EECS 182 Deep Neural Networks
Spring 2023 Anant Sahai

Homework 4

This homework is due on Friday, February 24, 2022, at 10:59PM.

1. Understanding Dropout (Coding Question)

In this question, you will analyze the effect of dropout in a simplified setting. Please follow the instructions in the Jupyter notebook 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:

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

```
[[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)

2. Regularization and Dropout

You saw one perspective on the implicit regularization of dropout in HW, and here, you will see another one. Recall that linear regression optimizes the following learning objective:

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

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

$$\mathcal{L}(\tilde{\mathbf{w}}) = \mathbb{E}_{R\sim Bernoulli(p)}\left[||\mathbf{y} - (R \odot X)\tilde{\mathbf{w}}||_2^2\right] \tag{3}$$

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:

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

for some suitable matrix Γ.

(a) **Show that we can manipulate (3) 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.

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

$$||y - Pw||_2^2 = y^T y - 2w^T P^T y + w^T P^T P w \tag{6}$$

That is:

$$\mathbb{E}_{R \sim Bernoulli(p)}[||y - R \odot Xw||_2^2] = \mathbb{E}_R[y^T y - 2w^T P^T y + w^T P^T P w] \tag{7}$$

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

$$\mathbb{E}_R[P]_{ij} = \mathbb{E}_R[(R \odot X)_{ij}] = X_{ij}\mathbb{E}_R[R_{ij}] = pX_{ij} \tag{8}$$

It follows that:

$$\mathbb{E}_R[2w^T P^T y] = 2pw^T X^T y \tag{9}$$

and:

$$(\mathbb{E}_R[(P^T P)])_{ij} = \Sigma_{k=1}^N \mathbb{E}_R[R_{ki} R_{kj} X_{ki} X_{kj}] \tag{10}$$

where:

$$\mathbb{E}_R[(P^T P)]_{ij} = \begin{cases} \sum_{k=1}^N \mathbb{E}_R[R_{ki}R_{kj}X_{ki}X_{kj}] = \sum_{k=1}^N \mathbb{E}_R[R_{ki}]\mathbb{E}_R[R_{kj}]X_{ki}X_{kj} = p^2(X^T X)_{ij} & \text{if } i \neq j \\ \sum_{k=1}^N \mathbb{E}_R[R_{ki}^2 X_{ki} X_{kj}] = \sum_{k=1}^N \mathbb{E}_R[R_{ki}^2]X_{ki}X_{kj} = p(X^T X)_{ij} & \text{if } i = j \end{cases} \tag{11}$$

Finally, we note that :

$$(\mathbb{E}_R[(P^T P)])_{ij} - p^2(X^T X)_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ (p - p^2)(X^T X)_{ij} & \text{if } i = j \end{cases} \tag{12}$$

we now can put everything together as follow:

$$\begin{aligned}
\mathcal{L}(w) &= \mathbb{E}_R[||y - R \odot Xw||_2^2] && (13)\\
&= \mathbb{E}_R[y^T y - 2w^T P^T y + w^T P^T P w] && (14)\\
&= y^T y - 2pw^T X^T y + p^2 w^T X^T X w - p^2 w^T X^T X w + w^T \mathbb{E}_R[P^T P] w && (15)\\
&= ||y - pXw||_2^2 + (w^T \mathbb{E}_R[P^T P] w - p^2 w^T X^T X w) && (16)\\
&= ||y - pXw||_2^2 + w^T (\mathbb{E}_R[P^T P] w - p^2) w && (17)\\
&= ||y - pXw||_2^2 + (p^2 - p) w^T (\text{diag}(X^T X)) w && (18)\\
&= ||y - pXw||_2^2 + p(1-p) w^T (\text{diag}(X^T X)) w && (19)\\
&= ||y - pXw||_2^2 + p(1-p)||\check{\Gamma} w||_2^2 && (20)\\
& && (21)
\end{aligned}$$

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

(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 (4)?**

(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 Γ accordingly.

