ECEN 6123

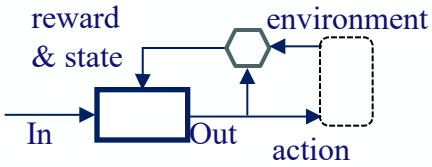
AI for Engineering Systems: Grid-Oriented Applications

Lecture 2: Coding Examples and Environment

Ying Zhang, Ph.D. in Electrical Engineering
Assistant Professor
Department of Electrical and Computer Engineering

Markov Decision Process

Reinforcement Learning

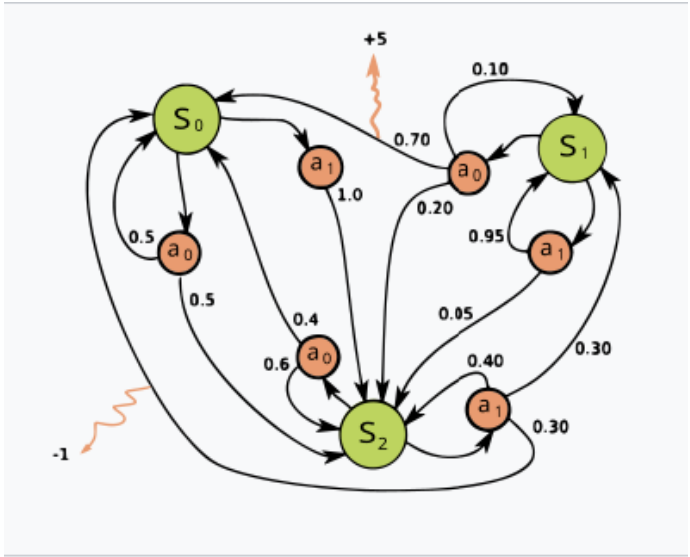


Application

- ✓ DeepMind's AlphaGo
- ✓ AlphaZero
- ✓ AlphaStar
- ✓ Fire-extinguish robots

Common Algorithms

- ○ Monte Carlo
- ○ Q-Learning
- ○ SARSA
- ○ **Deep Q Network (DQN)**
- ○ **Asynchronous Actor-Critic Agent (A3C)**
- ○ **Deep Deterministic Policy Gradient (DDPG)**



Example of a simple MDP with three states (green circles) and two actions (orange circles), with two rewards (orange arrows)

A Markov decision process is a 4-tuple (S, A, P_a, R_a)

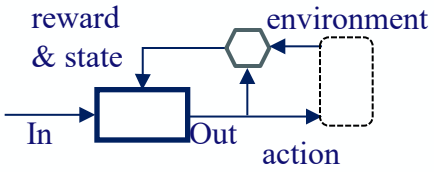
State, Action, Transition Probability, Reward

The goal in a Markov decision process is to find a good "policy" for the decision maker:

A function π that specifies the action $\pi(s)$ that the decision maker will choose when in state s.

Reinforcement Learning

Reinforcement Learning



Application

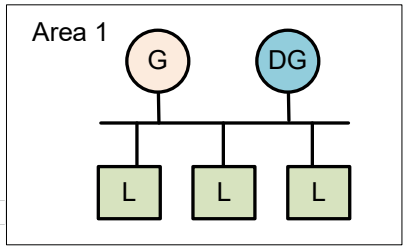
- ✓ DeepMind's AlphaGo
- ✓ AlphaZero
- ✓ AlphaStar
- ✓ Fire-extinguish robots

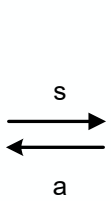
Common Algorithms

- ○ Dynamic programming
- ○ Monte Carlo
- ○ Q-Learning
- ○ SARSA
- ○ **Deep Q Network (DQN)**
- ○ **Asynchronous Actor-Critic Agent (A3C)**
- ○ **Deep Deterministic Policy Gradient (DDPG)**

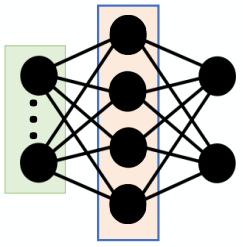
- RL uses machine learning to estimate the policy, which is a mapping between the state S and the action A.
- DRL uses deep NN, a combination of reinforcement learning and deep learning.
- The objective function is to maximize the accumulated reward of conduction this series of actions in a **time-series decision problem.**

DG: Renewable Generation
G: Generator
L: Controllable Load



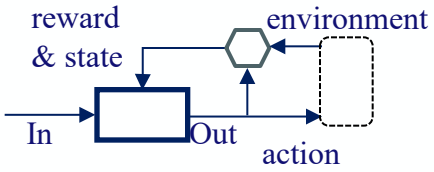


NN-based Agent as Decision-Maker



Reinforcement Learning

Reinforcement Learning



Application

- ✓ DeepMind's AlphaGo
- ✓ AlphaZero
- ✓ AlphaStar
- ✓ Fire-extinguish robots

Common Algorithms

- ○ Dynamic programming
- ○ Monte Carlo
- ○ Q-Learning
- ○ SARSA
- ○ **Deep Q Network (DQN)**
- ○ **Asynchronous Actor-Critic Agent (A3C)**
- ○ **Deep Deterministic Policy Gradient (DDPG)**

- RL defines a reward function based on "memory/experience", not longer loss function in NN) to reinforce this objective.

- **Experience**

Think about how human get experience.

Observation→action→New Observation→Consequence
All these four items form one piece of experience
How about OPF problem?

- RL borrows this process of forming experience by storing (s, a, s', r) tuple (**create its own dataset**).

