Class 1

Power System Event Classification with Convolutional Neural Network Using PMU Data – Model Development

Outline

- Background and Challenges
- Overview of Convolution Neural Network (CNN) and Event Classification Problem
- Types of Power System Events
- Data Preprocessing
- Convolutional Neural Network
- Assignment walk-through

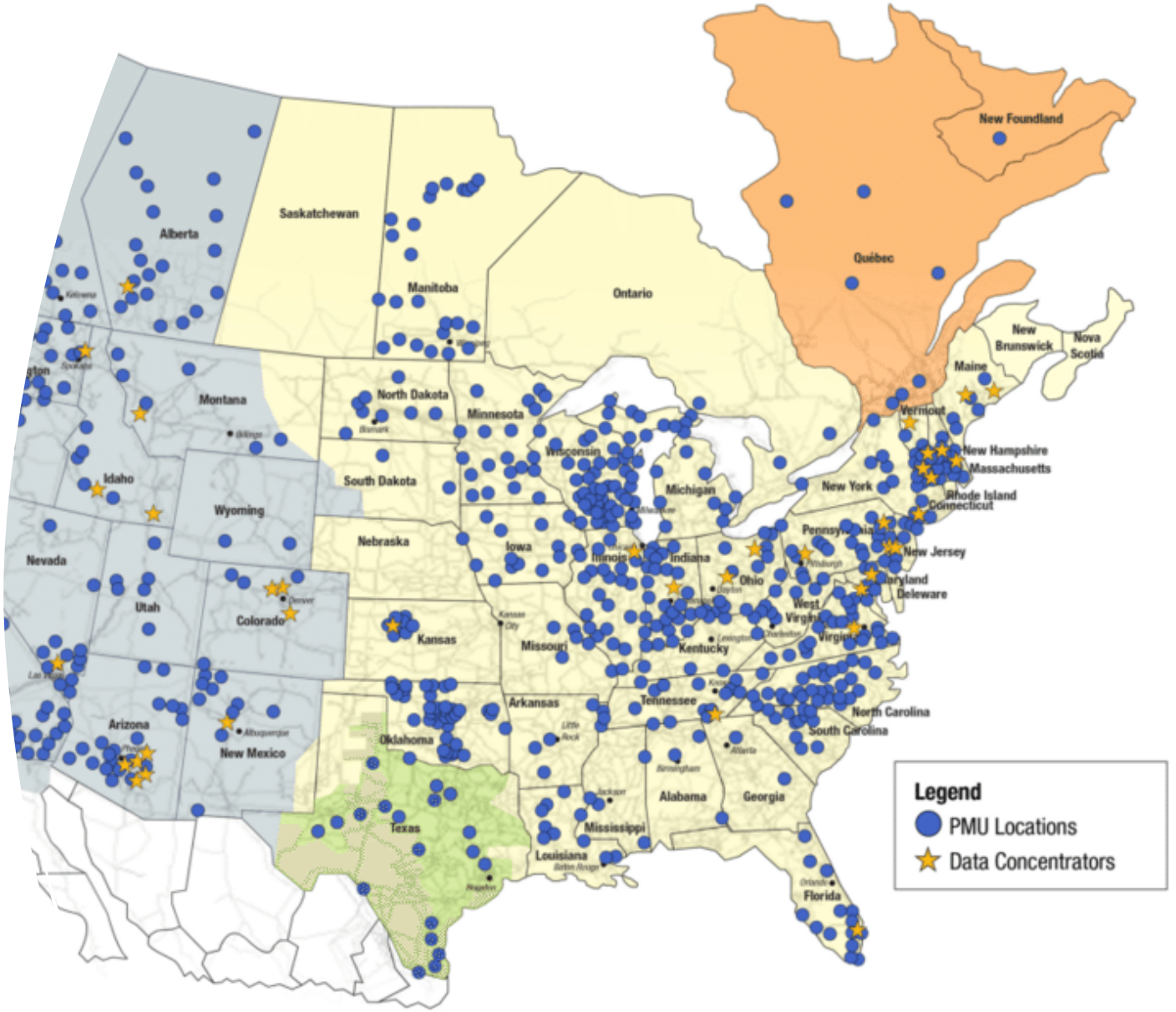
Outline

- Background and Challenges
- Overview of Convolution Neural Network (CNN) and Event Classification Problem
- Types of Power System Events
- Data Preprocessing
- Convolutional Neural Network
- Assignment walk-through

Background

- Phasor Measurement Units (PMU) are high resolution sensors deployed in high-voltage transmission systems.
  - Measure current and voltage phasors with high sampling frequency and accuracy.

- The number of PMUs has drastically increased in the past decade.
  - Less than 500 (2003) -- 2,000 (2018) in North America.



Why is event classification important?

Prompt and accurate power system event identification

- Can help system operator better understand the overall status of the whole power system.

- Enables system operators or control systems to take appropriate corrective control actions to avoid blackouts.

Challenges in Event Classification

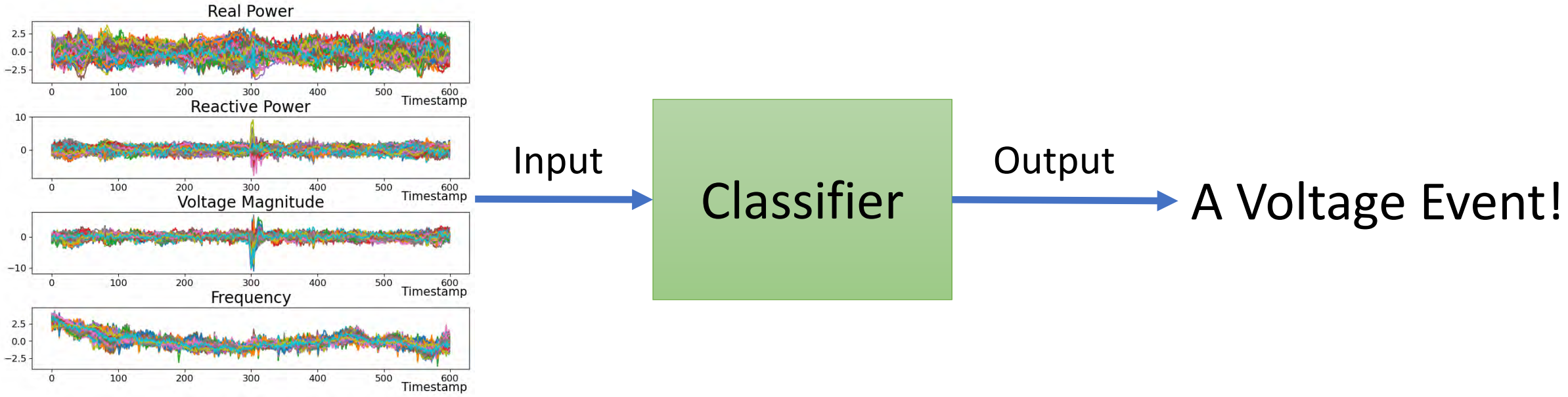
- Volume of Data
  - Large number of PMUs
  - High sampling frequency (30 Hz or more)
  - Multiple measurement channels (Voltage, Current, Frequency)
  - 26 GB data volume per day in case of 200 PMUs
- Detection Speed
  - Online event detection requires fast data processing to mitigate grid impact, thwarting large-scale power outages

Outline

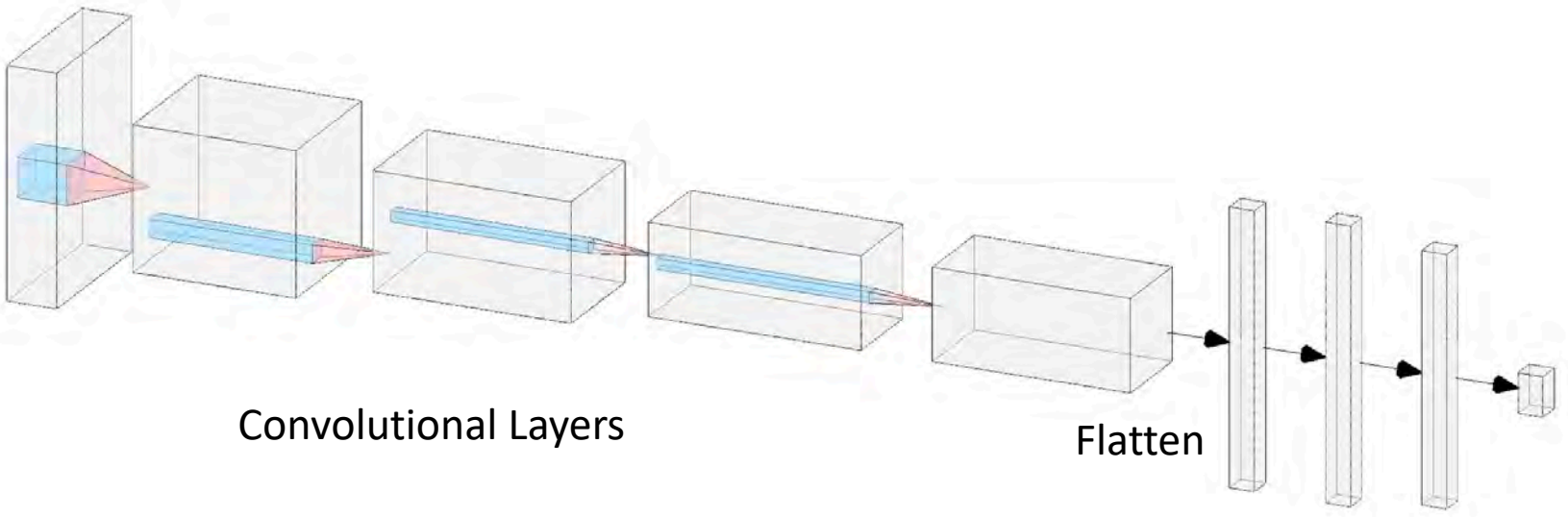
- Background and Challenges
- Overview of Convolution Neural Network (CNN) and Event Classification Problem
- Types of Power System Events
- Data Preprocessing
- Convolutional Neural Network
- Assignment walk-through

