EECS 182 – Homework 10 (Non-Coding Parts)
Solutions & Notes for LLM Interaction Assignment

NOTE: For any question that depends on YOUR specific training runs (e.g., accuracies, MACs, plots, screenshots), this document gives the conceptual answer and marks the parts you must fill in yourself.

---
1. Kernelized Linear Attention (Part II)

1(a)(i) Approximating softmax similarity with a Gaussian kernel

We have the softmax similarity
  sim_softmax(q, k) = exp((q^T k) / sqrt(D))

and the Gaussian kernel
  K_Gauss(q, k) = exp(-||q - k||^2 / (2*sigma^2)), with sigma^2 = sqrt(D).

Expand the squared norm:
  ||q - k||^2 = ||q||^2 + ||k||^2 - 2 q^T k.

Plug into the Gaussian kernel:
  K_Gauss(q,k)
    = exp(-(||q||^2 + ||k||^2 - 2 q^T k) / (2*sigma^2))
    = exp(-||q||^2 / (2*sigma^2))
      * exp(-||k||^2 / (2*sigma^2))
      * exp(q^T k / (sigma^2)).

With sigma^2 = sqrt(D), we get
  exp((q^T k) / sqrt(D))
    = exp((q^T k) / sigma^2)
    = K_Gauss(q,k)
      * exp(||q||^2 / (2*sigma^2))
      * exp(||k||^2 / (2*sigma^2)).

Therefore,
  sim_softmax(q,k)
    = exp(||q||^2 / (2*sigma^2))
      * exp(||k||^2 / (2*sigma^2))
      * K_Gauss(q,k),
with sigma^2 = sqrt(D).

1(a)(ii) Using random Fourier features to approximate softmax attention

Random Fourier features give a map phi_random(x) in R^{2 D_random} such that
  E_w[phi_random(q)^T phi_random(k)] ≈ K_Gauss(q,k).

Using the relationship from (i):
  sim_softmax(q,k)
    ≈ exp(||q||^2 / (2*sigma^2))

* exp(||k||^2 / (2*sigma^2))
* phi_random(q)^T phi_random(k).

Define
phi_tilde(q) = exp(||q||^2 / (2*sigma^2)) * phi_random(q),
phi_tilde(k) = exp(||k||^2 / (2*sigma^2)) * phi_random(k).

Then
sim_softmax(q,k) ≈ phi_tilde(q)^T phi_tilde(k).

Thus the attention can be approximated in the linearized kernel-attention form:
V'_i ≈ [phi_tilde(Q_i) * sum_{j=1}^N phi_tilde(K_j)^T V_j]
/ [phi_tilde(Q_i) * sum_{j=1}^N phi_tilde(K_j)].

Let the random feature dimension be \tilde D.

Computation for all positions i = 1,...,N:

1. Compute features:
- phi_tilde(K_j) for all j: O(N * D * \tilde D)
- phi_tilde(Q_i) for all i: O(N * D * \tilde D)

2. Precompute key-side summaries (once):
- S = sum_j phi_tilde(K_j)^T V_j, shape (\tilde D x M), cost O(N * \tilde D * M)
- z = sum_j phi_tilde(K_j), shape (\tilde D), cost O(N * \tilde D)

3. Per query i:
- Numerator: phi_tilde(Q_i) * S, cost O(\tilde D * M)
- Denominator: phi_tilde(Q_i) * z, cost O(\tilde D).

Total complexity (dominant terms):
O(N * D * \tilde D + N * \tilde D * M),
which is linear in N (sequence length), in contrast to full softmax attention O(N^2 * max(D, M)).

1(b) Causal masking with kernel attention

Causal attention: position i can only attend to positions j <= i.

Original kernel form (no mask):
V'_i = [phi(Q_i) * sum_{j=1}^N phi(K_j)^T V_j]
/ [phi(Q_i) * sum_{j=1}^N phi(K_j)].

Causal version:
V'_i = [sum_{j=1}^i sim(Q_i, K_j) * V_j]
/ [sum_{j=1}^i sim(Q_i, K_j)].

