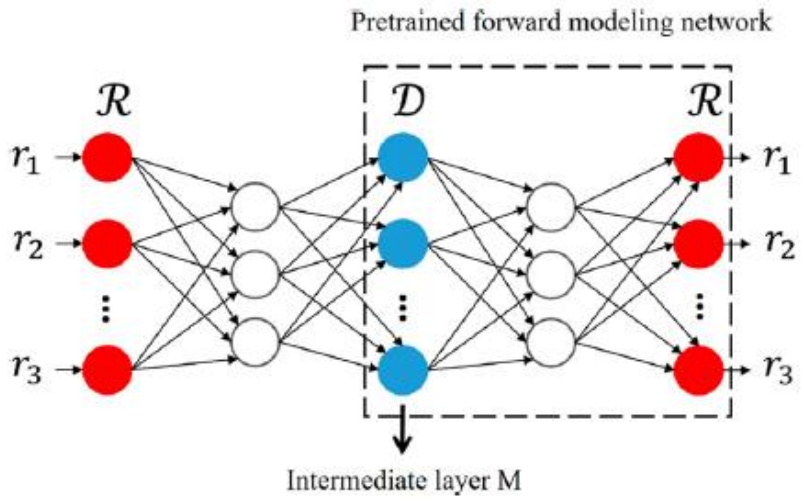
Training DNN for the Inverse Design of Nanophotonic Structures



SNU ECE Jeon Hyesung

Organic Electronics and Nanophotonics Laboratory

SNU Graduate School of Convergence Science and Technology

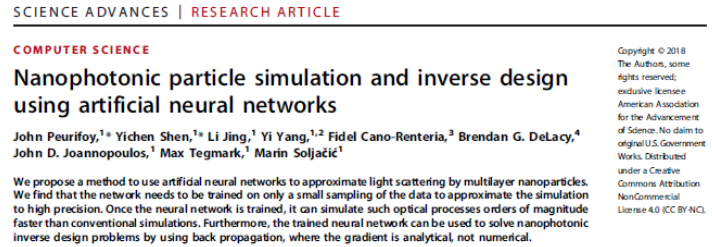
0. Dataset creation using TMM codes

- Capable of creating datasets : 192s to create 200,000 datasets
- Dataset of [Layer thickness(8), Transmittance(201)] = **[L, T]**

・Transmittance through 4 Stacks of SiO_2/SiNx layers, each of 40nm~200nm

・Focused on 400nm ~ 800nm wavelength spectrum

・75% train set, 25% valid set, **50,000 test set**



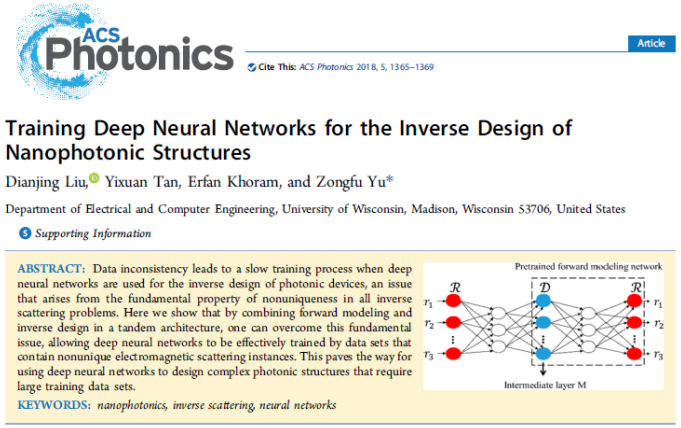
SCIENCE ADVANCES | RESEARCH ARTICLE

COMPUTER SCIENCE

Nanophotonic particle simulation and inverse design using artificial neural networks

John Peurifoy,[1]* Yichen Shen,[1]* Li Jing,[1] Yi Yang,[1,2] Fidel Cano-Renteria,[3] Brendan G. DeLacy,[4] John D. Joannopoulos,[1] Max Tegmark,[1] Marin Soljačić[1]

We propose a method to use artificial neural networks to approximate light scattering by multilayer nanoparticles. We find that the network needs to be trained on only a small sampling of the data to approximate the simulation to high precision. Once the neural network is trained, it can simulate such optical processes orders of magnitude faster than conventional simulations. Furthermore, the trained neural network can be used to solve nanophotonic inverse design problems by using back propagation, where the gradient is analytical, not numerical.



