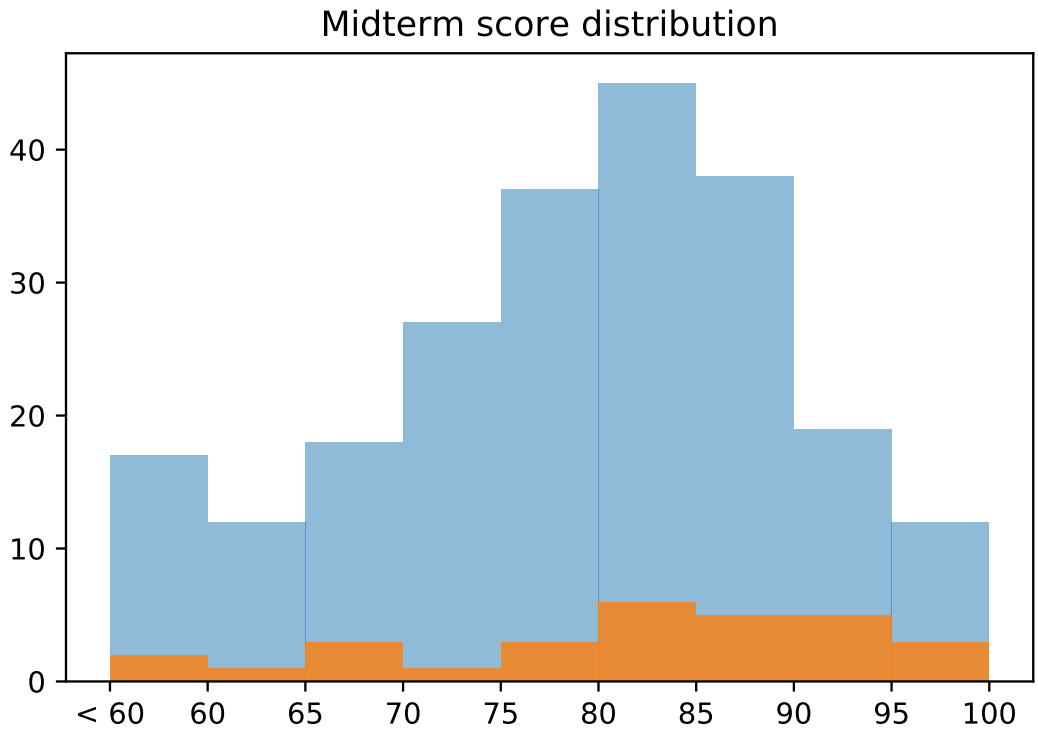
ECE 239AS, Winter 2019 **Midterm Overview**
Department of Electrical Engineering Prof. J.C. Kao
UCLA ECE TAs W. Chuang & M. Kleinman & K. Liang & A. Wickstrom

Midterm statistics (out of 100 maximum points):



Overall statistics (across whole class; 254 exams)
High score: 100
Mean score: 78.4
Median score: 80.5
Standard deviation: 11.8

Graduate student statistics (225 exams)
High score: 100
Mean score: 78
Median score: 80
Standard deviation: 11.4

Undergraduate student statistics (29 exams)
High score: 100
Mean score: 80.6
Median score: 84
Standard deviation: 14.4

ECE 239AS, Winter 2019 **Midterm**

Department of Electrical Engineering Prof. J.C. Kao
UCLA ECE TAs W. Chuang & M. Kleinman & K. Liang & A. Wickstrom

UCLA True Bruin academic integrity principles apply.
4 cheat sheets allowed
6:00 - 7:50pm.
Wednesday, 20 Feb 2019.

State your assumptions and reasoning.
No credit without reasoning.
Show all work on these pages.

Name: _____

Signature: _____

ID#: _____

Problem 1	_____	/ 20
Problem 2	_____	/ 20
Problem 3	_____	/ 20
Problem 4	_____	/ 20
Problem 5	_____	/ 20
Total	_____	/ 100

1. **Training, validation, and testing.**

   (a) (5 points) Briefly explain the purpose of the training, validation, and testing set. Comment on how many times each set should be used when doing k-fold cross validation.

