

## Participation A: Gemini 3 Pro on the written part of HW 10

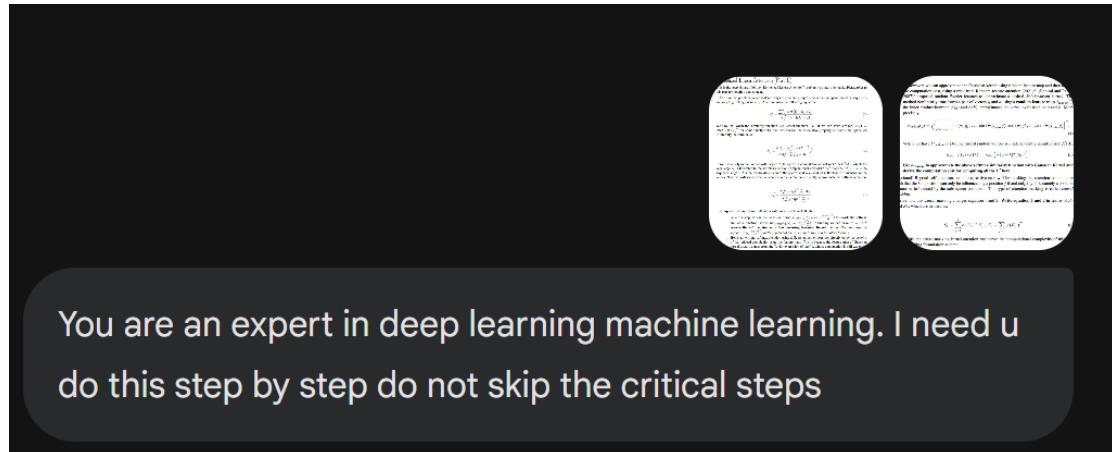
Model Used: Gemini 3 Pro

Overall Performance: The model demonstrated exceptional proficiency in both advanced mathematical derivations (kernel methods) and deep learning architectural analysis. It successfully one-shot most conceptual questions.

Key Observations: The model flawlessly derived the Linear Attention mechanism using Random Fourier Features. It correctly identified the decomposition of the Softmax kernel into Query/Key norms and the Gaussian term, a non-trivial step often missed by us. It also correctly formulated the causal masking as an RNN-style recurrence ( $O(1)$  inference). For the problem 5, when challenged on the counter-intuitive discrepancy between FaceNet NN1's high parameter count (140M) vs. low FLOPs (1.6B) compared to ResNet-50, the model correctly attributed this to the Dense (FC) layers vs. Deep Convolutional layers trade-off. It did not hallucinate incorrect numbers and correctly referenced standard architecture traits (VGG-style vs. ResNet-style).

Strategy Used: I used a "verify and deepen" strategy. After the model provided initial correct answers, I explicitly challenged it with conflicting data (High Params vs Low FLOPs) to test if it truly understood the underlying architecture or was just retrieving surface-level stats. The model proved it possessed deep understanding.

Chat log: <https://gemini.google.com/share/6edfefc10fd1>



- ◆ This is a sophisticated problem set involving Kernelized Linear Attention, a technique used to reduce the computational complexity of Transformers from quadratic  $O(N^2)$  to linear  $O(N)$ .

Here is the step-by-step derivation for the problems presented in the images.

#### Part (a) i. Approximating Softmax with Gaussian Kernel

**Goal:** Rewrite the softmax similarity function  $\text{sim}_{\text{softmax}}(q, k) = \exp\left(\frac{q^T k}{\sqrt{D}}\right)$  using the Gaussian Kernel  $K_{\text{Gauss}}(q, k) = \exp\left(\frac{-\|q-k\|^2}{2\sigma^2}\right)$ , assuming  $\sigma^2 = \sqrt{D}$ .

