Ain Shams University
Faculty of Engineering
Cairo – Egypt

Electronics and Electrical Communications Engineering Department

Machine Learning Techniques for PAPR Reduction in Multicarrier Communication Systems

A Report

Submitted in partial fulfillment of the requirements of the degree of B.Sc. in **Electronics and Communications Engineering**

Submitted by

Youssef Salah El-Din Mustafa
Marwan Yasser Yehia
Moustafa Ahmed Hussein
Ahmed Fawzy Ahmed
Rana Mohamed Yassine
Raneem Gaber Hassan
Metwaly Yahia Metwaly
Amr Saeed Kamel

Supervised By

Dr. Michael Ibrahim

Cairo 2022

Acknowledgment

All praise is due to ALLAH, Lord of the worlds. If it were not for ALLAH's help and mercy, we would not be able to complete this project.

We are extremely thankful to our supervisor Dr. Michael Ibrahim for his guidance, support, understanding and for sharing with us a very helpful papers and references that helped us a lot during our project. It was a great honour to learn from his precious experience.

We would like to thank our families for their support and patience, and thanks for all the team members for the cooperation, support and helping each other, and for the effort they put in this project.

Abstract

Orthogonal frequency division multiplexing (OFDM) is a promising high data rate transmission scheme due to its outstanding performance in frequency selective fading channels. However, the high peak-to-average power ratio (PAPR) of OFDM systems will cause the transmitted signal to enter into the nonlinear region of the high power amplifier (HPA), which leads to the signal distortion. In previous research, various schemes were proposed to solve the PAPR problem. The Tone Reservation (TR) technique has been received considerable attention, where a small number of subcarriers are reserved to generate the peak-canceling signal that is used to reduce the PAPR of the transmitted signal. Since the peak reduction tones (PRT) can be directly removed at the receiver side, the TR technique is capable of reducing PAPR without causing any additional signal distortion. However, the PAPR performance of the TR technique is significantly limited by the finite candidate set.

Selective mapping (SLM) is an attractive method to reduce the peak-to-average power ratio (PAPR) without causing any signal distortion in orthogonal frequency division multiplexing (OFDM) systems. The main drawback of this technique is the requirement of sending side information (SI) to the receiver for each data block which results in huge bandwidth and power overheads.

In recent years, artificial intelligence (AI), especially deep learning, has been achieved great success in many fields, the intelligent communication has been considered as one of the mainstream directions in the field of wireless communications. The basic idea of the intelligent communication is that introducing AI into the levels of wireless communication systems for improving the performance of wireless communication systems. At present, deep learning has showed good performances in channel estimation, signal detection, channel decoding, modulation recognition, and end-to-end wireless systems.

In terms of the PAPR reduction, we propose a Tone Reservation network **TRNet** based on deep learning to further enhance the PAPR performance of the conventional TR technique by adaptively generating the peak cancelling signals according to the characteristics of input data.

We also propose a PAPR reduction scheme, known as PAPR reducing network **PRNet**, based on the autoencoder architecture of deep learning. In the **PRNet**, the constellation mapping and demapping of symbols on each subcarrier is determined adaptively through a deep learning technique such that both the bit error rate (BER) and the PAPR of the OFDM system are jointly minimized.

Contents

Table of Figures

CHAPTER I - Literature Review on OFDM

OFDM System

Introduction

Any communication system nowadays has a main target, that is to transmit data with high rate, minimum loss, and high bandwidth efficiency. To achieve these goals there are many schemes introduced over the last years. We can categorize communication systems techniques into two main category, single carrier technique and multi carrier technique.

In Single Carrier Transmission, means one Radio Frequency carrier is used to carry the information. Hence information in the form of bits is carried by one single RF carrier. Simply the signal will be upconverted. it is observed that it will not be suitable for High Data Rate, As the required data rate increases, the symbol duration Ts must become very small to achieve the required data rate [1], and the system bandwidth becomes very large. So, in case of wired channel, we simply represent the channel as a LPF which has a certain cut off frequency, so the channel is band-limited, as mentioned we target a high data rate which corresponds to large band width so if this band width has frequency components beyond this cut off frequency the high order frequency components will be filtered out, so if we represent the transmitted bits as a rectangular pulse in time domain the sharp edges or the fast rate of change of the signal will be removed ,another aspect that in frequency domain the higher frequency component of the sinc function will be removed so this equivalent in time domain in spreading of the rectangular pulse, and since we transmit pulses sequentially we will observe inter symbol interference between symbols. A practical solution to this problem is to use a pulse shaping filter as sinc or raised cosine instead of rectangular pulse to avoid ISI. Although in wireless channel the channel band width become somehow very large with respect to band width of the signal so no filtering will occur to high frequency components of the transmitted signal but there is another problem that the signal will suffer from multi path fading. When the signal bandwidth becomes larger than the coherence bandwidth which is defined as the band width at which the channel is flat, (The coherence bandwidth is inversely proportional to the root-mean-square (rms) delay spread) in the wireless channel, the link suffers from multi-path

fading and the channel is described as frequency selective channel which means that every path will have a different channel gain and delayed with certain time (dispersion in time domain), which result in inter-symbol interference (ISI) .to avid ISI duration of pulses should be larger than the maximum delay spread which equivalent to narrow band width which is in contrast with our target (high data rate).

The channel has variation in both time and frequency which translated to (fast – slow) fading channel and (flat – frequency selective) fading channel. So, the channel should be compensated with an equalizer. In general, adaptive equalizers are employed to deal with the ISI resulted from the time-varying multi-path fading channel. Furthermore, the complexity of equalizer increases as the data rate increases [2]. So, the conclusion from this discussion is that single carrier transmission will not be suitable for high data rate. However, there are some applications for it in Satellite communication systems, GSM, CDMA, and other radio systems use single carrier for transmission and reception.

In Multi-Carrier Transmission, the wide band signal of a single carrier, which is higher than the coherence bandwidth, thus suffer from frequency selective channel can be divided into N independent flat sub-channels as in Figure 1, so every flat sub channel requires a simple equalizer instead of complex one, and the signal will not suffer from frequency selective channel as sub-band is smaller than the coherence bandwidth.

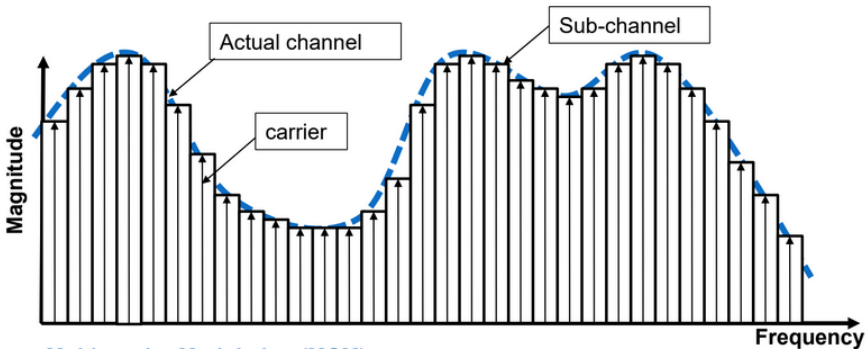


Figure 1 frequency response of multichannel transmission system [3]

The different symbols are transmitted over an equally separated subcarriers in parallel form as seen in Figure 2Figure 2

but there is a main drawback in this structure that we need multiple local oscillators to generate subcarriers. The transmitted signal ,X(t)=$S_0 e^{j2\pi fct} + S_1 e^{j2\pi(fc+\Delta f)t} + \cdots + S_{N-1} e^{j2\pi(fc+(N-1)\Delta f)t}$

$$x(t) = \sum_{K=0}^{N-1} S_K e^{j2\pi(fc+k\Delta f)t} \quad eq.(1)$$

where Δf is the frequency separation between every subcarrier and S_k is the transmitted signals carried on subcarrier of frequency $fc + k\Delta f$.

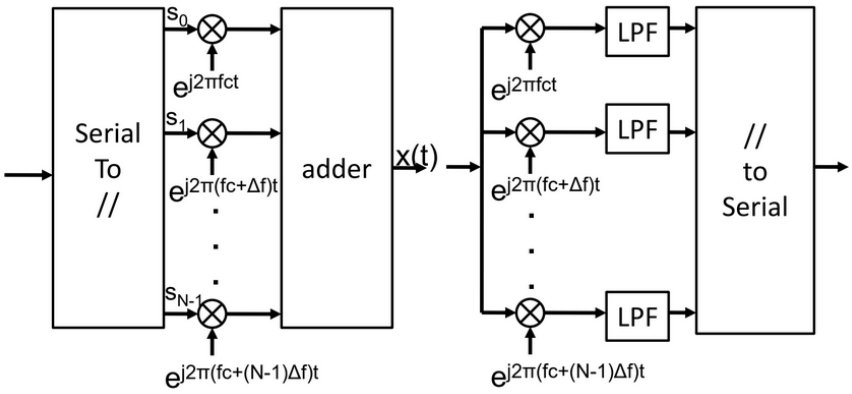


Figure 2 Basic structure of multicarrier system

FDMA (frequency division multiple access) method is the oldest, and most simple, multiaccess method, each user is assigned a frequency (sub)band as illustrated Figure 3 and all symbols are sent simultaneously. Despite the simplicity of this method but it has some disadvantages for narrow sub-Bands frequencies, Local oscillators thus must be very accurate and stable as jitters in the carrier frequency result in adjacent channel interference. Also, to select the desired signal we need a very steep filter. However, both local oscillators and sharp filters are expensive, another solution to this problem is including guard bands between sub- carriers to avoid interference as they are not orthogonal, so this will lead to low spectral efficiency.

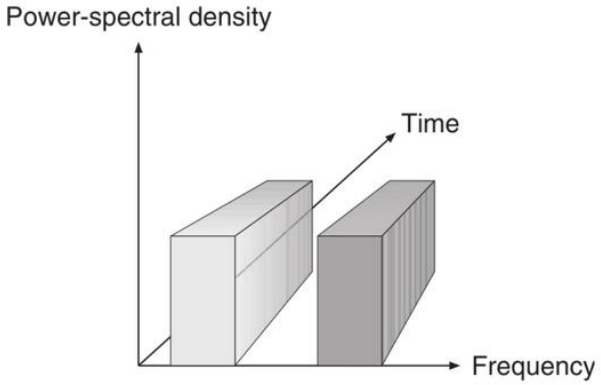


Figure 3 principle of FDMA [14]

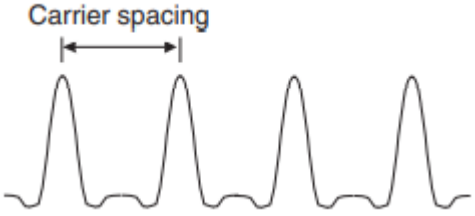


Figure 4 Spectrum of FDMA signal [2]

What is an OFDM System?

Orthogonal frequency division multiplexing (OFDM) is a multicarrier transmission system that is suitable for high data rate transmission, instead of single carrier transmission schemes that has many drawbacks in high data rate transmission. The concept of OFDM techniques was first proposed by RW Chang in 1965. In 1967, Saltzberg analyzed the performance of OFDM systems. In 1970, OFDM technology was patented at the USPD and then after it was used in military communication systems. In 1971, SB Weinstern and PM Ebert employed the Discrete Fourier Transform (DFT) into the MCM. in practice, the Fast Fourier Transform (FFT) implementation of the DFT has made OFDM modulation and demodulation feasible and very successful. In the 1980s, ISI was decreased when Peled and Ruiz added cyclic prefix (CP) into the basic OFDM signals. Nowadays OFDM has been adopted for many broadcast standards, such as Digital Audio Broadcasting (DAB) standards, Digital Video Broadcasting (DVB) standards. and being used in many wireless communication systems including the IEEE 802.11, Long-Term Evolution (LTE) and LTE Advanced systems and it has been adopted to be used in many wireless standards such as WLAN and WMAN, due to its robustness in multipath propagation and the relatively low complexity transceiver design. [3]. Also, OFDM technology is used in optical communication, namely, optical OFDM(O-OFDM), The principle of O-OFDM is similar to that of OFDM. The only difference is that the signal is transferred from wireless signal in electrical domain to optical signal in optical domain.

In the following section we will discuss the concept of OFDM.

If we compare eq. (1) with IDFT inverse discrete Fourier transform equation which state that

$$x(n) = \frac{1}{N} \sum_{K=0}^{N-1} x(k) e^{\frac{j2\pi Kn}{N}} \quad (2)$$

Where $0 \leq n \leq N-1$

IDFT equation is considered as the digital representation to multi carrier analog equation with scaling factor 1/N and frequency difference 1/N (here the signal becomes discrete in both frequency and time domain). So, we can replace multi carrier analog transmitter by IDFT block, also in the receiver, we can replace correlators by DFT block which is simpler and cheaper.

OFDM does not use oscillators for each subchannel so it over comes the disadvantage of the initial multi access scheme and furthermore, the spectra of subcarriers are overlapped for better bandwidth efficiency, where the wideband is fully divided into N orthogonal narrowband subchannels. All subcarriers are of the finite duration T, the spectrum of the OFDM signal can be considered as the sum of the frequency shifted sinc functions in the frequency domain as illustrated in Figure 5, where the overlapped neighboring sinc functions are spaced by

$$\Delta f = \frac{1}{Tsym} = \frac{1}{NTs} \quad (3)$$

Where T_{sym} is the OFDM symbol duration, N is the number of subcarriers, and Ts is the symbol time.

In practice, discrete Fourier transform (DFT) and inverse DFT (IDFT) processes are useful for implementing these orthogonal signals. Note that DFT and IDFT can be implemented efficiently by using fast Fourier transform (FFT) and inverse fast Fourier transform (IFFT) respectively, but the input to FFT or IFFT must be 2^j element.

Due to orthogonality between subcarriers as stated in equation 4, OFDM doesn't requires filters to separate sub-bands in the receiver side and can utilize the bandwidth efficiently.

$$\frac{1}{T_{sym}} \int_0^{Tsym} e^{j2\pi(f_k)t} * e^{-j2\pi(f_i)t} dt = \begin{cases} 1 & for\ K = i \\ 0 & otherwise \end{cases} \quad (4)$$

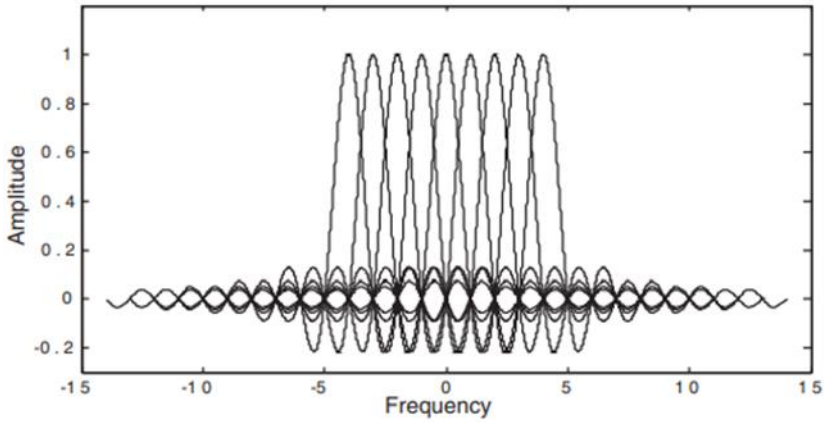


Figure 5 The spectrum of OFDM signal (linear scale) [1]

Due to multipath channel, the receiver has many versions of the transmitted data which causes two main problems [4]:

- ISI (inter symbol interference).
- ICI (inter carrier interference).

ISI problem: for a discrete-time channel with impulse response h(t) (three delayed passes from each other) as shown in figure 6 with the three passes added to each other each path has different gain and delayed with a certain time, if we take a certain FFT trigger point within a certain range taking N samples at the receiver, it will receive a distorted signal due to ISI. Simply this problem can be solved by adding a guard interval between every OFDM symbol and it must be larger than the maximum delay spread as shown in figure 8. For sure guard interval will reduce bandwidth efficiency but not as much as in FDM. from here we can calculate a time efficiency that define the percentage we can benefit from transmission time in sending information.

$$\eta = \frac{NT_s}{NT_s + TG}$$

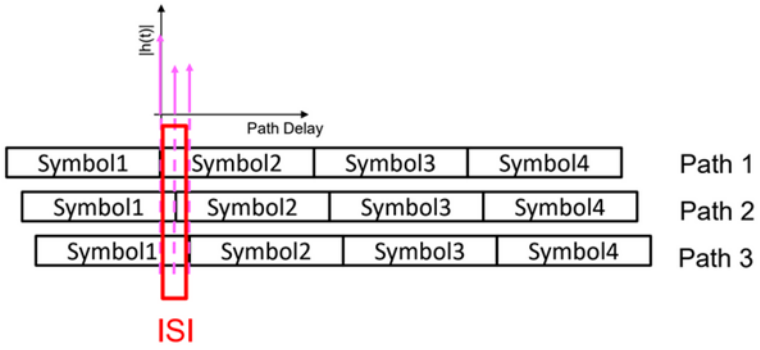


Figure 6 Impulse response of discrete time channel and effect of ISI [4]

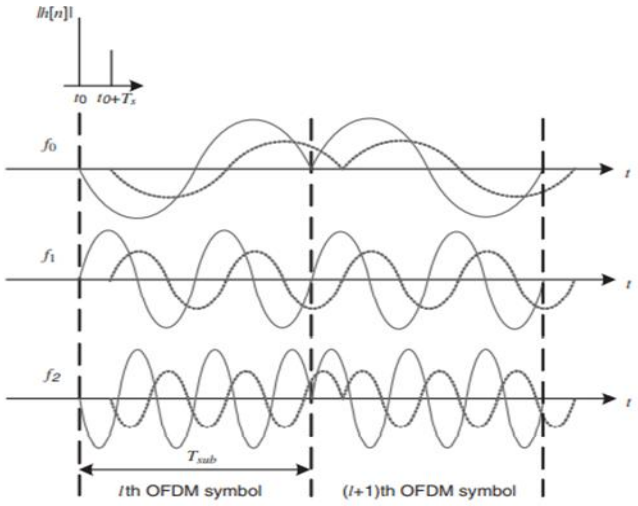


Figure 7 ISI effect of a multipath channel on the received signal

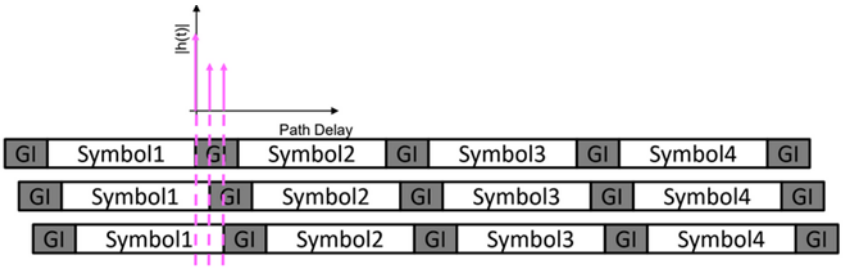


Figure 8 OFDM symbols with no ISI after adding GI [4]

ICI Problem: ICI means the crosstalk between different sub-carriers which means that they are no longer orthogonal to each other. Wireless propagation channels are also time varying, and thus time selective and frequency dispersive. Due to the Doppler effect, time selectivity creates ICI. A Doppler shift (frequency offset) of one subcarrier can cause ICI in many adjacent subcarriers (see figure 9). The spacing between the subcarriers is inversely proportional to symbol duration. Thus, if symbol duration is large, even a small Doppler shift can result in appreciable ICI. [5]

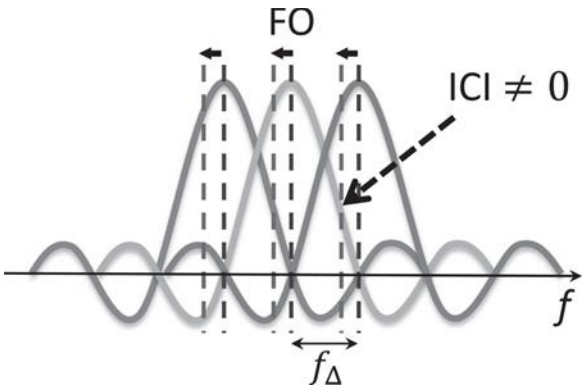


Figure 9 ICI due to carrier frequency offset

Simply OFDM can overcome these two problems by adding a Guard interval called Cyclic Prefix, as shown in figure 10. CP is to extend the OFDM symbol by copying the last samples of the OFDM symbol into its front. Let TG denote the length of CP in terms of samples. Then, the extended OFDM symbols now have the duration of Tsym =Tsub + TG.

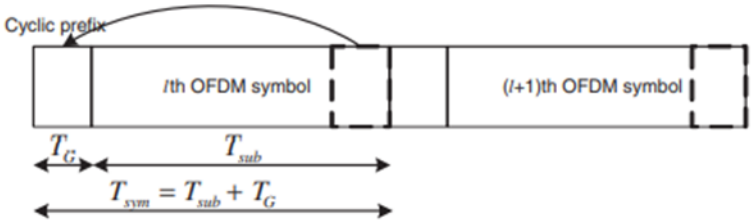


Figure 10 OFDM symbols with CP [2]

CP must be longer than maximum delay spread of the channel to avoid both ISI and ICI, but this is under a certain timing FFT window start point. More specifically, if the FFT window start point is earlier than the lagged end of the previous symbol, ISI occurs; if it is later than the beginning of a symbol, not only ISI (caused by the next symbol), but ICI also occurs (see figure 11 [2]).

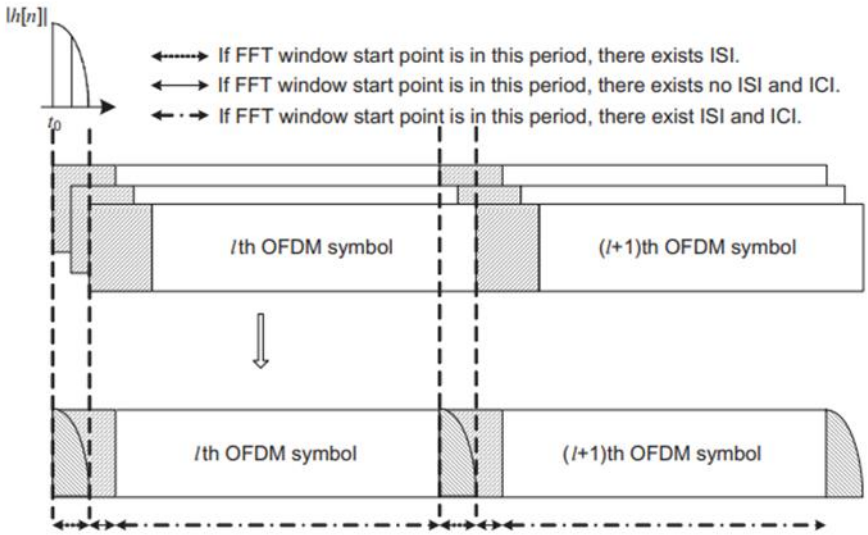


Figure 11 ISI/ICI effect depending on the FFT window start point [2]

Figure 12 illustrate how the OFDM signal distributed in both frequency and time domain in which X-axis represent frequency, Y-axis represent time and Z-axis represent the amplitude of the signal.

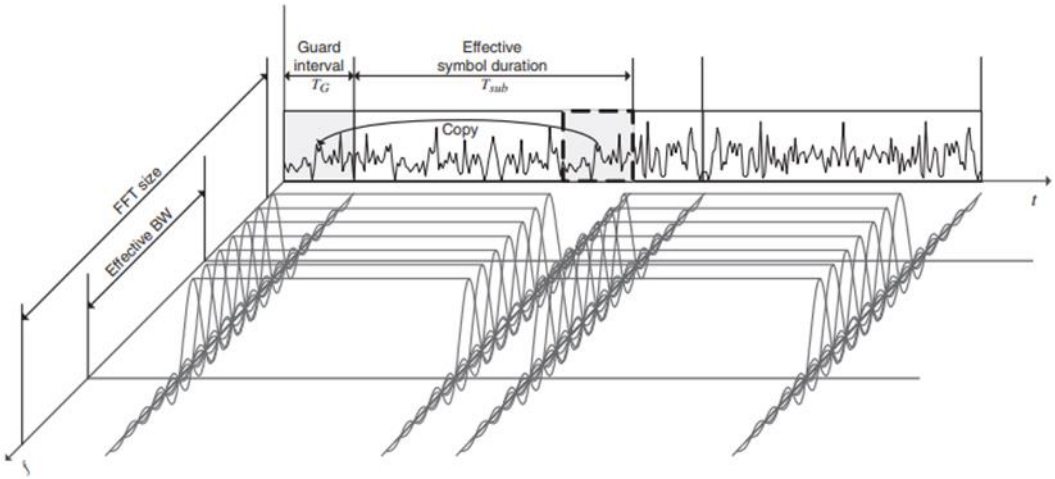


Figure 12 Time/frequency-domain description of OFDM symbols with CP

Implementation of OFDM system

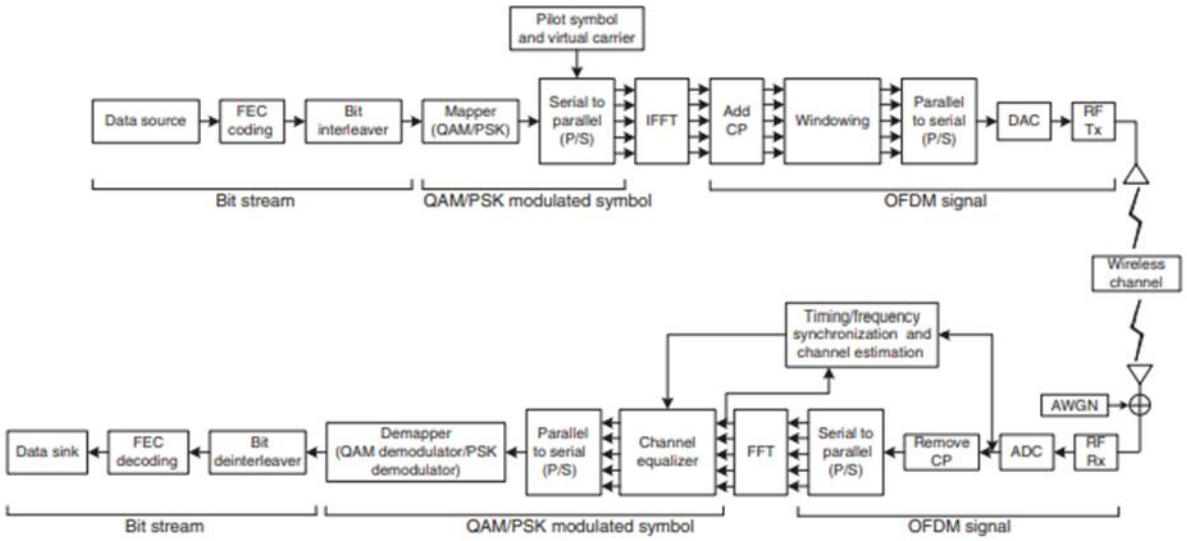


Figure 13 Block diagram of transmitter and receiver in an OFDM system [2]

OFDM transmitter first part is preparing the bit stream that will be processed after that to be suitable for transmission over a noisy and unreliable channel, so using a forward error correction coding (FEC coding) is an essential part in any digital communication system, it is defined as adding a redundancy bits to the information message to protect it against errors, also it can be defined as a process of detecting and correcting bit errors in digital communication systems without the need for retransmission. **Linear Block Code and Convolutional code** are examples of FEC coding. The Linear Block Code is a memory less operation as codewords are independent from each other. the Information sequence is segmented into message blocks of fixed length, so it is having the term Block, where each k-bit information message is encoded into an n-bit distinct codeword such that (n>k). the transmitted code word will have the following structure in figure 14, and Modulo-2 sum of any two codewords is also a codeword, so it has the term Linear. The code word is **V=U.G** where G is the generator matrix, U message matrix and V is the code word. The generator matrix should be linearly independent in order to obtain a distinct code word. [6]

While **convolutional code** operates on the incoming message sequence continuously in a serial manner (bit by bit) not as blocks as in Linear block code, also it has shift registers (memory elements) which means that the encoding of the current state will depend on the previous state and past elements. convolutional code is specified by three parameters (n, k, K), where:

- **n** is the output bit length
- **k** number of inputs per clock
- **K** is called the constraint length of the encoder where the encoder has K-1 memory elements.

(k/n) is the coding rate and determines the number of data bits per coded bit [6].

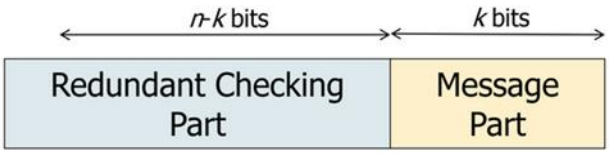


Figure 14 Structure of codeword

After FEC coding data we use bit interleaver the rearrange the distribution of bits on subcarriers to avoid burst errors as we mentioned that we use channel coding so if we insert all the bits of the codeword sequentially on subcarriers any deep fading occurs to one bit will affect on the whole codeword as the adjacent subcarriers are correlated and FEC will not be sufficient to overcome this effect so that's why we use bit interleaver. then we modulate the signal, the message bits get mapped into a sequence of PSK or QAM symbols, which will be subsequently converted into N parallel streams. Each of N symbols from serial-to-parallel (S/P) conversion is carried out by the different subcarrier. Due to the S/P conversion, the duration of transmission time for N symbols is extended to NTs, which forms a single OFDM symbol with a length of Tsym (i.e., Tsym = NTs).

The OFDM signal is generated by implementing Inverse Fast Fourier Transform (IFFT), which is used to convert signal from frequency domain to time domain, then adding CP and converting the OFDM signal from parallel-to-serial (P/S) and from digital to analog using DAC to be ready for transmission through a wireless channel, which will add noise to the signal (assume AWGN).

