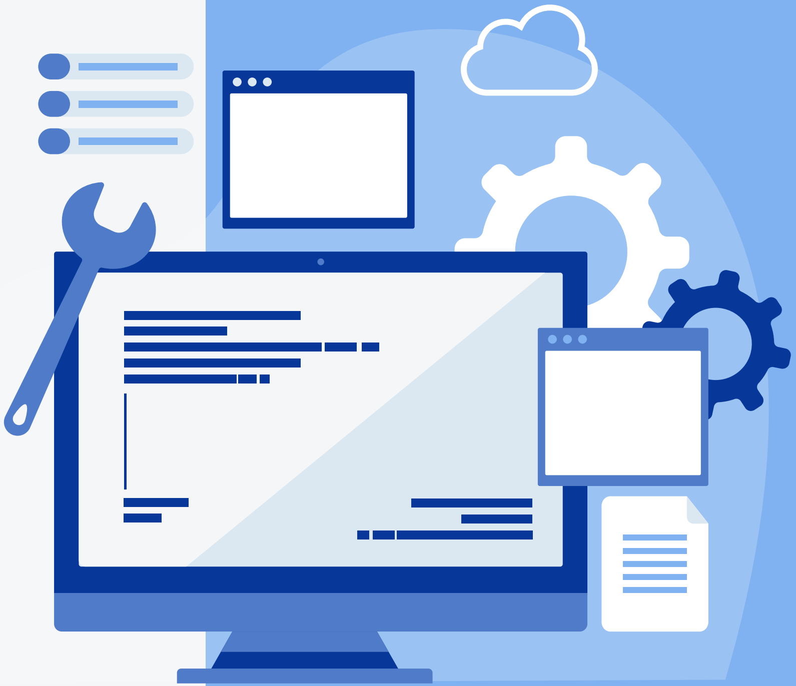
0+1 NJL model from an ANN framework

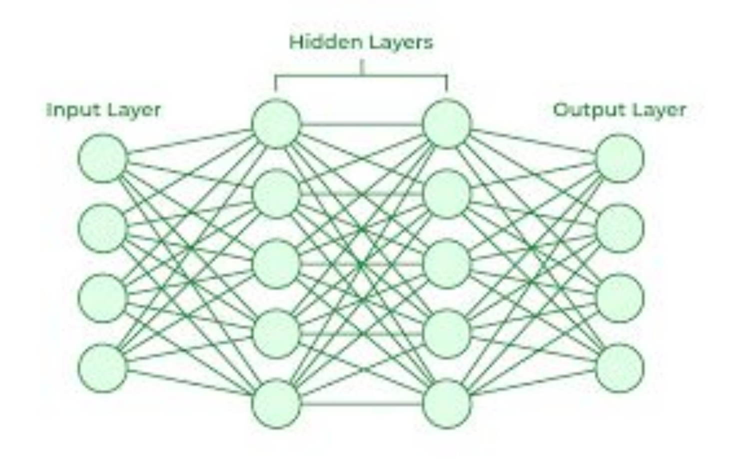


A Raya
IFM-UMSNH

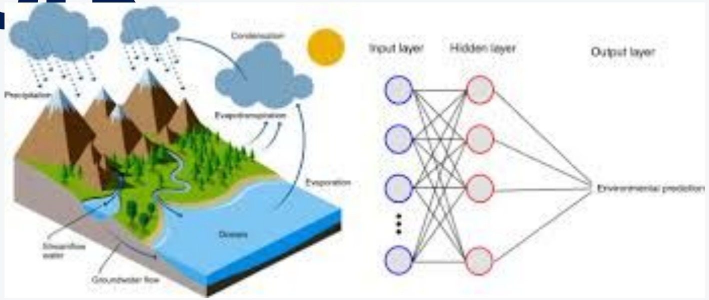
1st LA Workshop on Electromagnetic Effects in QCD

What is an ANN?

- An Artificial Newral Network (ANN) is a computer system modeled after the human brain's network of neurons
- It consists in interconnected nodes (neurons) organized into layers: input layer, hidden layers, and output layer
- ANNs are used to recognize patterns, make predictions, and solve complex problems



How does an ANN work?



- **Input Layer:** Receives data. Each neuron represents an input feature
- **Hidden Layers:** Process the input data. Neurons apply mathematical functions to the inputs.
- **Output Layer:** Produces the final prediction or classification.
- **Example:** Predicting Rainfall

  Inputs: Temperature, humidity, wind speed, atmospheric pressure

  Output: Probability of rain (Yes/No).

