Pattern Is All You Need

Learning Atomic Properties from Aggregate Labels in Neural Networks

Submitted in partial fulfillment of the requirements of
BITS F421T Thesis

By

Kanishk Yadav

Under the supervision of

Prof. Oleksandr Voznyy
&
Prof. K. Sumithra

UNIVERSITY OF
TORONTO

December 2023

Acknowledgement

I extend my sincere gratitude to my supervisor, Prof. Oleksandr Voznyy, for his invaluable guidance and unwavering support throughout the course of this project. I consider myself fortunate to have had the opportunity to work under his mentorship, and I deeply appreciate his continuous encouragement, insight, and constructive feedback.

I also wish to express my heartfelt thanks to my co-supervisor, Prof. K. Sumithra, for her support and academic oversight during this project. Her involvement has contributed significantly to shaping the direction and depth of the work.

Finally, I am grateful to all members of the Clean Energy Lab at the University of Toronto for their collaborative spirit and insightful discussions. Working alongside such a dedicated and talented team has greatly enriched my research experience.

Certificate

This is to certify that the thesis entitled "Pattern Is All You Need" submitted by Kanishk Yadav, ID No 2019B2A71452H in partial fulfillment of the requirement of BITS F421T Thesis embodies the original work done by him under our supervision.

Prof. Oleksandr Voznyy	Prof. K Sumithra
Supervisor	Co-supervisor
University of Toronto	Birla Institute of Technology & Science
Date: 04-December-2023	Date: 04-December-2023

List of Abbreviations & Symbols

GNN: Graph Neural Network(s)

MSE: Mean Squared Error

DFT: Density Functional Theory

WNN: Wide Neural Network(s)

BCE: Binary Cross-Entropy

NN: Neural Network

LR: Learning Rate

ReLU: Rectified Linear Unit

ELU: Exponential Linear Unit

X: Input Feature Matrix

y: Target Output

AE: Autoencoder

Abstract

The objective of this project is to explore how machine learning models can recover individual atomic properties using only aggregate molecular information—such as the sum of atomic values—without direct atomic supervision. This setup serves as a simplified testbed for developing data-efficient neural architectures applicable to quantum chemical prediction tasks, where fully labeled data is often scarce or expensive to generate via Density Functional Theory (DFT).

We design and evaluate a series of summation-based learning tasks in which the goal is to infer real-valued atomic contributions from their groupwise sums. Atoms are represented using both binary and one-hot encodings, and models are trained across various data regimes and architectural configurations. Early models using position-fixed Siamese-style subnetworks exhibited systematic outlier behavior, which we traced to positional bias in the data flow. This motivated a redesign toward a position-invariant shared-weight architecture, which significantly improved generalization.

Our findings reveal that model performance is shaped not only by data quantity, but also by the structure and symmetry of the input and network. The results underscore the importance of permutation invariance and architectural consistency in learning atomic-scale properties from weak, aggregate supervision—offering insights into the design of scalable, interpretable models for computational materials science.

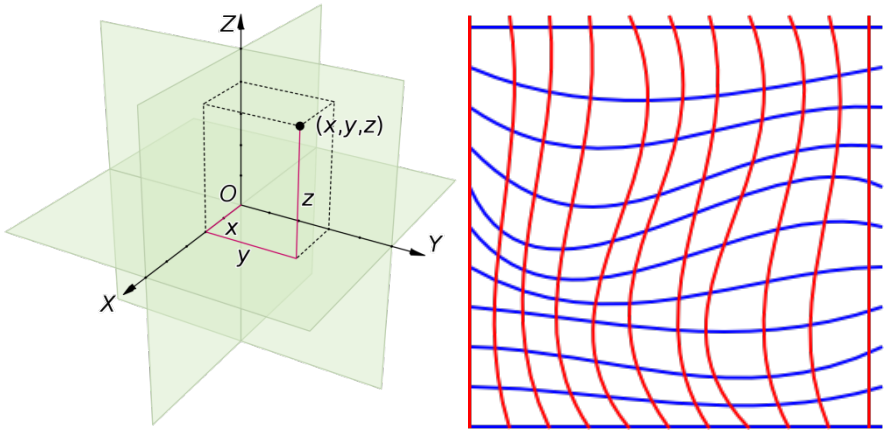
Chapter 1

Overview

In recent decades, the field of Deep Learning has revolutionized a wide range of machine learning tasks. Applications such as speech recognition, image classification, and video processing have seen significant advances due to deep learning. However, these tasks share a common characteristic: the data used by the machine learning algorithms is structured in a way that is compatible with conventional models.

Data can be geometrically represented in two primary forms:

- **Euclidean Space**: A geometric framework based on the postulates of the Greek mathematician Euclid, characterized by flatness—parallel lines never meet, and the angles of a triangle always sum to 180°.
- **Non-Euclidean Space**: A geometric framework that deviates from Euclid's postulates, where spaces can be curved, and the sum of triangle angles can differ from 180°.



While traditional deep learning models excel in processing data from Euclidean space, many real-world problems require operating in non-Euclidean spaces, such as **graphs**—data structures with complex relationships and interdependencies between elements.

In the same way that **words** are the basic units in Natural Language Processing, **atoms** serve as the fundamental units in chemistry. These atoms form larger

compounds, which can be naturally represented as graphs. However, representing such graphs in a machine-readable format is challenging due to the irregularity and complexity of chemical structures.

Graphs are inherently irregular: they consist of a variable number of nodes and neighbors, and the arrangement of these nodes is unordered. Each node (atom) is influenced by its neighbors through specific relationships (bonds), which determine the overall properties of the molecule. To analyze such data with machine learning, it is essential to convert graphs into numerical **embeddings** that preserve these structural dependencies.

Graph Neural Networks (GNNs) are a powerful solution to this problem. Unlike traditional neural networks, which assume that data points are independent and identically distributed (i.i.d.), GNNs explicitly exploit the relational structure of graph data. They work by aggregating information from neighboring nodes and updating each node's representation iteratively. As a result, GNNs are highly effective for tasks like node classification, graph classification, and link prediction, due to their ability to capture both local and global patterns in graphs.

Problem Statement

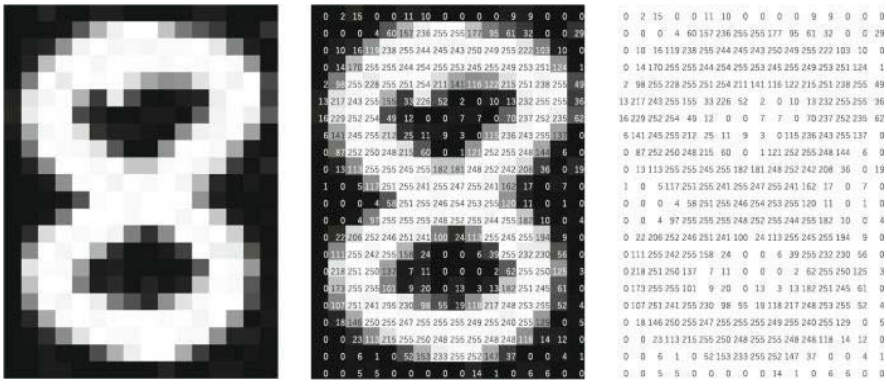
This thesis investigates a fundamental question: *Why is it difficult to predict the electronic structure properties of inorganic materials using Graph Neural Networks?* To approach this problem, we begin by constructing a simplified setting—gradually building our understanding of how GNNs operate and where they may fail.

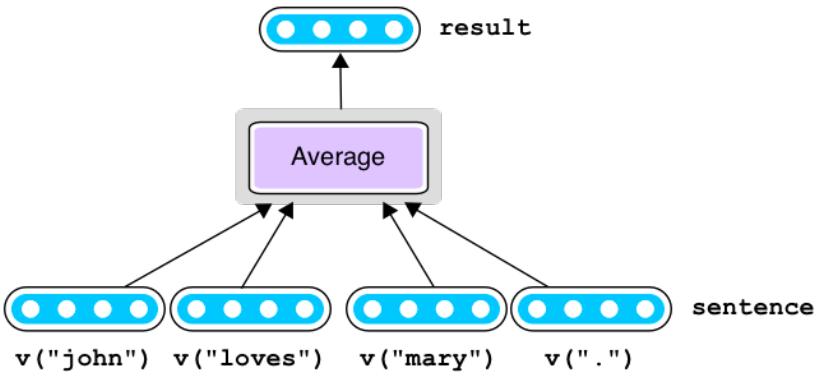
Modern deep learning frameworks are predominantly designed for data with fixed size and regular structure, such as sequences or grids. Convolutional Neural Networks work well on images, and recurrent or transformer-based models work well on sentences. These data types have an inherent ordering or spatial layout that the model can leverage.

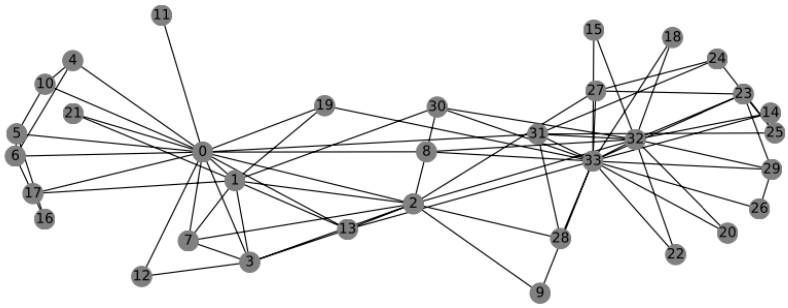
In contrast, graph data lacks any natural notion of order or fixed dimensionality. The topology of a graph can vary, and its nodes can be permuted arbitrarily without changing the underlying structure. For a model to work correctly with graph data, it must produce the same output regardless of the order in which nodes are presented.

Consider the following examples:

1. An *image*: structured as a grid with fixed pixel positions.
2. A *sentence*: a linear sequence of tokens, with a defined reading order.
3. A *graph*: an irregular structure, where the connectivity and ordering of nodes are not fixed.







Let us take a molecule like ethanol (C_2H_5OH). It can be represented as a graph in multiple ways depending on node ordering. Two examples:

1. **Graph A** (Node order: C, C, H, H, H, H, H, O, H):
    - Nodes: C1, C2, H1–H5, O, H6
    - Bonds: C1–C2, C1–H1/H2/H3, C2–H4/H5, C2–O, O–H6

	C1	C2	H1	H2	H3	H4	H5	O	H6
C1	[0,	1,	1,	1,	1,	0,	0,	0,	0]
C2	[1,	0,	0,	0,	0,	1,	1,	1,	0]
H1	[1,	0,	0,	0,	0,	0,	0,	0,	0]
H2	[1,	0,	0,	0,	0,	0,	0,	0,	0]
H3	[1,	0,	0,	0,	0,	0,	0,	0,	0]
H4	[0,	1,	0,	0,	0,	0,	0,	0,	0]
H5	[0,	1,	0,	0,	0,	0,	0,	0,	0]
O	[0,	1,	0,	0,	0,	0,	0,	0,	1]
H6	[0,	0,	0,	0,	0,	0,	0,	1,	0]

2. **Graph B** (Node order: H, C, O, H, C, H, H, H, H):
    - Nodes: H1, C1, O, H2, C2, H3–H6
    - Bonds: C1–C2, C1–H3/H4/H5, C2–H6/H1, C2–O, O–H2

	H1	C1	O	H2	C2	H3	H4	H5	H6
H1	[0,	0,	0,	0,	1,	0,	0,	0,	0]
C1	[0,	0,	0,	0,	1,	1,	1,	1,	0]
O	[0,	0,	0,	1,	1,	0,	0,	0,	0]
H2	[0,	0,	1,	0,	0,	0,	0,	0,	0]
C2	[1,	1,	1,	0,	0,	0,	0,	0,	1]
H3	[0,	1,	0,	0,	0,	0,	0,	0,	0]
H4	[0,	1,	0,	0,	0,	0,	0,	0,	0]
H5	[0,	1,	0,	0,	0,	0,	0,	0,	0]
H6	[0,	0,	0,	0,	1,	0,	0,	0,	0]

Both graphs represent the same molecule, but if a machine learning model is sensitive to node ordering, it may incorrectly assign different properties to each representation. This inconsistency would lead to unreliable predictions, especially during testing.

GNNs are specifically designed to address this **permutation invariance** issue and are the focus of this work.

Chapter 2

Graph Neural Networks

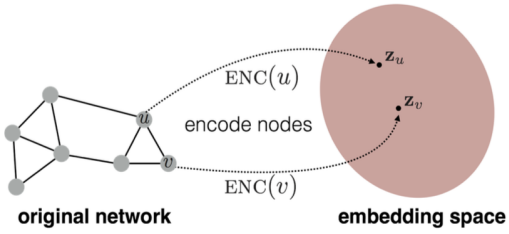
Having discussed the challenges posed by graph-structured data, we now turn to the question: How can we convert atoms in a molecule into numerical representations that can be effectively understood and processed by machine learning models?

The answer lies in the use of **embeddings**. Embeddings are a mathematical technique for representing any type of data—whether images, text, audio signals, or graphs—in a low-dimensional vector space that captures meaningful patterns and relationships. In the context of graphs, the objective is to map nodes to a low-dimensional embedding space where structurally or functionally similar nodes lie closer together. Techniques such as **node2vec** draw inspiration from **word2vec** in NLP, where words with similar meanings are placed near each other in vector space.

