EECS 182 Deep Neural Networks
Spring 2023 Anant Sahai

# Midterm Exam

Exam location: Dwinelle 155

PRINT your student ID: 3038737132

PRINT AND SIGN your name: Lin, Mengying
(last) (first) (signature)

PRINT your discussion section: _____
Row Number (front row is 1): _____ Seat Number (left most is 1): _____

Name and SID of the person to your left: _____

Name and SID of the person to your right: _____

Name and SID of the person in front of you: _____

Name and SID of the person behind you: _____

## Section 0: Pre-exam questions (5 points)

**1.** Honor Code: Please copy the following statement in the space provided below and sign your name below. (1 point or $-\infty$)

As a member of the UC Berkeley community, I act with honesty, integrity, and respect for others. I will follow the rules and do this exam on my own.

**2.** What's your favorite thing about this semester? (4 points)

Do not turn this page until the proctor tells you to do so. You can work on Section 0 above before time starts.

PRINT your name and student ID: _____

# 3. Convolutional Neural Networks (19 points)

(a) (6 pts) Consider a convolutional neural network layer with:

- Input shape: [10, 3, 32, 32] (batch size, number of input channels, input height, input width)
- Number of filters: 64

For each of the following configurations of kernel size, stride, and padding, **calculate the output dimensions** $[n, c, h, w]$ where $n$ is the batch size, $c$ is the number of channels, and $h, w$ are the output height and width. Assume that each layer has the same input shape and number of filters as specified above.

i. Kernel size: 3x3, stride: 1, padding: 1

Output size: $F \times C_{\times} H_0 \times W_0$.

$H_0 = \left\lfloor \frac{H+2P-K}{S} \right\rfloor + 1 = \left\lfloor \frac{32+2-3}{1} \right\rfloor + 1 = 32$

$W_0 = \left\lfloor \frac{W+2P-K}{S} \right\rfloor + 1 = \left\lfloor \frac{32+2-3}{1} \right\rfloor + 1 = 32$ Output size: $64 \times 3 \times 32 \times 32$.

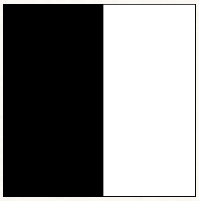
ii. Kernel size: 4x4, stride: 2, padding: 0

$H_0 = \left\lfloor \frac{H+2P-K}{S} \right\rfloor + 1 = \left\lfloor \frac{32+0-4}{2} \right\rfloor + 1 = 15$

$W_0 = 15$.

Output size: $64 \times 3 \times 15 \times 15$.

(b) (3 pts) **Design a 3x3 filter that detects vertical edges like the one shown in the image below.**



$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

(c) (3 pts) **Design a 3x3 filter to blur an image.**

*(Hint: blurring involves averaging a pixel's value with those of its neighbors.)*

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

PRINT your name and student ID: _____

(d) (7 pts) In a regular convolutional operation, the kernel slides over the input data in a contiguous manner. However, in dilated convolution, the kernel is "dilated" by introducing gaps between its elements, resulting in a larger receptive field for each output pixel.

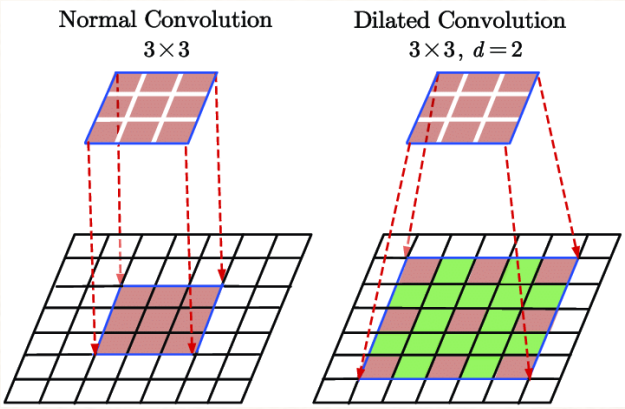



**Figure 1:** Dilated convolution kernels. Source: "Brain MRI Super-Resolution Using 3D Dilated Convolutional Encoder–Decoder Network" by Du et al.

The dilation ($d$) determines the spacing between kernel elements — a dilation of $d$ introduces $d-1$ gaps between two kernel elements. The example above illustrates a 3x3 1-dilated kernel ($d=1$ is equivalent to a regular convolutional kernel) and a 3x3 2-dilated ($d=2$) kernel.

i. You are given an input matrix $M$ and 2x2 filter $k$ below. **Compute their dilated convolution with $d=2$.** Assume that gaps in the kernel are filled with zeros, stride is 1 and padding is 0.

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad k = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 2 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 5 & 0 & 6 \\ 0 & 0 & 0 & 0 & 0 \\ 7 & 0 & 8 & 0 & 9 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 2 & 3 \\ 4 & 5 & 5 & 6 \\ 4 & 5 & 5 & 6 \\ 7 & 8 & 8 & 9 \end{bmatrix}$$

ii. Consider a two-layer architecture:

`DilatedConv1(3x3, d=1)` → `DilatedConv2(3x3, d=2)`

Both layers use the same sized kernel (3x3), but `DilatedConv1` has a dilation $d=1$ and `DilatedConv2` has a dilation $d=2$. **Compute the size of the receptive field at the output of the final layer (DilatedConv2).**

Recall that the receptive field is the region in the *original input* image whose pixels affect the output for a pixel in the specified layer.

img0 → DC1 → img1 → DC2 → img2.

1 pixel in img2 corresponds to 1/2/4 pixels in img1. 1 pixel in img1 corresponds to 9 pixels in img0.

Hence the receptive field is 9/18/36.

