmoid and TanH. ReLU can also avoid the saturated gradient problem as it is linear for $x > 0$. Nevertheless, the biggest drawback of ReLU is called "dying ReLU" problem. When the input of a ReLU neuron is less than zero, the gradient will also be zero during the backpropagation causing that the weights of this neuron will not be updated. If too many such neurons die, the neural network will not learn any more.

2.3.1.2 Machine learning tasks

The task of machine learning is to establish mathematic models such as neural networks to learn or predict from data. The establishment of the neural network structure is directly based on the training data, a set of input data used for training the network. Typically, machine learning tasks are classified into two broad categories, supervised learning and unsupervised learning depending on whether the training data is labeled (i.e., data is classified) or there is a feedback for each training data [32].

Supervised learning
In standard supervised learning tasks, the input data always have labels which are the desired output of the input. Given a set of N training examples $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_N, y_N)\}$, where $\mathbf{x}_i$ refers to the i-th training example's feature vector and y_i refers to the i-th training example's label, the goal is to find a function $f : X \to Y$ that maps the input data to their labels as much as possible.

Different from standard supervised learning tasks, If the training data is not labeled but there is a feedback (reward or punishment) corresponding to each training data, this kind of learning tasks belongs to reinforcement learning. Reinforcement learning seeks a set of actions that the agent should take in a dynamic environment to maximize the cumulative rewards and get the best performance.

Unsupervised learning
In unsupervised learning, the input data don't have any labels. The goal can be finding hidden laws or learning features of the input data. Typical algorithms used in unsupervised learning includes clustering [33], autoencoders, generative adversarial networks (GAN) [34], self-organizing map (SOM) [35], adaptive resonance theory (ART) [36, 37], etc.

2.3.1.3 Cost function

In order to train a neural network, a loss function is needed to calculate the difference, the so-called "loss" between the real output and the desired output of each training example. For instance, in supervised learning tasks, the loss is calculated between the neural network's output $\hat{y}_i$ and their labels y_i for each training example. The following equation shows two common-used loss functions. A cost function $J(\mathbf{W}, \mathbf{b}) = \frac{1}{N}\sum_{i=1}^{N} L(\hat{y}_i, y_i)$ parameterized by $\mathbf{W}$ and $\mathbf{b}$ is the average loss of all the training examples, where N is the number of training examples. The neural network can be trained by a certain optimizer to find the best weights and bias to minimize the objective cost function.

$$L(\hat{y}_i, y_i) = \begin{cases} (\hat{y}_i - y_i)^2, & \text{MSE} \\ y_i \log(\hat{y}_i) & \text{Cross entropy} \end{cases} \tag{2.7}$$

Sometimes a regularization item is added to the cost function to avoid over-fitting. Eq. 2.8 shows a common regularization method called L2 regularization:

$$J(\mathbf{W}, \mathbf{b}) = \frac{1}{N}\sum_{i=1}^{N} L(\hat{\mathbf{y}}_i, \mathbf{y}_i) + \frac{\lambda}{2N}\|\mathbf{W}\|_2^2 \tag{2.8}$$

where λ is the regularization parameter and $\lambda > 0$.

2.3.1.4 Practical optimizers

Some of the most popular and common optimizers used for deep learning are introduced in the following.

Gradient descent

Gradient descent is one of the most common ways to minimize an objective function $J(\theta)$ by updating the parameters θ recursively in the direction of the negative gradient of $J(\theta)$ at the current point, where θ are the weights and biases of the neural network. For example, suppose $J(\theta)$'s graph has a bowl shape, from a top view shown in Fig. 2.4 [38], $J(\theta)$ is represented by a set of blue contour lines on which the value of $J(\theta)$ is constant. The red arrows represent the direction of the negative gradient at some points. We start from a point θ_t at which the negative gradient is $-\nabla_\theta J(\theta_t)$. Then the parameter θ is updated by:

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta J(\theta_t) \tag{2.9}$$

where α is a learning step size. Finally, gradient descent will lead to the minimal value of $J(\theta)$, that is around the smallest circle in Fig. 2.4.

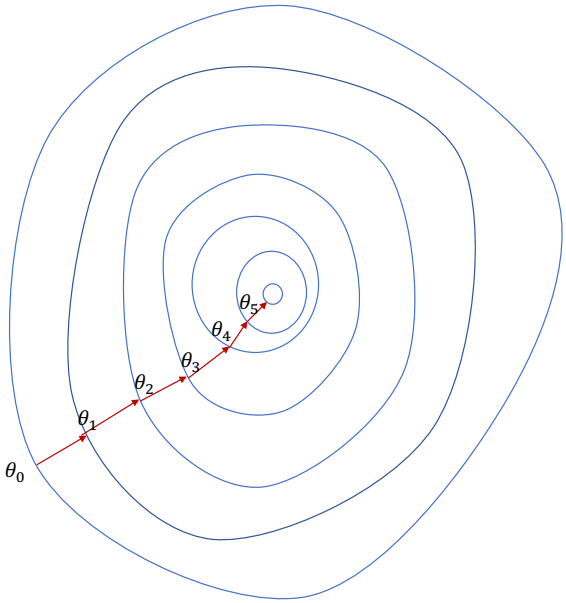

Figure 2.4: Illustration of gradient descent on a level set

If we calculate the negative gradient of our loss function $L(\theta)$ over the entire training set, that is to average the gradients of all training data points, the method is batch gradient descent (BGD). The direction of the calculated average negative gradient using BGD is the fastest direction to reach a local minimum. However, calculating the loss of all data points to perform only one update is inefficient when the training data set is large as every data point's gradient is required. Stochastic gradient descent (SGD) is designed to accelerate the training speed. SGD method updates the parameter by calculating the loss of only one stochastic training example (x_i, y_i) [39]:

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta J_i(\theta_t; (x^{(i)}, y^{(i)})) \tag{2.10}$$

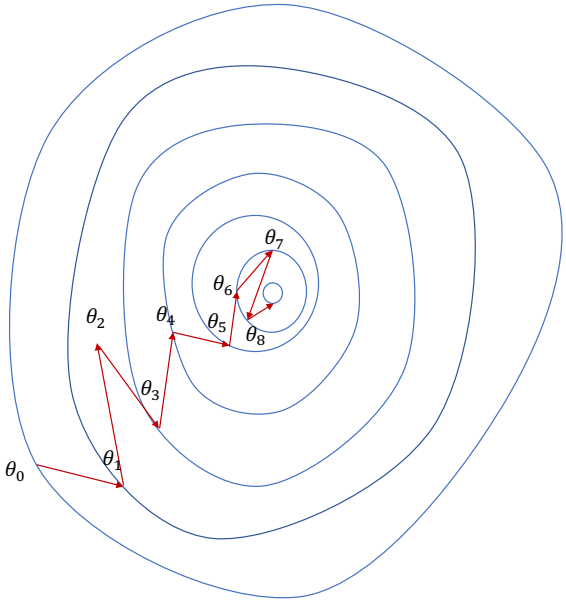

Figure 2.5: Illustration of stochastic gradient descent on a level set

Although SGD method needs more steps to reach the global minimum than BGD as the gradient is not the best, as shown in Fig. 2.5, the convergence is much faster when the training set is large because less computation is used for updating. Another compromise method is mini-batch gradient descent [39]. This method performs an update over a small set of n training examples $(x^{i:i+n}, y^{(i:i+n)})$:

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta J_i(\theta_t; (x^{(i:i+n)}, y^{(i:i+n)})) \tag{2.11}$$

Gradient descent with momentum

The learning steps of SGD or mini-batch gradient descent are fluctuated, which makes learning slow. Momentum is a method that takes past gradients into account to smooth out the steps of gradient descent and can accelerate the learning process [39]. First, we initialize $v_0 = 0$. The recursive updating rules at time t are:

$$v_t = \beta v_{t-1} + (1 - \beta) \nabla_\theta J_i(\theta_t) \tag{2.12}$$

$$\theta_{t+1} = \theta_t - \alpha v_t \tag{2.13}$$

where the factor β is called momentum and the most common value for β is 0.9. When $\beta = 0$, then method it's the same as gradient descent without momentum. Higher β means that more past gradients are taken into consideration so the learning steps are smoother.

RMSProp

Root mean square propagation (RMSProp) is another method that can also speed up mini-batch gradient descent. This method takes a moving average of the gradient's square and then divide the gradient by the square root of this mean square:

$$v_t = \beta v_{t-1} + (1 - \beta)(\nabla_\theta J_i(\theta_t))^2 \tag{2.14}$$

$$\theta_{t+1} = \theta_t - \alpha \frac{\nabla_\theta J_i(\theta_t)}{\sqrt{v_t + \epsilon}} \tag{2.15}$$

where ϵ is a small value used to avoid that the gradient is divided by zero.