- **Compared with conventional optimization (OR)**

RL is model-free optimization leveraging experiences

- **Compared with supervised learning**
- RL's optimization objective is not loss function between labeled and estimated input
- it dynamically evolved according the experience tuple, which is updated during the learning process. (**Adaptivity**)

Semi-supervised Learning

Also known as “partially supervised learning”

Semi-supervised Learning

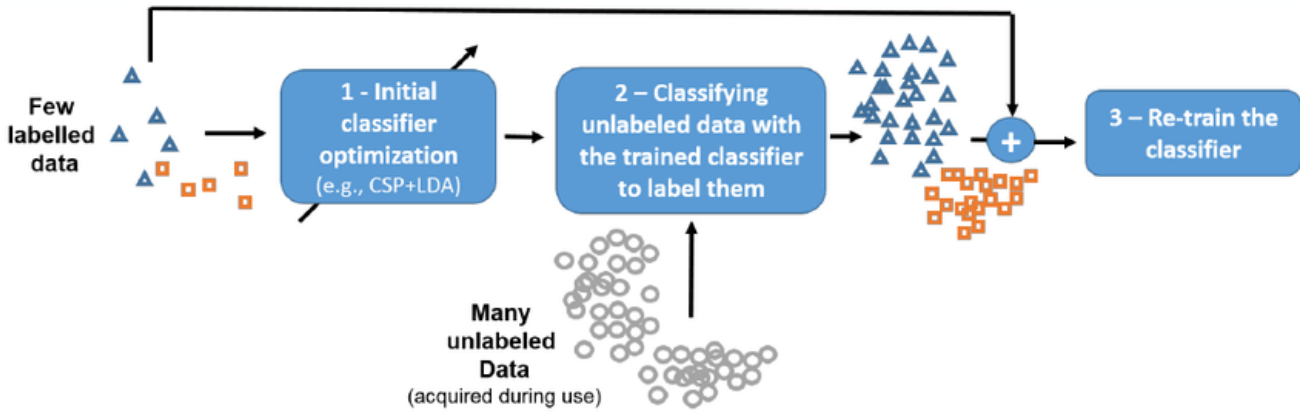
many unlabeled &
few labeled data

Application

- ✓ Google Photos
- ✓ Webpage classification

Common Algorithms

- ○ Combination of unsupervised and supervised learning

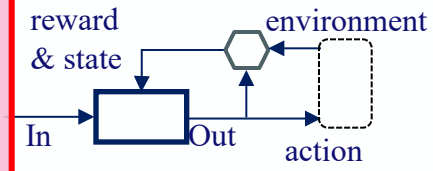


Source: https://www.researchgate.net/figure/Principle-of-semi-supervised-learning-1-a-model-eg-CSP-LDA-classifier-is-first_fig4_277605013

Deep Learning

Deep Learning is part of the machine learning family based on artificial neural network with many layers. Deep learning can be supervised, unsupervised and semi-supervised.

Reinforcement Learning



Application

- ✓ DeepMind's AlphaGo
- ✓ AlphaZero
- ✓ AlphaStar
- ✓ Fire-extinguish robots

Common Algorithms

- Monte Carlo
- Q-Learning
- SARSA
- Deep Q Network (DQN)
- Asynchronous Actor-Critic Agent (A3C)
- Deep Deterministic Policy Gradient (DDPG)

(Deep) Neural Network (DNN)

- **Universal approximation theorem[1]**

❑ A deep neural network typically consists of

- Linear transformations

$$h^{(k+1)} = Wh^{(k)}$$

- Nonlinear activation functions

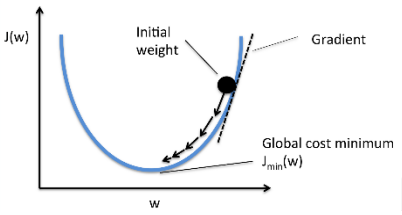
$$h^{(k+2)} = f(h^{(k+1)})$$

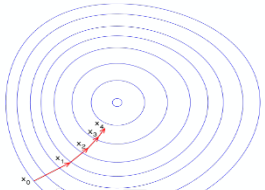
- A loss function

$$J(w,b) = \frac{1}{2}\left\|h_{W,b}(x) - y\right\|^2 + \text{regularization}$$

❑ A deep neural network can be trained using stochastic gradient descent

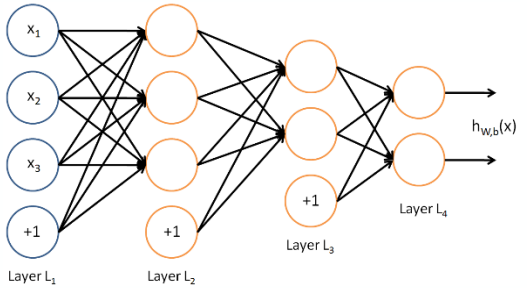
$$w_{i+1} = w_i - \frac{\partial J}{\partial w} \cdot \Delta$$





Source: wiki

Source: https://neurotracker.net/2017/12/28/7-major-developments-neuroscience-2017/



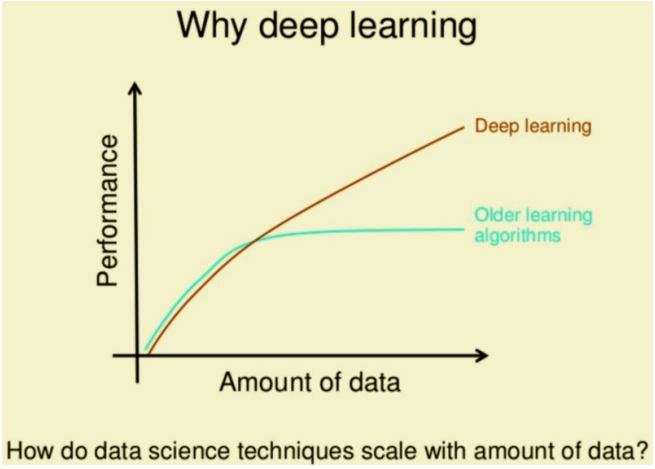
$$a_1^{(2)} = f\left(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}\right)$$
$$a_2^{(2)} = f\left(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}\right)$$
$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$
$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$

