EECS 182 Deep Neural Networks
Fall 2025 Anant Sahai and Gireeja Ranade

# Homework 5

**This homework is due on Friday, October 10, 2025, at 10:59PM.**

## 1. Convolutional Networks

Note: Throughout this problem, we will use the convention of NOT flipping the filter before dragging it over the signal. This is the standard notation with neural networks (ie, we assume the filter given to us is already flipped)

(a) **List two reasons we typically prefer convolutional layers instead of fully connected layers when working with image data.**

**Solution:** The basic reason is so that we can get an inductive bias that is suited for the regularities often present in image data so that we can learn without having to use inordinate amounts of training data. Valid reasons for using convolutional layers to do this include:

- Convolutions with finite sized filters respect the kind of locality and by layering them, we can reflect the heirarchical part/whole structure decompositions that we see in images.
- Convolutions are nicely compatible with weight-sharing which builds in a basic invariance/equivariance to translations which is reasonable since in most image-related contexts, the desired answers for a translated image are closely related to the original image. The weight-sharing allows the numbers of parameters to be much smaller.
- Convolutions build in the idea that left/right and up/down can be different and allow different weights for those.
- There is a long history of using filterbanks and convolutional structures in hand-designed approaches to computer vision and image processing so it is natural to simply replace those with learned weights in similar structures.
- There is biological inspiration based on what we know about how eyes work for the locality properties and the kinds of edge-detecting responses that convolutions capture naturally.
- Furthermore, other people have used convolutional layers to great success when working with vision and in Deep Learning, it is natural to try to build on the successes of others even when we don't understand why they succeeded.

(b) Consider the following 1D signal: $[1, 4, 0, -2, 3]$. After convolution with a length-3 filter, no padding, stride=1, we get the following sequence: $[-2, 2, 11]$. **What was the filter?**

*(Hint: Just to help you check your work, the first entry in the filter that you should find is 2. However, if you try to use this hint directly to solve for the answer, you will not get credit since this hint only exists to help you check your work.)*

**Solution:** If we represent the original filter as $[h_1, h_2, h_3]$, we can set up the system of equations:

$$\begin{aligned} h_1 + 4h_2 + 0h_3 &= -2 \\ 4h_1 + 0h_2 - 2h_3 &= 2 \\ 0h_1 - 2h_2 + 3h_3 &= 11 \end{aligned}$$

Solving, we get $[h_1, h_2, h_3] = [2, -1, 3]$

This could be done using Gaussian elimination or substitution. By substitution, we see $h_1 = -2 - 4h_2$ and $h_3 = \frac{11}{3} + \frac{2}{3}h_2$. Plugging into the second equation we get $-8 - 16h_2 - \frac{22}{3} - \frac{4}{3}h_2 = 2$ which simplifies to $\frac{52}{3} = -\frac{52}{3}h_2$ which solves to $h_2 = -1$. Back substituting gives us $h_1 = 2$ and $h_3 = \frac{9}{3} = 3$.

The Gaussian elimination approach involves similar numbers.

(c) Transpose convolution is an operation to help us upsample a signal (increase the resolution). For example, if our original signal were $[a, b, c]$ and we perform transpose convolution with pad=0 and stride=2, with the filter $[x, y, z]$, the output would be $[ax, ay, az + bx, by, bz + cx, cy, cz]$. Notice that the entries of the input are multipled by each of the entries of the filter. Overlaps are summed. Also notice how for a fixed filtersize and stride, the dimensions of the input and output are swapped compared to standard convolution. (For example, if we did standard convolution on a length-7 sequence with filtersize of 3 and stride=2, we would output a length-3 sequence).

If our 2D input is $\begin{bmatrix} -1 & 2 \\ 3 & 1 \end{bmatrix}$ and the 2D filter is $\begin{bmatrix} +1 & -1 \\ 0 & +1 \end{bmatrix}$ **What is the output of transpose convolution with pad=0 and stride=1?**

**Solution:** Transpose convolution is designed to do an operation that can upsample a signal while simultaneously filtering it. That application is easiest to understand in the context of a stride that is greater than 1 — the original input signal is turned into a scaled sequence of "impulses" separated by the stride and then the filter acts on each of those impulses to smear them out and let them interact (linearly) with the other impulses to result in the final signal.

For this problem, the stride was set to 1 to give a nontrivial problem without making the arithmetic and size of the output too big.

The first conceptual part of this question is simply understanding what the output dimensions should be. If this were a standard convolution with pad=0, then we would only get a $1 \times 1$ sized output. But this is a "transpose convolution" and so pad=0 means that we don't want to drop any of the outputs that would naturally arise from the transpose convolution operation. In this case, this means that we will get a $3 \times 3$ output. We can "type-check" this by looking at what we would get with a regular transpose convolution starting with a $3 \times 3$ and applying a $2 \times 2$ filter with zero pad and stride 1 — we indeed would get a $2 \times 2$ back.

Given that the input is $\begin{bmatrix} -1 & 2 \\ 3 & 1 \end{bmatrix}$ we can look at the filter responses to each of the terms as

$$-1\begin{bmatrix} +1 & -1 & 0 \\ 0 & +1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + 2\begin{bmatrix} 0 & +1 & -1 \\ 0 & 0 & +1 \\ 0 & 0 & 0 \end{bmatrix} + 3\begin{bmatrix} 0 & 0 & 0 \\ +1 & -1 & 0 \\ 0 & +1 & 0 \end{bmatrix} + 1\begin{bmatrix} 0 & 0 & 0 \\ 0 & +1 & -1 \\ 0 & 0 & +1 \end{bmatrix} \tag{1}$$

$$= \begin{bmatrix} -1 & +1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & +2 & -2 \\ 0 & 0 & +2 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ +3 & -3 & 0 \\ 0 & +3 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & +1 & -1 \\ 0 & 0 & +1 \end{bmatrix} \tag{2}$$

$$= \begin{bmatrix} -1 & 3 & -2 \\ 3 & -3 & 1 \\ 0 & 3 & 1 \end{bmatrix}. \tag{3}$$

