ECE-GY 6143, Spring 2020

Final Exam

- Total duration: 120 minutes.
- You **can** use one double-sided page as a cheat sheet.
- You **cannot** consult your notes, textbooks, Google, Chegg, your classmates, or any other form of external help.
- Maximum points: 60. Any score above 60 will be rounded to 60.
- Once you are finished, please scan, or take a picture of, your answers and upload on NYUClasses before 4pm ET. You will have to include your cheat sheet, if you used one.
- **No late submissions will be accepted**.
- Good luck and stay safe!

---

1. **(1 point)** Please write down the time at the *start* and *end* of your exam. The difference should not exceed 120 minutes. Please also write down your *name*, *netid*, and *signature*. By doing so, you affirm the NYU Tandon School of Engineering student code of conduct, and that you have faithfully abided by the rules of this exam.

Name: Haotian Yi

netid: hy1651

signature: Haotian Yi

start: 10:40

end: 12:25

Your NetID is a combination of your initials and a few digits.

- If your netid **ends** with a 0, 1, 2, 3, or 4, answer **only** Part I (Questions 2-8).
- If your netid **ends** with a 5, 6, 7, 8, or 9, answer **only** Part II (Questions 9-15).

Part I

2. **(5 points)** Mark the following statements as either true or false, and briefly (in 1 sentence or less) justify your reasoning.

   a. Gradient descent is capable of finding the minimum of any differentiable function.

   False. Because if one function is not convex, it has no minimum; or if function has one or more local minimum, Gradient Descent may be stuck at local minimum.

   b. The training time of kernel SVMs is higher than the training time of k-nearest neighbor classifiers.

   True. k-nearest neighbour classifier actually just compare distance of each train set point with test point, it does not do much computation in training process. training time O(1) (load data into training set in python)

   c. If a recurrent neural network is unrolled up to T time steps, then the number of independent weights in the unrolled network increases by a factor of T.

   False, weights in RNN are constant and they do not vary, (weight sharing).

   d. The principal component directions obtained from PCA are always perpendicular to each other.

   True, they are orthogonal.

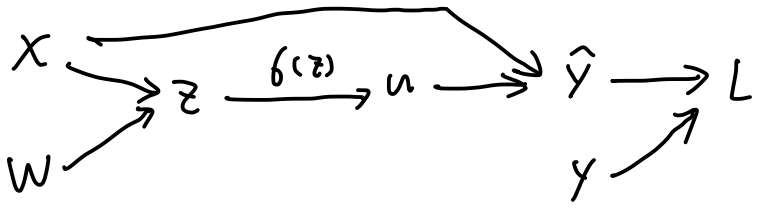
   e. In Reinforcement Learning, it is possible to approximate derivatives of the loss without explicit access to the gradient of the reward.

   True. Gradient of the reward is converted to another related quantity.

3. **(10 points)** Consider a 1-hidden layer *residual* neural network (ResNet) with sigmoid activation trained with squared error loss:

$$\hat{y} = x + \sigma(Wx), \quad \mathcal{L} = \|\hat{y} - y\|_2^2.$$

a. Draw the computational graph for this neural network.



b. Derive the forward and backward passes for backpropagation in this network.

Forward:

$z = XW$

$u = \sigma(z)$

$\hat{y} = x + u$

$L = (\hat{y} - y)^2$

Backward:

$\partial_L L = 1$

$\partial_{\hat{y}} L = 2(\hat{y} - y)$

$\partial_u \hat{y} = 1$

$\partial_z u = \sigma'(z) = \sigma(z)(1 - \sigma(z)) = u(1-u)$

$\partial_W z = X$

thus: $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial u} \cdot \frac{\partial u}{\partial z} \cdot \frac{\partial z}{\partial W} = 2(\hat{y} - y) \cdot u(1-u) \cdot X$

$= 2(\hat{y} - y) \cdot \sigma(z)(1 - \sigma(z)) \cdot X$

4. **(5 points)** The following represents python code for an algorithm we discussed in class. Assume that X is a data matrix and C is a matrix that has been randomly initialized.

   a. Identify the algorithm.

   b. Does this algorithm always terminate in a reasonable amount of time? If yes, justify why. If no, justify why.

