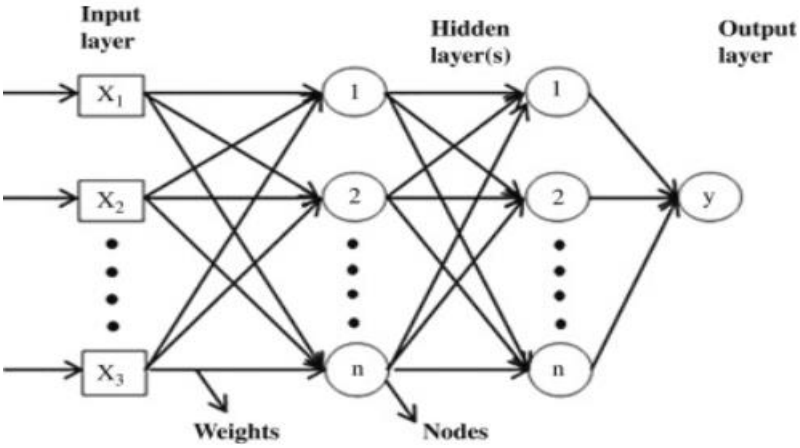
INTRODUCTION

- **Objective**
  The objective of this project is to decode an encrypted signal having mi symbols using an ANN model which has been trained accordingly, when prior probabilities of message symbols are unknown.

- **Abstract**
  A signal is typically sampled then modulated through PCM (pulse coded modulation, before being sent for encoding. This is an effort to ensure integrity, accuracy, and fault-tolerance in transmitted data, as channel noise invariably distorts the source message. The work of the decoder at the receiver end is to decode the encrypted (encoded) signal before proceeding further. Techniques have been developed to decode encoded PCM signals containing m_i symbols (where $M=2^{L \text{ bits}}$). In our case, Artificial Neural Networks have been used because of their adaptive learning, self-organization, and real time operation.

- **ANN MODEL (Artificial Neural Network model)**

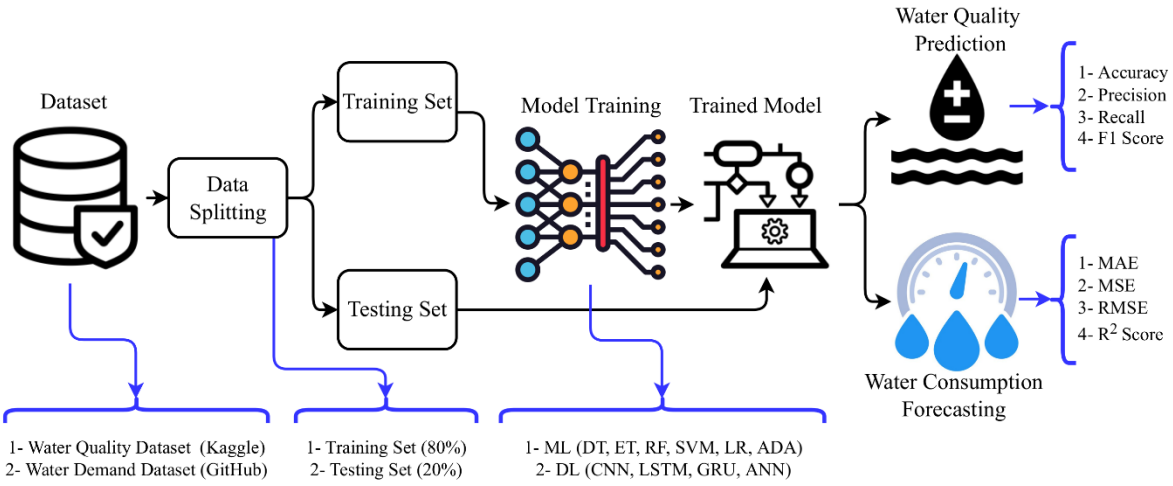
Artificial Neural Networks (ANNs) are a type of machine learning model that is inspired by the structure and function of the human brain.
The basic building block of an ANN is a neuron, which receives input signals, performs a weighted sum of the inputs, applies an activation function, and produces an output. Multiple neurons are organized in layers, typically including an input layer, one or more hidden layers, and an output layer.



Each neuron in a layer is connected to neurons in adjacent layers through weighted connections, and the weights are adjusted during training to optimize the performance of the network.

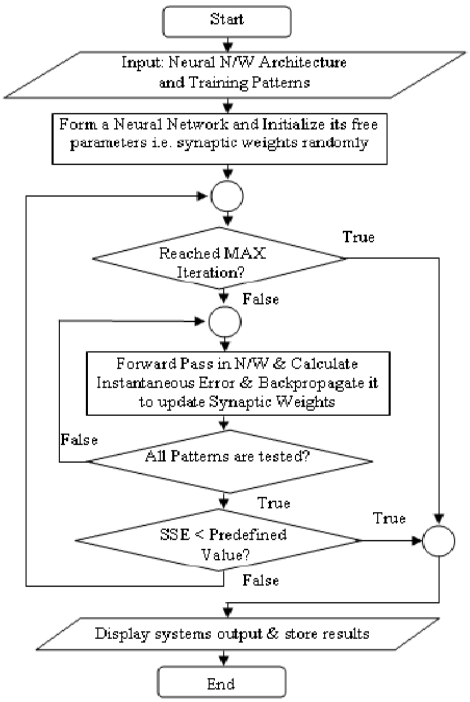
Training an ANN model

ANNs are trained using labelled data, where the input data is paired with corresponding output labels or targets. During training, the ANN learns to adjust the weights of the connections based on the errors between its predicted outputs and the actual targets, using a process called backpropagation. Once trained, ANNs can be used to make predictions or decisions on new, unseen data.



