AI UNCODED: DEMYSTIFYING NEURAL NETWORKS & CLUSTERING

A Practical Guide to Understanding FNNs, RNNs, and Customer Data Analysis

Minor In AI, IIT Ropar

8th May, 2025

Welcome

Have you ever walked into a local shop and wondered why some customers seem to get special treatment? Maybe they get personalized recommendations or exclusive discounts. That's not just luck; often, it's the result of data analysis! Imagine you're the friendly neighborhood shopkeeper, and you want to understand your customers better to serve them better. You have a record of how often each customer visits your store in a week (Frequency) and how much money they spend on average per visit (Spend). This information can be a goldmine if you know how to extract valuable insights from it.
This book is your guide to unlocking the secrets hidden within data. We'll explore some exciting AI techniques, starting with neural networks and then moving to clustering. We'll use real-world examples to make these concepts easy to grasp and fun to learn. Get ready for an exciting journey into the world of AI!

1 Neural Networks: A Quick Recap

Neural networks are at the heart of modern AI. They are inspired by the structure of the human brain and can learn complex patterns from data. Let's recap two fundamental types of neural networks that were discussed.

1.1 Feedforward Neural Networks (FNNs)

Imagine a simple task: Given the numbers 1, 2, and 3, predict the next number in the sequence. Most of us would instantly say "4." A feedforward neural network can learn to do this too. It treats the sequence (1, 2, 3) as a single input and predicts the output (4) in one go.

Key Characteristics of FNNs

- **Information flow:** Information flows in one direction – from input to output.
- **Analogy:** A straightforward calculation. You have the ingredients (input), and you follow a recipe (the network) to get the final dish (output).

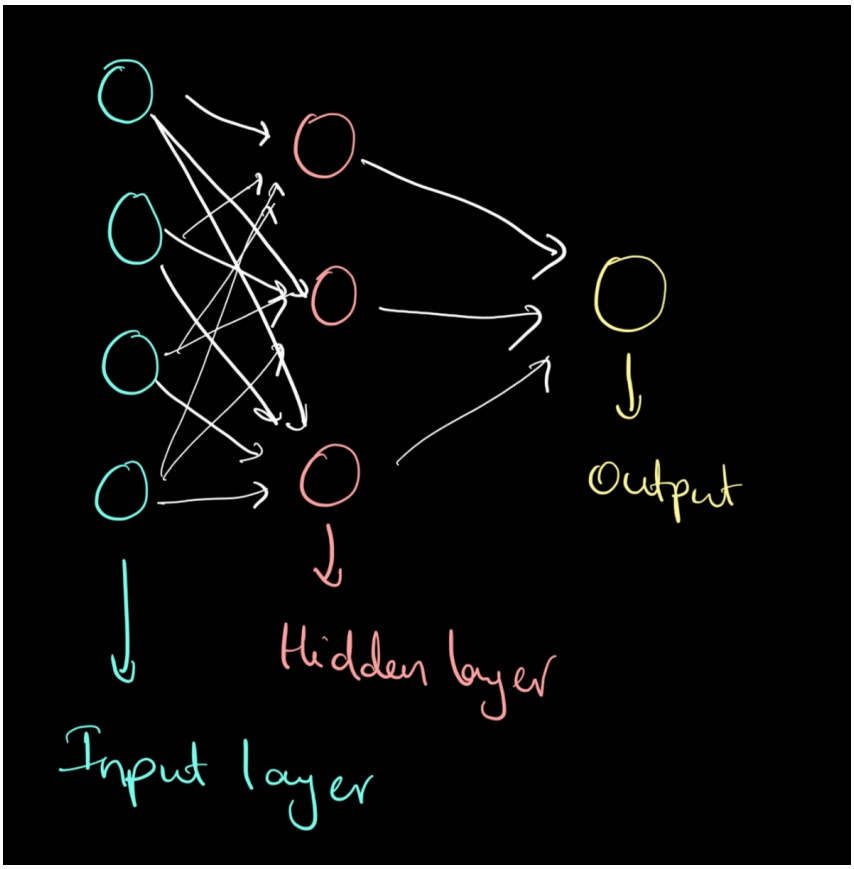

Figure 1: A simple diagram of a feedforward neural network with input layer, hidden layers, and an output layer.