An **encoder** is a function that transforms each node in the original graph into its corresponding embedding vector (typically of dimension m). The full set of node embeddings defines the embedding space, typically represented as a matrix $\mathbb{Z}$ of dimension $\mathbb{N} \times \mathbb{M}$, where $\mathbb{N}$ is the number of nodes and $\mathbb{M}$ is the embedding dimension.

A **decoder** operates on this embedding space and attempts to reconstruct relationships between the nodes. For example, a decoder might predict the presence of an edge between two nodes based on the similarity of their embeddings or predict a graph-level property based on the aggregation of node features. The decoder's role is to translate the learned embeddings into useful predictions that align with the training data.

The objective of training a GNN model is to minimize a loss function that captures the difference between the model's predictions and the ground-truth labels. For graph property prediction, this often involves aggregating predictions across nodes or edges and comparing them to observed values. The model learns to produce embeddings that optimize this loss, thereby capturing the underlying structure and semantics of the graph data.



Chapter 3

Literature Review

Generalization in a Linear Perceptron in the Presence of Noise

One foundational study investigates the generalization capabilities of a basic **linear perceptron model**, particularly in the presence of noise. The central aim of the study is to evaluate how well such a model can extrapolate from a set of training examples to the broader space of possible inputs—especially when the training data is corrupted by noise.

The perceptron model in this study is trained using the **delta rule**, which adjusts weights in proportion to the error between predicted and actual outputs. The specific variant used is the **Adaline** (*Adaptive Linear Neuron*) learning rule, a classical method for linear function approximation.

A key metric introduced in the study is the **generalization error**, defined as the average squared error between the model's output and the target output across all possible inputs. This metric provides insight into how well the model has captured the underlying distribution rather than overfitting to the training examples.

The authors explore the effects of **static noise** (where the same corrupted input persists across iterations) and **dynamic noise** (where noise varies across batches or time). A striking result is the observation of a **dynamical phase transition**: below a critical number of training examples, the perceptron fails to generalize effectively, but performance improves dramatically beyond this threshold.

Another important phenomenon addressed is **overfitting**. In the presence of noise, especially static noise, the model may begin to memorize spurious patterns, leading to high error on unseen data. The authors demonstrate that introducing **weight decay**—a regularization term that penalizes large weight magnitudes—can mitigate overfitting and improve generalization.

Although the study focuses on a simple linear model, its insights are highly relevant for understanding more complex neural networks. In particular, the observed behaviors—phase transitions, overfitting, and the benefits of regularization—apply directly to situations involving deep learning under weak supervision or noisy labels.

This comprehensive investigation sheds light on the delicate balance between model complexity, data quantity, and noise, offering foundational lessons that inform the design and analysis of the models used in this thesis.

Chapter 4

The Summation Problem

4.1 Motivation and Objective

Our goal is to reverse-engineer the properties of atoms by studying the molecules they form. To put this into simpler terms, we want to learn the properties of individual atoms (as real numbers) based solely on the known properties of molecules (sum of those real numbers), without directly observing the atom values themselves.

To tackle this, we employ a deep learning–based approach capable of inferring the properties of individual atoms given the aggregate property of the molecule they comprise. For this approach to be effective, the model should not be biased by the number of times each atom appears during training. For instance, if hydrogen atoms occur more frequently than carbon atoms, the model may learn better representations for hydrogen, which would hinder the generalizability of the learned atom values.

4.2 Three-Atom Summation Model

We start with a simplified version of the problem involving the sum of **three atoms**. We define a vocabulary of **128 unique atoms**, each represented using a **7-bit binary encoding**. Each atom is mapped to a real number in the range [1, 10].

For example, the 7-bit vector:

0	0	0	1	1	1	0

represents atom index 14, mapped to a real value in that range.

Our representations range from 0 to 127 (in binary)

0	0	0	0	0	0	0

to

1	1	1	1	1	1	1

Since our task involves three atoms, the number of unique combinations is:

128×128×128 = 2,097,152


```
"0000000": 2.668487740156844,
"0000001": 4.32266942083329,
"0000010": 1.8406354393396818,
"0000011": 4.87595575521007,
"0000100": 9.568665102241914,
"0000101": 2.898534975065279,
"0000110": 3.2600029260703565,
"0000111": 7.166216717705712,
"0001000": 9.645308179248719,
"0001001": 8.372003638093208,
"0001010": 3.874747605689689,
"0001011": 5.644060764770714,
"0001100": 0.09086843775491404,
"0001101": 5.903909896208223,
"0001110": 0.8619501992432965,
"0001111": 5.887117550844233,

                .
                .
"1110110": 5.4338490565083655,
"1110111": 4.350527004876547,
"1111000": 0.9835983889179045,
"1111001": 7.761221611027187,
"1111010": 3.4010655205202447,
"1111011": 1.2530354022278278,
"1111100": 9.756865057071614,
"1111101": 2.804094771263501,
"1111110": 6.729376887981758,
"1111111": 3.907766568414861
```

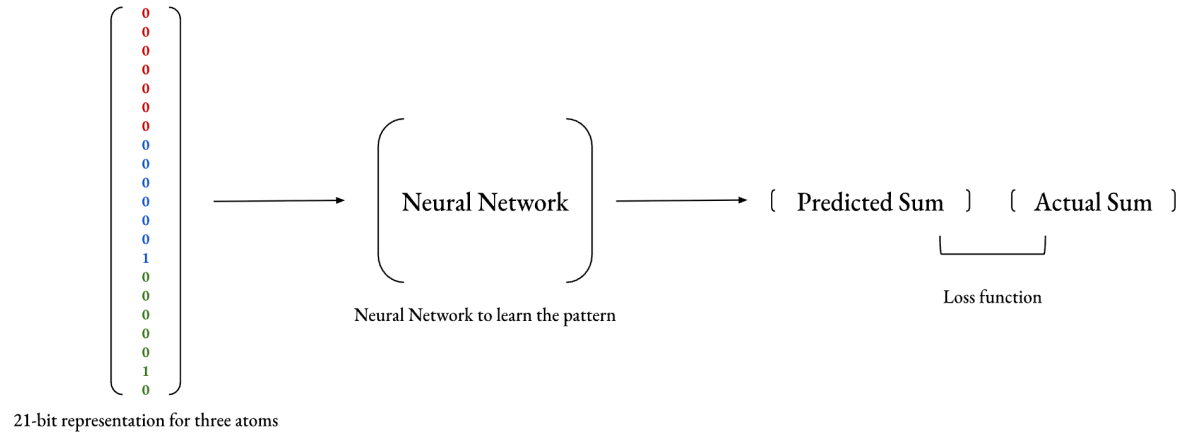

Each input to the model is a **21-bit vector** formed by concatenating three 7-bit encodings. The output is the scalar sum of the three atomic values.

To design our model for training, we first evaluate the sum of every possible combination of three atoms and store them. Consider a particular combination of three atoms from the mappings stated above (0000000, 0000001, 000010).

0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0

We map the value of this 21-bit representation to its corresponding sum i.e.,
2.668 + 4.322 + 1.840 = 8.83

We now design our model architecture,



4.3 Initial Model and Limitations

We designed a feedforward neural network to map 21-bit inputs to a scalar output. While the model showed good performance on test data, we realized it was learning **co-occurrence statistics** rather than true **atom-wise representations**. That is, it memorized frequent triplet patterns rather than learning each atom's contribution independently.

To fix this, we restructured the model. Instead of passing the entire 21-bit vector as one input, we passed each 7-bit atom separately into the **same neural network submodule** (i.e., using shared weights). The outputs were then aggregated (summed) to produce the final prediction.

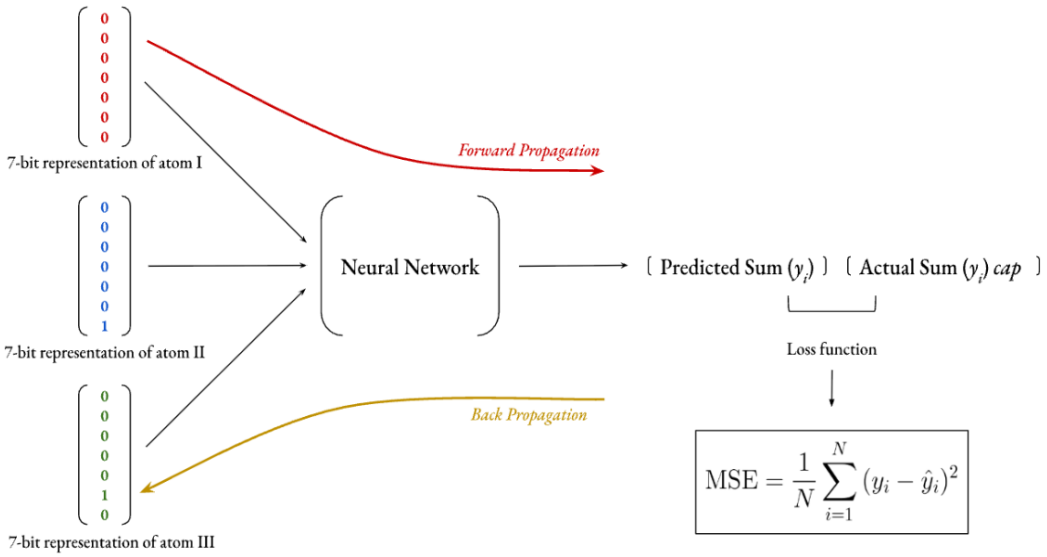


Figure 4.1: Diagram of shared-weight architecture for 3 atoms.

This enforces invariance to the position of atoms and encourages the model to learn consistent representations across different input contexts.

4.4 Transition to Five-Atom Problem

Once the 3-atom setup was validated, we moved on to a more computationally expensive version: the **five-atom summation task**. Although the full combination space is large (1285), we worked with a selected subset of **1255 combinations** for experimentation.

Our goal now was to explore two central questions:

1. How much training data is required for accurate predictions?
2. Does linear regression outperform neural networks in this weakly supervised setup?

4.5 Model Architecture and Variants

The five-atom model retained the shared-weight idea: each 7-bit atom encoding passed through the same neural network module, and the resulting outputs were summed to produce the total.

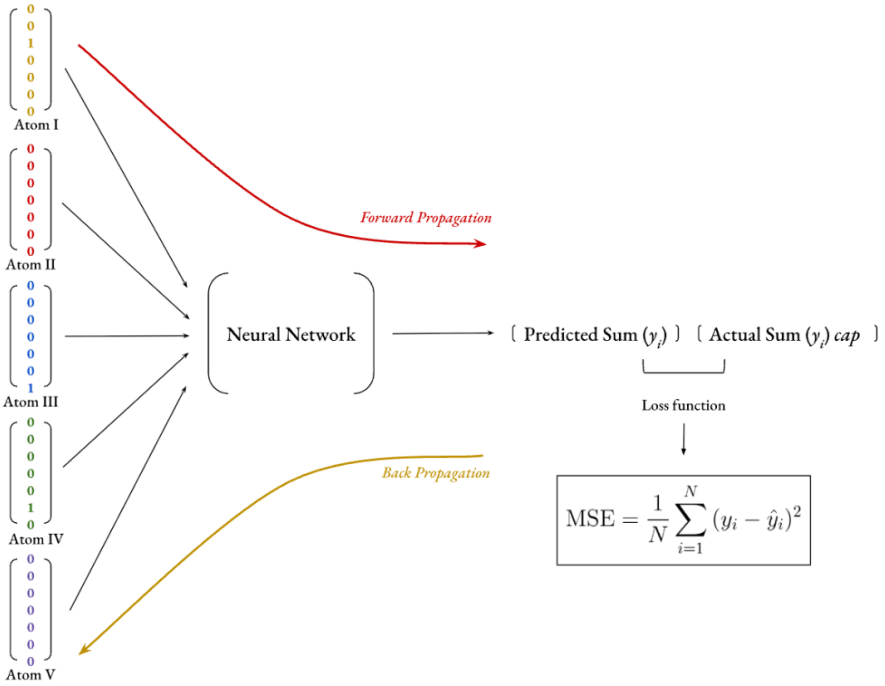


Figure 4.2: Five-atom architecture using different subnetwork for each atom.

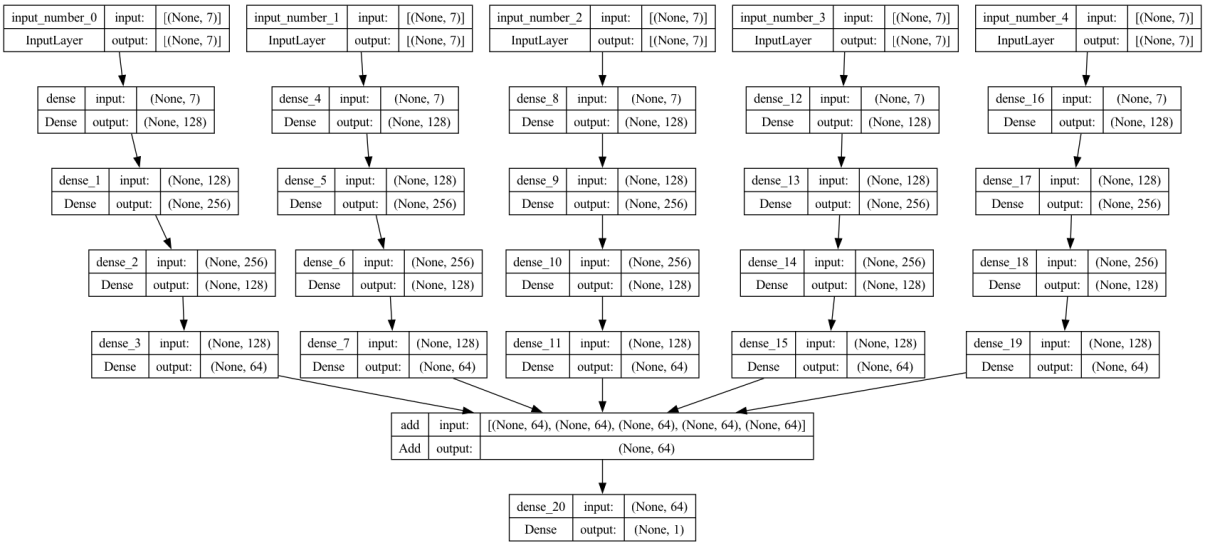


Figure 4.3: Detailed architecture using different subnetwork for each atom.

