ECE C147/C247, Winter 2022 **Midterm**
Department of Electrical and Computer Engineering Prof. J.C. Kao
University of California, Los Angeles TAs: T. Monsoor, T. Wang, P. Lu, Y. Li

UCLA True Bruin academic integrity principles apply.
Open: Book, computer.
Closed: Internet, except to visit Bruin Learn and Piazza.
4:00pm-5:50pm.
Wednesday, 16 Feb 2022 (or Saturday, 19 Feb 2022).

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

Name: Sunay Bhat

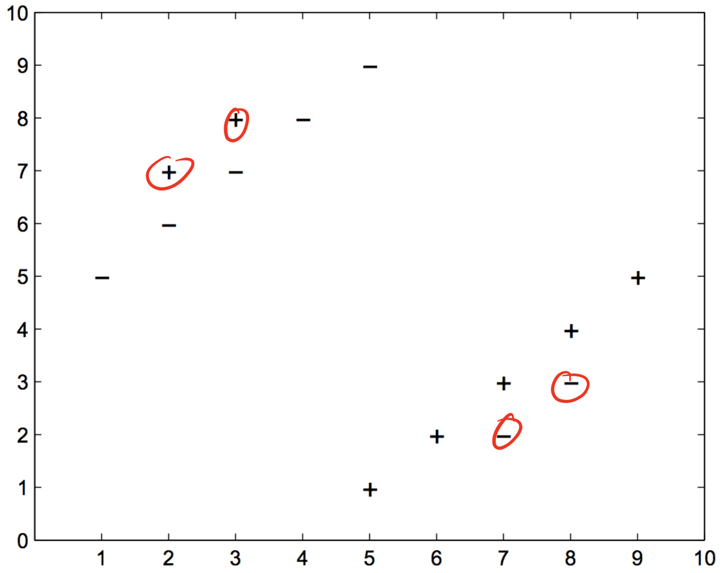
Signature: ____________

ID#: 905 629 072

Problem 1 _____ / 25
Problem 2 _____ / 40
Problem 3 _____ / 25
Problem 4 _____ / 15
BONUS _____ / 5 bonus points

Total _____ / 105 points + 5 bonus points

1. **ML basics** (25 points).



14 total

4 incorrect

(a) (5 points) Consider a k-nearest neighbors binary classifier which assigns the class of a test point to be the class of the majority of the k-nearest neighbors, according to a Euclidean distance metric. Using the data set shown above to train the classifier and choosing $k = 5$, what is the classification error on the training set? Assume that a point can be its own neighbor.

Answer as a decimal with 4 significant figures, e.g. (6.051, 0.1230, 1.234e+7) or a fraction.

$\boxed{4/14 = 0.2857 \ *}$

* Note that the assumption is that the classification error is incorrect divided by total datapoints. There are many other classification errors that could have been used.

(b) (7 points) Assume we have a training and test set drawn from the same distribution, and we would like to classify points in the test set using a k-nearest neighbors classifier.

i. (3 points) In order to minimize the classification error on this test set, we should always choose the value of k which minimizes the training set error.

Select one:

A. True

[✓ **B**. False]

K =1 will always minimize the training error, but might not generalize well (will overfit).

ii. (4 points) Consider two methods for optimizing the hyperparameters.

- **Method 1** chooses the hyperparameters that minimize the training set error.
- **Method 2** splits the data into training and validation sets, and chooses the hyperparameters that minimize the validation error.

Which method is better? Justify with no more than 3 sentences. **Select one:**

A. Method 1

[✓ **B**. Method 2]

We generally choose hyper parameters over a validation set to ensure these parameters do not overfit the training data. Without a separate validation set, we could not effectively choose a hyper parameter that generalizes well and minimizes overfitting.

(c) (5 points) Please select all true statements about k-nearest neighbors:

(Note: Justification is not necessary, but may result in partial credit if the answer is incorrect.)

Select all that apply:

[✓ **A**] Increasing k will generally result in a smoother decision boundary.

[✓ **B**] Icreasing k will generally reduce the impact of noise or outliers in the data.

C Increasing k increases the likelihood of overfitting the data.

