# CS 189/289A, Final

Fall 2025

Name: _____

Email: _____@berkeley.edu

Student ID: _____

Room where you're taking the exam: _____

Name and SID of the person on your right: _____

Name and SID of the person on your left: _____

## Instructions:

This exam consists of **84 points** spread out over **6 questions** and the Honor Code certification. The exam must be completed in **170 minutes** unless you have accommodations supported by a DSP letter.

Note that you should **select exactly one choice** for questions with **circular bubbles**. Please **fully** shade in the circle to mark your answer. For all math questions, **please simplify your answer**. Please also **show your work** if a large box is provided.

For all Python questions, you may assume `torch` has been imported and `torch.nn` has been imported as `nn`.

**You MUST write your Student ID number at the top of each page.**

## Honor Code [1 Pt]:

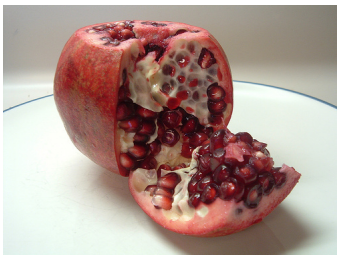
As a member of the UC Berkeley community, I act with honesty, integrity, and respect for others. I am the person whose name is on the exam, and I completed this exam in accordance with the Honor Code.

Signature: _____

This page has been intentionally left blank.

# 1 ImageNett–💯 [20 Pts]

ImageNet-1K is a curated subset of the ImageNet dataset that contains 1,281,167 images across 1,000 object categories. Each image has 3 RGB channels, but heights ($H$) and widths ($W$) vary across datapoints. A sample of images is shown below.




You want to train a model to classify images into one of the 1,000 object categories. The images in the dataset vary in size, but the model you plan to use expects inputs of the **same** dimension.

(a) [4 Pts] Decide if the following statements about data preprocessing and augmentation strategies are true or false in this setting.

(i) Resizing all images to a fixed resolution (e.g., $256 \times 256$) produces input vectors of the same length for every example, but can distort aspect ratios and shapes.

◯ **True** ◯ False

(ii) Padding each image with zeros to match a common height and width preserves aspect ratios but introduces additional background pixels that affect distance-based methods.

◯ **True** ◯ False

(iii) Randomly cropping a fixed-size patch from an image is a form of data augmentation that can improve generalization but may remove parts of visualized objects.

◯ **True** ◯ False

(iv) Standardizing pixel values in each image is sufficient to allow any model to train on variable-sized images.

◯ True ◯ **False**

For the next two parts, we work with the same data but suppose every image is resized to a fixed resolution of (Channels, Height, Width) $=$ $(\mathbf{3}, \mathbf{256}, \mathbf{256})$ **pixels**. Each image is flattened into a vector, and these vectors are stacked into a data matrix $\mathbf{X}$. Let $\mathbf{x}_n$ denote the $n$-th row of $\mathbf{X}$, corresponding to the $n$-th flattened image.

Images have a corresponding matrix of labels $\mathbf{T}$, where each row $\mathbf{t}_n$ is a one-hot vector indicating the category of the image $\mathbf{x}_n$, given by

$$t_{n,k} = \begin{cases} 1 & \text{if the image belongs to category } k \\ 0 & \text{otherwise} \end{cases}.$$

**As a reminder, the dataset contains 1,281,167 images across 1,000 categories.**

(b) [2 Pts] You first apply a $k$-nearest neighbors classifier, where $k_{\text{NN}}$ represents the number of neighbors. This classifier labels a query point $\mathbf{x}_{\text{query}}$ by the majority label among its $k_{\text{NN}}$ nearest neighbors.

(i) How does the **bias** of the $k$-nearest neighbors classifier change as the number of neighbors, $k_{\text{NN}}$, **increases**?

◯ **Bias increases** ◯ Bias decreases ◯ No change

(ii) How does the **variance** of the $k$-nearest neighbors classifier change as the number of neighbors, $k_{\text{NN}}$, **decreases**?

◯ **Variance increases** ◯ Variance decreases ◯ No change

(c) Recall that a multi-class logistic regression model is given by:

$$\mathbf{y}(\mathbf{x}; \mathbf{W}) = \text{softmax}(\mathbf{W}^\top \mathbf{x}),$$

where $\mathbf{x}$ is a flattened image vector that has been augmented with a constant 1, and $\mathbf{W}$ includes the corresponding bias weights. Suppose you would like to train a logistic regression model to predict the categories $\mathbf{T}$ of the images $\mathbf{X}$.

(i) [1 Pt] What is the dimensionality of the logistic regression model's output $\mathbf{y}(\mathbf{x}; \mathbf{W})$? *Your answer should be formatted as* $\mathbf{y}(\mathbf{x}; \mathbf{W}) \in \mathbb{R}^?$*, where* $?$ *is a* ***number****.*

**Solution:** $\mathbb{R}^{1000}$

(ii) [2 Pts] How many trainable parameters does this logistic regression model have? *Your answer should be a* ***numerical*** *expression, but you do not need to calculate it fully.*

**Solution:** $256 \times 256 \times 3 \times 1000 + 1000 = 196,609,000$ (that's a lot!)

Let $K$ be the number of categories. Assume that the logistic regression model from part (c) is trained with cross-entropy loss with $\ell_2$ regularization using Stochastic Gradient Descent (SGD) with a batch size of $\mathbf{1}$. At iteration $\tau$, the loss is given by:

$$\mathcal{L}^{(\tau)}(\mathbf{W}) \;=\; -\sum_{k=1}^{K} t_k^{(\tau)} \log y_k(\mathbf{x}^{(\tau)}; \mathbf{W}) \;+\; \frac{\lambda}{2}\|\mathbf{W}\|_2^2,$$

where $\mathbf{x}^{(\tau)}, \mathbf{t}^{(\tau)}$ are the specific datapoint and label at iteration $\tau$, respectively.

(d) [3 Pts] At iteration $\tau$, what is the gradient of the loss with respect to the weights $\mathbf{W}$?

*Hint: Suppose* $\mathbf{y} = \mathrm{softmax}(\mathbf{z})$. *The gradient with respect to* $\mathbf{z}$ *of the cross-entropy loss of the output* $\mathbf{y}$ *for a target* $\mathbf{t}$ *is equal to* $\mathbf{y} - \mathbf{t}$.

**Solution:** Let $\mathbf{y}^{(\tau)}$ be the vector whose $k$th element is $y_k(\mathbf{x}^{(\tau)}; \mathbf{W})$. From the hint, we know that $\frac{\partial \mathcal{L}^{(\tau)}}{\partial \mathbf{z}} = \mathbf{y}^{(\tau)} - \mathbf{t}^{(\tau)}$. To find $\nabla_{\mathbf{W}} \mathcal{L}^{(\tau)}$, we use the chain rule:

$$\frac{\partial \mathcal{L}^{(\tau)}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}^{(\tau)}}{\partial \mathbf{z}} \cdot \frac{\partial \mathbf{z}}{\partial \mathbf{W}}$$

Since $\mathbf{z} = \mathbf{W}^\top \mathbf{x}^{(\tau)}$, we have $\frac{\partial \mathbf{z}}{\partial \mathbf{W}} = \mathbf{x}^{(\tau)}$ (as a column vector).

Combining these, we get:

$$\frac{\partial \mathcal{L}^{(\tau)}}{\partial \mathbf{W}} = (\mathbf{y}^{(\tau)} - \mathbf{t}^{(\tau)})(\mathbf{x}^{(\tau)})^\top$$

Adding the regularization term $\frac{\lambda}{2}\|\mathbf{W}\|_2^2$, whose gradient is $\lambda \mathbf{W}$, we get:

$$\nabla_{\mathbf{W}} \mathcal{L}^{(\tau)}(\mathbf{W}) = (\mathbf{y}^{(\tau)} - \mathbf{t}^{(\tau)})(\mathbf{x}^{(\tau)})^\top + \lambda \mathbf{W}$$

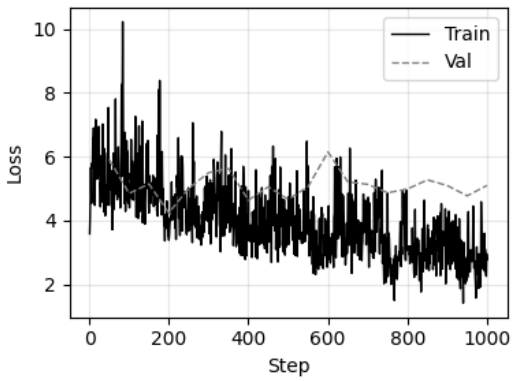
(e) [2 Pts] Using your answer to part (d), write the SGD update rule that updates $\mathbf{W}^{(\tau)}$ to $\mathbf{W}^{(\tau+1)}$, using $\eta$ as the learning rate.