We tested the following model variants:

1. **Type 1**: 3 hidden layers (baseline)

```
def shared_subnetwork(input_tensor):
    x = Dense(128, activation='relu')(input_tensor)
    h1 = Dense(256, activation='relu')(x)
    h2 = Dense(128, activation='relu')(h1)
    h3 = Dense(64, activation='relu')(h2)
    return h3
```

2. **Type 2**: Type 1 + weight decay

```
def shared_subnetwork(input_tensor, decay=0.001):
    x = Dense(128, activation='relu', kernel_regularizer=l2(decay))(input_tensor)
    h1 = Dense(256, activation='relu', kernel_regularizer=l2(decay))(x)
    h2 = Dense(128, activation='relu', kernel_regularizer=l2(decay))(h1)
    h3 = Dense(64, activation='relu', kernel_regularizer=l2(decay))(h2)
    return h3
```

3. **Type 3**: Type 2 + dropout

```
def shared_subnetwork(input_tensor, decay=0.001, dropout_rate=0.1):
    x = Dense(256, activation='relu', kernel_regularizer=l2(decay))(input_tensor)
    x = Dropout(dropout_rate)(x)  # Dropout after first dense layer

    h1 = Dense(512, activation='relu', kernel_regularizer=l2(decay))(x)
    h1 = Dropout(dropout_rate)(h1)  # Dropout after second dense layer

    h2 = Dense(256, activation='relu', kernel_regularizer=l2(decay))(h1)
    h2 = Dropout(dropout_rate)(h2)  # Dropout after third dense layer

    h3 = Dense(128, activation='relu', kernel_regularizer=l2(decay))(h2)
    h3 = Dropout(dropout_rate)(h3)  # Dropout after fourth dense layer

    return h3
```

4.6 Model Analysis

To evaluate the performance of the five-atom neural network model, we performed multiple experiments by varying the number of training samples, epochs, and model configurations. The goal was to understand how model performance scales with training data and whether traditional regularization techniques can help the network generalize better under a weak supervision setup.

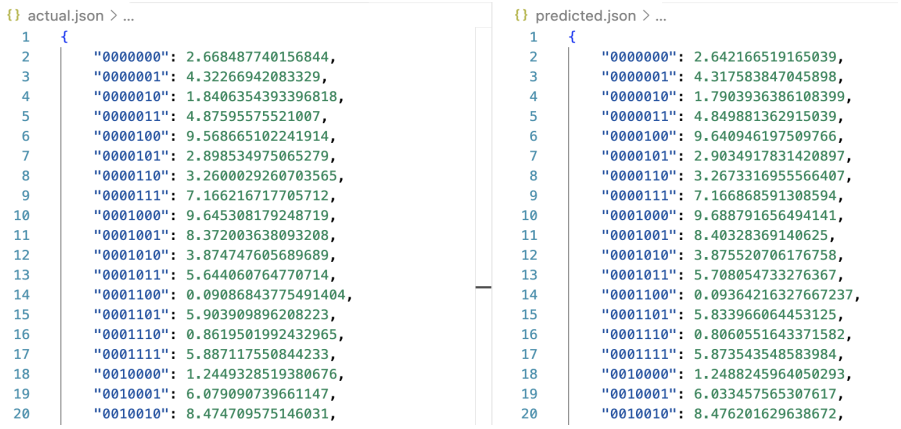
We experimented with the following configurations:

1. A 3-layer fully connected neural network.
2. Variations with added weight decay and dropout.
3. Different training data sizes and learning durations.

The results for **six** different training runs are summarized below:

Run Index	Epochs	Total Samples	Test MSE
1	30	10^6	2.237×10^{-11}
2	200	10^5	9.936×10^{-11}
3	1.7×10^3	10^4	1.853×10^{-9}
4	4×10^3	5×10^3	4.869×10^{-7}
5	6×10^3	3×10^3	1.89×10^{-4}
6	10×10^3	10^3	4.102

We perform a Train-Validation-Test split of 60-20-20 (learning rate: 10^{-5}) and after training the models, our testing looks something as shown below. '*actual.json*' file contains the original target mappings of each atom we wish to achieve and '*predicted.json*' contains the values the model predicted after training.



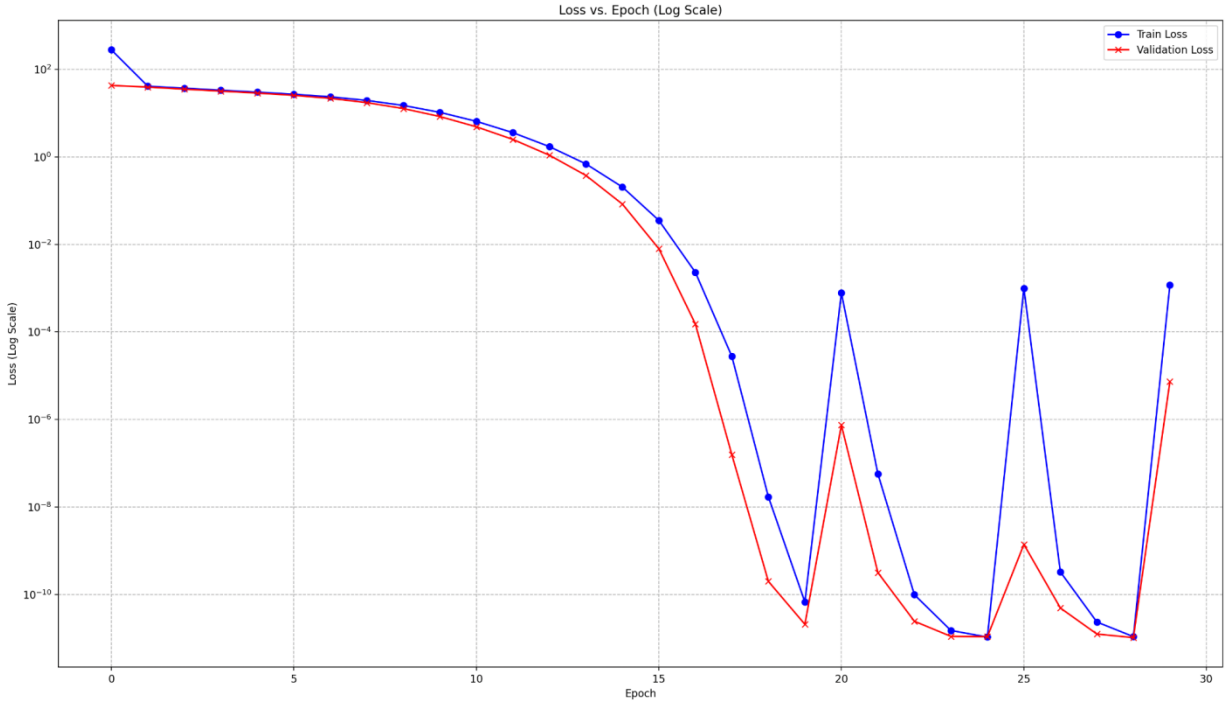
actual.json > ...

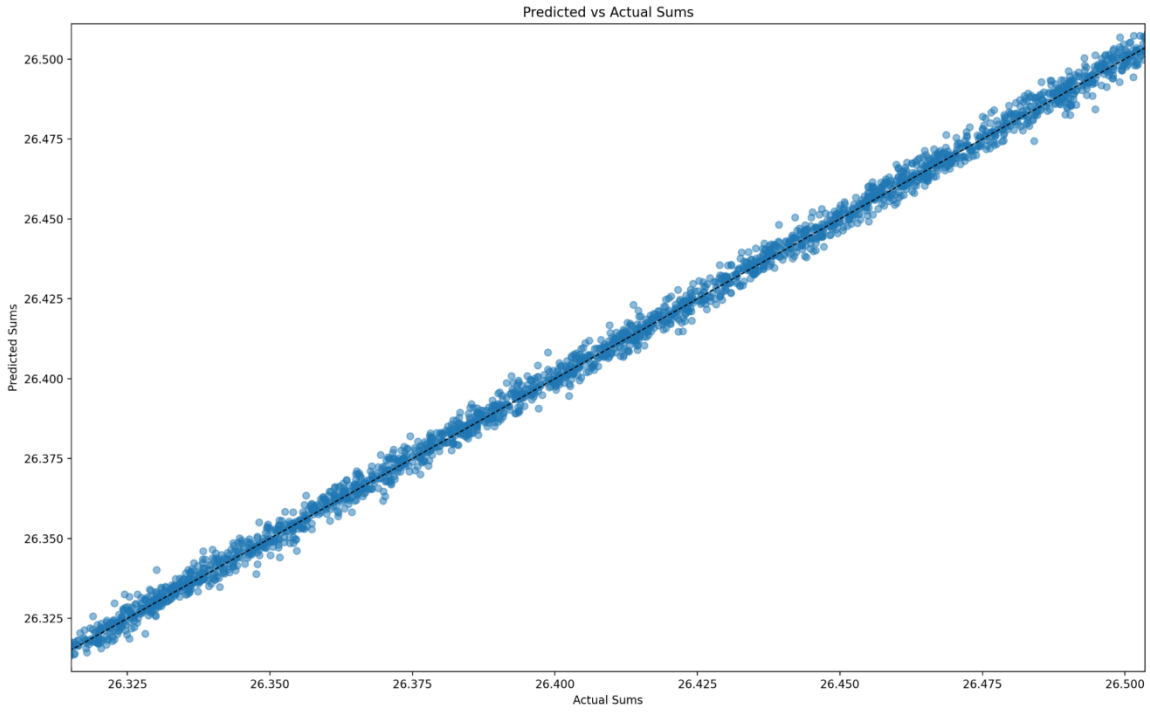
```
{
    "0000000": 2.668487740156844,
    "0000001": 4.32266942083329,
    "0000010": 1.8406354393396818,
    "0000011": 4.87595575521007,
    "0000100": 9.568665102241914,
    "0000101": 2.898534975065279,
    "0000110": 3.2600029260703565,
    "0000111": 7.166216717705712,
    "0001000": 9.645308179248719,
    "0001001": 8.372003638093208,
    "0001010": 3.874747605689689,
    "0001011": 5.644060764770714,
    "0001100": 0.09086843775491404,
    "0001101": 5.903909896208223,
    "0001110": 0.8619501992432965,
    "0001111": 5.887117550844233,
    "0010000": 1.2449328519380676,
    "0010001": 6.079090739661147,
    "0010010": 8.474709575146031,
```

predicted.json > ...

```
{
    "0000000": 2.642166519165039,
    "0000001": 4.317583847045898,
    "0000010": 1.7903936386108399,
    "0000011": 4.849881362915039,
    "0000100": 9.640946197509766,
    "0000101": 2.9034917831420897,
    "0000110": 3.2673316955566407,
    "0000111": 7.166868591308594,
    "0001000": 9.688791656494141,
    "0001001": 8.40328369140625,
    "0001010": 3.875520706176758,
    "0001011": 5.708054733276367,
    "0001100": 0.09364216327667237,
    "0001101": 5.833966064453125,
    "0001110": 0.8060551643371582,
    "0001111": 5.873543548583984,
    "0010000": 1.2488245964050293,
    "0010001": 6.033457565307617,
    "0010010": 8.476201629638672,
```