ACS Photonics

Article

Cite This: ACS Photonics 2018, 5, 1365−1369

Training Deep Neural Networks for the Inverse Design of Nanophotonic Structures

Dianjing Liu, Yixuan Tan, Erfan Khoram, and Zongfu Yu*

Department of Electrical and Computer Engineering, University of Wisconsin, Madison, Wisconsin 53706, United States

Supporting Information

ABSTRACT: Data inconsistency leads to a slow training process when deep neural networks are used for the inverse design of photonic devices, an issue that arises from the fundamental property of nonuniqueness in all inverse scattering problems. Here we show that by combining forward modeling and inverse design in a tandem architecture, one can overcome this fundamental issue, allowing deep neural networks to be effectively trained by data sets that contain nonunique electromagnetic scattering instances. This paves the way for using deep neural networks to design complex photonic structures that require large training data sets.

KEYWORDS: *nanophotonics, inverse scattering, neural networks*

1. Forward Model (L to T)

- Architecture : Input L(8) – 300-300-300-300 – Output T(201)

```
model = keras.Sequential([
    layers.Dense(300, activation='relu', input_shape=input_shape),
    layers.Dense(300, activation='relu'),
    layers.Dense(300, activation='relu'),
    layers.Dense(300, activation='relu'),
    layers.Dense(201)
])
```

Model architecture

```
early_stopping = callbacks.EarlyStopping(
    min_delta=0.001,
    patience=50,
    restore_best_weights=True,
)
```

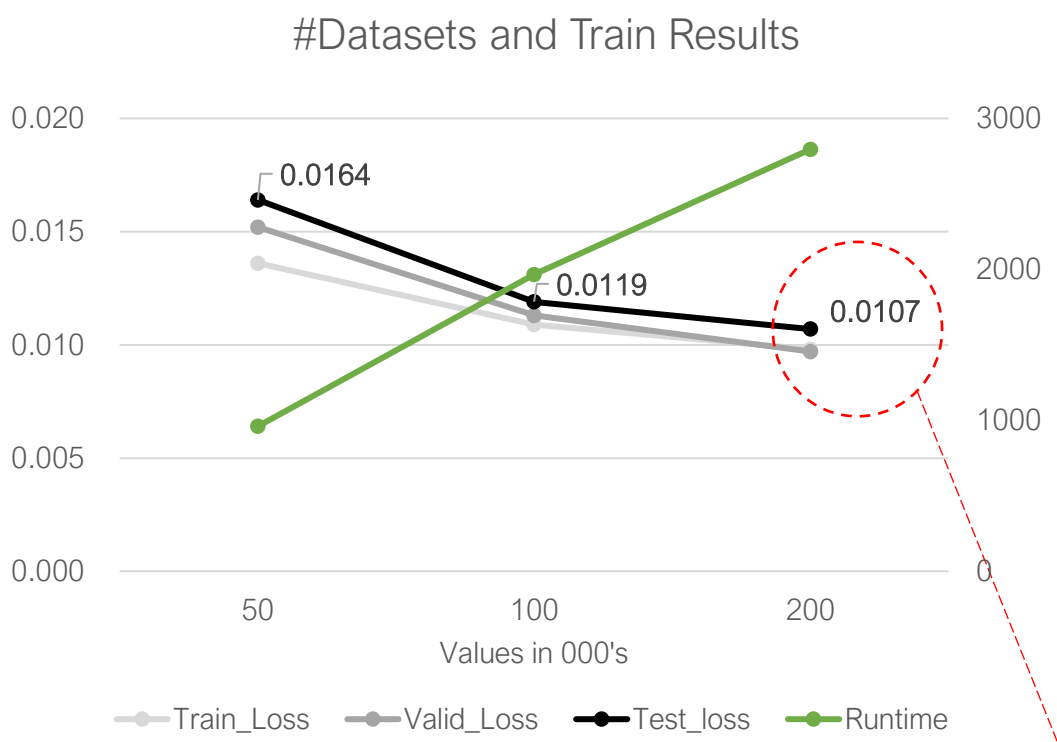
Overfitting 방지

```
model.compile(
    optimizer='adam',
    loss='mae',
)

compile_start = time.time()
history = model.fit(
    X_train, y_train,
    validation_data=(X_valid, y_valid),
    batch_size=100,
    epochs=1000,
    callbacks=[early_stopping],
    verbose=0,
)
compile_end = time.time()
```

