



**KA** You are a teaching assistant in a Deep Learning class at UC Berkeley. I have attached a question from a HW10 about kernelized linear attention. Solve Q1 a) i. and explain the reasoning. Make sure to check your work.

Decomposed Gaussian kernel to express softmax similarity function.

20s

## Solution to Q1(a)(i): Rewriting Softmax Similarity Using Gaussian Kernel

### Given Information

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

### Solution

Step 1: Expand the squared norm in the Gaussian kernel

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



Step 2: Substitute into the Gaussian kernel

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



Step 3: Use properties of exponentials to separate terms

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



Step 4: Apply the assumption  $\sigma^2 = \sqrt{D}$

With this substitution, the middle term becomes:

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



So we have:

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



Step 5: Solve for the softmax similarity

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



### Verification

Multiplying out the right-hand side:

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

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



### Interpretation

The softmax similarity decomposes into three factors:

1. Gaussian Kernel  $K_{\text{Gauss}}(q, k)$  — captures the distance-based similarity
2. Query term  $\exp\left(\frac{\|q\|^2}{2\sqrt{D}}\right)$  — depends only on  $q$
3.  $\exp\left(\frac{\|k\|^2}{2\sqrt{D}}\right)$  — depends only on  $k$

3. Key term  $\exp\left(\frac{\|q\|^2}{2\sqrt{D}}\right)$  — depends only on  $k$

This decomposition is useful because it shows that softmax attention can be viewed through the lens of kernel methods, though as part (ii) notes, directly using the Gaussian kernel's feature map doesn't immediately yield computational benefits.

Excellent! Can you please solve Q1 a) ii. as well now? Remember to check your work.

Analyzed random feature approximation and derived computational complexity.

## Solution to Q1(a)(ii): Random Feature Approximation and Computational Cost

### Goal

Use the random feature map  $\phi_{\text{random}}$  to approximate the softmax similarity function, then derive the computational cost for computing all  $V'$ .

### Step 1: Recall the Setup

From part (i), we showed:

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

The random feature map is:

$$\phi_{\text{random}}(q) = \sqrt{\frac{1}{D_{\text{random}}}} [\sin(\omega_1^T q), \dots, \sin(\omega_{D_{\text{random}}}^T q), \cos(\omega_1^T q), \dots, \cos(\omega_{D_{\text{random}}}^T q)]^T$$

where  $\omega_i \sim \mathcal{N}(0, \sigma_D^2 I)$ , and critically:

$$\mathbb{E}_\omega[\phi_{\text{random}}(q)^T \phi_{\text{random}}(k)] = \exp\left(-\frac{\|q - k\|^2}{2\sigma^2}\right) = K_{\text{Gauss}}(q, k)$$

### Step 2: Define a Modified Feature Map for Softmax

To approximate softmax similarity, we define:

$$\psi(q) = \exp\left(\frac{\|q\|^2}{2\sqrt{D}}\right) \cdot \phi_{\text{random}}(q)$$

→ good notation

$$\psi(k) = \exp\left(\frac{\|k\|^2}{2\sqrt{D}}\right) \cdot \phi_{\text{random}}(k)$$

Then:

scalar scalar

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

✓

### Step 3: Apply to Attention (Using Associativity)

The attention output is:

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

✓

Using our feature map approximation:

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

✓

Using associativity (the key insight from equation 2):

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

✓

### Step 4: Computational Cost Analysis

Let me define dimensions:

One-shot  
the first problem  
good notation  
no errors.  
Read the question correctly.

- $N$  = sequence length
- $D$  = dimension of queries/keys
- $M$  = dimension of values
- $D_{\text{random}}$  = number of random features (so  $\psi$  has dimension  $2D_{\text{random}}$ )

Cost Breakdown:

Step	Operation	Cost
1	Compute $\phi_{\text{random}}(Q_i)$ for all $i$ : requires $\omega_i^T Q$ for each random vector	$O(N \cdot D_{\text{random}} \cdot D)$
2	Compute $\phi_{\text{random}}(K_j)$ for all $j$	$O(N \cdot D_{\text{random}} \cdot D)$
3	Compute scaling factors $\exp(\ Q_i\ ^2/2\sqrt{D})$ and $\exp(\ K_j\ ^2/2\sqrt{D})$	$O(N \cdot D)$
4	Compute $S = \sum_{j=1}^N \psi(K_j) V_j^T$ — matrix of size $(2D_{\text{random}} \times M)$	$O(N \cdot D_{\text{random}} \cdot M)$
5	Compute $Z = \sum_{j=1}^N \psi(K_j)$ — vector of size $2D_{\text{random}}$	$O(N \cdot D_{\text{random}})$
6	For each $i$ : compute $\psi(Q_i)^T S$ and $\psi(Q_i)^T Z$	$O(N \cdot D_{\text{random}} \cdot M)$

Total Computational Cost:

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

### Verification & Comparison

Original softmax attention:  $O(N^2 \cdot \max(D, M))$  — quadratic in sequence length

Random feature attention:  $O(N \cdot D_{\text{random}} \cdot (D + M))$  — linear in sequence length!