[✓ **D**] It is possible to use cross-validation to select the value of k.

E We should never select the k that minimizes the error on the validation dataset.

F None of the above.

(d) (8 points) Consider a classifier trained till convergence on some training data D^{train}, and tested on a separate test set D^{test}. You evaluate the test error, and find that it is very high. You then compute the training error and find that it is close to 0.

i. (3 points) Has this classifier (1) underfit, (2) reasonably fit, or (3) overfit the data?

It has overfit the data (3).

ii. (5 points) Which of the following are expected to help improve this classifier? (Note: Justification is not necessary, but may result in partial credit if the answer is incorrect.)
Select all that apply:

[✓ **A**.] Increase the training data size.

B. Decrease the training data size.

C. Increase model complexity.

[✓ **D**.] Decrease model complexity.

E. Train on a combination of D^{train} and D^{test} and test on D^{test}.

F. Conclude that Machine Learning does not work.

E will help test error by "cheating" the test set (seeing it while training), and not actually improve the classifier.

2. **Detecting signature forgery using similarity network** (40 points)

Bank of Westwood has been receiving many complaints from its clients about their signatures being forged. In order to address this problem, the bank has decided to hire you for designing a machine learning system for detecting signature forgery. You have learned about the similarity network recently and want to use it for this problem.

A similarity network is a Fully Connected Feedforward network that accepts distinct inputs but share the same weights. To be precise, $\{(\mathbf{x}^{(i)}, \hat{\mathbf{x}}^{(i)}), y^{(i)}\}$ constitutes the i^{th} training example, where ($\mathbf{x}^{(i)} \in \mathbb{R}^d$,$\hat{\mathbf{x}}^{(i)} \in \mathbb{R}^d$) represents the i^{th} pair of single input example and $y^{(i)} \in \{+1, -1\}$ is the output label for the i^{th} pair. For this problem,

- If the i^{th} pair of input $(\mathbf{x}^{(i)}, \hat{\mathbf{x}}^{(i)})$ is composed of signature images both of which are genuine, then the label for the i^{th} example is +1 ($y^{(i)} = +1$).
- If the i^{th} pair of input $(\mathbf{x}^{(i)}, \hat{\mathbf{x}}^{(i)})$ is composed of signature images both of which are forged, then the label for the i^{th} example is -1 ($y^{(i)} = -1$).
- If the i^{th} pair of input $(\mathbf{x}^{(i)}, \hat{\mathbf{x}}^{(i)})$ is composed of signature images one of which is genuine and the other is forged, then the label for the i^{th} example is -1 ($y^{(i)} = -1$).

The architecture of the similarity network is given below:

$$\begin{aligned}
\mathbf{h}_1 &= \texttt{ReLU}(\mathbf{W}_1\mathbf{x}) \\
\hat{\mathbf{h}}_1 &= \texttt{ReLU}(\mathbf{W}_1\hat{\mathbf{x}}) \\
\mathbf{z} &= \mathbf{W}_2\mathbf{h}_1 \\
\hat{\mathbf{z}} &= \mathbf{W}_2\hat{\mathbf{h}}_1 \\
s &= \cos\langle \mathbf{z}, \hat{\mathbf{z}} \rangle = \frac{\mathbf{z}^T\hat{\mathbf{z}}}{\|\mathbf{z}\|_2\|\hat{\mathbf{z}}\|_2} \\
\mathcal{L} &= -y \cdot s
\end{aligned}$$

(a) (30 points) Having defined the architecture of the similarity network, you are now ready to learn the parameters of the network using stochastic gradient descent. The main ingredient of the gradient descent algorithms are the gradients. In the following parts, we will be walking you through the gradient computation process. To aid the gradient computations, we have drawn out the computational graph for you below. You may directly use any results derived in class.

