A Term Paper On

Application of Multi-Linear Perceptron Neural Network for Predicting Antarctic Sea-Ice Concentration using Meteorological Variables

By

Meghraj Goswami

2022A2B41869H

Under the supervision of

Prof. Jagadeesh Anmala

Submitted in Partial Fulfilment of the Requirements of

CE F417: Application of AI in Civil Engineering

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(November 2025)

Acknowledgements

I would like to express my sincere gratitude to Professor Jagadeesh Anmala, for his guidance, support, and encouragement throughout the semester. His expertise and insights have been instrumental in enhancing my understanding of the subject matter.

Abstract

This project uses feedforward Multi-Layer Perceptron (MLP) Neural Network models for forecasting Sea-Ice concentration (SIC) using meteorological variables derived from ERA5 reanalysis: sea-surface temperature (SST), 2m air temperature (T2M), 10m u and v wind components (U10, V10) and mean sea-level pressure (MSL). The MLP models are trained using various combinations of input variables and model hyperparameters. The performance of the models is evaluated using statistical metrics such as Root Mean Square Error (RMSE), Mean Absolute Error (MAE) and Coefficient of Correlation (R). This project highlights the potential of MLP Neural Networks in accurately predicting SIC, which is crucial for understanding climate dynamics all over the world.

Contents

1 Introduction

Sea-ice plays an important role in the global climate system, influencing heat exchange, ocean circulation and weather patterns. Sea-Ice Concentration (SIC) in Antarctica varies seasonally, reaching its maximum in the Antarctic winter (July to September) and its minimum in the Antarctic summer (January to March). This variation is strongly influenced in non-linear ways by various meteorological variables, including temperature (SST), wind patterns (U10, V10), and atmospheric pressure (MSL). Accurate prediction of SIC is crucial for understanding climate dynamics and policy-making related to climate change [1].

Traditional methods for predicting SIC often rely on physical models that can be computationally expensive and require extensive calibration, and may not capture the complex non-linear relationships between SIC and meteorological variables. Moreover, the accuracy of these models can be limited by the resolution and quality of input data.

On the other hand, machine learning techniques, particularly neural networks, are known to capture complex patterns in data. Multi-Layer Perceptron (MLP) Neural Networks are a type of feedforward artificial neural network (ANN) that can model non-linear relationships between input and output variables. MLPs consist of multiple layers of interconnected neurons that can learn from large datasets through iterative training and optimization. Due to this, they are generally well-suited for cases where the input data is not at par with the quality or resolution required by physical models [2] [3].

This project aims to explore the application of MLP Neural Networks for predicting Antarctic sea-ice concentration using atmospheric and oceanic variables as input variables derived from ERA5 reanalysis data [4]. The performance of the MLP models are evaluated using various statistical metrics, and the results are compared to gauge the relative effectiveness of the models.

2 Study Area and Data

2.1 Study Area

The study area for this project is the Antarctic region, specifically the Southern Ocean surrounding the continent of Antarctica. Antarctica occupies an area of about 14 million square kilometers, which is mostly covered by 29 million cubic kilometers of ice - about 90% of the world's ice and 80% of its fresh water.

The Antarctic sea-ice cover is typically divided into five sectors: the Ross Sea, the Weddell Sea, the Indian Ocean sector, the West Pacific sector and the Bellingshausen-Amundsen Seas. Each sector exhibits distinct sea-ice characteristcs due to variations in local atmospheric and oceanic conditions [5].

The weather in Antarctica is characterized by high-speed katabatic winds, which are created due to cold and dense air blowing downhill from the polar plateau. These winds influence formation and distribution of sea-ice in the surrounding Southern Ocean [6] [7].

Antarctica also experiences extreme temperature patterns due to being plunged into darkness during the Antarctic winter and experiencing continuous sunlight during the Antarctic summer [8]. Depending on the region, temperatures in Antarctica fluctuate between -80°C in winter to 10°C in summer [9]. Precipitation is generally low, with most of it falling as snow.

Atmospheric pressure patterns in Antarctica also play a significant role in influencing sea-ice dynamics. The mainland is usually surrounded by a low-pressure belt, while the interior experiences high-pressure, creating conditions for the formation of the characteristic katabatic winds [6].

The seasons in Antarctica are opposite to those in the Northern Hemisphere, and are categorized as follows [5]:

- **Summer:** January to March
- **Autumn:** April to June
- **Winter:** July to September
- **Spring:** October to December

2.2 Data Source

The data used in this project is taken from the ERA5 reanalysis dataset, which is maintained by the European Centre for Medium-Range Weather Forecasts (ECMWF) [4]. The dataset uses historical meteorological observations and weather prediction models to generate comprehensive data, from 1940 to the present. It has a spatial resolution of 0.25° x 0.25° and a temporal resolution of 1 hour, although a monthly frequency has been chosen for this project. This makes it suitable for studying climate patterns and trends.

The specific variables used in this project are:

- Sea-Surface Temperature (SST)
- 2m Air Temperature (T2M)
- 10m U Wind Component (U10)
- 10m V Wind Component (V10)
- Mean Sea-Level Pressure (MSL)

The target variable for prediction is the Sea-Ice Concentration (SIC) in the Antarctic Southern ocean region:

- North: -55°
- South: -75°
- West: -180°
- East: 180°

The dataset spans a decade from January 2014 to December 2024, incorporating seasonal and inter-annual variability.