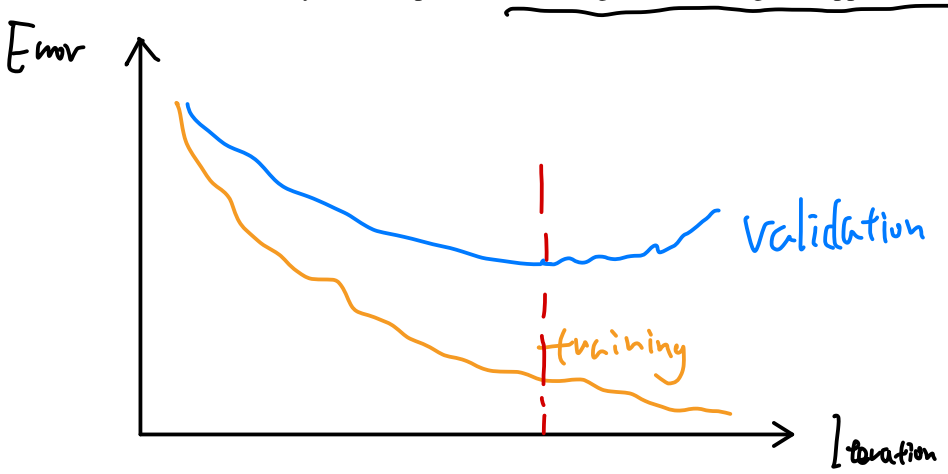
a. this is k-means algorithm

b. Yes. Because k-nearest algorithm only decrease the value of $F = \sum_{j=1}^{k} \sum_{x_i \in S_j} \|x_i - m_j\|^2$, it can't infinitely iterate which means F can decrease infinitely. And there are at most k^n possible configuration for points & corresponding centers, each configuration won't be revisit due to property above, so it has $O(k^n)$ for iteration.

```
def dist(a, b):
    return np.linalg.norm(a - b)

C_old = np.zeros(C.shape)

clusters = np.zeros(len(X))
error = dist(np.matrix.flatten(C), np.matrix.flatten(C_old))

while error != 0:
    for i in range(len(X)):
        distances = dist(X[i], C)
        cluster = np.argmin(distances)
        clusters[i] = cluster

    C_old = copy.copy(C)

    for i in range(k):
        points = [X[j] for j in range(len(X)) if clusters[j] == i]
        C[i] = np.mean(points, axis=0)

    error = dist(C, C_old, None)
```

5. **(10 points)** This is a three-part question.

   a. Consider neural network learning, and sketch typical curves of loss versus iteration count for the training and validation sets. (Assume for your example that overfitting to the training set happens at some point.)

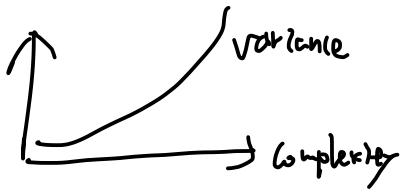


   b. Name two different strategies for preventing overfitting in neural network learning.

① use ridge regression (with L2 norm regularizer)

② PCA : reduce dimension and model complexity

because : complexity ↑ ⇒ variance ↑ (overfitting)



   c. Suppose that your training loss does not seem to ever decrease below a certain value, and you want to redo the learning. What fundamental change would you make to your network in order to increase your chances of getting lower training loss?

This may be because of imbalance between parameters and dataset. Maybe we can do dataset augmentation to increase amount of parameter to get a relatively more complex model so we can reduce bias to get lower training loss.

6. **(10 points)** In weighted linear regression, the goal is to minimize a loss function of the form:

$$L(w) = \frac{1}{2}\sum_{i=1}^{n} \beta_i (y_i - \langle x_i, w\rangle)^2.$$

where x_i are d-dimensional input data, y_i are scalar labels, and β_i are positive weights.

a. Show that the above loss function can be written in matrix notation:

$$L(W) = \frac{1}{2}(y - Xw)^T B (y - Xw).$$

Specify the sizes and contents of the variables y, X, B.

(assume zero mean)

sizes:

(1) y : $n \times 1$

(2) X : $n \times d$

(3) β : $n \times n$

$\frac{1}{2}\sum_{i=1}^{n}(y_i - \langle x_i, w\rangle)^2$ could be rewrite as $\frac{1}{2}\|y - Xw\|_2^2$, y's size $n\times 1$, X's size: $n \times d$, and $(y-Xw)$ of size of $n \times 1$. I want $L(w)$ transformed into the form of L2-norm. so $L(w) = \frac{1}{2}\sum_{i=1}^{n}\left[\sqrt{\beta_i}(y_i - \langle x_i, w\rangle)\right]^2$

$\Rightarrow L(w) = \frac{1}{2}\|b(y - Xw)\|_2^2$, $b = \begin{bmatrix} \sqrt{\beta_1} & & \\ & \sqrt{\beta_2} & \\ & & \ddots \\ & & & \sqrt{\beta_n} \end{bmatrix}$ (size: $n \times n$),

so $L(w) = \frac{1}{2}[b(y-Xw)]^T[b(y-Xw)] = (y-Xw)^T b^T b (y-Xw)$

$= (y-Xw)^T \beta (y-Xw)$, and $\beta = \begin{bmatrix} \beta_1 & & \\ & \beta_2 & \\ & & \ddots \\ & & & \beta_n \end{bmatrix}$

b. Suppose the conditional distribution of the data and labels follow a Gaussian model:

$$p(y_i|x_i) \propto \exp\left(-\frac{(y_i - \langle x_i, w\rangle)^2}{2\sigma_i^2}\right)$$

We would like to learn the parameters from n independent samples from the above distribution. Show that the maximum likelihood estimate can be written in the form of weighted linear regression of the form in part (a) and explicitly write down β_i in terms of σ_i.

