EECS 182 Deep Neural Networks
Fall 2025 Anant Sahai and Gireeja Ranade

Homework 10

This homework is due on November 14, at 10:59PM.

1. Kernelized Linear Attention (Part II)

This is the second part of of the "Kernelized Linear Attention" problem you saw previously. Please refer to this part for notation and context.

In Part I of this problem, we considered ways to efficiently express the attention operation when sequences are long (e.g. a long document). Attention uses the following equation:

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

We saw that when the similarity function is a kernel function (i.e. if we can write $\text{sim}\left(Q_{i\cdot}, K_{j\cdot}\right) = \Phi(Q_{i\cdot})\Phi(K_{j\cdot})^T$ for some function Φ), then we can use the associative property of matrix multiplication to simplify the formula to

$$V'_{i\cdot} = \frac{\phi\left(Q_{i\cdot}\right) \sum_{j=1}^{N} \phi\left(K_{j\cdot}\right)^T V_{j\cdot}}{\phi\left(Q_{i\cdot}\right) \sum_{j=1}^{N} \phi\left(K_{j\cdot}\right)^T}. \tag{2}$$

If we use a polynomial kernel with degree 2, this gives a computational cost of $\mathcal{O}\left(ND^2M\right)$, which for very large N is favorable to the softmax attention computational cost of $\mathcal{O}\left(N^2 \max\left(D, M\right)\right)$. ($N$ is the sequence length, D is the feature dimension of the queries and keys, and M is the feature dimension of the values. Now, we will see whether we can use kernel attention to directly approximate the softmax attention:

$$V'_{i\cdot} = \frac{\sum_{j=1}^{N} \exp(\frac{Q_{i\cdot}K_{j\cdot}^T}{\sqrt{D}}) V_{j\cdot}}{\sum_{j=1}^{N} \exp(\frac{Q_{i\cdot}K_{j\cdot}^T}{\sqrt{D}})}. \tag{3}$$

(a) Approximating softmax attention with linearized kernel attention

i. As a first step, we can use Gaussian Kernel $\mathcal{K}_{\text{Gauss}}(q,k) = \exp(\frac{-||q-k||_2^2}{2\sigma^2})$ to rewrite the softmax similarity function, where $\text{sim}_{\text{softmax}}(q,k) = \exp(\frac{q^Tk}{\sqrt{D}})$. Assuming we can have $\sigma^2 = \sqrt{D}$, **rewrite the softmax similarity function using Gaussian Kernel.**. (*Hint: You can write the softmax* $\exp(\frac{-||q-k||_2^2}{2\sigma^2})$ *as the product of the Gaussian Kernel and two other terms.)*

ii. However, writing softmax attention using a Gaussian kernel does not directly enjoy the benefits of the reduced complexity using the feature map. This is because the feature map of Guassian kernel usually comes from the Taylor expression of $\exp(\cdot)$, whose computation is still expensive

[1]. However, we can approximate the Guassian kernel using random feature map and then reduce the computation cost, using a trick from Random feature attention (2021)[2]. (Rahimi and Recht, 2007)[3] proposed random Fourier features to approximate a desired shift-invariant kernel. The method nonlinearly transforms a pair of vectors q and k using a **random feature map** $\phi_{\text{random}}()$; the inner product between $\phi(q)$ and $\phi(k)$ approximates the kernel evaluation on q and k. More precisely:

$$\phi_{\text{random}}(q) = \sqrt{\frac{1}{D_{\text{random}}}}\left[\sin\left(\mathbf{w}_1 q\right), \ldots, \sin\left(\mathbf{w}_{D_{\text{random}}} q\right), \cos\left(\mathbf{w}_1 q\right), \ldots, \cos\left(\mathbf{w}_{D_{\text{random}}} q\right)\right]^\top. \tag{4}$$

Where we have D_{random} of D-dimensional random vectors $\mathbf{w}_{i\cdot}$ independently sampled from $\mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_D)$,

$$\mathbb{E}_{\mathbf{w}_{i\cdot}}\left[\phi(q) \cdot \phi(k)\right] = \exp\left(-\|q-k\|^2 / 2\sigma^2\right). \tag{5}$$

Use ϕ_{random} to approximate the above softmax similarity function with Gaussian Kernel and derive the computation cost for computing all the V' here.

