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:

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

(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 $\text{sim}_{\text{softmax}}(q, k) = \exp(\frac{q^T k}{\sqrt{D}})$. 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.*)

Solution:

$$\text{sim}_{\text{softmax}}(q, k) = \exp(\frac{q^T k}{\sqrt{D}}) = \exp(\frac{||q||_2^2}{2\sigma^2}) * \exp(\frac{-||q-k||_2^2}{2\sigma^2}) * \exp(\frac{||k||_2^2}{2\sigma^2})$$

$$= \exp(\frac{||q||_2^2}{2\sigma^2}) * \mathcal{K}_{\text{Gauss}}(q,k) * \exp(\frac{||k||_2^2}{2\sigma^2})$$

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:

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

Where we have D_{random} of D-dimensional random vectors $\mathbf{w}_i$. 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.

Solution:

$$\begin{aligned}\text{sim}_{\text{softmax}}(Q,K) &= \exp\left(\frac{||Q||_2^2}{2\sigma^2}\right) * \mathcal{K}_{\text{Gauss}}(Q,K) * \exp\left(\frac{||K||_2^2}{2\sigma^2}\right)\\ &\approx \exp\left(\frac{||Q||_2^2}{2\sigma^2}\right) * \phi_{\text{random}}(Q)^T \phi_{\text{random}}(K) * \exp\left(\frac{||K||_2^2}{2\sigma^2}\right)\end{aligned}$$

The computation cost of ϕ_{random} for a single vector is $O(D_{random}D)$ where D is for computing each coordinate $\sin(w_i.q)$ or $\cos(w_i.q)$, and there are $2D_{random}$ of coordinates.
Cost of computing the entire matrix $\phi_{random}(Q)$ is $O(D_{random}DN)$.
Cost of computing the entire matrix $\phi_{random}(K)$ is $O(D_{random}DN)$.
Cost of computing $\exp\left(\frac{||k||_2^2}{2\sigma^2}\right)$ is $O(D)$, due to the cost of taking the L2 norm. The cost of the whole $\exp\left(\frac{||K||_2^2}{2\sigma^2}\right)$ is $O(ND)$. It is a vector with N entries. As always, we only consider the **incremental cost** of computing – given that everything in previous paragraphs had been computed, how much more does it cost to compute everything in this paragraph?
Cost of $\exp\left(\frac{||Q||_2^2}{2\sigma^2}\right)$ is $O(ND)$. It is a vector with N entries.
Now, define the matrix $A := \exp\left(\frac{||Q||_2^2}{2\sigma^2}\right)\phi_{\text{random}}(Q)^T$ and the matrix $B := \phi_{\text{random}}(K)\exp\left(\frac{||K||_2^2}{2\sigma^2}\right)$. They have shapes $N \times 2D_{random}$ and $2D_{random} \times N$ respectively. Cost of computing each is $O(D_{random}N)$.

[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).

Let B' be the $2D_{random} \times 1$ matrix obtained by summing the columns of B. Cost of computing is $O(D_{random}N)$.
Now it remains to do three matrix multiplications and one division to get V'. The ordering of the matrix multiplications is crucial.
Multiply BV. This costs $O(D_{random}NM)$.
Multiply $A(BV)$. This costs $O(ND_{random}M)$.
Multiply AB'. This costs $O(ND_{random})$.
Divide each row of V by each entry of AB'. This costs $O(NM)$
The total cost is found by summing all of them:

$$\mathcal{O}\left(N(M+D)D_{\text{random}}\right)$$

(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:

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

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

Solution: The causal masking changes equation 1 as follows,

$$V'_{i\cdot} = \frac{\sum_{j=1}^{i} \text{sim}\left(Q_{i\cdot}, K_{j\cdot}\right) V_{j\cdot}}{\sum_{j=1}^{i} \text{sim}\left(Q_{i\cdot}, K_{j\cdot}\right)}. \tag{7}$$