3 Literature Review

3.1 Sea-Ice Variability

Sea-ice variability in the Antarctic region has been a subject of extensive research in recent times due to its significant impact on global climate systems [10]. Studies have shown that sea-ice extent and concentration are influenced by a combination of atmospheric and oceanic factors, including temperature, wind patterns, and ocean currents [11]. For example, Parkinson and Cavalieri [5] analyzed satellite data to identify trends in Antarctic SIC, highlighting the complex relationship between natural variability and human influences.

Jena et al. [12] investigate the mechanisms behind sea-ice expansion in the Indian Ocean sector of Antarctica, emphasizing the role of meteorological variables such as air temperature and wind patterns. Their findings suggest that changes in these variables can lead to significant variations in sea-ice concentration, signifying the need for accurate predictive models.

3.2 Neural Networks in Climate Science

Machine learning techniques, particularly neural networks, have gained prominence in climate science for their ability to model complex, non-linear relationships in large datasets. MLP Neural Networks have been successfully applied in various climate-related prediction tasks, including temperature forecasting, precipitation prediction, and sea-ice extent estimation.

Gardner and Dorling [2] provide a comprehensive review of the applications of MLP Neural Networks in atmospheric sciences, highlighting their effectiveness in capturing non-linear patterns in atmospheric data. Similarly, Liu et al. [3] discuss the broader applications of artificial neural networks in global climate change research, emphasizing their potential in improving predictive accuracy.

These studies highlight the potential of MLP Neural Networks in climate science, particularly for accurate predictions. However, there remains a need for further research to optimize model architectures and hyperparameters for specific applications, such as Antarctic SIC prediction.

4 Methods and Methodology

4.1 Tools and Software

The following tools and software are used in this project:

- **Primary language:** Python 3.13.7 [13]
- **Neural network library:** scikit-learn (MLPRegressor) [14]
- **Data handling:** xarray, NumPy
- **Preprocessing:** StandardScaler, SimpleImputer
- **Hyperparameter tuning:** GridSearchCV
- **Model serialization:** joblib
- **GUI:** DearPyGUI
- **Dataset format:** NetCDF (ERA5)

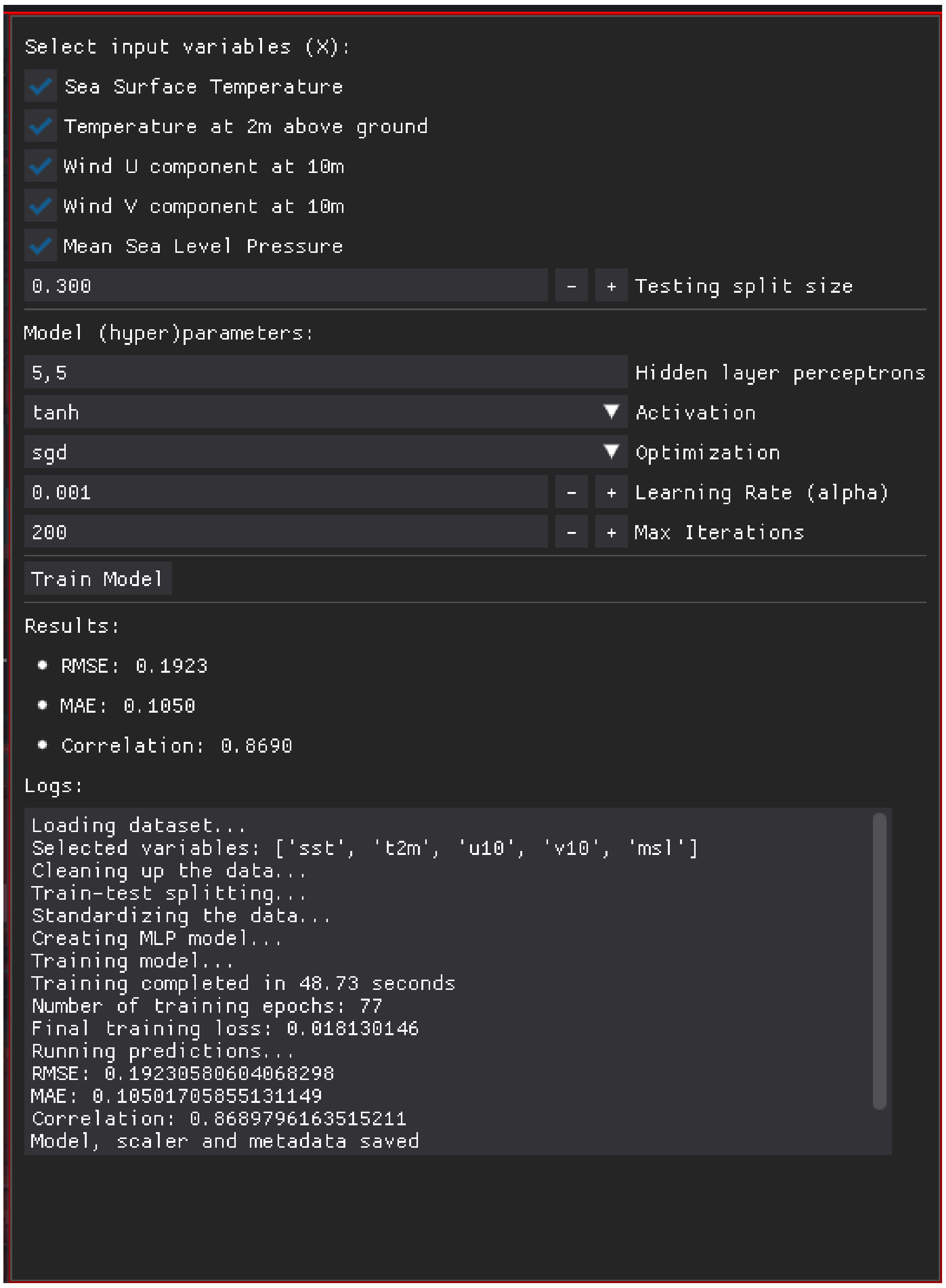
All computations were performed on a Linux 6.17.8 system running Arch Linux, with multi-core CPU acceleration enabled during grid search. The Python codebase has been added to a public GitHub repository [15], as well as in the Appendix section of this report.

