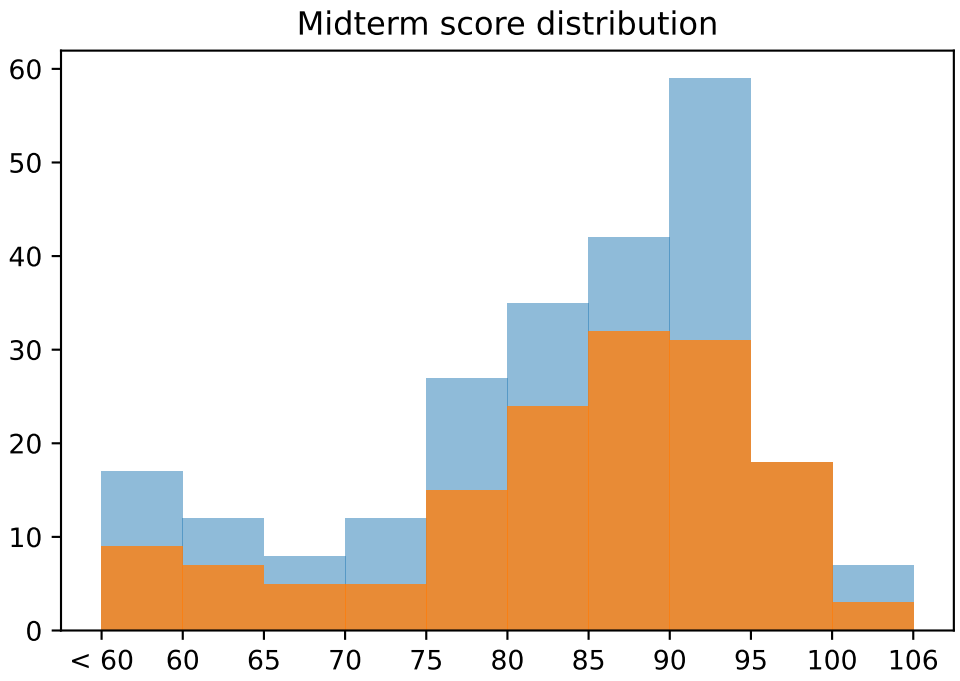
ECE C147/C247, Winter 2023 **Midterm statistics**
Neural Networks and Deep Learning Prof. J.C. Kao
UCLA ECE TAs: T.M, P.L, R.G, K.K, N.V, S.R, S.P, M.E



C147 statistics (orange)
High score: 103
Mean score: 83.6
Median score: 86
Standard deviation: 12.2

C247 statistics (blue)
High score: 104.5
Mean score: 82.2
Median score: 85.5
Standard deviation: 13.8

2-sample KS p-value (C147 vs C247): $p = 0.55$.

Comments:

- We are happy with the exam results; the class collectively did solid.
- Recall the class is not curved; it is graded on an absolute scale. Based on these results, the absolute scales for this class are unlikely to be relaxed.

- If you have grading questions, please submit through Gradescope. Regrades should be submitted if you believe we applied the rubric mistakenly. Inquiries into particular questions should be directed to the person who graded that question. Regrades will close one week from today.
- The questions were graded by the following TAs:
  - 1a: Naveen
  - 1b: Mahmoud
  - 1c: Sindhu
  - 1d: Shivam
  - 2: Tonmoy
  - 3: Kalai
  - 4: Rakshith
  - Bonus: Pan

ECE C147/C247, Winter 2023 **Midterm Solution**
Neural Networks & Deep Learning Prof. J.C. Kao
UCLA ECE TAs: T.M, P.L, R.G, K.K, N.V, S.R, S.P, M.E

UCLA True Bruin academic integrity principles apply.
Open: Four pages of cheat sheet allowed, Calculator.
Closed: Book, computer, Internet.
2:00pm-3:50pm PDT.
Wednesday, 22 Feb 2023.

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

Name: _____

Signature: _____

ID#: _____

Problem 1	_____ / 20
Problem 2	_____ / 30
Problem 3	_____ / 25
Problem 4	_____ / 25
BONUS	_____ / 7 bonus points
Total	_____ / 107 points

1. **Multiple choice and short answer** (20 points). For multiple choice, you do not need to justify your answers. If you provide justifications, they may be used to award partial credit if applicable.

   (a) (5 points) Consider a Fully connected (FC) network with the $\tanh(\cdot)$ activation for classification on the CIFAR-10 dataset. The network has no biases, only weights. Your goal is to initialize the weights. Please select **all true statements** (multiple may be true).

   (A) Initializing all weights to zero results in zero gradients and therefore no learning.

   (B) Initializing all weights to very large random values will result in vanishing gradients.

   (C) Initializing all weights to the same exact value (say every weight is equal to 2) will cause all weights to always be equal, even after several gradient descent steps.