By choosing $\mathbf{w} = p\check{\mathbf{w}}$ (and thus $\check{\mathbf{w}} = \frac{1}{p}\mathbf{w}$), we would have

$$\begin{aligned}
\mathcal{L}(w) &= ||y - pX\check{\mathbf{w}}||_2^2 + p(1-p)||\check{\Gamma}\check{\mathbf{w}}||_2^2 && (22)\\
&= ||y - X\mathbf{w}||_2^2 + p(1-p)||\check{\Gamma}\frac{\mathbf{w}}{p}||_2^2 && (23)\\
&= ||y - X\mathbf{w}||_2^2 + ||\sqrt{\frac{(1-p)}{p}}\check{\Gamma}\mathbf{w}||_2^2 && (24)\\
&= ||y - X\mathbf{w}||_2^2 + ||\Gamma\mathbf{w}||_2^2 && (25)
\end{aligned}$$

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

(c) With the understanding that the Γ in (4) is an invertible matrix, **change variables in (4) to make the problem look like classical ridge regression:**

$$\mathcal{L}(\widetilde{\mathbf{w}}) = ||\mathbf{y} - \widetilde{X}\widetilde{\mathbf{w}}||_2^2 + \lambda||\widetilde{\mathbf{w}}||_2^2 \quad (26)$$

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

Solution: To make the regularization term in (4) look like ridge regression, we'll let $\widetilde{\mathbf{w}} = \Gamma\mathbf{w}$, so $\mathbf{w} = \Gamma^{-1}\widetilde{\mathbf{w}}$. Plugging this into (4) gives us

$$||\mathbf{y} - X\Gamma^{-1}\tilde{\mathbf{w}}||_2^2 + ||\tilde{\mathbf{w}}||_2^2 \quad (27)$$

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 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.**

Solution:

Let's say each x_j, $\tilde{x}_j$ are the j-th column vector of X and $\tilde{X}$, respectively. Recall that Γ 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 of that you found in the previous part.

$$\tilde{X} = cX\check{\Gamma}^{-1} = \begin{bmatrix} \tilde{x}_1 & \tilde{x}_2 & \dots & \tilde{x}_d \end{bmatrix} = c \begin{bmatrix} x_1 & x_2 & \dots & x_d \end{bmatrix} \begin{bmatrix} \frac{1}{\|x_1\|_2} & 0 & \dots & 0 \\ 0 & \frac{1}{\|x_2\|_2} & \dots & 0 \\ & & \ddots & \\ 0 & 0 & \dots & \frac{1}{\|x_d\|_2} \end{bmatrix}$$

$$= \begin{bmatrix} c\frac{x_1}{\|x_1\|_2} & c\frac{x_2}{\|x_2\|_2} & \dots & c\frac{x_d}{\|x_d\|_2} \end{bmatrix}$$

Therefore, $\tilde{x}_j = c\frac{x_j}{\|x_j\|_2}$ and $\left\|\tilde{x}_j\right\|_2 = c$. 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 2c.

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

3. Weights and Gradients in a CNN

In this homework assignment, we aim to accomplish two objectives. Firstly, we seek to comprehend that the weights of a CNN are a weighted average of the images in the dataset. This understanding is crucial in answering a commonly asked question: does a CNN memorize images during the training process? Additionally, we will analyze the impact of spatial weight sharing in convolution layers. Secondly, we aim to gain an understanding of the behavior of max-pooling and avg-pooling in backpropagation. By accomplishing these objectives, we will enhance our knowledge of CNNs and their functioning.

Let's consider a convolution layer with input matrix $\mathbf{X} \in \mathbb{R}^{n\times n}$,

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,n} \end{bmatrix}, \tag{28}$$

weight matrix $\mathbf{w} \in \mathbb{R}^{k\times k}$,

$$\mathbf{w} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,k} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,1} & w_{k,2} & \cdots & w_{k,k} \end{bmatrix}, \tag{29}$$

and output matrix $\mathbf{Y} \in \mathbb{R}^{m \times m}$,

$$\mathbf{Y} = \begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,m} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m,1} & y_{m,2} & \cdots & y_{m,m} \end{bmatrix}. \tag{30}$$

For simplicity, we assume the number of the input channel (of $\mathbf{X}$ is) and the number of the output channel (of output $\mathbf{Y}$) are both 1, and the convolutional layer has no padding and a stride of 1.