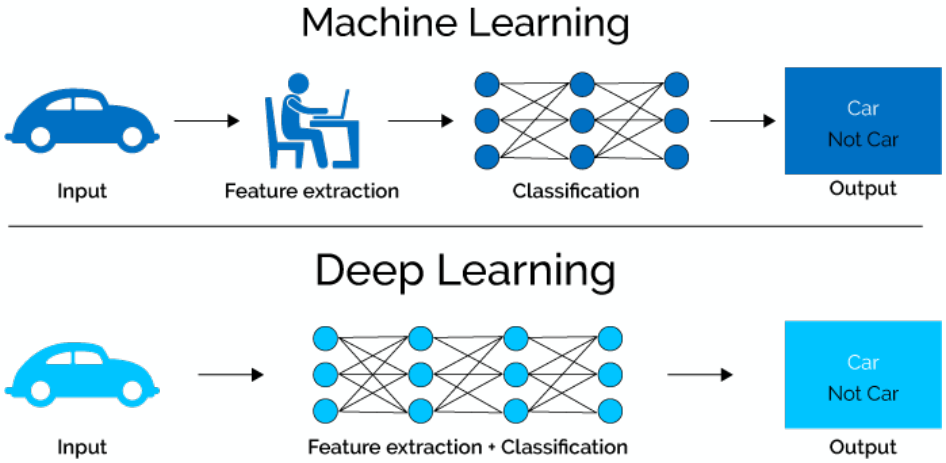
Neural network

[1] https://www.sciencedirect.com/science/article/abs/pii/089360809190009T

Deep Learning in a Nutshell

- Deep learning is a general-purpose framework for representation learning
  - Given an objective
  - Learn representation that is required to achieve objective
  - Directly from raw inputs
  - Using minimal domain knowledge
  - Represent the world using nested hierarchy of concepts (each using simpler ones)

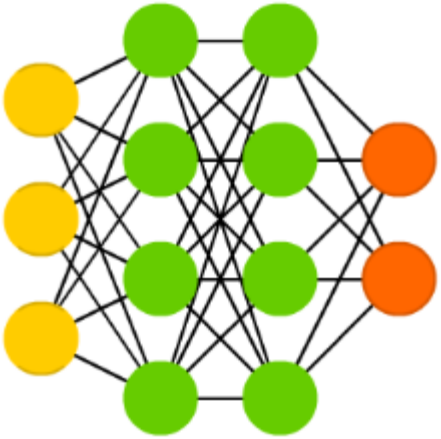




Source: https://towardsdatascience.com

Types of DNNs

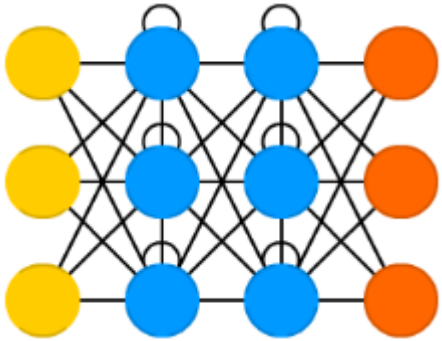
Artificial Neural Network (ANN)



Application

- ✓ **Tabular data**
- ✓ Image data
- ✓ Text data

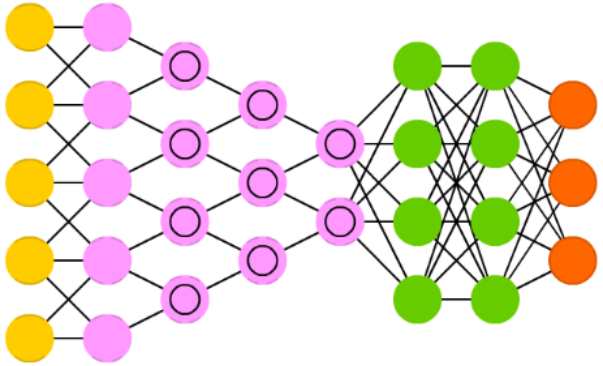
Recurrent Neural Network (RNN)



Application

- ✓ **Time Series data**
- ✓ Text data
- ✓ Audio data

Convolution Neural Network (CNN)