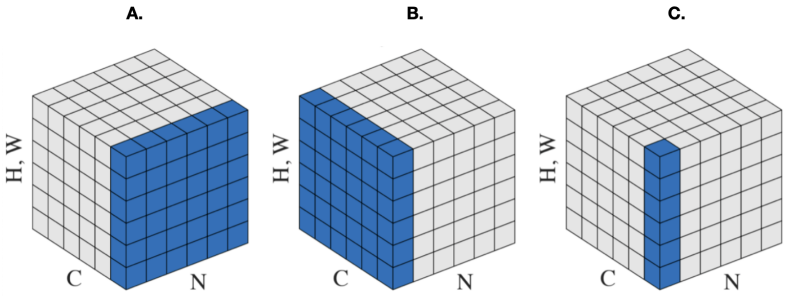
Notice above how the filter matrix has just been shifted each time by the relevant stride (1) times the position of the corresponding element in the input matrix. This is the "dragging" operation at the root

of all convolutions as applied for transpose convolution.

## 2. Batch Normalization for CNN

(a) 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) 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.**

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

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

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

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.

## 3. Depthwise Separable Convolutions

Depthwise separable convolutions are a type of convolutional operation used in deep learning for image processing tasks. Unlike traditional convolutional operations, which perform both spatial and channel-wise convolutions simultaneously, depthwise separable convolutions decompose the convolution operation into two separate operations: Depthwise convolution and Pointwise convolution.

This can be viewed as a low-rank approximation to a traditional convolution. For simplicity, throughout this problem, we will ignore biases while counting learnable parameters.

(a) Suppose the input is a three-channel $224 \times 224$-resolution image, the kernel size of the convolutional layer is $3 \times 3$, and the number of output channels is 4.

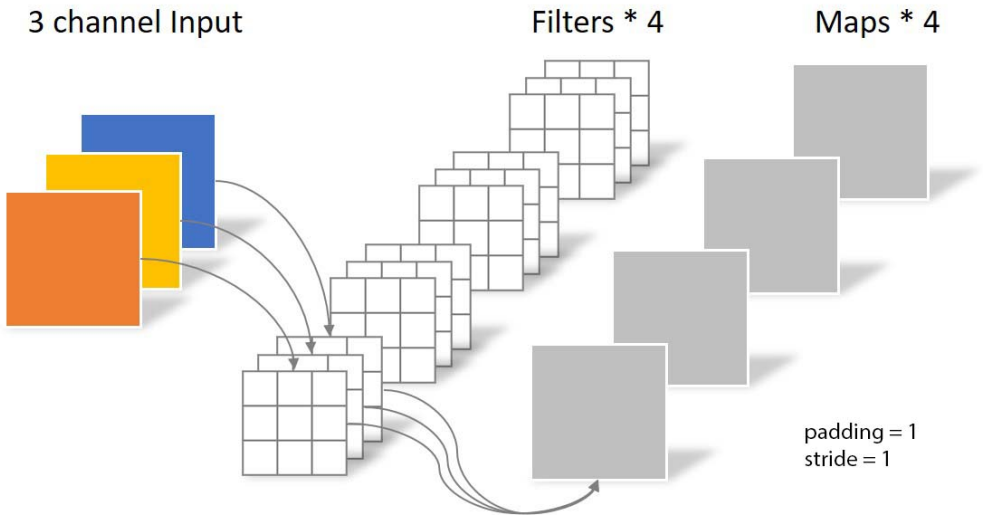



**Figure 1:** Traditional convolution.

**What is the number of learnable parameters in the traditional convolution layer?**

**Solution:** The number of weight parameters of a traditional convolution is the number of output channels $\times$ the number of input channels $\times$ kernel size $\times$ kernel size. Thus the number of parameters are $4 \times 3 \times 3 \times 3 = 108$.

There are also bias parameters, but we told you to ignore them in this problem. There would be one bias parameter per output channel in a traditional convolution — giving $108 + 4 = 112$ learnable parameters.

(b) Depthwise separable convolution consists of two parts: depthwise convolutions (Fig.2) followed by pointwise convolutions (Fig.3). Suppose the input is still a three-channel $224 \times 224$-resolution image. The input first goes through depthwise convolutions, where the number of output channels is the same as the number of input channels, and there is no "cross talk" between different channels. Then, this intermediate output goes through pointwise convolutions, which is basically a traditional convolution with the filter size being $1 \times 1$. Assume that we have 4 output channels.

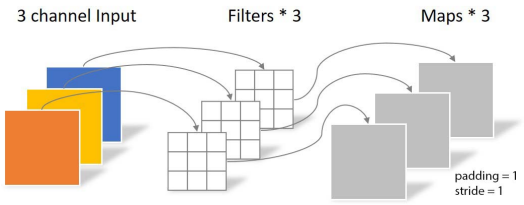



**Figure 2:** Depthwise convolution.

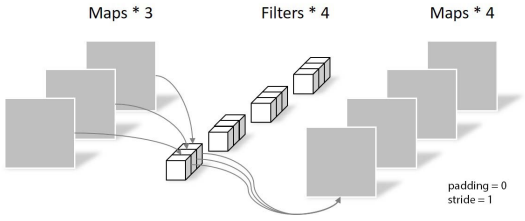



**Figure 3:** Pointwise convolution

**What is the total number of learnable parameters of the depthwise separable convolution layer which consists of both depthwise and pointwise convolutions?**

**Solution:** Since there is no "cross-talk" of different channels in the depthwise convolutions, the number of weight parameters in these is $3 \times 3 \times 3 = 27$.

For the pointwise convolutions that follow, the number of weight parameters is $4 \times 3 \times 1 \times 1 = 12$. The total number of learned weight parameters of the depthwise separable convolution is $27 + 12 = 39$.

Note that the depthwise separable convolutions require much fewer parameters.