A graphical user interface (GUI) has also been developed using DearPyGUI to enable easy and interactive experimentation with different hyperparameter configurations, without modifying the codebase directly.

4.2 Data Preprocessing

In its raw form, the ERA5 dataset contains missing values, outliers and inconsistencies that need to be taken care of before training the MLP models. The following preprocessing steps are applied:

- **Handling Missing Values:** Missing values in the dataset are handled using imputing, interpolation and removal of records with excessive missing data. The mean of data points is used for imputation.
- **Train-Test Splitting:** The dataset is split into training (70%) and testing (30%) sets. The training set is used to train the MLP models, while the testing set is used to evaluate their performance.
- **Standardization:** The input variables are standardized, or in other words they are rescaled to have a mean of 0 and a standard deviation of 1. This ensures that all input variables contribute equally when training the model.


Select input variables (X):

- ✓ Sea Surface Temperature
- ✓ Temperature at 2m above ground
- ✓ Wind U component at 10m
- ✓ Wind V component at 10m
- ✓ Mean Sea Level Pressure

0.300 Testing split size

Model (hyper)parameters:

5,5 Hidden layer perceptrons
tanh Activation
sgd Optimization
0.001 Learning Rate (alpha)
200 Max Iterations

Results:

- RMSE: 0.1923
- MAE: 0.1050
- Correlation: 0.8690

Logs:

```
Loading dataset...
Selected variables: ['sst', 't2m', 'u10', 'v10', 'msl']
Cleaning up the data...
Train-test splitting...
Standardizing the data...
Creating MLP model...
Training model...
Training completed in 48.73 seconds
Number of training epochs: 77
Final training loss: 0.018130146
Running predictions...
RMSE: 0.19230580604068298
MAE: 0.10501705855131149
Correlation: 0.8689796163515211
Model, scaler and metadata saved
```

Figure 1: Graphical User Interface for sea-ice concentration prediction

4.3 MLP Neural Network Architecture

A Multi-Layer Perceptron (MLP) Neural Network is a type of feedforward artificial neural network composed of an input layer, one or more hidden layers, and an output layer. Each layer consists of multiple neurons, and neurons in one layer are connected to those in the next layer through weighted connections. Backpropagation is used to train the network by adjusting the weights based on the error difference between predicted and actual outputs. MLPs are widely used for supervised learning problems, such as regression and classification.

The MLP Neural Network architecture used in this project comprises the following components:

- **Input Layer:** The input layer consists of neurons corresponding to the number of meteorological variables used as input features (maximum of 5 in this case).
- **Hidden Layers:** The MLP architecture includes a number of hidden layers, each with varying number of neurons. The number of hidden layers and neurons per layer are hyperparameters that are defined by the user and can be tuned for optimal performance.
- **Output Layer:** The output layer consists of a single neuron that predicts the Sea-Ice Concentration (SIC) value.
- **Learning Rate:** The learning rate (alpha) controls the step size while updating the model weights during training. A suitable learning rate is chosen to ensure convergence.
- **Activation Functions:** Non-linear activation functions such as Rectified Linear Unit (ReLU), Sigmoidal or Hyperbolic Tangent (TanH) are used in the hidden layers to introduce non-linearity into the model.
- **Loss Function:** Mean Squared Error (MSE) is used as the loss function to measure the difference between predicted and actual SIC values during training.
- **Optimization Algorithm:** Based on the grid search results, Stochastic Gradient Descent (SGD) optimizer is used to update the weights of the network during training. Other optimizers like Adaptive Moment Estimation (Adam) can also be used.

