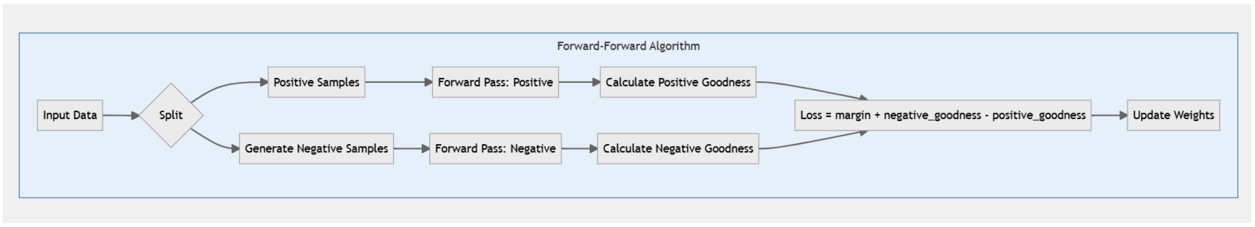
Take Home Assignment 2
Member #1: Arya Bhavesh Shah
Roll No: 25290002
arya.shah@iitgn.ac.in

Assignment Details:
Based on the class discussion, go through the Forward Forward paper: https://arxiv.org/pdf/2212.13345, and implement sections 3.1 and 3.2 of the paper using code.

Bonus points for
- implementations for sections 3.3 and 3.4
- thinking about implementation of alternatives of the goodness measure.

Introduction

Before diving right into the experiment I would like to quickly recap the concept of Geoff's thought process in the Forward Forward paper and how it contrasts with the traditional backpropagation method:
Contrary to how backpropagation worked with one forward pass where the message gets passed forward through the network and when the final loss is calculated, we trace backward through the entire network telling each node exactly how much they messed up and how to fix it, which is both computationally expensive and doesn't really match how our brains work, the forward forward approach uses two forward passes with completely different types of data. The positive pass' job is to learn what is right, we show the network the real, correct examples and each layer in the network tries to get "excited" or have high goodness when it sees these real examples. Goodness is just like a confidence score of each layer depicting how confident the layer feels about what it is seeing. The negative pass learns what is wrong. The network is shown fake or incorrect examples and each layer tries to have low goodness for these wrong examples. This teaches the network to reject incorrect patterns.
So, essentially, the network learns by maximizing the difference between goodness values for positive and negative samples.
Hinton tested this on simple image recognition tasks and found it works well achieving results comparable to traditional methods, 1.4% error rate on MNIST. The direction of his research shows that there are possibilities for creating AI systems that learn more like biological brains do efficiently without needing to constantly trace backward through a complex chain of errors.


Forward-Forward Algorithm
Input Data
Split
Positive Samples
Forward Pass: Positive
Calculate Positive Goodness
Generate Negative Samples
Forward Pass: Negative
Calculate Negative Goodness
Loss = margin + negative_goodness - positive_goodness
Update Weights

Section 3.1 The Backpropagation Baseline

In this section, Geoff establishes what performance we should expect from a good learning algorithm, making use of the MNIST Dataset as his benchmarking dataset which performs extremely well with traditional backpropagation methods.
The CNNs achieve 0.6% error rate and the permutation invariant networks achieve 1.4% error rate, where they treat pixels like a random bag, ignoring the spatial layout.

This is the baseline that makes use of traditional backpropagation hence it was easy to implement. We are unable to match the 1.4% error rate but we are able to get somewhere in the range of 4.6% as shown below:

```
Epoch 20/20 - Train Loss: 0.1548, Val Accuracy: 95.60%
Testing:
100%|████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████
█████| 79/79 [00:01<00:00, 56.85it/s]
Test Accuracy: 95.39%, Test Error: 4.61%
```

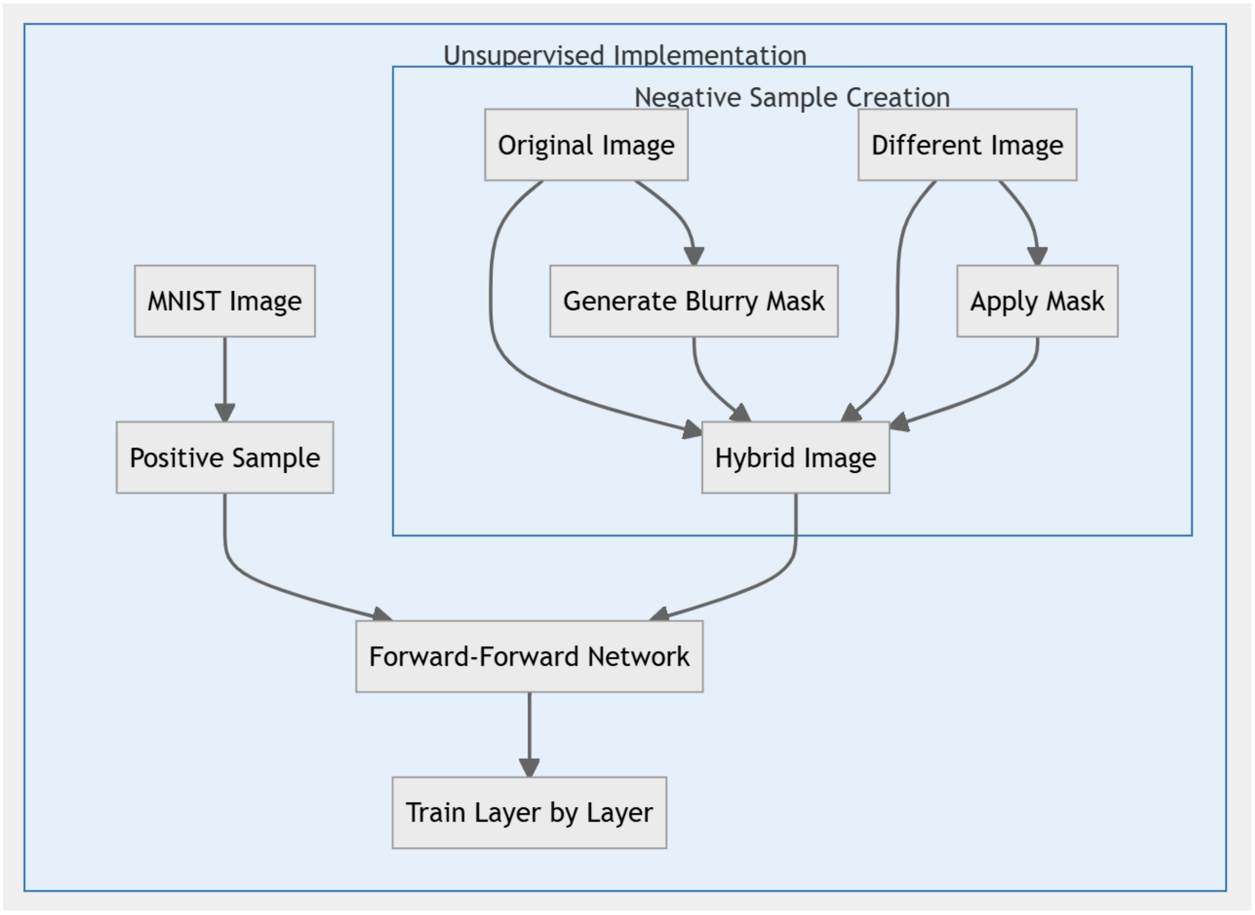
Section 3.2 A simple unsupervised example of FF