Adam

Adam optimizer [40] is an important common-used algorithm proposed recently. This method not only take a exponentially weighted average of the gradient but also the gradient's square (the first moment and second moment of the gradient). Then update parameters in the same way as RMSProp but using the bias-corrected version of the gradient and gradient's square:

$$v_t = \beta_1 v_{t-1} + (1 - \beta_1) \nabla_\theta J_i(\theta_t) \tag{2.16}$$

$$s_t = \beta_2 s_{t-1} + (1 - \beta_2)(\nabla_\theta J_i(\theta_t))^2 \tag{2.17}$$

$$v_t^{corrected} = \frac{v_t}{1 - \beta_1^t} \tag{2.18}$$

$$s_t^{corrected} = \frac{s_t}{1-\beta_2^t} \tag{2.19}$$

$$\theta_{t+1} = \theta_t - \alpha \frac{v_t^{corrected}}{\sqrt{s_t^{corrected} + \epsilon}} \tag{2.20}$$

β_1 and β_2 are also two hyper-parameters need to be chose. It's proven to be effective for a wide variety of different kinds of neural network architectures.
Selecting an appropriate learning rate is important for training. If the learning rate is too large, the neural network will not converge while too small learning rates will make the training process very slow. The choice of the learning rate depends on the specific problem at hand. We might start with a large learning rate such as 0.1 to observe the training process, and then try exponentially smaller values 0.01, 0.001, etc. After knowing the reasonable smaller range of the learning rate for example $[0.001, 0.01]$, then we might uniformly sample from this range until finding the best suitable one.

Sometimes a learning rate can make the training process fast at first but fluctuated around the minimum and not really converged. If we change this learning rate into a smaller one, then the whole training process is not efficient. It is usually useful to reduce the learning rate gradually during the training process. Here are the formulas of some examples of reducing the learning rate:

$$\alpha = \begin{cases} \alpha_0 \frac{1}{1+\eta t} \\ \alpha_0 \eta^t \\ \alpha_0 \frac{c}{\sqrt{t}} \end{cases} \tag{2.21}$$

where α_0 denotes the initial value of the learning rate, t denotes the current number of training epochs, $\eta \in (0, 1)$ is the decay rate and $c > 0$ is a constant adjusted according to the need.

2.3.1.5 Backpropagation

The training process in deep learning is the process of minimizing the cost function. Gradient descent is the most common solution which needs to calculate gradients first. Backpropagation [41] is used to calculate and backpropagate all parameters' gradients accroding to which the parameters can be updated.

Let $\mathbf{W}^l$ and $\mathbf{b}^l$ denote the weight matrix and bias vector of the l^{th} hidden layer, $f^{[l]}(\cdot)$ denote the l^{th} layer's activation function, $f^{[l]'}(\cdot)$ is the derivative of $f^{[l]}(\cdot)$, $\mathbf{a}^l$ is the output vector of the l^{th} hidden layer and $\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l$. Suppose the neural network has $L+1$ layers in total (the 0^{th} layer represents the input layer and the L^{th} is the output layer). Fig. 2.6 illustrated how a neural network's parameters are updated through backpropagation.

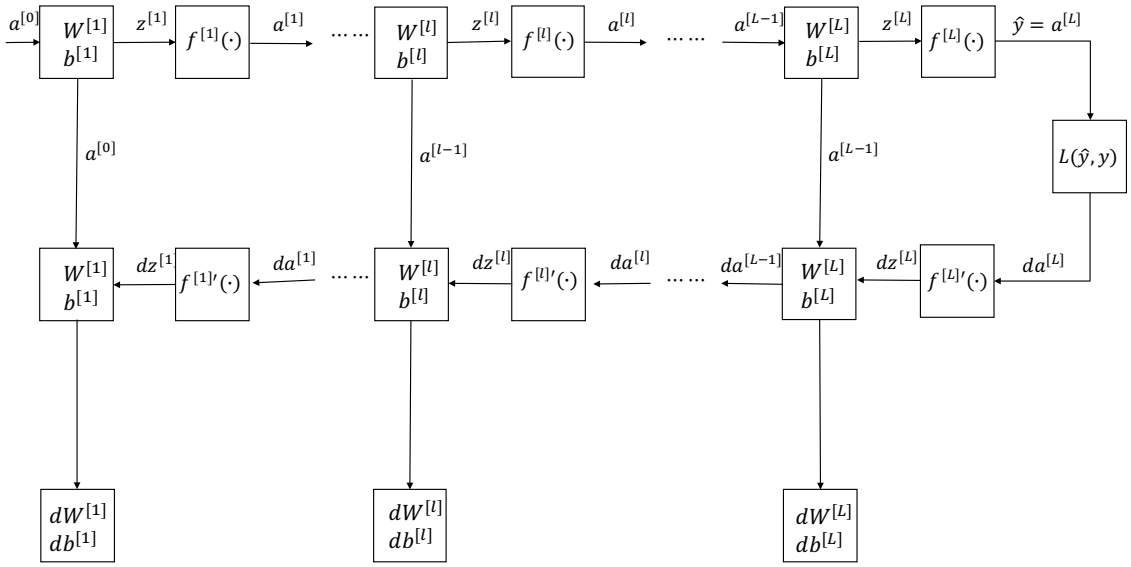

Figure 2.6: Backpropagation: First, the input $a^{[0]}$ is fed into the neural network and forward-propagated until the loss function $L(\hat{y}, y)$ at the end. Then the gradients of each layers' weights and biases $dW^{[l]}$ and $db^{[l]}$ are calculated from the end to the front. Finally, all the weights and biases are updated according to the gradients.

First, all the parameters are initialized. Then the loss $L(\mathbf{a}^{[L]}, \mathbf{y})$ is obtained after forward propagation. As the name suggests, we start calculating the gradient from the last layer. Let dx represents the partial derivative of the loss function $L(\hat{\mathbf{y}}, \mathbf{y})$ with respect to x. $d\mathbf{a}^{[L]}$ is easy to be obtained:

$$d\mathbf{a}^{[L]} = \frac{\partial L(\mathbf{a}^{[L]}, \mathbf{y})}{\partial \mathbf{a}^{[L]}} \tag{2.22}$$

According to chain rule, $d\mathbf{z}^{[L]}$ is calculated by:

$$d\mathbf{z}^{[L]} = \frac{\partial L(\mathbf{a}^{[L]}, \mathbf{y})}{\partial \mathbf{z}^{[L]}} = \frac{\partial L(\mathbf{a}^{[L]}, \mathbf{y})}{\partial \mathbf{a}^{[L]}} \frac{\partial \mathbf{a}^{[L]}}{\partial \mathbf{z}^{[L]}} = d\mathbf{a}^{[L]} f^{[L]'}(\mathbf{z}^{[L]}) \tag{2.23}$$

Similarly, we can get the derivative of the loss function with respect to $\mathbf{W}^{[L]}$ and $\mathbf{b}^{[L]}$:

$$d\mathbf{W}^{[L]} = \frac{\partial L(\mathbf{a}^{[L]}, \mathbf{y})}{\partial \mathbf{W}^{[L]}} = \frac{\partial L(\mathbf{a}^{[L]}, \mathbf{y})}{\partial \mathbf{z}^{[L]}} \frac{\partial \mathbf{z}^{[L]}}{\partial \mathbf{W}^{[L]}} = d\mathbf{z}^{[L]} \mathbf{a}^{[L-1]} \tag{2.24}$$

$$d\mathbf{b}^{[L]} = \frac{\partial L(\mathbf{a}^{[L]}, \mathbf{y})}{\partial \mathbf{b}^{[L]}} = \frac{\partial L(\mathbf{a}^{[L]}, \mathbf{y})}{\partial \mathbf{z}^{[L]}} \frac{\partial z^{[L]}}{\partial \mathbf{b}^{[L]}} = d\mathbf{z}^{[L]} \tag{2.25}$$

For the $(L-1)^{th}$ layer, according to $\mathbf{z}^{[L]} = \mathbf{W}^{[L]} \mathbf{a}^{[L-1]} + \mathbf{b}^{[L]}$, $d\mathbf{a}^{[L-1]}$ can be obtained by:

$$d\mathbf{a}^{[L-1]} = \frac{\partial L(\mathbf{a}^{[L]}, \mathbf{y})}{\partial \mathbf{a}^{[L-1]}} = \frac{\partial L(\mathbf{a}^{[L]}, \mathbf{y})}{\partial \mathbf{z}^{[L]}} \frac{\partial z^{[L]}}{\partial \mathbf{a}^{[L-1]}} = d\mathbf{z}^{[L]} \mathbf{W}^{[L]} \tag{2.26}$$

In general, we can have the following rules:

$$d\mathbf{z}^{[l]} = d\mathbf{a}^{[l]} f^{[L]'}(\mathbf{z}^{[L]}) \tag{2.27}$$

$$d\mathbf{a}^{[l-1]} = d\mathbf{z}^{[l]}\mathbf{W}^{[l]} \tag{2.28}$$

$$d\mathbf{W}^{[l]} = d\mathbf{z}^{[l]}\mathbf{a}^{[l-1]} \tag{2.29}$$

$$d\mathbf{b}^{[l]} = d\mathbf{z}^{[l]} \tag{2.30}$$

where $l = L, L-1, ..., 2, 1$.

