Metasurfaces

Egor Makarenkov, Co-authors

July 2024

1 Introduction

Metasurfaces are artificially engineered surfaces that can control electromagnetic waves in ways that traditional materials cannot. These surfaces comprise numerous tiny structures known as "meta-atoms," which are much smaller than the wavelength of the incident electromagnetic waves. By precisely arranging these meta-atoms, we can manipulate light, radio waves, and other forms of electromagnetic radiation in unique ways, such as bending light, focusing electromagnetic waves, and even cloaking objects from detection [1]. For example, V-shaped meta-atoms are nanostructures in metasurfaces designed to manipulate electromagnetic waves. Consisting of two arms forming a "V," their angle and length can be adjusted to control optical properties like phase, amplitude, and polarization. These meta-atoms create resonances that enable applications such as beam steering and polarization conversion. Also, fishnet meta-atoms feature a periodic array of holes resembling a fishnet, composed of alternating metal and dielectric layers. This structure induces plasmonic resonances, leading to negative permittivity and permeability at specific frequencies. We use fishnet meta-atoms for superlensing, cloaking, and creating materials with negative refractive indices. [2, 1] Some metasurfaces change capacitance and inductance based on excitation [1, 3].

Over recent years, comprehensive techniques have been developed to manufacture metasurfaces with prescribed frequency, capacitance (of the top layer), capacitance (of the bottom layer), inductance, resistance, angle of the incident wave, reflectance, transmittance, label (R or T dominance), reflectance + transmittance, and absorptivity [8, 1]. However, the relation of these parameters to reflectance and transmittance properties of metasurfaces is highly implicit [reference, maybe ask Sakib] and is hard to use in practical computations.

In this paper, we use 2 methods to predict reflectance and transmittance for the other given parameters of metasurfaces based on previously collected data. The first method we attempted to use was multiple linear regression (MLR). This is a form of linear regression that, unlike normal linear regression, uses over 2 variables. This technique works by predicting the value of one variable based on several other variables. It does this by finding the best-fit line that describes the relationship between the dependent variable and the independent

variables. The equation of this line helps make predictions based on new data. The second method we use is deep learning neural networks. We build a neural network model for prediction. It reads and cleans the data using pandas (Python library), converts non-numeric columns, and splits the data into training and test sets. It then scales the data and uses TensorFlow to create a neural network model, trains it, and evaluates its performance using mean squared error.

The paper is organized as follows. In the next section, we introduce the parameters of the data that we use and discuss the concepts of transmittance and reflectance in some detail. In Section 3, we review the methods that we use to make conclusions. And finally, in Section 4, we summarize the results obtained. The Data Sheet and Python code used for computations are included in the Appendix.

2 Preliminaries

The spread of electromagnetic energy can be modeled by waves. For example, as seen in the two-slit experiment [reference], when light passes through two slits in a wall, the image on the second wall is comprised of multiple components, not just two, and the waves-based modeling framework occurs in the explanation of this phenomenon. The number of wave oscillations that pass a fixed point in a second is called the wave's frequency ("Frequency" in the Data Sheet).

Design of metasurfaces in the form of connected layers offers additional parameters for control of the properties of metasurfaces. In the simulations of this paper, a metasurface of 2 layers is considered, and capacitances of the two layers are termed C_V and $C_V_$bottom, while inductance and resistance are denoted L_V and R_V.

The angle between the wave and the normal to metasurface is measured as *theta*. Finally, reflectance R is defined as the portion (from 0 to 1) of the power of the wave that is reflected by the metasurface, transmittance T is the portion of the power that passes through the metasurface, and absorptivity is the power that is absorbed by metasurface. The total of transmittance, reflectance, and absorptivity is 1.

3 Methods

In this paper, we use 2 methods to predict reflectance and transmittance for the other given parameters of metasurface based on previously collected data.

