Precipitation Nowcasting using Neural Networks

In this exercise, you are going to build a set of deep learning models on a real world task using PyTorch. PyTorch is an open source machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing, primarily developed by Facebook's AI Research lab (FAIR).

Setting up to use the gpu

Before we start, we need to change the environment of Colab to use GPU. Do so by:

Runtime -> Change runtime type -> Hardware accelerator -> GPU

Deep Neural Networks with PyTorch

To complete this exercise, you will need to build deep learning models for precipitation nowcasting. You will build a subset of the models shown below:

- Fully Connected (Feedforward) Neural Network
- A Feedforward network with dropout
- A convolution neural network model

and one more model of your choice to achieve the highest score possible.

We provide the code for data cleaning and some starter code for PyTorch in this notebook but feel free to modify those parts to suit your needs. Feel free to use additional libraries (e.g. scikit-learn) as long as you have a model for each type mentioned above.

This notebook assumes you have already installed PyTorch with python3 and had GPU enabled. If you run this exercise on Colab you are all set.

Precipitation Nowcasting

Precipitation nowcasting is the the task of predicting the amount of rainfall in a certain region given some kind of sensor data. The term nowcasting refers to tasks that try to predict the current or near future conditions (within 6 hours).

You will be given satellite images in 3 different bands covering a 5 by 5 region from different parts of Thailand. In other words, your input will be a 5x5x3 image. Your task is to predict the amount of rainfall in the center pixel. You will first do the prediction using just a simple fully-connected neural network that view each pixel as different input features.

Since the your input is basically an image, we will then view the input as an image and apply CNN to do the prediction. Finally, we can also add a time component since weather prediction can benefit greatly using previous time frames. Each data point actually contain 5 time steps, so each input data point has a size of 5x5x5x3 (time x height x width x channel), and the output data has a size of 5 (time). You will use this time information when you work with RNNs.

Finally, we would like to thank the Thai Meteorological Department for providing the data for this assignment.

In [ ]:
```
# !nvidia-smi
```

```
Thu Apr 10 03:37:34 2025
+-----------------------------------------------------------------------------
---------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version:
12.4     |
|-----------------------------------------+------------------------+-----------
---------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Unc
orr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Co
mpute M. |
|                                         |                        |
MIG M. |
|=========================================+========================+===========
=========|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |
0 |
| N/A   52C    P8             10W /   70W |       0MiB /  15360MiB |      0%
Default |
|                                         |                        |
N/A |
+-----------------------------------------+------------------------+-----------
---------+

+-----------------------------------------------------------------------------
---------+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name                              GP
U Memory |
|        ID   ID                                                               Us
age      |
|=============================================================================
=========|
|  No running processes found
|
+-----------------------------------------------------------------------------
---------+
```

In [ ]:
```
# For summarizing and visualizing models
# !pip install torchinfo
```

```
Collecting torchinfo
  Downloading torchinfo-1.8.0-py3-none-any.whl.metadata (21 kB)
Downloading torchinfo-1.8.0-py3-none-any.whl (23 kB)
Installing collected packages: torchinfo
Successfully installed torchinfo-1.8.0
```

Weights and Biases

Weights and Biases (wandb) is an experiment tracking tool for machine learning. It can log and visualize experiments in real time. It supports many popular ML frameworks, and obviously PyTorch is one of them. In this notebook you will learn how to log general metrics like losses, parameter distributions, and gradient distribution with wandb.

To install wandb, run the cell below

In [ ]:
```
# !pip install wandb
```

```
Requirement already satisfied: wandb in /usr/local/lib/python3.11/dist-packages
(0.19.9)
Requirement already satisfied: click!=8.0.0,>=7.1 in /usr/local/lib/python3.11/di
st-packages (from wandb) (8.1.8)
Requirement already satisfied: docker-pycreds>=0.4.0 in /usr/local/lib/python3.1
1/dist-packages (from wandb) (0.4.0)
Requirement already satisfied: gitpython!=3.1.29,>=1.0.0 in /usr/local/lib/python
3.11/dist-packages (from wandb) (3.1.44)
Requirement already satisfied: platformdirs in /usr/local/lib/python3.11/dist-pac
kages (from wandb) (4.3.7)
Requirement already satisfied: protobuf!=4.21.0,!=5.28.0,<6,>=3.19.0 in /usr/loca
l/lib/python3.11/dist-packages (from wandb) (5.29.4)
Requirement already satisfied: psutil>=5.0.0 in /usr/local/lib/python3.11/dist-pa
ckages (from wandb) (5.9.5)
Requirement already satisfied: pydantic<3 in /usr/local/lib/python3.11/dist-packa
ges (from wandb) (2.11.2)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.11/dist-packages
(from wandb) (6.0.2)
Requirement already satisfied: requests<3,>=2.0.0 in /usr/local/lib/python3.11/di
st-packages (from wandb) (2.32.3)
Requirement already satisfied: sentry-sdk>=2.0.0 in /usr/local/lib/python3.11/dis
t-packages (from wandb) (2.25.1)
Requirement already satisfied: setproctitle in /usr/local/lib/python3.11/dist-pac
kages (from wandb) (1.3.5)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packa
ges (from wandb) (75.2.0)
Requirement already satisfied: typing-extensions<5,>=4.4 in /usr/local/lib/python
3.11/dist-packages (from wandb) (4.13.1)
Requirement already satisfied: six>=1.4.0 in /usr/local/lib/python3.11/dist-packa
ges (from docker-pycreds>=0.4.0->wandb) (1.17.0)
Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/lib/python3.11/dist-
packages (from gitpython!=3.1.29,>=1.0.0->wandb) (4.0.12)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.1
1/dist-packages (from pydantic<3->wandb) (0.7.0)
Requirement already satisfied: pydantic-core==2.33.1 in /usr/local/lib/python3.1
1/dist-packages (from pydantic<3->wandb) (2.33.1)
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python
3.11/dist-packages (from pydantic<3->wandb) (0.4.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python
3.11/dist-packages (from requests<3,>=2.0.0->wandb) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-pac
kages (from requests<3,>=2.0.0->wandb) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/di
st-packages (from requests<3,>=2.0.0->wandb) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/di
st-packages (from requests<3,>=2.0.0->wandb) (2025.1.31)
Requirement already satisfied: smmap<6,>=3.0.1 in /usr/local/lib/python3.11/dist-
packages (from gitdb<5,>=4.0.1->gitpython!=3.1.29,>=1.0.0->wandb) (5.0.2)
```

Setup

1. Register Wandb account (and confirm your email)
2. `wandb login` and copy paste the API key when prompt

In [ ]:
```
# !wandb login
```

```
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://
wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit:
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: ittikorn (ittikorn-chulalongkorn-university) to ht
tps://api.wandb.ai. Use `wandb login --relogin` to force relogin
```

In [1]:
```
import os
import numpy as np
import pickle
import pandas as pd
import matplotlib.pyplot as plt
import urllib
import wandb
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision.transforms as transforms

from sklearn import preprocessing
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from tqdm.notebook import tqdm

import intel_npu_acceleration_library
from intel_npu_acceleration_library.compiler import CompilerConfig

torch.__version__ # 2.6.0+cu124
```

Out[1]:
```
'2.6.0+cpu'
```

Loading the data

You can load the dataset from huggingface hub by running the following.

In [ ]:
```
# !pip install huggingface_hub
```

```
Requirement already satisfied: huggingface_hub in /usr/local/lib/python3.11/dist-
packages (0.30.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-package
s (from huggingface_hub) (3.18.0)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist
-packages (from huggingface_hub) (2025.3.2)
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.11/dist-
packages (from huggingface_hub) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-pack
ages (from huggingface_hub) (6.0.2)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-package
s (from huggingface_hub) (2.32.3)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-pac
kages (from huggingface_hub) (4.67.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/pytho
n3.11/dist-packages (from huggingface_hub) (4.13.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python
3.11/dist-packages (from requests->huggingface_hub) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-pac
kages (from requests->huggingface_hub) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/di
st-packages (from requests->huggingface_hub) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/di
st-packages (from requests->huggingface_hub) (2025.1.31)
```

In [ ]:
```
# from huggingface_hub import hf_hub_download
# hf_hub_download(repo_id="ecitslos/HWnowcastingdata", filename="nowcastingHWdat
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWa
rning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (h
ttps://huggingface.co/settings/tokens), set it as secret in your Google Colab and
restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access publi
c models or datasets.
  warnings.warn(
nowcastingHWdataset.tar.gz:   0%|          | 0.00/272M [00:00<?, ?B/s]
```

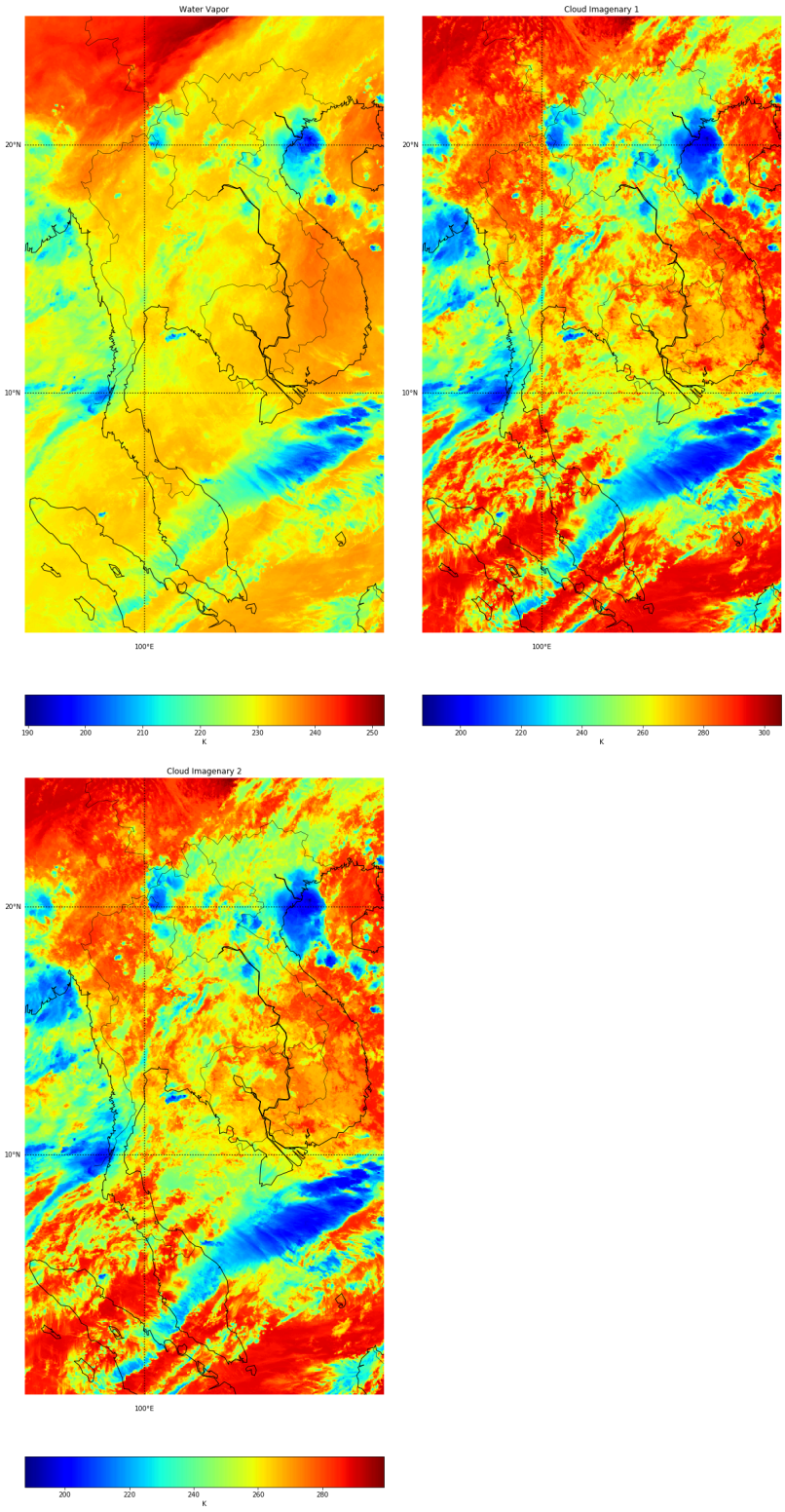
Out[ ]:
```
'nowcastingHWdataset.tar.gz'
```

