Midterm for CSC413/2516, Neural Networks and Deep Learning Winter 2020 Friday, Feb. 14, 6:10-8:00pm

Name: _____

Student number: _____

This is a closed-book test. It is marked out of 15 marks. Please answer ALL of the questions. Here is some advice:

- The questions are NOT arranged in order of difficulty, so you should attempt every question.
- Questions that ask you to "briefly explain" something only require short (1-3 sentence) explanations. Don't write a full page of text. We're just looking for the main idea.
- None of the questions require long derivations. If you find yourself plugging through lots of equations, consider giving less detail or moving on to the next question.
- Many questions have more than one right answer.

.

Q1: _____ / 2
Q2: _____ / 1
Q3: _____ / 1
Q4: _____ / 2
Q5: _____ / 1
Q6: _____ / 2
Q7: _____ / 2
Q8: _____ / 3
Q9: _____ / 1

Final mark: _____ / 15

1. Consider training a convolutional neural network on RGB images. Let the input image size be 100x100 pixels with three color channels. There are 16 convolution filters in the first conv layer. They have the same kernel size 5×5 . Assume the output size of the convolution is the same as the image. There is no bias unit in this conv net.
Marking: 0.25/1 is given when getting both (a)(b) questions wrong but having 16 filters and 5×5 kernel correct in the answers.

   (a) **[0.5pt]** How many learnable weights are there in the first conv layer?
   Solution: $5 \times 5 \times 16 \times 3$
   Marking: 0.5 for correct answer. 0 for wrong answer.
   For all the questions in 1, If the student is not considering input color channels 3, but gets everything else correct for the question, only penalize such error once.

   (b) **[0.5pt]** How many add-multiply operations do we need to compute the output of the conv layer?
   Solution: $100 \times 100 \times 5 \times 5 \times 16 \times 3$
   Marking: 0.5 for correct answer. 0 for wrong answer.
   Any error with the padding for height and width '100' is not penalized. Counting add-multiply (either separately for add&mult or add-multiply as one op) is ok.

   (c) **[0.5pt]** If we increase the input image size from 100x100 to 200x200, how many more parameters do we need to add to the first conv layer?
   Solution: No additional parameter needed.
   Marking: 0.5 for correct answer. 0 for wrong answer.

   (d) **[0.5pt]** If we increase the input image from 100x100 to 200x200, how does the number of add-multiply operations change?
   Solution: It will increase by a factor of 4.
   Marking: 0.5 for correct answer. 0 for wrong answer.
   As long as the explanation or equation shows and increase by a factor of 4 (or 4-1=3 times additional parameters, it gets 0.5

2. **[1pt]** When training a deep neural network, we encounter out of memory error during back-propagation. Assume there is no bug in the code. We also want to keep the same model architecture and the dataset. Describe two potential ways to avoid the out of memory error.

   Solution: 1) reduce the minibatch size. 2) To keep the minibatch size the same, we can perform backprop on a partition of the training examples and accumulate the gradients across the different partitions. 3) recompute hidden activation during backprop, so we only need to store the error signal and the most immediate hidden activations.

   Marking:

   - 0.5 for each of the correct answers from the above
   - Must not change the model architecture in any way
   - This question does not assume training CNN over images. Any answer making this assumption do not get any partial marks.
   - Must not change the dataset in anyways (e.g. downsampling input)
   - To obtain full 0.5 marks for answers relating to 1), must mention smaller / reducing mini-batch size. Simply mentioning SGD or mini-batch is not sufficient and 0.25 is given. This is because the question does not specify which optimizer is already being used for training, so it could already be using SGD with mini-batch (which is the case for most of the time).
   - No marks given for answers simply suggesting better hardware or increased memory.
   - No marks given for simply stating saving intermediate values to disk or releasing unnecessary variables. Need to explain this by relating to reducing the number of intermediate activations saved.
   - 0.25 given for just mentioning reducing number of intermediate activations stored. No marks for ambiguous answers like reducing number of activations and must mention activation stored.
   - No marks given for mentioning VJP since back-prop already uses it.

3. **[1pt]** In Homework 1, we hand-designed a neural network to perform sorting. In this question, you will design a neural network that finds the median of a list of 5 scalar inputs, $x_1, x_2, x_3, x_4, x_5 \in \mathbb{R}$. Draw the architecture and the weights of the median network. (Hint: You may first show a neural network that sorts 5 numbers. Then use the sorting network as a building block for the final solution.)

