CS4787 Final Exam

Fall 2022

NAME:	
Net ID:	
Email:	

Please make sure to write your NetID on each page of the exam. This will facilitate the reincorporation of the pages of your exam in the event of the failure of its fastening apparatus.

No use of notes, computers, or mobile devices is allowed on this exam. Use of a calculator is allowed, but you are not expected to need one—all computations are designed to be doable by hand. This exam is subject to Cornell's Code of Academic Integrity: please sign this document below to indicate that you understand and commit to abide by the Code.

I promise to abide by Cornell's Code of Academic Integrity.

Signature: ________________________________

1 [??] True/False Questions

Please identify if these statements are either True or False. Please justify your answer **if false**. Correct "True" questions yield 1 point. Correct "False" questions yield 2 points, one for the answer and one for the justification. Note that a justification that merely states the logical negation of the statement will not be considered as a valid justification.

1. **(T/F)** The parameter κ in the LCB acquisition function refers to the condition number of that function.

2. **(T/F)** Adam uses a different learning rate for different parameters based on an exponential moving average of the squares of the stochastic gradients it has observed so far during training.

3. **(T/F)** If the parameter vectors w_t reached after t iterations of SGD diverge to infinity (i.e. $\lim_{t\to\infty} \|w_t\| = \infty$) then so must the total loss $f(w_t)$.

4. **(T/F)** Early stopping can save resources by ending execution early. As a trade-off, models trained using early stopping have worse validation error than if they had run for more epochs.

5. **(T/F)** Momentum changes the theoretical convergence rate of gradient descent on strongly convex losses roughly by "replacing" the condition number κ in its convergence rate by $\sqrt{\kappa}$.

6. **(T/F)** The TPU stands for "transformer processing unit" and was designed to accelerate the self-attention layers of transformer networks.

7. **(T/F)** The number of flops required to compute the empirical risk is independent of the condition number κ.

8. **(T/F)** A matrix with less than 75% nonzeros can be compressed to take up less memory by storing it in sparse "COO" format.

9. **(T/F)** Neural network pruning is a type of compression method that works by removing weights or activations that are usually zero or close to zero.

10. **(T/F)** We can get a sense of how well an Autoencoder will work to reduce the dimensionality of a dataset to a particular dimension d by looking at the eigenvalues of the covariance matrix of the dataset.

11. **(T/F)** Deep neural network inference is only ever run in the cloud.

12. **(T/F)** Backpropagation allows us to compute a gradient ∇f of a function f in time proportional to the square of the time it would take to compute f.

2 [19] Short Answers

1. (4 pts) Choose and name two metrics that are important for inference. For each one, say what is is and how it is measured. Then give an example of a technique we've discussed in class that produces a trade-off between those two metrics.

2. (3 pts) Suppose that F is a function that maps from $\mathbb{R}^{m \times n}$ to $\mathbb{R}^{n \times p}$, i.e. if $x \in \mathbb{R}^{m \times n}$, then $F(x) \in \mathbb{R}^{n \times p}$. The derivative of F is a function DF that maps $\mathbb{R}^{m \times n}$ to some vector space V, i.e. $DF(x) \in V$. What is the dimension of V as a vector space (i.e. how large is a basis for that vector space)? Justify your answer.

3. (3 pts) Consider the loss function $f : \mathbb{R}^2 \to \mathbb{R}$ defined by

$$f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = 4(x_1 - 3)^2 + (x_2 + 1)^2$$

Does this loss function have a condition number? If so, say what it is, and show how you derived it. If not, explain why f has no condition number.

4. (3 pts) What is batch normalization? Briefly explain how it works, and how its behavior differs between training time and evaluation/inference time.

5. (4 pts) Your colleague Hans (from the prelim) has now trained an ensemble of four feedforward ReLU deep neural networks (i.e. all the layers are fully connected linear layers with ReLU nonlinearity) all with the same architecture, producing an ensemble with parameters that take up a total of 256 GB of space (across all four networks) when stored as dense arrays of single-precision floats. Hans knows that the vast majority (over 95%) of this space is taken up by weights for layers other than the first and last layer.

   Hans's partner Franz uses knowledge distillation to reduce the size of Hans's network, training a single new, smaller network that has the same architecture as one of Hans's original networks *except* (i) each of the hidden layers' widths d_i is one-quarter the width of the corresponding layer in the original network, and (ii) all the weights in Franz's new network are stored as dense arrays using bfloat16.

