Midterm for CSC421/2516, Neural Networks and Deep Learning Winter 2019 Friday, Feb. 15, 6:10-7:40pm

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

Final mark: _____ / 15

1. **[1pt]** In our discussion of language modeling, we used the following model for the probability of a sentence.

$$p(w_1, \ldots, w_T) = p(w_1)\, p(w_2 \mid w_1) \cdots p(w_T \mid w_1, \ldots, w_{T-1}) \qquad \text{(step 1)}$$
$$p(w_t \mid w_1, \ldots, w_{t-1}) = p(w_t \mid w_{t-3}, w_{t-2}, w_{t-1}) \qquad \text{(step 2)}$$

For each of the two steps, say what assumptions (if any) must be made about the distribution of sentences in order for that step to be valid. (You may assume that all the necessary conditional distributions are well-defined.)

Step 1: No assumption or chain rule of probability.

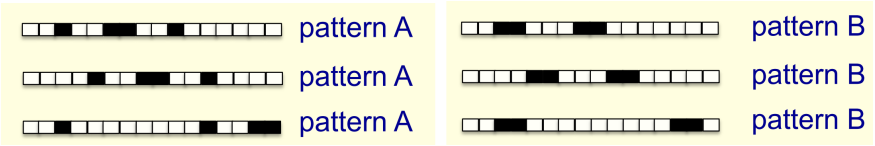
Marking: (+0.5) for correct answer. Answers that mentioned axioms of probability also were given full marks.

Step 2: Markov assumption (of order three).

Marking: (+0.5) for correct answer. Answers that explained Markov assumption in words were also given full marks.

Mean: 0.70/1

2. **[1pt]** Consider the following binary classiciation problem from Lecture 3, which we showed was impossible for a linear classifier to solve.



The training set consists of patterns A and B in all possible translations, with wrap-around. Consider a neural network that consists of a 1D convolution layer with a linear activation function, followed by a linear layer with a logistic output. Can such an architecture perfectly classify all of the training examples? Why or why not?

No. Convolution layers are linear, and any composition of linear layers is still linear. We showed the classes are not linearly separable.

Marking: (+0.5) Correct answer and partial justification. (+0.5) Correct justification. A complete answer includes mention of the whole neural network computing a linear function up to the final non-linearity and the data being linearly inseparable.

Mean: 0.62/1

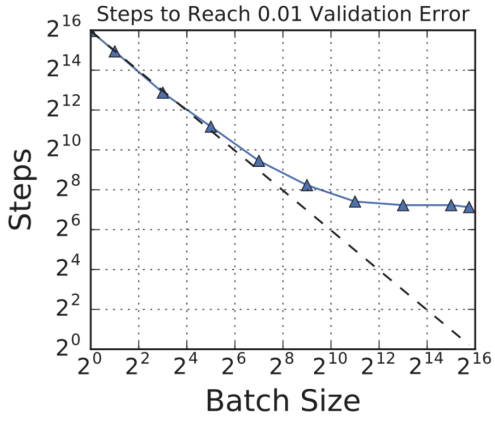
3. **[1pt]** Recall that `autograd.numpy.dot` does some additional work that `numpy.dot` does not need to do. Briefly describe the additional work it is doing. You may want to refer to the inputs and outputs to `autograd.numpy.dot`.

   In addition, autograd.numpy.dot add a node to the computation graph and stores its actual input and output value during the forward computation.

   Marking: Full marks were given to most students for mentioning the construction of a computation graph. (-0.5) marks for being too vague and just mentioning keywords. (-1) marks off for saying something incorrect.

   Mean: 0.84/1

4. **[2pts]** Recall the following plot of the number of stochastic gradient descent (SGD) iterations required to reach a given loss, as a function of the batch size:



.

   (a) **[1pt]** For small batch sizes, the number of iterations required to reach the target loss decreases as the batch size increases. Why is that?

   Larger batch sizes reduces the variance in the gradient estimation of SGD. Hence, larger batch converges faster than smaller batch.

   Marking: Most mentions of variance or noise being decreased were sufficient to get full marks for this question. (-1) for not mentioning anything regarding noise/variance or accuracy of gradient estimate given by SGD.

(b) **[1pt]** For large batch sizes, the number of iterations does not change much as the batch size is increased. Why is that?

As the batch size grows larger, SGD effectively becomes full batch gradient descent.

Marking: Full marks given for mentioning that large batches approximate full-batch gradient descent, so not much noise to be reduced in gradient estimation. (-1) if answer has no mention of full-batch gradient descent, (-0.5) if answer is vague.

Mean: 1.00/2

5. **[1pt]** Suppose we are doing gradient descent on a quadratic objective:

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

We showed that the dynamics of gradient descent with learning rate α could be analyzed in terms of the spectral decomposition $\mathbf{A} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^\top$, where $\mathbf{Q}$ is an orthogonal matrix containing the eigenvectors of $\mathbf{A}$, and $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_D)$ is a diagonal matrix containing the eigenvalues of $\mathbf{A}$ in ascending order.

