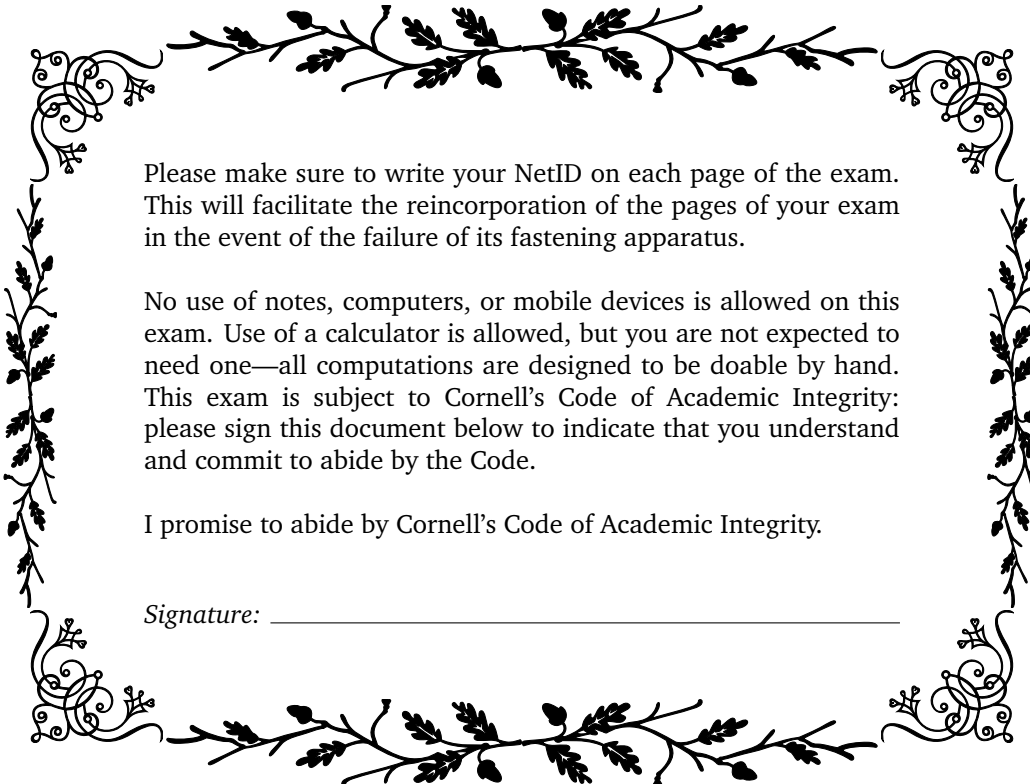


CS4787 Final Exam Solutions

Fall 2022

NAME:	
Net ID:	
Email:	

(Total points possible: 104)



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 [?: 20] 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.
F. This is a hyperparameter that trades off between exploration and exploitation.
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. T.
3. (T/F) If the parameter vectors w_t reached after t iterations of SGD diverge to infinity (i.e. $\lim_{t \rightarrow \infty} \|w_t\| = \infty$) then so must the total loss $f(w_t)$. F. This is not necessarily true, for example consider gradient descent on $f(w) = \exp(w)$.
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. F. When a model is overfitting over time, then early stopping also improves validation error by preventing that overfitting.
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 $\sqrt{\kappa}$. T.
6. (T/F) The TPU stands for “transformer processing unit” and was designed to accelerate the self-attention layers of transformer networks. F. The “T” in “TPU” stands for “tensor processing unit” and it accelerates general deep learning, not just transformers.

7. (T/F) The number of flops required to compute the empirical risk is independent of the condition number κ . T.
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. F. For this to work we'd need more like 33% nonzeros, and less if we expect to get computational benefit
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. T.
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. F. This is true of PCA, not autoencoders.
11. (T/F) Deep neural network inference is only ever run in the cloud. F. It's also run on edge devices, such as smartphones/laptops, as well as in other embedded contexts.
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 . F. Gradients via backprop take time linearly proportional to the time it takes 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 it 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.