4.4 Model Training and Evaluation

The MLP Neural Network models are trained using the preprocessed training dataset. The training process involves multiple iterations, or epochs, where the model iteratively perturbs and adjusts its weights to minimize the output of the loss function. The following steps are involved in model training and evaluation:

The trained models are evaluated using the testing dataset. The following statistical metrics are used to assess model performance:

- Root Mean Square Error (RMSE)
- Mean Absolute Error (MAE)
- Coefficient of Correlation (R)

The performance of different MLP models is also empirically compared based on the evaluation metrics using the following formula:

$$\frac{1}{RMSE} + \frac{1}{MAE} + 10R \tag{1}$$

5 Results

To fine-tune the optimal hyperparameters for the MLP Neural Network, an exhaustive grid search was conducted over a user-defined hyperparameter space. The hyperparameters considered in the grid search include:

- Input Variables: SST, T2M, U10, V10, MSL
- Architecture:
  - One-Layer and Two-Layer Networks
  - 2-6 Neurons in each hidden layer
- Activation Functions: ReLU, TanH, Sigmoidal
- Optimization Schemes: Adam, SGD
- Learning Rates: 0.001, 0.005, 0.01
- Maximum Epochs: 200
- Cross-Validation: 3-Fold

For the one-layer networks, a total of $4 * 3 * 2 * 3 = 72$ candidate models are tested, amounting to 216 training runs with 3-fold cross-validation. For the two-layer networks, a total of $16 * 3 * 2 * 3 = 288$ candidate models are tested, amounting to 864 training runs with 3-fold cross-validation.

5.1 Best One-Layer Model

Out of the defined hyperparameter space, the best-performing one-layer MLP model based on the evaluation metrics had the following hyperparameters:

- Input Variables: SST, T2M, U10, V10, MSL
- Architecture: One hidden layer with 4 Neurons
- Activation Function: TanH
- Optimization Scheme: SGD
- Learning Rate: 0.005
- Maximum Epochs: 200

The performance of this model on the testing dataset is as follows:

- Epochs to Convergence: 51
- Training Time: 30.02 seconds
- Final Training Loss (MSE): 0.0345
- RMSE: 0.1919
- MAE: 0.1099
- R: 0.8705

5.2 Best Two-Layer Model

Out of the defined hyperparameter space, the best-performing two-layer MLP model based on the evaluation metrics had the following hyperparameters:

- Input Variables: SST, T2M, U10, V10, MSL
- Architecture: Two hidden layers with 5 Neurons each
- Activation Function: TanH
- Optimization Scheme: SGD
- Learning Rate: 0.001
- Maximum Epochs: 200

The performance of this model on the testing dataset is as follows:

- Epochs to Convergence: 77
- Training Time: 50.44 seconds
- Final Training Loss (MSE): 0.0181
- RMSE: 0.1923
- MAE: 0.1050
- R: 0.8690

6 Discussion

The results obtained from the grid searches indicate that both the best one-layer and two-layer MLP models constrained in the defined hyperparameter space performed comparably well in predicting Antarctic SIC using meteorological variables. The RMSE and MAE values for both models are very similar, with the one-layer model slightly outperforming the two-layer model in terms of RMSE, while the two-layer model had a slightly better MAE. The R values for both models are also very close, indicating that both models explain a significant portion of the variance in the SIC data.

The choice of activation function (TanH) and optimization scheme (SGD) appears to be effective for this specific problem, as both models utilized these configurations. The learning rates chosen for the best models (0.005 for one-layer and 0.001 for two-layer) also seem to have contributed positively to the training process.

The marginal differences in performance metrics suggest that increasing the complexity of the model by adding an additional hidden layer did not lead to a significant improvement. Therefore, guided by the principle of parsimony, the simpler one-layer network is the preferred model, and is termed a parsimonious model. This is likely because the relationship between the input variables and SIC is not complex enough to warrant a higher dimensional network, or the additional layer led to overfitting on the training data.

Overall, both models demonstrate the capability of MLP Neural Networks in capturing the non-linear relationships between input variables and the target. However, with near identical performane, the parsimonious model (the one-layer network) is preferred for the balance between simplicity and predictive accuracy for this specific application.

7 Conclusion

This project explored the application of MLP Neural Networks for predicting Antarctic SIC using five key meteorological variables derived from ERA5 reanalysis data. An exhaustive grid search was conducted over a defined hyperparameter space to identify the best-performing one-layer and two-layer MLP models.

The best performing models used:

- Hyperbolic Tangent Activation Function
- Stochastic Gradient Descent Optimization Scheme

and achieved:

- RMSE: 0.192
- MAE: 0.107
- R: 0.87

The results indicate that even simple networks can effectively model large-scale climate phenomena like sea-ice variability when trained on physically meaningful input variables. More complex architectures did not yield significant improvements while increasing computational costs. This highlights the importance of model parsimony in machine learning applications for climate science.

Future work could explore additional input variables, longer training periods, and alternative neural network architectures like Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs) to further enhance predictive capabilities. In conclusion, this project demonstrates the potential of MLP Neural Networks as a valuable tool for climate modeling and prediction tasks.

A Python Code Used

The complete Python codebase used for data preprocessing, model training, hyperparameter tuning and evaluation has also been made available on GitHub [15].