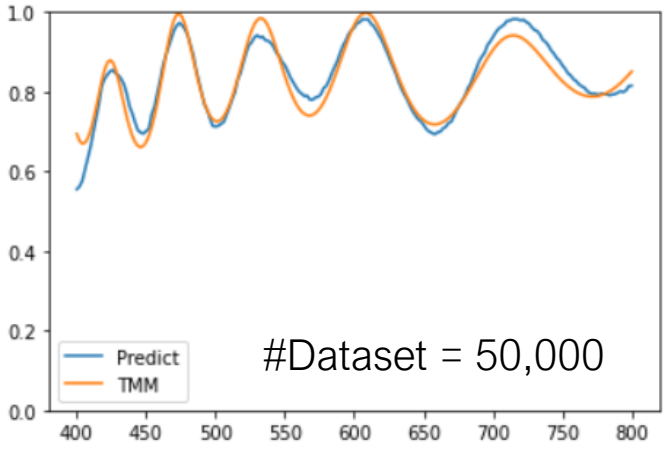
Loss function as MAE

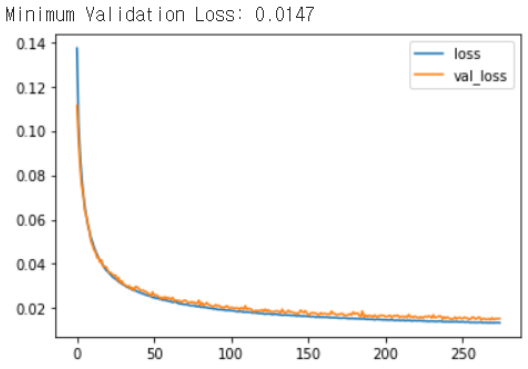
Model training variables

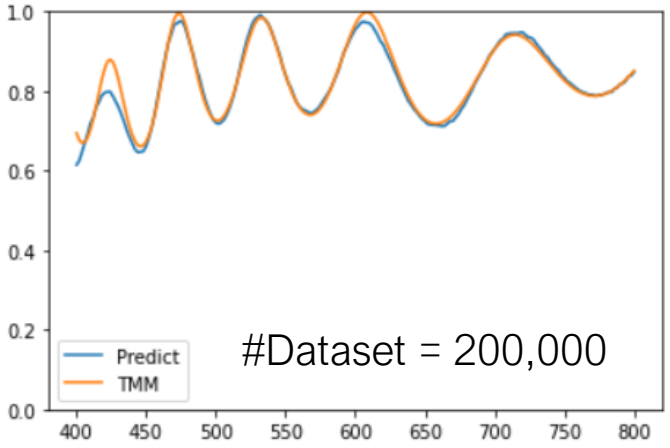
1. Forward Model (L to T)

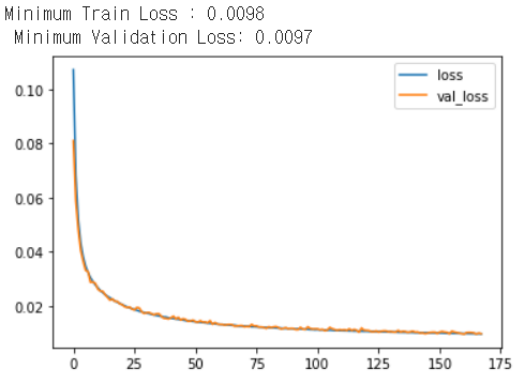


Prediction runtime : 2.7s
TMM code runtime : 47.14s

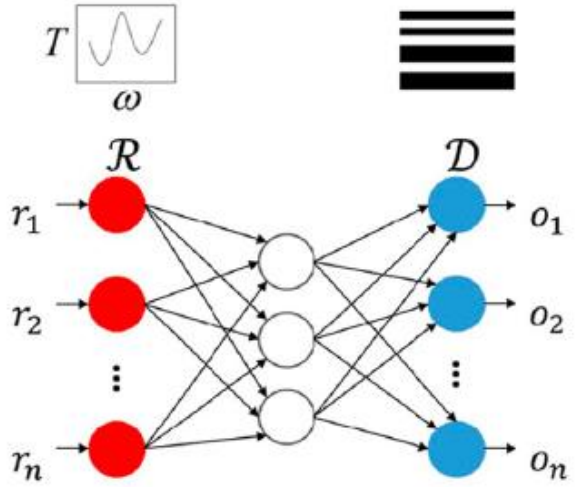








2. Backward Model (T to L)



a. Forward model in **Loss function**

- Input T, output L from dataset

① MAE vs **Custom Loss function** using Forward model (f_model)

```
def inverse_loss(y_true, y_pred):
  loss = K.mean(K.square(f_model(y_pred) - f_model(y_true)))*100
  return loss
```