Channel in communication system introduces attenuation, phase difference, delay in addition to noise to the signal. We can represent the transmitted signal as in eq.5

$$\boldsymbol{y(t) = \alpha e^{j\theta} x(t - \tau d) + v(t)} \qquad \boldsymbol{eq.5}$$

Where $\boldsymbol{\alpha}$ is the attenuation,$\boldsymbol{\theta}$ the phase shift, τd is the delay of the wireless channel, x(t) is the transmitted signal and v(t) is the AWGN. In addition, in wireless channels there are multi path components of the transmitted signal received at the receiver, these different passes may be Line of sight, scattered, reflected, or diffracted and they result in fading and delay dispersion, Delay dispersion means that if we transmit a signal of duration T, the received signal has a longer duration $T^{\backslash}$ which in turn results

in inter symbol interference ISI. These issues have a large impact on the received signal.

We should take this impact into account to be able to recover the transmitted signal correctly.

Before introducing different channel models, we want to briefly explain the definition and types of fading. There are two types of fading small-scale fading and large-scale fading.

Small scale fading is defined as, the changing of the total signal amplitude due to interference of the different MPCs, this interference may be constructive or destructive as in figure 15, it depends on the run length of the MPC so small movement can result in large change in signal amplitude this type of fading occurs on spatial scales on the order of one-half wavelength. That's why we call it small scale fading.

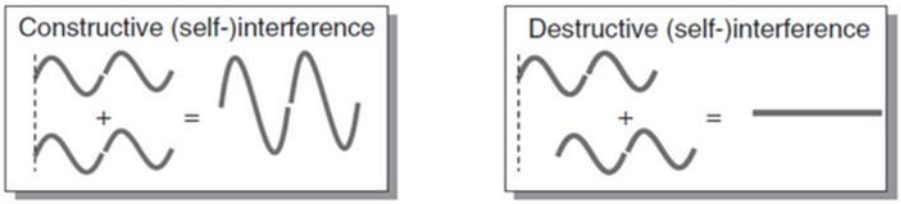


Figure 15 Principle of small-scale fading [7]

For large scale fading it occurs due to existence of large obstacles in the propagation medium. Such obstacles can lead to shadowing of one or several MPCs. And to overcome this shadow the receiver should move large distance to be away from this obstacle. [7]

The transmitted signal suffers from path loss or small-scale fading or large-scale fading or most probably a combination of them so in communication system we introduce different models of the communication channel to be closer to what is really affecting the transmitted signal as much as we can. However, in many circumstances, it is too complicated to describe all reflection, diffraction, and scattering processes that determine the different Multi Path Components (MPCs). Rather, it is preferable to describe the probability that a channel parameter attains a certain value [5].

Due to the large number of IOs, a deterministic description of the wireless channel is not efficient anymore, that is why we consider stochastic description methods. This stochastic description is essential for the whole field of wireless communications [5].we will discuss two of channel models that are widely used.

Rayleigh distribution:

Assuming a large number of MPC without a dominant component (LOS is not guaranteed), and giving the constituting waves a complex representation with random phase and amplitude, and according to the central limit theorem: when superimposing N statistically independent random variables.
none of which is dominant, the associated probability density function (pdf) approaches a normal distribution for $N \to \infty$ so we can say that the real and imaginary values of the received fields have a normal (Gaussian distribution as shown in figure 16 and 17.
A zero-mean Gaussian random variable has the pdf:

$$pdfx(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-x^2}{2\sigma^2}\right)$$

Where σ^2 denotes the variance.

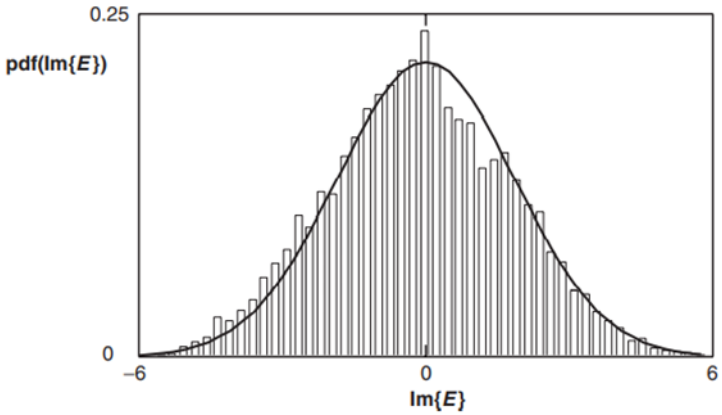


Figure 16 A Gaussian pdf for the imaginary part [5]

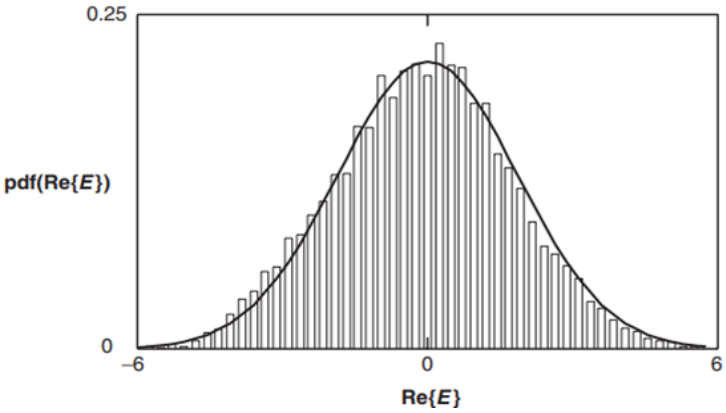


Figure 17 A Gaussian pdf for the real part [5]

A Rayleigh distribution describes the magnitude of a complex stochastic variable whose real and imaginary parts are independent and normally distributed. thus,

starting with the statistics of the real and imaginary parts, the statistics of amplitude and phase of the received signal can be derived.

It can be shown that the pdf of the phase ψ is a uniform distribution:

$$pdf_{\Psi}(\Psi) = \frac{1}{2\pi}$$

Where $0 \leq \Psi < 2\,\pi$

And that the pdf of the amplitude r is a Rayleigh distribution:

$$pdfr(r) = \frac{r}{\sigma^2}\exp\left(\frac{-r^2}{2\sigma^2}\right)$$

Where $0 \leq r < \infty$

For r < 0 the pdf is zero, as absolute amplitudes are positive.

A Rayleigh distribution has the following properties, which are also shown in figure 18 Mean value $\bar{r} = \sigma\sqrt{\frac{\pi}{2}}$,

- Mean square value $\overline{r^2} = 2\sigma^2$
- Variance $\overline{r^2} - (\bar{r})^2 = 2\sigma^2 - \sigma^2\frac{\pi}{2}$,
- Location of maximum max {pdf (r)} occurs at r = σ [5]

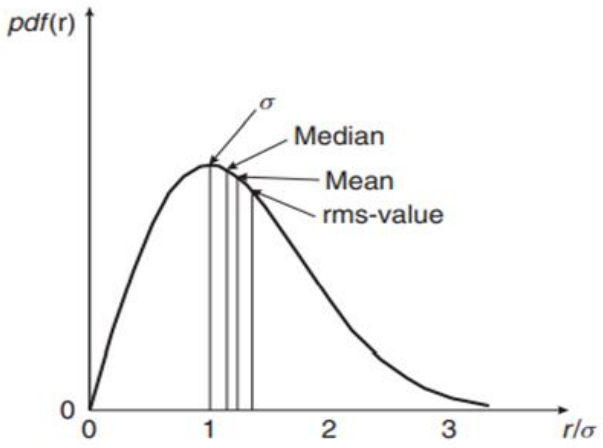


Figure 18 Pdf of a Rayleigh distribution. [5]

The Rayleigh distribution is widely used in wireless communications. This is due to several reasons:

- It is an excellent approximation in a large number of practical scenarios, as confirmed by a multitude of measurements. However, it is noteworthy that there are scenarios where it is not valid. These can occur, e.g., in Line of Sight (LOS) scenarios, some indoor scenarios, and in (ultra) wideband scenarios.
- It describes a worst-case scenario in the sense that there is no dominant signal component, and thus there is a large number of fading dips. Such a worst-case assumption is useful for the design of robust systems.

- It depends only on a single parameter, the mean received power – once this parameter is known, the complete signal statistics are known
- Mathematical convenience: computations of error probabilities and other parameters can often be done in closed form when the field strength distribution is Rayleigh.

Rice distribution

when a dominant MPC – e.g., an LOS component or a dominant specular component – is present, in this case, described by the Rician distribution, the fading becomes more benign since the specular components can no longer cancel each other. The histogram of the absolute value of the field strength is shown in figure 19.It is clear that the probability of deep fades is much smaller than in the Rayleigh-fading case.
Assume that the LOS component has zero phase, so that it is purely real. The real part thus has a nonzero-mean Gaussian distribution, while the imaginary part has a zero-mean Gaussian distribution. So, we get the joint pdf of amplitude r and phase ψ as follow:

$$pdf_{r,\psi}(r,\psi) = \frac{r}{2\pi\sigma^2}\exp\left(\frac{-r^2 + A^2 - 2rA\cos(\psi)}{2\sigma^2}\right.$$

where A is the amplitude of the dominant component.
The pdf of the amplitude is given by the Rice distribution:

$$pdf_r(r) = \frac{r}{\sigma^2}\exp\left(\frac{-r^2 + A^2}{2\sigma^2}\right).I_o\left(\frac{rA}{\sigma^2}\right) \qquad 0 \le r < \infty$$

I0(x) is the modified Bessel function of the first kind, and the mean square value of a Rice-distributed random variable r is given by:

$$\overline{r^2} = 2\sigma^2 + A^2$$

The ratio of the power in the LOS component to the power in the diffuse component, $A^2/(2\sigma^2)$, is called the Rice factor Kr. figure 20 shows the Rice distribution for three different values of the Rice factor. The stronger the LOS component, the rarer the occurrence of deep fades. For Kr → 0, the Rice distribution becomes a Rayleigh distribution, while for large Kr it approximates a Gaussian distribution with mean value A [5].
so wireless channel can be represented by different models depending on a lot of factors such as the nature of the environment that the signal travels through, the receiver and transmitter motion, the channel is time varying or stable and too many other factors must be taken into consideration, so to overcome the impact of the channel on the transmitted signal we have to do channel estimation and equalization in the receiver side this will be discussed in the next section.

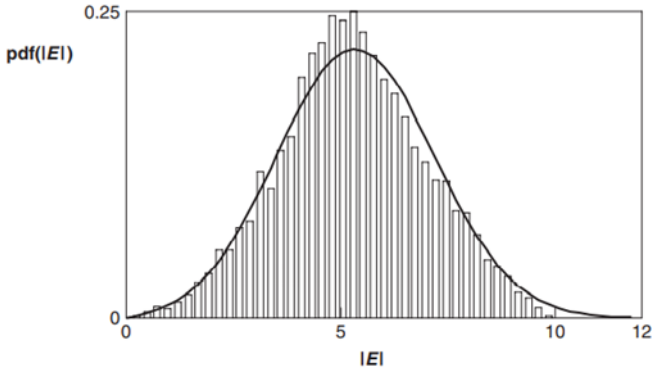


Figure 19 Histogram of the amplitudes in the presence of a dominant MPC. [5]

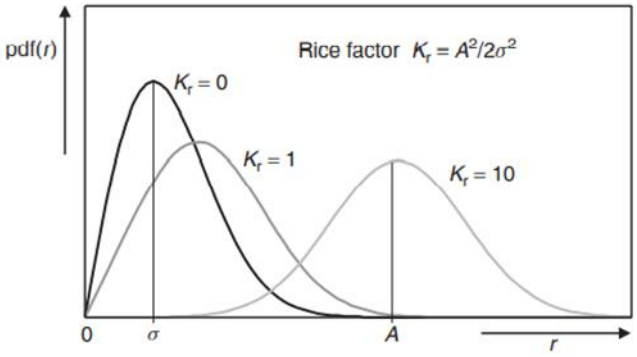


Figure 20 Rice distribution for three different values of Kr [5]

At the receiver, for wireless communication system, the radio channel is highly dynamic to make the design of the receiver to be a challenging task.
It starts from converting the signal from analog to digital using ADC and CP is removed from signal, then a S/P conversion is done to get the signal ready for Fast Fourier Transform (FFT) operation, which will convert the signal from time domain to frequency domain.

Channel estimation and equalization:

A channel equalizer is used at the receiver, as received signal is usually distorted by the channel characteristics. To recover the transmitted bits, the channel effect must be estimated and compensated. Due to orthogonality between subcarrier the component of the received signal can be expressed as the product of the transmitted signal and channel frequency response at the subcarrier. Thus, the transmitted signal can be recovered by estimating the channel response just at each subcarrier. In order to choose the channel estimation technique for the OFDM system under consideration, many different aspects of implementations, including the required performance, computational complexity and time-variation of the channel must be considered.

Completing receiver block diagram, converting the signal from P/S, then demodulation the PSK or QAM symbols, and applying decoding of the received bits, for Block decoding it multiplies the received vector r where **r = v + noise** by the syndrome matrix **S** to determine if any errors occurred and determine which (if any) bits were in error out of the n sent. while for convolutional codes in order to overcome the ability of channel noise signal we should define an important parameter which is the minimum hamming distance ($d_{\min}$)between any two distinct codewords, the number of errors that can be corrected by the code is known as correcting capability , in order to correct t errors the minimum distance $d_{\min}$ >2t. one of the most common and efficient decoder is Viterbi decoder. Then after these processes the signal is demodulated and path through bit deinterleave then FEC decoder and at the end of these processes we can retrieve the message bits. And that is a brief description of how OFDM system is implemented. [2].

OFDM Design Parameters

- Guard time TG=4 Trms such that Trms is that average delay spread.
- Cyclic prefix Tcp highest value =$\frac{Tsym}{4}$, the length of cyclic prefix should be greater than the length of the channel -1.
- Frequency separation Δf=$\frac{1}{Tsym}$.
- Maximum number of subcarriers = $\frac{Bw}{\Delta f}$.
- Bits/OFDM symbol=data rate*$T_{sym.}$
- FFT size L=2 ^{m} ≥ Nmax so the extra bits will be zero padded.

Advantages of OFDM

OFDM makes an efficient utilization of the spectrum due overlapping between orthogonal subcarriers (no large guard band exist between subcarriers)

By dividing the channel into narrowband flat fading sub channels, OFDM is more resistant to frequency selective fading than single carrier systems.

It can easily adapt to severe channel conditions without complex time-domain equalization (using simple equalizers). [1]

It reduces ISI and ICI by adding cyclic prefix in the transmitted time domain symbol. Unlike conventional FDM, tuned sub-channel receiver filters are not required due to orthogonality between subcarriers so receiver can differentiate between them easily. [1]

Provides high data rate.

Easy to be implemented because of IFFT and FFT algorithm.

Disadvantages of OFDM

It is sensitive to Doppler shift (frequency synchronization problems).
Requires time synchronization algorithm to avoid ISI
It suffers loss of efficiency caused by cyclic prefix.
And the main disadvantage is the peak to average power ratio (PAPR) which will be discussed in the next section.

PAPR

Introduction

One of the major problems of OFDM is that the peak amplitude of the emitted signal can be considerably higher than the average amplitude, this causes degradation in the efficiency of the power amplifier in the transmitter. High peak to average power causes nonlinear distortion in the transmitted OFDM signal when it is passed through a nonlinear HPA. This nonlinear distortion causes serious in-band distortion as well as adjacent channel interference due to spectrum regrowth in the transmitted signa, also it causes reduction in the efficiency of the OFDM system which means that the receiver will not be able to receive the signal properly (increasing in bit error rates in the receiver). Mainly this problem originates from the fact that an OFDM signal is the superposition of N sinusoidal signals on an independently different subcarriers and each of different phase value, And since all subcarriers are added together in time domain via IFFT this may result in adding all subcarriers constructively at a certain instance causing high peak ,so that the amplitude of the signal will be proportional to N, and the power will be proportional to N^2 so increasing number of subcarriers will result in high PAPR [5] .

so, it is predicted that OFDM systems are known to have a high PAPR (Peak-to-Average Power Ratio), compared with single-carrier systems.

We can also look at this issue from a slightly different point of view: the contributions to the total signal from the different subcarriers can be viewed as random variables (they have quasi-random phases, depending on the sampling time as well as the value of the symbol with which they are modulated). If the number of subcarriers is large, we can invoke the central limit theorem so we can assume that the real and imaginary parts of the time-domain complex OFDM signal x(n) or s(t) (after IFFT at the transmitter) have a Gaussian distributions with zero mean and variance σ^2, as illustrated in figure 21, so the amplitude of the resulting output signal is $|x(n)| = \sqrt{(Re\{x(n)\})^2 + (Im\{x(n)\})^2}$, follows a Rayleigh distribution as shown in figure 22.which again illustrates the PAPR characteristics of the OFDM signal. And it is important to note that the output signal amplitude and hence the power are random, so the PAPR is not a deterministic quantity.

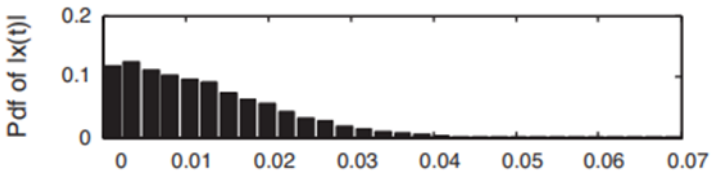


Figure 21 Magnitude distribution of the OFDM signal [2]

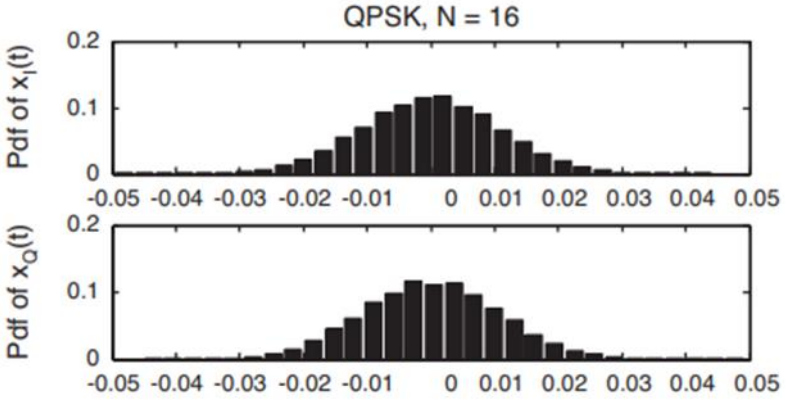


Figure 22 Magnitude distribution of in phase and quadrature components

Definition

Peak to average power ratio is the ratio between the maximum power and the average power of the complex passband signal as shown in equation 5, it is usually represented in dB. [2]. usually, it can be expressed in terms of Complementary Cumulative Distribution Function (CCDF) figure 23 shows our simulation results of PAPR distribution using CCDF. its approximately equals to 11 dB

$$\therefore \text{PAPR}\{x[n]\} = \frac{max_{0\leq n\leq N-1} x[n]^{\,2}}{\text{E}[|x[n]|2} \quad (5)$$

Here, the denominator is the average power of x[n] over 0≤n≤N−1.

The PAPR of OFDM signal is always less than or equal to N, where N is the number of subchannels, for example, for N = 256 the PAPR can be as high as 24 dB ($10log_{10}(256)$).

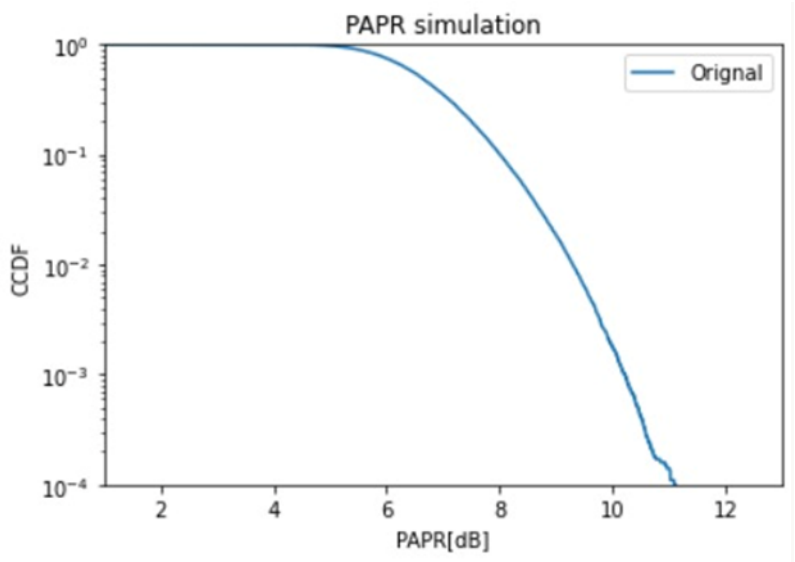


Figure 23 Our simulation for PAPR distribution

The above power characteristics can also be described in terms of their magnitudes (not power) by defining the crest factor (CF) as

CF =$\sqrt{PAPR}$

Note:

During our simulation of the OFDM system we remove the imaginary part of the signal that will be transmitted through the channel by using Hermitian matrix.

Simply Hermitian matrix is a square matrix that its non-diagonal elements are all complex numbers and are conjugate to each other. We observe that it increases the number of subcarriers per symbol by (N*2) +2 as the data will have the following structure, 0→ Data→0→conjugate of the data.

Drawbacks of PAPR

- It decreases SQNR (Signal-to-Quantization Noise Ratio) that reflects the relationship between the maximum nominal signal strength and the quantization error (also known as quantization noise) of ADC (Analog-to-Digital converter) and DAC (Digital-to-Analog Converter).
- The main drawback of PAPR is degrading the efficiency of the high-power amplifier in the transmitter.

Most radio systems employ the **HPA** in the transmitter to obtain sufficient transmission power, Practical power amplifiers are linear only over a finite range of

input amplitudes, so if input becomes much larger than its nominal value, output will be distorted due to saturation characteristics of power amplifiers. Nonlinearity in power amplifier response leads to nonlinear amplification of OFDM signal.
Figure 24 shows the input-output characteristics of high-power amplifier (HPA) in terms of the input power Pin and the output power Pout. Due to the saturation characteristic of the amplifier, the maximum possible output is limited by P_{out}^{max} when the corresponding input power is given by P_{in}^{max}. Therefore, the nonlinear region can be described by IBO (Input Back-Off) or OBO (Output Back-Off) [2]

$$\text{IBO} = 10\,log_{10}\frac{P_{in}^{max}}{P_{in}}, \qquad o\text{BO} = 10\,log_{10}\frac{P_{out}^{max}}{P_{out}} \quad (6)$$

Note that the nonlinear characteristic of HPA (High Power Amplifier), excited by a arge input, causes the out-of-band radiation that affects signals in adjacent bands, and in-band distortions that result in rotation, attenuation, and offset on the received signal. [2]

To prevent saturation and clipping of the OFDM signal peaks, the amplifiers must be operated with sufficient back-OF. However, increasing back-off will reduce the efficiency of the power amplifier. [8]

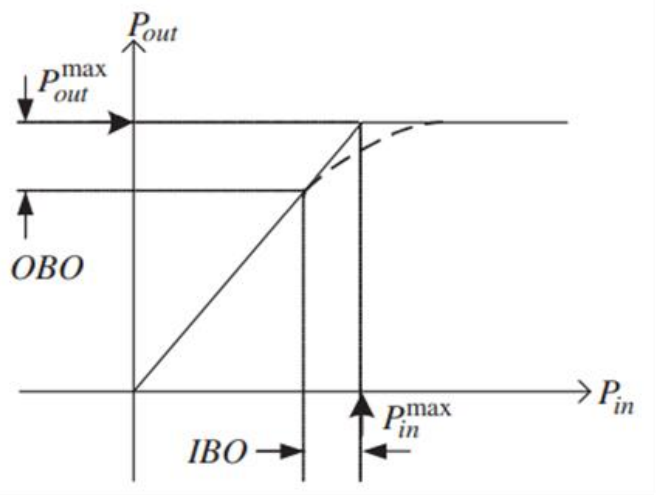


Figure 24 input-output characteristic of an HPA [2]

There are three main methods to deal with the high Peak-to-Average Power Ratio (PAPR):

1. Use a power amplifier with a very high linear rang that can amplify linearly the possible peak value of the transmit signal. But this is not practical as it requires expensive and power-consuming class-A amplifiers.
2. Use a non-linear amplifier and accept the fact that amplifier characteristics will lead to distortions in the output signal. Those nonlinear distortions destroy orthogonality between subcarriers, and lead to increased out-of-band emissions
3. Use PAPR reduction techniques. These will be described in the next section.

Peak-to-Average Ratio Reduction Techniques

An effective PAPR reduction technique should be given the best tradeoff between the capacity of PAPR reduction and transmission power, data rate loss, implementation complexity and Bit-Error-Ratio (BER) performance etc.

They are classified into the different approaches:

- Clipping technique: this approach includes block-scaling technique, clipping and filtering technique, peak windowing technique, peak cancellation technique, Fourier projection technique, and decision-aided reconstruction technique, although this technique reduces the peak signal significantly, but it distorts the transmitted signal by clipping the signal peaks and increase the bit error rate (BER).
- Coding technique is to select such codewords that minimize or reduce the PAPR. It causes no distortion and creates no out-of-band radiation, but it suffers from bandwidth efficiency as the code rate is reduced. It also suffers from complexity to find the best codes and to store large lookup tables for encoding and decoding, especially for a large number of subcarriers.
- Scrambling technique: this approach is to scramble an input data block of the OFDM symbols and transmit one of them with the minimum PAPR so that the probability of incurring high PAPR can be reduced. While it does not suffer from the out-of-band power, the spectral efficiency decreases, and the complexity increases as the number of subcarriers increases, it includes SLM (Selective Mapping), PTS (Partial Transmit Sequence), TR (Tone Reservation), and TI (Tone Injection) techniques.
- Adaptive predistortion technique.
- DFT-spreading technique. Spreading process achieved by dividing N subcarriers into K sub bands and all subcarriers in each sub-band will go through m -point DFT.DFT operation is applied before IFFT. The effect of PAPR reduction depends on the way of assigning the subcarriers to each terminal, there are two ways of assigning subcarriers among users localized allocation where subcarriers are allocated one sub band by one sub band. And interleaved allocation where subcarriers are chosen from different sub band in turn. This technique is particularly useful for mobile terminals in uplink transmission. It is known as the Single Carrier-FDMA (SC-FDMA), which is adopted for uplink transmission in the 3GPP LTE standard [2].

Clipping and filtering

The clipping approach is the simplest PAPR reduction scheme which limits the maximum of transmit signal to a pre-specified level. Generally, clipping is performed at the transmitter. However, the receiver needs to estimate the clipping that has occurred and to compensate the received OFDM symbol accordingly. At most one clipping occurs per OFDM symbol, and thus the receiver has to estimate two parameters: location and size of the clip to be able to recover the transmitted OFDM symbol. However, it is difficult to get these information.so clipping has the following drawbacks:

- Causes in-band signal distortion, resulting in BER performance degradation.
- Also causes out-of-band radiation, which imposes out-of-band interference signals to adjacent channels, Although the out-of-band signals caused by clipping can be reduced by filtering, but it cannot reduce the in-band distortion as it may affect high-frequency components of in-band signal (aliasing) when the clipping is performed with the Nyquist sampling rate in the discrete-time domain.
- The signal after filtering operation may exceed the clipping level specified for the clipping operation, (peak regrowth). [2]

$$x[n] = \begin{cases} x[n], & |x| < A \\ A, & |x| \geq A \end{cases} \quad eq.(7)$$

where A is the pre-specified clipping level

An important parameter in this technique is the clipping ratio (CR)which is the ratio between the clipping level normalized by the RMS value of OFDM signal

$$CR = \frac{A}{\sigma} \quad eq.(8)$$

It has been known that $\sigma = \sqrt{N}$, $\sigma = \sqrt{\frac{N}{2}}$ in the baseband and passband OFDM signals with N subcarriers, respectively. Figure 25 shows the CCDF of PAPR for the clipped and filtered OFDM signals. We notice that the smaller the clipping ratio (CR) is, the greater the PAPR reduction effect. [2]

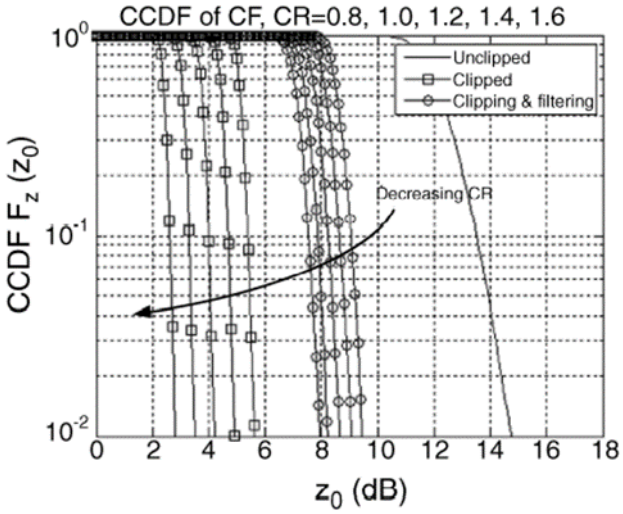


Figure 25 PAPR distribution using clipping tech.

Partial transmit sequence

The partial transmit sequence (PTS) technique partitions an input data block of N symbols into V disjoint subblocks as follows:

$$x = [x^0, x^1, x^2, x^{v-1}]^T \quad eq.(9)$$

where X^i are the subblocks that are consecutively located and are of equal size, Partitioning schemes of subblock could be adjacent, interleaved, and pseudo-random. Among these, the pseudo-random one has been known to provide the best performance. Phase rotation is applied to each subblock then each partition is multiplied by a corresponding complex phase factor $b^v = e^{j\theta v}$, implementation of partial transmit sequence can be shown in figure 26. The phase vector is chosen so that the PAPR can be minimized, choosing the optimum phase shift introduce complexity as the number of subblocks increase. also, PTS technique requires V IFFT operations for each data block and log2$\lfloor W^V \rfloor$ bits of side information. Figure 27 shows PAPR reduction of this technique.