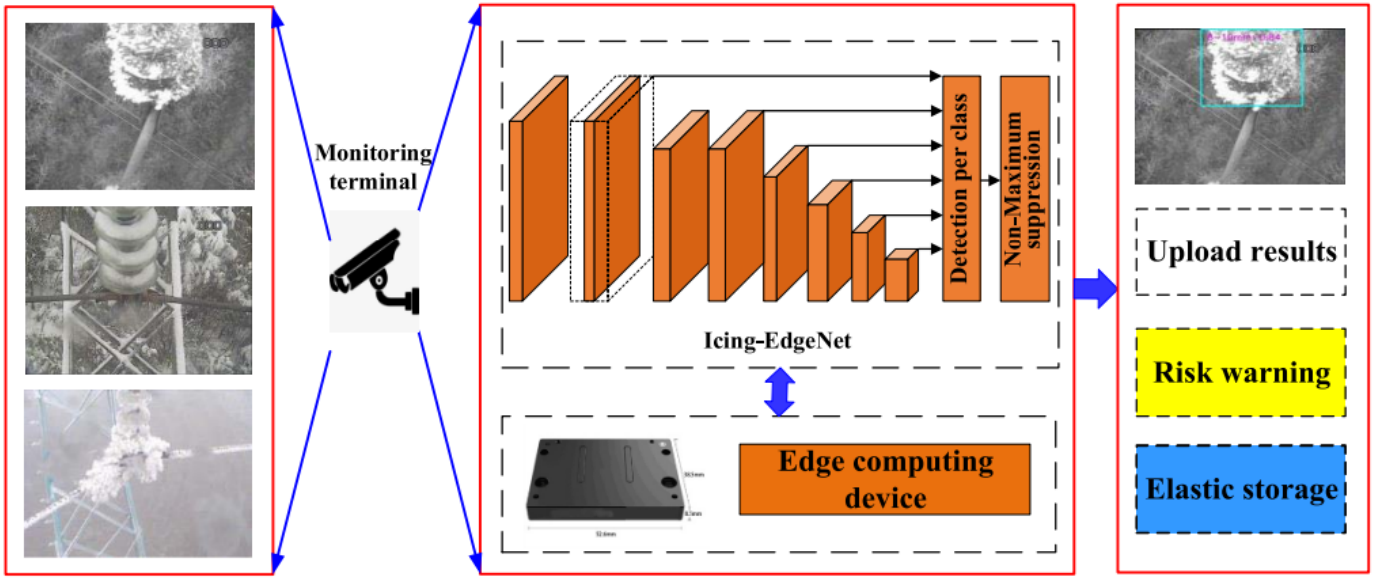
Application

- ✓ **Image data**

Pics from: https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

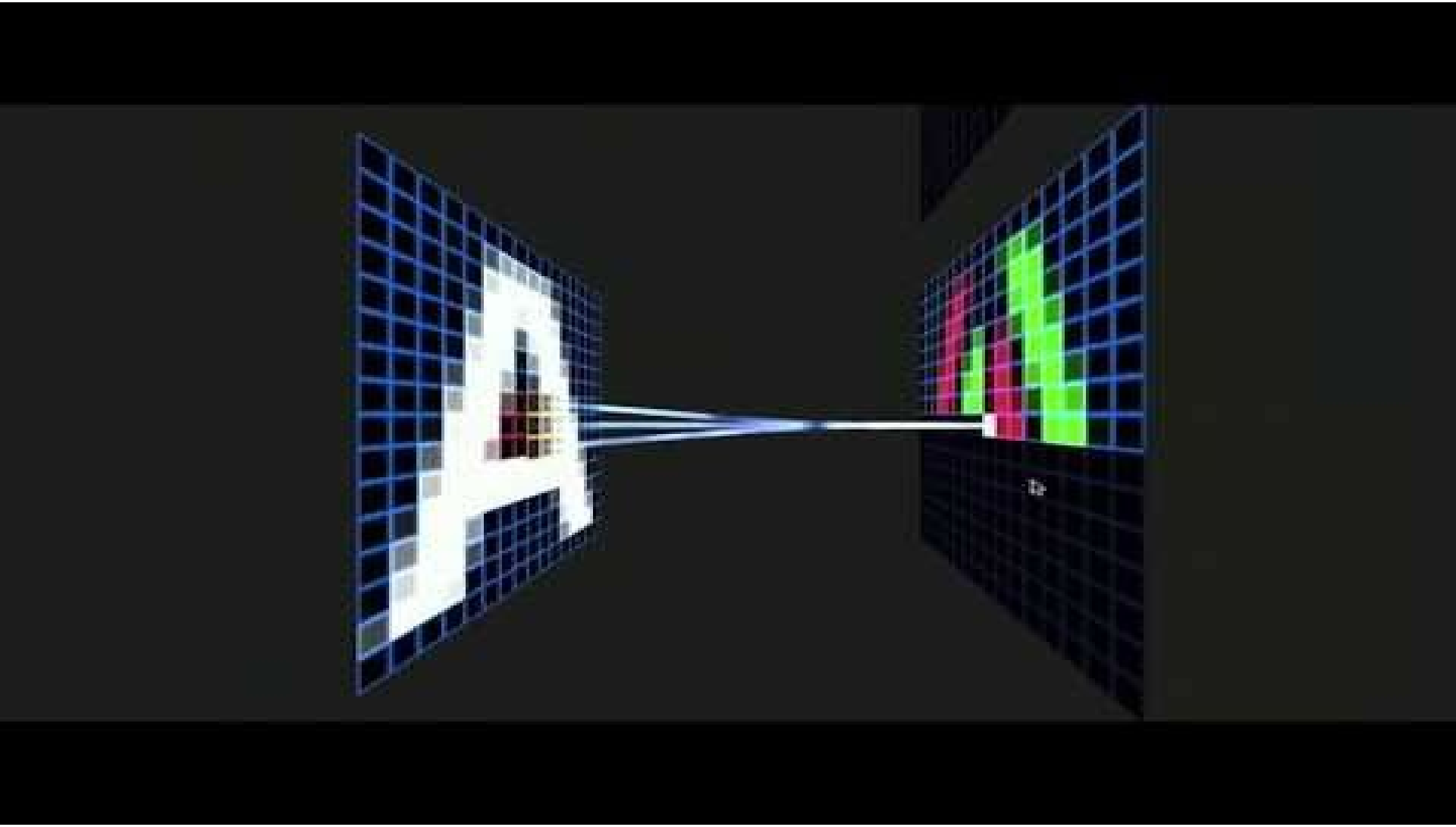
CNN Applications in Power Systems

- CNN-based Icing monitoring of transmission lines
- Measurement methods based on image recognition is gradually applied for icing monitoring but fails to distinguish the icing thickness
- The novelty is lightweight algorithm with edge intelligence, considers the camera of the monitoring terminal is covered with ice, which results in blurred images.



- "Icing-EdgeNet: A Pruning Lightweight Edge Intelligent Method of Discriminative Driving Channel for Ice Thickness of Transmission Lines," in IEEE Transactions on Instrumentation and Measurement, vol. 70, pp. 1-12, 2021, Art no. 2501412,

An Illustrative Example: How CNN works?



Video link: https://www.youtube.com/watch?v=f0t-OCG79-U&t=3s

General Procedure of Training a DNN

- We will use Tensorflow as an example. The other package is similar.
- 1. Install the library/packages
- 2. Prepare the data
- 3. Define the DNN model
- 4. Compile the model
- 5. Train the model
- 6. Evaluate and tune
- 7. Save and deploy

How to Define and Train a NN?

1. Step 1: install tensorflow

```
pip install tensorflow
```

2. Step 2: Prepare the data

For our example, let's use a simple dataset like MNIST, which consists of hand-written digits. TensorFlow provides easy access to this dataset.

```
import tensorflow as tf

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0  # Normalize data
```

- 3. Step 3. Define the model

  We will create a simple DNN with a few layers. Tensorflow makes this easy with the Keras API.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10) ])
```

- 4. Step 4. Compile the model

  Next, we compile the model with an optimizer, a loss function, and a metric

```
loss_fn =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])
```

- 5. Train the model

  Now, we train the model with our training data.