1.2 Recurrent Neural Networks (RNNs)

Now, what if you wanted to learn *how* the sequence (1, 2, 3) progresses to 4? That's where recurrent neural networks come in. An RNN processes the sequence step-by-step. It first learns the relationship between 1 and 2, then between 2 and 3, and finally uses this information to predict 4.

Key Characteristics of RNNs

- **Information flow:** Information can loop back, allowing the network to remember previous inputs.
- **Analogy:** Reading a sentence word by word. You understand the meaning of each word in the context of the words that came before.

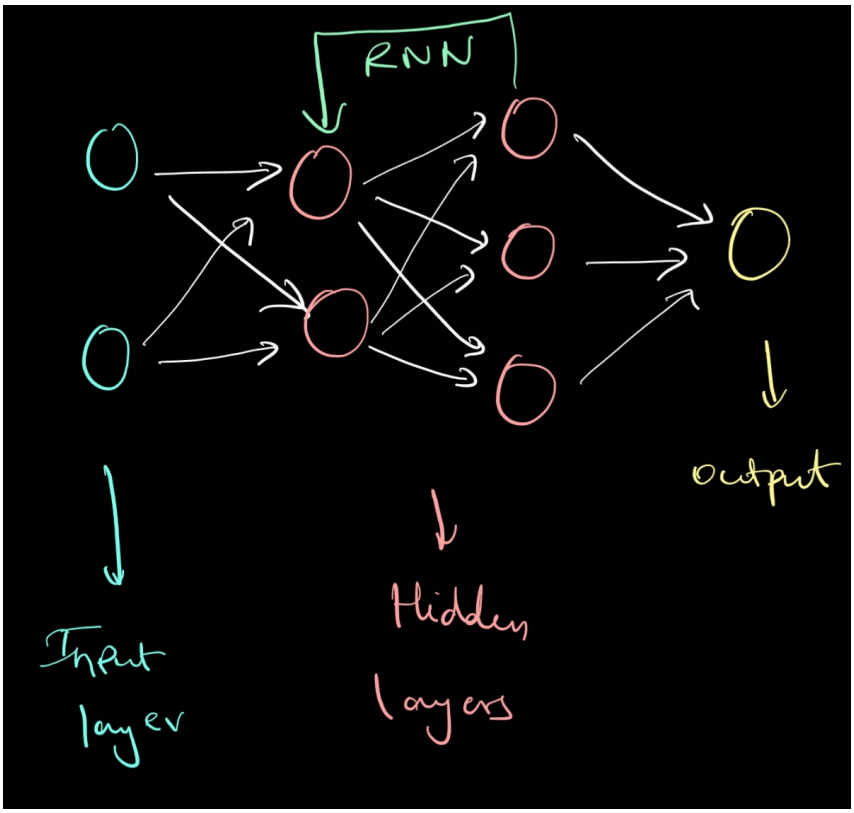

Figure 2: A simple diagram of a recurrent neural network showing the feedback loop.

1.3 Feedforward vs. Recurrent: Key Differences

Feature	Feedforward Neural Network (FNN)	Recurrent Neural Netwo
Data Handling	Treats input as a single unit	Processes input sequentially
Memory	No memory of previous inputs	Remembers previous inputs
Best Use Cases	Image recognition, static data	Time series, sequential data
Example	Predicting the output number from a bunch of input number	Time series forecasting

Table 1: Comparison between FNN and RNN

Let's understand how this works in code. We'll use the `Keras` library, which simplifies building neural networks. However, a caution: too much simplification can hide the underlying complexity. When that happens, you may lose control over the fine-grained parameters of a network and not understand exactly how the model is working.