In [ ]:
```
# !tar -xvf './nowcastingHWdataset.tar.gz'
```

```
dataset/features-m10.pk
dataset/features-m6.pk
dataset/features-m7.pk
dataset/features-m8.pk
dataset/features-m9.pk
dataset/labels-m10.pk
dataset/labels-m6.pk
dataset/labels-m7.pk
dataset/labels-m8.pk
dataset/labels-m9.pk
```

Data Explanation

The data is an hourly measurement of water vapor in the atmosphere, and two infrared measurements of cloud imagery on a latitude-longitude coordinate. Each measurement is illustrated below as an image. These three features are included as different channels in your input data.



We also provide the hourly precipitation (rainfall) records in the month of June, July, August, September, and October from weather stations spreaded around the country. A 5x5 grid around each weather station at a particular time will be paired with the precipitation recorded at the corresponding station as input and output data. Finally, five adjacent timesteps are stacked into one sequence.

The month of June-August are provided as training data, while the months of September and October are used as validation and test sets, respectively.

Reading data

In [2]:
```
def read_data(months, data_dir='dataset'):
    features = np.array([], dtype=np.float32).reshape(0,5,5,5,3)
    labels = np.array([], dtype=np.float32).reshape(0,5)
    for m in months:
        filename = 'features-m{}.pk'.format(m)
        with open(os.path.join(data_dir,filename), 'rb') as file:
            features_temp = pickle.load(file)
        features = np.concatenate((features, features_temp), axis=0)

        filename = 'labels-m{}.pk'.format(m)
        with open(os.path.join(data_dir,filename), 'rb') as file:
            labels_temp = pickle.load(file)
        labels = np.concatenate((labels, labels_temp), axis=0)

    return features, labels
```

In [3]:
```
# use data from month 6,7,8 as training set
x_train, y_train = read_data(months=[6,7,8])

# use data from month 9 as validation set
x_val, y_val = read_data(months=[9])

# use data from month 10 as test set
x_test, y_test = read_data(months=[10])

print('x_train shape:',x_train.shape)
print('y_train shape:', y_train.shape, '\n')
print('x_val shape:',x_val.shape)
print('y_val shape:', y_val.shape, '\n')
print('x_test shape:',x_test.shape)
print('y_test shape:', y_test.shape)
```

```
x_train shape: (229548, 5, 5, 5, 3)
y_train shape: (229548, 5)

x_val shape: (92839, 5, 5, 5, 3)
y_val shape: (92839, 5)

x_test shape: (111715, 5, 5, 5, 3)
y_test shape: (111715, 5)
```

features

- dim 0: number of entries
- dim 1: number of time-steps in ascending order
- dim 2,3: a 5x5 grid around rain-measued station
- dim 4: water vapor and two cloud imagenaries

labels

- dim 0: number of entries

- dim 1: number of precipitation for each time-step

Three-Layer Feedforward Neural Networks

In [4]:
```
# Dataset need to be reshaped to make it suitable for feedforword model
def preprocess_for_ff(x_train, y_train, x_val, y_val):
    x_train_ff = x_train.reshape((-1, 5*5*3))
    y_train_ff = y_train.reshape((-1, 1))
    x_val_ff = x_val.reshape((-1, 5*5*3))
    y_val_ff = y_val.reshape((-1, 1))
    x_test_ff = x_test.reshape((-1, 5*5*3))
    y_test_ff = y_test.reshape((-1, 1))

    return x_train_ff, y_train_ff, x_val_ff, y_val_ff, x_test_ff, y_test_ff

x_train_ff, y_train_ff, x_val_ff, y_val_ff, x_test_ff, y_test_ff = preprocess_fo
print(x_train_ff.shape, y_train_ff.shape)
print(x_val_ff.shape, y_val_ff.shape)
print(x_test_ff.shape, y_test_ff.shape)
```

```
(1147740, 75) (1147740, 1)
(464195, 75) (464195, 1)
(558575, 75) (558575, 1)
```

TODO#1

Explain each line of code in the function preprocess_for_ff()

Ans:

- Line 1: Reshape training data to 2D array each row as flattened 5x5x3 grid
- Line 2: Reshape training label to 2D array with 1 column
- Line 3: Reshape validation data to 2D array each row as flattened 5x5x3
- Line 4: Reshape validation label to 2D array with 1 column
- Line 5: Reshape test data to 2D array each row as flattened 5x5x3
- Line 6: Reshape test label to 2D array with 1 column
- Line 8: Return all values

Dataset

To prepare a DataLoader in order to feed data into the model, we need to create a `torch.utils.data.Dataset` object first. (Learn more about it here)

Dataset is a simple class that the DataLoader will get data from, most of its functionality comes from `__getitem__(self, index)` method, which will return a single data point (both input and label). In real world scenarios the method can do some other stuffs such as

1. Load images

If your input (x) are images. Oftentimes you won't be able to fit all the training images into your RAM. Thus, you should pass an array (or list) of image path into the dataloader, and the `__getitem__` will be the one who dynamically loads the actual image from the harddisk for you.

2. Data Normalization

Data normalization helps improve stability of training. Unnormalized data can cause gradients to explode. There are many variants of normalization, but in this notebook we will use either minmax or z-score (std) normalization. Read this (or google) if you wish to learn more about data normalization.

3. Data Augmentation

In computer vision, you might want to apply small changes to the images you use in training (adjust brightness, contrast, rotation) so that the model will generalize better on unseen data. There are two kinds of augmentation: static and dynamic. Static augmentation will augment images and save to disk as a new dataset. On the other hand, rather than applying the change initially and use the same change on each image every epoch, dynamic augmentation will augment each data differently for each epoch. Note that augmentation is usually done on the CPU and you might be bounded by the CPU instead. PyTorch has a dedicated documentation about data augmentation if you want to know more.

In [5]:
```
class RainfallDatasetFF(Dataset):
    def __init__(self, x, y, normalizer):
        self.x = x.astype(np.float32)
        self.y = y.astype(np.float32)
        self.normalizer = normalizer
        print(self.x.shape)
        print(self.y.shape)

    def __getitem__(self, index):
        x = self.x[index] # Retrieve data
        x = self.normalizer.transform(x.reshape(1, -1)) # Normalize
        y = self.y[index]
        return x, y

    def __len__(self):
        return self.x.shape[0]
```

In [6]:
```
def normalizer_std(X):
    scaler = preprocessing.StandardScaler().fit(X)
    return scaler

def normalizer_minmax(X):
    scaler = preprocessing.MinMaxScaler().fit(X)
    return scaler
```

In [7]:
```
normalizer = normalizer_std(x_train_ff) # We will normalize everything based on

train_dataset = RainfallDatasetFF(x_train_ff, y_train_ff, normalizer)
```

```
val_dataset = RainfallDatasetFF(x_val_ff, y_val_ff, normalizer)
test_dataset = RainfallDatasetFF(x_test_ff, y_test_ff, normalizer)
```

```
(1147740, 75)
(1147740, 1)
(464195, 75)
(464195, 1)
(558575, 75)
(558575, 1)
```

DataLoader

DataLoader feeds data from our dataset into the model. We can freely customize batch size, data shuffle for each data split, and much more with DataLoader class. If you're curious about what can you do with PyTorch's DataLoader, you can check this documentation

In [8]:
```
train_loader = DataLoader(train_dataset, batch_size=1024, shuffle=True, pin_memo
val_loader = DataLoader(val_dataset, batch_size=1024, shuffle=False, pin_memory=
test_loader = DataLoader(test_dataset, batch_size=1024, shuffle=False, pin_memor
```

Loss Function

PyTorch has many loss functions readily available for use. We can also write our own custom loss function as well. But for now, we will use PyTorch's built-in mean squared error loss

In [9]:
```
loss_fn = nn.MSELoss()
```

TODO#2

Why is the loss MSE?

Ans: This is a regression task to predict continuous value. MSE also penalize larger error more heavily to minimize significant of small differences between predicted and actual values.

Device

Unlike Tensorflow/Keras, PyTorch allows user to freely put any Tensor or objects (loss functions, models, optimizers, etc.) in CPU or GPU. By default, all objects created will be in CPU. In order to use GPU we will have to supply `device = torch.device("cuda")` into the objects to move it to GPU. You will usually see the syntax like `object.to(device)` for moving CPU object to GPU, or `o = Object(..., device=device)` to create the object in the GPU.

