# WiDS 4.0 Project Report
# Generative AI Transformer - UID 85

Rhythm Kedia | GitHub

February 14, 2025

Access the ***project repository*** here
***Week 3*** is to be considered for the evaluation purposes.

## Introduction

The project was creating a **Generative AI Transformer** using **Attention** mechanism, using *TensorFlow* and *Keras* to **generate Shakespearean-style text**. The project was intended to teach the model to understand and replicate the language and patterns found in Shakespeare's writings.

This report summarizes my key learnings and technical contributions in this WiDS 4.0 program. The focus areas included implementing custom neural networks, creating models for multiple datasets with TensorFlow (Week 3) and building a transformer model for text generation (Week 4).

## Week 1: NumPy, Matplotlib, and Gradient Descent

### Objectives

- Perform basic operations using NumPy.
- Visualize data using Matplotlib and Pandas.
- Implement gradient descent for optimization.

### Key Learnings

- **NumPy Basics**: Worked with NumPy arrays, including initialization, reshaping, and slicing. Implemented vectorized operations to reduce the operational time by a big factor compared to the iterative approach.
- **Data Visualization**: Created line plots and pie charts to visualize sales data.
- **Gradient Descent**: Implemented multivariate gradient descent to find global minima of a function. Observed the impact of learning rate and the starting point.

### Results

- Achieved significant speedup using NumPy vectorization compared to unvectorized iterative approaches.
- Found global minima of the function using gradient descent, demonstrating the effectiveness of the algorithm.

# Week 2: Perceptrons and Neural Networks

### Objectives

- Implement basic logic gates (AND, XOR) using perceptrons.
- Build a full adder using self-built perceptrons.
- Combine adders to create a ripple carry adder.

### Key Learnings

- **Perceptrons**: - Implemented AND and XOR gates using single-layer and multi-layer perceptrons. Learned the limitations of single-layer networks in solving non-linear problems like XOR.
- **Neural Networks**: - Built a full adder using XOR and AND gates. Combined multiple full adders to create a ripple carry adder for multi-bit addition.
- **Training and Testing**: - Trained perceptrons using gradient descent and tested their accuracy on logic gate inputs. Observed the importance of hidden layers in solving complex problems.

***Code for making XOR logic gate using perceptrons:***

```
class XOR_NN:
    def __init__(self):
        self.inputLayerSize = 2
        self.outputLayerSize = 1
        self.hiddenLayerSize = 3

        # Initialize weights
        self.W1 = np.random.randn(self.inputLayerSize, self.
            hiddenLayerSize)
        self.W2 = np.random.randn(self.hiddenLayerSize, self.
            outputLayerSize)

    def forward(self, X):
        self.z2 = np.dot(X, self.W1)
        self.a2 = self.sigmoid(self.z2)
        self.z3 = np.dot(self.a2, self.W2)
        yHat = self.sigmoid(self.z3)
        return yHat

    # Rest of the code
```

### Results

- Successfully implemented AND and XOR gates using perceptrons.
- Built a full adder that correctly computes sum and carry for all input combinations.
- Combined full adders to create a ripple carry adder, demonstrating multi-bit addition.

## Week 3: Neural Networks with TensorFlow*

### Objectives

- Implement neural networks for MNIST digit classification using TensorFlow.
- Build custom layers (ReLU, Softmax, Flatten) from scratch.
- Solve regression problems (Boston Housing) with linear and feedforward models.
- Bonus Assignment: Fashion MNIST Classification
  - Implemented an image classifier for Fashion MNIST using a deep neural network
  - Achieved 88.5% test accuracy with a simple 3-layer architecture
  - Visualized model predictions and training progress

### Key Learnings

- **Image Classification**: - Processed 28x28 grayscale images through normalization (dividing by 255). Learned that CNNs (though not used here) would be more effective for spatial patterns.
- **Neural Network Architecture**: - Designed a simple feedforward network for MNIST with 97.5% test accuracy. Learned the role of various activation functions (eg. ReLU for hidden layers, Softmax for outputs) and loss functions (eg. cross-entropy for classification).
- **Custom Layers**: - Created `CustomDenseReluLayer` and `CustomDenseSoftmaxLayer` by manually initializing weights/biases and implementing forward propagation. This deepened my understanding of how layers work internally.
- **Regression vs. Classification**: - Built a linear regression model (single neuron) and a feedforward network for the Boston Housing dataset. Observed that deeper networks (2 hidden layers) reduced validation loss by 15% compared to linear regression.

