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.

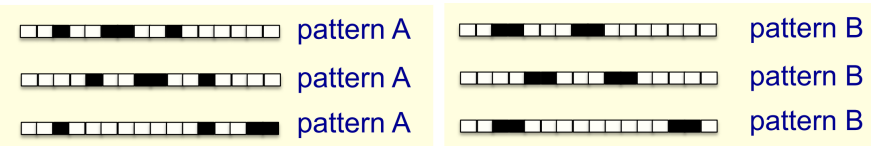
$$p(w_1, \ldots, w_T) = p(w_1)\, p(w_2 \mid w_1) \cdots p(w_T \mid w_1, \ldots, w_{T-1}) \qquad \text{(step 1)}$$
$$p(w_t \mid w_1, \ldots, w_{t-1}) = p(w_t \mid w_{t-3}, w_{t-2}, w_{t-1}) \qquad \text{(step 2)}$$

   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:

   Step 2:

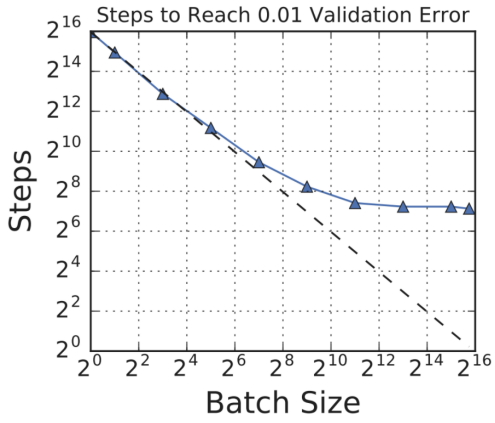
2. **[1pt]** Consider the following binary classiciation problem from Lecture 3, which we showed was impossible for a linear classifier to solve.



   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?

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`.

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?

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

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.

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.

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.

   (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.)

$$\overline{r} =$$

$$\overline{z_j} =$$

   (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.

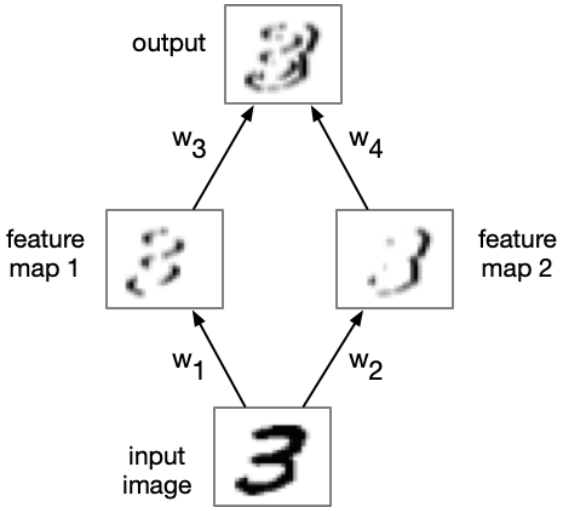
```
def softmax_vjp(Z, Y_bar):
```

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.

$w_1 =$ $\quad$ $w_2 =$

(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.

$w_3 =$ $\quad$ $w_4 =$

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.

(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.)

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

(Scratch work or continued answers)