1

Introduction to Radar Processing and Deep Learning

At the end of this chapter, reader will have understanding on

- How radar data cubes are processed to extract range, velocity, and angle of the multiple detected targets and are tracked over time.
- Different target representations used for radar target recognition.
- Introduction to deep learning architectures used for radar target recognition.

1.1 Basics of Radar Systems

Radar is an acronym that stands for radio detection and ranging. It is basically an electromagnetic system used to detect the presence of one or more targets of interest and estimate their range, angle, and velocity relative to the radar. Instead of just measuring the target's location and velocity, modern radars can predict the target given the reflected radar signals. The main objective of radar compared to infrared and optical sensors is to discover distant targets under difficult climate conditions and to determine their spatial location while tracking them over time with precision. The general working principle and signal processing fundamental details are explained in the following sections.

1.1.1 Fundamentals

The radar system generally consists of a transmitter that produces an electromagnetic signal, which is radiated into space by the transmit antenna. When this signal strikes an object, it gets reflected or re-radiated in many directions. This reflected echo signal is received by the receive antenna, which delivers it to the receiver circuitry, where it is processed to detect the target and also localize it over time along with certain characteristics of the target. A simplified version of a typical continuous wave radar front-end with the most important building blocks can be

Methods and Techniques in Deep Learning: Advancements in mmWave Radar Solutions, First Edition.
Avik Santra, Souvik Hazra, Lorenzo Servadei, Thomas Stadelmayer, Michael Stephan, and Anand Dubey.

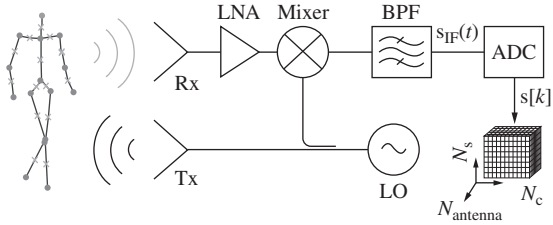

Figure 1.1 Block diagram of the continuous wave radar front-end and its receive chain including the mixer, band pass filter, and analog-to-digital converter. The digitized samples $s_{IF}(t)$ are stored into a data matrix $s[k, l]$. The radar in this case is sensing a human target in the field of view.

seen in Figure 1.1. The chosen waveform is generated by a local oscillator (LO) and transmitted via the transmit (Tx) antenna. The receive (Rx) antenna then captures the incoming signal reflections from the target at a distance. After amplifying the received signal, it is mixed with the original transmitted waveform and is passed through subsequent analog bandpass filtering (BPF). This removes any high-frequency components that could cause aliasing as well as low-frequency components from direct coupling of the LO signal into the receiver. After mixing and filtering the signal that has been shifted to an intermediate frequency (IF), and it is referred as $s_{IF}(t)$. The IF bandwidth B_{IF} is determined by the upper cut-off frequency of the bandpass filter, which is typically in the order of tens of kHz to few MHz.

1.1.2 Signal Modulation

To detect and differentiate multiple targets along its range, relative velocity, and azimuth-elevation angle dimensions, linear frequency modulated continuous wave (FMCW) is used as the most standard sensing waveform [1]. Usually, consecutive identical chirps are transmitted within a frame with a predefined time spacing referred to as chirp repetition time. The received IF signal is arranged within a two dimensional matrix, and the intratime, i.e., within a chirp, is referred as fast-time, while the intertime, i.e., across chirps, is referred to as slow-time. If the target is static, the round trip delay in the received signal is manifested as a frequency offset along the fast-time dimension after down-mixing at the receiver. But if the target or the radar is not stationary, the received signal will have an additional frequency offset caused by the Doppler shift manifested across slow-time dimension.

Figure 1.2 shows the concept of a FMCW modulation in detail. The LO generates a chirp signal $s_{LO}(t)$ with starting frequency $f_{0,LO}$, bandwidth B_c, duration T_c, and resulting sweep rate $\mu_{LO} = B_c/T_c$. By taking advantage of time integral over

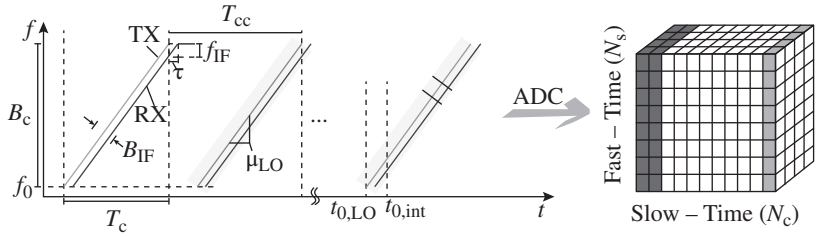

Figure 1.2 Illustration of typical modern radar sensors with several identical transmit chirps within a frame and the digitized IF samples $s_{IF}[k]$ are then stored chirpwise in a data matrix for coherent processing.

Tx frequency, instantaneous phase is calculated as shown in Eq. (1.1), where $\varphi_{0,LO}$ corresponds to the initial phase of the LO:

$$f_{Tx}(t) = f_{0,LO} + B_c/T_c t \tag{1.1a}$$

$$\phi_{Tx}(t) = 2\pi \int_0^{T_c} f_{Tx}(t)dt = 2\pi f_{0,LO} t + \pi \frac{B_c}{T_c} t^2 + \varphi_{0,LO} \tag{1.1b}$$

Assuming unity amplitude for a single chirp, $s_{LO}(t)$ can be formulated by

$$s_{Tx}(t) = \begin{cases} \cos\left(\phi_{Tx}(t)\right) & 0 \le t \le T_c \\ 0 & \text{else} \end{cases} \tag{1.2}$$

If this transmit signal gets reflected by some object, also referred to as target, the reflection will be received at the radar with a time delay τ, which is proportional to the target's distance to the radar. Additionally, signals of multiple reflections, like extended targets, are superimposed on each other at the receiver. For an arbitrary number of point targets N_{tgt} composing a spatially distributed target, the received signal $s_{Rx}(t)$ can thus be expressed as follows:

$$s_{Rx}(t) = \sum_{i=1}^{N_{tgt}} s_{Tx}(t - \tau_i(t)) + n \tag{1.3}$$

where n represents thermal receiver noise or clutter and $\tau_i(t)$ is the round trip delay to the ith target located at distance R_i and moving with a relative radial velocity of v_i. As a result, $\tau_i(t)$ can be described as $\frac{2R_i - 2v_i t}{c_0}$, where c_0 is the speed of light. For ease of notations, the noise term is dropped for all the following considerations. The received and amplified signal is mixed with the original transmitted signal ($s_{mix}(t) = s_{Tx}(t) \cdot s_{Rx}(t)$). As discussed before, both transmitted and received signal follows cosine waveform. Thus, the down-mixed signal $s_{mix}(t)$ can be transformed into two components using trigonometric formulation.

$$s_{mix}(t) = s_{Rx}(t) \cdot s_{Tx}(t) = \frac{1}{2}(\cos(\phi_{diff}(t)) + \cos(\phi_{sum}(t))) \tag{1.4}$$

here $\phi_{\text{diff}}(t)$ contains the difference of Tx and Rx signal frequencies and $\phi_{\text{sum}}(t)$ contains the sum frequencies respectively. The sum component is removed by the following BPF and the resulting IF signal $s_{\text{IF}}(t)$ is obtained as follows:

$$\begin{aligned} s_{\text{IF}}(t) &= \frac{1}{2}\cos(\phi_{\text{diff}}(t)) \\ &\approx \frac{1}{2}\cos\left(2\pi\left(\overbrace{\underbrace{\frac{2f_0R}{c_0}}_{\phi_0} + \left(\underbrace{\frac{2B_cR}{c_0T_c}}_{f_R} - \underbrace{\frac{2f_0v}{c_0}}_{f_D}\right)}^{f_b}t - \frac{2B_cv}{c_0T_c}t^2\right)\right) \end{aligned} \tag{1.5}$$

This shows that intermediate received signal contains both distance-dependent frequency f_R and also speed-dependent frequency shift f_D which are factors of modulation parameters. This includes chirp duration T_c, chirp repetition time T_{cc}, sweep frequency or bandwidth B_c, and number of chirps in a frame N_c as main configuration parameter for the design of a FMCW waveform. As a result, these parameters control the range and Doppler resolution, as presented in Eqs. (1.6) and (1.7), respectively. The maximum observable range and the maximum unambiguous Doppler is given in Eqs. (1.8) and (1.9).

$$\Delta f_R = \frac{2B_cR}{c_0T_c} \tag{1.6a}$$

$$\Delta R_{\text{min}} = 1.21\frac{c_0}{2B_c} \tag{1.6b}$$

$$\Delta f_v = \frac{2f_0v}{c_0} \tag{1.7a}$$

$$\Delta v_{\text{min}} = 1.21\frac{c_0}{2f_0N_cT_{cc}} \tag{1.7b}$$

$$R_{\text{max}} = \frac{f_sc_0T_c}{2B_c} \tag{1.8}$$

$$V_{\text{max}} = \frac{c_0}{2T_{cc}f_0} \tag{1.9}$$

1.2 FMCW Signal Processing

The IF signal, obtained from Eq. (1.5), is then digitized by an analog-to-digital converter with sampling period T_s at the discrete time instants kT_s, where $k \in [0, ..., N_s - 1]$. Consequently, the discrete time signal $s[k]$ contains N_s samples per chirp. Typically, modern short-range radar sensors rapidly transmit several identical chirps in a so-called chirp sequence modulation. The digitized IF samples $s[k]$ are then stored chirp wise in a data matrix $s[k, l]$ for coherent processing. Figure 1.3 summarizes the FMCW signal processing multistage pipeline, where

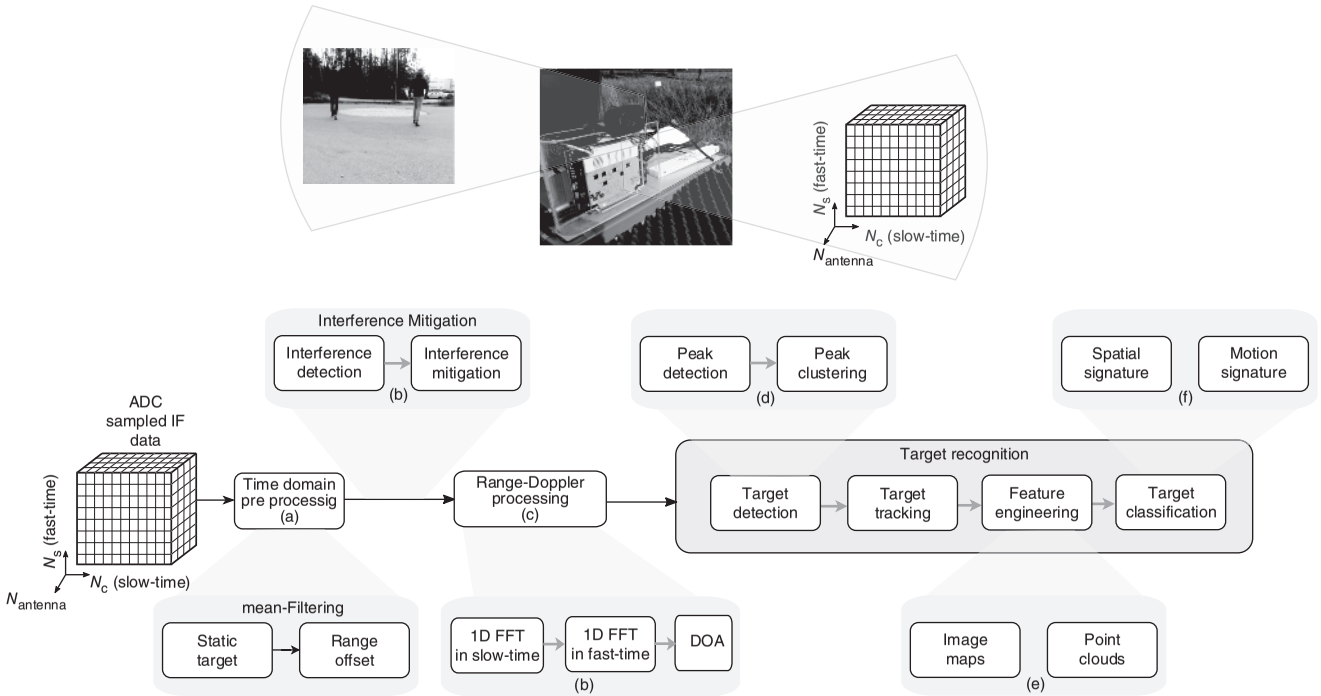

Figure 1.3 Summary of FMCW signal processing pipeline including both pre- and postprocessing over $s[k, l]$ chirp matrix for target detection, tracking, and classification.

$s[k, l]$ is first preprocessed in time-domain for removal of spectral leakage or static targets followed by interference mitigation. Later, the preprocessed $s[k, l]$ is transformed to frequency domain for target detection. Once a target is detected, then the measurement is fed into a tracking algorithm for temporal smoothening. At the end, the tracked target's features are extracted using their motion or spatial signature in the form of images or point-clouds, respectively, which are used for target recognition.

The frequency domain analysis for range-Doppler processing is explained in the following section.

1.2.1 Frequency-Domain Analysis

As indicated by Eqs. (1.6) and (1.7) both range and radial velocity information of the target are functions of frequency shifts in the received signal. As a result, frequency-domain analysis is used to determine respective target's parameters instead of time-domain analysis. In contrast to time-domain signals where signal changes over time (amplitude or power) can be observed, frequency-domain analysis reveals how much of the signal lies within each given frequency band over a range of frequencies, which also include change in phase information. The most common frequency-domain transform methods are Fourier transform, short-time Fourier transform (STFT) and wavelet transforms. All three transforms are inner products of a family of basis functions with a time-domain signal. The parameterization and the basis functions determine the properties of the transforms.