1.4 Code Example: Feedforward Neural Network

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Define the model
model = keras.Sequential([
    layers.Dense(10, activation='relu', input_shape=(3,)), # Input layer with 3 features
    layers.Dense(64, activation='relu'),                   # Hidden layer
    layers.Dense(1)                                        # Output layer
```

```
])

# Compile the model
model.compile(optimizer='adam', loss='mse')

# Prepare the data
X = tf.constant([[1, 2, 3]], dtype=tf.float32)
y = tf.constant([4], dtype=tf.float32)

# Train the model
model.fit(X, y, epochs=200)

# Make a prediction
X_test = tf.constant([[2, 3, 4]], dtype=tf.float32)
prediction = model.predict(X_test)
print("Feedforward Prediction:", prediction)
```

Listing 1: Implementing a simple Feedforward Neural Network

In this code, we create a simple feedforward network. We define the layers, activation functions, and train the model to predict the next number in the sequence.

1.5 Code Example: Recurrent Neural Network

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np

# Define the model
model = keras.Sequential([
    layers.LSTM(10, activation='relu', input_shape=(3, 1)), # LSTM layer
    layers.Dense(1)                                          # Output layer
])

# Compile the model
model.compile(optimizer='adam', loss='mse')

# Prepare the data (important: reshape for RNN)
X = np.array([[[1], [2], [3]]], dtype=np.float32)  # Batch, time steps, features
y = np.array([4], dtype=np.float32)

# Train the model
model.fit(X, y, epochs=200)

# Make a prediction
X_test = np.array([[[2], [3], [4]]], dtype=np.float32)  # Reshape test data as
    well
prediction = model.predict(X_test)
print("Recurrent Prediction:", prediction)
```

Listing 2: Implementing a simple Recurrent Neural Network

Notice the key difference in how the data is prepared. For the RNN, we reshape the input data into a 3D array. This signifies that we are treating the sequence as a series of time steps.

1.6 What's Next?

You now have an understanding of what FeedForward and Recurrent Neural Networks are, and what the key differences are. In the next chapter, we move on to applying these concepts to real world customer data.

2 Clustering Customers: Finding Hidden Groups

Let's go back to the shopkeeper's problem. We have data on customer frequency and spend. How can we use this data to understand our customers better? That's where clustering comes in.

Clustering is a technique for grouping similar data points together. In our case, we can group customers with similar buying patterns.

2.1 A Simple Example: The Spreadsheet Approach

Imagine our shopkeeper has data on six customers:

Customer	Frequency (Visits/Week)	Spend (Average Amount)
C1	2	500
C2	10	800
C3	4	300
C4	11	1200
C5	3	350
C6	9	1000

Table 2: Customer Data

Let's also imagine we randomly selected Customers 1 and 4 as the initial centroids to calculate from.

We want to group these customers into two clusters based on their frequency and spend. A simple way to do this is using the Euclidean distance formula:

$$\textbf{Distance} = \sqrt{(Frequency_2 - Frequency_1)^2 + (Spend_2 - Spend_1)^2}$$

This formula calculates the "straight-line" distance between two points in a 2D space (frequency and spend).

2.2 Step-by-Step Clustering

1. **Choose Initial Centroids:** Select two random customers as the starting points for our clusters. Let's choose C1 (2, 500) as Centroid A and C4 (11, 1200) as Centroid B.
2. **Calculate Distances:** For each customer, calculate the distance to Centroid A and Centroid B using the Euclidean distance formula.
3. **Assign Customers to Clusters:** Assign each customer to the cluster with the closest centroid.
4. **Recalculate Centroids:** Calculate the new centroids by finding the average frequency and spend for each cluster.
5. **Repeat:** Repeat steps 2-4 until the clusters no longer change.

