CS230: Deep Learning

Fall Quarter 2022

Stanford University

Midterm Examination

Suggested duration: 180 minutes

Problem	Full Points	Your Score
Multiple Choice	14	
Short Answer	30	
Feed-Forward Neural Network	15	
Backpropagation	19	
Discrete Functions in Neural Networks	11	
Debugging Code	18	
Total	88	

The exam contains 20 pages including this cover page.

- This exam is open book, but collaboration with anyone else, either in person or online, is strictly forbidden pursuant to The Stanford Honor Code.
- In all cases, and especially if you're stuck or unsure of your answers, **explain your work, including showing your calculations and derivations!** We'll give partial credit for good explanations of what you were trying to do.

Name: _____

SUNETID: _____@stanford.edu

The Stanford University Honor Code:
I attest that I have not given or received aid in this examination, and that I have done my share and taken an active part in seeing to it that others as well as myself uphold the spirit and letter of the Honor Code.

Signature: _____

Question (Multiple Choice, 14 points)

For each of the following questions, circle the letter of your choice. Each question has AT LEAST one correct option unless explicitly mentioned. No explanation is required.

(a) **(2 points)** Imagine you are tasked with building a model to diagnose COVID-19 using chest CT images. You are provided with 100,000 chest CT images, 1,000 of which are labelled. Which learning technique has the best chance of succeeding on this task? **(SELECT ONLY ONE)**

(i) Transfer Learning from a ResNet50 that was pre-trained on chest CT images to detect tumors

(ii) Train a GAN to generate synthetic labeled data and train your model on all the ground truth and synthetic data

(iii) Supervised Learning directly on the 1,000 labeled images

(iv) Augment the labeled data using random cropping and train using supervised learning

Solution: (i)

(i) The model was pre-trained on the same type of data, namely CT images. Therefore, transfer learning from such a model would make the most sense, despite the pre-training being done for a slightly different task.

(ii) GANs require a lot of data to train and note that in this case, we would have to train a GAN to generate both CT images with and without COVID-19. As we only have 1k labelled images, we can then only train separate GANs for our known COVID and non-COVID images, which we don't have enough of.

(iii) There wouldn't be enough data in this case, and your model would easily overfit to the 1k examples.

(iv) Although using random cropping can help, similar to (iii), there just isn't enough data to train a model in a fully supervised manner from scratch.

(b) **(2 points)** Imagine you are tasked with training a lane detection system that can detect between two different types of lanes: lanes in the same direction as the car moves and lanes in the opposite direction. Assume all the images are taken in common two-way streets in California from a car's front-view camera. What are the following data augmentation techniques can be used for your task?

(i) Flipping vertically (across x-axis)

(ii) Flipping horizontally (across y-axis)

(iii) Adding artificial fog to your images

(iv) Applying random masking to a (small) portion of the image

Solution: (iii), (iv)

(i) The images after flipping vertically do not make sense anymore, ie, your car will never be driving upside-down

(ii) The images after flipping horizontally breaks the directions of traffic as lanes in the opposite direction should always appear on the left side of the image

(iii) Adding artificial fog to the images is a good technique to apply in this case, as it can simulate more realistic driving conditions

(iv) Applying random masking is a good data augmentation technique to use here to cover the cases where there are obstacles on the road that occlude your vision.

(c) **(2 points)** You are training a binary classifier and are unsatisfied with the F1-score as a good metric to combine precision and recall into a single number. You are considering alternatives to F1-score. Which of the following would be reasonable candidate metric(s):

(i) |precision - recall|

(ii) recall/precision

(iii) precision $\times$ recall

(iv) max(precision, recall)

Solution: (iii)

(i) Optimizes for gap between precision and recall

(ii) Maximizes recall at the expense of precision

(iii) Is proportional to the geometric mean of precision and recall

(iv) Only focuses on the better metric, e.g. can not discriminate between (0.9, 0.4) and (0.9, 0.7).

(d) **(2 points)** Dropout can be considered as a form of ensembling over variants of a neural network. Consider a neural network with N nodes, each of which can be dropped during training independently with a probability $0 < p < 1$. What is the total number of unique models that can be realized on applying dropout?