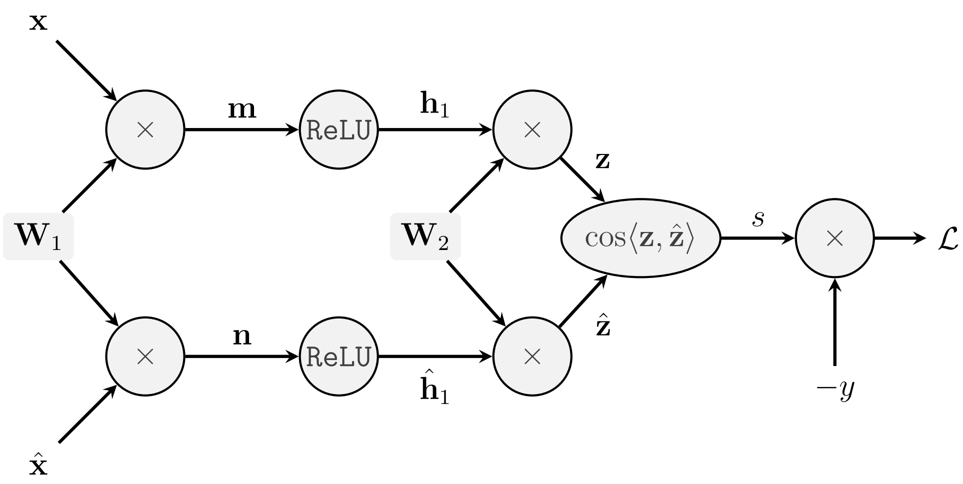


Figure 1: Computational graph of the similarity network

i. (10 points) Compute $\nabla_{\mathbf{z}}\mathcal{L}$ and $\nabla_{\hat{\mathbf{z}}}\mathcal{L}$ and denote them as $\delta_{\mathbf{z}}$ and $\delta_{\hat{\mathbf{z}}}$ respectively. For all the following parts, you can use $\delta_{\mathbf{z}}$ and $\delta_{\hat{\mathbf{z}}}$ to refer to $\nabla_{\mathbf{z}}\mathcal{L}$ and $\nabla_{\hat{\mathbf{z}}}\mathcal{L}$ respectively.

Hint: Recall the derivative quotient rule for scalars:

$$\frac{d}{dz}\left(\frac{f(z)}{g(z)}\right) = \frac{f'(z)g(z) - g'(z)f(z)}{g(z)^2}$$

for $f'(z) = \frac{df(z)}{dz}$ and $g'(z) = \frac{dg(z)}{dz}$.

$$\frac{\partial \mathcal{L}}{\partial s} = -y$$

$$\nabla_z \mathcal{L} = -y \frac{\partial s}{\partial z} \qquad \frac{\partial s}{\partial z} = \left[\frac{\|z\|_2 \|\hat{z}\|_2 \hat{z} - z^T \hat{z} \left(\frac{\|\hat{z}\|_2}{\|z\|_2} \right) z}{\|z\|_2^2 \, \|\hat{z}\|_2^2} \right]$$

$$\nabla_z \mathcal{L} = -y \left[\frac{\|z\|_2 \|\hat{z}\|_2 \hat{z} - z^T \hat{z} \left(\frac{\|\hat{z}\|_2}{\|z\|_2} \right) z}{\|z\|_2^2 \, \|\hat{z}\|_2^2} \right]$$

similarly

$$\nabla_{\hat{z}} \mathcal{L} = -y \left[\frac{\|z\|_2 \|\hat{z}\|_2 z - z^T \hat{z} \left(\frac{\|z\|_2}{\|\hat{z}\|_2} \right) \hat{z}}{\|z\|_2^2 \, \|\hat{z}\|_2^2} \right]$$

ii. (5 points) Compute $\nabla_{\mathbf{W}_2}\mathcal{L}$. For all the following parts, you can use $\delta_{\mathbf{W}_2}$ to refer to $\nabla_{\mathbf{W}_2}\mathcal{L}$.

$$\nabla_{w_2}\mathcal{L} = \frac{\partial z}{\partial w_2}\delta_z + \frac{\partial \hat{z}}{\partial w_2}\partial_{\hat{z}}$$

$$\searrow \partial_z h_1^T \qquad \searrow \partial_{\hat{z}} \hat{h}_1^T$$

$$\nabla_{w_2}\mathcal{L} = \delta_z h_1^T + \delta_{\hat{z}} \hat{h}_1^T$$

iii. (5 points) Compute $\nabla_{\mathbf{h}_1}\mathcal{L}$ and $\nabla_{\hat{\mathbf{h}}_1}\mathcal{L}$. For all the following parts, you can use $\delta_{\mathbf{h}_1}$ and $\delta_{\hat{\mathbf{h}}_1}$ to refer to $\nabla_{\mathbf{h}_1}\mathcal{L}$ and $\nabla_{\hat{\mathbf{h}}_1}\mathcal{L}$ respectively.