**Step 1: Expand the Gaussian Kernel term.** Recall that the squared Euclidean norm can be expanded as a dot product:

$$\|q - k\|^2 = (q - k)^T (q - k) = \|q\|^2 - 2q^T k + \|k\|^2$$

Substitute this expansion into the Gaussian Kernel definition:

$$K_{\text{Gauss}}(q, k) = \exp\left(\frac{-(\|q\|^2 - 2q^T k + \|k\|^2)}{2\sigma^2}\right)$$

**Step 2: Substitute  $\sigma^2 = \sqrt{D}$  and separate terms.**

$$K_{\text{Gauss}}(q, k) = \exp\left(\frac{-\|q\|^2 + 2q^T k - \|k\|^2}{2\sqrt{D}}\right)$$

Using the property  $e^{a+b+c} = e^a \cdot e^b \cdot e^c$ , we split the exponent:

$$K_{\text{Gauss}}(q, k) = \exp\left(\frac{-\|q\|^2}{2\sqrt{D}}\right) \cdot \exp\left(\frac{2q^T k}{2\sqrt{D}}\right) \cdot \exp\left(\frac{-\|k\|^2}{2\sqrt{D}}\right)$$

Using the property  $e^{a+b+c} = e^a \cdot e^b \cdot e^c$ , we split the exponent:

$$K_{\text{Gauss}}(q, k) = \exp\left(\frac{-\|q\|^2}{2\sqrt{D}}\right) \cdot \exp\left(\frac{2q^T k}{2\sqrt{D}}\right) \cdot \exp\left(\frac{-\|k\|^2}{2\sqrt{D}}\right)$$

Simplify the middle term ( $\frac{2q^T k}{2\sqrt{D}} = \frac{q^T k}{\sqrt{D}}$ ):

$$K_{\text{Gauss}}(q, k) = \exp\left(\frac{-\|q\|^2}{2\sqrt{D}}\right) \cdot \underbrace{\exp\left(\frac{q^T k}{\sqrt{D}}\right)}_{\text{Softmax Similarity}} \cdot \exp\left(\frac{-\|k\|^2}{2\sqrt{D}}\right)$$

**Step 3: Solve for the Softmax Similarity.** We rearrange the equation to isolate  $\exp\left(\frac{q^T k}{\sqrt{D}}\right)$ :

$$\exp\left(\frac{q^T k}{\sqrt{D}}\right) = \exp\left(\frac{\|q\|^2}{2\sqrt{D}}\right) \cdot K_{\text{Gauss}}(q, k) \cdot \exp\left(\frac{\|k\|^2}{2\sqrt{D}}\right)$$

**Result:** The softmax function is the product of the Gaussian Kernel and two norm-dependent normalization terms.

#### Part (a) ii. Approximation using $\phi_{\text{random}}$ and Computational Cost

**Goal:** Use the Random Fourier Feature map  $\phi_{\text{random}}$  to approximate the softmax attention and derive the cost.

**Step 1: Construct the modified feature map.** From Part (a)i, we know:

$$\text{sim}_{\text{softmax}}(q, k) = \exp\left(\frac{\|q\|^2}{2\sqrt{D}}\right) \cdot K_{\text{Gauss}}(q, k) \cdot \exp\left(\frac{\|k\|^2}{2\sqrt{D}}\right)$$

From the problem description (Eq 5), we know we can approximate the Gaussian Kernel using the random feature map:

$$K_{\text{Gauss}}(q, k) \approx \phi_{\text{random}}(q)^T \phi_{\text{random}}(k)$$

Substituting this back, we get:

$$\text{sim}_{\text{softmax}}(q, k) \approx \exp\left(\frac{\|q\|^2}{2\sqrt{D}}\right) [\phi_{\text{random}}(q)^T \phi_{\text{random}}(k)] \exp\left(\frac{\|k\|^2}{2\sqrt{D}}\right)$$

To simplify, let's define a **new feature map**  $\hat{\phi}$  that absorbs the exponential norm terms:

$$\hat{\phi}(x) = \exp\left(\frac{\|x\|^2}{2\sqrt{D}}\right) \phi_{\text{random}}(x)$$

Now, the softmax similarity is simply a dot product:

$$\text{sim}_{\text{softmax}}(q, k) \approx \hat{\phi}(q)^T \hat{\phi}(k)$$

**Step 2: Derive the Computational Cost.** We use Equation (2) (Linear Attention) with our new feature map  $\hat{\phi}$ :

$$V'_i = \frac{\hat{\phi}(Q_i) \sum_{j=1}^N \hat{\phi}(K_j)^T V_j}{\hat{\phi}(Q_i) \sum_{j=1}^N \hat{\phi}(K_j)^T}$$