**Solution:**

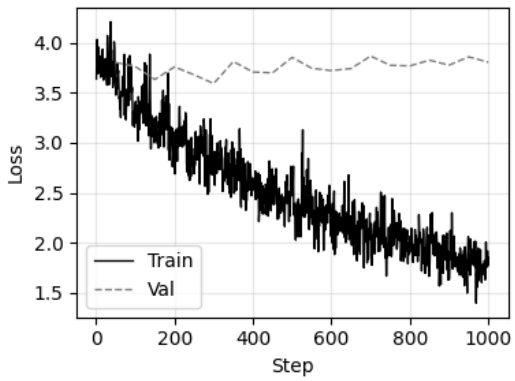
$$\begin{aligned}
\mathbf{W}^{(\tau+1)} &= \mathbf{W}^{(\tau)} - \eta \nabla_{\mathbf{W}} \mathcal{L}^{(\tau)}(\mathbf{W}^{(\tau)}) \\
&= \mathbf{W}^{(\tau)} - \eta \left[ (\mathbf{y}^{(\tau)} - \mathbf{t}^{(\tau)})(\mathbf{x}^{(\tau)})^\top + \lambda \mathbf{W}^{(\tau)} \right] \\
&= \mathbf{W}^{(\tau)} - \eta (\mathbf{y}^{(\tau)} - \mathbf{t}^{(\tau)})(\mathbf{x}^{(\tau)})^\top - \eta \lambda \mathbf{W}^{(\tau)}
\end{aligned}$$

(f) [1 Pt] You decide to train the model with a larger batch size.

Plot **A**




Plot **B**




Which plot corresponds to the loss curve with a **larger** batch size?

◯ Plot **A** uses a larger batch size.

◯ **Plot B uses a larger batch size.**

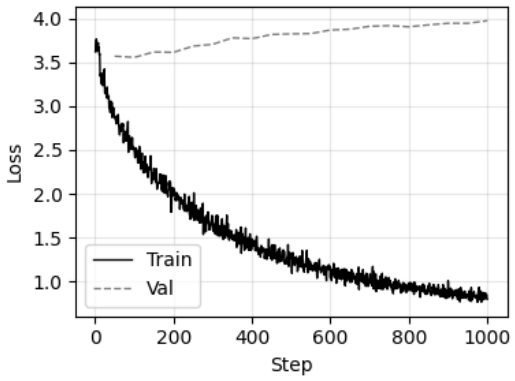
(g) [1 Pt] What is an advantage of using a larger batch size?

◯ **A larger batch size allows for greater data parallelism.**

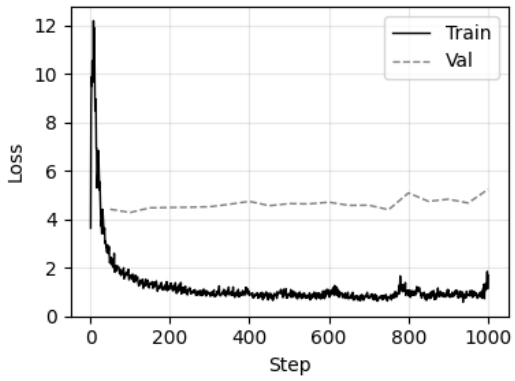
◯ A larger batch size reduces memory usage during training.

(h) [1 Pt] Keeping batch size fixed, you train the model with different learning rates.

Plot **A**




Plot **B**




Which plot corresponds to the loss curve with a **smaller** learning rate?

◯ **Plot A uses a smaller learning rate.**

◯ Plot **B** uses a smaller learning rate.

(i) [1 Pt] What is a disadvantage of using a smaller learning rate?

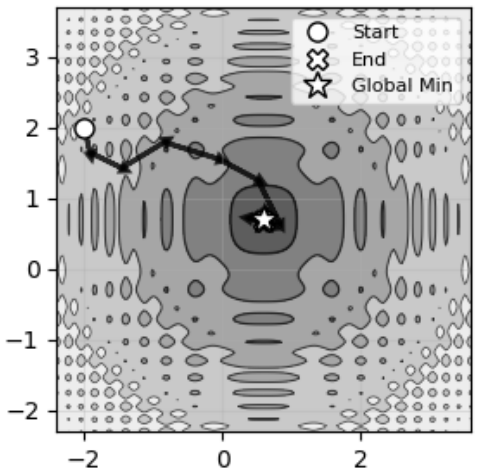
◯ Smaller learning rates may increase the computational cost per gradient update.

◯ **Smaller learning rates may require more gradient updates for the model to converge.**

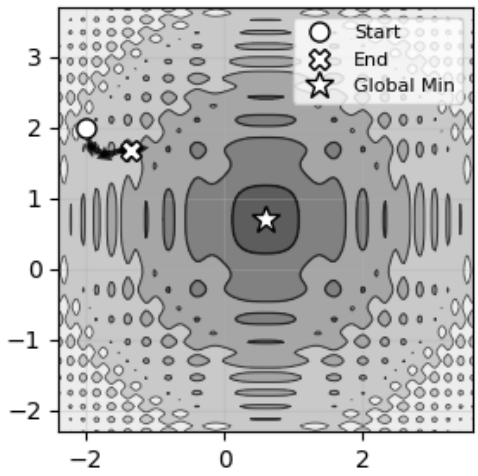
For the next two questions, suppose each contour plot represents the loss surface $\mathcal{L}(\mathbf{w})$, where the horizontal axis is $w_1$, the vertical axis is $w_2$, and the black arrows show the optimizer's trajectory as $(w_1, w_2)$ are updated.

(j) [1 Pt] You decide to use an **unregularized** loss and **SGD with momentum**.

Plot **A**




Plot **B**

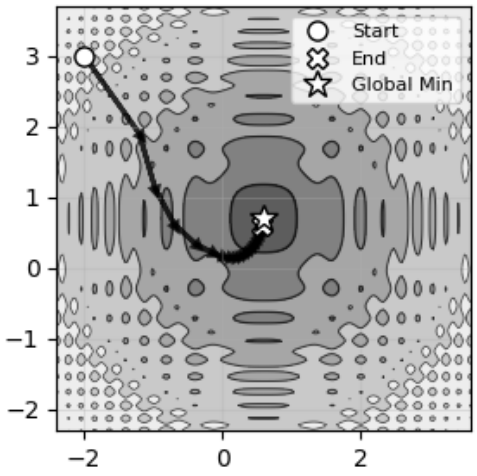



Which plot corresponds to the trajectory with the **larger** momentum term $\mu$?

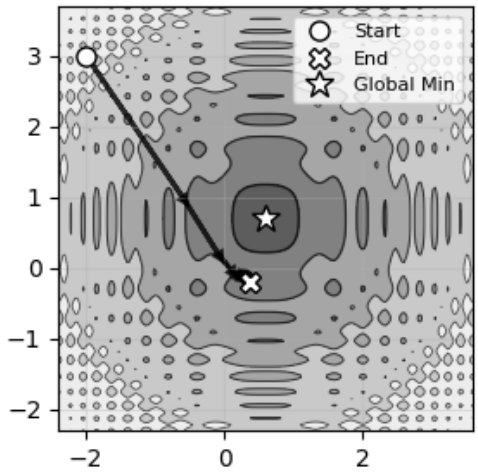
◯ **Plot A shows a larger $\mu$.** ◯ Plot **B** shows a larger $\mu$.

(k) [1 Pt] You then decide to use **unregularized** loss and **AdamW**.