(i) $\lfloor N \times p \rfloor$

(ii) $(\lfloor N \times p \rfloor)^N$

(iii) $2^{\lfloor N \times p \rfloor}$

(iv) 2^N

Solution: (iv)
Each node has 2 possibilities: to be kept or to be dropped, and we have N nodes total. Note that this is independent of p, as we ask for the *total* number of unique models, not the expected value.

(e) **(2 points)** In practice when using Early Stopping, one needs to set a "buffer" hyperparameter, which determines the number of epochs model training continues when no improvement in validation performance is observed before training is terminated. After training is terminated, the model with the best validation performance is used. What is the benefit of setting the buffer parameter to a value $k = 5$ epochs instead of 0:

(i) Robustness to noise in validation performance from epoch to epoch

(ii) Reduced training time on average

(iii) Reduced inference time on average

(iv) None of the above

Solution: (i)
In real, especially smaller datasets, validation performance can exhibit small fluctuations from epoch to epoch. In such a case, using a buffer parameter (typically referred to as "patience") of 0 increases the chances of prematurely stopping training when validation performance in a given epoch does not improve due to random fluctuation. Instead, setting patience to a value such as $k = 5$ epochs is more likely to ensure that we are not underfitting.

(f) **(2 points)** Suppose that you are training a deep neural network and observe that the training curve contains a lot of oscillations, especially at early stages of training. Which of the following techniques can help stabilize training?

(i) Early stopping

(ii) Learning rate scheduling

(iii) Data augmentation

(iv) Gradient clipping

Solution: (ii), (iv)

(i) can help stabilize training at the beginning stages when there is high uncertainty

(ii) helps reduce overfitting, not necessarily the training stability

(iii) constrains the maximum magnitude for gradients, and hence, the step size

(iv) incorrect, as you are simply stopping training early

(g) **(2 points)** You have a 2-layer MLP with Sigmoid activations in the hidden layers that you want to train with SGD. Your network weights are initialized from $\mathcal{N}(10, 1)$. From the very first epoch, you observe that some weights in the first layer are not getting updated or are updated very slowly compared to the second layer. Which of the following can fix this issue?

(i) Initializing the weights to be from $\mathcal{N}(0, 1)$

(ii) Adding more hidden layers

(iii) Switching the activation function to tanh

(iv) Switching the activation function to leaky ReLU

Solution: (i) and (iv) will help. Initializing the weights to very high values will cause large pre-activations causing sigmoid to output close to 1 and the derivative will be almost 0. Tanh has a similar problem as well. Switching to leaky ReLU or initializing the weights to smaller values can help.

Question (Short Answer, 30 points)

The questions in this section can be answered in less than 3 sentences. Please be concise in your responses.

(a) Imagine that you are building an app to optimize wait times in US emergency rooms while prioritizing severe cases. You build a deep learning-based app that works as follows:

- **Input:** a patient's demographic information (i.e, ethnicity, age), health history and reasons for emergency
- **Output:** ranking of patients currently in the emergency room from most to least severe

You trained and tested your model using 3 months worth of data from hospitals in the US, before deploying it to several hospitals in the San Francisco Bay Area.

(i) **(2 points)** Now you want to deploy your app internationally. Do you think your app will work well? Why or why not?

Solution: No, as the model is trained on US data only, it is heavily biased to American citizens and the same model cannot be directly applied to users in other countries where the data distribution is different.

You noticed that the app tends to rank African American and Hispanic patients lower than patients from other ethnic backgrounds, even if those patients came into the emergency department with more severe cases.

(ii) **(1 point)** Why is this a problem?

Solution: Many answers accepted.
Example solution: This is a problem because the model is perpetuating a racial bias that hurts specific populations.

(iii) **(2 points)** What may have caused this problem?
Hint: Think about how the model was trained and the input data that was provided

Solution: Many answers accepted.
Example solution: The training data may have been taken from hospitals that treat other populations over African American and Hispanic populations. As a result, the model may have learned to rank these populations lower.

(iv) **(2 points)** How can we fix this problem?

