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)

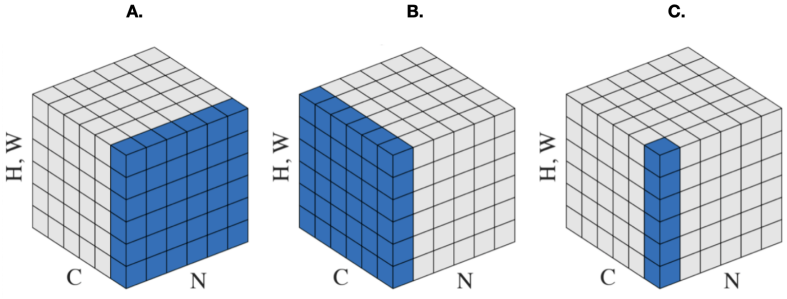
(a) **List two reasons we typically prefer convolutional layers instead of fully connected layers when working with image data.**

(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.)

(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?**

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

(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 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.)

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 224×224-resolution image, the kernel size of the convolutional layer is 3×3, and the number of output channels is 4.

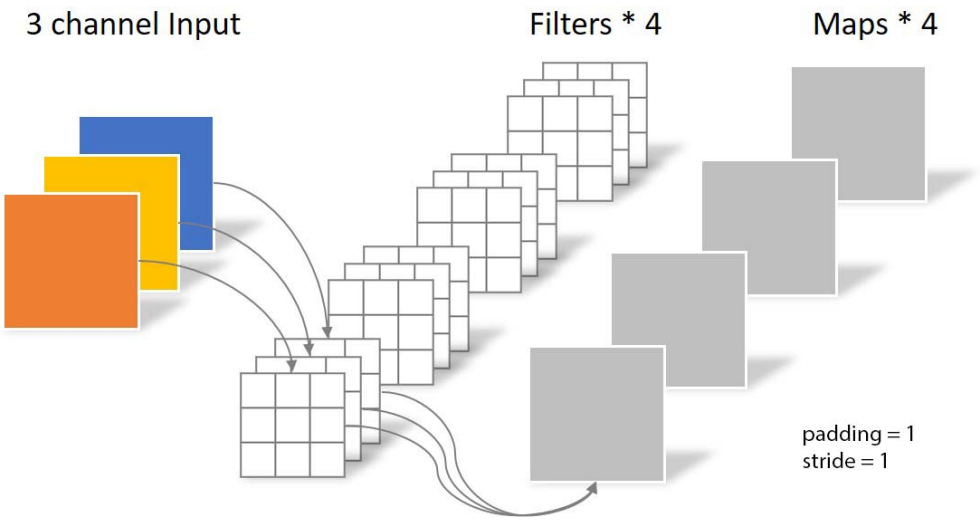


Figure 1: Traditional convolution.

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

(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 224×224-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 1×1. Assume that we have 4 output channels.

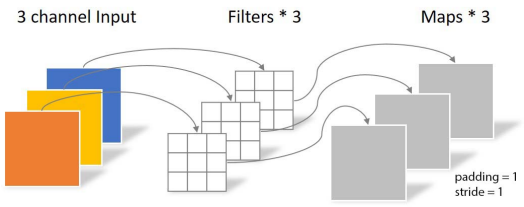


Figure 2: Depthwise convolution.

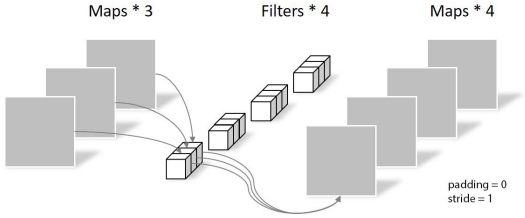


Figure 3: Pointwise convolution

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

4. Regularization and Dropout

Recall that linear regression optimizes the following learning objective:

$$\mathcal{L}(\mathbf{w}) = ||\mathbf{y} - X\mathbf{w}||_2^2 \tag{1}$$

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

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

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:

$$\mathcal{L}(\mathbf{w}) = ||\mathbf{y} - X\mathbf{w}||_2^2 + ||\Gamma\mathbf{w}||_2^2 \tag{3}$$

for some suitable matrix Γ, it turns out we can manipulate (2) to eliminate the expectations and get:

$$\mathcal{L}(\check{\mathbf{w}}) = ||\mathbf{y} - pX\check{\mathbf{w}}||_2^2 + p(1-p)||\check{\Gamma}\check{\mathbf{w}}||_2^2 \tag{4}$$

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 (2) to eliminate the expectations and get:**

$$\mathcal{L}(\check{\mathbf{w}}) = ||\mathbf{y} - pX\check{\mathbf{w}}||_2^2 + p(1-p)||\check{\Gamma}\check{\mathbf{w}}||_2^2 \tag{5}$$

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.