Plot **A**




Plot **B**



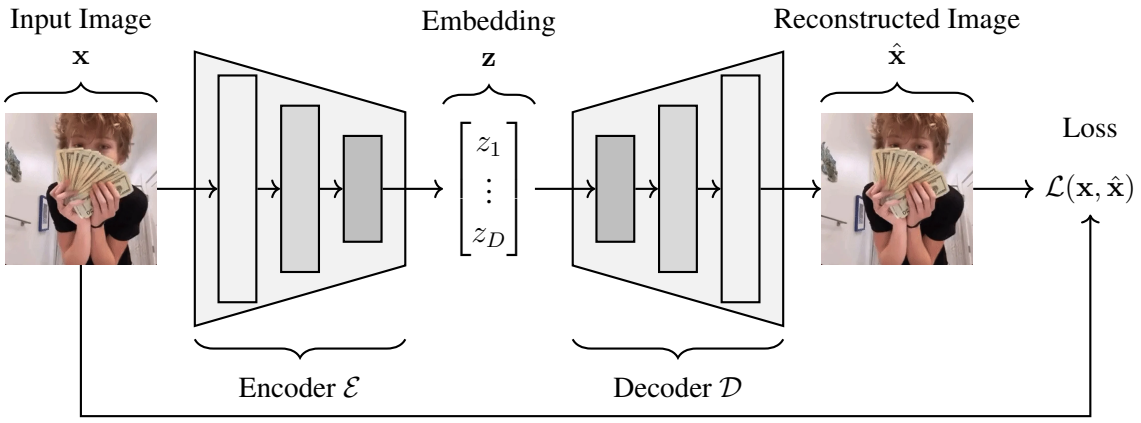

Which plot corresponds to the trajectory with the **larger** weight decay rate $\lambda$?

◯ Plot **A** shows a larger $\lambda$. ◯ **Plot B shows a larger $\lambda$.**

# 2 That One Question [15 Pts]

You found that naive logistic regression achieved poor accuracy on ImageNet-1K. Drawing on lessons from CS 189/289A, you decide to apply self-supervised learning by first training an autoencoder to produce embeddings that will support your downstream classification task.

You train an autoencoder $\{\mathcal{E}, \mathcal{D}\}$ on the ImageNet-1K dataset, which encodes input images $\mathbf{x}$ into embeddings $\mathbf{z} = \mathcal{E}(\mathbf{x})$ and decodes them into reconstructed images $\hat{\mathbf{x}} = \mathcal{D}(\mathbf{z})$.




(a) Consider the **first** layer of the CNN-based encoder. Suppose this layer uses a $5 \times 5$ kernel, with 3 input channels, 32 output channels, padding of 1, and a stride of 2.

(i) [2.5 Pts] How many trainable parameters does this layer have, including bias terms? *Your answer should be **numerical** expression, but you do not need to calculate it fully.*

**Solution:** $3 \times 32 \times 5 \times 5 + 32 \;=\; 2432$

(ii) [1.5 Pts] Recall that each image is a tensor of shape $(3, 256, 256)$ after preprocessing. What is the dimension of the first layer's output? Format answers as (C, H, W).

**Solution:** $(32, 127, 127)$

(iii) [1 Pt] Suppose we follow this layer with a mean-pooling layer. How many trainable parameters does the mean-pooling layer have?

**Solution:** $0$

(b) Suppose we build an encoder $\mathcal{E}_{(5)}$ using $L$ convolutional layers, each with 32 filters, kernels with shape $(5 \times 5)$, stride of 2, and padding of 1. Assume there are no pooling layers. This encoder outputs $\mathbf{z}_{(5)} = \mathcal{E}_{(5)}(\mathbf{x})$, which we flatten into $\mathbf{z}_{(5)} \in \mathbb{R}^{D_{(5)}}$.

Similarly, consider an encoder $\mathcal{E}_{(3)}$ built from $L$ convolutional layers, each with 32 filters, kernels with shape $(3 \times 3)$, stride of 2, and no padding. Again, assume there are no pooling layers. This encoder outputs $\mathbf{z}_{(3)} = \mathcal{E}_{(3)}(\mathbf{x})$, which we flatten into $\mathbf{z}_{(3)} \in \mathbb{R}^{D_{(3)}}$.

(i) [1 Pt] Compare the number of trainable parameters in $\mathcal{E}_{(5)}$ and $\mathcal{E}_{(3)}$.

- ◯ **$\mathcal{E}_{(5)}$ has more trainable parameters than $\mathcal{E}_{(3)}$.**
- ◯ $\mathcal{E}_{(5)}$ has fewer trainable parameters than $\mathcal{E}_{(3)}$.
- ◯ $\mathcal{E}_{(5)}$ and $\mathcal{E}_{(3)}$ have the same number of trainable parameters.

(ii) [1 Pt] Compare the sizes of $D_{(5)}$ and $D_{(3)}$.

- ◯ $D_{(5)}$ is larger than $D_{(3)}$.
- ◯ $D_{(5)}$ is smaller than $D_{(3)}$.
- ◯ **$D_{(5)}$ and $D_{(3)}$ are the equal.**

(iii) [1 Pt] Compare the receptive fields of the entries $(\mathbf{z}_{(5)})_i$ and $(\mathbf{z}_{(3)})_j$, where $i \in \{1, \ldots, D_{(5)}\}$ and $j \in \{1, \ldots, D_{(3)}\}$.

- ◯ **$(\mathbf{z}_{(5)})_i$ has a larger receptive field than $(\mathbf{z}_{(3)})_j$.**
- ◯ $(\mathbf{z}_{(5)})_i$ has a smaller receptive field than $(\mathbf{z}_{(3)})_j$.
- ◯ $(\mathbf{z}_{(5)})_i$ and $(\mathbf{z}_{(3)})_j$ have receptive fields that are the same size.

(c) Suppose we are currently training our autoencoder $\{\mathcal{E}, \mathcal{D}\}$ with a reconstruction loss, $\mathcal{L}_{\text{Recon}}(\mathbf{x}, \hat{\mathbf{x}})$, which computes the squared $\ell_2$ distance between the input image and the reconstructed image:

$$\mathcal{L}_{\text{Recon}}(\mathbf{x}, \hat{\mathbf{x}}) \;=\; \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2$$

We want to modify the loss function to encourage the encoder's embeddings $\mathbf{z} = \mathcal{E}(\mathbf{x})$ to be sparse (i.e., having many zero entries). To achieve this, we add a new term, $\mathcal{L}_{\text{Sparse}}(\mathbf{z})$, to the loss function:

$$\mathcal{L}_{\text{New}}(\mathbf{x}, \hat{\mathbf{x}}, \mathbf{z}) \;=\; \mathcal{L}_{\text{Recon}}(\mathbf{x}, \hat{\mathbf{x}}) \;+\; \lambda\, \mathcal{L}_{\text{Sparse}}(\mathbf{z})$$

(i) [2 Pts] Write a single loss function, $\mathcal{L}_{\text{Sparse}}(\mathbf{z})$, that induces sparsity in $\mathbf{z}$.

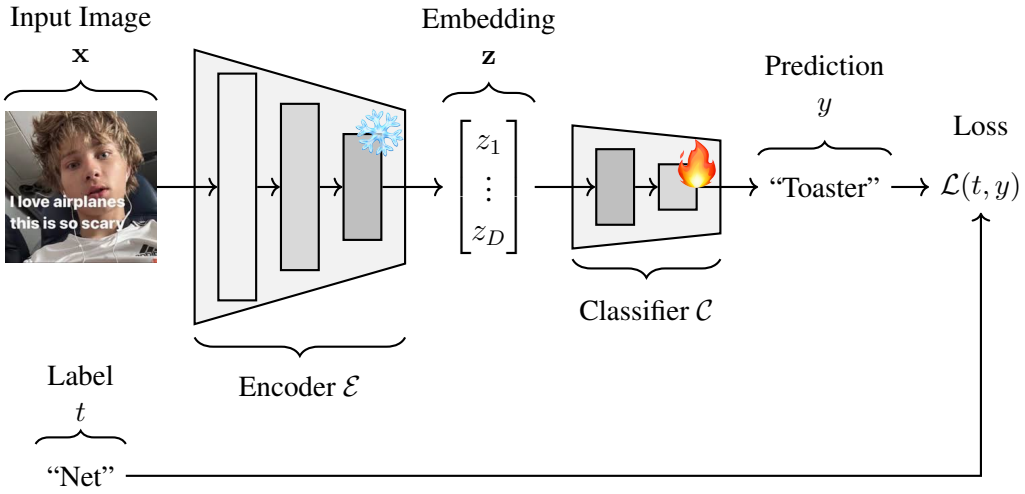
**Solution:** $\mathcal{L}_{\text{Sparse}}(\mathbf{z}) = \|\mathbf{z}\|_1$ or $\mathcal{L}_{\text{Sparse}}(\mathbf{z}) = \|\mathbf{z}\|_0$

(ii) [1 Pt] Given a training set of distinct images, is it possible for a model to achieve 0 training loss on $\mathcal{L}_{\text{New}}(\mathbf{x}, \hat{\mathbf{x}}, \mathbf{z})$?