Solution: Many answers are accepted.
Example solutions: model de-biasing, re-training the model with less biased data, remove "ethnicity" as an input to the model

(b) Graph Neural Networks (GNNs) are a family of neural networks that can operate on graph-structured data. Here, we describe a basic 2-layer GNN. Consider a graph with k nodes labeled $\{1, 2, \ldots, k\}$. For simplicity, assume that each node i is associated with a scalar input x_i. The first layer of our GNN, parameterized by scalar parameters $w^{[1]}$ and $b^{[1]}$ performs the following operation to compute $a_i^{[1]}$ at each node i:

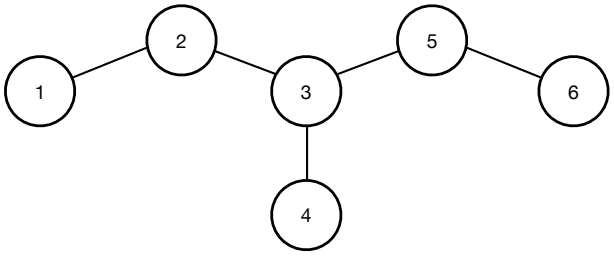
$$a_i^{[1]} = \text{ReLU}\left(x_i + w^{[1]}\left(\sum_{n \in N(i)} x_n\right) + b^{[1]}\right) \tag{1}$$

where $N(i)$ is the set of neighbors of node i in the graph (i.e, all nodes that share an edge with node i). The second layer, parameterized by scalar parameters $w^{[2]}$ and $b^{[2]}$, analogously computes $a_i^{[2]}$ for each node i:

$$a_i^{[2]} = \text{ReLU}\left(a_i^{[1]} + w^{[2]}\left(\sum_{n \in N(i)} a_n^{[1]}\right) + b^{[2]}\right) \tag{2}$$

Answer the following questions for the graph in the figure below, with labels as shown in the nodes.

(i) **(2 points)** What is $\partial a_1^{[2]} / \partial x_6$?



Solution: 0. After the k^{th} GNN layer, each node assimiliates information from nodes up to k hops away. Since nodes 1 and 6 are more than 2 hops away, the output of the 2nd GNN layer for node 1 does not depend on the input value for node 6.

(ii) **(2 points)** You are allowed to add one additional node (suppose this is node 7) and accompanying edges such that the value of $\partial a_1^{[2]}/\partial x_6$ changes from the value computed in part (i). Describe how you would do this with fewest number of edges accompanying node 7.

Solution: Node 7 would have edges connecting it to nodes 1 and 6. This brings nodes 1 and 6 within 2 hops of each other.

(c) Consider the graph in figure below representing the training procedure of a GAN. The figure shows the cost function of the generator plotted against the output of the discriminator when given a generated image $G(z)$. Concerning the discriminator's output, we consider that 0 means that the discriminator thinks the input "has been generated by G", whereas 1 means the discriminator thinks the input "comes from the real data".

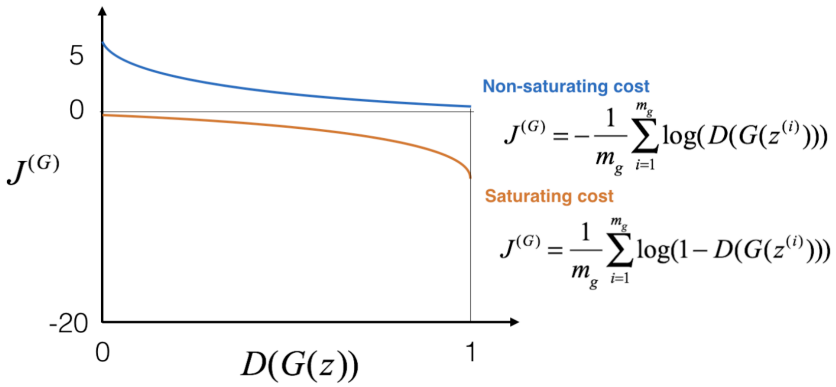


Figure 1: GAN training curve

(i) **(2 points)** After one round of training the generator and discriminator, is the value of $D(G(z))$ closer to 0 or closer to 1? Explain.