In [18]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cpu
```

Model

Below, the code for creating a 3-layers fully connected neural network in PyTorch is provided. Run the code and make sure you understand what you are doing. Then, report the results.

In [13]:
```
class FeedForwardNN(nn.Module):
    def __init__(self, hidden_size=200):
        super(FeedForwardNN, self).__init__()
        self.ff1 = nn.Linear(75, hidden_size)
        self.ff2 = nn.Linear(hidden_size, hidden_size)
        self.ff3 = nn.Linear(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, 1)

    def forward(self, x):
        hd1 = F.relu(self.ff1(x))
        hd2 = F.relu(self.ff2(hd1))
        y = F.relu(self.ff3(hd2))
        y = self.out(y)
        return y.reshape(-1, 1)
```

TODO#3

What is the activation function in the final dense layer? and why? Do you think there is a better activation function for the final layer?

Ans: Linear, the network is expected to predict values over a continuous range. It is ideal.

In [58]:
```
# Hyperparameters and other configs
config = {
    'architecture': 'feedforward',
    'lr': 0.01,
    'hidden_size': 200,
    'scheduler_factor': 0.2,
    'scheduler_patience': 2,
    'scheduler_min_lr': 1e-4,
    'epochs': 10
}

# Model
model_ff = FeedForwardNN(hidden_size=config['hidden_size'])
model_ff = model_ff.to(device)
optimizer = torch.optim.Adam(model_ff.parameters(), lr=config['lr'])
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
    optimizer,
    'min',
    factor=config['scheduler_factor'],
    patience=config['scheduler_patience'],
    min_lr=config['scheduler_min_lr']
)
```

In [15]:
```
from torchinfo import summary
summary(model_ff, input_size=(1024, 75))
```

Out[15]:
```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
FeedForwardNN                            [1024, 1]                 --
├─Linear: 1-1                            [1024, 200]               15,200
├─Linear: 1-2                            [1024, 200]               40,200
├─Linear: 1-3                            [1024, 200]               40,200
├─Linear: 1-4                            [1024, 1]                 201
==========================================================================================
Total params: 95,801
Trainable params: 95,801
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 98.10
==========================================================================================
Input size (MB): 0.31
Forward/backward pass size (MB): 4.92
Params size (MB): 0.38
Estimated Total Size (MB): 5.61
==========================================================================================
```

TODO#4

Explain why the first linear layer has number of parameters = 15200

Ans:

- Input size - 5x5x3 = 75
- Hidden layer size = 200
- Weights = 75x200 = 15,000
- Bias = 200
- Total params = 15,000 + 200 = 15,200

Training

In [59]:
```
torch.set_num_threads(os.cpu_count())
print(os.cpu_count())
```

```
22
```

In [ ]:
```
# compiler_conf = CompilerConfig(dtype=torch.float32, training=True)
# model_ff = intel_npu_acceleration_library.compile(model_ff, compiler_conf)
```

In [60]:
```
train_losses = []
val_losses = []
learning_rates = []

# Start wandb run
wandb.init(
    project='precipitation-nowcasting-2025',
    config=config,
```

```
)

# Log parameters and gradients
wandb.watch(model_ff, log='all')

for epoch in range(config['epochs']):  # loop over the dataset multiple times

    # Training
    train_loss = []
    current_lr = optimizer.param_groups[0]['lr']
    learning_rates.append(current_lr)

    # Flag model as training. Some layers behave differently in training and
    # inference modes, such as dropout, BN, etc.
    model_ff.train()

    print(f"Training epoch {epoch+1}...")
    print(f"Current LR: {current_lr}")

    for i, (inputs, y_true) in enumerate(tqdm(train_loader)):
        # Transfer data from cpu to gpu
        inputs = inputs.to(device)
        y_true = y_true.to(device)

        # Reset the gradient
        optimizer.zero_grad()

        # Predict
        y_pred = model_ff(inputs)

        # Calculate loss
        loss = loss_fn(y_pred, y_true)

        # Compute gradient
        loss.backward()

        # Update parameters
        optimizer.step()

        # Log stuff
        train_loss.append(loss)

    avg_train_loss = torch.stack(train_loss).mean().item()
    train_losses.append(avg_train_loss)

    print(f"Epoch {epoch+1} train loss: {avg_train_loss:.4f}")

    # Validation
    model_ff.eval()
    with torch.no_grad(): # No gradient is required during validation
        print(f"Validating epoch {epoch+1}")
        val_loss = []
        for i, (inputs, y_true) in enumerate(tqdm(val_loader)):
            # Transfer data from cpu to gpu
            inputs = inputs.to(device)
            y_true = y_true.to(device)

            # Predict
            y_pred = model_ff(inputs)
```

```
            # Calculate loss
            loss = loss_fn(y_pred, y_true)

            # Log stuff
            val_loss.append(loss)

        avg_val_loss = torch.stack(val_loss).mean().item()
        val_losses.append(avg_val_loss)
        print(f"Epoch {epoch+1} val loss: {avg_val_loss:.4f}")

        # LR adjustment with scheduler
        scheduler.step(avg_val_loss)

        # Save checkpoint if val_loss is the best we got
        best_val_loss = np.inf if epoch == 0 else min(val_losses[:-1])
        if avg_val_loss < best_val_loss:
            # Save whatever you want
            state = {
                'epoch': epoch,
                'model': model_ff.state_dict(),
                'optimizer': optimizer.state_dict(),
                'scheduler': scheduler.state_dict(),
                'train_loss': avg_train_loss,
                'val_loss': avg_val_loss,
                'best_val_loss': best_val_loss,
            }

            print(f"Saving new best model..")
            torch.save(state, 'model_ff.pth.tar')

    wandb.log({
        'train_loss': avg_train_loss,
        'val_loss': avg_val_loss,
        'lr': current_lr,
    })

wandb.finish()
print('Finished Training')
```

```
Tracking run with wandb version 0.19.9
Run data is saved locally in
c:\Users\user\Desktop\CEDT\Code\AIML\NeuralNetwork1\wandb\run-
20250415_185451-pxc9sd09
Syncing run glamorous-shape-50 to Weights & Biases (docs)
View project at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-
nowcasting-2025
View run at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-
2025/runs/pxc9sd09
Training epoch 1...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 1 train loss: 1.9262
Validating epoch 1
  0%|          | 0/454 [00:00<?, ?it/s]
```

```
Epoch 1 val loss: 1.6667
Saving new best model..
Training epoch 2...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 2 train loss: 1.9221
Validating epoch 2
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 2 val loss: 1.6611
Saving new best model..
Training epoch 3...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 3 train loss: 1.9230
Validating epoch 3
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 3 val loss: 1.6582
Saving new best model..
Training epoch 4...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 4 train loss: 1.9234
Validating epoch 4
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 4 val loss: 1.6606
Training epoch 5...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 5 train loss: 1.9228
Validating epoch 5
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 5 val loss: 1.6618
Training epoch 6...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 6 train loss: 1.9236
Validating epoch 6
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 6 val loss: 1.6611
Training epoch 7...
Current LR: 0.002
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 7 train loss: 1.9235
Validating epoch 7
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 7 val loss: 1.6614
Training epoch 8...
Current LR: 0.002
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 8 train loss: 1.9234
Validating epoch 8
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 8 val loss: 1.6607
Training epoch 9...
Current LR: 0.002
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 9 train loss: 1.9234
Validating epoch 9
  0%|          | 0/454 [00:00<?, ?it/s]
```

```
Epoch 9 val loss: 1.6620
Training epoch 10...
Current LR: 0.0004
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 10 train loss: 1.9241
Validating epoch 10
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 10 val loss: 1.6612
```

Run history:

Run summary:

lr	0.0004
train_loss	1.92406
val_loss	1.66125

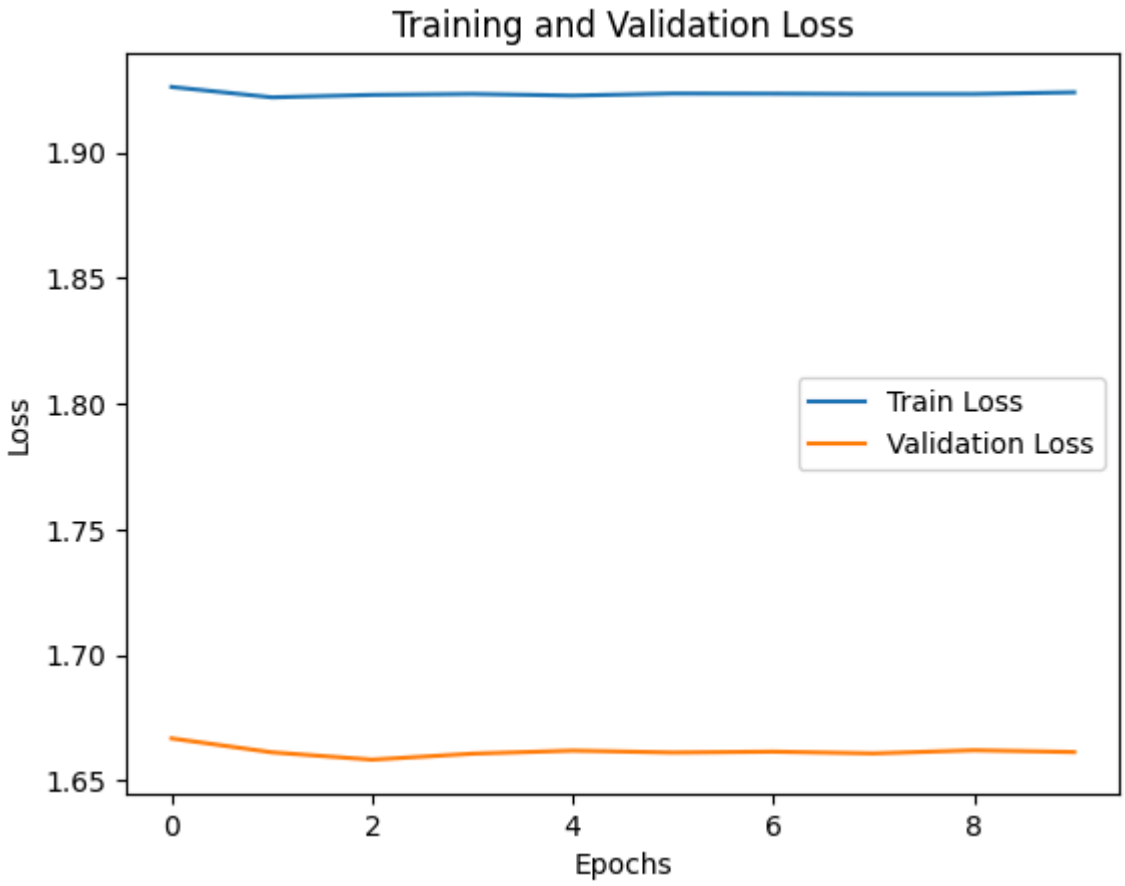
View run **glamorous-shape-50** at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/pxc9sd09
View project at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025
Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)
Find logs at: `.\wandb\run-20250415_185451-pxc9sd09\logs`