   (b) (10 points) Consider a dataset where a signal evolves through time. The data is sampled at an extremely high rate, meaning that adjacent time points are highly correlated. Consider also that this dataset has few trials. Your colleague comes up with an idea to increase the number of effective trials: he decides to subsample the data with no overlapping time points (so, for instance time points $1, 3, 5, 7, 9...$ could represent one effective trial and time points $2, 4, 6, 8, ...$ another). He tells you he now has twice the number of effective trials.

      i. (3 points) Why might it be beneficial to subsample the data to increase the number of effective trials?

      ii. (7 points) After subsampling the data, your colleague randomly split his trials (of which he now has double) into a training and validation set. He tells you that his validation accuracy is much higher than anything you have achieved. Suggest why this might be the case.

   (c) (5 points) Another colleague splits the data into 80% training, 10% validation, and 10% test. This colleague did not subsample the data. He has experimented with various architectures, each time training on the training set, validating on the validation set, and then testing on the testing set. Based on his testing accuracy, he would see which hyperparameters worked well, and make modifications to his model, reiterating the above process. Will the final test accuracy reported be an accurate proxy for how well his model will generalize to new data? Briefly explain.

Solution:

(a) (5 points) The training set is used to adjust the parameters of the network to fit the input examples. The validation set is used to select optimal hyperparameters. The testing set is to be used as a proxy to see how well the model (the architecture and hyperparameters) would generalize to new data. The test set should only be used once, after the cross-validation procedure, where each fold will be used k times for training/validation.

(b) (10 points)

   i. (3 points) Increasing the number of trials can allow for a more complicated/descriptive model to be learned. It can also help against overfitting and make the model more robust.

   ii. (7 points) The group was effectively training on the validation set, since the subsampled trials were highly correlated, and hence the parameters were adjusted to fit the highly correlated trials in the training set, which were effectively in the validation set as well. They should have first split the training and validation set, and then subsampled.

(c) (5 points) No, it will be an overestimate. By testing on the training set multiple times, they will have overfit to the testing set.

2. **Backpropagation.** Consider a 3 layer neural network (NN), with $\mathbf{x} \in \mathbb{R}^n$ as input and $\mathbf{y} \in \mathbb{R}^m$ as the target value. The NN is constructed as the following:

$$\begin{aligned} \mathcal{L} &= \frac{1}{2}||h_3(h_2(h_1(\mathbf{x}))) - \mathbf{y}||^2 \\ h_1(\mathbf{x}) &= \text{PReLU}(\mathbf{W}_1\mathbf{x}) \\ h_2(\mathbf{x}) &= \text{ELU}(\mathbf{W}_2\mathbf{x}) \\ h_3(\mathbf{x}) &= \mathbf{W}_3\mathbf{x} \end{aligned}$$

where

- $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{n\times n}$ and $\mathbf{W}_3 \in \mathbb{R}^{m\times n}$
- For the PReLU unit, $f(x) = \max(\alpha x, x)$.
- For the ELU unit, $f(x) = \max(\alpha(e^x - 1), x)$.

for $0 < \alpha < 1$.

(a) (5 points) Draw the computational graph for this neural network.

(b) (15 points) Calculate $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1}, \frac{\partial \mathcal{L}}{\partial \mathbf{W}_2}, \frac{\partial \mathcal{L}}{\partial \mathbf{W}_3}$ using backpropagation. You may define intermediate variables and write the gradients in terms of these intermediate variables.

Hint: For a ReLU function $f(x) = \max(0, x)$, we compute the gradient for $x < 0$ and $x > 0$ separately. For instance, $\frac{\partial f(x)}{\partial x} = \mathbb{1}\{x > 0\} \cdot 1 + \mathbb{1}\{x < 0\} \cdot 0$. Compute the gradient for PReLU and ELU with the same method.

