EECS545 Machine Learning
Homework #4

Due Date: Tuesday March 19, 2024 at 11:55 PM

Reminder: While you are encouraged to discuss problems in a small group (up to 5 people), you should write your solutions and code independently. Do not share your solution with anyone else in the class. If you worked in a group for the homework, please include the names of your collaborators in the homework submission. Your answers should be as concise and clear as possible. Please address all questions to `http://piazza.com/class#winter2024/eecs545` with a reference to the specific question in the subject line (e.g., Homework 4, Q1(a): foobar).

Submission Instruction: You should submit both **writeup** and **source code**. We may inspect your source code submission visually and rerun the code to check the results. Please be aware your points can be deducted if you don't follow the instructions listed below:

- Submit **writeup** to **Gradescope** (`https://www.gradescope.com/`)
  - Your writeup should contain your answers to **all** questions (typeset or hand-written), except for the implementation-only questions (marked with **(Autograder)**).
  - **For each subquestion, please select the associated pages from the pdf file in Gradescope. The graders are not required to look at pages other than the ones selected, and this may lead to some penalty when grading**.
  - Your writeup should be self-contained and self-explanatory. You should include the plots and figures in your writeup whenever asked, even when they are generated from the ipython notebook.
  - Please typeset (with LaTeX) or hand-write your solutions legibly. **Hand-writing that is difficult to read may lead to penalties**. If you prefer hand-writing, please use scanners or mobile scanning apps that provide shading corrections and projective transformations for better legibility; you **should *not*** just take photos and submit them without any image processing.
  - For all math derivations, you **should** provide detailed explanations as much as possible. If not, you may not be able to get a full credit if there are any mistakes or logical jumps.
- Submit **source code** to **Autograder** (`https://autograder.io/`)
  - Each ipython notebook file (i.e., `*.ipynb`) will walk you through the implementation task, producing the outputs and plots for *all* subproblems. You are required to write code on its matched python file (`*.py`). For instance, if you are working on `linear_regression.ipynb`, you are required to write code on `linear_regression.py`. Note that the outputs of your source code must match with your **writeup**.
  - We will read the data file in the `data` directory (i.e., `data/*.npy`) **from the same (current) working directory**: for example, `X_train = np.load('data/q3x.npy')`.
  - When you want to evaluate and test your implementation, please submit the `*.py` and `*.ipynb` files to Autograder for grading your implementations in the middle or after finish everything. You can partially test some of the files anytime, but please note that this also reduces your daily submission quota. You can submit your code up to **five** times per day.

- The autograder will give you information to help you troubleshoot your code. The following is the current list of error codes (will be updated accordingly if we encounter more common student implementation errors).

  0: Success

  1: The program exited with an unknown test failure.

  2: The program exited with an un-handled exception (usually a runtime error)

  3-5: Some other runtime error indicating an issue with the autograder code.

  40: NotImplementedError

  50: An uncategorized test failure different from the following error codes. Most likely indicates wrong outputs, but this may also include other assertion failures.

  51: There is an incorrect dtype on some output tensor.

  52: There is an incorrect shape on some output tensor.

  53: Some output tensors are not equal, or too different given the tolerance.

  54: Some output tensors are not equal with difference 1.0. Possibly due to rounding issues.

  55: The value of some input tensor was changed.

  60: The model's performance or converged loss is not good enough.

  61: The model might have diverged (e.g. encountered NaN or inf).

  62-63: The model has converged to a bad likelihood.

- Your program should run under an environment with following libraries:
  - Python 3.11 [1]
  - NumPy (for implementations of algorithms)
  - Matplotlib (for plots)
  - PyTorch (for implementing neural networks)

  Please do not use any other library unless otherwise instructed (no third-party libraries other than numpy and matplotlib are allowed). You should be able to load the data and execute your code on Autograder with the environment described above, but in your local working environment you might need to install them via `pip install numpy matplotlib torch torchvision`. For this assignment, you will need to submit the following files:
  - Q1
    - `cnn_layers.py`
    - `cnn.py`
    - `cnn.ipynb`
  - Q2
    - `rnn_layers.py`
    - `rnn.py`
    - `rnn.ipynb`
  - Q3
    - `transfer_learning.py`
    - `transfer_learning.ipynb`
  - Q4
    - `transformer.py`
    - `transformer_trainer.py`
    - `transformer.ipynb`

[1] We recommend using Miniconda (`https://docs.conda.io/en/latest/miniconda.html`). You can create `eecs545` environment with Python 3.11 as `conda create --name eecs545 python=3.11`. It is fine to use other python distributions and versions as long as they are supported, but we will run your program with Python 3.11 on Autograder.