(b) **How should we transform the $\check{\mathbf{w}}$ we learn using (5) (i.e. with dropout) to get something that looks a solution to the traditionally regularized problem (3)?**

(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.)

(c) With the understanding that the Γ in (3) is an invertible matrix, **change variables in (3) to make the problem look like classical ridge regression:**

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

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

(d) Continuing the previous part, with the further understanding that Γ 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.**

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.**

(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?**

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

(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?**

(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.**

(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).

(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.

(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?**

(i) Dropout on real data: **Run the notebook cells in part (I), and report on how they affect the final performance.**

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 $\mu, \sigma^2, \mathbf{Z}$.

(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 $\mathrm{dy}, \mathbf{X}, \mu, \sigma^2, \epsilon, \gamma, \beta$.

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

(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 μ and variance σ^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)**:

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

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

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

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 $\sigma^2 = \infty$.)

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: γ, β
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:

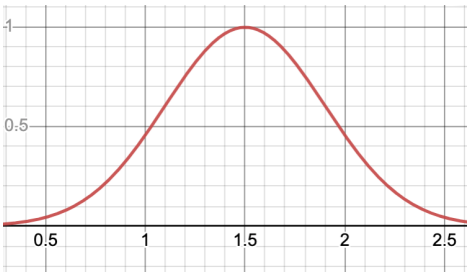


Figure 4: Gaussian with mean $\mu = 1.5$, variance $\sigma^2 = 0.16$

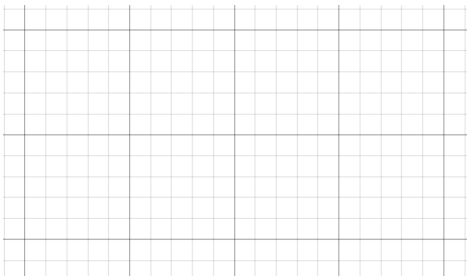


Figure 5: Blank grid for your answer

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 β. 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.)

(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 batch-norm 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.**

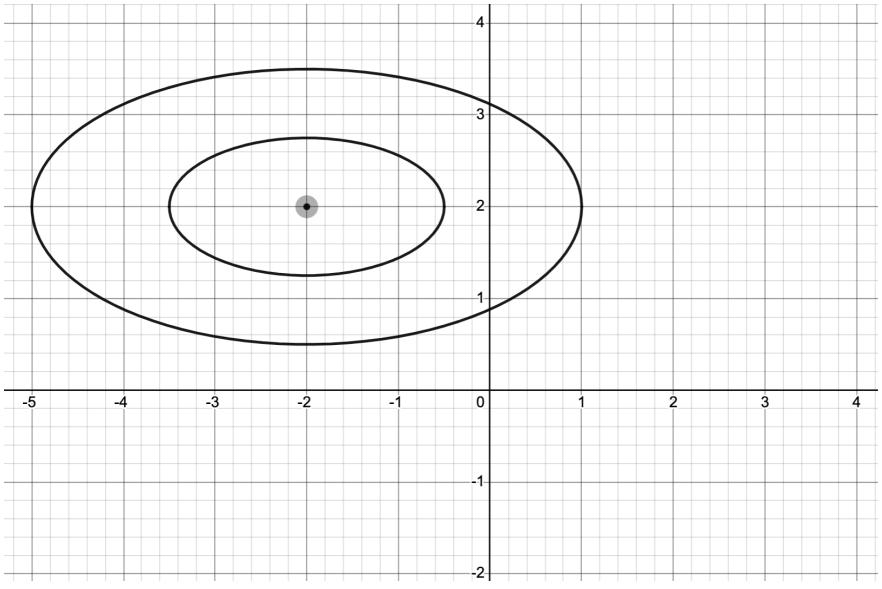


Figure 6: Draw your answer on the grid

Contributors:

- Saagar Sanghavi.
- Kevin Li.
- Peter Wang.

- Jerome Quenum.
- Anant Sahai.
- Olivia Watkins.
- Jake Austin.
- Linyuan Gong.
- Sheng Shen.
- Suhong Moon.