Overview of Power System Event Classification Problem

- Input X: A time-series of measurements from power system sensors.
- Output y: The prediction of event type.
- Classifier $f(X, \theta)$: Takes the input and outputs the event type prediction.



Overview of Convolution Neural Network (CNN)

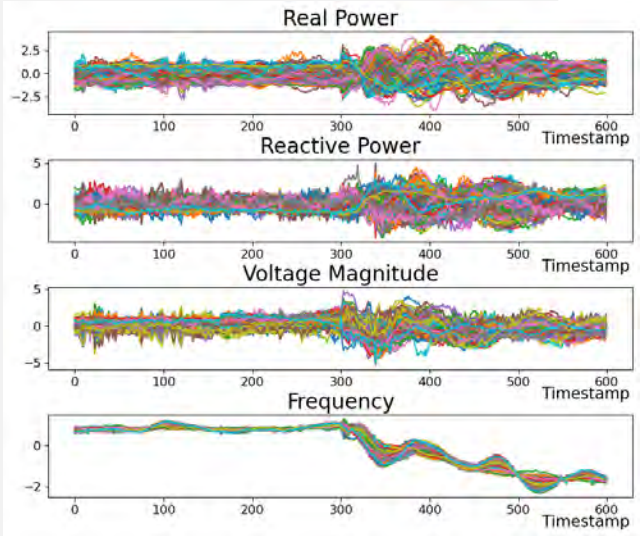


Outline

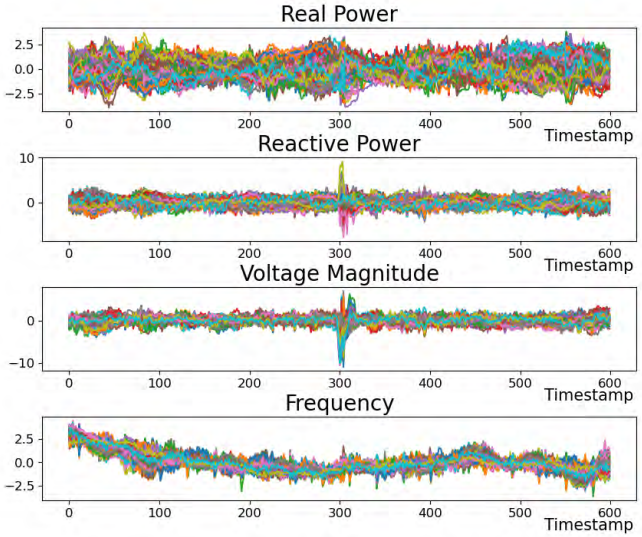
- Background and Challenges
- Overview of Convolution Neural Network (CNN) and Event Classification Problem
- Types of Power System Events
- Data Preprocessing
- Convolutional Neural Network
- Assignment walk-through

Types of Power System Events

- Voltage-related Event
  - Line outage events
- Frequency-related Event
  - Generator outage Events
- Oscillation Event



Frequency-related Event



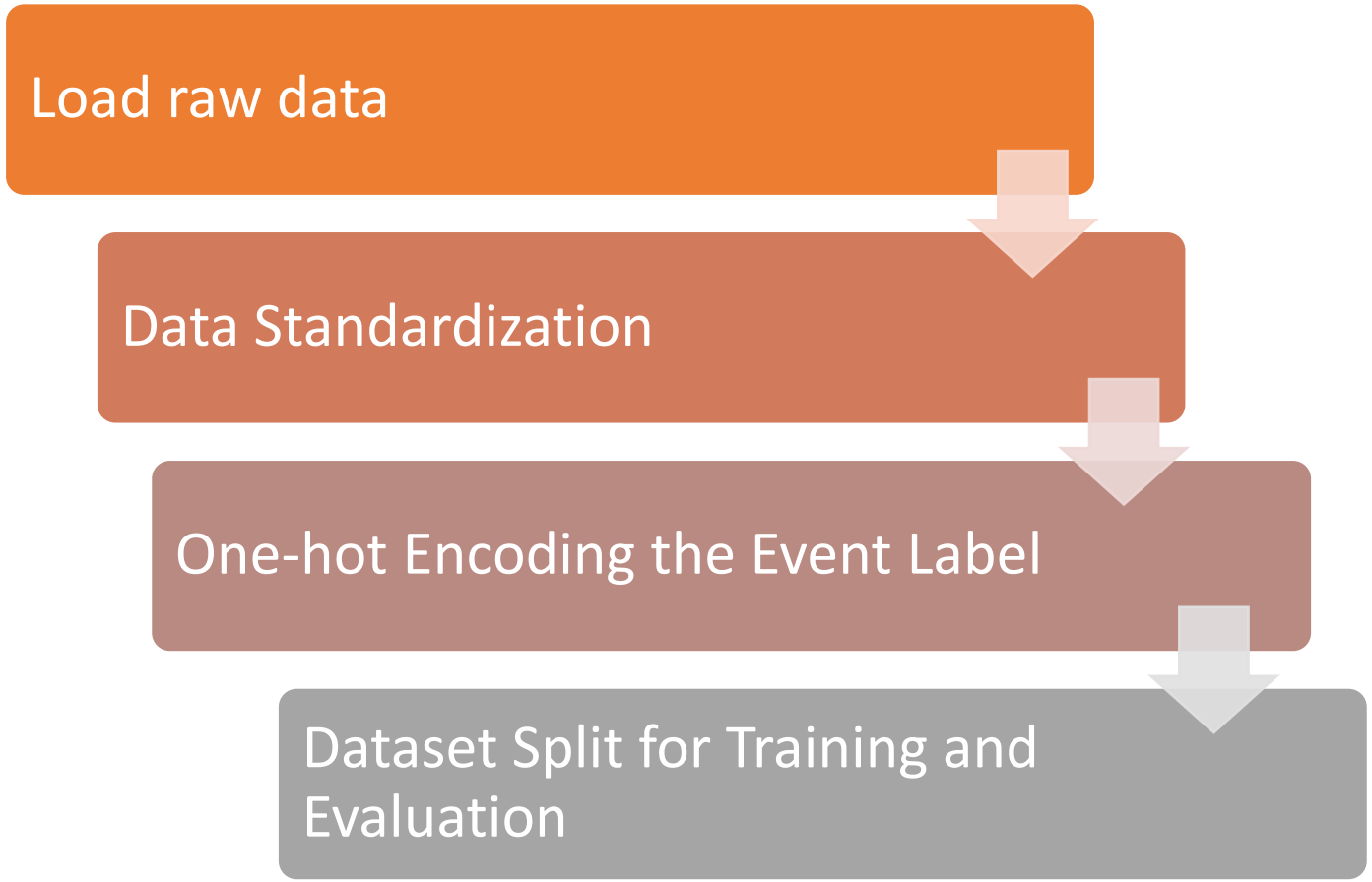
Voltage-related Event

Outline

- Background and Challenges
- Overview of Convolution Neural Network (CNN) and Event Classification Problem
- Types of Power System Events
- Data Preprocessing
- Convolutional Neural Network
- Assignment walk-through

Data Preprocessing

- The data preprocessing techniques involved in this course:



Load raw Data

- The dataset used in this course is the pmuBAGE, which can be downloaded from GitHub: https://github.com/NanpengYu/pmuBAGE. Note that the pmuBAGE is publicly available dataset library that generates mimic PMU measurements from actually recorded PMU measurement.
- The dataset consists of 84 synthetic frequency events and 620 synthetic voltage events.
- They are split into several partitions.