Here, Hinton tests whether FF can learn good internal representations without making use of any labels; just by distinguishing real images from the fake ones. He creates these negative examples by mixing two different images together using masks and then blending them together.
The process goes as follows:

1. Show the network real MNIST digits (positive examples)
2. Show it these blended hybrid images (negative examples)
3. Train each layer to get "excited" for real images and "calm" for fake ones
4. Test by using the learned representations to classify digits with a simple classifier

The key components of my implementation are as follows:

- Positive Samples: Original unmodified images
- Negative Sample Generation: Created by blending parts of different images using masked regions
- Layer-wise Training: Each layer is trained independently before moving to the next



After training 4 hidden layers with 2000 neurons each Hinton achieved 1.37% error which was very close to the backpropagation baseline.
My implementation yielded the following results:

```
Epoch 50/50 - Val Accuracy: 82.96%
Testing:
100%|████████████████████████████████████████
██████████████████████████████████████████
███| 79/79 [00:01<00:00, 52.22it/s]
Test Accuracy: 81.90%, Test Error: 18.10%
```

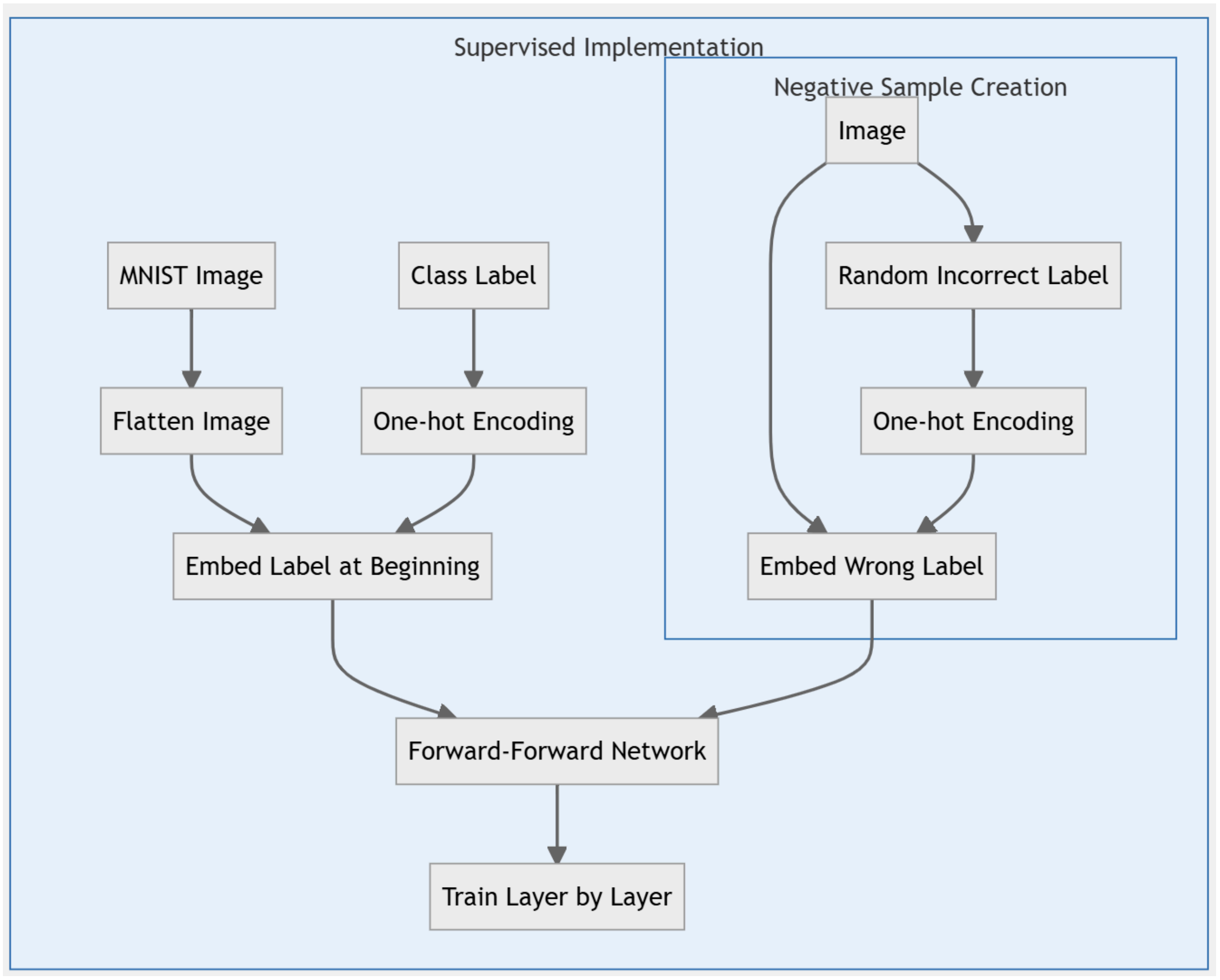
Section 3.3 A simple supervised example of FF

In this section, Geoff shows how if we have labels, how to use them in a FF model. Instead of just showing images, we include the correct label as part of the input. The first 10 pixels of each image are replaced with a one-hot encoding of the digit label.