If we had been interested in biases, we would have had one bias per output channel. Note: because everything is linear here, we do not need to have separate bias terms for the depth-wise convolutions as well as for the pointwise convolutions. It suffices to have biases on the pointwise convolutions. With the biases, the number of learned parameters is $39 + 4 = 43$. It would also be fine to have biases on both convolutions, which would give an answer of $39 + 3 + 4 = 46$.

# 4. Regularization and Dropout

Recall that linear regression optimizes the following learning objective:

$$\mathcal{L}(\mathbf{w}) = ||\mathbf{y} - X\mathbf{w}||_2^2 \tag{4}$$

One way of using *dropout* during SGD on the $d$-dimensional input features $\mathbf{x}_i$ involves *keeping* each feature at random $\sim_{i.i.d}$ $Bernoulli(p)$ (and zeroing it out if not kept) and then performing a traditional SGD step.

It turns out that such dropout makes our learning objective effectively become

$$\mathcal{L}(\check{\mathbf{w}}) = \mathbb{E}_{R \sim Bernoulli(p)} \left[ ||\mathbf{y} - (R \odot X)\check{\mathbf{w}}||_2^2 \right] \tag{5}$$

where $\odot$ is the element-wise product and the random binary matrix $R \in \{0,1\}^{n \times d}$ is such that $R_{i,j} \sim_{i.i.d} Bernoulli(p)$. We use $\check{\mathbf{w}}$ to remind you that this is learned by dropout.

Recalling how Tikhonov-regularized (generalized ridge-regression) least-squares problems involve solving:

$$\mathcal{L}(\mathbf{w}) = ||\mathbf{y} - X\mathbf{w}||_2^2 + ||\Gamma\mathbf{w}||_2^2 \tag{6}$$

for some suitable matrix $\Gamma$, it turns out we can manipulate (5) to eliminate the expectations and get:

$$\mathcal{L}(\check{\mathbf{w}}) = ||\mathbf{y} - pX\check{\mathbf{w}}||_2^2 + p(1-p)||\check{\Gamma}\check{\mathbf{w}}||_2^2 \tag{7}$$

with $\check{\Gamma}$ being a diagonal matrix whose $j$-th diagonal entry is the norm of the $j$-th column of the training matrix $X$.

(a) **Show that we can manipulate (5) to eliminate the expectations and get:**

$$\mathcal{L}(\check{\mathbf{w}}) = ||\mathbf{y} - pX\check{\mathbf{w}}||_2^2 + p(1-p)||\check{\Gamma}\check{\mathbf{w}}||_2^2 \tag{8}$$

**with $\check{\Gamma}$ being a diagonal matrix whose $j$-th diagonal entry is the norm of the $j$-th column of the training matrix $X$.**

**Solution:** Let $P = R \odot X$ where $\odot$ is the element-wise multiplication. Therefore, we have:

$$||y - Pw||_2^2 = y^T y - 2w^T P^T y + w^T P^T P w \tag{9}$$

That is:

$$\mathbb{E}_{R \sim Bernoulli(p)}[||y - R \odot Xw||_2^2] = \mathbb{E}_R[y^T y - 2w^T P^T y + w^T P^T P w] \tag{10}$$

Since the expected value of a matrix is the matrix of the expected value of its elements, we have that

$$\mathbb{E}_R[P]_{ij} = \mathbb{E}_R[(R \odot X)_{ij}] = X_{ij}\mathbb{E}_R[R_{ij}] = pX_{ij} \tag{11}$$

It follows that:

$$\mathbb{E}_R[2w^T P^T y] = 2pw^T X^T y \tag{12}$$

and:

$$(\mathbb{E}_R[(P^T P)])_{ij} = \Sigma_{k=1}^N \mathbb{E}_R[R_{ki} R_{kj} X_{ki} X_{kj}] \tag{13}$$

where:

$$\mathbb{E}_R[(P^T P)]_{ij} = \begin{cases} \sum_{k=1}^N \mathbb{E}_R[R_{ki}R_{kj}X_{ki}X_{kj}] = \sum_{k=1}^N \mathbb{E}_R[R_{ki}]\mathbb{E}_R[R_{kj}]X_{ki}X_{kj} = p^2(X^T X)_{ij} & \text{if } i \neq j \\ \sum_{k=1}^N \mathbb{E}_R[R_{ki}^2 X_{ki} X_{kj}] = \sum_{k=1}^N \mathbb{E}_R[R_{ki}^2]X_{ki}X_{kj} = p(X^T X)_{ij} & \text{if } i = j \end{cases} \tag{14}$$

Finally, we note that :

$$(\mathbb{E}_R[(P^TP)])_{ij} - p^2(X^TX)_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ (p-p^2)(X^TX)_{ij} & \text{if } i = j \end{cases} \tag{15}$$

we now can put everything together as follow:

$$\begin{aligned}
\mathcal{L}(w) &= \mathbb{E}_R[||y - R \odot Xw||_2^2] && (16)\\
&= \mathbb{E}_R[y^Ty - 2w^TP^Ty + w^TP^TPw] && (17)\\
&= y^Ty - 2pw^TX^Ty + p^2w^TX^TXw - p^2w^TX^TXw + w^T\mathbb{E}_R[P^TP]w && (18)\\
&= ||y - pXw||_2^2 + (w^T\mathbb{E}_R[P^TP]w - p^2w^TX^TXw) && (19)\\
&= ||y - pXw||_2^2 + w^T(\mathbb{E}_R[P^TP]w - p^2)w && (20)\\
&= ||y - pXw||_2^2 + (p^2 - p)w^T(\text{diag}(X^TX))w && (21)\\
&= ||y - pXw||_2^2 + p(1-p)w^T(\text{diag}(X^TX))w && (22)\\
&= ||y - pXw||_2^2 + p(1-p)||\check{\Gamma}w||_2^2 && (23)\\
& && (24)
\end{aligned}$$

where $\text{diag}(X^TX)$ refers to the matrix where the non-diagonal elements of $X^TX$ are set to 0, and $\check{\Gamma} = (\text{diag}(X^TX))^{1/2}$, which exists as $X^TX$ is PSD and therefore has non-negative diagonal elements.