The first method we attempted to use was multiple linear regression (MLR). Multiple Linear Regression (MLR) is a technique used to predict the value of a dependent variable based on multiple independent variables. In practice, MLR models the relationship between these variables by fitting a linear equation to the observed data.

```
model_T.fit(X_train, T_train)
```

```
model_R.fit(X_train, R_train)
```

The general form of the MLR equation is:
$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \epsilon$
Where:

- $\hat{y}$ is the predicted value of the dependent variable,
- β_0 is the intercept,
- $\beta_1, \beta_2, \ldots, \beta_p$ are the coefficients of the independent variables $x_1, x_2, \ldots, x_p$,
- ϵ is the error term, accounting for the variation not explained by the model.

In MLR, the coefficients (β) are estimated using the Ordinary Least Squares (OLS) method, which minimizes the sum of squared residuals between observed values and predictions. The objective function for OLS in MLR is [5]

$$\text{Minimize} \quad \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

This method seeks to find the line (or hyperplane) that best fits the data by reducing the error between the predicted and actual values.

Implementation with Pandas, Scikit-Learn, and PySpark:

- **Pandas and Scikit-Learn**: In Python, MLR is commonly implemented using libraries like Pandas for data manipulation and scikit-learn for building and evaluating models. After preparing the dataset using Pandas, the LinearRegression class from scikit-learn can be used to fit the model.

```
X_train, X_test, T_train, T_test, R_train, R_test =
    train_test_split(X, T, R, test_size=0.2,
    random_state=42)
```

The process includes splitting the data, fitting the model, making predictions, and evaluating the performance using metrics like Mean Squared Error (MSE) and R-squared (R^2). [5]

```
mse_T = mean_squared_error(T_test, T_pred)
r2_T = r2_score(T_test, T_pred)

mse_R = mean_squared_error(R_test, R_pred)
r2_R = r2_score(R_test, R_pred)
```

So while the core principles of MLR remain consistent across platforms, the implementation in scikit-learn is more suitable for small to medium datasets and focuses on user-friendly methods for model evaluation. PySpark, on the other hand, is designed for large-scale data processing and can handle more complex computations in distributed systems.

The second method we used was deep learning neural networks. This approach involves training a deep neural network (DNN) with three hidden layers to perform regression on a dataset. The model includes an input layer that accepts n features from the dataset, followed by three hidden layers. The first and second hidden layers have 128 neurons, while the third has 64 neurons. All hidden layers use the ReLU activation function, defined as $\text{ReLU}(x) = \max(0, x)$. The output layer consists of two units to predict two target variables. This can be seen in the following snippet from our code:

```
# Define a neural network model with two output units
model = tf.keras.Sequential([
    # First hidden layer with 128 units and ReLU activation
    tf.keras.layers.Dense(128, activation='relu',
        input_shape=(X_train.shape[1],)),

    # Dropout layer with a 30% dropout rate to prevent
        overfitting
    tf.keras.layers.Dropout(0.3),

    # Second hidden layer with 128 units and ReLU activation
    tf.keras.layers.Dense(128, activation='relu'),

    # Dropout layer with a 30% dropout rate to prevent
        overfitting
    tf.keras.layers.Dropout(0.3),

    # Third hidden layer with 64 units and ReLU activation
    tf.keras.layers.Dense(64, activation='relu'),

    # Dropout layer with a 30% dropout rate to prevent
        overfitting
    tf.keras.layers.Dropout(0.3),

    # Output layer with 2 units for predicting L and Q
    tf.keras.layers.Dense(2)
])
```

Listing 1: Neural Network Model Definition

The network is trained using mean squared error as the loss function and optimized with the Adam optimizer. Features are standardized, and the data is split into training and testing sets for evaluation. This can be seen in the following snippet from our code:

```
    # Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(
    learning_rate=0.001), loss='mean_squared_error')
```

Next, we train the model using the fit method, where X_train and y_train are the input features and target values. The training runs for 100 epochs with

a batch size of 32, and 20% of the data is used for validation. This can be seen in the following snippet of our code:

```
    # Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size
    =32, validation_split=0.2)
```

4 Results

In this section, we will explain the results of both models, the linear regression model, and the neural network model. We will explain how we interpreted the results, what prompted us to switch from linear regression to neural networks, and how we calculated the percent improvement from linear regression to neural network. We will go through all of that in chronological order. So first we finished training the linear regression model which had 2 models in it; the transmission model and the reflectance model. The results that the code gave us were Mean-Squared Error and R-squared. We will be using R-squared to interpret the accuracy of both models. For the transmission model, the accuracy was approximately 66.4%, and 64.4% for the reflection model. Now we will show how we got those numbers.