- ◯ Yes, it is possible for a model to achieve 0 training loss on $\mathcal{L}_{\text{New}}(\mathbf{x}, \hat{\mathbf{x}}, \mathbf{z})$.
- ◯ **No, it is not possible for a model to achieve 0 training loss on $\mathcal{L}_{\text{New}}(\mathbf{x}, \hat{\mathbf{x}}, \mathbf{z})$.**

Suppose you have successfully trained an Encoder $\mathcal{E}$ and Decoder $\mathcal{D}$ on ImageNet-1K using the standard reconstruction loss $\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2$.

(d) You now freeze the encoder $\mathcal{E}$ and train a classifier to predict the category of images in ImageNet-1K.




(i) [1 Pt] Suppose the encoder has $67,000,000$ parameters. Suppose the classifier has $1,738$ parameters. What is the total **trainable** parameters in the entire model?

**Solution:** 1738

(ii) [1 Pt] In this context, the pre-trained encoder can be viewed as a:

- ◯ **A fixed basis function $\phi(\mathbf{x})$.**
- ◯ A learnable transformation updated during classifier training.
- ◯ A linear transformation that projects the data into a lower-dimensional space.
- ◯ A data augmentation module that generates synthetic data.

(iii) [1 Pt] What is an advantage of using a frozen encoder?

- ◯ **It reduces the number of trainable parameters, which may prevent overfitting.**
- ◯ It allows the encoder to update its weights to adapt to the downstream task.
- ◯ It increases the number of trainable parameters, allowing the model to fit the data.
- ◯ It guarantees the pre-trained features are perfectly aligned with the downstream task.

(iv) [1 Pt] What is a disadvantage of using a frozen encoder?

- ◯ It requires calculating gradients for the encoder, making training computationally expensive.
- ◯ **The fixed feature representations are unable to adapt to the downstream task.**
- ◯ It generally leads to greater overfitting than fine-tuning the entire model.
- ◯ It causes the encoder to forget the features learned during pre-training.

# 3 No Vibes Just Torch [18 Pts]

Each code snippet contains a **single** line of code that contains a bug. Your task is to identify the line that contains the bug and rewrite it to use the correct code.

(a) `MyConv` consists of a ReLU between two 1D convolutions. Padding is applied to maintain the shape of the tensor after each convolution.


Find and correct the bug in the `MyConv` implementation.

```
class MyConv(nn.Module):
    def __init__(self, hidden_size):
        super().__init__()
        self.conv1 = nn.Conv1d(
              in_channels=hidden_size,
              out_channels=hidden_size,
              kernel_size=3,
A             stride=1,
B             padding=1
        )
        self.activation = nn.ReLU()
        self.conv2 = nn.Conv1d(
             in_channels=hidden_size,
             out_channels=hidden_size,
             kernel_size=5,
C            stride=1,
D            padding=1
        )

    def forward(self, x):
        ... (implementation omitted)...
```

(i) [1 Pt] Identify the line that contains a bug.

◯ Line **A** ◯ Line **B** ◯ Line **C** ◯ **Line D**

(ii) [2 Pts] Rewrite the line to use the correct code.

**Solution:** `padding=2`

(b) Find and correct the bug in the `MyLN` implementation of Layer Normalization. Assume inputs are of shape (B, N, D).

```
    class MyLN(nn.Module):
        def __init__(self, feature_dim):
            super().__init__()
A           self.weight = nn.Parameter(torch.ones(feature_dim))
B           self.bias = torch.zeros(feature_dim, requires_grad=True)
C           self.norm_dim = -1

        def forward(self, x):
            mean = x.mean(dim=self.norm_dim, keepdim=True)
            std = x.std(dim=self.norm_dim, keepdim=True)
            x_norm = (x - mean) / (std + 1e-5)
D           return self.weight * x_norm + self.bias
```

(i) [1 Pt] Identify the line that contains the bug.

◯ Line **A** ◯ **Line B** ◯ Line **C** ◯ Line **D**

(ii) [2 Pts] Rewrite the line to use the correct code.

**Solution:** `self.bias = nn.Parameter(torch.zeros(feature_dim))`

(c) Find and correct the bug in the `MyDrop` implementation of a dropout layer with test-time rescaling.

```
    class MyDrop(nn.Module):
        def __init__(self, dropout_prob):
            super().__init__()
A           self.p = dropout_prob

        def forward(self, x):
            if self.training:
B               mask = (torch.rand_like(x) > self.p).float()
C               return x * mask
            else:
D               return x
```

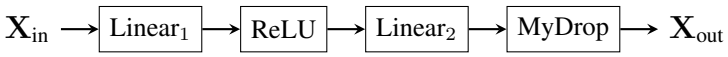
(i) [1 Pt] Identify the line that contains the bug.

◯ Line **A** ◯ Line **B** ◯ Line **C** ◯ **Line D**

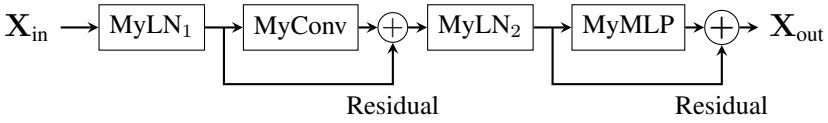
(ii) [2 Pts] Rewrite the line to use the correct code.

**Solution:** `return x * (1 - self.p)`

`MyMLP` consists of two `nn.Linear` layers with a `nn.ReLU` activation and a `MyDrop` layer.




`MyBlock` consists of two `MyLN`s, a `MyConv`, and a `MyMLP`, with residual connections.




(d) Find and correct the bug in the `MyMLP` and `MyBlock` implementation.

```
class MyMLP(nn.Module):
    def __init__(self, hidden_size, dropout_prob):
        super().__init__()
        self.linear1 = nn.Linear(hidden_size, hidden_size)
        self.activation = nn.ReLU()
        self.linear2 = nn.Linear(hidden_size, hidden_size)
        self.drop = MyDrop(dropout_prob)

    def forward(self, x):
        x_linear1 = self.linear1(x)
        x_act = self.activation(x_linear1)
        x_linear2 = self.linear2(x_act)
        x_out = self.drop(x_linear2)
        return x_out

class MyBlock(nn.Module):
    def __init__(self, hidden_size, dropout_prob):
        super().__init__()
        self.norm1 = MyLN()
        self.conv = MyConv(hidden_size)
        self.norm2 = MyLN()
        self.mlp = MyMLP(hidden_size, dropout_prob)

    def forward(self, x):
        x_in = x
        x_norm1 = self.norm1(x_in)
A       x_conv = self.conv(x_norm) + x_in
B       x_norm2 = self.norm2(x_conv)
C       x_mlp = self.mlp(x_norm2)
D       return x_mlp + x_norm2
```

(i) [1 Pt] Identify the line that contains the bug.

◯ **Line A** ◯ Line **B** ◯ Line **C** ◯ Line **D**

(ii) [2 Pts] Rewrite the line to use the correct code.

**Solution:** `x_conv = self.conv(x_norm1) + x_norm1`

(e) Find and correct the bug in the `train_epoch` function.

```
    def train_epoch(model, dataloader, device, criterion, optimizer):
        train_loss = 0
        model.train()
        for x, y in dataloader:
            x = x.to(device)
            y = y.to(device)
A           optimizer.step()
            pred = model(x)
B           loss = criterion(pred, y)
C           loss.backward()
D           optimizer.step()
            train_loss += loss.item()
        train_loss /= len(dataloader)
        return train_loss
```

(i) [1 Pt] Identify the line that contains the bug.

◯ **Line A** ◯ Line **B** ◯ Line **C** ◯ Line **D**

(ii) [2 Pts] Rewrite the line to use the correct code.

**Solution:** `optimizer.zero_grad()`

(f) Find and correct the bug in the `validate_epoch` function.

```
    def validate_epoch(model, dataloader, device, criterion):
        val_loss = 0
A       model.train()
        for x, y in dataloader:
B           x, y = x.to(device), y.to(device)
C           pred = model(x)
D           loss = criterion(pred, y)
            val_loss += loss.item()
        val_loss /= len(dataloader)
        return val_loss
```

(i) [1 Pt] Identify the line that contains the bug.

◯ **Line A** ◯ Line **B** ◯ Line **C** ◯ Line **D**

(ii) [2 Pts] Rewrite the line to use the correct code.

