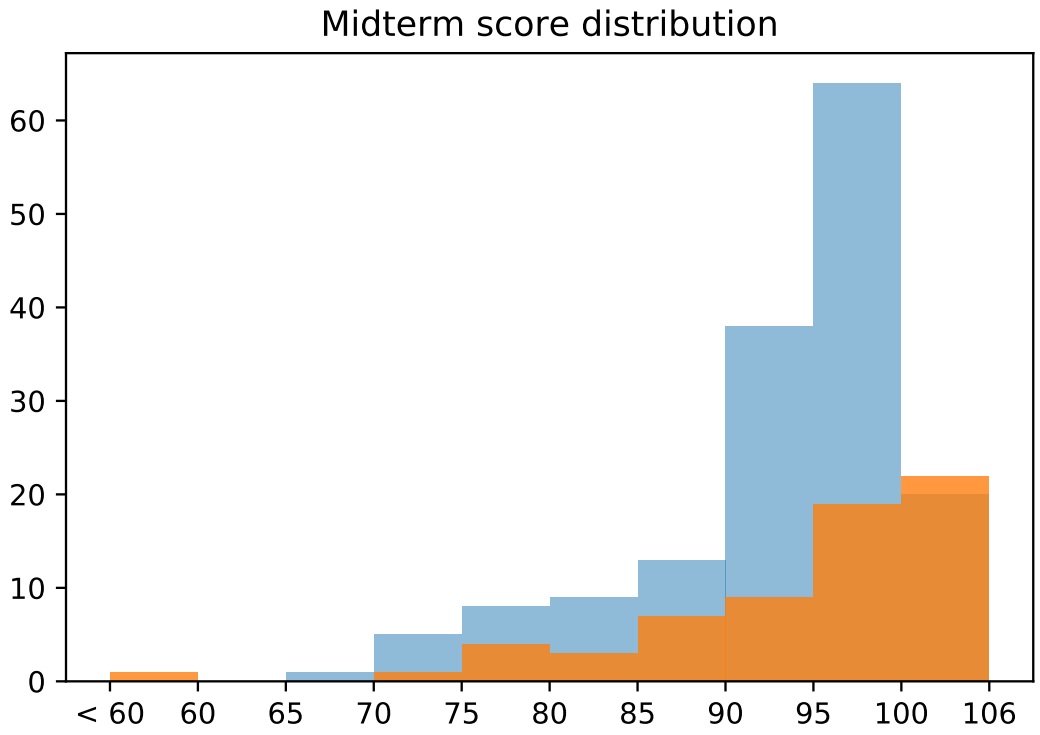
ECE C147/C247, Winter 2021 **Midterm Overview**
Department of Electrical Engineering Prof. J.C. Kao
UCLA ECE TAs N. Evirgen, A. Ghosh, S. Mathur, T. Monsoor, G. Zhao

Midterm statistics:



C147 (orange)
High score: 106
Median score: 97.25
Mean score: 94.6
Standard deviation: 9.4

C247 (blue)
High score: 103.5
Median score: 95.0
Mean score: 92.8
Standard deviation: 7.4

Comments:

- You as a class collectively scored very well on the exam. Congratulations!
- It's worth reminding everyone that the class is not curved.
- 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.

  Questions 1: Guangyuan
  Question 2: Shaan
  Question 3: Arunabh
  Question 4: Tonmoy
  Bonus: Noyan

- Regrades close one week from today (March 4, 2021).

ECE C147/C247, Winter 2021 **Midterm Solutions**

Department of Electrical and Computer Engineering
University of California, Los Angeles

Prof. J.C. Kao
TAs: N. Evirgen, A. Ghosh,
S. Mathur, T. Monsoor, G. Zhao

UCLA True Bruin academic integrity principles apply.
Open: Book, computer.
Closed: Internet, except to visit CCLE and Piazza.
12:00pm-1:50pm.
Thursday, 18 Feb 2021 (or at an approved alternate time).

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

Name: _____

Signature: _____

ID#: _____

Problem 1	_____ / 25
Problem 2	_____ / 30
Problem 3	_____ / 20
Problem 4	_____ / 25
BONUS	_____ / 6 bonus points
Total	_____ / 100 points + 6 bonus points

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

   (a) (5 points) What is the key difference between a classification and regression task? Discuss your answer in no more than 3 sentences.