2.3 Code Example: Manual Iteration

Here's a Python code example that demonstrates these calculations:

```
import math

# Data
customers = {
```

```
    "C1": (2, 500),
    "C2": (10, 800),
    "C3": (4, 300),
    "C4": (11, 1200),
    "C5": (3, 350),
    "C6": (9, 1000)
}

# Initialize centroids manually
centroid_A = customers["C1"]
centroid_B = customers["C4"]

def euclidean(p1, p2):
    return math.sqrt((p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2)

def compute_new_centroid(cluster):
    freq = sum(c[0] for c in cluster) / len(cluster)
    spend = sum(c[1] for c in cluster) / len(cluster)
    return (freq, spend)

# Run until convergence
iteration = 1
prev_assignment = {}
while True:
    print(f"\n--- Iteration {iteration} ---")

    cluster_A = []
    cluster_B = []
    assignment = {}

    for cid, (f, s) in customers.items():
        dA = euclidean((f, s), centroid_A)
        dB = euclidean((f, s), centroid_B)

        if dA < dB:
            cluster_A.append((f, s))
            assignment[cid] = 'A'
        else:
            cluster_B.append((f, s))
            assignment[cid] = 'B'

        print(f"{cid}: Dist to A={dA:.2f}, B={dB:.2f} => Cluster {assignment[cid
    ]}")

    if assignment == prev_assignment:
        print("\nConverged.")
        break

    prev_assignment = assignment
    centroid_A = compute_new_centroid(cluster_A)
    centroid_B = compute_new_centroid(cluster_B)

    print(f"\nNew Centroid A: {centroid_A}")
    print(f"New Centroid B: {centroid_B}")

    iteration += 1

# Final Output
print("\nFinal Clusters:")
print("Cluster A:", [cid for cid, grp in assignment.items() if grp == 'A'])
print("Cluster B:", [cid for cid, grp in assignment.items() if grp == 'B'])
```

Listing 3: Manual Implementation of K-means Clustering

This code manually implements the steps of clustering using the Euclidean distance and iterative updates.

3 Clustering Using Library Functions

Manually calculating distances and updating clusters can be tedious. Fortunately, libraries like `scikit-learn` provide easy-to-use functions for clustering.

3.1 K-Means Clustering

K-means is a popular clustering algorithm that aims to partition data into k clusters, where each data point belongs to the cluster with the nearest mean (centroid).

3.2 Code Example: K-Means with Specified Initialization

```python
import numpy as np
from sklearn.cluster import KMeans

# Data
customer_ids = ["C1", "C2", "C3", "C4", "C5", "C6"]
data = np.array([
    [2, 500],
    [10, 800],
    [4, 300],
    [11, 1200],
    [3, 350],
    [9, 1000]
])

# Initial centroids from C1 and C4
initial_centroids = np.array([
    [2, 500],    # C1
    [11, 1200]   # C4
])

# KMeans model with manually specified initial centroids
kmeans = KMeans(n_clusters=2, init=initial_centroids, n_init=1, max_iter=100,
    random_state=42)
kmeans.fit(data)

# Results
labels = kmeans.labels_

# Display results
for cid, label in zip(customer_ids, labels):
    print(f"{cid} -> Cluster {label}")

print("\nCluster Centers:")
print(kmeans.cluster_centers_)
```

Listing 4: K-means Clustering with Specified Initial Centroids

3.3 Code Example: K-Means with Random Initialization

If you don't have any prior knowledge about where your centroids should be initialized, you can have them randomly be initialized.

```python
import numpy as np
from sklearn.cluster import KMeans
```

```
# Data
customer_ids = ["C1", "C2", "C3", "C4", "C5", "C6"]
data = np.array([
    [2, 500],
    [10, 800],
    [4, 300],
    [11, 1200],
    [3, 350],
    [9, 1000]
])

# KMeans model without manually specified initial centroids
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
kmeans.fit(data)

# Results
labels = kmeans.labels_

# Display results
for cid, label in zip(customer_ids, labels):
    print(f"{cid} -> Cluster {label}")

print("\nCluster Centers:")
print(kmeans.cluster_centers_)
```