Training an ANN model involves the following steps:

1. Data Preparation: preparing the data, like collecting, cleaning, and organizing the data so that it can be fed into the model.
2. Initializing the Weights: The weights are initialized randomly at the beginning of the training process.
3. Forward Propagation: During forward propagation, the input data is passed through the layers of the ANN, and the output is generated. This output is compared for errors.
4. Backward Propagation: the error is propagated back through the layers of the ANN, and the weights are updated to reduce the error.
5. Iteration: Steps 3 and 4 are repeated multiple times until the error is minimized

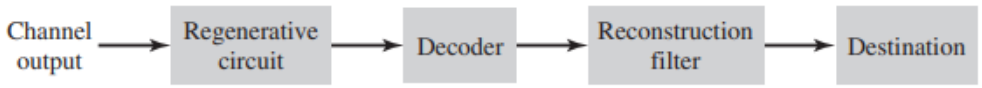


6. Testing: Once the ANN model is trained, it is tested on a separate set of data to evaluate its performance. This step lets us know whether the ANN model has been 'trained 'well.

Theory and Analysis

Decoding

The basic operations in the receiver are regeneration of impaired signals, decoding, and reconstruction of the train of quantized samples, as shown in Fig. 5.11(c).



(c)

FIGURE 5.11 The basic elements of a PCM system: (*a*) Transmitter, (*b*) transmission path, connecting the transmitter to the receiver, and (*c*) receiver.

After regeneration, the clean pulses are regrouped into code words and decoded, or mapped back into a quantized PAM signal. Some of the particular decoding algorithms to look into are the Maximum Likelihood Decoding algorithm (ML decoding) and Maximum a-Posteriori Decoding algorithm (MAP decoding), which intensively uses decision rules to create 'boundaries' to distinguish a 1 from a 0 from a AWGN channeled signal.

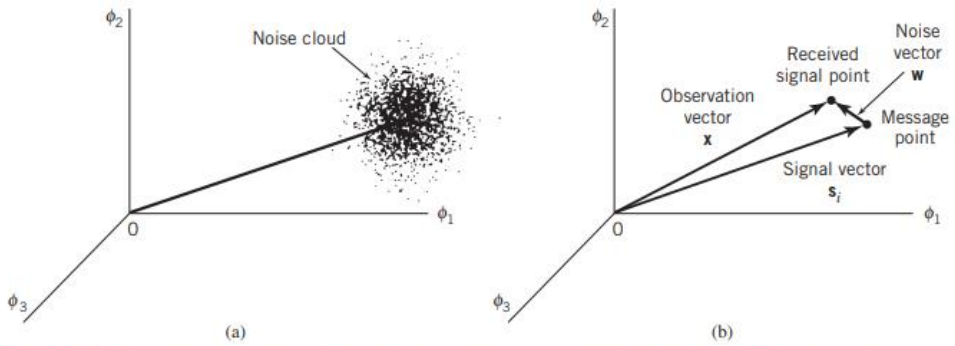


Figure 7.6 Illustrating the effect of (a) noise perturbation on (b) the location of the received signal point.

At the receiver, we are given the observation vector **x** and the requirement is to estimate the message symbol mi that is responsible for generating x. To emphasise this viewpoint, we go back to likelihood function

$$l(m_i) = f_X(\mathbf{x}|m_i), \qquad i = 1, 2, \ldots, M$$

Supposing that, in each time slot of T seconds, one of the M possible signals is transmitted with equal probability, 1/M. We should now come up with a decision rule to classify the obtained **x** to an estimate mi, that would minimize probability of error.

Given the observation vector $\mathbf{x}$, suppose that we make the decision $\hat{m} = m_i$. The probability of error in this decision, which we denote by $P_e(m_i|\mathbf{x})$, is simply

$$P_e(m_i|\mathbf{x}) = 1 - \mathbb{P}(m_i \text{ sent}|\mathbf{x}) \tag{7.47}$$

We set the optimum decision rule as:

Set $\hat{m} = m_i$ if

$$\mathbb{P}(m_i|\text{sent}|\mathbf{x}) \geq \mathbb{P}(m_k|\text{sent}|\mathbf{x}) \qquad \text{for all } k \neq i \text{ and } k = 1, 2, \ldots, M.$$

We see this as the decision rule employed as part of MAP decoding, where,

$$P\{m_i|x\} = \frac{f_x(x|m_i)P\{m_i\}}{f_x(x)}$$

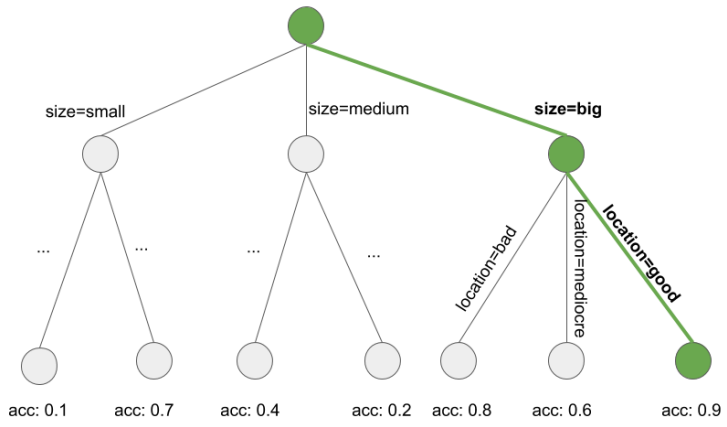
With P{mi} being the 'prior probability of symbol mi'.