Before delving into details, Figure 1.4 illustrates all the three transforms pictorially. While the classical technique to represent time signals in the frequency domain by calculating discrete Fourier transformation (DFT), it fails to detect time variant frequency effects, which are important for extended targets. As an alternative, DFT is modified by shortening the time window for each DFT and leads to short-time Fourier transformation (STFT), which improves resolution in time but at the cost of lower accuracy in the frequency domain, as seen in Figure 1.4. While the DFT has no temporal resolution and STFT have fixed

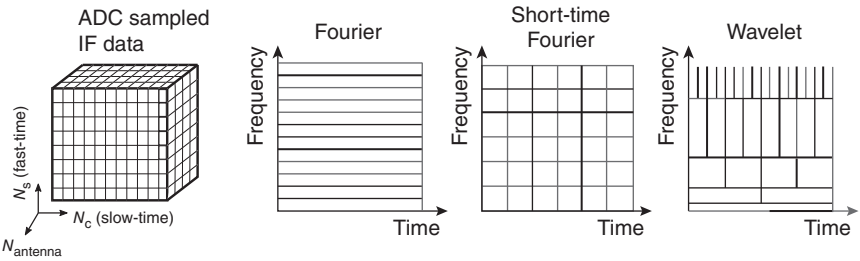

Figure 1.4 Pictorial representation of all three transforms, i.e., Fourier, STFT, and wavelets over analog-to-digital conversion (ADC) sampled chirp sequence data.

resolution for complete time-frequency, the wavelet transformation can adapt both the time and frequency dimensions and result into a high-frequency resolution at low frequencies while maintaining good time localization at high frequency.

1.2.1.1 Discrete Fourier Transform

While the Fourier series is used for oscillating or repetitive signal, Fourier transform is used for nonrepetitive signals. Thus, Fourier transform can be formulated as a special case of Fourier series when the time period $T \to \infty$. For the standard Fourier transform, the basis functions are simply the complex sinusoidal oscillations

$$b_{\omega}(t) = e^{j\omega t} \tag{1.10}$$

where t is the time axis of the signal and ω is the single frequency parameter that determines the basis function in the family. There is one basis function for every ω. The Fourier transform of the signal $x(t)$ is then simply the inner product written as an integral

$$x(\omega) = \mathcal{F}\{x(t)\}(\omega) = \langle b(\omega, t), x(t)\rangle = \int_{-\infty}^{\infty} e^{-j\omega\tau} x(\tau) d\tau \tag{1.11}$$

The negative sign in the exponential comes from the complex conjugation in the general inner product definition:

$$\langle a(t), b(t)\rangle = \int_{-\infty}^{\infty} a(\tau)^* b(\tau) d\tau \tag{1.12}$$

Since we are processing down-sampled raw radar analog-to-digital conversion (ADC) data matrix, all signals are considered as discrete time and valued (unlike continuous signals). Thus, discrete time Fourier transform (DTFT) is performed as follows:

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n) e^{-j\omega n} \tag{1.13}$$

where $x(n)$ is discrete time-value signal and $X(\omega)$ is continuous spectrum. Consequently, as $X(\omega)$ is of continuous nature, the DTFT cannot be processed directly on a digital machine. Therefore, a discrete spectrum response is required, which is usually done by using the discrete Fourier transform (DFT) and denoted by $X(k)$. This is done by sampling over the spectrum of the DTFT. The sampling frequency of DFT is defined as $\omega_k = 2\pi k/N$. Here, N corresponds to the total number of samples retrieved from the DTFT:

$$X\left(\frac{2\pi k}{N}\right) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} \tag{1.14}$$

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} \tag{1.15}$$

The above equation shows two fundamental mathematical operations, which are carried out for every sample of the input signal – multiplication and addition. The DFT is an iterative operation and requires a high-computational effort. The fast-Fourier transform (FFT) is an algorithm to compute the DFT efficiently. This is usually done using the Cooley–Tukey algorithm; however, there exist many other algorithms.

The FFT operation is applied over down-sampled raw radar ADC data matrix stored along the chirp, namely fast-time dimension, with consecutive chirps stored along the columns, referred to as slow-time dimension. Prior to the FFT operation, as seen in Figure 1.3, time-domain preprocessing can be done as an optional step. This helps to remove Tx–Rx leakage and clutter noise from down-sampled raw radar ADC data matrix. This is done by mean subtraction across fast-time and slow-time, respectively. This process is commonly known as moving target indicator (MTI) processing. Furthermore, an optional interference detection and mitigation method could also be applied to reduce the effect of interference and noise. By calculating the two-dimensional fast Fourier transform (FFT) on this data matrix, fast-time is converted to range-frequency and slow-time to Doppler frequency. This operation yields the spectrum $\boldsymbol{S}_{\mathrm{RD}}$, with indexed dimensions as range-frequency f_r and Doppler-frequency f_v.

In principle, FFT assumes that the signal contains a continuous spectrum that is one period of a periodic signal. However, the measured raw data matrix $s[k, l]$ may not contain integer number of periods. Therefore, the definiteness of the $s[k, l]$ may result in a discontinuity at the endpoints of the waveform in comparison to the original continuous-time signal and could introduce sharp transition changes into the consecutive measured signals. These artificial discontinuities lead to the additional high-frequency components not present in the original signal. This phenomenon is called spectral leakage where energy at one frequency leaks into other frequencies. This causes the sharp frequency spectrum to spread into wider signals and leads to ambiguity.

These effects are minimized using a technique called windowing. The windowing function reduces the amplitude of the discontinuities at the boundaries of each finite sequence acquired by the ADC. Windowing function consists of amplitude envelope that is multiplied elementwise with the original ADC matrix. The characteristic of windowing function is such that it varies smoothly and gradually toward zero at the edges. This makes the endpoints of the measurement $s[k, l]$ similar and therefore results in a continuous waveform without sharp transitions. In addition to it, in general, zero-padding along either of dimension of the data matrix $s[k, l]$ is done with a factor of power of 2. This interpolates the coarse spectrum to become more smooth but does not reveal extra information from the spectrum. To improve the resolution of the spectrum, length of the recorded signal needs to be increased. Additionally, it can also be interpreted as windowing, which is time-domain multiplication of rectangular function with the original signal.

1.2.1.2 Short-Time Fourier Transform (STFT)

Since FFT is performed over complete ADC sampled chirp matrix $s[k, l]$, it averages out signal frequency components. This approach is good for localization and detection of reflection from targets but fails to extract both spatial or motion information (commonly termed as signatures) for an extended targets like humans.[1] As a result, FFT is modified by adding time dimension to the base function ($b_{(\omega,t_0)}(t)$) parameters by multiplying the infinitely long complex exponential with a window to localize it. This transform is known as STFT, whose base functions are then

$$b_{(\omega,t_0)}(t) = w\left(t - t_0\right)\exp(i\omega t) \tag{1.16}$$

where $w(t)$ is the window functions that vanish outside some interval and $\left(\omega, t_0\right)$ are the time-frequency coordinates of the base function in the family. The inner product is formulated as follows:

$$\mathcal{S}\{s(t)\}\left(\omega, t_0\right) = \int_{-\infty}^{\infty} w(\tau)^* \exp(-i\omega\tau)s(\tau)d\tau \tag{1.17}$$

The advantage of STFT is that it can capture frequency information of target's signature over time, i.e., range signature or micro-Doppler signatures. This information can be treated as unique signature for target recognition or its attribute recognition. However, it is challenging to find a trade-off between time and frequency resolution while calculating the STFT. This is determined by the choice of the window function and sampling frequency.

1.2.1.3 Wavelets

In contrast to DFT and STFT, the wavelet transform adapts the window size to the frequency with the constant bandwidth constraint. This is designed in a scale invariant approach that doesn't even need the complex modulation basis function. The working principle of wavelet transforms can be understood with a generic base function that is localized and oscillates with zero mean, i.e., the integral over the complete space is zero. This basis function is referred as mother wavelet. The advantage of such a basis function (wavelet) is that both localization (time) and oscillation (frequency) resolution trade-off can be reduced, which is a constraint in STFT. Therefore, the family of base functions can be summarized as

$$b_{(\sigma,t_0)}(t) = w\left(\frac{t - t_0}{\sigma}\right) \tag{1.18}$$

where w is the mother wavelet and σ the scale parameter. Thus, the inner product becomes

$$\mathcal{W}\{s(t)\}\left(\sigma, t_0\right) = \int_{-\infty}^{\infty} w\left(\frac{t - t_0}{\sigma}\right)^* s(t) \tag{1.19}$$

1 With the availability of high-resolution radar, human targets are treated as doubly extended targets due to their multiple spatial reflections along range and along Doppler referred to as micro-Doppler components.

1.3 Target Detection and Clustering

As illustrated in Figure 1.3, irrespective of the target representation, i.e., images or point clouds, the most standard target detection has two stages, involving the detection followed by clustering. A simple approach to target detection is peak detection by determining if the sensed bin has higher amplitude than its neighboring bins.

Alternately, constant false alarm rate (CFAR) detector is used for detection of the targets. CFAR detector calculates detection probability for each bin by estimating varying noise power from neighboring cells as shown in Eq. (1.20). Here, T refers to detection threshold, α is scaling factor, P_{n} is estimated noise power, N counts the total number of neighboring reference cells and P_{fa} is the CFAR. Equation (1.20) represents cell averaging (CA)-CFAR. The drawback of CA-CFAR is that it occludes the weak targets near to strong target by having higher noise threshold and thus masking it out. As an alternative, ordered statistics (OS)-CFAR is being used for detection. The kth ordered data is selected as threshold instead of averaging over all reference cells.

$$T = \alpha P_{\mathrm{n}} \tag{1.20a}$$

$$P_{\mathrm{n}} = \frac{1}{N}\sum_{m=1}^{N} x_m \tag{1.20b}$$

$$\alpha = N\left(P_{\mathrm{fa}}^{-1/N} - 1\right) \tag{1.20c}$$

In contrast to rigid targets like car, truck, human as target of interest contains micro-motions which results in different velocity component in the received signal, commonly known as micro-Doppler components [2, 3]. This results to a spread of detected targets across Doppler dimension. Also, with the use of higher sweep bandwidth, the range resolution of the radar lies in the order of few centimeters. As a result, the reflection from target are not received as point target reflection but are spread across multiple range bins. These targets are also known as range-Doppler extended targets or doubly spread targets. As a result, all the detection from target needs to be clustered into one and thus necessitates clustering algorithm as the second stage. This also helps in reducing the computational complexity for the target-tracking algorithm, which after clustering tracks a single target parameter instead of tracking nonclustered group of target parameters. Density-based spatial clustering of applications with noise (DBSCAN) is used [4] as the state-of-the-art algorithm. Unlike most algorithms, DBSCAN runs clustering in one pass without having prior knowledge on number of clusters and is stable to the outliers (noise). The input hyperparameters required for DBSCAN are a minimum number of points M and minimum distance d between neighboring points to be part of cluster [5]. Given a set of target detections from

same and multiple targets in the 2D space, DBSCAN groups detections that are closely packed together, while at the same time removing as outliers detections that lie alone in low-density regions. To do this, DBSCAN classifies each point as either core point, edge point, or noise. A point is defined as core point if it has at least $M-1$ neighbors, i.e., points within the distance d. An edge point has less than $M-1$ neighbors, but at least one of its neighbors is a core point. All points that have less than $M-1$ neighbors and no core point as a neighbor do not belong to any cluster and will be classified as outlier and ignored. The two-stage approach has limitations in both the stages. While OS-CFAR fails to detect target in the case of clutter, multipath reflection, or interference for a fixed P_{fa}, DBSCAN fails to make cluster for sparse range-Doppler images (RDIs) and also is very sensitive to its hyperparameters. As a result, it may either lead to false detection, miss detections of real targets or target splits leading to multiple targets (when in reality it is only one target) in RDIs. However, the recent advancement in deep neural networks (DNNs) and its application in target segmentation makes DNNs an ideal algorithm for this problem. Unlike fixed rule-based methods, DNNs are capable of learning from low-level to high-level representations. The problem of two-stage target detection is treated as binary image segmentation problem in literature where target's cluster is considered as foreground and remaining information in range-Doppler map as background, as illustrated in Figure 1.5b. In [6, 7], authors have successfully demonstrated single-stage target detection on RDIs while suppressing the effect of scatterings from extended targets, multipath reflections and ghost targets. Further, in [8] single-stage target detection is introduced that preserves the instance of each target to identify them uniquely in case of a partial occlusion or overlapping of multiple targets. Additionally, [8] gives an alternative approach to combine interference mitigation and target detection by selectively looking into regions of interest.

However, the performance of target detection using image-based maps (most commonly range-Doppler and range-angle) rely strongly on the target's spread distribution pattern, which further depends on the target's aspect angle. However, by taking advantage of strong feature points from the target distribution and learning a relation between these feature points and values, the target detection can be made robust. Thus, it makes sense to transform the radar images to radar point-clouds, as shown in Figure 1.5c. Most approaches from literature propose to process radar point-clouds using certain representation transformations in order to use convolutional neural networks (CNNs), e.g., radar data is transformed into a grid map representation. However, [9–11] proposed a novel neural network, PointNet, that makes it possible to directly process point clouds. By making the representation invariant to transformation, PointNet additionally treats neighboring points to capture local structure and is invariant to structure of input point clouds. Although, radar data are very sparse compared to lidar data,

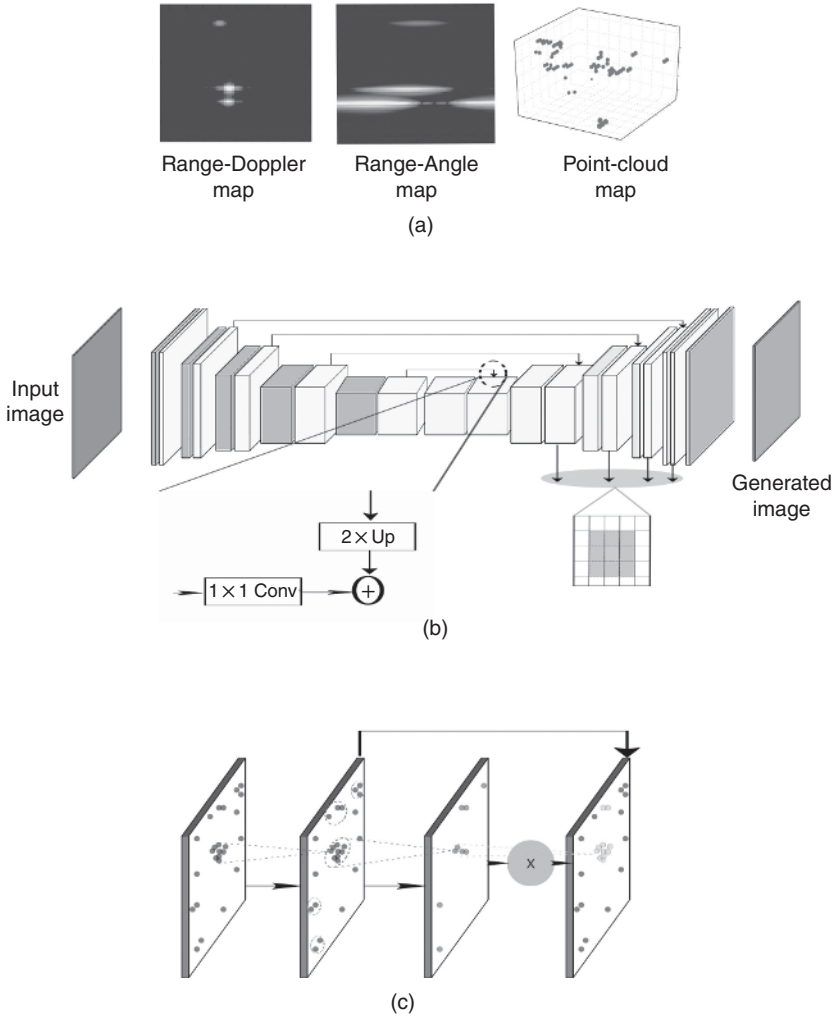

Figure 1.5 A visual summary on state-of-the-art target detection algorithms using (a) different target representation map followed by (b) convolution neural network architecture for image processing, and (c) PointNet architecture for processing of point-clouds.

radar data contains other attributes in the form of Doppler component or target radar cross section (RCS) information. As a result, multiple frameworks for target detection using radar point clouds were proposed [12–19]. The richness of data points for sparse radar point-clouds is improved either by accumulating the data over multiple frames or by augmenting data by repetition of target samples from available measurement by random sampling.