MAE of [f_model(pred) – f_model(true)]

```
def inverse_loss(y_true, y_pred):
  loss = K.mean(K.abs(f_model(y_pred) - f_model(y_true)))
  return loss
```

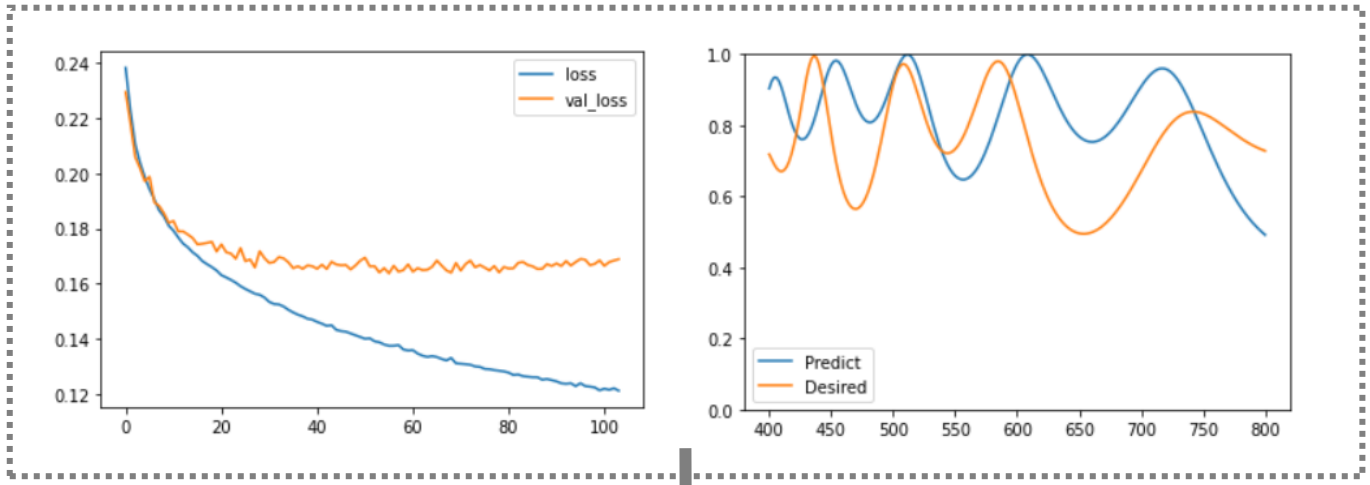
MSE of [f_model(pred) – f_model(true)]

```
model.compile(
    optimizer='adam',
    loss=inverse_loss,
)
```

② **200-500-200-20** vs 300-300-300-300

③ Using dataset of **200,000** ~ 50,000

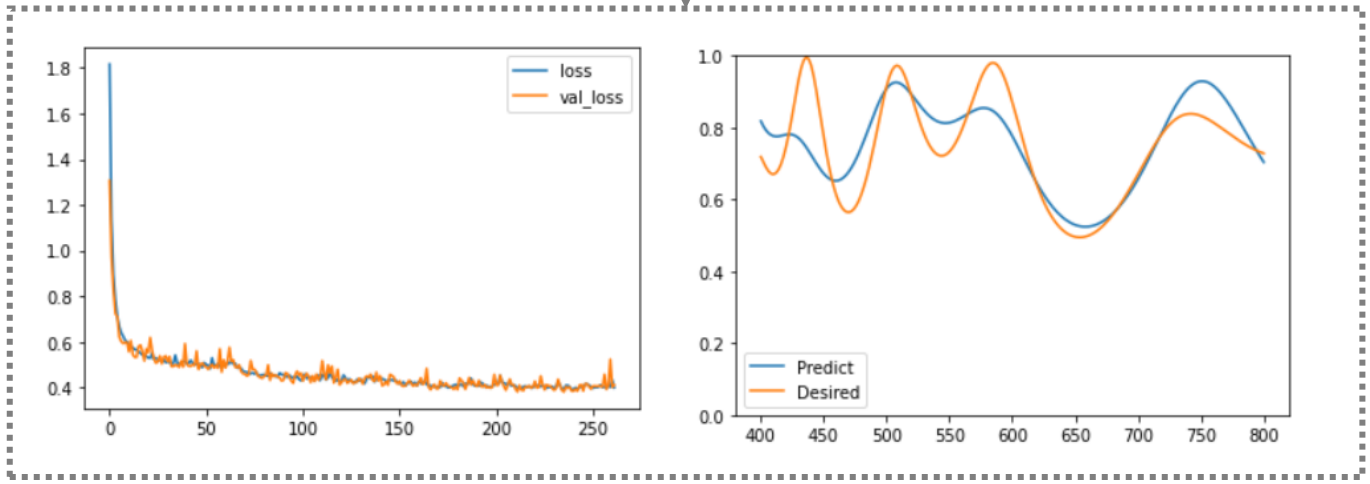
2. Backward Model (T to L)



