Lab Objective: To explore the convolutional and long short-term neural networks from Week 8 lecture and apply them for forecast of number of days of ground frost and snow based on other weather variables.

The dataset for this lab is slightly different to one used before

This one is *curated_data_24month_2010-2022_nonans.csv*

The initial part of this notebook is very important as it is reading in the tabular data and reformatting it into 3D data

Dataset starts as monthly data per coord with a year and month column

The params are split into categories, i.e. groundfrost_x, where x = 24

There are 24 columns per category

The code then collects the column ranges for each category and stores in a variable as an array of numbers

A subset for each category is then created for each with the shape **(8502, 23)**

Note that the splitting process removed the month, year and corrdinates

There are 7 categories in total which are then stacked using **numpy.stack()**

The function by default gives the shape '(7, 8502, 23)' which is difficult to understand

But is transformed to give the shape '(8502, 23, 7)'

The tabular view of the data retains its original shape at the categories are stacked on the Z axis

Rows are the data instances given by coordinates (place) and a month/year

Columns timesteps (?) going back or forward 23 months (?)

Tensors are the categories of params

2 output columns are extracted from the originally shaped data as raw vectors/arrays:

- ground_frost_label = data[:, ground_frost_24_col]
- snow_label = data[:, snow_24_col]

Each element of the outputs represents the outcome give a single (23, 7) input

This is an instance with 23 monthly datapoints and 7 params

The notebook used from **sklearn.model_selection import train_test_split** to split the data into train and test
It actually split into training and remaining, then further split the remaining into test and validation with a 50/50 split

The method of creating the splitting creates an array of all the possible ids [1 2 3 … N]

Then creates sub arrays of randomised is. These arrays can then be used to split the dataset(s)

Data is then normalized using from **sklearn.preprocessing import MinMaxScaler** and 0 to 1 range

The data is reshaped and flattened in order to apply the scaling to a shape of (195546, 7)

Data is often shaped back after preprocessing

After the data is scaled is it split using the id arrays created earlier

The notebook then goes on to setup a CNN class using PyTorch

The hidden layers are a succession of Conv1d and MaxPool1d before a final Linear and ReLU

Training and evaluation is very similar to the epochs looping in MLP

Notebook then executes the training class and function which is again very similar to before

The notebook then spends some time working out the output sizes of the convolutions

Note that the dimensions are shrinking on the time component (23 down to 7)

The param dimensions depend on the number of hidden layer filters used in each layer. These can go up or down

Max pool also shrinks the time component but retains the param dimensions

Some time is spent calculating the number params required. This is probably a good thing to repeat in the assessment

Augmentation types to experience with:
- dropout along the time dimension
- dropout along the variable dimension
- random noise along the time dimension

To enable robustness to missing sensors and noisy data, dropout was applied to the variable dimension and random noise added to the data.

When the data is noise, performance is lower with augmentation but the robustness is better

To achieve feature dropout, the code reshapes the data and uses self.dropout(out) to remove features

torch.nn.Dropout(p=...)

Dropping out is setting values to 0

This is a form of feature-level augmentation/regularization

The network is then forced to learn to make predictions even when certain input features are entirely absent for a given sequence. This makes the model more robust and less reliant on any single feature, which reduces overfitting and improves generalization.

Next the notebook goes onto focus on LSTMs

Three questions:

- Create and train a three-layer LSTMNN model (see https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html for the documentation for the PyTorch LSTM layer)
- How many weights does the LSTMNN have?
- How does your model perform in comparison with the CNN in Section 4?

An LSTM (Long Short-Term Memory) is a type of Recurrent Neural Network (RNN) specifically designed to process sequential data. It has internal mechanisms (gates) that allow it to learn long-term dependencies in sequences, overcoming the vanishing/exploding gradient problems of simpler RNNs.

The code set up for LSTM is very short

The package is setup to sequentially parse data of a 3 nature with the rows being taken as instances

There is an inbuilt function to get the params of the model: total_params = sum(w.numel() for w in model_LSTMNN.parameters())

But you could still calculate manual based on knowledge of the structure of a LSTM layer.