In the case of ML decoding, all M symbols are equally likely to occur, so $P\{mi\} = \frac{1}{M}$

This leads to our optimum decision rule for ML decoding algorithm,

$$f_x(x|m_i) \geq f_x(x|m_k), \quad 1 \leq k \leq M$$

The problem statement in our project states that no prior probabilities of message symbols are given. This suggests that reaching the solution by conventional method of ML and MAP decoding won't be an easy task. Hence, we rely on Artificial Neural Network (ANN) modeling to learn from a given dataset (training) and test itself against the problem we shall pose to it.



Using ANN for decoding

In the context of decoding with an ANN model, the model is trained on labelled data, where the input encoded signals are associated with known output symbols.
During training, the ANN learns the relationship between the input signals and output symbols, and adjusts its parameters (weights and biases) to minimize the error between predicted and actual output symbols. Once the ANN model is trained, it can be used for decoding new input signals by applying the learned mapping from inputs to outputs.

The advantages of using an ANN model over FIR filters for decoding include:

1. Adaptability to unknown probabilistic characteristics: ANNs can learn from data and adapt their parameters to optimize performance based on the available training data, making them more adaptable to unknown prior probabilities of message symbols compared to fixed FIR filters.

2. Non-linearity: ANNs can capture complex, non-linear relationships between input signals and output symbols, which may be difficult for linear FIR filters to model.

3. Flexibility: ANNs can be designed with varying architectures and sizes, allowing for flexibility in capturing different types of input-output relationships and accommodating different decoding requirements.

4. Robustness to noise: ANNs can be trained to be robust to noise and distortions in the input signals.

5. Scalability: ANNs can be trained with large datasets, allowing for scalability to handle large amounts of data and complex decoding tasks.

However, there are also some considerations when using ANN models for decoding, such as

- the need for large amounts of labelled training data,
- the potential for overfitting, and
- the computational complexity of training and inference.

Proper training, validation, and testing of ANN models are important to ensure their accuracy and reliability in decoding applications.

In summary, ANN models can offer advantages in decoding applications when the prior probabilities of message symbols are unknown, as they have the ability to learn from data and adapt to probabilistic characteristics. However, careful consideration and validation of the model's performance and requirements are necessary to ensure accurate and reliable decoding results.

DESIGN/ ANALYSIS

To build an Artificial Neural Network (ANN) model for decoding a PCM signal without prior probabilities of message symbols, we shall use a type of ANN called “Multilayer Perceptron" (MLP) using an algorithm called "Backpropagation” for our approach. The steps are as follows:

1. Data pre-processing: Convert the PCM signal to a binary sequence.

2. Splitting data: Split the binary sequence into training and testing sets.

3. Define the network architecture: A Multilayer Perceptron (MLP) with one hidden layer can be used to decode the PCM signal.

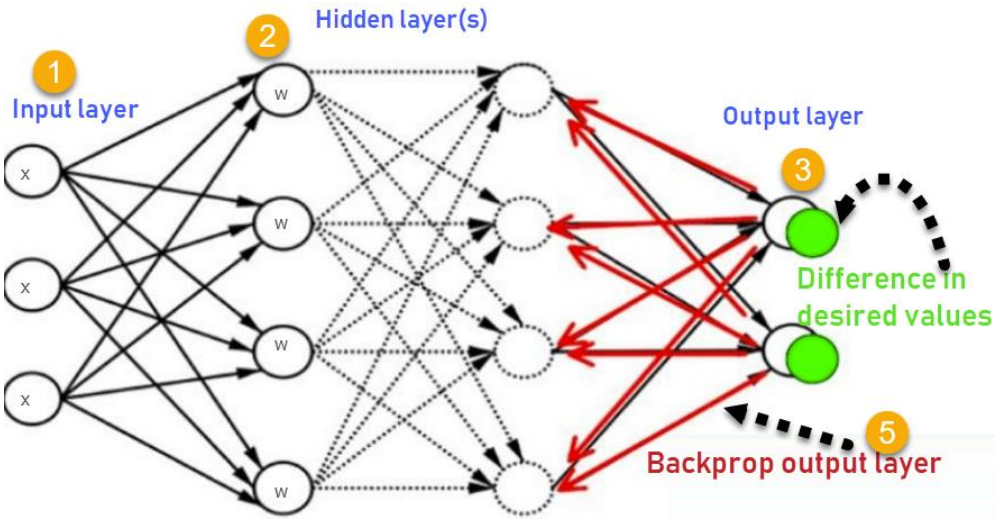
4. Initialize the weights and biases: The weights and biases of the MLP are initialized with small random values.

5. Forward pass: Feed the training data through the network and compute the output.

6. Backward pass: Calculate the error between the predicted output and the actual output, and use the Backpropagation algorithm to adjust the weights and biases to minimize the error.

7. Test the model: Use the testing set to evaluate the performance of the model.

8. Fine-tuning: If the performance is not satisfactory, fine-tune the architecture and repeat the training process



DESIGN USING ANN MODEL

First we Construct a Signal using Binary PCM Signal using Polar encoding Scheme.