Solution:

(a) We gave full points for a correctly drawn computational graph.

(b) Let

$$\begin{aligned} e &= \mathbf{W}_1 x \\ d &= ReLU(e) \\ c &= \mathbf{W}_2 d \\ b &= ELU(c) \\ a &= \mathbf{W}_3 b \end{aligned}$$

Therefore, $y = ||a||^2$.
$\frac{\partial \mathcal{L}}{\partial a} = a - y$

$$\frac{\partial \mathcal{L}}{\partial W_3} = \frac{a}{\partial W_3} \cdot \frac{\partial \mathcal{L}}{\partial a} = (a - y)b^T$$

$\frac{\partial a}{\partial b} = W_3^T$
$\frac{\partial b}{\partial c} = \text{diag}(\mathbb{1}\{c_1 < 0\} \cdot \alpha e^{c_1} + \mathbb{1}\{c_1 > 0\}, ..., \mathbb{1}\{c_n < 0\} \cdot \alpha e^{c_n} + \mathbb{1}\{c_n > 0\})$

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{\partial c}{\partial W_2} \cdot \frac{\partial b}{\partial c} \cdot \frac{\partial a}{\partial b} \cdot \frac{\partial \mathcal{L}}{\partial a}$$
$$= \frac{\partial b}{\partial c} \cdot W_3^T \cdot (a - y) \cdot d^T$$

$\frac{\partial c}{\partial d} = W_2^T$
$\frac{\partial d}{\partial e} = \text{diag}(\mathbb{1}\{e_1 < 0\} \cdot \alpha + \mathbb{1}\{e_1 > 0\}, ..., \mathbb{1}\{e_n < 0\} \cdot \alpha + \mathbb{1}\{e_n > 0\})$

$$\frac{\partial \mathcal{L}}{\partial W_1} = \frac{\partial e}{\partial W_1} \cdot \frac{\partial d}{\partial e} \cdot \frac{\partial c}{\partial d} \cdot \frac{\partial \mathcal{L}}{\partial c}$$
$$= \frac{\partial d}{\partial e} \cdot W_2^T \cdot \frac{\partial b}{\partial c} \cdot W_3^T \cdot (a - y) \cdot x^T$$

3. **Regularization.**

   (a) (5 points) You are tasked with training a feedforward neural network. Your boss asks you to shrink the effective size of the model by using regularization. How can you complete the task?

   (b) (5 points) Why can $L2$ regularization be referred to as "weight decay"?

   (c) (5 points) How does batch normalization help a neural network to be more robust to initialization?

   (d) (5 points) How does dropout act as a regularizer?

Solution:

(a) (5 points) This question was designed to test your knowledge of applying regularization methods covered in class to a given model. In the problem statement, shrinking the "effective size" of the model implies reducing the number of significant parameters without directly altering the model architecture. Points were subtracted if the proposed solution directly altered the architecture of the model, or invoked additional models.

You can shrink the effective size of the model by invoking L^1 regularization, which will impose a sparsity constraint on the model. That is, some parameters will have an optimal value of zero. This can be implemented by adjusting the loss and gradient of the model. Specifically:

$$\Omega(\theta) = \|\boldsymbol{W}\|_1$$
$$= \sum_{i,j} |W_{i,j}|$$

The loss function then becomes:

$$\tilde{J}(\boldsymbol{W}) = \alpha \|\boldsymbol{W}\|_1 + J(\boldsymbol{W})$$

which will encourage a subset of the weights to become zero. This results in a model with a smaller effective size.

Full points were given for adding an L^1 penalty to the loss function, and describing how this would effect the effective model size.