A.1 Base Code (main.py)

```python
import xarray as xr, numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.impute import SimpleImputer
import joblib, os, time, json

def flatten(var):
    return var.values.reshape(len(var.valid_time), -1)

def save_model(model, scaler, metadata, name):
    if not os.path.exists("models"): os.makedirs("models")
    if not os.path.exists("metadata"): os.makedirs("metadata")
    if not os.path.exists("scalers"): os.makedirs("scalers")

    joblib.dump(model, f"models/{name}_model.pkl")
    joblib.dump(scaler, f"scalers/{name}_scaler.pkl")
    with open(f"metadata/{name}_meta.json", "w") as f:
        json.dump(metadata, f, indent=4)

    print(f"Model, scaler and metadata saved for {name}")

X_vars = {
    "sst": "Sea Surface Temperature",
    "t2m": "Temperature at 2m above ground",
    "u10": "Wind U component at 10m",
    "v10": "Wind V component at 10m",
    "msl": "Mean Sea Level Pressure",
}

activation_fns = ["relu", "logistic", "tanh", "identity"]
optimization_fns = ["adam", "sgd", "lbfgs"]

print("Loading dataset...")
ds = xr.open_dataset("data_2014-2024.nc")
ds = ds.sel(latitude=slice(-55, -75))

X_vars = ["sst", "t2m", "u10", "v10", "msl"]
X_list = [flatten(ds[v]) for v in X_vars]

X = np.hstack(X_list)
Y = flatten(ds["siconc"])

# Cleaning data
print("Cleaning up the data...")

X = X[:, ~np.isnan(X).all(axis=0)]
Y = Y[:, ~np.isnan(Y).all(axis=0)]
```

```
X = SimpleImputer(strategy='mean').fit_transform(X)
Y = np.nan_to_num(Y, nan=0.0)

# Train-test split
print("Train-test splitting...")

X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.3, shuffle=False
)

# Standardizing data
print("Standardizing the data...")

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test  = scaler.transform(X_test)

# MLP Model parameter ranges

hidden_layers = []
activation = activation_fns[0]
optimization = optimization_fns[0]
alpha = 0.001
max_iter = 200

n_layers = 1
print('Note: Enter "done" when finished adding layers')

while True:
    range_n_neurons = input(f"Enter range of neurons in layer {n_layers}: "
                                  )
    if range_n_neurons.lower() == "done":
        break
    try:
        start, end = map(int, range_n_neurons.split(","))
        hidden_layers.append((start, end))
        n_layers += 1
    except:
        print("Invalid input. Please enter two integers separated by a
                                      comma.")

print("Creating MLP model...")
model = MLPRegressor(
    hidden_layer_sizes=hidden_layers,
    activation=activation,
    solver=optimization,
    learning_rate_init=alpha,
    max_iter=max_iter,
    verbose=True,
    random_state=1869
)

# Train
print("Training model...")
t0 = time.time()
model.fit(X_train, Y_train)
tf = time.time() - t0
print(f"Training completed in {tf:.2f} seconds")
print("Number of training epochs: " + str(model.n_iter_))
print("Final training loss: " + str(model.loss_))

# Predict
print("Running predictions...")
pred = np.clip(model.predict(X_test), 0, 1)

# Metrics
rmse = np.sqrt(mean_squared_error(Y_test, pred))
mae = mean_absolute_error(Y_test, pred)
corr = np.corrcoef(Y_test.flatten(), pred.flatten())[0, 1]

print(f"RMSE: {rmse}")
print(f"MAE: {mae}")
print(f"Correlation: {corr}")

metadata = {
    "variables": X_vars,
    "hidden_layers": hidden_layers,
    "activation": activation,
    "solver": optimization,
    "learning_rate": alpha,
    "max_iter": max_iter,
    "epochs": model.n_iter_,
    "rmse": rmse,
    "mae": mae,
    "corr": corr,
    "training_time": tf,
    "timestamp": time.time(),
}

# Save model
model_name = f"mlp_{hidden_layers}_{activation}_{optimization}_{alpha:.4f}_
                                     {model.n_iter_}"
save_model(model, scaler, metadata, model_name)
print(f"Saved model as: {model_name}")
```