**Solution:** `model.eval()`

# 4 Attention is All You Knead [10 Pts]

Remy scrambled his epic transformer recipe! He needs help identifying the ingredients for a 1-head attention block processing $N = 2$ tokens. The relevant matrices and their dimensions are:

- Input embeddings $\mathbf{X} \in \mathbb{R}^{N\times 3}$ (so $d_{model} = 3$)
- Queries and Keys $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{N\times 3}$ (so $d_k = 3$)
- Values $\mathbf{V} \in \mathbb{R}^{N\times 4}$ (so $d_v = 4$)

Because the recipe was scrambled, Remy isn't sure which of the matrices below corresponds to each step in the attention mechanism.

$$\mathbf{A} = \begin{bmatrix} -0.3 & 1.7 & 4.2 & 2.5 \\ -3.9 & -0.8 & -2.1 & -4.7 \\ 1.5 & 3.3 & 0.4 & -2.9 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1.2 & -3.8 & 0.6 \\ 2.9 & -1.4 & 0.7 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 0.7 & 0.3 \\ 0.5 & 0.5 \end{bmatrix}$$

$$\mathbf{D} = \begin{bmatrix} 1.0 & -0.2 & 0.3 \\ 2.2 & -0.5 & 2.2 \\ 3.0 & -1.2 & -0.7 \end{bmatrix} \quad \mathbf{E} = \begin{bmatrix} -0.3 & 2.3 \\ 0.9 & -6.7 \end{bmatrix} \quad \mathbf{F} = \begin{bmatrix} 0.4 & 0.2 \\ 0.6 & 0.8 \end{bmatrix}$$

(a) For each of the following steps in the transformer's attention mechanism, select which of the matrices could have been the output of that step. There is exactly **one** correct answer for each subpart in this question.

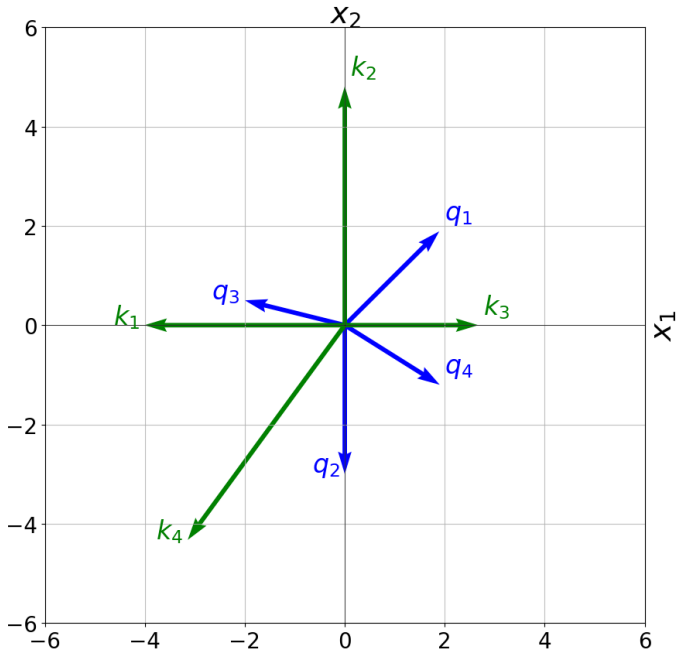
(i) [1 pt] Which of the above matrices could correspond to $\mathbf{QK}^\top$?

◯ **A** ◯ **B** ◯ **D** ◯ **E**

(ii) [1 pt] Which of the above matrices could correspond to softmax$(\mathbf{QK}^\top)$?

◯ **C** ◯ **D** ◯ **E** ◯ **F**

(b) [4 Pts] We now assume that the transformer's query and key vectors are 2-dimensional vectors. Remy passes 4 input tokens into the transformer's first attention block and plots the query $(q_1, \ldots, q_4)$ and key $(k_1, \ldots, k_4)$ vectors in the plot below.




Let $a_{ij}$ be the attention score indicating how much the $i$-th input token attends to the $j$-th input token. Based on the plot, fill in the boxes below to compare the two given attention scores with **exactly one** of $<$ , $=$ , or $>$.

(i) $a_{12}$ ☐ $a_{13}$ (ii) $a_{14}$ ☐ $a_{41}$

(iii) $a_{24}$ ☐ $a_{23}$ (iv) $a_{31}$ ☐ $a_{33}$

**Solution:**

(i) $a_{12} > a_{13}$

(ii) $a_{14} < a_{41}$

(iii) $a_{24} > a_{23}$

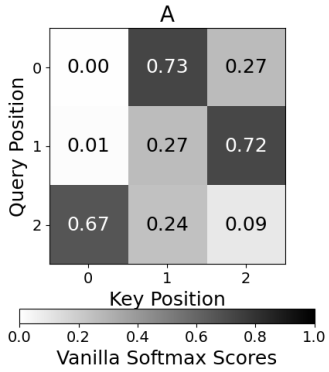
(iv) $a_{31} > a_{33}$

(c) Remy introduces $\mathrm{RemySoftmax}$, a variant of temperature-scaled softmax. For attention scores $a_{ij}$, the $\mathrm{RemySoftmax}$ score is defined as:

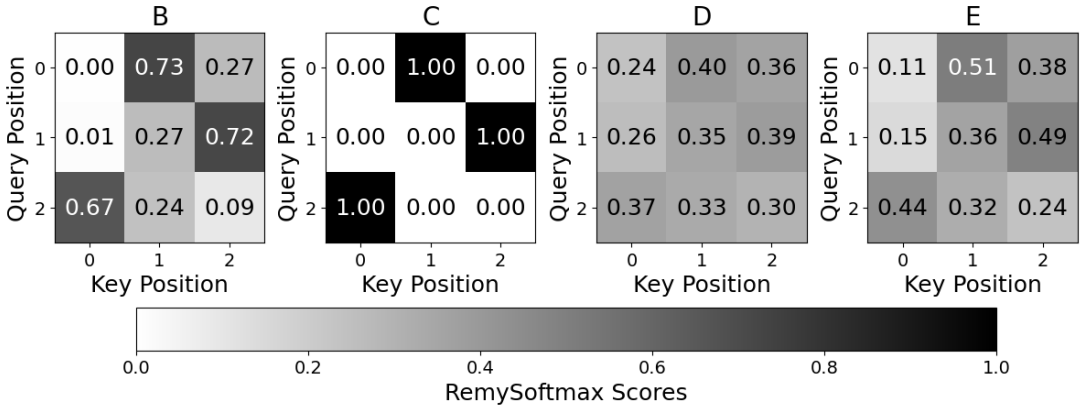
$$\mathrm{RemySoftmax}(a_{ij}) = \frac{\exp(\alpha \cdot a_{ij} + \beta)}{\sum_{j=1}^{N} \exp(\alpha \cdot a_{ij} + \beta)},$$

where $\alpha, \beta \in \mathbb{R}$ are constant factors.

Consider the heatmap of the original attention matrix $A$ below, whose $i^{\text{th}}$,$j^{\text{th}}$ element is the attention score $a_{ij}$ after applying the vanilla softmax function:




The heatmaps below are obtained by applying $\mathrm{RemySoftmax}$ using four different pairs of $(\alpha, \beta)$ values.




(i) [1 pt] Which heatmap corresponds to $\alpha = 0.1, \beta = 1$?

◯ **B** ◯ **C** ◯ **D** ◯ **E**

(ii) [1 pt] Which heatmap corresponds to $\alpha = 0.3, \beta = 10$?

◯ **B** ◯ **C** ◯ **D** ◯ **E**

(iii) [1 pt] Which heatmap corresponds to $\alpha = 1, \beta = 10$?

◯ **B** ◯ **C** ◯ **D** ◯ **E**

(iv) [1 pt] Which heatmap corresponds to $\alpha = 10, \beta = 1$?

◯ **B** ◯ **C** ◯ **D** ◯ **E**

**Solution:**

i. $\alpha = 0.1, \beta = 1$ corresponds to $D$

ii. $\alpha = 0.3, \beta = 10$ corresponds to $E$

iii. $\alpha = 1, \beta = 10$ corresponds to $B$

iv. $\alpha = 10, \beta = 1$ corresponds to $C$

Observe that adding $\beta$ has no effect on the heatmap because it is a constant shift of all the attention scores. Mathematically, we can show that:

$$\begin{aligned}
\text{RemySoftmax}(a_{ij}) &= \frac{\exp(\alpha \cdot a_{ij} + \beta)}{\sum_{j=1}^{N} \exp(\alpha \cdot a_{ij} + \beta)} \\
&= \frac{\exp(\alpha \cdot a_{ij}) \cdot \exp(\beta)}{\sum_{j=1}^{N} \exp(\alpha \cdot a_{ij}) \cdot \exp(\beta)} \\
&= \frac{\exp(\alpha \cdot a_{ij})}{\sum_{j=1}^{N} \exp(\alpha \cdot a_{ij})} \\
&= \text{Softmax}(a_{ij})
\end{aligned}$$

The scaling term $\alpha$ controls the "sharpness" of the RemySoftmax function: a higher $\alpha$ will result in a more concentrated distribution of attention scores, while a lower $\alpha$ will result in a more uniform distribution of attention scores.