The first step is to calculate the mean of actual values (for both T and R) with the 2 following equations (one for T and the other for R):

$$\bar{y}T = \frac{1}{n}\sum_{i=1}^{n} yT, i$$

$$\bar{y}R = \frac{1}{n}\sum_{i=1}^{n} yR, i$$

[6]
The next step is to calculate the total sum of squares (SST) for T and R:

$$\mathrm{S}ST_T = \sum_{i=1}^{n}(yT, i - \bar{y}T)^2$$

$$\mathrm{S}ST_R = \sum_{i=1}^{n}(yR, i - \bar{y}R)^2$$

[6, 7]
The third step is to calculate the residual sum of squares (SSR) for T and R:

$$\mathrm{S}SR_T = \sum_{i=1}^{n}(yT, i - \hat{y}T, i)^2$$

$$\mathrm{S}SR_R = \sum_{i=1}^{n} (yR, i - \hat{y}R, i)^2$$

[7]
And the last step is to calculate the R-Squared using the following equation.

$$\mathrm{R}_T^2 = 1 - \frac{SSR_T}{SST_T}$$

$$\mathrm{R}_R^2 = 1 - \frac{SSR_R}{SST_R}$$

Now we will show an example of how we can use these equations for a simple hypothetical dataset example.

- **Hypothetical Dataset:**
  - Actual Values for X: {3, 4, 5}
  - Predicted Values for X: {2.8, 4.1, 5.3}
- **Calculate Mean:**
$$\bar{y}T = \frac{3+4+5}{3} = 4$$
- **Calculate SST:**
$$SST_X = (3-4)^2 + (4-4)^2 + (5-4)^2 = 1 + 0 + 1 = 2$$
- **Calculate SSR:**
$$SSR_X = (3-2.8)^2 + (4-4.1)^2 + (5-5.3)^2 = 0.04 + 0.01 + 0.09 = 0.14$$
- **Calculate R^2:**
$$R_X^2 = 1 - \frac{0.14}{2} = 1 - 0.07 = 0.93$$

Using this method we got the numbers 66.4% and 64.4%. These results are not the best and show that linear regression had some trouble capturing the relationship between the variables with a linear pattern. because of that, we tried a method that can capture more complex patterns. That method is a deep neural network (DDN). We built the network and trained it.

Next after building the model, we decided to test it to see how big the improvement was from LR to DNN. To do this we needed to get data to test it on. So what we did was we took data from the original dataset and copied and pasted it into a new spreadsheet. After doing that we deleted that data from the original dataset so the models would have never seen the data before and would have to rely on their training to predict the transmission and reflection. We took out 3 lines of data from lines 2585, 4518, and 8126. Next in the code of both the lineasr regression model and the neural network model we added a

section that will input manual data after the model has been trained and predict the values. The code added for the LR model creates a new pandas DataFrame named `new_data` with specific columns a, b, c, e, f, and g, each containing a list of example values. This DataFrame is used to predict new values for Transmittance (T_new_pred) and Reflectance (R_new_pred) using pre-trained models `model_T` and `model_R`. The code then iterates through the predicted values, printing each set of predicted Transmittance and Reflectance values in a formatted string. And the code added for the neural network constructs a pandas DataFrame named `new_data`, containing columns `col1`, `col2`, `col3`, `col4`, `col5`, `col6`, and `col9`, each with a list of example values. The data is then standardized using a pre-fitted scaler, resulting in `new_data_scaled`. Predictions are generated from a model using this scaled data, producing the values L_pred and Q_pred. Finally, the predictions for each set of data are printed in a formatted string.

After adding that code we ran it and the results we got will be listed in the following table:

Actual Values (T)	**Actual Values (R)**	**(LR) Values (T)**	**(LR) Values (R)**	**(DNN) Values (T)**	**(DNN) Values (R)**
0.879548601	0.019619221	0.631705999	0.256948091	0.846831679	0.031582236
0.558632857	0.318089542	0.569205832	0.246141658	0.563892006	0.310687661
0.813646667	0.069711561	0.901711250	-0.0473694	0.819288611	0.075363367

Table 1: Comparison of Transmission and Reflectance Values

Next using the above data we will find the difference between the actual values and predicted values for both methods. We will organize this data in another table.