Steps to create an ANN framework

1. Define the Problem
2. Collect and Prepare Data
3. Design the Network Architecture
4. Initialize the Network
5. Choose Activation Functions
6. Define the Loss Function and Optimizer
7. Train the Network
8. Evaluate the Network
9. Deploy the Network

Step 1: Define the problem

- We want to predict whether it will rain based on weather conditions
- Our goal is to create an ANN that can accurately make this prediction
- Example:

  Inputs: Temperature, humidity, wind speed, atmospheric pressure

  Output: Probability of rain (Yes/No)

Step 2: Collect and Prepare Data

- **Gather historical weather data**

- **Preprocess the data:**
  - **- Normalize the data (e.g., scale temperature between 0 and 1)**
  - **- Split the data into training and testing sets**

- **Example:**

  Historical data for temperature, humidity, wind speed, and atmospheric pressure along with whether it rained or not.

Step 3: Design the Network Architecture

- Decide on the number of layers and neurons in each layer
- Simple architecture for our example:
  - Input Layer: 4 neurons (one for each weather parameter)
  - Hidden Layer: 5 neurons
  - Output Layer: 1 neuron (probability of rain)

Step 4: Initialize the Network

- **Randomly initialize the weights and biases**
- **Weights are adjusted during training to minimize prediction errors**
- **Example:**

  Initial weights are random values that will be fine-tuned

Step 5: Choose Activation Functions

- Activation functions introduce non-linearity to the network

- Common choices:

  - Hidden layers: ReLU (Rectified Linear Unit)
  - Output layer: Sigmoid (for binary classification)

- Example:

  Use ReLU for hidden layer and Sigmoid for output layer

Step 6: Define the Loss Function and Optimizer

- Loss function measures the error in predictions
  - Use binary cross-entropy for our example
- Optimizer updates the weights to minimize the loss
  - Use Adam optimizer

Step 7: Train the Network

- Feed the training data into the network
- Adjust the weights based on the error in predictions
- Iterate over multiple epochs to improve accuracy
- Example:
    - *Train the network with historical weather data to learn the patterns*

Step 8: Evaluate the Network

- Test the network on the testing set
- Calculate accuracy, precision, and recall to evaluate performance
- Example:
  - *Test with new weather data and check if the predictions match actual rainfall*

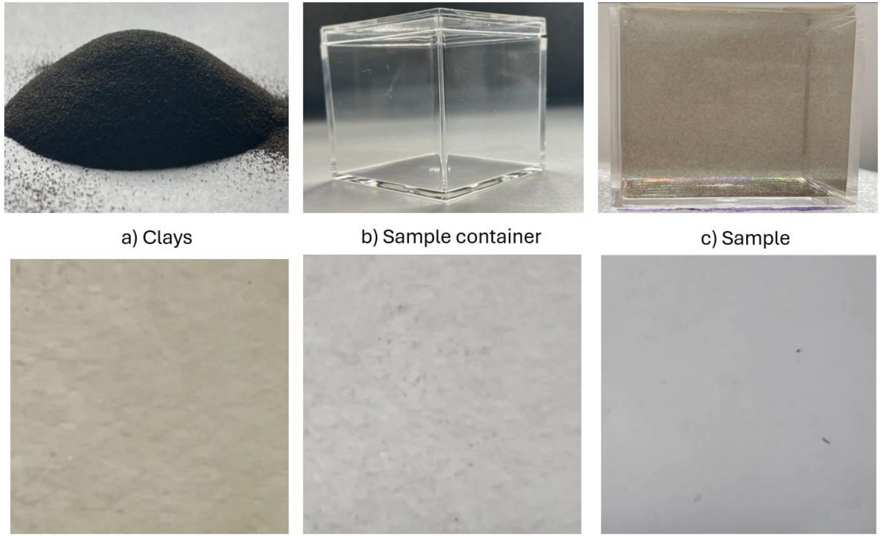
Step 9: Deploy the Network

- **Use the trained network to make predictions on new data**
- **Integrate the ANN into a weather prediction system**
- **Example:**
  - ***Use the ANN to predict if it will rain based on current weather conditions***

What did we learn?

