EECS 182 Deep Neural Networks
Spring 2023 Anant Sahai

Midterm Exam

Exam location: Dwinelle 155

PRINT your student ID: ____

PRINT AND SIGN your name: __________________, __________________ ______________________
(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.

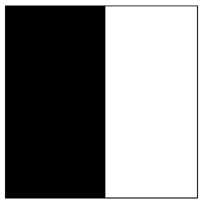
i. Kernel size: 3x3, stride: 1, padding: 1

Solution: Since the output dimensions are calculated as $d_{out} = \lfloor \frac{d_{in}+2p-k}{s} + 1 \rfloor$ where d_{in} is the input dimension, d_{out} and d_{out} are the output dimensions, p is the padding, k is the kernel size, and s is the stride, we can substitute these values to get the output dimension: $d_{out} = \lfloor \frac{32+2(1)-3}{1} + 1 \rfloor = 32$.
The output shape would be $[10, 64, 32, 32]$

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

Solution: Using the same formula above, we get the output shape $[10, 64, 15, 15]$

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



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

Solution:

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

(c) (3 pts) **Design a 3x3 filter to blur an image.**
(Hint: blurring involves averaging a pixel's value with those of its neighbors.)

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

Solution:

$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$

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.

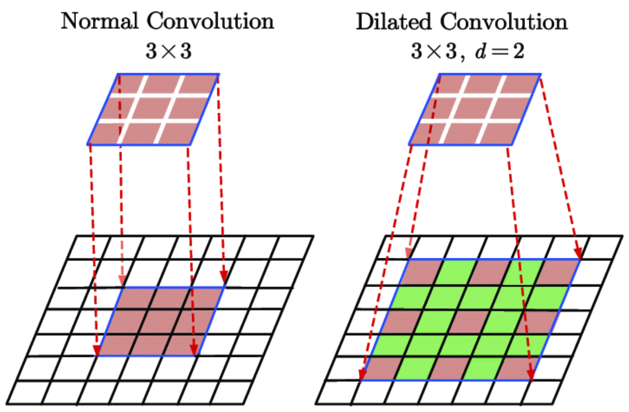


Figure 1: Dilated convolution kernels. Source: "Brain MRI Super-Resolution Using 3D Dilated Convolutional Encoder–Decoder Network" by Du et al.

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.

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

Solution: With dilation $d = 2$, the kernel becomes a 3x3 kernel:

$$k_d = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Applying regular convolution $M * k_d$, we get:

$$M * k_d = \begin{bmatrix} 20 \end{bmatrix}$$

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.

Solution: 7x7. Drawing the outputs might help.

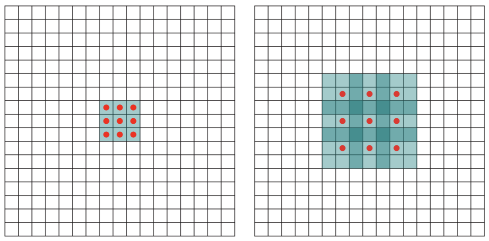


Figure 2: Receptive field size at DilatedConv1 and DilatedConv2 layers. Source: "Multi-Scale Context Aggregation by Dilated Convolutions", Yu et. al.

