

Model Used: Claude Sonnet 4.5

hw10.pdf

403 lines

PDF

- s You are an AI assistant that can solve Deep Neural Network problems. I will upload a pdf of all the problems, and then give you instructions for which problems to solve. Please make sure to show all your work and calculations in your output (including all equations you use to solve, etc.). If you are uncertain about a problem, please state what you are confused on rather than just hallucinating.

I've received your homework PDF for EECS 182 Deep Neural Networks. I can see it contains problems on:

1. Kernelized Linear Attention (Part II)
2. Coding Question: Hand-Design Transformers
3. Coding Question: Summarization
4. Example Difficulty and Early Exit
5. Read a Research Paper: FaceNet
6. Homework Process and Study Group

Please let me know which specific problem(s) you'd like me to help you solve, and I'll provide detailed solutions with all work shown, including:

- All equations and derivations
- Step-by-step calculations
- Clear explanations of the reasoning
- Any assumptions I'm making

I'm ready to work through whichever problem ↓ you need help with!

- s please solve part a) subpart i)

I'll solve Problem 1(a)(i): Rewriting the softmax similarity function using the Gaussian Kernel.

Problem 1(a)(i) Solution

Given:

- Gaussian Kernel: $K_{Gauss}(q, k) = \exp\left(\frac{-||q-k||^2}{2\sigma^2}\right)$
- Softmax similarity: $\text{sim}_{softmax}(q, k) = \exp\left(\frac{q^T k}{\sqrt{D}}\right)$
- Assumption: $\sigma^2 = \sqrt{D}$

Goal: Rewrite the softmax similarity function using the Gaussian Kernel.

Solution:

First, let me expand the squared norm in the Gaussian kernel:

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

Now, substitute this into the Gaussian kernel with $\sigma^2 = \sqrt{D}$:

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

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

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

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

Using properties of exponentials, we can split this into three terms:

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

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

Therefore, the softmax similarity function can be rewritten as:

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

This shows that the softmax similarity function equals the **Gaussian kernel multiplied by two normalization terms** that depend only on q and k individually (not on their interaction).

Analysis: Overall, Claude did a great job on part 1a) subpart i), arriving at the correct answer in a one-shot setting without any hallucinations. It clearly stated all the givens of the problem including the Gaussian kernel, softmax similarity, and the assumption that $\text{sqrt}(D) = \sigma^2$, and showed all algebraic steps, including the squared norm, substitution, and properties of exponentials, to reach the correct solution. However, since the answer key expressed the solution in terms of σ^2 rather than $\text{sqrt}(D)$, I prompted Claude to rewrite the answer accordingly, which it did correctly:

s

nice! can you also write this using sigma squared instead of sqrt(D)

Absolutely! Let me rewrite the solution using σ^2 instead of \sqrt{D} .

Problem 1(a)(i) Solution (with σ^2 notation)

Given:

- Gaussian Kernel: $K_{Gauss}(q, k) = \exp\left(\frac{-||q-k||^2}{2\sigma^2}\right)$
- Softmax similarity: $\text{sim}_{softmax}(q, k) = \exp\left(\frac{q^T k}{\sqrt{D}}\right)$
- Assumption: $\sigma^2 = \sqrt{D}$

Goal: Rewrite the softmax similarity function using the Gaussian Kernel.

Solution:

First, expand the squared norm in the Gaussian kernel:

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

Substitute this into the Gaussian kernel:

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

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

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

Split into three terms:

$$= \exp\left(\frac{-||q||^2}{2\sigma^2}\right) \cdot \exp\left(\frac{2q^T k}{2\sigma^2}\right) \cdot \exp\left(\frac{-||k||^2}{2\sigma^2}\right)$$

$$= \exp\left(\frac{-||q||^2}{2\sigma^2}\right) \cdot \exp\left(\frac{q^T k}{\sigma^2}\right) \cdot \exp\left(\frac{-||k||^2}{2\sigma^2}\right)$$

Therefore, the softmax similarity function can be rewritten as:

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

This representation makes it clear that the softmax similarity equals the **Gaussian kernel multiplied by two terms** that depend only on the individual norms of q and k .

Analysis: Although Claude had already performed most of these computations earlier, it recomputed the majority of the solution instead of simply substituting σ^2 for \sqrt{D} in the expression it had previously derived.

s great, now please solve part a) subpart ii)

I'll solve Problem 1(a)(ii): Approximating the softmax similarity function using random feature maps and deriving the computational cost.

Problem 1(a)(ii) Solution

Given:

- Random feature map: $\omega_{random}(q) = \sqrt{\frac{1}{D_{random}}} [\sin(w_1 q), \dots, \sin(w_{D_{random}} q), \cos(w_1 q), \dots, \cos(w_{D_{random}} q)]^\top$
- Where w_i are independently sampled from $\mathcal{N}(0, \sigma^2 I_D)$
- The approximation property: $\mathbb{E}_{w_i} [\omega(q) \cdot \omega(k)] = \exp\left(\frac{-||q-k||^2}{2\sigma^2}\right)$
- From part (i): $\text{sim}_{softmax}(q, k) = \exp\left(\frac{||q||^2}{2\sigma^2}\right) \cdot K_{Gauss}(q, k) \cdot \exp\left(\frac{||k||^2}{2\sigma^2}\right)$

Goal: Use ω_{random} to approximate the softmax similarity function and derive the computation cost for computing all V' .