Do not change the filename for the code and ipython notebook file because the Autograder may not find your submission. Please do not submit any other files except `*.py` and `*.ipynb` to Autograder.

- When you are done, please upload your work to Autograder (sign in with your UMich account). To receive the full credit, your code must run and terminate without error (i.e., with exit code 0) in the Autograder. Keep all the cell outputs in your notebook files (`*.ipynb`). We strongly recommend you run **Kernel → Restart Kernel and Run All Cells** (in the Jupyter Lab menu) before submitting your code to Autograder.

Change Log

- rev0 (2024/3/5): Initial Release

Credits

Some problems were adopted from Stanford CS229 and Bishop PRML.

PyTorch Implementation Tips

When initializing a tensor, please set it to the same **dtype** (X.dtype) and **device** (X.device) from the input parameter X, instead of explicitly set an arbitrary one. Please refer to this tutorial for more information if you encountered any dtype or device errors.

1 [34 points] CNNs for Multi-class Classification

In this problem, you will get the opportunity to implement the convolution neural network for classifying MNIST. For better visualization of the convolution operations, you can use the following link: `https://bit.ly/3HORjhn`.

X, Y will represent the input and the output of the layers, respectively. For this problem, we use the row-major notation here (i.e. each row of X and Y corresponds to one data sample) to be compatible with the multi-dimensional array structure of Numpy. Furthermore, L is the scalar valued loss function of Y.

Important Note: In this question, any indices involved (i, j, *etc.*) in the mathematical notation start from 1, and not 0. In your code, however, you should use 0-based indexing (standard Numpy notation).

Consider the following 2D arrays/tensors: $\mathbf{a} \in \mathbb{R}^{H_a \times W_a}$ and $\mathbf{b} \in \mathbb{R}^{H_b \times W_b}$. Note, $\mathbf{a}$ is the image and $\mathbf{b}$ is the filter with $H_a > H_b$ and $W_a > W_b$. Now, consider the following definitions.

Valid convolution: In this problem, we can define the valid convolution as follows:

$$\boxed{(\mathbf{a} *_{\text{valid}} \mathbf{b})_{i,j} = \sum_{m=i}^{i+H_b-1} \sum_{n=j}^{j+W_b-1} a_{m,n} b_{i-m+H_b, j-n+W_b}}$$

Here, $1 \leq i \leq H_a - H_b + 1$ and $1 \leq j \leq W_a - W_b + 1$. Please note that the convolution operation we discussed in class is **valid convolution**, and does not involve any zero padding. This operation produces an output of size $(H_a - H_b + 1) \times (W_a - W_b + 1)$.

Filtering: Moreover, it might also be useful to consider the **filtering** operation $*_{\text{filt}}$, defined by:

$$\boxed{(\mathbf{a} *_{\text{filt}} \mathbf{b})_{i,j} = \sum_{m=i}^{i+H_b-1} \sum_{n=j}^{j+W_b-1} a_{m,n} b_{m-i+1, n-j+1} = \sum_{p=1}^{H_b} \sum_{q=1}^{W_b} a_{i+p-1, j+q-1} b_{p,q}}$$

Here, $1 \leq i \leq H_a - H_b + 1$ and $1 \leq j \leq W_a - W_b + 1$. Please note that the filtering operation generates an output of size $(H_a - H_b + 1) \times (W_a - W_b + 1)$. In summary, the filtering operation is similar to the valid convolution, except that the filter is *not flipped* when computing the weighted sum. You can think of filtering as an inner product between the $H_b \times W_b$ sized kernel and image patch of the same size inside the input image where we don't flip the kernel.