```
Finished Training
```

TODO#5

Plot loss and val_loss as a function of epochs.

In [61]:
```
plt.plot(train_losses, label='Train Loss')
plt.plot(val_losses, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```



TODO#6

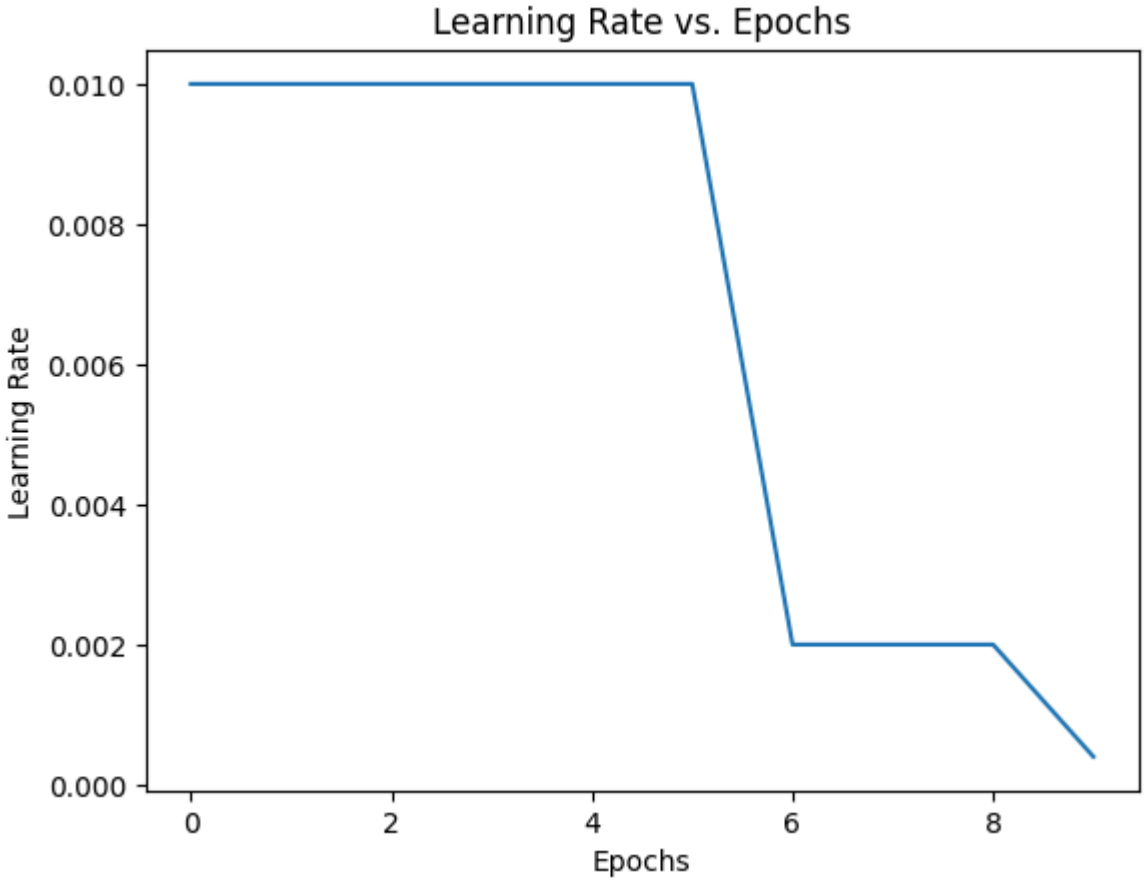
When does the model start to overfit?

Ans: Epoch 1

TODO#7

Plot the learning rate as a function of the epochs.

In [62]:
```
plt.plot(learning_rates)
plt.xlabel('Epochs')
plt.ylabel('Learning Rate')
plt.title('Learning Rate vs. Epochs')
plt.show()
```



TODO#8

What makes the learning rate change? (hint: try to understand the scheduler ReduceLROnPlateau)

Ans: Reduce learning rate when the metric stopped improving

Load Model

Use the code snippet below to load the model you just trained

In [22]:
```
checkpoint = torch.load('model_ff.pth.tar')
loaded_model = FeedForwardNN(hidden_size=config['hidden_size']) # Create model o
loaded_model.load_state_dict(checkpoint['model']) # Load weights
print(f"Loaded epoch {checkpoint['epoch']} model")
```

```
Loaded epoch 0 model
```

A more complex scheduling

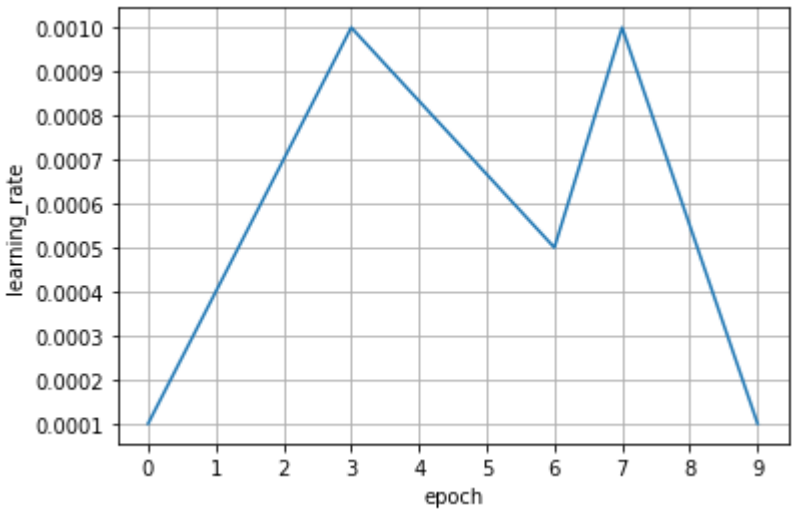
The scheduler can be very complicated and you can write your own heuristic for it.

TODO#9

Implement a custom learning rate scheduler that behaves like the following graph.

You might want to learn how to use PyTorch's built-in learning rate schedulers in order to build your own.

Learning rate should be function of epoch.



In [63]:
```python
# Implement scheduler here
class MyScheduler():
    def __init__(self, optimizer: torch.optim.Optimizer):
        self.optimizer = optimizer
        self.step(0)
        pass

    def step(self, epoch):
        # Changes the learning rate here
        if 0 <= epoch < 3:
          lr = 0.0001 + (0.001 - 0.0001) * (epoch / 3)
        elif 3 <= epoch < 6:
          lr = 0.001 - (0.001 - 0.0005) * ((epoch - 3) / 3)
        elif 6 <= epoch < 8:
          lr = 0.0005 + (0.001 - 0.0005) * ((epoch - 6) / 2)
        elif 8 <= epoch <= 9:
          lr = 0.001 - (0.001 - 0.0001) * (epoch - 8)
        else:
          lr = 0.0001  # fallback

        for param_group in self.optimizer.param_groups:
          param_group['lr'] = lr
        pass
```

In [64]:
```python
# Now train with your scheduler
# my_scheduler = MyScheduler(...)

model_custom = FeedForwardNN(hidden_size=config['hidden_size'])
model_custom = model_custom.to(device)
optimizer = torch.optim.Adam(model_custom.parameters(), lr=config['lr'])


my_scheduler = MyScheduler(optimizer=optimizer)

train_losses = []
val_losses = []
learning_rates = []
```

```
# Start wandb run
wandb.init(
    project='precipitation-nowcasting-2025',
    config=config,
)

# Log parameters and gradients
wandb.watch(model_custom, log='all')

for epoch in range(config['epochs']):  # Loop over the dataset multiple times

    # Training
    train_loss = []
    current_lr = optimizer.param_groups[0]['lr']
    learning_rates.append(current_lr)

    # Flag model as training. Some layers behave differently in training and
    # inference modes, such as dropout, BN, etc.
    model_custom.train()

    print(f"Training epoch {epoch+1}...")
    print(f"Current LR: {current_lr}")

    for i, (inputs, y_true) in enumerate(tqdm(train_loader)):
        # Transfer data from cpu to gpu
        inputs = inputs.to(device)
        y_true = y_true.to(device)

        # Reset the gradient
        optimizer.zero_grad()

        # Predict
        y_pred = model_custom(inputs)

        # Calculate loss
        loss = loss_fn(y_pred, y_true)

        # Compute gradient
        loss.backward()

        # Update parameters
        optimizer.step()

        # Log stuff
        train_loss.append(loss)

    avg_train_loss = torch.stack(train_loss).mean().item()
    train_losses.append(avg_train_loss)

    print(f"Epoch {epoch+1} train loss: {avg_train_loss:.4f}")

    # Validation
    model_custom.eval()
    with torch.no_grad(): # No gradient is required during validation
        print(f"Validating epoch {epoch+1}")
        val_loss = []
        for i, (inputs, y_true) in enumerate(tqdm(val_loader)):
            # Transfer data from cpu to gpu
            inputs = inputs.to(device)
```

```
            y_true = y_true.to(device)

            # Predict
            y_pred = model_custom(inputs)

            # Calculate loss
            loss = loss_fn(y_pred, y_true)

            # Log stuff
            val_loss.append(loss)

        avg_val_loss = torch.stack(val_loss).mean().item()
        val_losses.append(avg_val_loss)
        print(f"Epoch {epoch+1} val loss: {avg_val_loss:.4f}")

        # LR adjustment with scheduler
        my_scheduler.step(epoch+1)

        # Save checkpoint if val_loss is the best we got
        best_val_loss = np.inf if epoch == 0 else min(val_losses[:-1])
        if avg_val_loss < best_val_loss:
            # Save whatever you want
            state = {
                'epoch': epoch,
                'model': model_custom.state_dict(),
                'optimizer': optimizer.state_dict(),
                'train_loss': avg_train_loss,
                'val_loss': avg_val_loss,
                'best_val_loss': best_val_loss,
            }

            print(f"Saving new best model..")
            torch.save(state, 'model_custom.pth.tar')

    wandb.log({
        'train_loss': avg_train_loss,
        'val_loss': avg_val_loss,
        'lr': current_lr,
    })

wandb.finish()
print('Finished Training')
```

Tracking run with wandb version 0.19.9

Run data is saved locally in

`c:\Users\user\Desktop\CEDT\Code\AIML\NeuralNetwork1\wandb\run-20250415_193216-1m0e8eam`