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

Based on this formula, what is the value C such that the gradient descent iterates diverge for $\alpha > C$ but converge for $\alpha < C$? Briefly justify your answer.

$$|1 - \alpha\lambda_{max}| < 1$$

$$\alpha < \frac{2}{\lambda_{max}}$$

Marking: 0.5 if $\frac{1}{\lambda_{max}}$. 0.5 if no derivation is given in the justification.

Mean: 0.40/1

6. **[1pt]** Consider the GloVe cost function, in terms of matrices $\mathbf{R}$ and $\tilde{\mathbf{R}}$ containing word embeddings $\{\mathbf{r}_i\}, \{\tilde{\mathbf{r}}_j\}$

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

(We left out the bias parameters for simplicity.) Show that this cost function is not convex, using a similar argument to how we showed that training a multilayer perceptron is not convex.

This is a non-convex cost function.

Solution 1: When we permute the dimensions of the embedding vectors in $\mathbf{R}$ and $\tilde{\mathbf{R}}$ jointly, the cost function remains the same. To show convexity does not apply here, we can take the average of these permuted embedding vectors. The resulting embedding vectors will have all the same value for all the dimension, which will almost surely have a higher cost than the learnt embedding vectors. (Note: as an alternative to permutation symmetry, you can simply replace $\mathbf{R}$ with $-\mathbf{R}$ and $\tilde{\mathbf{R}}$ with $-\tilde{\mathbf{R}}$.)

Solution 2: We can interchange $\mathbf{R}$ and $\tilde{\mathbf{R}}$ directly and the cost function will remain the same. If we average the two embedding matrix $\frac{\mathbf{R}+\tilde{\mathbf{R}}}{2}$, we will have the same word embedding vector for both matrices, which will have higher cost than the original cost function. It is because the highest occurance will always be the inner product of the words with itself.

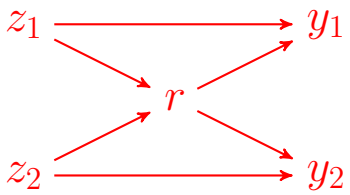
Marking: 0.5 for noting the swap-invariance or permutation-invariance. 0.5 for applying the invariance to get the average of solutions.

Mean: 0.32/1

7. **[3pts]** Recall that the softmax function takes in a vector $(z_1, \ldots, z_D)$ and returns a vector $(y_1, \ldots, y_D)$. We can express it in the following form:

$$r = \sum_j e^{z_j} \qquad\qquad y_i = \frac{e^{z_i}}{r}$$

(a) **[1pt]** Consider $D = 2$, i.e. just two inputs and outputs to the softmax. Draw the computation graph relating z_1, z_2, r, y_1, and y_2.



Marking: (+0.5) for having all nodes. (+0.5) for having all edges.

(b) **[1pt]** Determine the backprop updates for computing the $\overline{z_j}$ when given the $\overline{y_i}$. You do not need to justify your answer. (You may give your answer either for $D = 2$ or for the more general case.)

$$\overline{r} = -\sum_i \overline{y_i} \frac{e^{z_i}}{r^2}$$

$$\overline{z_j} = \overline{y_j} \frac{e^{z_j}}{r} + \overline{r} e^{z_j}$$

Marking: (+0.5) for each equation. Common mistakes were missing the partial derivative from $\bar{y}_j$ for $\bar{z}_j$ or missing the summation for $\bar{r}$.

(c) **[1pt]** Write a function to implement the vector-Jacobian product (VJP) for the softmax function based on your answer from part (b). For efficiency, it should operate on a mini-batch. The inputs are:

- a matrix `Z` of size $N \times D$ giving a batch of input vectors. N is the batch size and D is the number of dimensions. Each row gives one input vector $\mathbf{z} = (z_1, \ldots, z_D)$.
- A matrix `Y_bar` giving the output error signals. It is also $N \times D$.

The output should be the error signal `Z_bar`. Do not use a `for` loop.

```
def softmax_vjp(Z, Y_bar):
    R = np.sum(np.exp(Z), axis = 1, keepdims=True)
    R_bar = -np.sum(Y_bar * np.exp(Z), axis=1, keepdims=True)/R**2
    Z_bar = Y_bar * (np.exp(Z)/R) + R_bar * np.exp(Z)
    return Z_bar
```

Marking: Full marks were given if the general proecdure is correct, without taking into account the keyword arguments `axis` and `keepdims`. Otherwise (-0.5) for every mistake. Most common mistake was using the matrix multiplication operation `np.dot` as either element-wise multiplication, or using it instead of `np.inner`.