Then for all i, j,

$$y_{i,j} = \sum_{h=1}^{k} \sum_{l=1}^{k} x_{i+h-1,j+l-1} w_{h,l}, \tag{31}$$

or

$$\mathbf{Y} = \mathbf{X} * \mathbf{w}, \tag{32}$$

, where $*$ refers to the convolution operation. For simplicity, we omitted the bias term in this question.

Suppose the final loss is $\mathcal{L}$, and the upstream gradient is $d\mathbf{Y} \in \mathbb{R}^{m,m}$,

$$d\mathbf{Y} = \begin{bmatrix} dy_{1,1} & dy_{1,2} & \cdots & dy_{1,m} \\ dy_{2,1} & dy_{2,2} & \cdots & dy_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ dy_{m,1} & dy_{m,2} & \cdots & dy_{m,m} \end{bmatrix}, \tag{33}$$

where $dy_{i,j}$ denotes $\dfrac{\partial \mathcal{L}}{\partial y_{i,j}}$.

(a) **Derive the gradient to the weight matrix** $d\mathbf{w} \in \mathbb{R}^{k,k}$,

$$d\mathbf{w} = \begin{bmatrix} dw_{1,1} & dw_{1,2} & \cdots & dw_{1,k} \\ dw_{2,1} & dw_{2,2} & \cdots & dw_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ dw_{k,1} & dw_{k,2} & \cdots & dw_{k,k} \end{bmatrix}, \tag{34}$$

where $dw_{h,l}$ denotes $\dfrac{\partial \mathcal{L}}{\partial w_{h,l}}$. Also, **derive the weight after one SGD step with a batch of a single image**.

Solution:

The forward propagation rule is

$$y_{i,j} = \sum_{h=1}^{k} \sum_{l=1}^{k} x_{i+h-1,j+l-1} w_{h,l}. \tag{35}$$

Use chain rule

$$\frac{\partial \mathcal{L}}{\partial w_{h,l}} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial y_{i,j}} \frac{\partial y_{i,j}}{\partial w_{h,l}}, \tag{36}$$

and

$$\frac{\partial y_{i,j}}{\partial w_{h,l}} = x_{i+h-1,j+l-1}, \tag{37}$$

so we have

$$dw_{h,l} = \sum_{i=1}^{m} \sum_{j=1}^{m} x_{i+h-1,j+l-1} dy_{i,j} \tag{38}$$

$$d\mathbf{w} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,n} \end{bmatrix} * \begin{bmatrix} dy_{1,1} & dy_{1,2} & \cdots & dy_{1,m} \\ dy_{2,1} & dy_{2,2} & \cdots & dy_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ dy_{m,1} & dy_{m,2} & \cdots & dy_{m,m} \end{bmatrix} \tag{39}$$

$$= \mathbf{X} * d\mathbf{Y}. \tag{40}$$

For one SGD step with learning rate η, suppose the input and output are $\mathbf{X}$ and $\mathbf{Y}$, then

$$\mathbf{w_{t+1}} = \mathbf{w_t} - \eta d\mathbf{w} = \mathbf{w_t} - \eta \mathbf{X} * d\mathbf{Y} \tag{41}$$

We can conclude that **during the training of CNN weights with SGD, the weights of a CNN are a weighted average of images patches in the dataset**. This is because the gradient, which is a weighted

average of image patches in the dataset, is used to update the weights in SGD, so the trained weight is the initial weight plus a weighted average of the gradients.

(b) The objective of this part is to investigate the effect of spatial weight sharing in convolution layers on the behavior of gradient norms with respect to changes in image size.

For simplicity of analysis, we assume $x_{i,j}, dy_{i,j}$ are independent random variables, where for all i, j:

$$\mathbb{E}[x_{i,j}] = 0, \tag{42}$$
$$\mathrm{Var}(x_{i,j}) = \sigma_x^2, \tag{43}$$
$$\mathbb{E}[dy_{i,j}] = 0, \tag{44}$$
$$\mathrm{Var}(dy_{i,j}) = \sigma_g^2. \tag{45}$$

Derive the mean and variance of $dw_{h,l} = \dfrac{\partial \mathcal{L}}{\partial W_{h,l}}$ for each i, j a function of n, k, σ_x, σ_g. What is the asymptotic growth rate of the standard deviation of the gradient on $dw_{h,l}$ with respect to the length and width of the image n?