Syncing run **treasured-lake-51** to Weights & Biases (docs)

View project at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

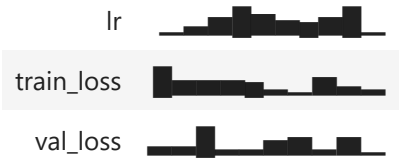
View run at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/1m0e8eam

```
Training epoch 1...
Current LR: 0.0001
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 1 train loss: 1.9191
Validating epoch 1
```

```
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 1 val loss: 1.6564
Saving new best model..
Training epoch 2...
Current LR: 0.00039999999999999996
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 2 train loss: 1.9186
Validating epoch 2
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 2 val loss: 1.6565
Training epoch 3...
Current LR: 0.0007
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 3 train loss: 1.9186
Validating epoch 3
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 3 val loss: 1.6574
Training epoch 4...
Current LR: 0.001
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 4 train loss: 1.9186
Validating epoch 4
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 4 val loss: 1.6563
Saving new best model..
Training epoch 5...
Current LR: 0.0008833333333333334
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 5 train loss: 1.9185
Validating epoch 5
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 5 val loss: 1.6562
Saving new best model..
Training epoch 6...
Current LR: 0.0006666666666666668
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 6 train loss: 1.9182
Validating epoch 6
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 6 val loss: 1.6568
Training epoch 7...
Current LR: 0.0005
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 7 train loss: 1.9180
Validating epoch 7
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 7 val loss: 1.6570
Training epoch 8...
Current LR: 0.00075
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 8 train loss: 1.9187
Validating epoch 8
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 8 val loss: 1.6561
Saving new best model..
Training epoch 9...
Current LR: 0.001
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 9 train loss: 1.9184
Validating epoch 9
```

```
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 9 val loss: 1.6570
Training epoch 10...
Current LR: 0.00010000000000000005
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 10 train loss: 1.9182
Validating epoch 10
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 10 val loss: 1.6560
Saving new best model..
```

Run history:



Run summary:

lr	0.0001
train_loss	1.91815
val_loss	1.65599

View run **treasured-lake-51** at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/1m0e8eam
View project at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025
Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)
Find logs at: `.\wandb\run-20250415_193216-1m0e8eam\logs`

```
Finished Training
```

[Optional] Wandb

You should now have a project in wandb with the name `precipitation-nowcasting`, which you should see the latest run you just finished inside the project. If you look into the run, you should be able to see plots of learning rate, train loss, val loss in the `Charts` section. Below it should be `Gradients` and `Parameters` section.

Wandb Observation

Optional TODO#1

Write your own interpretation of the logs from this example. A simple sentence or two for each section is sufficient.

Your answer:

Evaluation

In [65]:
```
##############################################################################
# TODO#10:                                                                   #
# Write a function to evaluate your model. Your function must predicts       #
# using the input model and return mean square error of the model.           #
#                                                                            #
# Hint: Read how to use PyTorch's MSE Loss                                   #
# https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html            #
##############################################################################
#                             WRITE YOUR CODE BELOW                          #
##############################################################################
def evaluate(data_loader, model):
    """
    Evaluate model on validation data given by data_loader
    """
    # compiler_conf = CompilerConfig(dtype=torch.float32)
    # model = intel_npu_acceleration_library.compile(model, compiler_conf)

    # write code here
    # Start wandb run
    wandb.init(
      project='precipitation-nowcasting-2025',
      config=config,
    )

    # # Log parameters and gradients
    wandb.watch(model, log='all')
    model.eval()  # Set the model to evaluation mode
    mse_loss = nn.MSELoss()  # Define the MSE loss function
    total_loss = 0.0
    total_samples = 0

    with torch.no_grad():  # Disable gradient computation for evaluation
        for inputs, targets in data_loader:
            inputs, targets = inputs.to(device), targets.to(device)  # Move data
            predictions = model(inputs)  # Get model predictions
            loss = mse_loss(predictions, targets)  # Compute MSE loss
            total_loss += loss.item() * inputs.size(0)  # Accumulate loss
            total_samples += inputs.size(0)  # Accumulate sample count

    mse = total_loss / total_samples  # Compute mean squared error
    wandb.finish()
    return mse
```

In [66]:
```
# We will use majority rule as a baseline.
def majority_baseline(label_set):
    unique, counts = np.unique(label_set, return_counts=True)
    majority = unique[np.argmax(counts)]
    baseline = 0
    label_set = label_set.reshape(-1,1)
    for r in label_set:
        baseline += (majority - r) ** 2 / len(label_set)
    return baseline
```

In [67]:
```
print('baseline')
print('train', majority_baseline(y_train))
print('validate', majority_baseline(y_val))
```

```
baseline
train [1.94397725]
validate [1.6746546]
```

In [68]:
```
print('FF-model')
print('train', evaluate(train_loader, model_ff))
print('validate', evaluate(val_loader, model_ff))
```

```
FF-model
```

Tracking run with wandb version 0.19.9

Run data is saved locally in

`c:\Users\user\Desktop\CEDT\Code\AIML\NeuralNetwork1\wandb\run-20250415_195928-gicl47ps`

Syncing run **ethereal-sponge-52** to Weights & Biases (docs)

View project at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

View run at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/gicl47ps

View run **ethereal-sponge-52** at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/gicl47ps

View project at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)

Find logs at: `.\wandb\run-20250415_195928-gicl47ps\logs`

```
train 1.9234000869832373
```

Tracking run with wandb version 0.19.9

Run data is saved locally in

`c:\Users\user\Desktop\CEDT\Code\AIML\NeuralNetwork1\wandb\run-20250415_200106-tr04wsj6`

Syncing run **major-darkness-53** to Weights & Biases (docs)

View project at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

View run at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/tr04wsj6

View run **major-darkness-53** at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/tr04wsj6
View project at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025
Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)
Find logs at: `.\wandb\run-20250415_200106-tr04wsj6\logs`

```
validate 1.6637271392069553
```

Dropout

You might notice that the 3-layered feedforward does not use dropout at all. Now, try adding dropout (dropout rate of 20%) to the model, run, and report the result again.

To access PyTorch's dropout, use `nn.Dropout`. Read more about PyTorch's built-in Dropout layer here

In [69]:
```
################################################################################
# TODO#11:                                                                     #
# Write a feedforward model with dropout                                       #
################################################################################
#                              WRITE YOUR CODE BELOW                           #
################################################################################
class FeedForwardNN(nn.Module):
    def __init__(self, hidden_size=200):
        super(FeedForwardNN, self).__init__()
        self.ff1 = nn.Linear(75, hidden_size)
        self.dropout1 = nn.Dropout(0.2) # Add dropout layer
        self.ff2 = nn.Linear(hidden_size, hidden_size)
        self.dropout2 = nn.Dropout(0.2) # Add dropout layer
        self.ff3 = nn.Linear(hidden_size, hidden_size)
        self.dropout3 = nn.Dropout(0.2) # Add dropout layer
        self.out = nn.Linear(hidden_size, 1)

    def forward(self, x):
        hd1 = F.relu(self.ff1(x))
        hd1 = self.dropout1(hd1) # Apply dropout
        hd2 = F.relu(self.ff2(hd1))
        hd2 = self.dropout2(hd2) # Apply dropout
        y = F.relu(self.ff3(hd2))
        y = self.dropout3(y) # Apply dropout
        y = self.out(y)
        return y.reshape(-1, 1)

  # Model
model_dropout = FeedForwardNN(hidden_size=config['hidden_size'])
model_dropout = model_dropout.to(device)
optimizer = torch.optim.Adam(model_dropout.parameters(), lr=config['lr'])
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
    optimizer,
    'min',
    factor=config['scheduler_factor'],
    patience=config['scheduler_patience'],
    min_lr=config['scheduler_min_lr']
)
```

In [70]:
```
################################################################################
# TODO#12:                                                                     #
# Complete the code to train your dropout model                                #
################################################################################
print('start training ff dropout')
################################################################################
#                              WRITE YOUR CODE BELOW                           #
################################################################################

train_losses = []
val_losses = []
learning_rates = []

# Start wandb run
wandb.init(
    project='precipitation-nowcasting-2025',
    config=config,
)

# Log parameters and gradients
wandb.watch(model_dropout, log='all')

for epoch in range(config['epochs']):  # loop over the dataset multiple times

    # Training
    train_loss = []
    current_lr = optimizer.param_groups[0]['lr']
    learning_rates.append(current_lr)

    # Flag model as training. Some layers behave differently in training and
    # inference modes, such as dropout, BN, etc.
    model_dropout.train()

    print(f"Training epoch {epoch+1}...")
    print(f"Current LR: {current_lr}")

    for i, (inputs, y_true) in enumerate(tqdm(train_loader)):
        # Transfer data from cpu to gpu
        inputs = inputs.to(device)
        y_true = y_true.to(device)

        # Reset the gradient
        optimizer.zero_grad()

        # Predict
        y_pred = model_dropout(inputs)

        # Calculate loss
        loss = loss_fn(y_pred, y_true)

        # Compute gradient
        loss.backward()

        # Update parameters
        optimizer.step()

        # Log stuff
        train_loss.append(loss)
```

```
    avg_train_loss = torch.stack(train_loss).mean().item()
    train_losses.append(avg_train_loss)

    print(f"Epoch {epoch+1} train loss: {avg_train_loss:.4f}")

    # Validation
    model_dropout.eval()
    with torch.no_grad(): # No gradient is required during validation
        print(f"Validating epoch {epoch+1}")
        val_loss = []
        for i, (inputs, y_true) in enumerate(tqdm(val_loader)):
            # Transfer data from cpu to gpu
            inputs = inputs.to(device)
            y_true = y_true.to(device)

            # Predict
            y_pred = model_dropout(inputs)

            # Calculate loss
            loss = loss_fn(y_pred, y_true)

            # Log stuff
            val_loss.append(loss)

        avg_val_loss = torch.stack(val_loss).mean().item()
        val_losses.append(avg_val_loss)
        print(f"Epoch {epoch+1} val loss: {avg_val_loss:.4f}")

        # LR adjustment with scheduler
        scheduler.step(avg_val_loss)

        # Save checkpoint if val_loss is the best we got
        best_val_loss = np.inf if epoch == 0 else min(val_losses[:-1])
        if avg_val_loss < best_val_loss:
            # Save whatever you want
            state = {
                'epoch': epoch,
                'model': model_dropout.state_dict(),
                'optimizer': optimizer.state_dict(),
                'scheduler': scheduler.state_dict(),
                'train_loss': avg_train_loss,
                'val_loss': avg_val_loss,
                'best_val_loss': best_val_loss,
            }

            print(f"Saving new best model..")
            torch.save(state, 'model_dropout.pth.tar')

    wandb.log({
        'train_loss': avg_train_loss,
        'val_loss': avg_val_loss,
        'lr': current_lr,
    })

wandb.finish()
print('Finished Training')
```

```
start training ff dropout
```