The following solutions received partial credit:

- **Dropout**: During each forward pass, it may seem that the model is smaller because some of the activations are "zeroed-out". But in practice, over the course of an actual set of training epochs, the probability is quite high that each unit gets included in at least one of the subnetworks within the model. That is, all of the units most likely end up being trained on some of the input data, and as such, comprise an integral part of the model. So at test time, a model using dropout still has the same number of parameters as a vanilla network with an otherwise identical architecture. Besides, the weights corresponding to the units that get dropped during a given forward pass aren't actually zeroed out – they are simply not updated during the subsequent backpropagation and weight update.
  Finally, scaling the activations by p during test time doesn't reduce the size of the model, as the number of nonzero weights stays the same.
- **L2**: L^2 regularization shrinks the magnitude of the weights, but the overall size of the model stays the same. Specifically, L^2 does not cause parameters to be sparse. L1 regularization imposes sparsity, which shrinks the effective size of the model.
- **Definitions of regularization**: This category of solutions described generally how regularizers may be used to reduce model complexity, but did not describe a specific example to reduce the effective size of the given model.

(b) (5 points) L^2 regularization favors models with smaller weights. With the L^2 penalty in the loss function, the weights will have a tendency to decay over time compared to a model without L^2 penalty.
Let ϵ, α equal nonzero learning rate and regularization hyperparameter, respectively.
Specifically:

$$\begin{aligned}
\Omega(\theta) &= \frac{1}{2}\|\boldsymbol{W}\|_2^2 \\
&= \frac{1}{2}\sum_{i,j} W_{i,j}^2 \\
\tilde{J}(\theta) &= \frac{\alpha}{2}\boldsymbol{W}^\top \boldsymbol{W} + J(\theta) \\
\nabla_W \tilde{J}(\theta) &= \alpha \boldsymbol{W} + \nabla_W J(\theta)
\end{aligned}$$

For each gradient step to update the weights:

$$\begin{aligned}
\boldsymbol{W} &\leftarrow \boldsymbol{W} - \epsilon\left(\alpha \boldsymbol{W} + \nabla_W J(\theta)\right) \\
&\leftarrow (1 - \epsilon\alpha)\boldsymbol{W} - \epsilon \nabla_W J(\theta)
\end{aligned}$$

By modifying the gradient update with a weight decay term $(1 - \epsilon\alpha)$, for nonzero ϵ, α, the weight matrix shrinks by a constant factor on each training iteration, before applying the gradient update. Across training, this will lead to weight matrices with smaller, more diffuse entries than vanilla gradient update.

(c) (5 points) This question was designed to test your intuition on the batchnorm algorithm, as well as your understanding how neural networks are sensitive to initialization.

By enabling the outputs at each layer to be normalized, batch normalization (BN) ensures that none of the activations will decay to zero or explode to infinity after cascading through multiple layers. As a result, the gradients are more robust to vanishing or exploding during backpropagation.
Why is this a benefit?
Vanishing gradients make training near impossible because of information loss: it becomes impossible to effectively propagate the gradient upstream when you multiply by zero at some deep layer. Exploding gradients are impossible to train because as they tend toward infinity, your gradient becomes so large that you either take large steps far away from the starting point into some new subspace of the loss landscape, or they are large enough to cause overflow errors.

Poor initializations include those drawn from distributions with variance too low or too high, which would facilitate activations during the forward pass to either vanish or explode after composition through multiple layers. Additionally, initializing from a distribution with a large mean would contribute to multiplicative exponential growth of activations, even with small variance.

The following solutions received partial credit:

- Mentioning that BN normalizes activations, but failing to describe how this adds robustness
- Stating that "parameters in lower layers change the statistics of input to a given layer" without describing when and how this can be problematic
- Stating that gradient descent expects unchanging distributions when updating layers, but failing to describe relevance to initialization robustness.

(d) (5 points) By applying a random mask to activations at each training iteration, dropout aims to approximate training a large (2^N) ensemble of models, each comprised of the units that passed through a mask. If we view dropout as training these multiple subnetworks within the full model, and averaging their predictions at test time, dropout can be seen as an approximation to bagging. The slight distinction is that in dropout, all the models share a significant portion of weights.