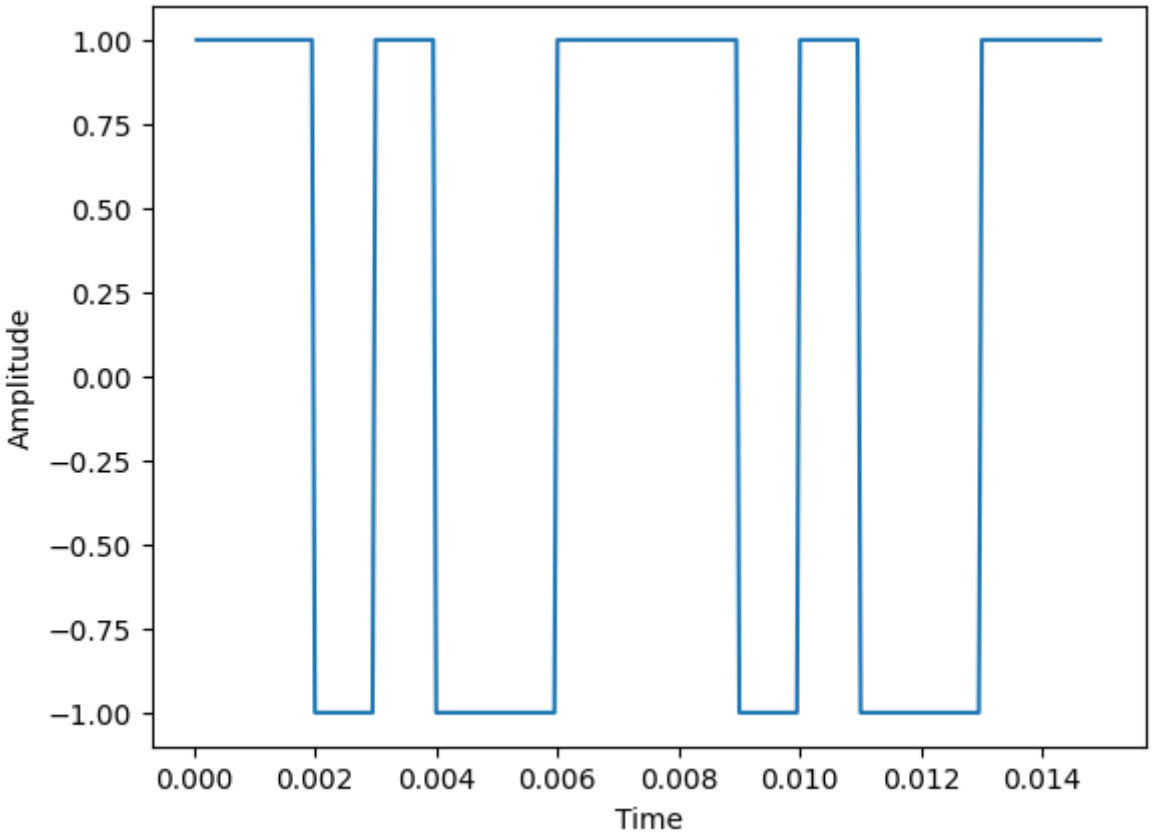
We Define a 1KHz PCM Signal with ones and Zeros with a bit duration of 1ms

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Signal Specifications
sym_length = 20 #Length of each symbols
n_sym = 100 #No of Symbols
fc = 5e03 #Frequency of carrier
Tb = 1e-03 #bit duration
Fs = 4*fc #Sampling Frequency
Ts = 1/Fs #Sampling period

# Signal Bm
rand_n = np.random.randint(2,size=n_sym)
print(rand_n)
```

```
Bm = [1 1 0 1 0 0 1 1 1 0 1 0 0 1 1 1 1 0 1 0 1 1 0 0 0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 0 1 1 0 0 1 0 0 0 1 1 0 0 0
1 1 1 0 1 1 0 1 1 1 1 0 0 1 0 1 0 0 0 1 0 0 1 0 1 1 0 0 1 0 0 0 0]
```

We then Plot a Polar Signal for the Same

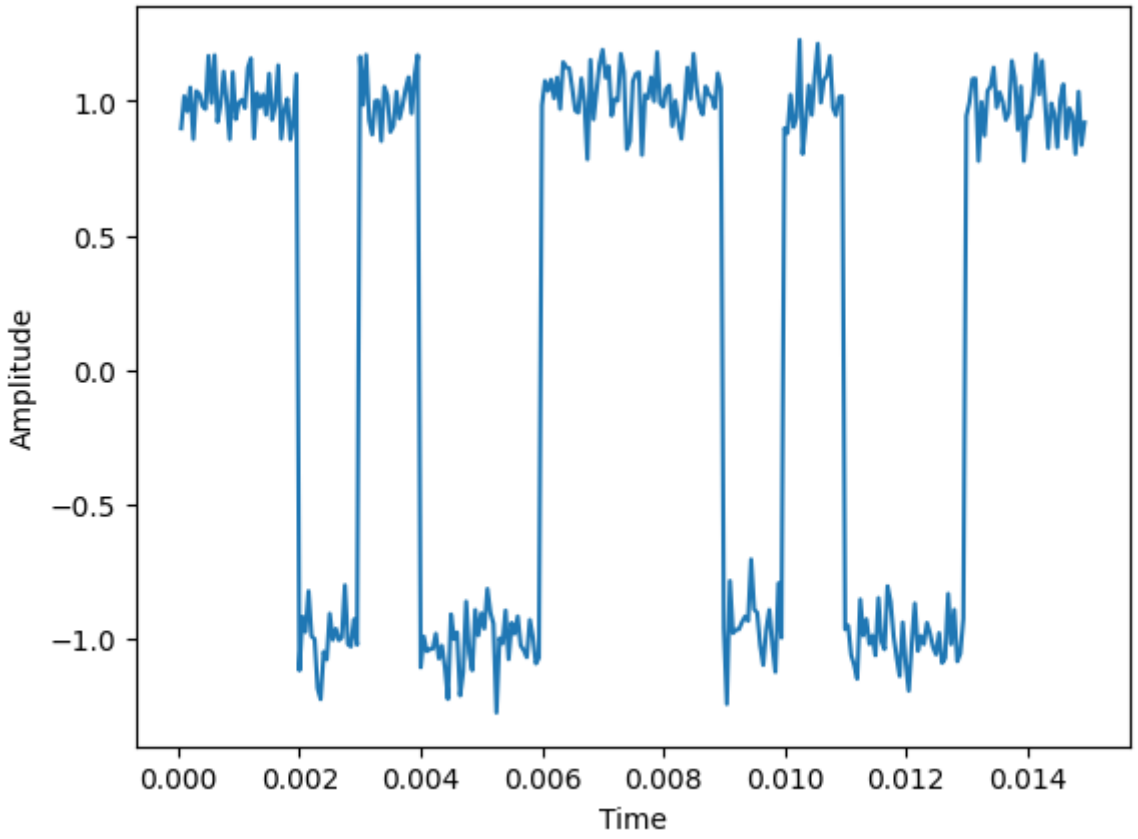
```
# generate a signal
sig = np.zeros(sym_length * n_sym)
id_n = np.where(rand_n == 1)