Tracking run with wandb version 0.19.9

Run data is saved locally in `c:\Users\user\Desktop\CEDT\Code\AIML\NeuralNetwork1\wandb\run-20250415_200147-aawlgjkj`

Syncing run **fallen-glade-54** to Weights & Biases (docs)

View project at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

View run at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/aawlgjkj

```
Training epoch 1...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 1 train loss: 1.9230
Validating epoch 1
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 1 val loss: 1.6587
Saving new best model..
Training epoch 2...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 2 train loss: 1.9217
Validating epoch 2
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 2 val loss: 1.6596
Training epoch 3...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 3 train loss: 1.9221
Validating epoch 3
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 3 val loss: 1.6612
Training epoch 4...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 4 train loss: 1.9234
Validating epoch 4
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 4 val loss: 1.6609
Training epoch 5...
Current LR: 0.002
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 5 train loss: 1.9244
Validating epoch 5
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 5 val loss: 1.6606
Training epoch 6...
Current LR: 0.002
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 6 train loss: 1.9232
Validating epoch 6
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 6 val loss: 1.6594
Training epoch 7...
Current LR: 0.002
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 7 train loss: 1.9223
Validating epoch 7
  0%|          | 0/454 [00:00<?, ?it/s]
```

```
Epoch 7 val loss: 1.6603
Training epoch 8...
Current LR: 0.0004
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 8 train loss: 1.9226
Validating epoch 8
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 8 val loss: 1.6599
Training epoch 9...
Current LR: 0.0004
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 9 train loss: 1.9223
Validating epoch 9
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 9 val loss: 1.6594
Training epoch 10...
Current LR: 0.0004
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 10 train loss: 1.9221
Validating epoch 10
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 10 val loss: 1.6594
```

Run history:

Run summary:

lr	0.0004
train_loss	1.92205
val_loss	1.65942

View run **fallen-glade-54** at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/aawlgjkj

View project at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)

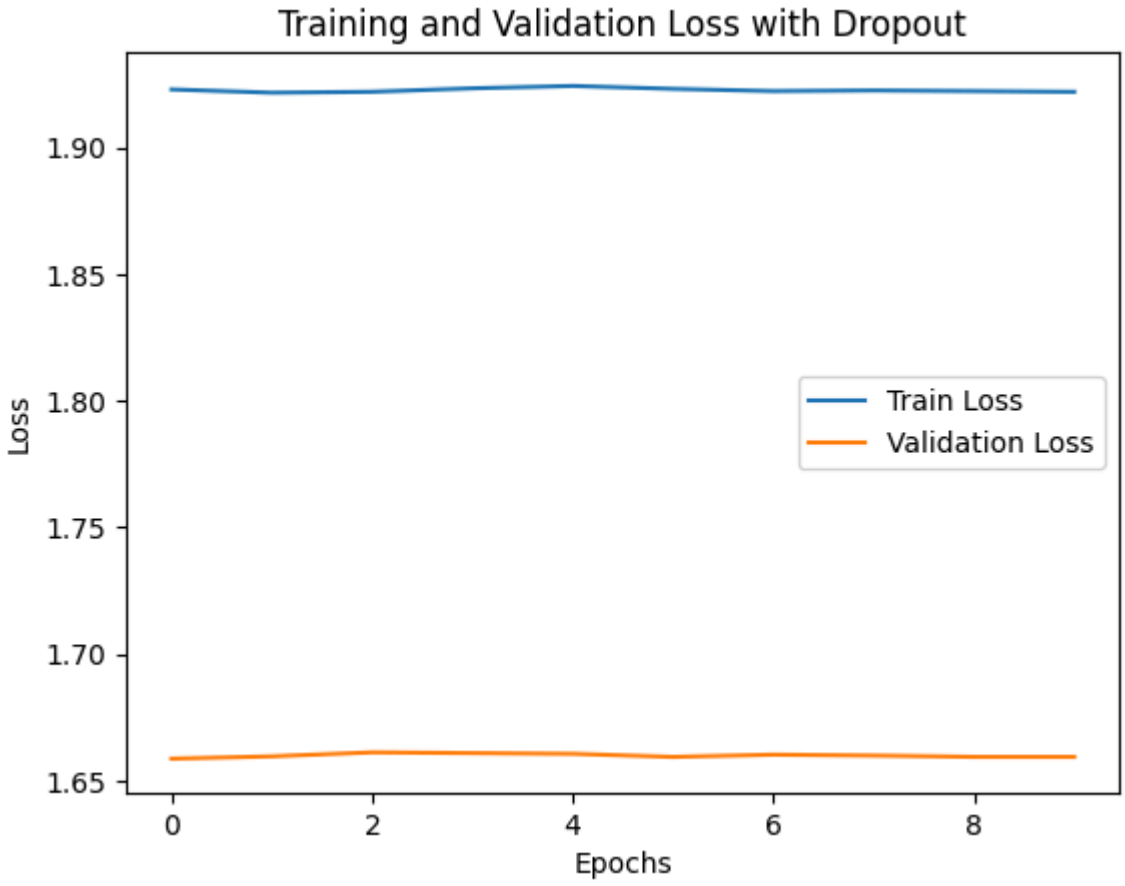
Find logs at: `.\wandb\run-20250415_200147-aawlgjkj\logs`

```
Finished Training
```

TODO#13

Plot the losses and MSE of the training and validation as before. Evaluate the dropout model's performance

In [71]:
```
# Plot here
plt.plot(train_losses, label='Train Loss')
plt.plot(val_losses, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss with Dropout')
plt.legend()
plt.show()
```



In [72]:
```
# Evaluate
print('train', evaluate(train_loader, model_dropout))
print('validate', evaluate(val_loader, model_dropout))
```

Tracking run with wandb version 0.19.9
Run data is saved locally in
`c:\Users\user\Desktop\CEDT\Code\AIML\NeuralNetwork1\wandb\run-20250415_203142-mr4wkvr1`
Syncing run **fiery-galaxy-55** to Weights & Biases (docs)
View project at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025
View run at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/mr4wkvr1

View run **fiery-galaxy-55** at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/mr4wkvr1

View project at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)

Find logs at: `.\wandb\run-20250415_203142-mr4wkvr1\logs`

```
train 1.921962004007472
```

Tracking run with wandb version 0.19.9

Run data is saved locally in `c:\Users\user\Desktop\CEDT\Code\AIML\NeuralNetwork1\wandb\run-20250415_203339-cox6dc4z`

Syncing run **spring-dragon-56** to Weights & Biases (docs)

View project at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

View run at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/cox6dc4z

View run **spring-dragon-56** at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/cox6dc4z

View project at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)

Find logs at: `.\wandb\run-20250415_203339-cox6dc4z\logs`

```
validate 1.661889920036683
```

Convolution Neural Networks

Now let's try to incorporate the grid sturcture to your model. Instead of passing in vectors, we are going to pass in the 5x5 grid into the model (5lat x 5long x 3channel). You are going to implement you own 2d-convolution neural networks with the following structure.

```
==========================================================================
Layer (type:depth-idx)                   Output Shape
Param #
==========================================================================
Conv2DNN                                 --
--
├─Conv2d: 1-1                            [1024, 200, 3, 3]
5,600
├─Linear: 1-2                            [1024, 200]
360,200
├─Linear: 1-3                            [1024, 200]
40,200
├─Linear: 1-4                            [1024, 1]
201
==========================================================================
Total params: 406,201
Trainable params: 406,201
Non-trainable params: 0
```

These parameters are simple guidelines to save your time.
You can play with them in the final section which you can choose any normalization methods, activation function, as well as any hyperparameter the way you want.

Hint: You should read PyTorch documentation to see the list of available layers and options you can use.

In [45]:
```
################################################################################
# TODO#15:                                                                     #
# Write a PyTorch convolutional neural network model.                          #
# You might want to use the layer torch.flatten somewhere                      #
################################################################################
#                             WRITE YOUR CODE BELOW                            #
################################################################################

class Conv2DNN(nn.Module):
    def __init__(self):
        super(Conv2DNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=200, kernel_size=3)
        self.fc1 = nn.Linear(200 * 3 * 3, 200)
        self.fc2 = nn.Linear(200, 200)
        self.out = nn.Linear(200, 1)

    def forward(self, x):
        x = x.view(-1, 3, 5, 5)  # Reshape input to match the expected dimension
        x = F.relu(self.conv1(x))  # Apply convolution and ReLU activation
        x = torch.flatten(x, start_dim=1)  # Flatten the output
        x = F.relu(self.fc1(x))  # Fully connected layer with ReLU
        x = F.relu(self.fc2(x))  # Fully connected layer with ReLU
        x = self.out(x)  # Output layer
        return x.reshape(-1, 1)

model_conv = Conv2DNN()
model_conv = model_conv.to(device)
optimizer = torch.optim.Adam(model_conv.parameters(), lr=config['lr'])
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
    optimizer,
    'min',
    factor=config['scheduler_factor'],
    patience=config['scheduler_patience'],
    min_lr=config['scheduler_min_lr']
)
summary(model_conv, input_size=(1024, 3, 5, 5))
```

Out[45]:
```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
Conv2DNN                                 [1024, 1]                 --
├─Conv2d: 1-1                            [1024, 200, 3, 3]         5,600
├─Linear: 1-2                            [1024, 200]               360,200
├─Linear: 1-3                            [1024, 200]               40,200
├─Linear: 1-4                            [1024, 1]                 201
==========================================================================================
Total params: 406,201
Trainable params: 406,201
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 461.83
==========================================================================================
Input size (MB): 0.31
Forward/backward pass size (MB): 18.03
Params size (MB): 1.62
Estimated Total Size (MB): 19.96
==========================================================================================
```