Solution:**Step 1: Approximate the Gaussian Kernel**

Using the random feature map approximation:

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

Step 2: Approximate the Softmax Similarity

Substituting into our result from part (i):

$$\text{sim}_{softmax}(Q_{i\cdot}, K_{j\cdot}) = \exp\left(\frac{\|Q_{i\cdot}\|^2}{2\sigma^2}\right) \cdot K_{Gauss}(Q_{i\cdot}, K_{j\cdot}) \cdot \exp\left(\frac{\|K_{j\cdot}\|^2}{2\sigma^2}\right)$$

$$\approx \exp\left(\frac{\|Q_{i\cdot}\|^2}{2\sigma^2}\right) \cdot \omega_{random}(Q_{i\cdot})^T \omega_{random}(K_{j\cdot}) \cdot \exp\left(\frac{\|K_{j\cdot}\|^2}{2\sigma^2}\right)$$

Step 3: Define Modified Feature Maps

Let's define:

$$\tilde{\omega}(Q_{i\cdot}) = \exp\left(\frac{\|Q_{i\cdot}\|^2}{2\sigma^2}\right) \cdot \omega_{random}(Q_{i\cdot})$$

$$\tilde{\omega}(K_{j\cdot}) = \exp\left(\frac{\|K_{j\cdot}\|^2}{2\sigma^2}\right) \cdot \omega_{random}(K_{j\cdot})$$

Then:

$$\text{sim}_{softmax}(Q_{i\cdot}, K_{j\cdot}) \approx \tilde{\omega}(Q_{i\cdot})^T \tilde{\omega}(K_{j\cdot})$$

Step 4: Apply Kernelized Linear Attention Formula

From equation (2) in the problem, we can now write:

$$V'_{i\cdot} = \frac{\tilde{\omega}(Q_{i\cdot}) \sum_{j=1}^N \tilde{\omega}(K_{j\cdot})^T V_{j\cdot}}{\tilde{\omega}(Q_{i\cdot}) \sum_{j=1}^N \tilde{\omega}(K_{j\cdot})^T}$$

Step 5: Computational Cost Analysis

Now let's compute the cost for all V' (all N output positions):

Dimensions:

- $Q, K \in \mathbb{R}^{N \times D}$ (N sequence positions, D feature dimension)
- $V \in \mathbb{R}^{N \times M}$ (M value dimension)
- $\omega_{random}(Q_{i\cdot}), \omega_{random}(K_{j\cdot}) \in \mathbb{R}^{2D_{random}}$ (the feature map has $2D_{random}$ dimensions due to sin and cos components)
- $\tilde{\omega}(Q_{i\cdot}), \tilde{\omega}(K_{j\cdot}) \in \mathbb{R}^{2D_{random}}$

Step-by-step cost:

1. **Compute $\omega_{random}(Q)$ for all positions:** $O(N \cdot D \cdot D_{random})$
 - For each of N positions, compute D_{random} sin and cos operations on D-dimensional vectors
2. **Compute $\omega_{random}(K)$ for all positions:** $O(N \cdot D \cdot D_{random})$
3. **Compute norms and exponentials for $\tilde{\omega}(Q)$ and $\tilde{\omega}(K)$:** $O(N \cdot D)$ (negligible compared to other terms)
4. **Compute $\sum_{j=1}^N \tilde{\omega}(K_{j.})^T V_{j.}$:** $O(N \cdot 2D_{random} \cdot M)$
 - This is a matrix multiplication: $(N \times 2D_{random})^T \times (N \times M) = (2D_{random} \times M)$
5. **Compute $\sum_{j=1}^N \tilde{\omega}(K_{j.})^T$:** $O(N \cdot 2D_{random})$
 - Sum over N vectors of dimension $2D_{random}$
6. **For each position i, compute numerator and denominator:** $O(N \cdot 2D_{random} \cdot M)$
 - Numerator: $\tilde{\omega}(Q_{i.})$ (dimension $2D_{random}$) times the sum from step 4 (dimension $2D_{random} \times M$)
 - For all N positions: $O(N \cdot 2D_{random} \cdot M)$

Total Computational Cost:

$$O(N \cdot D \cdot D_{random} + N \cdot D_{random} \cdot M) = O(N \cdot D_{random} \cdot (D + M))$$

Or more simply, if D and M are comparable:

$$O(N \cdot D_{random} \cdot \max(D, M))$$

Comparison with standard softmax attention: Standard softmax attention has cost $O(N^2 \max(D, M))$, so this random feature approximation achieves significant speedup when $D_{random} \ll N$.

Analysis: Again, Claude was able to get the correct answer in a one-shot setting, which was very impressive! It also goes through each calculation step by step, and its computation of

wrandom(Q), wrandom(K), and the norms and exponentials of wrandom(Q) and wrandom(K) match the calculations performed in the answer key.

s now solve part b)

I'll solve Problem 1(b): Deriving the causal masking formulation for kernelized linear attention.

Problem 1(b) Solution

Given:

- Causal masking: position i can only be influenced by position j if and only if $j \leq i$
- Definitions: $S_{i\cdot} = \sum_{j=1}^i \omega(K_{j\cdot})^T V_{j\cdot}$ and $Z_{i\cdot} = \sum_{j=1}^i \omega(K_{j\cdot})^T$

Goal: Derive how causal masking changes equations (1) and (2), and express them in terms of $S_{i\cdot}$ and $Z_{i\cdot}$.

Step 1: Equation (1) with Causal Masking

Original equation (1):

$$V'_{i\cdot} = \frac{\sum_{j=1}^N \text{sim}(Q_{i\cdot}, K_{j\cdot}) V_{j\cdot}}{\sum_{j=1}^N \text{sim}(Q_{i\cdot}, K_{j\cdot})}$$

With causal masking (only attending to positions $j \leq i$):

$$V'_{i\cdot} = \frac{\sum_{j=1}^i \text{sim}(Q_{i\cdot}, K_{j\cdot}) V_{j\cdot}}{\sum_{j=1}^i \text{sim}(Q_{i\cdot}, K_{j\cdot})}$$

The key change: the summation upper limit changes from N to i .

