EECS 182 Deep Neural Networks
Fall 2023 Anant Sahai

Homework 5

1. 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.

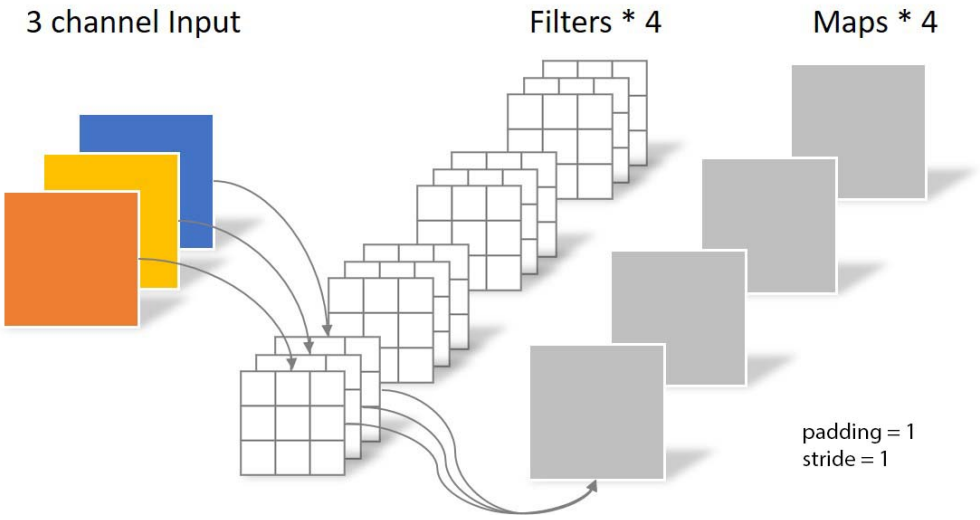

Figure 1: Traditional convolution.

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.

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

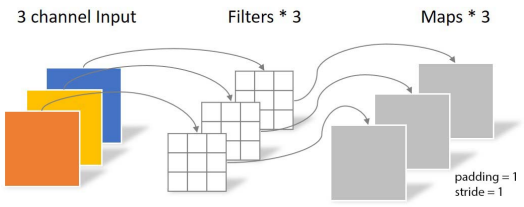

Figure 2: Depthwise convolution.

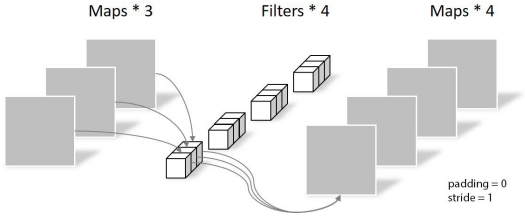

Figure 3: Pointwise convolution

2. 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.

(a) **Show that we can manipulate (2) to eliminate the expectations and get:**

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

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.

(b) **How should we transform the $\check{\mathbf{w}}$ we learn using (4) (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:**

$$\mathcal{L}(\widetilde{\mathbf{w}}) = ||\mathbf{y} - \widetilde{X}\widetilde{\mathbf{w}}||_2^2 + \lambda||\widetilde{\mathbf{w}}||_2^2 \tag{5}$$

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.**

3. 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.

4. Analyzing Distributed Training

For real-world models trained on lots of data, the training of neural networks is parallelized and accelerated by running workers on distributed resources, such as clusters of GPUs. In this question, we will explore three popular distributed training paradigms:

All-to-All Communication: Each worker maintains a copy of the model parameters (weights) and processes a subset of the training data. After each iteration, each worker communicates with every other worker and updates its local weights by averaging the gradients from all workers.

Parameter Server: A dedicated server, called the parameter server, stores the global model parameters. The workers compute gradients for a subset of the training data and send these gradients to the parameter server. The server then updates the global model parameters and sends the updated weights back to the workers.

Ring All-Reduce: Arranges n workers in a logical ring and updates the model parameters by passing messages in a circular fashion. Each worker computes gradients for a subset of the training data, splits the gradients into n equally sized chunks and sends a chunk of the gradients to their neighbors in the ring. Each worker receives the gradient chunks from its neighbors, updates its local parameters, and passes the

updated gradient chunks along the ring. After $n-1$ passes, all gradient chunks have been aggregated across workers, and the aggregated chunks are passed along to all workers in the next $n-1$ steps. This is illustrated in Figure 4.