Answers may vary.

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.
This vector space V is the space of linear maps from $\mathbb{R}^{m \times n}$ to $\mathbb{R}^{n \times p}$. This space's dimension is the product of the dimensions of the input space and the output space, so the dimension of V is mn^2p .

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

Yes, this function has a condition number. Its second derivative matrix has eigenvalues 8 and 2 at all times, so its strong convexity constant is 2, its L-smoothness constant is 8, and its condition number is $\kappa = 8/2 = 4$.

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

Batch norm is a neural network layers that can improve convergence and generalization. It works by normalizing mean and variance of activations across a minibatch; it retains expressivity by adding a learned affine transformation afterwards. While minibatch statistics are used at training time, at inference time instead the statistics of the whole training set (or, more commonly, a running average) are used.

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.

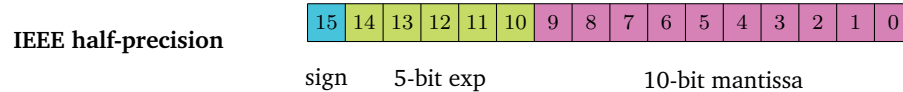
Franz diminishes Hans's memory by a factor of 4 by removing the ensemble, by a factor of 4^2 from diminishing the widths, and by a factor of 2 by going from 32-bit to bfloat. The total savings is then $128\times$, reducing Hans's network size to a bit more than 2 GB (because the first and last layers are diminished less).

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.

Ferdinand is wrong; we often expect regret to increase. Good performance would have regret increasing sublinearly, not decreasing to 0.

3 [18] Quantization and Low-Precision Arithmetic

- (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.



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

$$s \cdot 2^{e-15} \cdot (1.m_9m_8 \dots m_0)_b.$$

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 $2^{15} \cdot (1.111111111)_b = 2^{15} \cdot (2 - 2^{-10}) = 65504$.

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.

The largest representable number is going to be something like $2^{2^{B-1}}$. For $B = 7$, this is 2^{128} , which is huge. That's more than enough to represent this. Note that for $B = 6$, this is 2^{32} , about 4 billion, which is not enough.

- (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?

Micron's new format has fewer mantissa bits, so it will have a larger machine epsilon and greater rounding error than IEEE 16-bit floats.

- (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.

$1280 = 256 \cdot 5 = 2^8 \cdot 5 = 2^{10} \cdot 1.01_b$. Our exponent field will be $10 + 15 = 25 = 16 + 8 + 1 = 11001_b$, so the whole bit pattern will be 0110010100000000.

4. (2 pts) Can Micron's new 16-bit floating-point format actually represent the number 100 billion exactly? Justify your answer.
 They couldn't do it because 100 billion has too many "digits" of significant figures. It's basically $2^{11} \cdot 5^{11}$ so the mantissa needs to store 5^{11} , and that will involve way too many "binary decimal places", since it's obviously going to be more than 11.

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. This represents the real number 0 (or negative 0, if the sign bit is set). This arises whenever zero is the result of a computation, e.g. at the output of a ReLU that's off.

 - (b) The exponent field is all zeros, and the mantissa field is not all zeros. This represents a denormal number, which are the smallest-in-magnitude representable numbers of the format. These arise whenever a number is too small in magnitude to be represented as a normal number, but not smaller than the smallest denormal number. Often happens during gradient computation due to "vanishing gradient."

 - (c) The exponent field is all ones (e.g. 255 for a 32-bit float), and the mantissa field is all zeros. This represents positive or negative infinity, depending on the sign bit. These can result from overflow or division by zero. This can easily happen if training diverges.

 - (d) The exponent field is all ones (e.g. 255 for a 32-bit float), and the mantissa field is not all zeros. This represents NaN, not a number. These can result from invalid computations like $0/0$.

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, \dots, x_n and scalars c_1, \dots, 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]

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

This is not a kernel because it is not symmetric in x and y .