MAE / 300-300-300-300 / 50,000

Valid loss of scaled L : 0.1643

Test loss of T : 0.1069



Custom Loss / 200-500-200-20 / 200,000

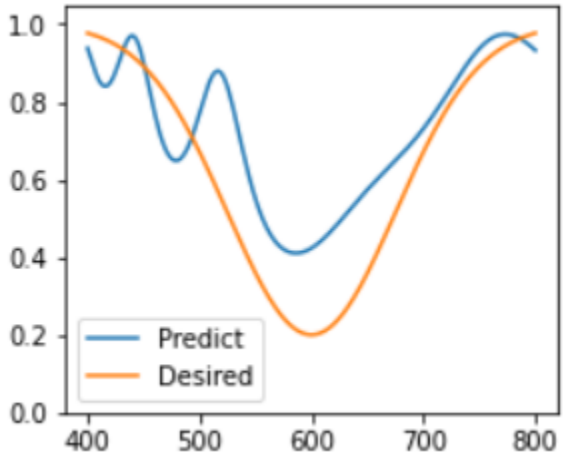
Valid loss of scaled L : 0.3117

Test loss of T : 0.0468

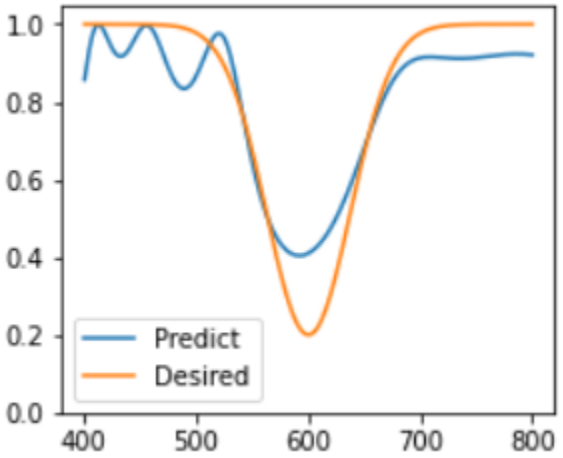
2. Backward Model (T to L)

- Artificial Spectrum (Gaussian)