(b) **How should we transform the $\check{\mathbf{w}}$ we learn using (8) (i.e. with dropout) to get something that looks a solution to the traditionally regularized problem (6)?**
*(Hint: This is related to how we adjust weights learned using dropout training for using them at inference time. PyTorch by default does this adjustment during training itself, but here, we are doing dropout slightly differently with no adjustments during training.)*

**Solution:** Choose $\mathbf{w} = p\check{\mathbf{w}}$.
The idea here is to absorb the $p$ term into $\mathbf{w}$, and then choose $\Gamma$ accordingly.
By choosing $\mathbf{w} = p\check{\mathbf{w}}$ (and thus $\check{\mathbf{w}} = \frac{1}{p}\mathbf{w}$), we would have

$$\begin{aligned}
\mathcal{L}(w) &= ||y - pX\check{\mathbf{w}}||_2^2 + p(1-p)||\check{\Gamma}\check{\mathbf{w}}||_2^2 && (25)\\
&= ||y - X\mathbf{w}||_2^2 + p(1-p)||\check{\Gamma}\frac{\mathbf{w}}{p}||_2^2 && (26)\\
&= ||y - X\mathbf{w}||_2^2 + ||\sqrt{\frac{(1-p)}{p}}\check{\Gamma}\mathbf{w}||_2^2 && (27)\\
&= ||y - X\mathbf{w}||_2^2 + ||\Gamma\mathbf{w}||_2^2 && (28)
\end{aligned}$$

where we choose $\Gamma = \sqrt{\frac{1-p}{p}}\check{\Gamma}$

(c) With the understanding that the $\Gamma$ in (6) is an invertible matrix, **change variables in (6) to make the problem look like classical ridge regression:**

$$\mathcal{L}(\widetilde{\mathbf{w}}) = ||\mathbf{y} - \widetilde{X}\widetilde{\mathbf{w}}||_2^2 + \lambda||\widetilde{\mathbf{w}}||_2^2 \tag{29}$$

**Explicitly, what is the changed data matrix $\widetilde{X}$ in terms of the original data matrix $X$ and $\Gamma$?**

**Solution:** To make the regularization term in (6) look like ridge regression, we'll let $\widetilde{\mathbf{w}} = \Gamma\mathbf{w}$, so $\mathbf{w} = \Gamma^{-1}\widetilde{\mathbf{w}}$. Plugging this into (6) gives us

$$||\mathbf{y} - X\Gamma^{-1}\tilde{\mathbf{w}}||_2^2 + ||\tilde{\mathbf{w}}||_2^2 \tag{30}$$

From this, we see our modified data matrix should be

$$\tilde{X} = X\Gamma^{-1}.$$

We also accept solutions which rescale the given answer by a positive constant.

(d) Continuing the previous part, with the further understanding that $\Gamma$ is a *diagonal* invertible matrix with the $j$-th diagonal entry proportional to the norm of the $j$-th column in $X$, **what can you say about the norms of the columns of the effective training matrix** $\widetilde{X}$ and **speculate briefly on the relationship between dropout and batch-normalization.**

**Solution:**

Let's say each $\mathbf{f}_j$, $\tilde{\mathbf{f}}_j$ are the j-th column vector of $X$ and $\tilde{X}$, respectively. Recall that $\Gamma$ is defined to be a diagonal matrix whose $j$-th diagonal entry is the norm of the $j$-th column of $X$ and $\tilde{X} = cX\Gamma^{-1}$, where $c$ is a rescaling positive constant that you found in the previous part.

$$\tilde{X} = cX\check{\Gamma}^{-1} = \begin{bmatrix}\tilde{\mathbf{f}}_1 & \tilde{\mathbf{f}}_2 & \dots & \tilde{\mathbf{f}}_d\end{bmatrix} = c\begin{bmatrix}\mathbf{f}_1 & \mathbf{f}_2 & \dots & \mathbf{f}_d\end{bmatrix}\begin{bmatrix}\frac{1}{\|\mathbf{f}_1\|_2} & 0 & \dots & 0 \\ 0 & \frac{1}{\|\mathbf{f}_2\|_2} & \dots & 0 \\ & & \ddots & \\ 0 & 0 & \dots & \frac{1}{\|\mathbf{f}_d\|_2}\end{bmatrix}$$

$$= \begin{bmatrix}c\frac{\mathbf{f}_1}{\|\mathbf{f}_1\|_2} & c\frac{\mathbf{f}_2}{\|\mathbf{f}_2\|_2} & \dots & c\frac{\mathbf{f}_d}{\|\mathbf{f}_d\|_2}\end{bmatrix}$$

Therefore, $\tilde{\mathbf{f}}_j = c\frac{\mathbf{f}_j}{\|\mathbf{f}_j\|_2}$ and $\left\|\tilde{\mathbf{f}}_j\right\|_2 = c$. In other words, dropout makes each column vector of the matrix $X$ constant-norm of $c$ in effect, where $c$ is the appropriate rescaling constant from part 2$c$.

Note that this is similar to batch-normalization's standardization and scaling operations, which makes the column vectors have the same variance.

# 5. Understanding Dropout (Coding Question)

In this question, you will analyze the effect of dropout in a simplified setting. Please follow the instructions in q_coding_dropout.ipynb and answer the questions in your submission of the written assignment. The notebook does not need to be submitted.

