Afternoon section

1. **[1pt]** *We saw that we can perform polynomial regression (i.e. fit the coefficients of a degree-d polynomial of a scalar variable x) using linear regression with a feature map $\boldsymbol{\phi}(x)$. Assume there is no explicit bias parameter, so the model is $y = \mathbf{w}^\top \boldsymbol{\phi}(x)$. What is $\boldsymbol{\phi}(x)$ if we are fitting a degree-4 polynomial? You don't need to explain your answer.*

$$\boldsymbol{\phi}(x) = \begin{pmatrix} 1 \\ x \\ x^2 \\ x^3 \\ x^4 \end{pmatrix}$$

Marking: Half a point off if missing bias, or if missing x^4.

Mean: 0.69/1

2. **[1pt]** *Suppose we have implemented a function that computes the derivative $\mathrm{d}f/\mathrm{d}x$ for a univariate function f. In order to test the correctness of our implementation using finite differences, we compare the function's output to another value. What is that value, and how is it used?*

We pick a random input to f and compute its exact derivative using our routine. We compare that to the finite differences estimate

$$\frac{f(x+h) - f(x-h)}{2h}.$$

We check that the relative error between the exact derivative and the finite difference approximation is small. The relative error between two values a and b is defined as

$$\frac{|a-b|}{|a|+|b|}.$$

Marking: The point is allocated to the first part ("what is the value?") since the second part ("how is it used?") was too vague. We were looking for an explicit formula for finite differences. Either the one-sided or two-sided version is OK.

Mean: 0.64/1

3. **[2pts]** *Suppose we have two multilayer perceptrons, A and B. TRUE or FALSE: if A has more units than B, then A must also have more connections than B. Explain why it is true or provide a counterexample.*

   FALSE. An example would be a network with a bottleneck layer. E.g, suppose A has three layers with sizes 100, 10, 100, and B has two layers with sizes 100 and 100. Then A has 2000 connections while B has 10,000 connections.

   Marking: One point for saying FALSE, and one point for a good counterexample (or, if the answer was TRUE, a justification that shows understanding). Half a point off if the answer uses convolution (since MLPs are fully connected).

   Mean: 1.56/2

4. **[3pts]** *In this question, we'll design a binary linear classifier to compute the NAND (not-AND) function. This function receives two binary-valued inputs x_1 and x_2, and returns* 0 *if both inputs are* 1*, and returns* 1 *otherwise.*

   (a) **[1pt]** *Give four constraints on the weights w_1 and w_2 and the bias b, i.e. one constraint for each of the 4 possible input configurations.*

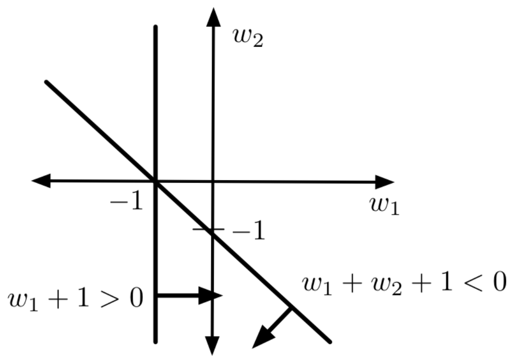
   $$\begin{aligned} b &> 0 \\ w_1 + b &> 0 \\ w_2 + b &> 0 \\ w_1 + w_2 + b &< 0 \end{aligned}$$

   Marking: Half a point off for each mistake. We accepted $\leq$ instead of $<$ even though the inequalities really should be strict.

   (b) **[2pts]** *Consider a slice of weight space corresponding to $b = 1$. Sketch the constraints in weight space corresponding to the two input configurations ($x_1 = 1, x_2 = 0$) and ($x_1 = 1, x_2 = 1$). (Make sure to indicate the half-spaces with arrows.)*

   Marking: 1 point for each of the two constraints.