```
model.fit(x_train, y_train, epochs=5)
```

- 6. Evaluate and Tune

  Evaluate the model's performance on the test dataset. If the performance is unsatisfactory, you may consider tuning the model architecture or hyperparameters.

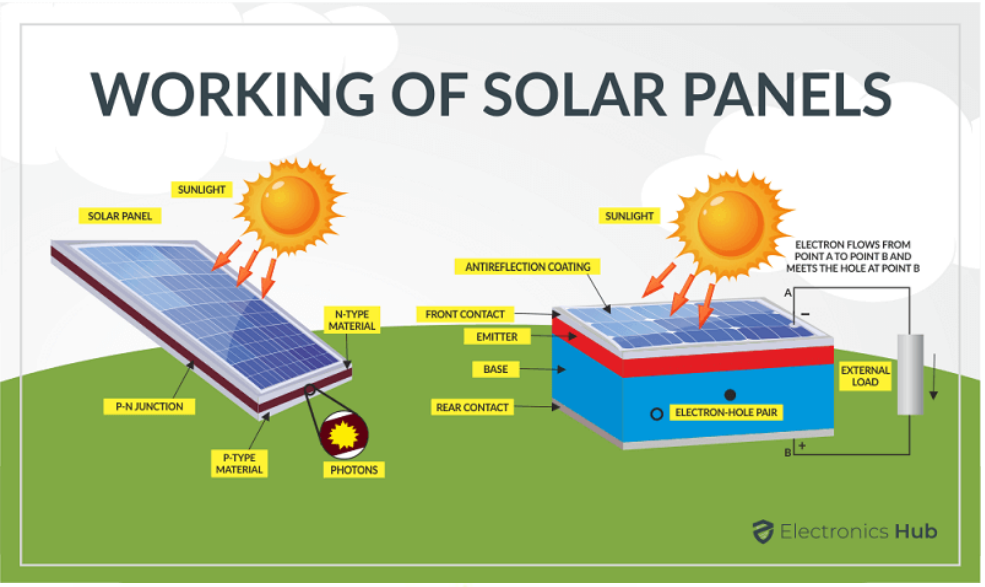
```
model.evaluate(x_test,  y_test, verbose=2)
```

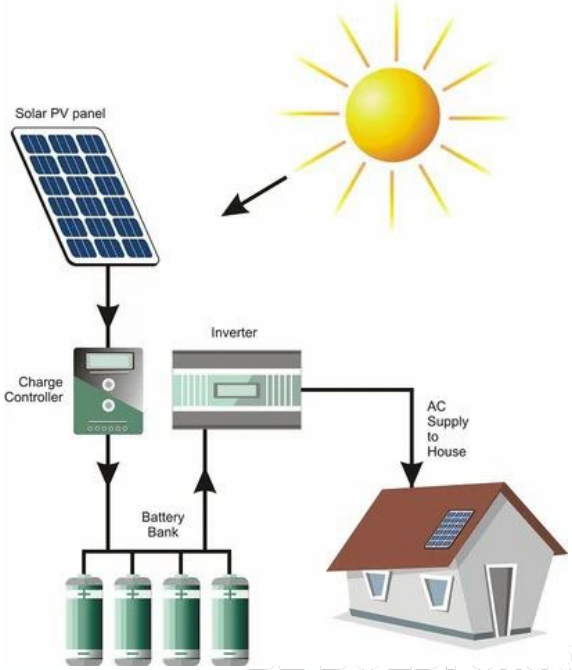
- 7. Save and deploy