If we use gradient descent and set a learning rate α, each layer's weights and biases at time t are updated by the following rules until the loss converges.

$$\mathbf{W}_{t+1}^{[l]} = \mathbf{W}_{t}^{[l]} - \alpha d\mathbf{W}_{t}^{[l]} \tag{2.31}$$

$$\mathbf{b}_{t+1}^{[l]} = \mathbf{b}_{t}^{[l]} - \alpha d\mathbf{b}_{t}^{[l]} \tag{2.32}$$

The reason why regularization works is that with a regularization item added to the cost function, the new gradient of the cost function $d\mathbf{W}'$ with respect to $\mathbf{W}$ will be $d\mathbf{W} + \frac{\lambda}{N}\mathbf{W}$, where $d\mathbf{W}$ represents the gradient of the cost function without regularization. Then $\mathbf{W}$ is updated by:

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \alpha d\mathbf{W}'_t = \mathbf{W}_t - \alpha(d\mathbf{W}_t + \frac{\lambda}{N}\mathbf{W}_t) = (1 - \frac{\alpha\lambda}{N})\mathbf{W}_t - \alpha d\mathbf{W}_t \tag{2.33}$$

Thus, $(1 - \frac{\alpha\lambda}{N})\mathbf{W}_t$ realizes a "weight decay". If the regularization parameter λ is big enough, the weights $\mathbf{W}$ will be limited to small values, the network will more linear to reduce over-fitting.

2.3.2 Autoencoder

In machine learning, an AE is an artificial neural network used for unsupervised learning to learn data codings [42]. The coding can be a lower representation of the data (e.g. an image), thus realizing dimensionality reduction. An AE usually consists of two parts: an encoder function $z = f_\theta(x)$ maps an input x to a code z and a decoder $g_\phi(f_\theta(x)) = \hat{x}$ attempts to reconstruct the input $\hat{x}$ from z. Figure 3.2 illustrates a simple structure of an AE. The difference from supervised learning is that the training data don't have any manual labels, or in other words, the labels are the data themselves. The learning process is to minimize the loss function:

$$L(x, g_\phi(f_\theta(x))) \tag{2.34}$$

Just copying the input seems to be meaningless. However, what we really care about is the code z. It can be a low-dimensional representation of the input, e.g. an image, and then the original input can be reconstructed only using this memory-efficient representation. Thus, AE realizes image compression. The dimension of the code z also can be the same as the input or even more than the input. Unfortunately, the AE sometimes merely copies the input and fails to extract anything useful of the input when there aren't any constraint [43]. Regularized AEs are designed to force the encoder and the decoder to learn useful information from the input data by adding items to the loss function or adding noise to the distribution of input data, such as sparse AEs and denoising AEs (DAEs) [44]. These variants of the AE

effectively prevent encoders and decoders from only copy their input to the output. Actually the above-mentioned low-dimensional representation method is also a kind of constraint under which the AE may learn the most prominent features with limited dimension of representation.

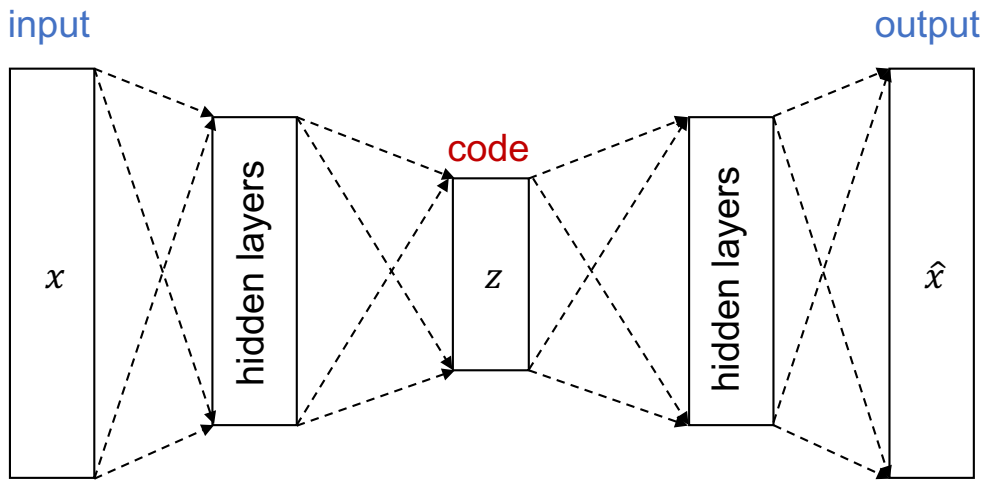

Figure 2.7: Illustration of the structure of an AE

Here we briefly introduce the DAE. Suppose the raw input data x has a distribution $p(x)$. The input of a DAE is a noisy version $\tilde{x}$ of the x. Figure 2.8 shows how a DAE works. The corrupted data $\tilde{x}$ is generated according to a conditional probability $c(\tilde{x}|x)$ or by setting some percentage of x to zeros. Then $\tilde{x}$ is fed to the encoder and the decoder, yielding a code $z = f_\theta(x)$ and an output $y = g_\phi(f_\theta(\tilde{x}))$. The loss function is defined as:

$$L(x, g_\phi(f_\theta(\tilde{x}))) \tag{2.35}$$

This approach force the AE to avoid learning identity functions. In [45], the idea is that the training process of a DAE is to estimate the conditional probability $p(x|\tilde{x})$. Then $p(x)$ can be obtained with knowing $p(x|\tilde{x})$ and $c(\tilde{x}|x)$ according to a Markov chain. It has been proved that the trained representation implicitly estimate the data generating distribution [45]. Thus the DAE can be used as a generative model to generate data samples from the learned distribution [43]. The trained representation also shows better robustness to noise and can be used to initialize a deep network [44].

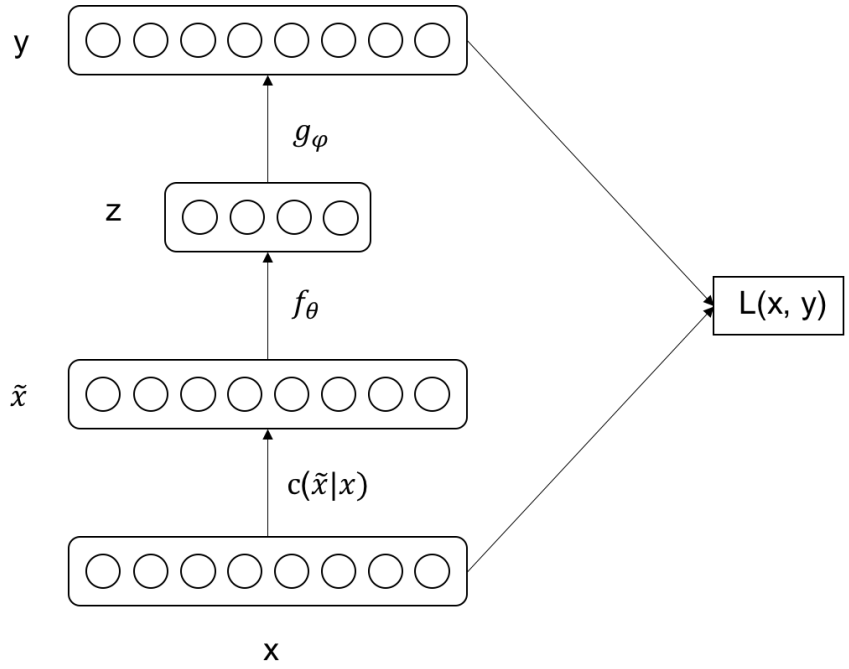

Figure 2.8: General structure of a DAE

2.3.3 TensorFlow

TensorFlow [46] is a open source software library used for artificial intelligence learning. Tensors means N-dimensional arrays and Flow implies calculation based on data flow graphs. As the name suggests, TensorFlow is the calculation process of the flow of tensors from one end of the flow graphs to the other. TensorFlow transmits complex data structures to artificial intelligence neural networks for analysis and processing. Figure 2.9 [46] illustrates how data flows of a simplest one-layer neural network in TensorFlow. The nodes represent computation units and edges imply incoming or output data of the nodes. When using TensorFlow, first the graph model needs to be constructed by setting up the relations among all the edges. Then data start to flow only when a session is created.

The biggest advantage of TensorFlow is that TensorFlow supports distributed computing for heterogeneous devices, which can greatly speed up learning process. TensorFlow can also automatically identify and execute operations that can be executed in parallel [46].