Hint: there should be no m in your solution because m can be derived from n and k.

Hint: you cannot assume that $x_{i,j}$ and $dy_{i,j}$ follow normal distributions in your derivation or proof.

Solution:

From the previous part, for each i, j, we have

$$dw_{i,j} = \sum_{h=1}^{m}\sum_{l=1}^{m} dy_{h,l} x_{i+h-1,j+l-1}. \tag{46}$$

Given the assumption that $dy_{h,l}$ and $x_{i,j}$ are independent random variables, we have $\mathbb{E}[dy_{h,l}x_{i,j}] = \mathbb{E}[dy_{h,l}]\mathbb{E}[x_{i,j}]$ for every i, j, h, l, so:

$$\mathbb{E}[dw_{i,j}] = \sum_{h=1}^{m}\sum_{l=1}^{m} \mathbb{E}[dy_{h,l}]\mathbb{E}[x_{i+h-1,j+l-1}] \tag{47}$$
$$= 0. \tag{48}$$

As for the variance, similarly, we have $\mathbb{E}[dy_{h,l}^2 x_{i,j}^2] = \mathbb{E}[dy_{h,l}^2]\mathbb{E}[x_{i,j}^2]$ for every i, j, h, l, so:

$$\mathrm{Var}(dw_{i,j}) = \mathbb{E}[dw_{i,j}^2] - \mathbb{E}[dw_{i,j}]^2 \tag{49}$$
$$= \mathbb{E}[dw_{i,j}^2] - 0 \tag{50}$$
$$= \mathbb{E}\left[\left(\sum_{h=1}^{m}\sum_{l=1}^{m} dy_{h,l} x_{i+h-1,j+l-1}\right)^2\right] \tag{51}$$
$$= \sum_{h=1}^{m}\sum_{l=1}^{m} \mathbb{E}[dy_{h,l}^2]\mathbb{E}[x_{i+h-1,j+l-1}^2] \tag{52}$$

$$= \sum_{h=1}^{m}\sum_{l=1}^{m}(\mathrm{Var}(dy_{h,l}) + \mathbb{E}[dy_{h,l}]^2)(\mathrm{Var}(x_{i+h-1,j+l-1}) + \mathbb{E}[x_{i+h-1,j+l-1}]^2) \tag{53}$$

$$= \sum_{h=1}^{m}\sum_{l=1}^{m}\mathrm{Var}(dy_{h,l})\mathrm{Var}(x_{i+h-1,j+l-1}) \tag{54}$$

$$= m^2\sigma_x^2\sigma_g^2. \tag{55}$$

Because there is no padding and stride is 1, we have $m = n - k + 1$, so

$$\mathrm{Std}(dw_{i,j}) = \sqrt{\mathrm{Var}(dw_{i,j})} = \sqrt{(n-k+1)^2\sigma_x^2\sigma_g^2} = \Theta(n), \tag{56}$$

i.e., the stardard variance of the gradient on each parameter grows linearly as the image size increases.

(c) **For a network with only 2x2 max-pooling layers (no convolution layers, no activations), what will be $d\mathbf{X} = [dx_{i,j}] = [\frac{\partial \mathcal{L}}{\partial x_{i,j}}]$? For a network with only 2x2 average-pooling layers (no convolution layers, no activations), what will be $d\mathbf{X}$?**

HINT: Start with the simplest case first, where $\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}$. *Further assume that top left value is selected by the max operation. i.e.*

$$y_{1,1} = x_{1,1} = \max(x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2}) \tag{57}$$

Then generalize to higher dimension and arbitrary max positions.