   Approximately how much memory will Franz's compressed model take up? Justify your answer.

6. (2 pts) Your friend Ferdinand is training a model using online learning, and is evaluating its performance by measuring its regret $R(\hat{w}, T)$ compared to some fixed model $\hat{w}$. Ferdinand observes that the regret $R(\hat{w}, T)$ does not converge to 0 as T increases, but instead seems to be diverging to infinity. On the basis of this, Ferdinand concludes that there is something wrong with his setup. Is Ferdinand's conclusion reasonable? If so, what might be wrong with his setup? If not, explain why Ferdinand is wrong.

Net ID: ______________________

3 [18] Quantization and Low-Precision Arithmetic

1. (3 pts) Micron Technology plans to spend \$100 billion building a mega-complex of computer chip plants in Syracuse's northern suburbs, bringing chip manufacturing to upstate New York. To commemorate the occasion, they want to propose a new 16-bit floating point format for use with machine learning. Recall that the IEEE standard 16-bit floating point format has a 5-bit exponent and a 10-bit mantissa as illustrated below.

IEEE half-precision	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	sign	5-bit exp					10-bit mantissa									

   Recall that the formula for the number represented by a half-precision float with sign $s \in \{-1, 1\}$, exponent field $e \in \{0, 1, \ldots, 2^5 - 2\}$ and mantissa bits $m_9, m_8, \ldots, m_0$ is

$$s \cdot 2^{e-15} \cdot (1.m_9 m_8 \ldots m_0)_b.$$

   where the *exponent bias* 15 is set to be $2^{5-1} - 1 = 16 - 1 = 15$ (following the formula that the exponent bias for a B-bit exponent is $2^{B-1} - 1$) and where the last term is interpreted as a number written in binary. The largest representable number in this format is $2^{15} \cdot (1.1111111111)_b = 2^{15} \cdot (2 - 2^{-10}) = 65504$.

   For marketing reasons, Micron wants to propose a floating-point format with enough range to represent numbers larger than 100 billion. What is the minimum number of exponent bits B needed to do this with a 16-bit floating-point format that otherwise follows the standard rules? Be sure to show your work.

2. (2 pts) Will Micron's new 16-bit floating-point format have a machine epsilon that is larger, smaller, or the same size as the IEEE standard 16-bit floats? What does this imply about the rounding error of the various formats?

3. (3 pts) Micron plans to build its plants on a 1280-acre site. To help remember this fact without using too much memory, they want to store the number 1280 in IEEE-half precision. What is the bit-pattern of this number? Be sure to show your work.

4. (2 pts) Can Micron's new 16-bit floating-point format actually represent the number 100 billion exactly? Justify your answer.

5. (6 pts) In class, we discussed four different "special cases" of floating point numbers. Choose **three (3)** of the following four categories. For each of the three categories you chose, describe in words what they represent and give an example of how this case may arise in the course of training a machine learning model.

   (a) The exponent field is all zeros, and the mantissa field is also all zeros.

   (b) The exponent field is all zeros, and the mantissa field is not all zeros.

   (c) The exponent field is all ones (e.g. 255 for a 32-bit float), and the mantissa field is all zeros.

   (d) The exponent field is all ones (e.g. 255 for a 32-bit float), and the mantissa field is not all zeros.

6. (2 pts) What are the relative advantages and disadvantages of bfloat16 and IEEE half-precision floats in a deep learning application? (List one way in which each of these formats is superior.)

4 [25] Kernels and Feature Extraction

1. (8 pts) Your friend Prospero wants to use a Gaussian process to predict the high temperature in Ithaca. Prospero believes that the load varies based on the time of the year, and he wants to have his GP predict the (normalized) temperature as a function of the number of days since January 1. However, Prospero is unsure as to how to choose a kernel for this task. He has made a list of some kernels he thinks might be reasonable to use, and sends them to you for feedback. Unfortunately, upon looking at Prospero's list, you find that, unfortunately, none of Prospero's "kernels" are actually kernels!

   Recall that a kernel is a function $k(x, y)$ which can be written as an inner product of feature maps $k(x, y) = \langle \phi(x), \phi(y) \rangle$: equivalently, one that is symmetric ($k(x, y) = k(y, x)$) and positive definite in the sense that for any $x_1, \ldots, x_n$ and scalars $c_1, \ldots, c_n$,

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(x_i, x_j) \geq 0,$$

   or equivalently that any Gram matrix (a.k.a. kernel matrix) for the kernel is positive semidefinite.