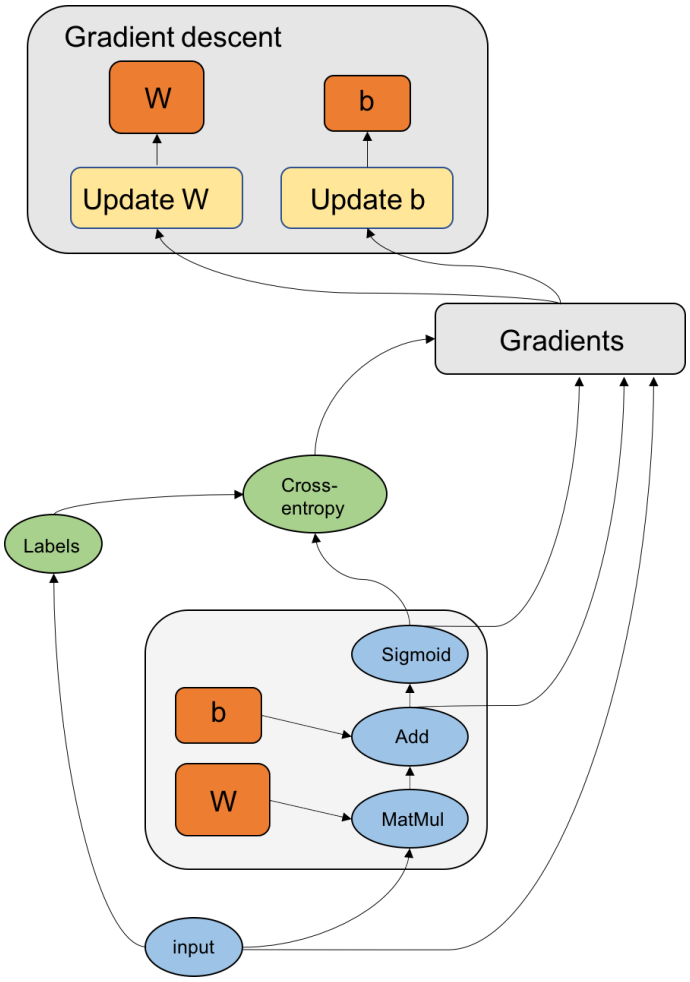

Figure 2.9: Data flow graph of an one-layer neural network in TensorFlow

3

Proposed autoencoder structure

3.1 Introduction

It has recently been proposed to interpret all components of a communication system, consisting of a transmitter, channel, and receiver, as an AE [10]. The goal of a communication system is to restore the same information as what has been transmitted, which is what exactly an AE does. Traditional components of a communication system including coding and decoding, modulation and demodulation are designed sub-optimal and we don't know if the whole communication system is joint optimized. The AE allows for end-to-end learning of good transmitter and receiver structures in a single process. We apply an AE to find good constellations and detectors for the channel model from Eq. 2.2.

3.2 Proposed AE structure

A simplest communication system basically comprises a transmitter, a channel and a receiver, as shown in Fig. 3.1. These components can be interpreted as an AE [10] in which the transmitter and the receiver can be jointly optimized through end-to-end learning.

Figure 3.1: A simpliest communication system

Fig. 3.2 illustrates the proposed auto-encoder structure. The goal is to transmit a message s chosen from a set of M possible messages $\{1, 2, ..., M\} \triangleq \mathcal{M}$. Following [10], the messages are first mapped to M-dimensional "one-hot" vectors where the s-th element is 1 and all other elements are 0. The one-hot vectors denoted by $\mathbf{u}$ are the inputs to a transmitter NN, which consists of multiple dense layers of neurons. The values of the two transmitter output neurons (z_r and z_i) in Fig. 3.2 are used to form the channel input. To meet the average power constraint, a normalization is applied using M different training inputs to the NN. The channel input x is drawn randomly from an M-point constellation with $\mathbb{E}\{|X|^2\} = P_{\text{in}}$, where P_{in} is the input power. Then the normalized output is assumed to be sent over the channel (2.2), leading to an observation y. The real and imaginary parts of y are taken as the input to a receiver NN, the output of which we denote by $f_y(s') \in [0, 1], s' \in \mathcal{M}$, where

we assume a sigmoid in the last layer and then normalize the sum of the output to 1. Finally, we set $\hat{s} = \arg\max_{s'} f_y(s')$.

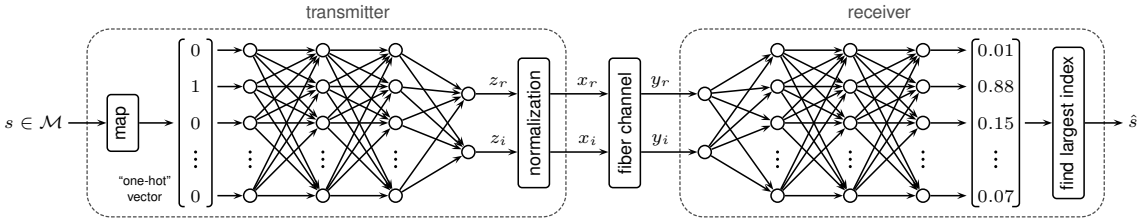

Figure 3.2: Autoencoder structure assuming 2 hidden layers in both the transmitter and receiver neural network

3.3 AE constellations and detectors

The AE is trained using many batches of training data averaging over different messages and channel noise configurations. In particular, the weights and biases of all neurons in both the transmitter and receiver NN are optimized with respect to $\frac{1}{N}\sum_{i=1}^{N} \ell(u_s^{(i)}, f_y(s')^{(i)})$, where

$$\ell(u_s^{(i)}, f_y(s')^{(i)}) = -u_s^{(i)} \log f_y(s')^{(i)}. \tag{3.1}$$

is the cross-entropy loss, N is the batch size (a multiple of M), and the superscript refers to different training data realizations, the subscript s refers to the s^{th} element of $\mathbf{u}^{(i)}$. The optimization is performed using a variant of stochastic gradient descent with an appropriate learning rate.

After training, the learned constellation can be obtained from the output of the normalization layer. $\mathbf{x}$ is two-dimensional and x_r and x_i can be regarded as the real and imaginary parts of the transmitted constellation points. The learned receiver part of the AE is a classifier that can decode message $\hat{s}$ from observed signal $\mathbf{y}$.

3.4 AE for bounding the capacity

For a given discrete channel, channel capacity is the supremum of the information rate at which signals can be transmitted through this channel with arbitrarily little errors when the information block length gose to infinity. Let random variables X and Y denote the input and output of the channel and $p_X(x)$ denotes the marginal distribution of X. Channel capacity is defined as [47]:

$$C = \lim_{N\to\infty} \frac{1}{N} \sup_{p_X(x)} I(X;Y) \tag{3.2}$$

where

$$I(X,Y) = \sum_x \int p(x,y) \log_2 \frac{p(y|x)}{p(y)} \mathrm{d}y \tag{3.3}$$

is the MI [18–21] between X and Y, channel is represented by the conditional probability distribution $p(y|x)$, $p(x, y)$ is the joint probability distribution of X and Y and N is the block length of information. From this definition, we can know that for a given channel model and a fixed modulation format, as well as the ML decoder for this channel, channel MI represents the highest information rate the communication system can reach, that is AIR of the system.

However, the problem is that we don't know the capacity of the fiber-optic channel because the channel MI is hard to calculate for complexity reasons [48]. A lower bound of channel MI can be obtained by replacing the channel law $p(y|x)$ with an auxiliary one $q(y|x)$ [49]. This method is called mismatched decoding. Both the channel MI and the AIR calculated by auxiliary channel law are lower bounds of the channel capacity. Even with known $p(y|x)$, the optimization of channel capacity in Eq. 3.2 over input distribution $p(x)$ is hard.

If we have a distribution $f_y(x)$ over x. Then, $f_y(x)p(y)$ is a valid joint distribution over x and y, so that, due to the non-negativity of the Kullback-Leibler divergence, $\mathrm{KL}(p(x, y)||p(y)f_y(x)) \geq 0$, that is:

$$\sum_x \int p(x, y) \log_2 \frac{p(x, y)}{p(y)f_y(x)} \mathrm{d}y \geq 0 \tag{3.4}$$

Then we can deduct it as follows:

$$\sum_x \int p(x, y) \log_2 (p(x, y) - p(y)f_y(x)) \mathrm{d}y \geq 0 \tag{3.5}$$

$$\sum_x \int p(x, y) \log_2 p(x, y) \mathrm{d}y \geq \sum_x \int p(x, y) \log_2 p(y)f_y(x) \mathrm{d}y \tag{3.6}$$

According to Bayes' theorem, replace $p(x, y)$ with $p(y|x)p(x)$:

$$\sum_x \int p(x, y) \log_2 p(y|x)p(x) \mathrm{d}y \geq \sum_x \int p(x, y) \log_2 p(y)f_y(x) \mathrm{d}y \tag{3.7}$$

Both sides divide $p(x)p(y)$ inside log yielding:

$$\sum_x \int p(x, y) \log_2 \frac{p(y|x)}{p(y)} \mathrm{d}y \geq \sum_x \int p(x, y) \log_2 \frac{f_y(x)}{p(x)} \mathrm{d}y \tag{3.8}$$

The right-hand side of (3.8) can be regarded as the AIR of the AE, which can easily be evaluated via Monte Carlo integration. Both the mutual information and the AIR are lower bounds on the channel capacity.