Using sim(Q_i,K_j) = phi(Q_i) phi(K_j)^T:
V'_i = [phi(Q_i) * sum_{j=1}^i phi(K_j)^T V_j]
/ [phi(Q_i) * sum_{j=1}^i phi(K_j)].

Define cumulative quantities:

S_i = sum_{j=1}^i phi(K_j)^T V_j,
Z_i = sum_{j=1}^i phi(K_j)^T.

Then
V'_i = [phi(Q_i) * S_i] / [phi(Q_i) * Z_i].

Recurrence to compute S_i, Z_i:
S_i = S_{i-1} + phi(K_i)^T V_i, with S_0 = 0,
Z_i = Z_{i-1} + phi(K_i)^T, with Z_0 = 0.

Each update is O(\tilde D * M) for S_i and O(\tilde D) for Z_i, so total cost is O(N * \tilde D * M), linear in N.

---

2. Hand-Design Transformers (Conceptual Comparison)

2(a) Content-based selection: learned vs. hand-designed

Similarities:
- Both the hand-designed and the learned model produce queries Q and keys K such that q_i^T k_j is large when token j has the desired content and small otherwise.
- Value projections W_V typically pass through or lightly transform the token embedding to be copied, so attention focuses on positions with the desired content.
- Attention maps show most of the mass on positions with the target content pattern.

Differences:
- The learned weights are usually less “clean”:
  - Q/K matrices may not be sparse in the same basis you hand-engineered, but still yield the correct relative dot products.
  - There may be extra non-zero entries that do not hurt because softmax only cares about relative magnitudes.
- The learned model may also use slightly different scaling (norms, biases), which is irrelevant as long as logits for desired positions dominate under softmax.

Conclusion:
- Gradient descent discovers a representation that effectively implements content-based selection, but in a distributed, non-hand-crafted way.

2(b) Position-based selection

Similarities:
- Both the hand-designed and learned models exploit positional encodings to produce Q and K that are highly aligned for specific (i, j) pairs corresponding to desired positions (e.g., “select the last token”).
- Attention maps often show stripes/bands along the diagonal capturing positional structure.

Differences:
- The learned model may mix content and position, rather than using purely positional keys

.
- Positional attention patterns may be broader or smoother, not perfectly “one-hot”.
- Some attention heads may specialize in positional selection while others mix position an
d content.

Conclusion:
- The learned transformer re-discovers positional mechanisms similar in spirit to hand-des
igned
ones, but with more redundancy and less interpretability.

---
3. Summarization Transformer

3(a) Training and validation loss curves

You must insert your own screenshots here from your runs.

Conceptual commentary:
- Ideally, training and validation loss both decrease over time.
- If validation loss stabilizes while training loss continues to fall, the model is starti
ng to overfit.
- A modest gap between training and validation loss is normal; a very large gap indicates
strong overfitting.

---
4. Example Difficulty and Early Exit

4(a)(i) What is example difficulty? What is the setup?

Example difficulty measures how “hard” a particular input is for the model to classify.
The setup:
- Train a ResNet to convergence on the dataset, so it is a strong classifier.
- At each layer, record intermediate activations and treat them as features.
- For each layer, train a KNN classifier on those features.
- For a given example, find the earliest layer whose KNN can classify it correctly with
high confidence. Earlier layers imply easier examples; later layers imply harder ones.

4(a)(ii) What properties make an example difficult in this dataset?

Examples tend to be difficult if they:
- Are noisy or low-contrast, with cluttered backgrounds.
- Have unusual poses or viewpoints not well represented in training.
- Are close to class boundaries or visually ambiguous.
- Are partially occluded or cropped.

These factors make representations less separable from other classes, pushing the
example to require deeper layers to become linearly separable.

4(a)(iii) Why must the ResNet be trained to convergence?

If the ResNet is undertrained:
- Its internal features are poor, and many examples will appear “hard” simply because the model has not learned good representations.
- Example difficulty would mostly reflect model underfitting, not intrinsic difficulty.