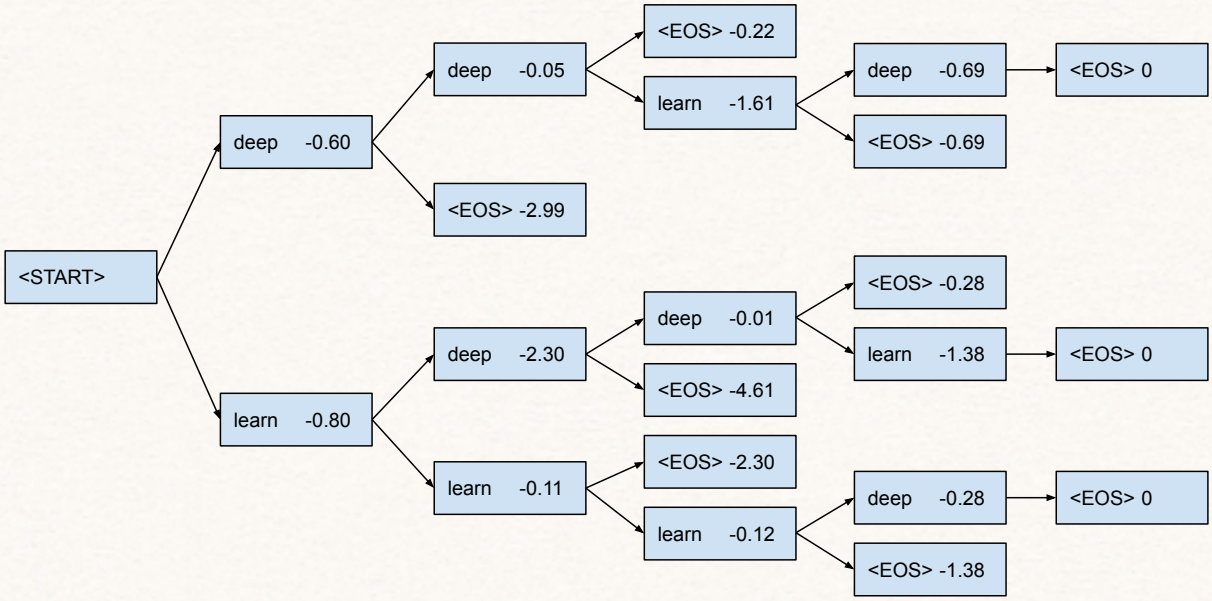
  

PRINT your name and student ID: _____

# 4. Beam Search (8 points)

Consider performing beam search with beam size $k = 2$ that expands the top $k$ sequences until an <EOS> token (end of sequence) is reached. Upon keeping a completed sequence as one of the top-$k$, the beam size for continuing the remaining search decreases by 1.

The log-probabilities of each word at a given timestep are shown next to the word.




**What are the 2 completed sequences that this decoder would consider?**

**What are their overall log-probabilities?**

<START> - deep - deep - <EOS>

<START> - learn - learn - learn - deep - <EOS>

p1 = -0.60 - 0.05 - 0.22 = -0.87

p2 = -0.80 - 0.11 - 0.12 - 0.28 + 0 = -1.31

PRINT your name and student ID: _____

[Extra page. If you want the work on this page to be graded, make sure you tell us on the problem's main page.]

PRINT your name and student ID: _____

## 5. Attention on Linear RNNs (14 points)

Consider a linear RNN with a two-dimensional state $h_t$ and a scalar input $u_t$ at each timestep:

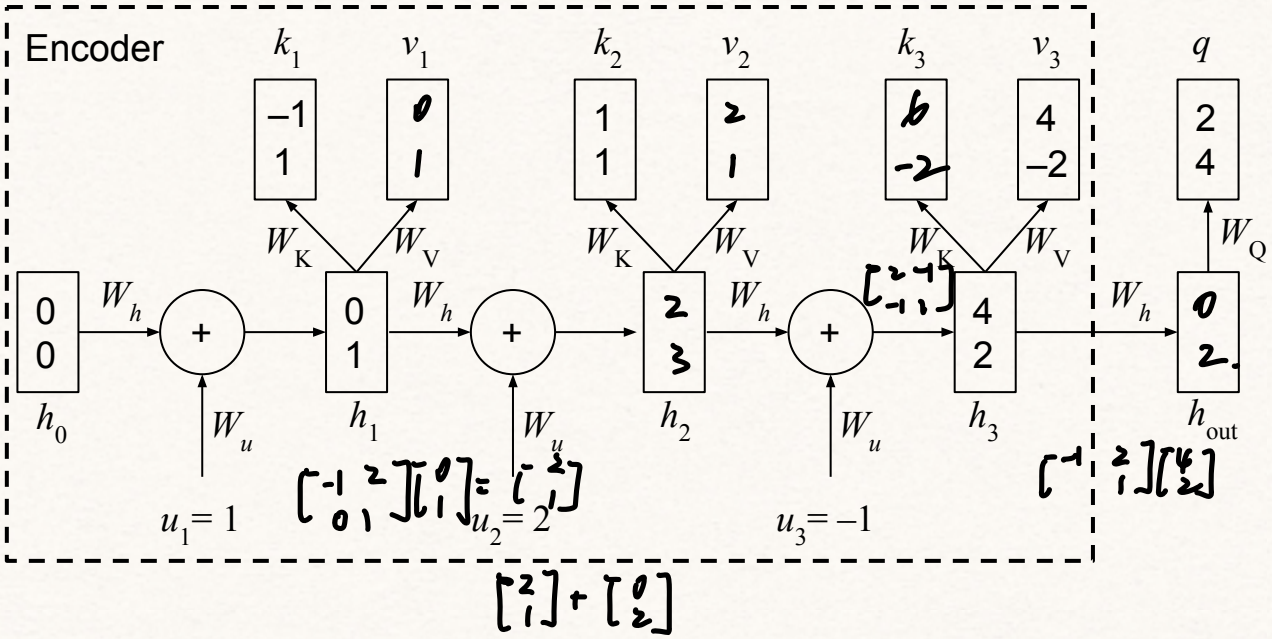
$$\mathbf{h}_t = W_h \mathbf{h}_{t-1} + W_u u_t$$

Suppose $W_h = \begin{bmatrix} -1 & 2 \\ 0 & 1 \end{bmatrix}$, $W_u = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, and we initialize $\mathbf{h}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$. The inputs are $u_1 = 1$, $u_2 = 2$, $u_3 = -1$. The decoder state is $\mathbf{h}_{out} = W_h \mathbf{h}_3$.