Data Standardization

- Definition: center a variable and normalize it by its standard deviation.
- Purpose: shift and rescale feature variables, so that they are in similar ranges. This will improve the convergence in the training process of machine learning algorithms.
- Example
  - A power system event sample $\{X_t\}$, $t = 1,2,3, \ldots$
  - Standardization $\widetilde{X_t} = \frac{X_t - \mu_X}{\sigma_X}$
  - Here μ_X is the mean value of $\{X_t\}$, σ_X is the standard deviation of $\{X_t\}$.

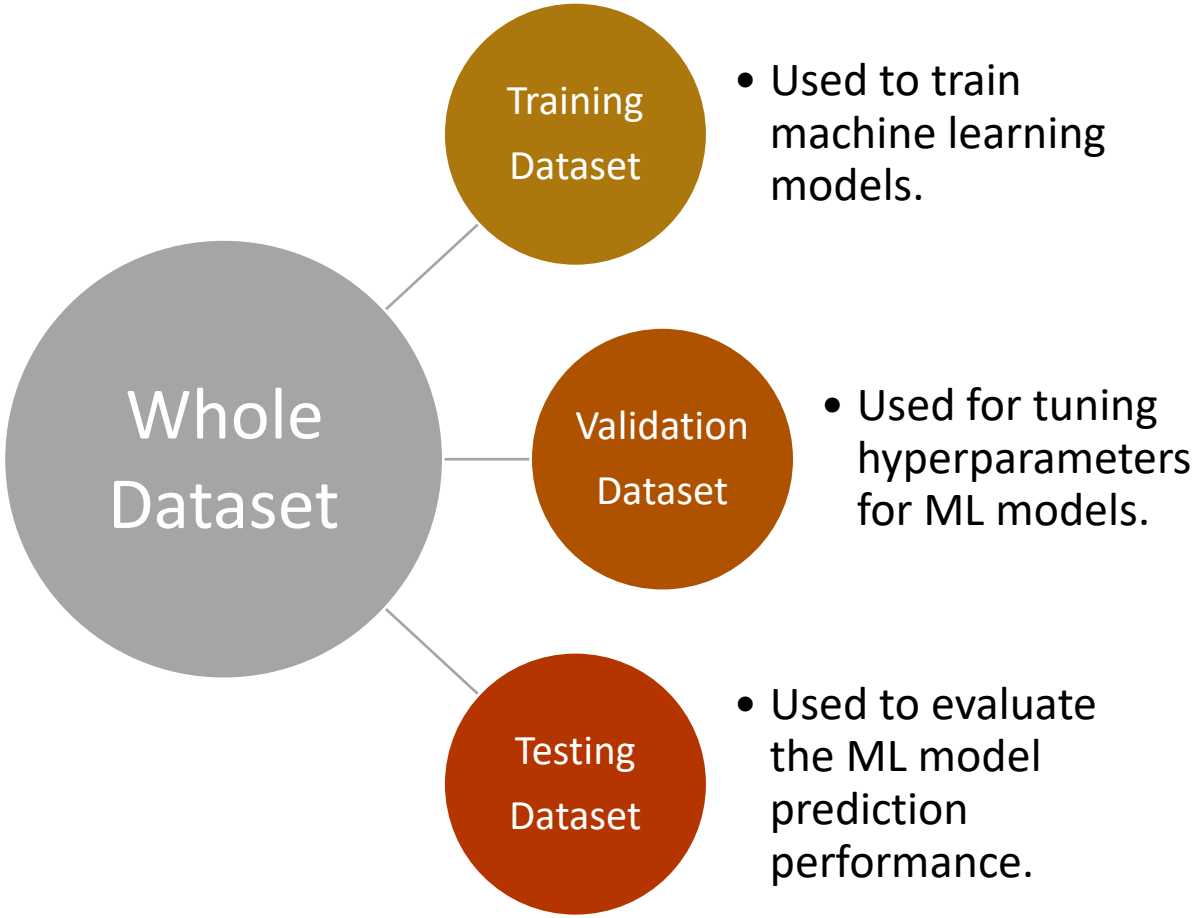
One-hot Encoding for the Label

- For categorical features.

Date Type
Voltage Event
Frequency Event
Frequency Event
Voltage Event
Voltage Event

Voltage Event	Frequency Event
1	0
0	1
0	1
1	0
1	0

Dataset Split for Train and Evaluation



Outline

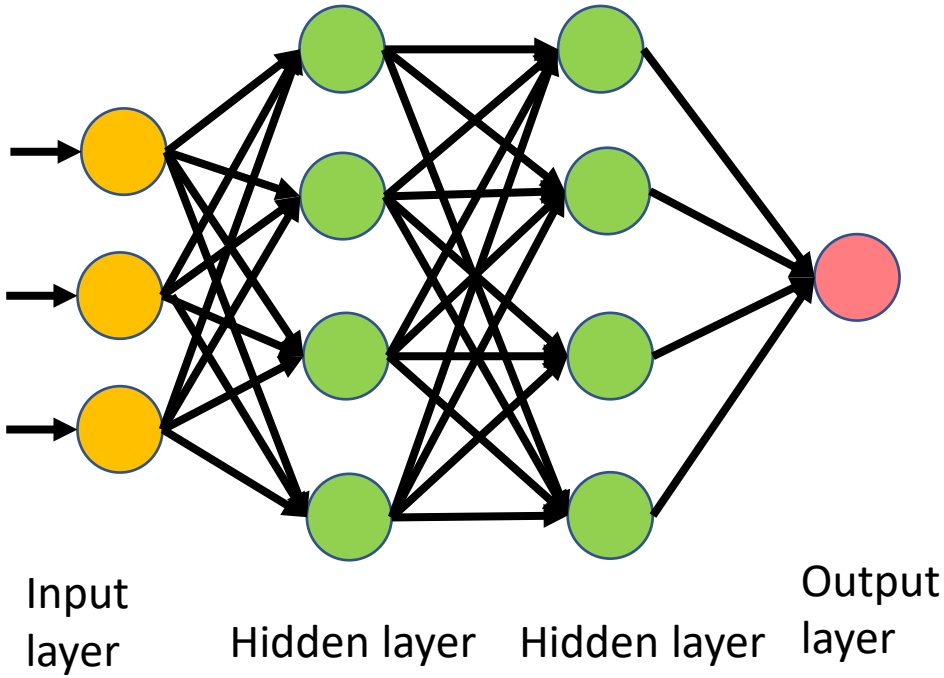
- Background and Challenges
- Overview of Convolution Neural Network (CNN) and Event Classification Problem
- Types of Power System Events
- Data Preprocessing
- Convolutional Neural Network
- Assignment walk-through

Convolutional Neural Network (CNN)

- Fully Connected Neural Network
- Convolutional Layer
- Max-Pooling Layer
- Convolutional Neural Network
- SoftMax and Loss function
- Back Propagation Algorithm

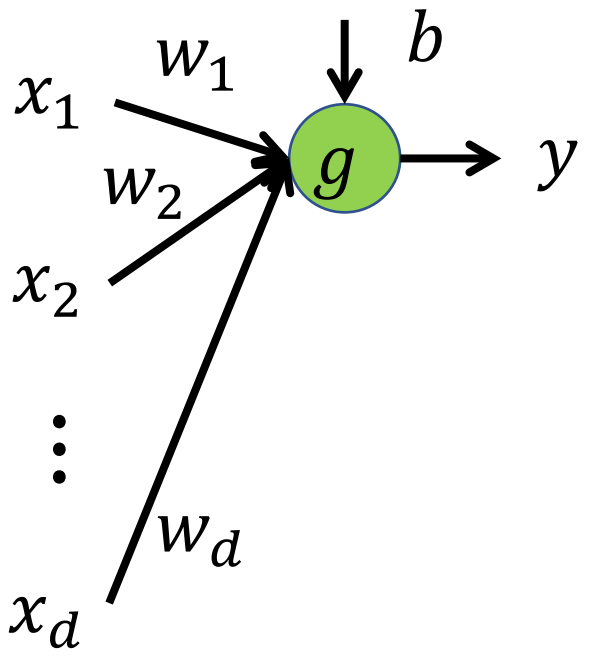
Start from Fully Connected Neural Network

- Fully Connected Neural Network
  - A fully connected neural network consists of a series of fully connected layers connecting every neuron in one layer to every neuron in the other.
  - Each neuron takes the previous layers' output as input and performs a linear transformation and an activation function over them.



Forward-propagating of a single neuron

- Input: $x_1, x_2, \ldots, x_d$ (from previous layer)
- Connection weight: $w_1, w_2, \ldots, w_d$.
- A bias b can be added.
- Neuron activation function: $g()$
- Output: $y = g(a)$, in which $a = b + \sum_{i=1}^{d} w_i x_i$
- Activation function $g()$ can be a linear or nonlinear function.