Method	**Reflectance**	**Transmittance**
LR Diff Set 1	0.256948091745451	0.247842601339270
DNN Diff Set 1	0.011963015642542	0.032716921770542
LR Diff Set 2	0.071947883428412	0.010572975550065
DNN Diff Set 2	0.007401881167678	0.005259149468397
LR Diff Set 3	0.117081012852246	0.088064583397441
DNN Diff Set 3	0.005651806604953	0.005641944260426

Table 2: Comparison of Reflectance and Transmittance Values for Different Methods

To calculate the percentage improvement for each set, use the following formula:

$$\text{Improvement} = \frac{(\text{LR} - \text{DNN})}{\text{LR}} \times 100\%$$

where LR represents the Linear Regression values and DNN represents the DNN values. This formula calculates the percentage decrease from the Linear Regression values to the DNN values for both Reflectance and Transmittance. Here are the approximate percentages we calculated in the form of another table:

Improvement Set	**(R) Improvement**	**(T) Improvement**
Improvement Set 1	95.36%	86.83%
Improvement Set 2	89.73%	50.19%
Improvement Set 3	95.18%	93.59%

Table 3: Percentage Improvement for Reflectance (R) and Transmittance (T)

Now we will find the average improvement among all 3 sets for Reflectance and Transmission using a table.

Average Improvement	**(R)**	**(T)**
	93.42%	76.54%

Table 4: Average Improvement for Reflectance (R) and Transmittance (T)

Now by finding the average of those 2 percentages, we can see that we get 84.98%. This means that the total improvement from the linear regression model to the neural network model was 84.98% which is very significant and good for accuracy. The results of the neural network in the table show that the neural network was in fact successful in finding a close trend between variables in the data.

References

[1] Hou-Tong Chen et al, A review of metasurfaces: physics and applications, 2016 Rep. Prog. Phys. 79 076401.

[2] Xi Gao, Xu Han, Wei-Ping Cao, Hai Ou Li, Hui Feng Ma, and Tie Jun Cui, Fellow, IEEE, Ultrawideband and High-Efficiency Linear Polarization Converter Based on Double V-Shaped Metasurface, IEEE TRANSACTIONS ON ANTENNAS AND PROPAGATION, VOL. 63, NO. 8, AUGUST 2015.

[3] Stanislav B. Glybovski, Sergei A. Tretyakov, Pavel A. Belov, Yuri S. Kivshar, Constantin R. Simovski, Metasurfaces: From microwaves to visible, Physics Reports 634 (2016) 1–72.

[4] Will Penny, Mathematics for Brain Imaging, Functional Imaging Laboratory, Wellcome Trust Centre for Neuroimaging, Institute of Neurology, UCL, 12 Queen Square, London WC1N 3BG, UK, April 8, 2008. Available: `http://www.fil.ion.ucl.ac.uk/~wpenny/mbi/`

[5] A. Testas, Multiple Linear Regression with Pandas, Scikit-Learn, and PySpark. In: Distributed Machine Learning with PySpark. Apress, Berkeley, CA, 2023. `https://doi.org/10.1007/978-1-4842-9751-3_3`

[6] Penn State, "STAT 462: Applied Regression Analysis", online course materials, Pennsylvania State University. Available: `https://online.stat.psu.edu/stat462/lesson/1/1.2`

[7] MathWorks, "R-squared and Other Regression Statistics", MATLAB Documentation. Available: `https://www.mathworks.com/help/stats/coefficient-of-determination-r-squared.html`

[8] Syed S. Bukhari, J(Yiannis) Vardaxoglou, and William Whittow, "A Metasurfaces Review: Definitions and Applications," Department of Materials, University of Oxford, Oxford, UK; Wolfson School of Mechanical, Electrical and Manufacturing Engineering, Loughborough University, UK, Appl. Sci., vol. 9, no. 13, pp. 2727, July 2019. Available: `https://doi.org/10.3390/app9132727`