```
model.save('my_model.h5')
```

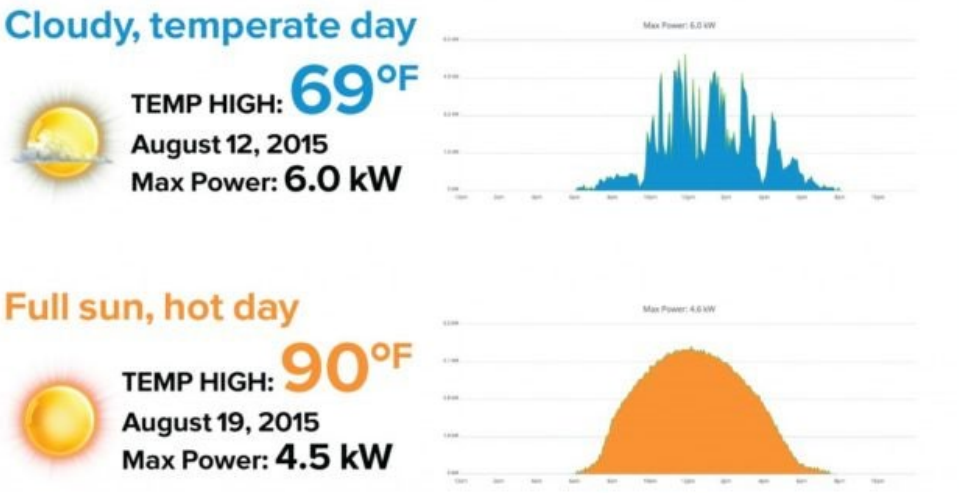
This process will give you a DNN trained on the MNIST dataset. You can then use this model to predict new, unseen data. Remember, this is a basic example. Real-world applications often require more complex data preprocessing, model architectures, and tuning steps.

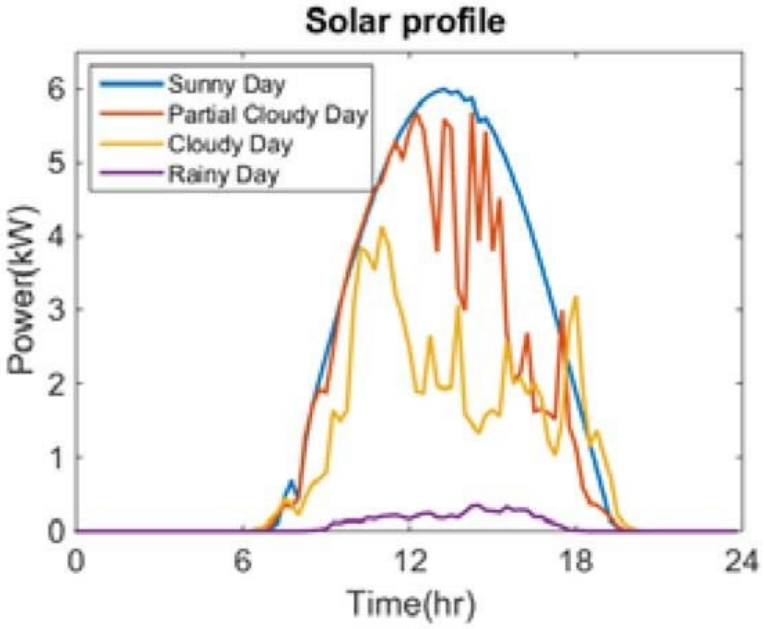
Use Neural Network to Predict Solar Power Generation

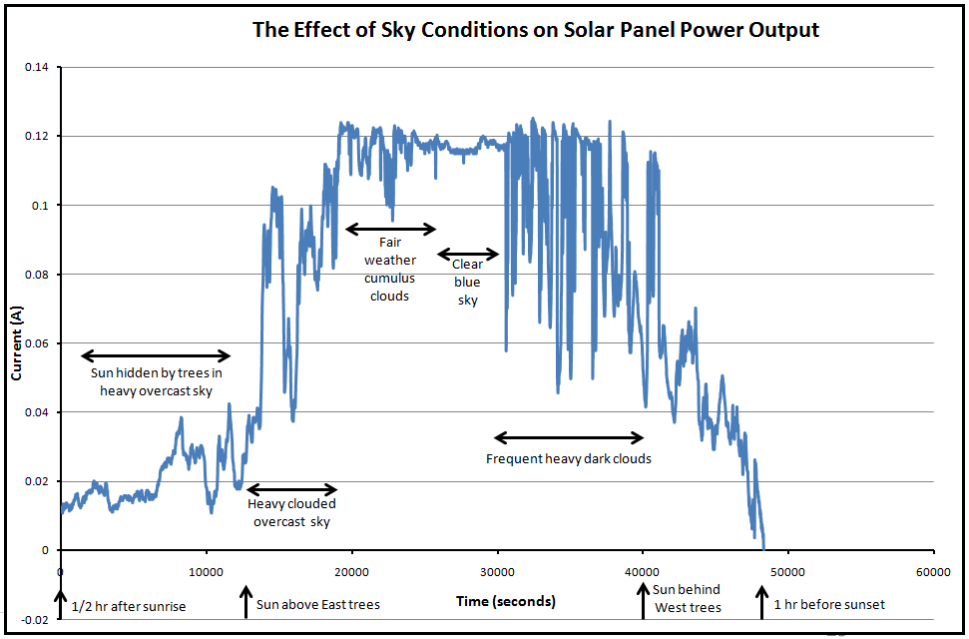




What Affect the Solar Power Output?







Use Neural Network to Predict Solar Power Generation

Data Collection:

Collect historical solar power output data. The more data, the better the model will potentially be.
It would be beneficial if you could also have data about environmental conditions (e.g., cloud cover, temperature, solar irradiance, etc.).

Data Preprocessing:

Check for missing values and handle them (either by imputation or removal).
Normalize or standardize the data to aid the training of the neural network.
Create a time series sequence data for training. For instance, if you want to predict the output of the next hour, you might want to use the past n hours as input.

- **Modeling:**

Design a deep neural network. For time-series data, recurrent layers (like LSTM or GRU) can be beneficial.
Split the data into training, validation, and testing sets.
Train the model using the training set and validate it using the validation set.

- **Evaluation:**

Evaluate the model's performance on the test set.
Use metrics like MAE (Mean Absolute Error) or RMSE (Root Mean Squared Error) for evaluation.

- **Prediction:**

Use the trained model to predict future solar power output.

Data Preprocessing in Learning Models

Check for missing values and handle them (either by imputation or removal).
Normalize or standardize the data to aid the training of the neural network.
Create a time series sequence data for training. For instance, if you want to predict the output of the next hour, you might want to use the past n hours as input.

Data Normalization
The process of adjusting values measured on different scales to a common scale.
Prevents features with larger scales from dominating how models are trained.
Improve model accuracy and convergence speed.

Data Normalization in Machine Learning

- **Why normalize data?**
  - Variables that are measured at different scales do not contribute equally to the model fitting & model learned function and might end up creating a bias.
  - Ensures equal contribution of each feature.
  - Reduces complexity in optimization, especially in gradient descent.
  - Helps avoid issues like exploding or vanishing gradients.
  - Often required in models like neural networks and k-means clustering.
- **Common methods**
  - Min-Max scaling
  - Standardization (Z-score normalization)
  - Robust scaling
  - Log scaling
  - L2 normalization

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
```

- `# Load data`

```
df = pd.read_csv('solar_power_data.csv')
data = df['power_output'].values.reshape(-1, 1)
```

- `# Normalize data`

```
scaler = MinMaxScaler()
data_normalized = scaler.fit_transform(data)
```

- # Create sequences

```
X, y = [], []
for i in range(24, len(data_normalized)):  # using past 24 hours to predict next hour
    X.append(data_normalized[i-24:i, 0])
    y.append(data_normalized[i, 0])
X, y = np.array(X), np.array(y)
```

- # Split data into training and testing sets

```
train_size = int(0.8 * len(X))
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

- # Reshape data for LSTM layer

```
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

- # Build the LSTM model

```
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(LSTM(units=50))
model.add(Dense(units=1))

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=50, batch_size=32)
```

- # Predict

```
y_pred = model.predict(X_test)
y_pred_original = scaler.inverse_transform(y_pred)
y_test_original = scaler.inverse_transform(y_test.reshape(-1, 1))
```

- # Evaluate

```
mae = mean_absolute_error(y_test_original, y_pred_original)
print(f"Mean Absolute Error: {mae:.2f}")
```

In-Class Reading
(30 min)

Data Normalization in Machine Learning

- **The process of adjusting values measured on different scales to a common scale.**
  - Prevents features with larger scales from dominating how models are trained.
  - Improve model accuracy and convergence speed.
- **Why normalize data?**
  - Variables that are measured at different scales do not contribute equally to the model fitting & model learned function and might end up creating a bias.
  - Ensures equal contribution of each feature.
  - Reduces complexity in optimization, especially in gradient descent.
  - Helps avoid issues like exploding or vanishing gradients.
  - Often required in models like neural networks and k-means clustering.
- **Common methods**
  - Min-Max scaling
  - Standardization (Z-score normalization)
  - Robust scaling
  - Log scaling
  - L2 normalization