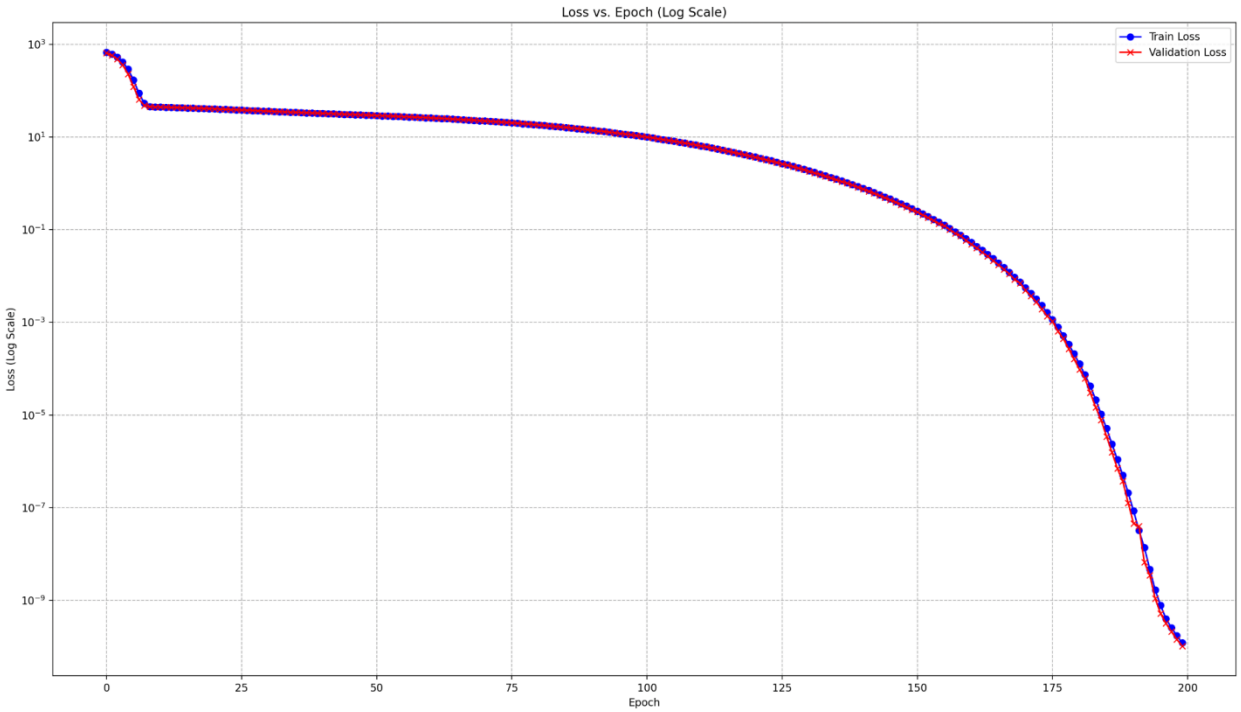
Run 1

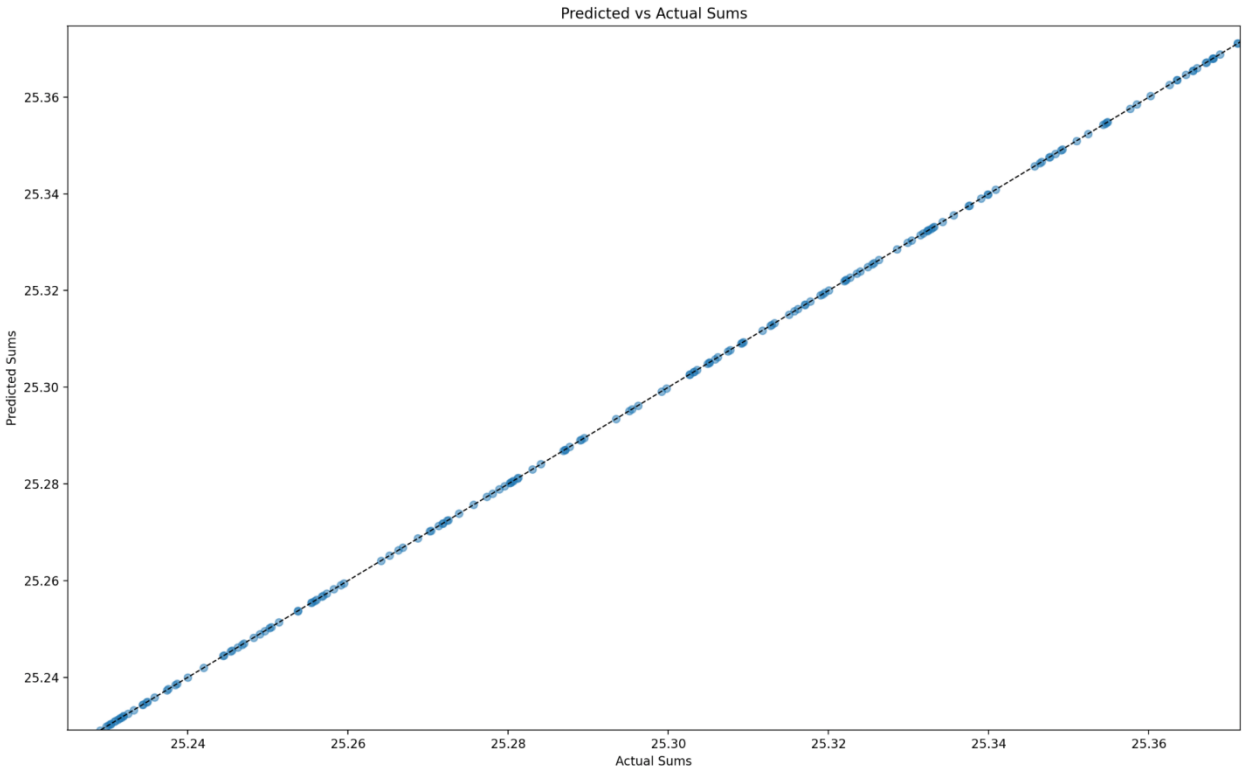




Conclusions: The model began to overfit before the 20th epoch. Although initial training loss decreased rapidly, validation loss plateaued early.

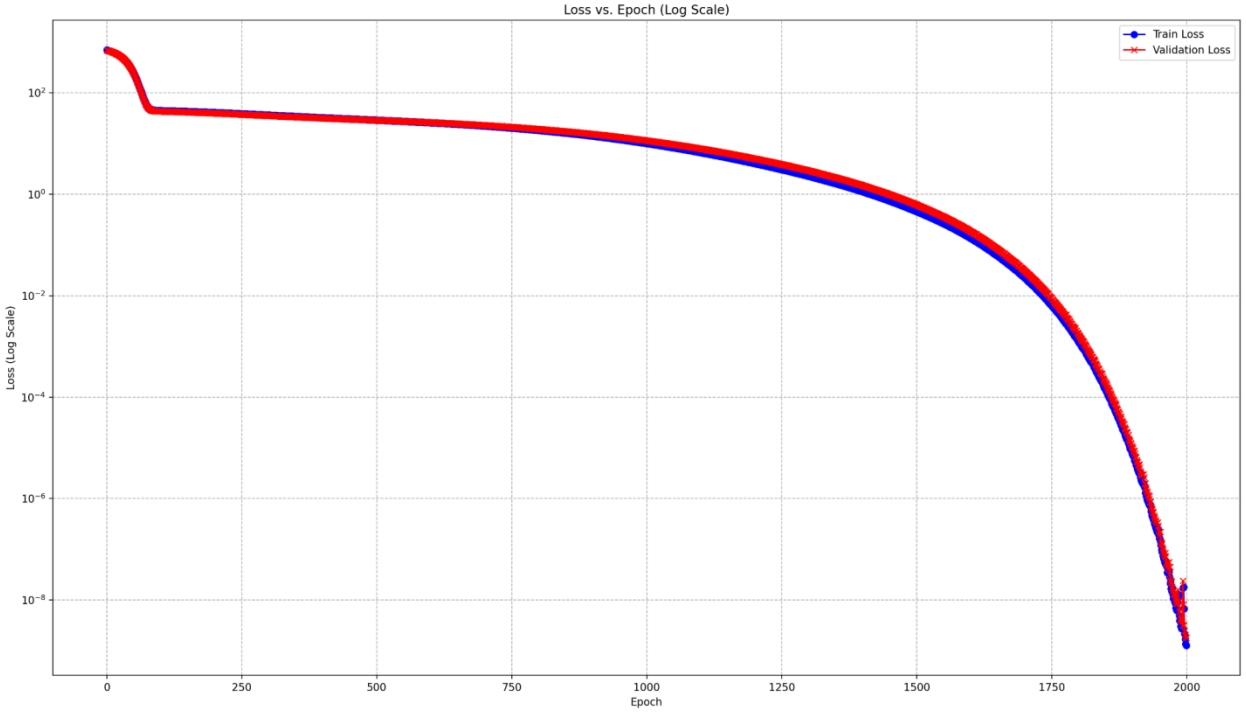
Run 2

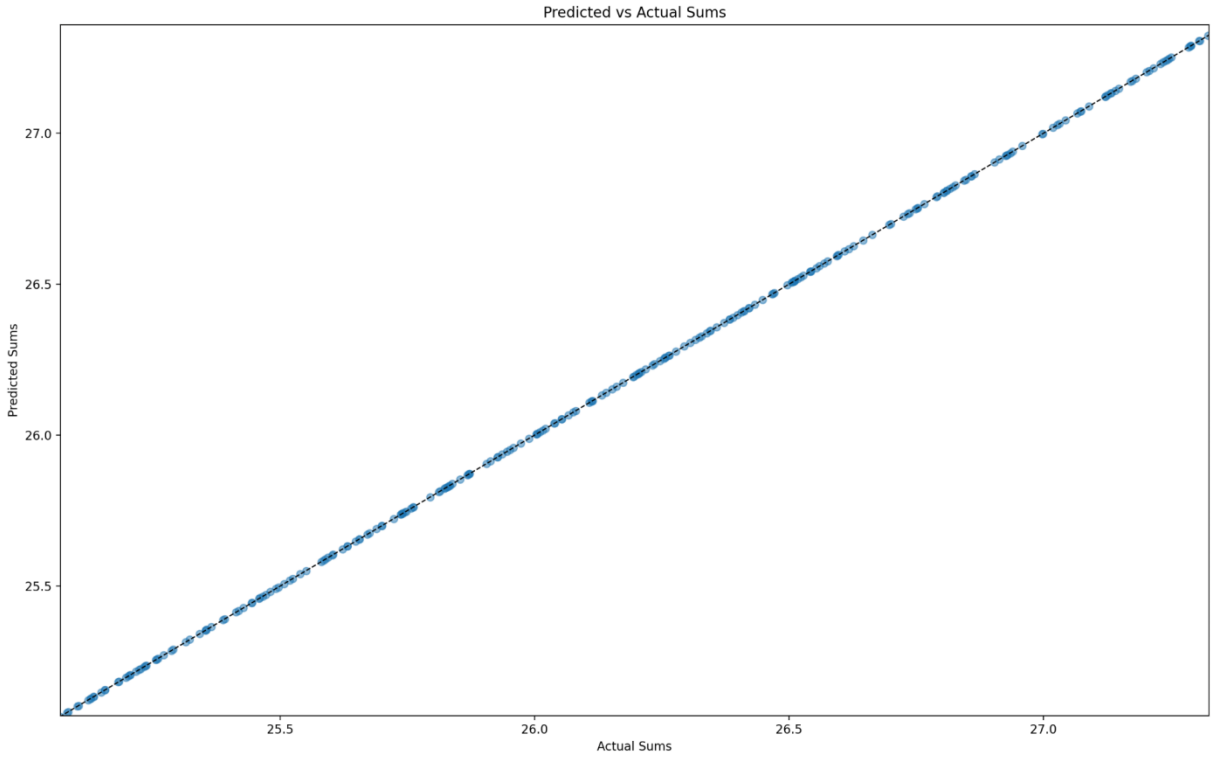




Conclusions: The model performed slightly better than Run 1 with longer training, but still overfitted after a certain point, despite a lower test MSE.