   Mean: 2.57/3



5. **[3pts]** *Consider a layer of a multilayer perceptron which has ReLU activations:*

$$z_i = \sum_j w_{ij} x_j + b_i$$
$$h_i = \mathrm{ReLU}(z_i)$$

(a) **[2pts]** *Give the backprop rules for computing the error signals $\overline{z_i}$, $\overline{x_j}$ and $\overline{w_{ij}}$ in terms of the error signals $\overline{h_i}$.*

$$\overline{z_i} = \begin{cases} \overline{h_i} & \text{if } z_i > 0 \\ 0 & \text{if } z_i \leq 0 \end{cases}$$
$$\overline{x_j} = \sum_i \overline{z_i} w_{ij}$$
$$\overline{w_{ij}} = \overline{z_i} x_j$$

Marking: Half a point off for each mistake.

(b) **[1pt]** *Consider a pair of units (x_j, h_i). Based on your answer to part (a), for what values of x_j and z_i are we guaranteed that $\overline{w_{ij}} = 0$?*

Based on the above equations, we are guaranteed that $\overline{w_{ij}} = 0$ (for any value of $\overline{h_i}$) if and only if $z_i \leq 0$ or $x_j = 0$.

Marking: Half a point for each case. We didn't take off for writing $<$ instead of $\leq$.

Mean: 2.23/3

6. **[1pt]** *Recall that Autograd includes a module, `autograd.numpy`, which provides similar functionality to `numpy`, except that each of the functions does some additional bookkeeping needed for autodiff. Briefly explain one thing that `autograd.numpy.sum` does which `numpy.sum` does not.*

   The functions in `autograd.numpy.sum` are responsible for building the computation graph data structure to be used in backprop. The nodes in the graph are stored as `Node` instances, and the `Node`s have attributes for the value, the function that was executed, and the arguments to the function (i.e. parents in the computation graph). The function `autograd.numpy.sum` retrieves the values from its `Node` arguments, feeds them to `numpy.sum`, and packages the result into a `Node` class.

   Marking: Full credit will be given for giving enough of the above explanation to demonstrate an understanding of what `autograd.numpy.sum` is doing. Half credit for saying it "builds the computation graph" without giving more specifics.

   Mean: 0.51/1

7. **[1pt]** *Compute the convolution of the following two arrays:*

$$\begin{pmatrix} 4 & 1 & -1 & 3 \end{pmatrix} * \begin{pmatrix} -2 & 1 \end{pmatrix}$$

   Your answer should be an array of length 5. You do not need to show your work, but it may help you get partial credit.

$$\begin{pmatrix} -8 & 2 & 3 & -7 & 3 \end{pmatrix}$$

   Marking: No points off for obvious typos or arithmetic errors. Half credit for the following answers, which involve flipping-related mistakes:

$$\begin{pmatrix} 8 & -2 & -3 & 7 & -3 \end{pmatrix}$$
$$\begin{pmatrix} -4 & 7 & 3 & -5 & 6 \end{pmatrix}$$
$$\begin{pmatrix} 4 & -7 & -3 & 5 & -6 \end{pmatrix}$$

   Mean: 0.78/1

8. **[1pt]** *Both the neural probabilistic language model and n-gram language models make a Markov assumption. What is this Markov assumption? (One sentence should suffice.)*

The distribution for the next word is independent of the rest of the sentence given the last K words (where K is the context length).

Mean: 0.76/1

9. **[2pts]** *When we discussed resource constraints for neural nets, we noted that the activations need to be stored in memory at training time, but not at test time. For simplicity, focus on the case of a multilayer perceptron.*

   (a) *Why do the activations need to be stored at training time (in order to do backprop)?*

   The backprop equations require the activations, and we need to store them because we go backwards through the computation graph.

   (b) *Why don't they need to be stored at test time?*

   At test time, we only need to do the forward pass. After one layer's activations are used to compute the next layer's activations, they aren't used anymore, so we can get rid of them.