Step 2: Equation (2) with Causal Masking

Original equation (2) (for kernel functions where $\text{sim}(Q_{i \cdot}, K_{j \cdot}) = \omega(Q_{i \cdot})\omega(K_{j \cdot})^T$):

$$V'_{i \cdot} = \frac{\omega(Q_{i \cdot}) \sum_{j=1}^N \omega(K_{j \cdot})^T V_{j \cdot}}{\omega(Q_{i \cdot}) \sum_{j=1}^N \omega(K_{j \cdot})^T}$$

With causal masking:

$$V'_{i \cdot} = \frac{\omega(Q_{i \cdot}) \sum_{j=1}^i \omega(K_{j \cdot})^T V_{j \cdot}}{\omega(Q_{i \cdot}) \sum_{j=1}^i \omega(K_{j \cdot})^T}$$

Step 3: Express in Terms of $S_{i \cdot}$ and $Z_{i \cdot}$

Using the given definitions:

- $S_{i \cdot} = \sum_{j=1}^i \omega(K_{j \cdot})^T V_{j \cdot}$
- $Z_{i \cdot} = \sum_{j=1}^i \omega(K_{j \cdot})^T$

Equation (1) in terms of $S_{i \cdot}$ and $Z_{i \cdot}$:

Since $\text{sim}(Q_{i \cdot}, K_{j \cdot}) = \omega(Q_{i \cdot})\omega(K_{j \cdot})^T$:

$$V'_{i \cdot} = \frac{\omega(Q_{i \cdot})S_{i \cdot}}{\omega(Q_{i \cdot})Z_{i \cdot}}$$

Equation (2) in terms of $S_{i \cdot}$ and $Z_{i \cdot}$:

This is the same expression:

$$V'_{i \cdot} = \frac{\omega(Q_{i \cdot})S_{i \cdot}}{\omega(Q_{i \cdot})Z_{i \cdot}}$$

Step 4: Computational Complexity Analysis

The key insight is that we can compute $S_{i\cdot}$ and $Z_{i\cdot}$ **recursively**:

$$S_{i\cdot} = S_{(i-1)\cdot} + \omega(K_{i\cdot})^T V_{i\cdot}$$

$$Z_{i\cdot} = Z_{(i-1)\cdot} + \omega(K_{i\cdot})^T$$

With base cases: $S_{0\cdot} = 0$ and $Z_{0\cdot} = 0$.

Computational cost for position i :

1. **Compute $\omega(K_{i\cdot})$:** $O(D \cdot D_{feature})$ where $D_{feature}$ is the dimension of the feature map
 - For polynomial kernel of degree 2: $D_{feature} = O(D^2)$
 - For random feature map: $D_{feature} = 2D_{random}$
2. **Update $S_{i\cdot}$:** $O(D_{feature} \cdot M)$
 - Compute $\omega(K_{i\cdot})^T V_{i\cdot}$ (outer product or matrix multiply)
3. **Update $Z_{i\cdot}$:** $O(D_{feature})$
 - Add $\omega(K_{i\cdot})^T$ to running sum
4. **Compute $V'_{i\cdot}$:** $O(D_{feature} \cdot M)$
 - Compute $\omega(Q_{i\cdot}) S_{i\cdot}$ and $\omega(Q_{i\cdot}) Z_{i\cdot}$

Total cost for all N positions:

For **polynomial kernel (degree 2)** with $D_{feature} = O(D^2)$:

$$O(N \cdot D^2 \cdot M)$$

For **random feature map** with $D_{feature} = 2D_{random}$:

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

Key advantage: The complexity is **linear in N** instead of quadratic!

- **Standard causal attention:** $O(N^2 \max(D, M))$
- **Causal kernelized attention:** $O(N \cdot D_{feature} \cdot M)$

When N is very large (long sequences), this provides significant computational savings since we avoid the $O(N^2)$ term entirely by using the recursive formulation.

Summary

The causal masking changes the summation from $\sum_{j=1}^N$ to $\sum_{j=1}^i$, which allows us to use a recursive update scheme with S_i . and Z_i . This reduces the computational complexity from $O(N^2)$ to $O(N)$ in the sequence length, making it highly efficient for long sequences in autoregressive settings.

Analysis: Claude seems to be representing all the phis in the problem with ws, but it gets the correct answer for how causal masking changes equation 1 and equation 2 to simplify equation 8. Claude also correctly reports that the complexity of linear transformers with causal masking is linear with respect to N (the sequence length), but provides a lot more in-depth computation to get to this answer rather than just using the insight that S_i and Z_i can be computed from S_{i-1} and Z_{i-1} in constant time.

facenet.pdf
1,081 lines

PDF

s now, I will upload a pdf of a research paper. please read that research paper thoroughly, and then answer each part of question 5 as I ask you to.

I've received and read the FaceNet paper thoroughly. I'm ready to answer the questions from Problem 5. Please go ahead and ask me each part, and I'll provide detailed answers based on the paper!

Just to confirm, I should answer:

- 5(a) through 5(j) as you ask them

Let me know which part you'd like me to start with!

s

please answer 5a)

Problem 5(a) Solution

Question: What are the two neural network architectures considered by the authors?