Full credit was awarded for describing how dropout approximates bagging, and how this constitutes regularization.
Partial credit was awarded for listing one or more of the following properties of dropout, with extra points for justification:

- Similar to corrupting the input data with noise
- Encourages units to work in diverse contexts, which prevents them from overfitting to specific features by "smoothing out" their expressive power
- Encouraging redundant encoding

4. **Optimization.**

(a) (12 points) Suppose, oddly, that an optimizer takes several gradient steps and arrives back at a same parameter setting, $\mathbf{W}$, that is *exactly* the same as in a prior step. This setting of $\mathbf{W}$ does not correspond to a local optima of the loss function. Imagine you are

optimizing this neural network using a batch algorithm, i.e., using the entire training set to calculate a gradient. You are also using a fixed learning rate.

i. (4 points) Consider that the optimizer was a naive gradient descent optimizer (with no momentum or adaptive gradients). Is the gradient step that you will take the second time you are at this parameter setting, $\mathbf{W}$, (circle one)
   - larger (in magnitude)
   - smaller (in magnitude)
   - the exact same, or
   - can't be determined (i.e. not enough information)

   than the gradient step the first time the optimizer was at the parameter setting, $\mathbf{W}$? Justify your answer.

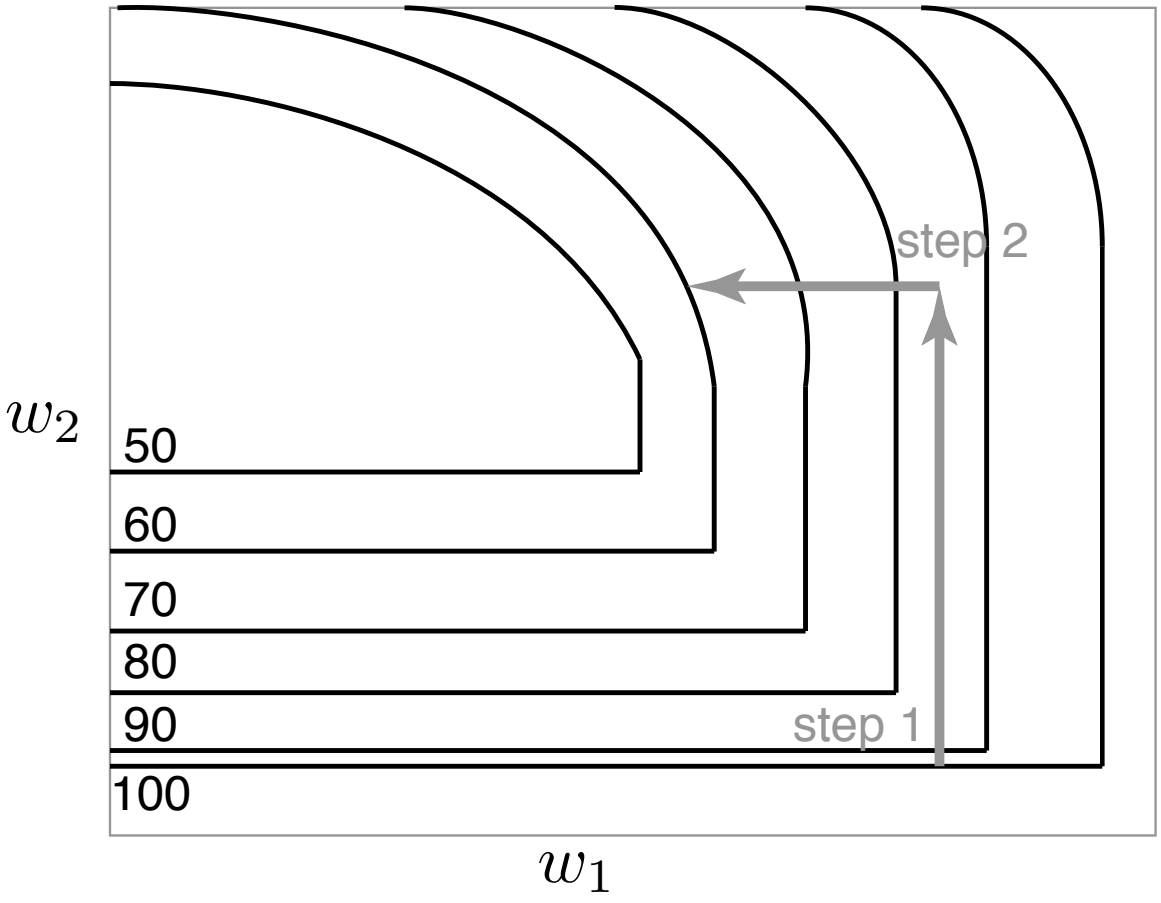
ii. (4 points) Consider that the optimizer was Adagrad. Is the gradient step that you will take the second time you are at this parameter setting, $\mathbf{W}$, (circle one)
   - larger (in magnitude)
   - smaller (in magnitude)
   - the exact same, or
   - can't be determined (i.e. not enough information)