(b) (Optional) Beyond self-attention, an autoregressive case will be masking the attention computation such that the i-th position can only be influenced by a position j if and only if $j \leq i$, namely a position cannot be influenced by the subsequent positions. This type of attention masking is called *causal masking*.

Derive how this causal masking changes equation 1 and 2. **Write equation 1 and 2 in terms of $S_{i\cdot}$ and $Z_{i\cdot}$**, which are defined as:

$$S_{i\cdot} = \sum_{j=1}^{i} \phi\left(K_{j\cdot}\right)^T V_{j\cdot}, \quad Z_{i\cdot} = \sum_{j=1}^{i} \phi\left(K_{j\cdot}\right)^T, \tag{6}$$

to simplify the causal masking kernel attention and derive the computational complexity of this new causal masking formulation scheme.

2. Coding Question: Hand-Design Transformers

Please follow the instructions in this notebook. You will implement a simple transformer model (with a single attention head) from scratch and then create hand-designed attention heads of the transformer model capable of solving a basic problem.

Once you finished with the notebook, answer the following questions in your submission of the written assignment:

(a) Design a transformer that selects by contents. Compare the variables of your hand-designed Transformer with those of the learned Transformer. **Identify the similarities and differences between the two sets of variables and provide a brief explanation for each difference.**

[1] https://www.csie.ntu.edu.tw/~cjlin/talks/kuleuven_svm.pdf

[2] Peng, Hao, et al. "Random feature attention." arXiv preprint arXiv:2103.02143 (2021).

[3] Rahimi, Ali, and Benjamin Recht. "Random features for large-scale kernel machines." Advances in neural information processing systems 20 (2007).

(b) Design a transformer that selects by positions. Compare the variables of your hand-designed Transformer with those of the learned Transformer. **Identify the similarities and differences between the two sets of variables and provide a brief explanation for each difference.**

(c) (Optional) Designing a transformer that selects by position. Fill out Section (b) in the notebook and **comment on the similarities and differences between the weights and intermediate outputs of the learned and hand-coded model.**

3. Coding Question: Summarization

Please follow the instructions in this notebook. You will implement a Transformer using fundamental building blocks in PyTorch. You'll apply the Transformer encoder-decoder model to a sequence-to-sequence NLP task: document summarization. Refer to the Attention is All You Need paper for details on the model architecture.

Once you finished with the notebook, answer the following questions in your submission of the written assignment:

(a) **Please submit the screenshots of the training loss and the validation loss displayed on Tensorboard.**

4. Example Difficulty and Early Exit

Deep Learning Practioners have recognized that within the same task, particular examples in the test set can actually be harder to perform predictions on that others. Why is that? What kinds of things are easier to learn? We explore the notion of example difficulty, proposed by Baldock et. al. that will allow us to perform deeper investigations on the topic. Furthermore, utilizing concepts from example difficulty can aid in development of techniques than can boost inference speeds, sample efficiency, and other key properties that allow scalability in both the micro and macro levels.

(a) Please run the notebook cells of `example_difficulty.ipynb`. Don't forget to download the data at `data.npy` and `test_data.npy` and place these files in the same directory as the notebook. Note that the notebook could take a while to run, so start the notebook and work on something else while it runs. Then answer the following conceptual questions.

i. **Briefly explain what example difficulty is in your own words. What is the setup for us to analyze example difficulty?**
ii. **What kinds of properties do you think will make an example from this dataset difficult?**
iii. **Why do we need to train the ResNet to convergence to be able to analyze the example difficulty?**
iv. **In pytorch, what does adding hooks do? Why do we need to use them?**
v. **Why do we train KNN classifiers at each layer of the ResNet? What data are we training them on?**
vi. **Why do you think so many examples exit in the earlier layers?**
vii. **Explain why the accuracy of the ResNet has a negative relationship with the prediction layer.**
viii. **Explain the distinction between the hard and easy examples. Does this surprise you?**
ix. **What kinds of patterns do you notice? Based on the composition of the layers, does it make sense?**

(b) Please fill in the notebook cells in `early_exit.ipynb`. Don't forget to download the data at `data.npy` and `test_data.npy` and place these files in the same directory as the notebook. Also include `architectures.py` in the same directory. Then answer the following conceptual questions.