- When $\alpha$ is very large, almost all the terms in the exponent of the softmax function will be very small and similiar to each other. Thus, the softmax function will be very close to a uniform distribution.
- On the other hand, when $\alpha$ is very small, the larger the pre-softmax attention score, the faster its value grows, and the higher the exponent $e^{a_{ij}/\alpha}$ will be. Thus, the softmax function will be very dominated by the maximum value in each row.

The heatmap that is most likely to correspond to $\alpha = 0.1, \beta = 1$ is the one that has the most uniform values: **choice D**.

For $\alpha = 0.3, \beta = 10$, the heatmap these parameters best match is **choice E** because the attention scores are still relatively uniform, but more concentrated than $D$. Note that since adding $\beta$ has no effect on the heatmap, it is sufficient to only consider the scaling term $\alpha$ when comparing which heatmap should correspond to $\alpha = 0.1$ versus $\alpha = 0.3$.

For $\alpha = 1, \beta = 10$, since the shifting term $\beta$ has no effect on the softmax distribution, the heatmap of this RemySoftmax will be identical to the heatmap of vanilla softmax **(choice B)**.

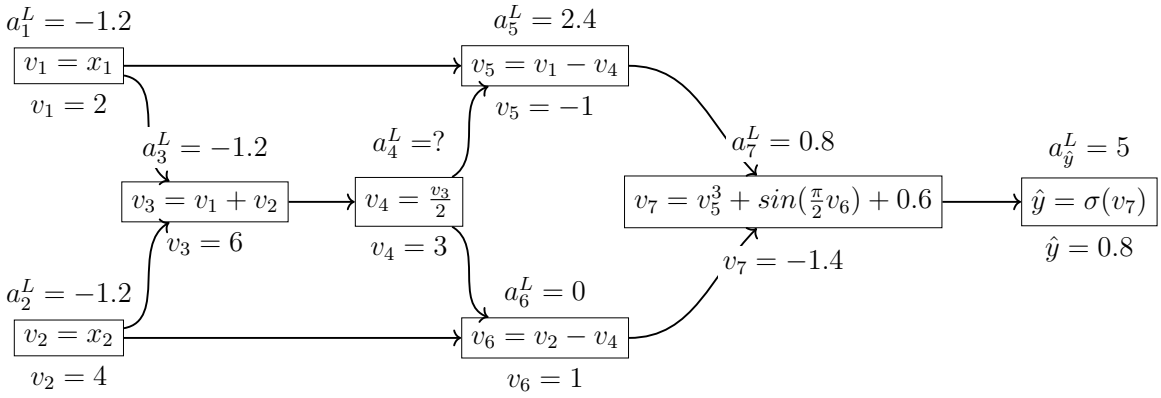
Finally, for $\alpha = 10, \beta = 1$, the heatmap that best matches these parameters is **choice C** because the attention scores are most concentrated in the cells that had the highest vanilla-softmax attention scores.

# 5 Rat-propagation [10 Pts]

Remy wants to design a model to predict whether or not food critic Anton Ego will like his ratatouille. The last layer of his model is represented by the following computational graph, which takes in two scalar inputs, $v_1$ and $v_2$, and produces a scalar output $\hat{y}$.

- $\sigma$ is the sigmoid activation function $\sigma(\gamma) = \dfrac{1}{1+e^{-\gamma}}$.
- $\sin$ is the sine function.
- $v_1$ and $v_2$ are the inputs from one datapoint passed to the last layer.

Remy performs one forward pass of the model with the inputs $v_1 = 2$ and $v_2 = 4$ and records the intermediate values $(v_3, \ldots, v_7)$ of his model's last layer in his computational graph below each node. He also records the adjoint variables $a_i^L = \frac{\partial L}{\partial v_i}$ in the computational graph above each node.




(a) [3 Pts] Given the computational graph above, write out the full expression for $\hat{y}$ in terms of **only** $v_1$ and $v_2$. You are allowed to include constants (such as $\pi$ and $e$) and the sigmoid and sine functions in your final answer.

**Solution:**

$$\begin{aligned}
\hat{y} &= \sigma(v_7) \\
&= \sigma(v_5^3 + sin(\frac{\pi}{2}v_6) + 0.6) \\
&= \sigma((v_1 - v_4)^3 + sin(\frac{\pi}{2}(v_2 - v_4)) + 0.6) \\
&= \sigma((v_1 - \frac{v_1 + v_2}{2})^3 + sin(\frac{\pi}{2}(v_2 - \frac{v_1 + v_2}{2})) + 0.6)
\end{aligned}$$

(b) [3 Pts] Suppose Remy knows that the downstream gradient with respect to the output is $\frac{\partial L}{\partial \hat{y}} = 5$. Derive $\frac{\partial L}{\partial v_4}$, the gradient of the loss $L$ with respect to $v_4$. (*Hint: You may use the adjoints provided on the computational graph.*)

You may use the large box below to show your work, but your final answer **must be written in the final answer box** to receive full credit. Your final answer should be a single **number**.

Final answer:

**Solution:** Define the adjoint variable $a_i^L = \frac{\partial L}{\partial v_i}$.

To find the derivative of the loss with respect to $v_4$, i.e. $a_4^L$, we need to consider all the paths through which $v_4$ affects the final loss $L$:

$$a_4^L = \sum_{i \in Ch(v_4)} a_i^L \cdot \frac{\partial v_i}{\partial v_4}$$

The children of $v_4$ are $v_5$ and $v_6$, so we have:

$$a_4^L = a_5^L \cdot \frac{\partial v_5}{\partial v_4} + a_6^L \cdot \frac{\partial v_6}{\partial v_4}$$

The local partial derivatives are:

$$\frac{\partial v_5}{\partial v_4} = -1$$
$$\frac{\partial v_6}{\partial v_4} = -1$$

Therefore:

$$\begin{aligned}\frac{\partial L}{\partial v_4} &= a_4^L \\ &= a_5^L \cdot (-1) + a_6^L \cdot (-1) \\ &= -a_5^L - a_6^L \\ &= -2.4 - 0 \\ &= -2.4\end{aligned}$$

(c) Remy decides to try using a multi-layer neural network to predict the ratings that Anton Ego would give different restaurants in Berkeley. He wants to try applying different normalization techniques in his neural network.

Assume that the inputs to layer $l$ are $X \in \mathbb{R}^{B \times n_l}$, where $B$ is the batch size and $n_l$ is the number of neurons in layer $l$. $\gamma$ (the scaling factor) and $\beta$ (the shifting factor) are learnable parameters that allow the normalization layer to scale and shift the normalized distribution.

- Let $x_i$ be the $i$-th datapoint in the batch at layer $l$, i.e. the $i$-th row of $X$.
- Let $x_{i,j}$ be the value of the $j$-th neuron in layer $l$ for the $i$-th datapoint.

(i) [1 pt] Which of the following correctly computes the mean for LayerNorm?

◯ $\mu_i = \frac{1}{n_l} \sum_{j=1}^{n_l} x_{i,j}, \ i \in \{1, \ldots, B\}$

◯ $\mu_i = \frac{1}{n_l} \sum_{i=1}^{B} \sum_{j=1}^{n_l} x_{i,j}, \ i \in \{1, \ldots, B\}$

◯ $\mu_i = \frac{1}{n_l} \sum_{i=1}^{n_l} x_{i,j}, \ j \in \{1, \ldots, n_l\}$

◯ $\mu_i = \frac{1}{B} \sum_{i=1}^{B} \sum_{j=1}^{n_l} x_{i,j}, \ j \in \{1, \ldots, B\}$

**Solution:** Layernorm is applied to each datapoint in the batch independently, so the mean is calculated for each datapoint separately. This means that the layerwise mean for the $i$-th datapoint is the average over all the components of $x_i$. Therefore, the correct equation is A: $\mu_i = \frac{1}{n_l} \sum_{j=1}^{n_l} x_{i,j}$ for $i = 1, 2, \ldots, B$.

(ii) [1 pt] For the $i$-th datapoint, assume layerwise mean $\mu_i$, standard deviation $\sigma_i$, and small positive constant $\epsilon$. What is the $j$-th component of the LayerNorm output $y_i$?