   than the gradient step the first time the optimizer was at the parameter setting, $\mathbf{W}$? Justify your answer.

iii. (4 points) Consider that the optimizer was RMSprop. Is the gradient step that you will take the second time you are at this parameter setting, $\mathbf{W}$, (circle one)
   - larger (in magnitude)
   - smaller (in magnitude)
   - the exact same, or
   - can't be determined (i.e. not enough information)

   than the gradient step the first time the optimizer was at the parameter setting, $\mathbf{W}$? Justify your answer.

(b) (8 points) The following figure is a contour plot where the contour lines denote the value of the loss as a function of two weight variables (corresponding to the x and y-axis). Imagine we have taken two gradient steps given by the gray arrows. Sketch on the same plot the weight update steps taken by SGD, SGD+momentum, and Adagrad after step 2. Do not perform any calculations; the sketch should be arrived at through intuition. Give at most a two sentence explanation for each arrow you've drawn.



Solution:

(a) (a)

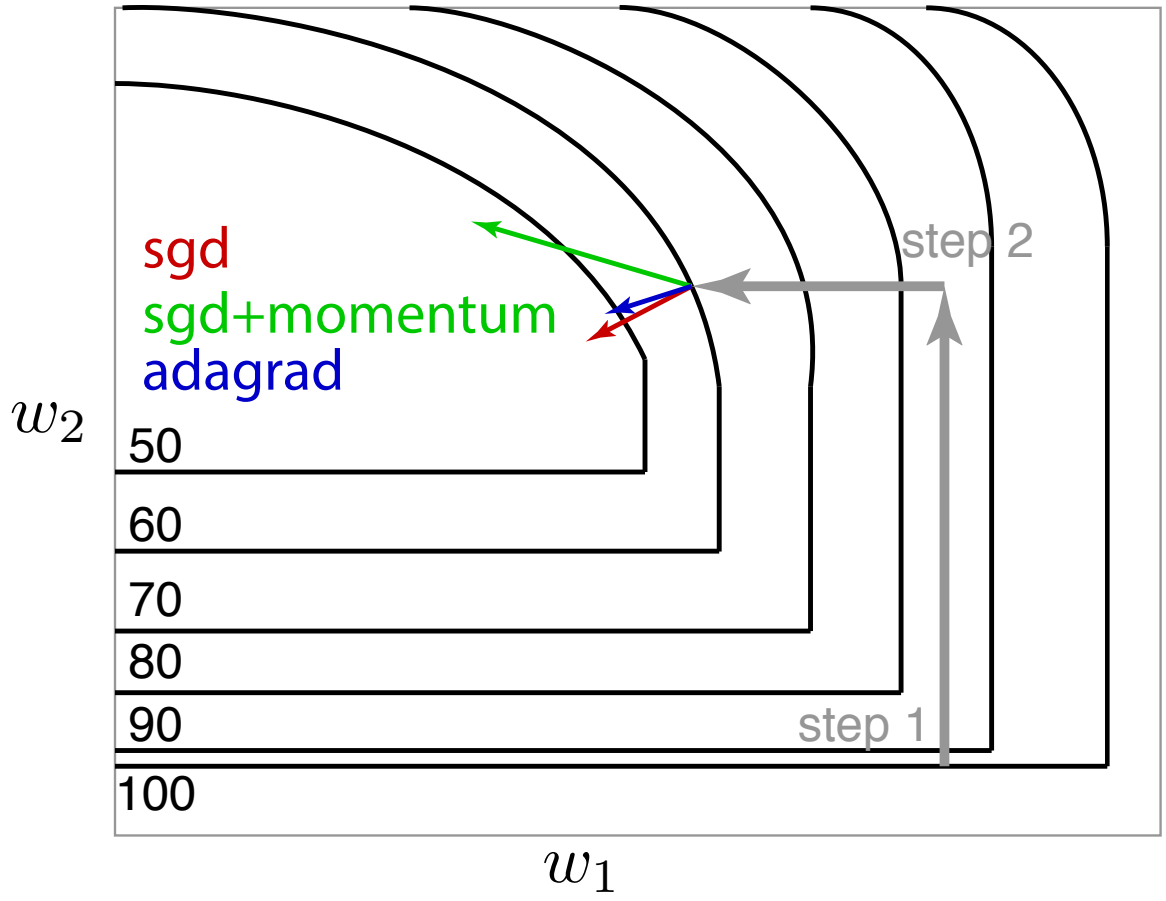
i. Exactly the same. SGD maintains no history and all the data is used to calculate the gradient. Therefore the gradient will be exactly the same. Since the learning rate is also exactly the same, the step is exactly the same.

