Week 1: Tools for Data Science

1.1 NumPy (Numerical Python)

We started with NumPy, which is the foundation for almost all scientific computing in Python.

Why we use it: Standard Python lists are flexible but slow. NumPy introduces the "ndarray" (n-dimensional array), which stores data more efficiently in memory.

Key Learning: The most important thing I learned here was "vectorization." This means we can perform calculations on entire arrays at once without writing for loops, which makes the code run much faster. We also looked at broadcasting, which allows us to add or multiply arrays of different sizes.

1.2 Pandas

Next, we learned Pandas, which is used for data manipulation.

- **DataFrames:** This is basically like a programmable Excel sheet. We learned how to load data from CSV files and view it using df.head().
- **Cleaning Data:** Real-world data is rarely perfect. We practiced checking for missing values (NaN) and either dropping them or filling them with the mean/median. We also learned how to filter rows based on specific conditions.

1.3 Matplotlib

Finally, to visualize our data, we used Matplotlib. We learned that just looking at numbers isn't enough; we need to see trends. We created line plots to see changes over time, scatter plots to check the relationship between variables, and histograms to see the distribution of data.

Week 2: Machine Learning Foundations

2.1 Activation Functions

We learned that a Neural Network is just a lot of matrix multiplications, which are linear. To solve complex problems, we need "non-linearity." That's where activation functions come in:

- **Sigmoid:** Squishes numbers between 0 and 1. It's useful for probability but can cause gradients to vanish.
- **Tanh:** Similar to Sigmoid but ranges from -1 to 1, so the output is zero-centered.
- **ReLU (Rectified Linear Unit):** This is the most popular one. It just outputs the input if it's positive, and 0 if it's negative. It's very fast to compute.
- **Softmax:** We use this in the last layer for classification because it converts raw numbers into probabilities that sum up to 1.

2.2 Loss Functions

To train a model, we need to measure how "wrong" it is.

- **Mean Squared Error (MSE):** We used this for regression problems (like predicting a price).
- **Binary Cross Entropy:** Used when there are only two classes (Yes/No).
- **Categorical Cross Entropy:** Used when we have multiple classes (like predicting digits 0-9).

2.3 Unsupervised Learning: K-Means Clustering

Concept: The idea is to separate data into k different clusters. The clusters are chosen so that points in the same cluster are close to each other spatially, usually measured by Euclidean Distance.

The Algorithm:

1. Pick k random points as "centroids".
2. Calculate the distance from every data point to these centroids.
3. Assign each point to the nearest cluster.
4. Update the centroid position by taking the average (mean) of all points in that cluster.
5. Repeat until the centroids stop moving.

2.4 Neural Network Training

Finally, we put it all together.

- **Forward Propagation:** Passing input data through the layers to get a prediction.
- **Backpropagation:** The process of calculating gradients—basically finding out which weight contributed how much to the error.
- **Optimizers:** We use algorithms like Stochastic Gradient Descent (SGD) to update the weights and reduce the loss.

Week 3: Deep Learning & Computer Vision (CNNs)

This week was about processing images. We learned that standard Neural Networks (MLPs) are bad for images because they flatten the input, destroying the spatial structure. Also, for a large image (like 1000x1000), an MLP would need trillions of parameters, which is impossible to train.

3.1 Convolutional Neural Networks (CNNs)

To fix this, we use Convolution. The key ideas are:

- **Translation Invariance:** A cat in the top-left corner is the same as a cat in the bottom-right. The weights shouldn't depend on the absolute position.
- **Locality:** To understand a pixel, we only need to look at its neighbors, not the whole image at once.

3.2 The Convolution Operation

Mathematically, this is actually a "Cross-Correlation" operation.

- We slide a small window (kernel) over the image.
- At each step, we multiply the kernel values with the image pixels and sum them up.

- **Feature Maps:** The output is called a feature map. Each channel in this map represents a specific feature (like a vertical edge) detected at different locations.

3.3 Key Terms

- **Stride:** How many pixels the kernel moves at a time. Increasing stride makes the output smaller and reduces redundant info.
- **Padding:** Adding zeros around the border so the image doesn't shrink too fast.
- **Receptive Field:** The part of the original image that a specific neuron is looking at.

3.4 Batch Normalization

We also learned a technique called Batch Normalization to make training faster.

- It normalizes the output of a layer to have a mean of 0 and standard deviation of 1.
- This fixes a problem called "Internal Covariate Shift," where the distribution of inputs keeps changing during training.
- It also acts as a regularizer, helping to prevent overfitting.

Week 4: Sequence Modeling & RNNs

In the final week, we moved to sequence data. Standard networks assume all inputs are independent, but for things like text or stock prices, the order matters.

4.1 Recurrent Neural Networks (RNNs)

RNNs are designed to handle sequences.

- **Hidden State:** The main difference is that RNNs have a "hidden state" (H_t). This state acts like a memory of what happened in the previous steps.
- **The Logic:** To calculate the current state, the network looks at the current input (X_t) AND the previous hidden state (H_t).

    - Formula: H_t = phi(X_t *W_xh + H_{t-1}* W_{hh} + b_h).

4.2 Language Models

We applied this to text data (Language Modeling).

- **Tokenization:** First, we break sentences into tokens (words or characters).
- **One-Hot Encoding:** We represent these tokens as vectors where only one position is 1 and the rest are 0.
- **Perplexity:** To measure how good our language model is, we use "Perplexity." A lower perplexity means the model is less surprised by the next word. It is basically the exponential of the cross-entropy loss.

4.3 Training Challenges

Training RNNs is difficult because of "Backpropagation Through Time".

- **Exploding Gradients:** Because we keep multiplying matrices over many time steps, the gradients can become huge and crash the training.
- **Solution (Gradient Clipping):** We learned a trick called Gradient Clipping. If the gradient vector gets too large (larger than a threshold (Theta), we shrink it down. This keeps the training stable.