In [46]:
```
##############################################################################################
# TODO#16:                                                                                   #
# Complete the code to train your cnn model                                                  #
##############################################################################################
print('start training conv2d')
##############################################################################################
#                              WRITE YOUR CODE BELOW                                         #
##############################################################################################
train_losses = []
val_losses = []
learning_rates = []

# Start wandb run
wandb.init(
    project='precipitation-nowcasting-2025',
    config=config,
)

# Log parameters and gradients
wandb.watch(model_conv, log='all')

for epoch in range(config['epochs']):  # loop over the dataset multiple times

        # Training
        train_loss = []
        current_lr = optimizer.param_groups[0]['lr']
        learning_rates.append(current_lr)

        # Flag model as training. Some layers behave differently in training and
        # inference modes, such as dropout, BN, etc.
        model_conv.train()

        print(f"Training epoch {epoch+1}...")
        print(f"Current LR: {current_lr}")
```

```
        for i, (inputs, y_true) in enumerate(tqdm(train_loader)):
            # Transfer data from cpu to gpu
            inputs = inputs.to(device)
            y_true = y_true.to(device)

            # Reset the gradient
            optimizer.zero_grad()

            # Predict
            y_pred = model_conv(inputs)

            # Calculate loss
            loss = loss_fn(y_pred, y_true)

            # Compute gradient
            loss.backward()

            # Update parameters
            optimizer.step()

            # Log stuff
            train_loss.append(loss)

        avg_train_loss = torch.stack(train_loss).mean().item()
        train_losses.append(avg_train_loss)

        print(f"Epoch {epoch+1} train loss: {avg_train_loss:.4f}")

        # Validation
        model_conv.eval()
        with torch.no_grad(): # No gradient is required during validation
            print(f"Validating epoch {epoch+1}")
            val_loss = []
            for i, (inputs, y_true) in enumerate(tqdm(val_loader)):
                # Transfer data from cpu to gpu
                inputs = inputs.to(device)
                y_true = y_true.to(device)

                # Predict
                y_pred = model_conv(inputs)

                # Calculate loss
                loss = loss_fn(y_pred, y_true)

                # Log stuff
                val_loss.append(loss)

            avg_val_loss = torch.stack(val_loss).mean().item()
            val_losses.append(avg_val_loss)
            print(f"Epoch {epoch+1} val loss: {avg_val_loss:.4f}")

            # LR adjustment with scheduler
            scheduler.step(avg_val_loss)

            # Save checkpoint if val_loss is the best we got
            best_val_loss = np.inf if epoch == 0 else min(val_losses[:-1])
            if avg_val_loss < best_val_loss:
                # Save whatever you want
                state = {
                    'epoch': epoch,
```

```
                    'model': model_conv.state_dict(),
                    'optimizer': optimizer.state_dict(),
                    'scheduler': scheduler.state_dict(),
                    'train_loss': avg_train_loss,
                    'val_loss': avg_val_loss,
                    'best_val_loss': best_val_loss,
                }
                print(f"Saving new best model..")
                torch.save(state, 'model_conv.pth.tar')

        wandb.log({
            'train_loss': avg_train_loss,
            'val_loss': avg_val_loss,
            'lr': current_lr,
        })
wandb.finish()
print('Finished Training')
```

```
start training conv2d
```

View run **efficient-glitter-40** at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/9mv379pr

View project at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)

Find logs at: `.\wandb\run-20250415_163504-9mv379pr\logs`

Tracking run with wandb version 0.19.9

Run data is saved locally in `c:\Users\user\Desktop\CEDT\Code\AIML\NeuralNetwork1\wandb\run-20250415_163554-q77ftfa5`

Syncing run **fearless-microwave-41** to Weights & Biases (docs)

View project at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

View run at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/q77ftfa5

```
Training epoch 1...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 1 train loss: 2.2825
Validating epoch 1
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 1 val loss: 1.6580
Saving new best model..
Training epoch 2...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 2 train loss: 1.9197
Validating epoch 2
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 2 val loss: 1.6581
Training epoch 3...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 3 train loss: 1.9198
Validating epoch 3
  0%|          | 0/454 [00:00<?, ?it/s]
```

```
Epoch 3 val loss: 1.6589
Training epoch 4...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 4 train loss: 1.9193
Validating epoch 4
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 4 val loss: 1.6580
Training epoch 5...
Current LR: 0.002
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 5 train loss: 1.9189
Validating epoch 5
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 5 val loss: 1.6573
Saving new best model..
Training epoch 6...
Current LR: 0.002
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 6 train loss: 1.9188
Validating epoch 6
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 6 val loss: 1.6577
Training epoch 7...
Current LR: 0.002
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 7 train loss: 1.9189
Validating epoch 7
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 7 val loss: 1.6574
Training epoch 8...
Current LR: 0.002
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 8 train loss: 1.9189
Validating epoch 8
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 8 val loss: 1.6581
Training epoch 9...
Current LR: 0.0004
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 9 train loss: 1.9182
Validating epoch 9
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 9 val loss: 1.6562
Saving new best model..
Training epoch 10...
Current LR: 0.0004
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 10 train loss: 1.9182
Validating epoch 10
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 10 val loss: 1.6565
```

Run history:

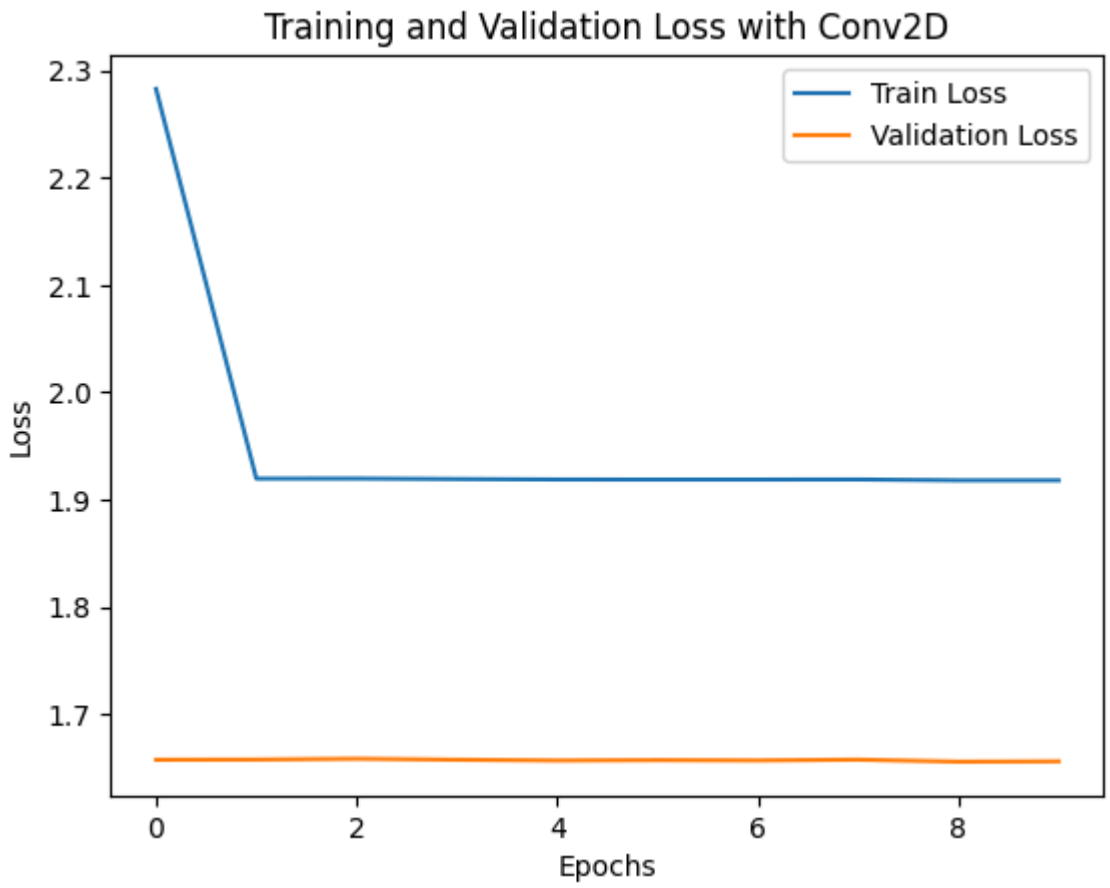
Run summary:

lr	0.0004
train_loss	1.91815
val_loss	1.65648

View run **fearless-microwave-41** at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/q77ftfa5
View project at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025
Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)
Find logs at: `.\wandb\run-20250415_163554-q77ftfa5\logs`

```
Finished Training
```

In [47]:
```
# Plot losses
plt.plot(train_losses, label='Train Loss')
plt.plot(val_losses, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss with Conv2D')
plt.legend()
plt.show()
```



In [48]:
```
# Evaluate
print('train', evaluate(train_loader, model_conv))
print('validate', evaluate(val_loader, model_conv))
```

Tracking run with wandb version 0.19.9

Run data is saved locally in

`c:\Users\user\Desktop\CEDT\Code\AIML\NeuralNetwork1\wandb\run-20250415_173223-s6w62nvx`

Syncing run **classic-frog-42** to Weights & Biases (docs)

View project at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

View run at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/s6w62nvx

View run **classic-frog-42** at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/s6w62nvx

View project at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)

Find logs at: `.\wandb\run-20250415_173223-s6w62nvx\logs`

```
train 1.9180470301905654
```

Tracking run with wandb version 0.19.9

Run data is saved locally in

`c:\Users\user\Desktop\CEDT\Code\AIML\NeuralNetwork1\wandb\run-20250415_173418-zb7lktdu`

Syncing run **cool-water-43** to Weights & Biases (docs)

View project at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025
View run at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/zb7lktdu
View run **cool-water-43** at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/zb7lktdu
View project at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025
Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)
Find logs at: `.\wandb\run-20250415_173418-zb7lktdu\logs`

```
validate 1.658926238071226
```

Final Section

PyTorch playground

Now, train the best model you can do for this task. You can use any model structure and function available.
Remember that trainig time increases with the complexity of the model. You might find printing computation graphs helpful in debugging complicated models.
Your model should be better than your models in the previous sections

Some ideas:

- Tune the hyperparameters
- Change the model architecture
- Use or ignore the time sequence information

You should tune your model on training and validation set.
The test set should be used only for the last evaluation.