4

Performance Analysis

4.1 Simulation scenario

For the numerical results, we assume $L = 5000$ km, $\gamma = 1.27$, and $P_N = -21.3$ dBm in the fiber model (2.2). The number of iterations to simulate the model is set to $K = 50$, which is sufficient to approximate the true asymptotic channel PDF [6]. The AE is trained separately for different values of P_{in} using the Adam optimizer in TensorFlow. The AE structure parameters for $M = 16$ and $M = 256$ are summarized in Tab. 4.1 and Tab. 4.2. The data flow graph of the AE for $M = 16$ in Tensorflow is illustrated in Figure 4.1.

Table 4.1: Autoencoder parameters for $M = 16$

	transmitter			receiver		
layer	1	2	3	1	2–3	4
neurons	16	50	2	2	50	16
$f(\cdot)$	-	tanh	iden.	-	tanh	sigm.

Table 4.2: Autoencoder parameters for $M = 256$

	transmitter			receiver		
layer	1	2–6	7	1	2–7	8
neurons	256	256	2	2	256	256
$f(\cdot)$	-	tanh	iden.	-	tanh	sigm.

4.2 Selection of number of layers and activation functions

The number of neurons both in the input layer and output layer must be M because the input is M-dimensional "one-hot" vectors and the output is a distribution over the input. The chosen number of layers as well as the number of neurons in each hidden layer need to be enough to get good performance. For $M = 16$ system, the hyper-parameters are selected to get the minimum SER, while the goal of $M = 256$ system is to make the AIR as high as possible. Different activation functions and ways of initialization have different effects on the neural networks. The choice needs to be careful especially when the network is quite large like $M = 256$ system,

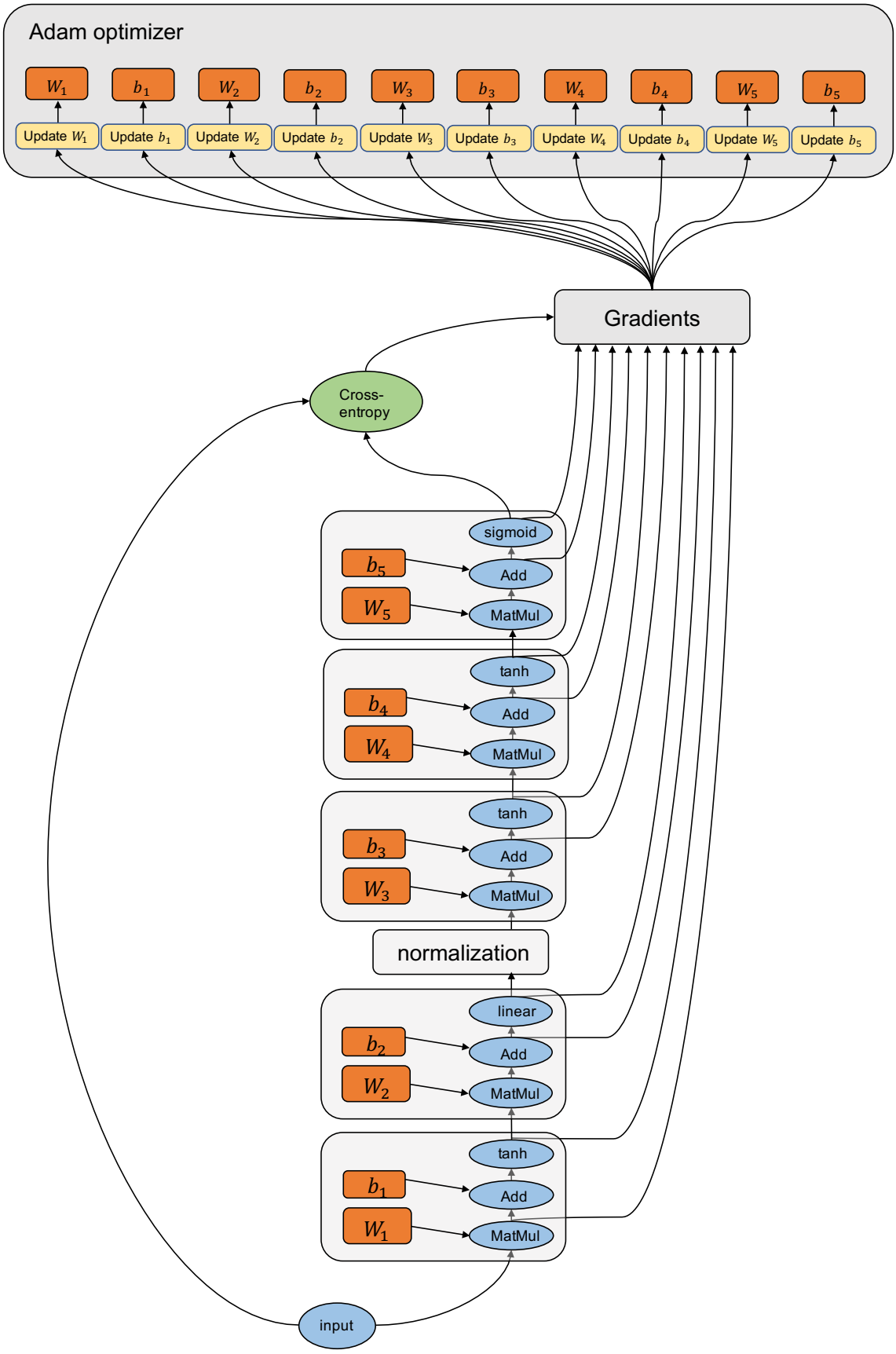

Figure 4.1: Data flow graph of the AE for $M = 16$ in Tensorflow

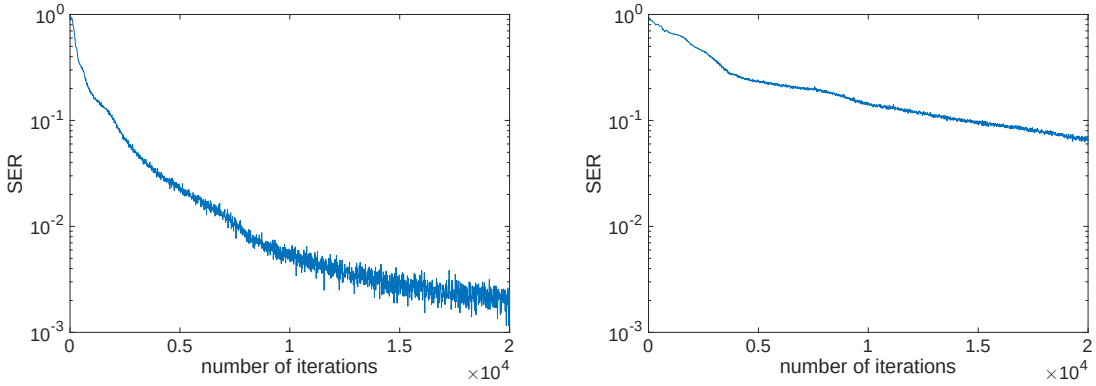

Figure 4.2: SER as a function of number of training iterations for appropriate network patameters (left) and insufficient parameters (right) for $M = 16$.

otherwise the network cannot converge, for example when encountered with "dying ReLU" problem or vanishing gradient problem. After trying different structures many times, we select the above-mentioned neural network structure which can lead to good performances. If without enough number of layers or neurons, the performance cannot converge to the global optima. For example, Figure 4.2 shows the SER as a function of number of training iterations for appropriate network parameters (Table 4.1) and insufficient parameters for $M = 16$.

4.3 Symbol Error Rate

We start by comparing the SER, i.e., $p(s \neq \hat{s})$, of the AE to the SER of an ML detector applied to (a) standard 16-QAM and (b) the signal constellation optimized by the AE. The results are shown in Figure 4.3. The optimal input power for 16-QAM under ML detection is around -2 dBm, after which the SER increases due to NLPN. The SER of the AE decreases with input power, showing that the AE can find more suitable constellations in the presence of NLPN. If we replace the receiver part of the AE with an ML detector, the SER improves only slightly. This indicates that the AE can not only learn good constellations, but also learn to approximate the correct channel distribution, thus achieving near-ML performance. To visualize this, in Fig. 4.4, we compare the effective decision regions implemented by the AE after training (right) to the optimal ML decision regions for the optimized AE constellation at $P_{\text{in}} = 0$ dBm (left), showing excellent agreement.