Full convolution: Finally, for deriving the gradient through convolution layer, the full convolution operation may be useful. This operation is defined as follows:

$$\boxed{(\mathbf{a} *_{\text{full}} \mathbf{b})_{i,j} = \sum_{m=i-H_b+1}^{i} \sum_{n=j-W_b+1}^{j} a_{m,n} b_{i-m+1, j-n+1}}$$

Here, $1 \leq i \leq H_a + H_b - 1$ and $1 \leq j \leq W_a + W_b - 1$. The full convolution can be thought of as zero padding $\mathbf{a}$ on all sides with one less than the size of the kernel, and then performing valid convolution using the modified input tensor $\mathbf{a}$. Concretely, this means that we will pad $\mathbf{a}$ by $H_b - 1$ rows on the top and bottom, followed by $W_b - 1$ columns on the left and right. In the definition of full convolution, $a_{m,n} = 0$ if $m < 1$ or $n < 1$ or $m > H_a$ or $n > W_a$. This operation produces an output of size $(H_a + H_b - 1) \times (W_a + W_b - 1)$.

Convolution Layer: Here, assume the input to the layer to be $X \in \mathbb{R}^{N \times C \times H \times W}$, where N is the number of images in the batch, C is the number of channels, H is the height of the image, W is the width of the image. Furthermore, consider a convolutional kernel $K \in \mathbb{R}^{F \times C \times H' \times W'}$, where F represents the number of filters present in this layer. The output of this layer is given by $Y \in \mathbb{R}^{N \times F \times H'' \times W''}$ where we have $H'' = H - H' + 1$ and $W'' = W - W' + 1$.

From the definitions above, we can consider the output of the layer $Y_{n,f}$ to be defined as

$$\boxed{Y_{n,f} = \sum_{c=1}^{C} X_{n,c} *_{\text{valid}} \overline{K}_{f,c}}$$

In this case, $\overline{K}_{f,c}$ represents the flipped filter, which can be more concretely defined for each element as $\overline{K}_{f,c,i,j} = K_{f,c,H'+1-i,W'+1-j}$.

Convolution Backwards: Finally, considering the upstream gradient to be denoted by $\frac{\partial L}{\partial Y_{n,f}}$, use the following formulae to implement the backward pass in `layers.py`.

$$\boxed{\frac{\partial L}{\partial X_{n,c}} = \sum_{f=1}^{F} K_{f,c} *_{\text{full}} \left(\frac{\partial L}{\partial Y_{n,f}}\right)}$$

and

$$\boxed{\frac{\partial L}{\partial K_{f,c}} = \sum_{n=1}^{N} X_{n,c} *_{\text{filt}} \left(\frac{\partial L}{\partial Y_{n,f}}\right)}$$

If you are interested, the derivation is included in the appendix of this problem set.

(a) **[14 points]** (Autograder) Use the definitions and derivations above to implement `conv_forward` and `conv_backward` in the `cnn_layers.py` script.

Note that the output expression $Y_{n,f} = \sum_{c=1}^{C} X_{n,c} *_{\text{valid}} \overline{K}_{f,c}$ can also be represented by the expression $Y_{n,f} = \sum_{c=1}^{C} X_{n,c} *_{\text{filt}} K_{f,c}$. Therefore, you can implement $Y_{n,f} = \sum_{c=1}^{C} X_{n,c} *_{\text{filt}} K_{f,c}$ in the `conv_forward` function in the `cnn_layers.py` script.

You should vectorize your implementations as much as possible following the instruction and hint in `cnn_layers.py`. Even with a vectorized implementation, training on (c) may take a significant amount of time. (The instructor solution takes around 1 hour to finish).

(b) **[14 points]** (Autograder) Implement CNN for softmax multi-class classification using the starter code provided in `cnn.py`. Note that forward, relu, softmax, max pool layers have been implemented for you in `layers.py`. Unlike the two-layer-net you implemented in HW3, we will initialize using a uniform distribution (instead of a normal distribution). This will significantly improve training performance in the next part.