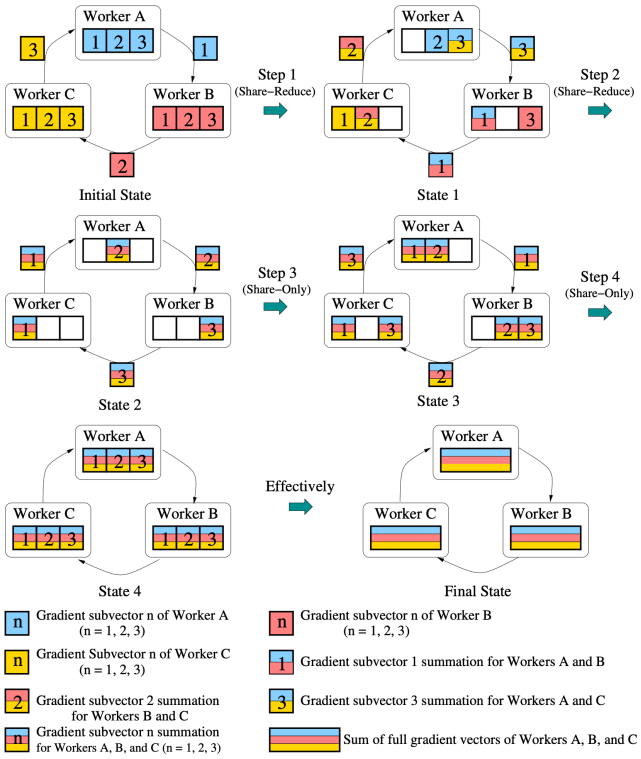

Figure 4: Example of Ring All-Reduce in a 3 worker setup. Source: Mu Et. al, *GADGET: Online Resource Optimization for Scheduling Ring-All-Reduce Learning Jobs*

For each of the distributed training paradigms, fill in the total number of messages sent and the size of each message. Assume that there are n workers and the model has p parameters, with p divisible by n.

	Number of Messages Sent	**Size of each message**
All-to-All		p
Parameter Server	$2n$	
Ring All-Reduce	$n(2(n-1))$	

5. Coding Question: Batchnorm, Dropout and Convolutions

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.

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 for completion.

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.**

6. Understanding Dropout (Coding Question)

In this question, you will analyze the effect of dropout in a simplified setting. Please follow the instructions in `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.**

7. Directed and Undirected Graphs

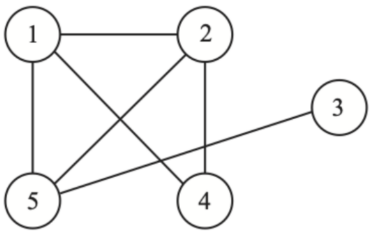

Figure 5: Simple Undirected Graph

Figure 5 shows a simple undirected graph whose adjacency matrices we want to make sure you can write down. Generally, an unnormalized adjacency matrix between the nodes of a directed or undirected graph is given by:

$$A_{i,j} = \begin{cases} 1 : \text{if there is an edge between node i and node j,} \\ 0 : \text{otherwise.} \end{cases} \tag{6}$$

This will be a symmetric matrix for undirected graphs. For a directed graph, we have:

$$A_{i,j} = \begin{cases} 1 : \text{if there is an edge from node i to node j,} \\ 0 : \text{otherwise.} \end{cases} \tag{7}$$

This need not to be symmetric for a directed graph, and is in fact typically not a symmetric matrix when we are thinking about directed graphs (otherwise, we'd probably be thinking of them as undirected graphs).

Similarly, the degree matrix of an undirected graph is a diagonal matrix that contains information about the degree of each vertex. In other words, it contains the number of edges attached to each vertex and it is given by:

$$D_{i,j} = \begin{cases} deg(v_i) : \text{if i == j,} \\ 0 : \text{otherwise.} \end{cases} \tag{8}$$

where the degree $\deg(v_i)$ of a vertex counts the number of times an edge terminates at that vertex.

For directed graphs, the degree matrix could be ***In-Degree*** when we count the number of edges coming into a particular node and ***Out-Degree*** when we count the number of edges going out of the node. We'll use the terms in-degree matrix or out-degree matrix to make it clear which one we are invoking.

Sometimes, imbalanced weights may undesirably affect the matrix spectrum (eigenvalues and eigenvectors). This occurs when a vertex with a large degree results in a large diagonal entry in the Laplacian matrix dominating the matrix properties. To solve that issue, a normalization scheme is applied which aims to make the influence of such vertices more equal to that of other vertices, by dividing the entries of the Adjacency matrix by the vertex degrees.

In that sense, a normalized adjacency matrix is given by:

$$A^{Normalized} = AD^{-1} \tag{9}$$

and a symmetrically normalized adjacency matrix is given by

$$A^{SymNorm} = D^{-1/2}AD^{-1/2} \tag{10}$$

Additionally, the Laplacian matrix relates many useful properties of a graph. In fact, the spectral decomposition of the Laplacian matrix of a graph allows for the construction of low-dimensional embeddings that appear in many machine learning applications. In other words, there is a relation between the properties of a graph and the spectra (eigenvalues and eigenvectors) of matrices associated with the graph, such as its adjacency matrix or Laplacian matrix.

