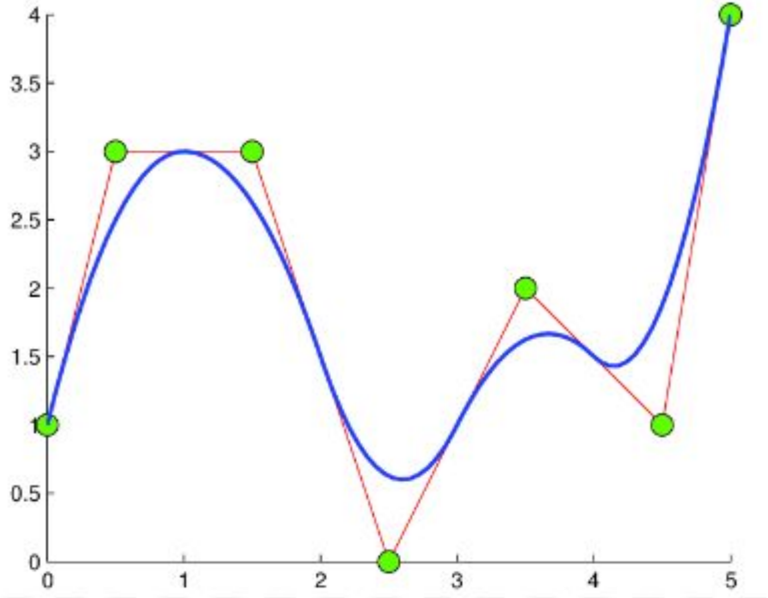
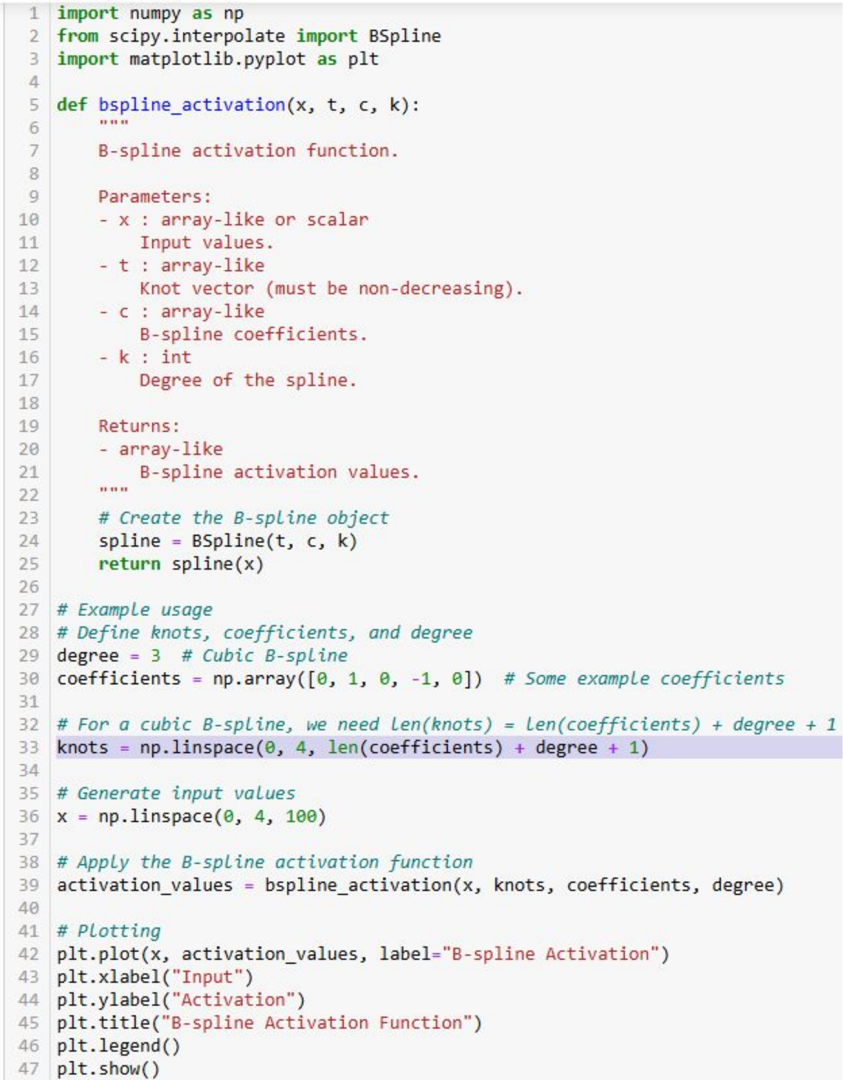
# Gordon Research