for i in id_n[0]:
    temp = int(i * sym_length)
    sig[temp:temp + sym_length] = 1

id_n = np.where(rand_n == 0)
for i in id_n[0]:
    temp = int(i * sym_length)
    sig[temp:temp + sym_length] = -1

# Time Axis
t = np.arange(0, n_sym*Tb, Ts)
print(len(t),len(sig))

plt.figure()
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.plot(t[1:100],sig[1:100]
```



This Signal is Transmitted over a AWGN Channel

Simulating The Received Noisy Signal with some given SNR

```
# Adding Noise
# Set a target SNR
target_snr_db = 20
# Calculate signal power and convert to dB
sig_avg_watts = np.mean(sig**2)
sig_avg_db = 10 * np.log10(sig_avg_watts)
# Calculate noise according to [2] then convert to watts
noise_avg_db = sig_avg_db - target_snr_db
noise_avg_watts = 10 ** (noise_avg_db / 10)
# Generate an sample of white noise
mean_noise = 0
noise = np.random.normal(mean_noise, np.sqrt(noise_avg_watts), len(sig**2))
sig = sig + noise

# plot noisy signal
plt.figure()
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.plot(t[1:300],sig[1:300])
```



We need the following Steps to Construct a ANN Model For Symbol Detection

1. Gather and preprocess data: Collect a dataset of binary PCM signals and preprocess the data by converting it into a numerical format that the ANN model can process.
2. Define the architecture of the ANN model: Choose an appropriate architecture for the ANN model, such as the number of input and output nodes, hidden layers, and activation functions.
3. Train the ANN model: Use the preprocessed dataset to train the ANN model. During training, the model will learn to recognize patterns in the data and make accurate predictions about whether a binary PCM signal is present or not.
4. Evaluate the model: Once the model is trained, evaluate its performance using a separate dataset. This will help you determine the accuracy of the model and identify any areas where it may need improvement.
5. Test the model: Finally, test the model on new, unseen data to see how well it performs in real-world situations. This will help you determine whether the model is accurate enough to be used in practical applications.

```
# creating a dataset
new_n = []
for i in range(len(rand_n)):
    for j in range(sym_length):
        new_n.append(rand_n[i])

signals = sig
labels = new_n
```

```
# Convert the dataset to a pandas DataFrame and save it to a CSV file
dataset = pd.DataFrame({'Signal': signals, 'Symbol': labels})
dataset.to_csv('noisy_signals.csv', index=False)

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense, Dropout

# Load the dataset containing the PCM signals and symbols
dataset = pd.read_csv('noisy_signals.csv')

# Split the dataset into input features (PCM signals) and output targets (sym
bols)
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rand
om_state=0)

# Standardize the input features (mean=0, std=1)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Define the ANN model
model = Sequential()
model.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dropout(0.2))
model.add(Dense(units=32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(units=1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accurac
y'])

# Train the model
model.fit(X_train, y_train, batch_size=32, epochs=50, validation_data=(X_test
, y_test))

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print('Test accuracy:', accuracy)

# Use the model to predict symbols in new PCM signals
test_sig = np.array(sig).reshape(len(sig),1)
print(test_sig)
new_pcm_signal = sc.transform(test_sig)
prediction = model.predict(new_pcm_signal)
```