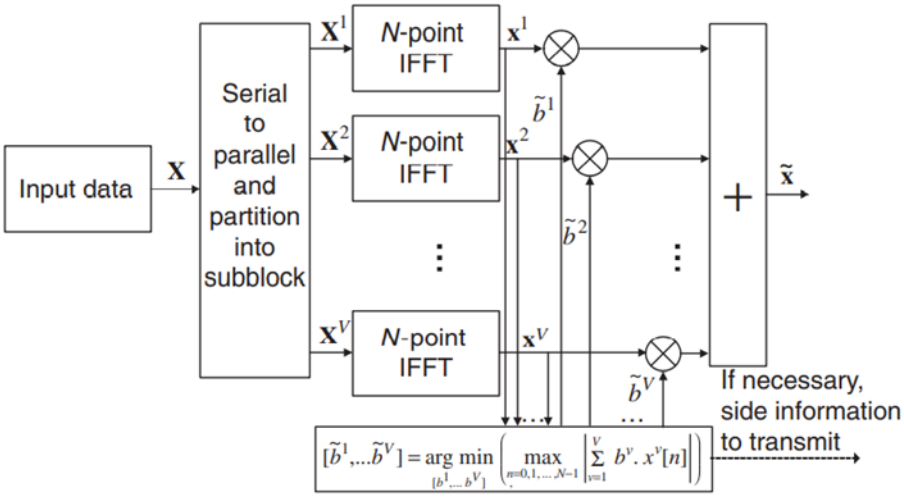


Figure 26 Block diagram of partial transmit sequence (PTS) technique for PAPR reduction

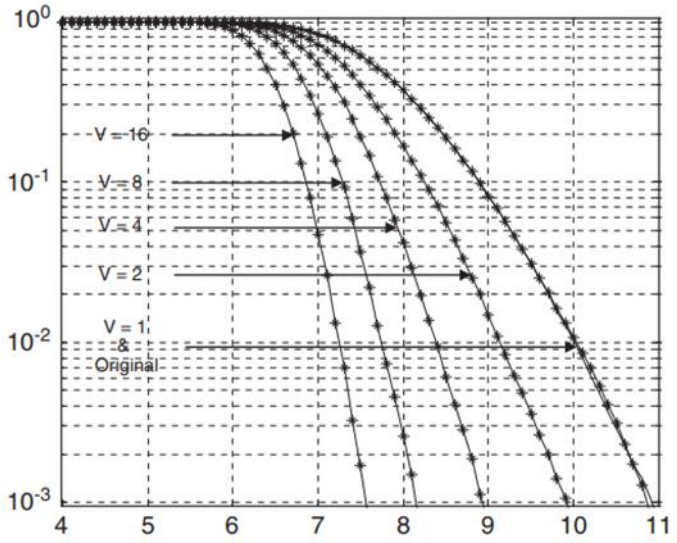


Figure 27 PAPR performance of a 16-QAM/OFDM system with PTS technique when the number of subblocks varies

Tone injection

Tone Injection technique can be used to reduce the PAPR without reducing the data rate. It allows the PRTs to be overlapped with data tones. TI is based on mapping of original data that causes large peaks to several new positions this could be done by increasing the constellation size so that each of the points in the original constellation can be mapped into several equivalent points in the expanded constellation where the extra degrees of freedom can be exploited for PAPR reduction. The receiver must know how to map the redundant positions on the original one. TI is distortion less technique and does not exhibit data rate loss. However, transmitter is more complex

as it requires additional IFFT operation [10]. TI technique also requires more signal power for transmission of signal to transmit the symbols in the expanded constellation.

Selective Mapping

Here, the input data block $X = [x[0], x[1] \dots, x[N-1]]$ is multiplied with U different phase sequences $p^u = [p_0^u, p_1^u, \dots, p_{N-1}^u]^T$ where $p_v^u = e^{\vartheta_v^u}, v = 0,1, \dots, N-1$, and u = $0,1, \dots \dots, U-1$, which produce a modified data blocks $X^u = [x^u[0], x^u[1], \dots \dots, x^u[N-1]]^T$which represent the same data, These are then forwarded into IFFT operation simultaneously. And then the PAPR is calculated for each vector separately. The sequence with the smallest PAPR is selected for final transmission. For the receiver to be able to recover the original data block, the information about the selected phase sequence P^u should be transmitted as a side information figure 28 illustrate the block diagram of SLM [2].

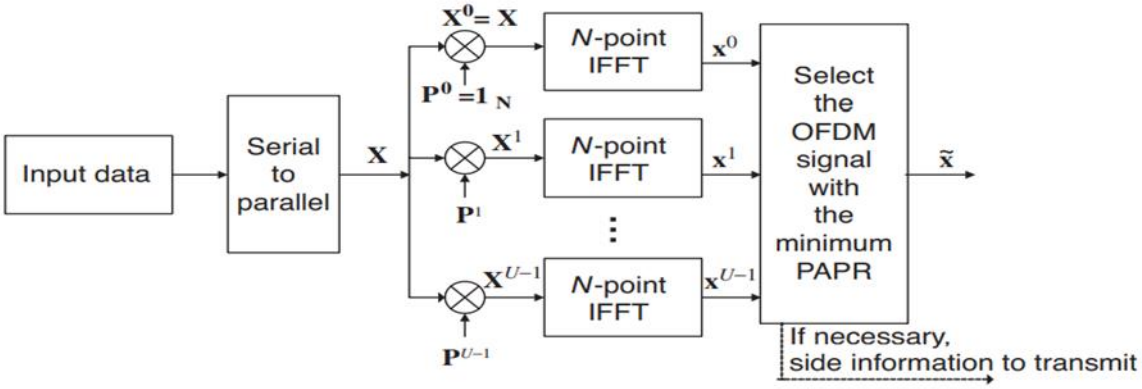


Figure 28 shows the block diagram of selective mapping (SLM) technique for PAPR reduction

Performance Analysis

To justify the performance of the SLM method we compare our results with research results in [9].

we have done some computer simulations considering the following simulation parameters:

- 64 sub carrier.
- U=2, 4, 16, 8, 32, 64. We generate a random orthogonal phase matrix using a built in function called ortho_group imported from scipy.stats. however, in [9]

generates the phase sequence by Hadamard matrix. But both techniques give approximately the same performance.

- 10k symbols
- 4-QAM modulation.

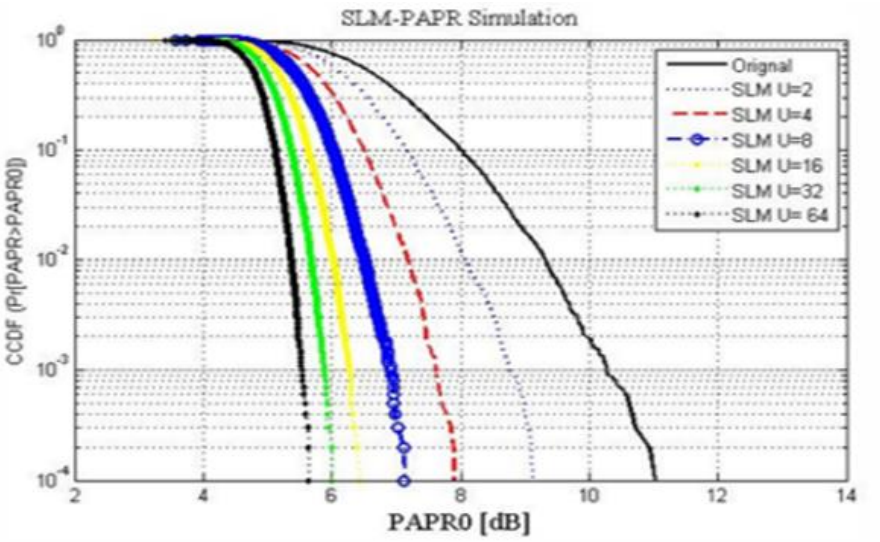


Figure 29 CCDF of the PAPR for the SLM technique for (a) paper results [12]

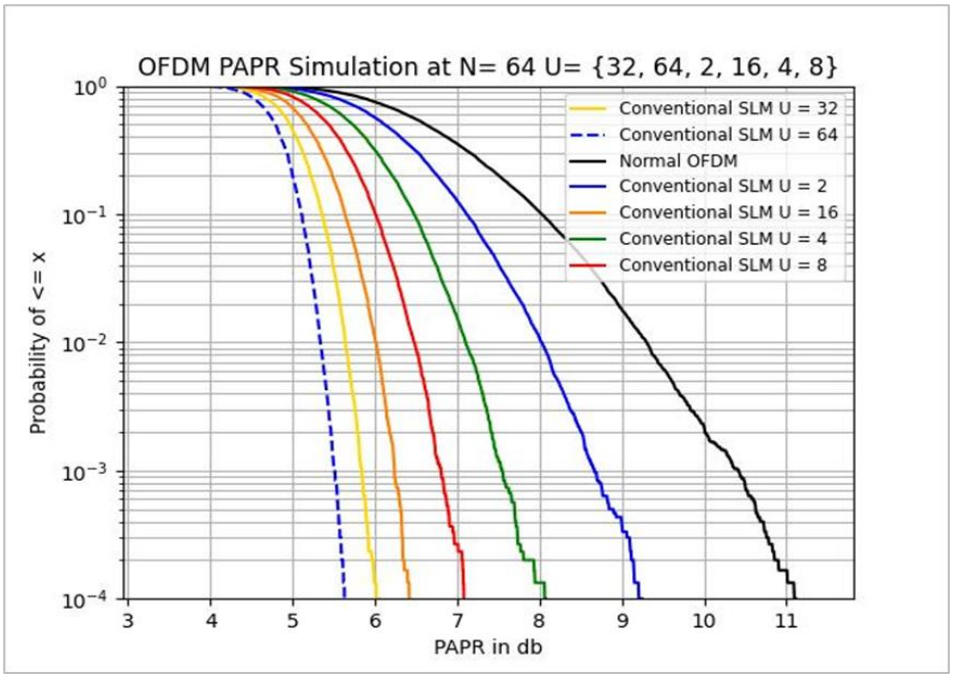


Figure 30 our simulation results at different number of phases

From fihure 30 we notice that as the number of phase sequence multiplied by the signal increases, reduction of PAPR also increases However this will lead to more IFFT operation and increasing the side information that should be sent to the receiver to recover OFDM symbols.

Also if we compare SLM and PTS, SLM is better than PTS with respect to number of data vectors. As number of data vectors or sub blocks increases, complexity in PTS increases substantially.

Tone reservation PAPR reduction technique

In this proposed technique, some OFDM subcarriers are reserved. These reserved subcarriers don't carry any data information, are only used for reducing PAPR, they are called peak reduction tones (PRTs). They are chosen such that OFDM symbol has a low PAPR. Position of these tones are complemented to the positions of data i.e., C[K], K ∈ R and X[K] ∈ R^c [10]. In this technique, additional power is required for transmitting the PRT symbols and the effective data rate decreases since the PRT tones work as an overhead. note that when we add the two signals in time domain no distortion occurs to the OFDM symbol since the subcarriers are orthogonal.

If we compare TR with SLM, we observe that in tone reservation no side information is needed, we just need only one IFFT operation, so it is less complex.

There are different methods for tone reservation technique: kernel based, and SCR-based TR methods have explained below.

Kernel based (TR-K) method

This method creates a reference kernel vector p, which is an impulse function has FFT size, in the kernel vector the reserved tones positions will set to one, and all other values are updated to zero. The kernel will update with a single peak and all the other samples would be zero, but this is impractical because of the limited bandwidth. In every iteration i the maximum amplitude A^i of OFDM signal x^i and its position m^i must be found. The position of the maximal amplitude of the kernel vector is circularly shifting to the position m^i, the adding of kernel vector to the original input reduces the peak to a previously determined wanted clipping level. The following equations will explain the method:

$$x^{i+1} = x^i - \alpha^i p(m^i) \quad (1)$$

$$\alpha^i = \frac{x(m^i)}{A^i}\left(A^i - A_{max}\right) \quad (2)$$

where A_{max} is the clipping amplitude and $p(m^i)$ is the circularly shifted time domain kernel, then the PAPR is calculated [[10] , [11]]. If the number of iterations reaches

predetermined maximum iteration number, control escapes the process and resulting signal is transmitted. If not, clipping operation is executed iteratively [10].

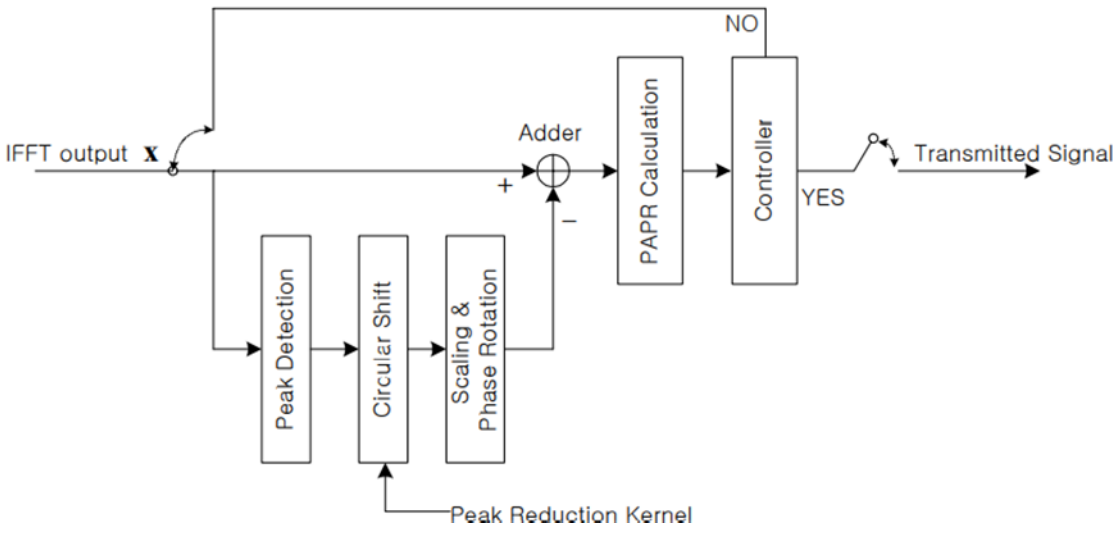


Figure 31 Kernel Algorithm [2]

Performance Analysis

To justify the performance of the TR-k method we compare our results with research results in [11] as shown in figure 32 and 33.

we have done some computer simulations considering the following simulation parameters:

- 1024 sub carrier.
- 32 PRTs.
- 10k symbols.
- Random set optimization for PRTs.
- Iterations = 20.
- QPSK nodulation

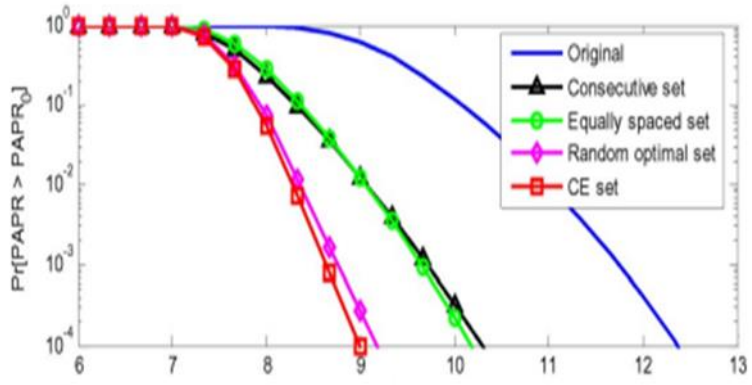


Figure 32 CCDF of the PAPR for the TR schemes with random PRT sets for paper results [12]

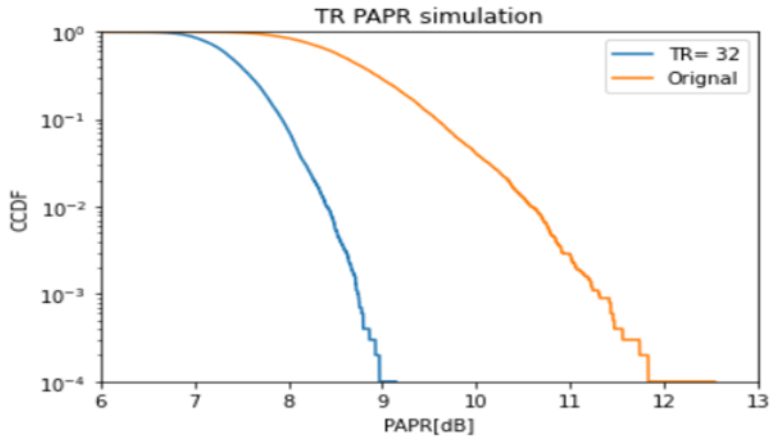


Figure 33 our simulation results

Note: Paper results are smoother as the use of 1M symbols in paper simulation, and we only used 10K symbols in our simulation.

Scaling signal to clipping noise ratio based (TR-S-SCR) method

In the SCR scheme, the peak reduced signal $\boldsymbol{x(n)}$ is iteratively updated by using a simple gradient algorithm given by:

$$x^{m+1}(n) = x^m(n) - \mu . c^m_{max}(n) \qquad (3)$$

$$c^m_{max}(n) = \left(x^m(n^m_{max}) - A.e^{j\theta_{max}}\right)p(n - n^m_{max}) \qquad (4)$$

Where $\mu . c^m_{max}(n)$ is the peak reduction signal, μ is a constant scaling factor, $A = \sqrt{\frac{1}{LN}\sum_{n=0}^{LN-1}|x^m(n)|^2}$ is the required clipping threshold, CR is the clipping ratio, $p(n - n^m_{max})$ denotes the circularly shifted sequence of time domain kernel to max n^m_{max} which is the position of the peak amplitude of $x^m(n)$ in the m-th iteration [9].

To improve the convergence rate, we present an S-SCR scheme which employs the gradient algorithm with additional peak regeneration restraint to obtain the optimized scaling factor. First, given the clipping threshold A, the directly clipped signal can be expressed as:

$$\dot{x}^m(n) = \begin{cases} x^m(n) & |x^m(n)| < A \\ A.e^{j\theta_n} & |x^m(n)| > A \end{cases} \quad (5)$$

then the clipping noise is defined:

$$f^m(n) = x^m(n) - \dot{x}^m(n) \quad (6)$$

Then, we express the position of the peaks which satisfy $|f^m(n)| > 0$ with U entries as $S = \{S_0 S_1, \dots, S_{U-1}\}$.

By using the LSA algorithm, the optimal solution of μ can be calculated as:

$$\mu = \frac{\sum_{n\in S}|x^m(n)||f^m(n)|}{\sum_{n\in S}|x^m(n)|^2} \quad (7)$$

Furthermore, for the purpose of avoiding an undesirable peak regeneration, we design a 1×LN scaling vector Ψ that restricts the scaling adjusting only on the peak value in one iteration. Thus, we get:

$$\Psi(i) = \begin{cases} \mu & i = n_{max} \\ 1 & i \neq n_{max} \end{cases} \quad (8)$$

Accordingly, we can rewrite the peak reduced signal $x(n)$ with optimal solution of the scaling factor as:

$$x^{m+1}(n) = x^m(n) - \Psi\left(x^m(n^m_{max}) - A.e^{j\theta_{max}}\right)p(n - n^m_{max}) \text{ [12]}$$

Performance Analysis

To justify the performance of the S-SCR method we compare our results with research results in [12] as shown in figures 34 and 35.

We have done some computer simulations considering the following simulation parameters:

- 256 sub carrier.
- T/N=1/8.
- 10k symbols.
- Random set optimization for PRTs.
- Iterations = 18.
- 16-QAM modulation.

- CR= 2dB.
- Oversampling factor (W)=4.

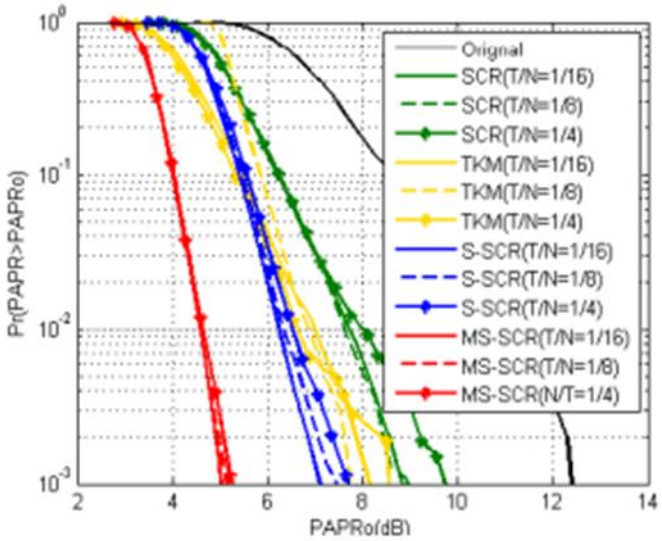


Figure 34 CCDF of the PAPR for the SCR technique for paper results [6]

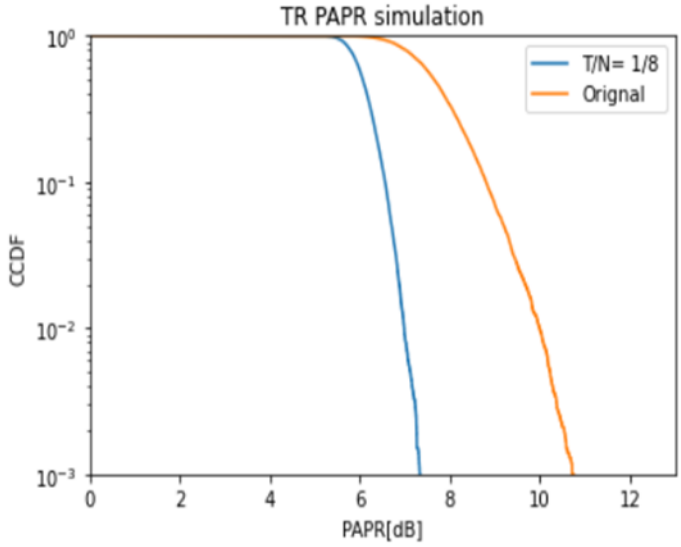


Figure 35 our simulation results

CHAPTER II - Literature Review on Machine Learning

Introduction to ML

Machine Learning is about extracting knowledge from data and thus making predictions upon the collected information. It is a research field at the intersection of statistics, artificial intelligence and computer science, it's also known as predictive analysis or statistical learning. Applications of Machine Learning has become ubiquitous in everyday life. From automatic recommendations of which films to watch, to what food to order or which products to buy, to recognizing your friends in your photos. Many modern websites and applications have machine learning algorithms at their core like Facebook, Amazon, Netflix, it is very likely that every part of the site contains multiple machine learning models.

Why Machine Learning?

In the early days of 'intelligent' applications, many systems used hand coded rules of 'if' and 'else' decisions to adjust to user input, however using hand coded rules to make decisions has two major drawbacks:

- The logic required to make a decision is specific to a single task and changing the task might require a rewrite to the whole system.
- Designing rules requires a deep understanding of how a decision should be made by the user.

Also hand coded approach failed to detect faces in images, on the other hand Machine Learning can use a large collection of images and by designing an algorithm, it can know the characteristics needed to identify a face.

Machine Learning Applications

The most successful machine learning algorithms are those that automate decision making processes by generalizing from known examples which is known as Supervised Learning, the user provides an algorithm with pairs of inputs and desired outputs, the algorithm finds a way to produce the desired output given an input, the algorithm can produce an output to an input it has never seen before without the help of human.

Examples of supervised machine learning tasks:

- Determine whether a tumor is benign based on medical images

The input here is the image of the tumor, and the output is whether it's benign or not, you need a dataset of medical images and an expert opinion.

- Estimating the channel network in MIMO network

  It's important to estimate the noise because it's essential in many tasks of wireless network such as network management and event detection, assume the channel model is z = Hs + u, where s is the transmitted signal, z is the received signal and u is the noise in the channel. So, the goal is to estimate u given s and z [13].

- Determine the Zip code from handwritten digits on an envelope

  Here the input is a scan of the handwriting, and the desired output is the actual digits in the zip code. To create a dataset for building a machine learning model, you need to collect many envelopes. Then you can read the zip codes yourself and store the digits as your desired outcomes.

Another type of machine learning algorithms is the unsupervised learning, in which there is only input, but no output is given to the algorithm. While there are many successful applications of unsupervised learning, but it's still hard to understand and evaluate.

Examples of unsupervised learning:

- Segmenting customers into groups with similar preferences

  Given a set of customer records, you may want to identify which customers are similar, for a shopping site this might be 'parents', 'gamers', 'bookworms'…. etc. Because you don't know in advance what these group of people are, there are no known outputs.

- News selection

  Google News uses unsupervised learning to categorize articles on the same story from various online news outlets.

- Identifying topics in a set of blog posts

For a large collection of text data, you might want to summarize it and find relevant themes in it. You might not know what these topics are, or how many topics there might be, so there are no known outputs.

Why Python?

Python has become the lingua franca for many data science applications. It combines the power of programming languages with the ease of use of domain-specific scripting languages like MATLAB or R. Python has libraries for data loading, visualization, statistics, natural language processing, image processing and more.

Machine learning and data analysis are fundamentally iterative processes, in which the data drives the analysis. It is essential for these processes to have tools that allow quick iteration and easy interaction. As a general-purpose programming language, Python also allows for the creation of complex graphical user interfaces (GUIs) and web services, and for integration into existing systems.

Essential Libraries and Tools

Jupyter Notebook

The Jupyter Notebook is an Interactive environment for running code in the browser. It is a great tool for exploratory data analysis and used by many data scientists. The Notebook makes it easier to incorporate code, text, and images.

Google Colab

Colab is basically a free Jupyter notebook environment running wholly in the cloud. Most importantly, Colab does not require a setup, plus the notebooks that you will create can be simultaneously edited by your team members – in a similar manner you edit documents in Google Docs. The greatest advantage is that Colab supports most popular machine learning libraries which can be easily loaded in your notebook.

NumPy

NumPy is one of the fundamental packages for scientific computing in Python. It contains functionality for multidimensional arrays, high-level mathematical functions such as linear algebra operations and the Fourier transform, and pseudorandom number generators. We will be using NumPy a lot in this book, refer to its documentation.

SciPy

SciPy is a collection of functions for scientific computing in Python. It provides, among other functionality, advanced linear algebra routines, mathematical function optimization, signal processing, special mathematical functions, and statistical distributions. The most important part of SciPy for us is sparse this provides sparse matrices, which are used whenever we want to store a 2D array that contains mostly zeroes, refer to its documentation.

Matplotlib

matplotlib is the primary scientific plotting library in Python. It provides functions for making publication-quality visualizations such as line charts, histograms, scatter plots, and so on. Visualizing your data and different aspects of your analysis can give you important insights, and we will be using matplotlib for all our visualizations, refer to its documentation.

Pandas

pandas is a Python library for data wrangling and analysis. a panda Data Frame is a table, similar to an Excel spreadsheet, pandas provides a great range of methods to modify and operate on this table. In contrast to NumPy, which requires that all entries in an array be of the same type, pandas allows each column to have a separate type (for example, integers, dates, floating-point numbers, and strings). Another valuable tool provided by pandas is its ability to ingest from a great variety of file formats and data- bases, like SQL, Excel files, and comma-separated values (CSV) files.

TensorFlow

Developed by Google, TensorFlow is an open-source end to end platform for creating Machine Learning applications. It is a symbolic Math library that uses dataflow and differentiable programming to perform various tasks focused on training and inference of deep neural networks. Google uses machine learning in all its products to improve search engine, translations and recommendations. It is called TensorFlow because it takes input as multidimensional array called Tensors, then construct a sort of flowchart of operations called a **Graph** that you want to perform on the input, the input goes at one end, and then flows through this system of multiple operations and comes out the other end as output. TensorFlow is the best library of all because it is built to be accessible for everyone. TensorFlow library incorporates different API to build at scale deep learning architecture like CNN or RNN. TensorFlow is based on graph computation; it allows the developer to visualize the construction of the neural network with TensorBoad. This tool is helpful to debug the program. Finally, TensorFlow is built to be deployed at scale. It runs on CPU and GPU.For more information refer to its documentation.

SUPERVISED LEARNING

Supervised Machine Learning is one of the most used and successful types of machine learning, it is used whenever we want to predict a certain outcome from a given input, providing input/output pairs we build a model from these pairs which comprise our training set, the output here is considered the **Labels** for the input. Our goal is to make accurate predictions for new never seen data. Supervised Learning always require human effort to build datasets.

There are two major types of Supervised Learning problems Classification and Regression.

Classification and Regression

In Classification the goal is to predict a class label, which is a choice from a predefined list of probabilities. Classification is sometimes separated into binary classification, which is classification between two classes and multiclass classification, which is classification between more than two classes, for example predicting a website language from the text and the classes here would be a pre-defined list of possible languages. In binary classification we try to answer a question with yes/no, for example classifying emails into spam or not spam.

In Regression the goal is to predict a continuous number, for example predicting a person's income given his/her age, location and education, the income here is a continuous number that can be in any range. Regression analysis tries to find the value of the parameters for the function that best fits an input dataset.

An easy way to distinguish between classification and regression to ask whether there is continuity in the output, for example if the predicted income is $5001 or $4999 while the true output is $4000, we don't care about that difference. But in classification, if we want to recognize the language of a website there is no continuity between the languages, as there is no language between French and German.

Generalization, overfitting and underfitting

In Supervised learning, we want to build the model on the training data and then be able to make accurate predictions on new unseen data that has the same characteristics

as the training data. If the model is able to make accurate predictions on unseen data, we say the model is able to **generalize** from the training set to the test set.