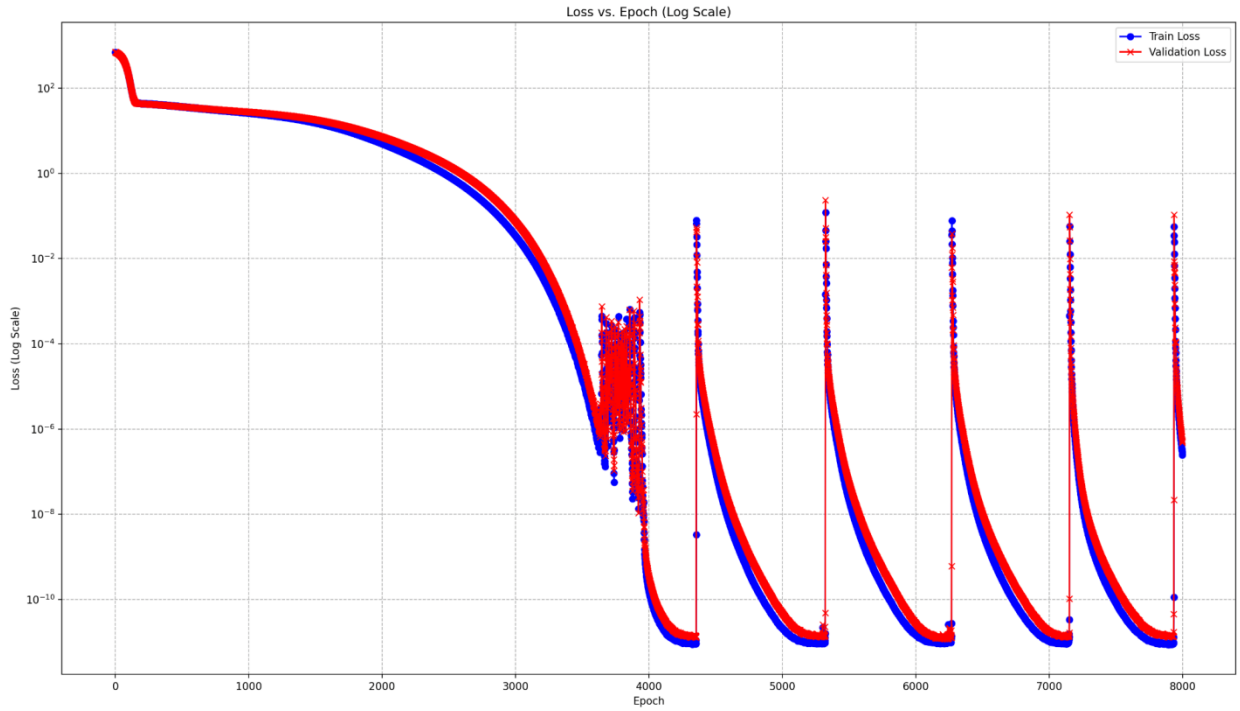
Run 3

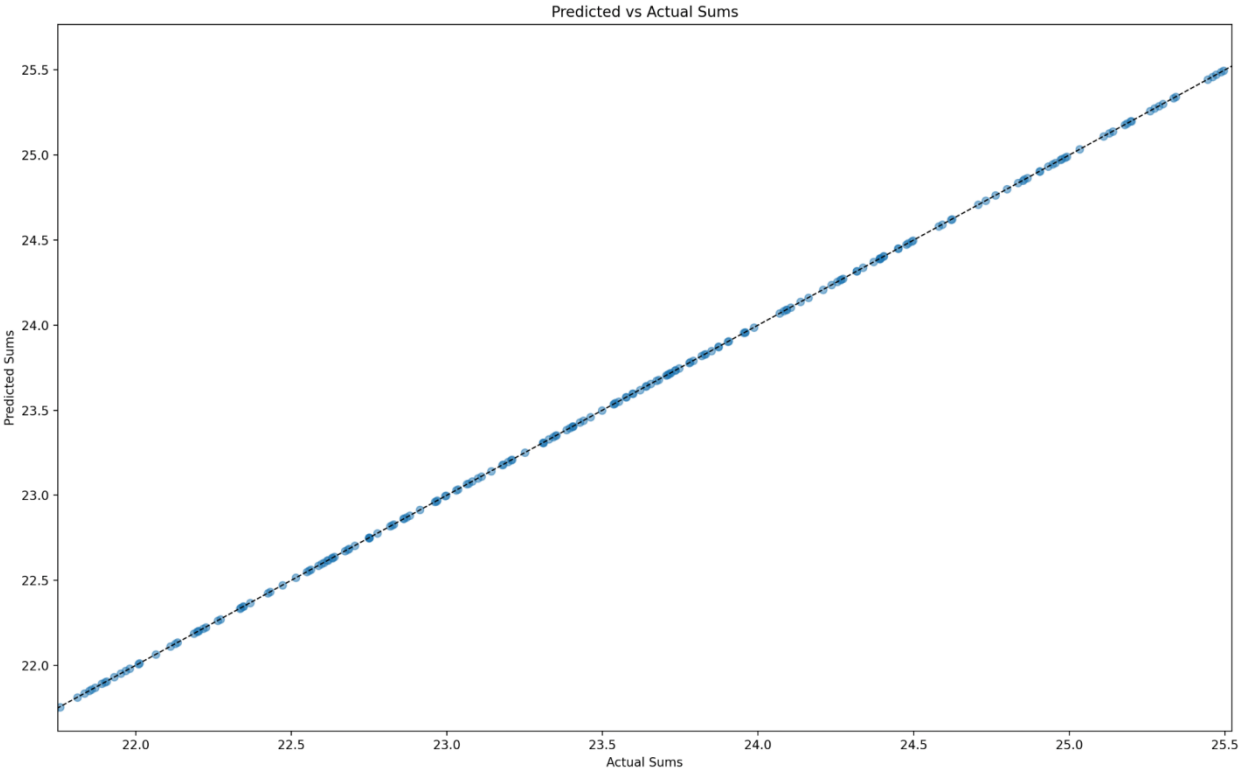




Conclusions: Test MSE was higher than Run 2, suggesting that increasing epochs alone does not improve performance with very small training sets.

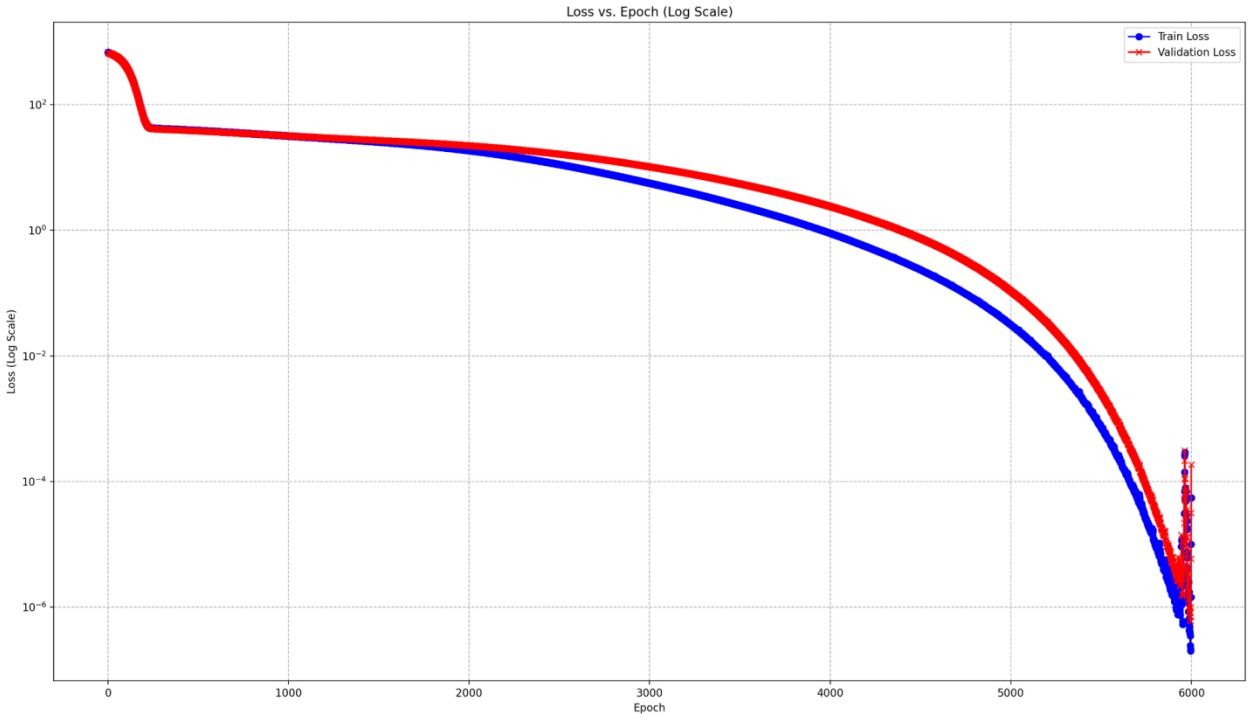
Run 4

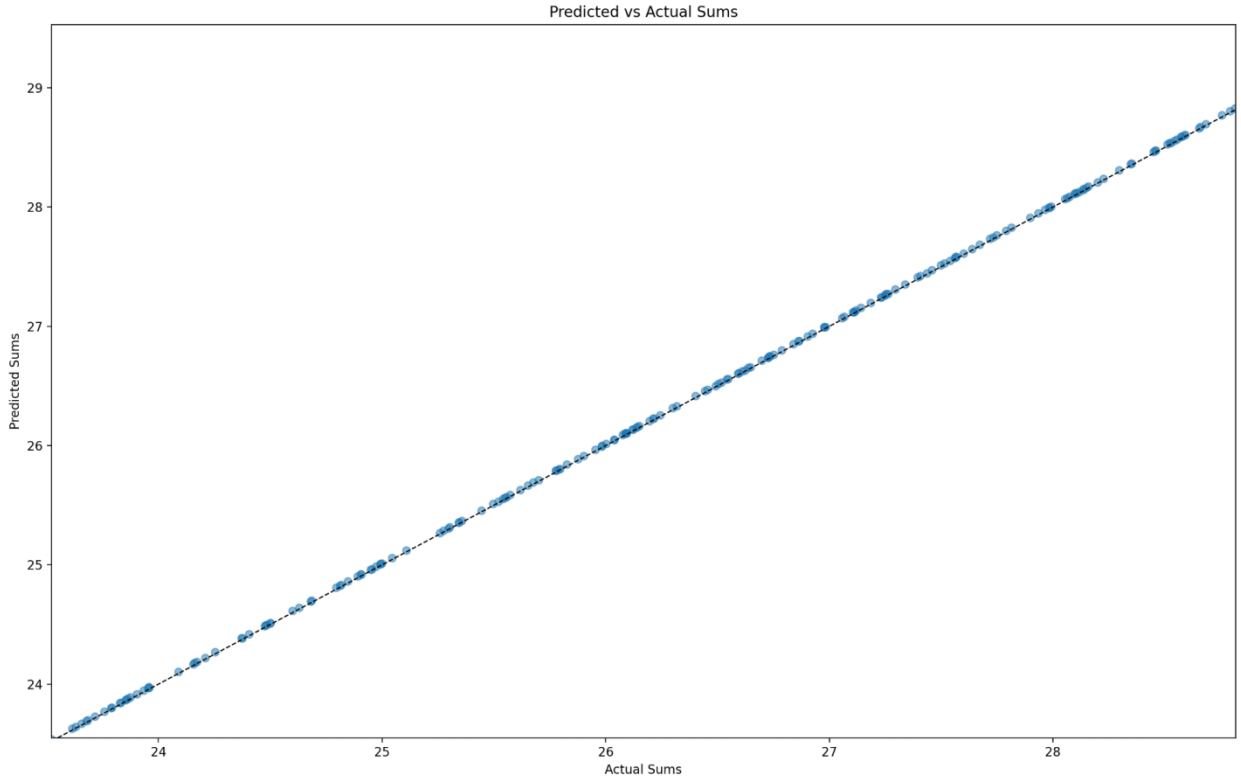




Conclusions: Significantly more training data improved generalization. Test MSE dropped to 4.869×10^{-7}.

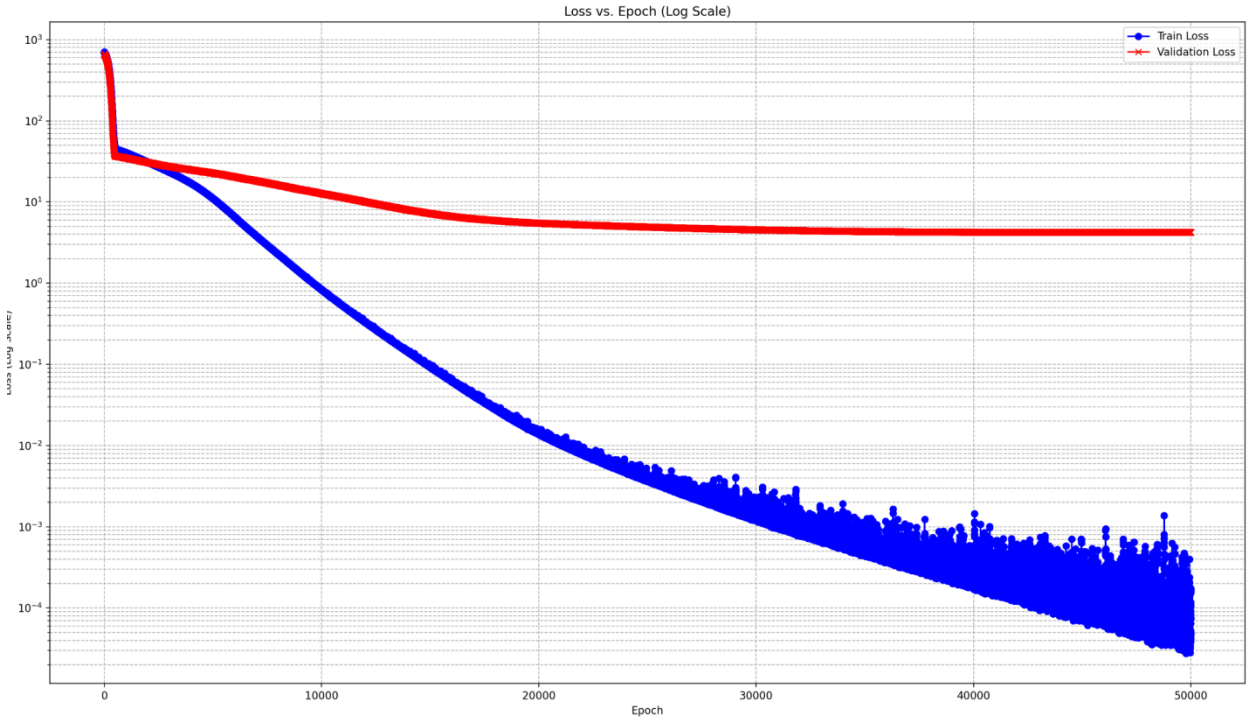
Run 5

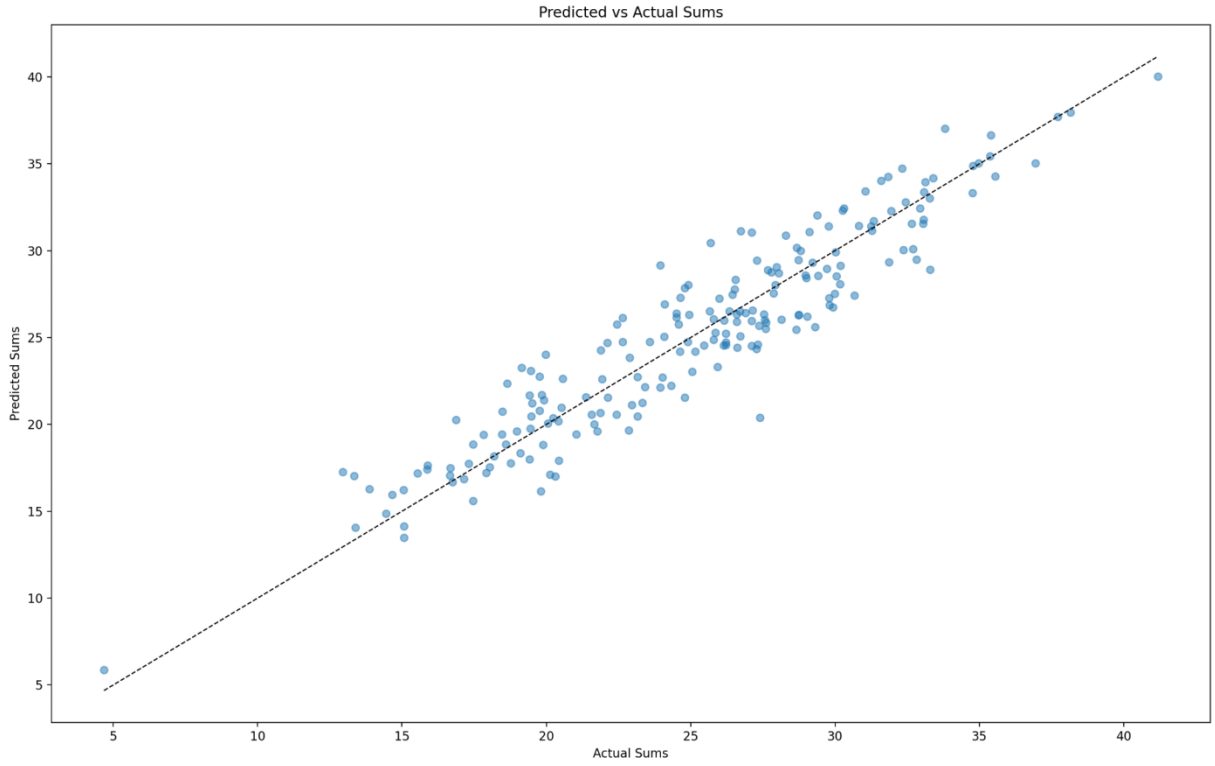




Conclusions: Model continued to improve, achieving a test MSE of 1.89×10^{-4}. However, performance was still inferior to simpler models, prompting exploration of linear regression.