   Solution: A, B.

   (b) (5 points) A trained neural network achieves high training accuracy but poor validation accuracy on CIFAR-10 dataset. Which of the following are reasonable approaches to decrease the gap between the training and validation accuracy? **Please select all true statements** (multiple may be true).

   (A) Increase the number of layers in the neural network.

   (B) Perform dataset augmentation on input images (including crops, color augmentations, and reflections).

   (C) Use dropout.

   Solution: B, C

   (c) (5 points) Your friend just finished a deep learning course. He comes to you with an idea he believes is brilliant: He thought of a new activation function that he wants to use in neural network training:

$$f(z) = \begin{cases} 1, z \geq 0 \\ -1, z < 0 \end{cases}$$

   Would you recommend he use this activation function? Justify your answer in no more than two sentences.

   Solution: $f(z)$ has zero gradients for all inputs (except at 0 where it's non-differentiable) which means there won't be learning.

   (d) (5 points) Consider a Convolutional neural network (CNN) with N conv-pool layers. Each conv-pool layer uses five filters of size 3×3, stride $= 1$ and padding $= 1$, followed by relu(), followed by 2×2 max pooling with stride 2. The input image is of size $256 \times 256 \times 3$. What is the number of trainable parameters (including bias) in the first conv-pool layer?

Solution: The number of parameters in the first layer are:

$$\begin{aligned} \text{Parameters} &= (3 \cdot 3 \cdot 3 + 1) \cdot 5 \\ &= 140 \end{aligned}$$

Max-pool layer does not have any parameters.

2. **Binary classification on unbalanced data: backpropagation with weighted cross-entropy loss** (30 points)

Bronchitis is an inflammation of the lining of your bronchial tubes, which carry air to and from your lungs. The symptoms of bronchitis are wheezing and coughing. If you develop these symptoms, you visit a pulmonologist and they order a chest X-ray to diagnose the disease. However, analyzing the X-ray manually takes time and is not scalable. Therefore, doctors at UCLA have contacted you to build a machine learning model for predicting whether the patient has bronchitis or not given the chest X-ray. Since bronchitis is not a common disease, the dataset they have provided you is highly imbalanced:

- 10000 chest X-rays from patients without bronchitis
- 1000 chest X-rays from patients with bronchitis

(a) (2 points) Name two data augmentation techniques that can help you to alleviate the class imbalance problem.

Solution: Some data augmentation techniques that can help you to alleviate the class imbalance problem are:

- Translation
- Cropping
- Reflection
- Gaussian Blurring

(b) (21 points) Instead of data augmentation, you want to design a neural network that can learn well even from the imbalanced dataset. You have come up with the following architecture:

$$
\begin{aligned}
\mathbf{z}_1 &= \mathbf{W}_1\mathbf{x}^{(i)} + \mathbf{b}_1 \\
\mathbf{a}_1 &= ReLU(\mathbf{z}_1) \\
z_2 &= \mathbf{W}_2\mathbf{a}_1 + b_2 \\
\hat{y}^{(i)} &= \sigma(z_2) \\
\mathcal{L}^{(i)} &= \alpha \cdot y^{(i)} \cdot \log(\hat{y}^{(i)}) + \beta \cdot (1 - y^{(i)}) \cdot \log(1 - \hat{y}^{(i)}) \\
J &= -\frac{1}{m}\sum_{i=1}^{m} \mathcal{L}^{(i)}
\end{aligned}
$$

where $\hat{y}^{(i)} \in \mathbb{R}$, $y^{(i)} \in \mathbb{R}$, $\mathbf{x}^{(i)} \in \mathbb{R}^{D_x \times 1}$, $\mathbf{W}_1 \in \mathbb{R}^{D_{a_1} \times D_x}$, $\mathbf{W}_2 \in \mathbb{R}^{1 \times D_{a_1}}$, and $\sigma(\cdot)$ is the sigmoid activation function. Note that m is the size of the dataset. Also note that the chest X-ray are flattened into vectors of length D_x before being fed into the neural network. If the i^{th} chest X-ray belongs to a patient with bronchitis, then $y^{(i)} = 1$. If the i^{th} chest X-ray belongs to a patient without bronchitis, then $y^{(i)} = 0$.

i. (2 points) What are the dimensions of $\mathbf{b}_1$ and b_2?

Solution: $\mathbf{b}_1 \in \mathbb{R}^{D_{a_1}}, b_2 \in \mathbb{R}$

ii. (2 points) Why are the hyperparameters α and β useful? Answer in no more than 3 sentences.

Solution: Weighting how much each class contributes to the loss function can help gradient descent because the network will take larger steps when learning from instances of the underrepresented class.

iii. (2 points) What values of α and β should you pick such that the samples from each class contribute equally in the training process?