Examples of Activation Functions

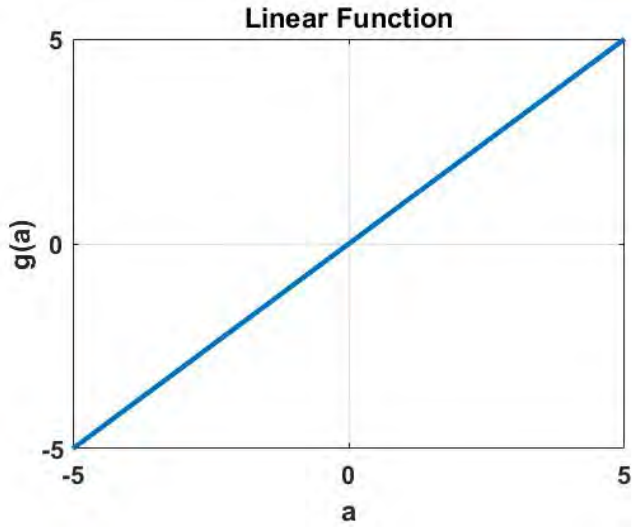


Figure 1: $g(a) = a$

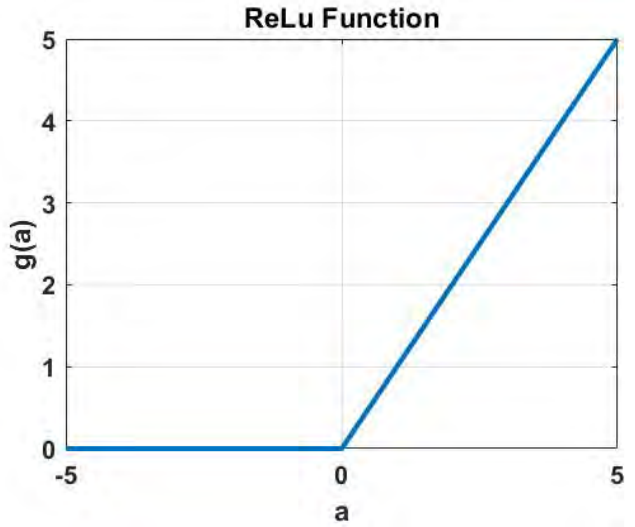


Figure 3: $g(a) = \max(0, a)$

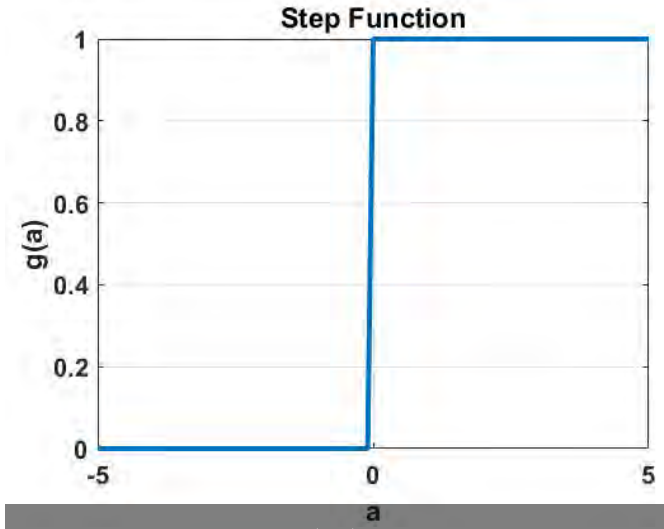


Figure 2: $g(a) = 1$ if $a \geq 0$, otherwise $g(a) = 0$

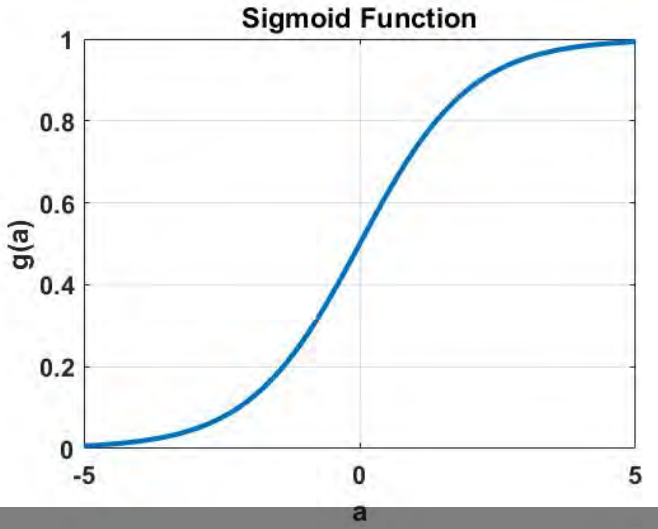


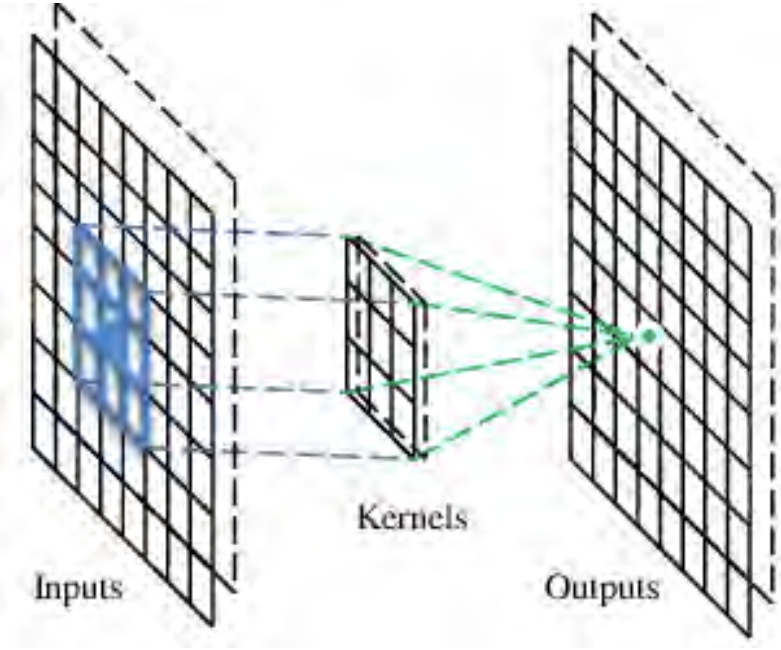
Figure 4: $g(a) = 1/(1 + e^{-a})$

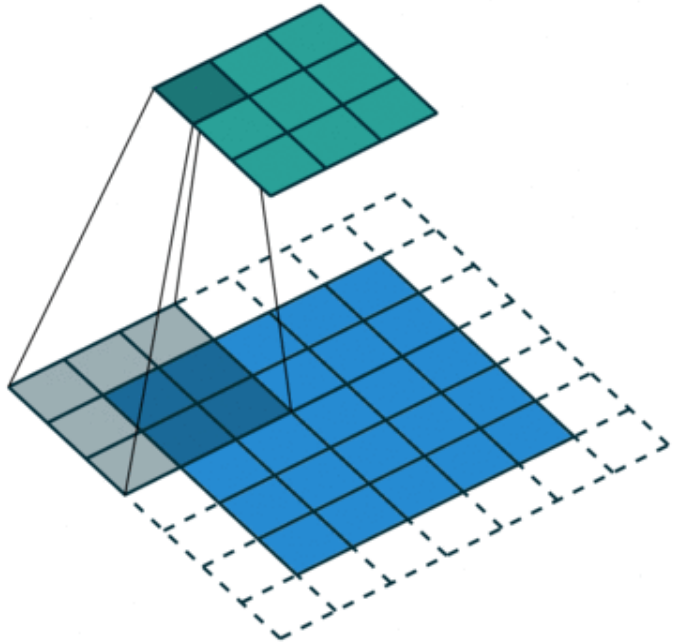
Too many parameters in the Fully Connected Neural Network

- Suppose the input of the neural network is a series of power system data (30 Hz):
  - 20 second (600 timestamp)
  - 100 sensors in total
  - 4 measurements (active power, reactive power, voltage magnitude, frequency)
- The input size is $20 \times 100 \times 4 = 8000$
- Two hidden layers, each of which has 512 neurons
- Output size: 2 classes
- Total Parameters: $8000 \times 512 + 512 \times 512 + 512 \times 2 \approx 4$ million!

The Convolutional layer

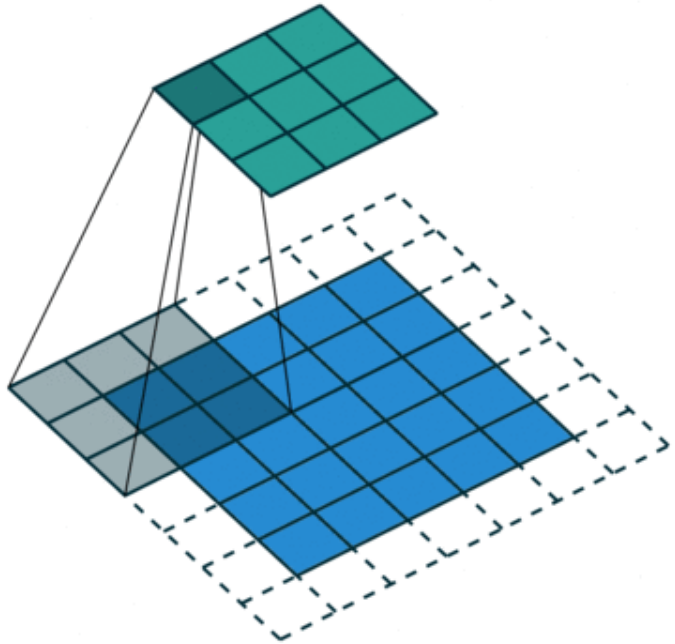
- The neurons in a Convolutional layer will only be connected to a small region of the previous layer instead of all the neurons in a fully connected manner.
- A convolutional layer contains multiple filters that perform convolutional operations.
- A Convolutional layer helps Reduce the parameter number.