Answer:

The two neural network architectures considered by the authors are:

1. Zeiler & Fergus Architecture (with 1×1 convolutions)

This is described as **NN1** in the paper (see Table 1). It is based on the Zeiler & Fergus [22] model with additional $1 \times 1 \times d$ convolutional layers inspired by the "Network in Network" approach [9].

Key characteristics:

- **22 layers deep**
- **140 million parameters**
- **1.6 billion FLOPS per image**
- Uses multiple interleaved layers of convolutions, non-linear activations (ReLU), local response normalizations, and max pooling layers
- The 1×1 convolutional layers are inserted between the standard convolutional layers

2. GoogLeNet-style Inception Architecture

This architecture is based on the Inception model from Szegedy et al. [16], which was the winning approach for ImageNet 2014. The paper explores several variants of this architecture (NN2, NN3, NN4, NNS1, NNS2).

Key characteristics:

- **Much more parameter efficient:** 20× fewer parameters (around 6.6M-7.5M) compared to NN1
- **More computationally efficient:** Up to 5× fewer FLOPS (between 500M-1.6B)
- Uses **mixed layers** that run several different convolutional and pooling layers in parallel and concatenate their responses
- Some variants are dramatically reduced in size to run on mobile phones (e.g., NNS2 has only 4.3M parameters and 20M FLOPS)

The paper's largest Inception-based model is **NN2**, which has **7.5 million parameters** and **1.6 billion FLOPS** (see Table 2).

Analysis: Claude correctly identifies the two neural architectures considered by the paper as the Zeiler & Fergus model and the Inception model, clearly describing their authors and key characteristics. It accurately notes that the Zeiler & Fergus model uses multiple interleaved layers of convolutions, non-linear activations, local response normalizations, and max-pooling layers.

s

please answer 5b)

Problem 5(b) Solution

Question: Briefly describe the triplet loss and how it differs from a typical supervised learning objective.

Answer:

Triplet Loss Description

The triplet loss is designed to directly optimize the embedding space for face similarity. It operates on **triplets of images**:

1. **Anchor image** (x_i^a): A face image of a specific person
2. **Positive image** (x_i^p): Another face image of the **same person** as the anchor
3. **Negative image** (x_i^n): A face image of a **different person**

The Goal: The triplet loss aims to ensure that the anchor image is closer to the positive image than to the negative image by at least a margin α .

Mathematical Formulation:

The constraint the loss tries to enforce is:

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

The actual loss function being minimized is:

$$L = \sum_i^N [\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha]_+$$

where $[\cdot]_+$ means $\max(0, \cdot)$ and $f(x)$ is the embedding function that maps an image to a d-dimensional space (128-D in their experiments).

How it Differs from Typical Supervised Learning

Typical Supervised Learning (Classification):

- Uses a **classification layer** trained over a set of known identities
- Assigns each face to one of K identity classes
- Uses cross-entropy loss or softmax loss
- The network learns to output class probabilities
- To get embeddings, you must take an **intermediate bottleneck layer** as the representation
- This approach is **indirect** - you hope the bottleneck generalizes to new identities not seen during training
- Results in **large representation sizes** (often 1000s of dimensions)

Triplet Loss (FaceNet's Approach):

- **Directly optimizes the embedding itself** (the output of the network)
- Does not require a classification layer over identities
- Learns a **metric space** where distances directly correspond to face similarity
- The loss directly reflects what we want: same identity faces close together, different identity faces far apart
- Results in a **compact embedding** (128 dimensions)
- More **efficient and direct** - the learned representation is immediately usable for verification, recognition, and clustering
- Allows faces of one identity to live on a **manifold** rather than being projected to a single point
- **Generalizes naturally to new identities** not seen during training without any additional processing

The key philosophical difference is that triplet loss learns **relative similarities** between faces rather than absolute identity classifications, making it more flexible and producing more compact, generalizable representations.

Analysis: Claude clearly explains that the triplet loss ensures the anchor image is closer to the positive image than to the negative image by at least a margin of alpha, and correctly notes that it

does not require a classification layer. It also provides a helpful comparison between typical supervised learning and the triplet loss used in FaceNet, clarifying the differences between the two approaches. Overall, the response is very detailed and robust.

s please answer 5c)

Problem 5(c) Solution

Question: What is the challenge with generating all possible triplets? Briefly describe how the authors address this challenge.

Answer:

The Challenge with Generating All Possible Triplets

The fundamental challenge is that generating all possible triplets would result in:

1. **Computational Infeasibility:** With a training set of 8M identities and 100M-200M images, the total number of possible triplets would be astronomically large (on the order of $O(N^3)$ where N is the number of images). It would be infeasible to compute the argmin and argmax across the entire training set.
2. **Slow Convergence:** Many triplets would be "easy" triplets that already satisfy the constraint in Equation (1): $\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2$. These easy triplets don't violate the constraint, so they contribute zero to the loss and provide no learning signal. Passing them through the network wastes computational resources and slows down training.
3. **Poor Training Quality:** Including mislabeled or poorly imaged faces as hard positives/negatives could dominate the training and lead to poor model performance.

How the Authors Address This Challenge

The authors use a strategy called **online triplet mining** with the following approach:

1. Online Generation within Mini-batches

Instead of generating triplets offline across the entire dataset, they generate triplets **online during training** by selecting hard positives and hard negatives from **within each mini-batch**.

- Use **large mini-batches** (around 1,800 exemplars)
- Sample approximately **40 faces per identity** in each mini-batch
- Compute argmin and argmax only within the mini-batch (not across the entire dataset)

2. Smart Triplet Selection Strategy

For Positives: Use **all anchor-positive pairs** within a mini-batch (not just the hardest positive). The authors found this approach more stable and converged slightly faster.