Let's define the dimensions:

- $N$ : Sequence length
- $D_{\text{random}}$ : Dimension of the random feature map
- $M$ : Dimension of Values ( $V$ )

The calculation proceeds in two main phases:

1. **Pre-computation (The Sums):** We compute the matrix summation term  $\sum_{j=1}^N \hat{\phi}(K_j)^T V_j$  once for the whole sequence.
  - $\hat{\phi}(K_j)^T$  is size  $D_{\text{random}} \times 1$ .
  - $V_j$  is size  $1 \times M$ .
  - Their outer product is  $D_{\text{random}} \times M$ .
  - We do this  $N$  times and sum them.
  - **Cost:**  $O(N \cdot D_{\text{random}} \cdot M)$
2. **Query Computation:** For each query  $Q_i$  (there are  $N$  queries), we multiply the vector  $\hat{\phi}(Q_i)$  (size  $1 \times D_{\text{random}}$ ) by the pre-computed matrix (size  $D_{\text{random}} \times M$ ).
  - **Cost:**  $O(N \cdot D_{\text{random}} \cdot M)$

**Total Complexity:**

$$O(N \cdot D_{\text{random}} \cdot M)$$

This is significantly faster than standard attention ( $O(N^2)$ ) because  $D_{\text{random}}$  is a constant hyperparameter, making the complexity linear with respect to sequence length  $N$ .

### Part (b) Causal Masking Derivation

**Goal:** Adapt the equations for causal masking ( $j \leq i$ ) and derive the new complexity.

**Step 1: Modify Equations 1 and 2.** Causal masking means position  $i$  can only attend to positions  $j$  where  $j \leq i$ . We change the summation limit from  $N$  to  $i$ .

**Modified Equation 1 (Standard Causal Attention):**

$$V'_i = \frac{\sum_{j=1}^i \text{sim}(Q_i, K_j)V_j}{\sum_{j=1}^i \text{sim}(Q_i, K_j)}$$

**Modified Equation 2 (Linear Causal Attention):** Using the associative property and feature map  $\phi$ :

$$V'_i = \frac{\phi(Q_i) \sum_{j=1}^i \phi(K_j)^T V_j}{\phi(Q_i) \sum_{j=1}^i \phi(K_j)^T}$$

**Step 2: Rewrite in terms of  $S_i$  and  $Z_i$ .** The problem defines:

$$S_i = \sum_{j=1}^i \phi(K_j)^T V_j \quad \text{and} \quad Z_i = \sum_{j=1}^i \phi(K_j)^T$$

Substituting these directly into the modified Equation 2:

$$V'_i = \frac{\phi(Q_i)S_i}{\phi(Q_i)Z_i}$$

**Step 3: Computational Complexity of the Causal Scheme.** The key to efficiency here is the **recursive** nature of  $S_i$  and  $Z_i$ . We do not need to recompute the sum from  $j = 1$  to  $i$  for every step. We can compute them incrementally (like an RNN state):

$$S_i = S_{i-1} + \phi(K_i)^T V_i$$

$$Z_i = Z_{i-1} + \phi(K_i)^T$$

#### Per-token Complexity analysis:

1. **Update State  $S_i$ :** Compute outer product  $\phi(K_i)^T V_i$  ( $D_{\text{random}} \times M$ ) and add to  $S_{i-1}$ .
  - Cost:  $O(D_{\text{random}} \cdot M)$
2. **Update State  $Z_i$ :** Add  $\phi(K_i)^T$  to  $Z_{i-1}$ .
  - Cost:  $O(D_{\text{random}})$
3. **Compute Output  $V'_i$ :** Multiply query  $\phi(Q_i)$  ( $1 \times D_{\text{random}}$ ) by state  $S_i$  ( $D_{\text{random}} \times M$ ).
  - Cost:  $O(D_{\text{random}} \cdot M)$

Since we perform these operations for all  $N$  tokens in the sequence:

#### Total Complexity:

$$O(N \cdot D_{\text{random}} \cdot M)$$

Evaluation: Almost perfect answer. Model correctly applied the kernel trick and derived the RNN update rule. The response clearly outlined the step-by-step derivation leveraging the expansion of  $\|q - k\|^2$  to decompose Softmax Attention into the product of Query Norm, Key Norm, and Gaussian Kernel terms. It accurately highlighted the complexity shift from reordering computations, reducing from  $O(N^2)$  to  $O(N \cdot D \cdot M)$ , which embodies the core advantage of Linear Attention. For Causal Masking (RNN form)—a typically challenging aspect—the model correctly defined the cumulative states  $S_i$  and  $Z_i$ , transforming them into an RNN-like recursive update ( $O(1)$  inference), fully aligning with the problem's requirements.