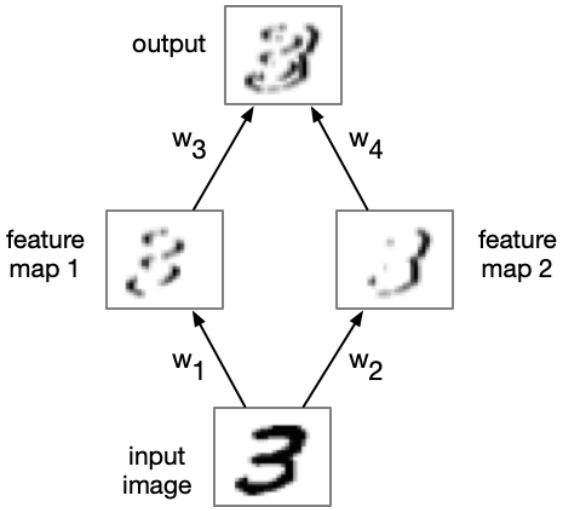
Mean: 2.23/3

8. **[2pts]** In this question, you will design a convolutional network to detect vertical boundaries in an image. The architecture of the network is as shown on the right.

The ReLU activation function is applied to the first convolution layer. The output layer uses the linear activation function.

For this question, you may assume either the standard definition of convolution (which flips and filters) or the version used in conv nets (which skips the filtering step). Conveniently, the same answer works either way.

In order to make the figure printable for the exam paper, we use white to denote 0 and darker values to denote larger (more positive) values.



(a) **[1pt]** Design two convolution kernels for the first layer, of size 3×3. One of them should detect dark/light boundaries, and the other should detect light/dark boundaries. (It doesn't matter which is which.) You don't need to justify your answer.

$w_1 =$

-1	0	1
-1	0	1
-1	0	1

$w_2 =$

1	0	-1
1	0	-1
1	0	-1

Marking: (+0.5) for a kernel that has a positive gradient in the left-right direction. (+0.5) for a kernel that has a negative gradient in the left-right direction. Answers that satisfied the above criteria but weren't horizontally symmetric were given partial marks.

(b) **[1pt]** Design convolution kernels of size 3×3 for the output layer, which computes the desired output. You don't need to justify your answer.

$w_3 =$

0	0	0
0	1	0
0	0	0

$w_4 =$

0	0	0
0	1	0
0	0	0

Marking: (+1) for any two kernels that add the feature maps from the previous layer. Kernels that do this while inadvertently blurring the image were also given full marks.

Mean: 1.60/2

9. **[3pts]** Recall the logistic activation function σ and the tanh activation function:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$
$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Both activation functions have a sigmoidal shape.

(a) **[1pt]** Give the Jacobian matrix $\partial \mathbf{y}/\partial \mathbf{z}$ of the tanh activation function, applied elementwise to all of the units in a layer. You may give your answer in terms of $\tanh'(z)$, the univariate derivative of the tanh function.

$\frac{\partial Y}{\partial Z}$ is a diagonal matrix, where $\tanh'(z_i)$ are the terms along the diagonal.

Marking:

- 1/1 for giving the correct Jacobian matrix. Any valid notation is fine; mainly drawing out the matrix or giving an equation for $\frac{\partial y_j}{\partial z_i}$
- 0.5/1 if the answer demonstrated an understanding of roughly the correct solution (diagonal matrix with $tanh(z_i)$ along diagonal) but made small errors or omissions.
- 0/1 For anything else, including for a vector of $tanh(z_i)$

(b) **[1pt]** One of the difficulties with the logistic activation function is that of saturated units. Briefly explain the problem, and whether switching to tanh fixes the problem. (You may refer to your answer from part (a) or sketch the activation functions.)

No, switching to tanh does not fix the problem. The derivative of $\sigma(z)$ is small for large negtive or positive z. The same problem persists in $tanh(z)$. Both function has a sigmoidal shape. We can see tanh is effectively a scaled and translated sigmoid function: $tanh(z) = 2\sigma(2z) - 1$

Marking: Half a point for an explanation of the problem and half a point for whether switching to tanh fixes the problem.

- Explanation: 0.5/0.5 for a correct explanation with relevant details about saturation A description of saturation or its effect on optimization are both OK.
- Explanation: 0/0.5 for an incorrect explanation or if the explanation demonstrated incorrect understanding, even if parts were correct.

- Switching to tanh: 0.5/0.5 for a "no" answer (or an explanation that obviously implies "no" if yes/no is not explicitly stated).

(c) **[1pt]** Briefly explain one way in which using tanh instead of logistic activations makes optimization easier.

tanh activations are centered around zero, whereas sigmoid are centered around 0.5. Centering the data/hidden activation can help optimization due to similar effect to the batch normalization without the variance division.

Marking:

- 1/1 for a factually correct explanation concerning normalization or centering the data abount 0.
- 0.5/1 or 1/1 for a factually correct description of some property that is plausibly relevant to the ease of optimization. Depends on the depth of reasoning, demonstrated understanding, and relevance to optimization ease.
- 0/1 if the answer makes factually incorrect claims or does not describe something that would make optimization easier.

Mean: 1.45/3

(Scratch work or continued answers)