This is the key benefit: for long sequences where  $N \gg D_{\text{random}}$ , we get significant speedup. The tradeoff is approximation quality, which improves as  $D_{\text{random}}$  increases.

### Summary

The random feature map transforms the problem from computing  $N^2$  pairwise similarities to:

1. Pre-computing two aggregates ( $S$  and  $Z$ ) in  $O(N)$  time
2. Using these aggregates for each query in  $O(1)$  time per query

This "kernel trick in reverse" (explicit feature maps) enables linear-time attention.



KA Excellent. Now I am attaching another problem from the HW. This involves reading a paper and answering questions about it. I have attached a pdf of the paper for you as well. Read the paper carefully and answer the questions.

Catalogued FaceNet paper questions and prepared systematic responses.

## FaceNet Paper: Question Answers

I've read through the FaceNet paper carefully. Here are the answers to each question:

(a) What are the two neural network architectures considered by the authors?

The authors consider two deep convolutional network architectures:

1. **Zeiler&Fergus-style architecture (NN1)**: Based on the model from Zeiler & Fergus [22], consisting of multiple interleaved layers of convolutions, non-linear activations, local response normalizations, and max pooling layers. They additionally add  $1 \times 1 \times d$  convolution layers inspired by the "Network in Network" work. This model has 10 million parameters and requires 1.6 billion FLOPS.
2. **Inception-style architecture (NN2, NN3, NN4, NNS1, NNS2)**: Based on the GoogLeNet Inception model from Szegedy et al. [16]. These use mixed layers that run several different convolutional and pooling layers in parallel and concatenate their responses. These models have 20x fewer parameters (around 6.6M-7.5M) and up to 5x fewer FLOPS.

(b) Briefly describe the *triplet loss* and how it differs from a typical supervised learning objective.

Triplet Loss Description:

Expanding on the details of this computation would be helpful

Good and correct conclusion

The triplet loss operates on triplets of images: an **anchor** ( $x_i^a$ ), a **positive** ( $x_i^p$ ) of the same identity as the anchor, and a **negative** ( $x_i^n$ ) of a different identity. The loss enforces that the anchor is closer to the positive than to the negative by at least a margin  $\alpha$ :

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

Notation could be better, would've liked to see the inequality first

How it differs from typical supervised learning:

Typical Supervised Learning	Triplet Loss
Uses a classification layer trained over known identities	Directly optimizes the embedding space
Encourages all faces of one identity to collapse to a single point	Allows faces of one identity to live on a manifold while enforcing distance to other identities
Requires fixed set of classes at training time	Generalizes to new identities not seen during training
Often requires post-processing (PCA, SVM) on bottleneck features	Embedding is directly usable for verification/clustering

(c) What is the challenge with generating all possible triplets? Briefly describe how the authors address this challenge.

The Challenge:

Generating all possible triplets would result in an enormous number of triplets ( $O(N^3)$  for  $N$  images), most of which would be "easy" triplets that already satisfy the constraint  $\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2$ . These easy triplets:

- Don't contribute to training (zero gradient)
- Slow down convergence
- Waste computation

The Authors' Solution:

1. **Online triplet mining:** Instead of generating triplets offline, they generate triplets online within large mini-batches (~1,800 exemplars).
2. **Smart batch construction:** They sample ~40 faces per identity per mini-batch to ensure meaningful anchor-positive pairs.
3. **Semi-hard negative selection:** Rather than selecting the hardest negatives (which can cause collapsed models early in training), they select semi-hard negatives that satisfy:  $\|f(x_i^a) - f(x_i^p)\|_2^2 < \|f(x_i^a) - f(x_i^n)\|_2^2 < \|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha$
4. **Use all anchor-positive pairs:** They use all valid anchor-positive pairs within a mini-batch while being selective about negatives.

lost information

not in paper directly but inferred expression correctly

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

FaceNet Models:

Model	Parameters	FLOPS
NN1 (Zeiler&Fergus)	140M	1.6B
NN2 (Inception)	7.5M	1.6B
NN3 (Inception 160×160)	~7.5M	~500M-1.6B
NNS1 (mini Inception)	26M	220M
NNS2 (tiny Inception)	4.3M	20M

big orange should be around 1B from Fig k.

ResNet-50 Comparison:

Based on the original ResNet paper (He et al., 2015), ResNet-50 has approximately **25.6** million parameters and requires about 3.8-4.1 billion FLOPs (for 224×224 input).