```
print('Predicted symbols', np.round(prediction))
```

Dataset with Noisy Signals and Their Corresponding Symbols

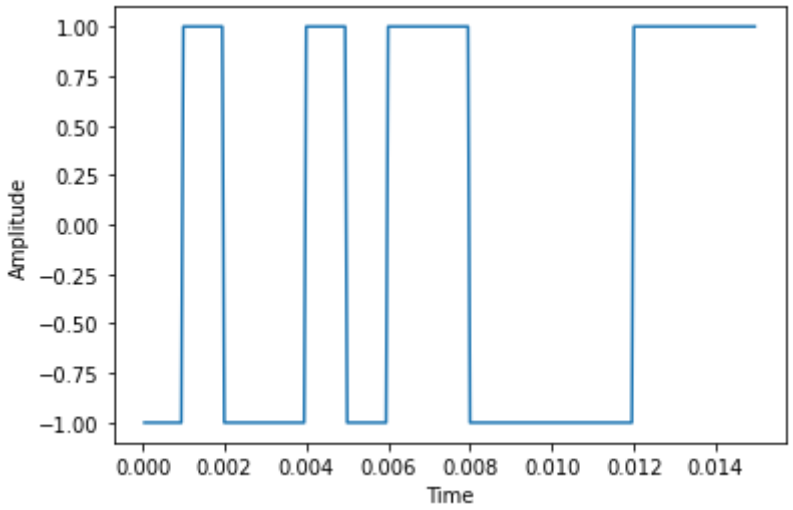
Signal	Symbol
0.9740488849543414	1
0.898067022632247	1
1.0174801188955898	1
-1.1188958062590812	0
-0.9191408531270693	0
-0.9754657633537476	0
1.1643366550121366	1
-1.1071569793407192	0
-0.9772476670441745	0
.....	.....

Out of this About 80% would be Used as Training Data, and the Rest as Testing Data

Results

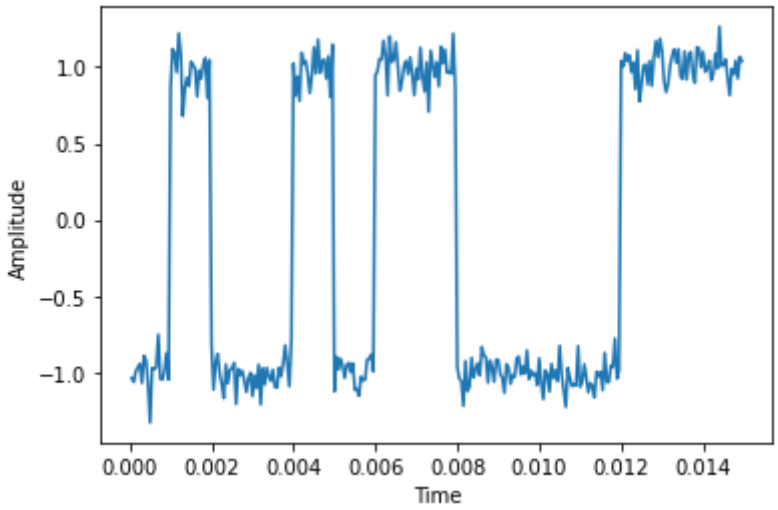
Test Data -

Let us say Message Signal - **[0 1 0 0 1 0 1 1 0 0]**



Let us say the receiver receives the following Signal -

[[-1.00924005], [0.86617397], [-0.81576782], [1.15448636], [1.02225824], [1.12216999], [0.94402741], [0.98953712], [-0.96299824], [-0.91720862]]



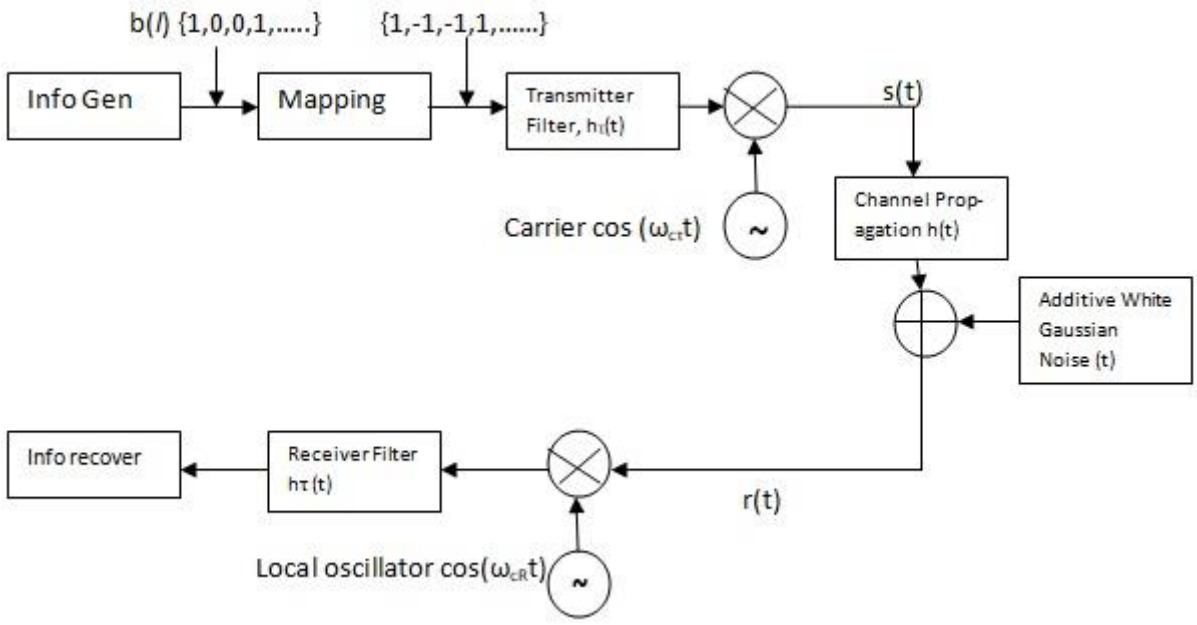
Output of ANN Model