   For each of Prospero's proposed "kernels" $k(x, y)$ below, where x and y are both real numbers in $[0, 365)$, prove that it *isn't* a kernel.

   [A]
$$k(x, y) = \exp\left(-\left(\frac{x - y}{365} \right)^3 \right).$$

   [B]
$$k(x, y) = |x - y|.$$

   [C]
$$k(x, y) = 1 - xy.$$

   [D]
$$k(x, y) = \exp\left((x - y)^2 \right).$$

2. (4 pts) Now Prospero proposes yet another kernel,

$$k(x, y) = \cos(x - y).$$

Is Prospero's new function a kernel? Justify your answer using the definition of "kernel" given above. (You may find the identity $\cos(\alpha - \beta) = \cos(\alpha)\cos(\beta) + \sin(\alpha)\sin(\beta)$ to be useful.)

3. (9 pts) In class, we talked about kernel linear models of the form

$$f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(w^T\phi(x_i); y_i)$$

where n is the number of training examples, $\mathcal{L}$ is some loss function, and $x_i \in \mathbb{R}^d$, and y_i are our training examples and training labels, and $\phi : \mathbb{R}^d \to \mathbb{R}^D$ is some feature map that corresponds to a kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ where $k(x, y) = \langle\phi(x), \phi(y)\rangle$. (Note that D could be ∞ in which case the feature map is to an infinite-dimensional Hilbert space.)

We discussed six ways to compute SGD for this objective:

① Transform the features on the fly and compute SGD on w.

② Pre-compute and cache the transformed features, forming $z_i = \phi(x_i)$, and compute SGD on w.

③ Run a kernelized SGD and compute the kernel values $K(x_j, x_i)$ on the fly as they are needed.

④ Run a kernelized SGD on and pre-compute the kernel values for each pair of examples, forming the Gram matrix, store it in memory, and then use this during training as needed.

⑤ Pre-compute an approximate feature map ψ (with a smaller dimension than D) and use it to compute approximate features on the fly to compute SGD.

⑥ Pre-compute an approximate feature map, then also pre-compute and cache the transformed approximate features, forming vectors $z_i = \psi(x_i)$. Then run SGD.

For each of the following scenarios, **fill in** the circle associated with the **best** way of computing SGD in this scenario. Unless otherwise indicated, suppose that we will run SGD for a large number of iterations, such that the extra computational cost of any pre-computation will be effectively amortized across all the iterations of SGD. Also, **cross out** ($\times$) any circles associated with ways that are **computationally infeasible** to run on an ordinary desktop CPU in the described scenario. Briefly explain your answer.

(a) The feature map has $D = 32$. The dimension of the examples $d = 16000$ is relatively large, while the number of examples $n = 10^7$ is also quite large.

① ② ③ ④ ⑤ ⑥

(b) The kernel function is a polynomial kernel $k(x, y) = (1 + x^T y)^6$. The dimension of the examples $d = 100$ is relatively large, while the number of examples $n = 20$ is relatively small. An approximate feature map $\psi : \mathbb{R}^d \to \mathbb{R}^{D_{\text{approx}}}$ sufficient for our purposes could be computed with dimension $D_{\text{approx}} = 100000$.

① ② ③ ④ ⑤ ⑥

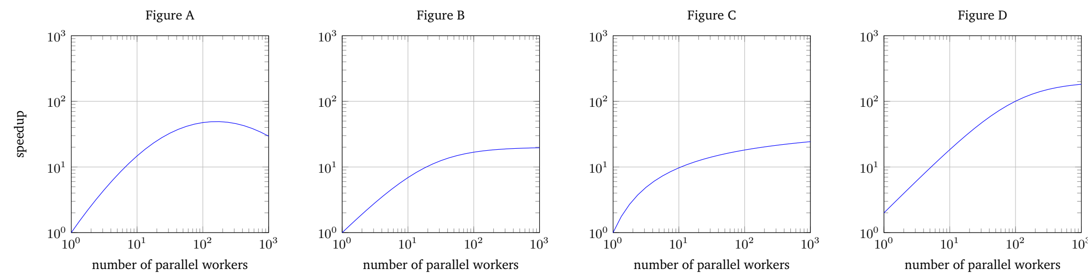
(c) The kernel function is the RBF kernel $K(x, y) = \exp(-\gamma \cdot \|x - y\|^2)$. The dimension of the examples $d = 32$ is relatively small, while the number of examples $n = 10^7$ is relatively large. An approximate feature map $\psi : \mathbb{R}^d \to \mathbb{R}^{D_{\text{approx}}}$ sufficient for our purposes could be computed with dimension $D_{\text{approx}} = 20000$.

