DL Assignment 2 - Part 1: Theoretical Part

Sruthisri Venkateswaran (sv94 - 50604295), Saroja Vuluvabeeti (sarojavu - 50593902)

April 4, 2025

Part I.I: Autoencoders for Anomaly Detection

Task 1: Calculation of Trainable Parameters

In a fully connected (FC) layer, the number of trainable parameters includes both weights and biases:

$$\text{Weights} = n_{\text{in}} \times n_{\text{out}} \tag{1}$$

$$\text{Biases} = n_{\text{out}} \tag{2}$$

$$\text{Total Parameters} = (n_{\text{in}} \times n_{\text{out}}) + n_{\text{out}} \tag{3}$$

For each layer:

- **Input Layer (1000) → Hidden Layer 1 (512)**

$$1000 \times 512 + 512 = 512,512 \tag{4}$$

- **Hidden Layer 1 (512) → Bottleneck Layer (32)**

$$512 \times 32 + 32 = 16,416 \tag{5}$$

- **Bottleneck Layer (32) → Hidden Layer 2 (512)**

$$32 \times 512 + 512 = 16,896 \tag{6}$$

- **Hidden Layer 2 (512) → Output Layer (1000)**

$$512 \times 1000 + 1000 = 513,000 \tag{7}$$

Final total trainable parameters:

$$512,512 + 16,416 + 16,896 + 513,000 = \mathbf{1,058,824} \tag{8}$$

Task 2: L2 Regularization in MSE Loss

L2 regularization, also called weight decay, prevents overfitting by adding a penalty term to the loss function to discourage large weights.

Loss Function with L2 Regularization

The standard Mean Squared Error (MSE) loss:

$$L_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{9}$$

With L2 regularization:

$$L = L_{\text{MSE}} + \lambda \sum_{j} ||W_j||^2 \tag{10}$$

where λ is the regularization strength.

During backpropagation, the weight update rule becomes:

$$W_j = W_j - \eta \left(\frac{\partial L_{\text{MSE}}}{\partial W_j} + \lambda W_j \right) \tag{11}$$

where η is the learning rate. This effectively shrinks the weights towards zero, reducing overfitting.

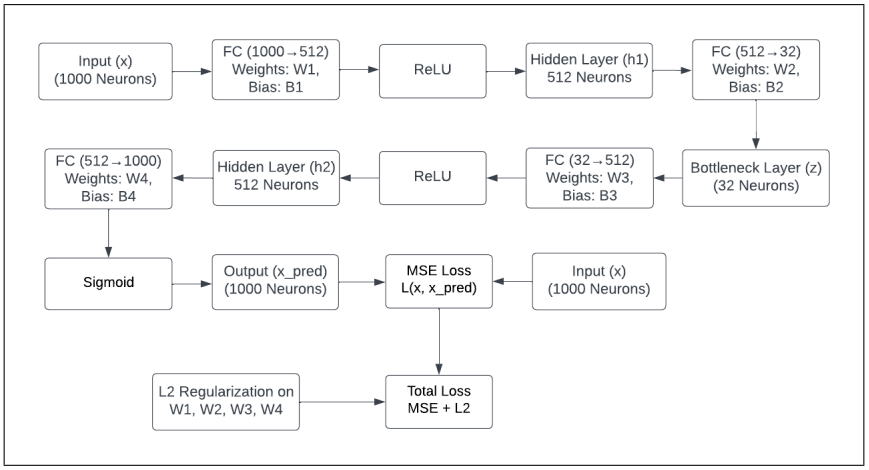

Figure 1: Computational Graph of Autoencoder for Anomaly Detection

Task 3: Computational Graph

Refer Figure 1

Task 4: Challenges and Limitations

1. **Defining Anomalies**:

   It is difficult to set a reconstruction error threshold.

   Autoencoders reconstruct input data but need a threshold to classify anomalies.

2. **Overfitting to Normal Data**:

   If trained only on normal data, the model may generalize too well and reconstruct anomalies, making detection difficult.

3. **Data Imbalance**:

   Manufacturing anomaly datasets are often imbalanced (more normal than defective samples).

   Training an autoencoder without enough anomalies makes detection unreliable.

4. **Deployment Challenges**:

   Requires real-time inference in manufacturing, but deep models can be slow.

   Needs continuous retraining as new anomalies arise.

Task 5: Improvements and Extensions

1. **Alternative Architectures**:

   Instead of vanilla autoencoders, try Variational Autoencoders (VAEs) or Denoising Autoencoders to improve generalization.

2. **Loss Function Modifications**:

   Use Reconstruction Error + Anomaly Score instead of only MSE.