11/6/24

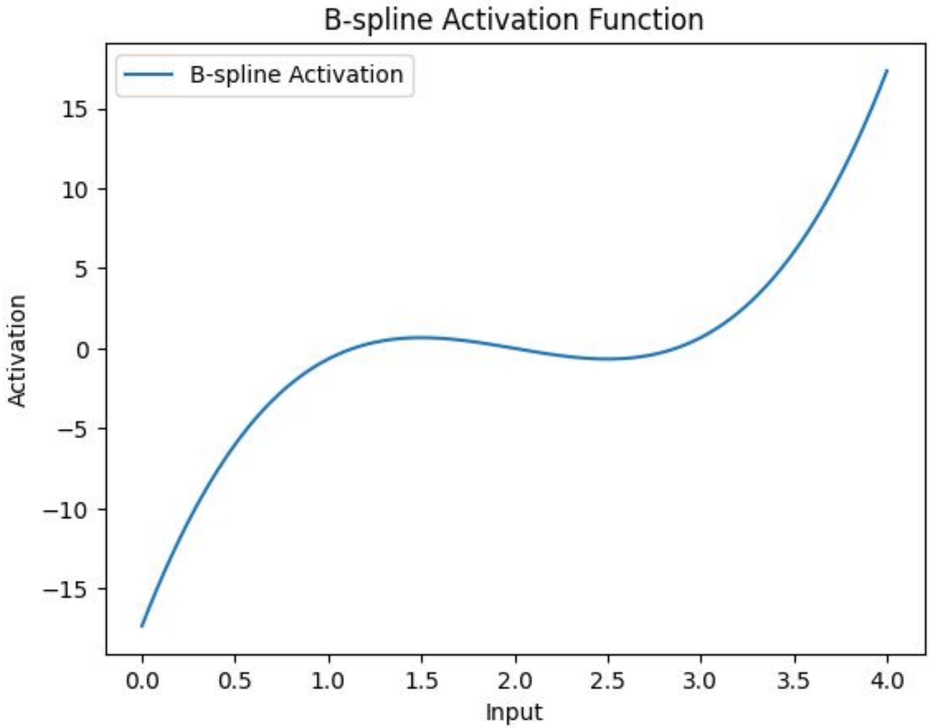
# B-Splines

- Consists of basis functions
- Formation:
  - Connects polynomial segments (ensures smoothness)
- Order:
  - Higher = smoother (more CD)
- Control points:
  - Influence shape of curve (magnets)
- Knot vector:
  - Splits curve into sections
  - Wider = smoother curve
    - fewer segments to form sharper turns



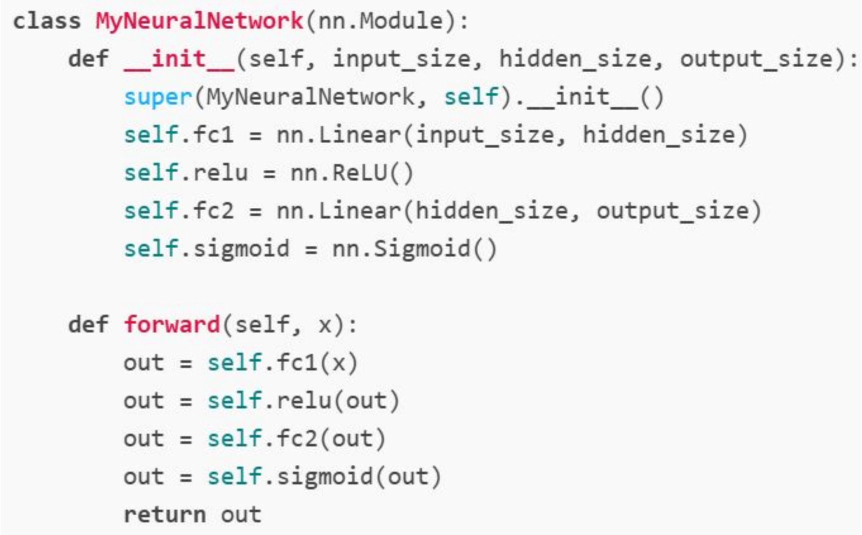


```python
import numpy as np
from scipy.interpolate import BSpline
import matplotlib.pyplot as plt

def bspline_activation(x, t, c, k):
    """
    B-spline activation function.

    Parameters:
    - x : array-like or scalar
        Input values.
    - t : array-like
        Knot vector (must be non-decreasing).
    - c : array-like
        B-spline coefficients.
    - k : int
        Degree of the spline.

    Returns:
    - array-like
        B-spline activation values.
    """
    # Create the B-spline object
    spline = BSpline(t, c, k)
    return spline(x)

# Example usage
# Define knots, coefficients, and degree
degree = 3  # Cubic B-spline
coefficients = np.array([0, 1, 0, -1, 0])  # Some example coefficients

# For a cubic B-spline, we need len(knots) = len(coefficients) + degree + 1
knots = np.linspace(0, 4, len(coefficients) + degree + 1)

# Generate input values
x = np.linspace(0, 4, 100)

# Apply the B-spline activation function
activation_values = bspline_activation(x, knots, coefficients, degree)

# Plotting
plt.plot(x, activation_values, label="B-spline Activation")
plt.xlabel("Input")
plt.ylabel("Activation")
plt.title("B-spline Activation Function")
plt.legend()
plt.show()
```