◯ $y_{i,j} = \frac{x_{i,j} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$

◯ $y_{i,j} = \gamma \cdot \frac{x_{i,j} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} + \beta$

◯ $y_{i,j} = \frac{x_{i,j} - \mu_i}{\sigma_i^2 + \epsilon}$

◯ $y_{i,j} = \gamma \cdot \frac{x_{i,j} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} + \beta \cdot \mu_i$

**Solution:** LayerNorm consists of two steps: normalization and a shift/scale.

First, we normalize each component by subtracting the mean and dividing by the standard deviation (with $\epsilon$ added for numerical stability):

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

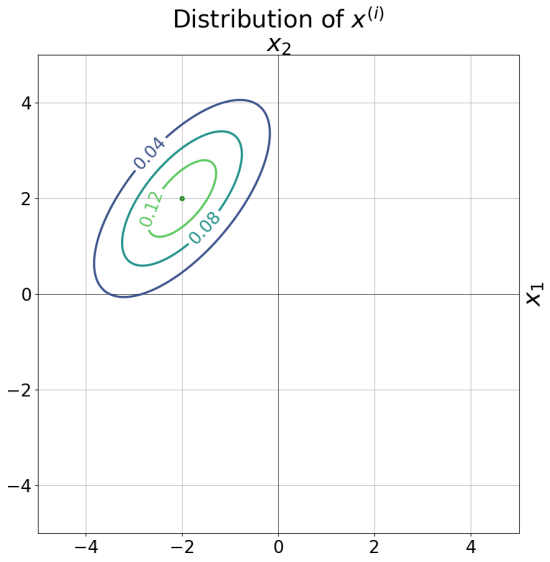
However, this normalization can limit the representational power of the network. Therefore, LayerNorm includes learnable affine parameters $\gamma$ (scale) and $\beta$ (shift)

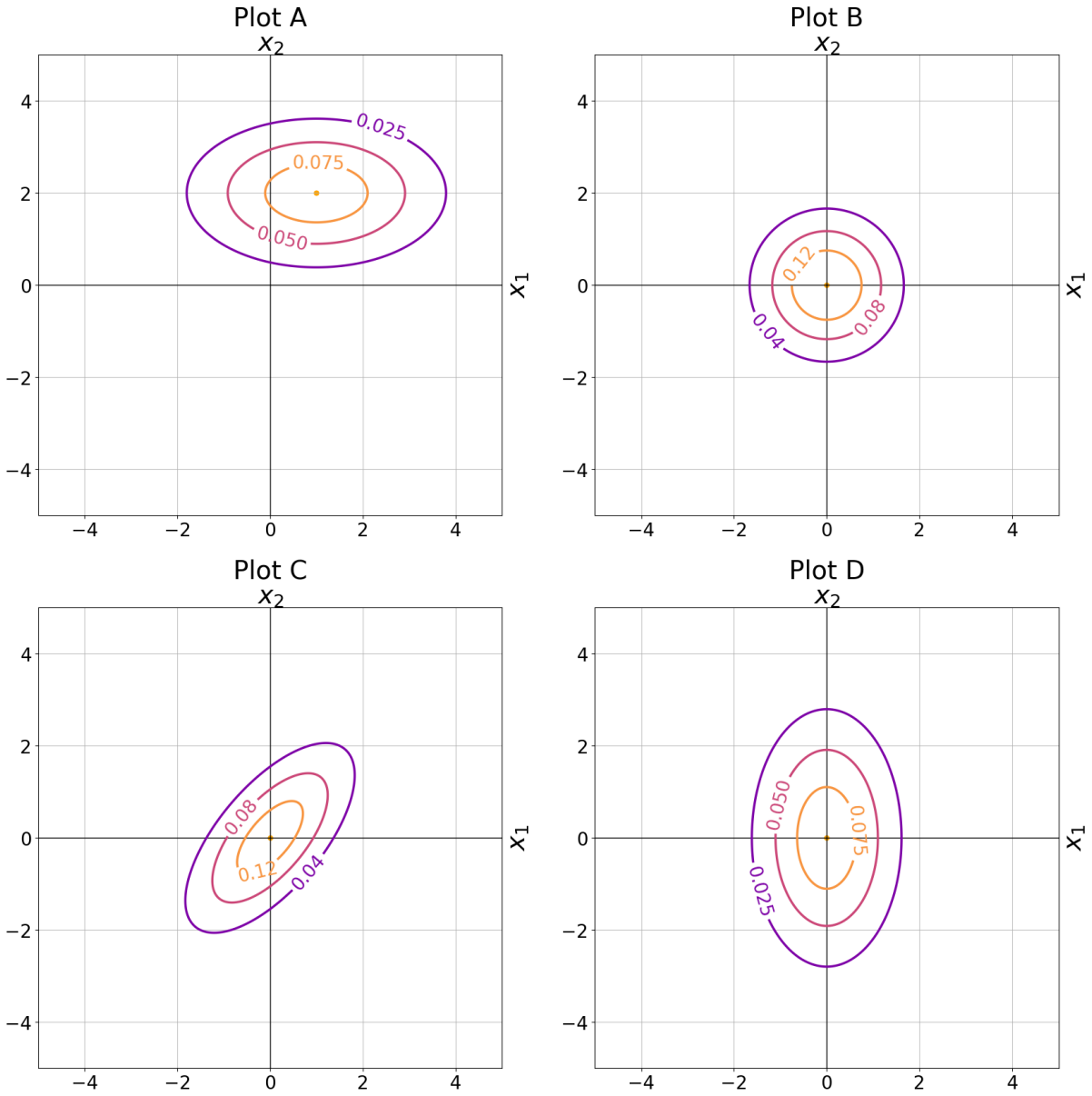
that are applied element-wise:

$$y_{i,j} = \gamma \cdot \hat{x}_{i,j} + \beta = \gamma \cdot \frac{x_{i,j} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} + \beta$$

11

(d) Assume that the input is a batch $\mathbf{X}_{\text{batch}} \in \mathbb{R}^{B \times 2}$ of $B$ vectors, where the $i$-th row is $\mathbf{x}^{(i)} = \begin{bmatrix} x_1^{(i)} & x_2^{(i)} \end{bmatrix}^\top$. Remy fits a 2D Gaussian to the data and plots it.






(i) [1 pt] Which plot shows the output distribution after applying a BatchNorm layer with parameters $\gamma = \begin{bmatrix} 1 & 1 \end{bmatrix}^\top$ and $\beta = \begin{bmatrix} 0 & 0 \end{bmatrix}^\top$?

◯ Plot **A** ◯ **Plot B** ◯ Plot **C** ◯ Plot **D**

**Solution:** The correct answer is B. Batchnorm first demeans and normalizes the data so that all dimensions have unit variance. This results in the distribution being circular isocontours centered at the origin. Then, it scales and shifts the data by the learnable parameters $\gamma$ and $\beta$.

Since $\beta = [0, 0]$, the distribution is not shifted. Since $\gamma = [1, 1]$, the distribution is scaled by 1 in both dimensions.

(ii) [1 pt] Which plot shows the output distribution after applying a BatchNorm layer with parameters $\gamma = \begin{bmatrix} 3 & 1 \end{bmatrix}^\top$ and $\beta = \begin{bmatrix} 1 & 2 \end{bmatrix}^\top$?

◯ **Plot A** ◯ Plot **B** ◯ Plot **C** ◯ Plot **D**

**Solution:** The correct answer is A.

With a gamma of $[3, 1]$, the circular isocontours are scaled by 3 in the $x_1$ direction and 1 in the $x_2$ direction. This results in a elliptical whose horizontal axis is 3 times as long as the vertical axis for each isocontour value.

The beta $[1, 2]$ shifts the isocontours so that they are centered at $(1, 2)$.

# 6 Everything, Everywhere, All At Once [10 Pts]

(a) [4 Pts] Match each machine learning algorithm to the task it would be **most applicable** to. Each algorithm should be matched to **exactly one** task.

(A) Forecast crop yield from dozens of environmental factors, selecting only the few factors that truly influence growth.

(B) Produce new images of ways to take back the axe from Stanford starting from noise.

(C) Classify different road signs from a Waymo vehicle's camera feed.

(D) Generate the next DNA nucleotide (A, G, C, T) in a DNA sequence.

(E) Identify defective products on a factory line by learning to reconstruct normal items and flagging those that don't match the learned patterns.

(F) Group patients based on lab results, where some patients show characteristics of more than one condition.

(G) Predict a boba shop's rating based on its price, distance, and number of toppings.