1.4 Target Tracking

Conventional radar signal processing usually applies a tracking algorithm after the clustering to filter measurements over time and track individual targets. False detections arising from ghost targets are usually also eliminated during the tracking. In order to avoid decreasing the measurement accuracy as well as the introduction of inherent noise, the usage of recursive filters is preferred in literature [20–27]. The most widely used tracking algorithms are Kalman filters. The performance of Kalman filters relies on the state vector (i.e., the parameters to be tracked), measurement noise, process noise, and the transition from measurement to the input state space. In the case of target detection, the tracker's state vector can be described as $\mathbf{x} = \begin{bmatrix} px & py & v & \theta \end{bmatrix}^T$, where px, py, v, and θ are the position coordinates along the x- and y-axes, the radial velocity and the azimuth angle, respectively. Due to low variations in the spatial dimension, azimuth information is also used as part of the state vector and improves the robustness of the target localization. Generally, heading angle and turn-rate bring additional information for a dynamic target with nonlinear motion.

Once the target detections are available at each frame, the next step is to associate them with existing tracks or create or delete tracks as necessary. The detected targets and their estimated parameters are prone to errors due to measurement noise, missed detections due to occlusions, and false alarms due to ghost targets and interference sources. The task of tracking is to maintain and update the state and identity of these targets over time reducing such false positive and true negatives. There are two important aspects to tracking, first is the track management part, which handles track initiation, track maintenance, and measurement to track association. The other part is the track-filtering step, which is achieved through either alpha-beta filter, Kalman filter, or particle filter.

1.4.1 Track Management

Track management involves logic blocks that record target tracks when it needs to be initialized or confirmed, maintains or removes a track signifying exit of a target from the field-of-view and also handles the measurement to track association, which decides on the assignment of a detected target or measurement to an existing track or generation of a new track if sufficient criteria are fulfilled. The track management block acts as a controller to handle initiation, maintenance, and deletion of a target track.

Track initiation is the process of creating a new track from an unassociated radar measurement. During track initialization, when a new unassociated radar measurement is encountered, the measurement is assigned to a new track, but the status of the track is kept tentative. The status of these tracks is changed to confirmed only when N_{det} out of N_{fr} ($N_{\text{det}} \leq N_{\text{fr}}$) subsequent measurements have a

positive detection/measurement of the target. This helps in reducing false positives from ghost targets or interference to spin off a target track.

Track maintenance is the process in which a decision is made if a track has to be terminated. If a track was not associated with a measurement during the association phase for consecutive N_{fr} frames, there is a high likelihood that the target no longer exists in the radar's field of view and thus such tracks are terminated. Alternately, due to missed detection, there are chances that the radar did not detect that target for few measurements within the N_{fr} frame window but would appear at future frame time. In this case, the tracks are retained, and only the prediction is updated.

The task of the measurement to track association is to assign the current measurement to an existing track. This step is also referred to as gating in literature. At time step $n + 1$, there are N tracks, i.e.,

$$X = \{(\mu_1, \phi_1), (\mu_2, \phi_2), \ldots, (\mu_N, \phi_N)\} \tag{1.21}$$

where $\{\mu_i, \phi_i\}_{i=1}^{N}$ are the state mean and state error covariance matrices of the tracks. Additionally, the M measurements coming from target detections at time step $n + 1$ are

$$Z = \{z_1, z_2, \ldots, z_M\} \tag{1.22}$$

A measurement of track association function $\Phi\colon Z \rightarrow X$ maps the uncertain measurements to certain tracks. The measurement to track association is a difficult problem due to multiple targets and the associated false positives and true negatives of the target detections. One of the simplest measurements to track association algorithm is suboptimal nearest neighbor (SNN), whereby each measurement is assigned to a track closest to it in Euclidean or Mahalanobis distance sense, and the assignment starts with the strongest detected target and thus is referred to as SNN. The alternate approach includes probabilistic data association filter or joint probabilistic data association filter.

1.4.2 Track Filtering

The Kalman filter is the most well-known track-filtering algorithm and utilizes a series of noisy measurements observed over time to estimate the internal state of a linear dynamic system. The estimated state variables are more accurate since the algorithm estimates a joint probability distribution over the variables for each time frame. The algorithm has two steps – prediction step and the measurement step. In the prediction step, the filter predicts the next internal state based on the process model accounting for the model uncertainties. While in the measurement step, the measurement is combined with the predicted state in an adaptive weighted average referred to as "Kalman gain" to update the internal state estimate and

its uncertainty. The Kalman filter operates under Bayesian principle, so when the measurement data are observed with less uncertainty compared to the predicted data, the Kalman gain is high, which means it relies more on the measured data to update the state. On the other hand, when the predicted data have less uncertainty compared to that of the measurement data, the Kalman gain is low, meaning it weighs the predicted data more for its state update. Kalman filters are a generic algorithm which finds use in several signal processing to control theoretic applications and is not limited to only radar signal processing. The process model used by the Kalman filter is similar to that of a hidden Markov model except that the state space is rather continuous and assumes the state and observed variables to follow a normal distribution.

One of the most standard used process model or state transition function is a linear constant velocity (CV) motion model. Let the state variable be $\psi = \begin{bmatrix} p^x & p^y & v^x & v^y \end{bmatrix}^T$, where p^x and p^y denote the position coordinates in x and y coordinates, whereas v^x and v^y denote the velocity in the x- and y-axes, respectively. The CV process model can be expressed as follows:

$$\begin{cases} p^x(k) = p^x(k-1) + v^x(k-1)\delta t \\ p^y(k) = p^y(k-1) + v^y(k-1)\delta t \\ v^x(k) = v^x(k-1) + n^{\mathrm{p}}_{vx} \\ v^y(k) = v^y(k-1) + n^{\mathrm{p}}_{vy} \end{cases} \tag{1.23}$$

where k is the time step, δt represents the frame time, i.e., the radar refresh rate. n^{p} denotes the Gaussian process noise and captures the kinematic changes which are not considered in the linear process update model. In the compact matrix–vector form, the process model can be expressed as follows:

$$\begin{bmatrix} p^x(k) \\ p^y(k) \\ v^x(k) \\ v^y(k) \end{bmatrix} = \begin{bmatrix} 1 & 0 & \delta t & 0 \\ 0 & 1 & 0 & \delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p^x(k-1) \\ p^y(k-1) \\ v^x(k-1) \\ v^y(k-1) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ n^{\mathrm{p}}_{vx} \\ n^{\mathrm{p}}_{vy} \end{bmatrix} \tag{1.24}$$

or $\psi(k+1) = F\psi(k) + n^{\mathrm{p}}$. Now, since the state variables are assumed to be Gaussian distributed, the process noise is modeled as zero mean Gaussian noise, and thus the mean and covariance of the state variables can be expressed as follows:

$$\begin{aligned} x_{k|k-1} &= \mathbf{E}\{\psi_{k|k-1}\} = Fx_{k|k-1} \\ P_{k|k-1} &= \mathbf{E}\{\phi_{k|k-1}\phi^T_{k|k-1}\} = FP_{k-1|k-1}F^T + Q \end{aligned} \tag{1.25}$$

where the state variables with suffix $k-1|k-1$ denotes the posterior probability and $k|k-1$ denotes the prior probability, and $\mathbf{E}\{.\}$ is the expectation operator and Q denotes the process noise covariance matrix. In the case of CV model, the process

noise would include kinematic changes due to jerks and acceleration of the target. Considering the acceleration component, the noise process can be expressed as follows:

$$n^{\mathrm{p}} = \begin{bmatrix} \frac{1}{2}a_x\delta t^2 \\ \frac{1}{2}a_y\delta t^2 \\ a_x\delta t \\ a_y\delta t \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\delta t^2 & 0 \\ 0 & \frac{1}{2}\delta t^2 \\ \delta t & 0 \\ 0 & \delta t \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix} = Ga \tag{1.26}$$

Thus, the process noise covariance matrix is defined as follows:

$$Q = \mathbf{E}\{n^{\mathrm{p}} n^{\mathrm{p}T}\} = G \begin{bmatrix} a_x^2 & 0 \\ 0 & a_y^2 \end{bmatrix} G^T \tag{1.27}$$

where a_x^2 and a_y^2 represent the variance of acceleration noise in x- and y-axes, and the covariance acceleration noise in x- and y-axes are 0. Thus, the process noise covariance Q is initialized based on the maximum target's acceleration expected in the system.

At the next step, the measurement model needs to account for the transformation of the internal state to the radar measurements, i.e., the radial distance ρ, the azimuth angle or bearing angle θ, and the radial velocity or range rate υ.

$$\begin{cases} \rho = \sqrt{p^{x2} + p^{y2}} \\ \upsilon = \frac{p^x v^x + p^y v^y}{\sqrt{p^{x2} + p^{y2}}} \\ \theta = \tan^{-1}\left(\frac{p^y}{p^x}\right) \end{cases} \tag{1.28}$$

That is, $z = H(\psi) + n^m$, where n^m represents the measurement zero-mean Gaussian noise with covariance matrix $R = \mathbf{E}\{n^m n^{mT}\}$. This indicates that unlike the process model, the measurement model which maps the predicted state $\begin{bmatrix} p^x & p^y & v^x & v^y \end{bmatrix}^T$ to the measurement space $z = \begin{bmatrix} \rho & \upsilon & \theta \end{bmatrix}^T$ is a nonlinear transformation, which implies that the simple Kalman filter cannot be applied in this case. The two common approaches to handle such a nonlinear transformation issue are namely extended Kalman filter (EKF) and the unscented Kalman filter (UKF). While in EKF, the nonlinear equation is approximated through first-order Taylor's expansion. It approximates and propagates the state distribution through the first-order Taylor series linearization, which expands the nonlinear state around a single-point. As a result, the EKF is not able to capture the uncertainty of the distribution, introducing large errors in the estimation of the true posterior mean and covariance, respectively. Alternately, UKFs are used, which uses deterministic sampling filters, i.e., a sigma-point Kalman filter (SPKF), to approximate

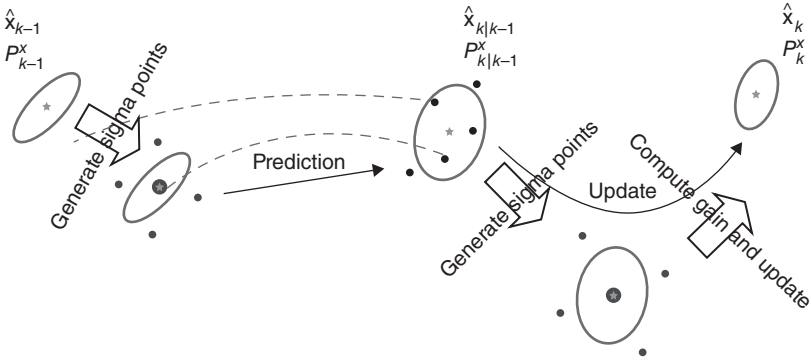

Figure 1.6 Graphical representation of predict and update stage for an UKF where mean and variance are estimated at each stage by approximation over sigma points.

the measurement model by a minimal set of sample points. These sample points can capture the true mean and covariance of the Gaussian random variable (GRV). Figure 1.6 gives a visual overview on the prediction and update operation of the UKF to track mean ($\hat{x}_{k-1}$) and covariance (P^x_{k-1}) of the input state vector at a time instance $k-1$, while Algorithm 1.1 gives a mathematical understanding on the implementation of the UKF.

In the UKF, an unscented transformation (UT) is applied at both prediction and update steps, which include a nonlinear state transformation based on f and h, respectively. As input, the state vector x_{k-1} of dimension n_x with mean $\hat{x}^{(i)}_{k|k-1}$ and the covariance $P^x_{k|k-1}$ are provided. At prediction stage, sigma points $\hat{x}^{(i)}_{k-1|k-1}$ are generated, which undergoes the UT $f(.)$ to estimate the predicted mean $\hat{x}_{k|k-1}$ and the covariance $P^x_{k|k-1}$ of the state vector. Since the predicted mean and variance changed due to the transformation, a new set of sigma points is calculated. Afterward, the new sigma points are transformed into measurement space using $h(.)$ as a transformation function. A CV system is considered with the localization state vector $\mathbf{x}$. A nonlinear measurement model $h(.)$ accounts for the transformation of the state vector into the measurement domain. Mapping part of the localization parameters (radial range and azimuth angle) from the tracker's state vector to the measurement domain follows a nonlinearity (Cartesian to spherical), whereas mapping the radial velocity and augmented parameters (appearance embedding) corresponds to an identity mapping between state vector and measurement domain. However, the overall nonlinear transformation in the process model $x^P = g(x_a)$ and the measurement model $z^P = h(x^P_a)$ can be achieved through the UT, using the so-called "sigma points." These are generated to approximate the statistical properties of the state distribution [28].

Algorithm 1.1 Unscented Kalman filter.