We linearize the masked attention as described below,

$$V'_{i\cdot} = \frac{\phi\left(Q_{i\cdot}\right) \sum_{j=1}^{i} \phi\left(K_{j\cdot}\right)^T V_{j\cdot}}{\phi\left(Q_{i\cdot}\right) \sum_{j=1}^{i} \phi\left(K_{j\cdot}\right)^T}. \tag{8}$$

we can simplify equation 8 to

$$V'_{i\cdot} = \frac{\phi\left(Q_{i\cdot}\right) S_{i\cdot}}{\phi\left(Q_{i\cdot}\right) Z_{i\cdot}}. \tag{9}$$

Note that, $S_{i\cdot}$ and $Z_{i\cdot}$ can be computed from S_{i-1} and Z_{i-1} in constant time hence making the computational complexity of linear transformers with causal masking linear with respect to the sequence length.

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.**

Solution: Answers may vary, but weights, keys, and queries are likely more evenly distributed than those the student implemented. However, Vm, attention scores and values should be roughly the same.

Explanation: this occurs since the network is simple enough that there is only one correct attention pattern (only attend to matching positions) and only one correct thing to do with attended positions (use them to copy the original content into the output). In more complicated problems it's rare to be able to predict precisely what features the transformer learns.

(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.**

Solution: Answers may vary, but weights, keys, and queries are likely more evenly distributed than those the student implemented. However, attention scores and outputs should be roughly the same, since there is only one correct attention pattern (only attend to the first position) and only one correct thing to do with attended positions (use them to copy the first element into the output).

(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.**

Solution: Answers may vary, but all transformer outputs may look substantially different between the two models except the sign of the final output should be the same. This shows us that transformers can solve tasks in ways which are unintuitive to humans.

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.**

Solution: They should look like:

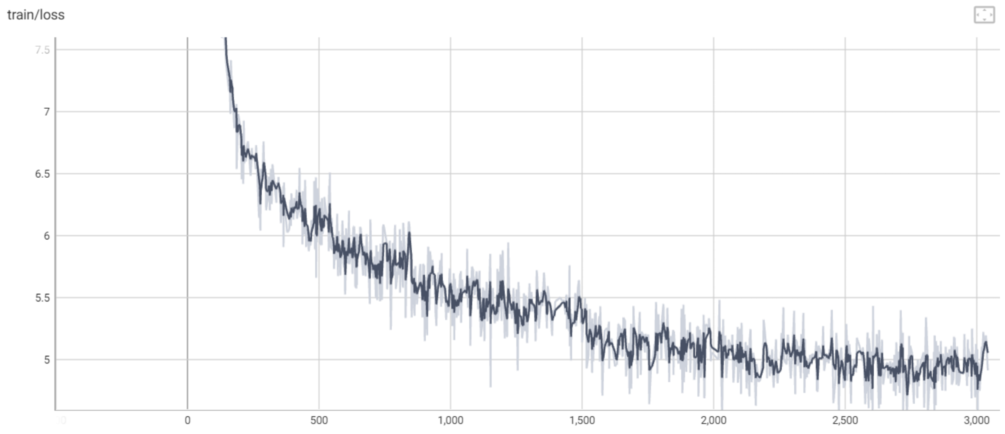

Figure 1: Training loss.