(H) Predict whether an email is spam based on its content.

| **Algorithm** | **Task (letter)** |
|---|---|
| GMM | |
| Linear Regression | |
| Linear Regression with L1 Regularization | |
| Logistic Regression | |
| CNNs | |
| Transformers | |
| Autoencoders | |
| Diffusion Models | |

**Solution:**

(i) GMM – F: Group patients based on lab results, where some patients show characteristics of more than one condition.

(ii) Linear Regression - G: Predict the rating of a boba shop based on its prices, distance from campus, and number of toppings.

(iii) Linear Regression with L1 Regularization – A: Forecast crop yield from dozens of environmental factors, selecting only the ones that truly influence growth.

(iv) Logistic Regression – H: Predict whether an email is spam based on its content.

(v) CNNs – C: Classify different road signs from a Waymo vehicle's camera feed.

(vi) Transformers – D: Generate the next DNA nucleotide (A, G, C, T) in a DNA sequence.

(vii) Autoencoders – E: Identify defective products on a factory line by learning to reconstruct normal items and flagging those that don't match the learned patterns.

(viii) Diffusion Models – B: Produce new images of ways to take back the axe from Stanford starting from noise.

(b) [1 Pt] Which of the following is **not** a means of addressing the "Curse of Dimensionality"–the phenomenon where, as the number of dimensions in a dataset increases, it becomes more difficult for algorithms to identify meaningful patterns in the data?

- ◯ Using a larger dataset
- ◯ **Reducing the variance of your model**
- ◯ Defining a different distance metric
- ◯ Reducing the number of features of your data

**Solution:** Reducing the variance of your model *is not* a solution to the "Curse of Dimensionality".

The "Curse of Dimensionality" refers to the fact that as you increase the number of features in your data, your datapoints will become very sparse. Almost all your data will lie far apart from each other in this high-dimensional space.

In Professor Efros's guest lecture, he mentioned three solutions to the curse of dimensionality:

1. Using more data
2. Defining a different kind of distance metric
3. Dimensionality reduction (e.g. PCA)

(c) [1 Pt] Assume you have a large language model that has been adequately pre-trained on a large dataset of text collected before 2020 but **has not been post-trained yet**.

You provide this input: "Who won the last World Cup?"

What behavior would you expect from this pre-trained-only model?

- ◯ It would search the web and report the most recent winner.
- ◯ It would answer based on its knowledge cutoff (e.g., "France won in 2018").
- ◯ **It would continue the text pattern, likely generating similar questions.**
- ◯ It would repeat the last token in the input indefinitely.

**Solution:** C: It would continue the text pattern, likely generating similar questions

A pre-trained language model is fundamentally an autoregressive text completion system. Without post-training, it does NOT: - Understand that text is a "question to answer" - Have instruction-following capabilities - Know to provide factual responses

Instead, it predicts the most likely continuation based on training data patterns. The input resembles a list of sports trivia questions (common in web text, quiz sites, etc.), so the

model will likely generate more questions like "Who won the last World Series? Who won the last Super Bowl?" - treating the input as an incomplete list to continue.

Post-training techniques (instruction-tuning, RLHF) teach the model to recognize queries as requests for information and respond accordingly.

(d) [1 Pt] Which of the following is a **true** statement about post-training and fine-tuning large language models?

- ◯ It's impossible to fine-tune a large language model using chat conversations from another type of LLM because the architecture details of the two models will be different.
- ◯ Learning the matrices $A \in \mathbb{R}^{D \times r}$ and $B \in \mathbb{R}^{r \times D}$ (where $AB$ is a low-rank approximation to $W \in \mathbb{R}^{D \times D}$) during Low Rank Adaptation fine-tuning requires learning the same number of parameters as fine-tuning the original $W$ matrix itself.
- ◯ You should raise the learning rate during fine-tuning on a new task because the model already knows general patterns from pre-training.
- ◯ **RLHF (Reinforcement Learning from Human Feedback) is more human-labor intensive than instruction fine-tuning.**

**Solution:** Choice D: RLHF (Reinforcement Learning from Human Feedback) is more human-labor intensive than instruction fine-tuning. RLHF usually requires collecting human preferences and labelling/scoring model outputs (either manually or using another reward model) to guide the model's behavior.

**Choice A** is **false** because the paper "Instruction Fine-Tuned LLMs" showed that it was possible to fine-tune an open-source chatbot on ShareGPT conversations to provide responses that were judged to be on par with over 90% of ChatGPT's original responses.

**Choice B** is **false** because learning the matrices $A \in \mathbb{R}^{D \times r}$ and $B \in \mathbb{R}^{r \times D}$ requires learning $2 \times D \times r$ new parameters compared to the $D \times D$ parameters in the original $W$ matrix. If $r$ is smaller than $D$ (which is usually the case for a low-rank approximation), this will require significantly less computation than fine-tuning the original $W$ matrix itself.

**Choice C** is **false** because you should use a lower learning rate when fine-tuning a model on a new task to avoid catastrophic forgetting. When fine-tuning a model on a new task, you want to avoid moving the model's weights too far such that it forgets the original pre-trained tasks.

(e) [1 Pt] Which of the following is a **true** statement about different LLM applications?

- ◯ **No gradient updates are performed during few-shot prompting.**
- ◯ RAG (Retrieval-Augmented Generation) requires training new parameters so the model learns how to read retrieved documents.

◯ Chain-of-thought prompting generally requires less test-time compute than zero-shot prompting because the model is more likely to answer the question correctly when it states its reasoning.

◯ Even with agentic systems, LLMs can't answer questions about events past their training cut-off date.

**Solution:** Choice A: Few-shot prompting does not involve any gradient updates. It is a form of in-context learning where the model learns how to answer questions based on the examples provided.

**Choice B** is **false** because RAG (Retrieval-Augmented Generation) does not require training new parameters. In RAG, we provide relevant documents to the model by including it in its context window.

**Choice C** is **false** because chain-of-thought prompting generally requires more test-time compute than zero-shot prompting, as the model needs to output its reasoning process.

**Choice D** is **false** because agentic systems can allow LLMs to access information outside of what it was trained on (e.g. by searching the web for today's weather).

(f) [1 Pt] Based on the paper "Deep Residual Learning for Image Recognition", which of the following is a **true** statement about residual connections?

◯ **If the original input $x$ and the sublayer output $F(x)$ have different dimensions, padding the original input with zeros so that it matches the dimensions of the sublayer's output will add 0 new learnable parameters.**

◯ Not adding any residual connections in their model at all performed better than adding residual connections.

◯ Applying a $1 \times 1$ convolution (with no bias) to project the original input with $c_{in}$ channels to match a sublayer output with $c_{out}$ channels adds $c_{in} \times c_{in}$ new learnable parameters.

◯ You always have to apply a $1 \times 1$ projection to the original input even if the sublayer output has the same dimensions as the original input.

**Solution:** **Choice A**: if the original input $x$ and the sublayer output $F(x)$ have different dimensions, padding the original input with zeros so that it matches the dimensions of the sublayer's output will add 0 new learnable parameters.

Padding the original input with zeros does not introduce any new learnable parameters because we never update/learn the padded zeros.

**Choice B** is false because the authors of the paper found that adding residual connections tended to increase train/test accuracy and resulted in more effective training.

**Choice C** is false because applying a 1x1 convolution (with no bias) to project the original input with $c_{in}$ channels to match a sublayer output with $c_{out}$ channels adds $c_{in} \times c_{out} \times h_{kernel} \times w_{kernel} = c_{in} \times c_{out}$ new learnable parameters.

**Choice D** is false because in the paper, the authors also explored architectures that directly added the input to the output of the sublayer when they had an equal number of channels.

(g) [1 Pt] Where do the query (Q), key (K), and value (V) vectors come from in a decoder-encoder transformer's **decoder cross-attention** layer?

- ◯ Q, K, and V all come from the encoder's final hidden states.
- ◯ **Q comes from the decoder's current hidden states, while K and V come from the encoder's final hidden states.**
- ◯ Q and K come from the decoder's current hidden states, while V comes from the encoder's input embeddings.
- ◯ Q, K, and V all come from the decoder's current hidden states.

**Solution:**

Choice B: Q comes from the decoder's current hidden states, while K and V come from the encoder's final hidden states.

In a transformer's decoder cross-attention layer, the decoder attends to the encoder's output representations. The decoder generates the query (Q) vectors from its current hidden states, reflecting what it is trying to predict next. The key (K) and value (V) vectors come from the encoder's final hidden states, which contain contextual information about the input sequence. This structure allows the decoder to condition its predictions on the encoded source sentence while generating the output step by step.

**You are done with the final! Congratulations!**

What was your favorite part of CS 189/289A? Also, share any feedback you have in the box below.