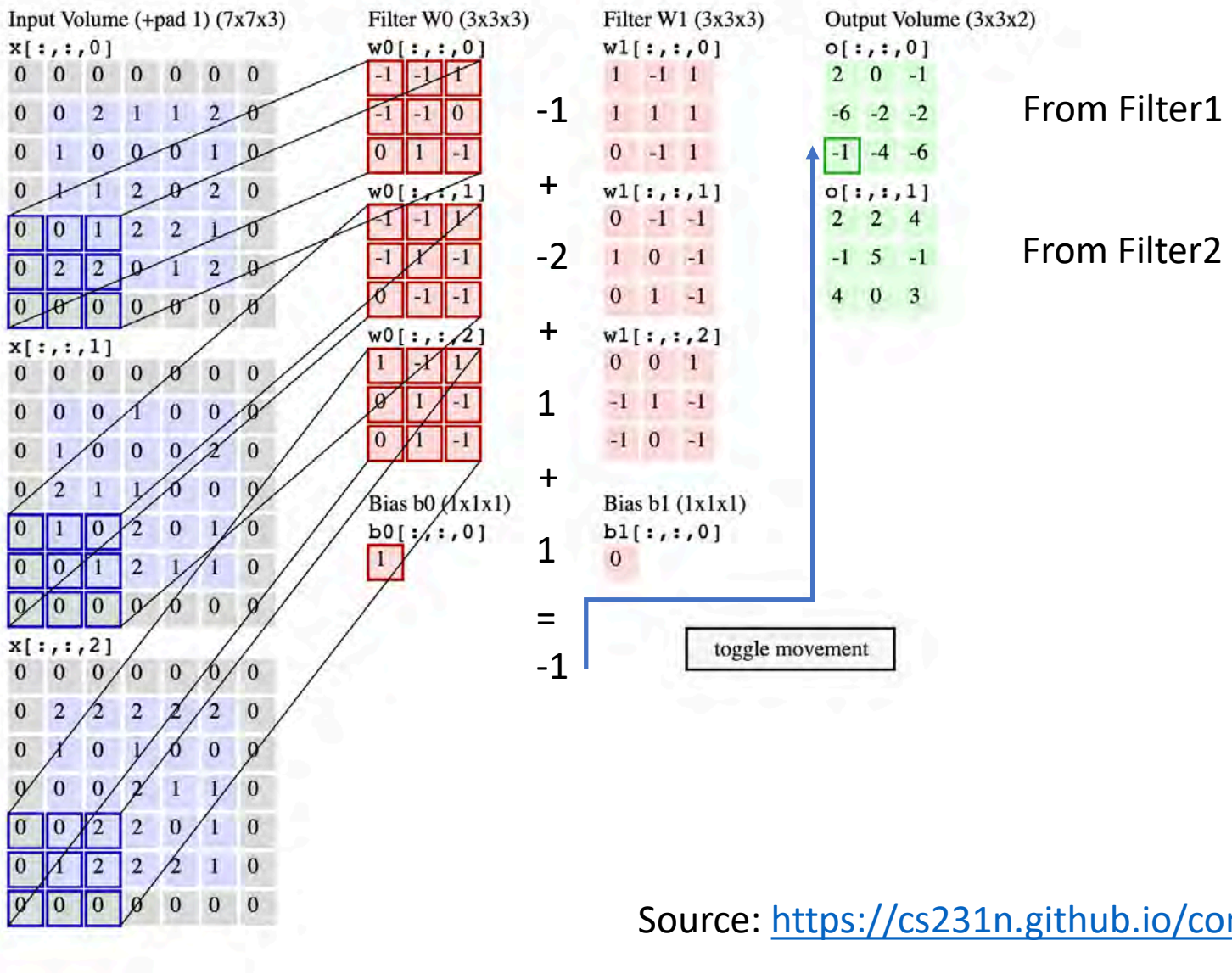


Slides and Zero Padding

- Suppose the input of a convolutional layer is $5 \times 5 \times 3$
- A slide captures the small region of 3×3. All 3×3 share and re-use the slide parameters.
- Add zero padding to avoid the size shrink. (Optional)
- Each slide contains $3 \times 3 \times 3$ parameters and produces the $5 \times 5 \times 1$ output.
- Apply multiple slides to produce multiple channels in the output.

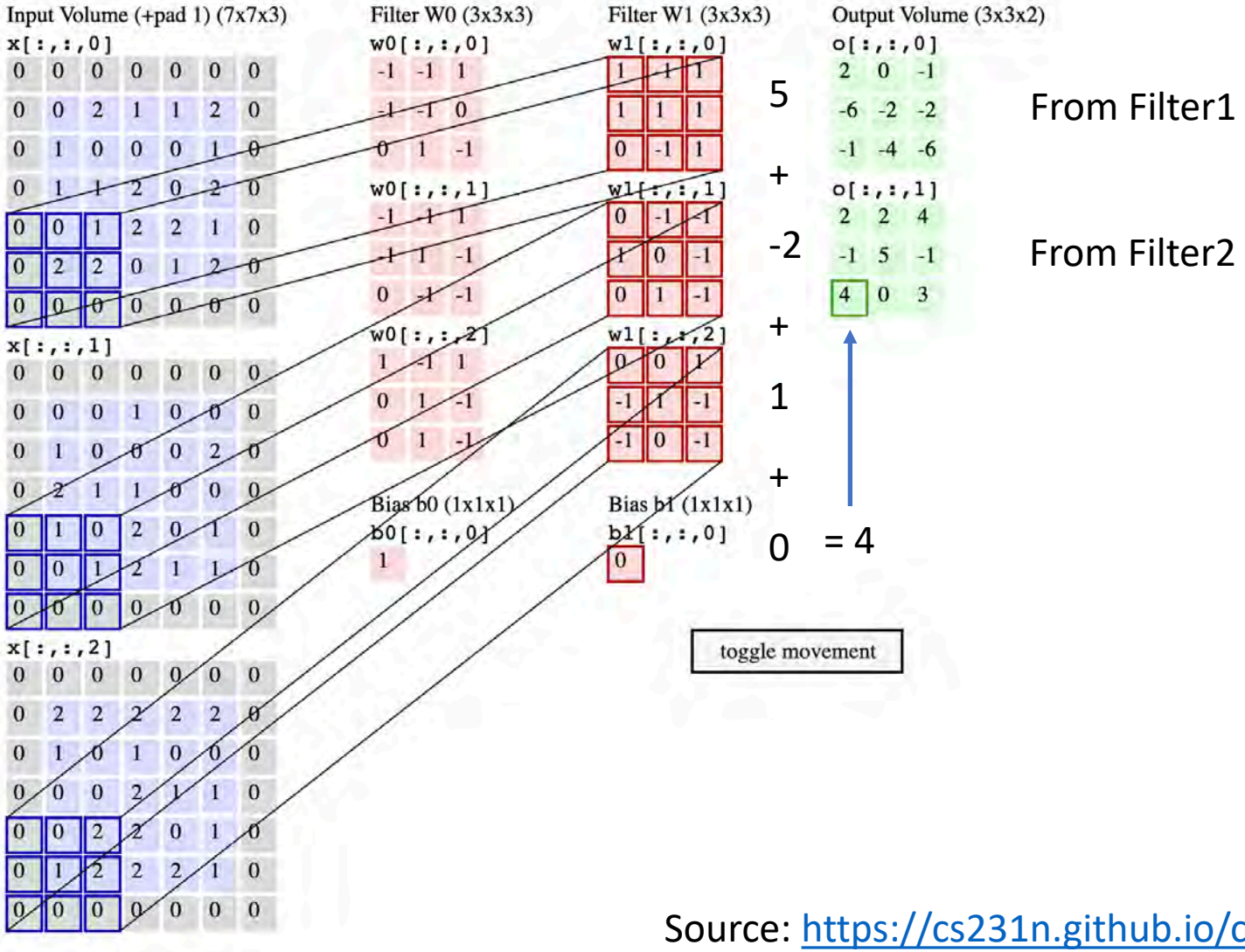


Convolutional Layer (Simple Demo)