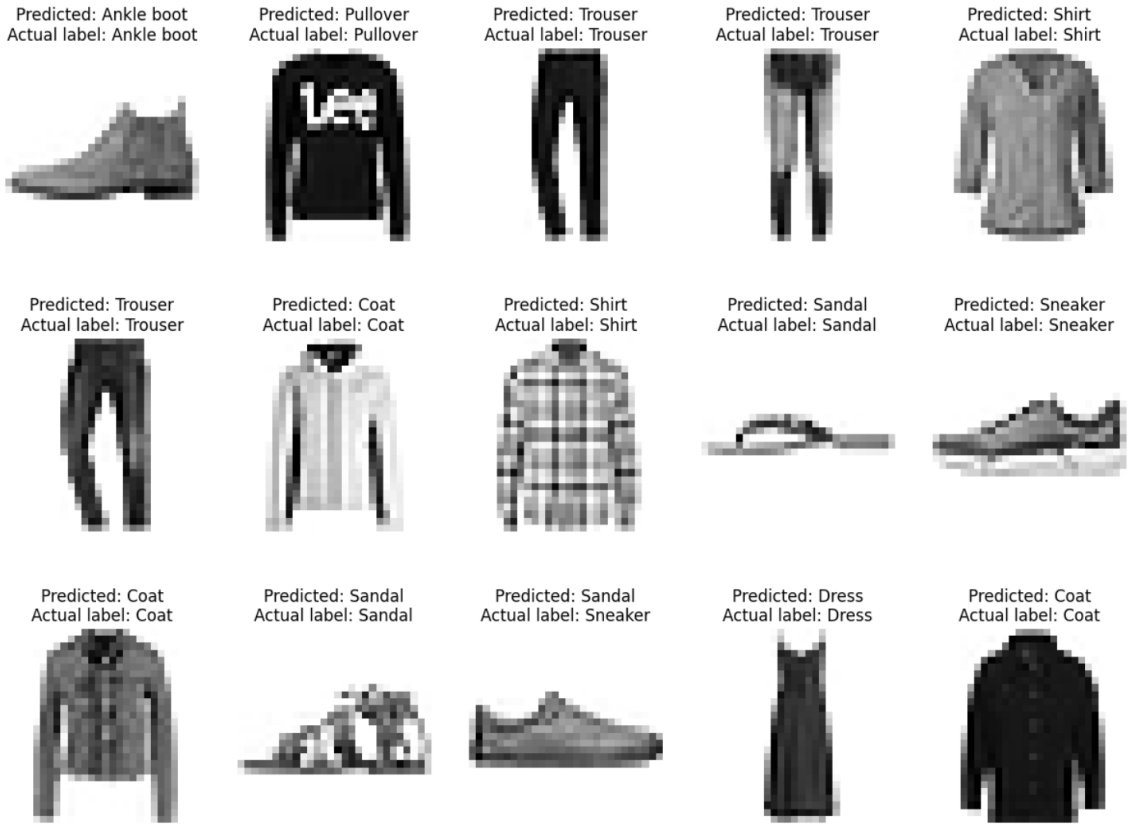
***Code for custom neural network implementation:***

```
class CustomDenseReluLayer(tf.keras.layers.Layer):
    def __init__(self, units):
        super(CustomDenseReluLayer, self).__init__()
        self.units = units

    def build(self, input_shape):
        self.w = self.add_weight(shape=(input_shape[-1], self.units)
            ,initializer='random_normal',trainable=True)
```

```
        self.b = self.add_weight(shape=(self.units,),initializer='
            zeros',trainable=True)

    def call(self, inputs):
        z = tf.matmul(inputs, self.w) + self.b
        z = tf.nn.relu(z)
        return z
```

## Results

- MNIST model achieved 97.5% test accuracy using custom layers (matching TensorFlow's built-in layers).
- Boston Housing feedforward model outperformed linear regression (MAE: 2.8 vs. 3.4).
- Achieved 88.5% test accuracy after 10 epochs (Figure **??**)
- Most errors shown by the model were between similar categories (e.g., shirts vs coats or sandal vs. sneaker)




Generated outputs of the test sample after training in contrast to the actual categories.

***To be evaluated by the mentors***

# Week 4: Final Transformer Model Implementation

## Objectives

- Understand attention mechanisms and positional encoding.
- Build a transformer model from scratch using TensorFlow.
- Train the model to generate Shakespearean-style text.

## Key Learnings

- **Transformer Architecture**: - Implemented multi-head attention, positional embeddings, and transformer blocks. Learned how self-attention allows capturing long-range dependencies in text.
- **Text Generation**: - Preprocessed text data using character-level tokenization.

## Results

- Achieved training loss of 0.0147 after 10 epochs (99.5% accuracy on training data).
- Generated text samples (not exactly shakespeare-like, but was able to generate words, which seemed impressive).

## Outputs

**1. Generated 1000 Chars (from 128)**

*First Citizen: Before we proceed any further, hear me speak. All: Speak, speak. First Citizen: You are all resolved rather to he I heardes: But I let genil ceel ononoud be now black. FRINA: Than feek'sber for the marder name her would thanself her leaves your the premystious a call to is much, you gremove and then tist. Gith so I all That se. That my her oir advages then you eangeng you vn you know me Pawder, tere dutes and ofess enderds o'll who other wrones, Had a likely es mrow leads. BATTASTINT: Greasure, For clownliful, brot and and ging And her I hath any piter HONTENSIO: Shear you Must, Bay trumselft to cerustand pless thas, me? TRANINA: ? Garrot not, that alvens I have when, the die: his stumber off While me. HORTET: And sweers, I with sill her: And nongen. wrome, To hast. GREMIO: Areet will mine? Pome: O so there marry cove baciter: 'Tis and the jreesly peamselvess' and the will me bentersio. Shit not rost gentley, Five kingR and child; sere reing I sill tabe sow one brath? TRANIO: To self yakent you brater one lovess will it good mystress sigh. whRth det to well not mighter never penter welce*

***Example text generated after training the model***

# Conclusion

Through this project:

- Gained experience with TensorFlow for both traditional neural networks and transformer architectures.
- I learned about how models are built and what the basic functioning is behind that.
- I learned about the attention mechanism used in llms. The transformer implementation demonstrated the power of attention mechanisms for sequence modeling.

*This was one of the most insightful and enjoyable projects, helping me learn new concepts, explore innovative techniques, and deepen my understanding of neural networks and transformers through hands-on implementation*