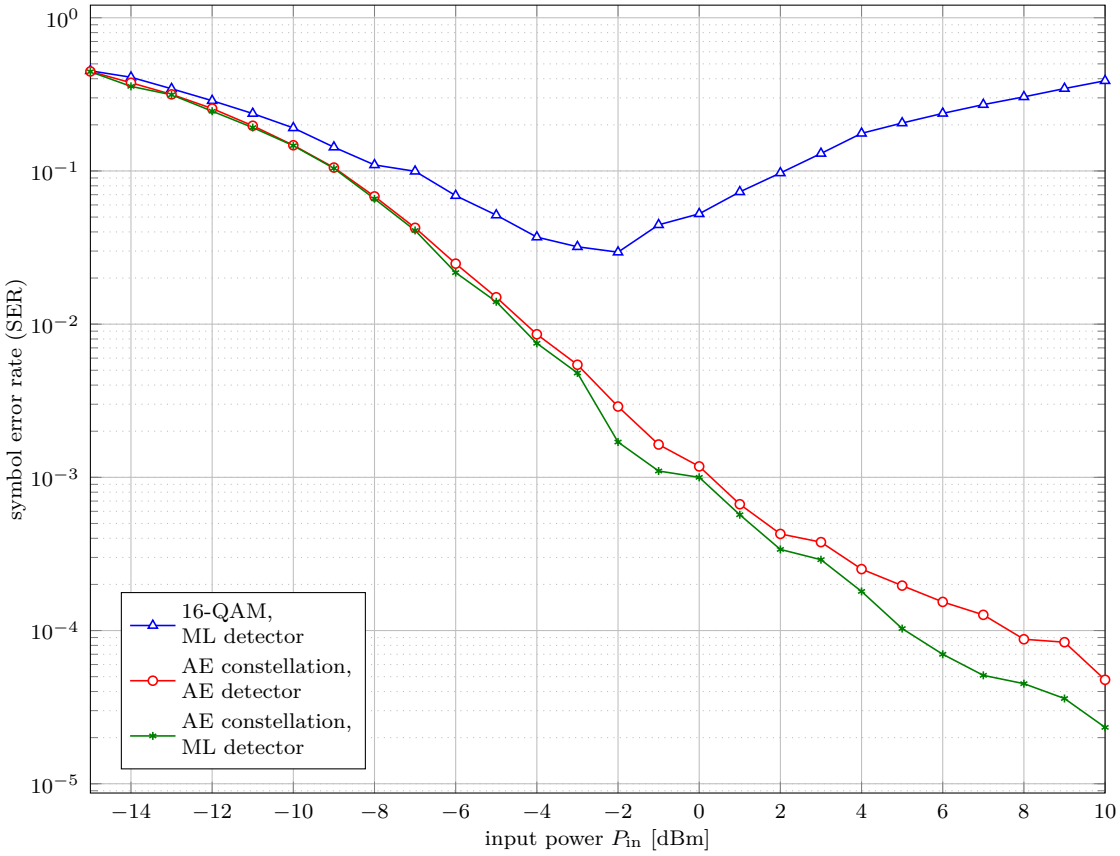

Figure 4.3: SER as a function of P_{in} for $M = 16$.

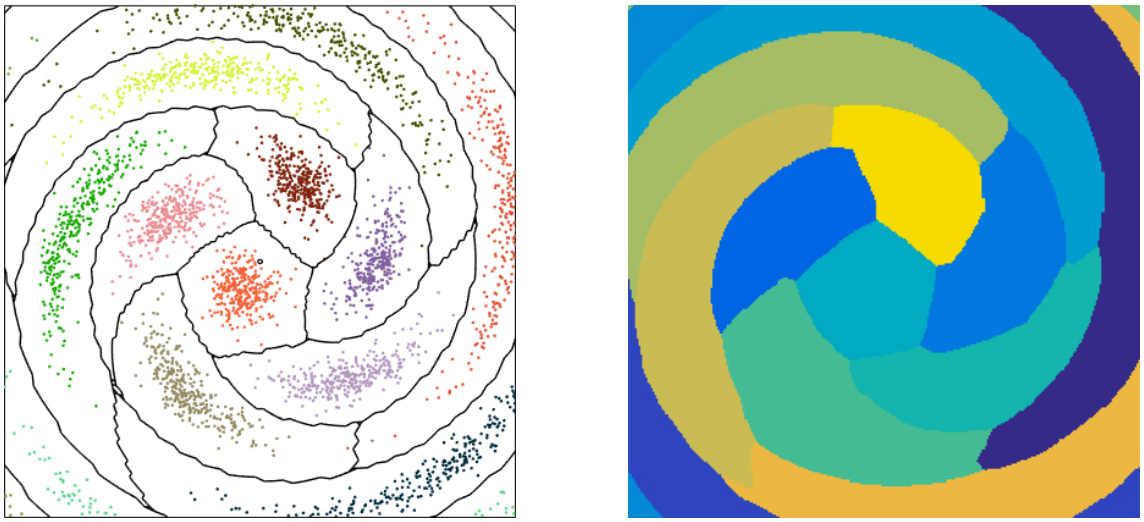

Figure 4.4: ML decision boundaries for the AE constellation at $P_{\text{in}} = 0$ dBm (left) and learned AE decision regions (right).

4.4 Learned constellations

First, for comparison, we apply the AE to a linear AWGN channel silimar to [10]. The AE parameters are summarized in Table 4.3. Training is done at $E_b/N_0 = 7$dB using Adam optimizer. Figure 4.5 illustrates the learned 16-point constellation. The constellation shape is pentagonal and the points are almost equally spaced as the noise in the channel is Gaussian distributed.

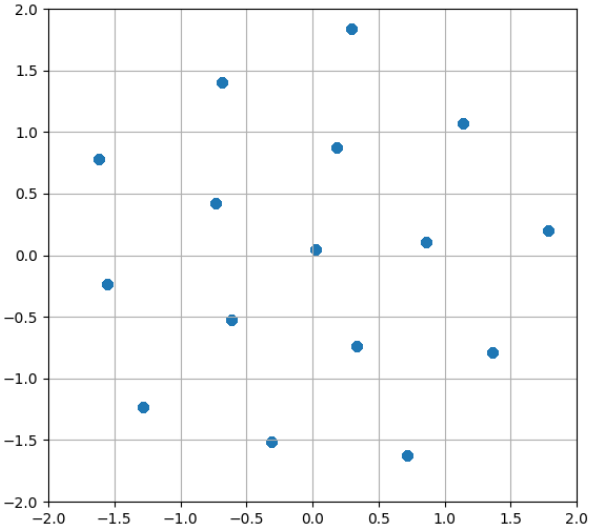

Figure 4.5: Learned 16-point constellation for AWGN channel

Table 4.3: AE parameters for AWGN channel ($M = 16$)

	transmitter		receiver		
layer	1	2	1	2	3
neurons	16	2	2	16	16
$f(\cdot)$	-	iden.	-	relu	sigm.