Metric	NN1	NN2	ResNet-50
Parameters	140M	7.5M	~25.6M
FLOPS	1.6B	1.6B	~3.8-4.1B

Comparison Summary:

- NN1 has ~5.5× more parameters than ResNet-50 but fewer FLOPS
- NN2 (Inception) has ~3.4× fewer parameters than ResNet-50 and ~2.5× fewer FLOPS
- The Inception architecture is remarkably efficient in terms of parameters

### (e) What do the authors mean by *semi-hard* negatives?

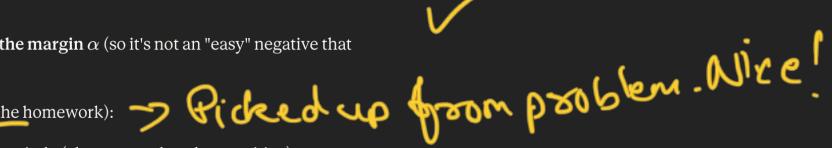
Semi-hard negatives are negative examples that satisfy:

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

Interpretation:

- The negative is **farther from the anchor than the positive** (so it's not a "hard" negative that violates the constraint)
- But the negative is **still within the margin  $\alpha$**  (so it's not an "easy" negative that contributes nothing)

Visually (referring to Figure 1 in the homework):

- 
- **Hard negatives:** Inside the inner circle (closer to anchor than positive) — can cause training collapse
  - **Semi-hard negatives:** Between the inner circle and the margin boundary — ideal for stable training
  - **Easy negatives:** Outside the margin — contribute zero gradient

The authors found that using purely hard negatives early in training can cause the model to collapse to  $f(x) = 0$ , making semi-hard negatives a more stable choice.





### (f) What are harmonic embeddings?

Harmonic embeddings are embeddings generated by different model versions (e.g., v1 and v2) that are **compatible** in the sense that they can be directly compared to each other.

Purpose:

This solves the **model upgrade problem**: When you have a large database of face embeddings computed with model v1, and you want to deploy an improved model v2, you normally would need to recompute all embeddings. With harmonic embeddings, v2 embeddings can be directly compared with existing v1 embeddings.

How they're trained:

The authors use a **harmonic triplet loss** that mixes v1 and v2 embeddings within the same triplets. As shown in Figure 9 of the paper, triplets can have anchors, positives, and negatives from either v1 or v2 embeddings, encouraging cross-version compatibility.



Results:

As shown in Figure 8, the mixed-mode performance (comparing NN2 embeddings to NN1 embeddings) performs at an intermediate level between pure NN1 and pure NN2, demonstrating successful compatibility.

### (g) How does the performance vary with embedding dimensionality?

From **Table 5** in the paper:

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



Key observations:

1. **128 dimensions performs best**, which is what the authors selected for their final model
2. The differences are **statistically insignificant** given the standard errors
3. Larger embeddings (512) don't necessarily perform better — they may require more training
4. The 128-D embedding can be **quantized to 128 bytes** without loss of accuracy, making it very compact for large-scale deployment
5. Smaller embeddings (64) are viable for mobile devices with only minor accuracy loss

### (h) How does the performance vary with increasing amounts of training data?

From **Table 6** in the paper:

Training Images	VAL @ $10^{-3}$ FAR
2.6M	$76.3\%$
6.1M	$65.4\%$

26M	85.1%
52M	85.1%
260M	86.2%

Key observations:

1. Massive improvement from 2.6M to 26M images: ~9 percentage points (60% relative error reduction)
2. Diminishing returns from 26M to 52M: No improvement
3. Small continued gains at 260M: Another ~1 percentage point improvement
4. The effect may be even larger for bigger models — this experiment was conducted on a smaller model due to time constraints
5. The authors trained their main models on 100M-200M face thumbnails consisting of about 8M different identities

(i) Briefly share your favorite **emergent** property/result of the learned behavior with a triplet loss from the paper.

My favorite emergent property is demonstrated in Figure 7 (Face Clustering): The learned embedding shows remarkable invariance to occlusion, lighting, pose, and age — all within a single cluster.

The figure shows a cluster from a user's personal photo collection where the same person is recognized across:

- Different ages (child to adult)
- Partial occlusions (sunglasses, hands covering face)
- Extreme pose variations (profile views, looking down)
- Dramatic lighting changes
- Different image qualities

doesn't have childhood pictures

This wasn't explicitly trained for — the triplet loss simply enforced that same-identity faces should be closer than different-identity faces. Yet the network learned these robust invariances as an emergent behavior necessary to satisfy the constraint across diverse training examples.

(j) Which approach taken by the authors interested you the most? Why? (~100 words)

The online semi-hard negative mining strategy interested me most because it elegantly solves a fundamental tension in metric learning. Hard negatives provide the most informative gradients but cause training instability; easy negatives are stable but uninformative. The semi-hard approach finds the sweet spot by selecting negatives that are challenging but not pathological.

What's particularly clever is implementing this within mini-batches rather than requiring expensive offline computation. By carefully constructing batches with ~40 faces per identity, they ensure sufficient anchor-positive diversity while efficiently mining semi-hard negatives online. This curriculum-inspired approach — starting with easier examples and progressively increasing difficulty — is now standard practice in contrastive learning methods like SimCLR and MoCo.

} nice tie with  
curriculum  
learning



Retry

Claude is AI and can make mistakes. Please double-check responses.

Reply...



Opus 4.5