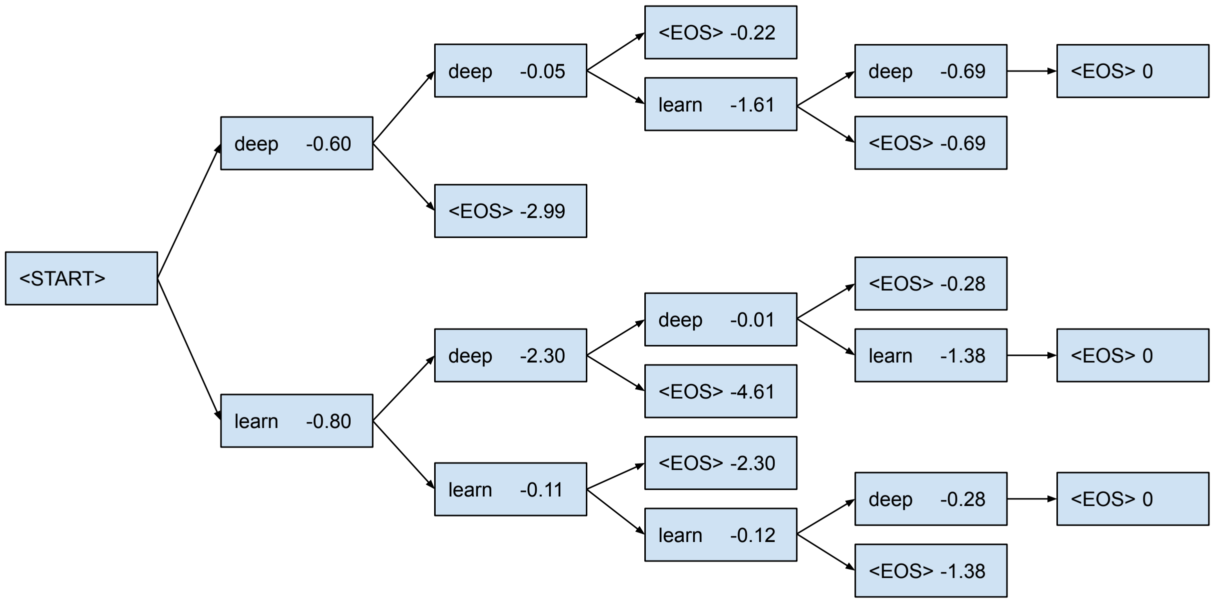
Figure 3: Receptive field size at each layer. Source: "Multi-Scale Context Aggregation by Dilated Convolutions", Yu et. al.

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?

Solution: At $t = 1$, we have only two hypotheses:
deep: -0.6
learn: -0.8

At $t = 2$, we expand and have:
deep deep: -0.65
~~deep <EOS>: -3.59~~
~~learn deep: -3.9~~
learn learn: -0.91

The crossed-out hypotheses are eliminated since we only keep the top two.

At $t = 3$, we expand and have:
deep deep <EOS>: -0.87
~~deep deep learn: -2.26~~
~~learn learn <EOS>: -3.21~~
learn learn learn: -1.03

Now, we continue expanding to get one more completed sequence:
learn learn learn deep <EOS>: -1.31
~~learn learn learn <EOS>: -2.41~~

Solution: As we found from our beam search, the most likely completed sequences are

deep deep <EOS>: -0.87
learn learn learn deep <EOS>: -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:

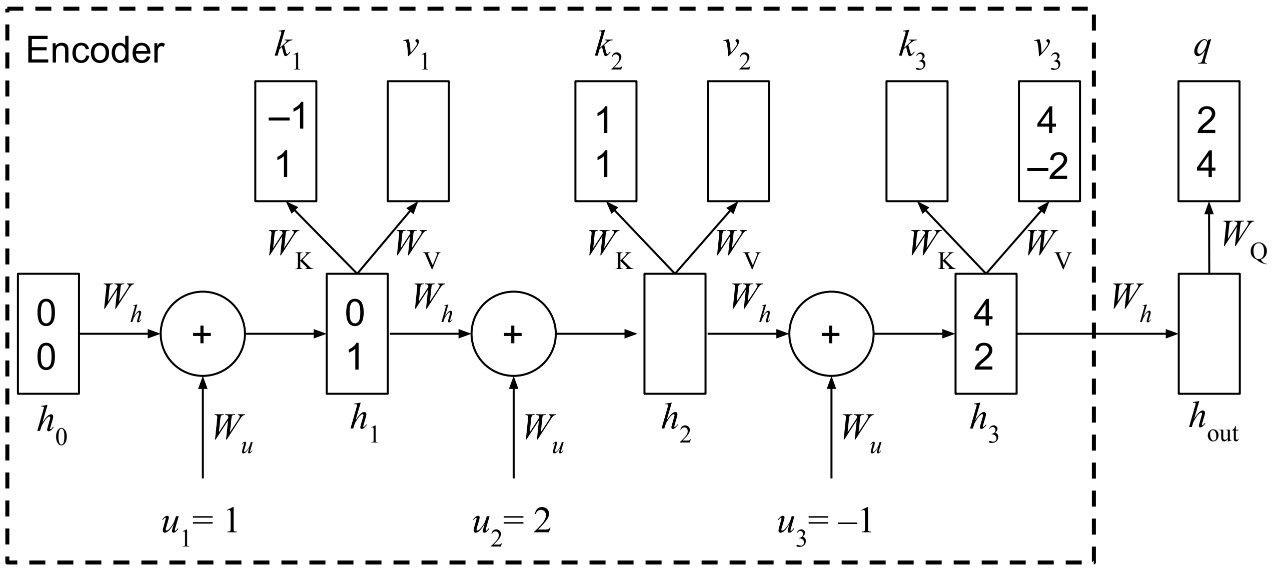
$$\mathbf{h}_t = W_h \mathbf{h}_{t-1} + W_u u_t$$