For Negatives: Use **semi-hard negative mining** instead of the absolute hardest negatives.

3. Semi-Hard Negatives

The key innovation is selecting **semi-hard negatives** that satisfy:

$$||f(x_i^a) - f(x_i^p)||_2^2 < ||f(x_i^a) - f(x_i^n)||_2^2$$

Definition: Semi-hard negatives are negatives that are:

- **Farther from the anchor than the positive** (so they're not the absolute hardest)
- But still **close enough** that the squared distance is within the margin α
- They lie **inside the margin** but don't violate the basic ordering

Why semi-hard negatives?

- Selecting the **absolute hardest negatives** (those closer to the anchor than the positive) can lead to **bad local minima** early in training
- Specifically, it can cause the model to **collapse** (i.e., $f(x) = 0$ for all inputs)
- Semi-hard negatives provide meaningful gradient signal while avoiding training instability

This approach ensures **fast convergence** by focusing on triplets that are challenging but not impossibly hard, while keeping the computation tractable through mini-batch sampling.

Analysis: Claude correctly identifies that generating all possible triplets is computationally infeasible, wastes resources, and can degrade training quality. It also accurately notes that the authors use online triplet mining to address these issues. The response is very detailed, which enhances the reader's understanding.

s

please answer 5d)

Problem 5(d) Solution

Question: How many parameters and floating point operations (FLOPs) do the authors use for their neural network? How does this compare to a ResNet-50?

Answer:

FaceNet Model Parameters and FLOPs

The authors explore multiple architectures with varying complexity. Here are the main models:

Zeiler & Fergus Architecture (NN1):

- **Parameters:** 140 million
- **FLOPs:** 1.6 billion per image

Inception-based Architectures:

- **NN2** (largest Inception model, 224×224 input):
 - **Parameters:** 7.5 million
 - **FLOPs:** 1.6 billion per image

- **NN3** (Inception, 160×160 input):
 - **Parameters:** ~6.6-7.5 million (Inception family range)
 - **FLOPs:** ~1.6 billion
- **NN4** (Inception, 96×96 input):
 - **Parameters:** ~6.6-7.5 million
 - **FLOPs:** 285 million per image
- **NNS1** (mini Inception for mobile):
 - **Parameters:** 26 million
 - **FLOPs:** 220 million per image
- **NNS2** (tiny Inception for mobile):
 - **Parameters:** 4.3 million
 - **FLOPs:** 20 million per image

ResNet-50 Specifications

From the original ResNet paper (He et al., 2016), Table 1:

- **Parameters:** ~25.6 million (commonly cited as 25.5M)
- **FLOPs:** 3.8 billion (commonly cited as ~3.8-4.1 billion FLOPs)

Comparison

Model	Parameters	FLOPs	Comparison
FaceNet NN1	140M	1.6B	~5.5× more parameters than ResNet-50, but ~42% fewer FLOPs
FaceNet NN2	7.5M	1.6B	~3.4× fewer parameters than ResNet-50, and ~58% fewer FLOPs
ResNet-50	25.6M	3.8B	Baseline

Key Observations

1. **NN1 vs ResNet-50:** The Zeiler & Fergus architecture (NN1) has significantly more parameters (5.5× more) but uses fewer FLOPs (~42% of ResNet-50). This suggests NN1 has larger fully connected layers but less complex convolutional operations.
2. **NN2 vs ResNet-50:** The Inception-based NN2 is much more efficient than ResNet-50 in both parameters (~3.4× fewer) and FLOPs (~58% fewer), yet achieves comparable or better performance on the face recognition task. This demonstrates the efficiency of the Inception architecture.

3. **Efficiency Trade-offs:** FaceNet shows that the Inception-based models (NN2, NN3, NN4) achieve excellent face recognition performance with dramatically fewer parameters ($20\times$ fewer than NN1) while maintaining similar computational requirements. This makes them particularly suitable for deployment scenarios.
4. **Mobile-optimized variants:** The smallest FaceNet models (NNS2 with 4.3M parameters and 20M FLOPs) are orders of magnitude smaller than ResNet-50, enabling mobile deployment.

The comparison highlights that FaceNet's Inception-based architectures are significantly more parameter-efficient than both NN1 and ResNet-50, while the choice between NN1 and NN2 represents a trade-off between parameter count and slight accuracy differences.

Analysis: Claude correctly identifies the number of parameters and FLOPs used for the paper's neural network, and even goes further to cite the number of parameters and FLOPs used for the NN3, NN4, NNS1, and NNS2 architectures. The model is also able to identify the correct number of parameters and FLOPs used by ResNet-50 which was not provided in the FaceNet reading I supplemented Claude with, which was interesting to see. It also provides a much more detailed comparison between the paper's architecture and ResNet-50, which helps further supplement the reader's understanding.

s

please answer 5e)

Problem 5(e) Solution

Question: What do the authors mean by semi-hard negatives?

Answer:

Definition of Semi-Hard Negatives

Semi-hard negatives are negative examples (faces of different identities) that satisfy a specific distance relationship with respect to the anchor and positive examples.

