**HW\_1\_1: Simulate a Function: Cos 2∏x\*x3**

I have used three neural networks of varying depth, but comparable parameter counts:

1. **Model 1 (Shallow):** 1 hidden layer (249 units), 748 parameters.
2. **Model 2 (Deep):** 3 hidden layers (18, 20, 15 units), 747 parameters.
3. **Model 3 (Deeper):** 5 hidden layers (5, 10, 20, 15, 8 units), 742 parameters.

A graph of a training model

AI-generated content may be incorrect.

Figure 1: Training loss vs epochs for shallow, 3-layer, and 5-layer models*.*

A graph with red and green lines

AI-generated content may be incorrect.

Figure 2: Predicted curves of all models compared with the ground-truth function.

**Results**

* **Training Loss:**

The shallow model converged more slowly and required significantly more epochs to reduce the loss, while deeper models achieved faster convergence. Model 3 (5 hidden layers) reached the lowest training loss the fastest.

* **Function Approximation:**

The shallow model (Model 1) and the deepest model (Model 3) captured the target function curve closely, matching oscillations and amplitude. The deep model (Model 2) with 3 hidden layers failed to approximate the function properly (particularly when the input values ranged from -2.2 to +2.2), showing almost flat predictions in this region.

**Comments:**

Depth improves representation power and optimization efficiency, even when total parameter counts are similar, but careful consideration of depth is required. It is not always the case that more hidden layers guarantee effective learning of the nonlinear mapping, while shallow models tend to underfit the function.

**HW\_1\_2: Train on Actual Tasks (CIFAR-10)**

**Models:**

I trained three CNN architectures of increasing complexity on the CIFAR-10 dataset:

1. **CNN Model 1 (545,098 params):** Small model with two convolutional blocks.
2. **CNN Model 2 (2,360,906 params):** Medium model with additional layers and batch normalization.
3. **CNN Model 3 (3,249,994 params):** Large model with three convolutional blocks and more channels.