Solution:

In the simplest case, output $\mathbf{Y}$ has size 1x1. For the max pooling case,

$$\frac{\partial y_{11}}{\partial x_{i,j}} = \begin{cases} 1, & \text{if } (i,j) = 1,1 \\ 0, & \text{otherwise} \end{cases} \tag{58}$$

Combining all four partial derivatives, we have

$$d\mathbf{X} = \begin{bmatrix} dx_{11} & dx_{12} \\ dx_{21} & dx_{22} \end{bmatrix} = \begin{bmatrix} dy_{11} & 0 \\ 0 & 0 \end{bmatrix} \tag{59}$$

For the average pooling case,

$$\frac{\partial y_{11}}{\partial x_{i,j}} = 1/4 \tag{60}$$

$$d\mathbf{X} = \begin{bmatrix} dx_{11} & dx_{12} \\ dx_{21} & dx_{22} \end{bmatrix} = \begin{bmatrix} \frac{dy_{11}}{4} & \frac{dy_{11}}{4} \\ \frac{dy_{11}}{4} & \frac{dy_{11}}{4} \end{bmatrix} \tag{61}$$

In the general setting, for max pooling, we can notice that x and y have a one to one mapping. Each x value is involved in calculation of exactly one y value. Let $k = i//2$, $l = j//2$, where $//$ performs

the floordiv operation, let

$$\delta_{i,j} = \frac{\partial y_{kl}}{\partial x_{i,j}} = \begin{cases} 1, & \text{if } x_{i,j} = \max(x_{i,j}, x_{i+1,j}, x_{i,j+1}, x_{i+1,j+1}) \\ 0, & \text{otherwise} \end{cases} \tag{62}$$

, then

$$dx_{i,j} = \sum_{y_{mn}} dy_{mn} \frac{\partial y_{mn}}{\partial x_{i,j}} = dy_{kl}\delta_{i,j} \tag{63}$$

$d\mathbf{X}$ is the matrix contructed by each $dx_{i,j}$. It will have similar pattern to the simplest case. For each 2x2 block, only one of the input pixel has gradient of magnitude one flowing back, and the rest three inputs have zero gradient.

For the average pooling general case,

$$\delta_{i,j} = \frac{\partial y_{kl}}{\partial x_{i,j}} = \frac{1}{4} \tag{64}$$

$$dx_{i,j} = \sum_{y_{mn}} dy_{mn} \frac{\partial y_{mn}}{\partial x_{i,j}} = dy_{kl}\delta_{i,j} = \frac{dy_{kl}}{4} \tag{65}$$

Average pooling distributes the gradient evenly across each 2x2 input blocks.

(d) Following the previous part, **discuss the advantages of max pooling and average pooling** in your own words.

Hint: you may find it helpful to finish the question "Inductive Bias of CNNs (Coding Question)" before working on this question

Solution: In general, it depends on the specific problem setting to use max pooling or average pooling.

Advantages of max pooling: it learns invariant features. Max pooling has the advantage of retaining strong features and highlighting the presence of an object in the image.

When average pooling is needed: it is used to reduce the spatial dimensions of the feature map while retaining some information about the distribution of values in the region. Average pooling is better at capturing smooth transitions or subtle changes in the feature map. For example, average pooling usually follows the last convolution layer of feature learning.

4. Inductive Bias of CNNs (Coding Question)

In this problem, you will follow the EdgeDetection.ipynb notebook to understand the inductive bias of CNNs.

(a) Overfitting Models to Small Dataset: **Fill out notebook section (Q1).**

(i) Can you find any interesting patterns in the learned filters?

Solution: Somewhat looks like edge detection filters. (Every reasonable answer is correct)

(b) Sweeping the Number of Training Images: **Fill out notebook section (Q2).**

(i) Compare the learned kernels, untrained kernels, and edge-detector kernels. What do you observe?

Solution: When CNN's performance is low (< 90%), Kernels look almost random. Interestingly,

with the dumb luck of initialization, there are some kernels that already look like edge detectors. When CNN's performance is high (> 90%), Kernels look like edge detectors. (Every reasonable answer is correct)

(ii) We freeze the convolutional layer and train only final layer (classifier). In a high data regime, the performance of CNN initialized with edge detectors is worse than CNN initialized with random weights. Why do you think this happens?

Solution: As mentioned earlier, inductive bias benefits both performance and training efficiency. But not always that's the case. The training result can be worse if we inject too much into the training process/model or incorrectly. We can understand it through the lens of bias-variance tradeoff. If we introduce inductive biases to the model or training process, it increases bias while decreasing variance. But note that a large number of data also reduce the variance. The edge detection problem is so simple that 50 images per class are large enough to decrease the decent amount of variance. But we inject too many inductive biases into the model: initializing with edge detection filters and freezing them. Therefore, the bias term dominates the generalization error in the high data regime. For more information, please refer to this paper.