```
Epoch 1/50
50/50 [==============================] - 1s 5ms/step - loss: 0.3708 - accuracy: 0.9625 - val_loss: 0.1500
- val_accuracy: 1.0000
Epoch 2/50
50/50 [==============================] - 0s 3ms/step - loss: 0.0782 - accuracy: 1.0000 - val_loss: 0.0255
- val_accuracy: 1.0000
Epoch 3/50
50/50 [==============================] - 0s 2ms/step - loss: 0.0196 - accuracy: 1.0000 - val_loss: 0.0078
- val_accuracy: 1.0000
Epoch 4/50
50/50 [==============================] - 0s 2ms/step - loss: 0.0084 - accuracy: 1.0000 - val_loss: 0.0033
- val_accuracy: 1.0000
Epoch 5/50
50/50 [==============================] - 0s 2ms/step - loss: 0.0041 - accuracy: 1.0000 - val_loss: 0.0018
- val_accuracy: 1.0000
Epoch 6/50
50/50 [==============================] - 0s 2ms/step - loss: 0.0026 - accuracy: 1.0000 - val_loss: 0.0011
- val_accuracy: 1.0000
.....
.....
.....
Epoch 47/50
50/50 [==============================] - 0s 2ms/step - loss: 3.5704e-05 - accuracy: 1.0000 - val_loss:
1.6946e-06 - val_accuracy: 1.0000
Epoch 48/50
50/50 [==============================] - 0s 2ms/step - loss: 4.7002e-05 - accuracy: 1.0000 - val_loss:
1.5570e-06 - val_accuracy: 1.0000
Epoch 49/50
50/50 [==============================] - 0s 2ms/step - loss: 3.6665e-05 - accuracy: 1.0000 - val_loss:
1.4393e-06 - val_accuracy: 1.0000
Epoch 50/50
50/50 [==============================] - 0s 2ms/step - loss: 3.1633e-05 - accuracy: 1.0000 - val_loss:
1.3280e-06 - val_accuracy: 1.0000
13/13 [==============================] - 0s 1ms/step - loss: 1.3280e-06 - accuracy: 1.0000
Test accuracy: 1.0
Predicted symbols [[0.]
 [1.]
 [0.]
 [0.]
 [1.]
 [0.]
 [1.]
 [1.]
 [0.]
 [0.]]
```

As we can see the Predictions are - **[[0.], [1.], [0.], [0.], [1.], [0.], [1.], [1.], [0.], [0.]]**
Which are the same as the original message!

BPSK Detection Model -

Let us Simulate a Typical BPSK Communication Scheme

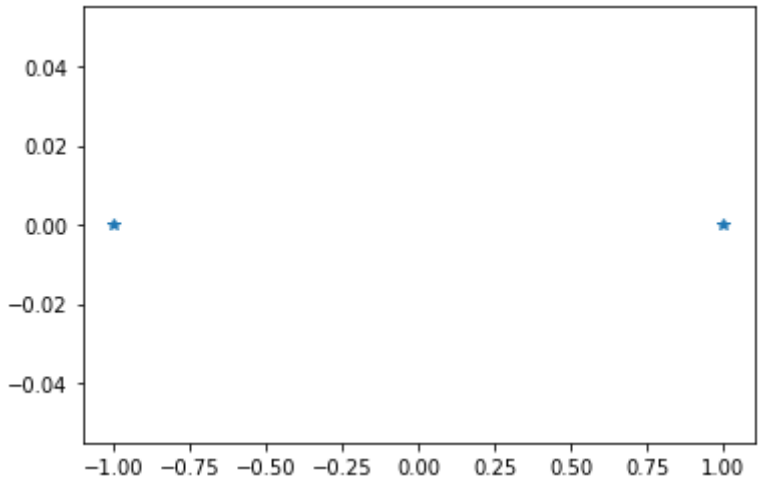



```
#Eb/N0 Vs BER for BPSK over AWGN (complex baseband model)
import numpy as np #for numerical computing
import matplotlib.pyplot as plt #for plotting functions
from scipy.special import erfc #erfc/Q function


#---------Input Fields------------------------
nSym = 100 # Number of symbols to transmit
EbN0dBs = np.arange(start=-
2,stop = 8, step = 2) # Eb/N0 range in dB for simulation
BER_sim = np.zeros(len(EbN0dBs)) # simulated Bit error rates


M=2 #Number of points in BPSK constellation
m = np.arange(0,M) #all possible input symbols
A = 1; #amplitude
constellation = A*np.cos(m/M*2*np.pi)  #reference constellation for BPSK


#------------ Transmitter---------------
inputSyms = np.random.randint(low=0, high = M, size=nSym) #Random 1's and 0's
 as input to BPSK modulator
print("Original Sequence -",inputSyms)
s = constellation[inputSyms] #modulated symbols