   Solution: Classification is used when your target is categorical, while regression is used when your target variable is continuous. Both classification and regression belong to the category of supervised machine learning algorithms.

   Examples of classification problems include:

   - Predicting yes or no
   - Estimating gender
   - Breed of an animal
   - Type of color

   Examples of regression problems include:

   - Estimating sales and price of a product
   - Predicting the score of a team
   - Predicting the amount of rainfall

   (b) (7 points) Explain the key difference between stochastic gradient descent (SGD) and full-batch gradient descent (GD). Does SGD or GD provide a more accurate estimate of the gradient? Discuss your answer in no more than 4 sentences.

   Solution: Both algorithms are methods for finding a set of parameters that minimize a loss function by evaluating parameters against data and then making adjustments.

   In standard gradient descent, you'll evaluate all training samples for each set of parameters. This is akin to taking big, slow steps toward the solution.

   In stochastic gradient descent, you'll evaluate only 1 training sample for the set of parameters before updating them. This is akin to taking small, quick steps toward the solution.

   In terms of accuracy, batch gradient is more accurate as it utilizes full batch.

   (c) (7 points) Why is it very difficult to train neural networks with many layers (10+) initialized with small weights ($\text{var}(w_{ij}) \ll 2/(n_{\text{in}} + n_{\text{out}})$, i.e., much smaller than the Xavier initialization)? Discuss your answer in no more than 3 sentences.

   Solution: Since the weights are small, the activation magnitude decreases with each successive layer. The last layer's activations are close to zero. These leads to the backpropagated gradients being small.

   (d) (6 points) Consider a fully connected neural network initialization where all the parameters are initialized to the same constant value (instead of randomly drawn from a normal distribution). Further, consider that this constant is judiciously chosen so that activations do not explode or vanish across layers. Is this initialization a good idea? Why or why not? Discuss your answer in no more than 4 sentences.

Hint: consider the values of the activations in a single layer.

Solution: Breaking symmetry is intention for the random initilization. If we initialize all weights to the same value (e.g. one), each hidden unit will get exactly the same signal. E.g. if all weights are initialized to 1, each unit gets signal equal to sum of inputs (and outputs sigmoid(sum(inputs))). The gradient of all neurons in one layer will be the same. As a result, the training will only effectively update neuron in different layers.

2. **Backpropagation** (30 points)

In this problem we will be making a computational graph for the final layer of a neural network, and performing backpropagation on it. Let the input to our layer be $\mathbf{x} \in \mathbb{R}^n$, the bias be $\mathbf{b} \in \mathbb{R}^m$, the weight matrix be $\mathbf{W}_k \in \mathbb{R}^{m \times n}$, and the loss function be $\mathcal{L}$.

$$\begin{aligned}
\mathcal{L} &= \frac{1}{2}||\mathbf{f} - \mathbf{y}||^2 \\
\mathbf{f} &= \max(\alpha \cdot \mathbf{g}, \mathbf{g}), 0 < \alpha < 1 \\
\mathbf{g} &= \mathbf{h} + \mathbf{b} \\
\mathbf{h} &= \mathbf{W}_k \mathbf{x}
\end{aligned}$$

(a) (11 points) Draw a computational graph for this neural network layer. Please label the edges with $\mathbf{f}$, $\mathbf{g}$, and $\mathbf{h}$ the same way we have above. You don't need to create a graph for the loss.

Solution: Points given for a correct computational graph.

(b) (14 points) Determine $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_k}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{b}}$ using backpropagation. In your calculations, you may notice that $\frac{\partial \mathbf{h}}{\partial \mathbf{W}_k}$ is a 3D tensor. To avoid this complication, you may use the fact that

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_k} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}} \mathbf{x}^T.$$

Please make sure to box your final answers.

Solution: We will compute each gradient step by step. First $\frac{\partial \mathcal{L}}{\partial \mathbf{f}}$.