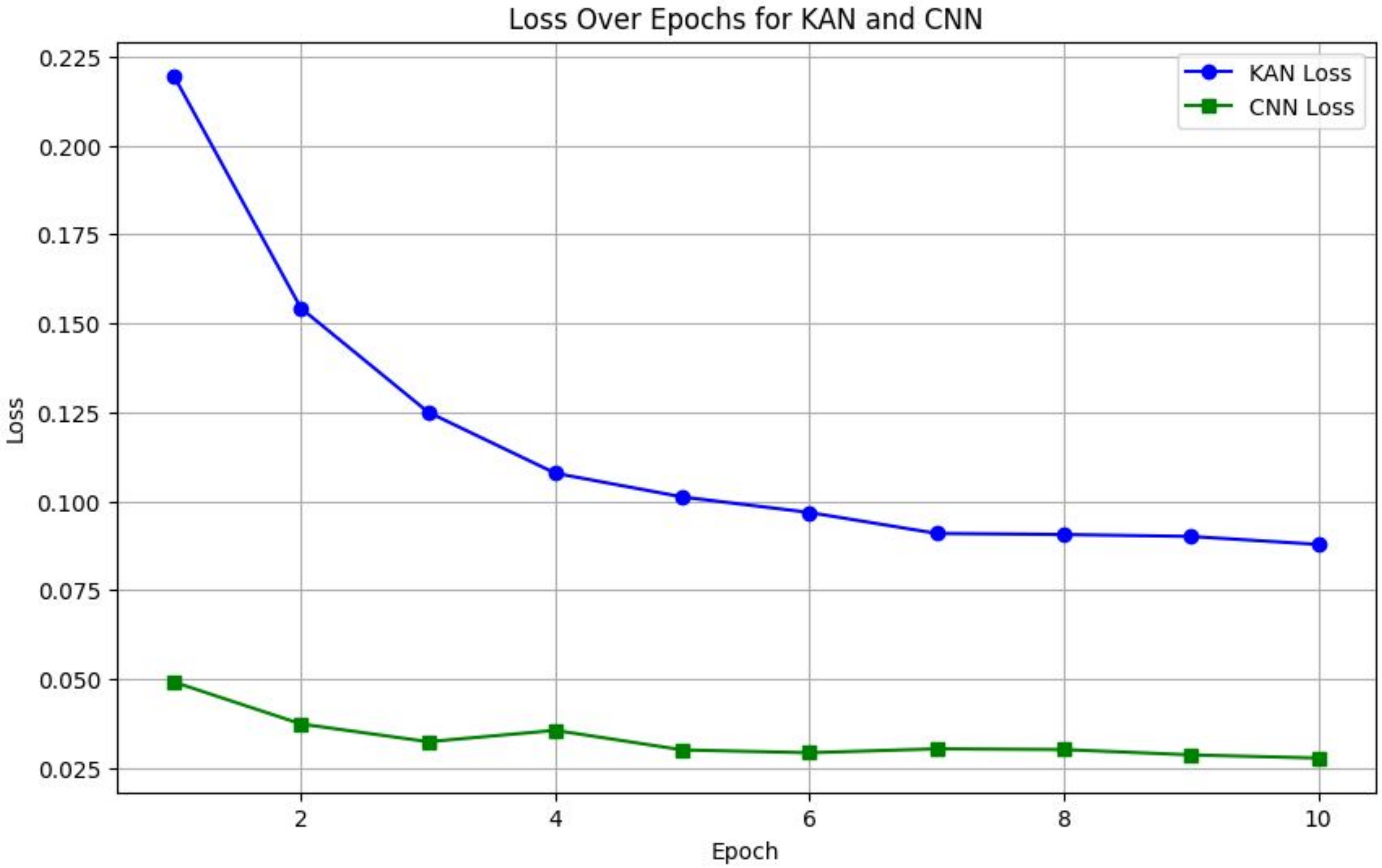


# Difference between init and forward

- Init:
    - Networks structure
    - Initialize parameters
    - Defines layers
- Forward:
    - Input data -> layers -> output
    - Predictions
    - Compute loss during training
- Both needed for PyTorch NN



```python
class MyNeuralNetwork(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(MyNeuralNetwork, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        out = self.sigmoid(out)
        return out
```


Loss Over Epochs for KAN and CNN
KAN Loss
CNN Loss
Loss
0.225
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
Epoch
2
4
6
8
10