Mathematically, a semi-hard negative x_i^n satisfies:

$$\|f(x_i^a) - f(x_i^p)\|_2^2 < \|f(x_i^a) - f(x_i^n)\|_2^2$$

where:

- x_i^a is the anchor image
- x_i^p is the positive image (same identity as anchor)
- x_i^n is the negative image (different identity)
- $f(\cdot)$ is the embedding function

Explanation

A semi-hard negative is a negative example that is:

1. **Farther from the anchor than the positive is** - This maintains the correct ordering: the positive is closer to the anchor than the negative
2. **But still within the margin α** - The distance to the negative is close enough that it lies inside the margin and therefore contributes to the loss
3. **Not the absolute hardest negative** - The hardest negatives would be those that are actually closer to the anchor than the positive, violating the basic ordering

Visual Understanding

Think of it in terms of distance ranges:

- **Easy negatives:** Very far from the anchor, satisfy $\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2$ (don't contribute to loss)
- **Semi-hard negatives:** Satisfy $\|f(x_i^a) - f(x_i^p)\|_2^2 < \|f(x_i^a) - f(x_i^n)\|_2^2 < \|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha$ (inside the margin, useful for training)
- **Hard negatives:** Violate ordering with $\|f(x_i^a) - f(x_i^n)\|_2^2 \leq \|f(x_i^a) - f(x_i^p)\|_2^2$ (too hard, can cause training instability)

Why Use Semi-Hard Negatives?

The authors choose semi-hard negatives for important practical reasons:

1. **Avoid Training Collapse:** Selecting the absolute hardest negatives (those closer to the anchor than the positive) can lead to **bad local minima** early in training. Specifically, it can cause the model to collapse where $f(x) = 0$ for all inputs.
2. **Stable Training:** Semi-hard negatives provide a strong learning signal (they're still challenging) while maintaining training stability.
3. **Meaningful Gradients:** Semi-hard negatives lie within the margin, so they actively contribute to the loss and provide useful gradient information to improve the model.
4. **Progressive Difficulty:** As training progresses and the model improves, what was once semi-hard becomes easy, and the model naturally finds new semi-hard examples, creating a curriculum-like learning process.

In summary, semi-hard negatives strike the optimal balance between being challenging enough to provide useful training signal while being safe enough to avoid training instabilities that could cause the model to collapse.

Analysis: Claude correctly defines a semi-hard negative and explains why it's useful, while also providing clear mathematical intuition that helps the reader understand the concept more fully. I actually think the answer key is incorrect for this part since it states that semi hard negatives are "negative examples in a triplet that are closer to the anchor image than the positive example," which is the opposite of how the paper (and Claude!) defines it.

s

please answer 5f)

Problem 5(f) Solution

Question: What are harmonic embeddings?

Answer:

Definition of Harmonic Embeddings

Harmonic embeddings are a set of embeddings generated by **different models** (e.g., v1 and v2) that are **compatible** with each other, meaning they can be directly compared in the same embedding space.

More specifically, harmonic embeddings allow embeddings from an older model version (v1) to be meaningfully compared with embeddings from a newer, improved model version (v2) using the same distance metrics.

The Problem Harmonic Embeddings Solve

In typical scenarios:

- Model v1 generates embeddings for a large dataset of images
- Later, an improved model v2 is developed with better accuracy
- **Problem:** The embeddings from v1 and v2 live in incompatible spaces and cannot be directly compared
- **Solution needed:** Either re-compute all embeddings with v2 (computationally expensive) or ensure compatibility

How Harmonic Embeddings Work

The authors achieve harmonic compatibility by training the new model (v2) with a special **harmonic triplet loss** that mixes embeddings from both versions:

Harmonic Triplet Loss

The training procedure generates triplets that combine v1 and v2 embeddings in various ways (see Figure 9 in the paper):

- Anchors can be from v1 or v2
- Positives can be from v1 or v2
- Negatives can be from v1 or v2
- Semi-hard negatives are selected from the **combined set** of both v1 and v2 embeddings

Training Procedure

1. Start with an independently trained improved model (NN2)
2. Initialize v2 from this model
3. **First stage:** Retrain only the last layer (embedding layer) from random initialization using the harmonic triplet loss
4. **Second stage:** Continue training the entire v2 network with the harmonic loss

Benefits of Harmonic Embeddings

1. **Smooth Upgrade Path:** When rolling out a new model version, you don't need to recompute embeddings for the entire existing database immediately
2. **Backward Compatibility:** New embeddings (v2) can be compared directly to old embeddings (v1) without conversion
3. **Intermediate Performance:** When comparing v2 embeddings to v1 embeddings, performance is intermediate between pure v1 and pure v2 (see Figure 8):
 - v1 vs v1: baseline performance
 - v2 vs v2: best performance (improved model)
 - v1 vs v2 (mixed): intermediate performance, but still better than v1 alone
4. **Gradual Migration:** Allows for a gradual transition from old to new embeddings without version incompatibility issues

Conceptual Interpretation

Figure 10 in the paper provides intuition: the v2 embeddings are learned such that most are placed near their corresponding v1 embeddings, maintaining compatibility. However, incorrectly placed v1 embeddings can be "perturbed" slightly in v2 to improve their location in the embedding space, thus improving overall verification accuracy while maintaining compatibility.

Practical Application

In a real-world deployment:

- A company has computed embeddings for billions of faces using model v1
- They develop an improved model v2
- Instead of recomputing all embeddings (expensive), they train v2 to be harmonic with v1
- New faces get v2 embeddings that can be compared to existing v1 embeddings
- Over time, they can gradually migrate to all v2 embeddings

This is particularly valuable for large-scale face recognition systems where re-embedding entire databases is prohibitively expensive.

Analysis: Claude again correctly defines harmonic embeddings as embeddings produced by different models (v1 and v2) that remain comparable within the same embedding space. It also provides additional interpretation of these embeddings in the corresponding figures of the paper, information I didn't ask for but found interesting nonetheless.

s

please answer 5g

Problem 5(g) Solution

Question: How does the performance vary with embedding dimensionality?