Prediction: Generate sigma points $\hat{x}^{(i)}_{k-1|k-1}, i = 0, 1, \ldots, 2n_x$

$$\hat{x}^{(i)}_{k|k-1} = f\left(\hat{x}^{(i)}_{k-1|k-1}\right), i = 0, 1, \ldots, 2n_x,$$

$$\hat{x}_{k|k-1} = \sum_{i=0}^{2n} W_i^{(m)} \hat{x}^{(i)}_{k|k-1},$$

$$P^x_{k|k-1} = \sum_{i=0}^{2n_x} W_i^{(c)} \left(\hat{x}^{(i)}_{k|k-1} - \hat{x}_{k|k-1}\right)\left(\hat{x}^{(i)}_{k|k-1} - \hat{x}_{k|k-1}\right)^T + Q$$

Update: Generate sigma points $\hat{x}^{(i)}_{k|k-1}, i = 0, 1, \ldots, 2n_x$

$$\hat{y}^{(i)}_{k|k-1} = h\left(\hat{x}^{(i)}_{k|k-1}\right), i = 0, 1, \ldots, 2n_x,$$

$$\hat{y}_{k|k-1} = \sum_{i=0}^{2n_x} W_i^{(m)} \hat{y}^{(i)}_{k|k-1},$$

$$P^y_{k|k-1} = \sum_{i=0}^{2n_x} W_i^{(c)} \left(\hat{y}^{(i)}_{k|k-1} - \hat{y}_{k|k-1}\right)\left(\hat{y}^{(i)}_{k|k-1} - \hat{y}_{k|k-1}\right)^T + R,$$

$$P^{xy}_{k|k-1} = \sum_{i=0}^{2n_x} W_i^{(c)} \left(\hat{x}^{(i)}_{k|k-1} - \hat{x}_{k|k-1}\right)\left(\hat{y}^{(i)}_{k|k-1} - \hat{y}_{k|k-1}\right)^T,$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + P^{xy}_{k|k-1}\left(P^y_{k|k-1}\right)^{-1}\left(y_k - \hat{y}_{k|k-1}\right),$$

$$P^x_{k|k} = P^x_{k|k-1} - P^{xy}_{k|k-1}\left(P^y_{k|k-1}\right)^{-1}\left(P^{xy}_{k|k-1}\right)^{\top}.$$

1.5 Target Representation

Unlike computer vision, target recognition for radar sensors rely on the target representation, which further depends on the choice of frequency transforms. As discussed before in Sections 1.1.2 and 1.2.1, both signal modulation and frequency-transform parameters are critical to specify the required resolution for any target representation for an application. While the signal modulation parameter defines both maximum and minimum measurable range-Doppler quantities, the frequency transform parameters are used to find a trade-off between time-frequency resolution of target's spatial and motion signatures. In broad terms, target recognition is achieved by target detection, tracking, and feature extraction for classification. As a result, accuracy of target classification strongly depends on accurate detection and tracking of the target.

Unlike camera-based imaging, radar images appear dense in nature, but target information is very low compared to background or noise distribution. Similarly, in comparison to LiDAR point-clouds, where density of target point-clouds are richer, radar point-clouds are very sparse for a single frame which makes it hard for target detection and thus more complex for target classification. In case of radar-point clouds, as a most common practice, multiple frames are accumulated over time to make target representation richer. The overview on state-of-the-art target representations in the context of target recognition is summarized in the following sections.

1.5.1 Image Representation

Radar feature images are 2D representations of the target that can be used to extract additional information about the target. Based on the radar images, the type of the target, e.g., if the target is a pedestrian, a car, or an animal, or the currently executed activity such as standing idle, walking, or riding a bike, can be recognized. The relevant 2D radar features that can be extracted for target classification are Doppler spectrograms, range angle images, and video of RDIs, which are presented in the sequel.

1.5.1.1 Doppler Spectrogram

The Doppler spectrogram is generated by first detecting the target along the range bins, followed by FFT along slow-time for the detected range bin. The Doppler spectra from consecutive frames are stacked one after another to generate a 2D image. In the case of a single target, the Doppler spectrum can be achieved by marginalization of the RDI across the range axis. The Doppler spectrum contains both the macro-Doppler component as well as micro-Doppler components due to hand and leg movements while performing an activity. The stacked Doppler spectrum across consecutive frames is referred to as Doppler spectrogram that captures information about the instantaneous Doppler spectral content and the variation of the Doppler spectral content over time. The Doppler spectra of the slow-time data from kth radar frame on the selected range bins can be expressed as follows:

$$S(p,k)^n = \left| \sum_{m=1}^{N_{st}} w(m)\mathrm{s}(m,k)^n \ \exp\left(-\frac{j2\pi mp}{N_{st}}\right) \right|$$
$$S(p,k) = \sum_{n=1}^{N} S(p,k)^n \tag{1.29}$$

where $w(m)$ is the window function along slow-time indexed by m, $s(m,k)^n$ is the slow-time data across N_c chirps in the kth frame for the nth range bin, and p is the FFT points along slow-time. $S(p,k)^n$ represents the Doppler spectrogram along the nth range bin, and N represents the range bins along which the Doppler FFT

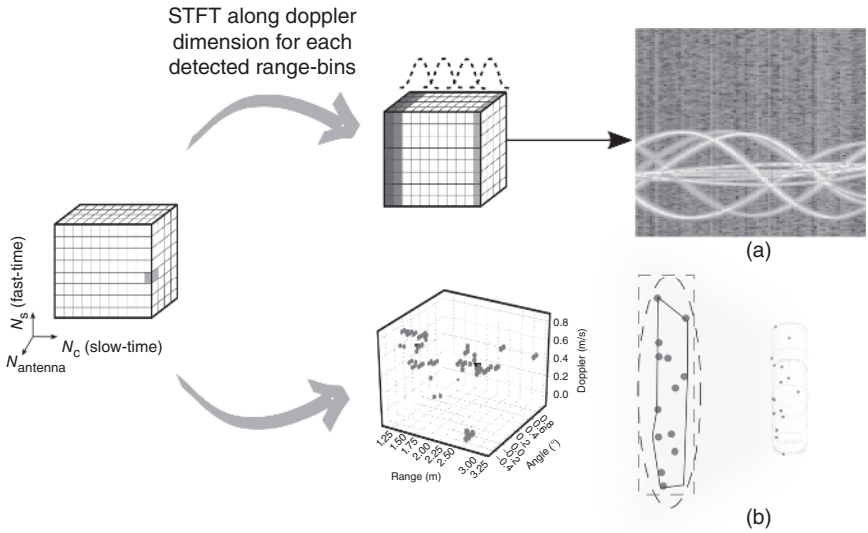

Figure 1.7 Target classification using (a) Doppler signature image and (b) spatial point-cloud maps.

is summed to generate the Doppler spectra for that frame. Figure 1.7 illustrates an example of typical motion signature of a cyclist which is generated using STFT, as described in Section 1.2.1.

1.5.1.2 Range Angle Images

Two receive antennas separated by a distance of d and receiving reflections from angle θ implies the second antenna receives the reflected signal with an additional path length of $d\sin(\theta)$ as shown in Figure 1.8 leading to phase difference of

$$\delta\phi = \left(\frac{2\pi}{\lambda}\right) d\sin(\theta) \tag{1.30}$$

Thus, the angle of arrival θ can be estimated as follows:

$$\hat{\theta} = \sin^{-1}\left(\frac{\lambda\delta\phi}{2\pi d}\right) \tag{1.31}$$

This angle of arrival estimation is referred to as phase monopulse technique.

Using the same concept, digital beamforming generates the weights for predefined angles sweeping across the entire angular field of view to create range-angle image, where the peak in the spectrum denotes the angle of arrival of the target at that particular range. Similarly, there are several other approaches to obtain the range angle image. A simplest approach takes an FFT along the virtual channel after applying a window function and zero-padding.

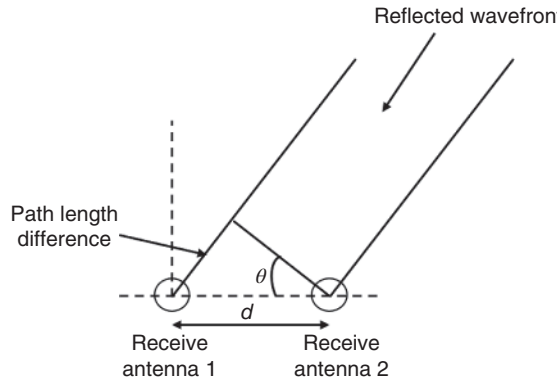

Figure 1.8 Two receive antennas with the angle of arrival θ and the path length difference between the two.

The other approach is referred to as Capon beamforming or minimum variance unbiased estimator. The $N_t \times N_r$ de-ramped beat signal can be stacked into a vector and the Kronecker product of the steering vector of the Tx array $a^{\text{Tx}}(\theta)$ and the steering vector of the Rx array $a^{\text{Rx}}(\theta)$, i.e., $a^{\text{Tx}}(\theta) \otimes a^{\text{Rx}}(\theta)$ can be used to resolve the relative angle θ of the scatterer, where $\otimes$ represents the Kronecker product. Subsequently beamforming of the MIMO array signals can be regarded as synthesizing the received signals with the Tx and Rx steering vectors. The azimuth imaging profile for a range bin l can be generated using the Capon spectrum from the beamformer. The Capon beamformer is computed by minimizing the variance/power of noise while maintaining a distortion-less response toward a desired angle. The corresponding quadratic optimization problem is

$$\begin{aligned} &\min_{w} w^H C w \\ s.t. \quad & w^H(a^{Tx}(\theta) \otimes a^{Rx}(\theta)) = 1 \end{aligned} \tag{1.32}$$

where C is the covariance matrix of noise, and the above optimization has a closed form expression given as $w_{\text{capon}} = \frac{C^{-1}a(\theta)}{a^{\text{H}}(\theta)C^{-1}a(\theta)}$, with θ being a desired angle. On substituting w_{capon} in the objective function, the spatial spectrum is given as follows:

$$P_l(\theta) = \frac{1}{\left(a^{\text{Tx}}(\theta) \otimes a^{\text{Rx}}(\theta)\right)^H C_l^{-1} \left(a^{\text{Tx}}(\theta) \otimes a^{\text{Rx}}(\theta)\right)} \quad \text{with } l = 0, ..., L \tag{1.33}$$

However, estimation of noise covariance at each range bin l is difficult in practice; hence, $\hat{C}_l$ is estimated which contains the signal component as well and can be estimated using sample matrix inversion technique $\hat{C}_l = \frac{1}{N}\sum_{k=1}^{K} s_l^{\text{IF}}(k) s_l^{\text{IF}}(k)^{\text{H}}$, where K denotes the number of snapshot used for signal plus noise covariance estimation and $s_l^{\text{IF}}(k)$ is the de-ramped IF signal at range bin l with k being the frame index.

1.5.1.3 Video of Range-Doppler Images

The video of RDIs computed at frame time k can be expressed as follows:

$$\mathrm{v}_{\mathrm{RDI}}(p,l,k) = \left| \sum_{m=1}^{N_{\mathrm{st}}} \sum_{n=1}^{N_{\mathrm{ft}}} w(m,n)\mathrm{s}(m,n,k) \exp\left(-j2\pi\left(\frac{mp}{N_{\mathrm{st}}} + \frac{nl}{N_{\mathrm{ft}}}\right)\right)\right| \quad (1.34)$$

where $w(m,n)$ is the 2D weighting function along the fast-time and slow-time, while $\mathrm{s}(m,n,k)$ is the ADC data on kth frame. The indices n,m sweep along the fast-time and slow-time axis, respectively, while l,p sweep along the range and Doppler axes, respectively. N_{st} and N_{ft} are the FFT size along the slow-time and fast-time, respectively.

1.5.2 Point-Cloud Maps

In comparison to images, the radar point-cloud representation enables a target classification framework, as shown in Figure 1.7b. The processing of point-cloud target classification follows a principle similar to image-based target classification where first neighborhood selection of all points is done using a clustering technique. The most common techniques used are DBSCAN. Thereafter, hand-crafted feature extraction and selection are performed, which is passed to linear or nonlinear supervised classifier. The hand-crafted feature engineering is mainly based on point-cloud distribution within the target cluster along each dimension. This includes mean value, maximum–minimum value within the cluster, and eigenvalues of the covariance matrix of x–y coordinates of clustered target points. Additionally, concepts like mean-shift estimation for each point or entire cluster also help to make point cloud features more robust. Additionally, with availability of deep learning frameworks [9–11] to process point-clouds directly, enable an end-to-end framework for feature engineering and classification. The most common practices are as follows:

- Processing point-cloud maps directly by PointNet to generate a class score for each point which can be clustered into one class.
- Processing point-cloud clusters separately instead of maps directly.
- Mapping point-clouds to a global coordinate grid map and using vision-based detector network architectures like you-only look once (YOLO) [29], single shot detector (SSD) [30], and region-based fully convolutional network (RFCN) [31].
- Combining PointNet features with a recurrent neural network (RNN) for temporal smoothening.

Similar to images, target classification using radar point-clouds is very challenging due to inherent nature of data distribution. Although the point-cloud maps contain information such as angle, range, velocity, RCS, orientation, and dimension for each detected target point, the point distribution is very sparse in nature. Further, the number of data points for weak targets such as pedestrian

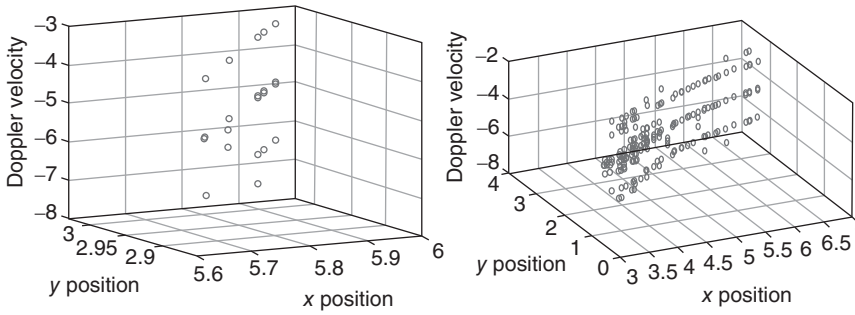

Figure 1.9 Illustration of a 3D radar point cloud where coordinate axis are *x*, *y* positional coordinates and Doppler velocity; (a) and (b) show how data aggregation changes the form and information content of a point cloud.

and cyclist are in the range of 0–10 within a single frame, depending on the target's viewing angle and distance to radar sensor. Further, this also creates a data imbalance within the data distribution of targets despite of having the same number of examples for each target of interest. Additionally, radar point-cloud maps are very prone to viewpoint and temporal variations across consecutive measurements of the same object even during a static scenario. As an alternative to this problem, multiple frames are accumulated. These variations occur due to the radar sensors statistical nature and due to the use of the classical radar processing, which uses handcrafted features (e.g., detector thresholds) to detect and localize radar detections. As an alternative to it, multiple frames are accumulated to generate dense point-cloud maps. Figure 1.9 illustrates how the point cloud changes due to aggregation of multiple point-clouds over time.