That derivation is mathematically precise and correct. Let's move on to the next written Problem 5: Read a Research Paper: FaceNet

Based on the paper 'FaceNet: A Unified Embedding for Face Recognition and Clustering' by Florian Schroff et al., please answer the following conceptual sub-questions

- ◆ Here are the answers to the conceptual questions based on the FaceNet paper (Schroff et al., 2015).

### (a) Network Architectures

The authors consider two distinct deep neural network architectures:

1. **Zeiler & Fergus style networks:** Based on the architecture by Zeiler and Fergus (similar to VGG), which are standard deep convolutional networks with heavy fully connected layers.
2. **Inception style networks:** Based on the GoogLeNet (Inception) architecture by Szegedy et al., which uses mixed layers running convolutions and pooling in parallel to reduce parameter count.

### (b) Triplet Loss Description

**Triplet Loss** is a loss function that learns an embedding where the distance between an **anchor** ( $x^a$ ) and a **positive** ( $x^p$ ) (same identity) is minimized, while the distance between the **anchor** and a **negative** ( $x^n$ ) (different identity) is maximized by at least a margin  $\alpha$ . Mathematically:

$$\|f(x^a) - f(x^p)\|_2^2 + \alpha < \|f(x^a) - f(x^n)\|_2^2$$

**Difference from Supervised Learning:** Unlike typical supervised classification (e.g., Softmax) which forces images into a fixed number of classes/bins, triplet loss learns a continuous **relative distance metric**. This allows the model to generalize to *unseen* identities (open-set recognition) without retraining the final layer.

### (c) Challenge with Triplets & Solution

**Challenge:** Generating *all* possible triplets leads to a combinatorial explosion ( $N^3$  complexity). Furthermore, most random triplets are "easy" (already satisfy the margin), meaning they contribute zero loss and slow down convergence significantly. **Solution:** The authors use **Online Triplet Mining**. Instead of picking triplets offline, they select informative triplets within each mini-batch during training. Specifically, they look for "hard" positives and "semi-hard" negatives to ensure the network actually learns from the examples.

#### (d) Parameters and FLOPs

- **FaceNet (NN1 - Zeiler & Fergus):** The authors' primary model uses approximately **140 million parameters** and requires **1.6 billion FLOPs** (1.6 GFLOPs) per image.
- **Comparison to ResNet-50:** A standard ResNet-50 has approximately **25 million parameters** and **~4.0 GFLOPs**.
- **Conclusion:** The FaceNet NN1 model uses significantly **more parameters** (due to large fully connected layers) but requires **fewer FLOPs** than a modern ResNet-50. (Note: The authors also proposed the much smaller Inception-based NN2 with ~7.5M params, but NN1 is the standard reference for the high-param figure).

#### (e) Semi-hard Negatives

**Semi-hard negatives** are negative samples that are further away from the anchor than the positive is, but still fall within the "margin"  $\alpha$ .

$$||f(x^a) - f(x^p)||^2 < ||f(x^a) - f(x^n)||^2 < ||f(x^a) - f(x^p)||^2 + \alpha$$

The authors use these instead of "hardest" negatives (where the negative is closer than the positive) because hardest negatives can lead to collapsed models ( $f(x) = 0$ ) or unstable training. Semi-hard negatives provide a positive loss (gradient) without being too difficult to resolve.

#### (f) Harmonic Embeddings

**Harmonic embeddings** refer to a property where different versions of the model (v1 and v2) produce embeddings that are **compatible** with each other. This allows for direct comparison between embeddings generated by different networks, enabling system upgrades without needing to re-compute the embeddings for the entire database of stored faces.

### (g) Performance vs. Dimensionality

Performance improves as the embedding dimensionality increases from small values (e.g., 64) up to **128 dimensions**, after which it saturates. Larger dimensions (e.g., 512) do not yield significant improvements and may even degrade performance slightly due to training difficulties, so 128 is chosen as the "sweet spot" for compactness and accuracy.