Solution: Since the ratio of the class samples is 10, so we can pick $\alpha = 10, \beta = 1$.

iv. (3 points) Compute $\nabla_{\hat{y}^{(i)}} \mathcal{L}^{(i)}$ and denote it as $\delta_{\hat{y}^{(i)}}$. For all the following parts, you can refer to this computed gradient as $\delta_{\hat{y}^{(i)}}$.

Solution: Using linearity and chain rule of differentiation,

$$\nabla_{\hat{y}^{(i)}} \mathcal{L}^{(i)} = \alpha \frac{y^{(i)}}{\hat{y}^{(i)}} - \beta \frac{1 - y^{(i)}}{1 - \hat{y}^{(i)}} = \delta_{\hat{y}^{(i)}}$$

v. (3 points) Compute $\nabla_{b_2} \mathcal{L}^{(i)}$ and denote it as δ_{b_2}. For all the following parts, you can refer to this computed gradient as δ_{b_2}.

Hint: The derivative of $\sigma(z)$ with respect to z is: $\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))$

Solution: The local derivative at the sigmoid operator is given by,

$$\nabla_{z_2} \hat{y}^{(i)} = \sigma(z_2)[1 - \sigma(z_2)].$$

Then backpropagating to z_2 we get

$$\nabla_{z_2} \mathcal{L}^{(i)} = \sigma(z_2)[1 - \sigma(z_2)] \nabla_{\hat{y}^{(i)}} \mathcal{L}^{(i)}.$$

Since plus operator passes the gradient, so

$$\nabla_{b_2} \mathcal{L}^{(i)} = \sigma(z_2)[1 - \sigma(z_2)] \nabla_{\hat{y}^{(i)}} \mathcal{L}^{(i)}$$
$$\delta_{b_2} = \sigma(z_2)[1 - \sigma(z_2)] \delta_{\hat{y}^{(i)}}$$

vi. (3 points) Compute $\nabla_{\mathbf{W}_2} \mathcal{L}^{(i)}$ and denote it as $\delta_{\mathbf{W}_2}$. For all the following parts, you can refer to this computed gradient as $\delta_{\mathbf{W}_2}$.

Solution: Since multiplication operator switches the gradient, so

$$\nabla_{\mathbf{W}_2} \mathcal{L}^{(i)} = \mathbf{a}_1^T \delta_{b_2} = \delta_{\mathbf{W}_2}$$

vii. (3 points) Compute $\nabla_{\mathbf{b}_1} \mathcal{L}^{(i)}$ and denote it as $\delta_{\mathbf{b}_1}$. For all the following parts, you can refer to this computed gradient as $\delta_{\mathbf{b}_1}$.

Solution: Backpropagating to $\mathbf{a}_1$ we get

$$\begin{aligned}\nabla_{\mathbf{a}_1}\mathcal{L}^{(i)} &= \mathbf{W}_2^T\nabla_{b_2}\mathcal{L}^{(i)} \\ &= \mathbf{W}_2^T\delta_{b_2}\end{aligned}$$

Now, backpropagating to $\mathbf{z}_1$ we get

$$\nabla_{\mathbf{z}_1}\mathcal{L}^{(i)} = \mathbb{I}(\mathbf{z}_1 > 0) \odot \nabla_{\mathbf{a}_1}\mathcal{L}^{(i)}.$$

Since plus operator passes the gradient, so

$$\nabla_{\mathbf{b}_1}\mathcal{L}^{(i)} = \nabla_{\mathbf{z}_1}\mathcal{L}^{(i)} = \delta_{\mathbf{b}_1}.$$

viii. (3 points) Compute $\nabla_{\mathbf{W}_1}\mathcal{L}^{(i)}$ and denote it as $\delta_{\mathbf{W}_1}$.

Solution: Since multiplication operator switches the gradient, so

$$\begin{aligned}\nabla_{\mathbf{W}_1}\mathcal{L}^{(i)} &= \nabla_{\mathbf{b}_1}\mathcal{L}^{(i)}{\mathbf{x}^{(i)}}^T \\ \delta_{\mathbf{W}_1} &= \delta_{\mathbf{b}_1}{\mathbf{x}^{(i)}}^T\end{aligned}$$

(c) (5 points) You learned in class that regularization increases the generalization ability of the model. Suppose you want to add L_2 regularization with strength 1 to parameter $\mathbf{W}_2$. That is, the updated loss function with L_2 regularization is:

$$\hat{J} = J + \|\mathbf{W}_2\|_2^2.$$

Assuming you are using vanilla gradient descent with learning rate ϵ, write down the update rule for weight parameter $\mathbf{W}_2$.

Solution:

$$\mathbf{W}_2^{(k+1)} = \mathbf{W}_2^{(k)} - \epsilon[\nabla_{\mathbf{W}_2}J^{(k)} + 2\mathbf{W}_2^{(k)}]$$

(d) (2 points) Suppose you used L_1 regularization instead. How would you expect the weights learned using L_1 regularization to differ from those learned using L_2 regularization?