Data Normalization - Min-Max Scaling

- Min-Max Scaling is a scaling technique that linearly transforms the range of data values so that the minimum and maximum values of the transformed data are a specific minimum and maximum value, commonly 0 and 1.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

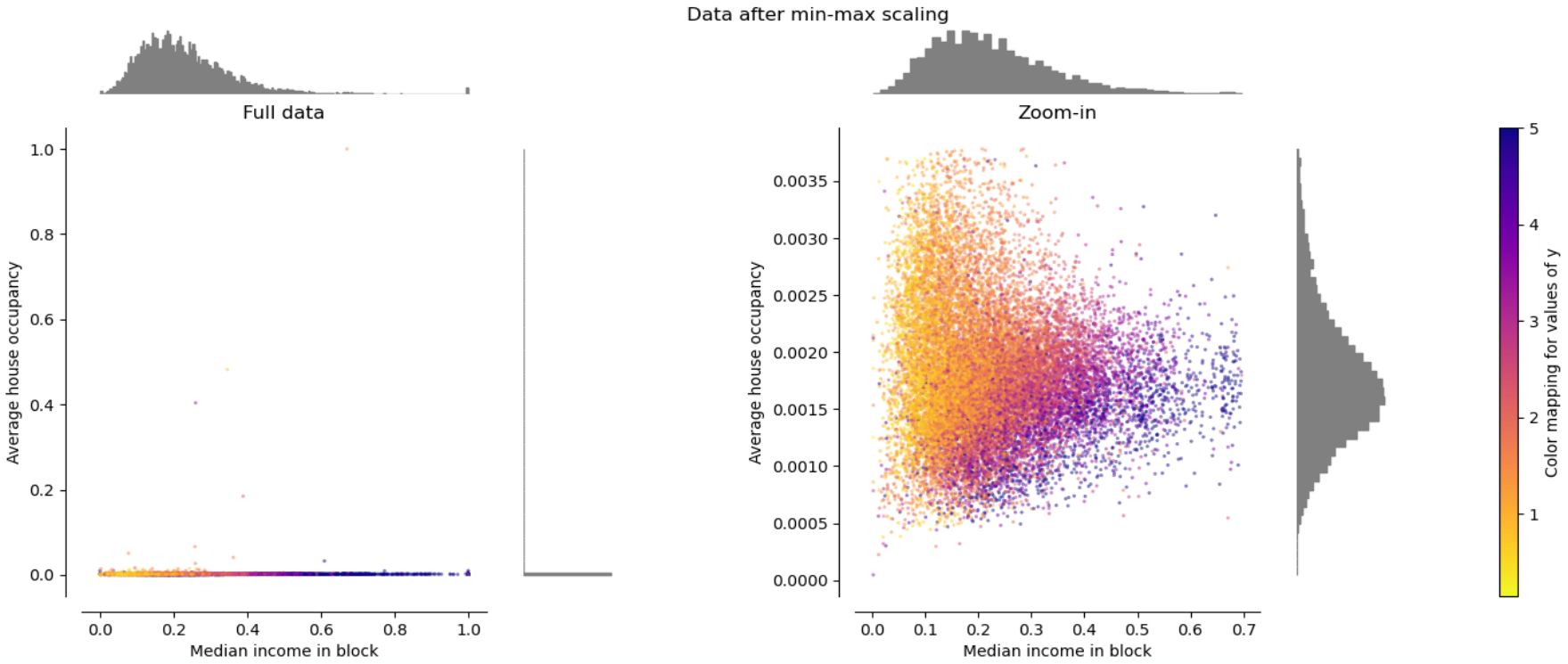
- Why use it?
  - **Uniform scale**: useful when you want all of your variable to be on the same scale
  - **Preserve relationships**: maintain the relationships between variables (as it's a linear transformation
  - **Algorithm sensitivity**: essential for algorithms that are sensitive
- Considerations and limitations
  - **Outliers**: Min-Max Scaling is sensitive to outliers. If outliers are present, they can compress the majority of the data into a very narrow range.
  - **Not Centered Data**: This technique does not center the data around zero, and it doesn't handle skewed distributions well.
  - **Feature Distribution**: It doesn't change the distribution of the feature; it merely rescales the range.

Data Normalization - Min-Max Scaling 2

- When to Use Min-Max Scaling?
  - When you need values in a bounded interval.
  - In neural network models that need input data within the range of the activation function.
  - When the data doesn't contain significant outliers.

```
from sklearn.preprocessing import MinMaxScaler
import numpy as np

# Sample data
data = np.array([[100, 0.001],
                 [8, 0.05],
                 [50, 0.005],
                 [88, 0.07],
                 [4, 0.1]])

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the data
scaled_data = scaler.fit_transform(data)

print(scaled_data)
```

Data Normalization - Min-Max Scaling 3

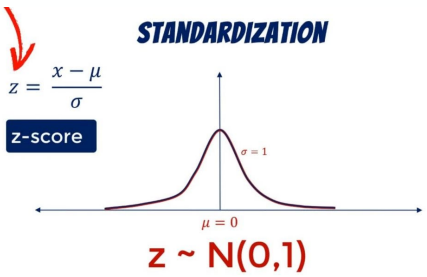


Source: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html

Data Normalization - Standardization

- Standardization is a scaling technique where the values are centered around the mean with a unit standard deviation. This means after standardization, the mean of the transformed features is zero, and the standard deviation is one.

$$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation } (x)}$$



- Why use it?
  - **Handling different scales**: Particularly useful when your data features have different units or scales.
  - **Centers data**: It centers the data around zero, which can be important for models that assume zero-centricity.
  - **Normalization of distribution**: While it doesn't make data normally distributed, it makes it easier to handle for models that assume normally distributed features.
  - **Algorithm performance**: Essential for many ML algorithms like support vector machines, logistic regression, and other algorithms that use gradient descent as an optimization technique.