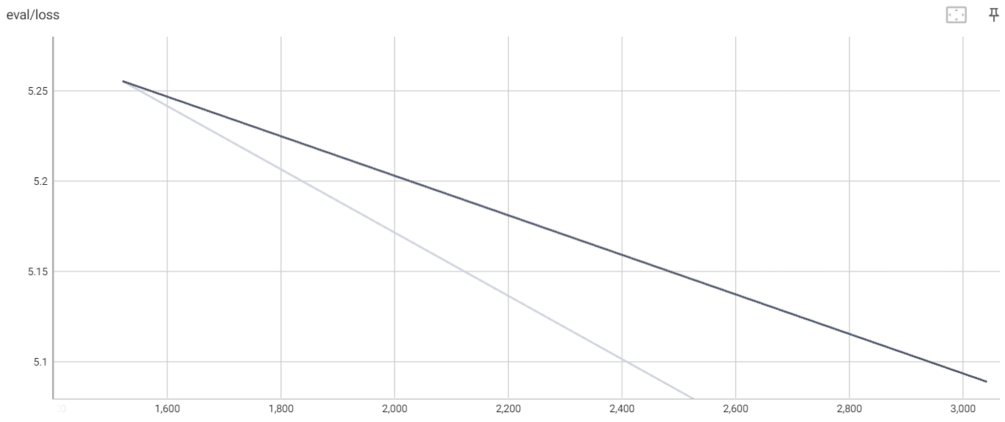

Figure 2: Validation loss.

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?**

Solution: Example difficulty is the relative measure of the computational difficulty of making a prediction. Some examples may be easily classified with few layers in a neural network, while others may have to be processed through multiple layers. The setup for us to examine example difficulty is to train a ResNet 18, attach KNN classifiers to each layer, then analyze what is known as prediction depth of each layer. Through visualizing various properties of the prediction depths, we can gain a better understanding of traits that might make an example more difficult or easier.

ii. **What kinds of properties do you think will make an example from this dataset difficult?**

Solution: The amount of noise added to the image, the shape of the image, and the size of the shape all seem to contribute to the example difficulty.

iii. **Why do we need to train the ResNet to convergence to be able to analyze the example difficulty?**

Solution: Not training to convergence could open up confounding variables (e.g, earlier layers could learn faster). This would add in complexity to our analysis, since we are primarily discussing prediction depth.

iv. **In pytorch, what does adding hooks do? Why do we need to use them?**

Solution: Hooks allow us to capture intermediate values in pytorch models. This gains us access to activations without having to write a model from scratch.

v. **Why do we train KNN classifiers at each layer of the ResNet? What data are we training them on?**

Solution: The purpose of training KNN classifiers is to create a measure of prediction depth. We want this prediction depth in order to determine how hard an example was. We are training the KNN classifiers on the intermediate representations in the ResNet.

vi. **Why do you think so many examples exit in the earlier layers?**

Solution: There are quite a lot of easy examples in this dataset for a ResNet to classify.

vii. **Explain why the accuracy of the ResNet has a negative relationship with the prediction layer.**

Solution: We can hypothesize that the examples that exited earlier were the easiest, and therefore would've been the easiest to classify with a ResNet in general.

viii. **Explain the distinction between the hard and easy examples. Does this surprise you?**

Solution: The hard examples are much smaller than the easy examples. This is quite interesting that the network can distinguish these

ix. **What kinds of patterns do you notice? Based on the composition of the layers, does it make sense?**

Solution: A lot of the examples exit in layer 4, which is after a max pool. I find this to be interesting. A lot of examples also exit after layer 8, which is the final ResNet block. This makes 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?**

Solution: Accuracy should be around 90 percent. Inference speed will vary. Total MACS: 3336213504000.

ii. **What was the accuracy with an early exit ResNet-18? Inference Speed? Total MACS?**

Solution: Similar 90 percent accuracy. Inference speed will also vary. MACS will be around 2305700339712. This can vary depending on the entropy tolerance.

iii. **How did early exit do? Compare accuracy and MACS.**

Solution: Accuracy should go down by around 5 percent. MACS should go down by around 50 percent or more, depending on the tolerance.

iv. **What was the lowest MACs you found. What might this say?**

Solution: Mac ratio should at least be 3x. This could indicate that we don't need this high capacity for our task.

v. **When would we use early exit, versus just using a smaller model? What factors should we consider?**

Solution: open question.

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.