Building a model that is too complex for the amount of the information we have is called **overfitting**. Overfitting occurs when you fit a model too closely to the particularities of the training set and obtain a model that works well on the training set but unable to generalize to new data. On the other hand building too simple model will result in **underfitting.**

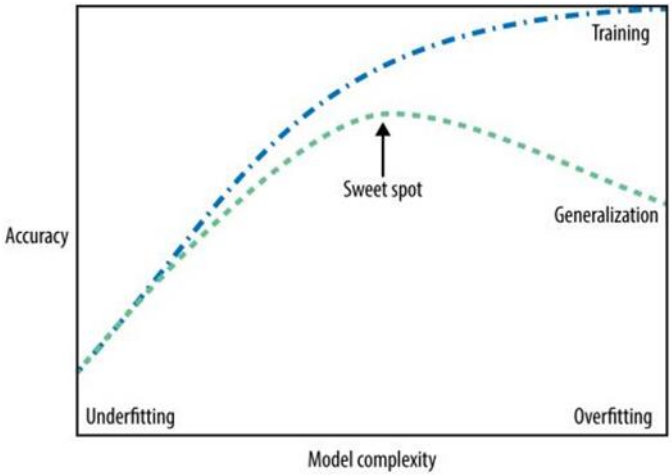


Figure 36 complexity vs accuracy

There is a sweet spot in between that will yield to the best Generalization.

Supervised Machine Learning Algorithms

We will discuss the main algorithms shortly, as the main focus is on Neural Networks, as it is the algorithm that we used in our project.

Linear and Logistic Regression

Linear models make a prediction using a *linear function*, In Linear regression, the goal is to minimize a cost function by finding appropriate parameters for the function over the input data that best approximates the target values. A cost function is the function of the error which describes how far the predicted value from the true value. A popular cost function is the Mean Squared Error **(MSE)** where we take the difference between the square of the true and predicted values, the sum over all the input examples gives us the error of the algorithm. There are many other cost functions that we will discuss later. In Linear Regression the output is the multiplication of the input and weight vector, then we iterate over the training data to compute MSE then

minimize it using the Gradient Descent to update the weights, $\hat{y} = w[0] * x[0] + w[1] * x[1] + ... + w[p] * x[p] + b$, where b is a bias.

Note that: Neural Networks and Linear/Logistic Regression have a lot in common.

Logistic Regression is similar to Linear Regression except that the output of the linear regression is passed to a logistic function that limits its value to [0,1].

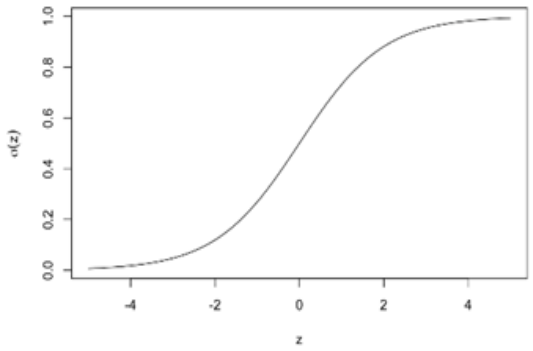


Figure 37 sigmoid function

Support Vector Machines

SVM is a popular supervised machine learning algorithm that is used mainly for classification, it is the most popular member of the kernel method class of algorithms. SVM tries to find a hyperplane which separates the sample in the dataset. The limitation in the linear model is that it can only separate the sample in the dataset by a line, which is not accurate for linearly inseparable classes. But the kernel trick uses a kernel function that transforms the data by adding more dimensions to it, so if the data classes is linearly inseparable in two dimensions, it will be separable in three dimensions. We expand the set of input features by adding another feature in terms of the input feature, say that we have two features (feature0 , feature1) the third feature can be feature0^2 , now each data point can be represented as (feature0 , feature1 , feature0^2).

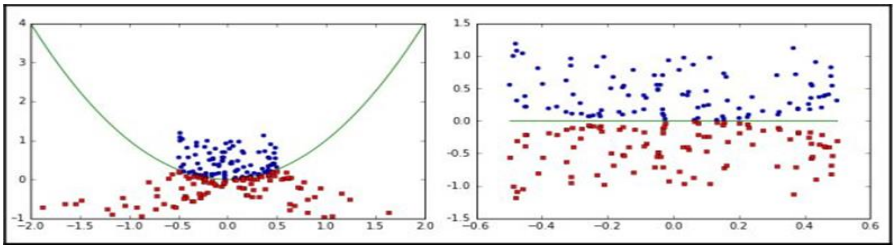


Figure 38 Before applying Kernel Vs After Applying it

Decision Trees

Another popular supervised algorithm is the Decision Trees, in which a classifier is created in the form of a tree, decision nodes and decision leaves.

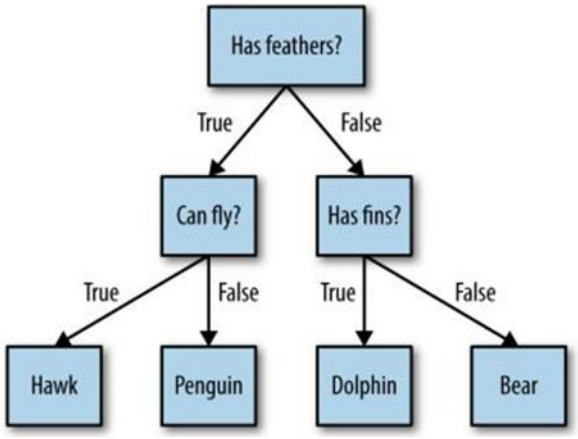


Figure 39 simple animal classifier based on decision trees

Naive Bayes

Naive Bayes classifiers are similar to Linear models, but they tend to be faster in training but slightly worse generalization performance, the reason behind this efficiency is that those classifiers learn parameters by looking at each feature individually and collect per class statistics from each feature.

Unsupervised Learning

The second family of Machine Learning algorithms is the unsupervised learning, here we don't label the data, no known output, no teacher to instruct the learning algorithm. The learning algorithm is just shown the data and asked to extract knowledge from this data.

Types of Unsupervised Learning

Clustering

The most common and the simplest example of unsupervised learning is the clustering, this is a technique that attempts to separate the data into subsets based on

the similarities between the samples. Different clustering algorithms use different metrics to measure similarity.

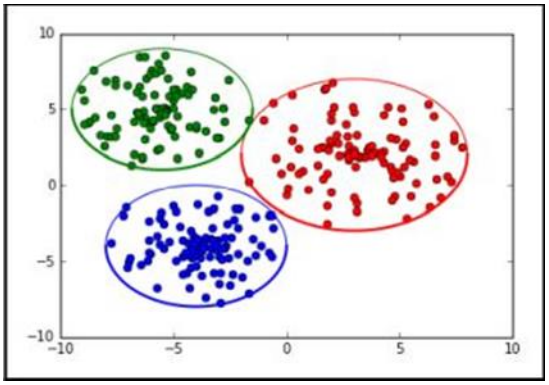


Figure 40 clustering

Recurrent neural networks

It is a type of artificial neural networks that uses sequential data or time series data. These algorithms are used for ordinal or temporal problems such as language translation, natural language processing, speech recognition and image captioning. We will discuss this type of networks later in the Neural Networks chapter.

AutoEncoders

Another interesting application of unsupervised learning is generative models, we will train a generative model with a large amount of data of a certain domain such as images or text and the model will try to generate new data like the one used in training.

Neural Networks

Neural Networks have been around for many years, and they have gone through several periods in which they've fallen out of favor, but recently they gained too much hype over the classical Machine Learning algorithms, thanks to the fast computers and fast Graphical Processing Units **GPU** that outperforms the tradition **CPU**, and due to better algorithms and neural net design. So, the main reasons behind so much hype built on neural networks are **Big Data, Algorithms (Back Propagation) and Computational Power.** The more data you feed to the network the better accuracy and performance you will get. The amount of Computational power doubles every

two years, with so many companies producing much powerful hardware, so it became easier to process that big amount of data. Nvidia is the market leader when it comes to GPUs, also Google is releasing TPUs (ASICs for deep learning), and by that it is easier to train deep neural networks.

The Need for Neural Networks

Neural Networks have been around for many years, and they have gone through several periods in which they've fallen out of favor, but recently they gained too much hype over the classical Machine Learning algorithms, thanks to the fast computers and fast Graphical Processing Units **GPU** that outperforms the tradition **CPU**, and due to better algorithms and neural net design. So, the main reasons behind so much hype built on neural networks are **Big Data, Algorithms (Back Propagation) and Computational Power.** The more data you feed to the network the better accuracy and performance you will get. The amount of Computational power doubles every two years, with so many companies producing much powerful hardware, so it became easier to process that big amount of data. Nvidia is the market leader when it comes to GPUs, also Google is releasing TPUs (ASICs for deep learning), and by that it is easier to train deep neural networks.

Figure 41 amount of data vs performance

The main advantage of Neural Networks is that they nearly outperform every traditional machine learning algorithm, but they also have their disadvantages. You mainly use neural networks when you have big amount of data, and the accuracy is your main goal.

The following are some of neural network disadvantages:

- Hard to interpret most of the times.

Most of the times a neural network will give you good results if you are using it with the right problem, but if it doesn't perform well, you will have a lot of trouble finding out why it didn't go as expected, especially a deep neural network, which is the case with us.

- They require too much data

  Most of the times you will find yourself struggling with the network if you have little data, for example a Naïve Bayes model will give better results for the little data case.

- They take time to be developed.

  It's highly easy to develop a prototype using **Keras**, but **Keras** doesn't allow the freedom of customization, that's where **TensorFlow** comes to provide you with much better options to customize stuff as per your needs.

- They take too much time in the training phase

  Deep Neural Network always take more time to train than traditional machine learning algorithms depending on the computational power we have and how deep the network is also how big the data is. So, it's recommended to choose the algorithm wisely.

Introduction to Neural Networks

We can describe a neural network as a mathematical model for information processing. A neural net is not a fixed program, but rather a model, a system that processes information, or inputs. A more general description of a neural network would be as a computational graph of mathematical operations [14].

The operation of a neural network in a simple form is as follows:

- Information processing occurs over elements called **Neurons**.
- Neurons are connected together, and they exchange information through information link.
- Connection links between neurons may be strong or weak and this determine how information is processed.
- Each Neuron has an internal state that is determined by all the incoming connections from other neurons.
- Each neuron has an **Activation Function** that is calculated on its state and determines its output signal, we will talk about activation functions later.

The two main characteristics of a neural net is identified as:

- **The neural net architecture:** This describes how the neurons are connected together whether it's feedforward, Recurrent, multi or single layered, also the number of layers and number of neurons in each layer.
- **The learning:** This describes what is commonly defined as the **Training**, the most common way to train a neural network is with gradient descent and backpropagation.

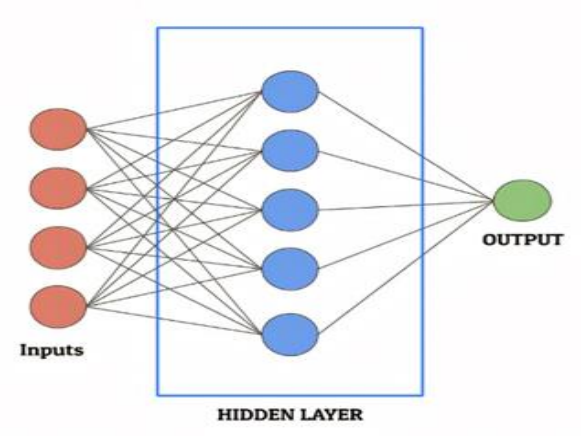


Figure 42 simple neural network

Introduction to Neurons

A neuron is a mathematical function that takes one or more input values, and outputs a single numerical value [14].

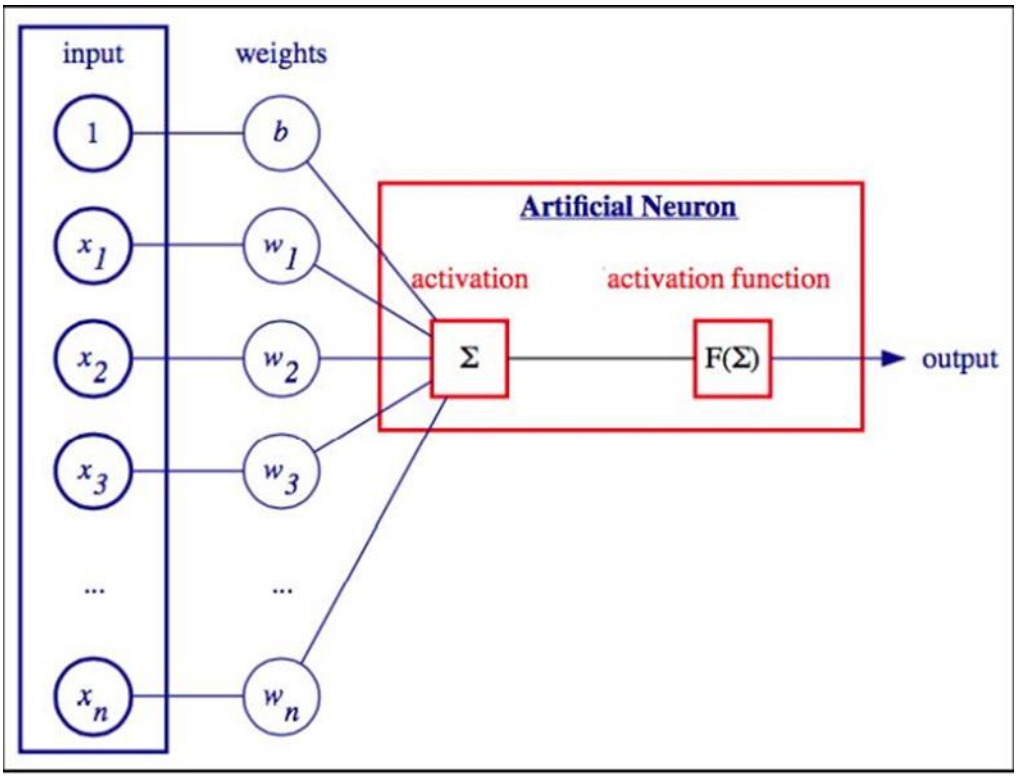


Figure 43 different elements of neurons

The neuron is defined as follows:

$$y = f(\sum_i x_i w_i + b)$$

1. First, we compute the weighted sum of inputs and weights $\sum x_i w_i$ also known as activation value. Here, x_i are either numerical inputs or the outputs of other neurons, if the neuron is a part of a neural network. The weights w_i are numerical values that represent the strength of inputs or strength of connection between neurons. b is the bias value which is always one.
2. Then we use the result of the weighted sum and input it to an activation function, which is also known as transfer function. There are many types of activation functions, but they all have to satisfy the requirement to be non-linear.

The activation value defined previously can be interpreted as the dot product between the vector w and the vector x, they both are perpendicular if their dot product equals to zero. $x \,.\, w = 0$ defines a hyperplane in the feature space R^n, in which n is the dimension of x. For example, if we have a single input value and the activation function is $f(x) = x$, the output of the neuron becomes $y = wx + b$, which is a linear equation. This shows that the neuron defines a line. If we visualize the same for

two or more inputs, we will see that the neuron defines a plane or a hyperplane, for an arbitrary number of input dimensions.

The bias b represents shift between the hyperplane and the center of coordinate systems.

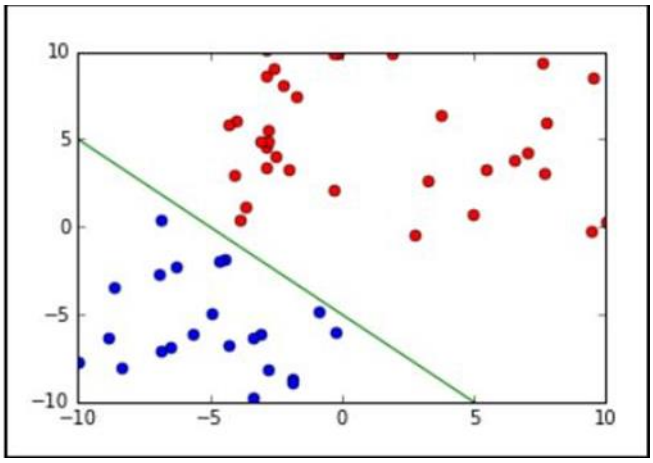


Figure 44 1D hyperplane

Note that the neuron works only with linearly separable classes. To overcome this limitation, we will need to organize the neurons in a neural network.

Introduction to Layers

A neural network can have an indefinite number of neurons, which are organized in interconnected layers. The input layer represents the dataset and the initial conditions. For example, if the input is a grayscale image, the output of each neuron in the input layer is the intensity of one pixel of the image. For this very reason, we don't generally count the input layer as a part of the other layers. When we say 1-layer net, we mean that it is a simple network with just a single layer, the output, in addition to the input layer [14].

the output layer can have more than one neuron. This is especially useful in classification, where each output neuron represents one class. For example, in the case of the Modified National Institute of Standards and Technology (MNIST) dataset, we'll have 10 output neurons, where each neuron corresponds to a digit from 0-9. We will talk about a full classifier example later.

In the following diagram, you can see the 1-layer feedforward network. In this case, we explicitly show the weights w for each connection between the neurons, but usually, the edges connecting neurons represent the weights implicitly. Weight w_{ij} connects the i-th input neuron with the j-th output neuron. The first input, 1, is the bias unit, and the weight, b1, is the bias weight:

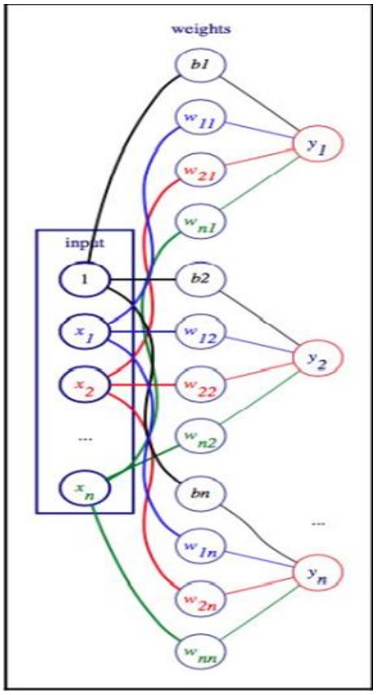


Figure 45 1- layer feedforward network

In the preceding diagram, we see the 1-layer neural network wherein the neurons on the left represent the input with bias b, the middle column represents the weights for each connection, and the neurons on the right represent the output given the weights w.

The neurons of one-layer can be connected to the neurons of other layers, but not to other neurons of the same layer. In this case, the input neurons are connected only to the output neurons. But why do we need to organize the neurons in layers in the first place? One argument is that the neuron can convey limited information (just one value). But when we combine the neurons in layers, their outputs compose a vector and, instead of single activation, we can now consider the vector in its entirety. In this way, we can convey a lot more information, not only because the vector has multiple values, but also because the relative ratios between them carry additional information.

Multi-layer Neural Networks

1-layer neural nets can only classify linearly separable classes. But there is nothing that prevents us from introducing more layers between the input and the output. These extra layers are called hidden layers. The following diagram demonstrates a 3-layer fully connected neural network with two hidden layers. The input layer has k input neurons, the first hidden layer has n hidden neurons, and the second hidden layer has

m hidden neurons. The output, in this example, is the two classes y1 and y2. On top is the always-on bias neuron. A unit from one-layer is connected to all units from the previous and following layers (hence fully connected). Each connection has its own weight, w, that is not depicted for reasons of simplicity.

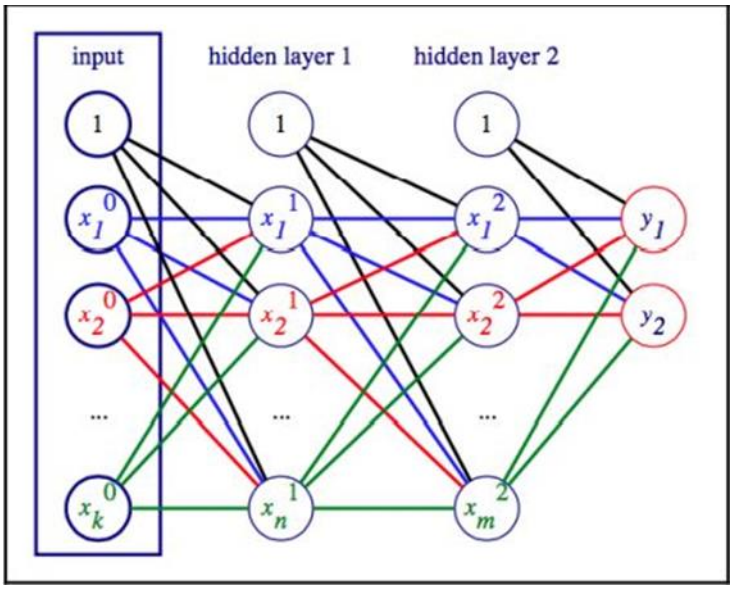


Figure 46 multi-layer network

Activation Functions

Single-Layer networks can classify linearly dependent classes while multi-Layer networks can classify linearly inseparable classes but with one more condition. If the neurons don't have activation functions their output will be the weighted sum of the input with the weights $\sum_i w_i \ . x_i$ which is a linear function. Then the entire network which is a composition of neurons will be a composition of linear functions, which is also a linear function. This means that even if we add more layers the network will be equivalent to a simple Regression model with all its limitations. So, we must add nonlinearity to the network in the form of activation functions. Usually, all neurons in the same layer have same activation function but different layers may have different activation functions.

Activation functions must be efficient, and it should reduce the computation time because neural networks sometimes are trained on millions of data. Activation functions also have a major effect on the neural network ability to converge and the convergence speed, or in some cases, activation functions might prevent neural networks from converging in the first place. Activation function also helps to normalize the output of any input in the range between 1 to -1 or 0 to 1.

The activation function is an important part of an artificial neural network. They decide whether a neuron should be activated or not and it is a non-linear transformation that can be done on the input before sending it to the next layer of neurons or finalizing the output.

Properties of Activation Functions:

- Non-Linear
- Continuously differentiable
- Monotonic

  A monotonic function is a function that decreases or increases entirely on a certain interval, in other words its derivative doesn't change its sign.
- Approximates identity near the origin

  The weight vector w and bias b are initialized with values close to zero by the gradient descend method. Consequently, wx T + b will be close to zero. If G approximates the identity function near zero, its gradient will be approximately equal to its input. In other words, δ G ≈ wx T + b ⟸⟹ wx T + b ≈ 0. In terms of the gradient descend, it is a strong gradient which helps the training algorithm to converge faster.

Activation functions are split into 3 main types:

- Binary step function

  A binary step function is generally used in the Perceptron linear classifier. It thresholds the input values to 1 and 0, if they are greater or less than zero, respectively. It works well with binary classification problems but can't help in the multi class problems. $f(x) = \begin{cases} 0, x < 0 \\ 1, x > 0 \end{cases}$

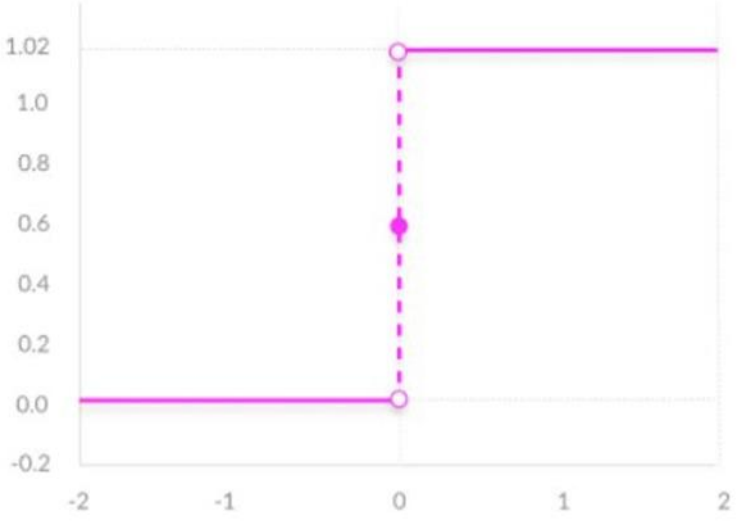


Figure 47 binary step function

- Linear activation function

  $f(x) = ax$,

  The main issue with this function is that the gradient is constant, hence we can't use the gradient descent and backpropagation can't be applied.

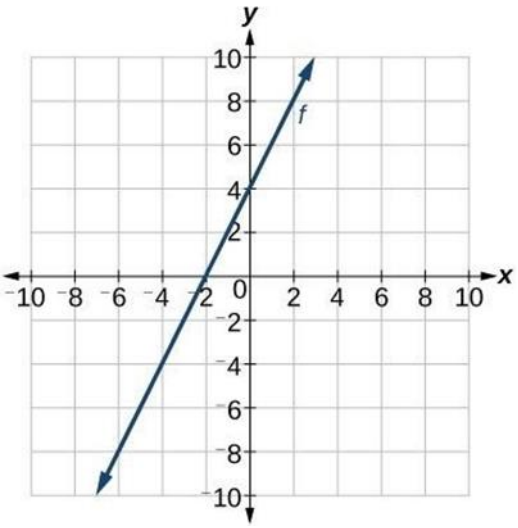


Figure 48 linear function

Non-Linear activation functions

Sigmoid

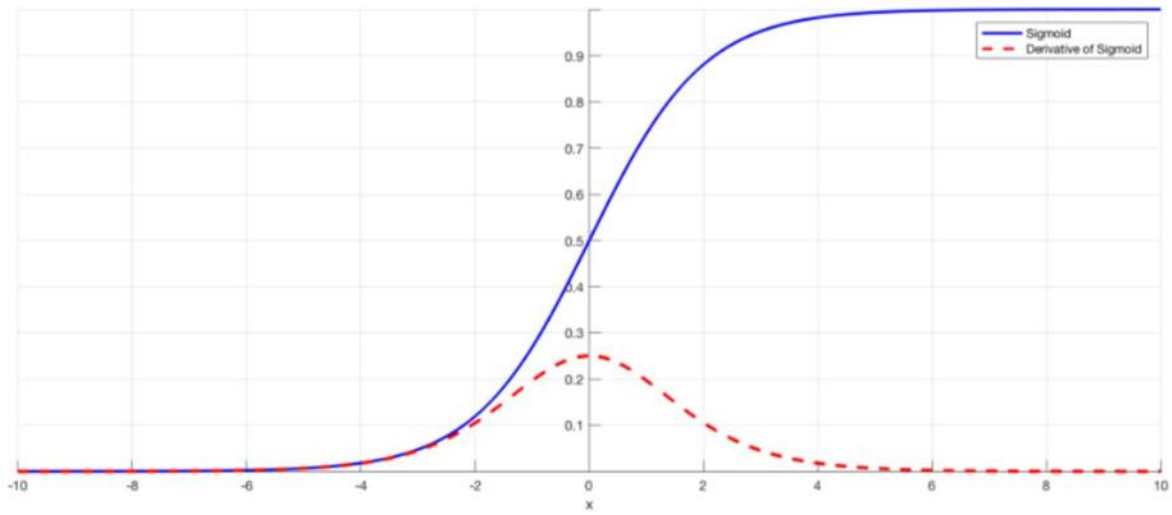


Figure 49 sigmoid in blue and its gradient in red

This function is one of the most commonly used, but it suffers from a major problem which is the **Vanishing Gradient** problem. The gradient diminishes dramatically as it is propagated backward through the network. The error may be so small by the time it reaches layers close to the input of the model that it may have very little effect.
A small gradient means that the weights and biases of the initial layers will not be updated effectively with each training session. Since these initial layers are often crucial to recognizing the core elements of the input data, it can lead to overall inaccuracy of the whole network.
Another major problem in the sigmoid activation functions is the **Exploding Gradient** problem.
Exploding gradients are a problem where large error gradients accumulate and result in very large updates to neural network model weights during training. These large updates in turn results in an unstable network. At an extreme, the values of weights can become so large as to overflow and result in NaN values.

Hyperbolic Tangent

The function produces outputs in scale of [-1, 1] and it is a continuous function. In other words, function produces output for every x value.

$$y = \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} ,$$

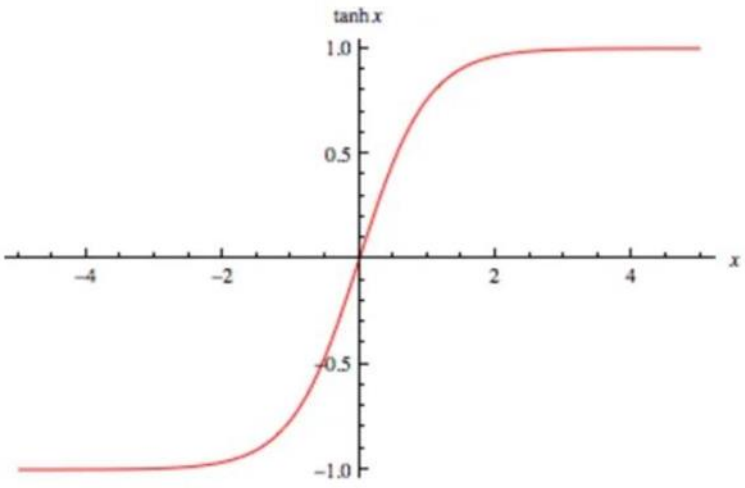


Figure 50 tanh function

Inverse hyperbolic tangent

It's similar to tanh but output is in the range $\left[\frac{-\pi}{2}, \frac{\pi}{2}\right]$

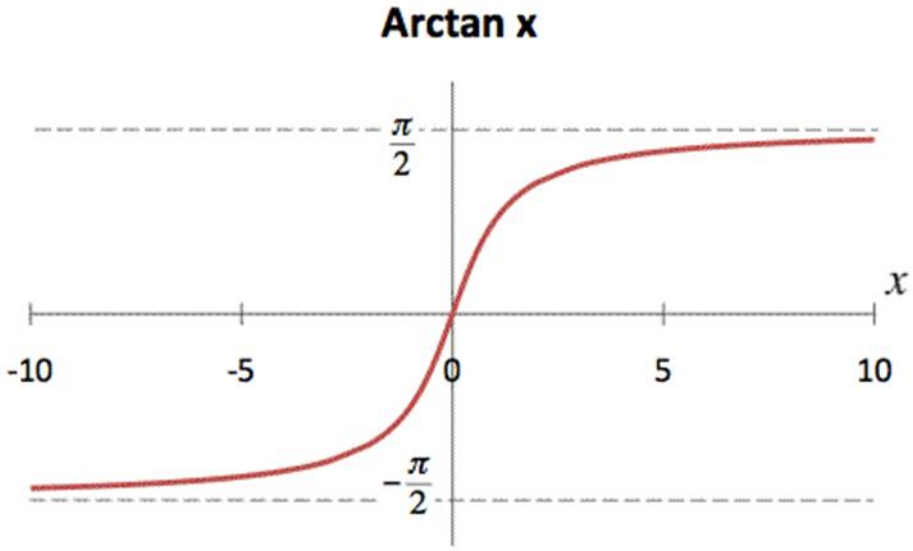


Figure 51 arctan function

Softmax

The softmax function is sometimes called the soft argmax function or multi-class logistic regression. This is because the softmax is a generalization of logistic regression that can be used in multi class classification, and its formula is very similar to the sigmoid. The softmax can be used only when the classes are mutually exclusive.

