Tokens to Thought A Contextual Transformer

Amogh Agrawal (Roll No. 24b1092)

January 7, 2026

Overview

During the initial two weeks of the project, it was more of a slow build-up before becoming a sprint. In the first week, I became familiar with NumPy, Matplotlib and gradient descent; in the second week, I was tasked with the application of the same tools for constructing logic gates and adders using only NumPy arrays. If I spotted an answer I thought matched what I was trying to do, I rewrote it using my own words to reinforce my memory.

1 Week 1 Assignment Summary

1.1 NumPy Warm-Up Notebook

Recreating the starter matrix by reshaping a flat buffer might look like a small flourish, yet it pushed me to think about how NumPy organises data. I repeated that mindset throughout the notebook: random matrices come from `np.random.default_rng` so that seeds are explicit, zero and one tensors are formed with `broadcast_to` and `full` to highlight broadcasting, and transposes lean on `swapaxes` instead of the usual shorthand. Even the column-vector multiplication exercise became an excuse to assert shapes and make sure every operation lined up without guesswork. The slicing and broadcasting sections carry the same spirit—I relied on `np.arange` for index construction and in-place universal functions for memory-friendly arithmetic. By the time I compared the vectorised and non-vectorised loops, the gap in execution time felt intuitive rather than surprising.

1.2 Matplotlib & Pandas Exercises

Visualizing sales data is not just following some instructions or matching them to a list; instead, it creates a narrative via a chart. I began utilizing explicit axes instead of stateful matplotlib pyplot to provide consistent formatting, then moved to styling my elements. For example, I used dotted crimson lines with thick markers for profit, a grid-backed panel to delineate products, and legends that read in complete sentences rather than abbreviated column names. To create the pie chart, I compiled the yearly totals into a tidy DataFrame, sorted them, then formatted the percentages and absolute values in the pie chart using the formatter that I created. The very slight "explode" feature adds space around the individual wedges, while maintaining an equal aspect ratio ensures that all the wedges maintain their circular integrity.

1.3 Multivariate Gradient Descent

The gradient-descent notebook was my chance to write the optimisation loop I always wished I had on hand. I wrapped the polynomial in a clean `objective` function, derived its gradient explicitly, and then scripted a descent routine with tolerance-based stopping, capped iterations, and a simple backtracking step that halves the learning rate whenever a move fails to improve the objective. Running the algorithm from very different starting points showed how sensitive gradient descent can be to step size, yet the logged trajectory made each journey traceable. The returned dictionary captures the final position, function value, gradient norm, iteration count, and even the complete path for future visualisations.

2 Week 2 Assignment Summary

2.1 Perceptron Foundations

After establishing the week-one basics, I created a tiny perceptron class from the ground up: all weights get set to initial values before each training session; adjustments occur on an individual basis for all samples; and the process will end when a complete iteration occurs without any sign of errors. I was extremely pleased to find that the predictions made on the AND gate dataset were 100% accurate. This indicates that the model successfully classifies databases that are linearly separable, exactly as has been theorised.

2.2 XOR Gate Exploration

Naturally, it was impossible for XOR to be solved using a single perceptron. The mismatch report serves as a constant reminder of limitations that exist when trying to model non-linear decision boundaries. To bridge this dividing line, a two-layer neural network was created with hyperbolic tangent hidden units and a sigmoid output. The code for everything from Xavier initialization of parameters to using the cross-entropy loss function and employing vectorisation during backpropagation can be found within one Python script file. After performing several thousand gradient descend iterations, the network successfully recreated the XOR truth table which felt akin to witnessing an old idea become a reality.

2.3 Adder Circuits

The network's ability to perform a binary sum with an XOR operation was enough to justify creating a full adder. The design of the network retained its two-layer architecture with an increase in the number of hidden units from 2 to 6 units within the hidden layer. This change increased the capacity of the hidden layer to provide the necessary outputs for the three input bits and two output bits (the sum and carry). Multiple predictions made through the use of a ripple-carry demonstration combine the individual components of the prediction. Bits are processed from least significant to greatest significant, with the adder calculating an intermediate carry and sum, while a straightforward helper provides the user with a final decimal representation of the bit string the user has provided to verify its correctness. The user was very excited about the capability of the network to compute two four-bit numbers correctly and on its own. It certainly made for a good way to spend a week.