$$\nabla_{h_1}\mathcal{L} = \frac{\partial z}{\partial h_1}\delta_z = W_2^T \delta_z$$

$$\nabla_{\hat{h}_1}\mathcal{L} = \frac{\partial z}{\partial \hat{h}_1}\partial_{\hat{z}} = W_2^T \delta_{\hat{z}}$$

$$\nabla_{h_1}\mathcal{L} = W_2^T \delta_z$$

$$\nabla_{\hat{h}_1}\mathcal{L} = W_2^T \delta_{\hat{z}}$$

iv. (5 points) Compute $\nabla_{\mathbf{m}}\mathcal{L}$ and $\nabla_{\mathbf{n}}\mathcal{L}$. For all the following parts, you can use $\delta_{\mathbf{m}}$ and $\delta_{\mathbf{n}}$ to refer to $\nabla_{\mathbf{m}}\mathcal{L}$ and $\nabla_{\mathbf{n}}\mathcal{L}$ respectively. Use the symbol $\odot$ to denote elementwise multiplication (Hadamard product).

$$\nabla_m \mathcal{L} = \frac{\partial h_1}{\partial m} \delta h_1 = \mathbb{1}(m>0) \odot \delta h_1$$

$$\nabla_n \mathcal{L} = \frac{\partial \hat{h}_1}{\partial n} \delta \hat{h}_1 = \mathbb{1}(n \geq 0) \ominus \delta \hat{h}_1$$

$$\nabla_m \mathcal{L} = \mathbb{1}(m>0) \odot \delta h_1$$

$$\nabla_n \mathcal{L} = \mathbb{1}(n \geq 0) \odot \delta \hat{h}_1$$

v. (5 points) Compute $\nabla_{\mathbf{W}_1}\mathcal{L}$.

$$\nabla_{w_1} \mathcal{L} = \frac{\partial m}{\partial w_1} \cdot \delta m + \frac{\partial n}{\partial w_1} \delta n$$

$$\nabla_{w_1} \mathcal{L} = \delta_m x^T + \delta_n \hat{x}^T$$

(b) (9 points) In the similarity network architecture, $\mathbf{z}$ and $\hat{\mathbf{z}}$ represents the embedding vectors for input signature images $\mathbf{x}$ and $\hat{\mathbf{x}}$ respectively. Suppose we are given a training sample, $\{(\mathbf{x}^{(g)}, \hat{\mathbf{x}}^{(g)}), +1\}$.

i. (3 points) Compute the loss for the training sample if $\mathbf{z}^{(g)} = \hat{\mathbf{z}}^{(g)}$.

$$\mathcal{L} = -\frac{z^{g\,T}\hat{z}^{g}}{\|z^{g}\|_2\|\hat{z}^{g}\|_2} = -1$$

cosine similarity for same vectors angle is zero $\cos(0) = 1$

ii. (3 points) Compute the loss for the training sample if $\mathbf{z}^{(g)}$ and $\hat{\mathbf{z}}^{(g)}$ are orthogonal to each other

$$\mathcal{L} = -1\,(0) = 0$$

$-\cos(90°) = 0$

iii. (3 points) Compute the loss for the training sample if $\mathbf{z}^{(g)} = -\hat{\mathbf{z}}^{(g)}$.

$$\mathcal{L} = -1\,(-1) = 1$$

$\cos(180°) = -1$

(c) (1 points) Based on your answer to part (b), explain if the loss function is forcing the embedding vectors in the right direction.

It is. Based on the above calculation, we see that if the embeddings are similar we will get the highest value from the cosine similarity, which corresponds to minimizing the loss. Since we have two genuine signatures in this training sample, we should get similar embeddings. Note that if one or both were forgeries, the y sign flips and we would be trying to increase the angle (decrease similarity) between the two embeddings.