1.6 Target Recognition

After accurate target localization using detection and tracking along either of two dimensions, i.e., range, Doppler, and angle, detected target is classified into desired class. Similar to detection, multiple frameworks are proposed for target classification using both images and point-clouds. The most common approach for target classification using images is done by extracting hand-engineered features from detected target characteristic along range-angle and range-Doppler dimensions [32–34]. The most common characteristics are mean value of range, Doppler, direction of arrival, and normalized reflected power for all detected peaks from unique target cluster. Additionally, addition of variance and deviation in range, radial Doppler, and object size in *x*,*y*-dimension in feature set increase the richness of target features. Alternative to hand-crafted features, feature

descriptors such as speed up robust features (SURFs) or scale-invariant feature transform (SIFT) can be used over the detected target region for a fixed grid size. Later, these feature set can be passed to a linear or nonlinear classifiers such as decision tree, support vector machine (SVM), or K-nearest neighbor (KNN). Such methods are very specific to sensor measurement and are prone to noise, and thus often, struggle to generalize over different measurements for sensor at diverse location or with different gain pattern. The alternate state-of-the-art approaches involve deep learning algorithms. The notable deep learning architecture is presented in the sequel.

1.6.1 Feedforward Network

Feedforward neural networks or multilayer perceptrons (MLPs) approximate some function f that does the mapping $y = f(x; w)$ and learns the weights w of the network that results in the function approximation given a certain objective. These models are called feedforward because information flows through the function being evaluated from x, through the intermediate computations used to define f, and finally to the output y. There are no feedback connections in which outputs of the model are fed back into itself. Figure 1.10 presents a feedforward network with one hidden layer.

1.6.2 Convolutional Neural Networks (CNN)

In deep learning literature, the most widely used networks are CNNs [35–37] for tasks such as object detection, face recognition, image segmentation, or super-resolution. In CNNs, the image classification is performed by incorporating various layers namely convolution layers, pooling layers, and dense layers with cross-entropy (CE) loss. The network sees an image as a multidimensional array of pixels with the dimensions $h \times w \times c$ (h = height, w = width, c = color channel).

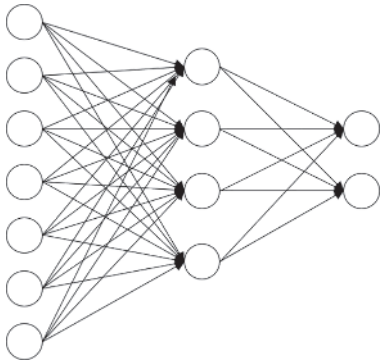

Figure 1.10 Example of a feedforward neural network with one hidden layer.

As an example, an image of size $32 \times 32 \times 3$ represents an RGB image (three-color channels), while an image of size $32 \times 32 \times 1$ represents a grayscale image.

The initial spatial feature extraction in a CNN is done by convolution layers. In a convolution layer, a filter or a kernel (weight matrix) is sliding through the entire image. The output of the convolution layer is defined as the dot product of the sliding kernel and the underlying input image in each sliding step. One must note that the filter must have the same number of channels as in the input image.

The output dimension of the convolution layers are calculated as follows:

$$\begin{aligned} w_{\text{out}} &= \frac{w_{\text{in}} - f_w + 2p}{s} + 1 \\ h_{\text{out}} &= \frac{h_{\text{in}} - f_h + 2p}{s} + 1 \end{aligned} \tag{1.35}$$

where $w_{\text{in}}, h_{\text{in}}$ are the width and the height of the input image, f_w and f_h are the width and height of the filter or kernel, p, s are the padding and the stride factors and are set ≥ 1. The weights of the filters are learned by the network through back propagation. The components of the convolution layers are described as follows:

1. Strides: While applying convolution, shifts are made to move the filter across the entire image. The stride defines the step size of the shift, e.g., if the stride is one, then the filter is shifted by a single pixel, and if the stride is two, then it is shifted by two pixels, and so on.
2. Padding: When a filter does not fit the input image properly, there are two options:
   - Zero-padding – Pad the image with zeros so that the filter fits perfectly.
   - Valid-padding – Dropping the part of images that did not fit the filter perfectly.
3. Activation Function: Some of the standard activation functions used in a CNN are as follows:
   - Sigmoid (Logistic Activation): This activation function is originally inspired from the "real neuron." The output of this activation function is between [0,1]. Its major drawbacks are that a saturated neuron will not learn and the activation is computationally expensive.

$$\begin{aligned} f(x) &= \frac{1}{1 + \exp(-x)} \\ f'(x) &= f(x)(1 - f(x)) \end{aligned} \tag{1.36}$$

   - Hyperbolic Tangent Activation: The output range of hyperbolic tangent activation is between [−1,1] and is zero centered. Similarly like sigmoid activation, this activation also does not train saturated neurons.

$$\begin{aligned} f(x) = \tan h(x) &= \frac{1 - \exp(-x)}{1 + \exp(-x)} \\ f'(x) &= 1 - \tan h^2(x) \end{aligned} \tag{1.37}$$

- Rectified Linear Unit (ReLu): In ReLu activation, there is no saturation if $x > 0$ and is more computationally efficient and leads to faster convergence. In such an activation, the output is always positive and the inactive neurons are not optimized:

$$f(x) = \max(0, x)$$
$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{1.38}$$

- Leaky Rectified Linear Unit and Parametric ReLu: These activation functions are an improvement over the normal ReLu which overcomes the problem of dead neurons. In case of leaky Relu, α is a small constant, such as 0.01. In case of parametric ReLu, α is a hyperparameter learned through back propagation.

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x < 0 \end{cases}$$
$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{if } x < 0 \end{cases} \tag{1.39}$$

Figure 1.11 presents the commonly used activation functions described above namely sigmoid function, hyperbolic function, rectified linear unit (ReLu), and leaky rectified linear unit.

One of the other standard activation function typically used in the output layer for classification problems is the softmax layer. Since the squared error is not suitable for cases where classes are mutually exclusive, a better approach is to assign probabilities to each class with the constraint that the outputs should sum up to 1. The softmax function forces the output to represent a probability distribution across the possible classes L. Its function and its derivatives are given as follows:

$$p_k = \frac{\exp(x_k)}{\sum_{j=1}^{L} \exp(x_j)}$$
$$p'_k = f(x_k)(1 - f(x_k)) \tag{1.40}$$

The cost function typically associated with the softmax layer is the negative log likelihood of the correct prediction called CE or log loss cost function and is defined as follows:

$$E(t, p) = -\sum_{k=1}^{L} t_k \log\left(\frac{\exp(u_k)}{\sum_{j=1}^{L} \exp(u_j)}\right) \tag{1.41}$$

4. Pooling/Subsample Layers: Pooling layers or more specifically spatial pooling layers perform subsampling or down sampling of the input image while retaining the most relevant information. This process helps in the reduction of parameters when the images are too large.

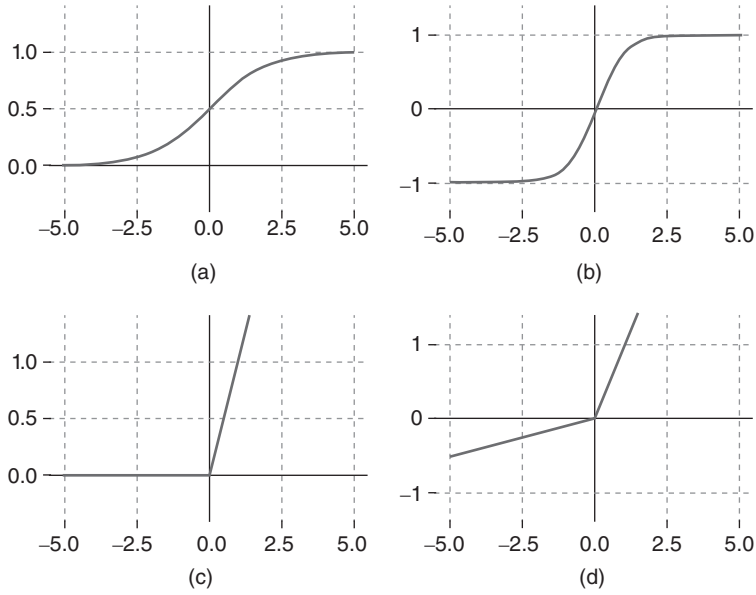

Figure 1.11 Illustration of various activation functions: (a) sigmoid function, (b) hyperbolic function, (c) ReLu function, and (d) leaky ReLu.

Three majorly used pooling layers are as follows:

- Max pooling – Taking the largest element within the defined nonparametric filter size.
- Average pooling – Taking the average of all the elements within the defined nonparametric filter size.
- Sum pooling – Taking the sum of all the elements within the defined nonparametric filter size.

5. Dense/Fully Connected Layers: At the end of the CNN, a single or multiple dense layers are used to which the flattened (1D array) output of the previous convolution and pooling layers is fed. In a CNN used for classification the last activation function is typically a sigmoid or a softmax activation.

Figure 1.12 illustrates an example of a CNN architecture that includes convolution layers, pooling layers, or subsample layers followed by dense or fully connected layers at the later stage.

1.6.3 Recurrent Neural Network (RNN)

MLPs and CNNs cannot directly address the problem of information propagation through time. Several applications such as gesture sensing and tracking require

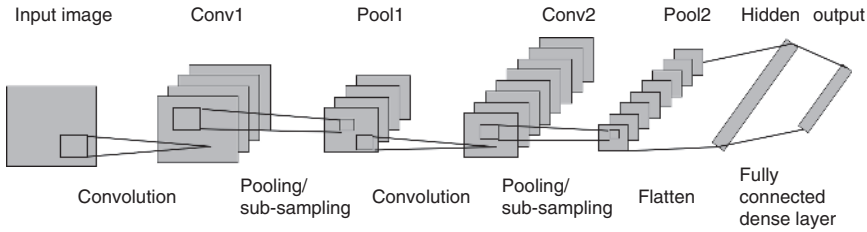

Figure 1.12 Example of a CNN architecture.

the neural network to keep a history of the past events to make a decision. RNN addresses this issue, by having self-loop structures, allowing information to persist over time. Using these self-loops, RNNs are able to link previous information to the current task and make a decision based on the previous events. One of the biggest hindrance toward wide adoption of RNNs in 1990s was the problem of vanishing gradient. Not only does the information flows through time but also the error has to be backpropageted through time. To do so, the self-loops are unfolded over time, which leads to a very deep network, where the gradient has to propagate through many layers. However, if the weights are less than one, further multiplication by gradient that is also less than one would result in a very small number after a few multiplications. Thus, the gradient flows over time through the RNNs would easily become zero, which means no further propagation of information. As a result, the RNNs were unable to retain information or learn information from quite distant past.

RNNs can be described as follows: a RNN maps a given temporal input sequence $\mathbf{x}(k) = (x_1(k), x_2(k), x_3(k))$ to a sequence of hidden values $\mathbf{h}(k) = (h_1(k), \dots, h_T(k))$ and outputs a sequence of activations $\mathbf{a}(k+1) = (a_1(k+1), \dots, a_T(k+1))$ by iterating the following recursive equation:

$$\mathbf{h}(k) = \sigma(W_{\text{hx}}\mathbf{x}(k) + \mathbf{h}(k-1)W_{\text{hh}} + \mathbf{b}_{\text{h}}) \tag{1.42}$$

where σ is the nonlinear activation function, $\mathbf{b}_{\text{h}}$ is the hidden bias vector, W_{hx} is the input-hidden weight matrix and W_{hh} is the hidden-hidden weight matrix.

The activation for these recurrent units is defined as follows:

$$\mathbf{a}(k+1) = h(k)W_{\text{ha}} + \mathbf{b}_{\text{a}} \tag{1.43}$$

where W_{ha} denotes the hidden-activation weight matrix and $\mathbf{b}_{\text{a}}$ denotes the activation bias vector.

RNNs have the problem of vanishing or exploding gradient, which is solved through a long-short term memory (LSTM) [38] or a gated recurrent unit (GRU) [39]. LSTMs extend RNNs with memory cells using the concept of gating: a mechanism based on componentwise multiplication of the input, which defines the

behavior of each individual memory cell. The LSTM updates its cell state, according to the activation of the gates. The input provided to an LSTM is fed into different gates that control which operation is performed on the cell memory: write (input gate), read (output gate), or reset (forget gate). These gates act on the signals they receive and block or pass information based on its strength and importance, by virtue of their own learned filter weights. These weights are learned during backpropagation, which means the weights of the cells decide when to allow data to be entered, to be retained, or to be deleted.

The vectorial representation (vectors denoting all units in a layer) of the update of an LSTM layer is as follows:

$$\begin{cases} \mathbf{i}(k) = \sigma_i(W_{ai}\mathbf{a}(k) + W_{hi}\mathbf{h}(k-1) + W_{ci}\mathbf{c}(k-1) + \mathbf{b}_i) \\ \mathbf{f}(k) = \sigma_f(W_{af}\mathbf{a}(k) + W_{hf}\mathbf{h}(k-1 + W_{cf}\mathbf{c}(k-1) + \mathbf{b}_f) \\ \mathbf{c}(k) = \mathbf{f}(k)\mathbf{c}(k-1) + \mathbf{i}(k)\sigma_c(W_{ac}\mathbf{a}(k) + W_{hc}\mathbf{h}(k-1) + \mathbf{b}_c) \\ \mathbf{o}(k) = \sigma_o(W_{ao}\mathbf{a}(k) + W_{ho}\mathbf{h}(k-1) + W_{co}\mathbf{c}(k) + \mathbf{b}_o) \\ \mathbf{h}(k) = \mathbf{o}(k)\sigma_h(\mathbf{c}(k)) \end{cases} \tag{1.44}$$

where **i**, **f**, **o**, and **c** are the input gate, forget gate, output gate, and cell activation vectors, respectively, all of which are of the same size as vector **h** defining the hidden value. Terms σ represent nonlinear activation functions. The term $\{\mathbf{x}(1), \mathbf{x}(2), \cdots, \mathbf{x}(K)\}$ is the input to the memory cell layer at time k. W_{ai}, W_{hi},W_{ci}, W_{af}, W_{hf}, W_{cf}, W_{ac}, W_{hc}, W_{ao}, W_{ho}, and W_{co} are weight matrices, with subscripts representing from-to relationships b_i, b_f, b_c, and b_o are bias vectors.

Figure 1.13 illustrates one unit of a LSTM block. Depending on the application, there are different configurations how the RNN models can be used. RNNs are not only able to map one input to one output but can also map one input to multiple outputs, multiple inputs to one output or multiple inputs to multiple outputs.