① ② ③ ④ ⑤ ⑥

4. (4 pts) What is a Gaussian process? What role do kernels play in Gaussian processes? What role does a Gaussian process play in Bayesian Optimization?

5 [13] Parallelism and Distributed Learning

1. (3 pts) Which one of the following figures displays a parallel speedup as predicted by Amdahl's law?



   State what Amdahl's law is, then use it to explain your answer.

2. (4 pts) Draw a diagram that illustrates how distributed learning using SGD with the *parameter server* approach works. Then write simple high-level pseudocode (e.g. your pseudocode can say things like "compute a gradient" instead of writing maths notation) that explains how this algorithm works.

3. (6 pts) Your friend Minerva comes to you with the following threaded implementation of SGD.

```python
# Xs, Ys          training examples and labels (d * n), (c * n)
# alpha           step size
# W0              the initial value of the parameters (c * d)
# B               minibatch size (guaranteed to be a multiple of num_threads)
# num_epochs      number of epochs (passes through the training set) to run
# num_threads     how many threads to use
def sgd_threaded(Xs, Ys, W0, alpha, B, num_epochs, num_threads):
  (d, n) = Xs.shape
  Bt = int(B / num_threads)  # batch size per worker
  W = numpy.copy(W0)
  Gts = [numpy.zeros(W0.shape) for it in range(num_threads)]

  # construct the barrier object
  iter_barrier = Barrier(num_threads + 1)

  # a function for each thread to run
  def thread_main(ithread):
    for it in range(num_epochs):
      for ibatch in range(int(n/B)):
        # work done by thread in each iteration
        ii = range(ibatch*B, (ibatch+1)*B)
        # computes the average gradient of the examples with indexes in ii
        Gts[ithread][:] = multinomial_logreg_grad_i(Xs, Ys, ii, W)
        iter_barrier.wait()
        iter_barrier.wait()

  worker_threads = [Thread(target=thread_main, args=(it,)) for it in range(num_threads)]
  for t in worker_threads:
    t.start()

  for it in range(num_epochs):
    for ibatch in range(int(n/B)):
      # work done on a single thread at each iteration
      W -= alpha * sum(Gts)
      iter_barrier.wait()
      iter_barrier.wait()

  for t in worker_threads:
    t.join()

  return W # return the learned model
```

Unfortunately, Minerva observes three issues with her program.

- Her parallel code doesn't seem to be running any faster than the sequential version.
- She observes different output when running her program with different numbers of threads.
- Sometimes she even observes different output from multiple runs with the same number of threads.

Find the cause of these three (3) errors in her program, and briefly explain why the errors occur, what lines of code they are caused by, and how you would change the source to fix them. (Hint: the errors are found across three (3) different lines of the program's source.)

6 [9] Hyperparameter Optimization

1. (7 pts) Your friend Jessica is doing a hyperparameter grid search. She shows you the following code.

```
def grid_search():
  # form the grid
  alphas = [-2.0, -1.0, 0.0, 1.0, 2.0, 3.0] # step size
  betas  = [0.01, 0.03, 0.1, 0.3, 1.0, 3.0] # momentum
  gammas = [0.0, 0.01, 0.03, 0.1] # l2 regularization
  # initial weights for training
  W0 = np.zeros(model_size)
  W = W0
  # array to hold hyperparam runs
  best_error = 0.0
  best_results = None
  # iterate over the grid
  for i in range(len(alphas)):
    for j in range(len(betas)):
      for k in range(len(gammas)):
        # train with these hyperparameters
        W = train(W, alphas[i], betas[j], gammas[k])
        err = get_test_error(W)
        if (err < best_error):
          best_error = err
          best_results = (W, alphas[i], betas[j], gammas[k])
  return best_results
```

Without looking at the rest of Jessica's code, identify **five** separate bugs in her program (each localized on a single line), and explain how you would fix them.

2. (2 pts) Jessica fixes her code by following your suggestions, in a way that does not change the grid size for any of the parameters she is searching over. What is the number of times Jessica's corrected code will need to run the train function? Justify your answer.