Solution: Closer to 0. This is because the generator is not well-trained and the discriminator can easily separate real data from generated data.

(ii) **(2 points)** Two cost functions are presented in Figure 1 above. Which one would you choose to train your GAN? Justify your answer.

Solution: Non-saturating cost, as the gradient is the highest when the cost is largest.

(iii) **(2 points)** True or false. Your GAN is finished training when $D(G(z))$ is close to 1. Please explain your answer for full credit.

Solution: False. For a well-trained generator, the discriminator should not be able to discriminate between real and generated examples, and so $D(G(z))$ should be closer to 0.5.

(d) We would like to train a self-supervised generative model that can learn encodings z of a given input image X by reconstructing the same input image as $\hat{X}$. For our example, lets say our input images are MNIST digits. Consider the architecture shown below:

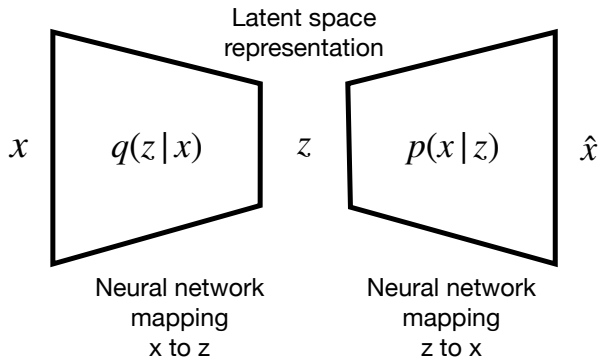


Figure 2: Architecture of proposed generative model

Assume the encoder $q(z \mid x)$ is parameterized to output a normal distribution over z. Alice, Bob and Carol propose 3 different loss functions to train this model end-to-end.

- **Alice:** $\mathbb{KL}(q(z \mid x) \,||\, \mathcal{N}(0, \mathrm{I}))$
- **Bob:** $\mathrm{MSE}(X - \hat{X}) + \mathbb{KL}(q(z \mid x) \,||\, \mathcal{N}(0, \mathrm{I}))$
- **Carol:** $\mathrm{MSE}(X - \hat{X})$

The entire network is end-to-end differentiable for all 3 loss functions.

Here $\mathbb{KL}$ is the KL-divergence which is a measure of similarity of two different probability distributions. $\mathcal{N}(0, \mathrm{I})$ is the multivariate standard Normal distribution where I is the identity matrix. MSE is the mean squared error.

(i) **(3 points)** In plain English, intuitively, explain what each loss function is trying to optimize.

Solution:
Alice: Wants $q(z \mid x)$ to be as close to $\mathcal{N}(0, 1)$ as possible. This is because we want to keep the distribution of encodings to be relatively spread-out, so that most of the latent-space is "covered" by a digit.
Bob: Wants the same as Alice, but also want to minimize reconstruction error (wants X and $\hat{X}$ to be as similar as possible)
Carol: Wants to minimize reconstruction error of X

(ii) **(3 points)** Say we choose the dimension of z to be 2 so we can plot the z's on a graph. Consider the three graphs below where each of the two axes is a dimension of z. The different colours indicate different MNIST digits as indicated by the legend. The plots are numbered left to right as (1), (2) and (3).