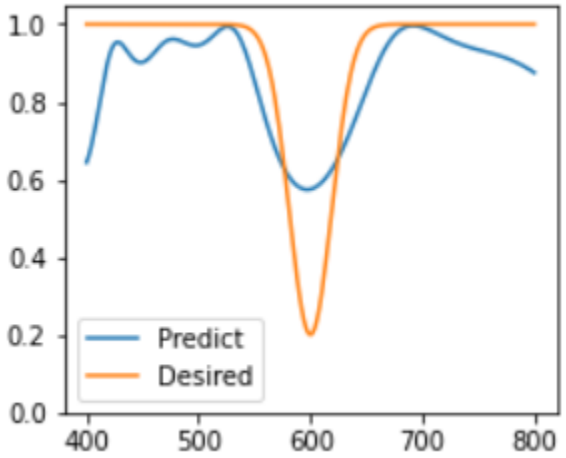
MAE error : 0.1296



MAE error : 0.0754

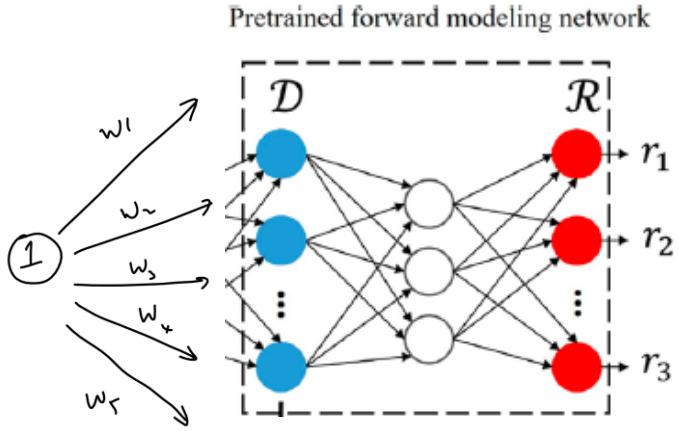


MAE error : 0.0946



Average MAE error : 0.0998

2. Backward Model (T to L)



b. Solving one specific Layer **(Optimization)**

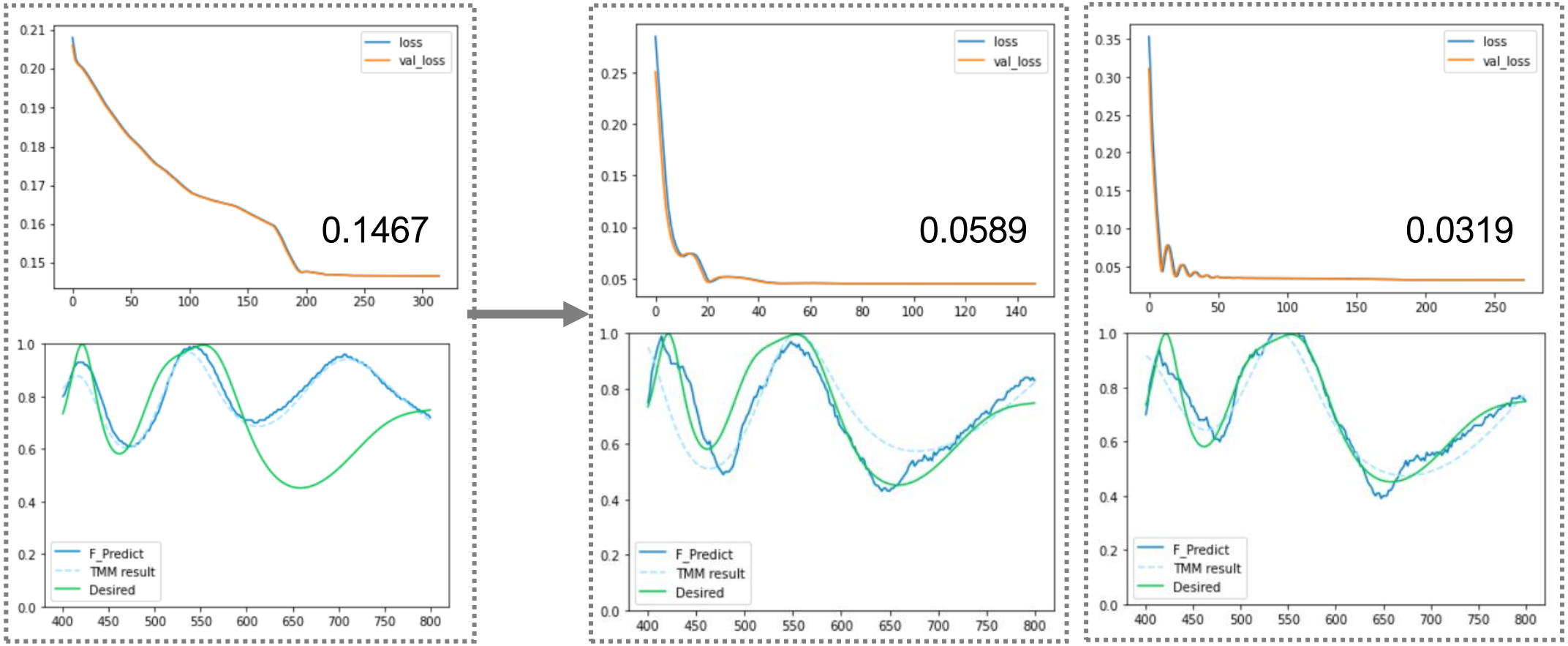
- Training (Optimizing) for one specific layer design … it's a trick!
- Input 1(1) – … – 8 – 300-300-300-300 – Output T_1(201) Forward model

```
w_model = model = keras.Sequential([
    layers.Dense(300, activation='relu', input_shape=input_shape),
    layers.Dense(8, activation='relu'),
])

model = keras.Sequential([
    w_model,
    f_model
])
f_model.trainable = False

model.compile(
    optimizer='adam',
    loss='mae',
)
```

```
compile_start = time.time()
history = model.fit(
    X_train, y_train,
    validation_data=(X_valid, y_valid),
    batch_size=1,
    epochs=1000,
    callbacks=[early_stopping],
    verbose=0,
)
```