3. **Training neural networks** (25 points)

(a) (4 points) Which of the following activation functions where vanishing gradients usually happen? **Select all that apply.** (Note: Justification is not necessary, but may result in partial credit if the answer is incorrect.)

A. ReLU
[✓ B. Tanh]
[✓ C. Sigmoid]
D. Leaky ReLU
E. Identity

These two functions saturate at high and low values (near zero gradients).

(b) (5 points) What is **true** about batch normalization? **Select all that apply.** (Note: Justification is not necessary, but may result in partial credit if the answer is incorrect.)

A. Batch normalization slows down the training process by requiring more iterations.
B. Batch normalization is a non-learnable transformation.
[✓ C. Batch normalization is a non-linear transformation to make the output of each layer have unit statistics.]
D. Batch normalization introduces noise to a hidden layer's activation.
E. Batch normalization is not applicable at test time.

(c) (5 points) Which of the following are **true** about regularization? **Select all that apply.** (Note: Justification is not necessary, but may result in partial credit if the answer is incorrect.)

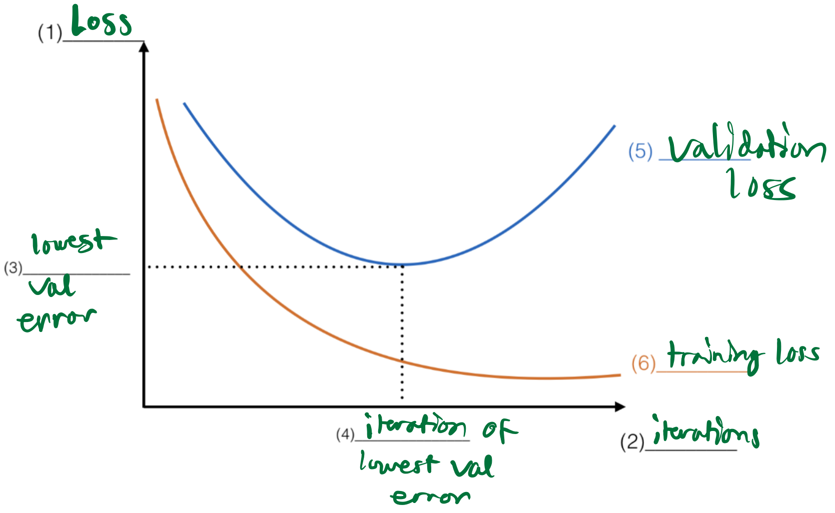
[✓ A. L1 regularization often results in some weights being 0.]
B. Adding a regularization penalty will always reduce the training loss.
[✓ C. Dropout acts as regularization.]
[✓ D. Unsuccessful regularization attempts (such as having too large a weight on a parameter norm penalty) could lead to model underfitting.]
E. None of the above

(d) (5 points) Which of the following are **true**? **Select all that apply.** (Note: Justification is not necessary, but may result in partial credit if the answer is incorrect.)

[✓ A.] In transfer learning, we can freeze most parameters of the original network.
[✓ B.] Data augmentation could help address the class imbalance problem (having different number of examples for each class) for image classification.
C. Multitask learning is not applicable if you have a small amount of data for a particular task.
[✓ D.] Ensemble methods are an effective way to improve performance.
E. None of the above.

C was not selected since it really depends more on the model complexity and it might actually be beneficial if you can share features with tasks that have more data.

(e) (6 points) Early stopping is a popular regularization method that constantly evaluates the training and validation loss on each training iteration, and returns the model with the lowest validation error. Now, you are going to draw an illustration of early stopping and introduce the concept of it to your friend. Fill in the blanks in the figure with precise answers.



Hint:
(1) and (2) describe the axis legends.
(3) and (4) describe specific values on the vertical and horizontal axes.
(5) and (6) describe the names of curves.

4. **Gradient-based optimization algorithms** (15 points)

We have learned several optimization algorithms. Given a loss function $\mathcal{L}(\theta)$, the algorithms make use of the gradient information $\mathbf{g} = \nabla_\theta \mathcal{L}$ to iteratively update the parameters θ. The update rule, however, varies for different algorithms.