Listing 5: K-means Clustering with Random Initial Centroids

3.4 Code Example: Visualizing The Clusters

You can also visualize the clusters to gain better insight.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Data
customer_ids = ["C1", "C2", "C3", "C4", "C5", "C6"]
data = np.array([
    [2, 500],
    [10, 800],
    [4, 300],
    [11, 1200],
    [3, 350],
    [9, 1000]
])

# Initial centroids from C1 and C4
initial_centroids = np.array([
    [2, 500],    # C1
    [11, 1200]   # C4
])

# Fit KMeans
kmeans = KMeans(n_clusters=2, init=initial_centroids, n_init=1, max_iter=100,
    random_state=42)
kmeans.fit(data)
labels = kmeans.labels_
centroids = kmeans.cluster_centers_

# Plot
colors = ['red', 'blue']
for i in range(2):
```

```
    cluster_points = data[labels == i]
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], c=colors[i], label=f
    'Cluster {i}', s=100)

# Plot centroids
plt.scatter(centroids[:, 0], centroids[:, 1], c='black', marker='X', s=200,
    label='Centroids')

# Add customer IDs as labels
for i, txt in enumerate(customer_ids):
    plt.annotate(txt, (data[i, 0] + 0.1, data[i, 1] + 10))

plt.xlabel('Frequency')
plt.ylabel('Spend')
plt.title('Customer Clusters using K-Means')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Listing 6: Visualizing K-means Clusters

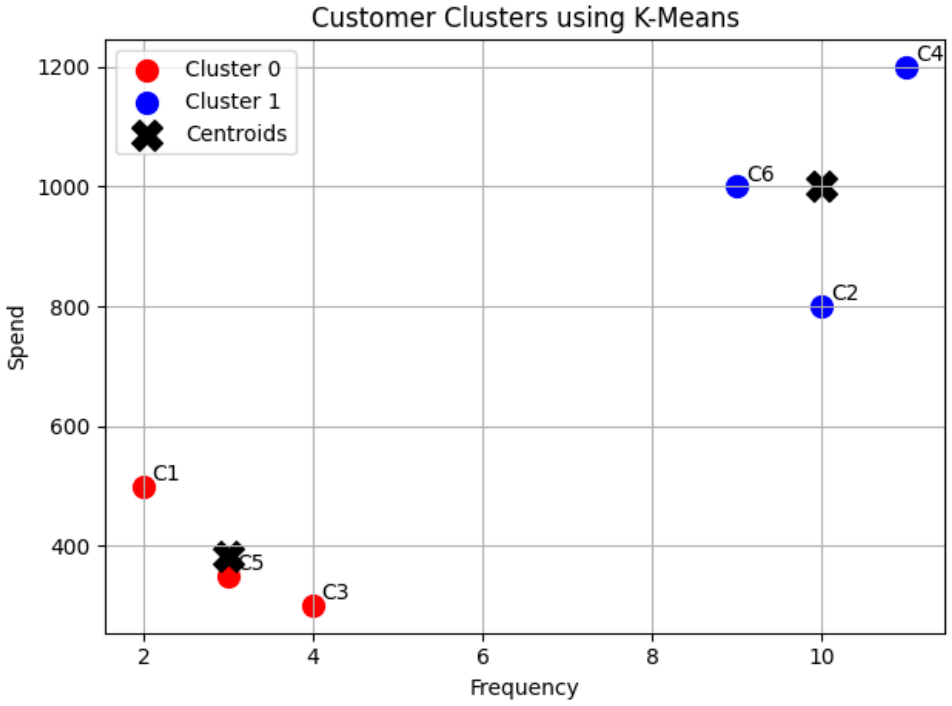

Figure 3: A scatter plot showing customers clustered into different segments, with centroids marked.

This book has been an introduction to neural networks and clustering. You now know the difference between FeedForward and Recurrent Neural Networks, and what sort of tasks each kind of Neural Network is suited for. You also learned how to manually segment data, how clustering works, and what you can do with Clustering.

Remember, AI is a journey. This book is only a starting point. Keep exploring, experimenting, and building!