Gudermannian

The Gudermannian function relates circular functions and hyperbolic functions without explicitly using complex numbers.

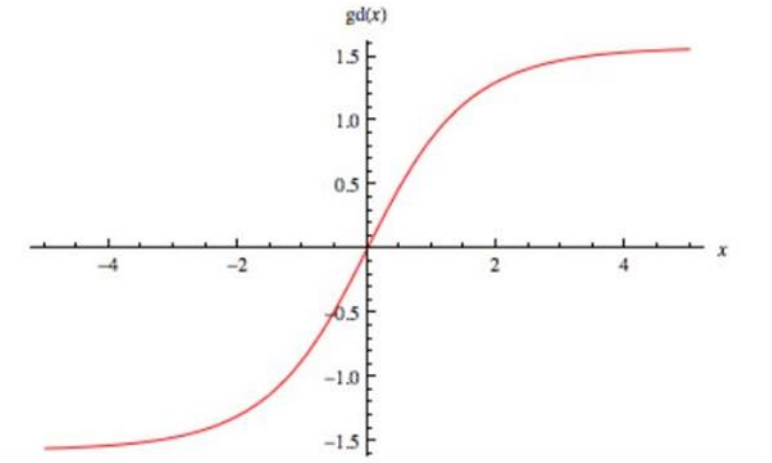


Figure 52 Gundermannian

GELU

(Gaussian Error Linear Units)

An activation function used in the most recent Transformers such as Google's BERT and OpenAI's GPT-2. This activation function takes the form of this equation:

GELU(x)=0.5x(1+tanh(√2/π(x+0.044715×3))).

So, it's just a combination of some functions (e.g. hyperbolic tangent tanh) and approximated numbers.

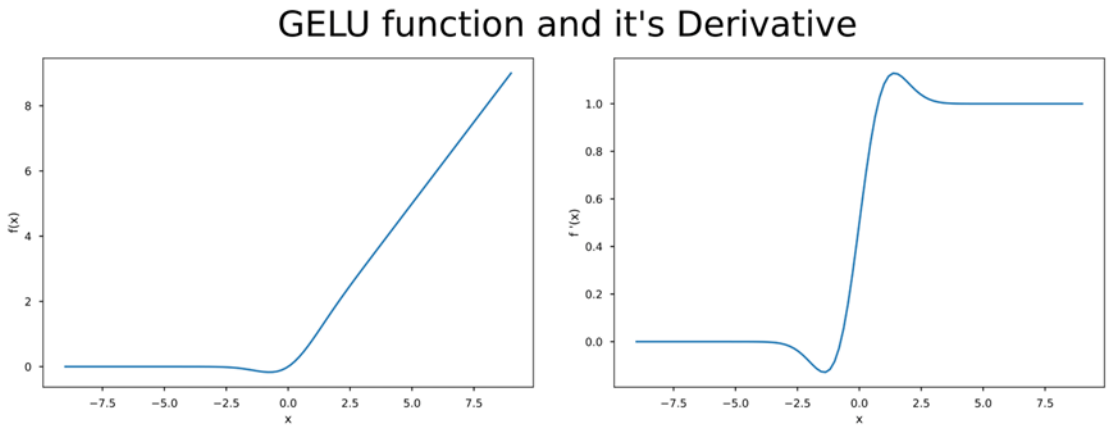


Figure 53 Gelu on left and its derivative on right

RELU

The sigmoid and hyperbolic tangent activation functions cannot be used in networks with many layers due to the vanishing gradient problem. The rectified linear activation function overcomes the vanishing gradient problem, allowing models to learn faster and perform better. The rectified linear activation is the default activation when developing multilayer Perceptron and convolutional neural networks.
ReLU is the most commonly used activation function in neural networks and the mathematical equation for ReLU is:
ReLU(x) = max(0,x) , So, if the input is negative, the output of ReLU is 0 and for positive values, it is x.

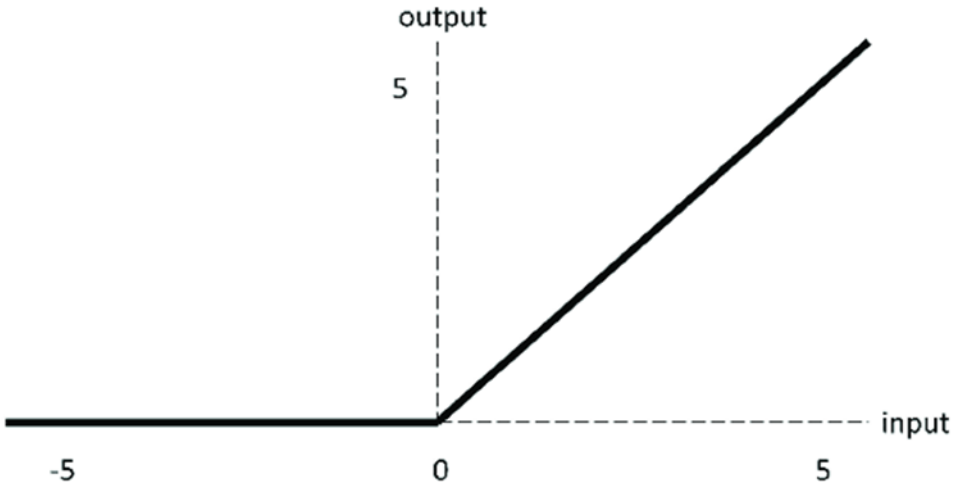


Figure 54 ReLU function

Though it looks like a linear function, it's not. ReLU has a derivative function and allows for backpropagation.

There is one problem with ReLU. Let's suppose most of the input values are negative or 0, the ReLU produces the output as 0 and the neural network can't perform the back propagation. This is called the Dying ReLU problem. Also, ReLU is an unbounded function which means there is no maximum value.

Pros:

- Less time and space complexity
- Avoids the vanishing gradient problem

Cons:

- Introduces the dead relu problem
- Does not avoid the exploding gradient problem

Leaky ReLU

The dying ReLU problem is likely to occur when:

- Learning Rate is too high
- There is a large negative bias

Leaky ReLU is the most common and effective method to solve a dying ReLU problem. It adds a slight slope in the negative range to prevent the dying ReLU issue.

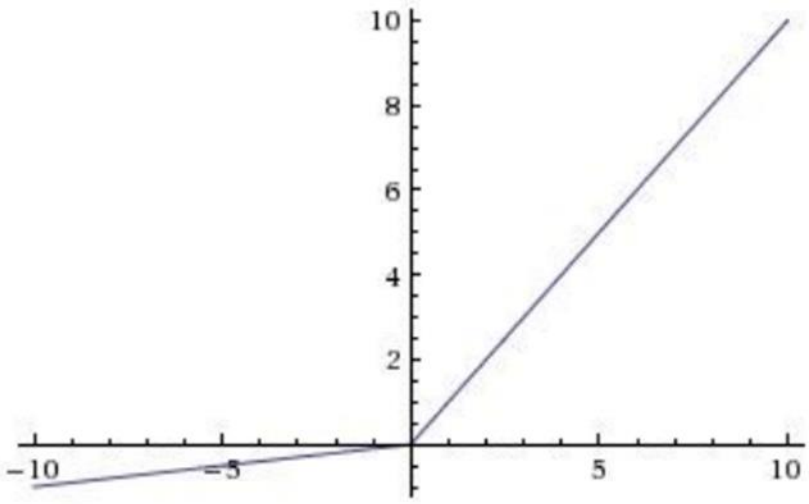


Figure 55 leaky ReLU

Parametric Relu

PReLU is actually not so different from Leaky ReLU. Parametric ReLU is the most common and effective method to solve a dying ReLU problem but again it doesn't solve exploding gradient problem.

$$f(x_i) = \begin{cases} x_i, & \text{if } x_i > 0 \\ \alpha_i x_i, & \text{otherwise} \end{cases}$$

Exponential Linear Unit (ELU)

ELU speeds up the learning in neural networks and leads to higher classification accuracies, and it solves the vanishing gradient problem. ELUs have improved learning characteristics compared to the other activation functions. ELUs have negative values that allow them to push mean unit activations closer to zero like batch normalization but with lower computational complexity.

The mathematical expression for ELU is:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}, \quad f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ f(x) + \alpha & \text{if } x \leq 0 \end{cases}.$$

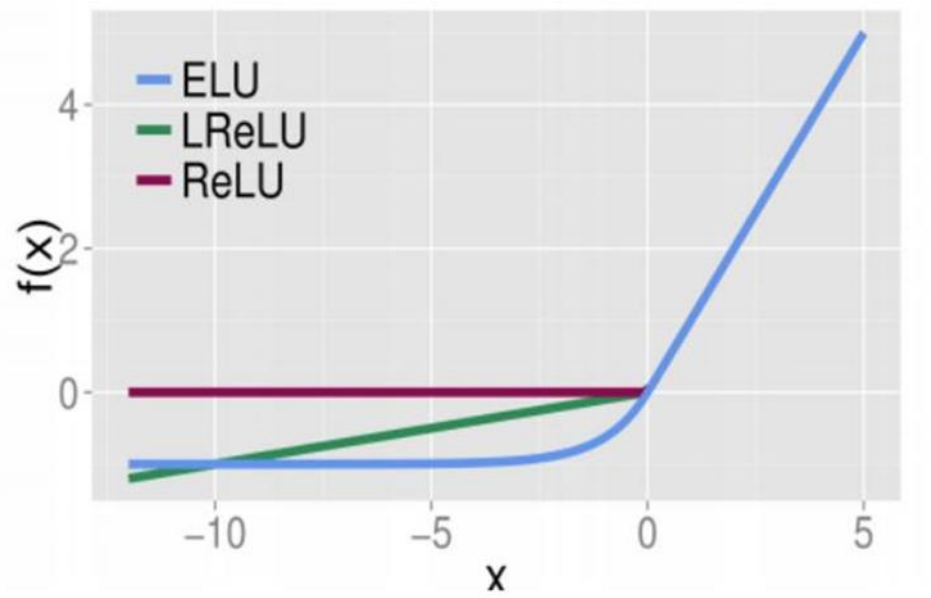


Figure 56 ELU graph in blue

Scaled Exponential Linear Unit (SELU)

SELU incorporates normalization based on the central limit theorem. SELU is a monotonically increasing function, where it has an approximately constant negative output for large negative input. SELU's are mostly commonly used in Self Normalizing Networks (SNN).

The output of a SELU is normalized, which could be called internal normalization, hence the fact that all the outputs are with a mean of zero and standard deviation of one. The main advantage of SELU is that the Vanishing and exploding gradient problem is impossible and since it is a new activation function, it requires more testing before usage

$$\text{selu}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leqslant 0 \end{cases}$$

Figure 57 SeLu mathematical formula

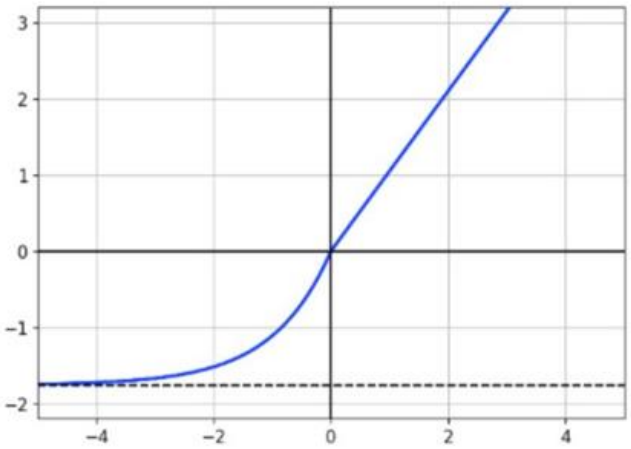


Figure 58 Selu graph

Softplus or Smooth ReLU

The derivative of the softplus function is the logistic function.
The mathematical expression is:

$$f(x) = \ln(1 + e^x)$$

And the derivative is:

$$f'(x) = \frac{1}{1+e^{-x}}$$

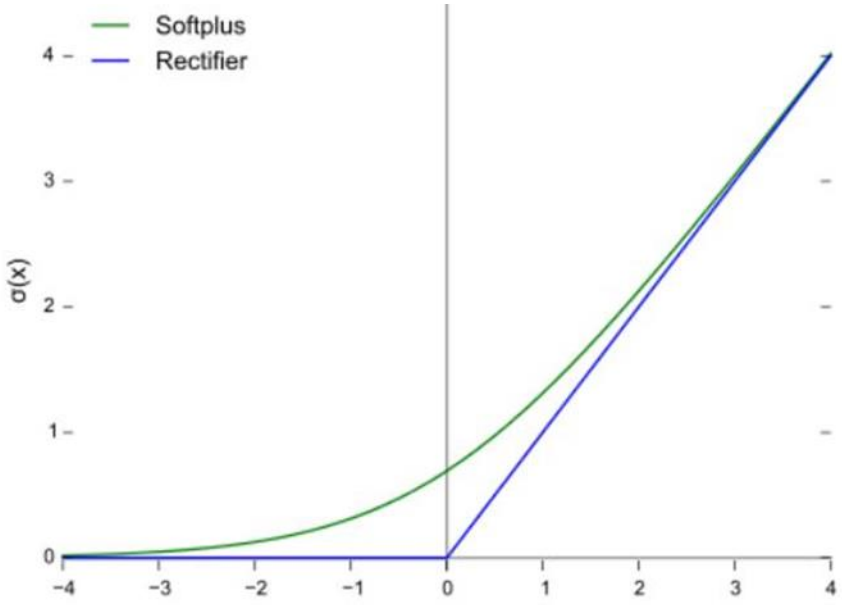


Figure 59 softplus vs RELU

Swish Function

The Swish function was developed by Google, and it has superior performance with the same level of computational efficiency as the ReLU function. ReLU still plays an important role in deep learning studies even for today. But experiments show that this new activation function overperforms ReLU for deeper networks
The mathematical expression for Swish Function is:

$$y = x \,.\, \text{sigmoid}(x)$$

$$y = x \,.\, (1/(1+e^{-x})) = x \,/\, (1+e^{-x})$$

The modified version of swish function is:

$$y = x \,.\, \text{sigmoid}(\beta.x)$$

$$y = x \,.\, (1/(1+e^{-\beta x})) = x \,/\, (1+e^{-\beta x})$$

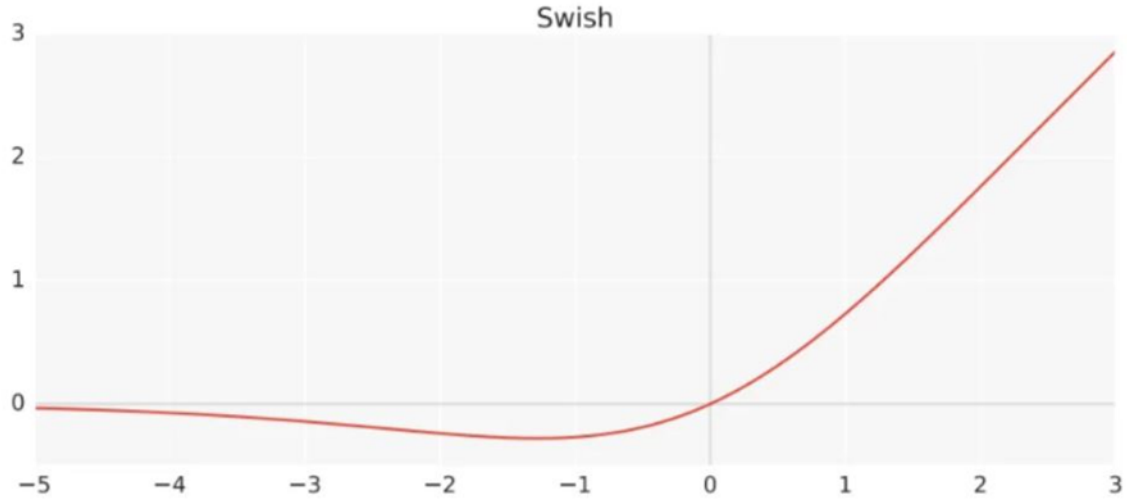


Figure 60 Swish function

Here, β is a parameter that must be tuned. If β gets closer to ∞, then the function looks like ReLU. Authors of the Swish function proposed to assign β as 1 for reinforcement learning tasks. [15].

Linear Algebra essentials for Neural Networks

Introduction

The foundation of machine learning and deep learning systems wholly base upon mathematics principles and concepts. It is imperative to understand the fundamental foundations of mathematical principles. During the baseline and building of the model, many mathematical concepts like the curse of dimensionality, regularization, binary, multi-class, ordinal regression, and others must be artistic in mind.

The basic unit of deep learning, commonly called a neuron, is wholly based on its mathematical concept, and such involves the sum of the multiplied values involving input and weight. Its activation functions like Sigmoid, ReLU, and others, have been built using mathematical theorems.

These are the essential mathematical areas to understand the basic concepts of machine learning and deep learning appropriately:

- Linear Algebra
- Vector Calculus
- Matrix Calculus
- Matrix Decomposition
- Probability and Distributions
- Analytic Geometry

Linear algebra plays a requisite role in machine learning due to vectors' availability and several rules to handle vectors. We mostly tackle classifiers or regressor problems in machine learning, and then error minimization techniques are applied by computing from actual value to predicted value. Consequently, we use linear algebra to handle the before-mentioned sets of computations. Linear algebra handles large amounts of data, or in other words, "linear algebra is the basic mathematics of data."

These are some of the areas in linear algebra that we use in machine learning (ML) and deep learning [16]:

- Vector and Matrix
- System of Linear equations
- Vector space

- Basis

Also, these are the areas of machine learning (ML) and deep learning, where we apply linear algebra's methods:

- Derivation of Regression Line.
- Linear Equation to predict the target value.
- Dimensionality Reduction.
- Mean Square Error or Loss function.
- Regularization.
- Covariance Matrix.
- Convolution.

NOTE: ***We*** will use some sample codes using NumPy and TensorFlow.

Introduction to Tensors and Matrices

Vector

You can think of a vector as simply a list of scalar values. We call these values the elements (entries or components) of the vector. When our vectors represent examples from our dataset, their values hold some real-world significance. For example, if we were training a model to predict the risk that a loan defaults, we might associate each applicant with a vector whose components correspond to their income, length of employment, number of previous defaults, and other factors. If we were studying the risk of heart attacks hospital patients potentially face, we might represent each patient by a vector whose components capture their most recent vital signs, cholesterol levels, minutes of exercise per day, etc. In math notation, we will usually denote vectors as bold-faced, lower-cased letters (e.g., **x, y, z)** [17].

```
x = tf.range(4)
x
<tf.Tensor: shape=(4,), dtype=int32, numpy=array([0, 1, 2, 3])>
```

Matrix

A matrix is an essential part of linear algebra. It stores m*n elements of data, and we use it for the computation of the linear equation system or the linear mappings. It is an m*n tuple of real-value elements.

The number of rows and columns is called the dimension of the matrix.

Figure 61 matrix representation

```
x = tf.reshape(tf.range(20), (5,4))
x
array([[ 0, 1, 2, 3], [ 4, 5, 6, 7], [ 8, 9, 10, 11], [12, 13, 14, 15], [16,
17, 18, 19]])
```

Tensors

Just as vectors generalize scalars, and matrices generalize vectors, we can build data structures with even more axes. Tensors ("tensors" in this subsection refer to algebraic objects) give us a generic way of describing -dimensional arrays with an arbitrary number of axes. Vectors, for example, are first-order tensors, and matrices are second-order tensors. Tensors are denoted with capital letters of a special font face, e.g., **X**, **Y**

Tensors will become more important when we start working with images, which arrive as **n** - dimensional arrays with 3 axes corresponding to the height, width, and a channel axis for stacking the colour channels (red, green, and blue).

```
X = tf.reshape(tf.range(24), (2, 3, 4))
X
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]], dtype=int32)
```

Matrix Multiplication

Matrix multiplication is a dot product of rows and columns where the row of the matrix is multiplied and summed up with another matrix column.

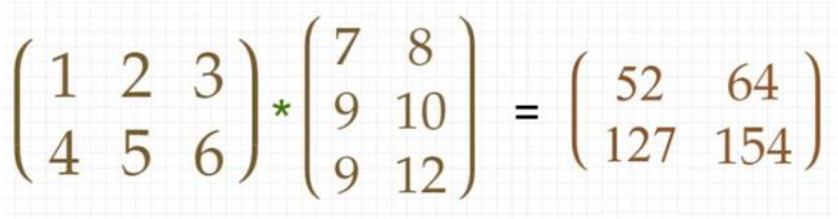


Figure 62 matrix multiplication

in the above TensorFlow example we multiplied (5x4) matrix with another (4x3) matrix, the number of columns in the first matrix must be equal to the number of rows of the second matrix and the resultant matrix has a shape of (number of rows of 1st matrix and the number of columns of 2nd matrix).

Transpose of a Matrix

For A ∈ R^m*n the matrix B ∈ R^n*m with bij = aij is called transpose of A. It is represented as B = A^T. It is just flipping the axes.

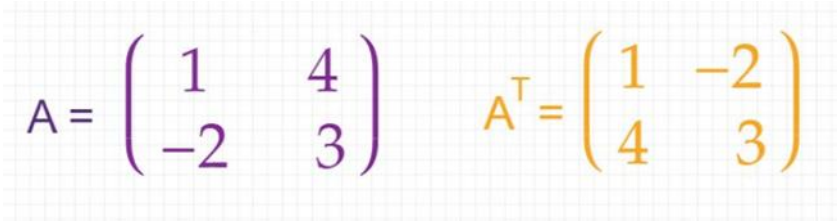
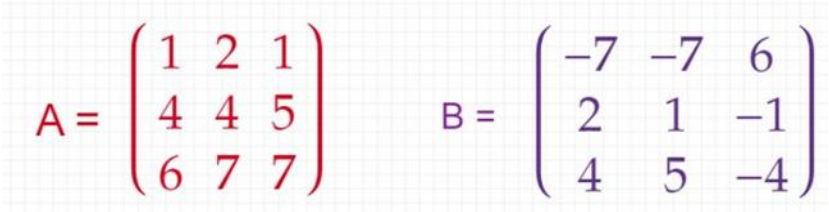


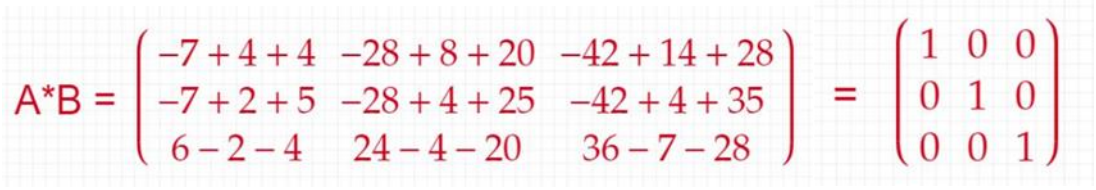
Figure 63 Transpose of a Matrix

```
y = tf.reshape(tf.range(8), (4,2))
z = tf.transpose(y)
z
array([[0, 2, 4, 6], [1, 3, 5, 7]])
```

Inverse of a Matrix

Consider a square matrix **A** ∈ R^*n*n*. Let matrix **B**∈R^*n*n* has the property that **AB** = Identity matrix = **BA, B** is called the inverse of **A** and denoted by **A**^-1.





Orthogonal Matrix

A square matrix A∈R^$n*n$ is an orthogonal matrix if and only if its columns are orthonormal (unit length) so that:

$$A A^T = I = A^T A$$

$$\text{Where, } A^{-1} = A^T$$

Figure 64 orthogonal matrix

Diagonal Matrix

A square matrix A∈R^$n*n$ is a diagonal matrix where all the elements are zero except those on the main diagonal like:

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 65 diagonal matrix

Norms

Some of the most useful operators in linear algebra are norms. Informally, the norm of a vector tells us how big a vector is. The notion of size under consideration here concerns not dimensionality but rather the magnitude of the components.

In linear algebra, a vector norm is a function $\boldsymbol{f}$ that maps a vector to a scalar, satisfying a handful of properties. Given any vector $\boldsymbol{x}$, the first property says that if we scale all the elements of a vector by a constant factor $\boldsymbol{\alpha}$, its norm also scales by the absolute value of the same constant factor:

$$f(\alpha \mathbf{x}) = |\alpha| f(\mathbf{x}).$$

The second property is the familiar triangle inequality:

$$f(\mathbf{x} + \mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y}).$$

The third property simply says that the norm must be non-negative:

$$f(\mathbf{x}) \geq 0.$$

That makes sense, as in most contexts the smallest size for anything is 0. The final property requires that the smallest norm is achieved and only achieved by a vector consisting of all zeros.

$$\forall i, [\mathbf{x}]_i = 0 \Leftrightarrow f(\mathbf{x}) = 0.$$

There are many types of Norms, but the general formula is:

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}.$$

There are general classes of the p – norm:

- L1 norm or Manhattan Norm

  Which is expressed as the sum of the absolute values of the vector elements:

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|.$$

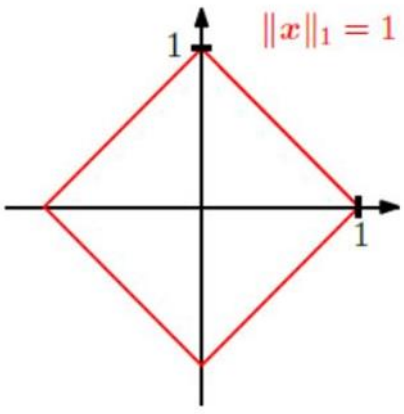


Figure 66 L1 norm representation

As shown in figure 66 the red lines symbolize the set of vectors for the L1 norm equation.

- L2 norm or Euclidean Norm

  The square root of the sum of the squares of the vector elements

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2},$$

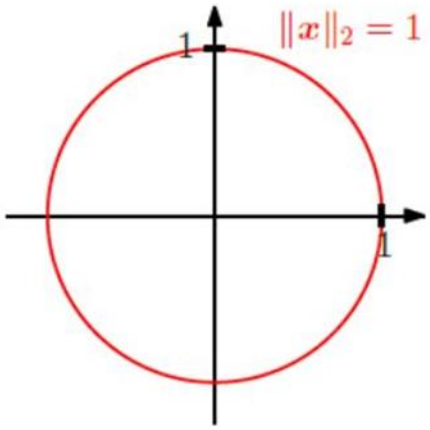


Figure 67 L2 norm representation

Calculus Essentials in Neural Networks

In Machine Learning, what we really care about is producing a model that performs well on data that we have never seen before. But we can only fit the model to data that we can see. Thus, we can decompose the task of fitting models into two key concerns:

1. **Optimization**: the process of fitting our models to observed data.
2. **Generalization**: the mathematical principles and practitioners' wisdom that guide as to how to produce models whose validity extends beyond the exact set of data examples used to train them.

In this section, we're going to mention a very brief primer on differential calculus that is commonly used in deep learning in order to help understanding optimization problems and methods.

Derivatives and Differentiation

We begin by addressing the calculation of derivatives, a crucial step in nearly all deep learning optimization algorithms. In deep learning, we typically choose loss functions that are differentiable with respect to our model's parameters.

For example, for a function f , which has scalar both input and output. The derivative of f is defined as

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Simply, this means that for each parameter, we can determine how rapidly the loss would increase or decrease, were we to *increase* or *decrease* that parameter by an infinitesimally small amount. If $f'(a)$ exists, f is to be differentiable at a . if f is differentiable at every number of an interval, then this function is differentiable on this interval.

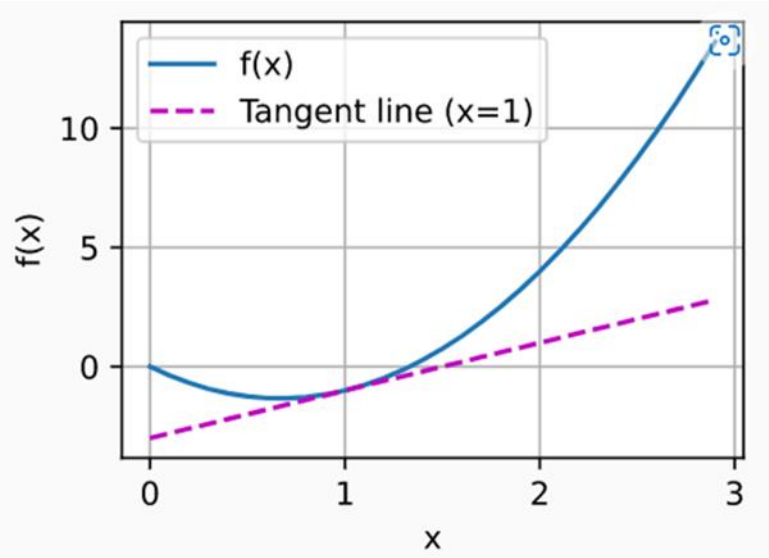


Figure 68 derivative of a function is the slope of this function

The derivative $f'(x)$ can be interpreted as the *instantaneous* rate of change of $f(x)$ with respect to x . the so-called instantaneous rate of change is based on the variation h in x , which approaches 0.

