Analysis of the Paper: Benchmarking Machine Learning Techniques for THz Channel Estimation Problems

Mohamad Lakkis

October 9, 2024

1 Introduction

The paper focuses on the application of machine learning (ML) techniques for **channel estimation (CE)** in **Terahertz (THz) communication systems**, which are critical for future wireless networks such as 6G. Channel estimation is necessary for predicting how signals will propagate through the environment, particularly in challenging scenarios where high data rates and low-latency communications are required. THz signals face significant challenges due to their high frequency, which causes them to be sensitive to environmental factors such as reflection, scattering, and absorption.

The authors explore various machine learning algorithms to estimate the channel, comparing techniques such as **Neural Networks (NN)**, **logistic regression (LR)**, and **Projected Gradient Ascent (PGA)**. Among these, the results show that PGA provides the best performance, especially under low Signal-to-Noise Ratio (SNR) conditions, which makes it a promising candidate for THz channel estimation.

2 Challenges in THz Channel Estimation

The estimation of channel response parameters $\mathbf{H}$ in THz communication systems is particularly challenging due to the nature of THz signals. These signals operate at frequencies between 300 GHz and 10 THz, which means they have a very small wavelength and are susceptible to a variety of physical limitations such as high attenuation and molecular absorption. In addition, the hardware limitations, such as the need for high-speed **Analog-to-Digital Converters (ADCs)**, add to the complexity. These converters often operate with low resolution (e.g., 1-bit ADCs), which can introduce significant quantization errors.

Another challenge is that the **signal-to-noise ratio (SNR)** must be carefully managed to ensure reliable channel estimation. THz systems operate in

environments where thermal noise is substantial due to the large bandwidths involved. Therefore, traditional channel estimation techniques, which rely on pre-defined models, may not be adequate. Instead, the authors explore **data-driven approaches** using machine learning to improve the accuracy of channel estimation.

3 Handling Complex Numbers

Since the transmitted and received signals in THz systems are complex-valued, the authors of the paper address this by converting the complex definitions into real-valued representations. This conversion is necessary because most machine learning algorithms operate on real numbers.

- The received noisy symbols $\mathbf{y}_n$, the channel matrix $\mathbf{H}$, and the transmitted signal $\mathbf{x}_n$ are all split into their real and imaginary components. For example, $\mathbf{y}_n = [\mathrm{Re}(\mathbf{y}_n), \mathrm{Im}(\mathbf{y}_n)]$, and similar decompositions are applied to the channel matrix and the transmitted signal. - After this decomposition, the real and imaginary parts are processed as separate real-valued inputs, which allows machine learning algorithms to process the data effectively without handling complex numbers directly.

This approach simplifies the problem, allowing the system to operate in a real-valued framework while still preserving the necessary information from the original complex-valued signals.

4 Neural Network-Based Channel Estimation

In the **Neural Network (NN) architecture** proposed by the authors, the input signals $\mathbf{x}_n$ are passed through multiple neurons, each of which applies a weighted transformation followed by a non-linear activation function. The weights of the network correspond to the channel response parameters $\mathbf{H} = W$, and the bias terms correspond to the noise z. Note that for now assume that $\mathbf{H}, W$ are real valued. We will explain later in depth on how to handle the complex numbers.

The neural network aims to learn the mapping between the input feature $\mathbf{x}_n$ and the output signal $\mathbf{y}_n$. The transformation is represented by the equation:

$$y_n = f(H_i x_n + z_i)$$

where $f(\cdot)$ is the activation function applied to the weighted sum of inputs. Different activation functions are explored, including the **sign function** $\mathrm{sign}(x)$, which helps map the network's output to binary values, making it suitable for decision-making processes such as decoding binary signals.

The training of the neural network involves optimizing the weights $\mathbf{H}_i$ and bias terms z_i to minimize the **loss function**. They chose the loss function which measures the difference between the predicted output $\mathbf{y}_n$ and the expected output $\mathbf{y}_n^e$ as follows:

$$[\hat{H}, \hat{z}] = \underset{H,z}{\operatorname{argmin}}[\mathbb{J}(H, z)]$$

where

$$\mathbb{J}(H, z) = \frac{1}{N} \sum_{n=1}^{N} \|y_n - tanh(H_i x_n + z_i)\|_2^2 + \lambda \|H\|_2^2$$

where λ is the regularization parameter.
Note that during the training, the activation function used is the sign function.(so while calculating the loss we are not directly looking at our outputs, because what we care about explicitly is H and not being able to retrieve the output from H, but of course being able to retrieve the output from H is crucial to make it accurate, but here we are just noting on why the loss function doesn't directly use the outputs of the NN)

Once the loss is computed, the weights $\mathbf{H}_i$ are updated using **gradient descent**, where the gradients of the loss function with respect to the weights are calculated, and the weights are updated accordingly:

$$H_i \leftarrow H_i - \eta \nabla_{H_i} \text{Loss}$$

Here, η is the learning rate, which controls the step size of the gradient descent update.