Solution: Weights learned using L_1 regularization will be more sparse than the weights learned using L_2 regularization.

3. **Regularization** (25 points)

(a) (5 points) **Ensembling**. Why does ensembling several neural networks usually increase generalization performance? Justify your answer in no more that 4 sentences.

Solution: If neural networks make somewhat independent errors, then they are likely to make errors on different trials. We will only get a trial wrong when several neural networks make the same errors. Hence, lower correlation between the model, better accuracy of the ensemble model.

(b) (10 points) **L-∞ norm**. Your friend, Alice, is helping you to train a Fully connected (FC) network with n layers. The parameters in layer i are represented as a weight matrix $\mathbf{W}_i$. She suggests using $L-\infty$ regularization on $\mathbf{W}_i$ (also known as Chebyshev norm) as shown below:

$$\mathcal{L}_{reg}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + R(\mathbf{W})$$

where

$$R(\mathbf{W}) = \frac{\lambda}{n}\sum_{i=1}^{n} ||\mathbf{W}_i||_\infty.$$

For $\mathbf{A} \in \mathbb{R}^{p\times m}$, the $L-\infty$ norm returns the maximum row sum of $\mathbf{A}$. Mathematically, it is defined as

$$||\mathbf{A}||_\infty = \max_{1\le k\le p}\sum_{j=1}^{m}|A_{kj}|$$

Compute the gradient $\nabla_{\mathbf{W}_i}R(\mathbf{W})$. This expression is the gradient of $R(\mathbf{W})$ with respect to the weight matrix $\mathbf{W}_i$. For this question, assume the gradient of $|w|$ with respect to w is sign(w) (i.e., -1 if w is negative, $+1$ if w is positive, and 0 if w is zero).

Solution: Given $L-\infty$ expression is

$$R(\mathbf{W}) = \frac{\lambda}{n}\sum_{i=1}^{n} ||\mathbf{W}_i||_\infty$$

Based on the dimensions of input, hidden layer and output dimensions, the individual weight matrix dimensions will be known to us. Without loss of generality, let the weight matrix of the i^{th} hidden layer $\mathbf{W}_i \in \mathbb{R}^{m\times p}$ comprising of row vectors $\mathbf{w}_r^i$ and elements represented as w_{rc}^i then

$$R(\mathbf{W}) = \frac{\lambda}{n}\sum_{i=1}^{n}\left(\max_{1\le r\le m}\sum_{c=1}^{p}|w_{rc}^i|\right)$$

Let j denote the index of row with maximum row-sum. Then the above equation reduces to

$$R(\mathbf{W}) = \frac{\lambda}{n}\sum_{i=1}^{n}\sum_{c=1}^{p}|w_{jc}^i|$$

Therefore,

$$\nabla_{\mathbf{W}_i} R(\mathbf{W}) = \frac{\lambda}{n} \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \text{sign}(w^i_{j1}) & \text{sign}(w^i_{j2}) & \cdots & \text{sign}(w^i_{jp}) \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

$$\nabla_{\mathbf{W}_i} R(\mathbf{W}) = \frac{\lambda}{n} \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \text{sign}(\mathbf{w}^i_j) \\ \mathbf{0} \end{bmatrix}$$

(c) **Batch vs Layer Normalization** (10 points)

Recall that batch normalization transforms the activation of a single neuron over B examples in a batch, $\{x^{(1)}, x^{(2)}, \ldots, x^{(B)}\}$ into $\{y^{(1)}, y^{(2)}, \ldots, y^{(B)}\}$ according to the following set of equations:

$$\mu_B = \frac{1}{B} \sum_{j=1}^{B} x^{(j)}$$

$$\sigma_B^2 = \frac{1}{B} \sum_{j=1}^{B} (x^{(j)} - \mu_B)^2$$

$$\hat{x}^{(j)} = \frac{x^{(j)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, \quad \forall\, j = 1, 2, \cdots, B$$

$$y^{(j)} = \gamma \hat{x}^{(j)} + \beta, \quad \forall\, j = 1, 2, \cdots, B$$

There is another type of normalization called "layer normalization." In "layer normalization," the input is $\mathbf{x} \in \mathbb{R}^D$, the activity of D artificial neurons, and the output is $\mathbf{y} \in \mathbb{R}^D$, the layer normalized activity of the D artificial neurons, according to the following equations:

$$\mu_\ell = \frac{1}{D} \sum_{i=1}^{D} x_i$$

$$\sigma_\ell^2 = \frac{1}{D} \sum_{i=1}^{D} (x_i - \mu_l)^2$$

$$\hat{x}_i = \frac{x_i - \mu_\ell}{\sqrt{\sigma_\ell^2 + \epsilon}}, \quad \forall\, i = 1, 2, \cdots, D$$

$$y_i = \gamma \hat{x}_i + \beta, \quad \forall\, i = 1, 2, \cdots, D$$

where x_i and y_i are the i^{th} elements of $\mathbf{x}$ and $\mathbf{y}$, respectively. Therefore, μ_ℓ is the mean of all the D artificial neurons and σ_ℓ^2 is the variance across the D artificial neurons in the l^{th} layer.

i. (3 points) Why does batch normalization cause a network to be more robust to random weight initializations?

ii. (4 points) Describe the major differences between Layer and Batch normalization (no more than 4 sentences).

iii. (3 points) Name one condition when it may be preferable to use layer normalization instead of batch normalization (no more than 3 sentences).