Answer:

Experimental Setup

The authors tested the model NN1 (Zeiler & Fergus architecture) with different embedding dimensionalities on their hold-out test set (section 4.1). They measured the validation rate (VAL) at 10^{-3} false accept rate (FAR).

Results

Table 5 in the paper shows the following results:

Embedding Dimensionality	VAL @ 10^{-3} FAR	Standard Error
64 dimensions	86.8%	$\pm 1.7\%$
128 dimensions	87.9%	$\pm 1.9\%$
256 dimensions	87.7%	$\pm 1.9\%$
512 dimensions	85.6%	$\pm 2.0\%$

Key Observations

- 128 dimensions is optimal:** The best performance is achieved at 128 dimensions (87.9% $\pm 1.9\%$), which is why the authors selected this dimensionality for all their other experiments.
- Performance is relatively stable:** The differences between 64, 128, and 256 dimensions are **statistically insignificant** given the standard errors. All three perform comparably in the 86-88% range.
- Larger is not always better:** Surprisingly, 512 dimensions performs **worse** (85.6%) than the smaller dimensionalities. The authors note that larger embeddings may require more training time to achieve the same accuracy.
- Diminishing returns:** Increasing from 64 to 128 dimensions provides a modest improvement (~1.1%), but doubling again to 256 dimensions provides no additional benefit (actually slightly worse at 87.7%).
- Compact representation is sufficient:** The results demonstrate that a relatively compact 128-dimensional embedding is sufficient to capture the discriminative information needed for face recognition.

Practical Implications

Why 128 dimensions is ideal:

1. **Good accuracy:** Achieves the best performance in their experiments
2. **Compact storage:** During training, they use 128-dimensional float vectors, but these can be **quantized to 128 bytes** (one byte per dimension) without loss of accuracy
3. **Memory efficient:** Each face is represented by only 128 bytes, making it ideal for:
 - Large-scale clustering
 - Large-scale recognition systems
 - Storage of billions of face embeddings
4. **Mobile deployment:** Even smaller embeddings (64 dimensions) are possible with only minor accuracy loss, which could be useful for mobile devices with strict memory constraints

Additional Context

The authors note that:

- The embedding lives on a **d-dimensional hypersphere** (normalized to unit length: $\|f(x)\|_2 = 1$)
- This normalization constraint may influence the optimal dimensionality
- The choice of 128 dimensions balances performance, compactness, and practical deployment considerations

In summary, **performance is relatively insensitive to embedding dimensionality** in the range of 64-256 dimensions, with **128 dimensions providing the best balance** of accuracy and compactness. This is a remarkably efficient representation compared to previous approaches that used thousands of dimensions.

Analysis: Claude correctly notes that 128 dimensions is optimal and highlights the diminishing returns that occur when increasing the embedding size beyond 256 dimensions. It also points out that the performance differences among 64, 128, and 256 dimensions are statistically insignificant. Additionally, Claude reaches the same conclusion as the answer key that larger embedding dimensions may require more training time to reach the same accuracy (even though I didn't explicitly ask for that reasoning), which was great to see!

s please answer 5h)

Problem 5(h) Solution

Question: How does the performance vary with increasing amounts of training data?

Answer:

Experimental Setup

The authors evaluated the impact of training data size using a smaller model with 96×96 pixel inputs. The model architecture is similar to NN2 (Inception-based) but without the 5×5 convolutions in the Inception modules. All models were trained for 700 hours and evaluated on their personal photos test set (section 4.2), measuring the validation rate (VAL) at 10^{-3} false accept rate.

Results

Table 6 in the paper shows the following results:

Training Data Size	VAL @ 10^{-3} FAR
2.6 million images	76.3%
26 million images	85.1%
52 million images	85.1%
260 million images	86.2%

Key Observations

1. Dramatic improvement from millions to tens of millions:

- Going from 2.6M to 26M images ($10\times$ increase) produces a **huge jump** from 76.3% to 85.1%
- This represents an **8.8 percentage point improvement**
- The authors note this is approximately a **60% relative reduction in error rate**

2. Plateau at tens of millions:

- Doubling from 26M to 52M images shows **no improvement** (both at 85.1%)
- This suggests the model has largely saturated its capacity at this training set size

3. Small gains from hundreds of millions:

- Going from 52M to 260M images ($5\times$ increase) yields only a **modest improvement** to 86.2%
- This is just a **1.1 percentage point gain** for a massive increase in data
- The improvement **tapers off** significantly

4. Logarithmic-like returns:

- The performance gains follow a pattern of diminishing returns
- Each order of magnitude increase in data provides progressively smaller benefits

Analysis

Why the pattern?

1. **Early gains (2.6M → 26M):** The model is data-starved and benefits enormously from more diverse examples of faces in different poses, lighting conditions, expressions, etc.
2. **Plateau (26M → 52M):** The model has seen enough diversity that additional similar examples provide diminishing value. The network architecture may be limiting further gains.
3. **Tapering (52M → 260M):** At this scale, the additional data captures increasingly rare or edge cases. While helpful, the incremental benefit is small.