Source: https://cs231n.github.io/convolutional-networks/

Convolutional Layer (Simple Demo)

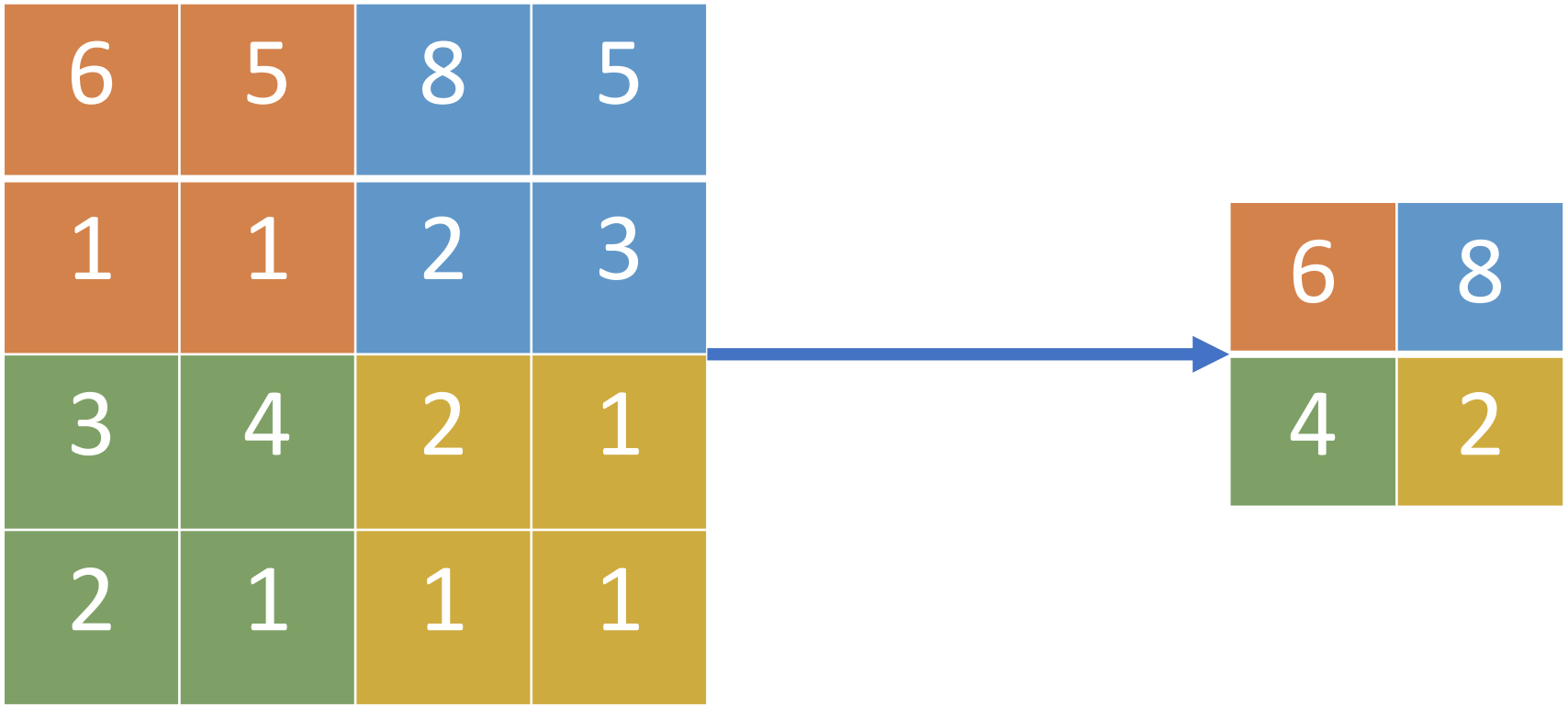


Source: https://cs231n.github.io/convolutional-networks/

Max-Pooling Layer

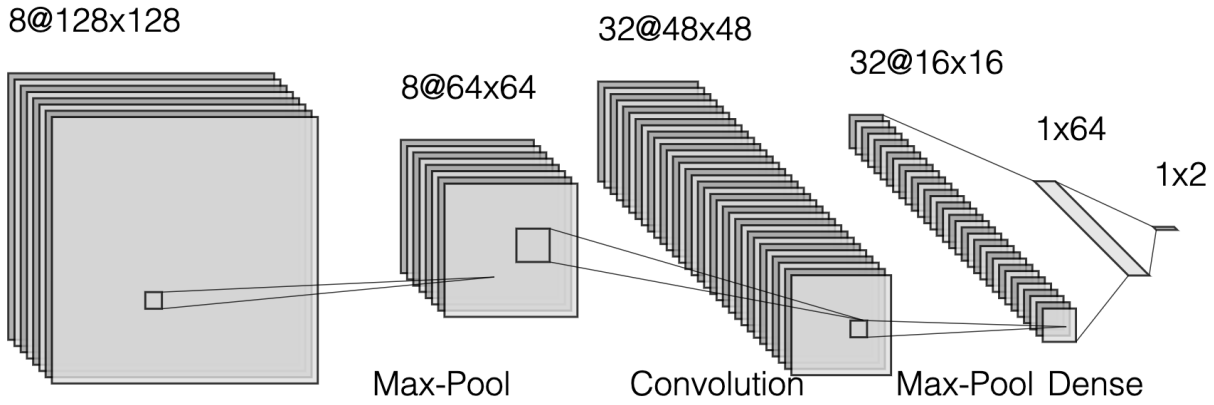
- The max-pooling layer is commonly used in convolutional neural networks.
  - It progressively reduces the spatial size of the representation to reduce the number of parameters.
  - It also controls overfitting (will be covered later).
  - The most common form is a pooling layer with filters of size 2×2 applied with a stride of 2 at every depth. In this case, every MAX operation would take a max over 4 numbers.

Max-Pooling Layer (Simple Demo)



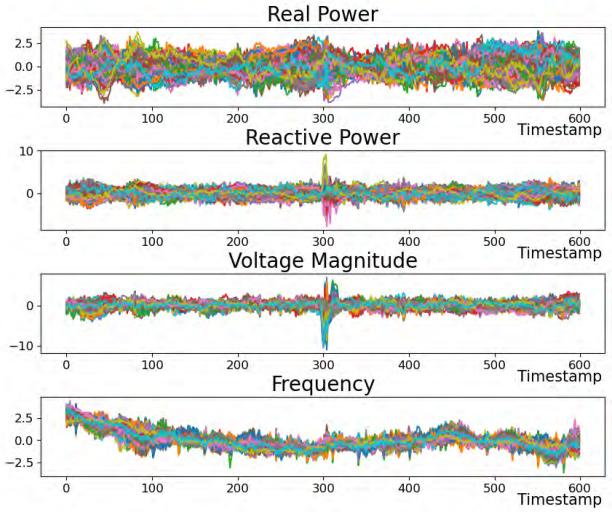
Convolutional Neural Network (CNN)

- A Convolutional Neural Network contains some convolutional layers (and some other layers).
- For instance, the right-hand side Convolutional Neural Network has two convolutional layers and two max-pooling layers and two fully connected layers.



Softmax Function

A Power System Event Sample

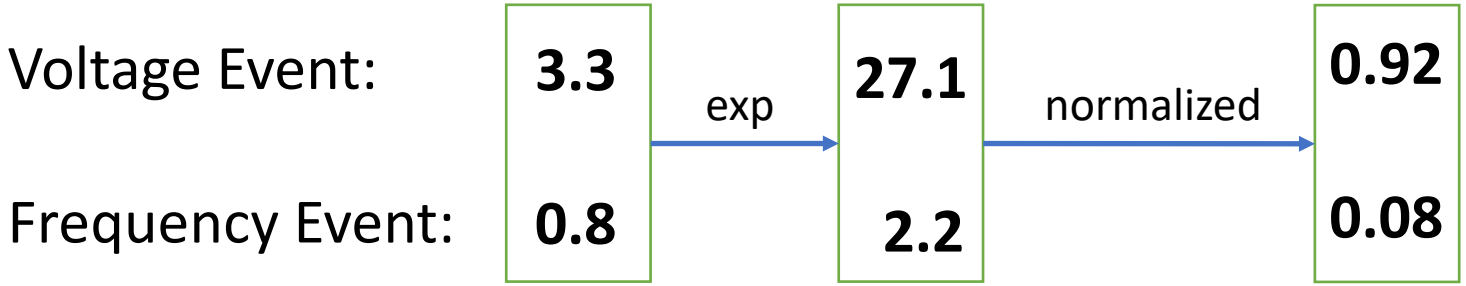


- Softmax is used to convert the raw classifier scores as probabilities:
- $s = f(X, \theta)$
- $P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$ (Softmax Function)

The scores of two classes

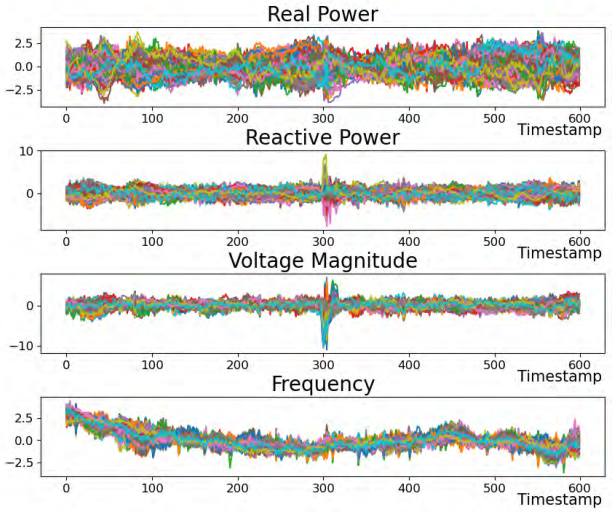
Probabilities must be >= 0

Probabilities must sum to 1



Cross Entropy Loss Function

A Power System Sample



- Cross Entropy is used to evaluate how well the estimated probabilities compare to the ground truth.
- $L = -\sum_c y_c \log(p_c)$
- c represent each classes.