i. **What was the accuracy with a regular ResNet? Inference Speed? Total MACS?**
ii. **What was the accuracy with an early exit ResNet-18? Inference Speed? Total MACS?**
iii. **How did early exit do? Compare accuracy and MACS.**
iv. **What was the lowest MACs you found. What might this say?**
v. **When would we use early exit, versus just using a smaller model? What factors should we consider?**

5. Read a Research Paper: FaceNet

In class, you have learnt how self-supervised learning can be used to learn useful representations from large datasets without labels (e.g., learning features from ImageNet). While these features may inherently pick out some notion of "similarity" between different images in the dataset, they are not incentivized to *cluster* different data points based on any interesting similarity measure.

The paper "FaceNet: A Unified Embedding for Face Recognition and Clustering" explores how we can view task of face recognition through the lens of self-supervised (or to be more accurate, *slightly* supervised) learning.

Personally I found this blog post to help a lot more than reading the paper. The two figures are particularly helpful.

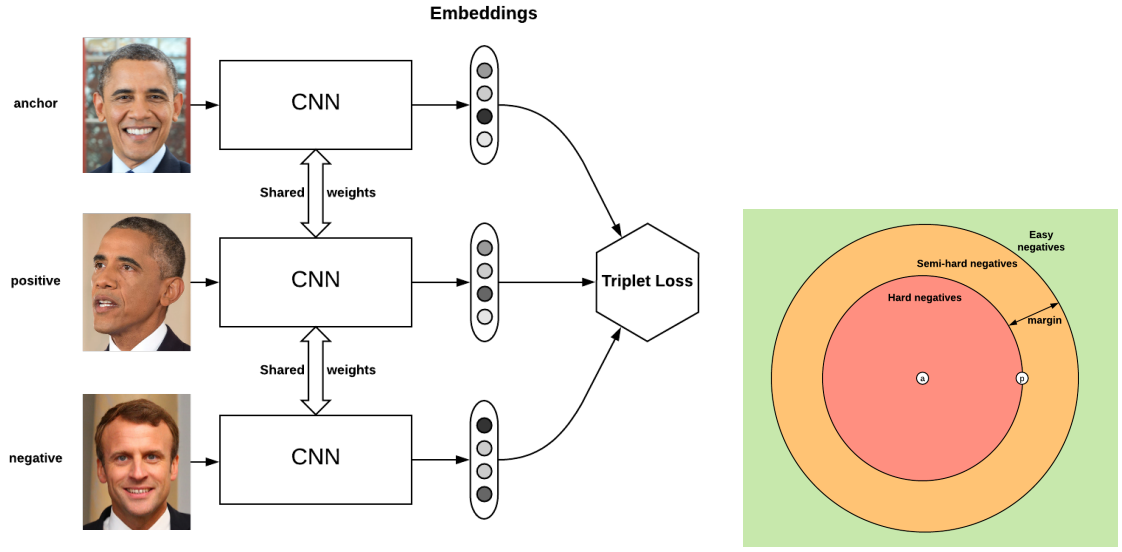

Figure 1: Figures from blog post.

Read the paper and answer the questions below.

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

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

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

(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?** Read Table 1 of the original paper[4] on ResNet to find out the FLOPs of ResNet-50. Its parameter count can be found by searching online.

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

(f) **What are harmonic embeddings?**

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

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

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

(j) **Which approach taken by the authors interested you the most? Why?** ($\approx$ 100 words)

6. Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student!
We also want to understand what resources you find helpful and how much time homework is taking, so we can change things in the future if possible.

(a) **What sources (if any) did you use as you worked through the homework?**

(b) **If you worked with someone on this homework, who did you work with?**
List names and student ID's. (In case of homework party, you can also just describe the group.)

(c) **Roughly how many total hours did you work on this homework?**

Contributors:

- Sheng Shen.
- Shaojie Bai.
- Angelos Katharopoulos.
- Hao Peng.
- Olivia Watkins.
- Jake Austin.
- Dhruv Shah.
- Anant Sahai.
- Linyuan Gong.
- Hao Liu.
- Allie Gu.
- CS182 Staff from past semesters.
- Liam Tan.

[4] He, Kaiming, et al. "Deep residual learning for image recognition." (2016).