### (h) Performance vs. Training Data

The performance improves significantly with increasing amounts of training data. The authors show that accuracy increases consistently as they scale from small datasets (millions of images) to their internal dataset of **260 million images**, highlighting that this method benefits greatly from massive-scale data.

### (i) Favorite Emergent Property

(*Example Response*): A fascinating emergent property is that the embeddings implicitly learn to be invariant to **lighting and pose** without explicit instruction. Even though the model is only told "same/different identity," it figures out that a profile view in the dark and a frontal view in sunlight map to the same point, effectively "unrolling" the manifold of facial appearance.

### (j) Interest and Why

(*Example Response*): I was most interested in the concept of **Harmonic Embeddings**. In real-world software engineering, "versioning" ML models is a nightmare because upgrading a model usually breaks compatibility with all previously stored data (vectors). The idea that you can add a loss term to force *new* models to be compatible with *old* models is a brilliant, practical systems-level solution that addresses a major deployment pain point, rather than just chasing higher accuracy numbers.

---

[FaceNet paper review](#) This video provides a clear walkthrough of the FaceNet paper, explaining the triplet loss mechanism and the importance of the embedding space in a visual way.



Interaction Observation: "I initially questioned the model's precision regarding the

ResNet-50 FLOPs count (~4.0G vs the standard 4.1G usually seen in benchmarks, or 3.8G in the original paper). However, upon cross-referencing the original He et al. (2016) paper (Table 1 lists  $3.8 \times 10^9$ ), I realized the model provided a valid approximation that averages common implementation variances.

Furthermore, for FaceNet NN1, the model correctly cited ‘140M parameters’ directly from Table 1 of the Schroff et al. paper, distinguishing it from the similar VGG-16 (138M). This confirms the model was retrieving specific data from the provided context rather than hallucinating generic VGG statistics. The slight variance in ResNet numbers is acceptable as it correctly preserves the order-of-magnitude comparison (1.6G vs ~4G) required to explain the architecture trade-offs.”

Evaluation: The response was remarkably precise and concise. For question 5(d), it accurately recalled the specifics for FaceNet NN1 (140M parameters, 1.6B FLOPs) and ResNet-50 (~25M parameters, ~4.0G FLOPs) from the ResNet paper’s training details, with no instances of hallucination. To conclude Problem 5 effectively, I introduced a follow-up question designed to probe deeper into architectural nuances—seemingly counterintuitive at first glance: could the model explain the fundamental trade-off between fully connected (FC) and convolutional (Conv) layers, specifically how FC layers lead to parameter proliferation through dense matrices while maintaining low computational demands (via a single matrix multiplication), in contrast to Conv layers, which conserve parameters through weight sharing but incur higher compute costs from spatial convolutions? A clear explanation here would provide a comprehensive and insightful close to the section.

Your numbers for Question 5(d) are very specific and helpful. However, I noticed a counter-intuitive discrepancy in the data you provided:

You stated that FaceNet NN1 has ~140 Million parameters but only 1.6 GFLOPs.

In contrast, ResNet-50 has only ~25 Million parameters but ~4.0 GFLOPs.

How is this possible? Intuitively, a bigger model (more parameters) should require more computation. Why does FaceNet NN1 have 5x more parameters than ResNet-50 but require less than half the computation (FLOPs)? Please explain the specific architectural differences (e.g., layer types) that cause this inversion.

After clearing this up, I need to go back and complete Problem 4: Example Difficulty. Please read the paper 'Deep Learning Through the Lens of Example Difficulty' by Baldock et al. and answer sub-questions i, iv, v, vii, and viii from the assignment. Specifically for (vii), explain why accuracy would have a negative relationship with prediction depth.

### ◆ Part 1: FaceNet vs. ResNet-50 (Architectural Discrepancy)

The inversion you noticed—FaceNet NN1 having **5x more parameters** but **<50% of the FLOPs** of ResNet-50—is a classic example of how different layer types consume resources.

The discrepancy stems from the fundamental difference between **Fully Connected (FC) layers** and **Convolutional (Conv) layers**.