We also generate the keys, values, and queries for decoder-side cross-attention as purely linear functions of the state using weight matrices:

$$W_K = \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix} \qquad W_V = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \qquad W_Q = \begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix}$$

A partially filled in computation graph is provided below. The edges with weights $W$ on them indicate that the vector is left-multiplied by that weight matrix as it passes along that edge.




(a) (10 pts) **Fill in the blanks on the diagram above for the missing hidden states, keys, and values.** *(Hint: You can also use some of the given numbers to check your work.)*

(b) (4 pts) **What would be the output of attention for the decoder's query?** To simplify calculations, use an argmax instead of softmax. For example, softmax$([1, 3, 2])$ becomes $\text{argmax}([1, 3, 2]) = [0, 1, 0]$.

PRINT your name and student ID: _____

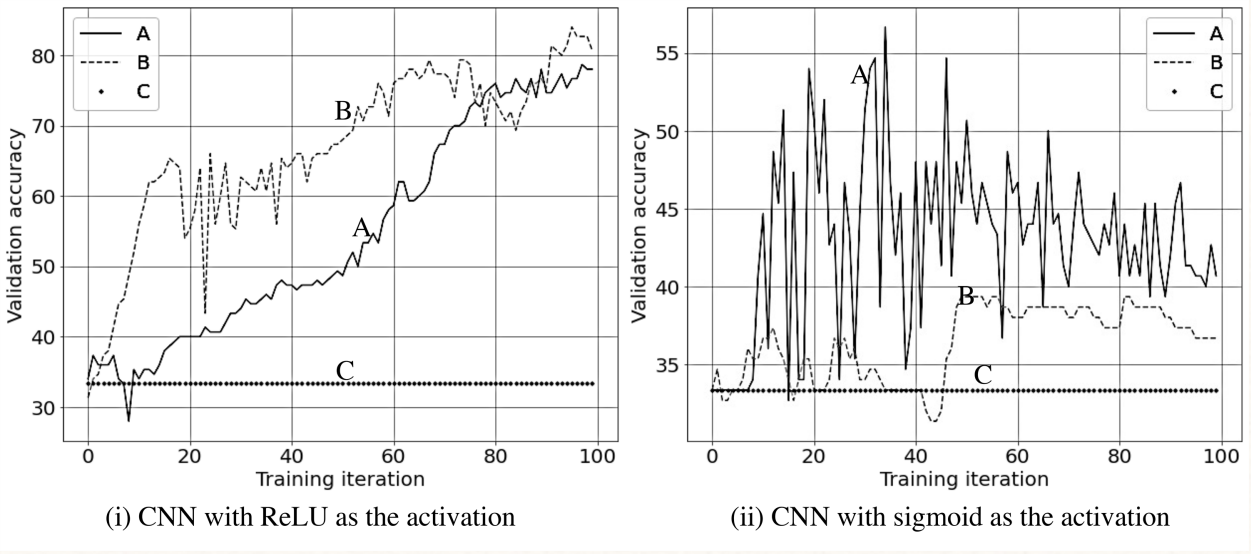
[Extra page. If you want the work on this page to be graded, make sure you tell us on the problem's main page.]

PRINT your name and student ID: _____

# 6. Debugging neural networks (21 points)

(a) (9 pts) Recall that we learned three initialization methods in lecture: Zero (all weight values are set to 0 in the beginning), Xaiver, and He. Suppose we train two different deep CNNs to classify images: (i) using ReLU as the activation function, and (ii) using sigmoids as the activation function. For each of them, we try initializing weights with the three different initialization methods aforementioned, while the biases are always initialized to all zeros. We plot the validation accuracies with different training iterations below:




(i) CNN with ReLU as the activation

(ii) CNN with sigmoid as the activation

**Which initialization methods are A, B, and C, (circle your choice) and very briefly explain your reasoning below it.**

A is *Zero* *Xavier* *He*

B is *Zero* *Xavier* *He*

C is *Zero* *Xavier* *He*

Zero initialization will lead to zero gradients. He initialization doubles the variance compared with Xavier, which is because Relu zeros out input with 0.5 probability. Therefore it performs the best with Relu.

PRINT your name and student ID: _____

(b) (12 pts) You are designing a neural network to perform image classification. You want to use data augmentation to regularize the training for your model. You write the following code:

```
import random
from torch.utils.data import DataLoader, Subset

train_dataset = load_train_dataset()

augmented_dataset = apply_augmentations(train_dataset)

num_data = len(augmented_dataset)
indices = list(range(num_data))
random.shuffle(indices)
split = int(0.8 * num_data)
train_idxs, val_idxs = indices[:split], indices[split:]

train_data = Subset(augmented_dataset, train_idxs)
val_data = Subset(augmented_dataset, val_idxs)
test_data = load_test_dataset()
test_data = apply_augmentations(test_data).
train_loader = DataLoader(train_data, batch_size=32)
val_loader = DataLoader(val_data, batch_size=32)
test_loader = DataLoader(test_dataset, batch_size=32)

# Train the model
for epoch in range(10):
    for images, labels in train_loader:
        # Training code here
    # Validation code here

# Testing code here
```

On running this code, you find that you get a *high training accuracy and validation accuracy, but a significantly lower testing accuracy as compared to the validation accuracy.*

Your TA suggests that there is a bug in your code that is causing this behavior. **Identify the bug, briefly explain why it causes the observed behavior, and show how to fix the bug.**

PRINT your name and student ID: _____

# 7. Learning from Point Clouds (22 points)

A point cloud is a discrete *set* of data points in space. Because of this *set*-valued nature of point clouds, concepts from graph neural nets are often relevant in their processing.