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.)*

Solution: We have $v_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $h_2 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$, $v_2 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$, $k_3 = \begin{bmatrix} 6 \\ -1 \end{bmatrix}$, $h_{\text{out}} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$

(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 argmax$([1, 3, 2]) = [0, 1, 0]$.

Solution: Compute inner products of query with the keys:

$$\langle q, k_1 \rangle = 2$$

$$\langle q, k_2 \rangle = 6$$
$$\langle q, k_3 \rangle = 4$$

We take argmax rather than softmax, and have $\text{argmax}([2, 6, 4]) = [0, 1, 0]$
Now, take weighted sum of value vectors (in this case all are zeroed out except for the one corresponding to the highest dot-product between query and key)
$0 \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} + 1 \cdot \begin{bmatrix} 2 \\ 1 \end{bmatrix} + 0 \cdot \begin{bmatrix} 4 \\ -2 \end{bmatrix}$
So final output is $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$

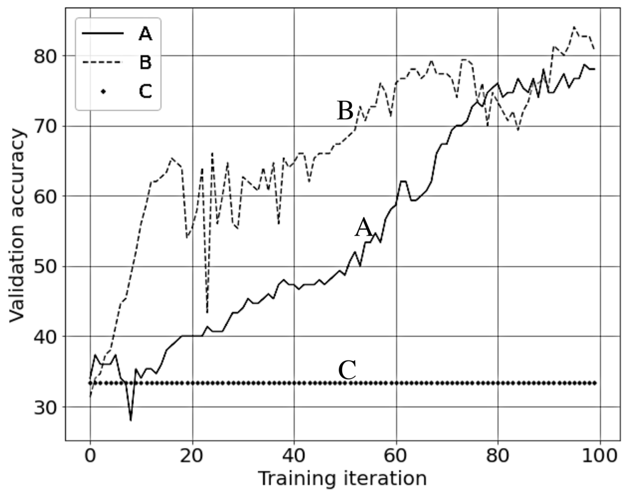
PRINT your name and student ID: ____

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

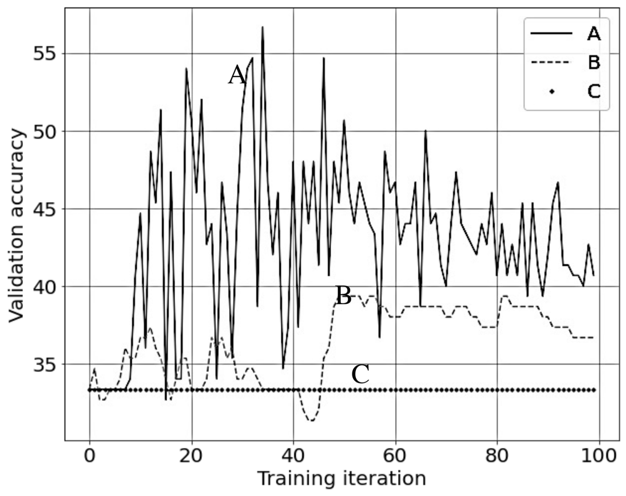
PRINT your name and student ID: ____________________

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,
and very briefly explain your reasoning below it. Please use ■ for your selections.