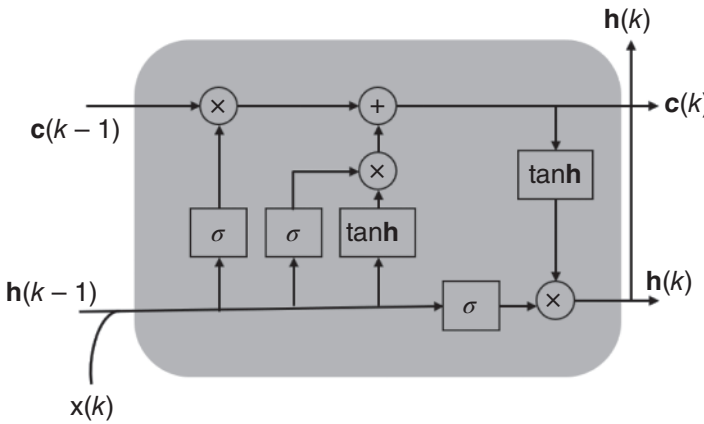

Figure 1.13 Example of a LSTM cell.

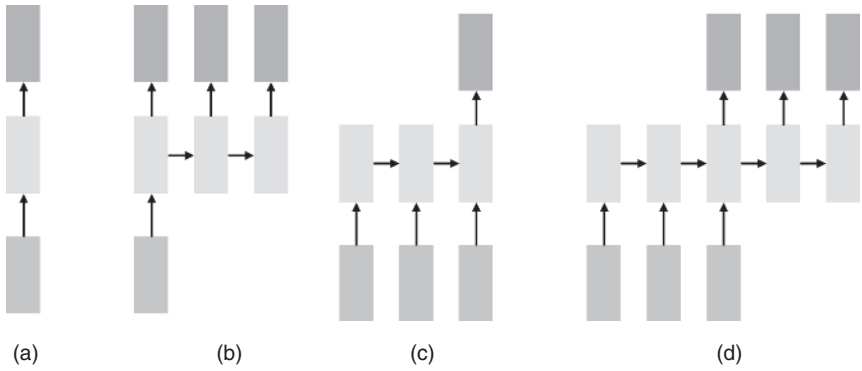

Figure 1.14 Different types of RNN models: (a) one-to-one, (b) one-to-many, (c) many-to-one, and (d) many-to-many.

These four configurations are illustrated in Figure 1.14. The many-to-one configuration is used in radar-gesture sensing and many-to-many configuration is used in human activity classification.

1.6.4 Autoencoder and Variational Autoencoder

Autoencoders comprise an encoder neural network followed by a decoder neural network with the aim of reconstructing the input data at the output. The design of the autoencoder imposes a bottleneck in the network that encourages a compressed representation of the original input. In general, autoencoders aim to leverage the key structure in the data to compress the input into the network's bottleneck or latent space representation, which is enough to reconstruct the original input data. It is thus used in dimension reduction and denoising applications among others.

The model involves an encoder function g parameterized by θ and a decoder function f parameterized by ϕ. The bottleneck layer is given as follows:

$$z = g(x; \theta) \tag{1.45}$$

where x is the input data and z the encoded latent vector. The reconstructed input at the output of the decoder can be expressed as follows:

$$\hat{x} = f(g(x; \theta); \phi) \tag{1.46}$$

The autoencoder network is then iteratively optimized using the reconstruction loss such as mean squared error (MSE):

$$L(\theta, \phi) = \frac{1}{N} \sum_{i=1}^{N} |x^i - \hat{x}^i|^2 \tag{1.47}$$

Compared to autoencoders that aim to project the input data onto a single latent vector, variational autoencoders [40–42] instead aim to learn or encode the input data onto a distribution in the latent space. Variational autoencoders can be viewed as applying a regularization during training that prevents the network from overfitting. The input data x is encoded as a distribution over the latent space, i.e., $p_\theta(z|x)$, then a point is sampled from that latent distribution $z \sim p_\theta(z|x)$, which is then fed into the decoder to reconstruct the input data as $\hat{x} = g_\phi(z)$. The reconstruction loss such as the mean-square error is used along with the Kullback–Leibler (KL) divergence of $p_\theta(z|x)$ to a Gaussian distribution with mean 0 and variance 1, i.e., $\mathcal{N}(0,1)$, to back propagate and learn the weights of the network.

In practice, the encoded distributions are chosen to be normal so that the encoder can be trained to return the mean and the covariance matrix that describes these Gaussians. The reason why an input is encoded as a distribution with some variance instead of a single point is that it allows to express very naturally the latent space regularization: the distributions returned by the encoder are enforced to be close to a standard normal distribution such that the entire feature space is close to a standard normal distribution. We can notice that the KL divergence between two Gaussian distributions has a closed form that can be directly expressed in terms of the means and the covariance matrices of the two distributions. The loss function of a variational auto-encoder (VAE) can be written as follows:

$$L(\theta, \phi) = \frac{1}{N}\sum_{i=1}^{N} |x^i - \hat{x}^i|^2 + KL\left(p_\theta(z|x), \mathcal{N}(0,1)\right) \tag{1.48}$$

where N is the number of examples.

The KL divergence is the expectation of the log difference between the probability of data sampled from the approximating distribution and the target distribution and thus is defined as follows:

$$D_{\mathrm{KL}}(p||q) = \sum_{i=1}^{N} p(x_i) \log\left(\frac{p(x_i)}{q(x_i)}\right) \tag{1.49}$$

The KL divergence has the following properties:

1. KL divergence is 0 when both distributions are approximately the same:

$$D_{\mathrm{KL}}(p||q) = 0 \quad \text{iff} \quad p \sim q \tag{1.50}$$

2. KL divergence is always positive for any two distributions:

$$D_{\mathrm{KL}}(p||q) > 0 \quad \text{if} \quad p \neq q \tag{1.51}$$

3. To ensure $D_{\mathrm{KL}}(p||q)$ is finite, the support of p needs to be contained in q else if by Eq. (1.49) $q(x) \to 0$, then $D_{\mathrm{KL}}(p||q) \to \infty$.

4. The KL divergence is an asymmetric metric, i.e.,

$$D_{\mathrm{KL}}(p\|q) \neq D_{\mathrm{KL}}(q\|p) \tag{1.52}$$

which means that $D_{\mathrm{KL}}(p\|q)$ is not a distance metric

Conceptually, the VAE architecture of learning a distribution in the latent space makes the space continuous, meaning two closely spaced points in the latent space generate more similar content than two far spaced points, and complete, meaning any point sampled from the latent space generates a meaningful output at the decoder of the VAE. Since during backpropagation the gradient cannot flow through a probabilistic layer, the sampling process of extracting $z \sim p(z|x)$ requires a special technique, referred to as "reparameterization trick." The reparameterization trick suggests to randomly sample ϵ from a zero mean and unit variance Gaussian, and then shift ϵ by the latent distribution's mean μ and scale it by the latent distribution's variance σ. Figure 1.15 presents the reparameterization trick used to sample the random variable from the latent distribution making it deterministic. The reparameterization trick allows to optimize the parameters of the distribution while still maintaining the ability to randomly sample from that distribution.

1.6.5 Generative Adversarial Network

Generative adversarial networks (GANs) introduced by Goodfellow in 2014 [43] are a breakthrough in the field of unsupervised learning using neural

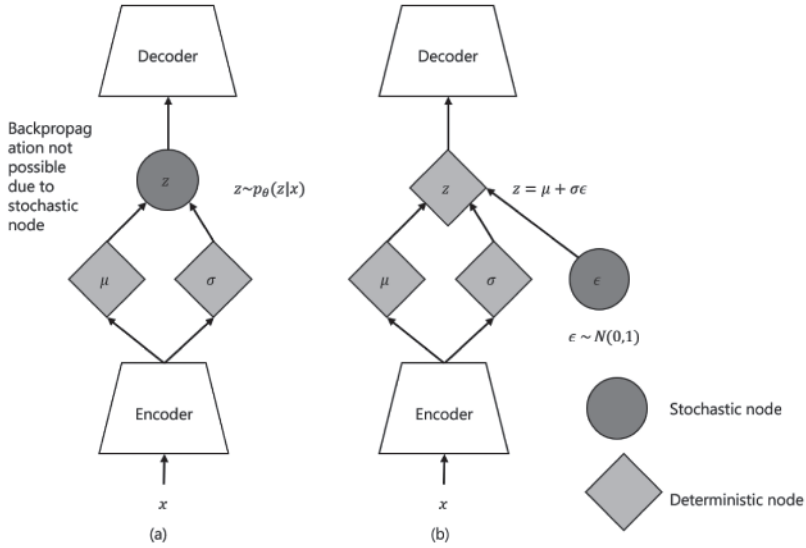

Figure 1.15 Illustration of variational autoencoder architecture depicting: (a) original form highlighting the issue during backpropagation, and (b) reparameterization trick.

networks. The technique is one of the most promising unsupervised learning approaches due to its capability of modeling high-dimensional distributions and less computationally expensive training process when compared with previous unsupervised learning methods such as VAEs, Boltzmann machines, and others. The working principle of GANs is a two-player minmax game, where two neural networks called the generator and discriminator are playing against each other. The generator tries to fool the discriminator by generating real-looking data, while the discriminator's task is to classify real and fake data. During training, the generator progressively becomes better at creating images that look real, while the discriminator becomes better at telling them apart. The minmax game has a global (and unique) optimum for $p_g = p_r$, where p_g is the generative distribution and p_r is the real data distribution. The process reaches equilibrium when the discriminator can no longer distinguish real from fake images. Once trained, only the generator is used to generate new realistic data similar to the real data distribution. Figure 1.16 illustrates the operating principle of a generator and discriminator that are used in training a vanilla GAN network.

During training, the discriminator classifies both real data and fake data from the generator and penalizes the discriminator weights for misclassifying a real instance as fake or a fake instance as real. Thus, incrementally getting better at classifying real and fake data. The generator part of a GAN learns to create fake data by incorporating feedback from the discriminator, in the sense that the generator loss penalizes the generator for failing to fool the discriminator. If the generator succeeds perfectly, then the discriminator has a 50% accuracy meaning it is unable to tell real from fake data anymore. If the GAN continues training past this point, then the generator starts to train on completely random feedback, and its own quality may collapse.

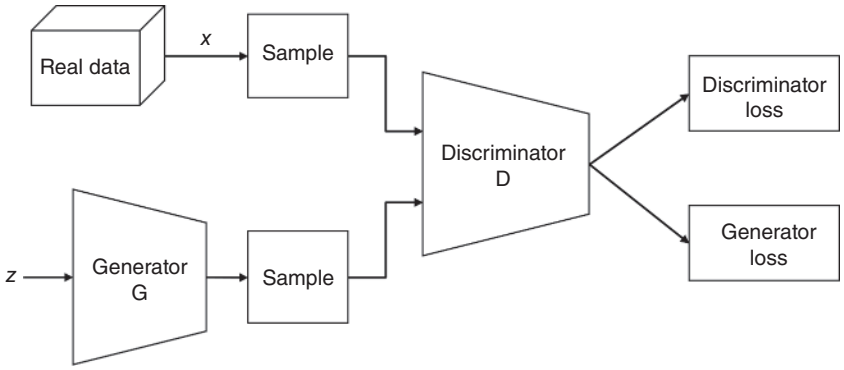

Figure 1.16 Illustration of a vanilla GAN architecture outlining the principle of a generator and a discriminator.

1.6.5.1 Minmax Loss

In the case of minmax loss, the objective of the discriminator is to maximize the expectation of the log likelihood of data drawn from the real distribution, i.e., $\max_D \mathbb{E}_{x\sim\mathbb{P}_r} \log(D(x))$, while minimizing the expectation of the log likelihood of data generated from the generator which samples from the random distribution $\mathbb{P}_z$, i.e., $\min_D \mathbb{E}_{z\sim\mathbb{P}_z} \log(D(G(z)))$ or equivalently $\max_D \mathbb{E}_{z\sim\mathbb{P}_z} \log(1 - D(G(z)))$. Thus, the objective of the discriminator function is

$$\max_D \mathbb{E}_{x\sim\mathbb{P}_r} \log(D(x)) + \max_D \mathbb{E}_{z\sim\mathbb{P}_z} \log(1 - D(G(z))) \tag{1.53}$$

On the other hand, the objective of the generator is to $\min_G \mathbb{E}_{z\sim\mathbb{P}_z} \log(1 - D(G(z)))$ such that the fake examples produced by the generator resemble the real data at the output of the discriminator. Thus, combining the two aspects and competing objective can be formulated as D and G are playing minmax game with the combined loss function:

$$\min_G \max_D [\mathbb{E}_{x\sim\mathbb{P}_r} \log(D(x)) + \mathbb{E}_{z\sim\mathbb{P}_z} \log(1 - D(G(z)))] \tag{1.54}$$

This is fine since $\mathbb{E}_{x\sim\mathbb{P}_r} \log(D(x))$ is independent of the generator optimization. It can be shown that the generator is trying to minimize the Jensen–Shannon (JS) divergence between $\mathbb{P}_r$ and $\mathbb{P}_g$. The JS divergence is bounded between 0 and 1 and is defined as follows:

$$D_{\mathrm{JS}}(p\|q) = \frac{1}{2} D_{\mathrm{KL}}\left(p\|\frac{p+q}{2}\right) + \frac{1}{2} D_{\mathrm{KL}}\left(q\|\frac{p+q}{2}\right) \tag{1.55}$$

It is worth noting that unlike the KL divergence used in VAEs, the JS divergence is symmetric and in case of two distributions being disjoint would result in a maximum value of log(2), irrespective of the two distributions. In comparison, the KL divergence would be ∞ in such a scenario. From Eq. (1.55), it is easy to see that the minimum value of $D_{\mathrm{JS}}(p\|q)$ is obtained when $p \sim q$. Consequently, the generator is trying to achieve $\mathbb{P}_g \sim \mathbb{P}_r$, which means that the generator generates data that resemble the real data. The discriminator maximizes the loss by trying to approach $D(x)$ to 1 and $D(G(z))$ to 0, thus attaining the optimal value of $D^*(x) = \frac{1}{2}$, which is the Nash equilibrium.

The minmax loss for GAN suffers from vanishing gradients and mode collapse. If the discriminator is too good, then the generator training can fail due to vanishing gradients. Furthermore, the generator in a random input GAN is expected to generate a variety of outputs. However, if a generator produces an especially plausible output, the generator may learn to produce only that output. If the generator starts producing the same output over several iterations, then the discriminator's best strategy is to reject that output always. But if the next iteration of discriminator gets stuck in a local minimum and does not find the best strategy, then it is too easy for the next generator iteration to find the most plausible output for the current discriminator. As a result, the generator gets trapped in a local minimum

and generates limited set of outputs, and this phenomenon is referred as mode collapse.