Score from the classifier

Probabilities must be >= 0

Probabilities must sum to 1

Ground Truth (Correct Probs)

Voltage Event: **3.3**

Frequency Event: **0.8**

exp → **27.1**, **2.2**

normalized → **0.92**, **0.08**

Compare

$L = -\log(0.92)$

1, **0**

Parameter Training

- The CNN is trained by tuning its parameters:
  - Weights of connections $W^{(k)}$
  - Biases $\boldsymbol{b}^{(k)}$
- The parameters are tuned to minimize a loss function.
  - The loss function measures how far away the prediction is from the target (true) values.
  - Examples: cross-entropy, mean squared error, mean absolute error.
  - The choice of loss function depends on the problem.
- This tuning process is called “Training”.
- Training is performed on a training dataset.
  - The training dataset contains the samples of PMU measurement event samples and the corresponding event labels.

Minimize the Loss Function

- The loss function is minimized by iterative, gradient-based optimization algorithms (e.g. stochastic gradient descent)
- Suppose:
  - The CNN output is $\hat{\boldsymbol{y}}$, and the target value (true value) is $\boldsymbol{y}$. The loss function is $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$.
  - The $\hat{\boldsymbol{y}} = f(X, \boldsymbol{\theta})$, which represents the function of the CNN, $\boldsymbol{\theta}$ is the set of the parameters in CNN.
- The basic iterative, gradient-based framework of the optimization is as follows:

1: Initialize $\boldsymbol{\theta}$
2: Repeat until convergence:
3: Calculate $\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$ and $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$.
4: Calculate $\nabla_{\boldsymbol{\theta}} L$, i.e., the gradient of loss function value with respect to $\boldsymbol{\theta}$.
5: Update $\boldsymbol{\theta}$ based on $\nabla_{\boldsymbol{\theta}} L$ so that $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ is decreased.

Backpropagation

- In a CNN combined with multiple layers, the backpropagation is used to calculate the gradient of every parameter with respect to the loss: $\nabla_{\boldsymbol{\theta}} L$.
- Backpropagation is based on the "chain rule" of calculus.
- Suppose $f()$ and $g()$ are two differentiable functions.
- If $y = g(x)$, and $z = f\big(g(x)\big) = f(y)$.
  - The chain rule states that $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$.
- If $\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors, $\boldsymbol{y} = g(\boldsymbol{x})$ and $z = f(\boldsymbol{y})$,
  - The chain rule states that $\frac{\partial z}{\partial \boldsymbol{x}_i} = \sum_j \frac{\partial z}{\partial \boldsymbol{y}_j} \cdot \frac{\partial \boldsymbol{y}_j}{\partial \boldsymbol{x}_i}$

Back Propagation Algorithm

- Based on the chain rule, the backpropagation of an l-layer CNN is as follows.
- The gradient is calculated backward from layer l to layer 1. That's why it is called backpropagation.
- $\odot$ represents pointwise multiplication.

1: Perform a forward propagation to get the value of $\hat{\boldsymbol{y}}$ and the $\boldsymbol{a}^{(i)}$ and $\boldsymbol{h}^{(i)}$ of each layer $i = 1, \dots, l$.
2: Calculate the gradient $\boldsymbol{J} \leftarrow \nabla_{\boldsymbol{h}^{(l)}} L = \nabla_{\hat{\boldsymbol{y}}} L = \nabla_{\hat{\boldsymbol{y}}} L(\hat{\boldsymbol{y}}, \boldsymbol{y})$.
3: for layer $k = l, l-1, \dots, 1$ do
4: $\quad \boldsymbol{J} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} L = \boldsymbol{J} \odot g'(\boldsymbol{a}^{(k)})$
5: $\quad \nabla_{\boldsymbol{b}^{(k)}} L = \boldsymbol{J}$
6: $\quad \nabla_{W^{(k)}} L = \boldsymbol{J} \cdot \left(\boldsymbol{h}^{(k-1)} \right)^T$
7: $\quad \boldsymbol{J} \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} L = \left(W^{(k)} \right)^T \cdot \boldsymbol{J}$
8: end for

Outline

- Background and Challenges
- Overview of Convolution Neural Network (CNN) and Event Classification Problem
- Types of Power System Events
- Data Preprocessing
- Convolutional Neural Network
- Problem setup and dataset
- Assignment walk-through

Assignment walk-through

Python: Installation and Environment Setup

Walk-Through of Python Code

Python Installation

- Download the Anaconda from the following website: https://www.anaconda.com/download
- Download and install the Anaconda, an open-source package and environment management system for Python.
- After the installation, run “pip install PACKAGE_NAME” to install the packages.

Homework 1: Power System Event Classification with Convolutional Neural Network Using PMU Data

- In this homework, you will:
  - 1. Download the pmuBAGE data.
  - 2. Preprocessing pipeline to process and build the dataset.
  - 3. Build the convolutional neural network.
  - 4. Train the convolutional neural network on the built dataset.
  - 5. Evaluate the trained CNN model.

Load Needed Libraries/Packages

- To start, we need to load the libraries and packages that will used in the code.
- The random seed that controls the random process (e.g., random split of dataset) is also defined at the beginning.

In [1]:
```
# Load the tensorflow, which is a framework for deep learning.
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
# Load numpy library as "np", which can handle large matrices and provides some mathematical functions.
import numpy as np
# Load pandas as "pd", which is useful when working with data tables.
import pandas as pd
# Load random, which provide some randomize functions.
import random
# Load a function pyplot as "plt" to plot figures.
import matplotlib.pyplot as plt

# Setup the random seed for reproducibility
seed = 4281
random.seed(seed)
np.random.seed(seed)
tf.random.set_seed(seed)
```

Download and Load the Dataset

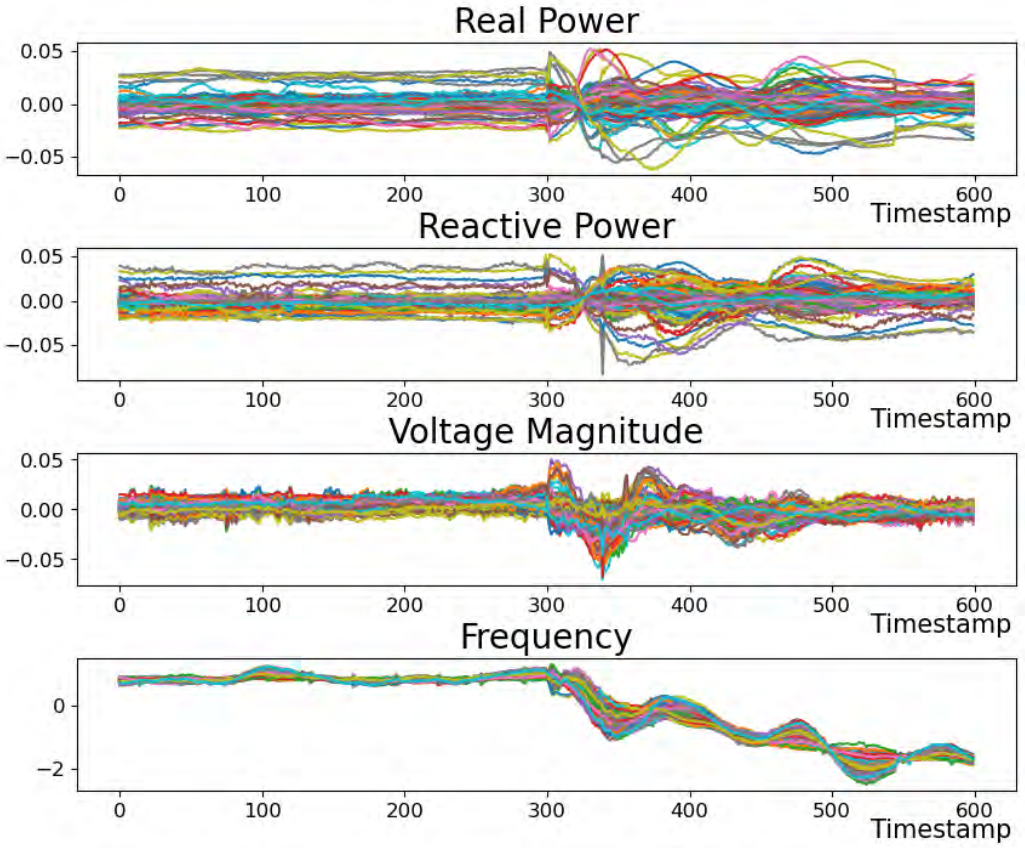
- We need to download the data from GitHub: https://github.com/NanpengYu/pmuBAGE. The downloaded dataset needs to be put in the same folder as the homework Jupyter Notebook.
- Then, we load the dataset which will be used in the code. The data has been prepared in multiple “npy” files, and each “npy” file contains several 20-second PMU measurements from 100 sensors.

```
# Load each tensors of voltage events and concatenate them as a big tensor.
voltage_tensor_list = []
for idx in range(voltage_tensor_number):
    voltage_sub_tensor = np.load(f"{pmuBAGE_data_dir}/voltage/voltage_{idx}.npy")
    voltage_tensor_list.append(voltage_sub_tensor)
voltage_tensor = np.concatenate(voltage_tensor_list, axis=0)


# Load each tensors of frequency events and concatenate them as a big tensor.
frequency_tensor_list = []
for idx in range(frequency_tensor_number):
    frequency_sub_tensor = np.load(f"{pmuBAGE_data_dir}/frequency/frequency_{idx}.npy")
    frequency_tensor_list.append(frequency_sub_tensor)
frequency_tensor = np.concatenate(frequency_tensor_list, axis=0)

# Transpose the big tensor as (event_idx, timestamp, PMU_idx, measurements)
voltage_tensor = np.transpose(voltage_tensor, (0, 3, 2, 1))
frequency_tensor = np.transpose(frequency_tensor, (0, 3, 2, 1))

# Print the shape of the voltage event
```