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial f_i} &= \frac{\partial}{\partial f_i}\left[\frac{1}{2}(\mathbf{f}-\mathbf{y})^T(\mathbf{f}-\mathbf{y})\right] \\
&= \frac{\partial}{\partial f_i}\left[\frac{1}{2}(\mathbf{f}^T\mathbf{f} - 2\mathbf{y}^T\mathbf{f} + \mathbf{y}^T\mathbf{y})\right] \\
&= \frac{1}{2}(2f_i - 2y_i) \\
&= f_i - y_i.
\end{aligned}$$

Therefore $\frac{\partial \mathcal{L}}{\partial \mathbf{f}} = \mathbf{f} - \mathbf{y}$.

By the (denominator layout) chain rule, $\frac{\partial \mathcal{L}}{\partial \mathbf{g}} = \frac{\partial \mathbf{f}}{\partial \mathbf{g}} \frac{\partial \mathcal{L}}{\partial \mathbf{f}}$.

$$\begin{aligned}
\frac{\partial f_j}{\partial g_i} &= \frac{\partial}{\partial g_i}\left[\max(\alpha g_j, g_j)\right] \\
&= \begin{cases} 0 & \text{if } i \neq j \\ \alpha & \text{if } i = j \text{ and } \alpha g_j > g_j \\ 1 & \text{if } i = j \text{ and } \alpha g_j \leq g_j \end{cases}
\end{aligned}$$

Therefore $\frac{\partial \mathbf{f}}{\partial \mathbf{g}} = \text{diag}(I(\alpha g_1 > g_1)\alpha + (1 - I(\alpha g_1 > g_1)), \ldots, I(\alpha g_m > g_m)\alpha + (1 - I(\alpha g_m > g_m)))$, and $\frac{\partial \mathcal{L}}{\partial \mathbf{g}} = \frac{\partial \mathbf{f}}{\partial \mathbf{g}}(\mathbf{f} - \mathbf{y})$.

Next we compute $\frac{\partial \mathbf{g}}{\partial \mathbf{h}}$ and $\frac{\partial \mathbf{g}}{\partial \mathbf{b}}$.

$$\begin{aligned}
\frac{\partial g_j}{\partial h_i} &= \frac{\partial}{\partial h_i}(h_j + b_j) \\
&= \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{else} \end{cases}
\end{aligned}$$

Therefore $\frac{\partial \mathbf{g}}{\partial \mathbf{h}} = I$, and analogously $\frac{\partial \mathbf{g}}{\partial \mathbf{b}} = I$. Therefore $\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}} = \frac{\partial \mathcal{L}}{\partial \mathbf{g}}$.

Finally we can use the provided formula to derive that $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_k} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}} \mathbf{x}^T$.

(c) (5 points) Suppose we have a neural network where the very first layer is an affine layer (i.e. a linear transformation) with weight matrix $\mathbf{W}_1 \in \mathbb{R}^{m \times n}$. Further suppose that there is a special nonzero input $\mathbf{z}$ such that $\mathbf{W}_1 \mathbf{z} = 0$. In linear algebra this is called a *nullspace vector*. It turns out that for all inputs $\mathbf{x} \in \mathbb{R}^n$ and for all scalars $\lambda \in \mathbb{R}$,

$$\mathbf{W}_1 \mathbf{x} = \mathbf{W}_1 (\mathbf{x} + \lambda \mathbf{z}).$$

This is a cool property because this means we can "obfuscate" our neural network input $\mathbf{x}$ by adding $\lambda \mathbf{z}$, and the output would be the same!

It turns out this can be used in training too. Suppose we take every vector $\mathbf{x}$ in our training data and replace it with $\mathbf{x} + \lambda \mathbf{z}$, and then we train on that new dataset. Which parameter updates will be identical to training on the normal data, and which updates will be different? Justify your answer.

Solution: Fix a given input $\mathbf{x}$, and consider its obfuscated variant $\mathbf{x} + \lambda \mathbf{z}$. Because the neural networks activations are going to be the same in the layers after the first linear transformation, all of those gradients are going to be exactly the same. The only gradients that will be affected are the ones that are directly computed using the input. If the input is $\mathbf{x}$, then the gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} = \frac{\partial L}{\partial \mathbf{h}} \mathbf{x}^T$; if the input is $\mathbf{x} + \lambda \mathbf{z}$, then the gradient is $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}} (\mathbf{x} + \lambda \mathbf{z})^T$. This gradient will be the only one that is different, so the rest of the network will train well.