Solution:

i. Random weight initializations can cause problems of exploding/vanishing gradients and exacerbate internal covariate shift. For an appropriate batch size, the random weight initialization does not affect Batch Normalization as strongly because the mean and variance of the output of Batch Normalization is constrained and learnt by model.

ii. In batch normalization, the normalization occurs on a per neuron per batch that is the mean and variance calculated is for all instances in a batch per channel. In Layer normalization, the mean and variance is calculated for a single instance across all channels that is across a layer.

iii. As layer normalization normalizes the activations across an entire layer, it performs better than Batch normalization when the batch size is reduced. It also performs better when modelling sequential data. Layer Normalization is extensively used in RNNs.

4. **Gradient-based optimization algorithms** (25 points)
We have learnt several optimization algorithms in class. We list them below along with their update rules for your convenience. For all the algorithms below,

$$\mathbf{g}_t = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta_{t-1}})$$

where $\mathbf{g}_t$ is the gradient at t^{th} iteration of the loss function with respect to the parameter $\boldsymbol{\theta}$ evaluated at $\boldsymbol{\theta_{t-1}}$.

Vanilla Gradient Descent At t^{th} iteration,

$$\boldsymbol{\theta_t} = \boldsymbol{\theta_{t-1}} - \varepsilon \mathbf{g}_t$$

Gradient Descent with momentum At t^{th} iteration,

$$\mathbf{v}_t = \alpha \mathbf{v}_{t-1} - \varepsilon \mathbf{g}_t$$
$$\boldsymbol{\theta_t} = \boldsymbol{\theta_{t-1}} + \mathbf{v}_t$$

Nesterov Momentum At t^{th} iteration,

$$\mathbf{v}_t = \alpha \mathbf{v}_{t-1} - \varepsilon \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta_{t-1}} + \alpha \mathbf{v}_{t-1})$$
$$\boldsymbol{\theta_t} = \boldsymbol{\theta_{t-1}} + \mathbf{v}_t$$

Adagrad At t^{th} iteration,

$$\mathbf{a}_t = \mathbf{a}_{t-1} + \mathbf{g}_t \odot \mathbf{g}_t$$
$$\boldsymbol{\theta_t} = \boldsymbol{\theta_{t-1}} - \frac{\varepsilon}{\sqrt{\mathbf{a}_t} + \nu} \odot \mathbf{g}_t$$

RMSProp At t^{th} iteration,

$$\mathbf{a}_t = \beta \mathbf{a}_{t-1} + (1-\beta) \mathbf{g}_t \odot \mathbf{g}_t$$
$$\boldsymbol{\theta_t} = \boldsymbol{\theta_{t-1}} - \frac{\varepsilon}{\sqrt{\mathbf{a}_t} + \nu} \odot \mathbf{g}_t$$

Adam with bias correction At t^{th} iteration,

$$\mathbf{v}_t = \beta_1 \mathbf{v}_{t-1} + (1-\beta_1) \mathbf{g}_t$$
$$\mathbf{a}_t = \beta_2 \mathbf{a}_{t-1} + (1-\beta_2) \mathbf{g}_t \odot \mathbf{g}_t$$
$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_1^t}$$
$$\hat{\mathbf{a}}_t = \frac{\mathbf{a}_t}{1-\beta_2^t}$$
$$\boldsymbol{\theta_t} = \boldsymbol{\theta_{t-1}} - \frac{\varepsilon}{\sqrt{\hat{\mathbf{a}}_t} + \nu} \odot \hat{\mathbf{v}}_t$$

Comparing optimization algorithms

(a) (5 points) Consider a loss surface with 1-D parameter $w \in \mathbb{R}$ as shown in Figure 1. In the plot we marked the starting point of the optimization algorithm. We made 4 vanilla gradient descent updates as marked in the figure. We also stored the running average of the momentum v_t, for these 4 vanilla gradient descent steps, using the update rule

$$v_t = \alpha v_{t-1} - \varepsilon g_t$$

We choose $\alpha = 0.9$. For the 5^{th} step, you must choose to update the weight using "Gradient Descent with Momentum" or "Nesterov Momentum." which weight update will lead to a faster convergence to the minimum which is marked with a star in Figure 1. Why? Justify your answer in no more than 5 sentences.