Important Caveats:

The authors note this was evaluated on a **smaller model** due to time constraints. They explicitly state:

"the effect may be even larger on larger models"

This suggests that larger, more complex models (like NN1 or NN2) might:

- Continue to benefit more substantially from hundreds of millions of images
- Have higher capacity to learn from the additional data
- Not plateau as quickly as this smaller model

Practical Implications

1. **Minimum data requirements:** At least 26 million images are needed for strong performance on face recognition
2. **Massive data helps:** Using 100M-200M images (as FaceNet does in most experiments) provides the best results, even if the incremental gains are smaller
3. **Scale matters:** The 60% error reduction from 2.6M to 26M images demonstrates why large-scale datasets are crucial for face recognition
4. **Cost-benefit trade-off:** Organizations must balance the cost of collecting/labeling hundreds of millions of images against the relatively modest additional performance gains

Context in Deep Learning

This result aligns with general deep learning principles:

- Deep networks are data-hungry
- Performance typically improves logarithmically with data (not linearly)
- Larger models can better leverage larger datasets
- There's a point where architectural improvements matter more than additional data

In summary, **training data size has a massive impact on performance**, with the most dramatic improvements coming from scaling from millions to tens of millions of images. While hundreds of millions of images provide additional benefits, the gains taper off significantly, though **larger models might continue to benefit more** than the smaller model tested here.

Analysis: Claude correctly notes that performance increases by 8.8% when scaling from 2.6M to 26M images, whereas increasing to 260M yields only an additional 1.1% gain. It also discusses the practical implications of these results and how they relate to broader deep learning principles. Again, this was something I didn't directly ask for but appreciated, as it added valuable context and improved my understanding as a reader.

s

please answer 5i)

Problem 5(i) Solution

Question: Briefly share your favorite emergent property/result of the learned behavior with a triplet loss from the paper.

Answer:

My favorite emergent property from the FaceNet paper is the **remarkable invariance to extreme variations in pose, lighting, age, and occlusion** demonstrated by the face clustering results in Figure 7.

What Makes This Special

Figure 7 shows a cluster of faces from a single user's personal photo collection that were automatically grouped together using agglomerative clustering on the learned embeddings. What's striking is the incredible diversity within this single cluster:

- **Dramatic lighting variations:** From harsh shadows to bright illumination
- **Extreme pose changes:** Frontal views, profile views, looking up/down
- **Age progression:** The same person at different ages spanning years
- **Occlusions:** Glasses, hands covering parts of the face, accessories
- **Different expressions:** Smiling, serious, various emotional states
- **Different contexts:** Indoor/outdoor, different backgrounds

Why This Is My Favorite

1. **Truly emergent behavior:** The network was never explicitly told to be invariant to these factors. It learned these invariances purely from the triplet loss objective and the data. The only instruction was "make same-person faces closer than different-person faces" - everything else emerged naturally.
2. **Practical magic:** This is exactly what you'd want from a real-world face recognition system. Users don't carefully pose for consistent photos - they have decades of casual photos in wildly different conditions. The fact that FaceNet can cluster them correctly is genuinely impressive.
3. **Manifold learning:** This demonstrates that the triplet loss allows "faces for one identity to live on a manifold" as the authors describe. Rather than collapsing all instances of a person to a single point (as classification-based approaches might), the embedding preserves the natural variations while maintaining the critical similarity structure.
4. **End-to-end simplicity:** All of this sophisticated invariance comes from a simple, elegant loss function and no complex preprocessing (no 3D alignment, just tight crops). The paper even mentions these PIE dataset image pairs were "previously considered to be very difficult for face verification systems."
5. **Real-world validation:** Figure 1 showing the distance scores between same-person and different-person pairs across various pose/illumination combinations (with a simple threshold of 1.1 separating them perfectly) is a beautiful validation that the learned metric space actually works as intended.

This emergent property exemplifies what deep learning does best: learning complex, high-dimensional invariances that would be extremely difficult to hand-engineer, through a simple objective function applied to large-scale data.

Analysis: Interesting to see what Claude answered for this question! It not only answers the question but also provides reasoning for why the emergent property of invariance to extreme variations in pose, lighting, age, and occlusion is significant and noteworthy.

s please answer 5j)

Problem 5(j) Solution

Question: Which approach taken by the authors interested you the most? Why? (≥ 100 words)

Answer:

The approach that interested me most is the **online semi-hard negative mining strategy** within mini-batches. This technique elegantly solves a fundamental tension in metric learning: you need hard examples to learn effectively, but the absolute hardest examples can destabilize training.

What makes this approach fascinating is its curriculum-like nature without explicit curriculum design. By selecting negatives that are harder than the positive but not violating the ordering constraint, the method creates a naturally adaptive difficulty progression. As the model improves, yesterday's semi-hard negatives become easy, and the model automatically finds new semi-hard examples from the mini-batch. This creates a self-paced learning dynamic.

The practical engineering insight is equally impressive: computing argmin/argmax across millions of images is infeasible, but doing it within large mini-batches (around 1,800 exemplars with ~40 faces per identity) is tractable and provides sufficient diversity. This shows how the authors balanced theoretical ideals with computational reality.

The contrast with their finding that using all anchor-positive pairs (rather than just hard positives) worked better demonstrates that the negative selection is the critical component. This asymmetry—be aggressive with negatives but conservative with positives—is a valuable insight for metric learning more broadly.

Finally, the warning about training collapse (where $f(x) = 0$ for all inputs) from selecting absolute hardest negatives highlights the importance of understanding optimization dynamics beyond just the mathematical formulation of the loss. This practical wisdom about what can go wrong during training and how to prevent it represents the kind of hard-won knowledge that distinguishes successful deployed systems from theoretical papers.

The semi-hard mining strategy exemplifies excellent research: theoretically motivated, practically implementable, and generalizable beyond the specific application domain.

Analysis: Interesting to see what Claude answered for this question! Like the previous part, it not only answers the question by identifying its favorite approach from the paper (the online semi-hard negative mining strategy) but also explains why this strategy is relevant and noteworthy.