2. Backward Model (T to L)

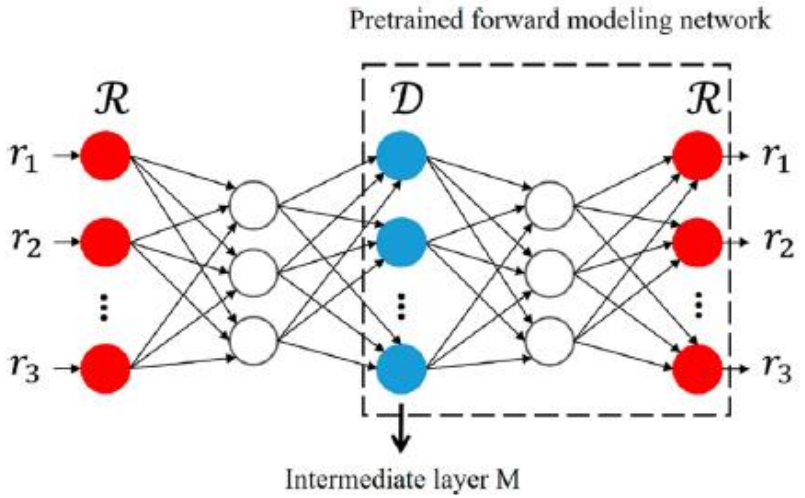
Had to try a lot of trials…



Forward model having problems in predicting near ‘training input boundary’?

```
Predicted Layer :  [[ 40.  71. 110.  40.  40.  40.  70.  94.]]
Desired Layer :  [113. 165.  90.  97.  63.  86. 150.  98.]
```

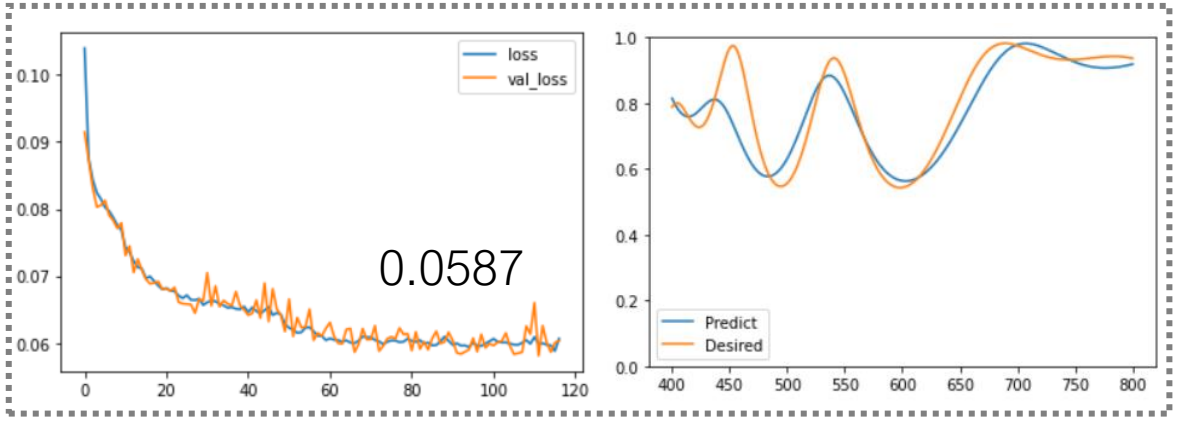
2. Backward Model (T to L)