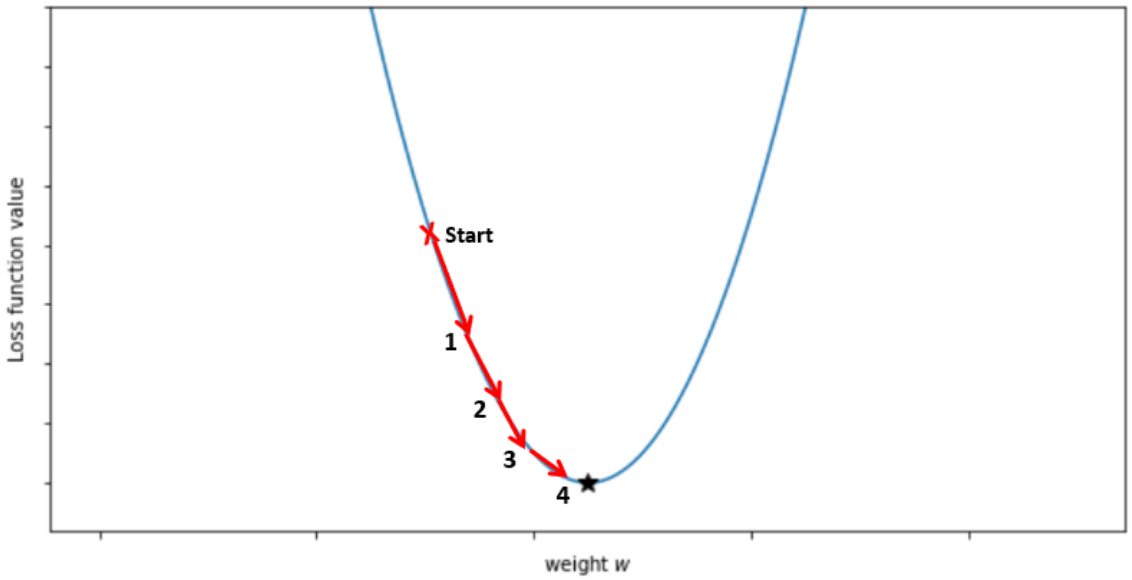


Figure 1: Figure for question 4a

Solution: It is better to choose "Nesterov Momentum" in the above scenario. The loss function has a steep slope in steps 1-4 so the accumulated momentum will be large. We know that our weight update is always opposite to the direction of gradient so in our case it would be a large positive update on weights(as slope is negative for steps 1-4), this will overshoot the minima for and reach to the right of it in case of plain "GD+ momentum".

However in case of Nesterov Momentum we look before we leap. So although the momentum weight update to the right of point 4 the slope at point $\theta_{t-1} + \alpha v_{t-1}$ is

positive which will work against the momentum and push the weight updates towards to minima, hence we converge on the minima faster.

(b) (6 points) Consider the contour plot of a loss function with 2-D weights $\mathbf{w} \in \mathbb{R}^2$. You and your friend in C147/C247 want to play a guessing game.Your friend uses 3 optimization algorithms "Gradient Descent with Momentum", "Adagrad" and "Adam" to find the minima of the loss function whose contour plot is shown in Figure 2.

The learning rate (ε) used for all the 3 algorithms is the same. Despite different weight initializations the algorithms happen to meet at a common point at some iteration t as marked in Figure 2. At iteration $t+1$, the 3 algorithms mentioned above update the weights to points $\mathbf{a}, \mathbf{b}, \mathbf{c}$ as marked in Figure 2.

You need to pair which optimization algorithm amongst "GD+momentum", "Adagrad" and "Adam" correspond to the weight updates to point $\mathbf{a}, \mathbf{b}$ and $\mathbf{c}$ respectively. Justify your answer in no more than 6 sentences.