Note, some students thought this question asked how an experimenter would intervene in the algorithm, or tried to address why we were in this situation. I gave partial credit here, since the question did use the phrasing "you will take." However, we have never talked (nor is it standard practice, except in annealing the learning rate) about intervening into a gradient descent algorithm.

ii. Smaller. Adagrad keeps a running sum of the squared gradients, $\mathbf{a}$. $\mathbf{a}$ will increase over more iterations, leading to the Adagrad steps being smaller.

iii. Can't be determined. Since RMSprop normalizes by an exponential history of squared gradients, and we don't know the gradients before reaching this point, we cannot determine if the step will be bigger or smaller.

(b)



5. **Convolutional neural networks.** Consider a convolutional layer C followed by a max pooling layer P. The input to layer C is $120 \times 120 \times 50$. Layer C has 20 filters, each of which is of size 4×4. The convolution padding is 1 and the stride is 2. Layer P performs max pooling over each of the layer C′s output feature maps, over a 3×3 receptive field, and stride 1. Given $x_1, x_2, ..., x_n$ all scalars, we assume:

   - A scalar multiplication $x_i \cdot x_j$ accounts for one FLOP;
   - A scalar addition $x_i + x_j$ accounts for one FLOP;
   - A max operation $\max(x_1, x_2, ..., x_n)$ accounts for $n - 1$ FLOPs.

   You do not need to calculate the products you write out (e.g., answers maybe left in terms like "$(x \cdot y \cdot w) \cdot z$").

   (a) (5 points) What is the total number of trainable parameters? Please account for the bias term.

   (b) (4 points) What is the size of layer C's output feature map?

   (c) (4 points) What is the size of layer P's output feature map?

   (d) (7 points) How many FLOPs are there in layers C and P during one forward pass? Please include the bias term when calculating the FLOPs.

Solution:

(a) $(4 \times 4 \times 50 + 1) \times 20$

(b) $60 \times 60 \times 20$

(c) $58 \times 58 \times 20$

(d) $20 \times (60 \times 60) \times (4 \times 4 \times 50 \times 2) + (8 \times 58 \times 58 \times 20)$