In Fig.2, we consider a simple network to process a 2d point cloud $X = \{x_i\}_{i=1}^n \in \mathbb{R}^{n\times 2}$, where $n$ is the number of points. The original features of each point are its horizontal and vertical coordinates.

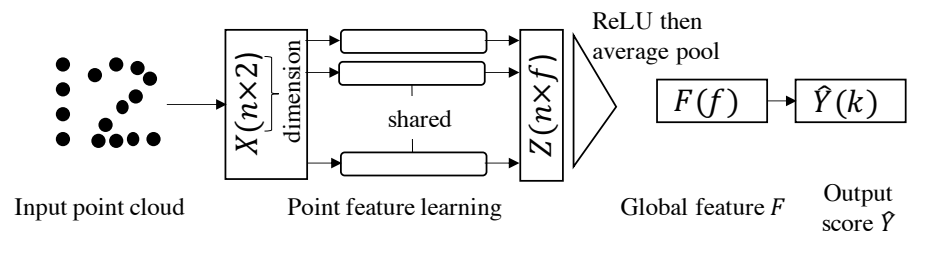



**Figure 2:** 2d point cloud processing network.

For example, an input point cloud with ground-truth of digit 1 could be represented as $\begin{bmatrix} 0 & 4 \\ 0 & 3 \\ 0 & 2 \\ 0 & 1 \end{bmatrix}$ where each row is a different point in the point cloud.

(a) (8 pts) The *point feature learning* module in Fig.2 learns $f$-dimensional features for each point separately. Specifically, it learns (shared) weights $W_1 \in \mathbb{R}^{2\times f}$ to get hidden layer outputs $Z = XW_1$. We then apply the nonlinear activation function element-wise and then use average pooling to yield the $f$-dimensional global feature vector $F \in \mathbb{R}^f$.

Suppose we swap the first two points of the input point cloud $X$, i.e. $\{x_1, x_2, ..., x_n\}$ to $\{x_2, x_1, ..., x_n\}$. **Show that the global feature $F$ will not change.**

*Note:* In reality, the network here is *permutation invariant*, as changing the ordering of the $n$ input points in $X$ will not affect the global feature $F$.

Denote $X = \{x_1 \cdots x_n\}$, $\hat{X} = \{x_2, x_1 \cdots x_n\}$.

$F = \text{Avg}(\text{Relu}(XW))$

$= \frac{1}{n}\sum_{i=1}^{n} \text{Relu}(x_i W)$

$= \text{Avg}(\text{Relu}(\hat{X}W))$

Hence it won't affect F.

PRINT your name and student ID: _____

(b) (8 pts) One drawback of the point feature learning discussed in part (a) is that the spatial inter-relationships of different points is not considered.

One idea is to use the Euclidean distance as a similarity measure to group points together into local neighborhoods, and to then process each point together with contextual information about that neighborhood. Your friend proposed several different point grouping methods listed below.

**Select all methods that guarantee permutation-invariance, i.e. the global feature vector $F$ will not change with different orderings of the points**.

☑ For each point $x_i$ of the point cloud $X$, we find the top-$m$ nearest neighbor points. We then augment each point coordinates with its nearest neighbors' coordinates to make $\widetilde{X} \in \mathbb{R}^{n\times 2(m+1)}$. The order of a concatenated group of points would be: the center point, 1st closest, 2nd closest point, ..., $m^{\text{th}}$ closest point. Now $Z = \widetilde{X}\widetilde{W_1}$ where $\widetilde{W_1} \in \mathbb{R}^{2(m+1)\times f}$ are the shared learnable weights.

☐ For each point $x_i$ of the point cloud $X$, we find the top-$m$ nearest neighbor points. As in the previous choice, we then augment each point coordinates with its nearest neighbors' coordinates and get $\widetilde{X} \in \mathbb{R}^{n\times 2(m+1)}$. *The difference from the previous choice is that the order of a concatenated group of nearby points simply follows their order in the original $X$.* The $\widetilde{W_1}$ is the same as the previous choice.

☑ For each point $x_i$ of the point cloud $X$, we *instead find all neighboring points with the distance to $x_i$ smaller than a predefined radius $r$. We then augment each point coordinates with its neighbors' coordinates with the order being the center point, the 1st closet point within $r$, the 2nd closet point within $r$, ..., the furthest point within $r$.* Because different points might have a different number of neighbors within radius $r$, the shared learnable weights $\widetilde{W_1} \in \mathbb{R}^{2(n+1)\times f}$ are applied by using the relevant-size truncation of $\widetilde{W_1}$ for every point.

☐ For each point $x_i$ of the point cloud $X$, as in the previous option, we find all neighboring points with the distance to $x_i$ smaller than a predefined radius $r$. But instead of concatenating the representation of the point with its neighboring points, we instead extend the point's representation with just the distance $d$ from $x_i$ to the furthest point within the radius $r$ resulting in an $\widetilde{X} \in \mathbb{R}^{n\times 3}$. The $\widetilde{W_1} \in \mathbb{R}^{3\times f}$.