There're a few other equivalent notations for derivatives. Given $y = f(x)$, where $\boldsymbol{x}$ is the independent variable and $\boldsymbol{y}$ is the dependent of the function f , The following expressions are equivalent:

$$f'(x) = y' = \frac{dy}{dx} = \frac{df}{dx} = \frac{d}{dx}f(x) = Df(x) = D_x f(x)$$

where symbols $\frac{\mathrm{d}}{\mathrm{dx}}$ and D are differentiation operators that indicate operation of differentiation. Here are some common functions and its derivative:

- $\frac{dy}{dx}(k) = 0$, where k is a constant.
- $\frac{dy}{dx}(x^n) = nx^{n-1}$, which is called the power rule.
- $\frac{dy}{dx}\left(e^{f(x)}\right) = e^{f(x)}f'(x)$
- $\frac{dy}{dx}(\ln(x)) = {}^{1}/_{x}$

To differentiate a function that is formed from a few simpler functions such as the above common functions, the following rules can be handy for us. Suppose that functions f and g are both differentiable and C is a constant, we have:

- The constant multiple rule

$$\frac{d}{dx}[Cf(x)] = C\frac{d}{dx}f(x),$$

- The Sum Rule

$$\frac{d}{dx}[f(x)+g(x)] = \frac{d}{dx}f(x) + \frac{d}{dx}g(x),$$

- The Product Rule

$$\frac{d}{dx}[f(x)g(x)] = f(x)\frac{d}{dx}[g(x)] + g(x)\frac{d}{dx}[f(x)],$$

- The Quotient Rule

$$\frac{d}{dx}\left[\frac{f(x)}{g(x)}\right] = \frac{g(x)\frac{d}{dx}[f(x)] - f(x)\frac{d}{dx}[g(x)]}{[g(x)]^2}.$$

Partial Derivatives

So far, we have dealt with the differentiation of functions of just one variable, but in deep learning, functions often depend on *many* variables. Thus, we need to extend the ideas of differentiation to these *multivariate* functions. Let:

$y = f(x_1, x_2, \ldots, x_n)$ be a function with n variables. The *partial derivative* of y with respect to its i^{th} parameter x_i is:

$$\frac{\partial y}{\partial x_i} = \lim_{h \to 0} \frac{f(x_1, \ldots, x_{i-1}, x_i + h, x_{i+1}, \ldots, x_n) - f(x_1, \ldots, x_i, \ldots, x_n)}{h}$$

To calculate $\frac{\partial y}{\partial x_i}$, we can simply treat $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$ as constants and calculate the derivative of y with respect to x_i.

There're a few other equivalent notations for derivatives as well:

$$\frac{\partial y}{\partial x_i} = \frac{\partial f}{\partial x_i} = f_{x_i} = f_i = D_i f = D_{x_i} f$$

Gradients

Gradient is a way of packing together every partial derivative information of a function where its representation symbol (∇) is called nabla, del or just gradient.

So, assume that we have a function F(x,y) = $x^2 \sin(y)$,

So Partial of F with respect to X $\frac{\partial f}{\partial x} = 2x \sin(y)$

And Partial of F with respect to Y $\frac{\partial f}{\partial y} = x^2 \cos(y)$

∇F is a vector that has these two partial derivatives in it

$$\nabla f(x,y) = \begin{bmatrix} 2x \sin(y) \\ x^2 \cos(y) \end{bmatrix}$$

So, gradient is a function that takes in a point in two-dimensional space and point it outputs two-dimensional vector.

Chain Rule

However, some gradients can be hard to find. Due to multivariate functions in deep learning are composite, so we probably will not be using the previous method to differentiate these functions. Instead, we will be using chain rule to be able to differentiate these composite functions.

History of chain rule

The chain rule seems to have first been used by Gottfried Wilhelm Leibniz He used it to calculate the derivative of ($\sqrt{a + bz + cz^2}$) as the composite of the square root function and the function ($a + bz + cz^2$) . He first mentioned it in a 1676 memoir (with a sign error in the calculation). The common notation of the chain rule is due to Leibniz. Guillaume de l'Hôpital used the chain rule implicitly in his Analyse des infiniment petits. The chain rule does not appear in any of Leonhard Euler's analysis books, even though they were written over a hundred years after Leibniz's discovery.

Let's say y = f(u) and u = g(x), then the chain rule states that:

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} * \frac{\partial u}{\partial x}$$

Generally speaking, if the function has a random number of variables let's say that a differential function y has variables $u_1, u_2, u_3 \ldots \ldots u_m$ where each differential function u_i has a variable $x_1, x_2, x_3 \ldots \ldots x_n$ then the chain rule states that:

$$\frac{dy}{dx_i} = \frac{dy}{du_1}\frac{du_1}{dx_i} + \frac{dy}{du_2}\frac{du_2}{dx_i} + \cdots + \frac{dy}{du_m}\frac{du_m}{dx_i}$$

Summary

- Differential calculus and integral calculus are two branches of calculus, where the former can be applied to the ubiquitous optimization problems in deep learning.
- A derivative can be interpreted as the instantaneous rate of change of a function with respect to its variable. It is also the slope of the tangent line to the curve of the function.
- A gradient is a vector whose components are the partial derivatives of a multivariate function with respect to all its variables.
- The chain rule enables us to differentiate composite functions.

Training Neural Networks

Once the architecture of the neural network has been defined and includes the feed forward network, the number of hidden layers, the number of neurons per layer, and the activation function, we'll need to set the weights, which, in turn, will define the internal states for each neuron in the network. First, we'll see how to do that for a 1-layer network using an optimization algorithm called gradient descent, and then we'll extend it to a deep feed forward network with the help of backpropagation.

Every neural network is an approximation of a function, so each neural network will not be equal to the desired function, but instead will differ by some value called error. During training, the aim is to minimize this error. Since the error is a function of the weights of the network, we want to minimize the error with respect to the weights. The error function is a function of many weights and, therefore, a function of many variables. Mathematically, the set of points where this function is zero represents a hypersurface, and to find a minimum on this surface, we want to pick a point and then follow a curve in the direction of the minimum.

We would like to stop here and discuss different Neural Networks Loss functions.

The loss function in a neural network quantifies the difference between the expected outcome and the outcome produced by the machine learning model. From the loss function, we can derive the gradients which are used to update the weights. The average over all losses constitutes the cost [18]. To understand how the gradients are calculated and used to update the weights, we will discuss later how the backpropagation algorithm works.

A machine learning model such as a neural network attempts to learn the probability distribution underlying the given data observations. In machine learning, we commonly use the statistical framework of maximum likelihood estimation as a basis for model construction. This basically means we try to find a set of parameters and a prior probability distribution such as the normal distribution to construct the model that represents the distribution over our data [18].

Different types of loss functions

Cross Entropy based Loss functions

Cross-entropy-based loss functions are commonly used in classification scenarios. Cross entropy is a measure of the difference between two probability distributions. In a machine learning setting using maximum likelihood estimation, we want to calculate the difference between the probability distribution produced by the data generating process (the expected outcome) and the distribution represented by our model of that process.
The resulting difference produced is called the loss. It increases exponentially as the prediction diverges from the actual outcome.

If the actual outcome is 1, the model should produce a probability estimate that is as close as possible to 1 to reduce the loss as much as possible.

If the actual outcome is 0, the model should produce a probability estimate that is as close as possible to 0.

Cross entropy is also referred to as the negative log-likelihood.

Binary Cross Entropy

As the name implies, the binary cross-entropy is appropriate in binary classification settings to get one of two potential outcomes. The loss is calculated according to the following formula, where y represents the expected outcome, and y hat represents the outcome produced by our model.

$$L = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

The binary cross-entropy is appropriate in conjunction with activation functions such as Sigmoid that produce a probability relating to a binary outcome.

Categorical Cross Entropy

The categorical cross-entropy is applied in multiclass classification scenarios. In the formula for the binary cross-entropy, we multiply the actual outcome with the logarithm of the outcome produced by the model for each of the two classes and then sum them up. For categorical cross-entropy, the same principle applies, but now we sum over more than two classes. Given that M is the number of classes, the formula is as follows.

$$L = \sum_{j=1}^{M} y_j \log(\hat{y}_j)$$

The categorical cross-entropy is appropriate in combination with an activation function such as the Softmax that can produce several probabilities for the number of classes that sum up to 1.

Sparse Categorical Cross Entropy

In deep learning frameworks such as TensorFlow or Pytorch, you may come across the option to choose sparse categorical cross-entropy when training a neural network.

Sparse categorical cross-entropy has the same loss function as categorical cross-entropy. The only difference is how you present the expected output y, If your y's are encoded in an integer format, you would use sparse categorical cross-entropy, while if they are one hot encoded, i.e. ones and zeroes then you would use Categorical Cross Entropy.

Mean Squared Error

Mean squared error is used in regression settings where your expected and your predicted outcomes are real-number values.

The formula for the loss is straightforward. It is just the squared difference between the expected value and the predicted value.

$$L = (y - \hat{y})^2$$

Note that the cost is the average overall losses for all the individual examples. For the MSE it's written as $C = \frac{1}{N}\sum_{i=1}^{N}(y - \hat{y})^2$.

NOTE: the Cost function generally is that thing that we want the model to minimize as small as the Model can do. So, we can use this to our advantage to minimize any function we want and that enabled the Machine Learning to be a part of many different applications. And that exactly what we used in our Main Project, which is Reducing the PAPR of OFDM system, we literally made the loss function at some point our PAPR and made our model minimize it as possible as it can.

We talked earlier in the *Supervised Learning chapter* about the Linear Regression and Logistic Regression, in fact the Neural Network is a bunch of Logistic Regressions, where linear regression is a special case of a neural network; that is, it's a single neuron with the identity activation function.

Gradient Descent

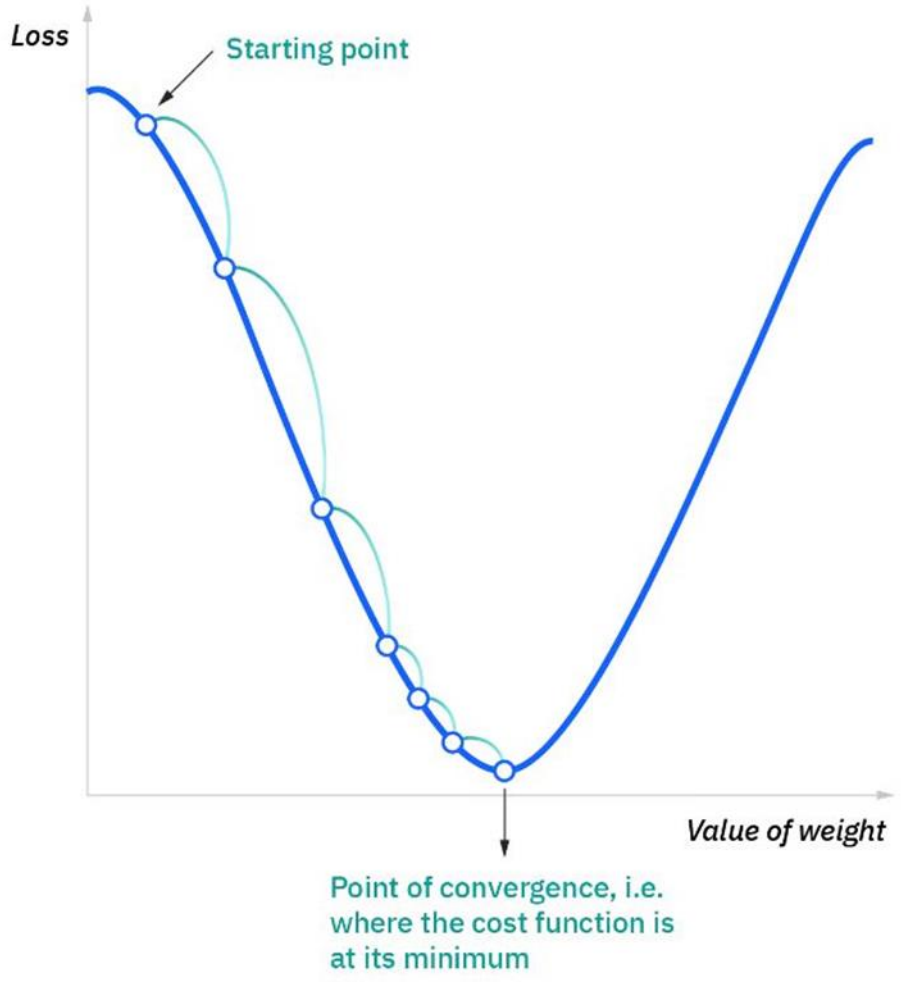


Figure 69 achieving minimum of loss function

Our goal is to go to the point where the cost function is minimum.

The starting point is just an arbitrary point for us to evaluate the performance. From that starting point, we will find the derivative (or slope), and from there, we can use a tangent line to observe the steepness of the slope. The slope will inform the updates to the parameters—i.e., the weights and bias. The slope at the starting point will be steeper, but as new parameters are generated, the steepness should gradually reduce until it reaches the lowest point on the curve, known as the point of convergence. [19]

The goal of gradient descent is to minimize the cost function, or the error between predicted and actual y. In order to do this, it requires two data points —a direction and a learning rate. These factors determine the partial derivative calculations of future iterations, allowing it to gradually arrive at the local or global minimum (i.e., point of convergence). More detail on these components can be found below:

- **Learning rate** (also referred to as step size or the alpha) is the size of the steps that are taken to reach the minimum. This is typically a small value, and it is evaluated and updated based on the behavior of the cost function. High learning rates result in larger steps but risks overshooting the minimum. Conversely, a low learning rate has small step sizes. While it has the advantage of more precision, the number of iterations compromises overall efficiency as this takes more time and computations to reach the minimum.
- **The cost (Loss) function** measures the difference, or error, between actual y and predicted y at its current position. This improves the machine learning model's efficacy by providing feedback to the model so that it can adjust the parameters to minimize the error and find the local or global minimum. It continuously iterates, moving along the direction of steepest descent (or the negative gradient) until the cost function is close to or at zero. At this point, the model will stop learning. Additionally, while the terms, cost function and loss function, are considered synonymous, there is a slight difference between them. It's worth noting that a loss function refers to the error of one training example, while a cost function calculates the average error across an entire training set. As we talked earlier about the Loss functions.

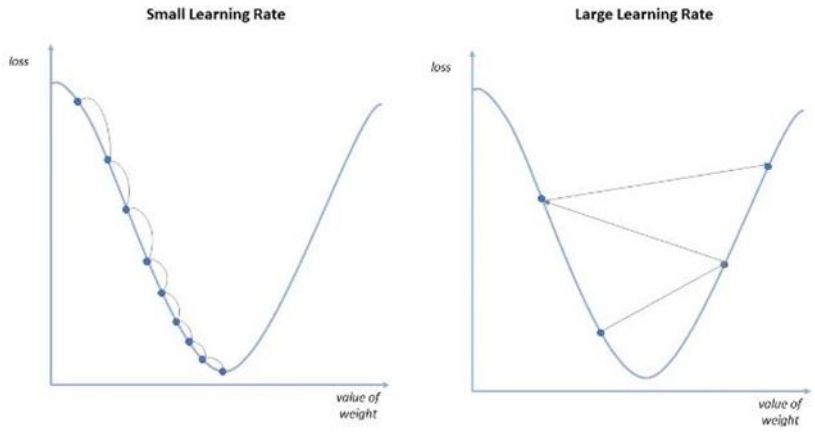


Figure 70 learning rate effect

Types of Gradient Descent

There are three types of gradient descent learning algorithms: batch gradient descent, stochastic gradient descent and mini-batch gradient descent.

- **Batch Gradient Descent**

  Batch gradient descent sums the error for each point in a training set, updating the model only after all training examples have been evaluated. This process referred to as a training epoch.
  While this batching provides computation efficiency, it can still have a long processing time for large training datasets as it still needs to store all of the data into memory. Batch gradient descent also usually produces a stable error gradient and convergence, but sometimes that convergence point isn't the most ideal, finding the local minimum versus the global one.

- **Stochastic Gradient Descent**

  Stochastic gradient descent (SGD) runs a training epoch for each example within the dataset and it updates each training example's parameters one at a time. Since you only need to hold one training example, they are easier to store in memory. While these frequent updates can offer more detail and speed, it can result in losses in computational efficiency when compared to batch gradient descent. Its frequent updates can result in noisy gradients, but this can also be helpful in escaping the local minimum and finding the global one.

- **Mini-batch gradient descent**

  Mini-batch gradient descent combines concepts from both batch gradient descent and stochastic gradient descent. It splits the training dataset into small batch sizes and performs updates on each of those batches. This approach strikes a balance between the computational efficiency of batch gradient descent and the speed of stochastic gradient descent.

Challenges with Gradient Descent

While gradient descent is the most common approach for optimization problems, it does come with its own set of challenges. Some of them include:

- Local minima and saddle points

For convex problems, gradient descent can find the global minimum with ease, but as nonconvex problems emerge, gradient descent can struggle to find the global minimum, where the model achieves the best results.
Recall that when the slope of the cost function is at or close to zero, the model stops learning. A few scenarios beyond the global minimum can also yield this slope, which are local minima and saddle points. Local minima mimic the shape of a global minimum, where the slope of the cost function increases on either side of the current point. However, with saddle points, the negative gradient only exists on one side of the point, reaching a local maximum on one side and a local minimum on the other. Its name inspired by that of a horse's saddle.
Noisy gradients can help the gradient escape local minimums and saddle points.

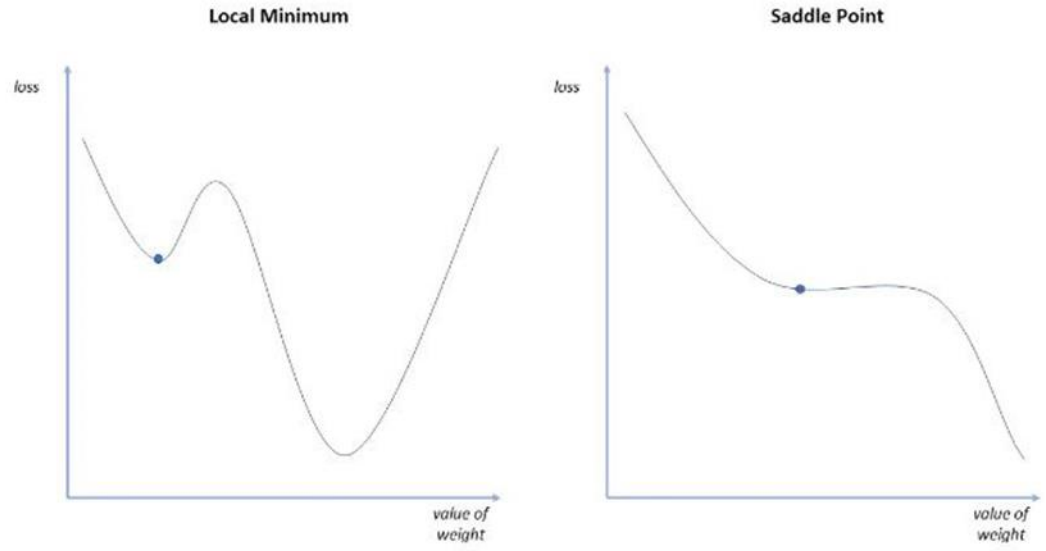


Figure 71 local minimum vs saddle point

- Vanishing and Exploding gradients

In deeper neural networks and particular recurrent neural networks, we encounter two other problems when the model is trained with gradient descent and backpropagation.

1. Vanishing Gradient

This occurs when the gradient is too small. As we move backwards during backpropagation, the gradient continues to become smaller, causing the earlier layers

in the network to learn more slowly than later layers. When this happens, the weight parameters update until they become insignificant i.e., 0 resulting in an algorithm that is no longer learning.

2. Exploding Gradients

   This happens when the gradient is too large, creating an unstable model. In this case, the model weights will grow too large, and they will eventually be represented as NaN. One solution to this issue is to leverage a dimensionality reduction technique, which can help to minimize complexity within the model.

The Training Process

Backpropagation

So far, we have learned how to update the weights of 1-layer networks with gradient descent. We started by comparing the output of the network (that is, the output of the output layer) with the target value, and then we updated the weights accordingly. But, in a multi-layer network, we can only apply this technique for the weights that connect the final hidden layer to the output layer. That's because we don't have any target values for the outputs of the hidden layers. What we'll do instead is calculate the error in the final hidden layer and estimate what it would be in the previous layer. We'll propagate that error back from the last layer to the first layer; hence, we get the name backpropagation. [14]

We always start from the output layer and propagate backwards, updating weights and biases for each layer.

The idea is simple, adjust the weights and biases throughout the network, so that we get the desired output in the output layer. Say we wanted the output neuron to be 1.0, then we would need to nudge the weights and biases so that we get an output closer to 1.0.

We can only change the weights and biases, but activations are direct calculations of those weights and biases, which means we indirectly can adjust every part of the neural network, to get the desired output except for the input layer, since that is the dataset that you input. [16]

Computing Gradients

Now, before the equations, let's define what each variable means.

We'll define w_{ij} as the weight between the i-th neuron of layer L, and the j-th neuron of layer L+1, In other words, we use subscripts i and j, where the element with subscript i belongs to the layer preceding the layer containing the element with subscript j.

L = last layer

L-1 = second last layer

l = layer

C = cost function

W = weight

b = bias

J = cost function (error)

Y= the activation function (sigmoid, ReLU, and so on) output. [14]

So, we can write the following for the last layer of our neural network (using partial derivatives):

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}}$$

Since we know that $\frac{\partial a_j}{\partial w_{ij}} = y_i$, we have the following:

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial a_j}y_i$$

For the previous (hidden) layers, the same formula holds:

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}}$$

And we also know that $\frac{\partial y_j}{\partial a_j}$ is the derivative of the activation function, which we can calculate. Then, all we need to do is calculate the derivative $\frac{\partial J}{\partial y_j}$. Let's note that this is the derivative of the error with respect to the activation function in the "second" layer - as $y = x * w + b$ -. We can now calculate all the derivatives, starting from the last layer and moving backward.

In the following equation y_i is the output of the first layer (and input for the second), while y_j is the output of the second layer. Applying the chain rule, we have the following:

$$\frac{\partial J}{\partial y_i} = \sum_j \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial y_i} = \sum_j \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial a_j}\frac{\partial a_j}{\partial y_i}$$

Once again, we can calculate both $\frac{\partial y_j}{\partial a_j}$ and $\frac{\partial a_j}{\partial y_i} = w_{ij}$; once we know $\frac{\partial J}{\partial y_j}$, we can calculate $\frac{\partial J}{\partial y_i}$.

Since we can calculate $\frac{\partial J}{\partial y_j}$ for the last layer, we can move backward and calculate $\frac{\partial J}{\partial y_i}$ for any layer, and therefore $\frac{\partial J}{\partial w_{ij}}$ for any layer.

To summarize if we have a sequence of layers where the following applies:

$$y_i \gg y_j \gg y_k$$

We then have these two fundamental equations:

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}}$$

$$\frac{\partial J}{\partial y_j} = \sum_k \frac{\partial J}{\partial y_k}\frac{\partial y_k}{\partial y_j}$$

By using these two equations, we can calculate the derivatives for the cost with respect to each layer. If we set $\delta_j = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial a_j}$, then δ_j represents the variation in cost with respect to the activation value, and we can think of δ_j as the error at neuron y_j. So, $\delta_i = (\sum_j \delta_j w_{ij})\frac{\partial y_i}{\partial a_i}$

The update rule for the weights of each layer is given by the following equation:

$$w_{ij} \gg w_{ij} - \mu\delta_j y_i$$

INTRODUCTION TO DEEP LEARNING

In this chapter, we will cover some of the basics of deep learning and deep neural networks. The word Deep is said on Networks with lots of hidden layers. You may ask why we need neural networks with lots of hidden layers. Without going in too much detail, one reason is that approximating a complex function might require a huge number of neurons in the hidden layer, making it impractical to use. There is also another, more important, reason for using deep networks, which is not directly related to the number of hidden layers, but to the level of learning. A deep network does not simply learn to predict output Y given input X; it also understands basic features of the input. It's able to learn abstractions of features of input examples, to understand the basic characteristics of the examples, and to make predictions based on those characteristics. This is a level of abstraction that is missing in other basic machine learning (ML) algorithms and in shallow neural networks.

We could define deep learning as a class of machine learning techniques, where information is processed in hierarchical layers to understand representations and features from data in increasing levels of complexity. In practice, all deep learning algorithms are neural networks, which share some common basic properties. They all consist of interconnected neurons that are organized in layers. Where they differ is network architecture (or the way neurons are organized in the network), and sometimes in the way they are trained. [14]

Main Classes of Deep Neural Network

- **Multi-Layer perceptrons (MLPs):** A neural network with feed-forward propagation, fully connected layers, and at least one hidden layer.
- **Convolutional Neural Networks (CNNs):** A CNN is a feedforward neural network with several types of special layers. For example, convolutional layers apply a filter to the input image (or sound) by sliding that filter all across the incoming signal, to produce an n-dimensional activation map. There is some evidence that neurons in CNNs are organized similarly to how biological cells are organized in the visual cortex of the brain. We've mentioned CNNs several times up to now, and that's not a

coincidence – today, they outperform all other ML algorithms on a large number of computer vision and NLP tasks.

- **Recurrent Neural Networks:** This type of network has an internal state (or memory), which is based on all or part of the input data already fed to the network. The output of a recurrent network is a combination of its internal state (memory of inputs) and the latest input sample. At the same time, the internal state changes, to incorporate newly input data. Because of these properties, recurrent networks are good candidates for tasks that work on sequential data, such as text or time series data.
- **Autoencoders:** An autoencoder is a feed forward neural network that tries to reproduce its input. In other words, the target value (label) of an autoencoder is equal to the input data. We can say that it tries to learn an identity function.
  Since our "labels" are just the input data, the autoencoder is an unsupervised algorithm. The model is forced to prioritize which aspects of the input should be copied; it often learns useful properties of the data.

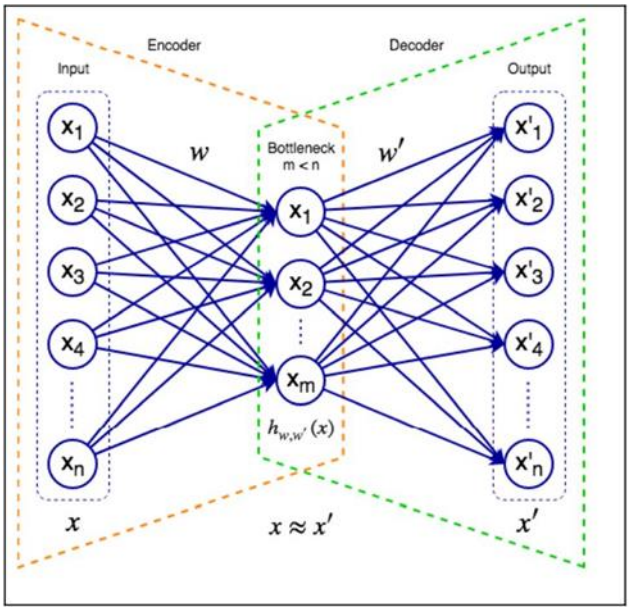


Figure 72 Autoencoder

But we must notice that when the network is learning the identity function, which means the output becomes equal to the input, the autoencoder is somehow useless. Denoising Autoencoders solve this problem by corrupting the data on purpose by randomly turning some of the input values to zero. In general, the percentage of input nodes which are being set to zero is about 50%. Other sources suggest a lower count, such as 30%. It depends on the amount of data and input nodes you have.

When calculating the Loss function, it is important to compare the output values with the original input, not with the corrupted input. That way, the risk of learning the identity function instead of extracting features is eliminated.

Denoising Autoencoders are an important and crucial tool for feature selection and extraction

We used that in our advantage in our second model **PAPR reduction using a Denoising Autoencoder**, as the transmitted symbol are vulnerable to the channel with all of its effects, so the transmitted symbols are heavily corrupted, and the benefit of the Denoising Autoencoders occurs here and we train it to get back the original symbols from the corrupted ones.

An autoencoder consists of an input, hidden (or bottleneck), and output layers. Although it's a single network, we can think of it as a virtual composition of two components:

- **Encoder**

  Maps the input data to the network's internal representation.
- **Decoder**

  Tries to reconstruct the input from the network's internal data representation. The decoder can also have a complex structure, which typically mirrors the encoder.

We can train the autoencoder by minimizing a loss function, which is known as the reconstruction error. It measures the distance between the original input and its reconstruction. We can minimize it in the usual way with gradient descent and backpropagation. Depending on the approach, we can use either mean square error (MSE) or binary cross-entropy (such as cross-entropy, but with two classes) as reconstruction errors.

Also, the Autoencoders can be used in compression, Because the network tries to reconstruct its input from a smaller feature space, it learns a compact representation of the data.

Training Deep Networks

We can use different algorithms to train a neural network. But in practice, we almost always use Stochastic Gradient Descent (SGD) and backpropagation, in which we talked about in details in Neural Networks Chapter.