A is ☐ *Zero* ☐ *Xavier* ☐ *He*

B is ☐ *Zero* ☐ *Xavier* ☐ *He*

C is ☐ *Zero* ☐ *Xavier* ☐ *He*

Solution: A is Xaiver, B is He initialization. From the plot we observe that B performs better for ReLU while A performs better for sigmoid. Recall that Xaiver is designed for sigmoid activations, whereas He initialization considers ReLU. He initialization assumes the inputs to the ReLU function are half non-positive and half positive and change the standard deviation of the normalization distribution to compensate for that property.

C is zero parameter initialization. This method causes neurons to perform the same calculation in each iteration and produces the same outputs. Thus the accuracy is the lowest and not updating for different iterations.

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()

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.**

Solution: In our buggy code, we perform data augmentation before we create train/validation splits. This causes training data to leak into the validation set and vice-versa. To correct this, we must perform data augmentation after creating the training/validation splits.

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.4, 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.

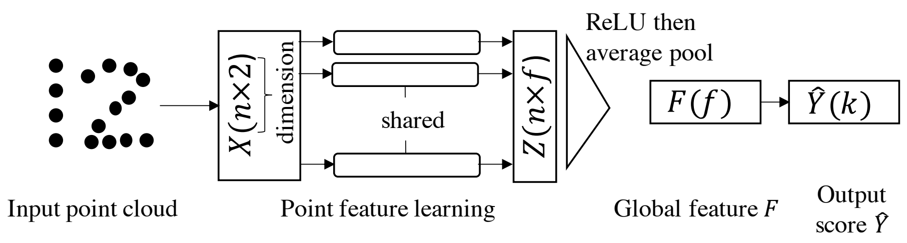


Figure 4: 2d point cloud processing network.

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.4 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.

Solution: Expand X row-wise, derive X_1:

$$Z = XW_1 = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} W_1 = \begin{bmatrix} x_1W_1 \\ x_2W_1 \\ \dots \\ x_nW_1 \end{bmatrix} \tag{1}$$

The global feature F is the average all row vectors of Z:

$$F = \frac{1}{n}\sum_{i=1}^{n} \text{ReLU}(x_iW_1) \tag{2}$$

We can see that swapping the order of x_1, x_2 will not affect F.

Note: Actually, permuting the order of points $\{x_1, x_2, ..., x_n\}$ will not change F as it is a sum of all input point coordinates transformed by a weight matrix. Additionally, ReLU is invariant to the permutation of the order of input points.

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. Please use ■ for your selections.

- ☐ 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^{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 $\widetilde{W_1} \in \mathbb{R}^{3\times f}$.

Solution: The first, the third and the fourth one are permutation-invariant. The second one follows a given order by X, and cannot guarantee permutation invariance.

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.5. 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 ($Z \to Z_1$), followed by another point feature learning layer to increase the dimension of the point features from f to $2f$ ($Z_1 \to Z_2$). (Note that this mimics pooling procedures in CNNs: the spatial size shrinks, followed by an increase of the feature dimensionality.)

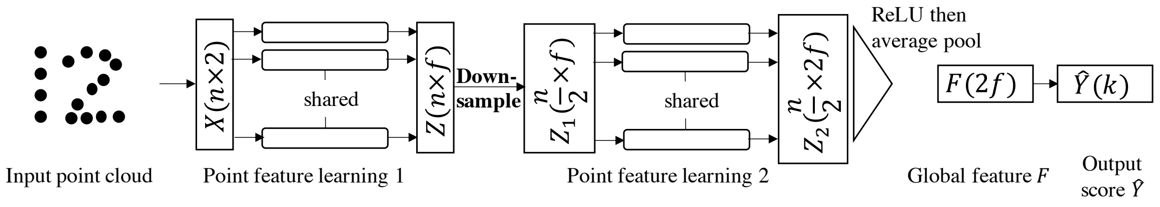


Figure 5: A deeper point cloud processing network.