It's worth mentioning that this is only valid on the first step of training, since we would need to recalculate $\mathbf{z}$ everytime we change W_1. We did not take away points if this was not mentioned.

3. **Regularization** (20 points)

(a) (7 points) Consider performing parameter norm L^2 regularization on the weight matrix of a softmax classifier, $\mathbf{W}$. The loss function is:

$$\mathcal{L}_{\text{new}} = \mathcal{L} + \lambda \|\mathbf{W}\|_F^2$$

where $\mathcal{L}$ is the loss without L^2 regularization. How can one choose the value of the regularization parameter (λ)? Discuss your answer in no more than 4 sentences.

Solution: Selecting the regularization parameter is a tricky business. If the value of λ is too high, it will lead to extremely small values of the regression coefficient, which will lead to the model underfitting (high bias – low variance). On the other hand, if the value of λ is 0 (very small), the model will tend to overfit the training data (low bias – high variance). There is no proper way to select the value of λ. What you can do is have a sub-sample of data and run the algorithm multiple times on different sets. Here, the person has to decide how much variance can be tolerated. Once the user is satisfied with the variance, that value of λ can be chosen for the full dataset. One thing to be noted is that the value of λ selected here was optimal for that subset, not for the entire training data.

(b) (7 points) True or False: Introducing regularization to the model always results in equal or better performance on examples not in the training set. Justify your answer in no more than 3 sentences.

Solution: False. If we introduce too much regularization, we can underfit the training set and this can lead to worse performance even for examples not in the training set.

(c) (6 points) How does dropout prevent overfitting? Discuss your answer in no more than 4 sentences.

Solution: Dropout prevents overfitting due to a layer's "over-reliance" on a few of its inputs. Because these inputs aren't always present during training (i.e. they are dropped at random), the layer learns to use all of its inputs, improving generalization.

4. **Loss function and optimization** (25 points)

Equipped with cutting-edge Deep Learning knowledge, you are working with a radiology lab at the UCLA. Specifically, you're asked to build a classifier that predicts the infection type from a given computerized tomography (CT) imaging of the lungs into four ($n_y = 4$) classes: (**coronavirus, respiratory syncytial virus, bacteria, fungus**). There's always exactly one infection per image. You decide to use cross-entropy loss to train your network. Recall that the cross-entropy (CE) loss for a single example is defined as follows:

$$\mathcal{L}_{CE}(\hat{y}, y) = -\sum_{i=1}^{n_y} y_i \log \hat{y}_i$$

where $\hat{y} = (\hat{y_1}, \hat{y_2}, \cdots, \hat{y_{n_y}})^T$ represents the predicted probability distribution over the classes and $y = (y_1, y_2, \cdots, y_{n_y})^T$ is the ground truth vector, which is zero everywhere except for the correct class (e.g $y = (1, 0, 0, 0)^T$ for **coronavirus** and $y = (0, 0, 1, 0)^T$ for **bacteria**). For this problem, you can assume $0 \log 0 = 0$.

(a) (3 points) Suppose you're given an example CT image of the lung infected by **bacteria**. If the model correctly predicts the resulting probability distribution as $\hat{y} = (0.25, 0.25, 0.3, 0.2)^T$, what is the value of the cross-entropy loss? You can give an answer in terms of logarithms.

Solution: $-\log 0.3$

(b) (3 points) After some training, the model now incorrectly predicts the infection type to be **fungus** with probability distribution $(0, 0, 0.4, 0.6)^T$ for the same image as in part (a). What is the new value of the cross-entropy loss for this example? You can give an answer in terms of logarithms.

Solution: $-\log 0.4$

(c) (4 points) Based on your answer from parts (a) and (b), do you notice some undesirable phenomenon? Explain what implementation choices led to this undesirable phenomenon. Discuss your answer in no more than 3 sentences.

Solution: Based on our answer to part (a) and (b), we can observe that the model achieves lower loss for a misprediction than for a correct prediction. This is because our objective is to minimize cross-entropy loss, rather than to directly maximize accuracy. While cross-entropy loss is a reasonable proxy to accuracy, there is no guarantee that a lower cross-entropy loss will lead to higher accuracy.