(c) **[6 points]** Train the CNN multi-class classifier on CIFAR-10 dataset with `cnn.ipynb`. Using the hyperparameters specified in the solver instance, train the CNN on the CIFAR-10 dataset and report the generated loss plot (`cnn.png`) and the accuracy obtained on the test set in your **writeup**. Please also submit `cnn.ipynb` to autograder.

2 [29 points] Application to Image Captioning

In this problem, you will apply the RNN module that has been partially implemented to build an image captioning model. The link to download data is provided in the comment in `rnn.ipynb`. Please download `coco_captioning.zip` using the link provided in the notebook and zip the data into the `data` directory.

(a) [**8 points**] (Autograder) At every timestep we use a fully-connected layer to transform the RNN hidden vector at that timestep into scores for each word in the vocabulary. This is very similar to the fully-connected layer that you implemented in HW3. Implement the forward pass in `temporal_fc_forward` function and the backward pass in `temporal_fc_backward` function in `rnn_layers.py`.

(b) [**16 points**] (Autograder) Now that you have the necessary layers in `rnn_layers.py`, you can combine them to build an image captioning model. Implement the forward and backward pass of the model in the `loss` function and the `sample` function for the RNN model in `rnn.py`.

(c) [**5 points**] With the RNN code now ready, run the script `rnn.ipynb` to get learning curves of training loss and the caption samples. Report the learning curves and the caption samples based on your well-trained network in your **writeup**. Please also submit `rnn.ipynb` to autograder.

3 [18 points] Transfer Learning

By working on this problem, you will be more familiar with PyTorch and can run experiments in two major transfer learning scenarios. If you are new to PyTorch, please refer to our PyTorch review materials on canvas. The link to download data is provided in the comment in `transfer_learning.ipynb`. Please unzip `hymenoptera_data.zip` into the `data` directory.
This problem is heavily based on the PyTorch transfer learning tutorial from this link. If you are stuck on this problem, feel free to consult the PyTorch tutorial.

(a) [**10 points**] Fill in the code in the `transfer_learning.py` file. This includes TODO blocks in the `train_model`, `finetune`, and `freeze` functions.

The `train_model` function is a general function for model training. In the `finetune` function, instead of random initialization, we initialize the network with a pre-trained network. The rest of the training looks as usual. In the `freeze` function, the weights are frozen in all of the network except that of the final fully connected layer. This last fully connected layer is replaced with a new one with random weights and only this layer is trained.

After implementing all above, run the script `transfer_learning.ipynb`.

Note: This part will not be autograded. Therefore, please submit your completed code to Autograder. You will get 2/10 points for submitting the files. We will manually grade based on your submitted code. Before submitting `transfer_learning.ipynb`, make sure that you run it thoroughly and have ALL the logs displayed(for example, the result figures of the last cell). For gradescope, please submit a screenshot of the Test Case of Autograder transfer learning section, which should show 2 blue bars of **passed** for submitted transfer_learning.py? and submitted transfer_learning.ipynb?

(b) [**4 points**] Report the train/val loss and accuracy per epoch log generated by `transfer_learning.ipynb` in your **writeup**.

(c) [**4 points**] Please include figures generated by the model's prediction in your **writeup** for both finetuned model predictions and frozen model predictions.

4 [19 points] Transformer Neural Networks

In this problem, you will implement a simple Transformer network in PyTorch and finetune it on interesting datasets. All the models and layers in PyTorch inherit from the `torch.nn.Module` class. You can refer to `https://pytorch.org/docs/stable/generated/torch.nn.Module.html` for more information about how it works. If you are new to PyTorch, please refer to our PyTorch review materials on canvas. Note: part(b) and (c) will not be autograded.

(a) [**5 points**] (Autograder) Let's implement an attention module in PyTorch. Follow the instruction to fill the TODO blocks in the `forward` function of `MaskedAttention` in `transformer.py`. For sanity check, please run `section(A)` of `transformer.ipynb` to check your implementation. Then, submit your `transformer.py` file to autograder.