$P(y|X) = \prod_{i=1}^{n} \alpha \exp\left(-\frac{(y_i - \langle x_i, w\rangle)^2}{2\sigma_i^2}\right)$ multiply with coefficient α to reflect proportional relationship: $P(y_i|X_i) \propto \exp(\cdots)$

do "$-\log$" to get Loss function

$L = -\log\left[\prod_{i=1}^{n} \alpha \exp\left(-\frac{(y_i - \langle x_i, w\rangle)^2}{2\sigma_i^2}\right)\right]$

$= -n\log k + \sum_{i=1}^{n} \frac{(y_i - \langle x_i, w\rangle)^2}{2\sigma_i^2} = \sum_{i=1}^{n} \frac{1}{2\sigma_i^2}(y_i - \langle x_i, w\rangle)^2 - n\log k$

$= \frac{1}{2}\sum_{i=1}^{n} \frac{1}{\sigma_i^2}(y_i - \langle x_i, w\rangle)^2 - n\log k$ ($-n\log k$ is an constant, actually we can ignore it)

SO $\frac{1}{\sigma_i^2}$ is like β_i in part (a)

7. **(10 points)** Suppose that a convolutional layer of a neural network has an input tensor $X[i, j, k]$ and computes an output as follows:

$$Z[i, j, m] = \sum_{k_1} \sum_{k_2} \sum_{n} W[k_1, k_2, n, m] X[i + k_1, j + k_2, n] + b[m]$$
$$Y[i, j, m] = \text{ReLU}(Z[i, j, m])$$

for some kernel W and bias b. Suppose X and W have shapes $(72, 96, 3)$ and $(5, 5, 3, 10)$ respectively.

a. What are the shapes of Z and Y?

b. What are the number of input and output channels?

c. How many multiply- and add- operations are required to perform a forward pass through this layer? Rough calculations are OK.

a. sizes:
Z: (68, 92, 10) Y: (68, 92, 10)

b. # input channel : 3, output channel: 10

c. $68 \times 92 \times 10 \times 5 \times 5 \times 3 = 4692000$ multiply- and add- operations.

8. (**10 points**) In class, we discussed a reinforcement learning strategy to play Flappy Bird. However, we used a rather simple scenario where the only goal for Flappy was to avoid hitting the ground, while minimizing the amount of upward boosts. Here, we explore modifications to this problem. your answers can be either in words, or in pseudocode, or in Python.

   a. What if we had to learn a policy such that Flappy always remains above the ground, and below 150 meters?

We can modify flappy_reward(states, actions)

add `if height > 150: reward -= 10 * (height-150)` after `if height < 0: reward += 10 * height`

   b. What if we had to learn a policy which performed a *logistic* regression over the previous 4 states?

maybe we can consider it as a multi-class logistic regression problem. We create multiple class labels y, y represents different choices of actions, we consider "param" and previous 4 "states" as input of logistic regression to get probability of choice. Each time we use output of logistic regression to make a choice of action, if reward is good, we consider this action is right (matches class) (like $\mathbb{1}\{y^i = k\}$)

To refresh your memory, the relevant pieces of Python code from the class exercise is given below.

```
def flappy_reward(states, actions):
  reward = 0
  for state, action in zip(states, actions):
    _, height = state
    if height < 0:
      reward += 10*height
    reward -= action
  return reward

def flappy_policy(param,states,actions):
  # let's ignore previous actions and focus on prev 2 states
  if len(states) == 1:
    states = states + states
  return np.dot(param,np.reshape(np.array(states[-2:]),(4,)))
```

Part II

9. (**10 points**) Suppose that a convolutional layer of a neural network has an input tensor $X[i, j, k]$ and computes an output as follows:

$$Z[i, j, m] = \sum_{k_1} \sum_{k_2} \sum_{n} W[k_1, k_2, n, m] X[i + k_1, j + k_2, n] + b[m]$$
$$Y[i, j, m] = \text{sigmoid}(Z[i, j, m])$$

for some filter W and bias b. Suppose X and W have shapes $(128, 128, 10)$ and $(5, 5, 10, 10)$ respectively.

a. What are the shapes of Z and Y?

b. How many multiply- and add- operations are required to perform a forward pass through this layer? Rough calculations are OK.

c. If this was the final layer of a neural network, and we were doing k-class classification with $k = 18$ classes, how would the size of the filter W need to change?