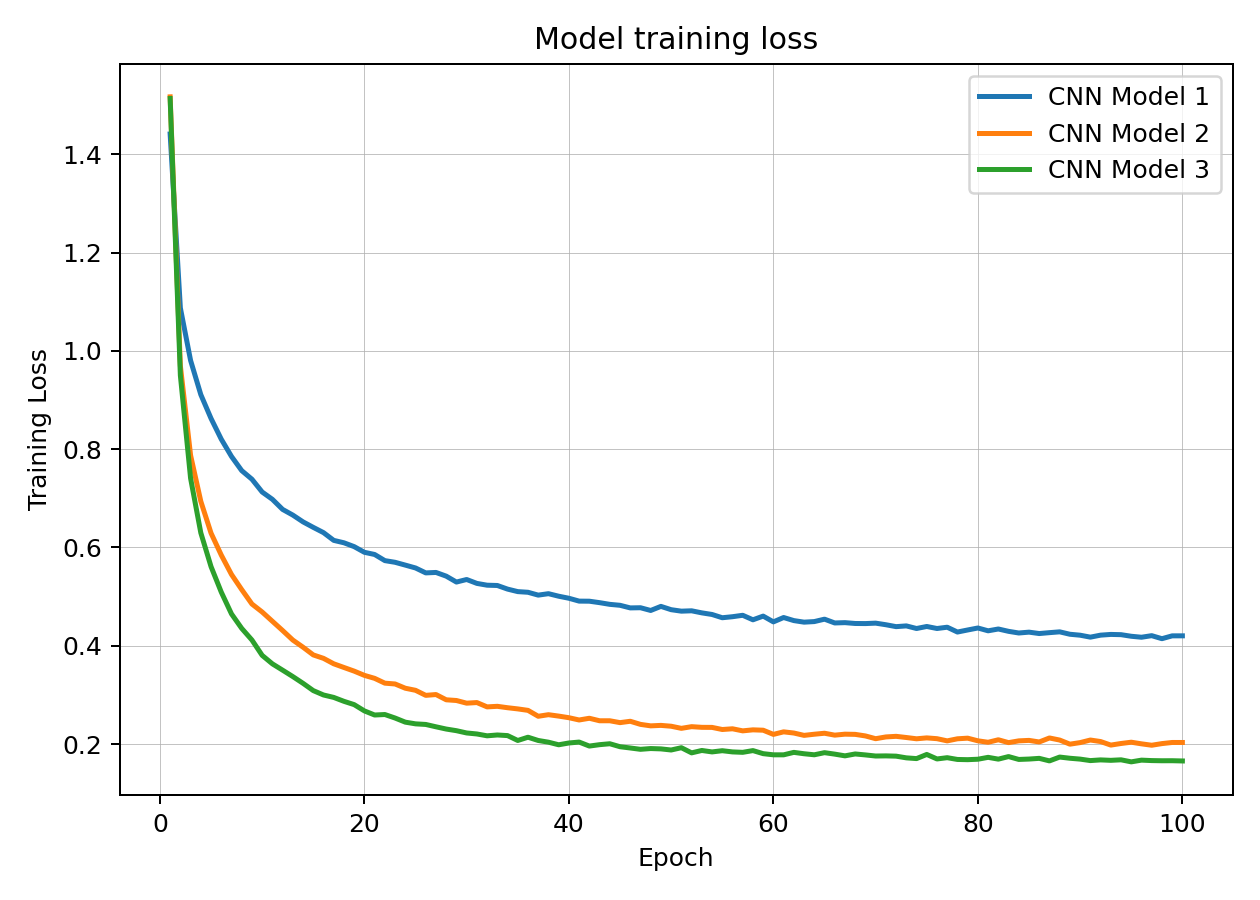


Figure 3. Training loss vs. epochs for three CNN models on CIFAR-10. Model 3 converged faster and reached the lowest loss among all models.

A graph of a training curve

AI-generated content may be incorrect.

Figure 4. Training accuracy vs. epochs for three CNN models on CIFAR-10. Deeper models (Model 2, Model 3) achieved higher accuracy compared to Model 1.

**Results:**

* **Training Loss:** The shallow model (Model 1) converged slower and plateaued at higher loss, while Model 2 and Model 3 achieved lower losses more quickly. The deepest model (Model 3) consistently achieved the lowest training loss.
* **Training Accuracy:** Model 3 reached the highest training accuracy (>93%), followed by Model 2 (~92%) and Model 1 (~85%). The deeper models demonstrated better feature extraction capacity and optimization efficiency.

**Comment:**

Deeper CNNs with more parameters generalize better within the training set, achieving higher accuracy and lower loss. However, increasing model complexity increases computation and memory requirements, which must be balanced against performance gains.

**HW\_1\_2: Visualize the optimization process**

**Experiment settings**

* **Tasks:** 1-D function fitting and CIFAR-10 classification.
* **Models:**
  + Function: Single input single output with hidden widths [18, 20, 15].
  + CIFAR-10: CNN (Conv-ReLU-Pool ×2 → FC).
* **Optimizer:** Adam with lr=10-3, weight\_decay=5\*10-4.
* **Runs:** 8 independent runs (different seeds).
* **Logging cadence:** every 3 epochs for 12 epochs total.
* **Recorded parameters:**
  + **Layer parameters:** first layer only (net.0 for Function, features.0 for CNN).
  + **Whole model parameters:** all trainable weights concatenated.
* **Dimensionality reduction:** stack all checkpoints across runs, PCA to 2-D, then plot run-wise trajectories with epoch colorbar.
* **Plots:** separate figures for first layer and whole model for each task.

A graph with lines and numbers

AI-generated content may be incorrect.

Figure 5 (Function, First layer): PCA-projected optimization trajectories of first layer parameter vectors across 8 runs; color encodes epoch.

A graph with colorful lines

AI-generated content may be incorrect.

Figure 6 (Function, Whole model): PCA-projected optimization trajectories of full parameter vectors across 8 runs; color encodes epoch.

A graph of a graph with colored dots

AI-generated content may be incorrect.

Figure 7 (CIFAR-10, First layer): Early-training trajectories for the first conv layer across 8 runs.

A graph with colorful dots and lines

AI-generated content may be incorrect.

Figure 8 (CIFAR-10, Whole model): Early-training whole-model trajectories; short, near-linear drifts typical of the first dozen epochs.