   Mean: 1.63/2

Night Section

1. **[2pts]** *When tuning hyperparameters, it's important to use a validation set.*

   - **[1pt]** *Briefly explain what can go wrong if you tune hyperparameters on the training set.*

     It will pick hyperparameters that perform the best on the training set, which doesn't account for overfitting. Hence, it will choose the most powerful model, the smallest weight decay parameter, etc.

   - **[1pt]** *Briefly explain what can go wrong if you tune them on the test set.*

     If the hyperparameters were tuned on the test set, then the test set performance won't be a realistic measure of how the model will generalize to new data.

   Mean: 1.61/2

2. **[1pt]** *Give the definition of a convex function. I.e., let f be a scalar-valued function that takes a vector $\mathbf{x}$ as input. The definition is that f is convex if and only if...*

A function f is convex if and only if $f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2)$ for any $\mathbf{x}_1$, $\mathbf{x}_2$, and $0 \leq \lambda \leq 1$.

Marking: Full credit for the alternative definition: $f(\lambda_1 \mathbf{x}_1 + \cdots + \lambda_N \mathbf{x}_N) \leq \lambda_1 f(\mathbf{x}_1) + \cdots + \lambda_N f(\mathbf{x}_N)$ for any $\{\mathbf{x}_i\}_{i=1}^N$ and $\{\lambda_i\}_{i=1}^N$ with each $\lambda_i \geq 0$ and $\sum_i \lambda_i = 1$. Half credit for the definition of a convex set. No credit for simply saying "bowl-shaped".

Mean: 0.57/1

3. **[3pts]** *Consider a layer of a multilayer perceptron with logistic activation function σ:*

$$z_i = \sum_j w_{ij} x_j + b_i \qquad\qquad h_i = \sigma(z_i)$$

The two layers have dimensions D_1 and D_2, respectively. Let M denote the mini-batch size. Recall the backprop equations for a multilayer perceptron:

$$\overline{z_i} = \sigma'(z_i) \circ \overline{h_i} \qquad\qquad \overline{x_j} = \sum_i \overline{z_i} w_{ij} \qquad\qquad \overline{w_{ij}} = \overline{z_i} x_j$$

Your job is to write NumPy code to implement the backward pass for this layer. Assume the following NumPy arrays have already been computed:

- *X, an $M \times D_1$ matrix representing the x_j values for a whole mini-batch. (I.e., each row contains the activations for one training example.)*
- *The matrices Z and H are defined analogously.*
- *W, a $D_2 \times D_1$ matrix representing the weights. I.e., the (i, j) entry of W is w_{ij}.*
- *H_bar, the matrix of error signals $\overline{h_i}$, the same size as H.*

Also, assume you are given a function **sigma_prime** *which computes the derivatives of the logistic function, elementwise.*

Write NumPy code that computes arrays Z_bar, X_bar, and W_bar, representing error signals for the corresponding variables. W_bar should be the average of the derivatives over the mini-batch. Your code should be vectorized, i.e. do not use **for** *loops. This can be done in three lines, one for each backprop equation.*

```
Z_bar = sigma_prime(Z) * H_bar
X_bar = np.dot(Z_bar, W)
W_bar = np.dot(Z_bar.T, X) / M
```

Marking: One point for each line. But each individual mistake is only 0.5 off (even if it's made in multiple lines).

