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

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: Ziyi Yang

Signature: Ziyi Yang 杨子羿

ID#: 405064363

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.

   Classification deals with discrete classes while regression deals with continuous quantities. The output of classification indicates a category whereas the output of regression is a quantity measuring a certain characteristic.

   (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.

   For S.G.D. in each iteration, we update the weights using the gradient of one or a minibatch of training examples. For full-batch G.D. we update using the gradient of all training examples. Thus GD provides a more accurate estimate of gradient but is more computationally expensive.

   (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.

   For small weights initialization, the activations of each layer also tend to be small. When the network is deep, the backpropagation would multiply gradients with many small values (weights and activations) such that as it propagates to the first few layers, the gradient update becomes nearly zero and the network is not learning.

(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.

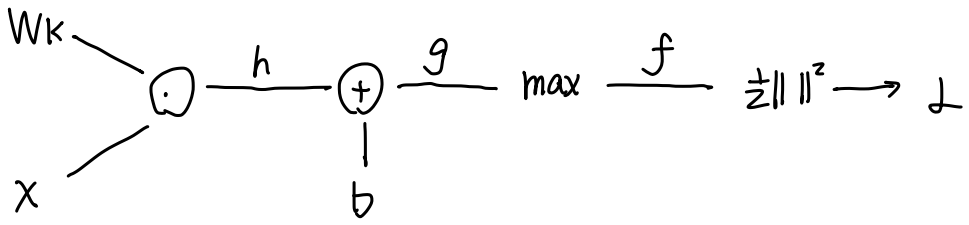
Hint: consider the values of the activations in a single layer.

No, this is not a good idea.
Since all weights are initialized with the same value, when we backpropagate we are also updating each weight with the same gradient descent in each layer. This means for a single layer the weights are always the same, and this severely restricts the capacity of the model and reduce the matrix to a 1D value.

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.



* max is a shorthand for function

$$f(x) = \max(\alpha x, x)$$

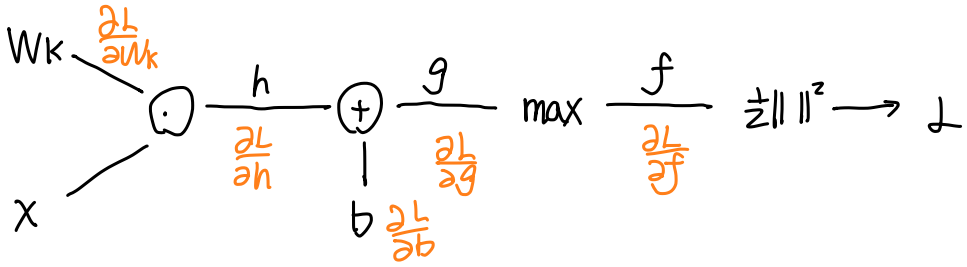
* $\frac{1}{2}|| \ ||^2$ is a shorthand for function

$$f(x) = \frac{1}{2}||x - y||^2$$

(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.



$$L = \frac{1}{2}\|f-y\|^2 = \frac{1}{2}(f-y)^T(f-y)$$

$$= \frac{1}{2}(f^T-y^T)(f-y) = \frac{1}{2}(f^Tf - f^Ty - y^Tf + y^Ty)$$

$$\frac{\partial L}{\partial f} = \frac{1}{2}(2f-2y) = f-y$$

$$f = \max(\alpha \cdot g, g)$$

$$\frac{\partial f_i}{\partial g_i} = \frac{\partial}{\partial g_i}(\max(\alpha g_i, g_i)) = \begin{cases} 0 & \text{if } i \neq j \\ \alpha & \text{if } g_i < 0 \\ 1 & \text{else} \end{cases} \quad i=j$$

$$\frac{\partial f}{\partial g} = \text{diag}\left(\mathbb{I}(g_1<0)\cdot\alpha + \mathbb{I}(g_1 \geq 0)\cdot 1, \ldots, \mathbb{I}(g_m<0)\cdot\alpha + \mathbb{I}(g_m \geq 0)\cdot 1\right)$$

↖ a $m \times m$ diagonal matrix

$$\frac{\partial L}{\partial g} = \frac{\partial f}{\partial g}\frac{\partial L}{\partial f} = \frac{\partial f}{\partial g}(f-y) \qquad \frac{\partial L}{\partial h} = \frac{\partial L}{\partial b} = \frac{\partial L}{\partial g} = \frac{\partial f}{\partial g}(f-y)$$

$$\frac{\partial L}{\partial W_k} = \frac{\partial L}{\partial h}x^T = \frac{\partial f}{\partial g}(f-y)x^T$$

in summary, $\frac{\partial L}{\partial b} = \frac{\partial f}{\partial g}(f-y)$

$$\frac{\partial L}{\partial W_k} = \frac{\partial f}{\partial g}(f-y)x^T$$

where $\frac{\partial f}{\partial g} = \text{diag}\left(\mathbb{I}(g_1<0)\cdot\alpha + \mathbb{I}(g_1 \geq 0)\cdot 1, \ldots, \mathbb{I}(g_m<0)\cdot\alpha + \mathbb{I}(g_m \geq 0)\cdot 1\right)$

is a $m \times m$ diagonal matrix

(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.