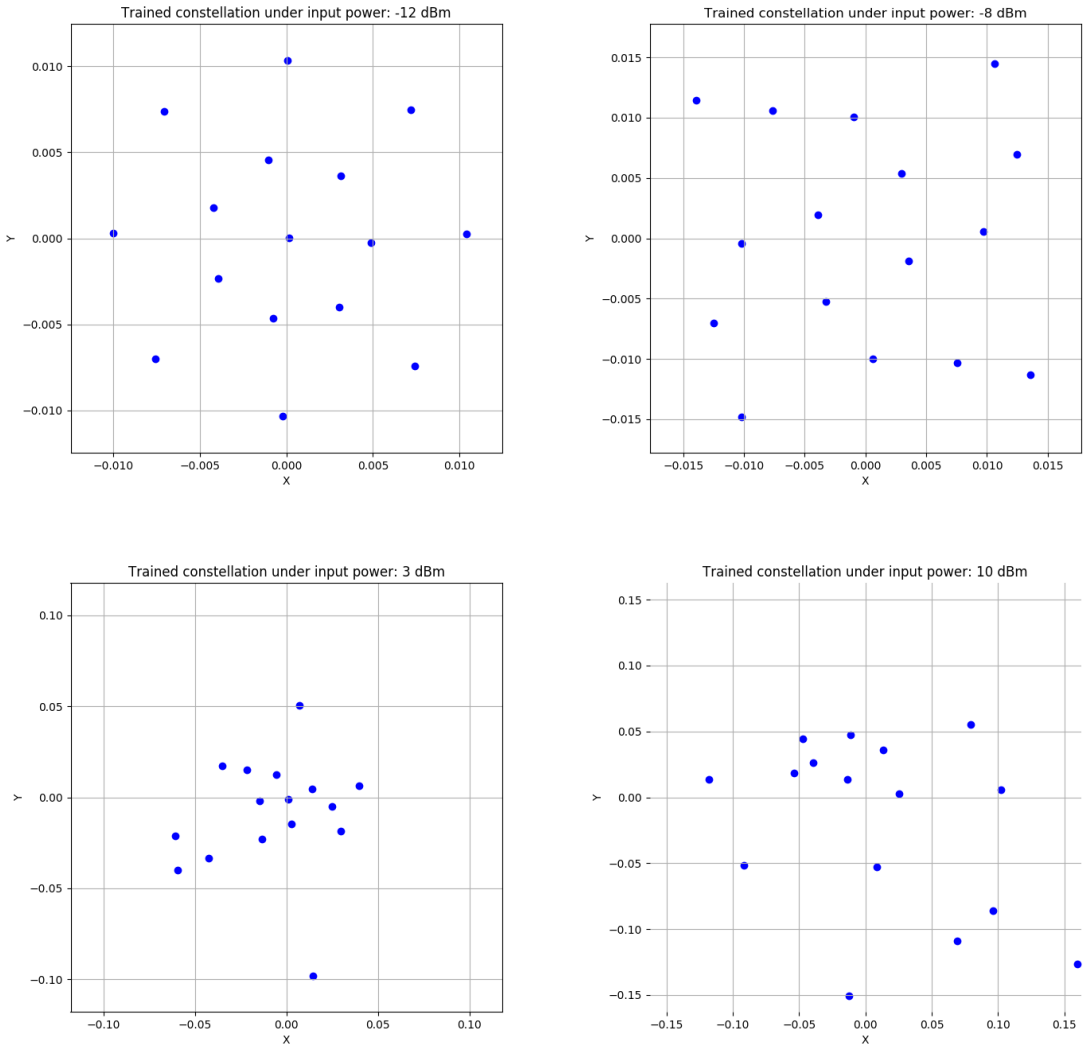

Figure 4.6: Learned 16-point constellations for the nonlinear fiber channel under $P_{\text{in}} = -12$ dBm (top left), $P_{\text{in}} = -8$ dBm (top right), $P_{\text{in}} = 3$ dBm (bottom left) and $P_{\text{in}} = 10$ dBm (bottom right).

Figure 4.6 shows the learned 16-point constellations for the simplified fiber channel under different P_{in}. From the figure, we see that at very low input power, the constellation is more like that of the AWGN channel. At -8 dBm, the constellation shape looks like a windmill which has four similar blades composed of four points. For highly-nonlinear regimes, the constellations look random but actually those constellation points with high energy all have different radii, so that after

rotation caused by the nonlinear fiber channel, they don't overlap with other points and the larger the radius of the point, the farther the point is away from other points.

Figure 4.7 illustrates the learned 256-point constellations under $P_{in} = -5$ and 6 dBm. In both constellations, the points with higher energy are farther away from other points. Compared with the constellation under -5 dBm, more constellation points gather in the center under $P_{in} = 6$ dBm.

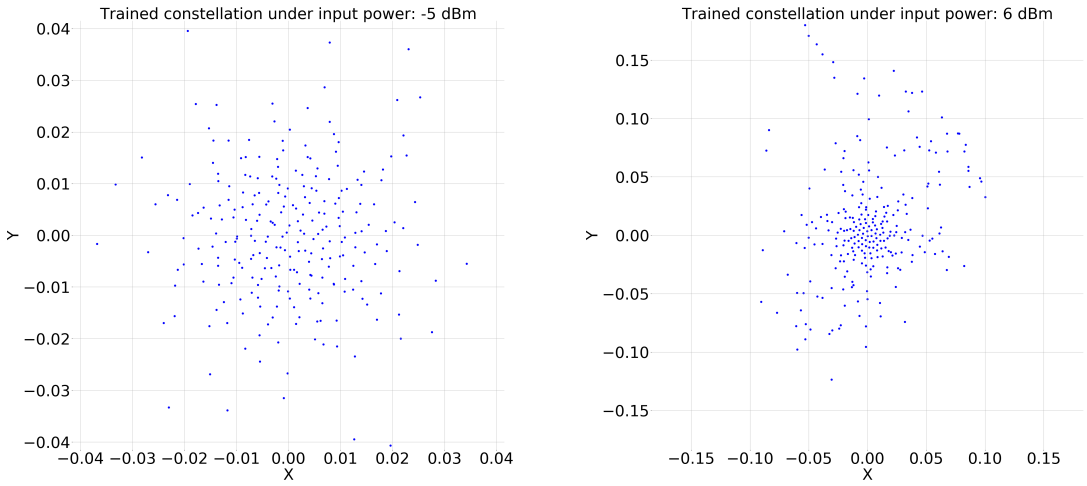

Figure 4.7: Learned 256-point constellations for the nonlinear fiber channel under $P_{\text{in}} = -5$ dBm (left), $P_{\text{in}} = 6$ dBm (right).

4.5 Achievable Information Rate

We calculate the AIR according to the right-hand side of Eq. 3.8, where $f_y(x)$ is the normalized output of the neural network and $p(x)$ is a uniform prior $1/M$. In Figure 4.8, the AIR of the AE for $M = 16$ and $M = 256$ is shown. We first compare the case $M = 16$ to the channel MI $I(X;Y)$ assuming 16-QAM as the input distribution. Note that the MI (3.3) can also be evaluated via Monte Carlo integration since the channel PDF $p(y|x)$ is known. As expected, the mutual information for 16-QAM decreases with input power, whereas the AIR of the AE flattens out at the maximum value $\log_2 16 = 4$. Lastly, we compare the AIR of the AE for $M = 256$ to three information-theoretic bounds on the channel capacity: the solid black line corresponds to a recently derived upper bound [3], whereas the dashed and dash-dotted lines correspond to lower bounds based on a Gaussian [3] and half-Gaussian [5] input distribution, respectively. The AIR of the AE closely follows the maximum of the two lower bounds, slightly exceeding them at the crossover point at around 0 dBm. These results indicate that the optimized AE constellations are close to being capacity-achieving and that the upper capacity bound can be further tightened.

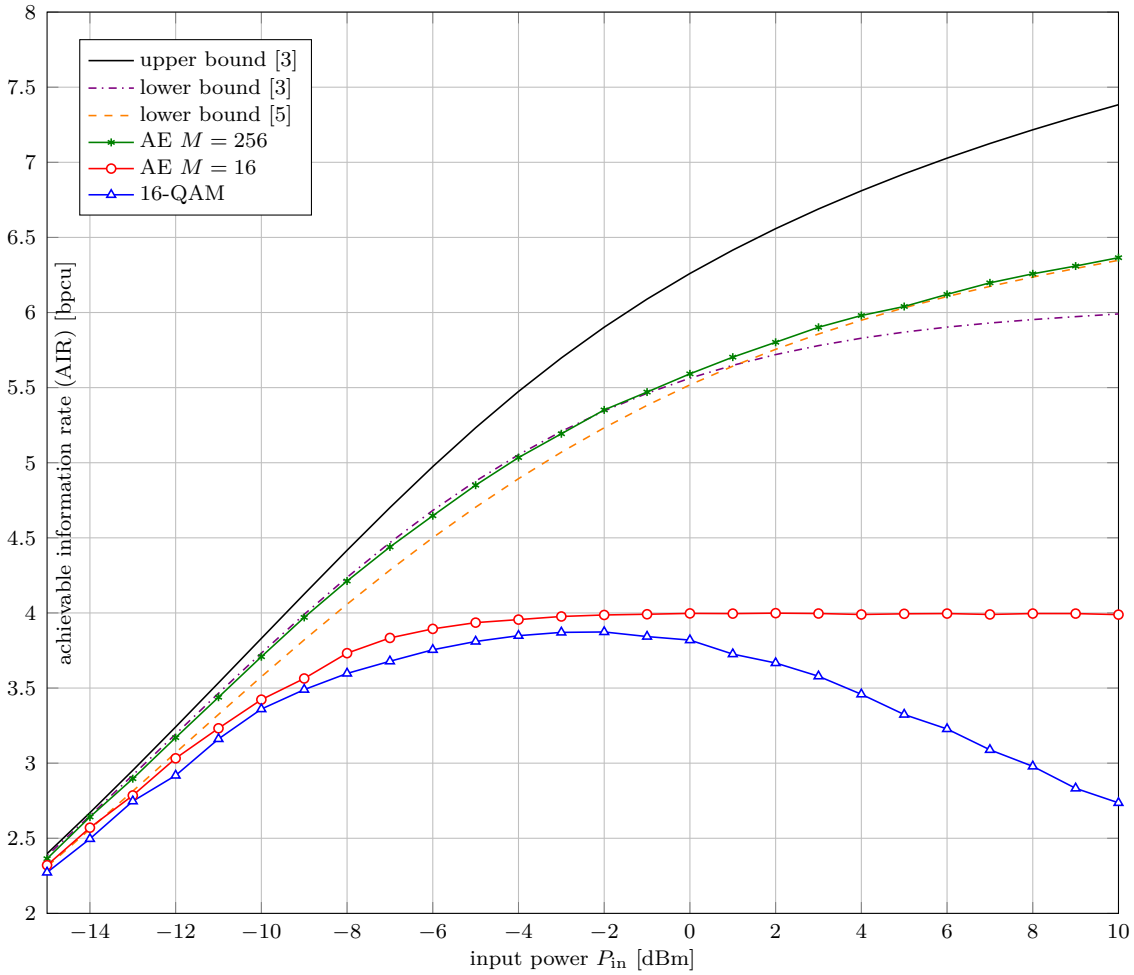

Figure 4.8: Comparison of the AIR of the AE to various information-theoretic capacity bounds and 16-QAM.