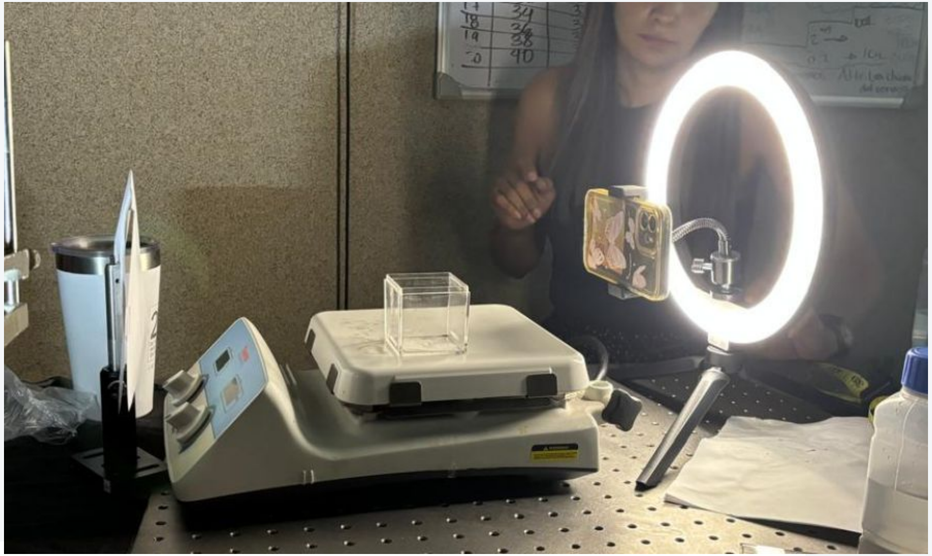
- ANNs are powerful tools for making predictions based on complex patterns in data
- Following the step-by-step process, we created an ANN to predict rainfall using weather data
- This framework can be adapted to solve various problems
- Example:
    - *- Beyond predicting rain, similar methods can be used for tasks like predicting stock prices, diagnosing medical conditions, and more*