Run 6





Conclusions: Despite long training and use of dropout and weight decay, the model failed to generalize, resulting in a test MSE of 4.102, several orders of magnitude higher than earlier runs.

Observations across runs:

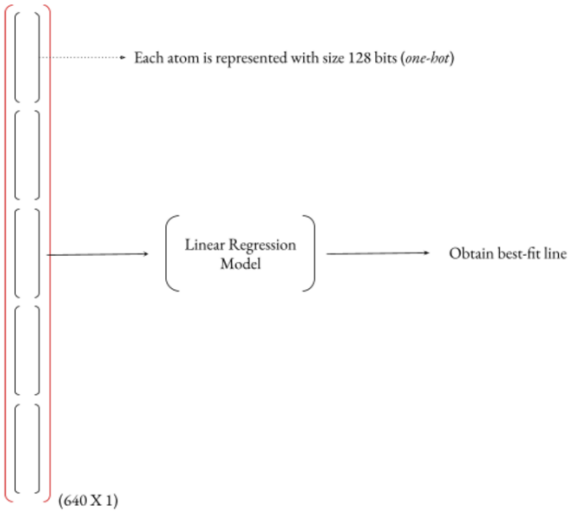
1. Neural networks are highly data-dependent in this setting.
2. The architecture helps generalization but only when trained on at least tens of thousands of examples.
3. Runs 5 and 6 confirm the boundary of the low-data regime, where even long training and regularization cannot rescue model performance.
4. These results led us to investigate whether simpler models like linear regression could succeed with better input representation.

4.7 Linear Regression Analysis

Following the declining performance of neural networks in **low-data settings**, especially in Runs 5 and 6, we turned to **linear regression** to test whether a simpler, interpretable model could learn atomic properties more efficiently—provided it was given the right input representation.

7-bit representations are not useful with linear regression. This is because, for linear regression to work, there should be a linear relationship between input and output. Consider the 2-atom problem, where two atoms (each 7-bit) are represented using a long 14-bit sequence meaning that each bit is treated as an independent feature, but the mapping from the 7-bit sequence to its decimal representation is non-linear.

Therefore, each atom is represented using 128-bit one hot encoding. For instance, atom 34 will have its 34th bit labeled as 1 and all other 127 bits labeled as 0. Our pipeline now looks something like:



Results and Predictions

Linear regression is easily able to extract the relationship between the atoms and their sums and predict the real values associated with each individual atom accurately with more training data (800 train, 200 test), which even neural networks (with longer training and lower batch size) are unable to do.

Test MSE: 1.218 x 10^{-27}

Test Sample 1:

One-hot vectors for the 5 numbers:

Number 1: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **1** 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Number 2: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 **1** 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Number 3: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **1** 0 0 0 0]

Number 4: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **1**
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Number 5: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **1** 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Actual Value: 19.28028
Predicted Value: 19.280279999999962

Test Sample 2:

One-hot vectors for the 5 numbers:

Number 1: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **1** 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Number 2: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **1** 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Number 3: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **1** 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Number 4: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **1** 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Number 5: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **1** 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Actual Value: 25.7471
Predicted Value: 25.7471

4.8 Summary

1. Linear regression with one-hot encoding outperformed neural networks in the low-data regime.
2. Binary encoding failed due to non-linearity.
3. When input is clean and separable, simpler models generalize better under data constraints.

These insights informed later architectural choices—including the importance of input structure, invariance, and representational clarity.

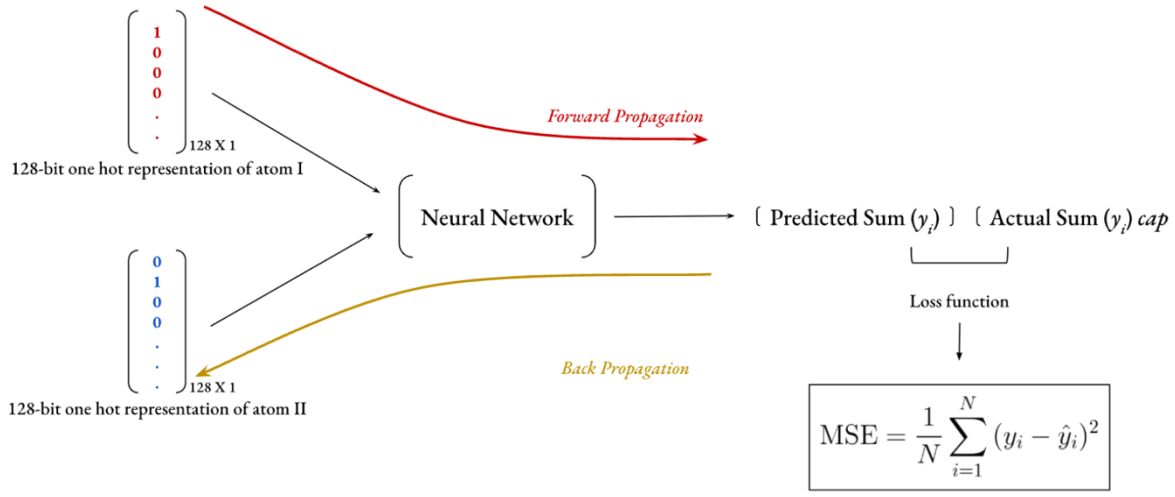
Chapter 5

Positional Bias and Outliers

In computer science, deep learning has been successful due to the abundance of data available for training machine learning models. In problems where abundant data is not available, one can easily generate artificial data and proceed but how does one generate artificial molecules or crystals that do not even exist?

5.1 Outlier Emergence

After observing stable generalization in high and mid-data regimes, we transitioned into a deliberately constrained setup using **just 1000 training samples**—a fraction of the total input space for the 5-sum task. By this stage, we had already shifted from 7-bit binary encodings to **128-bit one-hot representations** for each atom, eliminating representation-related nonlinearity.



We begin our analysis by exploring the 70-30 split and various batch sizes.

Model	Model Description	Split (Train - Test)	Batch size	MSE
1	5-atom	70 - 30	16	0.10250
2	5-atom	70 - 30	4	0.10248

Despite this, we observed a puzzling phenomenon: **a small number of atoms consistently showed large prediction errors**, even though the overall test MSE remained low. These **outliers** were present in the training set, some appearing multiple times, and the rest of the model was performing well.

This raised deeper questions:

1. Why was the model failing to learn correct values for just a few atoms?
2. Could the issue still be due to insufficient frequency of certain atoms?

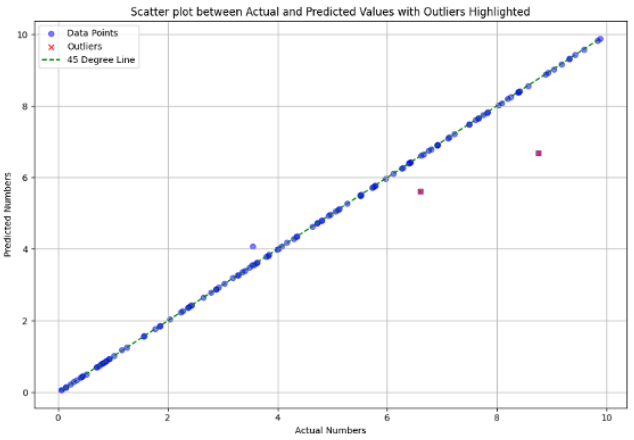
Our initial assumption was that these atoms were underrepresented in the dataset. We tested this by **artificially boosting their training frequency**, and while this did temporarily improve performance, the issue persisted in certain testing scenarios—hinting that the root cause was something else.

This marked the beginning of our investigation into what ultimately turned out to be a **positional bias**—a structural flaw in the way training data was generated, rather than a model or representation issue.

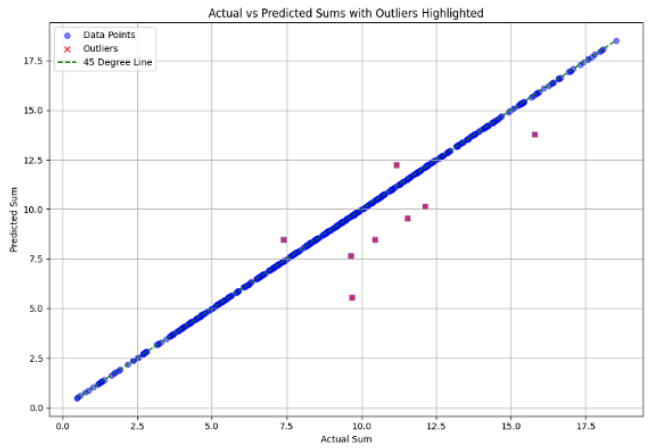
5.2 Initial Hypothesis: Frequency Bias

Our first explanation was straightforward: perhaps the model was underperforming for certain atoms simply because it had seen them **fewer times** during training. In a dataset of only 1000 combinations, the chance that every atom appears uniformly across positions is low.

To test this hypothesis, we **artificially increased the frequency** of the outlier atoms in the training dataset. Their representations were sampled more often, while all other aspects of the data remained unchanged.



Outliers	Predicted	Train	Test
6.60713	5.641942501	5	9
8.75369	6.709037304	4	5
3.54897	4.102982044	7	7



Actual Sum	Actual Composition	Predicted Sum
10.44849	6.60713,3.84136	8.464169
7.39033	3.84136,3.54897	8.463253
12.13113	6.60713,5.524	10.134339
9.68994	0.93625,8.75369	5.558916
9.64502	6.60713,3.03789	7.6554832
11.15857	7.6096,3.54897	12.23126
11.53182	6.60713,4.92469	9.568447
15.78911	6.60713,9.18198	13.793084

Figure 5.1: Frequency histogram of atoms before boosting the outliers.

This intervention worked—at least temporarily. The model's prediction accuracy for the previously failing atoms improved, and their outlier behavior seemed to disappear. At the time, we interpreted this as evidence that the network was learning **frequency-weighted atom embeddings**, i.e., the more often an atom appeared during training, the better the model could learn its contribution.

On increasing the number of occurrences of the outliers in training, we observe that [6.60713] and its corresponding sums (sums that 6.60713 forms) disappear from the predictions during testing which raises the question, "*Do some numbers require more representation in training than others?*"

However, this did not explain why **only a small, specific subset of atoms** failed, even though **others with similar frequencies** were predicted accurately. It suggested that frequency might not be the full story.

5.3 Discovery of Positional Dependence

Further investigation into the structure of the training data revealed a more subtle but critical issue: the **positional placement** of atoms was not random. Specifically, the outlier atoms had only appeared in **one fixed position** throughout the entire training dataset.

1. Atom **6.60713** always occurred in the **second position**.
2. Atom **8.75369** appeared only in the **first position**.
3. Atom **3.54897** was always found in the **first position**.

In contrast, during testing, atoms could appear in **any position** in the summation tuple. This meant that although the atom was present in the training data, the model had **never seen it in any other position** and thus failed to learn a general representation that worked across contexts.

The model had implicitly learned **position-specific representations**, because it never had the opportunity to disentangle identity from location. This revealed that the root issue was not about frequency or architecture — it was about **dataset bias**.