(c) Checking the Training Procedures: **Fill out notebook section (Q3).**

(i) List every epochs that you trained the model. Final accuracy of CNN should be at least 90% for 20 images per class.

Solution: There is no definite answer in this question. If you achieve 90% for 20 images per class with CNN model, your answer is correct

(ii) Check the learned kernels. What do you observe?

Solution: Learned kernels look like edge detectors. (Every reasonable answer is correct)

(iii) (optional) You might find that with the high number of epochs, validation loss of MLP is increasing whild validation accuracy increasing. How can we interpret this?

Solution: As the model overfits to training set, the model becomes overconfident.

(iv) (optional) Do hyperparameter tuning. And list the best hyperparameter setting that you found and report the final accuracy of CNN and MLP.

Solution: There is no definite answer in this question. If you find any hyperparameter configuration that performs better than the given, your answer is correct.

(v) How much more data is needed for MLP to get a competitive performance with CNN? Does MLP really generalize or memorize?

Solution: Any number you found is correct if the validation accuracy of MLP is similar to that of CNN. But you will find that MLP actually memorize the dataset. Considering the data generating process, we can find four significant variables for this dataset: 1) edge location, 2) edge width, 3) edge intensity and 4) background intensity. The first two variables mainly determine the patterns of images. As you can see in the data generation code, we randomly sample from [1, 2, 3, 4, 5] as the edge width and [$1 \sim 28$] as the edge location for each edge. Hence there are 280 possible cases. CNN achieves almost 100% validation accuracy with about 50 training images, but MLP needs a much larger number of data points to achieve matched validation accuracy.

(d) Domain Shift between Training and Validation Set: **Fill out notebook section (Q4).**

(i) Why do you think the confusion matrix looks like this? Why CNN misclassifies the images with edge to the images with no edge? Why MLP misclassifies the images with vertical edge to the images with horizontal edge and vice versa? (Hint: Visualize some of the images in the training and validation set.)

Solution: We can find that both models are overfitting to the training set. However, the patterns that they overfit are pretty different. CNN overfits to the edges that are located in the left half or

upper half of the image. Therefore, it is more likely to classify validatation set with edges that are located in the right half or lower half of the image to the images with no edge. However, MLP overfits to the edges that are located in the fraction of edges that are located in the left upper part and right lower part of the image. Therefore, MLP classifies the images with vertical edge to the images with horizontal edge and vice versa.

(ii) Why do you think MLP fails to learn the task while CNN can learn the task? (Hint: Think about the model architecture.)

Solution: CNN is translational invariant. Therefore, CNN can learn the task even though the edges are located in the different half of the image. However, MLP is not translational invariant. Therefore, MLP fails to learn the task.

(e) When CNN is Worse than MLP: **Fill out notebook section (Q5).**

(i) What do you observe? What is the reason that CNN is worse than MLP? (Hint: Think about the model architecture.)

Solution: CNN utilizes the local correlation of the image. However, the local correlation is destroyed by the random permutation. Therefore, CNN fails to learn the task. However, MLP is not affected by the random permutation because it is 'fully connected'. Therefore, MLP can learn the task.

(ii) Assuming we are increasing kernel size of CNN. Does the validation accuracy increase or decrease? Why?

Solution: The validation accuracy will increase. This is because as kernel gets larger, convolutional layer becomes more 'fully connected'. Therefore, CNN can learn the task even though the local correlation is destroyed by the random permutation.

(iii) How do the learned kernels look like? Explain why.

Solution: The learned kernels are not that different from the randomly initialized kernels. This is because CNN cannot capture the local correlation of the image. Recalling backpropagation of CNN, CNN kernels are optimized in the way that maximizes the correlation of grids of images. (Any reasonable answer is also correct)

(f) Increasing the Number of Classes: **Fill out notebook section (Q6).**

(i) Compare the performance of CNN with max pooling and average pooling. What are the advantages of each pooling method?

Solution: With 40 or less training images, max pooling performs better than average pooling. With 50 or more training image, average pooling is on par or slightly better than max pooling. Average pooling method smooths out the image and hence the sharp features may not be identified when this pooling method is used.Max pooling selects the brighter pixels from the image. It is useful when the background of the image is dark and we are interested in only the lighter pixels of the image. We cannot say that a particular pooling method is better over other generally. The choice of pooling operation is made based on the data at hand.