Now I will provide the dimension of each of these entries, to make it clear what are using in each case:
- Note $M_r = N = M_t$
- $\mathbf{y}_n$ is of dimension $M_r \times 1$, where M_r is the number of receive antennas.
- $\mathbf{H}_i$ is of dimension $1 \times M_r$, where M_t is the number of transmit antennas.
- $\mathbf{x}_n$ is of dimension $M_t \times 1$, where M_t is the number of transmit antennas.
- $\mathbf{z}_i$ is of dimension $M_r \times 1$, where M_r is the number of receive antennas.
-$H_i = [H_{i,1}, ..., H_{i,N}]$
- H is $M_r \times M_t$
Note that the the received unquantized signals $\boldsymbol{r}$ are then converted to $\boldsymbol{y} = \hat{g}(\boldsymbol{r}) = Sign(Re(\boldsymbol{r})) + jsign(Im(\boldsymbol{r}))$, and so this is the $\boldsymbol{y}$ that we working with.

Now we are arrive exactly on how we are handling these complex numbers in the neural network.

So the training on a high level is the one we just described, but how are we handling the complex numbers in the neural network?
So first let us start with the form of the matrices $W = W_r + iW_i$, $y = y_r + iy_i$, $b = b_r + ib_i$, $x = x_r + ix_i$.
What we care about as like any NN is:

$$f(Wx+b) = f((W_r+iW_i)(x_r+ix_i)+b_r+ib_i) = f((W_r x_r - W_i x_i)+i(W_r x_i+W_i x_r)+b_r+ib_i)$$

And in this particular example: $W = H, b = z$, the values that we actually care about to do the channel estimation are the actual weights and biases. And so from now on we process these numbers as real numbers, and we don't care about the complex numbers, so in other words, at each step we are keep tracking of two things the real part and the imaginary part, nothing hard to grasp. And of course while backpropagating we are essentially doing the same thing, we are just updating the real part and the imaginary part separately.
Lastly, regarding the loss the way we are computing it is

$$Loss = \frac{1}{N}\sum_{n=1}^{N}(|y^e_{r,n} - tanh(x_{r,n} \times H_{r,n} + z_{r,n})|_2^2 + |y^e_{i,n} - tanh(x_{i,n} \times H_{i,n} + z_{i,n})|_2^2)$$

$$+\lambda(|H_r|_2^2 + |H_i|_2^2)$$

Where the subscript r and i denote the real and imaginary parts respectively. And n denotes the n^{th} sample.

5 Log-Likelihood Minimization

The paper also explores the use of **log-likelihood minimization** as a method for optimizing the channel estimation. In this approach, the log-likelihood function is derived from the assumption that the real and imaginary components of the received signal are modeled by a Gaussian distribution. The loss function based on log-likelihood is designed to estimate a transformed variable $\mathbf{X}^{Re}$, where:

$$\hat{X}^{Re} = \underset{X^{Re}}{\operatorname{argmin}} \sum_{i=1}^{M_t}\sum_{j=1}^{M_r} \left[\mathbf{1}[y^{Re}_{i,j} = 1]\log(\phi(X^{Re}_{i,j}/\sigma)) + \mathbf{1}[y^{Re}_{i,j} = -1]\log(1 - \phi(X^{Re}_{i,j}/\sigma))\right]$$

where $\phi(\cdot)$ is the cumulative distribution function (CDF) of the standard normal distribution, and σ represents the standard deviation.

By minimizing this log-likelihood function, the model attempts to estimate the channel parameters that maximize the probability of the observed binary outputs.

6 Projected Gradient Descent (PGD)

Now we will go through the **Projected Gradient Descent (PGD)** algorithm. This algorithm is usually used when we have a constrained problem:
Unconstrained minimization

$$\min_{x \in \mathbb{R}^n} f(x).$$

- All $x \in \mathbb{R}^n$ is feasible.
- Any $x \in \mathbb{R}^n$ can be a solution.

Constrained minimization

$$\min_{x \in Q} f(x).$$

- Not all $x \in \mathbb{R}^n$ is feasible.
- Not all $x \in \mathbb{R}^n$ can be a solution.
- The solution has to be inside the set Q.

An example:

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2^2 \quad \text{s.t.} \quad \|x\|_2 \leq 1$$

can be expressed as

$$\min_{\|x\|_2 \leq 1} \|Ax - b\|_2^2.$$

Here $Q := \{v \in \mathbb{R}^n : \|v\|_2 \leq 1\}$ is known as the unit ℓ_2 ball.

Let us recap for a moment how gradient Descent Works for Unconstrained Problems: Reminder of the problem:

$$\min_{x \in Q} f(x).$$

- Starting from an initial point $x_0 \in \mathbb{R}^n$, **GD** iterates the following until a stopping condition is met:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k),$$

- $k \in \mathbb{N}$: the current iteration counter
- $k + 1 \in \mathbb{N}$: the next iteration counter
- x_k: the current variable
- x_{k+1}: the next variable
- ∇f is the gradient of f with respect to differentiation of x (i.e. $\nabla f = \frac{\partial f}{\partial x}$)
- $\nabla f(x_k)$ is the ∇f at the current variable x_k
- $\alpha_k \in (0, +\infty)$: gradient stepsize, could be fixed or learned at each iteration.
- Note: Here we are assuming that the function f is differentiable. If that is not the case, we can use subgradient descent.