   Introduce contrastive learning to separate normal and anomalous distributions.

3. **Data Preprocessing**:

   Normalize sensor data properly.

   Use feature engineering techniques like PCA to extract more useful signals.

4. **Integration with Other Models**:

   Combine autoencoders with one-class SVMs or isolation forests for anomaly detection.

   Use attention-based mechanisms to focus on the most critical sensor readings.

Part I.II: Transformers and Self-Attention

Task 1a. Linear Transformations

In the self-attention mechanism of the Transformer, the input sequence

$$x = (x_1, x_2, \ldots, x_N), \quad x_i \in \mathbb{R}^d$$

is first projected into three different representations using learned linear transformations: the **Query (q)**, **Key (k)**, and **Value (v)** vectors.

- **Query (q):** Represents the "question" that a particular token is asking about other tokens in the sequence.
- **Key (k):** Represents the "content" of the token; determines how relevant the token is when answering a query.
- **Value (v):** Contains the actual information of the token to be aggregated through attention.

Each vector is obtained via a linear transformation:

$$\begin{aligned} q_i &= W_q x_i + b_q \\ k_i &= W_k x_i + b_k \\ v_i &= W_v x_i + b_v \end{aligned}$$

where:

- $W_q, W_k, W_v \in \mathbb{R}^{d_k \times d}$ are learnable weight matrices.
- $b_q, b_k, b_v \in \mathbb{R}^{d_k}$ are learnable bias vectors.
- d is the dimensionality of the input embeddings.
- d_k is the dimensionality of the projected q, k, v vectors (usually $d_k = d_v$).

In matrix form, if $X \in \mathbb{R}^{N \times d}$ denotes the full input sequence (stacked row-wise), the transformations are:

$$\begin{aligned} Q &= XW_q^\top + b_q \\ K &= XW_k^\top + b_k \\ V &= XW_v^\top + b_v \end{aligned}$$

where $Q, K, V \in \mathbb{R}^{N \times d_k}$ are the query, key, and value matrices respectively, with each row corresponding to a transformed input token.

Task 1b. Scaled Dot-Product Attention

Once the query (q_i) and key (k_j) vectors are obtained, the attention score between the i-th query and j-th key is computed using a dot product:

$$\text{score}_{ij} = \frac{q_i \cdot k_j}{\sqrt{d_k}}$$

Here, d_k is the dimensionality of the key (and query) vectors. The division by $\sqrt{d_k}$ serves to scale the dot product. Without this normalization, for large values of d_k, the dot products can become large in magnitude, pushing the softmax function into regions with very small gradients. This leads to poor learning dynamics. Thus, the scaling improves numerical stability and learning efficiency.

The attention weights are then obtained by applying a softmax function over all keys:

$$\alpha_{ij} = \frac{\exp\left(\frac{q_i \cdot k_j}{\sqrt{d_k}}\right)}{\sum_{j'=1}^{N} \exp\left(\frac{q_i \cdot k_{j'}}{\sqrt{d_k}}\right)}$$

Task 1c. Weighted Sum

The attention weights α_{ij} are used to compute a weighted sum of the value vectors v_j. This results in an output vector z_i that captures the most relevant information for the i-th token, based on the entire sequence:

$$z_i = \sum_{j=1}^{N} \alpha_{ij} v_j$$

This operation effectively aggregates information from all positions in the input sequence, giving more importance to elements that are more relevant to the current token x_i.

Task 1d. Optional Output Transformation

After obtaining the attention output vector z_i, an optional linear transformation can be applied to project it back to the original embedding space:

$$\tilde{z}_i = W_o z_i + b_o$$

where:

- $W_o \in \mathbb{R}^{d \times d_k}$ is a learnable weight matrix.
- $b_o \in \mathbb{R}^d$ is a learnable bias vector.

This transformation allows the model to further process and adapt the contextual representation for downstream tasks. It also ensures compatibility in dimensionality with the residual connections in the Transformer architecture. As a result, it can impact how much influence the attention output has on the final representation by reprojecting and re-weighting features.