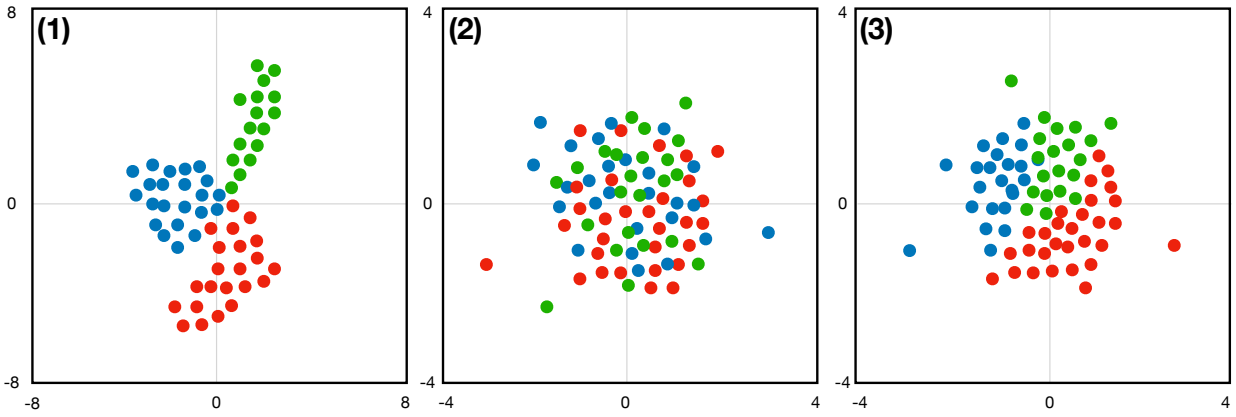


Figure 3: Plotted graphs for different loss functions. Plots are numbered, left to right as (1), (2) and (3).

Match each graph to Alice, Bob and Carol (draw lines connecting the two columns if you printed the midterm) and explain your reasoning for each.

Alice	**(1)**
Bob	**(2)**
Carol	**(3)**

Solution:
Alice: 2.

- This is clearly a plot of a $\mathcal{N}(0, 1)$ distribution along both axes (most of the points, you could say around 95% of them, are quite clustered in a circle at the center where mean = 0 with 5% of the points that are slightly more spread out)
- Notice that there is no separation between digits, as there is nothing in

the loss function that enforces that separation

Bob: 3.

- The overall shape still resembles plot 2, where most of the points are clustered in the center where mean = 0 and some other points are spread farther away, indicating that there is a KL divergence term in the loss.
- We also see separation of the different digits, which is caused by the reconstruction error, as similarly-looking digits would have lower reconstruction error and thus, our loss groups those digits together

Carol: 1.

- We evidently see that there is separation between the classes, which is due to the reconstruction error
- We also see that the points do not appear to be $N(0, 1)$ distributed (you can also see from the axes that the scale is a lot larger).

Question (Backpropagation, 19 points)

Consider the following neural network with arbitrary dimensions (ie, $\mathbf{x}$ is not necessarily 5-dimensional, etc.):

$$\begin{aligned}
\mathbf{z}^{[1]} &= \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]} \\
\mathbf{h} &= \text{ReLU}(\mathbf{z}^{[1]}) \\
\mathbf{z}^{[2]} &= \mathbf{W}^{[2]}\mathbf{h} + \mathbf{b}^{[2]} \\
\hat{\mathbf{y}} &= \sigma(\mathbf{z}^{[2]}) \\
\mathcal{L} &= \sum_{i=1}^{k} \max(0, 1 - \mathbf{y}_i \hat{\mathbf{y}}_i)
\end{aligned}$$

where σ is the sigmoid activation function, and $\odot$ is the operator for element-wise products, and $\mathbf{y}$ is a k-dimensional vector of 1's and 0's. Note that $\mathbf{y}_i$ represents the i-th element of vector $\mathbf{y}$, and likewise for $\hat{\mathbf{y}}_i$.

(i) **(3 points)** What is $\partial\mathcal{L}/\partial\hat{\mathbf{y}}_i$? You must write the most reduced form to get full credit.

Solution:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}_i} &= \frac{\partial}{\partial \hat{\mathbf{y}}_i} \max(0, 1 - \mathbf{y}_i \hat{\mathbf{y}}_i) \\
&= \begin{cases} 0, & \text{if } 1 - \mathbf{y}_i \hat{\mathbf{y}}_i < 0 \\ -\mathbf{y}_i, & \text{if } 1 - \mathbf{y}_i \hat{\mathbf{y}}_i \geq 0 \end{cases} \\
&= -\mathbf{y}_i \qquad \text{(refer to the note below)}
\end{aligned}$$

Notice that because we use the sigmoid activation function and $\mathbf{y}$ is a vector of 1's and 0's, $0 \leq \mathbf{y}_i \hat{\mathbf{y}}_i \leq 1$. Therefore, $0 \leq 1 - \mathbf{y}_i \hat{\mathbf{y}}_i \leq 1$. Hence, the first case cannot ever happen.