In [74]:
```
################################################################################
# TODO#17                                                                      #
# Write a function that returns your best PyTorch model. You can use anything  #
# you want. The goal here is to create the best model you can think of.        #
#                                                                              #
# Hint: You should read PyTorch documentation to see the list of available     #
# layers and options you can use.                                              #
################################################################################
#                             WRITE YOUR CODE BELOW                            #
################################################################################
class BestModel(nn.Module):
    def __init__(self):
        super(BestModel, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, pa
        self.bn1 = nn.BatchNorm2d(64)  # Add batch normalization
        self.conv2 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3,
        self.bn2 = nn.BatchNorm2d(128)  # Add batch normalization
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
```

```
        self.fc1 = nn.Linear(128 * 2 * 2, 256)
        self.dropout1 = nn.Dropout(0.4)  # Increase dropout rate
        self.fc2 = nn.Linear(256, 128)
        self.dropout2 = nn.Dropout(0.4)  # Increase dropout rate
        self.out = nn.Linear(128, 1)

    def forward(self, x):
        x = x.view(-1, 3, 5, 5)  # Reshape input to (batch_size, channels, heigh
        x = F.relu(self.bn1(self.conv1(x)))  # Apply convolution, batch normaliz
        x = self.pool(F.relu(self.bn2(self.conv2(x))))  # Apply convolution, bat
        x = torch.flatten(x, start_dim=1)  # Flatten for fully connected layers
        x = F.relu(self.fc1(x))
        x = self.dropout1(x)  # Apply dropout
        x = F.relu(self.fc2(x))
        x = self.dropout2(x)  # Apply dropout
        x = self.out(x)
        return x.reshape(-1, 1)

def get_best_model():
    model = BestModel()
    return model
```

In [75]:

```
################################################################################
# TODO#18                                                                      #
# Complete the code to train your best model                                   #
################################################################################
print('start training the best model')
################################################################################
#                            WRITE YOUR CODE BELOW                             #
################################################################################
# Initialize the best model
model_best = get_best_model()
model_best = model_best.to(device)

# Define optimizer and scheduler
optimizer = torch.optim.Adam(model_best.parameters(), lr=config['lr'])
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
    optimizer,
    'min',
    factor=config['scheduler_factor'],
    patience=config['scheduler_patience'],
    min_lr=config['scheduler_min_lr']
)

# Training loop
train_losses = []
val_losses = []
learning_rates = []

# Start wandb run
wandb.init(
    project='precipitation-nowcasting-2025',
    config=config,
)

# Log parameters and gradients
wandb.watch(model_best, log='all')

for epoch in range(config['epochs']):  # loop over the dataset multiple times
```

```
    # Training
    train_loss = []
    current_lr = optimizer.param_groups[0]['lr']
    learning_rates.append(current_lr)

    # Flag model as training
    model_best.train()

    print(f"Training epoch {epoch+1}...")
    print(f"Current LR: {current_lr}")

    for i, (inputs, y_true) in enumerate(tqdm(train_loader)):
        # Transfer data from cpu to gpu
        inputs = inputs.to(device)
        y_true = y_true.to(device)

        # Reset the gradient
        optimizer.zero_grad()

        # Predict
        y_pred = model_best(inputs)

        # Calculate loss
        loss = loss_fn(y_pred, y_true)

        # Compute gradient
        loss.backward()

        # Update parameters
        optimizer.step()

        # Log stuff
        train_loss.append(loss)

    avg_train_loss = torch.stack(train_loss).mean().item()
    train_losses.append(avg_train_loss)

    print(f"Epoch {epoch+1} train loss: {avg_train_loss:.4f}")

    # Validation
    model_best.eval()
    with torch.no_grad():  # No gradient is required during validation
        print(f"Validating epoch {epoch+1}")
        val_loss = []
        for i, (inputs, y_true) in enumerate(tqdm(val_loader)):
            # Transfer data from cpu to gpu
            inputs = inputs.to(device)
            y_true = y_true.to(device)

            # Predict
            y_pred = model_best(inputs)

            # Calculate loss
            loss = loss_fn(y_pred, y_true)

            # Log stuff
            val_loss.append(loss)

        avg_val_loss = torch.stack(val_loss).mean().item()
        val_losses.append(avg_val_loss)
```

```
        print(f"Epoch {epoch+1} val loss: {avg_val_loss:.4f}")

        # LR adjustment with scheduler
        scheduler.step(avg_val_loss)

        # Save checkpoint if val_loss is the best we got
        best_val_loss = np.inf if epoch == 0 else min(val_losses[:-1])
        if avg_val_loss < best_val_loss:
            # Save whatever you want
            state = {
                'epoch': epoch,
                'model': model_best.state_dict(),
                'optimizer': optimizer.state_dict(),
                'scheduler': scheduler.state_dict(),
                'train_loss': avg_train_loss,
                'val_loss': avg_val_loss,
                'best_val_loss': best_val_loss,
            }

            print(f"Saving new best model..")
            torch.save(state, 'model_best.pth.tar')

    wandb.log({
        'train_loss': avg_train_loss,
        'val_loss': avg_val_loss,
        'lr': current_lr,
    })

wandb.finish()
print('Finished Training')
```

```
start training the best model
```

Tracking run with wandb version 0.19.9

Run data is saved locally in

`c:\Users\user\Desktop\CEDT\Code\AIML\NeuralNetwork1\wandb\run-20250415_203747-eb821aqw`

Syncing run **true-hill-59** to Weights & Biases (docs)

View project at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

View run at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/eb821aqw

```
Training epoch 1...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 1 train loss: 1.9461
Validating epoch 1
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 1 val loss: 1.6601
Saving new best model..
Training epoch 2...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 2 train loss: 1.9212
Validating epoch 2
  0%|          | 0/454 [00:00<?, ?it/s]
```

```
Epoch 2 val loss: 1.6604
Training epoch 3...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 3 train loss: 1.9212
Validating epoch 3
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 3 val loss: 1.6626
Training epoch 4...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 4 train loss: 1.9211
Validating epoch 4
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 4 val loss: 1.6598
Saving new best model..
Training epoch 5...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 5 train loss: 1.9206
Validating epoch 5
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 5 val loss: 1.6590
Saving new best model..
Training epoch 6...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 6 train loss: 1.9207
Validating epoch 6
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 6 val loss: 1.6589
Saving new best model..
Training epoch 7...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 7 train loss: 1.9221
Validating epoch 7
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 7 val loss: 1.6619
Training epoch 8...
Current LR: 0.01
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 8 train loss: 1.9239
Validating epoch 8
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 8 val loss: 1.6606
Training epoch 9...
Current LR: 0.002
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 9 train loss: 1.9239
Validating epoch 9
  0%|          | 0/454 [00:00<?, ?it/s]
Epoch 9 val loss: 1.6616
Training epoch 10...
Current LR: 0.002
  0%|          | 0/1121 [00:00<?, ?it/s]
Epoch 10 train loss: 1.9222
Validating epoch 10
  0%|          | 0/454 [00:00<?, ?it/s]
```

```
Epoch 10 val loss: 1.6584
Saving new best model..
```

Run history:

Run summary:

lr	0.002
train_loss	1.92222
val_loss	1.65839

View run **true-hill-59** at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/eb821aqw

View project at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)

Find logs at: `.\wandb\run-20250415_203747-eb821aqw\logs`

```
Finished Training
```

In [76]:
```python
# Evaluate best model on validation and test set
val_loss = evaluate(val_loader, model_best)
test_loss = evaluate(test_loader, model_best)

print(f"Validation Loss: {val_loss:.4f}")
print(f"Test Loss: {test_loss:.4f}")
```

Tracking run with wandb version 0.19.9

Run data is saved locally in `c:\Users\user\Desktop\CEDT\Code\AIML\NeuralNetwork1\wandb\run-20250415_213835-a5hntm0c`

Syncing run **amber-hill-60** to Weights & Biases (docs)

View project at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

View run at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/a5hntm0c

View run **amber-hill-60** at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/a5hntm0c

View project at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)

Find logs at: `.\wandb\run-20250415_213835-a5hntm0c\logs`

Tracking run with wandb version 0.19.9

Run data is saved locally in `c:\Users\user\Desktop\CEDT\Code\AIML\NeuralNetwork1\wandb\run-20250415_213931-djez8nvf`

Syncing run **bright-star-61** to Weights & Biases (docs)

View project at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

View run at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/djez8nvf

View run **bright-star-61** at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/djez8nvf

View project at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)

Find logs at: `.\wandb\run-20250415_213931-djez8nvf\logs`

```
Validation Loss: 1.6609
Test Loss: 1.1634
```

In [73]:
```python
# Also evaluate your fully-connected and dropout model on the test set.
ff_loss = evaluate(test_loader, model_ff)
dropout_loss = evaluate(test_loader, model_dropout)

print(f"FF Model Test Loss: {ff_loss:.4f}")
print(f"Dropout Model Test Loss: {dropout_loss:.4f}")
```

Tracking run with wandb version 0.19.9

Run data is saved locally in `c:\Users\user\Desktop\CEDT\Code\AIML\NeuralNetwork1\wandb\run-20250415_203419-xgupk4lt`

Syncing run **smooth-tree-57** to Weights & Biases (docs)

View project at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

View run at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/xgupk4lt

View run **smooth-tree-57** at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/xgupk4lt

View project at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025

Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)

Find logs at: `.\wandb\run-20250415_203419-xgupk4lt\logs`

Tracking run with wandb version 0.19.9

Run data is saved locally in `c:\Users\user\Desktop\CEDT\Code\AIML\NeuralNetwork1\wandb\run-20250415_203508-4cjqkgd3`
Syncing run **hardy-aardvark-58** to Weights & Biases (docs)
View project at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025
View run at https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/4cjqkgd3
View run **hardy-aardvark-58** at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025/runs/4cjqkgd3
View project at: https://wandb.ai/ittikorn-chulalongkorn-university/precipitation-nowcasting-2025
Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)
Find logs at: `.\wandb\run-20250415_203508-4cjqkgd3\logs`

```
FF Model Test Loss: 1.1672
Dropout Model Test Loss: 1.1660
```

To get full credit for this part, your best model should be better than the previous models on the **test set**.

TODO#19

Explain what helped and what did not help here

Ans:

- **What helped:**
  - Adding dropout layers reduced overfitting by regularization.
  - Batch normalization improved training stability and convergence.
  - Using CNN with spatial data improve performance.
  - Hyperparameter tuning improved model performance.
- **What did not help:**
  - Increased model complexity lead to overfitting.
  - High learning rate cause instability.
  - Adding too many layers resulted in increased training time.

[Optional] Augmentation using data loader

Optional TODO#2

Implement a new dataloader on your best model that will perform data augmentation. Try adding noise of zero mean and variance of $10e^{-2}$.

Then, train your model.

In [ ]:
```
# Write Dataset/DataLoader with noise here
```

In [ ]:
```
print('start training the best model with noise')
################################################################################
#                              WRITE YOUR CODE BELOW                           #
################################################################################
```

In [ ]:
```
# Evaluate the best model trained with noise on validation and test set
```