**Observations & comments**

* **Distinct paths per initialization.** Each run traces a different path in PCA-space, confirming strong dependence on random initialization.
* **Consistent drift directions.** For both tasks, trajectories are largely one- or few-dimensional “rays”, suggesting most variance during training is captured by a small number of parameter directions.
* **Whole-model vs first-layer.**
  + Whole-model plots show longer, more coherent arcs (aggregate movement across layers).
  + First-layer plots show shorter, localized drifts and greater dispersion—early layers adapt, but much of the global movement is explained by later layers and the classifier head.
* **Function task:** trajectories are longer and smoother (many updates with full-batch style behavior), converging toward compact regions—consistent with stable loss descent on a simple supervised signal.
* **CIFAR-10:** with only 12 epochs, paths are short and mostly linear in PCA space (early-training regime); different runs still point in different directions, but all move away from the origin (initialization) toward task-specific basins.

**HW\_1\_2: Observe gradient norm during training**

**Experiment Settings:**

We observed the gradient norm and training loss across iterations for two tasks:

1. **Function Approximation** (Simple MLP).
2. **CIFAR-10 Classification** (CNN).

The basic setup for both models is like what I have done in **HW\_1\_2: Visualize the optimization process**. The gradient norm was recorded at each training step, and loss values were tracked in parallel. Both were plotted against the number of iterations.

**A graph of a graph showing a loss and a function

AI-generated content may be incorrect.**

Figure 9: Gradient norm and training loss vs. iterations for function approximation task.

A graph showing a loss and a loss

AI-generated content may be incorrect.

Figure 10: Gradient norm and training loss vs. iterations for CIFAR-10 CNN model.

**Results:**

* **Function Approximation (MLP):**
  + Loss initially spiked before gradually decreasing to a stable value.
  + Gradient norm showed higher variability, with peaks early in training, then settling down after ~3000 iterations.
  + The sharp oscillations reflect the difficulty of fitting the highly nonlinear target function with a simple model.
* **CIFAR-10 (CNN):**
  + Loss decreased smoothly and stabilized below 1.0 consistently after ~5000 iterations.
  + Gradient norm fluctuated within a moderate band (around 2–5), indicating stable optimization progress without exploding or vanishing gradients.

**Comments:**

* Gradient norm provides a useful diagnostic of training stability: large spikes indicate instability, while consistently small values suggest possible stagnation.
* In CIFAR-10, the CNN achieved a steady balance of gradient magnitudes, supporting effective convergence.
* In the function task, the larger swings in gradient norm highlight the challenges of optimization in shallower networks with nonlinear mappings.

**HW\_1\_2: What Happened When Gradient is Almost Zero**

1. **How I reached (near) zero gradient?**

* Training is done in two stages:
  1. **Stage-1 (task loss):** The model is trained normally on its loss function (MSE for the function, cross-entropy for CIFAR-10) for a few epochs.
  2. **Stage-2 (gradient minimization):** Instead of the loss, we now directly minimize the overall gradient size. This makes the weights settle at a point where the gradients are almost zero.
* The weights at the end of Stage-2 are considered a stationary point because the gradients are close to zero.

1. **How I defined and measured “minimal ratio”?**

* Once at a stationary point, I slightly perturbed the weights many times with small random noise.
* For each perturbed version, I checked if the loss went up compared to the base model.
* The minimal ratio is the fraction of perturbations that make the loss worse.
* Intuitively:
  + A higher ratio means the point is more stable and “flatter,” since most small changes increase the loss.
  + A lower ratio means the point is more “sharp” or saddle-like, with many directions still available to reduce the loss.

A graph with blue dots

AI-generated content may be incorrect.

Figure 11 (Function): Scatter plot of minimal ratio vs. base loss across 100 runs. Most points fall between 0.35 and 0.60 with similar loss, showing multiple stable regions.

A graph with blue dots

AI-generated content may be incorrect.

Figure 12 (CIFAR-10): Scatter plot of minimal ratio vs. base loss on a CIFAR-10 subset. Ratios again cluster around 0.40–0.60, with similar losses across runs.

**Observations**

* **Function task:** Minimal ratios mostly fell between 0.35 and 0.60. Losses were tightly grouped, showing that different runs reached similar performance but with different levels of flatness.
* **CIFAR-10:** Ratios were also between 0.40 and 0.60 with tightly clustered losses. The relationship between loss and ratio was weak, likely because of the smaller model size and limited training budget.

**Comment**

* In these experiments, many stationary points had similar loss values but different ratios, suggesting that models can end up in regions of the parameter space that generalize differently even if the training loss looks the same.