(ii) **(2 points)** What is $\partial\mathcal{L}/\partial\hat{\mathbf{y}}$? Refer to this result as $\overline{\hat{\mathbf{y}}}$. Please write your answer according to the shape convention, i.e., your result should be the same shape as $\hat{\mathbf{y}}$.

Solution:

$$\overline{\hat{\mathbf{y}}} = -\mathbf{y}$$

(iii) **(2 points)** What is $\partial\mathcal{L}/\partial\mathbf{z}^{[2]}$? Refer to this result as $\overline{\mathbf{z}^{[2]}}$. To receive full credit, your answer must include $\overline{\hat{\mathbf{y}}}$ and your answer must be in the most reduced form.

Solution:

$$\overline{\mathbf{z}^{[2]}} = \overline{\hat{\mathbf{y}}} \odot \hat{\mathbf{y}} \odot \left(1 - \hat{\mathbf{y}}\right)$$

(iv) **(2 points)** What is $\partial \mathcal{L} / \partial \mathbf{W}^{[2]}$? Please refer to this result as $\overline{\mathbf{W}^{[2]}}$. Please include $\overline{\mathbf{z}^{[2]}}$ in your answer.

Solution:

$$\overline{\mathbf{W}^{[2]}} = \overline{\mathbf{z}^{[2]}} \mathbf{h}^\top$$

(v) **(2 point)** What is $\partial \mathcal{L} / \partial \mathbf{b}^{[2]}$? Please refer to this result as $\overline{\mathbf{b}^{[2]}}$. Please include $\overline{\mathbf{z}^{[2]}}$ in your answer.

Solution:

$$\overline{\mathbf{b}^{[2]}} = \overline{\mathbf{z}^{[2]}}$$

(vi) **(2 points)** What is $\partial \mathcal{L} / \partial \mathbf{h}$? Please refer to this result as $\overline{\mathbf{h}}$. Please include $\overline{\mathbf{z}^{[2]}}$ in your answer.

Solution:

$$\overline{\mathbf{h}} = W^{[2]\top} \overline{\mathbf{z}^{[2]}}$$

(vii) **(2 points)** What is $\partial \mathcal{L} / \partial \mathbf{z}^{[1]}$? Refer to this result as $\overline{\mathbf{z}^{[1]}}$. Please include $\overline{\mathbf{h}}$ in your answer.

Solution:

$$\overline{\mathbf{z}^{[1]}} = \overline{\mathbf{h}} \odot \delta$$

where δ is a vector of same size as $\mathbf{z}^{[1]}$ where $\delta_i = \begin{cases} 1, \mathbf{z}_i^{[1]} > 0 \\ 0, \text{otherwise} \end{cases}$.

(viii) **(2 point)** What is $\partial \mathcal{L} / \partial \mathbf{W}^{[1]}$? Please refer to this result as $\overline{\mathbf{W}^{[1]}}$. Please include $\overline{\mathbf{z}^{[1]}}$ in your answer.

Solution:

$$\overline{\mathbf{W}^{[1]}} = \overline{\mathbf{z}^{[1]}} \mathbf{x}^\top$$

(ix) **(2 point)** What is $\partial\mathcal{L}/\partial\mathbf{b}^{[1]}$? Please refer to this result as $\overline{\mathbf{b}^{[1]}}$. Please include $\overline{\mathbf{z}^{[1]}}$ in your answer.

Solution:

$$\overline{\mathbf{b}^{[1]}} = \overline{\mathbf{z}^{[1]}}$$

Question (Discrete Functions in Neural Networks, 11 points)

In this problem, we will explore training neural networks with discrete functions. Consider a neural network encoder $\mathbf{z} = \text{softmax}[f_\theta(\mathbf{X})]$. You can think of f_θ as an MLP for this example. $\mathbf{z}$ is the softmax output and we want to discretize this output into a one-hot representation before we pass it into the next layer. Consider the operation `one_hot` where `one_hot`(z) returns a one-hot vector where the 1 is at the argmax location. For example, `one_hot`$([0.1, 0.5, 0.4]) = [0, 1, 0]$. Say we want to pass this output to another FC layer g_ϕ to get a final output y.