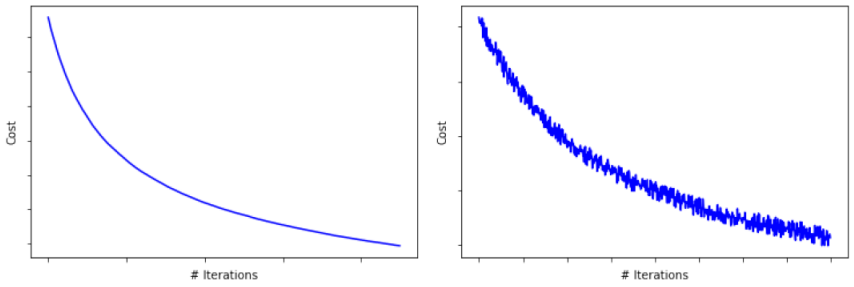
(d) (3 points) Given your observation from part (c), you decide to train your neural network with the accuracy as the objective instead of the cross-entropy loss. Is this a good idea? Give one reason. Note that the accuracy of a model is defined as

$$\text{Accuracy} = \frac{\text{Number of correctly-classified examples}}{\text{Total number of examples}}$$

Solution: It's difficult to directly optimize the accuracy because:

- it depends on the entire training data, making it impossible to use stochastic gradient descent.
- the classification accuracy of a neural network is not differentiable with respect to its parameters.

(e) (5 points) You want to learn the parameters of your model using optimization. Figure 1 shows how the cost decreases (as the number of iterations increases) when two different optimization routines are used for training. Which of the graphs corresponds to using batch gradient descent as the optimization routine and which one corresponds to using mini-batch gradient descent? Explain.



(a) Graph A (b) Graph B

Figure 1: Loss trajectory of two different optimization routines

Solution: Batch gradient descent - Graph A, Minibatch - Graph B. Batch gradient descent - the cost goes down at every single iteration (smooth curve). Mini-batch - does not decrease at every iteration since we are just training on a mini-batch (noisier)

(f) (2 points) Figure 2 shows how the cost decreases (as the number of iterations increases) during training. What could have caused the sudden drop in the cost? Explain one reason.

Solution: Learning rate decay

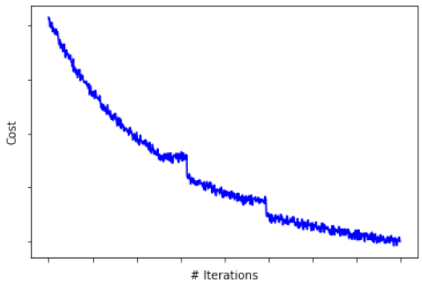


Figure 2: Loss trajectory with a sudden drop

(g) (5 points) After some exploration, you decided to use Adam with bias correction as the optimization routine. Show mathematically why the bias correction naturally disappears when the numbers of steps to compute the exponential moving averages gets large.

Solution: Recall from lecture, that we derived a correction term for a and v as follows:

$$\tilde{v} = \frac{1}{1-\beta_1^t} v$$
$$\tilde{a} = \frac{1}{1-\beta_2^t} a$$

As t gets large, $\beta_1^t \to 0$ and $\beta_2^t \to 0$. Therefore, the correction naturally disappears and

we have:

$$\tilde{v} = v$$
$$\tilde{a} = a$$

5. **Bonus** (6 points)

(a) (3 points) Consider a model based on a softmax with k output values. We replace the hard **0** and **1** classification targets with targets of $\frac{\epsilon}{k-1}$ and $1-\epsilon$ for some small constant ϵ to train the model. What do we accomplish with such change and when does it make sense? Please explain.

Solution: This is called label smoothing and it is a regularization method. It accounts for the mistakes in the dataset, so maximizing the likelihood of $logp(y|x)$ is less penalizing. Therefore model becomes more robust with better performance.

(b) (3 points) Let us assume you have unlimited compute power but limited training data for a vision task. Is using a fully connected network better than using a CNN? Please explain.

Solution: No. FC networks have far more parameters than CNNs, and so with limited training data would be more prone to overfitting.