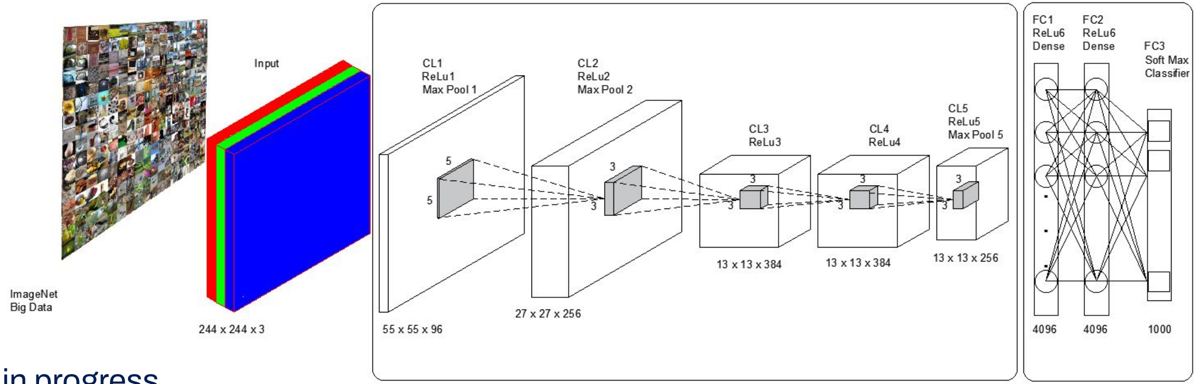
Example: Water polluted with solids


a) Clays b) Sample container c) Sample
d) High e) Medium f) Low

I Luviano, Y Concha, AR, work in progress

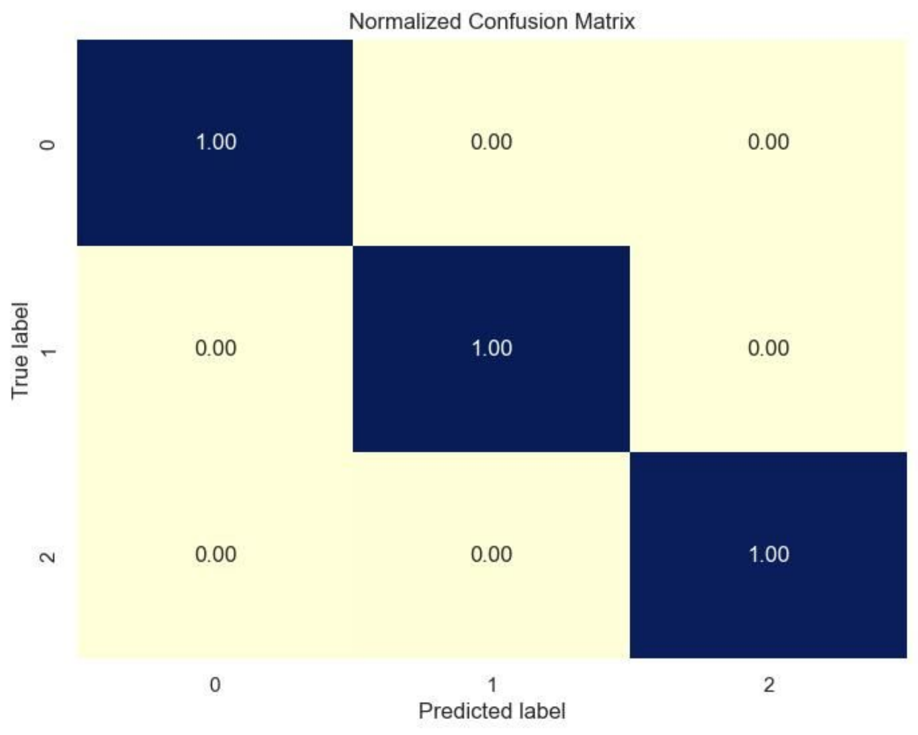
Example: Water polluted with solids





I Luviano, Y Concha, AR, work in progress

Example: Water polluted with solids



I Luviano, Y Concha, AR, work in progress

0+1 dimensional NJL model

Lagrangian

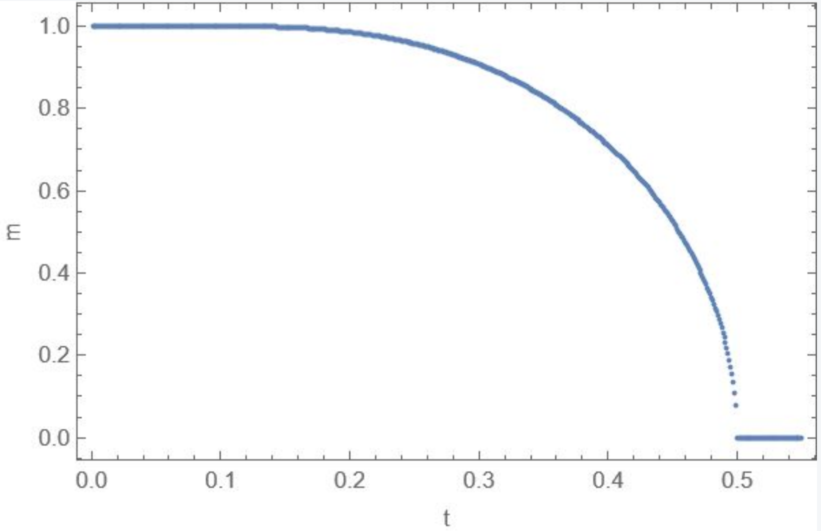
$$\mathcal{L} = \psi^\dagger \left(i\gamma_4 D^4 + im + i\mu\gamma_4\right)\psi + \frac{g^2}{2}\left[(\psi^\dagger\psi)^2 + (\psi^\dagger i\gamma_5\psi)^2\right]$$

Gap equation (chiral limit)

$$m^0 = 2G$$

Finite temperature

$$m = \tanh\left[\frac{m}{2t}\right]; \quad m = \frac{M}{m^0}; \quad t = \frac{T}{m^0}$$