(i) **(1 points)** Is there a problem with the neural network defined below?

$$y = g(\text{one\_hot}(\text{softmax}(f(\mathbf{X}))))$$

Solution: Yes, `one_hot` is not a differentiable operation.

(ii) **(2 points)** Consider the following function:

$$z = S_\tau(f(\mathbf{X})) = \text{softmax}(f(\mathbf{X})/\tau)$$

Here dividing by τ means every element in the vector is divided by τ. Obviously, when $\tau = 1$, this is exactly the same as the regular softmax function. What happens when $\tau \to \infty$? What happens when $\tau \to 0$?
Hint: You don't need to prove these limits, just showing a trend and justifying is good enough.

Solution: As $\tau \to \infty$, we get a uniform distribution. As $\tau \to 0$, we get the one-hot vector at the argmax location.

(iii) **(4 points)** Assume $f(\mathbf{X}) = \mathbf{w}^\top \mathbf{X}$ where $\mathbf{w}$ is a weight vector. What is the derivative of $S_\tau(f(\mathbf{X}))_i$ with respect to $\mathbf{w}$ for a fixed τ? In other words, what is $\partial S_\tau(\mathbf{w}^\top \mathbf{X})_i / \partial \mathbf{w}$, the derivative of the i-th element of $S_\tau(\mathbf{w}^\top \mathbf{X})$ with respect to $\mathbf{w}$? You must write your answer in the most reduced form to receive full credit.

Solution:

$$\begin{aligned}
\frac{\partial S_\tau(\mathbf{w}^\top \mathbf{X})_i}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}} \text{softmax}(\mathbf{w}^\top \mathbf{X}/\tau)_i \\
&= \frac{\partial}{\partial \mathbf{w}} \frac{\exp(\mathbf{w}^\top \mathbf{X}_i/\tau)}{\sum_j \exp(\mathbf{w}^\top \mathbf{X}_j/\tau)} \\
&= \frac{\mathbf{X}_i(\exp(\mathbf{w}^\top \mathbf{X}_i/\tau)/\tau)(\sum_j \exp(\mathbf{w}^\top \mathbf{X}_j/\tau)) - \exp(\mathbf{w}^\top \mathbf{X}_i/\tau)[\sum_j \mathbf{X}_j \exp(\mathbf{w}^\top \mathbf{X}_j/\tau)/\tau]}{(\sum_j \exp(\mathbf{w}^\top \mathbf{X}_j/\tau))^2} \\
&= \frac{1}{\tau}\Big(\mathbf{X}_i S_\tau(\mathbf{w}^\top \mathbf{X})_i - S_\tau(\mathbf{w}^\top \mathbf{X})_i \sum_j \mathbf{X}_j S_\tau(\mathbf{w}^\top \mathbf{X})_j\Big) \\
&= \frac{1}{\tau} S_\tau(\mathbf{w}^\top \mathbf{X})_i \Big(\mathbf{X}_i - \sum_j \mathbf{X}_j S_\tau(\mathbf{w}^\top \mathbf{X})_j\Big)
\end{aligned}$$

(iv) **(2 points)** How can we use this modified softmax function S to get discrete vectors in our neural networks? Perhaps we cannot get perfect one-hot vectors but can we get close?

Solution: Replace softmax with S. S is differentiable. No need for `one_hot` anymore. With low τ, this should give us basically one-hot vectors.

(v) **(2 points)** What problems could arise by setting τ to very low values?

Solution: High values means high variance, which means that the network will be difficult to train, i.e instability.