Visualization the pmu event data

- What does the data look like?
- Use the visualization function. Upon execution, we can visualize each data sample.

```
In [3]: def visualization(event, save_path='', isSave=False):
            plt.style.context(['ieee', 'no-latex'])

            fig, axes = plt.subplots(nrows=4, ncols=1, figsize=(10, 8))
            plt.subplots_adjust(hspace=0.55)
            plt.tick_params(labelsize=15)

            axes[0].set_title("Real Power", fontsize=20)
            axes[0].plot(event[:, :, 0])
            axes[0].tick_params(labelsize=12)
            axes[0].set_xlabel('Timestamp', fontsize=15)
            axes[0].xaxis.set_label_coords(0.92, -0.21)

            axes[1].set_title("Reactive Power", fontsize=20)
            axes[1].plot(event[:, :, 1])
            axes[1].tick_params(labelsize=12)
            axes[1].set_xlabel('Timestamp', fontsize=15)
            axes[1].xaxis.set_label_coords(0.92, -0.21)

            axes[2].set_title("Voltage Magnitude", fontsize=20)
            axes[2].plot(event[:, :, 2])
            axes[2].tick_params(labelsize=12)
            axes[2].set_xlabel('Timestamp', fontsize=15)
            axes[2].xaxis.set_label_coords(0.92, -0.21)

            axes[3].set_title("Frequency", fontsize=20)
            axes[3].plot(event[:, :, 3])
            axes[3].tick_params(labelsize=12)
            axes[3].set_xlabel('Timestamp', fontsize=15)
            axes[3].xaxis.set_label_coords(0.92, -0.21)

            if isSave == True:
                plt.savefig(save_path)
            plt.show()
            return 0
```



Preprocessing of the Dataset

- At this step, we need to:
  - Standardize each PMU time-series sequence.
  - Apply one-hot encoding to categorical features.
  - Combine multiple event types together.
  - Split the dataset to training, (validation), and testing.

Data Standardization

- Below is the detailed task of data standardization (as the input of the CNN).
- You need to complete the code by filling in the part of standardization on the PMU sequence.

```
##-------------------------------------------------------------------------##
##---------------------Students start filling below----------------------##
##-------------------------------------------------------------------------##

"""
    Use standardization to pre-process the pmu time series data.
    Input  -> two original tensors: voltage_tensor, frequency_tensor
    Output -> two standardized tensors: voltage_tensor_standardized, frequency_tensor_standardized
    Requirement Details:
        The tensor shape is (number_of_event, timestamps (600), pmus (100), measurements (4))
        For each time sequence (Single pmu measurement sequence, 600 timestamps), standardize them by Z-Score.
"""

# Voltage tensor

voltage_tensor_standardized = ...


# Frequency tensor

frequency_tensor_standardized = ...


##-------------------------------------------------------------------------##
##----------------------------End filling----------------------------------##
##-------------------------------------------------------------------------##

# Should be (620, 600, 100, 4)
print(voltage_tensor_standardized.shape)

# Should be (84, 600, 100, 4)
print(frequency_tensor_standardized.shape)
```

Apply One-Hot Encoding

- Below is the detailed task of one-hot encoding on the event labels.
- You need to complete the code by filling in the part of one-hot encoding for PMU event labels.

```
##------------------------------------------------------------------------##
##--------------------Students start filling below--------------------##
##------------------------------------------------------------------------##


"""
    Implement the one-hot encoding on the lablel of of the voltage and frequency event labels.
    Input  -> Original voltage and frequency labels (voltage_label, frequency_label)
    Output -> One-hot encoded voltage and frequency labels (voltage_label_onehot, frequency_label_onthot)
    Voltage label: "0" -> "[1, 0]"
    Frequency label: "1" -> "[0, 1]"
    You can use any library or tool for doing this.

"""

voltage_label_onehot = ...
frequency_label_onthot = ...

##------------------------------------------------------------------------##
##----------------------------End filling---------------------------##
##------------------------------------------------------------------------##

# Should be [1, 0]
print(voltage_label_onehot[0])
# Should be [0, 1]
print(frequency_label_onthot[0])
# Should be (620, 2)
print(voltage_label_onehot.shape)
# Should be (84, 2)
print(frequency_label_onthot.shape)
```

Prepare Input and Output Data

- Below is the detailed task of the dataset separation.
- You need to complete the code by filling in the part of the dataset separation on the PMU data for training and testing.

```
##-------------------------------------------------------------------##
##---------------------Students start filling below------------------##
##-------------------------------------------------------------------##

"""
    Seperate the samples and labels to train and test datasets.
    70% of the voltage and frequency samples and labels are combined as training dataset
    30% remainings are combined as testing dataset
    Input  -> X_voltage, X_frequency, y_voltage, y_frequency
    Output -> X_train, y_train, X_test, y_test
        X_train contains 70% of the X_voltage and X_frequency
        y_train contains 70% of the y_voltage and y_frequency
        X_test contains 30% of the X_voltage and X_frequency
        y_test contains 30% of the y_voltage and y_frequency
"""

# X_train
X_train = ...
# y_train
y_train = ...
# X_test
X_test = ...
# y_test
y_test = ...

##-------------------------------------------------------------------##
##-----------------------------End filling---------------------------##
##-------------------------------------------------------------------##

# Should be (492, 600, 100, 4)
print(X_train.shape)
# Should be (492, 2)
print(y_train.shape)
# Should be (212, 600, 100, 4)
print(X_test.shape)
# Should be (212, 2)
print(y_test.shape)
```

Build the CNN Model

- Below is an example of an unfinished CNN model.
- Following this example, you should add more layers and complete the CNN model.

In [ ]:

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(600, 100, 4)))

##------------------------------------------------------------------------------##
##--------------------Students start filling below----------------------##
##------------------------------------------------------------------------------##

"""
    Add more laybers in the model, at least three convolusional layers.
    Then add the Flatten and Dense layers to make the output same with the number of classes.
"""
model.add(...)

"""
    Define the Loss function
"""
loss_func = ...

"""
    Define the learning rate and optimizer
"""
lr = ...
optimizer = ...

##------------------------------------------------------------------------------##
##-----------------------------End filling------------------------------##
##------------------------------------------------------------------------------##

# Compile the neural network model
model.compile(optimizer=optimizer, loss=loss_func, metrics=['categorical_accuracy'])
```

Train the CNN Model and Evaluate the Performance

- Thanks to the Keras, the training and evaluation only need one-line code. We provide the code for this step.
- After finishing the previous tasks, you can evaluate your CNN's performance in the end.

6. Train the neural network

```
In [ ]: model.fit(X_train, y_train, epochs=20, batch_size=32)
```

7. Evaluate the Neural Network

```
In [ ]: loss, accuracy = model.evaluate(X_test, y_test)
        print(f"The accuracy of the neural network on the test dataset is: {accuracy}.")
```