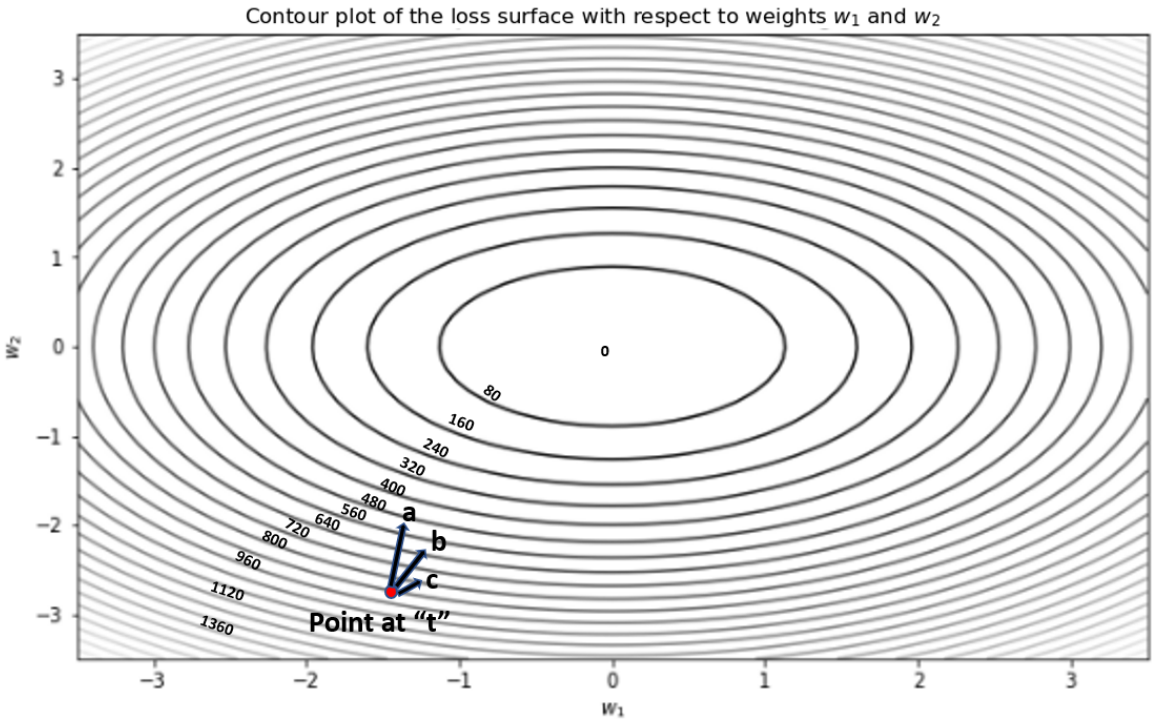


Figure 2: Figure for question 4b

Solution: We see from contour plot that at the point of interest, we we have large gradient in the **y** / $\mathbf{w}_2$ direction and small gradients in the **x** / $\mathbf{w}_1$ direction. We know that "GD with momentum" updates weights governed the direction of the higher gradient in this case it is along $\mathbf{w}_2$ so point **a** corresponds to update of GD+Momentum. The magnitude for a is also the largest.

Adagrad on the other hand anneals the learning rate along the direction where the gradient is large and the weight update is biased to the direction of smaller gradient which **x** direction in our case so point **c** corresponds to update of Adagrad. The magnitude for c is the smallest.

Adam on the other hand combines the above two so its update would not be dominantly **x** or **y** so point **b** corresponds to update of Adam. The magnitude for b is in between a and c

a → GD+momentum. **b** → Adam. **c** → Adagrad.

(c) (14 points) **Gradients as Random vectors**:
You plan to train a neural network with Mini-batch Gradient descent + Momentum. You initialize the weights by drawing samples from a multivariate Gaussian distribution with mean **0** and covariance $\eta^2\mathbf{I}$. In other words $\mathbb{E}(\boldsymbol{\theta_0}) = \mathbf{0}, \text{Cov}(\boldsymbol{\theta_0}) = \eta^2\mathbf{I}, \boldsymbol{\theta} \in \mathbb{R}^D$. We assume the gradients calculated at any arbitrary iteration k have the same expected value $\mathbb{E}(\mathbf{g}_k) = \boldsymbol{\mu}$. Assume the initial value for momentum $\mathbf{v_0} = \mathbf{0}$.

i. (6 points) Express $\boldsymbol{\theta_t}$ (the weights at iteration t) in terms of $\boldsymbol{\theta_0}$ (the initial weights), ε (the learning rate), α (momentum hyperparameter) and the gradients $\mathbf{g}_1, \mathbf{g}_2, \ldots \mathbf{g}_t$.

Solution: For GD+ momentum we have the following update rules,

$$\mathbf{v}_t = \alpha\mathbf{v}_{t-1} - \varepsilon\mathbf{g}_t$$
$$\boldsymbol{\theta_t} = \boldsymbol{\theta_{t-1}} + \mathbf{v}_t$$

The recursive equation for $\mathbf{v}_t$, $\boldsymbol{\theta_t}$ would yield,

$$\mathbf{v}_t = -\varepsilon \sum_{i=1}^{t} \alpha^{t-i}\mathbf{g}_i$$

$$\boldsymbol{\theta_t} = \boldsymbol{\theta_0} + \sum_{j=1}^{t} \mathbf{v}_j$$

$$\boldsymbol{\theta_t} = \boldsymbol{\theta_0} - \varepsilon \sum_{j=1}^{t} (\sum_{i=1}^{j} \alpha^{j-i}\mathbf{g}_i)$$

ii. (8 points) Calculate the expected value of the weights $\mathbb{E}(\boldsymbol{\theta_t})$ at any arbitrary iteration t. Your answer should be in terms of ε (the learning rate), α (momentum hyperparameter), $\boldsymbol{\mu}$(Expected value of gradients) and t only. Recall that $\mathbb{E}(\mathbf{g}_k) = \boldsymbol{\mu}$ for all k.