   Solution: Create a sorting network that has 5 outputs. The final output layer will have the weight vector $[0, 0, 1, 0, 0]$.

   Marking:

   - (0.75 pt) for correct sorting architecture + weights
     (a) Solution 1: VerifySort over all permutations. (0.5 pt) for VerifySort architecture and weights, and (0.25 pt) for using that to create sort, by considering every possible permutations. Another variant is "VerifyMedianInMiddle", where the network outputs true if the middle input is indeed the median, by checking if it is precisely larger than 2 of the inputs.
     (b) Solution 2: BubbleSort style (Min/Max, Sort-2 swapping). (0.50 pt) for proper max/min architecture, and (0.25 pt) for incorporating it into a bubble-sort.
     (c) If there are missing/incorrect implementation details (weights for VerifySort/2Sort is missing, or incorrect) then take off (-0.25 pt) each.
     (d) If no weights at all are given (i.e. no attempt); the subnetworks are just treated as black boxes, then, this is a (-0.5 pt) deduction.
   - (0.25 pt) Having the correct final output layer weight vector $[0, 0, 1, 0, 0]$, or show that the output only comes from the median unit among the sorted units.
   - Another acceptable solution is to implement a network to check whether each of the number is a median, and then only output the number where this is true. Similar marking scheme where points are awarded for having correct weights and architecture for extracting the correct output.

4. Recall the softmax cross entropy loss of D classes is given by:

$$\mathcal{L} = -\sum_{i=1}^{D} t_i \log y_i, \quad \text{where } y_i = \frac{exp\{z_i\}}{\sum_{j=1}^{D} exp\{z_j\}}$$

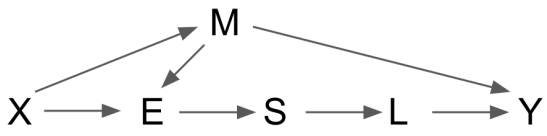
The above expression can be simplified in terms of the logits:

$$\mathcal{L} = -\sum_{i=1}^{D} t_i \log \frac{exp\{z_i\}}{\sum_{j=1}^{D} exp\{z_j\}} = -\sum_{i=1}^{D} t_i z_i + \log \sum_{j=1}^{D} exp\{z_j\}$$

In practice, the logits, z, can take on very small values, which cause numerical instability in the second term, $\log \sum_{j=1}^{D} exp\{z_j\}$. (Taking the log of a value close to zero needs to be avoided.) Most of the machine learning libraries use a numerically stable implementation for the second term, called "LogSumExp":

```
def logsumexp(X):
  M = np.max(X)
  E = np.exp(X - M)
  S = np.sum(E)
  L = np.log(S)
  Y = M + L
  return Y
```

(a) **[1pt]** Draw the computation graph of LogSumExp in terms of the intermediate variables, X, M, E, S, L, Y.



(b) **[1pt]** Derive the backprop updates for the whole LogSumExp function, i.e. $\overline{X}$ given $\overline{Y}$. Notice Y is a scalar (You may give your answers to either D=2 or for the more general case.) (Hint: Y is a scalar.)

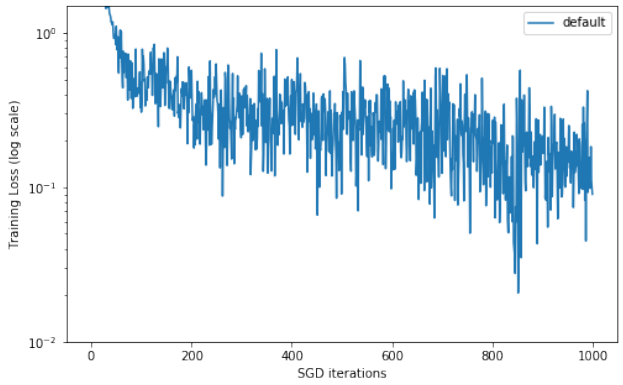
Solution:

$\overline{L} = \overline{Y}$
$\overline{S} = \overline{L}\frac{\partial L}{\partial S} = \frac{\overline{L}}{S}$