R Pioquinto, S Hernández-Ortiz, AR, work in progress

0+1 dimensional NJL model

Lagrangian

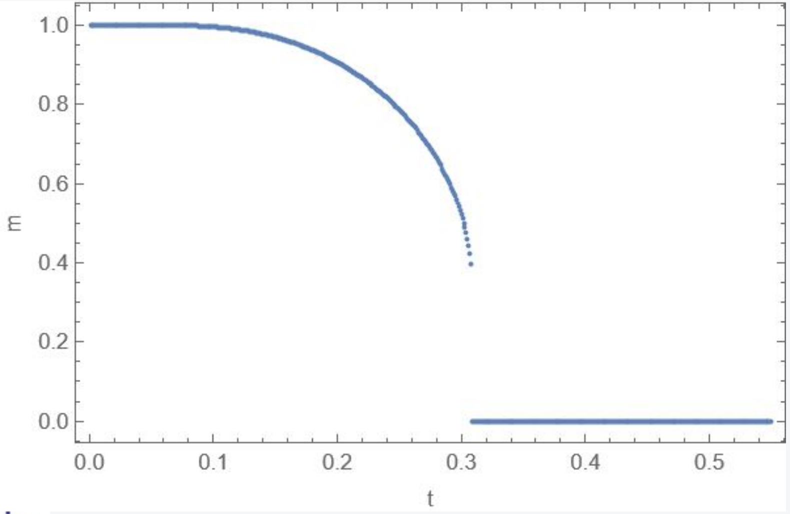
$$\mathcal{L} = \psi^\dagger\left(i\gamma_4 D^4 + im + i\mu\gamma_4\right)\psi + \frac{g^2}{2}\left[(\psi^\dagger\psi)^2 + (\psi^\dagger i\gamma_5\psi)^2\right]$$

Gap equation (chiral limit)

$$m^0 = 2G$$

Finite temperature and density

$$m = \frac{1}{2}\left[\tanh\left(\frac{m+\mu}{2t}\right) + \tanh\left(\frac{m-\mu}{2t}\right)\right]$$



R Pioquinto, S Hernández-Ortiz, AR, work in progress

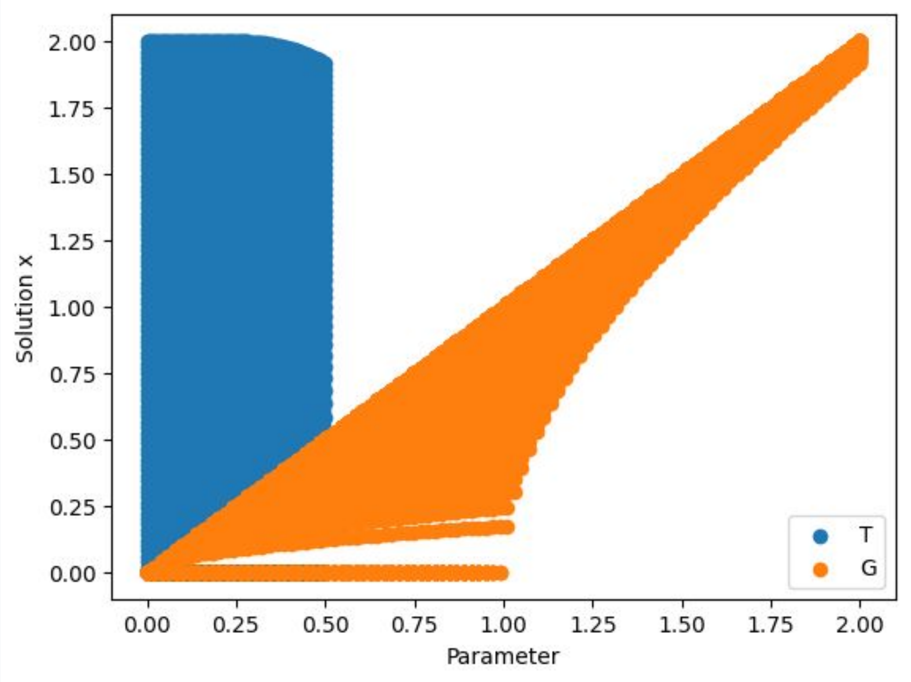
Step 1: Define the pro

- We want to solve a transcendental equation (gap equation) that depends on the coupling G and the temperature T to start with

$$1 = G \tanh \frac{m}{2t}$$

Step 2: Collect and Prepare Data

- We give several values of G and T and solve the gap equation even if solutions are unphysical



Step 3: Design the Network Architecture

- **Simple architecture for our problem:**
  - **Input Layer: 2 neurons, G and T**
  - **Hidden Layer: 2 layers, 64 neurons each**
  - **Output Layer: 1 neuron, the dynamical mass**

Step 4: Initialize the Network

- Randomly initialize the weights and biases
- Weights are adjusted during training to minimize prediction errors

```
# Define the neural network model
model = Sequential([
    Input(shape=(2,)),  # Input layer with 2 features (T and G)
    Dense(64, activation='relu'),
    Dense(64, activation='relu'),
    Dense(1)  # Output layer with 1 neuron (x)
])

model.compile(optimizer='adam', loss='mse')

# Train the model
model.fit(np.column_stack((T_flat, G_flat)), X_flat, epochs=100, batch_size=32, validation_split=0.2)
```