(a) (No dropout, least-square) **The mathematical expression of the OLS solution, and the solution calculated in the code cell.**

**Solution:**

$$(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{31}$$

```
[[1.08910891] [0.10891089]]
```

(b) (No dropout, gradient descent) **The solution in the code cell. Are the weights obtained by training with gradient descent the same as those calculated using the closed-form least squares method?**

**Solution:** `Weights: odict_values([tensor([[1.0891, 0.1089]])])`
Yes, it is the same as the last part.

(c) (Dropout, least-square) **The solution in the code cell.**

**Solution:**

```
x = [[20 2] [20 0] [ 0 2] [ 0 0]]
y = [[11] [11] [11] [11]]
w = [[0.36666667] [3.66666667]]
```

(d) (Dropout, gradient descent) **Describe the shape of the training curve. Are the weights obtained by training with gradient descent the same as those calculated using the closed-form least squares method?**

**Solution:**
The loss curve is very spiky. The solution is also different from the solution in the last part.

(e) (Dropout, gradient descent, large batch size) **Describe the loss curve and compare it with the loss curve in the last part. Why are they different? Also compare the trained weights with the one calculated by the least-square formula.**

**Solution:** The loss curve is less spiky. With batch-size 1, we alternate between really high and low loss for each data point depending on which elements are masked out. The worst loss is when both data points are dropped out, and this can't be reduced. The losses on the other data points are still nonzero since the network can't fit all equations simultaneously. Convergence is slow since each new datapoint tugs the weights in a different direction. With a larger batch size, we take the approximate average of the loss over the four datapoints, which means the gradient for each batch points in a similar direction, making it easier to converge (though loss at convergence is still nonzero, so the learned weight is close to the least-square solution, but they are still slightly different).

(f) Refer back to the cells you ran in part (e). **Analyze how and why adding dropout changes the following: (i) How large were the final weights $w_1$ and $w_2$ compared to each other. (ii) How large the contribution of each term (i.e. $10w_1$ + $w_2$) is to the final output. Why does this change occur?** (This does not need to be a formal math proof).

**Solution:** Without dropout, $w_1$ is 10x larger than $w_2$ and contributes 100x more to the output. This occurs because the update step for $w_1$ is always 10x larger than the update step for $w_2$. With dropout, $w_2$ is approximately 10x larger than $w_1$ and the contributions to the final output are approximately the same. The reason this is happening is as follows. We have 3 different potential cases with dropout: one where $w_1$ is dropped, one where $w_2$ is dropped and one where neither are dropped. When w1 is dropped, gradient descent is pulling $w_2$ towards 11. When $w_2$ is dropped, gradient descent is pulling $w_1$ towards 1.1. However, these are in conflict with the case where nothing is dropped out, since if $w_1 = 1$ and $w_2 = 10$, our output is 20 and our mean squared error is huge (81). This third case pulls $w_1$ and $w_2$ down by gradient descent, and specifically pulls $w_1$ down much more than $w_2$ since small changes in $w_1$ get multiplied by a factor of 10 and result in a much larger decrease in our error. These three conflicting objectives settle in the middle of the analytic solution and this dropped out loss when $w_1$ being smaller and $w_2$ being larger than the no-dropout solution due to case 3 pulling $w_1$ down 10 times harder.

(g) (Optional) Sweeping over the dropout rate **Fill out notebook section (G).** You should see that as the dropout rate changes, $w_1$ and $w_2$ change smoothly, except for a discontinuity when dropout rates are 0. Explain this discontinuity.

**Solution:** When dropout rates are positive, there is a unique loss-minimizing solution. Achieving it involves making $w_2$ larger than $w_1$. When dropout rates are zero, there are many loss-minimizing solutions, and the network chooses the norm-minimizing one, which makes $w_1$ larger than $w_2$.

(h) (Optional) Optimizing with Adam: Run the cells in part (H). **Does the solution change when you switch from SGD to Adam? Why or why not?**

**Solution:** The solution with dropout = .5 doesn't change since there's a unique loss-minimizing solution. The solution without dropout changes since there are many loss-minimizing solutions, and Adam's adaptive learning rate makes it converge to the solution where $w_1$ and $w_2$ are approximately equal instead of SGD's norm-minimizing solution.

(i) Dropout on real data: **Run the notebook cells in part (I), and report on how they affect the final performance.** **Solution:** Dropout increases the test accuracy on clean data, and the network learns to rely a bit less on the cheating feature (though is still relies heavily on it)

# 6. Batchnorm, Dropout and Convolutions (Coding Question)

In this assignment, you will implement batch normalization, dropout, and convolutions using NumPy and PyTorch. For this assignment, we recommend using Google Colab as some PCs or laptops may not support GPU-based PyTorch. You can sign up for a free trial of Colab Pro for students and teachers by following this link.

Attention: This coding task will take approximately 4 to 6 hours to complete, taking into account the time required for training the model. Please plan accordingly and start early to ensure adequate time to finish.

The assignment consists of three parts:

**Implementing Batch Norm and Dropout.** Open bn_drop.ipynb and follow the instructions in the notebook. You will implement the forward and backward propagation of dropout and batch normalization in NumPy. A GPU runtime is not required for this part.

**Implement convolution and spatial batch norm.** Open cnn.ipynb and follow the instructions in the notebook. You will implement the forward and backward propagation of convolutional layers and spatial dropout in NumPy. A GPU runtime is not required for this part.

**Use deep learning framework.** Open pytorch_cnn.ipynb and follow the instructions in the notebook. For this part, you will need to switch to a GPU runtime (details can be found in the notebook). You will implement a convolutional neural network with convolution layers, batch normalization, and dropout using PyTorch and train it on a GPU. You also have the opportunity to improve or design your own neural network.

Please answer the following question in your submission:

(a) **Draw the computational graph of training-time batch normalization** In input of the computational graph should be $\mathbf{X}, \gamma, \beta$, the output of the computational graph should be $\mathbf{Y}$, and the intermediate nodes are $\mu, \sigma^2, \mathbf{Z}$.