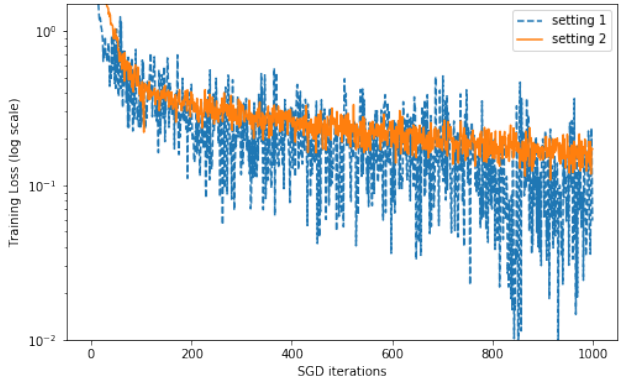
$\overline{E_i} = \overline{S}$
$\overline{M} = \overline{Y}\frac{\partial Y}{\partial M} + \overline{E}\frac{\partial E}{\partial M} = \overline{Y} - \overline{E}.e^{X-M} = \overline{Y} - \overline{E}.E$
$\overline{X_i} = \overline{M}\mathbb{1}(i = \text{argmax}(X)) + \overline{E_i}.E_i$

Marking: One minor mistake like incorrect sign is ignored.
Errors such as forgetting the indicator function in the last equation, computing the partial derivative incorrectly and ignoring part of the computation graph reduce the mark to 0.5.

5. **[1pt]** Consider training a neural network language model from Programming Assignment 1 using stochastic gradient descent (SGD). Given the starter code learning rate and batch size, we obtain the following training curve, that is the training loss v.s. the number of iteration plot.



To improve convergence, We tweaked learning rates and batch sizes in two different settings (setting 1 and setting 2):



Describe what has been changed in each setting in terms of learning rate and batch size. Briefly justify your answer.

setting 1: Solution: increase learning rate and/or decrease batch size, which increases the update variance.

Marking: Full (0.5) marks for correctly identifying either factor, so long as the second factor is (a) also listed and correct, (b) not listed or (c) kept neutral (i.e., "increase learning rate and keep batch size the same"). No marks are given if either factor is listed in the opposite direction (i.e., decreased learning rate or increased batch size).

setting 2: Solution: increase batch size and/or decrease learning rate, which reduces the update variance.

Marking: Full (0.5) marks for correctly identifying either factor, so long as the second factor is (a) also listed and correct, (b) not listed or (c) kept neutral (i.e., "increase batch size and keep learning rate the same"). No marks are given if either factor is listed in the opposite direction (i.e., increased learning rate or decreased batch size).

Additional notes:

- Many students switched Setting 1 & Setting 2. This was not penalized so long as it was clear, from the student explanation, which setting was being referenced.
- Minor repeated errors in both parts were not doubly penalized if majority of solution had correct explanation.

6. Recall the quadratic model from Lecture 4:

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{\theta}^\top \mathbf{A}\boldsymbol{\theta}$$

Spectral decomposition $\mathbf{A} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^\top$ helps us understand the curvature matrix $\mathbf{A}$ in terms of its diagonal eigenvalue matrix $\boldsymbol{\Lambda}$. We showed the dynamics of gradient descent updates can be described by recurrence:

$$\boldsymbol{\theta}_t = \mathbf{Q}(\mathbf{I} - \alpha\boldsymbol{\Lambda})^t\mathbf{Q}^\top\boldsymbol{\theta}_0.$$

Consider optimize two different quadratic problems with different curvature matrices, $\mathbf{A}_1$ and $\mathbf{A}_2$.

$$\mathcal{J}(\boldsymbol{\theta})_1 = \frac{1}{2}\boldsymbol{\theta}^\top \mathbf{A}_1\boldsymbol{\theta}, \quad \text{where } \mathbf{A}_1 = \mathbf{Q}_1\boldsymbol{\Lambda}_1\mathbf{Q}_1^\top, \quad \boldsymbol{\Lambda}_1 = \begin{bmatrix} 30 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & 10 \end{bmatrix}$$

$$\mathcal{J}(\boldsymbol{\theta})_2 = \frac{1}{2}\boldsymbol{\theta}^\top \mathbf{A}_2\boldsymbol{\theta}, \quad \text{where } \mathbf{A}_2 = \mathbf{Q}_2\boldsymbol{\Lambda}_2\mathbf{Q}_2^\top, \quad \boldsymbol{\Lambda}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.0001 \end{bmatrix}$$

(a) **[1pt]** On which quadratic problem ($\mathcal{J}(\boldsymbol{\theta})_1$ or $\mathcal{J}(\boldsymbol{\theta})_2$) will gradient descent converge faster? Briefly justify your answer.

Solution: The condition number for the two problems is $\kappa_1 = 3, \kappa_2 = 10,000$. Therefore, $\mathcal{J}(\boldsymbol{\theta})_1$ will converge faster because its condition number is lower.