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? Please use ■ for your selections.

☐ **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.

Solution: The first algorithm is more similar to a max pooling layer than the second algorithm. The first algorithm outputs the leading $N/2$ most different points and is a good fit for the pooling layer. The problem with the second algorithm is that it forces each point to bind with its nearest neighbor in the remaining set and by this constraint we lose the flexiblity to group densed points (more than 2 points) into one single point while keeping single significant individual points alone without having to bind with a neighbor. Such disjoint even division is seen in CNN pooling layers as the pixels' nearest neighbor distances are all the same, but not a good fit for GNN pooling layers where we have to respect the freedom of neighbors having different distances.

Note: Indeed, the first algorithm is called the furthest point sampling algorithm, and it has the flexiblity to output any number of most significant points.

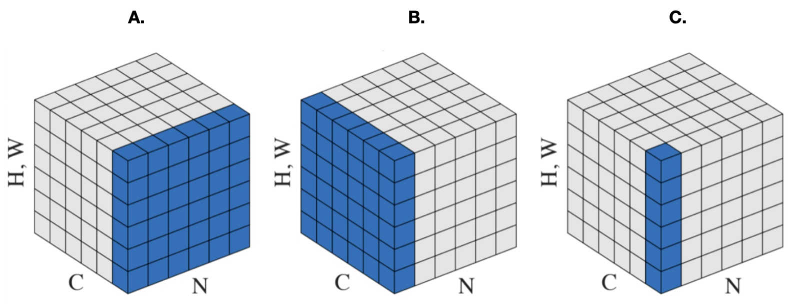
PRINT your name and student ID: ______________________________

[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?** Please use ■ for your selections.

  ☐ A ☐ B ☐ C

- **Which one denotes layer normalization?** Please use ■ for your selections.

  ☐ A ☐ B ☐ C

Solution: Batch norm is option A, Layer norm is option B.

(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 γ and shifting factor β. 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 μ 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 γ and β as needed.**

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.)

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

Solution:

Since values for a given x_i affects all the values of y_j we need to sum over partial derivatives with respect to all the different y_j.

$$\frac{\partial L}{\partial x_i} = \sum_{j=1}^{n} \frac{\partial L}{\partial y_j} * \frac{\partial y_j}{\partial x_i}$$

Now, let's calculate $\frac{\partial y_j}{\partial x_i}$. Note that we can write y_j as

$$y_j = \gamma \left(x_j - \frac{1}{n} \sum_{k=1}^{m} x_k \right) + \beta$$

So we can calculate it's derivative wrt x_i as

$$\frac{\partial y_j}{\partial x_i} = \begin{cases} \gamma * \left(1 - \frac{1}{n}\right) \text{, if } i = j \\ \gamma * \left(-\frac{1}{n}\right) \text{, if } i \neq j \end{cases}$$

We can combine the sum to get

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

$$= \gamma \left[\frac{\partial L}{\partial y_i} - \frac{1}{n} \sum_{j=1}^{n} \frac{\partial L}{\partial y_j} \right]$$

Based on this expression, we see that $\frac{\partial L}{\partial x_i} = 0$ when $n = 1$ and $\gamma \cdot \frac{\partial L}{\partial y_i}$ when n approaches infinity. This shows that with larger batch size, batchnorm has an ideal effect of isolating the gradients through each sample.

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

```
1: Given $\eta = 0.001, \beta_1 = 0.9$
2: Initialize:
3:     time step $t \leftarrow 0$
4:     parameter $\theta_{t=0} \in \mathbb{R}^n$
5: Repeat
6:     $t \leftarrow t + 1$
7:     $g_t \leftarrow \nabla f_t(\theta_{t-1})$
8:     $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
9:     $\theta_t \leftarrow \theta_{t-1} - \eta m_t$
10: Until the stopping condition is met
```

Algorithm 2 Adam Optimizer (without bias correction)

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

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

Solution:

$$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

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

- **L2 regularization** for vector-valued weights θ 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 γ 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 γ is chosen correctly, and **find such a** γ **in terms of** λ **and** η.