Data Normalization - Standardization 2

- Considerations and limitations
  - Doesn't bound values: Unlike Min-Max scaling, standardization doesn't bound values to a specific range, which might be a problem for some algorithms.
  - Outliers impact: While standardization is less sensitive to outliers than Min-Max scaling, extreme outliers can still skew the results.
  - Data distribution: If the data is not normally distributed, this might not be the best scaler to use.

```
from sklearn.preprocessing import StandardScaler
import numpy as np

# Sample data
data = np.array([[1, -1, 2],
                 [2, 0, 0],
                 [0, 1, -1]])

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit and transform the data
scaled_data = scaler.fit_transform(data)

print(scaled_data)
```

Data Normalization - Standardization 3

- When to Use Standardization Scaling?
  - In algorithms that assume features are normally distributed.
  - When your model requires feature scaling but not necessarily within a bounded range.
  - In cases where outliers are not predominantly skewing the data.

Standardization is especially beneficial in scenarios where the features of your dataset vary significantly in terms of units or scales. This method improves the behavior of many machine learning algorithms and can significantly impact the performance of your models.

Source: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html

Data Normalization - Robust Scaling

- Robust Scaling is a method that scales features using statistics that are robust to outliers. It involves scaling data according to the Interquartile Range (IQR), which is the range between the 1st quartile (25th percentile, Q_1) and the 3rd quartile (75th percentile, Q_3).

$$x_{\text{robust}} \leftarrow \frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)}$$

- Why use it?
  - **Outlier impact:** The primary advantage of Robust Scaling is its resilience to outliers, as it uses the median and IQR.
  - **Preserving outliers:** Unlike other methods that might try to "normalize" outliers, Robust Scaling keeps these values distinct, which can be important in datasets where outliers are significant.
  - **Range consistency:** It brings all features to the same scale but does not distort the differences in the range of values like Min-Max scaling.
- Considerations and limitations
  - **Not zero-centered:** The scaled data is not centered around zero and the scaling does not ensure a unit variance.
  - **Dataset specific:** It's most useful in datasets where outliers are expected or known to exist.
  - **Not always appropriate:** In cases where outliers are actually data errors, or in algorithms sensitive to the absolute size of the feature, Robust Scaling might not be the best approach.

Data Normalization - Robust Scaling 2

- When to use robust scaling?
  - In datasets where outliers are not only present but are also a meaningful part of the data.
  - When you want to reduce the effects of outliers, without ignoring them or treating them as errors.
  - In algorithms that are sensitive to the scale or distribution of the data but robust to outliers, like SVM or k-means clustering.

Robust Scaling is a powerful tool in your preprocessing toolkit, especially in real-world datasets where outliers can significantly impact the performance of your models.

Data Normalization - Robust Scaling 3

```
from sklearn.preprocessing import RobustScaler
import numpy as np

# Sample data
data = np.array([[1, 2, 3],
                 [2, 3, 4],
                 [3, 4, 5],
                 [4, 10, 6],
                 [10, 6, 7]])

# Initialize the RobustScaler
scaler = RobustScaler()

# Fit and transform the data
scaled_data = scaler.fit_transform(data)

print(scaled_data)
```

Source: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html

Virtual Environment

- Using a virtual environment in Python programming is a widely recommended practice, particularly for managing dependencies and ensuring project isolation.
- Isolation of projects
  - **Separate Dependencies**: Each virtual environment has its own set of installed Python packages, allowing different projects to have different dependencies without interference.
  - **Avoid Conflicts**: Different projects may require different versions of the same package. Virtual environments ensure that there are no conflicts between these requirements.
- Reproducibility
  - **Consistent Environments:** By using a virtual environment, you can ensure that all project dependencies are consistent across different development and production environments.
  - **Facilitates Collaboration:** When multiple people are working on a project, virtual environments make it easy to share and replicate the setup.
- Testing and Stability
  - **Safe Testing Ground:** Virtual environments provide a sandbox to safely install and test new packages or updates without affecting the global Python installation.
  - **Experimentation:** They allow for experimenting with different package versions and configurations without risking the stability of other projects.

Virtual Environment 2

- Simplified Dependency Management
  - **Requirements File:** A virtual environment can have its own requirements.txt file that lists all the necessary packages, making it easy to recreate the environment elsewhere.
  - **Clean Project Space:** Keeps your project space clean and manageable by avoiding global installation of packages that are only needed for specific projects.
- Easy Cleanup
  - **Remove Entire Environment:** If you no longer need a project and its dependencies, you can simply delete its virtual environment without affecting others.
- Consistent Across Different Systems
  - **Platform Independence:** Virtual environments work similarly across different operating systems, making it easier to work on cross-platform projects

Virtual Environment 3

- 1. Installation: (only need to do it once for your computer)
  Install the virtual environment package if not already installed:
  pip install virtualenv
- 2. Creating a Virtual Environment: (only need to do it once for one application)
  Create a new environment:
  python -m venv myenv
  myenv is the name of the virtual environment and can be anything you choose.
- 3. Activating the Virtual Environment: (do it everytime you run your app)
  On Windows: myenv\Scripts\activate
  On Unix or MacOS: source myenv/bin/activate
- 4. Deactivating the Virtual Environment:
  Simply run: deactivate
- 5. Managing Packages: (needed when you need to add/remove packages from an existing virtual environment)
  Install packages as usual with pip: pip install package-name
  These installations will be local to the virtual environment.

Virtual Environment 4 - installing multiple packages

Assume you have the following requirement.txt file

```
flask==1.1.2
requests==2.25.1
numpy==1.19.5
pandas==1.2.1
```

Create a virtual environment

```
python -m venv myenv
```

Activate the Virtual Environment:

-On windows

```
myenv\Scripts\activate
```

-On linux

```
source myenv/bin/activate
```

Install the package

```
pip install -r requirements.txt
```

After installation, verify it.

```
pip freeze
```

1. Updating requirements.txt: If you install new packages or update existing ones and want to update your requirements.txt, you can use pip freeze > requirements.txt to overwrite it with the current list of packages.
2. Version Control: It's a good practice to include requirements.txt in your version control system (like Git) to keep track of dependencies.
3. By using a requirements.txt file, you ensure that anyone else working on the project, or even you in a different environment, can set up an identical environment with the same dependencies, which is crucial for consistency and reproducibility in development.

After-class Reading Material

- https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#sphx-glr-auto-examples-preprocessing-plot-all-scaling-py

- https://keras.io/api/

- https://docs.python.org/3/library/venv.html