Marking: 0.5 marks for providing $\mathcal{J}(\boldsymbol{\theta})_1$ as faster convergence. 0.5 marks for providing justification (ratio of condition numbers). Deduct 0.25 marks if condition number or other similar representation is not explicitly mentioned, but ill-condition curvature is mentioned. Awarded only 0.25 marks if condition number was used incorrectly (ratio was commonly inversed). One common mistake was using the absolute magnitude of eigenvalues instead of ratio between largest/smallest.

(b) **[1pt]** If we can multiply all the Eigenvalues in $\boldsymbol{\Lambda}_1$ by a constant factor c, will it change the convergence speed of gradient descent? If so, by how much? Briefly justify your answer.

Solution: 1) it will not change the convergence because the condition number is still $\kappa_1 = 3$. 2) Assume learning rate does not change, the convergence speed is changed by a factor of c.

Marking: Full marks for stating affect on convergence **including** justification related to response. No marks awarded providing binary response without justification. 0.5 marks deducted if change in convergence provided for solution 2 was not provided or inaccurate.

7. A ReLU unit is considered "dead" if it always receives inputs smaller than 0 for *all* the training examples. Although dead ReLUs can show up in various different architectures, consider a two-hidden-layer MLP for this question.

   (a) **[0.5pt]** For a dead ReLU neuron, what is the gradient for its incoming weights and its outgoing weights? Provide a brief justification for your answer.

   Solution: For each training example, $\overline{W}^{(i)} = \frac{\partial ReLU(z^{(i)})}{\partial z}\overline{h^{(i)}}h^{(i-1)^T}$.

   Incoming weights, the Jacobian of the dead ReLU units is zero, so the gradient for the incoming weights is also zero.

   Outgoing weights, the dead ReLU neuron $h_j^{(i-1)}$ will always be zero, so its corresponding outgoing weights will receive zero gradient.

   Marking:

   - Correct answer: 0.4
   - Brief explanation: 0.1

   (b) **[0.5pt]** Provide two scenarios how we may encounter a dead neuron during training.

   Solution: 1) Learning rate being too large overshoot the bias unit. 2) Dead upon initialization. 3) the incoming weights are orthogonal to the row space of the design matrix and the corresponding bias units are negative. 4)Input data not centered. 5) weights and biases too large on non-centered data.

   Marking:

- Each correct scenario: 0.25 points
- Accept any explanation that refers to a non-relu unit (such as tanh).
- If the student answered with methods to *detect* dead ReLUs.

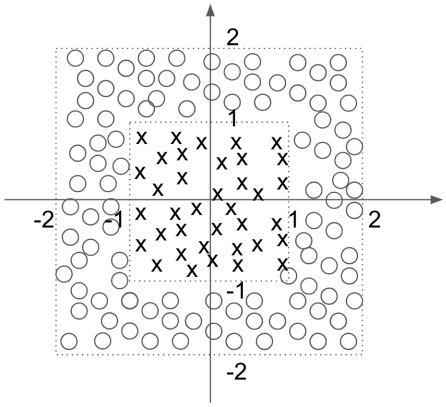
(c) **[1pt]** For a dead ReLU neuron in the second hidden layer to ever becomes dead, does it ever have the chance to get revived during training? Provide a brief justification for your answer.

Solution: Yes, dead ReLU neurons may be revived. The input hidden vector from layer below would change during training. The hidden activations from layer below may increase in magnitude and/or align with the incoming weights of the dead ReLU unit, such that the logit of the dead ReLU neuron might become positive.

Marking:

- Correct answer: 0.5 points
- Explanation: 0.5 points
- If answer is wrong but the explanation make a bit of sense: 0.25 points
- If the answer involves an external intervention, 0.5 points.
- No points for mentioning ill-conditioned curvature, as it's not related to dead units.

8. Consider the following 2D classification problem. Assume the training examples are drawn i.i.d. from a uniform square in the input space $x_1, x_2 \in [-2, 2]$. Data points are assigned to the positive class if they fall in the center square between -1 to 1, and to the negative class otherwise.



(a) **[1pt]** When there are *more* than three training examples, can an (underparameterized) linear classifier obtain zero training error on this dataset? Provide proof to justify your answer. (Hint: convex set)

Solution: Linear classifier will not always obtain zero training error. The mean of the training examples from either the positive or the negative class is $(0, 0)$. If the dataset is linearly separable then $(0, 0)$ can only be assigned to either of the classes. Thus, contradiction.