Step 5: Choose Activation Functions

- **We select:**
  - **Hidden layers: ReLU**
  - **Output layer**

```
# Define the neural network model
model = Sequential([
    Input(shape=(2,)),  # Input layer with 2 features (T and G)
    Dense(64, activation='relu'),
    Dense(64, activation='relu'),
    Dense(1)  # Output layer with 1 neuron (x)
])

model.compile(optimizer='adam', loss='mse')

# Train the model
model.fit(np.column_stack((T_flat, G_flat)), X_flat, epochs=100, batch_size=32, validation_split=0.2)
```

Step 6: Define the Loss Function and Optimizer

- Loss function
  - MSE
- Optimizer
  - Adam

```
# Define the neural network model
model = Sequential([
    Input(shape=(2,)),  # Input layer with 2 features (T and G)
    Dense(64, activation='relu'),
    Dense(64, activation='relu'),
    Dense(1)  # Output layer with 1 neuron (x)
])

model.compile(optimizer='adam', loss='mse')

# Train the model
model.fit(np.column_stack((T_flat, G_flat)), X_flat, epochs=100, batch_size=32, validation_split=0.2)
```

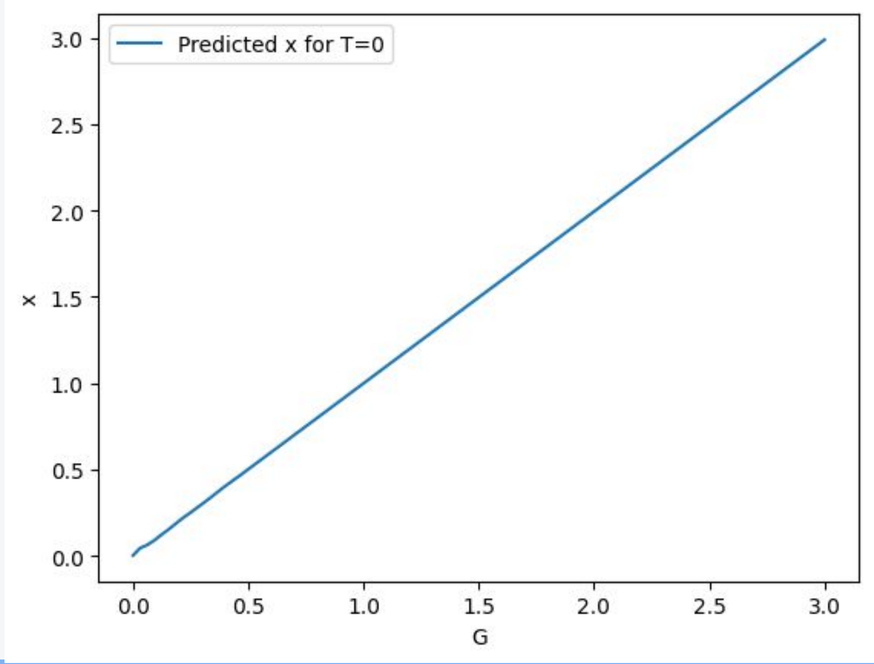
Step 7: Train the Network

- Feed the training data into the network (Sols of the gap eq. with several G's and T's
- Adjust the weights based on the error in predictions (MSE)
- Iterate up to 100 epochs

```
Epoch 95/100
250/250 [==============================] - 1s 3ms/step - loss: 3.3840e-05 - val_loss: 5.5604e-06
Epoch 96/100
250/250 [==============================] - 1s 3ms/step - loss: 3.2098e-05 - val_loss: 9.8271e-06
Epoch 97/100
250/250 [==============================] - 2s 9ms/step - loss: 7.4845e-05 - val_loss: 9.8643e-06
Epoch 98/100
250/250 [==============================] - 1s 2ms/step - loss: 5.9181e-05 - val_loss: 2.0244e-04
Epoch 99/100
250/250 [==============================] - 1s 2ms/step - loss: 5.5182e-05 - val_loss: 3.1287e-05
Epoch 100/100
250/250 [==============================] - 1s 2ms/step - loss: 5.1515e-05 - val_loss: 8.1444e-06
```