**Solution:** Here is the list of edges in the computational graph:

$$\mathbf{X} \to \mu \tag{32}$$

$$\mathbf{X} \to \sigma^2 \tag{33}$$

$$\mathbf{X} \to \mathbf{Z} \tag{34}$$
$$\mu \to \mathbf{Z} \tag{35}$$
$$\sigma^2 \to \mathbf{Z} \tag{36}$$
$$\mathbf{Z} \to \mathbf{Y} \tag{37}$$
$$\gamma \to \mathbf{Y} \tag{38}$$
$$\beta \to \mathbf{Y} \tag{39}$$

(b) **(Optional) Derive the closed-form back-propagation of a batch normalization layer (during training).** Include the answer in your written assignment.

Specifically, given $\mathrm{dy}_{i,j} = \dfrac{\partial \mathcal{L}}{\partial Y_{i,j}}$ for every $i, j$, Please derive $\dfrac{\partial \mathcal{L}}{\partial X_{i,j}}$ for every $i, j$ as a function of $\mathrm{dy}, \mathbf{X}, \mu, \sigma^2, \epsilon, \gamma, \beta$.

**Solution:**

$$\frac{\partial Y_{ij}}{\partial Z_{kl}} = \begin{cases} \gamma_j, & \text{if } i = k \text{ and } j = l \\ 0, & \text{otherwise} \end{cases} \tag{40}$$
$$\implies \frac{\partial \mathcal{L}}{\partial Z_{ij}} = \sum_{k,l} \frac{\partial \mathcal{L}}{\partial Y_{kl}} \frac{\partial Y_{kl}}{\partial Z_{ij}} \tag{41}$$
$$= \frac{\partial \mathcal{L}}{\partial Y_{ij}} \frac{\partial Y_{ij}}{\partial Z_{ij}} + 0 \tag{42}$$
$$= \mathrm{dy}_{ij} \gamma_j \tag{43}$$

$$\frac{\partial Z_{ij}}{\partial \mu_k} = \begin{cases} -\dfrac{1}{\sqrt{\sigma_j^2 + \epsilon}}, & \text{if } j = k \\ 0, & \text{otherwise} \end{cases} \tag{44}$$
$$\implies \frac{\partial \mathcal{L}}{\partial \mu_j} = \sum_i \frac{\partial \mathcal{L}}{\partial Z_{ij}} \frac{\partial Z_{ij}}{\partial \mu_j} \tag{45}$$
$$= \sum_i \mathrm{dy}_{ij} \gamma_j \left(-\frac{1}{\sqrt{\sigma_j^2 + \epsilon}}\right) \tag{46}$$
$$= -\frac{\gamma_j}{\sqrt{\sigma_j^2 + \epsilon}} \sum_i \mathrm{dy}_{ij} \tag{47}$$

$$\frac{\partial Z_{ij}}{\partial \sigma_k^2} = \begin{cases} -\dfrac{1}{2}(\sigma_j^2 + \epsilon)^{-3/2}, & \text{if } j = k \\ 0, & \text{otherwise} \end{cases} \tag{48}$$
$$\implies \frac{\partial \mathcal{L}}{\partial \sigma_j^2} = \sum_i \frac{\partial \mathcal{L}}{\partial Z_{ij}} \frac{\partial Z_{ij}}{\partial \sigma_j^2} \tag{49}$$

$$= \sum_i \mathrm{d}y_{ij}\gamma_j(-\frac{1}{2}(\sigma_j^2+\epsilon)^{-3/2}) \tag{50}$$

$$= -\frac{1}{2}\gamma_j(\sigma_j^2+\epsilon)^{-3/2}\sum_i \mathrm{d}y_{ij} \tag{51}$$

$$\frac{\partial \mu_k}{\partial X_{ij}} = \begin{cases} \frac{1}{n}, & \text{if } j=k \\ 0, & \text{otherwise} \end{cases} \tag{52}$$

$$\text{and } \frac{\partial \sigma_k^2}{\partial X_{ij}} = \begin{cases} \frac{2}{n}(X_{ij}-\mu_j), & \text{if } j=k \\ 0, & \text{otherwise} \end{cases} \tag{53}$$

$$\implies \frac{\partial \mathcal{L}}{\partial X_{ij}} = \frac{\partial \mathcal{L}}{\partial Z_{ij}}\frac{1}{\sqrt{\sigma_j^2+\epsilon}} + \sum_k \frac{\partial \mathcal{L}}{\partial \mu_k}\frac{\partial \mu_k}{\partial X_{ij}} + \sum_k \frac{\partial \mathcal{L}}{\partial \sigma_k^2}\frac{\partial \sigma_k^2}{\partial X_{ij}} \tag{54}$$

$$= \mathrm{d}y_{ij}\gamma_j\frac{1}{\sqrt{\sigma_j^2+\epsilon}} - \frac{\gamma_j}{\sqrt{\sigma_j^2+\epsilon}}\sum_k \mathrm{d}y_{kj}(\frac{1}{n}) \tag{55}$$

$$-\frac{1}{2}\gamma_j(\sigma_j^2+\epsilon)^{-3/2}\sum_k \mathrm{d}y_{kj}\frac{2}{n}(X_{kj}-\mu_j) \tag{56}$$

$$= \frac{\gamma_j}{\sqrt{\sigma_j^2+\epsilon}}\left(\mathrm{d}y_{ij} - \frac{1}{n}\sum_k \mathrm{d}y_{kj} - \frac{1}{n}\frac{1}{\sigma_j^2+\epsilon}\sum_k \mathrm{d}y_{kj}(X_{kj}-\mu_j)\right) \tag{57}$$

(c) **Explain what you see in this experiment. What does it suggest about dropout?**

**Solution:** Dropout reduces the gap between the validation accuracy and the training accuracy, and increases the validation accuracy. The fact shows that dropout can mitigate overfitting and improve the generalization of the neural network.

(d) **Briefly describe your neural network design and the procedure of hyperparameter tuning.**

## 7. Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student!
We also want to understand what resources you find helpful and how much time homework is taking, so we can change things in the future if possible.

(a) **What sources (if any) did you use as you worked through the homework?**

(b) **If you worked with someone on this homework, who did you work with?**
List names and student ID's. (In case of homework party, you can also just describe the group.)