Actual Sum	Actual Composition	Predicted Sum
10.44237	3.83524,**6.60713**	10.460015
10.1561	3.54897,**6.60713**	10.159552
12.11718	5.51005,**6.60713**	12.1122465
12.58256	5.97543,**6.60713**	12.60082
11.87097	5.26384,**6.60713**	11.873709
13.88699	**8.75369**,5.1333	13.9088335
9.77621	**8.75369**,1.02252	9.79257
17.16891	**8.75369**,8.41522	17.184547
9.16476	**8.75369**,0.41107	9.136297
7.03821	**3.54897**,3.48924	7.049985
10.16378	**3.54897**,6.61481	10.172703
10.1561	**3.54897**,6.60713	10.159552
4.38025	**3.54897**,0.83128	4.385064
11.9478	**3.54897**,8.39883	11.970627
8.47366	**3.54897**,4.92469	8.474126
3.76994	**3.54897**,0.22097	3.7793763
TRAINING		

Outliers	Predicted	Train	Test
6.60713	5.444995403	5	9
8.75369	6.589535236	4	5
3.54897	3.931107998	7	7

Actual Sum	Actual Composition	Predicted Sum
12.71769	6.11056,**6.60713**	12.740326
10.44849	**6.60713**,3.84136	8.109748
12.13113	**6.60713**,5.524	9.8544655
10.90429	4.29716,**6.60713**	10.947264
9.49163	2.8845,**6.60713**	9.489421
9.64502	**6.60713**,3.03789	8.148861
11.53182	**6.60713**,4.92469	9.20209
10.61958	4.01245,**6.60713**	10.483174
15.78911	**6.60713**,9.18198	13.484142
17.32052	**8.75369**,8.56683	17.31808
9.68994	0.93625,**8.75369**	5.374362
18.07626	**8.75369**,9.32257	18.069529
11.12177	**8.75369**,2.36808	11.128768
16.36329	**8.75369**,7.6096	16.362125
3.98323	**3.54897**,0.43426	4.003206
6.58686	**3.54897**,3.03789	6.6070037
7.34205	**3.54897**,3.79308	7.355052
3.68883	**3.54897**,0.13986	3.718056
6.9345407	3.84136,**3.54897**	7.39033
10.68102	**3.54897**,7.13205	10.71435
11.15857	7.6096,**3.54897**	11.872341
TESTING - I		

Table 5.1: Atom ID vs. position in training and test, with prediction error.

5.3 Discovery of Architectural Position Dependence

Upon reviewing the model architecture more carefully, we realized the true problem: although the model was visually symmetrical, it was **not weight-shared**.

Each input atom—based on its position in the summation—was passed through a **separate, independently parameterized subnetwork**. While each subnetwork had the same layer structure (e.g., Dense (128) → Dense (256) → Dense (256) → Dense (64)), their weights were **distinct**.

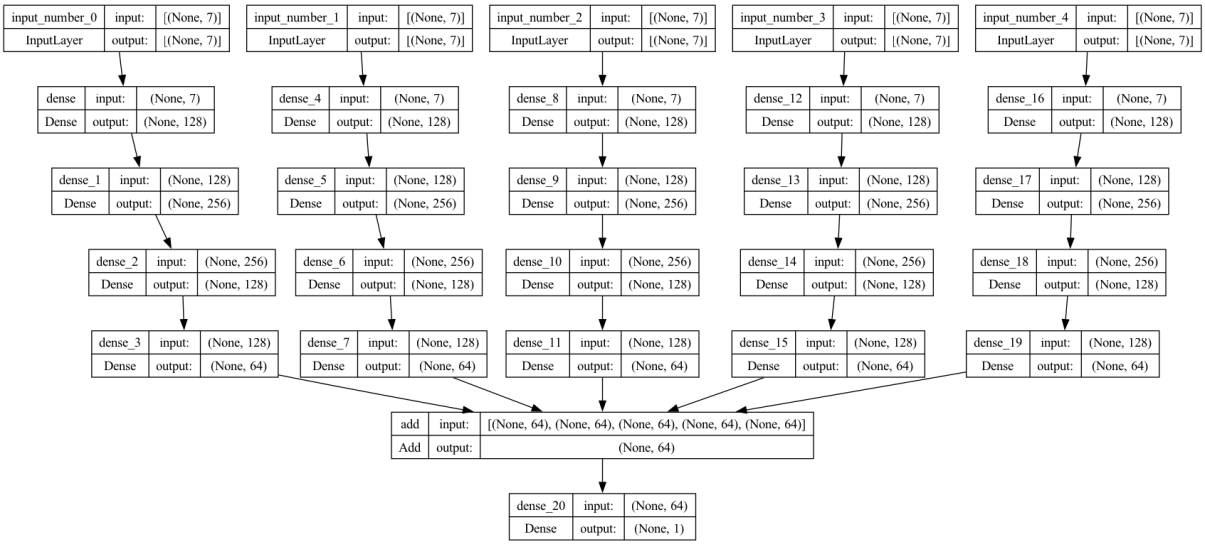


Figure 5.2: Model diagram showing separate subnetworks per atom position (for 7-bit input, similar for 128-bit input)

This meant the model learned a function of the form:

$$\text{Sum} = f_1(x_1) + f_2(x_2) + f_3(x_3) + f_4(x_4) + f_5(x_5)$$

where *f1, f2, f3* are **different functions**, one per atom position.

Because of this design:

1. Atom A at position 2 was learned by *f2*,
2. But if Atom A appeared at position 4 during testing, it was passed through *f4* — which had **never seen** Atom A during training.

Hence, the model learned **position-dependent behaviors**, not atom-specific embeddings.

5.4 Controlled Intervention: Positional Shuffling

To address this, we implemented a simple but critical correction: **shuffling the positions** of atoms in each tuple during training (adding these tuples to the dataset). This ensured that every atom was exposed across multiple positions allowing each subnetwork to learn its representation

After training on this shuffled dataset:

1. The model correctly predicted all atom values, including those that were previously outliers.
2. Prediction performance improved uniformly across all atom types.
3. Overall test MSE decreased, and **outlier behavior disappeared entirely**.

5.5 Fixing the Problem: Weight Sharing

We redesigned the architecture so that all atoms, regardless of their position, were passed through a **shared neural subnetwork** f_s, producing embeddings in the same latent space. These embeddings were then **summed** before the final regression layer.

This new formulation:

$$\text{Sum} = f_s(x_1) + f_s(x_2) + f_s(x_3) + f_s(x_4) + f_s(x_5)$$

ensured that:

1. All atoms were processed identically (f_s *is shared f(x)*),
2. The network learned a **position-invariant atomic mapping**,
3. And generalization improved across all test scenarios.

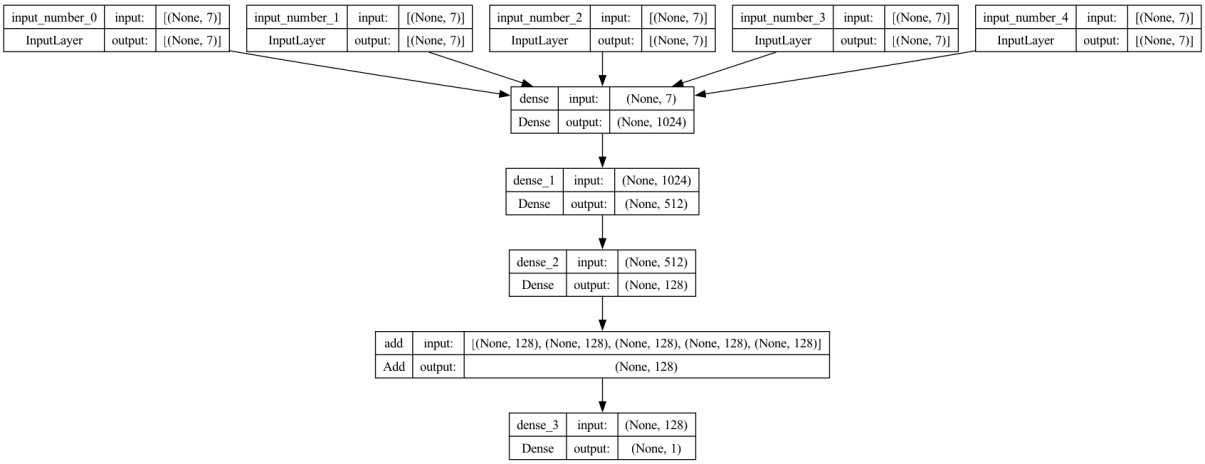
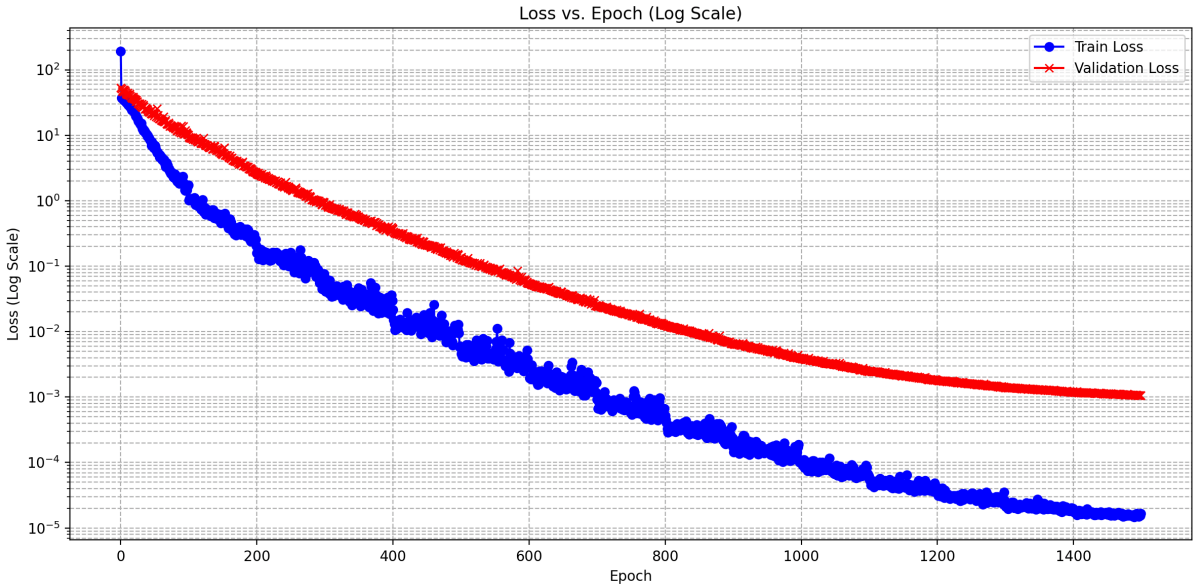
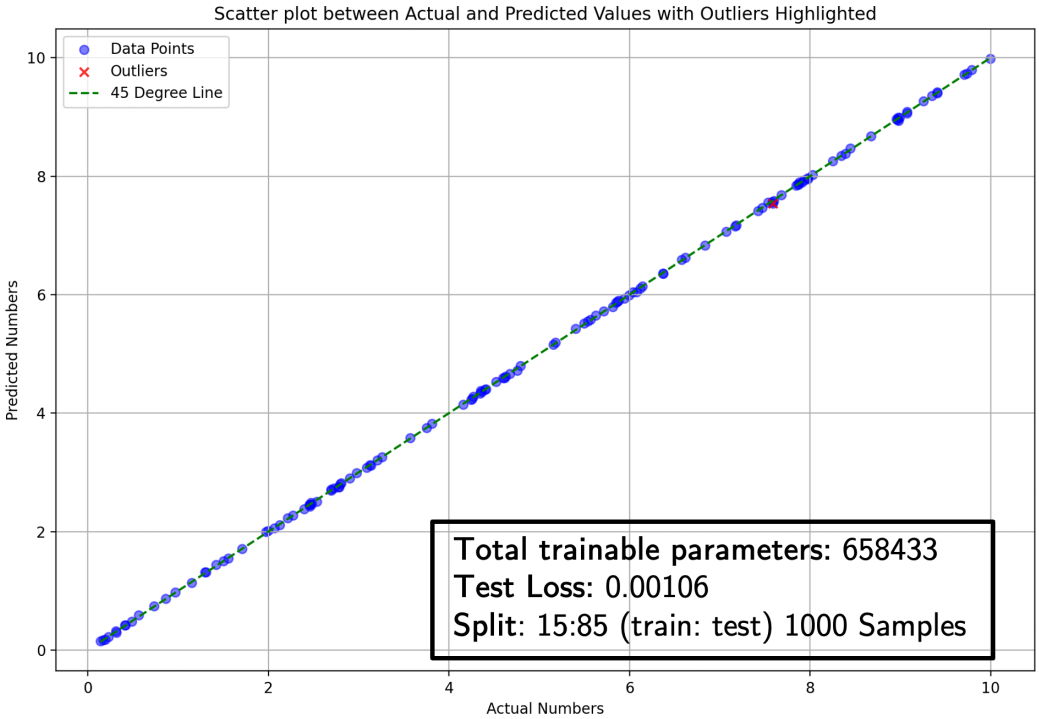


Figure 5.3: Final shared-weight architecture diagram (for 7-bit representation – similar for 128-bit representations)

After implementing this shared model:

- All outlier behavior disappeared.
- Atom predictions were accurate regardless of position.
- Test MSE improved, especially in low-data regimes.





5.6 Conclusion

In this chapter, we uncovered the root cause of outlier behavior observed under low-data training conditions. While initial suspicion fell on atom frequency imbalance, deeper investigation revealed that the core issue was **architectural positional bias**—arising from the use of **independent subnetworks** for each atom position. This design unintentionally tied atom identity to positional context, preventing generalization when atoms appeared in unseen positions.

Shuffling atom positions during training proved partially effective, as it allowed different subnetworks to observe the same atoms across positions. However, the true resolution came from replacing the position-specific architecture with a **shared-weight neural subnetwork**, enabling the model to learn a **position-invariant atomic representation**. This led to the complete disappearance of outliers, robust predictions across all atoms, and improved performance even in extreme low-data regimes.

These findings highlight a critical insight: **model architecture must reflect task symmetry**, especially when training data is limited. Without structural permutation invariance, even seemingly simple problems like summation can mislead the model into learning spurious positional dependencies.