5

Conclusion

Kerr effect is one of the most challenging problems in fiber-optic communication systems. To deal with this problem, traditional methods most concentrate on decoders designs and constellation shaping. Inspired by the concept of AE, we have presented an AE approach to communicate over a simplified nonlinear fiber channel. The approach allows for end-to-end learning of good signal constellations and the channel posterior distribution. It was shown that the autoencoder can learn constellations that are robust to nonlinear phase noise and outperform conventional M-QAM constellations. Moreover, near-ML performance can be obtained without explicit channel knowledge. We also evaluated the achievable information rate of the AE, showing that the obtained lower capacity bounds are comparable to, and sometimes slightly exceed, two existing lower bounds for the considered nonlinear fiber channel model.

However, there are some aspects still need to be improved. Dispersion is neglected in this simplified channel model which cannot represent the real fiber channel accurately but still captures nonlinear phase noise. The trainings of AEs under each input power are separate, which is not efficient. A more robust AE model is under study. Moreover, our AE algorithm always needs a certain channel model to calculate gradients in backpropagation. Thus, this approach is not suitable for real fiber channels as the exact mathematical relation between a real fiber channel input and output is unclear. Reinforcement learning [50] can be a promising method for optimizing the transmitter and the receiver separately without any channel knowledge.

Bibliography

[1] A. S. Tan et al., "An ML-Based Detector for Optical Communication in the Presence of Nonlinear Phase Noise," Proc. ICC (Kyoto, Japan, 2011).

[2] P. Weinberger, "John Kerr and his effects found in 1877 and 1878," Philosophical Magazine Letters, 88:12, 897-907 (2008).

[3] K. Keykhosravi et al., "A Tighter Upper Bound on the Capacity of the Nondispersive Optical Fiber Channel," in Proc. ECOC (Gothenburg, Sweden, 2017)

[4] K. S. Turitsyn et al., "Information capacity of optical fiber channels with zero average dispersion," Phys. Rev. Lett. **91** (2003).

[5] M. I. Yousefi and F. R. Kschischang, "On the per-sample capacity of nondispersive optical fibers," IEEE Trans. Inf. Theory **57**, 7522-7541 (2011).

[6] K. P. Ho, "Phase-Modulated Optical Communication Systems," Springer (2005).

[7] A. P. T. Lau et al., "Signal Design and Detection in Presence of Nonlinear Phase Noise," J. Lightw. Technol. **25**, 3008-3016 (2007).

[8] C. Häger et al., "Design of APSK constellations for coherent optical channels with nonlinear phase noise," IEEE Trans. Commun. **61**, 3362-3373 (2013).

[9] O. Geller et al., "A Shaping Algorithm for Mitigating Inter-Channel Nonlinear Phase-Noise in Nonlinear Fiber Systems," J. Lightwave Technol. **34**, 3884-3889 (2016).

[10] T. O'Shea and J. Hoydis, "An Introduction to Deep Learning for the Physical Layer," in IEEE Trans. Cogn. Commun. Netw., Vol. **3**, 563-575 (2017).

[11] S. Dörner et al., "Deep Learning Based Communication Over the Air," in IEEE J. Sel. Topics Signal Process. **12**, 132-143 (2018).

[12] D. Zibar et al., "Machine learning techniques in optical communication," J. Lightw. Techn. **34**, 1442-1452 (2016).

[13] C. Häger and H. D. Pfister, "Nonlinear Interference Mitigation via Deep Neural Networks," Proc. OFC (Los Angeles, USA, 2018).

[14] E. Ip and J. M. Kahn, "Compensation of Dispersion and Nonlinear Impairments Using Digital Backpropagation," J. Lightw. Technol. **26**, 3416-3425 (2008).

[15] ——, "Deep learning of the nonlinear Schrödinger equation in fiber-optic communications," Proc. ISIT, (Rome, Italy, 2018).

[16] B. Karanov et al., "End-to-end Deep Learning of Optical Fiber Communications," arXiv:1804.04097 [cs.IT] (2018).

[17] H. Lee et al., "Deep learning based transceiver design for multi-colored VLC systems," Opt. Express **26**, 6222-6238 (2018).

[18] D.-M. Arnold et al., "Simulation-based computation of information rates for channels with memory," IEEE Trans. Inf. Theory **52**, 3498-3508 (2006).

[19] I. B. Djordjevic et al., "Achievable information rates for high-speed long-haul optical transmission," J. Lightw. Techn. **23**, 3755-3763 (2005).
[20] M. Secondini et al., "Achievable information rate in nonlinear WDM fiber-optic systems with arbitrary modulation formats and dispersion maps," J. Lightw. Techn. **31**, 3839-3852 (2013).
[21] T. Fehenberger et al., "On achievable rates for long-haul fiber-optic communications," Opt. Express **23**, 9183-9191 (2015).
[22] Executive Office of the President, "Big Data: A Report on Algorithmic Systems, Opportunity, and Civil Rights," Obama White House (2016).
[23] G. P. Agrawal, "Fiber-Optic Communications Systems," 3rd ed. Wiley (2002).
[24] W. S. McCulloch et al., "A logical calculus of the ideas immanent in nervous activity," The bulletin of mathematical biophysics **5**, 115-133 (1943).
[25] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain," Psychological Review, 65-386 (1958).
[26] D. E. Rumelhart, "Neurocomputing: Foundations of Research," MIT Press (1988).
[27] H. White, "Artificial Neural Networks: Approximation and Learning Theory," Blackwell Publishers, Inc. (1992)
[28] T. Young et al., "Recent Trends in Deep Learning Based Natural Language Processing," CoRR, abs/1708.02709 (2017).
[29] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," Neural Networks, **61** 85-117 (2015).
[30] P. Vincent, "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion," J. Mach. Learn. Res. **11** 3371-3408 (2010).
[31] V. Gerven et al., "Editorial: Artificial Neural Networks as Models of Neural Information Processing," Frontiers in Computational Neuroscience 11 (2017).
[32] Wikipedia contributors, "Machine learning," *Wikipedia, The Free Encyclopedia.* Wikipedia, The Free Encyclopedia, 21 Aug. 2018. Web. 21 Aug. 2018.
[33] K. Bailey, "Numerical Taxonomy and Cluster Analysis," Typologies and Taxonomies, 34, ISBN 9780803952591 (1994).
[34] I. Goodfellow et al., "Generative Adversarial Network," arXiv:1406.2661 (2014).
[35] T. Kohonen et al., "Kohonen Network," Scholarpedia (2007).
[36] G. A. Carpenter et al., "Adaptive Resonance Theory," In Michael A. Arbib (Ed.), The Handbook of Brain Theory and Neural Networks, Second Edition, 87-90, Cambridge, MA: MIT Press (2003).
[37] S. Grossberg, (1987), "Competitive learning: From interactive activation to adaptive resonance," Cognitive Science (Publication), 11, 23-63 (1987).
[38] Wikipedia contributors. "Gradient descent," *Wikipedia, The Free Encyclopedia.* Wikipedia, The Free Encyclopedia, 28 May. 2018. Web. 31 May. 2018.
[39] S. Ruder, "An overview of gradient descent optimization algorithms," CoRR, abs/1609.04747 (2016).
[40] D. P. Kingma et al., "Adam: A Method for Stochastic Optimization," CoRR, abs/1412.6980 (2014).
[41] Rumelhart et al., "Learning representations by back-propagating errors," Nature 323.6088: 533-536 (1986).

[42] C.Y. Liou et al., "Autoencoder for words," Neurocomputing, 139: 84 (2014).
[43] I. Goodfellow et al., “Deep Learning,” MIT Press (2016).
[44] P. Vincent et al., “Extracting and Composing Robust Features with Denoising Autoencoders,” Proc. ICML (Helsinki, Finland, 2008).
[45] Y. Bengio et al., “Generalized Denoising Auto-Encoders as Generative Models,” CoRR, abs/1305.6663 (2013).
[46] M. Abadi, “TensorFlow: Large-scale machine learning on heterogeneous systems,” Software available from tensorflow.org (2015).
[47] T. M. Cover et al., “Elements of Information Theory,” 2nd ed. Hoboken, NJ: Wiley (2006).
[48] E. Agrell, “Capacity Bounds in Optical Communications,” in Proc. ECOC (Gothenburg, Sweden, 2017).
[49] G. Liga et al., “Information Rates of Next-Generation Long-Haul Optical Fiber Systems Using Coded Modulation,” J. Lightw. Technol. **35**, 113-123 (2017).
[50] F. A. Aoudia et al., “End-to-End Learning of Communications Systems Without a Channel Model,” CoRR, abs/1804.02276 (2018).