[B]

$$k(x, y) = |x - y|.$$

This is not a kernel because the resulting kernel matrix for the dataset $x_1 = 0, x_2 = 1$ would be $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ which has negative determinant, so its eigenvalues can't all be positive as required for a kernel.

[C]

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

This is not a kernel because $k(0, 0) = -1$.

[D]

$$k(x, y) = \exp((x - y)^2).$$

This is not kernel. The resulting kernel matrix for the dataset $x_1 = 0, x_2 = 1$ would be $\begin{bmatrix} 1 & e \\ e & 1 \end{bmatrix}$ which has negative determinant $1 - e^2$, so its eigenvalues can't all be positive as required for a kernel.

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 $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^D$ is some feature map that corresponds to a kernel $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ 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

Since D is small, it would be best to run (2), since transforming the features first allows us to work purely in the D -dimensional feature space without having to process the much larger d -dimensional training examples x_i at each step. $n^2 = 10^{14}$ which would correspond to about 4TB of memory to store the Gram matrix, so (4) is infeasible. Also, since D is small, we won't expect to see much benefit from computing an approximate feature map.

- (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 $\psi : \mathbb{R}^d \rightarrow \mathbb{R}^{D_{\text{approx}}}$ sufficient for our purposes could be computed with dimension $D_{\text{approx}} = 100000$.

① ② ③ ④ ⑤ ⑥

Since the number of examples is small, simply running (4), with a pre-computed kernel matrix, is going to perform the best. (1) and (2) are infeasible in this scenario because $D = 100^6 = 10^{12}$ so computing directly with w is impossible.

- (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 $\psi : \mathbb{R}^d \rightarrow \mathbb{R}^{D_{\text{approx}}}$ sufficient for our purposes could be computed with dimension $D_{\text{approx}} = 20000$.

① ② ③ ④ ⑤ ⑥

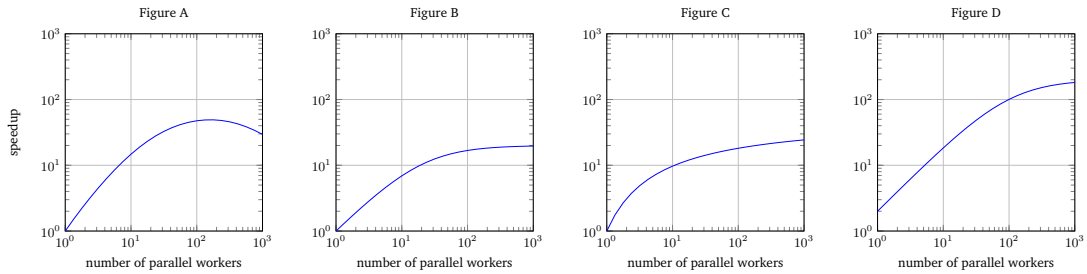
Since $D = \infty$, (1) and (2) are again infeasible in this scenario. (4) will also be infeasible because the Gram matrix will be too large. Finally, (6) would be infeasible because we would need to store transformed approximate features of size $10^7 \cdot 2 \cdot 10^4 = 2 \cdot 10^{10} \approx 400$ GB which is infeasible to fit in memory on an ordinary CPU. The best remaining choice is (5).

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?

A GP is basically an infinite-dimensional generalization of a multivariate Gaussian. The kernel function expresses the covariance operator of this process. Gaussian processes form the prior distribution for the search done in Bayes opt.

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.

It's B. A can't work because it's non-monotonic. C can't work because it predicts super-linear speedup, and D can't work because it has a non-unit speedup at one thread, which is nonsensical.

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.

Answers may vary. Diagram needs to include a parameter server and workers, and explain how they compute SGD.