Let $\mathbf{g}_t := \nabla_\theta \mathcal{L}(\theta_{t-1})$ be the gradient at θ_{t-1}. This question will discuss the following update rules from class, reproduced here for convenience:

Gradient Descent At the t^{th} iteration,

$$\theta_t \leftarrow \theta_{t-1} - \varepsilon \mathbf{g}_t,$$

where ε is the step size hyperparameter.

Gradient Descent with Momentum At the t^{th} iteration,

$$\begin{aligned} \mathbf{v}_t &\leftarrow \alpha \mathbf{v}_{t-1} - \varepsilon \mathbf{g}_t \\ \theta_t &\leftarrow \theta_{t-1} + \mathbf{v}_t \end{aligned}$$

where ε is the step size hyperparameter, and $\alpha \in [0, 1]$ is the running average parameter for momentum.

AdaGrad At the t^{th} iteration,

$$\begin{aligned} \mathbf{a}_t &\leftarrow \mathbf{a}_{t-1} + \mathbf{g}_t \odot \mathbf{g}_t \\ \theta_t &\leftarrow \theta_{t-1} - \frac{\varepsilon}{\sqrt{\mathbf{a}_t} + \nu} \odot \mathbf{g}_t, \end{aligned}$$

where ν is a small value to prevent zero-division and ε is the step size hyperparameter.

Adam At the t^{th} iteration,

$$\begin{aligned} \mathbf{v}_t &\leftarrow \beta_1 \mathbf{v}_{t-1} + (1 - \beta_1)\mathbf{g}_t \\ \mathbf{a}_t &\leftarrow \beta_2 \mathbf{a}_{t-1} + (1 - \beta_2)\mathbf{g}_t \odot \mathbf{g}_t \\ \tilde{\mathbf{v}}_t &= \frac{1}{1 - \beta_1^t} \mathbf{v}_t \qquad \text{(bias correction for first moment)} \\ \tilde{\mathbf{a}}_t &= \frac{1}{1 - \beta_2^t} \mathbf{a}_t \qquad \text{(bias correction for second moment)} \\ \theta_t &\leftarrow \theta_{t-1} - \frac{\varepsilon}{\sqrt{\tilde{\mathbf{a}}_t} + \nu} \odot \tilde{\mathbf{v}}_t, \end{aligned}$$

where ν is a small value to prevent zero-division, β_1 and β_2 are the running average parameter for the first and second moment estimation. ε is the step size hyperparameter.

(a) (10 points) **Getting out of a "trap"**. Figure 2 is the landscape of a loss function with an 1-D parameter $\theta \in \mathbb{R}$. As the plot shows, there is a "plateau" between $\theta = 3$ and $\theta = 6$.

In the plot, the arrows show 6 vanilla gradient descent steps (with a fixed step size ε) before reaching the red dot near a local minimum. Note that the 6th step is so small that

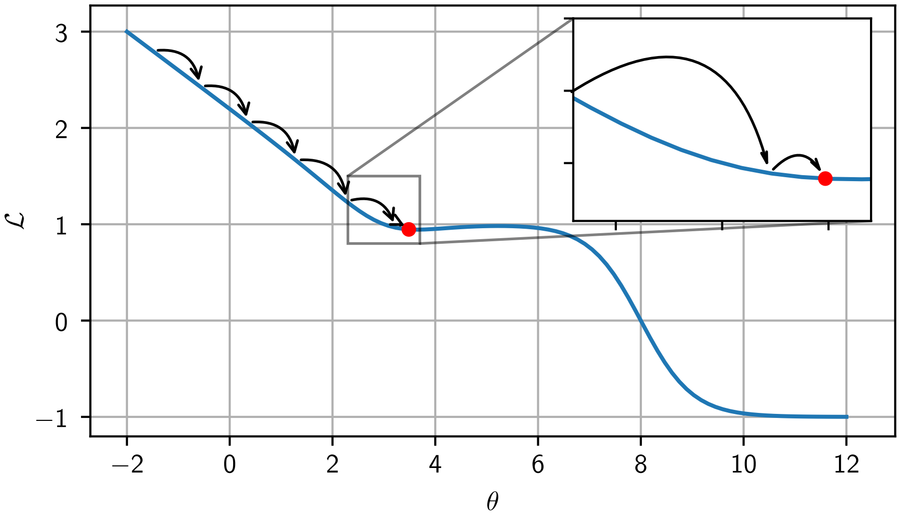


Figure 2: Loss landscape of $\mathcal{L}(\theta)$, and a gradient descent trajectory on it.

the details can only be shown in the zoom-in inset. This demonstrates that the plateau acts as a "trap" for gradient descent, where the gradient almost vanishes, leading to marginal update magnitude.