5. Memory considerations when using GPUs (Coding Question)

In this homework, you will run GPUMemory.ipynb to train a ResNet model on CIFAR-10 using PyTorch and explore its implications on GPU memory.

We will explore various systems considerations, such as the effect of batch size on memory usage and how different optimizers (SGD, SGD with momentum, Adam) vary in their memory requirements.

It is strongly recommended that you start early on this question, since colab daily GPU limits may require you to complete this question over a few days with breaks in between.

(a) Managing GPU memory for training neural networks (Notebook Section 1).

(i) **How many trainable parameters does ResNet-152 have**? What is the **estimated size of the model in MB**?

Solution: ResNet-152 has 58164298 parameters, and on a colab T4 GPU, it is estimated to be 232MB in size.

(ii) Which GPU are you using? **How much total memory does it have**?

Solution: Colab typically has Nvidia T4 with 15360 MB memory.

(iii) After you load the model into memory, **what is the memory overhead (MB) of the CUDA context loaded with the model**?

Solution: It can be anywhere between 500-1000 MB. On a colab T4 GPU, it is about 582 MB.

(b) Optimizer memory usage (Notebook Section 2).

(i) **What is the total memory utilization during training with SGD, SGD with momentum and Adam optimizers**? Report in MB individually for each optimizer.

Solution: Generally SGD would have lowest memory utilization, followed by SGD with momentum. Adam would have maximum memory usage. Example values from a Colab T4 GPU: SGD: 3192.0 MB, SGD with momentum: 3274.0 MB, ADAM: 3358.0 MB.

(ii) **Which optimizer consumes the most memory**? Why?

Solution: ADAM consumes more memory than SGD or SGD with momentum because it stores two moving averages of the gradients and their squares, which are used to estimate the first and second moments of the gradient. Because these moving averages are updated in each step, and they need to be stored in memory.

(c) Batch size, learning rates and memory utilization (Notebook Section 3)

(i) **What is the memory utilization for different batch sizes (4, 16, 64, 256)? What is the largest batch size you were able to train?**

Solution: Exact answers depend on the GPU used by the student. Example memory utilizations (trends are important, not absolute values): 4: 1406.0 MB, 16: 1706.0 MB, 64: 2738.0 MB, 256: 8744.0 MB. Batch sizes >= 1024 result in an out of memory error on a T4 GPU.

(ii) **Which batch size gave you the highest accuracy at the end of 10 epochs?**

Solution: Answers may vary, but typically would be between 16 and 256.

(iii) **Which batch size completed 10 epochs the fastest (least wall clock time)? Why?**

Solution: The largest batch size run (256 or 512) would have completed 10 epochs fastest because they can take advantage of hardware acceleration and parallelism, and they can use memory more efficiently. Specifically, GPUs are based on the SIMD (Single Instruction, Multiple Data) paradigm. This means that if we have a large batch of data, we can perform the same operation on each piece of data in parallel, increasing the throughput we see.

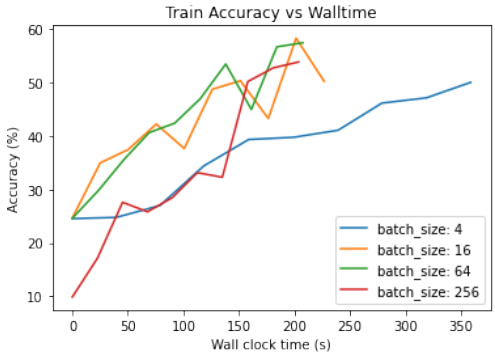
(iv) Attach your **training accuracy vs wall time plots** with your written submission.

Solution: Exact plots may vary. Example accuracy vs wall time plot:

6. 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? Write it down here where you'll need to remember it for the self-grade form.**

Contributors:

- Olivia Watkins.
- Anant Sahai.
- Jake Austin.
- Linyuan Gong.
- Saagar Sanghavi.
- Jerome Quenum.
- Peter Wang.
- Long He.
- Suhong Moon.
- Kumar Krishna Agrawal.
- Romil Bhardwaj.