A.2 GUI (gui.py)

```
import dearpygui.dearpygui as dpg
import xarray as xr, numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.impute import SimpleImputer
import joblib, os, time, json

def log(msg):
    dpg.set_value("log_window", dpg.get_value("log_window") + msg + "\n")

def flatten(var):
    return var.values.reshape(len(var.valid_time), -1)

def save_model(model, scaler, metadata, name):
    if not os.path.exists("models"): os.makedirs("models")
    if not os.path.exists("metadata"): os.makedirs("metadata")
    if not os.path.exists("scalers"): os.makedirs("scalers")

    joblib.dump(model, f"models/{name}_model.pkl")
    joblib.dump(scaler, f"scalers/{name}_scaler.pkl")
    with open(f"metadata/{name}_meta.json", "w") as f:
        json.dump(metadata, f, indent=4)

    log(f"Model, scaler and metadata saved")

X_vars = {
    "sst": "Sea Surface Temperature",
    "t2m": "Temperature at 2m above ground",
    "u10": "Wind U component at 10m",
    "v10": "Wind V component at 10m",
    "msl": "Mean Sea Level Pressure",
}

activation_fns = ["relu", "logistic", "tanh", "identity"]
optimization_fns = ["adam", "sgd", "lbfgs"]

def run_training():
    try:
        dpg.set_value("log_window", "")

        log("Loading dataset...")
        ds = xr.open_dataset("data_2014-2024.nc")
        ds = ds.sel(latitude=slice(-55, -75))

        selected_vars = [var for var in X_vars.keys() if dpg.get_value(f"
                                                   chk_{var}")]
        if len(selected_vars) == 0:
            log("ERROR: Select at least one X variable.")
            return
        log(f"Selected variables: {selected_vars}")

        X_list = [flatten(ds[v]) for v in selected_vars]
        X = np.hstack(X_list)
        Y = flatten(ds["siconc"])

        log("Cleaning up the data...")

        # Remove NaN columns
        X = X[:, ~np.isnan(X).all(axis=0)]
        Y = Y[:, ~np.isnan(Y).all(axis=0)]

        # Impute and fill NaNs
        X = SimpleImputer(strategy='mean').fit_transform(X)
        Y = np.nan_to_num(Y, nan=0.0)

        # Train-test split
        log("Train-test splitting...")
        X_train, X_test, Y_train, Y_test = train_test_split(
            X, Y, test_size=dpg.get_value("ttsplit"), shuffle=False
        )

        # Standardizing
        log("Standardizing the data...")
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_test  = scaler.transform(X_test)

        # Read (hyper)parameters from GUI
        hidden_layers = tuple(map(int, dpg.get_value("lst_hidden_layers").
                                           split(",")))
        activation = dpg.get_value("activation_combo")
        optimization = dpg.get_value("optimization_combo")
        alpha = dpg.get_value("alpha")
        max_iter = dpg.get_value("max_iters")

        # Create model
        log("Creating MLP model...")
        model = MLPRegressor(
            hidden_layer_sizes=hidden_layers,
            activation=activation,
            solver=optimization,
            learning_rate_init=alpha,
            max_iter=max_iter,
            verbose=True
        )

        # Train
        log("Training model...")
        t0 = time.time()
        model.fit(X_train, Y_train)
        tf = time.time() - t0
        log(f"Training completed in {tf:.2f} seconds")
        log("Number of training epochs: " + str(model.n_iter_))
        log("Final training loss: " + str(model.loss_))

        # Predict
        log("Running predictions...")
        pred = np.clip(model.predict(X_test), 0, 1)
        # pred = model.predict(X_test)
        # log(f"First 5 predicted sea-ice conc: {pred[:5]}")
        # log(f"First 5 actual sea-ice conc: {Y_test[:5]}")

        # Metrics
        rmse = np.sqrt(mean_squared_error(Y_test, pred))
        mae = mean_absolute_error(Y_test, pred)
        corr = np.corrcoef(Y_test.flatten(), pred.flatten())[0, 1]

        log(f"RMSE: {rmse}")
        log(f"MAE: {mae}")
        log(f"Correlation: {corr}")

        dpg.set_value("rmse_val", f"{rmse:.4f}")
        dpg.set_value("mae_val", f"{mae:.4f}")
        dpg.set_value("corr_val", f"{corr:.4f}")

        metadata = {
            "variables": selected_vars,
            "hidden_layers": hidden_layers,
            "activation": activation,
            "solver": optimization,
            "learning_rate": alpha,
            "max_iter": max_iter,
            "epochs": model.n_iter_,
            "rmse": rmse,
            "mae": mae,
            "corr": corr,
            "training_time": tf,
            "timestamp": time.time(),
        }

        # Save model
        model_name = f"mlp_{hidden_layers}_{activation}_{optimization}_{
                                      alpha:.4f}_{model.n_iter_}"
        save_model(model, scaler, metadata, model_name)
        log(f"Saved model as: {model_name}")

    except BaseException as e:
        log(f"ERROR:\n{e}")

dpg.create_context()

with dpg.window(tag="Sea-Ice Prediction MLP"):
    dpg.add_text("Select input variables (X):")
    for var in X_vars.keys():
        dpg.add_checkbox(label=X_vars[var], default_value=True, tag=f"chk_{
                                      var}")

    dpg.add_input_float(label="Testing split size",default_value=0.3, tag="
                                  ttsplit")

    dpg.add_separator()

    dpg.add_text("Model (hyper)parameters:")
    dpg.add_input_text(label="Hidden layer perceptrons (comma separated)",
                                  default_value="5,5,5", tag="
                                  lst_hidden_layers")
    dpg.add_combo(activation_fns, label="Activation", default_value=
                                  activation_fns[0], tag="
                                  activation_combo")
    dpg.add_combo(optimization_fns, label="Optimization", default_value=
                                  optimization_fns[0], tag="
                                  optimization_combo")
    dpg.add_input_float(label="Learning Rate (alpha)", default_value=0.001,
                                   tag="alpha", step=0.001)
    dpg.add_input_int(label="Max Iterations", default_value=200, tag="
                                  max_iters")

    dpg.add_separator()
    dpg.add_button(label="Train Model", callback=run_training)
    dpg.add_separator()

    dpg.add_text("Results:")
    dpg.add_text("RMSE:", bullet=True); dpg.add_same_line(); dpg.add_text("
                                  ", tag="rmse_val")
    dpg.add_text("MAE:", bullet=True); dpg.add_same_line(); dpg.add_text(""
                                  , tag="mae_val")
    dpg.add_text("Correlation:", bullet=True); dpg.add_same_line(); dpg.
                                  add_text("", tag="corr_val")
    dpg.add_text("Logs:")
    dpg.add_input_text(tag="log_window", multiline=True, width=500, height=
                                     200, readonly=True)

dpg.create_viewport(title="Sea-Ice Prediction MLP")
dpg.setup_dearpygui()
dpg.show_viewport()
dpg.set_primary_window("Sea-Ice Prediction MLP", True)
dpg.start_dearpygui()
dpg.destroy_context()
```