Marking:

- (1 pt) No, and gives the proof sketch as in the solution, or find a counter example
- (0.5 pt) No, and attempts to give a somewhat plausible proof, but the proof is not quite correct
- (0 pt) No, no proof or wrong proof
- (0.5 pt) Yes, and gives a special case where the points are linearly separable
- (0 pt) Yes, and try to argue that it is the general case

(b) **[1pt]** When there are *less* than or equal to three training examples, can an (over-parameterized) linear classifier obtain zero training error on this dataset? Provide proof to justify your answer.

Solution: 1) Yes, it can simply memorize the class assignment due to overcompelete linear system. 2) No. When all three training examples lie on a line (o, x, o), then a linear classifier is not able to classify this perfectly.

Marking:

- (1 pt) Yes, and point out that there are at least the same # of parameters than # of equations, or that the classifier can "memorize" all examples.
- (0.5 pt) Yes, but only intuitively / pictorially gives the justification
- (1 pt) No, and gives the co-linear counter example
- (0.5 pt) No, because ≤ 3 examples are not enough to learn the decision boundary (misunderstanding of question)
- (0 pt) Other answers

(c) **[1pt]** When there are *more* than three training examples, can a one-hidden-layer MLP consist of a single sigmoid unit (i.e. the hidden size is 1) achieve zero training error? You do not need to justify your answer.

Solution: No. A single sigmoid neuron MLP is no more powerful than a linear classifier.

Marking: 1 pt for No, and 0 pt for Yes

9. **[1pt]** Consider running gradient descent on a simplified GloVe model from Programming Assignment 1 without the bias. The training objective is given in terms of the embedding matrix $\mathbf{W}$ containing word embeddings $\{\mathbf{w}_i\} \in \mathbb{R}^d$:

$$\mathcal{J}(\mathbf{W}) = \sum_{i,j} (\mathbf{w}_i^\top \mathbf{w}_j - \log x_{ij})^2.$$

Instead of random initialization, assume we initialize all the word embeddings to be exactly the same vector $\mathbf{c} \in \mathbb{R}^d$. Show that for any embedding size d, gradient descent will learn a degenerate model that is equivalent to training with embedding size $d = 1$. (Hint: think about the row space and the rank of the embedding matrix.)

Solution: In this case, upon initialization, the weight matrix is rank-1 $\mathbf{W} = \mathbf{1}\mathbf{c}^T$. So the row space is a line in d-dimensional space. The gradient descent updates always lie in the row space of the weight matrix. Therefore, the weight matrix will always be a rank-1 matrix. This is equivalent to an embedding size of 1.

Marking:

To get full marks on this question you need to do the following:

(a) Recognize that we are doing ww^T matrix factorization (i.e. a type of matrix regression/approximation). You need this point since in general if you use data directly, your gradient will become data dependent, and could escape the rowspace.

(b) Make an argument about why the gradient descent updates are all scalar multiples of a row of the weight matrix

(c) From this point, conclude that the updates must lie in the rowspace. You should argue that the row is updated, but with a linear combination of the other rows. Therefore, the rowspace of the weight matrix is the same

(d) Finally, conclude that if the rowspace of the matrix is the same, then all word embeddings are collinear.

(e) Argue that collinear word embeddings mean we effectively have a dimension size of 1. (only the scaling of c will be important)

In general, the key point of this question is to show that the gradient descent updates lie in the rowspace. Hence, we will receive a rank-k embedding matrix, for any initial rank k of our initialization.

You will receive 0.5 mark for the gradient rowspace argument (a to c), and 0.5 mark for the collinear word embeddings (d to e). Critically, you must present a coherent argument (i.e. marks are not given for simply giving a bag-of-words of the correct components).

Common mistakes:

- Arguing that all rows receive the same gradient descent update. This is true only on the first batch of the first iteration.
- You should clarify the orientation of the weight matrix; I have assumed the canonical $N \times D$ notation
- The gradients are neither the same from batch to batch, nor are the gradient updates to each of the dimension the same. The rowspace argument is critical here

Additional remarks:

- pointing out that the dimension of the embedding matrix is good, but not enough unless you justify why. i.e. it is a useful way to conclude a proof, but cannot stand alone as a full answer.
- it is also acceptable to justify dimension via an eigenvalue decomposition argument
- if you can show the gradients are all in terms of c, this is also an acceptable way of showing the gradients never leave the rowspace. (i.e. you do not need to explicitly make an argument). But you should generally argue that the linear combination of gradients means we stay in the rowspace.

(Scratch work or continued answers)