Given a simple graph G with n vertices $v_1, ..., v_n$, its unnormalized Laplacian matrix $L_{n\times n}$ is defined element-wise as:

$$L_{i,j} = \begin{cases} deg(v_i) : \text{if i == j}, \\ -1 : \text{if i != j and } v_i \text{ is adjacent to } v_j, \\ 0 : \text{otherwise.} \end{cases} \tag{11}$$

or equivalently by the matrix:

$$L = D - A \tag{12}$$

where D is the degree matrix and A is the adjacency matrix of the graph.

We could also compute the symmetrically normalized Laplacian which is inherited from the adjacency matrix normalization scheme as shown below:

$$L^{SymNorm} = I - A^{SymNorm} \tag{13}$$

where I is the identity matrix, A is the unnormalized adjacency matrix, and L is the unnormalized Laplacian.

(a) **Show that $L^{SymNorm}$ could also be written as**:

$$L^{SymNorm} = D^{-1/2} L D^{-1/2} \tag{14}$$

where D is the dregree matrix, and L is the unnormalized Laplacian.

(b) **Write the unnormalized adjacency A, the degree matrix, D, and the symmetrically normalized adjacency matrix, $A^{SymNorm}$, of the graph in Figure. 5.**

(c) **Write the symmetrically normalized Laplacian matrix of the graph in Figure. 5.**

(d) **Compute A^2, A^3**

We now want to estimate the traffic flow of inner downtown Berkeley and we know the road network shown below. The goal of the estimation is to estimate the traffic flow on eache road segment. The flow estimates should satisfy the conservation of vehicles exactly at each intersection as indicated by the arrows.

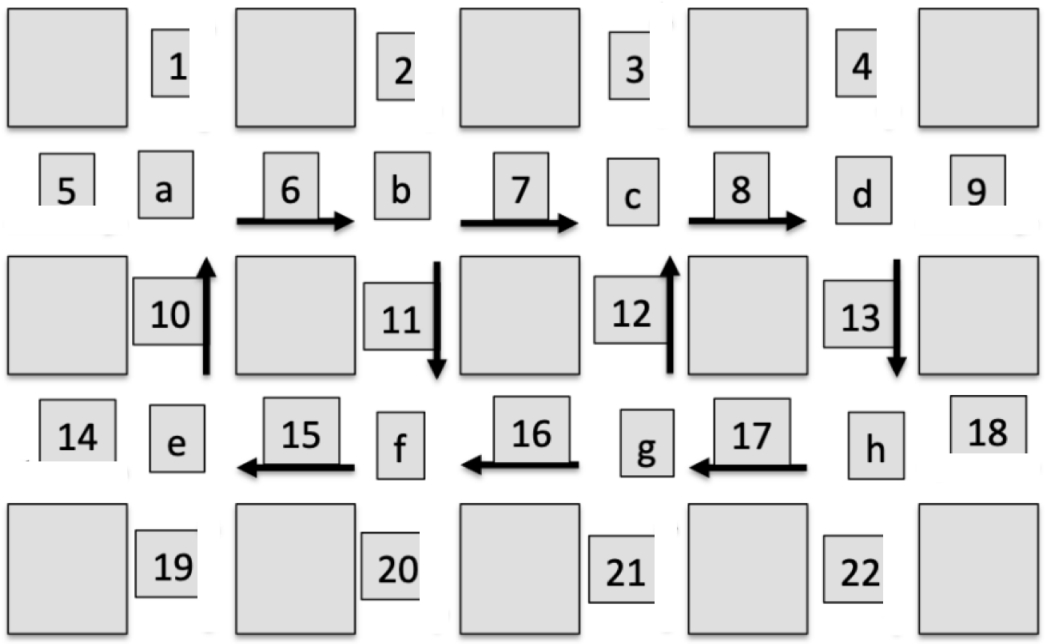

Figure 6: Simple Directed Graph

The intersections are labeled a to h. The road segments are labeled 1 to 22. The arrows indicate the direction of traffic.

Hint: think about the best way to represent the road network in terms of matrices, vectors, etc.

(e) **Write the unnormalized adjacency matrix of the graph in Figure 6.**

(f) **Write the In-degree D_{in} and Out-degree D_{out} matrix of the graph in Figure. 6.**

(g) **Write both of the symmetrically normalized In-degree and Out-degree Laplacian matrix of the graph in Figure. 6.**

(h) ***[Optional]*** It is good to read https://arxiv.org/pdf/1609.02907.pdf and https://distill.pub/2021/understanding-gnns/ to learn about the importance of the Adjacency and Laplacian matrices in graph representation.

8. 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?**

Contributors:

- Peter Wang.
- Saagar Sanghavi.
- Jerome Quenum.
- Anant Sahai.
- Romil Bhardwaj.
- Sheng Shen.
- Suhong Moon.
- Jake Austin.
- Kevin Li.
- Linyuan Gong.
- Olivia Watkins.
- Anrui Gu.
- Matthew Lacayo.
- Past EECS 282 and 227 Staff.