Since $W_1x = W_1(x+\lambda z)$, h, g, f, L would all stay the same.
When we backpropagate from $\frac{\partial L}{\partial f}$, $\frac{\partial L}{\partial g}$, $\frac{\partial L}{\partial h}$ and $\frac{\partial L}{\partial b}$ will be the same.
However, since $\frac{\partial L}{\partial W_k} = \frac{\partial L}{\partial h}(x+\lambda z)^T$ now, the update of W_k is different.

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.

λ is a hyperparameter, so we could test a list of λ values with validation set. Also by plotting the change of training and validation error across iterations, we can check if λ is large enough to effectively reduce the overfitting or if λ is so big as to result in underfitting. Choose λ that minimizes both training and validation errors.

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

False, it really depends on the value of λ and the model itself.
An overkill regularization would lead to underfitting such that both training and validation errors are high, which might be worse than without a regularization term.

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

Dropout applies a random mask to activations in each layer and approximates training a large ensemble of models. We can view dropout as training these subnetworks and thus an approximation of bagging. In a FC network neurons tend to develop co-dependency amongst each other during training which curbs the individual power and leads to overfitting. Dropout effectively forces different combinations of neurons to "re-adapt" and thus reduces co-dependency and overfitting.

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**).

(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.

$$L_{CE}(\hat{y}, y) = -\sum_{i=1}^{4} y_i \log \hat{y}_i = -y_3 \log \hat{y}_3 = -\log 0.3$$

$y_1 = y_2 = y_4 = 0$

(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.

$$L_{CE}(\hat{y}, y) = -\sum_{i=1}^{4} y_i \log \hat{y}_i = -(0 + 0 + \log 0.4 + 0)$$

$$= -\log 0.4$$

(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.

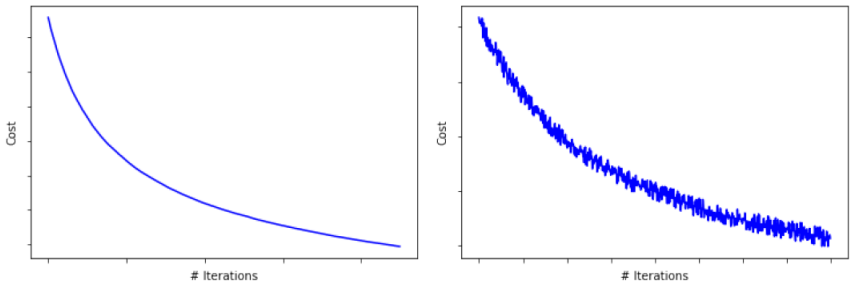
Even though in (b) we misclassify the infection type while in (a) the prediction is correct, the cross-entropy loss in (b) is smaller than in (a). This is because the y vector all wrong classes are 0. And based on the definition of cross-entropy, we only care how large is the probability of the right class. (in (b), $P(\hat{y}_3) = 0.4 > 0.3$ so loss is smaller than (a))

(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

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

This is not necessarily a good idea, since an overfitting has very high training accuracy but poor performance in validation and test set. This metric does not guarantee good generalization.

(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

Graph A is batch GD and B is mini-batch GD. Graph B is much more jagged, and this noise comes from random sampling in mini-batch. Since in each iteration, the gradients are chosen from different examples, the cost occasionally goes up. In full batch there is no such randomness, so graph A is very smooth.

(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.

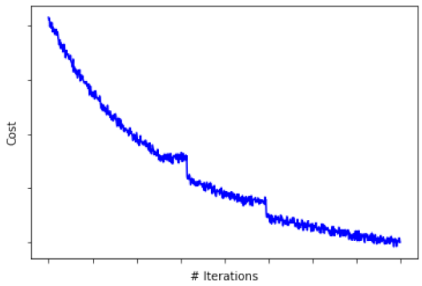


Figure 2: Loss trajectory with a sudden drop

The sudden drop indicates an adjustment in learning rate at certain point.

(g) (5 points) After some exploration, you decide 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.

Adam :

1st moment update : $v \leftarrow \beta_1 v + (1-\beta_1) g$

2nd moment update : $a \leftarrow \beta_2 a + (1-\beta_2)\, g \odot g$

bias correction : $\hat{v} = \frac{1}{1-\beta_1^t} v$

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

t is the number of iterations and $0 < \beta_1, \beta_2 < 1$

as $t \to \infty$, $\beta_1^t, \beta_2^t \to 0$ and thus

$$\lim_{t\to\infty} \frac{1}{1-\beta_1^t} = 1 \,, \quad \lim_{t\to\infty} \frac{1}{1-\beta_2^t} = 1$$

As the number of steps gets large, bias correction turns to 1, which is equivalent to doing nothing to v and a

5. **Bonus question: Intuition** (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.

(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.

Yes. Fully connected network has stronger connectivity than CNN, so when training data is limited, FC net tends to use data more compactly. Also now that we have unlimited computer power, the computational complexity of FC net is no longer a problem.