Training to convergence ensures that layer representations are strong enough that depth-based separability reflects genuine example difficulty.

4(a)(iv) In PyTorch, what do hooks do? Why are they needed here?

Hooks are callbacks attached to modules or tensors that run automatically during forward or backward passes. A forward hook lets you inspect or save the output of a layer without modifying the model’s code.

Here, hooks are used to capture intermediate activations at each ResNet layer so we can:
- Extract layer-wise features,
- Train KNN classifiers on those features,
- Analyze how example difficulty changes with depth.

4(a)(v) Why train KNN classifiers at each layer? On what data?

We train a KNN classifier at each layer to measure how separable examples are at that representation depth.

Data:
- For each layer, we use that layer’s activations on the training (or validation) set along with the ground-truth labels.

Motivation:
- If a KNN on shallow-layer features already classifies an example correctly, it is “easy”
.
- If we need deeper-layer features to do the same, the example is “harder.”

4(a)(vi) Why do many examples exit in earlier layers?

In an early-exit scheme, easy examples often become separable and yield high-confidence predictions using shallow-layer features. Thus they can exit early without requiring full-depth processing. Only a fraction of examples that are more ambiguous or complex need to propagate to deeper layers.

4(a)(vii) Why is ResNet accuracy negatively related to prediction layer for early-exit exa mples?

If you group examples by the layer at which they exit:

- Those exiting early are typically easy and thus have high accuracy.
- The hardest examples are forced to go deepest and are more likely to be misclassified.

Therefore, when plotting accuracy of “examples that exit at layer L” versus L, accuracy tends to decrease with layer index.

4(a)(viii) Distinction between hard and easy examples; is it surprising?

Easy examples:
- Correctly classified using shallow-layer features.
- Usually typical, high-quality, and unambiguous images.

Hard examples:
- Require deep-layer features for correct classification.
- Often noisy, occluded, or atypical.

This distinction matches our intuition and is not very surprising, but the layer-wise KNN analysis makes it quantitatively precise.

4(a)(ix) Patterns observed and relation to layer composition

Typical pattern:
- Early layers (edges/textures) separate examples distinguished by low-level cues.
- Middle layers (parts/shapes) handle more structural differences.
- Deep layers (semantic features) disambiguate fine-grained or ambiguous cases.

This aligns with the standard view of CNNs: depth corresponds to progressively higher-level, more abstract feature hierarchies.

4(b) Early Exit ResNet-18 – quantitative parts

You must fill in the quantitative results from your runs (accuracies, MACs, speed).

Conceptual answers:

(i) Regular ResNet:
- Report final accuracy, inference speed, and MACs for the baseline ResNet (no early exit).

(ii) Early-exit ResNet-18:
- Report accuracy, speed, and average MACs with early exits enabled.

(iii) Comparison:
- Early exit usually significantly reduces average MACs while maintaining similar accuracy, with a small accuracy drop depending on thresholds.

(iv) Lowest MACs:

- As thresholds are tuned to encourage earlier exits, MACs drop but accuracy eventually fa
lls off.
- The best setting represents a trade-off where much of ResNet's computation is redundant
  for easy examples.

(v) When is early exit preferable to a smaller model?

Early exit is preferable when:
- Example difficulty varies widely.
- You want adaptive compute: easy inputs should be cheap, hard ones can be expensive.
- Latency or power budgets vary per request.

A smaller model is preferable when:
- You want uniform, simple deployment and fixed runtime per input.
- Overall accuracy can be sacrificed in exchange for simplicity and smaller capacity.

---
5. FaceNet: A Unified Embedding for Face Recognition and Clustering
---

5(a) Two architectures considered

FaceNet explores:
1) A Zeiler & Fergus–style deep CNN (often called NN1).
2) GoogLeNet/Inception-style CNNs (NN2, NN3, NN4, etc.), which use Inception modules and
   are more parameter-efficient.

5(b) Triplet loss vs. standard supervised objective

Triplet loss operates on triplets (anchor x^a, positive x^p, negative x^n):
- It enforces that the distance between anchor and positive is smaller (by a margin) than
  the distance between anchor and negative:
    ||f(x^a) - f(x^p)||^2 + alpha < ||f(x^a) - f(x^n)||^2.