fig, ax1 = plt.subplots(nrows=1,ncols = 1)
ax1.plot(np.real(constellation),np.imag(constellation),'*')
```

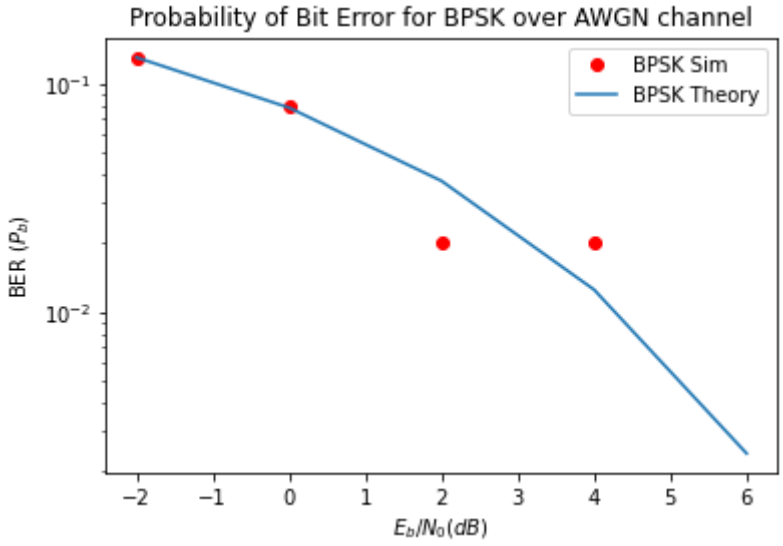
```
#---------- Channel -------------
#Compute power in modulatedSyms and add AWGN noise for given SNRs
for j,EbN0dB in enumerate(EbN0dBs):
    gamma = 10**(EbN0dB/10) #SNRs to linear scale
    print("SNR-",gamma)
    P=sum(abs(s)**2)/len(s) #Actual power in the vector
    N0=P/gamma # Find the noise spectral density
    n = np.sqrt(N0/2)*np.random.standard_normal(s.shape) # computed noise vec
tor
    r = s + n # received signal

    #-------------- Receiver ------------
    detectedSyms = (r <= 0).astype(int) #thresolding at value 0
    BER_sim[j] = float(np.sum(detectedSyms != inputSyms)/nSym) #calculate BER

print(BER_sim)
print(detectedSyms)
BER_theory = 0.5*erfc(np.sqrt(10**(EbN0dBs/10)))


fig, ax = plt.subplots(nrows=1,ncols = 1)
ax.semilogy(EbN0dBs,BER_sim,color='r',marker='o',linestyle='',label='BPSK Sim
')
ax.semilogy(EbN0dBs,BER_theory,marker='',linestyle='-',label='BPSK Theory')
ax.set_xlabel('$E_b/N_0(dB)$');ax.set_ylabel('BER ($P_b$)')
ax.set_title('Probability of Bit Error for BPSK over AWGN channel')
# ax.set_xlim(4,20);ax.grid(True);
ax.legend();plt.show()
```

Outputs -

Constellation Diagram



Theoretical vs Simulated Error



Original Sequence - [0 1 1 0 0 1 1 1 0 0]

SNR- 0.6309573444801932
[1.52057489 -0.58556757 -0.86930632 0.41502028 0.2798777 -2.45483509
-2.16024925 -1.33517175 1.14538785 1.96315009]

SNR- 1.0
[2.68602307 -1.69961389 -1.2155367 0.92590071 0.71607845 -1.34683164
-0.40770001 0.20395359 1.58361722 0.08733195]

SNR- 1.5848931924611136
[1.80910231 -1.03192953 0.37471313 1.25494161 0.89226201 -1.19084944
-1.53757542 -1.21454821 -0.09299699 0.78008212]

SNR- 2.51188643150958
[1.12390025 -0.94425659 -1.50456471 1.51382824 0.399768 -0.58659795
-1.25361022 0.42489691 0.77736487 0.86855934]

SNR- 3.9810717055349722
[1.47243173 -1.36077235 -0.70605556 0.96717911 0.46194256 -0.6876914
-1.12928826 -0.87500702 0.35191649 0.65248926]

Inference

Pros of using BPSK:

- BPSK is a relatively simple modulation scheme that is easy to implement and requires less computational resources compared to an ANN model.
- BPSK is well-suited for transmitting binary data over a noisy channel, as it can detect and correct errors in the signal.
- BPSK is widely used in communication systems and is supported by many devices and protocols.

Cons of using BPSK:

- BPSK may not be the best choice for detecting PCM signals that have more than two levels of quantization, as it can only transmit binary data.
- BPSK is susceptible to interference from other signals in the same frequency band, which can lead to errors in the received signal.

Pros of using an ANN model:

- An ANN model can potentially detect PCM signals with multiple levels of quantization, whereas BPSK can only transmit binary data.
- An ANN model can potentially detect more complex patterns in the data, which can improve the accuracy of the signal detection.
- An ANN model can potentially adapt to changing conditions and learn from new data, which can improve the robustness of the signal detection.

Cons of using an ANN model:

- An ANN model requires significant computational resources, which may make it less suitable for real-time applications or devices with limited processing power.
- An ANN model requires a large amount of training data to be effective, which may not be available in all situations.
- An ANN model may be more difficult to implement and maintain compared to a simpler modulation scheme like BPSK.

References

[1] Simon Haykin, Digital Communication Systems, John Wiley and Sons, 2014

[2] Simon Haykin and Michael Moher, Introduction into Analog and Digital Communications, Second edition, John Wiley and Sons, 2012

[3] Atif Haroon, Fahad Hussain, Muhammad Raheel Bajwa, “Decoding of Error Correcting codes using neural networks”, Blekinge Institute of Technology, 2012

[4] O. I. Abiodun et al., "Comprehensive Review of Artificial Neural Network Applications to Pattern Recognition," in IEEE Access, vol. 7, pp. 158820-158846, 2019, doi: 10.1109/ACCESS.2019.2945545.

[5] Jason Brownlee, “How to Develop an Encoder-Decoder Model for Sequence-to-Sequence Prediction in Keras”, November, 2017.