PRINT your name and student ID: _____

(c) (6 pts) *Point downsampling (reducing the number of points for deeper layers to process).* Let's consider a deeper network, as shown in Fig.3. We are adding a pooling layer (highlighted in bold text) after the first point feature learning layer to downsample half of the points in the cloud ($Z \rightarrow Z_1$), followed by another point feature learning layer to increase the dimension of the point features from $f$ to $2f$ ($Z_1 \rightarrow Z_2$). (Note that this mimics pooling procedures in CNNs: the spatial size shrinks, followed by an increase of the feature dimensionality.)

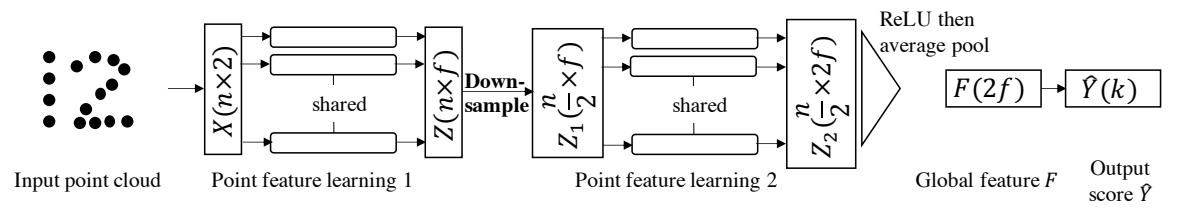



**Figure 3:** A deeper point cloud processing network.

Consider two candidate algorithms. Both start with the point cloud comprising $n$ points and both iteratively select points until you have at least $\frac{n}{2}$ samples. For both, we construct two sets: **sampled** and **remaining** to denote the set of sampled and remaining points. For both, we first pick a random point and use it to initialize **sampled** and we initialize **remaining** with all the other points. Then, the iterative processes are different for the two algorithms as described below.

**Which algorithm is more similar in spirit to using standard max pooling for downsampling in CNNs?**

○ **Algorithm 1:**

i. For each point in **remaining** find its nearest neighbor in **sampled**, saving the distance.

ii. Select the point in **remaining** whose nearest neighbor distance is the *largest* and move it from **remaining** to **sampled**. (*i.e.* We keep points far from those we already have.)

○ **Algorithm 2:**

i. For each point in **remaining** find its nearest neighbor in **sampled**, saving the distance.

ii. Select the point in **remaining** whose nearest neighbor distance is the *smallest* and move it from **remaining** to **sampled**. (*i.e.* We keep points close to those we already have.)

At the end, only the points in **sampled** get sent on to the next layer.

PRINT your name and student ID: _____

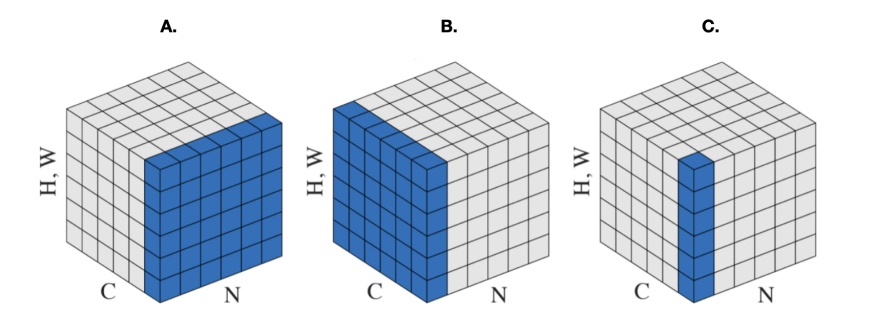
[Extra page. If you want the work on this page to be graded, make sure you tell us on the problem's main page.]

PRINT your name and student ID: _____

# 8. Normalization (14 points)

(a) (2 pts) Consider the following digram where the shaded blocks are the entries participating in one normalization step for a CNN-type architecture. $N$ represents the mini-batch, $H, W$ represent the different pixels of the "image" at this layer, and $C$ represents different channels.




- **Which one denotes the process of batch normalization?** Circle your selection below.

  A B C

- **Which one denotes layer normalization?** Circle your selection below.

  A B C

(b) (12 pts) Consider a simplified BN where we do not divide by the standard deviation of the data batch. Instead, we just de-mean our data batch before applying the scaling factor $\gamma$ and shifting factor $\beta$. For simplicity, consider scalar data in an $n$-sized batch: $[x_1, x_2, \ldots, x_n]$. Specifically, we let $\hat{x}_i = x_i - \mu$ where $\mu$ is the average $\frac{1}{n}\sum_{j=1}^{n} x_j$ across the batch and output $[y_1, y_2, \ldots, y_n]$ where $y_i = \gamma\hat{x}_i + \beta$ to the next layer. Assume we have a final loss $L$ somewhere downstream. **Calculate $\frac{\partial L}{\partial x_i}$ in terms of $\frac{\partial L}{\partial y_j}$ for $j = 1, \ldots, n$ as well as $\gamma$ and $\beta$ as needed.**

$$\frac{\partial L}{\partial x_i} = \sum_{j=1}^{n} \frac{\partial L}{\partial y_j} \cdot \frac{\partial y_j}{\partial \hat{x}_j} \cdot \frac{\partial \hat{x}_j}{\partial x_i}$$

$$= \sum_{j=1}^{n} \frac{\partial L}{\partial y_j} \cdot \gamma \cdot \frac{\partial \hat{x}_j}{\partial x_i}$$

$$\frac{\partial \hat{x}_j}{\partial x_i} = \begin{cases} 1 - \frac{1}{n} & (j = i) \\ -\frac{1}{n} & (j \neq i). \end{cases}$$

$$\frac{\partial L}{\partial x_i} = -\sum_{j \neq i} \frac{\partial L}{\partial y_j} \cdot \gamma \cdot \frac{1}{n} + \left(1 - \frac{1}{n}\right)\frac{\partial L}{\partial y_i}\gamma$$

Numerically, **what is $\frac{\partial L}{\partial x_1}$ when $n = 1$ and our input batch just consists of $[x_1]$ with an output batch of $[y_1]$?** (Your answer should be a real number. No need to justify.)