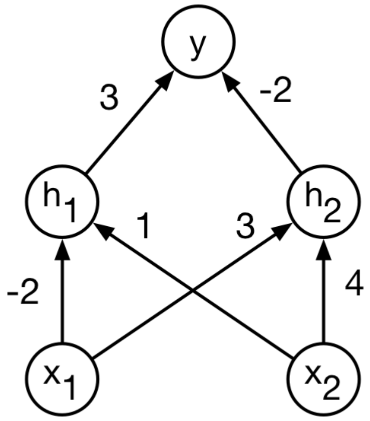
Mean: 1.67/3

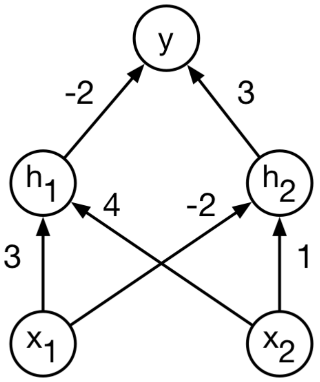
4. **[1pt]** *TRUE or FALSE: in logistic regression, if every training example is classified correctly, then the cost is zero. Briefly explain your answer.*

   FALSE. The model outputs a distribution over the categories, and due to the softmax, the probabilities will all be positive. Hence, the cross-entropy loss will be nonzero.

   Mean: 0.63/1

5. **[1pt]** *Consider the following multilayer perceptron shown on the left. All units use the logistic activation function, and there are no biases. On the right-hand diagram, give the weights for an equivalent network (i.e. one which computes the same function). You don't need to explain your answer.*





   Marking: It's clear that a lot of people were confused by the wording of this question, so we threw it out and gave everybody the point.

6. **[2pts]** *Briefly explain two reasons to use automatic differentiation rather than finite differences to compute the gradient for a neural network during training.*

   (a) Autodiff computes exact gradients (apart from floating point error), whereas finite differences only computes an approximation.

   (b) Autodiff only requires a single forward pass and a single backward pass, and the backward pass is only a constant factor more expensive than the forward pass. Finite differences requires a separate forward pass for every entry of the gradient.

Marking: Half a point for saying autodiff is already implemented by deep learning frameworks.

Mean: 1.46/2

7. **[1pt]** *Alice and Barbara are trying to redesign the LeNet conv net architecture to reduce the number of weights. Alice wants to reduce the number of feature maps in the first convolution layer. Barbara wants to reduce the number of hidden units in the last layer before the output. Whose approach is better? Why?*

   Barbara's approach is better because most of the weights are in the fully connected layers of LeNet (or a typical conv net architecture).

   Mean: 0.64/1

8. **[2pts]** *Recall the neural language model architecture from Assignment 1. Suppose there are 100 words in the dictionary, the embedding dimension is 30, the context length is 3 words, and the hidden layer has 60 units. You don't need to explain your answer, but doing so may help you get partial credit.*

   (a) **[1pt]** *How many parameters are needed for the embedding layer? (You may assume this layer has no biases.)*

   The parameters consist of a 30-dimensional embedding vector for each of the 100 words, or 3000 parameters total.

   (b) **[1pt]** *How many weights and biases are needed for the hidden layer?*

   The 3 representations of length 30 are concatenated into a vector of length 90. The hidden layer has 60 units. Since this is a fully connected layer, there are $90 \times 60 = 5400$ weights. There are 60 biases.

   Marking: Half a point off for assuming different embedding matrices for each context word.

   Mean: 1.31/2

9. **[2pts]** *Recall that a plateau is a flat region in the cost function. Give two examples of plateaux (plual of plateau) that can occur in neural net training, and briefly explain why they are plateaux.*

   Some examples:

   - If the loss function is flat (e.g. zero-one loss), then the gradient will be zero because changing the weights a small amount has no impact on the loss.

- In classification with a logistic output nonlinearity and squared error loss, the output derivative is small when the prediction is very wrong. Hence, the gradient will also be small.
- If the hard threshold activation function is used, the gradient will be zero because making a small change to the weights won't change the predictions (and hence won't change the cost).
- If the input z to the ReLU activation function is 0, then we have $\overline{z} = 0$, and hence the weights that feed into this unit have zero gradient. If this happens for every training example (i.e. this unit is dead), then these weights will have zero gradient and won't get updated.
- Similarly, if a logistic unit is saturated, the derivatives for the incoming weights will be very small.

Mean: 1.32/2