Solution:

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

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

The process above shows that L2 regularization and weight decay are equivalent when $\gamma = \lambda\eta$.

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:

$$\mathcal{L}_1(W;\mathbf{X}) = \|\mathbf{X} - W\mathbf{X}\|_F^2 \tag{3}$$

While the trivial solution $W = \mathbf{I}$ can minimize the reconstruction loss (3), we will now see how weight-decay (or equivalently in this case, ridge-style regularization) can help us achieve non-trivial purification.

$$\mathcal{L}_2(W;\mathbf{X},\lambda) = \underbrace{\|\mathbf{X} - W\mathbf{X}\|_F^2}_{\text{Reconstruction Loss}} + \lambda \underbrace{\|W\|_F^2}_{\text{Regularization Loss}} \tag{4}$$

Note above that λ 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:

$$W^{(\alpha)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad W^{(\beta)} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \tag{5}$$

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

$$\mathbf{X} = \begin{bmatrix} -2.17 & 1.98 & 2.41 & -2.03 \\ 0.02 & -0.01 & 0.01 & -0.02 \end{bmatrix} \tag{6}$$

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
α	________	________
β	0.001	________

Solution:

Encoder	Reconstruction Loss	Regularization Loss
α	0	2
β	0.001	1

ii. **For what values of the regularization parameter λ is the identity matrix $W^{(\alpha)}$ *not* a better solution for the objective $\mathcal{L}_2$ in (4), as compared to $W^{(\beta)}$?**

Solution: Plugging the numbers from the table into Eqn. 4, the β encoder is a better choice for

$$\lambda > 1 \times 10^{-3}.$$

This suggests that despite the encoder being square, regularization induces a bottleneck in the encoder, preventing it from learning an identity mapping.

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:

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

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.**

$$\widehat{W} = \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix} \cdot \begin{bmatrix} \frac{\sigma_1^2}{\sigma_1^2+\lambda} & & & \\ & \frac{\sigma_2^2}{\sigma_2^2+\lambda} & & \\ & & \ddots & \\ & & & \frac{\sigma_m^2}{\sigma_m^2+\lambda} \end{bmatrix} \cdot \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix} \tag{7}$$

Solution:

$$\hat{W} = \begin{bmatrix} \mathbf{U} \end{bmatrix} \cdot \begin{bmatrix} \frac{\sigma_1^2}{\sigma_1^2+\lambda} & & & \\ & \frac{\sigma_2^2}{\sigma_2^2+\lambda} & & \\ & & \ddots & \\ & & & \frac{\sigma_m^2}{\sigma_m^2+\lambda} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{U}^\top \end{bmatrix}$$

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?)
Solution: Sum of m completely decoupled ridge-regression approach:
Since the squared Forbenius norm is just the sum of squares of all the elements in the matrix, we can decouple them into sum of m l2 norms. Specifically:

$$\mathcal{L}_2(W;\mathbf{X},\lambda) = \|\mathbf{X} - W\mathbf{X}\|_F^2 + \lambda\|W\|_F^2 = \sum_{i=1}^{m} \|\mathbf{X}_i - W_i\mathbf{X}\|_2^2 + \lambda\|W_i\|_2^2$$

where $\mathbf{X}_i \in \mathbb{R}^{1\times n}$ and $W_i \in \mathbb{R}^{1\times m}$ are rows. We can also write it as:

$$\mathcal{L}_2(W;\mathbf{X},\lambda) = \sum_{i=1}^{m} \|\mathbf{X}_i^\top - \mathbf{X}^\top W_i^\top\|_2^2 + \lambda\|W_i^\top\|_2^2$$

which resembles the classical ridge regression optimization problem $\|y - X\beta\|_2^2 + \lambda\|\beta\|_2^2$ which can be minimized with $\hat{\beta} = (X^\top X + \lambda I)^{-1}X^\top y$.
Through pattern matching $\mathbf{X}_i^\top = y, \mathbf{X}^\top = X, W_i^\top = \beta$, we can arrive with the result:

$$\hat{W}_i^\top = (\mathbf{X}\mathbf{X}^\top + \lambda I)^{-1}\mathbf{X}\mathbf{X}_i^\top$$

$$\hat{W} = \mathbf{X}\mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top + \lambda I)^{-1}$$