Now the Question is Can we tune this algorithm to make it work for Constrained Problems?

And the answer is yes, the key is Euclidean projection operator. $\mathbb{R}^n \to \mathbb{R}^n$. Like before we will start with some assumptions:

- f is assumed to be continously differentiable. (i.e. $f \in C^1$ or $\nabla f(x)$ exists $\forall x$)
- In general one also assumes that f is globally L-Lipschitz continuous. (i.e. $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$)
- $\emptyset \neq Q \subset \mathbb{R}^n$ is convex and compact
- The constraint is represented by a set Q
- $Q \subset \mathbb{R}^n$ means Q is a subset of $\mathbb{R}^n$, the domain of f
- $Q \neq \emptyset$ means Q is not an empty set. *it is not useful for discussion if* Q is empty
- Q is a convex set $\forall x \forall y \forall \lambda \in (0,1)\,(x \in Q, y \in Q \implies \lambda x + (1-\lambda)y \in Q)$
- Q is compact *compact = bounded + closed*

Now PGD = GD + Projection.

- Starting from an initial point $x_0 \in \mathbb{R}^n$, **PGD** iterates the following until a stopping condition is met:
- $x_{k+1} = \text{Proj}_Q(x_k - \alpha_k \nabla f(x_k))$
- Proj_Q is the Euclidean projection operator onto Q, which maps a point in $\mathbb{R}^n$ to the closest point in Q.(i.e. $\text{Proj}_Q(x) = \underset{y \in Q}{\text{argmin}} \|x - y\|_2$)
- $\alpha_k \in (0, +\infty)$: gradient stepsize, could be fixed or learned at each iteration.
- $k \in \mathbb{N}$: the current iteration counter
- $k+1 \in \mathbb{N}$: the next iteration counter
- x_k: the current variable
- x_{k+1}: the next variable
- ∇f is the gradient of f with respect to differentiation of x (i.e. $\nabla f = \frac{\partial f}{\partial x}$)

7 Projected Gradient Ascent (PGA)

This is the same as PGD, but instead of minimizing the function, we are maximizing it. (so in ML, we changed the objective function, not the same ONE we were trying to minimize, because that doesn't make sense!)
And instead of taking a step in the opposite direction of the gradient, we are taking a step in the direction of the gradient.
Lastly Note that the proj function stays the same, if the point is outisde our desired set, we need to projected back by finding the closest point to it from this set. (In the Euclidean Projection we use Euclidean Distance!)

Relating this to what we are we doing in the paper:

Some Questions to answer:

- What is the constraint we are working under (i.e. what is the set Q)?
- How are we projecting this Matrix H that we got back to Q?

Very interesting questions, let's dive into them:

Our constrained Set Q

The constraint is that H should have a low rank $r \ll N$. In many real-world systems, the channels can often be well-approximated by matrices of much lower rank. For example, in multiple-input multiple-output (MIMO) systems, low-rank approximations are often used to reduce computational complexity. For more details, refer to the work in this paper [1] available at IEEE Xplore.
So the update step in the PGA algorithm is as follows:

$$H_{k+1} = H_k + \alpha_k \nabla f(H_k)$$

Where $\nabla f(H_k)$ is the gradient of the loss function with respect to H_k, and α_k is the step size.
But now we have a Problem How do we make sure this matrix $H_{k+1} \in Q$?

Projection Step to Q

There is many way to project a matrix to a low-rank space, but one of the most known way is to use the **Singular Value Decomposition (SVD)**. SVD is a matrix factorization technique that can decompose a matrix into its singular values and corresponding singular vectors. By keeping only the largest singular values (up to rank r), you can project the matrix back onto the space of low-rank matrices.
Simplex projection: Ensures the solution remains within certain constraints.(maybe H non-negative or any other constrain)
The SVD of a matrix H is given by:

$$H = U\Sigma V^T$$

where U and V are orthogonal matrices, and Σ is a diagonal matrix containing the singular values of H. To project H to a low-rank matrix, we keep only the r largest singular values and setting the rest to zero. This operation effectively reduces the rank of H to r. The projected matrix is then given by:

$$H_{\text{proj}} = U\Sigma_{\text{proj}}V^T$$

where Σ_{proj} is the diagonal matrix with only the r largest singular values. This projection step ensures that the updated matrix H_{k+1} remains within the low-rank constraint.

8 Conclusion

This paper provides a comprehensive comparison of various machine learning techniques for **THz channel estimation**. By converting the complex-valued channel estimation problem into a real-valued one, the authors are able to leverage powerful machine learning models such as neural networks and log-likelihood minimization techniques to achieve accurate channel estimation. The results indicate that methods like **Projected Gradient Ascent (PGA)** and **Neural Networks (NN)** perform well in challenging SNR conditions, making them suitable candidates for future THz communication systems.

References

[1] N. J. Myers, K. N. Tran, and R. W. Heath, "Low-rank mmwave mimo channel estimation in one-bit receivers," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2020, pp. 5005–5009.