So, the training setup looks as follows:

- Positive examples: Real Image + correct label
- Negative examples: Real Image + wrong label

I believe this works since the only difference between +ve and -ve examples is the label and the network learns to focus on image features that actually correlate with the labels ignoring other irrelevant details



The key components of my implementation include the following:

- Label Embedding: One-hot encoded labels are embedded into the beginning of the flattened image vectors
- Negative Sample Generation: Generated by pairing images with incorrect labels
- Layer-wise Training: Each layer is trained independently before moving to the next

Hinton achieved 1.36% test error rate after 60 epochs with a 4 layer network, compared to backpropagation’s 20 epochs for similar performance. On adding augmentations, he got 0.64% error, matching CNN results/
My implementation yielded the following results (25 epochs)

```
100%|██████████████████████████████████████████████████████████████████████████████████████████████████████| 79/79 [00:01<00:00, 52.22it/s]
Test Accuracy (Classifier): 81.02%, Test Error: 18.98%
```

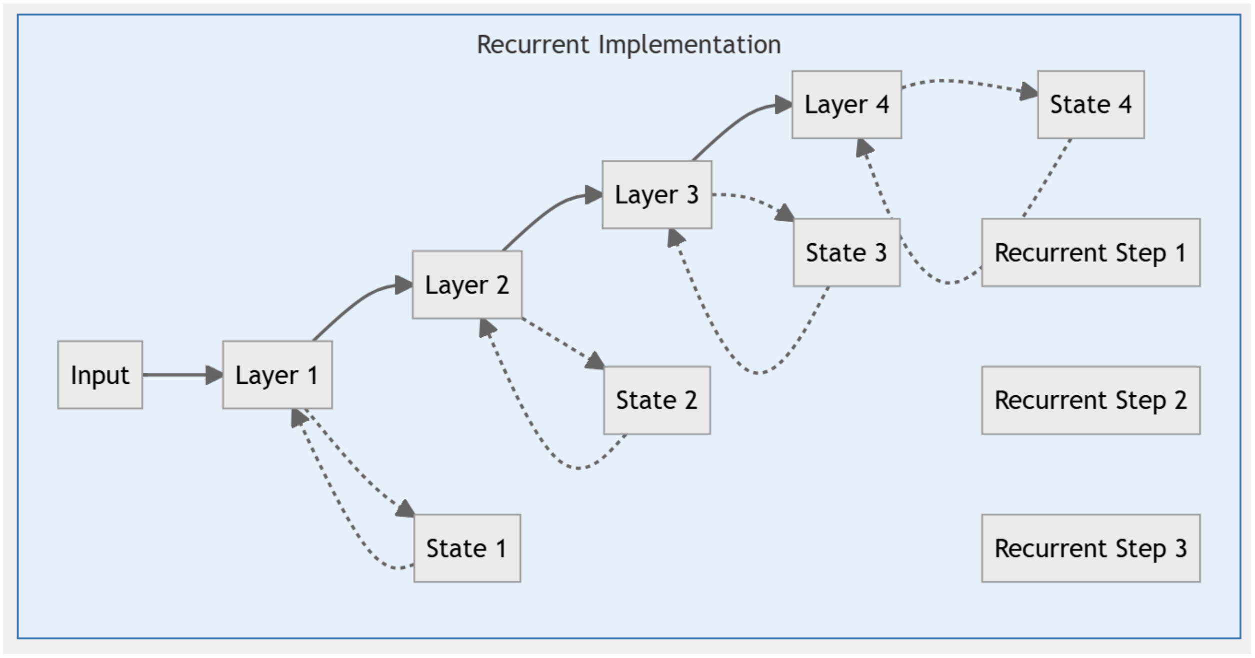
Section 3.4 Using FF to model top-down effects in perception (Recurrent Network)

Out of all the methods, this is the most sophisticated method which addresses the limitations of how later layers influence what earlier ones learn?
Instead of treating images as static, Geoff treats them like very boring videos that get processed by a recurrent network over time i.e feedback.
The architecture:

- Each layer receives input from both the layer below AND the layer above
- The network processes the same image multiple times, allowing information to flow both up and down
- This mimics how our brain's visual cortex actually works, with lots of feedback connections

Training process:

- Initialize all layers with a single bottom-up pass
- Run the network for several iterations, letting top-down and bottom-up information interact
- For each test image, try all 10 possible labels and see which one makes the network most "happy" overall



Key components of my implementation include:

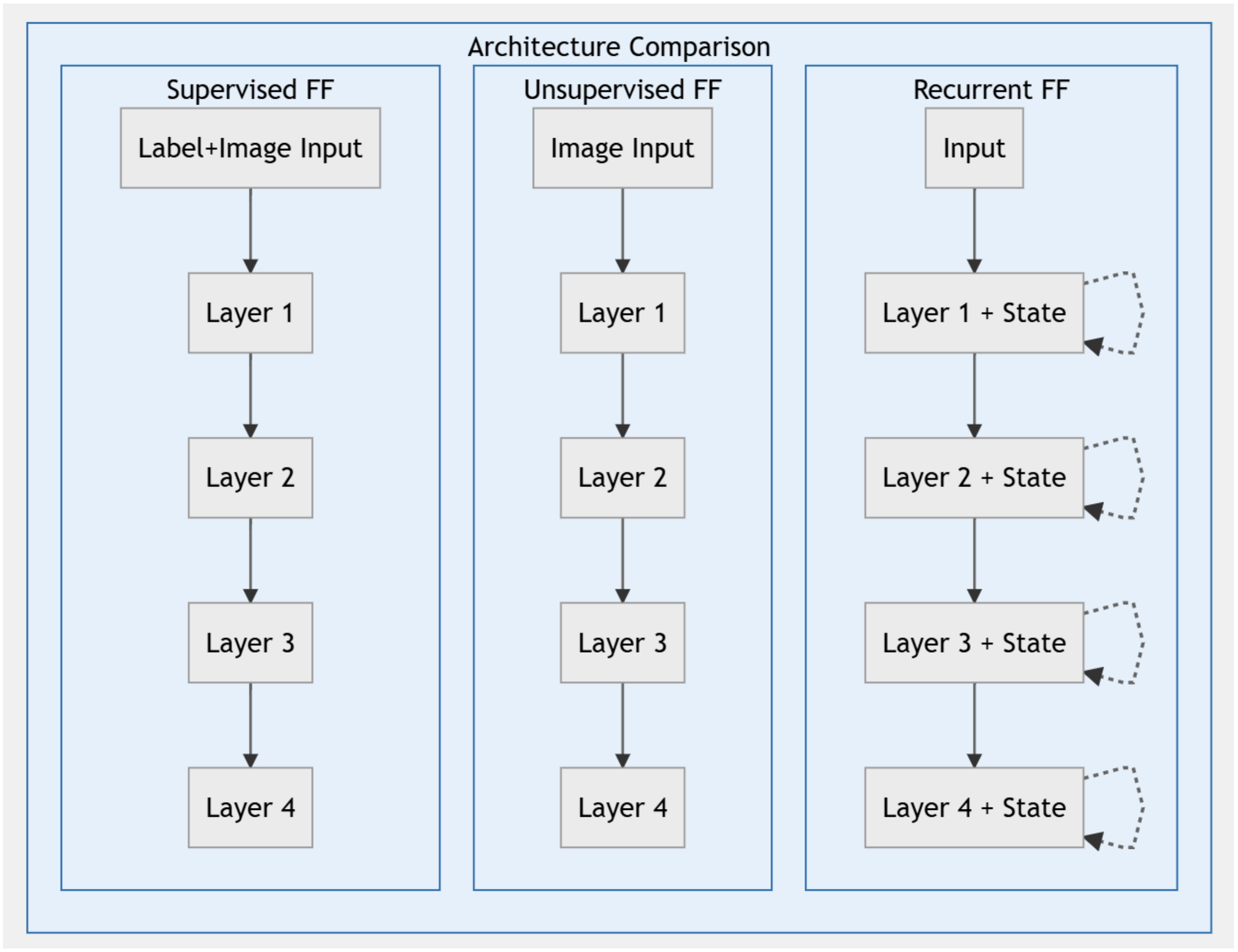
- State Recurrence: Each layer maintains a state that is updated over multiple iterations
- Mixed Recurrence: Information can flow both between layers and within the same layer across iterations
- Supervised Mode: Can operate with label embedding similar to the supervised implementation

Hinton's implementation gets a 1.31% test error for this.

My implementations yielded the following results:

```
Epoch 25/25 - Loss: 0.0917, Val Accuracy: 82.21%
Test Accuracy: 81.40%, Test Error: 18.60%
```

Thus the three architectures when placed next to each other look like the following:



And, to summarize the 4 sections, I provide the results in the table as shown below:

Method	**Final Test Accuracy**	**Test Error**
Backpropagation Baseline	95.60%	4.61%
Supervised Forward-Forward	81.02%	18.98%
Unsupervised Forward-Forward	81.9%	18.10%
Recurrent Forward-Forward	81.40%	18.60%

Alternatives to “Goodness”

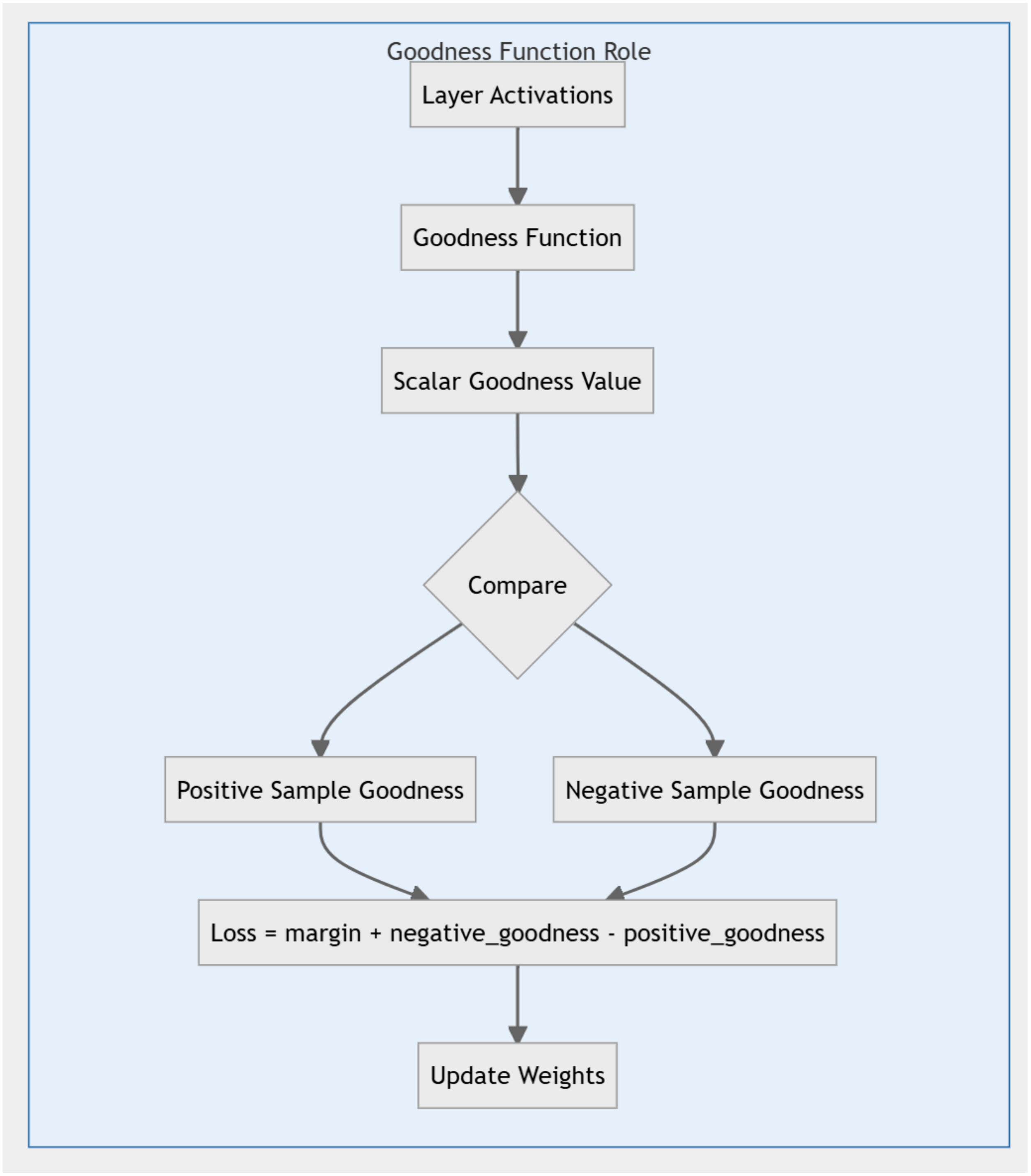
This was the most interesting part of this assignment and I can think of a research paper that can be written with the title starting “In search of goodness…”
In the paper, Geoff clearly mentions that the concept of goodness is yet to be explored further, there can be many goodness functions and it is not a one size fits all scenario.
In this section, I try out a couple of goodness alternatives that take inspiration from representation learning, contrastive learning and similar principles trying to benchmark it with the baseline sum of squares goodness function.