1.6.5.2 Wasserstein Loss

In the Wasserstein generative adversarial networks (WGANs), the discriminator does not classify input data as real or fake, but rather outputs a number. Discriminator training just tries to make the output bigger for real instances than for fake instances. Therefore, the discriminator in a WGAN is usually referred to as a critic rather than a discriminator. The discriminator tries to maximize the critic loss $D(x) - D(G(z))$, where $D(x)$ is the critic's output for a real instance, $G(z)$ is the generator's output for given z. $D(G(z))$ is the critic's output for fake data. Thus, it tries to maximize the difference between its output on real data and its output on fake data. The generator tries to maximize the generator loss $D(G(z))$. Thus, it tries to maximize the discriminator's output for its fake data. WGANs are less vulnerable to suffering model collapse and can avoid vanishing gradients issues.

1.6.6 Transformer

Transformer has been one of the most popular deep learning architectures recently due to its usability in a wide range of applications from natural language processing tasks to visual tasks and its state-of-the-art results in multiple public datasets. However, it is important to note that transformers come with high computational and memory requirements that might not be ideal for embedded solutions. Some works like Attention Is All You Need [44] focus on tackling the mentioned bottlenecks while mimicking the functions of a transformer. In the following paragraph, we provide an explanation of different blocks in a transformer which might ease readers to understand related works.

In [44] is introduced the idea of a transformer which is made of six encoders, six decoders, and uses machine translation as an application. A machine translation task takes a sentence or a phrase (sequence of words) as input and outputs the phrase translated into the target language. Each encoder is identical in architecture while having their own set of learnable weights and consists of a self-attention and a feedforward layer. The self-attention layer can be regarded as a context-aware encoding mechanism where it uses information from other words to encode it better. In a technical implementation perspective, the self-attention mechanism involves for each word three vectors namely Attention Is All You Need [44] query (Q), key (K), and value (V) which are generated by three different fully connected layers having output dimension smaller than that of the input embedding vector. In order to compute a score for each word against all other words in the phrase, a dot product is taken between the query vector of the word

and key vector of all the words in the phrase. The score is further divided by the square root of the dimension of the key vector d_K to stabilize the training by normalizing the gradients. Next, all the scores are passed through a softmax function to generate a normalized distribution. Finally, as shown in Eq. (1.56), the softmax output is multiplied with the value vector matrix to generate the output Z of the self-attention layer for the given position. This output is then simply fed to the following fully connected layer:

$$Z = \text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_K}}\right) V \tag{1.56}$$

Transformers also introduce the idea of multihead attention that involves having multiple self-attention layers initialized randomly in order to have different encodings to cover multiple subspaces. The multihead attention generates multiple outputs Z which are concatenated together and multiplied with a jointly trained weight vector W that projects them into a single vector which is fed to the

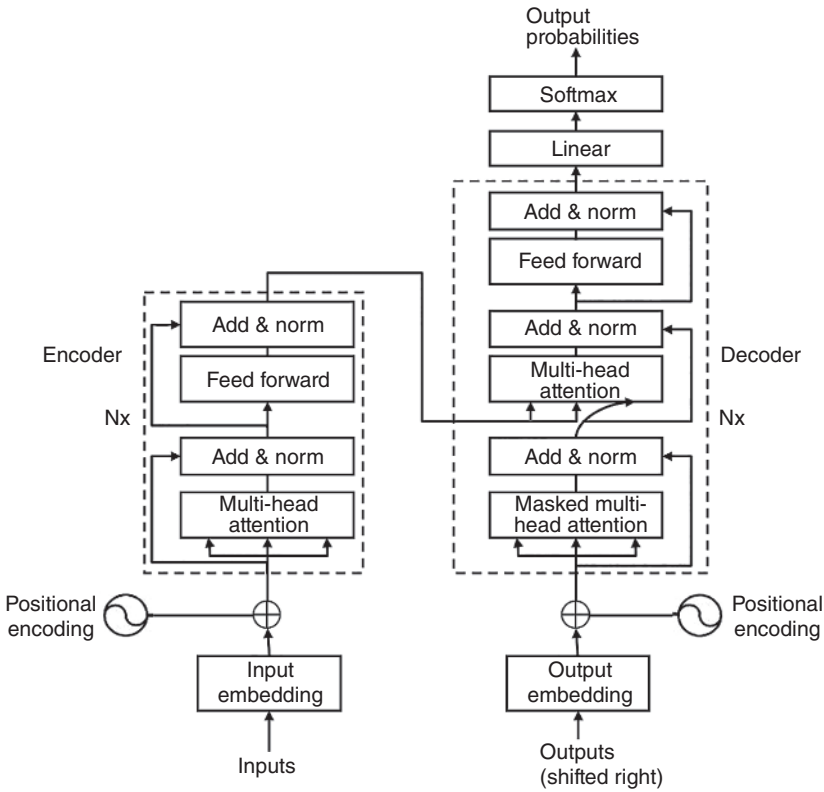

Figure 1.17 Transformer network architecture. Source: adapted from [44].

fully connected layer. In order to capture the order of words in a given sequence, a positional encoding is generated where each element in the encoding represent a sinusoid. This is then added to the word-embedding vector which results in the input vector to the encoder. Additionally, each encoder consists of a residual connection with a normalization layer. In the decoder, the incoming key and value vectors coming from the top encoder act as input which is used by the encoder–decoder attention layer. After each time step, the output of the decoder is fed back to the decoder along with a positional encoding which becomes the input for the self-attention layer in the decoder. The decoder self-attention layer is prevented from attending positions in future by masking all remaining places which are yet to be predicted to $-\infty$ and only the predicted output sequence goes as input to the self-attention layer in the decoder. The final layer of the decoder consists of a logit layer which has the dimension of all the possible words, and softmax is applied on it to select the one with highest probability as the predicted word.

The overall architecture of a transformer is depicted in Figure 1.17.

1.7 Training a Neural Network

For training a neural network, there are two steps namely forward pass and error backpropagation.

1.7.1 Forward Pass and Backpropagation

In the forward pass, the input is fed to the model and multiplied with weight vectors, and bias is added for each layer to compute the output of the model. The input x_i^l, activation u_j^l, and output o_j^l at lth layer of dense or fully connected layer is represented as follows:

$$\begin{aligned} x_i^{(l)} &= o_j(l-1) \\ u_j^{(l)} &= \sum_{i=1}^{N} w_{ij}^{(l)} x_i^{(l)} \\ o_j^{(l)} &= \sigma(u_j^{(l)}) \end{aligned} \tag{1.57}$$

where N denotes the number of neurons at the lth layer, w_{ij}^l are the weights that need to be learned for a task at the lth layer and $\sigma()$ is the activation function.

The backpropagation is described below. Considering one sample for which inputs $(x_1, x_2, \ldots, x_n)$ and expected outputs $(t_1, t_2, \ldots t_k, \ldots, t_m)$ and real outputs $(y_1, y_2, \ldots y_k, \ldots, y_m)$, the error for one sample is therefore $E = \frac{1}{2}\sum_{k=1}^{m}(y_k - t_k)^2$,

where y_k is a function of the weights $w_{ij}^{(l)}$. The weights are updated to minimize the error using the gradient descent algorithm and can be expressed as follows:

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \lambda \frac{\partial E}{\partial w_{ij}^{(l)}} \tag{1.58}$$

In Eq. (1.58), $\frac{\partial E}{\partial w_{ij}^{(l)}}$ can be computed as follows:

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \frac{1}{2} \sum_k \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial w_{ij}^{(l)}} \tag{1.59}$$

where $\frac{\partial E}{\partial y_k} = (y_k - t_k)$.

Since y_k is a function of $u_j^{(l)}$, it can be derived that

$$\frac{\partial y_k}{\partial w_{ij}^{(l)}} = \frac{\partial y_k}{\partial u_j^{(l)}} \frac{\partial u_j^{(l)}}{\partial w_{ij}^{(l)}} \tag{1.60}$$

$$\frac{\partial u_j^{(l)}}{\partial w_{ij}^{(l)}} = o_i^{(l-1)} \tag{1.61}$$

which is computed during the feed forward step.

Thus, putting it all together gives us

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = (y_j - t_j) \frac{\partial y_k}{\partial u_j^{(l)}} o_i^{(l-1)} = \frac{\partial E}{\partial u_j^{(l)}} o_i^{(l-1)} \tag{1.62}$$

Some of the important aspects during training of a neural network are the following:

1. Learning Rate: Each weight update is controlled by parameter λ known as the learning rate parameter. If the learning rate is too small, then it may result in very slow learning, can get trapped in local minima easily, and can keep running for many iterations. On the other hand, if the learning rate is large, then it may step over the minima, can fail to converge, and potentially diverge. So it is really important to choose a good learning rate based on the architecture, dataset, transfer function, etc. Figure 1.18 illustrates the effects of choosing small and large learning rates on the gradient descent.
2. Weight Initialization: It is important to randomize the weights during initialization; otherwise, symmetry in weights would prevent the network from learning. Usually, small random values are used which is highly important when the number of neurons in a layer grows, as the weighted sum may saturate the optimization function.
3. Overfitting and Underfitting: In machine learning, the objective is not only to minimize cost function on in-sample data, i.e., data available or seen, but also generalize on out-sample data, i.e., data not available or unseen during training.

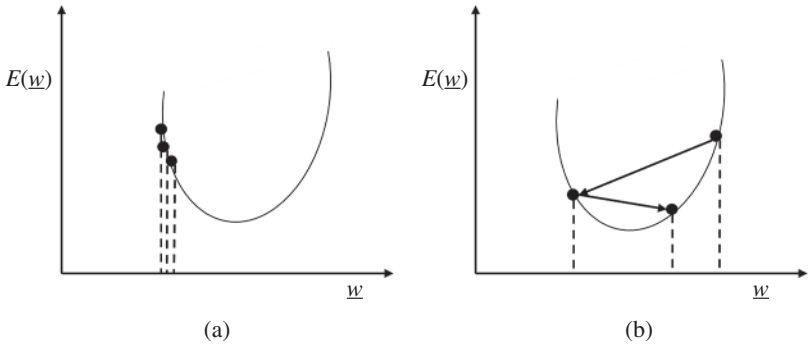

Figure 1.18 Illustration of the gradient descent when (a) the learning rate is small, and (b) the learning rate is large.

During training, the available dataset is divided into a training set, a validation set, and a test set. The training dataset is used to train the model, the validation dataset is used to set the hyperparameters of the model, and the test dataset is used for estimating the out-sample or generalization accuracy.

When the performance is poor on the training data, then it can be regarded as underfitting and is often due to poor choice of learning rate or if the neural network is under-dimensional. This error is referred to as "bias." The issue of underfitting is illustrated in the left column of Figure 1.19. The issue of overfitting arises when the performance is good on the training data, i.e., good approximation accuracy, but poor on the test or validation data, i.e., poor generalization accuracy. This phenomenon is also referred as "variance" and is illustrated in the right column of Figure 1.19. If the training set size is insufficient or the model complexity is too high for the data, the model memorizes or approximates the training data well but does not generalize well on test data, i.e., it

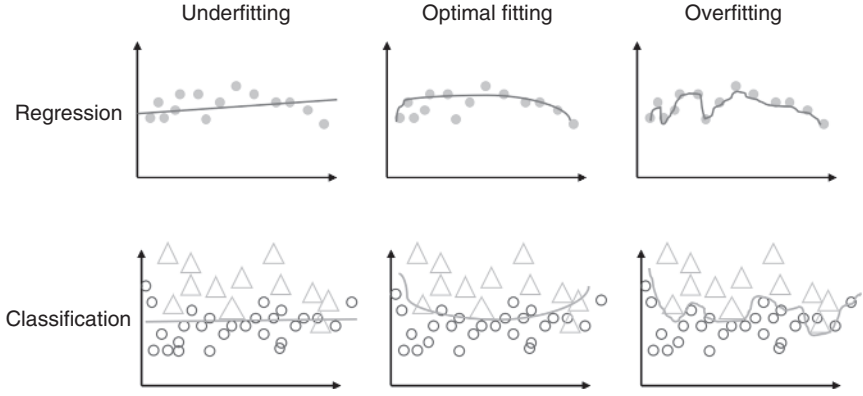

Figure 1.19 Illustration of underfitting and overfitting of a model.

overfits. The purpose of training a machine-learning model is to find a model as shown in the middle column of Figure 1.19, where the training error (bias) as well as the generalization error (variance) is minimized. Typically, training finds a model such that a balance between bias and variance can be achieved and often is referred as "bias-variance" trade-off. In case of deep learning, the "bias-variance" trade-off is not applicable since there are separate mechanisms to reduce bias and variance, thus the trade-off is not readily applicable.

4. Curse of Dimensionality: The other critical aspect in machine learning in general is the curse of dimensionality. Curse of dimensionality is closely related to overfitting. In high-dimensional spaces, most of the training data resides in the corners of the hypercube defining the feature space. Instances in the corners of the feature space are much more difficult to classify than instances around the centroid of the hyperactive sphere. Thus, as the number of features or dimensions grows, the amount of data we need to generalize accurately also grows exponentially.

1.7.2 Optimizers

Optimizers are methods that help to change weights and bias of the model such that a loss function is minimized. There are several modifications that have been proposed to the standard stochastic gradient descent (SGD) algorithm that $w_{t+1} = w_t - \lambda \partial_w E(w)$, where $E(w), \partial E(w)$ denotes the loss function and its derivative, respectively. w_{t+1} and w_t represents the weights after and before the update step and λ represents the learning rate. Following are list of optimizers that have been proposed in the that improves the standard SGD:

1. Momentum: It accelerates the SGD toward the relevant direction while reducing the oscillations. It basically adds a part of the previous weight updates to the current update vector ensuring that the direction of the previous update is retained to some extent while the current update gradient is used to fine-tune the final update direction. Momentum introduces another variable v_t and can be expressed as follows:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \lambda \partial_w E(w) \\ w_{t+1} &= w_t - v_t \end{aligned} \tag{1.63}$$

2. Nesterov Accelerated Gradient [45]: While momentum helps to reduce noise and also accelerates the convergence, it also introduces error. This is resolved in Nesterov accelerated gradient by including part of the previous weight updates to the current update vector to perform the weight update that is expressed as follows:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \lambda \partial_w E(w - \gamma v_{t-1}) \\ w_{t+1} &= w_t - v_t \end{aligned} \tag{1.64}$$

A typical value of $\gamma = 0.9$.