But we won't use the Vanilla Gradient Descent, we will use momentum with it as we explained momentum before in the previous chapter.

Regularization in Machine Learning

Regularization is a process of modifying the loss function to penalize specific values of the weight on learning. Regularization helps us avoid overfitting.

It is an excellent addition in machine learning for the operations below:

- ✓ Handle collinearity
- ✓ Filter out noise from data

By noise we mean those data points in the dataset which don't really represent the true properties of your data, but only due to a random chance.

- ✓ Prevent Overfitting

This technique prevents the model from overfitting by adding **extra information** to it.

- ✓ Generally, provide better performance

One of the standard regularization techniques is **Ridge Regression** or **L2 norm.**

Ridge regression is one of the types of linear regression in which we introduce a small amount of bias, known as Ridge regression penalty, so that we can get better long-term predictions.

The cost function is altered by adding the penalty term (shrinkage term), which multiplies the lambda with the squared weight of each individual feature. Therefore, the optimization function (cost function) becomes:

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2=\mathrm{RSS}+\lambda\sum_{j=1}^{p}\beta_j^2$$

Figure 73 cost function for ridge regression

Where RSS is the Residual Sum of Squares which is the older cost function without regularization.

Uses of Ridge Regression:

- When we have the independent variables which are having high collinearity.

Multicollinearity is a famous problem in Data Science, occurs when two or more independent variables (also known as predictor) are highly correlated with one

another in a regression model. We won't discuss it more than that, since it's not directly related to our project.

- If we have more parameters than the samples, then Ridge regression helps to solve the problems.

Limitations of Ridge Regression:

- Not helps in Feature Selection

It decreases the complexity of a model but does not reduce the number of independent variables since it never leads to a coefficient being zero rather only minimizes it. Hence, this technique is not good for feature selection.

- Model Interpretability

Its disadvantage is model interpretability since it will shrink the coefficients for least important predictors, very close to zero but it will never make them exactly zero. In other words, the final model will include all the independent variables, also known as predictors.

The other technique of Regularization is the **Lasso Regression** or **L1 norm.**

Lasso regression is another variant of the regularization technique used to reduce the complexity of the model. It stands for Least Absolute and Selection Operator.

It is similar to the Ridge Regression except that the penalty term includes the absolute weights instead of a square of weights. Therefore, the optimization function becomes:

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}|\beta_j| = \mathrm{RSS}+\lambda\sum_{j=1}^{p}|\beta_j|.$$

In this technique, the L1 penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero which means there is a complete removal of some of the features for model evaluation when the tuning parameter λ is sufficiently large. Therefore, the lasso method also performs Feature selection and is said to yield sparse models.

Limitations of Lasso Regression:

- Problems with some types of Datasets: If the number of predictors is greater than the number of data points, Lasso will pick at most n predictors as non-zero, even if all predictors are relevant.
- Multicollinearity Problem: If there are two or more highly collinear variables then LASSO regression selects one of them randomly which is not good for the interpretation of our model.

Differences between Lasso and Ridge Regressions:

- Ridge regression helps us to reduce only the overfitting in the model while keeping all the features present in the model.

  It reduces the complexity of the model by shrinking the coefficients whereas Lasso regression helps in reducing the problem of overfitting in the model as well as automatic feature selection.
- Lasso Regression tends to make coefficients to absolute zero whereas Ridge regression never sets the value of coefficient to absolute zero.

  In a nutshell, we can say that Regularization tries to reduce the variance of the model, without big increase in the bias, because the standard least square models tend to have some variance in it and this makes model unable to generalize well for future data sets other than the training set.

Important note on λ :

- It is a tuning parameter used in regularization that decides how much we want to penalize the flexibility of our model i.e., controls the impact on bias and variance.

  We used the λ concept in our project in a way that we want to minimize a certain cost function but at the same time we don't want to deteriorate another cost function. In short words we used it to balance between two cost functions, one for the PAPR and the other for the Bit Error Rate.

Dropout in Neural Networks

Dropout is a technique where randomly selected neurons are ignored during training. They are "dropped-out" randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass. You can imagine that if neurons are randomly dropped out of the network during training, that other neurons will have to step in and handle the representation as required to make predictions for the missing neurons. This is believed to result in multiple independent internal representations being learned by the network.

The effect is that the network becomes less sensitive to the specific weights of neurons. This in turn results in a network that is capable of better generalization and is less likely to overfit the training data.

Tips for using Dropout:

- Generally, use a small dropout value of 20%-50% of neurons with 20% providing a good starting point. A probability too low has minimal effect and a value too high results in under-learning by the network.
- Use a larger network. You are likely to get better performance when dropout is used on a larger network, giving the model more of an opportunity to learn independent representations.
- Use dropout on incoming (visible) as well as hidden units. Application of dropout at each layer of the network has shown good results.

Batch Normalization

Training deep neural networks with tens of layers is challenging as they can be sensitive to the initial random weights and configuration of the learning algorithm.

One possible reason for this difficulty is the distribution of the inputs to layers deep in the network may change after each mini batch when the weights are updated. This can cause the learning algorithm to forever chase a moving target. This change in the distribution of inputs to layers in the network is referred to the technical name "*internal covariate shift*."

One possible reason for this difficulty is the distribution of the inputs to layers deep in the network may change after each mini batch when the weights are updated. This can cause the learning algorithm to forever chase a moving target. This change in the distribution of inputs to layers in the network is referred to the technical name "*internal covariate shift*."

Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

How Batch Normalization is done

First step is **Normalizing the input**, which means transforming the data to have a mean zero and standard deviation of one.

The final step is **Rescaling and Offsetting,** In the final operation, the re-scaling and offsetting of the input take place. Here two components of the BN algorithm come into the picture, γ(gamma) and β (beta). These parameters are used for re-scaling (γ) and shifting(β) of the vector containing values from the previous operations.

Advantages of Batch Normalization

- Speed up the training

  By Normalizing the hidden layer activation, the Batch normalization speeds up the training process.
- Handles internal covariate shift

  It solves the problem of internal covariate shift. Through this, we ensure that the input for every layer is distributed around the same mean and standard deviation.
- Smoothens the Loss Function

  Batch normalization smoothens the loss function that in turn by optimizing the model parameters improves the training speed of the model.

PAPR REDUCTION USING DEEP LEARNING

Tone Reservation Scheme based on Deep Learning

Introduction

As we mentioned before in the OFDM part of this book how the OFDM has outstanding performance in frequency selective channels, but its main drawback is the high PAPR that makes the transmitted signal to enter into the non-linear region of the High-Power Amplifier (HPA), which leads to signal distortion. We talked also about various methods to reduce the high PAPR as Partial Transmit Sequence (PTS), Selective Mapping (SLM), Clipping and Filtering and Tone Reservation or Tone Injection. In particular, the Tone Reservation technique has received a big attention from researchers, although it reduces the Bandwidth efficiency of the transmitted signal as we reserve certain tones without carrying any information but only to reduce the PAPR and also it increases the power of the transmitted signal, but it's a distort less technique and doesn't any side information to the transmitted signal and thus no complex receiver is needed.

However, the TR technique's performance is limited due to the finite candidate set of the Peak Reduction vector, it also suffers from big execution time.

In recent years Deep Learning has gained big momentum in many fields as we mentioned earlier in this book, and one of the main applications that we are concerned about is the **Intelligent Communication.** The basic idea of the intelligent communication is introducing Artificial Intelligence into the field of wireless communication systems to improve the performance.

Main Idea

We propose a Tone Injection Network based on Deep Neural Network to enhance the classical Tone Reservation technique, that we used earlier in thus book.

In the proposed scheme we utilize the Feed Forward Neural Network to adaptively generate the peak cancelling signal according to the input signal characteristics.

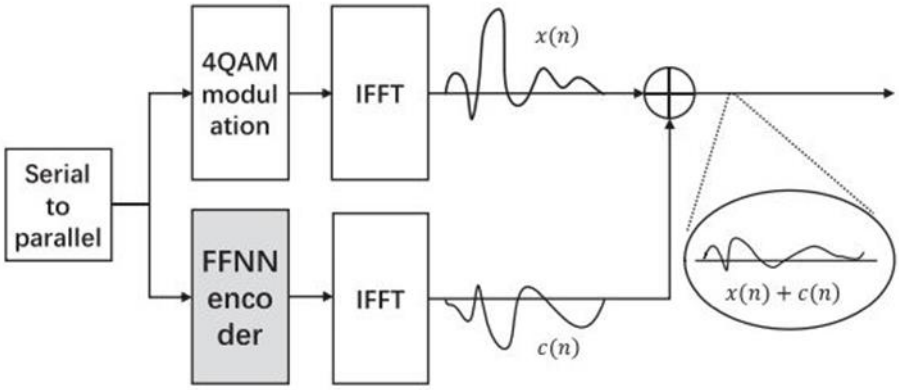


Figure 74 overview of the TRNet

The baseband signal is copied into two copies. One copy is used for generating the time domain signal ***x(n)*** by using the modulation scheme and the IFFT operation. The other duplicate is fed into the FFNN to generate the frequency domain reserved symbol vector ***C*** and hence we can get the time domain peak cancelling signal ***c(n)***. and finally, the peak cancelling signal is $\hat{x}_n = x_n + c_n$

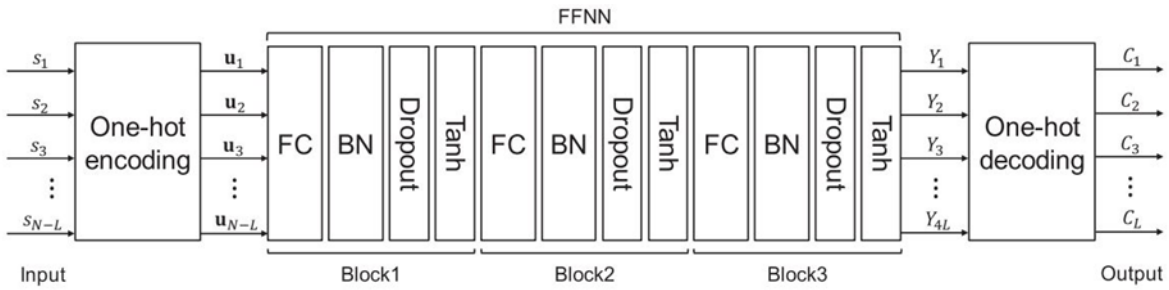


Figure 75 FFNN Block Diagram

As shown in [20] the baseband signal s is one hot encoded and then goes into the Feed Forward Neural Network which consists of three identical blocks that are connected in series. Each block consists of Fully Connected Layer (FC) that extracts the features of the input signal, then Batch Normalization (BN) which speeds up the training process, then dropout layer that prevents the model from overfitting and makes the model able to generalize better and non-Linear activation function which is the tanh.

Compared with the other proposed scheme that we will talk about later; this scheme only uses the Feed Forward Network at the Transmitter only which results in a lower cost to the system.

The proposed scheme reduces the PAPR of the OFDM system significantly without causing any distortion to the system, in addition to that the Neural Network based scheme uses a smaller number of reserved tones than the classical TR which improves the Bandwidth efficiency.

System Model

Considering OFDM system with N subcarriers, where the discrete time transmitted signal can be written as:

$$x[n] = \sum_{k=0}^{N-1} X_k e^{j2\pi nk/N}$$

Where $0 \leq n \leq N-1$, X_k is a PSK or QAM modulated symbol transmitted through the k^{th} subcarrier.

The PAPR can be defined as:

$$PAPR\{x[n]\} = \frac{\max\limits_{0 \leq n \leq N-1} x[n]^2}{\mathbb{E}\left[|x[n]|^2\right]}.$$

TR Technique

TR technique uses a part of tones for conveying the reserved symbols that is used to reduce the PAPR. By ignoring those tones by the Receiver, it can recover the transmitted useful data.

TR technique reserves L subcarriers out of N subcarriers; thus, we have $N - L$ subcarriers used for data symbol transmission. The original time domain signal is written as $\hat{x} = x + c = IFFT(X + C)$, where C is the reserved symbol vector in the frequency domain, c is the peak cancelling signal in the time domain. To avoid distortion X and C are designed to be orthogonal on each other.

The optimal reserved symbol vector is chosen that it gives the minimum PAPR.

The Tone Reservation Ratio (TRR) or the Data Rate Loss is defined as:

$TRR = \frac{L}{N}$, which is the ratio between the number of reserved tones to the total number of tones. The greater it becomes the more PAPR is reduced but the Bandwidth efficiency becomes worse, and hence we must find the best compromise between the PAPR and Bandwidth efficiency.

TRNet Structure

The neural network consists of five layers, one input layer, three hidden layers and one output layer. All layers are connected in series.

Each hidden layer consists of a Fully Connected layer (FC), Batch Normalization (BN), Dropout and tanh activation function.

The core of the FC layer is the matrix multiplication that is equivalent to a feature space transformation. Specifically, if the input of the FC layer is (x1, x2, ⋯ , xn) ∈ Rn, then the output of the FC layer can be expressed as

$$y_j^{(k)} = \sum_{i=1}^{N} w_{ji}^{(k)} x_i + b_j^{(k)},$$

Where $w_{ji}^{(k)}$ is the weight of i^{th} input in the k^{th} weight matrix layer connected to the j^{th} hidden layer neuron.

Batch Normalization layer normalizes the output of the FC layer, which avoids the problem of vanishing gradient and increases the training speed.

More specifically, BN can be expressed as

$$\widehat{\mathbf{y}}^{(k)} = \gamma \frac{\mathbf{y}^{(k)} - \mathbb{E}(\mathbf{y}^{(k)})}{\sqrt{D(\mathbf{y}^{(k)}) + \varepsilon}} + \beta$$

where ε is a small value that avoids the denominator to be zero. γ and β are adjustable parameters, which enhance the expressivity of the neural network. This means that the neural network can adaptively adjust the distribution in the training process to accelerate the convergence.

Additionally, in the proposed scheme, the trained model has the overfitting problem. In order to solve this problem, this letter uses the dropout algorithm that reduces the overfitting by ignoring half of the feature detectors in each training batch (half of the neurons in the hidden layer are zero).

The tanh is the activation function for TRNet, which can be expressed as $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ The input data of the activation function will be limited to [-1, 1] after the BN operation, and tanh is very close to y = x in this interval. Therefore, it can directly perform the matrix operation according to the nature of identity function, which can improve the training efficiency.

Data Pre-processing

Since Neural Networks are unable to deal with complex numbers directly, We tried two methods to overcome this problem. We splitted the data into real and imaginary which doubled the size of the subcarriers into 2N, but that's not a physical increase in the subcarriers it's just in the dimensions of the matrix that represent our data. Another solution but more complicated is using One Hot Encoding to re-parameterize the input and output data of the neural network. We tried both ways and they gave the same results, so we sticked with the simpler method which is splitting the complex data into real and imaginary.

Training of TRnet and Loss Function

As we discussed before the main target of a deep neural network model is to minimize a given Cost function whether it's MSE or Cross Entropy. But here in our case we made the Cost function to be the PAPR, and hence the model tries to minimize the PAPR of our OFDM system.

The Loss function can be written as

$$\begin{aligned} LOSS &= PAPR\{\hat{\mathbf{x}}\} \\ &= PAPR\{\mathbf{x}+\mathbf{c}\} = PAPR\{\mathrm{Q}(\mathbf{X}+\mathbf{C})\} \end{aligned}$$

Where x^ is the transmitted signal that is created by summing the peak-cancelling signal c generated by the FFNN encoder and the data signal x.

Performance Evaluation

Considering OFDM system with N = 256 subcarriers, L = 8 reserved tones, 150,000 symbols for training and 4 QAM modulation scheme.

The proposed Neural Network uses learning rate of 0.001 and Adam optimizer, also the batch size is set to 400. The execution time is approximately **13.01334** seconds for 10,000 symbols.

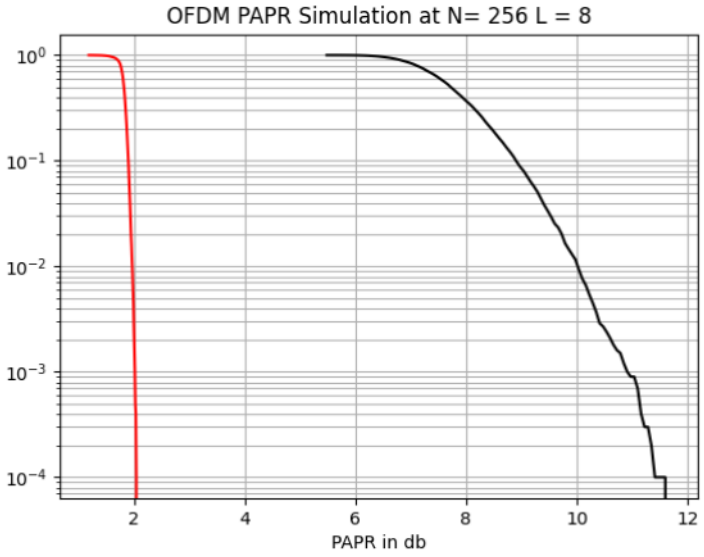


Figure 76 TRNet PAPR CCDF

We simulated the model which is proposed in [20]. But the model didn't give the claimed results as shown in figure 77.

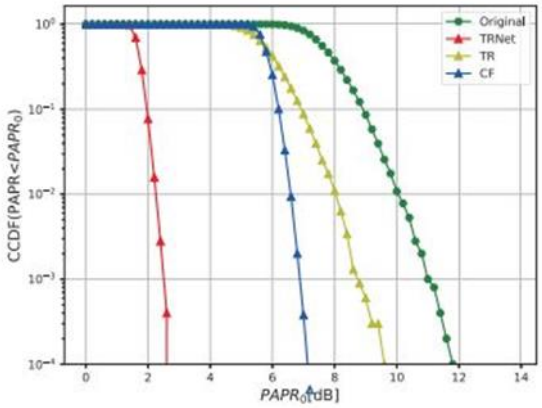


Figure 77 TRnet PAPR CCDF in IEEE letter

And after a lot of digging and trials, we found that if we removed the **tanh** activation function from the output layer, we would get the same or better performance that was proposed in [20]. But removing the **tanh** activation function in the output layer makes the output of the model to not be limited in the interval [-1,1], but it makes the model outputs any range of values. Due to the removal of the **tanh** the reserved tones will have higher power than the tones carrying information.

We tried to limit the values of the reserved tones by using a **custom activation function** which is a scaled version of the **tanh** activation function.

Firstly, we tried to rescale the **tanh** to be from [-5,5] and it did give the results shown in figure 78 Which gave similar results of the classical Tone Reservation shown in figure 79. But the main advantage of our model is having significantly less execution time than the classical method which takes a longer time, as it tries many combinations from a given candidate set, but the deep learning model generates peak cancelling vector adaptively according to the features of the input.

Only a few numbers of data sets are required for the learning and the model can converge quickly. This greatly improves the robustness of our scheme. In other words, a new model can be trained in a short time when waveform parameters and modulation parameters change, which provides a feasible solution for solving real-time problems.

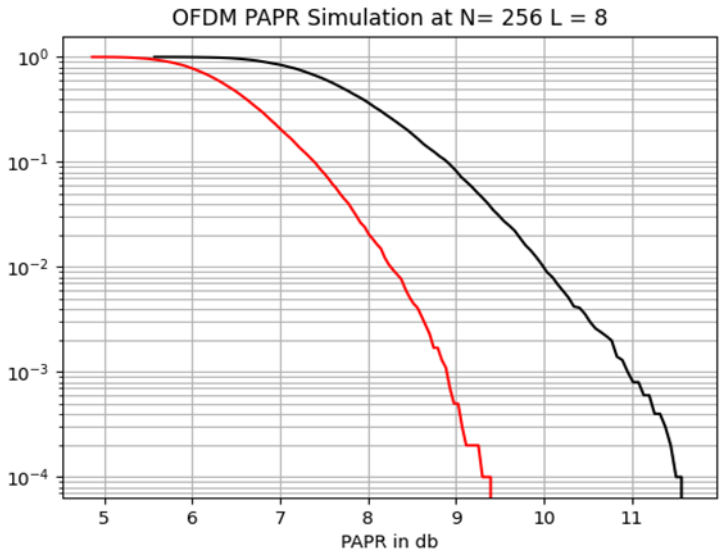


Figure 78 TRnet using tanh [-5,5]

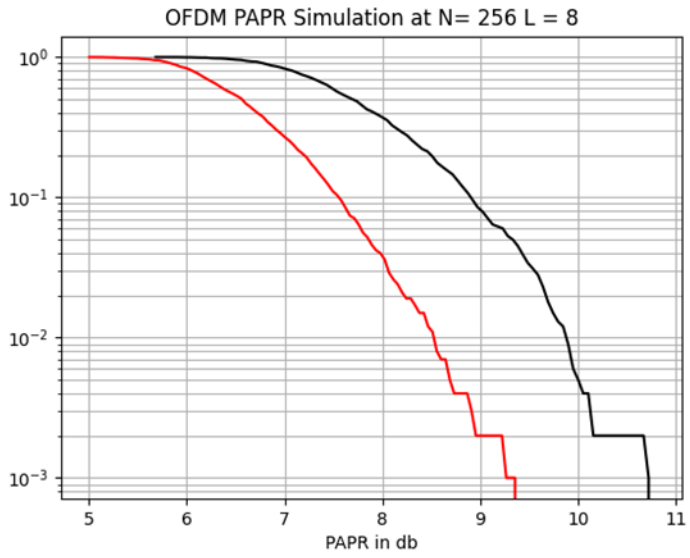


Figure 79 Classical TR

PAPR Reduction Network for OFDM based on Deep Learning

Motivation

We need to propose a solution that performs better in few criteria that classical OFDM PAPR reduction techniques failed at. Such as:

- Computational Complexity
- Overhead
- BER Performance (in case of clipping and filtering).

Idea

Deep learning technique can be adopted to solve the problem of high PAPR in OFDM systems. we investigate the reduction of PAPR by utilizing the specific type of DNN, i.e., an autoencoder, which is usually used for denoising corrupted data. In the proposed scheme, constellation mapping and de-mapping of symbols on each subcarrier in an OFDM system is learned through the training of DNN such that the transmit signal sequence has a low PAPR, while minimizing the degradation of the BER. Moreover, once trained, our proposed scheme can be performed with negligible overhead such that the real-time operation is possible.

This Reduction scheme has been named PRNet by the authors, it's the first attempt to apply a deep autoencoder for the reduction of PAPR in an OFDM system.

The DNN Model

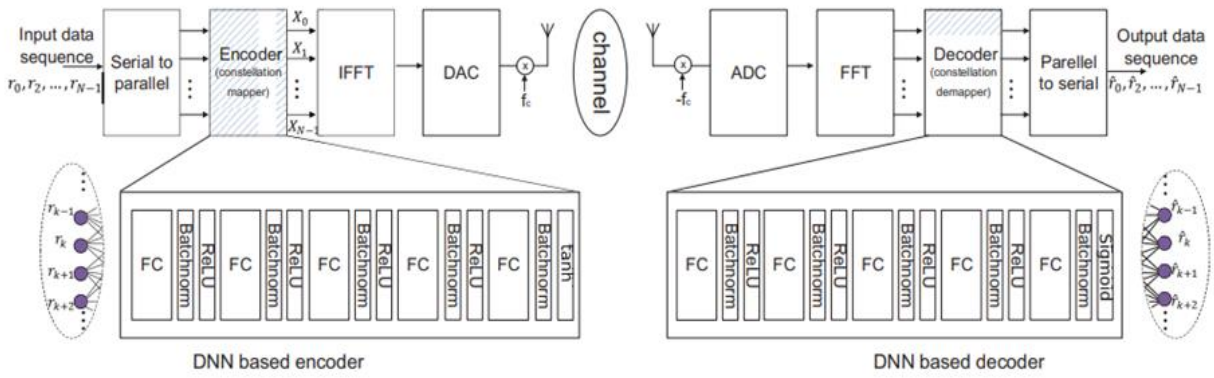


Figure 80 Deep NN block diagram

Our problem is single-in multi-out, there are 2 outputs we need to focus on. The encoder output and the decoder output (Tx and Rx). We have deviated and modified on the proposed scheme, in the original proposition a ReLU activation function was used on the output of the encoder then normalization measures were taken, we found that this hinders the performance of the model since output before IFFT step only can take positive values, which limits the number of possibilities that the network can move freely into. Hence, it limits the constellation that the network can go through. So, a Tanh activation function was used. The problem with tanh is the vanishing gradient problem but will not highly occur in our case. And it limits the output between [-1,1]. while keeping the non-linearity attribute.

Also, we changed the activation function on the output to be Sigmoid not ReLU, we don't know exactly why the author chose ReLU over Sigmoid, but our guess is that they may have input the data label encoded = {0, 1, 2, 3}. But in our approach, we have binary input and output so Sigmoid is the only logical activation function to limit the decoder output.

We input the data as normal binary input of size N*2, this facilitates encoding and decoding so that no categorical encoding is needed to parse the data. But it will be used on a constellation of 4 so it will be reduced to N in the process. Like 4-QAM or QPSK or a different constellation in between, more on that later.

We consider an OFDM system in which bandwidth is divided into N orthogonal subcarriers where an encoded symbol is transmitted on each subcarrier. We assume an independent and identically distributed (i.i.d.) input data sequence is mapped into I-Q constellations, using a constellation mapper that we denote as an encoder.

Joint Loss

To achieve the former objective, the encoder of the PRNet is trained to find the proper constellation mapping from the input data to the output and the decoder of the PRNet must be able to decode the received signal. Then, the proper loss function to achieve this objective can be written as the Binary Cross Entropy (log loss):

$$L_1 = -\frac{1}{K}\sum_{K=1}^{K} y_i . \log\left(p(y_i)\right) + (1 - y_i). \log\,(1 - p(y_i))$$

Where H is the channel effect which is assumed to be flat fading.

Also, the encoder must keep track of decreasing PAPR,

$$L_2 = PAPR\{IFFT(output_{encoder})\}$$

$$PAPR\{x[n]\} = \frac{\max_{0 \le n \le N-1} x[n]^2}{E\,[|x[n]|^2\,]}$$

So, the model ultimately will try to decrease the joint loss which will be:

$$L_{joint} = L_1 + \lambda L_2$$

Here, λ denotes the weight parameter that determines which loss, i.e., L_1 or L_2, is dominant. For example, when the value of λ is small, the autoencoder is trained to improve the BER but to put less effort into reducing the PAPR, and vice versa.

Hyperparameters

We set the number of nodes in each hidden layer to be 2048 using trial and error tuning, we used RMSprop optimizer with learning rate L = 0.0001.

We also used accuracy metrics with the 2nd output to be keras binary accuracy to depict bit error rate (or success rate to be precise) to better get an idea how well the model is converging. Its simpler to look at metrics rather than loss numbers that doesn't exactly mean real world accuracy.

Number of epochs is function of lambda and learning rate and SNR, so we figured out its best to use a bigger number and using early stopping callback to monitor validation loss and overfitting.

Channel and Noise Simulation

The output of the encoder is the signal encoded on a constellation predicted by the model to have as low PAPR as possible, then Inverse Fourier Transformed to Time Slots Signal.

By Stating the SNR of the system, we can determine the noise power and hence the noise voltage through the following equations.