This section provides a detailed explanation of various goodness functions implemented in our Forward-Forward algorithm implementation. Each goodness function offers a different approach to measuring the "goodness" of neural network activations, with unique mathematical properties and training characteristics.

The FF algorithm relies on the goodness function. It measures how good a layer's activation pattern is with the goal of maximizing goodness for positive samples and minimizing it for negative samples. From what I inferred from the paper, a “good” goodness function should be one that is able to achieve the following:

- Create a clear separation between positive and negative samples
- Promote useful representations in the network
- Be efficiently computable
- Work well with the network's training dynamics

Keeping time and resources in mind, I train each goodness function for a set number of 10 epochs only.



Squared Sum Goodness

The squared sum goodness is the simplest goodness function, computed as the sum of squared activations. It has been used in the original FF paper as well.

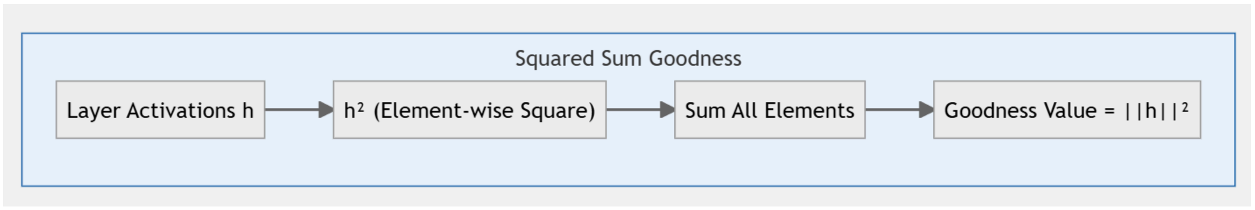
$$G_{sq}(h) = \sum_{i=1}^{n} h_i^2 = \|h\|_2^2$$

Where:

- h is the activation vector of a layer
- n is the number of units in the layer

Characteristics

- Simplicity: Easy to compute and optimize
- Unbounded: Values can grow very large during training
- Sensitivity to Magnitude: Favors larger activation values, potentially leading to activation explosions
- Original FF Algorithm: This was used in the original Forward-Forward paper



The paper mentions that "many other measures are possible" beyond the sum of squares, but this squared activity measure was chosen for its mathematical simplicity and practical effectiveness in the experiments.

This goodness function yielded me the following results:

```
Epoch 10/10 - Val Accuracy: 56.92%
Test Accuracy: 55.46%, Test Error: 44.54%
```

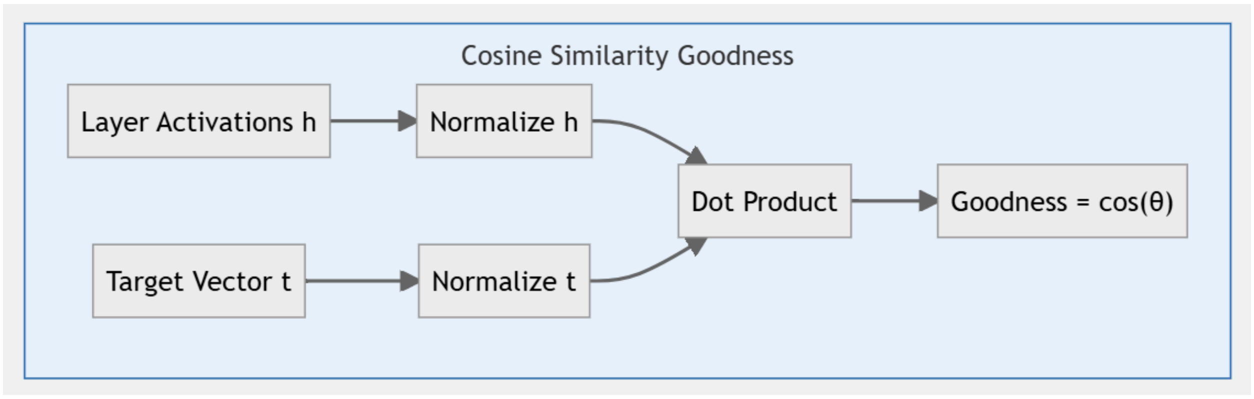
Cosine Similarity

Cosine similarity goodness measures the similarity between the activation vector and a target vector (typically a vector of ones):

$$G_{cos}(h) = \frac{h \cdot t}{||h||_2||t||_2} = \cos(\theta)$$

Where:

- h is the activation vector of a layer
- t is a target vector (often a vector of ones)
- θ is the angle between the two vectors



Characteristics

- Normalized: Values are bounded between -1 and 1
- Direction Sensitive: Focuses on the direction of the activation vector, not its magnitude
- Stability: Less prone to activation explosion compared to squared sum
- Target Direction: Encourages activations to align with the target direction

This goodness function yielded me the following results:

```
Epoch 10/10 - Val Accuracy: 78.27%
Test Accuracy: 77.06%, Test Error: 22.94%
```