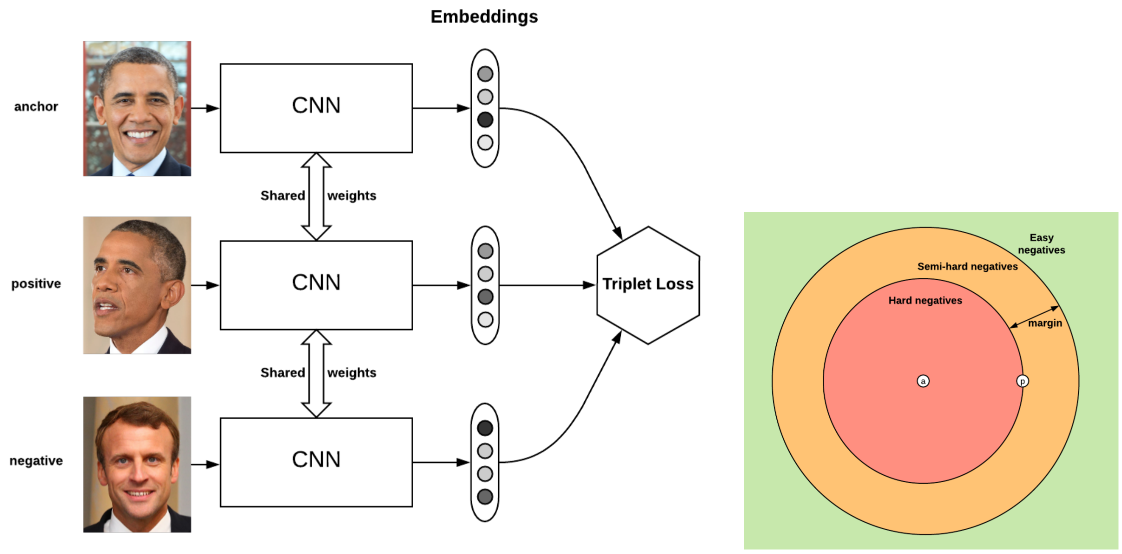

Figure 3: Figures from blog post.

Read the paper and answer the questions below.

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

Solution: The first architecture is based on the Zeiler & Fergus model which consists of multiple interleaved layers of convolutions, non-linear activations, local response normalizations, and max pooling layers. The second architecture is based on the Inception model of Szegedy et al.

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

Solution: Open-ended question, there's no "right" answer. The most important distinction is the fact that triplet loss doesn't require labels and can learn by simply specifying "positives" and "negatives" with respect to the "anchor".

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

Solution: Generating all possible triplets would simultaneously be computationally expensive (and likely infeasible for large datasets) and wasteful (since many triplets would trivially satisfy the condition set by equation 1). The authors use a variety of methods outlined in Section 3.3, including "hard" triplet mining for selecting the most informative triplets.

(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.

Solution: Table 1 suggests that the authors use a 140M (NN1) / 7.5M (NN2) parameter model for with 1.6B FLOPS. ResNet-50 uses approximately 26M parameters and 3.8B FLOPS.

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

Solution: Since the "hardest" negatives may lead to bad local minima (in practice), the authors use *semi-hard* negatives to balance the trade-off between having informative negative examples and avoiding trivial examples that violate the triplet constraint. Semi-hard negatives refer to negative examples in a triplet that are closer to the anchor image than the positive example, but are still far enough away from the anchor image to provide a meaningful training signal.

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

Solution: Harmonic embeddings refer to a way of embedding data points across different neural network models such that they satisfy a special property: embeddings generated by different models v1 and v2 are compatible in the sense that they can be compared to each other. This compatibility greatly simplifies the path to using multiple models and comparing their results.

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

Solution: The authors find that the performs improves slightly by increasing embedding dimensionality from 64 to 128, then drops on increasing it further. While it is natural to expect the performance to improve with higher dimensionality, the authors speculate that the performance drop may be due to larger embedding spaces requiring more training time to achieve similar (or better) performance. It is also likely that the difference in performance in Table 5 is statistically insignificant.

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

Solution: Expectedly, the validation performance increases with increasing amounts of training data, although Table 6 suggests plateauing of the performance. $10\times$ increase from 2.6M to 26M training images improves the performance by $\approx 9\%$, while a further $10\times$ increase only improves performance by 1%.

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

Solution: Open ended question, there's no "right" answer.

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

Solution: Open ended question, there's no "right" answer.

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.

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

(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.