0.

**What happens when $n \to \infty$?** (Feel free to assume here that all relevant quantities are bounded.)

$$\frac{\partial L}{\partial x_i} \to \frac{\partial L}{\partial y_i}\gamma.$$

PRINT your name and student ID: _____

[Extra page. If you want the work on this page to be graded, make sure you tell us on the problem's main page.]

PRINT your name and student ID: _____

# 9. Optimizers (10 points)

**Algorithm 1** SGD with Momentum

**Given** $\eta = 0.001, \beta_1 = 0.9$
**Initialize**:
  time step $t \leftarrow 0$
  parameter $\theta_{t=0} \in \mathbb{R}^n$
**Repeat**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla f_t(\theta_{t-1})$
  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $\theta_t \leftarrow \theta_{t-1} - \eta m_t$
**Until** the stopping condition is met

**Algorithm 2** Adam Optimizer (without bias correction)

**Given** $\eta = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$
**Initialize** time step $t \leftarrow 0$, parameter $\theta_{t=0} \in \mathbb{R}^n$, $m_{t=0} \leftarrow 0$, $v_{t=0} \leftarrow 0$
**Repeat**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla f_t(\theta_{t-1})$
  $m_t \leftarrow$ ____**(A)**____
  $v_t \leftarrow$ ____**(B)**____
  $\theta_t \leftarrow \theta_{t-1} - \eta \cdot \frac{m_t}{\sqrt{v_t}}$
**Until** the stopping condition is met

(a) (4 pts) Complete part **(A)** and **(B)** in the pseudocode of Adam.

(b) (6 pts) This question asks you to establish the relationship between

- **L2 regularization** for vector-valued weights $\theta$ refers to adding a squared Euclidean norm of the weights to the loss function itself:

$$f_t^{reg} = f_t(\theta) + \frac{\lambda}{2}||\theta||_2^2$$

- **Weight decay** refers to explicitly introducing a scalar $\gamma$ in the weight updates assuming loss $f$:

$$\theta_{t+1} = (1 - \gamma)\theta_t - \eta \nabla f(\theta_t)$$

where $\gamma = 0$ would correspond to regular SGD since it has no weight-decay.

**Show that SGD with weight decay using the original loss $f_t(\theta)$ is equivalent to regular SGD on the L2-regularized loss $f_t^{reg}(\theta)$** when $\gamma$ is chosen correctly, and **find such a $\gamma$ in terms of $\lambda$ and $\eta$.**

$\nabla f_t^{reg}(\theta) = \nabla f_t(\theta) + \lambda\theta$

$\theta_{t+1} = \theta_t - \eta(\nabla f_t(\theta_t) + \lambda\theta_t)$

$= (1 - \eta\lambda)\theta_t - \eta \nabla f_t(\theta_t)$

Let $\gamma = \eta\lambda$. $\theta_{t+1} = (1 - \gamma)\theta_t - \eta \nabla f_t(\theta_t)$.

Q.E.D.

PRINT your name and student ID: _____

[Extra page. If you want the work on this page to be graded, make sure you tell us on the problem's main page.]

PRINT your name and student ID: _____

# 10. Self-Supervised Linear Purification (31 points)

Consider a linear encoder — *square* weight matrix $W \in \mathbb{R}^{m \times m}$ — that we want to be a "purification" operation on $m-$dimensional feature vectors from a particular problem domain. We do this by using self-supervised learning to reconstruct $n$ points of training data $\mathbf{X} \in \mathbb{R}^{m \times n}$ by minimizing the loss:

$$\mathcal{L}_1(W; \mathbf{X}) = \|\mathbf{X} - W\mathbf{X}\|_F^2 \tag{1}$$

While the trivial solution $W = \mathbf{I}$ can minimize the reconstruction loss (1), we will now see how weight-decay (or equivalently in this case, ridge-style regularization) can help us achieve non-trivial purification.

$$\mathcal{L}_2(W; \mathbf{X}, \lambda) = \underbrace{\|\mathbf{X} - W\mathbf{X}\|_F^2}_{\text{Reconstruction Loss}} + \lambda \underbrace{\|W\|_F^2}_{\text{Regularization Loss}} \tag{2}$$

Note above that $\lambda$ controls the relative weighting of the two losses in the optimization.

(a) (8 pts) Consider the simplified case for $m = 2$ with the following two candidate weight matrices:

$$W^{(\alpha)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad W^{(\beta)} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \tag{3}$$

The training data matrix $\mathbf{X}$ is also given to you as follows:

$$\mathbf{X} = \begin{bmatrix} -2.17 & 1.98 & 2.41 & -2.03 \\ 0.02 & -0.01 & 0.01 & -0.02 \end{bmatrix} \tag{4}$$

i. **Compute the reconstruction loss and the regularization loss for the two encoders, and fill in the missing entries in the table below.**

| Encoder | Reconstruction Loss | Regularization Loss |
|---|---|---|
| $\alpha$ | 0 | $2\lambda$ |
| $\beta$ | 0.001 | $0.001 + \lambda$ |

ii. **For what values of the regularization parameter $\lambda$ is the identity matrix $W^{(\alpha)}$ *not* a better solution for the objective $\mathcal{L}_2$ in (2), as compared to $W^{(\beta)}$?**

let $2\lambda > 0.001 + \lambda$

$\lambda > 0.001$.

PRINT your name and student ID: _____

[Extra page. If you want the work on this page to be graded, make sure you tell us on the problem's main page.]