(b) [**9 points**] Follow the instruction in `transformer_trainer.py` to fill in the missing lines in function `run`. With this function, you can finetune the language model to learn math multiplication. Please run `section(B)` in `transformer.ipynb` and (1) report your final training and testing accuracy, and (2) attach your training loss plot(multiplication_loss.png) in your **writeup**. You should expect a training accuracy higher than 95% and testing accuracy higher than 85%.

(c) [**5 points**] Now let's move on to a more interesting Tiny Stories dataset. The Tiny Stories dataset contains a large collection of artificially generated childrens' stories. The dataset is designed to test whether small language models can learn to produce fluent and coherent text when trained on data with smaller vocabulary and simpler grammar structures (language that 3-4 year olds can understand). We will finetune the model you implemented and see what stories the trained language model will generate! The link to download the dataset is in the comment in `transformer.ipynb`. Run `section(C)` in `transformer.ipynb` and report (1) the final training loss, (2) the three generated stories in the end. Please also submit `transformer.ipynb` to autograder.

A Appendix: Convolution Backwards Derivation

Here we will derive the convolution gradient formulars used in Question 1. To prove this, we will derive the gradients of the filter convolution then use the chain rule to find the final convolution gradient. First, we will first introduce some indexing notation, this will help cut down on the writing and make it easier to see what's going on.

Notation. Let $\mathbf{m} \in \mathbb{R}^{H_m \times W_m}$ be some matrix. We define the "slice" of $\mathbf{m}$ as

$$(\mathbf{m})_{i,j}^{k,\ell}$$

which denotes a matrix of size $(k-i) \times (\ell - j)$, where its elements are defined as

$$\left[(\mathbf{m})_{i,j}^{k,\ell}\right]_{n,m} = \begin{cases} \mathbf{m}_{i+n-1,j+m-1} & \text{if } i+n-1, j+m-1 \text{ is contained in } \mathbf{m} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Intuitively, the slice of $\mathbf{m}$, $(\mathbf{m})_{i,j}^{k,\ell}$ can be thought of as zero-padding $\mathbf{m}$, then taking the i, j to k, ℓ indexes of the padded $\mathbf{m}$.

Consider the 2D arrays/tensors defined the same way as Question: $\mathbf{a} \in \mathbb{R}^{H_a \times W_a}$ and $\mathbf{b} \in \mathbb{R}^{H_b \times W_b}$. Let f be any scalar output function that takes $\mathbf{a} *_{\text{filt}} \mathbf{b}$ as input.

With this notation, we can show the following:

Claim:

$$\frac{\partial (\mathbf{a} *_{\text{filt}} \mathbf{b})_{i,j}}{\partial \mathbf{a}} = (\mathbf{b})_{2-i,2-j}^{H_a-i+1,W_a-j+1} \tag{2}$$

Proof.

$$\begin{aligned}
\frac{\partial (\mathbf{a} *_{\text{filt}} \mathbf{b})_{i,j}}{\partial a_{k,\ell}} &= \frac{\partial}{\partial a_{k,\ell}} \sum_{m=i}^{i+H_b-1} \sum_{n=j}^{j+W_b-1} a_{m,n} b_{m-i+1,n-j+1} && (3) \\
&= \sum_{m=i}^{i+H_b-1} \sum_{n=j}^{j+W_b-1} \frac{\partial}{\partial a_{k,\ell}} a_{m,n} b_{m-i+1,n-j+1} && (4) \\
&= \sum_{m=i}^{i+H_b-1} \sum_{n=j}^{j+W_b-1} b_{m-i+1,n-j+1} \mathbb{1}[m=k, n=\ell] && (5) \\
&= b_{k-i+1,\ell-j+1} && (6)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial (\mathbf{a} *_{\text{filt}} \mathbf{b})_{i,j}}{\partial \mathbf{a}} &= [b_{k-i+1,\ell-j+1}]_{k=1\ldots H_a, \ell=1\ldots W_a} && (7) \\
&= (\mathbf{b})_{1-i+1,1-j+1}^{H_a-i+1,W_a-j+1} && (8) \\
&= (\mathbf{b})_{2-i,2-j}^{H_a-i+1,W_a-j+1} && (9)
\end{aligned}$$