Task 2: Computational Graph

Refer Figure 2 in next page

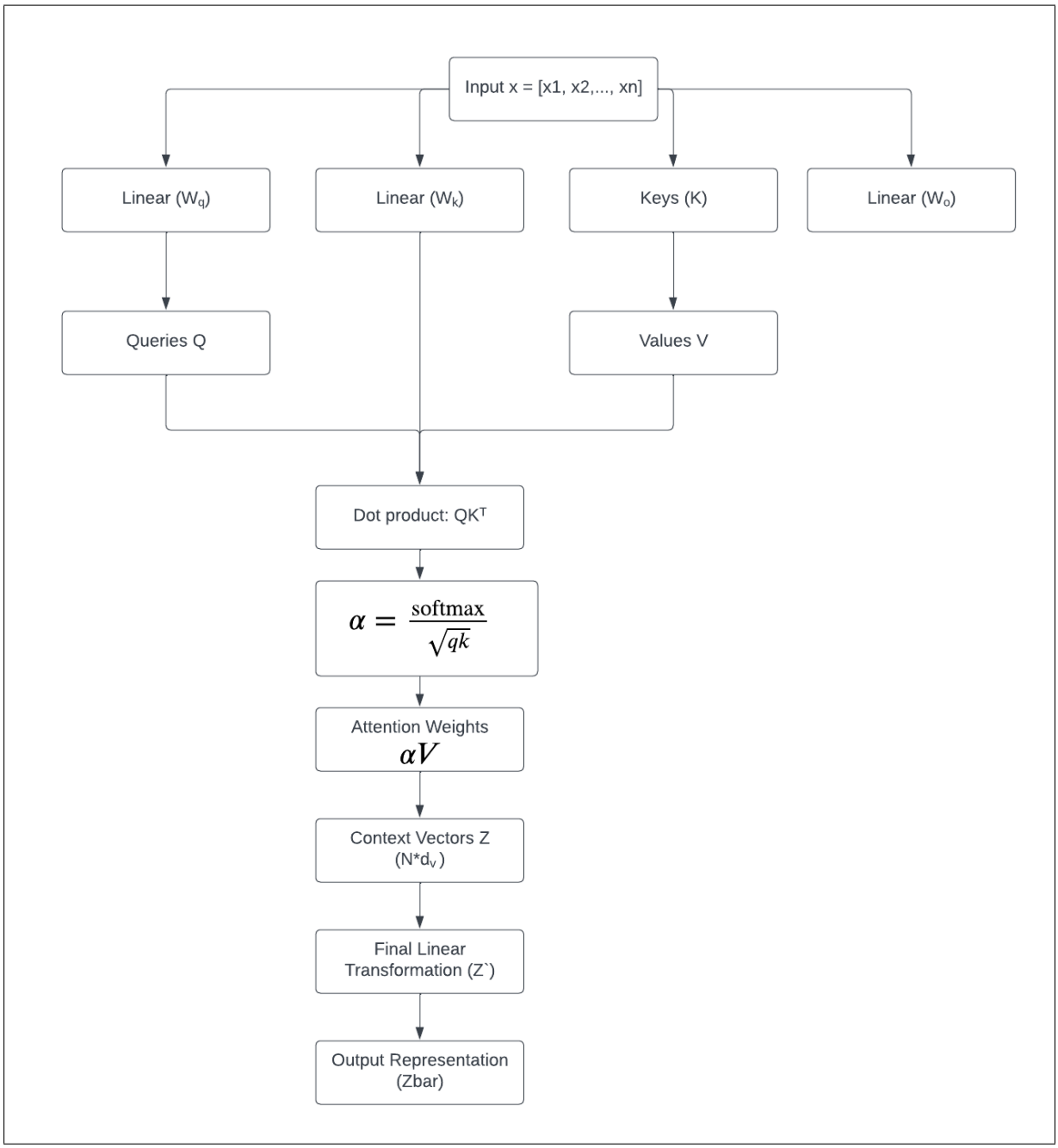

Figure 2: Flow of data through the self-attention mechanism

Team Member	Task#	Contribution (%)
sarojavu & sv94	Task 1	50% & 50%
sarojavu & sv94	Task 2	50% & 50%
sarojavu & sv94	Task 3	50% & 50%
sarojavu & sv94	Task 4	50% & 50%
sarojavu & sv94	Task 5	50% & 50%
sarojavu & sv94	**Total**	100% & 100%

Table 1: Contribution Table for Team Members sarojavu and sv94

References

- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. Chapter 14: Autoencoders.
- D. P. Kingma and M. Welling. "Auto-Encoding Variational Bayes." arXiv:1312.6114, 2013. `https://arxiv.org/abs/1312.6114`
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is All You Need." NeurIPS 2017. `https://arxiv.org/abs/1706.03762`
- Christopher Olah. "Illustrated Guide to Self-Attention." Distill, 2016. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`
- Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory." Neural Computation, 1997.
- PyTorch Official Documentation. `https://pytorch.org/docs/stable/index.html`
- Transformer tutorial with code examples: `https://kazemnejad.com/blog/transformer_architecture_positional_encoding/`