Question (Debugging Code, 18 points)

Consider the pseudocode below for an MLP model to perform regression. The model takes an input of dim 10, hidden layer of size 20 with ReLU activations and outputs a real number. There are biases in both layers.
Weights are initialized from the random normal distribution and biases to 0.
Point out the errors in the code with line numbers and suggest fixes to them.
Your fixes should suggest code changes and not just English descriptions. Functions/classes that are not implemented completely can be assumed to be correctly written and have no errors in them.

```
1  import numpy as np
2
3  def mse_loss(predictions, targets):
4      """
5      Returns the Mean Squared Error Loss given the
6      predictions and targets
7
8      Args:
9          predictions (np.ndarray): Model predictions
10         targets (np.ndarray): True outputs
11
12     Returns:
13         Mean squared error loss between predictions and targets
14     """
15     return 0.5 * \
16             (predictions.reshape(-1) - targets.reshape(-1))**2
17
18
19 def dropout(x, p=0.1):
20     """
21     Applies dropout on the input x with a drop
22     probability of p
23
24     Args:
25         x (np.ndarray): 2D array input
26         p (float): dropout probability)
27
28     Returns:
29         Array with values dropped out
30     """
31     ind = np.random.choice(x.shape[1]*x.shape[0], replace=False,
32                            size=int(x.shape[1]*x.shape[0]*p))
33     x[np.unravel_index(indices, x.shape)] = 0
34     return x / p
```

```
35
36
37 def get_grads(loss, w1, b1, w2, b2):
38     """
39     This function takes the loss and returns the gradients
40     for the weights and biases
41     YOU MAY ASSUME THIS FUNCTION HAS NO ERRORS
42     """
43     ...
44     return dw1, db1, dw2, db2
45
46 def sample_batches(data, batchsize):
47     """
48     This function samples of batches of size `batchsize`
49     from the training data.
50     YOU MAY ASSUME THIS FUNCTION HAS NO ERRORS
51     """
52     ...
53     return x, y
54
55 class Adam:
56     """
57     The class for the Adam optimizer that
58     accepts the parameters and updates them.
59     YOU MAY ASSUME THIS CLASS AND ITS METHODS HAVE
60     NO ERRORS
61     """
62     def __init__(self, w1, b1, w2, b2):
63         ...
64
65     def update(self):
66         """
67         Updates the params according to the
68         Adam update rule
69         """
70         ...
71
72 class MLP:
73     """
74     MLP Model to perform regression
75     """
76     def __init__(self):
77         super().__init__()
78         self.w1 = np.random.randn(10, 20)
79         self.b1 = np.zeros(10)
```

```
        self.w2 = np.random.randn(20, 1)
        self.b2 = np.zeros(20)
        self.optimizer = Adam(w1, b1, w2, b2)

    def forward(self, x):
        """
        Forward pass for the model

        Args:
        x (np.ndarray): Input of shape batchsize x 10

        Returns:
        out (np.ndarray): Output of shape batchsize x 1
        """
        x = self.w1 * x + b1
        x = dropout(x)
        x = self.w2 * x + b2
        return x


    def train(self, training_data, test_data):
        """
        This method trains the neural network and outputs
        predictions for the test_data

        Args:
        training_data (np.ndarray):
                Training data containing (x, y) pairs
                x is 10-dimensional and y is 1-dimensional
        test_data (np.ndarray): 100 test points of shape
                                    (100, 10)

        Returns:
        predictions (np.ndarray): The predictions for
                            the 100 test points.
                            Final shape is (100,1)
        """
        batchsize = 32
        for _ in range(num_epochs):
            for x, y in sample_batches(training_data, batchsize):
                # Shape of x is (32, 10) and y is (32, 1)
                out = self.forward(x)
                loss = mse_loss(x, y)
                dw1, db1, dw2, db2 = get_grads(loss, self.w1,
                                                self.b1, self.w2,
```

```
                                                    self.b2)
                self.optimizer.update()

        # Assume test_data is of shape (100, 10)
        predictions = self.forward(test_data)

        return predictions

```

Solution:
Line 15: Add `np.mean`
Line 19: Update the `dropout` function to take a `training` argument and only apply dropout when `training=True`
Line 34: Should be `x / (1 - p)`
Lines 78,80: Bias shapes should be 20 and 1 respectively
Lines 93, 95: Need to do matrix multiplication, not hadamard product
Line 93: Missing ReLU activation
Line 121: `loss = mse_loss(out, y)`

END OF PAPER