(c) **Roughly how many total hours did you work on this homework?**

**The following are old exam questions simply for your reference. You do not have to do these questions. Their coverage is redundant.**

# 8. Multiplicative Regularization beyond Dropout

In dropout, we get a regularizing effect by multipling the activations of the previous layer by iid coin tosses to randomly zero out many of them during each SGD update. Here, we will consider a linear-regression problem but instead of randomly multiplying each input feature with a 0 or a 1 during SGD updates, we will multiply each feature of our input with an iid random sample of a normal distribution with mean $\mu$ and variance $\sigma^2$. In other words, we perform the elementwise product $R \odot X$, where $R$ is a matrix where every iid entry $R_{ij} \sim \mathcal{N}(\mu, \sigma^2)$ and $\odot$ represents elementwise multiplication.

*It turns out that the expected training loss*

$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_{R_{ij} \sim \mathcal{N}(\mu,\sigma^2)}\left[||\mathbf{y} - (R \odot X)\mathbf{w}||_2^2\right]$$

*can be put in the form*

$$\mathcal{L}(\mathbf{w}) = ||\mathbf{y} - \_\_\textbf{(A)}\_\_X\mathbf{w}||_2^2 + \_\_\textbf{(B)}\_\_||\Gamma\mathbf{w}||_2^2$$

where $\Gamma = (\text{diag}(X^\top X))^{1/2}$.

**What are (A) and (B)?**

Select one choice for **(A)**:

- ◯ $\mu$
- ◯ $2\mu$
- ◯ $\frac{\mu}{2}$
- ◯ $\sigma$
- ◯ $2\sigma$
- ◯ $\frac{\sigma}{2}$

**Solution:** The right answer is $\mu$.

Select one choice for **(B)**:

- ◯ $\mu^2$
- ◯ $2\mu^2$
- ◯ $\frac{\mu^2}{2}$
- ◯ $\sigma^2$
- ◯ $2\sigma^2$
- ◯ $\frac{\sigma^2}{2}$

**Solution:** The right answer is $\sigma^2$.

**Show some work below** to justify your choices. Correct answers with incorrect or no supporting work will not receive full credit.

*(Hint: You might find it helpful to think about the case $\mu = 1$ and $\sigma^2 = 0$ to help you pick as well as $\mu = 0$ and $\sigma^2 = \infty$.)*

**Solution:**

Expanding out the objective and applying linearity of expectation, we have

$$\mathbb{E}\left[||\mathbf{y} - (R \odot X)\mathbf{w}||_2^2\right] = \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbb{E}[(R \odot X)]\mathbf{w} + \mathbf{w}^T\mathbb{E}[(R \odot X)^T(R \odot X)]\mathbf{w}$$

Since the expected value of a matrix is the matrix of the expected value of its elements, the linear term becomes $\mathbb{E}[(R \odot X)_{ij}] = X_{ij}\mathbb{E}[R_{ij}] = \mu X_{ij}$, so $\mathbb{E}[2\mathbf{y}^T(R \odot X)^T\mathbf{w}] = 2\mu\mathbf{y}^T X\mathbf{w}$.

For the quadratic term, we have

$$\mathbb{E}[((R \odot X)^T(R \odot X))])_{ij} = \Sigma_{k=1}^N \mathbb{E}[R_{ik}^T X_{ik}^T R_{kj} X_{kj}] = \Sigma_{k=1}^N \mathbb{E}[R_{ki}R_{kj}]X_{ki}X_{kj}$$

Again, we need to consider both the cases when $i = j$ and $i \neq j$. Recall that variance is defined as $\text{Var}[R_{ij}] = \mathbb{E}[R_{ij}^2] - \mathbb{E}[R_{ij}]^2$, so we can write the second moment as $\mathbb{E}[R_{ij}^2] = \mu^2 + \sigma^2$.

So the matrix for our quadratic term becomes:

$$= \begin{cases} \Sigma_{k=1}^{N} \mathbb{E}[R_{ki}R_{kj}X_{ki}X_{kj}] = \Sigma_{k=1}^{N} \mathbb{E}[R_{ki}]\mathbb{E}[R_{kj}]X_{ki}X_{kj} = \mu^2(X^TX)_{ij} & \text{if } i \neq j \\ \Sigma_{k=1}^{N} \mathbb{E}[R_{ki}^2]X_{ki}X_{kj} = (\mu^2 + \sigma^2)(X^TX)_{ij} & \text{if } i = j \end{cases}$$

Now, back to our original objective, we can write it as

$$\begin{aligned} \mathcal{L}(w) &= y^Ty - 2\mu \mathbf{y}^T X\mathbf{w} + \mu^2 \mathbf{w}^T X^T X\mathbf{w} + \sigma^2 \mathbf{w}^T \text{diag}(X^TX)\mathbf{w} \\ &= ||\mathbf{y} - \mu X\mathbf{w}||_2^2 + \sigma^2 ||\Gamma\mathbf{w}||_2^2 \end{aligned}$$

which is in the form that we desire.

# 9. Batch Normalization

Recall the pseudocode for a batchnorm layer (with learnable scale and shift) in a neural network:

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: $\gamma$, $\beta$
**Output:** $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// standardize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

(a) If our input data (1-dimensional) to batchnorm follows roughly the distribution on the left:

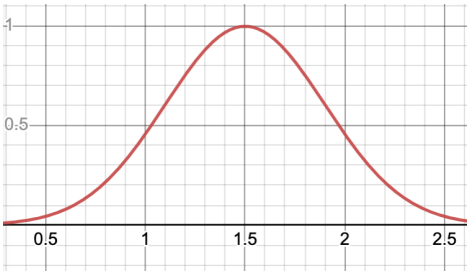



**Figure 4:** Gaussian with mean $\mu = 1.5$, variance $\sigma^2 = 0.16$

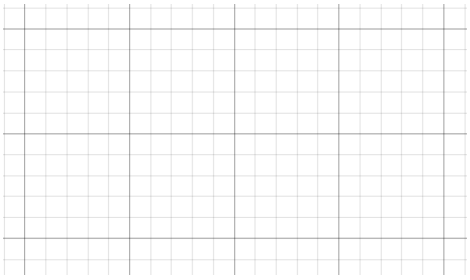


**Figure 5:** Blank grid for your answer

**What does our data distribution look like after batch normalization with $\beta = 3$ and $\gamma = 1$ parameters? Draw your answer on the blank grid above, give a scale to the horizontal axis, and label $\beta$.** You can assume that the batch-size is very large. **Briefly explain why the graph looks the way it does.**

*(Note: You do not have to give a scale to the vertical axis.)*

**Solution:** Shifted and scaled gaussian is still a gaussian. Note that mean is at $\beta = 3$, and stdev is $\gamma = 1$.