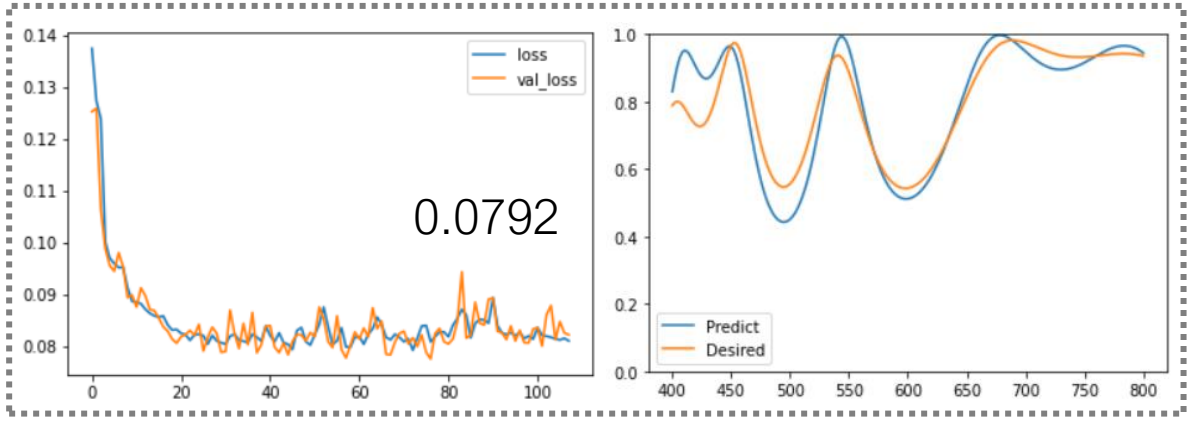
c. Forward model in **Transfer model**

- Input T, output T from dataset

① **200-500-200-20** vs 300-300-300-300



Test loss of T : 0.0643

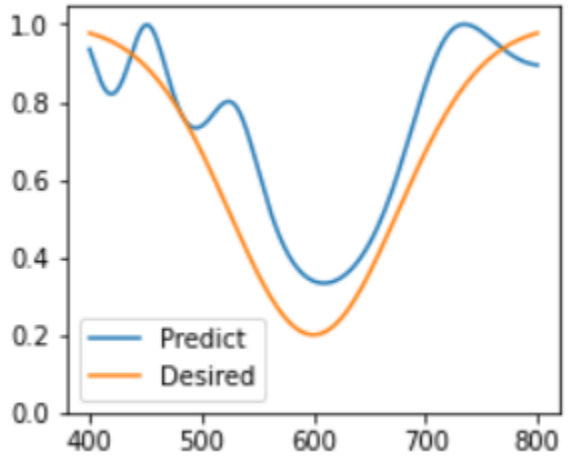


Test loss of T : 0.0974

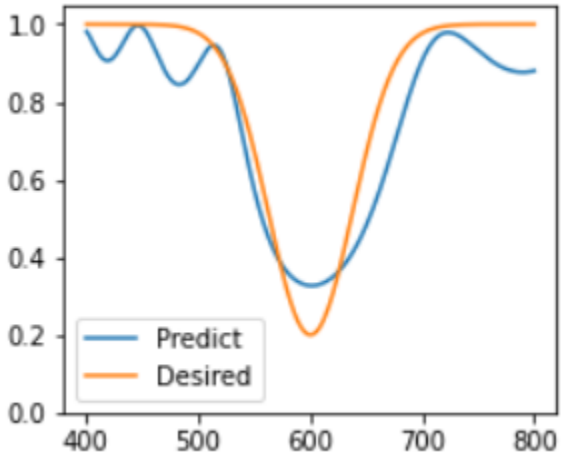
2. Backward Model (T to L)

- Artificial Spectrum (Gaussian)

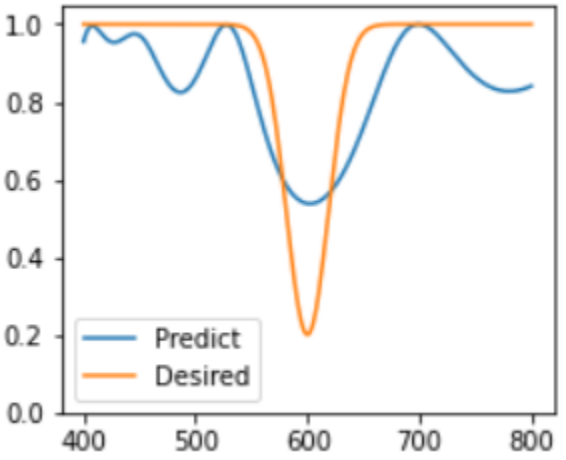
MAE error : 0.1234



MAE error : 0.0844



MAE error : 0.1115



Average MAE error : 0.1064

Conclusion & New Questions

- Implementing loss function using Forward model can be effective

- Compared to the direct transfer structure using Forward model

- For just one specific problem, optimization by DNN can be effective

- Choose appropriate model for user's purpose

- Prediction near the 'training input boundary' gets more incorrect?

- Tendency of spectrum mismatching near layer & wavelength boundaries