Next, in a similar manner:

Claim:

$$\frac{\partial(\mathbf{a} *_{\text{filt}} \mathbf{b})_{i,j}}{\partial \mathbf{b}} = (\mathbf{a})_{i,j}^{i+H_b-1,j+W_b-1} \tag{10}$$

Proof.

$$\begin{aligned}
\frac{\partial(\mathbf{a} *_{\text{filt}} \mathbf{b})_{i,j}}{\partial b_{k,\ell}} &= \frac{\partial}{\partial b_{k,\ell}} \sum_{p=1}^{H_b} \sum_{q=1}^{W_b} a_{i+p-1,j+q-1} b_{p,q} && (11) \\
&= \sum_{p=1}^{H_b} \sum_{q=1}^{W_b} \frac{\partial}{\partial b_{k,\ell}} a_{i+p-1,j+q-1} b_{p,q} && (12) \\
&= \sum_{p=1}^{H_b} \sum_{q=1}^{W_b} a_{i+p-1,j+q-1} \mathbb{1}[p=k, q=\ell] && (13) \\
&= a_{i+k-1,j+\ell-1} && (14)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial(\mathbf{a} *_{\text{filt}} \mathbf{b})_{i,j}}{\partial \mathbf{b}} &= [a_{i+k-1,j+\ell-1}]_{k=1\ldots H_b, \ell=1\ldots W_b} && (15) \\
&= (\mathbf{a})_{i+1-1,j+1-1}^{i+H_b-1,j+W_b-1} && (16) \\
&= (\mathbf{a})_{i,j}^{i+H_b-1,j+W_b-1} && (17)
\end{aligned}$$

Next, we can show the following (which can be directly used with the chain rule to find the convolution gradients).

Claim:

$$\frac{\partial f}{\partial \mathbf{a}} = \mathbf{b} *_{\text{full}} \frac{\partial f}{\partial \mathbf{a} *_{\text{filt}} \mathbf{b}} \tag{18}$$

Proof. Let $H' = H_a - H_b + 1$ and $W' = W_a - W_b + 1$ (the size of the output of $\mathbf{a} *_{\text{filt}} \mathbf{b}$).

$$\begin{aligned}
\left[\frac{\partial f}{\partial \mathbf{a}}\right]_{k,\ell} &= \left[\sum_{i=1}^{H'}\sum_{j=1}^{W'} \left(\frac{\partial f}{\partial \mathbf{a} *_{\text{filt}} \mathbf{b}}\right)_{i,j} \frac{\partial (\mathbf{a} *_{\text{filt}} \mathbf{b})_{i,j}}{\partial \mathbf{a}}\right]_{k,\ell} && (19)\\
&= \sum_{i=1}^{H'}\sum_{j=1}^{W'} \left(\frac{\partial f}{\partial \mathbf{a} *_{\text{filt}} \mathbf{b}}\right)_{i,j} \left(\frac{\partial (\mathbf{a} *_{\text{filt}} \mathbf{b})_{i,j}}{\partial \mathbf{a}}\right)_{k,\ell} && (20)\\
&= \sum_{i=1}^{H'}\sum_{j=1}^{W'} \left(\frac{\partial f}{\partial \mathbf{a} *_{\text{filt}} \mathbf{b}}\right)_{i,j} \left((\mathbf{b})_{2-i,2-j}^{H_a-i+1,W_a-j+1}\right)_{k,\ell} && (21)\\
&= \sum_{i=1}^{H'}\sum_{j=1}^{W'} \left(\frac{\partial f}{\partial \mathbf{a} *_{\text{filt}} \mathbf{b}}\right)_{i,j} (\mathbf{b})_{2-i+k-1,2-j+\ell-1} && (22)\\
&= \sum_{i=1}^{H'}\sum_{j=1}^{W'} (\mathbf{b})_{k-i+1,\ell-j+1} \left(\frac{\partial f}{\partial \mathbf{a} *_{\text{filt}} \mathbf{b}}\right)_{i,j} && (23)\\
&= \sum_{m=k-H'+1}^{k}\;\sum_{n=\ell-W'+1}^{\ell} (\mathbf{b})_{m,n} \left(\frac{\partial f}{\partial \mathbf{a} *_{\text{filt}} \mathbf{b}}\right)_{k-m+1,\ell-n+1} && (24)\\
&= \left(\mathbf{b} *_{\text{full}} \frac{\partial f}{\partial \mathbf{a} *_{\text{filt}} \mathbf{b}}\right)_{k,\ell} && (25)
\end{aligned}$$

and the following:

Claim:

$$\frac{\partial f}{\partial \mathbf{b}} = \mathbf{a} *_{\text{filt}} \frac{\partial f}{\partial \mathbf{a} *_{\text{filt}} \mathbf{b}} \tag{26}$$

Proof. Let $H' = H_a - H_b + 1$ and $W' = W_a - W_b + 1$ (the size of the output of $\mathbf{a} *_{\text{filt}} \mathbf{b}$).

$$\begin{aligned}
\left[\frac{\partial f}{\partial \mathbf{b}}\right]_{k,\ell} &= \left[\sum_{i=1}^{H'}\sum_{j=1}^{W'} \left(\frac{\partial f}{\partial \mathbf{a} *_{\text{filt}} \mathbf{b}}\right)_{i,j} \frac{\partial (\mathbf{a} *_{\text{filt}} \mathbf{b})_{i,j}}{\partial \mathbf{b}}\right]_{k,\ell} && (27)\\
&= \sum_{i=1}^{H'}\sum_{j=1}^{W'} \left(\frac{\partial f}{\partial \mathbf{a} *_{\text{filt}} \mathbf{b}}\right)_{i,j} \left(\frac{\partial (\mathbf{a} *_{\text{filt}} \mathbf{b})_{i,j}}{\partial \mathbf{b}}\right)_{k,\ell} && (28)\\
&= \sum_{i=1}^{H'}\sum_{j=1}^{W'} \left(\frac{\partial f}{\partial \mathbf{a} *_{\text{filt}} \mathbf{b}}\right)_{i,j} \left((\mathbf{a})_{i,j}^{i+H'-1,j+W'-1}\right)_{k,\ell} && (29)\\
&= \sum_{i=1}^{H'}\sum_{j=1}^{W'} \left(\frac{\partial f}{\partial \mathbf{a} *_{\text{filt}} \mathbf{b}}\right)_{i,j} (\mathbf{a})_{i+k-1,j+\ell-1} && (30)\\
&= \sum_{i=1}^{H'}\sum_{j=1}^{W'} (\mathbf{a})_{i+k-1,j+\ell-1} \left(\frac{\partial f}{\partial \mathbf{a} *_{\text{filt}} \mathbf{b}}\right)_{i,j} && (31)\\
&= \left(\mathbf{a} *_{\text{filt}} \frac{\partial f}{\partial \mathbf{a} *_{\text{filt}} \mathbf{b}}\right)_{k,\ell} && (32)
\end{aligned}$$

Given Equations 18 and 26, we can simply derive the convolution gradient using the chain rule. Recall that

$$Y_{n,f} = \sum_{c}^{C} X_{n,c} *_{\text{valid}} \overline{K}_{f,c} = \sum_{c}^{C} X_{n,c} *_{\text{filt}} K_{f,c} \tag{33}$$

As L is a scalar valued function which takes $Y_{n,f}$ as input,

$$\frac{\partial L}{\partial X_{n,c}} = \sum_{f=1}^{F} \frac{\partial L}{\partial Y_{n,f}} \frac{\partial Y_{n,f}}{\partial X_{n,c}} = \sum_{f=1}^{F} K_{f,c} *_{\text{full}} \frac{\partial L}{\partial Y_{n,f}} \tag{34}$$

and

$$\frac{\partial L}{\partial K_{f,c}} = \sum_{n=1}^{N} \frac{\partial L}{\partial Y_{n,f}} \frac{\partial Y_{n,f}}{\partial K_{f,c}} = \sum_{n=1}^{N} X_{n,c} *_{\text{filt}} \frac{\partial L}{\partial Y_{n,f}}. \tag{35}$$