Now substitute $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^\top$ to the above result, we have:

$$\hat{W} = \mathbf{U}\Sigma\mathbf{V}^\top(\mathbf{U}\Sigma\mathbf{V}^\top)^\top(\mathbf{U}\Sigma\mathbf{V}^\top(\mathbf{U}\Sigma\mathbf{V}^\top)^\top + \lambda I)^{-1}$$

$$\hat{W} = \mathbf{U}\Sigma\Sigma^\top\mathbf{U}^\top(\mathbf{U}\Sigma\Sigma^\top\mathbf{U}^\top + \lambda I)^{-1}$$

$$\hat{W} = \mathbf{U}\Sigma\Sigma^\top\mathbf{U}^\top\mathbf{U}(\Sigma\Sigma^\top + \lambda I)^{-1}\mathbf{U}^\top$$

$$\hat{W} = \mathbf{U}\Sigma\Sigma^\top(\Sigma\Sigma^\top + \lambda I)^{-1}\mathbf{U}^\top$$

Solution: Vector calculus approach:
Alternatively we can solve this problem by taking the gradient of the loss function with respect to the weight matrix and finding the optimal point by setting it to zero.

$$\nabla\mathcal{L}_2(W; \mathbf{X}, \lambda) = \nabla(\|\mathbf{X} - W\mathbf{X}\|_F^2 + \lambda\|W\|_F^2) = 0$$

Using property of the frobenius norm $\|A\|_F^2 = \mathrm{Tr}\left(A^T A\right)$ we can rewrite the objective function:

$$\|\mathbf{X} - W\mathbf{X}\|_F^2 + \lambda\|W\|_F^2 = \mathrm{Tr}\left((\mathbf{X} - W\mathbf{X})(\mathbf{X} - W\mathbf{X})^T\right) + \lambda\,\mathrm{Tr}\left(WW^T\right)$$

Next, expand out the terms in the trace and compute the gradient with respect to W.

$$\nabla_W\left[\mathrm{Tr}\left(\mathbf{X}\mathbf{X}^T\right) - \mathrm{Tr}\left(\mathbf{X}\mathbf{X}^T W\right) - \mathrm{Tr}\left(W\mathbf{X}\mathbf{X}^T\right) + \mathrm{Tr}\left(W\mathbf{X}\mathbf{X}^T W\right) + \lambda\,\mathrm{Tr}\left(WW^T\right)\right]$$

$$= -2\mathbf{X}\mathbf{X}^T + 2W\mathbf{X}\mathbf{X}^T + 2\lambda W = 0$$

Solving for W we get

$$W = \mathbf{X}\mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda I)^{-1}$$

The problem can then be completed by substituting in $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^\top$ as in the alternative solution.

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 λ 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 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?)

Solution:

For the first condition, we have

$$\frac{\sigma_i^2}{\sigma_i^2 + \lambda} \geq 0.8 \ \ \forall\, i \in \{1, 2, 3\}$$

Since σ_i are decreasing, we get

$$\frac{\sigma_3^2}{\sigma_3^2 + \lambda} \geq 0.8 \implies \frac{4^2}{4^2 + \lambda} \geq 0.8 \implies \lambda \leq 4$$

Similarly, solving for the second condition gives:

$$\frac{\sigma_4^2}{\sigma_4^2 + \lambda} \leq 0.6 \implies \frac{1}{1 + \lambda} \leq 0.5 \implies \lambda \geq 1$$

Thus,

$$\mathbf{1 \leq \lambda \leq 4}$$

PRINT your name and student ID: ____________________

[Doodle page! Draw us something if you want or give us suggestions or complaints. You can also use this page to report anything suspicious that you might have noticed.

If needed, you can also use this space to work on problems. But if you want the work on this page to be graded, make sure you tell us on the problem's main page.]