This directly shapes the geometry of the embedding space so that identities form tight clu
sters.

In contrast, standard supervised learning with cross-entropy trains the network to predict
 a
class label, not to maintain a meaningful distance metric between embeddings.

5(c) Challenge with generating all possible triplets and the solution

Problem:
- The number of possible triplets grows cubically with the dataset size and most are uninf
ormative
  (constraints already satisfied). Evaluating all triplets is computationally infeasible a
nd inefficient.

Solution:

- Use online triplet mining within mini-batches:
  - Construct large batches.
  - For each anchor, select hard or semi-hard positives and negatives inside the batch.
- This focuses learning on triplets that violate or nearly violate the margin, making trai
ning efficient.

5(d) Parameters/FLOPs and comparison to ResNet-50

From the paper:
- A large Zeiler & Fergus–style FaceNet model (NN1) has about 140M parameters and ~1.6B FL
OPs.
- Inception-based FaceNet models have far fewer parameters (e.g., ~7.5M) at similar FLOPs.

ResNet-50:
- ~25M parameters and ~3.8B FLOPs.

Comparison (NN1 vs ResNet-50):
- NN1 has many more parameters (~5–6x) but fewer FLOPs (~0.4x).
- Inception FaceNet models demonstrate that high performance can be achieved with fewer pa
rameters
  and competitive FLOPs compared to ResNet-50.

5(e) Semi-hard negatives

Semi-hard negatives satisfy:
  ||f(x^a) - f(x^p)||^2 < ||f(x^a) - f(x^n)||^2 < ||f(x^a) - f(x^p)||^2 + alpha.

They are harder than easy negatives (which are already far away) but not the absolute hard
est, which
may be noisy or cause instability. Semi-hard negatives provide informative gradients while
 avoiding
training collapse.

5(f) Harmonic embeddings

Harmonic embeddings refer to embeddings produced by different networks (e.g., old and new)
 that are
trained to be compatible in the same metric space. Using a harmonic triplet loss, FaceNet
learns
new embeddings that maintain distances consistent with older embeddings. This allows mixin
g embeddings
from multiple generations in a single system (e.g., a large face database) without recompu
ting everything.

5(g) Performance vs. embedding dimensionality

Empirically (from the paper's results):
- Performance improves when going from very small embedding sizes (e.g., 64) to moderate o

nes (e.g., 128).
- Accuracy is roughly flat between 128 and 256.
- Going to 512 dimensions offers little or no gain and may slightly hurt performance in th
eir setup.

Conclusion:
- 128-dimensional embeddings give a good trade-off between compactness and accuracy.

5(h) Performance vs. training data size

Empirically:
- Going from a few million to tens of millions of training images substantially improves v
alidation
performance.
- Increasing data beyond that yields diminishing returns: performance continues to improve
, but more slowly.

This matches the general pattern that more data helps, but with decreasing marginal benefi
t.

5(i) Favorite emergent property of triplet loss

One notable emergent property is that simple Euclidean distance in the learned embedding s
pace clusters
faces by identity extremely well, despite wide variations in pose, lighting, expression, a
nd age. The model
learns invariances to many nuisance factors without explicit supervision for those factors
, purely by
optimizing triplet loss.

5(j) Most interesting approach (example ~100 words)

A sample answer you can personalize:

The most interesting aspect to me is the online triplet mining with semi-hard negatives. I
nstead of relying
on all possible triplets, which is computationally infeasible, FaceNet focuses only on tho
se triplets that
are most informative under the current model. This makes triplet loss training practical a
t web scale while
also improving convergence. I find the idea of treating example selection as a core part o
f the learning
algorithm very appealing: the model is not just learning from a fixed set of instances but
dynamically
choosing which comparisons to emphasize to better shape the embedding space.

---
6. Homework Process & Study Group

You must fill in your own details here (resources used, collaborators, time spent, etc.).

End of Solutions (Non-Coding Parts)