PRINT your name and student ID: _____

(b) (15 pts) Now consider a generic square linear encoder $W \in \mathbb{R}^{m \times m}$ and the regularized objective $\mathcal{L}_2$ reproduced below for your convenience:

$$\mathcal{L}_2(W; \mathbf{X}, \lambda) = \underbrace{\|\mathbf{X} - W\mathbf{X}\|_F^2}_{\text{Reconstruction Loss}} + \lambda \quad \underbrace{\|W\|_F^2}_{\text{Regularization Loss}}$$

Assume $\sigma_1 > \cdots > \sigma_m \geq 0$ are the $m$ singular values in $\mathbf{X}$, that the number of training points $n$ is larger than the number of features $m$, and that $\mathbf{X}$ can be expressed in SVD coordinates as $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^\top$.

i. You are given that the optimizing weight matrix for the regularized objective $\mathcal{L}_2$ above takes the following form. **Fill in the empty matrices below**.

$$\widehat{W} = \begin{bmatrix} & \end{bmatrix} \cdot \begin{bmatrix} \frac{\sigma_1^2}{\sigma_1^2+\lambda} & & & \\ & \frac{\sigma_2^2}{\sigma_2^2+\lambda} & & \\ & & \ddots & \\ & & & \frac{\sigma_m^2}{\sigma_m^2+\lambda} \end{bmatrix} \cdot \begin{bmatrix} & \end{bmatrix} \tag{5}$$

ii. **Derive the above expression.**
*(Hint: Can you understand $\mathcal{L}_2(W; \mathbf{X}, \lambda)$ as a sum of $m$ completely decoupled ridge-regression problems?)*

$$\hat{W} = (X^T X + \lambda I)^{-1} X^T X$$
$$= (V\Sigma^2 V^T + \lambda I)^{-1} V\Sigma^2 V^T$$
$$= V(\Sigma^2 + \lambda I)^{-1} V^T V \Sigma^2 V^T$$
$$= V(\Sigma^2 + \lambda I)^{-1} \Sigma^2 V^T.$$

$$(\Sigma^2 + \lambda I)^{-1}\Sigma^2 = \begin{pmatrix} \frac{\sigma_1^2}{\sigma_1^2+\lambda} & & \\ & \ddots & \\ & & \frac{\sigma_m^2}{\sigma_m^2+\lambda} \end{pmatrix}$$

$$\hat{W} = V \begin{pmatrix} \frac{\sigma_1^2}{\sigma_1^2+\lambda} & & \\ & \ddots & \\ & & \frac{\sigma_m^2}{\sigma_m^2+\lambda} \end{pmatrix} V^T.$$

Q.E.D.

PRINT your name and student ID: _____

(c) (8 pts) You are given that the data matrix $\mathbf{X} \in \mathbb{R}^{8\times n}$ has the following singular values:

$$\{\sigma_i\} = \{10, 8, 4, 1, 0.5, 0.36, 0.16, 0.01\}$$

**For what set of hyperparameter values $\lambda$ can we guarantee that the learned purifier $\widehat{W}$ will preserve at least 80% of the feature directions corresponding to the first 3 singular vectors of $\mathbf{X}$, while attenuating components in the remaining directions to at most 50% of their original strength?**

*(Hint: What are the two critical singular values to focus on?)*

when $\frac{\sigma_i^2}{\sigma_i^2+\lambda} > 0.8$, $\lambda < \frac{\sigma_i^2}{4}$ for $i = 1, 2, 3$.

Hence $\lambda < \frac{4^2}{4} = 4$.

when $\frac{\sigma_i^2}{\sigma_i^2+\lambda} < 0.5$. $\lambda > \sigma_i^2$ for $i \neq 1, 2, 3$.

Hence $\lambda > 1$.

we have $1 < \lambda < 4$.

## 2. Hand-Design Transformers

Please follow the instructions in this notebook. You will implement a simple transformer model (with a single attention head) from scratch and then create a hand-designed the attention heads of the transformer model capable of solving a basic problem. Once you finished with the notebook,

- Download `submission_log.json` and submit it to "Homework 8 (Code)" in Gradescope.
- Answer the following questions in your submission of the written assignment:

(a) Design a transformer that selects by contents. Compare the variables of your hand-designed Transformer with those of the learned Transformer. **Identify the similarities and differences between the two sets of variables and provide a brief explanation for each difference.**

(b) Design a transformer that selects by positions. Compare the variables of your hand-designed Transformer with those of the learned Transformer. **Identify the similarities and differences between the two sets of variables and provide a brief explanation for each difference.**

a) Vm attention scores & attention are roughly the same. cuz the network is simple enough that there is only one correct attention pattern which attends to matching positions.

b): The hand designed Km and Qm are different from those that are learned. Others are the same. In fact Vm might differ as well. We only need to make sure the final linear combinations of vectors, which are the outputs, are almost the same.

## 3. Kernelized Linear Attention (Part II)

This is a continuation of "Kernelized Linear Attention" in HW 7. Please refer to this part for notation and context. In Part I of this problem, we considered ways to efficiently express the attention operation when sequences are long (e.g. a long document). Attention uses the following equation:

$$V_i' = \frac{\sum_{j=1}^{N} \text{sim}\left(Q_i, K_j\right) V_j}{\sum_{j=1}^{N} \text{sim}\left(Q_i, K_j\right)}. \tag{1}$$

We saw that when the similarity function is a kernel function (i.e. if we can write $\text{sim}\left(Q_i, K_j\right) = \Phi(Q_i)^T \Phi(K_j)$ for some function $\Phi$), then we can use the associative property of matrix multiplication to simplify the formula to

$$V_i' = \frac{\phi\left(Q_i\right)^T \sum_{j=1}^{N} \phi\left(K_j\right) V_j^T}{\phi\left(Q_i\right)^T \sum_{j=1}^{N} \phi\left(K_j\right)}. \tag{2}$$