Now consider the optimization algorithms mentioned above. Assume they all share the same ε and starting point as that are used for the plotted gradient descent steps, and that *Adam* and *AdaGrad* share the same ν.

i. (5 points) Which optimization algorithms would have a better chance to get out of the trap compared to *Gradient Descent*? Briefly explain your reasons.

First we need some form of momentum, (GD+p* or Adam), since we need to continue past the plateau even when the local gradient is zero. Between the two, GD+momentum is more likely to escape since it is not annealing the learning rate/step size down, and thus takes bigger steps along the plateau.

* p = momentum

ii. (5 points) After several updates from the same starting point, when the optimizers "just step into the plateau", please order the "update magnitude" given by *Gradient Descent with Momentum*, *AdaGrad*, and *Adam*. Briefly explain your reasons.

Here "update magnitude" refers to the norm of the update step, for example, at the t^{th} step, "update magnitude" is $\|\theta_t - \theta_{t-1}\|_2$.

From largest to smallest

2. GD + Momentum
3. Adam
4. Adagrad

GD+p has the momentum term (running avg) of past vectors with no gradient norm annealing. Adam has the same with annealing, so it will be smaller than GD+p, but larger than no momentum. Adagrad only has annealing, thus it will always shrink the step size and lacks the momentum term. Note momentum term would help increase step size here (to the right) since the past vectors point to the right.

(b) (5 points) Notice that the Adam algorithm designs the “bias correction” steps for the first and second moment estimation of the gradients. In this question, we are going to derive the correction factors.

We will treat the gradients along the optimization trajectory as random variables, and assume that $\mathbf{g}_1, \mathbf{g}_2, \ldots$, are i.i.d. with some distribution that has the first and second moment. That is, we assume

$$\mathbb{E}[\mathbf{g}_t] = \boldsymbol{\mu}, \quad t = 1, 2, \ldots$$
$$\mathbb{E}[\mathbf{g}_t^2] = \mathbf{s}, \quad t = 1, 2, \ldots$$

where for simplicity, we denote $\mathbf{g}_t \odot \mathbf{g}_t$ as $\mathbf{g}_t^2$.

We first expand the recursive relation and express $\mathbf{v}_t$ in terms of $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_t$.

This gives

$$\begin{aligned}\mathbf{v}_t &= (1-\beta_1)\mathbf{g}_t + \beta_1(1-\beta_1)\mathbf{g}_{t-1} + \beta_1^2(1-\beta_1)\mathbf{g}_{t-2} + \ldots + \beta_1^{t-1}(1-\beta_1)\mathbf{g}_1 \\ &= (1-\beta_1)\sum_{i=1}^{t} \beta_1^{t-i}\mathbf{g}_i \qquad (1)\end{aligned}$$

and similarly,

$$\mathbf{a}_t = (1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i}\mathbf{g}_i^2 \qquad (2)$$

Then consider the expectation of $\mathbf{v}_t$, $\mathbb{E}[\mathbf{v}_t]$, and compare with $\boldsymbol{\mu}$.

Show that the correction factor $\gamma_1 = \dfrac{1}{1-\beta_1^t}$ satisfies

$$\gamma_1 \mathbb{E}[\mathbf{v}_t] = \boldsymbol{\mu} = \mathbb{E}[\mathbf{g}_t].$$

You will see that $\gamma_2 = \dfrac{1}{1-\beta_2^t}$ corrects $\mathbb{E}[\mathbf{a}_t]$ to $\mathbf{s}$ (i.e. $\mathbb{E}[\mathbf{g}_t^2]$) in a similar way.