3. Adagrad [46]: The motivation of Adagrad is to have an adaptive learning rate for each parameter; however, earlier approaches have fixed learning rate. Adagrad ensures that different neurons of the hidden layer dependent on iterations have different learning rates. The intuition behind it is that large updates should occur for infrequent parameters and smaller for frequent parameters.

   For each weight updates, the learning rate is adapted as follows:

$$w_{t+1}t^i = w_t^i - \frac{\lambda}{\sqrt{G_t^i + \epsilon}} \partial_w E(w_t^i)$$
$$G_t^i = \sum_{tt=0}^{t} \left(\partial_w E(w_{tt}^i)\right)^2 \tag{1.65}$$

   since the sum of squared gradients grows continuously, it would lead to a smaller learning rates adaptively. The parameter ϵ helps in avoiding divide by zero issues.

4. RMSprop [47]: An issue of Adagrad is that after several iterations in DNNs, the learning rate becomes very small leading to the issue of dead neuron problems and results in no updates for these neurons. This issue is fixed by RMSprop, where learning can continue even after many parameter updates. In RMSprop, the learning rate is an exponential average of the gradients instead of the cumulative sum of squared gradients as in Adagrad. A moving average of a squared gradient for each weight is computed by limiting the gradient accumulation to a certain past and can be expressed as follows:

$$w_{t+1}^i = w_t^i - \frac{\lambda}{\sqrt{G_t^i + \epsilon}} \partial_w E(w_t^i)$$
$$G_t^i = \gamma G_{t-1}^i + (1-\gamma)\left(\partial_w E(w_t^i)\right)^2 \tag{1.66}$$

5. Adadelta [48]: Adadelta is another improvement over the Adagrad to continue learning after many parameter updates. But Adadelta is computationally expensive. Here the gradient accumulation is limited to a certain past update by computing a moving average of both the squared gradient and parameter updates for each weight parameter as follows:

$$w_{t+1}^i = w_t^i - \lambda_t \partial_w E(w_t^i) = w_t^i + v_t^i$$
$$G_{t+1}^i = \gamma G_t^i + (1-\gamma)\left(\partial_w E(w_{t-1}^i)\right)^2$$
$$\Delta w_{t+1}^i = \gamma \Delta w_t^i + (1-\gamma)\left(v_{t-1}^i\right)^2$$
$$\lambda_t = \frac{\sqrt{\Delta w_t^i + \epsilon}}{\sqrt{G_t^i + \epsilon}} \tag{1.67}$$

6. Adaptive Moment Estimation (ADAM) [49]: Adam optimizer is one of the most popular and widely used one today. It stores both the decaying average of the

past gradients similar to momentum and also the decaying average of the past squared gradients, similar to RMSprop and Adadelta. ADAM can be expressed as in the following equation, where momentum is added to RMSprop by using first and second moments, i.e., mean m_{t+1}^i and variance v_{t+1}^i of the gradient:

$$\begin{aligned} m_{t+1}^i &= \beta_1 m_t^i + (1-\beta_1)\partial_w E(w_t^i) \\ v_{t+1}^i &= \beta_2 v_t^i + (1-\beta_2)\left(\partial_w E(w_t^i)\right)^2 \\ w_t^i &= w_{t-1}^i - \frac{\lambda}{\sqrt{v_t^i + \epsilon}} m_t^i \end{aligned} \tag{1.68}$$

where β_1 and β_2 are the forgetting factor in the moving average implementation of the mean and variance of the gradient. Adam is easy to implement and computationally efficient and requires less memory owing to the moving average implementation.

1.7.3 Loss Functions

A neural network is formulated as an optimization problem. The candidate solution, which means the weights of the network, should minimize or maximize the score of the given objective function.

In the case of a regression problem, the objective is to predict a real-value quantity. In this case, linear activation unit is used at the output layer, and MSE is used as the loss function. The mean-square loss for regression is given as follows:

$$\mathrm{MSE}(y, \hat{y}) = |y - \hat{y}|^2 \tag{1.69}$$

where y and $\hat{y}$ are the true value and predicted value of the neural network, respectively.

For modeling a classification problem, the idea is to map the input variable to a class label implying that the objective is to predict the probability of an example for belonging to a particular class. Under maximum likelihood estimation, the training of the network is seeking to find a set of model weights that minimizes the difference between the model's predicted probability distribution given the dataset and the distribution of probabilities in the training dataset. This is called the CE loss, and in the case of binary classification is configured as a sigmoid activation at the output, while for multiclass classification, the softmax activation is used at the output. In both cases, the problem is formulated as predicting the maximum likelihood for a given input belonging to a particular class.

The binary CE loss for binary classification is given as follows:

$$\mathrm{CE}(p, \hat{y}) = -\hat{y}\log(p) - (1-\hat{y})\log(1-p) \tag{1.70}$$

where p is the probability of class 1, $1-p$ is the probability of class 0, and $\hat{y}$ is the predicted probability from the neural network.

1.8 Questions to the Reader

- Explain the processing pipeline of 2D angle-Doppler image-based detection. Explain a use-case where 2D image-Doppler image is preferred over range-Doppler or range-angle images.
- What are the adaptation in processing pipeline for sensing extended targets compared to point target processing?
- What are the advantages of OS-CFAR detector over CA-CFAR detector for extended targets? What is the need for guard cells in CA-CFAR? And why are they not absolutely necessary for OS-CFAR?
- What are the advantages and disadvantages of DBSCAN and Euclidean clustering?
- What does Cramer–Rao bound for estimating a radar parameter indicate? Derive the Cramer–Rao bound for angle estimation.
- What is the purpose of introducing nonlinearities in neural networks? How is it achieved in convolutional neural network or LSTM?
- Why is initializing all the weights of a neural network with the same value (e.g., 0.1) not a good idea?
- How does 2D CNN ensure invariance toward spatial translation? How can you extend 2D CNN to 3D CNN and which application can you think of?
- Explain the different configuration LSTM that can be used in a radar task, e.g., people counting or gesture sensing.
- Explain the reparameterization trick in VAE. Explain the advantage of GAN loss over VAE loss.
- What is mode collapse in GAN? How to identify a stable GAN training?
- What are the improvements proposed by Wasserstein GAN? How is the Lipschutz continuity implemented in practical Wasserstein GAN.
- What is bias–variance trade-off? What are the means of dealing with bias and variance in a neural network?
- What is global receptive field in deep CNN (DCNN)?
- What is the issue of Adagrad optimizer that RMSprop solves? What is the problem of dead neurons?

References

1 Meinel, H.H. (2014). Evolving automotive radar - from the very beginnings into the future. *The 8th European Conference on Antennas and Propagation (EuCAP 2014)*, pp. 3107–3114.

2 Li, Y., Du, L., and Liu, H. (2011). Moving vehicle classification based on micro-Doppler signature. *2011 IEEE International Conference on Signal*

Processing, Communications and Computing (ICSPCC), September 2011, pp. 1–4.

3 Dubey, A., Fuchs, J., Reissland, T. et al. (2020). Uncertainty analysis of deep neural network for classification of vulnerable road users using micro-Doppler. *2020 IEEE Topical Conference on Wireless Sensors and Sensor Networks (WiSNeT)*, pp. 23–26.

4 Xu, R. and Wunsch, D. (2005). Survey of clustering algorithms. *IEEE Transactions on Neural Networks* 16 (3): 645–678.

5 Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. *KDD-96* 96 (34): 226–231.

6 Dubey, A., Fuchs, J., Luebke, M. et al. (2020). Generative adversial network based extended target detection for automotive MIMO radar. *2020 IEEE International Radar Conference (RADAR)*, pp. 220–225.

7 Stephan, M. and Santra, A. (2019). Radar-based human target detection using deep residual U-net for smart home applications. *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pp. 175–182.

8 Dubey, A., Fuchs, J., Luebke, M. et al. (2020). Region based single-stage interference mitigation and target detection. *IEEE Radar Conference 2020*.

9 Qi, C., Su, H., Mo, K., and Guibas, L.J. (2016). PointNet: Deep learning on point sets for 3D classification and segmentation. *CoRR*, vol. abs/1612.00593. http://arxiv.org/abs/1612.00593.

10 Qi, C.R., Yi, L., Su, H., and Guibas, L.J. (2017). PointNet++: Deep hierarchical feature learning on point sets in a metric space. *CoRR*, vol. abs/1706.02413. http://arxiv.org/abs/1706.02413.

11 Guo, Y., Wang, H., Hu, Q. et al. (2019). Deep learning for 3D point clouds: a survey. *CoRR*, vol. abs/1912.12033. http://arxiv.org/abs/1912.12033.

12 Danzer, A., Griebel, T., Bach, M., and Dietmayer, K. (2019). 2D car detection in radar data with pointnets. *CoRR*, vol. abs/1904.08414. http://arxiv.org/abs/1904.08414.

13 Lee, S. (2020). Deep learning on radar centric 3D object detection.

14 Qi, C.R., Liu, W., Wu, C. et al. (2017). Frustum pointnets for 3D object detection from RGB-D data. *CoRR*, vol. abs/1711.08488. http://arxiv.org/abs/1711.08488.

15 Yang, Z., Sun, Y., Liu, S., and Jia, J. (2020). 3DSSD: Point-based 3D single stage object detector. *CoRR*, vol. abs/2002.10187. https://arxiv.org/abs/2002.10187.

16 Simon, M., Milz, S., Amende, K., and Gross, H. (2018). Complex-YOLO: Real-time 3D object detection on point clouds. *CoRR*, vol. abs/1803.06199. http://arxiv.org/abs/1803.06199.

17 Shi, S., Wang, Z., Wang, X., and Li, H. (2019). Part-a^2 net: 3D part-aware and aggregation neural network for object detection from point cloud. *CoRR*, vol. abs/1907.03670. http://arxiv.org/abs/1907.03670.

18 Yang, B., Wang, J., Clark, R. et al. (2019). Learning object bounding boxes for 3D instance segmentation on point clouds. *CoRR*, vol. abs/1906.01140. http://arxiv.org/abs/1906.01140.

19 Jiang, M., Wu, Y., and Lu, C. (2018). PointSIFT: A sift-like network module for 3D point cloud semantic segmentation. *CoRR*, vol. abs/1807.00652. http://arxiv.org/abs/1807.00652.

20 Duan, Z., Li, X.R., Han, C., and Zhu, H. (2005). Sequential unscented Kalman filter for radar target tracking with range rate measurements. *2005 7th International Conference on Information Fusion*, Volume 1, p. 8.

21 Lin, A. and Ling, H. (2006). Three-dimensional tracking of humans using very low-complexity radar. *Electronics Letters* 42 (18): 1062–1063.

22 Chang, S., Wolf, M., and Burdick, J.W. (2009). An MHT algorithm for UWB radar-based multiple human target tracking. *2009 IEEE International Conference on Ultra-Wideband*, pp. 459–463.

23 Kim, Y. and Ling, H. (2009). Through-wall human tracking with multiple Doppler sensors using an artificial neural network. *IEEE Transactions on Antennas and Propagation* 57 (7): 2116–2122.

24 Cortina, E., Otero, D., and D'Attellis, C.E. (1991). Maneuvering target tracking using extended Kalman filter. *IEEE Transactions on Aerospace and Electronic Systems* 27 (1): 155–158.

25 Chang, S., Sharan, R., Wolf, M. et al. (2009). UWB radar-based human target tracking. *2009 IEEE Radar Conference*, pp. 1–6.

26 Chang, S., Wolf, M., and Burdick, J.W. (2010). Human detection and tracking via ultra-wideband (UWB) radar. *2010 IEEE International Conference on Robotics and Automation*, pp. 452–457.

27 Sun, W., Huang, W., Ji, Y. et al. (2019). A vessel azimuth and course joint re-estimation method for compact HFSWR. *IEEE Transactions on Geoscience and Remote Sensing* 10: 1–11.

28 Vaishnav, P. and Santra, A. (2020). Continuous human activity classification with unscented Kalman filter tracking using FMCW radar *IEEE Sensors Letters* 4 (5): 1–4.

29 Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: unified, real-time object detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 779–788.

30 Liu, W., Anguelov, D., Erhan, D. et al. (2016). SSD: Single shot multibox detector. *European Conference on Computer Vision*, Springer, pp. 21–37.

31 Dai, J., Li, Y., He, K., and Sun, J. (2016). R-FCN: Object detection via region-based fully convolutional networks. In: *Advances in Neural Information Processing Systems*, vol. 29.

32 Prophet, R., Hoffmann, M., Vossiek, M. et al. (2018). Pedestrian classification with a 79 GHz automotive radar sensor. *2018 19th International Radar Symposium (IRS)*, pp. 1–6.

33 Prophet, R., Hoffmann, M., Ossowska, A. et al. (2018). Pedestrian classification for 79 GHz automotive radar systems. *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1265–1270.

34 Prophet, R., Hoffmann, M., Ossowska, A. et al. (2018). Image-based pedestrian classification for 79 GHz automotive radar. *2018 15th European Radar Conference (EuRAD)*, pp. 75–78.

35 LeCun, Y. and Bengio, Y. (1995). Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks* 3361 (10): 1995.

36 Krizhevsky, A., Sutskever, I., and Hinton, G.E. (2012). ImageNet classification with deep convolutional neural networks. In: *Advances in Neural Information Processing Systems*, vol. 25.

37 Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A.A. (2017). Inception-v4, inception-ResNet and the impact of residual connections on learning. *31st AAAI Conference on Artificial Intelligence*.

38 Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation* 9 (8): 1735–1780.

39 Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

40 Kingma, D.P. and Welling, M. (2019). An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning* 12 (4): 307–392.

41 Kipf, T.N. and Welling, M. (2016). Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*.

42 Pu, Y., Gan, Z., Henao, R. et al. (2016). Variational autoencoder for deep learning of images, labels and captions. In: *Advances in Neural Information Processing Systems*, vol. 29.

43 Goodfellow, I., Pouget-Abadie, J., Mirza, M. et al. (2014). Generative adversarial nets. In: *Advances in Neural Information Processing Systems*, vol. 27.

44 Vaswani, A., Shazeer, N., Parmar, N. et al. (2017). Attention is all you need. In: *Advances in Neural Information Processing Systems*, vol. 30.

45 Nesterov, Y. (2013). Gradient methods for minimizing composite functions. *Mathematical Programming* 140 (1): 125–161.

46 Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12 (7).

47 Tieleman, T. and Hinton, G. (2012). Lecture 6.5-Rmsprop, Coursera: Neural Networks for Machine Learning. *University of Toronto, Technical Report*, vol. 6.

48 Zeiler, M.D. (2012). ADADELTA: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

49 Kingma, D.P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.