10. (**5 points**) Mark the following statements as either true or false, and briefly (in 1 sentence or less) justify your reasoning.

    a. Gradient descent is capable of finding the minimum of any differentiable function.

    b. If a recurrent neural network is unrolled up to T time steps, then the number of independent weights in the unrolled network increases by a factor of T.

    c. Gaussian mixture models only can model circularly-shaped data distributions and not ellipses.

    d. In reinforcement learning, the reward function has to be differentiable, for otherwise learning a good policy is impossible.

    e. It is possible that the kernel feature mappings are extremely high dimensional, yet the kernel inner products are easy to calculate.

11. (**10 points**) This is a three-part question.

a. Consider neural network learning, and sketch typical curves of loss versus iteration count for the training and validation sets using stochastic gradient descent. (Assume for your example that overfitting to the training set happens at some point.)

b. Name two different strategies for preventing overfitting in neural network learning.

c. Briefly explain why a data augmentation strategy such as mirroring/flipping the training images may work for cat-vs-dog classifiers, but is **not** the right thing to do for distinguishing between English handwritten characters (a,b,c,d,...).

12. (**10 points**) Consider a 1-hidden layer *residual* neural network (ResNet) with sigmoid activation trained with squared error loss:

$$\hat{y} = x + \sigma(Wx), \quad \mathcal{L} = \|\hat{y} - y\|_2^2.$$

a. Draw the computational graph for this neural network.

b. Derive the forward and backward passes for backpropagation in this network.

13. (**10 points**) In class, we discussed a reinforcement learning strategy to play Flappy Bird. However, we used a rather simple scenario where the only goal for Flappy was to avoid hitting the ground, while minimizing the amount of upward boosts. Here, we explore modifications to this problem. your answers can be either in words, or in pseudocode, or in Python.

    a. What if we had to learn a policy such that Flappy always remains above the ground, and below 200 meters?

    b. What if we had to learn a policy which performed a *logistic* regression over the previous 6 states?

To refresh your memory, the relevant pieces of Python code from the class exercise is given below.

```
def flappy_reward(states, actions):
  reward = 0
  for state, action in zip(states, actions):
    _, height = state
    if height < 0:
      reward += 10*height
    reward -= action
  return reward

def flappy_policy(param,states,actions):
  # let's ignore previous actions and focus on prev 2 states
  if len(states) == 1:
    states = states + states
  return np.dot(param,np.reshape(np.array(states[-2:]),(4,)))
```

14. (**5 points**) The following represents python code for an algorithm we discussed in class. Assume that X is a data matrix and C is a matrix that has been randomly initialized.

    a. Identify this algorithm.

    b. What does the parameter k in the algorithm represent, and how do you choose it in practice?

```
def dist(a, b):
    return np.linalg.norm(a - b)

C_old = np.zeros(C.shape)

clusters = np.zeros(len(X))
error = dist(np.matrix.flatten(C), np.matrix.flatten(C_old))

while error != 0:
    for i in range(len(X)):
        distances = dist(X[i], C)
        cluster = np.argmin(distances)
        clusters[i] = cluster

    C_old = copy.copy(C)

    for i in range(k):
        points = [X[j] for j in range(len(X)) if clusters[j] == i]
        C[i] = np.mean(points, axis=0)

    error = dist(C, C_old, None)
```

15. (**10 points**) In weighted linear regression, the goal is to minimize a loss function of the form:

$$L(w) = \frac{1}{2}\sum_{i=1}^{n} \beta_i (y_i - \langle x_i, w \rangle)^2.$$

where x_i are d-dimensional input data, y_i are scalar labels, and γ_i are positive weights.

a. Show that the above loss function can be written in matrix notation:

$$L(W) = \frac{1}{2}(y - Xw)^T \Gamma (y - Xw).$$

Specify the sizes and contents of the variables y, X, Γ.

b. Suppose the conditional distribution of the data and labels follow a Gaussian model:

$$p(y_i|x_i) \propto \exp\left(-\frac{(y_i - \langle x_i, w \rangle)^2}{2\sigma_i^2}\right)$$

We would like to learn the parameters from n independent samples from the above distribution. Show that the maximum likelihood estimate can be written in the form of weighted linear regression of the form in part (a) and explicitly write down γ_i in terms of σ_i.