If we use a polynomial kernel with degree 2, this gives a computational cost of $\mathcal{O}\left(ND^2M\right)$, which for very large $N$ is favorable to the softmax attention computational cost of $\mathcal{O}\left(N^2 \max\left(D, M\right)\right)$. ($N$ is the sequence length, $D$ is the feature dimension of the queries and keys, and $M$ is the feature dimension of the values. Now, we will see whether we can use kernel attention to directly approximate the softmax attention:

$$V_i' = \frac{\sum_{j=1}^{N} \exp(\frac{Q_i^T K_j}{\sqrt{D}}) V_j}{\sum_{j=1}^{N} \exp(\frac{Q_i^T K_j}{\sqrt{D}})}. \tag{3}$$

(a) Approximating softmax attention with linearized kernel attention

i. As a first step, we can use Gaussian Kernel $\mathcal{K}_{\text{Gauss}}(q, k) = \exp(\frac{-||q-k||_2^2}{2\sigma^2})$ to rewrite the softmax similarity function, where $\text{sim}_{\text{softmax}}(q, k) = \exp(\frac{q^T k}{\sqrt{D}})$. Assuming we can have $\sigma^2 = \sqrt{D}$, **rewrite the softmax similarity function using Gaussian Kernel.**. (*Hint: You can write the softmax* $\exp(\frac{-||q-k||_2^2}{2\sigma^2})$ *as the product of the Gaussian Kernel and two other terms.*)

$K_{Gauss}(q_i,k) = e^{-\frac{(q_i-k)^T(q_i-k)}{2\sigma^2}} = e^{-\frac{q_i^2+k^2}{2\sigma^2}} \cdot e^{\frac{q_i^T k}{\sigma^2}} = e^{-\frac{q_i^2+k^2}{2\sigma^2}} sim_{softmax}(q_i,k)$

$sim_{softmax}(q_i,k) = e^{\frac{q_i^2+k^2}{2\sigma^2}} K_{Gauss}(q_i,k).$

ii. However, writing softmax attention using a Gaussian kernel does not directly enjoy the benefits of the reduced complexity using the feature map. This is because the feature map of Guassian kernel usually comes from the Taylor expression of $\exp(\cdot)$, whose computation is still expensive [1]. However, we can approximate the Guassian kernel using random feature map and then reduce the computation cost. (Rahimi and Recht, 2007)[2] proposed random Fourier features to approximate a desired shift-invariant kernel. The method nonlinearly transforms a pair of vectors $q$ and $k$ using a **random feature map** $\phi_{\text{random}}()$; the inner product between $\phi(q)$ and $\phi(k)$ approximates the kernel evaluation on $q$ and $k$. More precisely:

$$\phi_{\text{random}}(q) = \sqrt{\frac{1}{D_{\text{random}}}}\left[\sin\left(\mathbf{w}_1 q\right), \ldots, \sin\left(\mathbf{w}_{D_{\text{random}}} q\right), \cos\left(\mathbf{w}_1 q\right), \ldots, \cos\left(\mathbf{w}_{D_{\text{random}}} q\right)\right]^\top. \tag{4}$$

Where we have $D_{\text{random}}$ of $D$-dimensional random vectors $\mathbf{w}_i$ independently sampled from $\mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_D)$,

$$\mathbb{E}_{\mathbf{w}_i}\left[\phi(q) \cdot \phi(k)\right] = \exp\left(-\|q-k\|^2 / 2\sigma^2\right). \tag{5}$$

**Use $\phi_{\text{random}}$ to approximate the above softmax similarity function with Gaussian Kernel and derive the computation cost for computing all the $V'$ here.**

$sim_{softmax}(q_i,k) \approx e^{\frac{q_i^2+k^2}{2\sigma^2}} K_{Gauss}(q_i,k)$

$= e^{\frac{q_i^2+k^2}{2\sigma^2}} E_{wi}[\phi(q_i)\phi(k)]$

$= e^{\frac{q_i^2+k^2}{2\sigma^2}} E_{wi}[\phi_{random}(q_i)\phi_{random}(k)]$

$= e^{\frac{q_i^2+k^2}{2\sigma^2}} E_{wi}[\frac{1}{D_{random}} \sum_{i=1}^{D_{random}} \sin^2(w_i q_i) + \cos^2(w_i q_i)]$

$= e^{\frac{q_i^2+k^2}{2\sigma^2}} \cdot \phi_{random}(q_i)^T \phi_{random}(k).$

Then computation cost of computing $v'$ is ~~$O(NMD)$~~.

$O(N(M+D)D_{random})$.

$X \in \mathbb{R}^{N \times F}$

$W_Q \in \mathbb{R}^{F \times D}$

$W_K \in \mathbb{R}^{F \times D}$

$W_V \in \mathbb{R}^{F \times M}$

Computing $q_i^2 + k^2$: $O(N)$.

Computing $\phi_{random}(q_i)^T \cdot \phi_{random}(k)$: $O(ND D_{random})$

Computing $v_i' = \frac{\sum sim_{softmax}(q_i, k_j) v_i}{\sum sim_{softmax}(q_i, k_j)}$:

$O(NMD_{random})$.

**derive the computation cost for computing all the $V'$ here.**

(b) (Optional) Beyond self-attention, an autoregressive case will be masking the attention computation such that the $i$-th position can only be influenced by a position $j$ if and only if $j \leq i$, namely a position cannot be influenced by the subsequent positions. This type of attention masking is called *causal masking*.

Derive how this causal masking changes equation 1 and 2. **Write equation 1 and 2 in terms of $S_i$ and $Z_i$**, which are defined as:

$$S_i = \sum_{j=1}^{i} \phi\left(K_j\right) V_j^T, \; Z_i = \sum_{j=1}^{i} \phi\left(K_j\right), \tag{6}$$

to simplify the causal masking kernel attention and derive the computational complexity of this new causal masking formulation scheme.