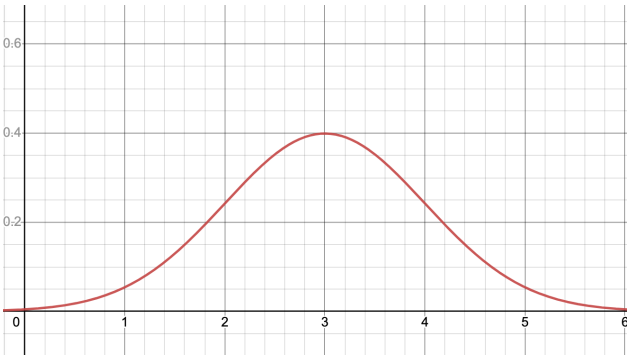



**Figure 6:** Solution

(b) Say our input data (now 2-dimensional) to the batchnorm layer follows a Gaussian distribution. The mean and contours (level sets) of points that are 1 and 2 stdev away from the mean are shown below. **On the same graph, draw what the mean, 1-SD, and 2-SD contours would look like after batchnorm without any shifting/scaling (i.e. $\beta = 0$ and $\gamma = 1$).** You can assume that the batch-size is very large. **Briefly explain why the graph looks the way it does.**

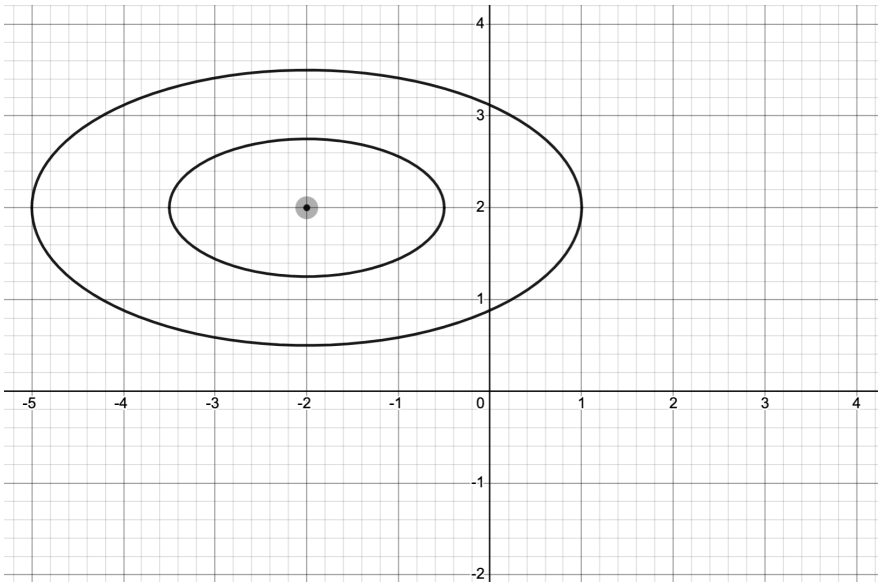



**Figure 7:** Draw your answer on the grid

**Solution:**

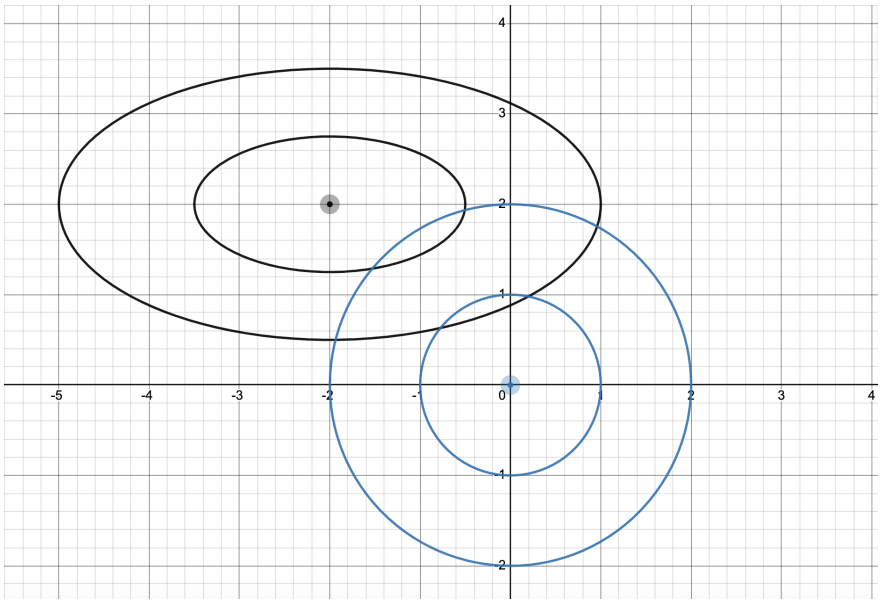



**Figure 8:** Solution

**Contributors:**

- Saagar Sanghavi.
- Kevin Li.
- Peter Wang.
- Jerome Quenum.
- Anant Sahai.
- Olivia Watkins.
- Jake Austin.
- Linyuan Gong.
- Sheng Shen.
- Suhong Moon.