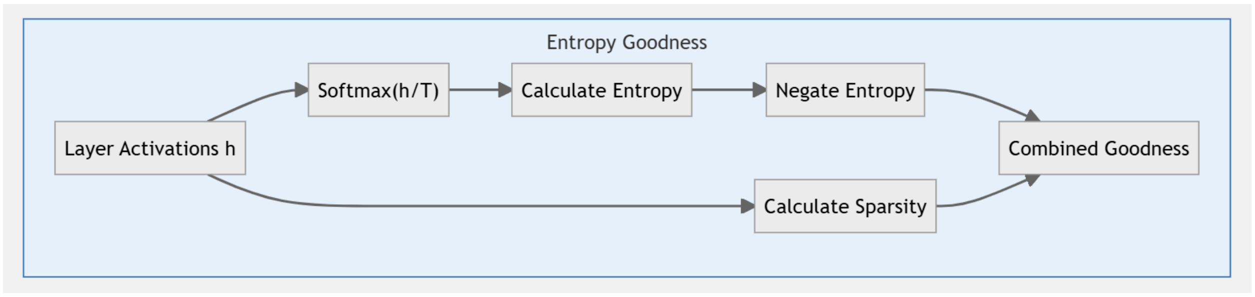
Entropy

Entropy goodness uses the negative entropy of the activation distribution as a measure of goodness

$$G_{ent}(h) = -H(p) = \sum_{i=1}^{n} p_i \log(p_i)$$

Where:

- h is the activation vector of a layer
- p=softmax(h/T) is the probability distribution over activations
- T is a temperature parameter
- H(p) is the entropy of distribution



Characteristics

- Concentration: Rewards low-entropy, concentrated activation patterns
- Interpretability: Encourages more deterministic, confident representations
- Temperature Sensitivity: Results are highly dependent on the softmax temperature
- Sparsity Promotion: With the sparsity bonus, encourages fewer but stronger activations

This goodness function yielded me the following results:

```
Epoch 10/10 - Val Accuracy: 83.75%
Test Accuracy: 82.39%, Test Error: 17.61%
```

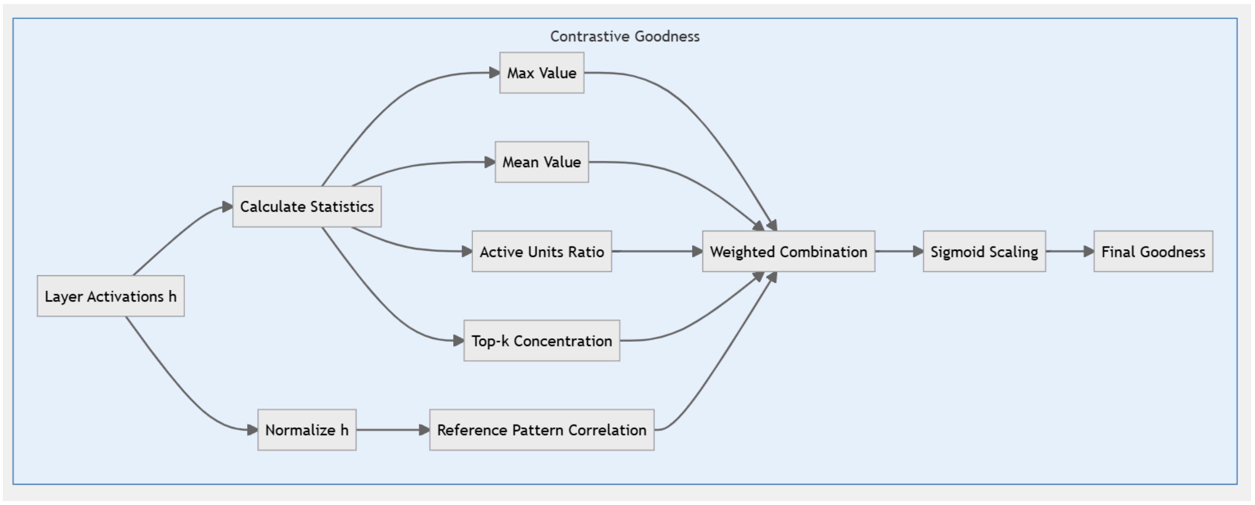
Contrastive Goodness

Contrastive goodness uses principles from contrastive learning, measuring the similarity of activations to reference patterns while rewarding certain statistical properties

$$G_{con}(h) = \alpha \cdot S_{max}(h, R) + \beta \cdot M(h) + \gamma \cdot (1 - A(h)) + \delta \cdot V(h)$$

Where:

- h is the normalized activation vector
- max(h,R) is the maximum similarity to reference patterns
- M(h) is the mean activation
- A(h) is the ratio of active units
- V(h) is the variance across activation dimensions
- α,β,γ,δ are weighting parameters



Characteristics

- Multi-faceted: Considers multiple statistical properties of activations
- Reference Patterns: Uses correlation with reference patterns
- Sparsity: Rewards sparser activations (fewer active units)
- Non-linear Transformation: Uses sigmoid scaling to increase separation between positive and negative examples
- Feature Memory: Maintains a memory bank of features (though this is used for monitoring only in our implementation)

This goodness function yielded me the following results:

```
Epoch 10/10 - Val Accuracy: 50.80%
Test Accuracy: 50.87%, Test Error: 49.13%
```

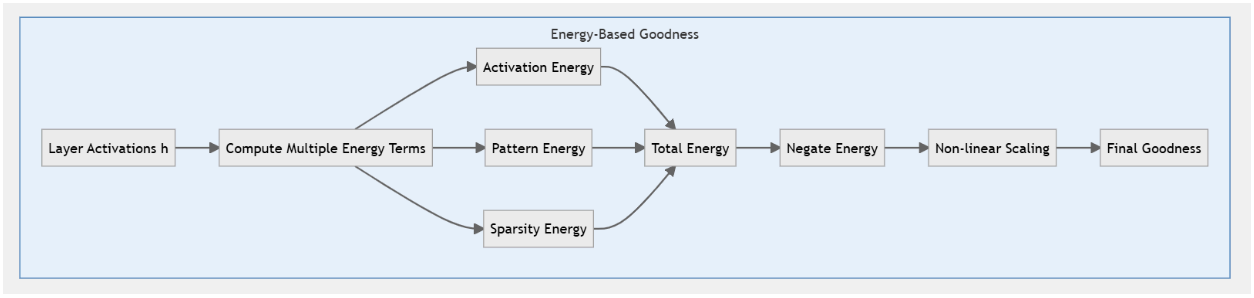
Energy Based Goodness

Energy-based goodness treats the network as an energy model, with lower energy corresponding to higher goodness

$$G_{eng}(h) = -E(h) = -(E_{act}(h) + E_{pat}(h) + E_{spa}(h))$$

Where:

- $E_{act}(h) = -\log \sum_i \exp(h_i)$ is the activation energy
- $E_{pat}(h) = -|S(h, p)|$ is the pattern matching energy
- $E_{spa}(h) = -A(h)$ is the sparsity energy
- $A(h)$ is the ratio of active units



Characteristics

- Energy Framework: Based on energy-based models where low energy = high goodness
- Multiple Energy Terms: Combines different energy components
- Pattern Matching: Includes similarity to reference patterns
- Sparsity: Rewards sparser activations
- Non-linear Scaling: Uses tanh scaling to bound and separate goodness values

This goodness function yielded me the following results:

```
Epoch 10/10 - Val Accuracy: 79.12%
Test Accuracy: 78.35%, Test Error: 21.65%
```

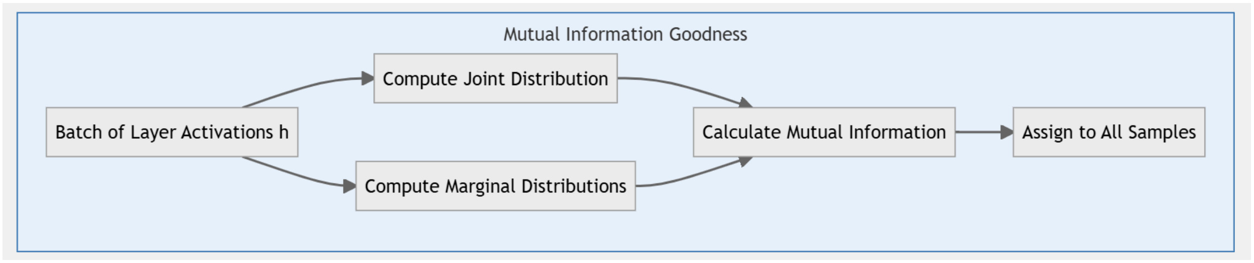
Mutual Information Goodness

Mutual information goodness measures the mutual information between activations within a batch

$$G_{MI}(h) = I(h_i, h_j) = \sum_{i,j} p(h_i, h_j) \log \frac{p(h_i, h_j)}{p(h_i)p(h_j)}$$

Where:

- $p(h_i, h_j)$ is the joint distribution of activations
- $p(h_i)$ and $p(h_j)$ are the marginal distributions
- $I(h_i, h_j)$ is the mutual information between activations


Mutual Information Goodness
Batch of Layer Activations h
Compute Joint Distribution
Compute Marginal Distributions
Calculate Mutual Information
Assign to All Samples

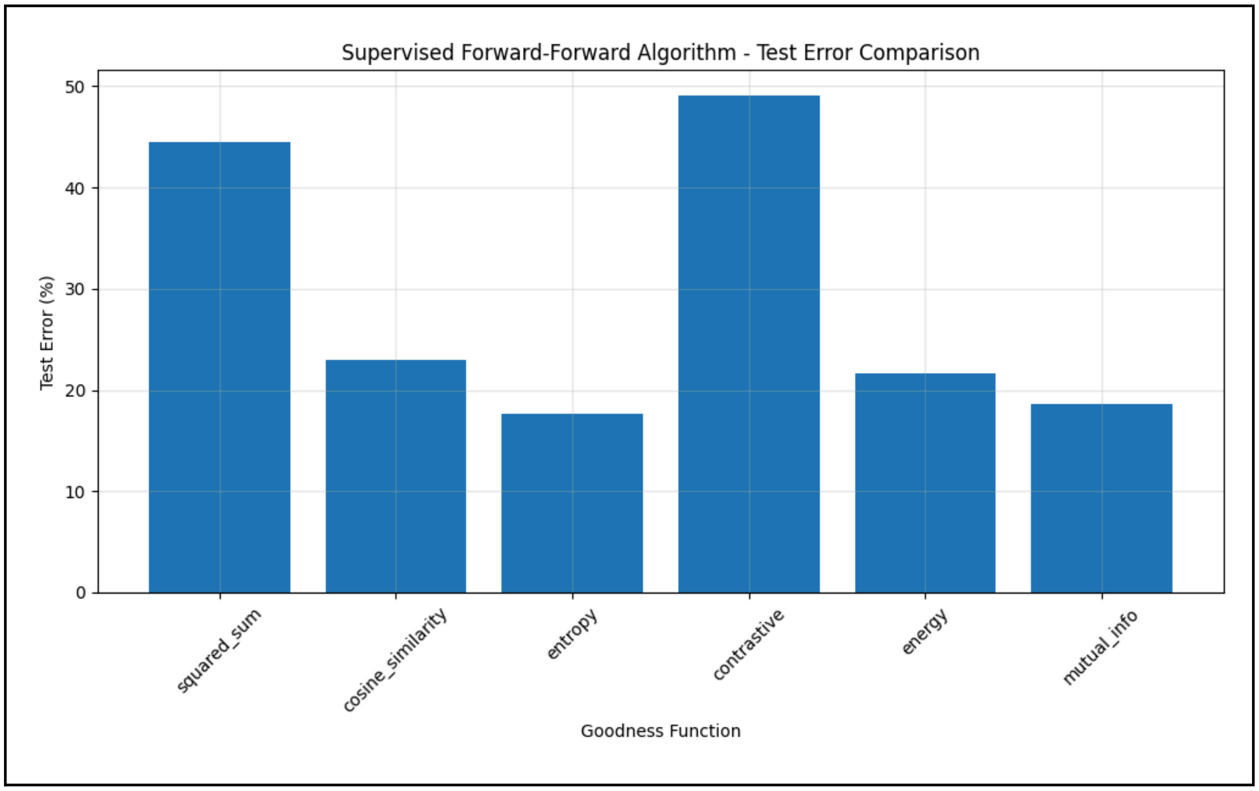
Characteristics:

- Information Theoretic: Based on information theory principles
- Batch-level: Computed across the entire batch rather than per sample
- Relationships: Captures statistical relationships between different activation dimensions
- Computational Cost: More computationally intensive than other goodness functions
- Uniform Values: Returns the same goodness value for all samples in a batch