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

```

1  # Xs, Ys          training examples and labels (d * n), (c * n)
2  # alpha           step size
3  # W0              the initial value of the parameters (c * d)
4  # B               minibatch size (guaranteed to be a multiple of num_threads)
5  # num_epochs      number of epochs (passes through the training set) to run
6  # num_threads     how many threads to use
7  def SGD_threaded(Xs, Ys, W0, alpha, B, num_epochs, num_threads):
8      (d, n) = Xs.shape
9      Bt = int(B / num_threads) # batch size per worker
10     W = numpy.copy(W0)
11     Gts = [numpy.zeros(W0.shape) for it in range(num_threads)]
12
13     # construct the barrier object
14     iter_barrier = Barrier(num_threads + 1)
15
16     # a function for each thread to run
17     def thread_main(ithread):
18         for it in range(num_epochs):
19             for ibatch in range(int(n/B)):
20                 # work done by thread in each iteration
21                 ii = range(ibatch*B, (ibatch+1)*B)
22                 # computes the average gradient of the examples with indexes in ii
23                 Gts[ithread][:] = multinomial_logreg_grad_i(Xs, Ys, ii, W)
24                 iter_barrier.wait()
25                 iter_barrier.wait()
26
27     worker_threads = [Thread(target=thread_main, args=(it,)) for it in range(num_threads)]
28     for t in worker_threads:
29         t.start()
30
31     for it in range(num_epochs):
32         for ibatch in range(int(n/B)):
33             # work done on a single thread at each iteration
34             W -= alpha * sum(Gts)
35             iter_barrier.wait()
36             iter_barrier.wait()
37
38     for t in worker_threads:
39         t.join()
40
41     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.)

- Line 36 or 37: There is a synchronization error here. The barrier on line 36 needs to be moved to line 33. is needed to make sure the next iteration's gradients are only computed after the parameters W are updated. This causes a race condition which makes multiple runs have different results.
- Line 21: This line is not selecting a subset of the minibatch to compute on each worker, but is instead assigning every worker to compute every element of the minibatch. To fix this, the line needs to be changed to "ii = range(ibatch*B + ithub*Bt, ibatch*B + (ithub+1)*Bt)". This is what is causing her parallel code to not run any faster.
- Line 34 or 35: G must be divided by the number of threads, since we want the average gradient from the workers, not the sum of all the gradients. This is what is causing Minerva to observe

Net ID: _____

different output with different numbers of threads.

6 [9] Hyperparameter Optimization

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

```

1  def grid_search():
2      # form the grid
3      alphas = [-2.0, -1.0, 0.0, 1.0, 2.0, 3.0] # step size
4      betas = [0.01, 0.03, 0.1, 0.3, 1.0, 3.0] # momentum
5      gammas = [0.0, 0.01, 0.03, 0.1] # l2 regularization
6      # initial weights for training
7      W0 = np.zeros(model_size)
8      W = W0
9      # array to hold hyperparam runs
10     best_error = 0.0
11     best_results = None
12     # iterate over the grid
13     for i in range(len(alphas)):
14         for j in range(len(betas)):
15             for k in range(len(gammas)):
16                 # train with these hyperparameters
17                 W = train(W, alphas[i], betas[j], gammas[k])
18                 err = get_test_error(W)
19                 if (err < best_error):
20                     best_error = err
21                     best_results = (W, alphas[i], betas[j], gammas[k])
22     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.

Three of the below would suffice.

- Jessica's step size hyperparameters are sometimes out of range. The step size should always be positive. Usually a logarithmic grid is more common.
- Jessica's momentum hyperparameters are sometimes out of range. Momentum should always be between 0 and 1, and she should adjust her grid to compensate.
- The initial best error should be initialized to be large, not to be zero. This will always return None.
- Jessica is running HPO on the test set with the get test error function! Not good! Use validation error instead.
- Jessica seems to be initializing training for subsequent grid entries starting at the model outputted from the previous grid entry—or at least, she is passing W from the previous grid entry into her train function, which is very suspicious. This is unfair. Instead, she should pass in W_0 .

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.

$$6 \times 6 \times 4 = 144.$$