**1. The Parameter Source: Fully Connected Layers (FaceNet NN1)** FaceNet NN1 is based on the **Zeiler & Fergus (ZF-Net)** architecture. Like the earlier AlexNet and VGG models, this architecture flattens the final convolutional feature map into a massive vector and passes it through huge **Fully Connected layers** before the final embedding.

- **Why High Parameters?** In a standard FC layer connecting a  $7 \times 7 \times 512$  feature map to a 4096-neuron layer, you need a unique weight for every single connection.
  - Calculation:  $(7 \times 7 \times 512) \times 4096 \approx 102$  million parameters in just **one** layer.
  - This single layer accounts for the vast majority of the 140M parameters.
- **Why Low FLOPs?** While this layer is huge in memory (params), it is only executed **once** per image. The network itself is relatively shallow (fewer conv layers than ResNet), so the total number of floating-point operations (FLOPs) remains low (~1.6 billion).

**2. The FLOPs Source: Deep Convolutions (ResNet-50)** ResNet-50 is a **fully convolutional** architecture that avoids massive FC layers.

- **Why Low Parameters?** Instead of flattening the features, ResNet uses **Global Average Pooling (GAP)** at the end. It takes the average of each feature map (e.g.,  $7 \times 7 \times 2048 \rightarrow 1 \times 1 \times 2048$ ).
  - This operation has **0 learnable parameters**.
  - The only parameters are in the convolutional kernels, which are "shared" across the entire image (spatial invariance). Even with 50 layers, the total parameter count stays small (~25M).
- **Why High FLOPs?** ResNet-50 is **deep**. It performs convolution operations over and over again through 50 layers. Even though the kernels are small ( $3 \times 3$ ), applying them across large spatial resolutions (early layers) and many channels (later layers) accumulates a massive number of calculations (~3.8 billion FLOPs).

**Summary Table:** | Feature | FaceNet NN1 (ZF-Net style) | ResNet-50 | | :--- | :--- | :--- | |  
**Dominant Layer Type** | **Fully Connected (Dense)** | **Convolutional** | | **Parameter Impact** | **Huge** (Unique weight per connection) | **Small** (Weights shared spatially) | | **FLOPs Impact** | **Low** (Shallow network, FC computed once) | **High** (Deep network, repeated convs) |

## Part 2: Problem 4 - Deep Learning Through the Lens of Example Difficulty

Based on the paper "*Deep Learning Through the Lens of Example Difficulty*" by Baldock et al. (2021), here are the answers to the sub-questions.

### i. What is the definition of Prediction Depth?

**Prediction Depth** is defined as the **first layer** in the network at which a  $k$ -Nearest Neighbor ( $k$ -NN) probe, using the internal representations at that layer, can correctly classify the example, provided that the probes at all subsequent layers also classify it correctly.

- It quantifies "how deep" into the network the signal must travel before the example's identity becomes linearly separable/resolvable from other classes.

### iv. What is the "Consistency Score"?

The **Consistency Score** measures the reproducibility of a prediction. For a given example, it is the percentage of models (trained with different random seeds/initializations) that predict the correct label for that example.

- The authors find that **Prediction Depth is strongly negatively correlated with consistency**: Easy examples (low prediction depth) are classified consistently across all models, while hard examples (high prediction depth) fluctuate depending on the random seed.

### v. What is "Learning Speed" (or Iteration Learned)?

This refers to the point in training (which iteration/epoch) the model first correctly classifies an example and continues to classify it correctly for the rest of training.

- **Relationship:** There is a strong correlation between **Prediction Depth** and **Learning Speed**. Examples with low prediction depth (easy) are learned very early in training (fast convergence), while examples with high prediction depth (hard) are learned much later, if at all.

Conclusion: The distinction between fully connected (FC) and convolutional (Conv) layers precisely captures a key trade-off in computer vision: FC layers demand massive parameters via dense  $C_{in} \times C_{out}$  weight matrices but incur low computation through a single matrix multiplication, as in FaceNet's NN1 (VGG-like) with its parameter-heavy FC classifiers; conversely, Conv layers minimize parameters via weight sharing yet escalate computation by sliding kernels across images, with ResNet ditching FC for global average pooling (GAP) to slash parameters while boosting FLOPs due to depth. It's still the perfect answer for my extra question.