A.3 Grid Search (gridsearch.py)

```
import xarray as xr, numpy as np, pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.impute import SimpleImputer
import joblib, os, time, json, itertools

def flatten(var):
    return var.values.reshape(len(var.valid_time), -1)

def save_model(model, scaler, metadata, name):
    if not os.path.exists("multi_models"): os.makedirs("multi_models")
    if not os.path.exists("multi_metadata"): os.makedirs("multi_metadata")
    if not os.path.exists("multi_scalers"): os.makedirs("multi_scalers")

    joblib.dump(model, f"multi_models/{name}_model.pkl")
    joblib.dump(scaler, f"multi_scalers/{name}_scaler.pkl")
    with open(f"multi_metadata/{name}_meta.json", "w") as f:
        json.dump(metadata, f, indent=4)

    print(f"Model, scaler and metadata saved for {name}")

X_vars_dict = {
    "sst": "Sea Surface Temperature",
    "t2m": "Temperature at 2m above ground",
    "u10": "Wind U component at 10m",
    "v10": "Wind V component at 10m",
    "msl": "Mean Sea Level Pressure",
}

print("Loading dataset...")
ds = xr.open_dataset("data_2014-2024.nc")
ds = ds.sel(latitude=slice(-55, -75))

X_vars = ["sst", "t2m", "u10", "v10", "msl"]
X_list = [flatten(ds[v]) for v in X_vars]

X = np.hstack(X_list)
Y = flatten(ds["siconc"])
```

```
# Cleaning data
print("Cleaning up the data...")
X = X[:, ~np.isnan(X).all(axis=0)]
Y = Y[:, ~np.isnan(Y).all(axis=0)]

X = SimpleImputer(strategy='mean').fit_transform(X)
Y = np.nan_to_num(Y, nan=0.0)

# Train-test split
print("Train-test splitting...")
X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.3, shuffle=False
)

# Standardizing data
print("Standardizing the data...")
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test  = scaler.transform(X_test)

# Hidden Layer Neuron Ranges
neuron_ranges = []
n_layers = 1
print('\n=== Layer Configuration ===')
print('Enter ranges (e.g., "4,8") to test all sizes between 4 and 8.')
print('Type "done" when you have defined all layers.')

while True:
    range_input = input(f"Enter range for Layer {n_layers} (start,end): ")
    if range_input.lower() == "done":
        break
    try:
        start, end = map(int, range_input.split(","))
        neuron_ranges.append(range(start, end + 1))
        n_layers += 1
    except ValueError:
        print("Invalid input. Please enter two integers separated by a
                                                  comma (e.g., 4,8).")

hidden_layer_combinations = list(itertools.product(*neuron_ranges))

print(f"\nGenerated {len(hidden_layer_combinations)} combinations to test."
                                      )

# Parameter Grid
param_grid = {
    'hidden_layer_sizes': hidden_layer_combinations,
    'activation': ["relu", "tanh", "logistic"],
    'solver': ["adam", "sgd"], # , "lbfgs"
    'learning_rate_init': [0.001, 0.005, 0.01],
    'max_iter': [200]
}

print("\nStarting Grid Search...")
print("Note: This may take a while :)")

mlp = MLPRegressor(random_state=1869)
grid_search = GridSearchCV(
    estimator=mlp,
    param_grid=param_grid,
    cv=3,       # 3-fold cross-validation
    scoring='neg_mean_squared_error',
    verbose=2,
    n_jobs=-1   # use all CPU cores
)

t0 = time.time()
grid_search.fit(X_train, Y_train)
tf = time.time() - t0

# Best Model Evaluation
print(f"\nGrid Search completed in {tf:.2f} seconds")
print(f"Best Parameters found: {grid_search.best_params_}")

best_model = grid_search.best_estimator_
print("Running predictions on Test Set with best model...")
pred = np.clip(best_model.predict(X_test), 0, 1)

# Metrics
rmse = np.sqrt(mean_squared_error(Y_test, pred))
mae = mean_absolute_error(Y_test, pred)
corr = np.corrcoef(Y_test.flatten(), pred.flatten())[0, 1]

print(f"Best Model RMSE: {rmse}")
print(f"Best Model MAE: {mae}")
print(f"Best Model Correlation: {corr}")

metadata = {
    "variables": X_vars,
    "best_params": grid_search.best_params_,
    "search_space_size": len(hidden_layer_combinations) * len(param_grid['
                                          activation']) * len(param_grid['
                                          solver']),
    "rmse": rmse,
    "mae": mae,
    "corr": corr,
    "training_time": tf,
    "timestamp": time.time(),
}

# Save best model
model_name = f"best_mlp_{int(time.time())}"
save_model(best_model, scaler, metadata, model_name)
print(f"Saved best model as: {model_name}")

results_df = pd.DataFrame(grid_search.cv_results_)
results_df["MSE"] = -results_df["mean_test_score"]
results_df["RMSE"] = np.sqrt(results_df["MSE"])
results_df = results_df.sort_values(by="RMSE")

results_df.loc[results_df['rank_test_score'] == 1, "test_RMSE"] = rmse
results_df.loc[results_df['rank_test_score'] == 1, "test_MAE"] = mae
results_df.loc[results_df['rank_test_score'] == 1, "test_CORR"] = corr

results_df.to_csv(f"multi_metadata/model_rankings_{int(time.time())}.csv",
                                     index=False)
```

```
print("Saved rankings to model_rankings.csv")
print(results_df.head(5)[["params", "RMSE"]])
```