Hint: The sum of a geometric series $p^0, p^1, \ldots, p^{n-1}$ is given by:

$$\sum_{j=1}^{n-1} p^j = \frac{1-p^n}{1-p}$$

(Space for question 4b.)

$$E[v_t] = E\left[(1-\beta_1)\sum_{i=1}^{t} \beta_1^{t-i} g_i\right]$$

$$= (1-\beta_1)\underbrace{\sum_{i=1}^{t} \beta_1^{t-i}} E[g_i] \rightarrow \mu$$

Linearity of E

- iid $E[g_t]=\mu \;\forall\, t$

Note: $\sum_{i=1}^{t} \beta_1^{t-i} = \sum_{j=0}^{t-1} \beta^j \, : \; \beta^{t-1} + \beta^{t-2} \ldots \beta^0 = \beta^0 + \beta^1 \ldots + \beta^{t-1}$

$$\Rightarrow (1-\beta_1)\sum_{j=0}^{t-1} \beta^j \mu$$

geometric series sum provided

$$= \cancel{(1-\beta_1)} \frac{(1-\beta_1^t)}{\cancel{(1-\beta_1)}} \mu$$

$$= (1-\beta_1^t)\,\mu$$

Thus $\gamma_1 E[v_t] = \frac{1}{\cancel{1-\beta_1^t}} \cdot \cancel{(1-\beta_1^t)}\,\mu = \mu$ ✓

$\gamma_2 E[a_t]$ is similar except $E[g^2] = s$:

$$\gamma_2 E[a_t] = \frac{1}{1-\beta_2^t}(1-\beta_2^t)\, s = s$$

5. **Bonus** (5 points) **Nesterov Momentum.**

Recall that in class, we discussed the Nesterov momentum update. For parameters θ, Nesterov momentum performs:

$$\begin{aligned} \mathbf{v} &\leftarrow \alpha \mathbf{v} - \epsilon \nabla_\theta \mathcal{L}(\theta + \alpha \mathbf{v}) \\ \theta &\leftarrow \theta + \mathbf{v} \end{aligned}$$

In class, we showed the result that by defining $\tilde{\theta}_{\text{old}} = \theta_{\text{old}} + \alpha \mathbf{v}_{\text{old}}$, the update becomes:

$$\begin{aligned} \mathbf{v}_{\text{new}} &= \alpha \mathbf{v}_{\text{old}} - \epsilon \nabla_\theta \mathcal{L}(\tilde{\theta}_{\text{old}}) \\ \tilde{\theta}_{\text{new}} &= \tilde{\theta}_{\text{old}} + \mathbf{v}_{\text{new}} + \alpha(\mathbf{v}_{\text{new}} - \mathbf{v}_{\text{old}}) \end{aligned}$$

followed by setting $\mathbf{v}_{\text{old}} = \mathbf{v}_{\text{new}}$ and $\tilde{\theta}_{\text{old}} = \tilde{\theta}_{\text{new}}$. Show that these two update rules are equivalent.

Initially, Vold is Vnew from the past iteration (Before v update)

$$V_{new} = \alpha V_{old} - \varepsilon \nabla_\theta \mathcal{L}(\theta_{old} + \alpha V_{old})$$

Since Vnew = Vold before

$$\Rightarrow \underline{V = \alpha V - \varepsilon \nabla_\theta \mathcal{L}(\theta + \alpha V)}$$

Some as old update

sub for Vnew

$$\tilde{\theta}_{new} = \tilde{\theta}_{old} + \cancel{\alpha V_{old}} - \varepsilon \nabla_\theta \mathcal{L}(\tilde{\theta}_{old}) + \alpha V_{new} - \cancel{\alpha V_{old}}$$

$$\tilde{\theta}_{new} = \tilde{\theta}_{old} + \overbrace{\alpha V_{new} - \varepsilon \nabla_\theta \mathcal{L}(\tilde{\theta}_{old})} = V$$

$$\Rightarrow \underline{\theta = \theta + V}$$

We are basically passing out the the gradient with the momentum each step for the gradient calc, and then subtracting it back out before updating the gradient. It is like a half step offset in the standard momentum process.