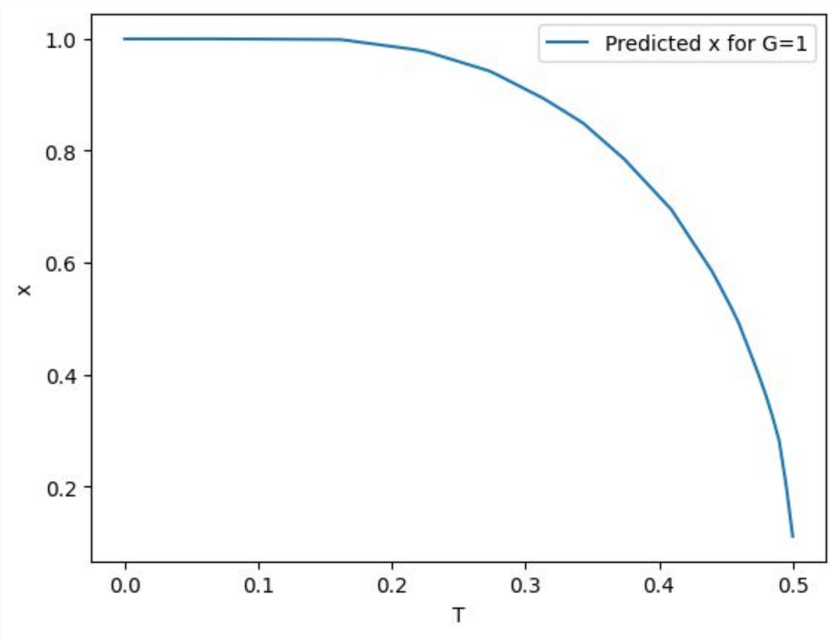
Step 8: Evaluate the Network

- Test the network on the testing set
- Calculate accuracy, precision, and evaluate performance



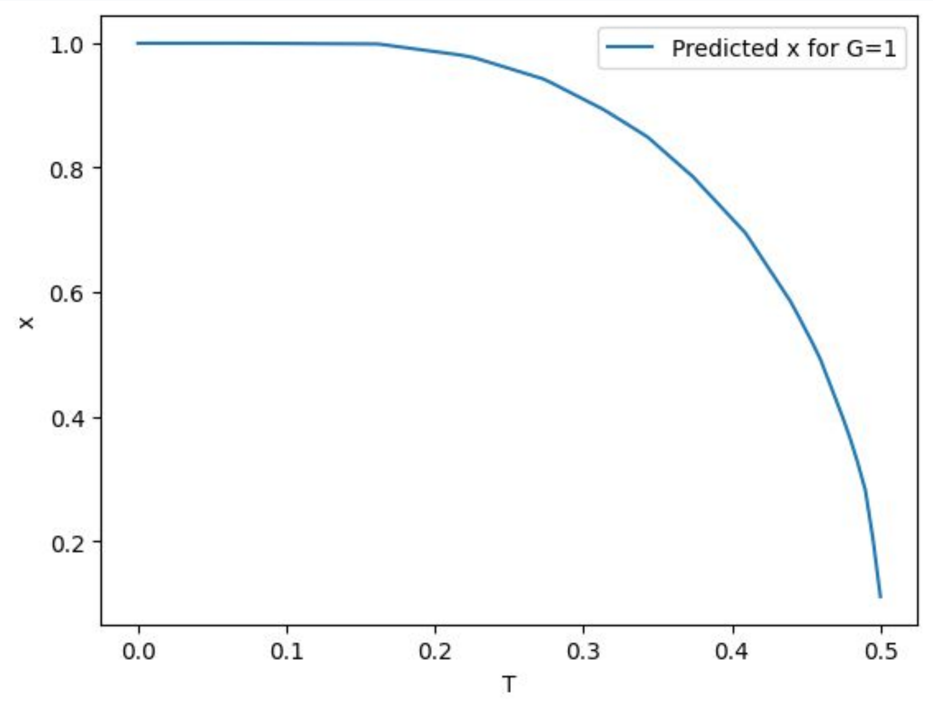
Step 8: Evaluate the Network

- Test the network on the testing set
- Calculate accuracy, precision, and evaluate performance



Step 9: Deploy the Network

- **Predictions**



Future Endeavors

- Introduce a chemical potential
- Work in higher dimensions
- Introduce additional parameters (magnetic fields, etc)
- Feed the ANN with physical information
- Compute other physical observables
- Develop a similar framework for other problems in QCD
    - *LSMq*
    - *FESR*
    - *Etc*

GRACIAS