A.4 Empirical Model Scoring (compare_models.py)

```
import json, os

def calc_model_score(rmse, mae, corr):
    return (1 / rmse) + (1 / mae) + (corr * 10)

dir = "multi_metadata"

model_names = [v.split("_meta.json")[0] for v in os.listdir(dir)]
model_metadata = {}

for model in model_names:
    with open(f"{dir}/{model}_meta.json", "r") as f:
        model_metadata[model] = json.load(f)

for model in model_metadata:
    rmse = model_metadata[model]["rmse"]
    mae = model_metadata[model]["mae"]
    corr = model_metadata[model]["corr"]

    score = calc_model_score(rmse, mae, corr)
    model_metadata[model]["score"] = score

print(json.dumps(model_metadata, indent=4))
```

References

[1] Kennicutt, M., Chown, S., Cassano, J., et al. (2014). *Polar research: Six priorities for Antarctic science*. Nature, 512, 23-25. https://doi.org/10.1038/512023a

[2] Gardner, M. W., Dorling, S. R. (1998). *Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences*. Atmospheric Environment, 32(14-15), 2627-2636. https://doi.org/10.1016/S1352-2310(97)00447-0

[3] Liu, Z., Peng, C., Xiang, W., et al. (2010). *Application of artificial neural networks in global climate change and ecological research: An overview*. Chinese Science Bulletin, 55, 3853-3863. https://doi.org/10.1007/s11434-010-4183-3

[4] Hersbach, H., Bell, B., Berrisford, P., Biavati, G., Horányi, A., Muñoz Sabater, *et al.* (2023). *ERA5 hourly data on single levels from 1940 to present*. Copernicus Climate Change Service (C3S) Climate Data Store. https://doi.org/10.24381/cds.adbb2d47

[5] Parkinson, C. L., Cavalieri, D. J. (2012). *Antarctic sea ice variability and trends, 1979-2010*. The Cryosphere, 6, 871-880. https://doi.org/10.5194/tc-6-871-2012

[6] Wang, Z., Turner, J., Wu, Y., Liu, C. (2019). *Rapid Decline of Total Antarctic Sea Ice Extent during 2014-16 Controlled by Wind-Driven Sea Ice Drift*. Journal of Climate, 32(17), 5381-5395. https://doi.org/10.1175/JCLI-D-18-0635.1

[7] Schemm, S. (2018). *Regional trends in weather systems help explain Antarctic sea ice trends*. Geophysical Research Letters, 45(13), 7165-7175. https://doi.org/10.1029/2018GL079109

[8] Turner, J., Anderson, P., Lachlan-Cope, T. A., Colwell, S., Phillips, T., Kirchgaessner, A., *et al.* (2009). *Record low surface air temperature at Vostok station, Antarctica*. Journal of Geophysical Research: Atmospheres, 114(D24102). https://doi.org/10.1029/2009JD012104

[9] Turner, J., Lu, H., King, J., Marshall, G. J., Phillips, T., Bannister, D., Colwell, S. (2021). *Extreme Temperatures in the Antarctic*. Journal of Climate, 34(7), 2653-2668. https://doi.org/10.1175/JCLI-D-20-0538.1

[10] Yang, J., Xiao, C., Liu, J., Li, S., Qin, D. (2021). *Variability of Antarctic sea ice extent over the past 200 years*. Science Bulletin, 66(23), 2394-2404. https://doi.org/10.1016/j.scib.2021.07.028

[11] De Santis, A., Maier, E., Gomez, R., Gonzalez, I. (2017). *Antarctica, 1979-2016 sea ice extent: total versus regional trends, anomalies, and correlation with climatological variables*. International Journal of Remote Sensing, 38(24), 7566-7584. https://doi.org/10.1080/01431161.2017.1363440

[12] Jena, B., Kumar, A., Ravichandran, M., Kern, S. (2018). *Mechanism of sea-ice expansion in the Indian Ocean sector of Antarctica: Insights from satellite observation and model reanalysis*. PLOS ONE, 13(10), e0203222. https://doi.org/10.1371/journal.pone.0203222

[13] Python Software Foundation. (2025). *Python Language Reference, version 3.13.7.* http://www.python.org

[14] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., *et al.* (2011). *Scikit-Learn: Machine Learning in Python.* Journal of Machine Learning Research, 12, 2825-2830. http://jmlr.org/papers/v12/pedregosa11a.html

[15] Goswami, M. (2025). *megz15/mlp-sic-pred: MLP Neural Network for Antarctic Sea-Ice Concentration Prediction.* GitHub. https://github.com/megz15/mlp-sic-pred