Additive White Gaussian Noise

$$P_{avg\ noise} = \frac{P_{avg\ signal}}{10^{\frac{SNR}{10}}} = = \frac{\frac{1}{N}\sum_N |IFFT(output_{encoder})|^2}{10^{\frac{SNR}{10}}}$$

$$X_{noise\ in\ voltage} = Normal\ Random\ Variable + 1\text{j} * Normal\ Random\ Variable$$

$$= X_{real}\left(\mu = 0\ , \sigma 2 = \sqrt{\frac{P_{avg\ noise}}{2}}\right) + 1j * X_{imag}\left(\mu = 0\ , \sigma 2 = \sqrt{\frac{P_{avg\ noise}}{2}}\right)$$

Flat Fading Channel

$$H = \frac{1}{2} * Rayleigh\ Distributed\ Random\ Variable \text{ of (scale} = 1)$$

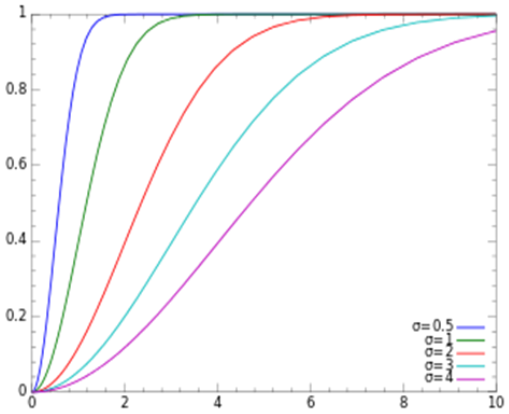


Figure 82 CDF

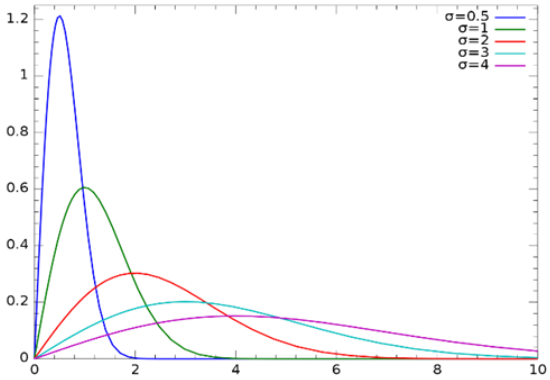


Figure 81 PDF

Channel and Noise Lambda Layer Code Snippet

```
def Channel_and_Noise(tensor):
        signal_avg = tf.reduce_mean(tf.square(tf.abs(tensor)))
        noise_avg = signal_avg / 10**(SNR/10)
        rayleigh = (1.0/2.0)*tfp.random.rayleigh((batchsize, 1))
        noise_real = (tf.sqrt(noise_avg/2.0))*tf.random.normal((batchsize , N))
        noise_imag = (tf.sqrt(noise_avg/2.0))*tf.random.normal((batchsize , N))
        #Corruption = Channel Effect + AWGN
        corruption =  tf.complex(tf.divide(noise_real, rayleigh) ,
                                 tf.divide(noise_imag, rayleigh))
        #Equalizing done on Signal
        x = tensor + corruption
        #Recieving
        x_fft = tf.signal.fft(x)
        return tf.keras.layers.Concatenate(axis = -1) ([tf.math.real(x_fft),
                                                        tf.math.imag(x_fft)])
```

Rayleigh Channel + AWGN "Simulated" Effect on BER

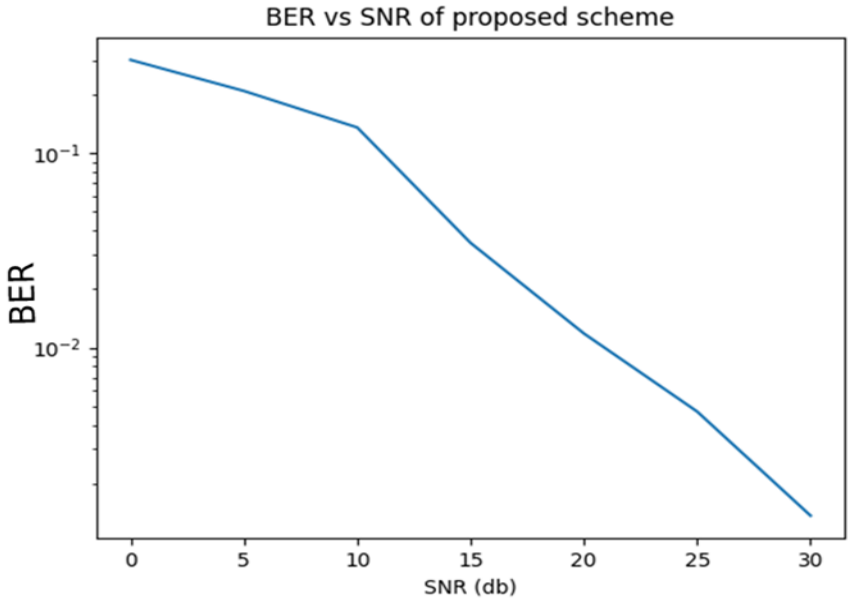


Figure 83 Decoder Predicted Output Error Rate

Rayleigh Channel + AWGN "Theoretical" Effect on BER

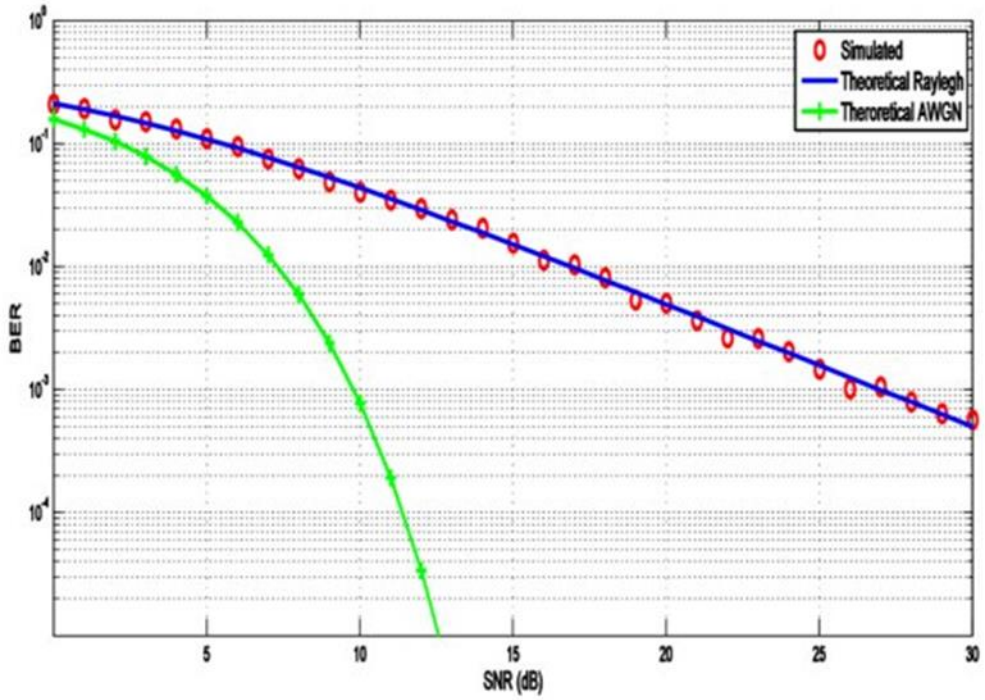


Figure 84 Rayleigh channel effect on QPSK

Training Dataset

The model is compiled as a multi-output model with Losses = [encoder: PAPR, decoder: Binary-Cross entropy] which somehow indicates the model performance on PAPR and BER at the receiver. Loss weights as = [Lambda, 1] and metrics to be binary accuracy.

We had a dataset of randomly generated binary bits of size (N*2), number of symbols in the hundreds of thousands. We then feed them to the network at batches, each batch consists of 400 symbols. Because we have an autoencoder approach we feed those as the (x and y) of the training. Because we are trying to match the decoder output to the input to receive the data as flawless as possible.

We use the shuffle property so there is a sense of randomness in the input batch each propagation. We will train for 400 epoch and set the callback to stop the learning when validation loss stops improving.

Training was done on an intel processor I7-10700K, 32GB of memory and RTX 3070 with TensorFlow-GPU to accelerate matrix operations.

Computational Performance Evaluation

Upon predicting 100,000 symbols on a fully trained model, we found the following:

	N = 64 (100K Symbols)	**N= 64 (Each Symbol)**
Simulation Time	1.5291564 Seconds	15.292 μSeconds

Training Results

TOTAL LOSS	ENCODER LOSS	DECODER LOSS	DECODER BINARY ACCURACY
0.045609	1.87045	0.0082	0.9996

Our Custom Model PAPR Simulation

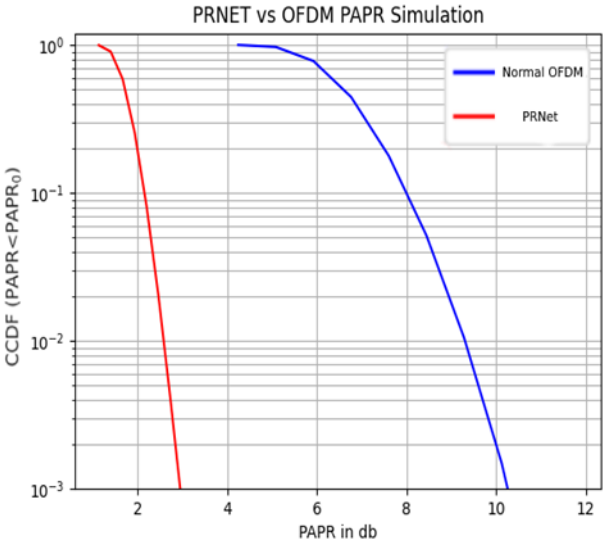


Figure 86 Our PRNet Model Performance

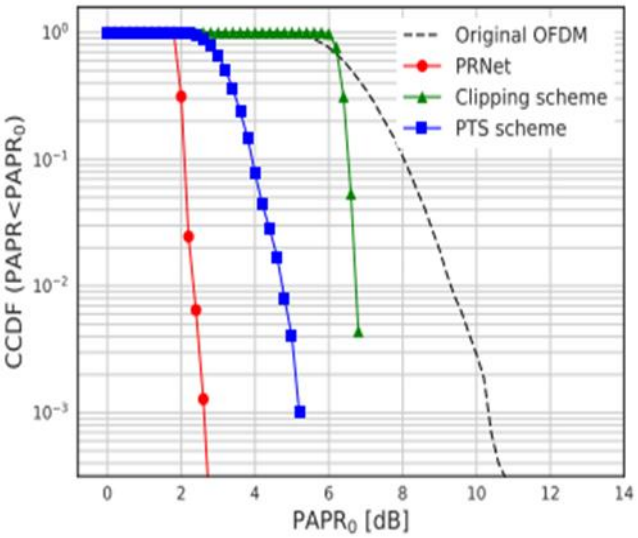
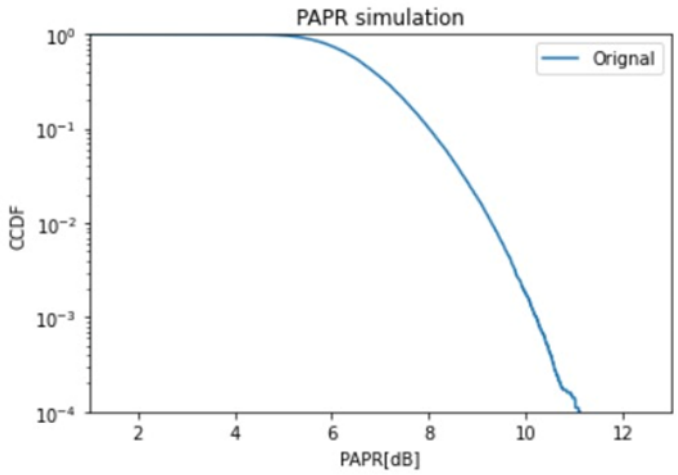


Figure 85 Proposed IEEE Performance

CHAPTER III – Simulation Results

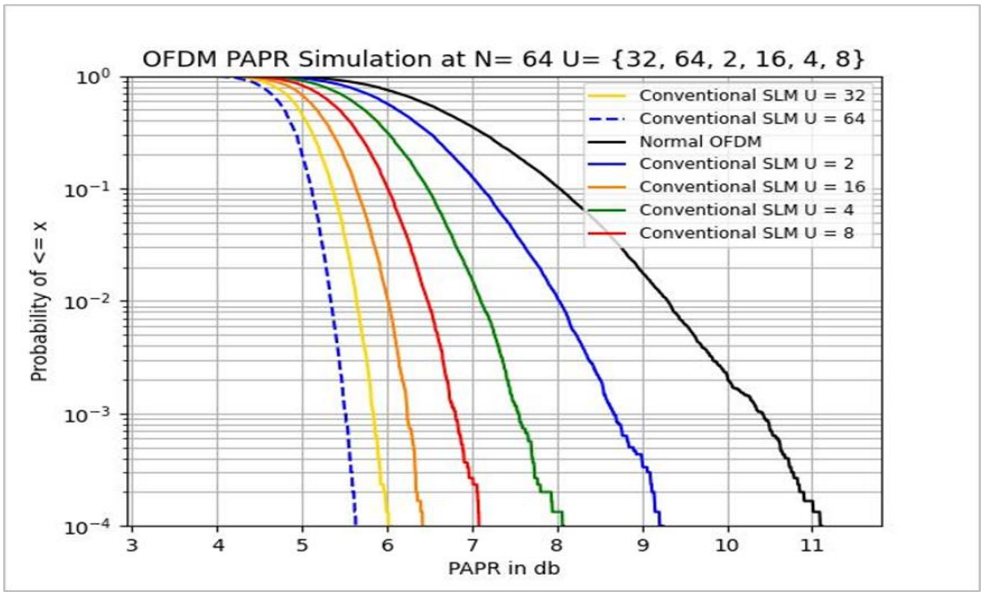
OFDM system without PAPR Reduction



- 64 Subcarriers
- 10,000 Symbols
- 4-QAM modulation

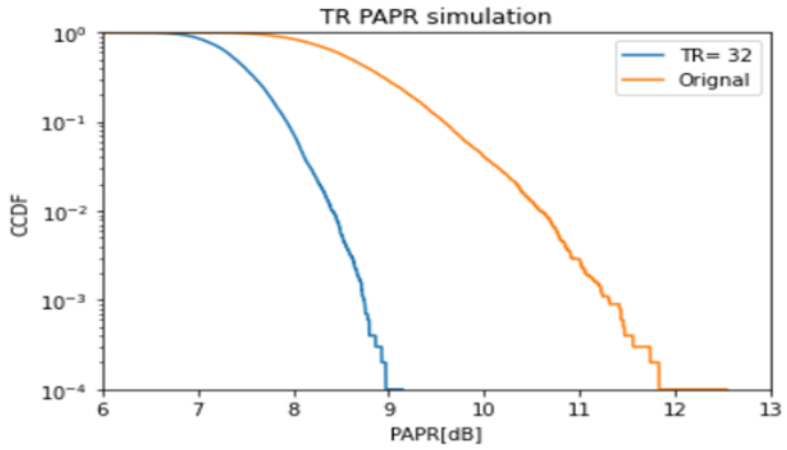
Classical techniques

SLM



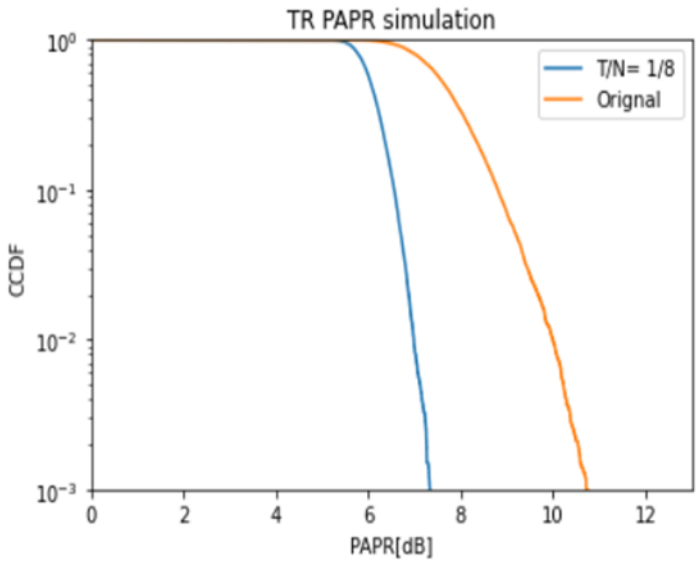
- 64 Subcarriers
- 30,000 symbols
- Number of phase sequences = (2, 4, 8, 16, 32, 64)
- 4-QAM modulation

Tone reservation (Kernel)



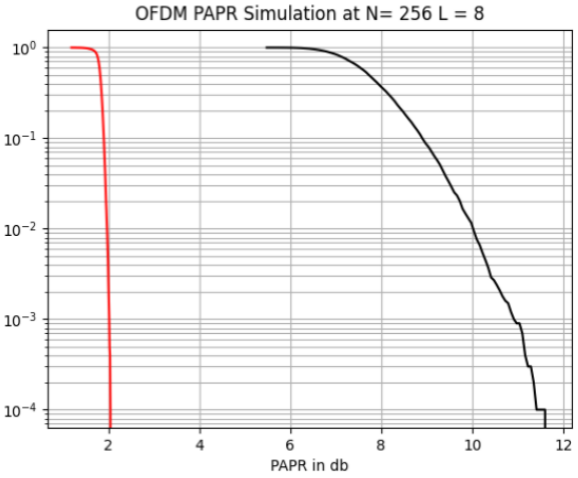
- 1024 Subcarriers
- 32 Reserved tones
- 20 iterations
- 10,000 symbols
- Random set optimization for PRTs

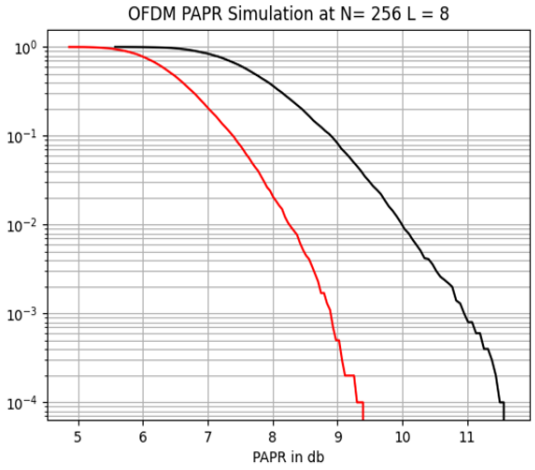
Tone reservation (S-SCR)



Deep Learning Techniques

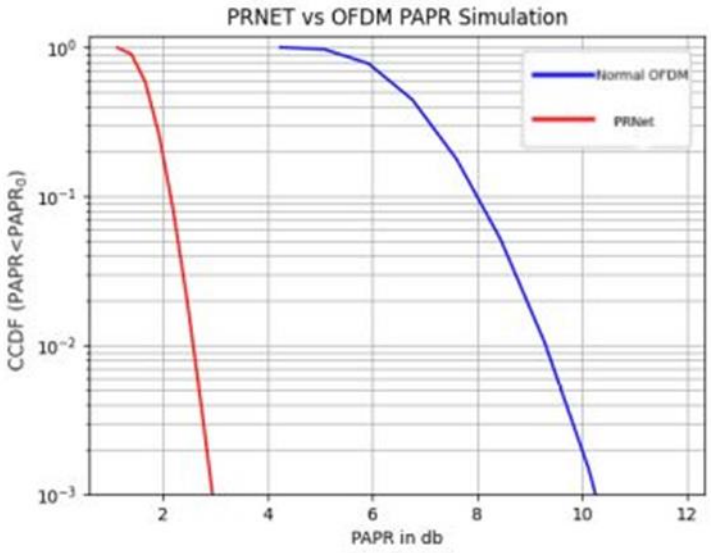
TRNet





- 256 Subcarriers.
- 8 Reserved Tones.
- 4-QAM modulation.
- Learning rate = 0.001 , 90 epochs.
- Execution time for 10,000 symbols is around 13 seconds.
- On the Left no activation function at the output layer.
- On the Right custom activation function is used.

PRNet



- 64 Subcarriers
- Joint Loss coefficient $\lambda = 0.002$
- Execution Time 1.52 seconds for 100,000 symbols.
- Execution Time 15.29 μSeconds per symbol.
- Training was done on an intel processor I7-10700K, 32GB of memory and RTX 3070 with TensorFlow-GPU to accelerate matrix operations.

CHAPTER IV – Conclusion and Future Work

Conclusion

We developed two models based on deep learning for PAPR reduction of OFDM systems. First one is named **TRNet** that adaptively generates the peak-canceling signal according to the characteristics of the input signal. The simulation results showed that the proposed scheme gives slightly better performance in PAPR reduction compared to the conventional TR technique and it outperforms it in terms of execution time.

The second model is named **PRNet** that uses the autoencoder architecture of deep learning. Our proposed scheme can reduce the PAPR while maintaining BER. Through simulations, we showed that our proposed scheme significantly outperforms conventional schemes in view of both PAPR and BER.

Future Works

We will consider solving the high signal power problem that heavily affected the TRNet model.

As Neural Networks are in current progress. It would be also interesting to develop faster training method for the PRNet for larger OFDM systems with more subcarriers and higher modulation order.

CHAPTER V – Appendices

In many cases radio frequency power is measured as a simple numeric value like (-40 dbm) , but some instruments as spectrum analyzer and some power sensors have the ability to display statistical power measurements that statistical power measurements can show us the probability of how that the measured power can take on a certain value or how that the values of measured power is distributed.

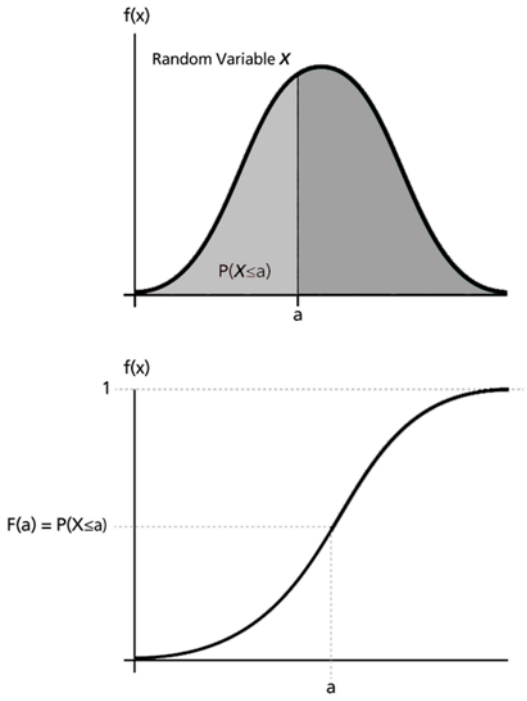
Mainly we have three statistical power measurements.

PDF → probability density function.

CDF → cumulative distribution function.

CCDF → complementary cumulative distribution function.

PDF



- It shows the relative probability that the measured power takes on a given value.
- The data we have from PDF similar to the one we get from histogram (it's a graphical representation organizes a group of data in range specified by the user) but the data we get from PDF is continuous not discrete like histogram.
- PDF can help us find out the probability of a measured power lies between. two limits by integrating on this interval.

- If we integrated on the whole PDF, we get CDF.

CDF :

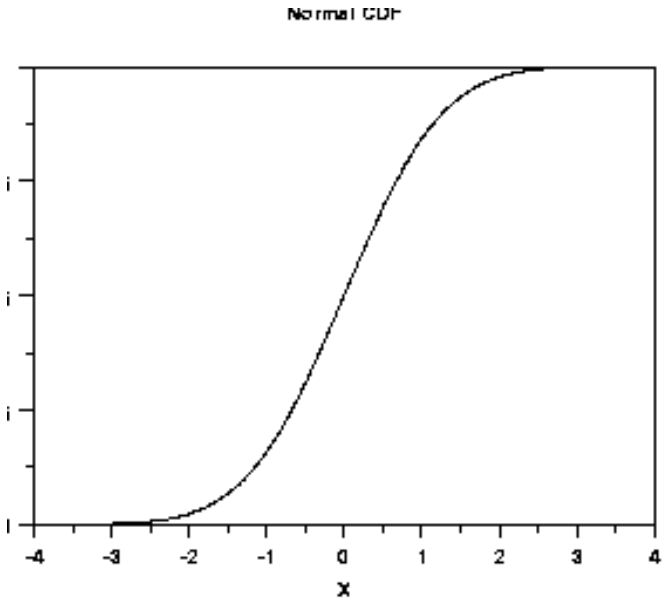


Figure 87 CDF distribution

- If we take any random point on curve, we will notice that CDF emphasizes minimum power values, in other words CDF is useful if you want to know how percentage of time signal's power is or below certain value
- Using CDF, we can compute CCDF where $CCDF = 1 - CDF$.

CCDF:

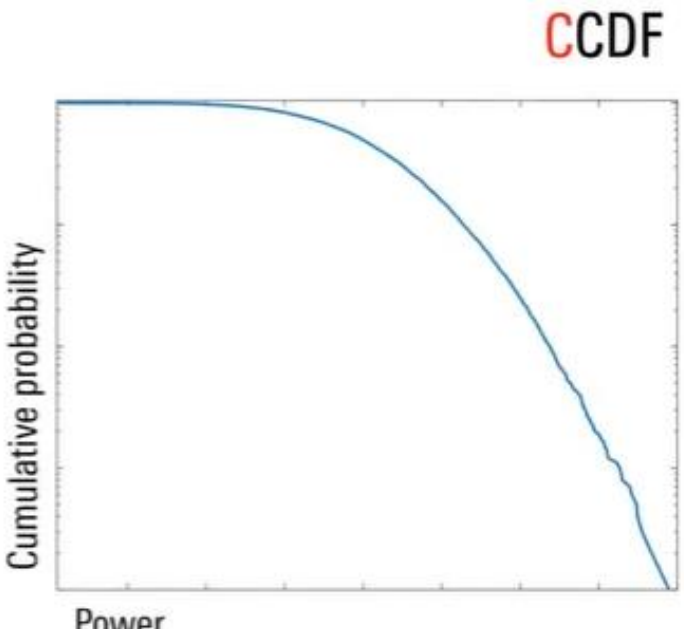


Figure 88 CCDF distribution

- on the contrary CCDF emphasizes maximum power values in other words CCDF is useful if you want to know how much percentage of time the signal's power reaches or exceeds certain value.
- observing CCDF curve, we will notice that

- the Y-axis contains the cumulative probabilities which is mostly plotted on log-scale (ex: -10 dBm←→20dBm)

- the X-axis. however, can be plotted in two ways. the first one is simply by plotting the absolute power values, and the second one is by plotting the relative power values so that the origin of graph is defined to be average power or (0 dB) and the values on the axis itself will be in dB above the average power

- the more the CCDF curve tilts to the right the more that the signal takes on values that are significantly higher than the average power

Example:

the signal with an amplitude of more or less constant over time will appear in a CCDF curve as an almost vertical line but the more that the signal takes great amplitude variations the trace will tilt more and more to the right causing that ration between the peak power to the average power (peak to average power ratio) to increase which leads to more challenges to device designers because signals with high PAPR requires things like [high dynamic range – analog to digital converter or highly linear amplifiers with peak power capability]

All of that demonstrate how important CCDF is because it shows us how often a giver peak power is reached or exceeded

- Not only that CCDF shows us how often a giver peak power is reached or exceeded but also helps us with other things like computing PAPR to modulated signal types because it's impossible to compute that mathematically using only signal's parameters

Note that:

even the signals with the same modulation can have a different CCDF curve so in order to design a device like a filter or an amplifier we need to measure CCDF curve for expected input signal as well as the signal used in testing and debugging

- Also, CCDF can be used to quantify device behavior by measuring the CCDF to device's input and output and make notes of any changes.

CHAPTER VI – References

References

[1] A. N. S. a. B. B. Chadha, Orthogonal frequency division multiplexing and its applications, arXiv preprint, 2013.

[2] Y. S. Cho, J. Kim, W. Y. Yang and C. G. Kang, MIMO-OFDM Wireless Communications with MATLAB®, Singapore: Wiley-IEEE Press, 2010.

[3] [Online]. Available: https://crimsonpublishers.com/rdms/fulltext/RDMS.000686.php .

[4] [Online]. Available: https://drive.google.com/drive/folders/1JECwZtMUq19K3MlQJEHvm1_j_b7toNLK?fbclid=IwAR2eFElJ-QSMAwZGsR17OZ3RQ5cx_SyA_rFWEW0F4-qI5ZlHHXiQ4bcDS94.

[5] A. F. Molisch., Wireless communications, California, USA: A John Wiley and Sons, Ltd., Publication, 2011.

[6] [Online]. Available: https://drive.google.com/drive/folders/1OefLVRsCxqe-3b5qUjyu-OR_GLKC6A7H?fbclid=IwAR2eFElJ-QSMAwZGsR17OZ3RQ5cx_SyA_rFWEW0F4-qI5ZlHHXiQ4bcDS94.

[7] [Online]. Available: https://drive.google.com/drive/folders/1V6MFGtPyDyaky7pY7Nc30abM1_NDeC-V?fbclid=IwAR2eFElJ-QSMAwZGsR17OZ3RQ5cx_SyA_rFWEW0F4-qI5ZlHHXiQ4bcDS94.

[8] G. S. YE (GEOFFREY) LI, ORTHOGONAL FREQUENCY DIVISION MULTIPLEXING FOR WIRELESS COMMUNICATIONS, New York: Springer Science+Business Media, Inc. , 2006.

[9] M. M. M. M. M. H. S. K. B. Snikdho Sworov Haquel, "An Algorithm for P APR Reduction by SLM Technique in OFDM with Hadamard Matrix Row Factor," Bangladesh, 2015.

[10] S.-R. Y. Y. K. P. P. J. Sung-Eun Park, "Tone Reservation method for PAPR Reduction scheme," IEEE Standards publication, Korea , 2003.

[11] J.-C. C. a. Chih-PengLi, "Tone Reservation Using Near-Optimal PeakReduction Tone Set Selection Algorithm forPAPR Reduction in OFDM Systems," *IEEE SIGNAL PROCESSING LETTERS*, vol. 17, no. PAPR REDUCTION IN OFDM SYSTEMS, pp. 3-4, 2010.

[12] X. L. a. W. W. Jingqi Wang, "SCR-based Tone Reservation Schemes With Fast Convergence for PAPR Reduction in OFDM System," IEEE Wirless Communication , 2018.

[13] R. He, Applications of Machine Learning in Wireless Communications, London: Institution of Engineering and Technology, 2019.

[14] S. S. R. Z. Vasilev, Python Deep Learning, Birmingham: Packt, 2019.

[15] "My Great Learning," [Online]. Available: https://www.mygreatlearning.com/blog/activation-functions/. [Accessed July 2022].

[16] T. A. e. team, "towards ai," Towards AI, 19 October 2020. [Online]. Available: https://pub.towardsai.net/basic-linear-algebra-for-deep-learning-and-machine-learning-ml-python-tutorial-444e23db3e9e. [Accessed July 2022].

[17] "Dive into Deep Learning," [Online]. Available: https://d2l.ai/chapter_preliminaries/linear-algebra.html.

[18] "programmathically," 21 September 2021. [Online]. Available: https://programmathically.com/an-introduction-to-neural-network-loss-functions/#:~:text=The%20loss%20function%20in%20a,all%20losses%20constitutes%20the%20cost.. [Accessed july 2022].

[19] "IBM," IBM Cloud Education, 27 October 2020. [Online]. Available: https://www.ibm.com/cloud/learn/gradient-descent#:~:text=Gradient%20descent%20is%20an%20optimization,each%20iteration%20of%20parameter%20updates.. [Accessed July 2022].

[20] Q. S. M. J. Benwei Wang, "A Novel Tone Reservation Scheme Based on Deep Learning for PAPR," *IEEE COMMUNICATIONS LETTERS*, vol. 24, no. 6, pp. 1271 - 1274, 2020.

[21] N. A. K. M. S. M. K. S. H. B. Z. A. K. Nadeem Javaid, "Ubiquitous HealthCare in Wireless Body Area Networks - A Survey," *Research Gate,* 2013.

[22] R. K. Masaya OHTA, "Asymmetric Autoencoder for PAPR Reduction of OFDM signals," *IEICE Communications Express,* pp. 1-7, 2022.