Chapter 6

Scaling the Model: 5-Sum to 60-Sum

6.1 Motivation for Scaling

After resolving positional bias in our 5-sum architecture through weight sharing and permutation handling, we set out to test whether the architecture could **generalize to larger summation sizes**. In real-world chemistry and materials systems, properties often result from the collective contribution of dozens of atoms. Thus, our key questions were:

1. Can the model maintain performance with a significantly larger number of atom inputs?
2. Does weak supervision still suffice when summation involves **60 atoms**?

To investigate, we extended the model to both **10-sum** and **60-sum** problems using the same shared-weight architecture.

6.2 10-Sum Experiment

We expanded the summation input from 5 atoms to 10. The architectural design remained consistent: each atom's 7-bit binary representation was passed through a **shared subnetwork**, and the outputs were summed to predict the total.

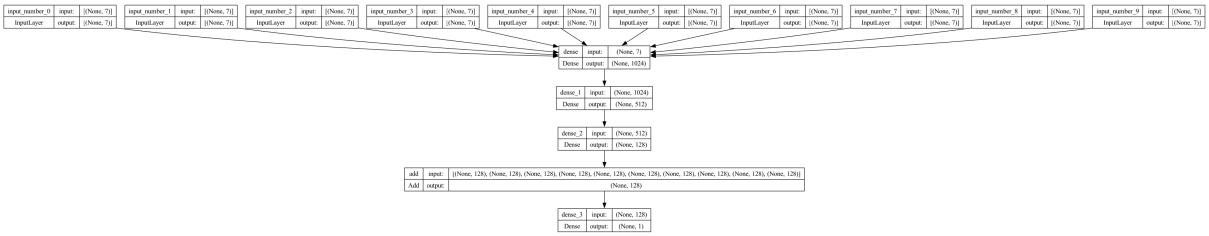


Figure 6.1: Model architecture for 10-sum problem (shared-weight setup)

This test was used primarily to confirm that:

1. Shared weights remain scalable to wider summation windows.
2. The model does not develop any new form of overfitting or instability with increasing atom count.

We do not report metrics for this run, but it served as an intermediate validation step before moving to a more ambitious 60-atom task.

6.3 60-Sum Problem: Performance at Scale

We next pushed the limits by scaling to a **60-atom summation task**, retaining the same neural design philosophy. Each atom was encoded with a 7-bit binary vector, passed through the **shared encoder**, and all 60 outputs were summed before regression.

Input (60 X 7) → (60 X 128) → (60 X 1024) → (60 X 512) → 1

Figure 6.2: Full model architecture for 60-sum problem (like 10-sum)

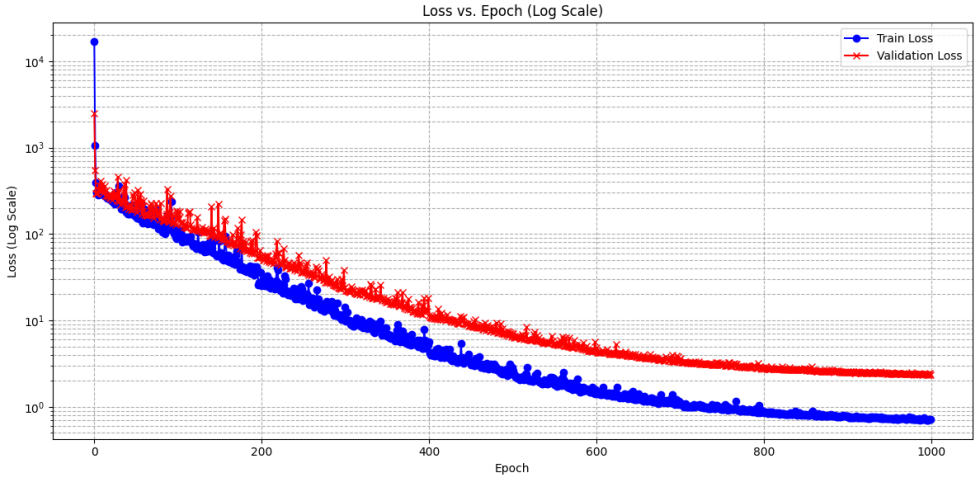
This experiment was conducted in two separate settings.

6.3.1 60-Sum with 1% Training Data (Extreme Low-Data Test)

1. **Split:** 01:99 (300 train / 29,700 test)
2. **Test Loss:** 2.3626
3. **Training Time:** 15 hours
4. **Total Parameters:** 658,433

Despite the extremely limited training data, the model:

1. Learned stable atom representations,
2. Avoided outlier behavior,
3. Preserved rank order, as seen in the **actual vs. predicted scatter plot**.



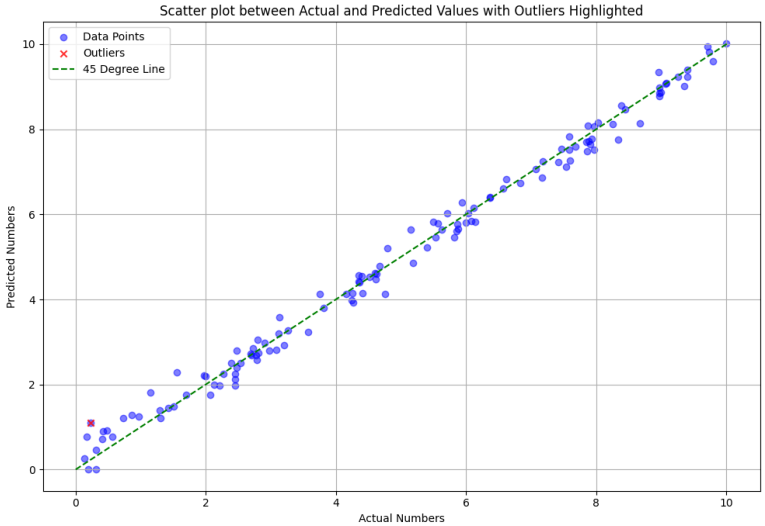


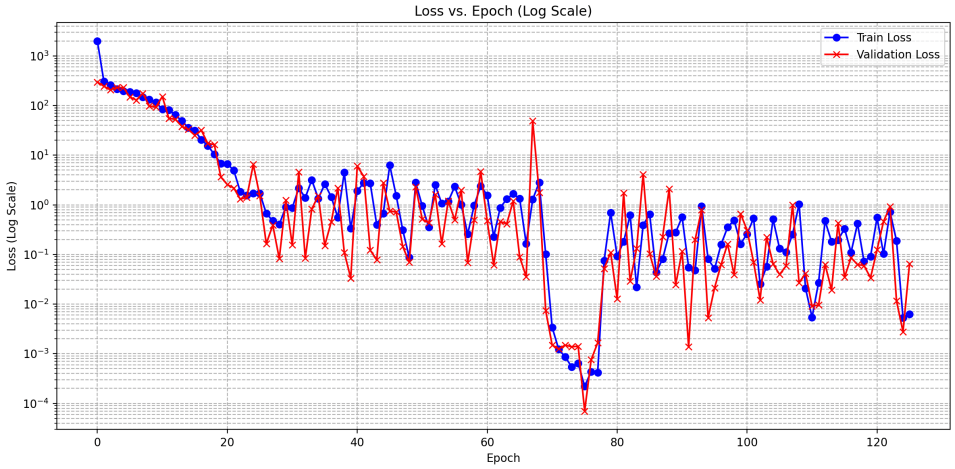
Figure 6.3: Loss vs. Epoch (log scale) and scatter plots for this run.

6.3.2 60-Sum with 10% Training Data (Improved Resolution)

1. **Split:** 10:90 (3,000 train / 27,000 test)
2. **Test Loss:** 7.02e-05
3. **Training Time:** 2 hours (with early stopping)

This configuration showed:

1. Substantial improvement in prediction accuracy,
2. Minimal error variance across atoms,
3. Perfect alignment in **actual vs. predicted** plots even for extreme sum values.



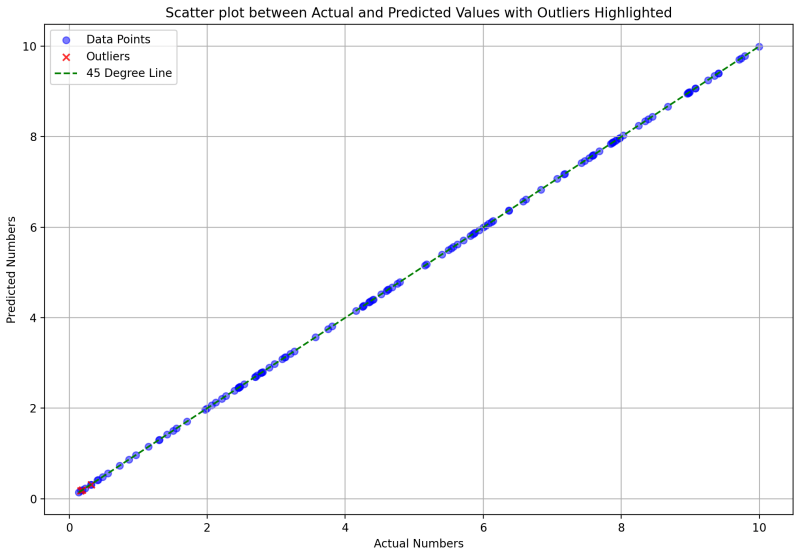


Figure 6.4: Final plots and accuracy breakdown.

The model accurately predicted atomic contributions even when extrapolated to unseen atomic combinations.

6.4 Conclusion of Scaling Study

These experiments demonstrate that our shared-weight, permutation-invariant architecture scales robustly to large summation sizes:

1. No new positional bias or outliers reappeared,
2. The model adapted to 60-atom inputs with minimal supervision,
3. Prediction accuracy improved consistently with increased data.

This confirms the feasibility of **learning atomic properties from aggregate labels** at scale —paving the way for future extensions to real-world molecular systems.

Chapter 7

Conclusion

7.1 Summary of Work

This thesis explored the challenge of learning individual atomic properties from aggregate molecular labels, a problem inspired by the nature of real-world data in chemistry, where only bulk or averaged properties are often measurable. We developed and rigorously evaluated a neural network framework that learns these atomic-level quantities using only the sum of atom values as supervision.

Beginning with the 3-sum and 5-sum toy problems, we constructed models that took as input binary or one-hot encodings of atoms and were tasked with predicting the sum of their unknown underlying values. Through these controlled experiments, we encountered and addressed fundamental challenges related to data distribution, architectural design, and generalization under weak supervision.

7.2 Key Contributions

1. **Weak-Supervision Learning Setup:**
   We formulated a clean abstraction of weak-label supervision through the n-sum problem—allowing us to benchmark neural architectures on their ability to reverse-engineer per-atom contributions from only total sums.

2. **Discovery and Resolution of Positional Bias:**
   We uncovered that visually symmetrical architectures were position-sensitive due to non-shared subnetworks. This architectural flaw led to consistent outliers when atoms appeared in positions not seen during training. We traced this to a structural positional bias, not a data or optimization issue.

3. **Shared-Weight Neural Design:**
   We introduced a **shared-weight architecture** where all atom inputs are passed through a common subnetwork. This enforced **permutation invariance** and eliminated the positional bias entirely. Outliers disappeared, generalization

improved, and performance stabilized across all positions and samples.

4. **Scalability Demonstrated (5-sum → 60-sum):**
   Our framework was extended from 5-sum to 60-sum inputs, showing no architectural limitations. The model was able to learn accurate atomic representations even with only 1% of the total data, highlighting its potential in **low-data regimes**—a typical scenario in materials science.

7.3 Broader Impact and Insight

This work presents a powerful insight:

> *"A neural network can learn correct atom-level representations using only aggregate-level supervision — if and only if the architecture respects the symmetry and permutation structure of the data."*

This makes our approach relevant for a wide range of problems in materials science, physics, and chemistry where direct supervision is difficult, expensive, or impossible. It also reinforces the idea that data augmentation and brute-force training are not sufficient without careful architectural alignment to the task.

7.4 Limitations and Open Questions

1. The experiments were conducted on synthetic data with known ground truth. Real molecular systems introduce additional noise, correlations, and structural complexity (e.g., bonding topology) not present in these setups.
2. The model is currently limited to simple summation tasks. Future work should explore nonlinear aggregate functions, such as energies or dipoles, where atom contributions interact.
3. Interpretability of learned atom representations remains an open challenge. While we achieve accuracy, understanding the semantic or physical meaning of embeddings in real systems requires further investigation.

7.5 Future Directions

1. **Apply to Real Chemical Systems:**
   Incorporate DFT-calculated properties for molecules and test whether the model can extract physically meaningful atomic descriptors.

2. **Graph-Based Extensions:**
   Extend the input representation from 1D atom encodings to **molecular graphs**, allowing the model to learn from both identity and local topology using GNNs.

3. **Beyond Summation:**
   Explore weak supervision for tasks involving **max, min, or nonlinear combinations** of atomic properties.

4. **Uncertainty Quantification:**
   Introduce probabilistic models to estimate uncertainty in atom predictions from ambiguous aggregate labels.

7.6 Final Remarks

This work underscores a critical lesson in machine learning:

> *"Inductive bias, not just data, determines what a model learns."*

Through a focused investigation of weak supervision in a toy chemical system, we show how a simple symmetry violation can derail learning, and how restoring that symmetry unlocks generalization and scalability.

Bibliography

https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book.pdf

Krogh, A., & Hertz, J. (1992). Generalization in a linear perceptron in the presence of noise. Journal of Physics, 25(5), 1135–1147. https://doi.org/10.1088/0305-4470/25/5/020