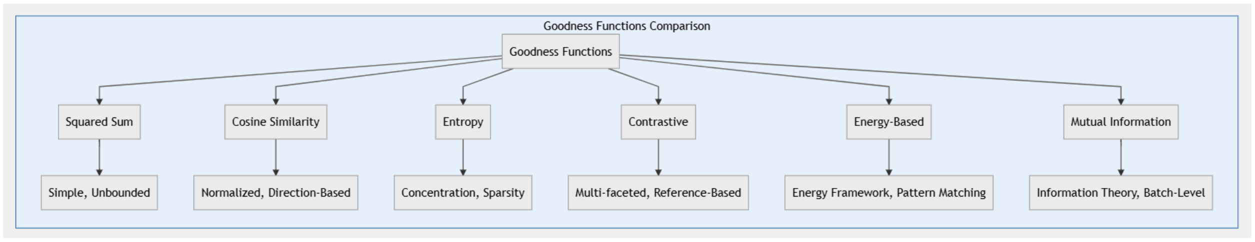
This goodness function yielded me the following results:

```
Epoch 10/10 - Val Accuracy: 82.44%
Test Accuracy: 81.34%, Test Error: 18.66%
```

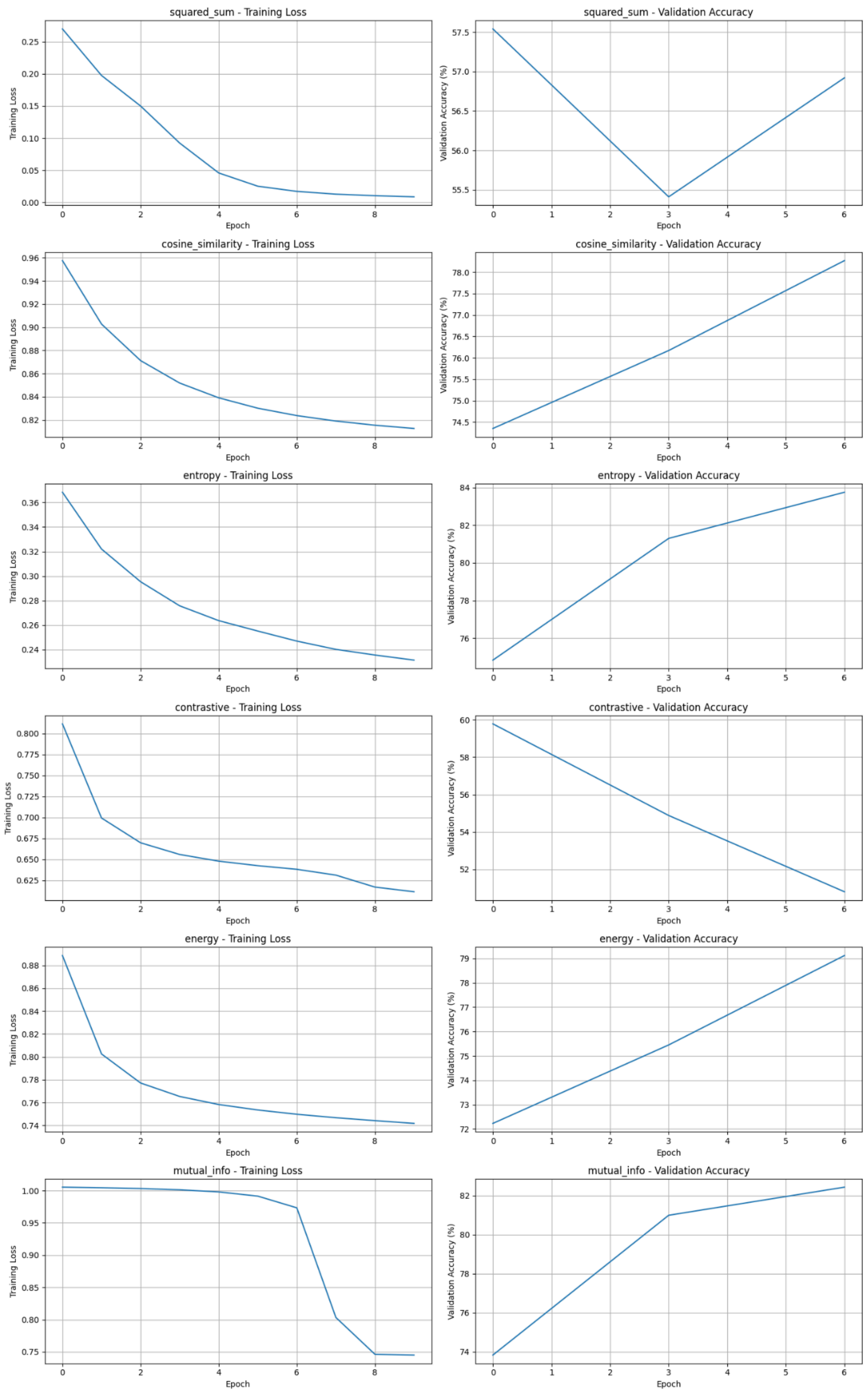
```
======================================================================
====
Goodness Function Comparison
======================================================================
====
Goodness Type       Test Accuracy   Test Error
------------------------------------------------------------------------------
squared_sum         55.46           44.54
cosine_similarity   77.06           22.94
entropy             82.39           17.61
contrastive         50.87           49.13
energy              78.35           21.65
mutual_info         81.34           18.66
======================================================================
====
```



It can be concluded that different goodness functions exhibit various behaviours, a complete summary is shown below:


Goodness Functions Comparison
Goodness Functions
Squared Sum
Cosine Similarity
Entropy
Contrastive
Energy-Based
Mutual Information
Simple, Unbounded
Normalized, Direction-Based
Concentration, Sparsity
Multi-faceted, Reference-Based
Energy Framework, Pattern Matching
Information Theory, Batch-Level

And below are the Training Loss and Validation accuracy curves for each of the goodness functions implemented and evaluated:


squared_sum - Training Loss
squared_sum - Validation Accuracy
cosine_similarity - Training Loss
cosine_similarity - Validation Accuracy
entropy - Training Loss
entropy - Validation Accuracy
contrastive - Training Loss
contrastive - Validation Accuracy
energy - Training Loss
energy - Validation Accuracy
mutual_info - Training Loss
mutual_info - Validation Accuracy
Training Loss
Validation Accuracy (%)
Epoch