Hint: The sum of terms in a geometric series $a, ar, ar^2, \ldots, ar^{n-1}$ is:

$$\sum_{p=0}^{n-1}(ar^p) = \frac{a(1-r^n)}{1-r}$$

Solution:

$$\mathbb{E}(\boldsymbol{\theta_t}) = \mathbb{E}(\boldsymbol{\theta_0}) - \varepsilon\mathbb{E}(\sum_{j=1}^{t}\sum_{i=1}^{j}\alpha^{j-i}\mathbf{g}_i)$$

$$\mathbb{E}(\boldsymbol{\theta_t}) = \mathbb{E}(\boldsymbol{\theta_0}) - \varepsilon\sum_{j=1}^{t}\sum_{i=1}^{j}\alpha^{j-i}\mathbb{E}(\mathbf{g}_i)$$

Expectation is linear and ε is a constant

$$\mathbb{E}(\boldsymbol{\theta_t}) = -\varepsilon\sum_{j=1}^{t}\sum_{i=1}^{j}\alpha^{j-i}\boldsymbol{\mu}$$

Substitute for $, \mathbb{E}(\mathbf{g}_k) = \boldsymbol{\mu}$

$$\mathbb{E}(\boldsymbol{\theta_t}) = -\varepsilon\sum_{j=1}^{t}\sum_{i=1}^{j}\alpha^{j-i}\boldsymbol{\mu}$$

$$\mathbb{E}(\boldsymbol{\theta_t}) = -\varepsilon\boldsymbol{\mu}\sum_{j=1}^{t}(1+\alpha+\alpha^2\ldots\alpha^{j-1})$$

$$\mathbb{E}(\boldsymbol{\theta_t}) = -\varepsilon\boldsymbol{\mu}\sum_{j=1}^{t}\frac{1-\alpha^j}{1-\alpha}$$

$$\mathbb{E}(\boldsymbol{\theta_t}) = \frac{-\varepsilon\boldsymbol{\mu}}{1-\alpha}\sum_{j=1}^{t}(1-\alpha^j)$$

$$\mathbb{E}(\boldsymbol{\theta_t}) = \frac{-\varepsilon\boldsymbol{\mu}}{1-\alpha}\left(t - \frac{\alpha(1-\alpha^t)}{1-\alpha}\right)$$

5. **Bonus** (7 points) **Covariance of Momentum**

For this question, you may assume that $\mathbb{E}(\mathbf{g}_t) = \mathbb{E}(\mathbf{v}_t) = \boldsymbol{\mu} = \mathbf{0}$, where $\mathbf{g}_t \in \mathbb{R}^D$ is the gradient vector at iteration "t" and $\mathbf{v}_t \in \mathbb{R}^D$ is the running average of the momentum at iteration t.

Further, it is given that

$$\mathbb{E}(\mathbf{g}_i\mathbf{g}_j^T) = \begin{cases}\sigma^2\mathbf{I} & i = j\\ \mathbf{0} & i \neq j\end{cases} \tag{1}$$

Compute $\mathbb{E}(\mathbf{v}_t \mathbf{v}_t^T)$.

Solution: Again we start with recursive form of momentum update equation

$$\mathbf{v}_t = -\varepsilon \sum_{i=1}^{t} \alpha^{t-i} \mathbf{g}_i$$

$$\mathrm{Cov}(\mathbf{v}_\mathrm{t}) = \mathbb{E}(\mathbf{v}_t {\mathbf{v}_t}^T)$$

Zero mean for momentum update

$$\mathrm{Cov}(\mathbf{v_t}) = \varepsilon^2 \mathbb{E}(\sum_{i=1}^{t} \alpha^{t-i} \mathbf{g}_i [\sum_{j=1}^{t} \alpha^{t-j} \mathbf{g}_j]^T)$$

$$= \varepsilon^2 \mathbb{E}(\sum_{i=1}^{t} \sum_{j=1}^{t} \alpha^{t-i} \alpha^{t-j} \mathbf{g}_i \mathbf{g}_j^T)$$

$$= \varepsilon^2 [\sum_{i=1}^{t} \alpha^{2(t-i)} \mathbb{E}(\mathbf{g}_i \mathbf{g}_i^T)]$$

$$= \varepsilon^2 \sigma^2 \mathbf{I} \sum_{i=1}^{t} \alpha^{2(t-i)}$$

$$\mathrm{Cov}(\hat{\mathbf{v}}_t) = \varepsilon^2 \sigma^2 (\frac{1-\alpha^{2t}}{1-\alpha^2}) \mathbf{I}$$