CS 6301.503 Spring 2019 Scott C. Waggener (scw180000)

Homework 3 (Calculus) Problem 1

Solution: Complete ■

CS 6301.503 Spring 2019 Scott C. Waggener (scw180000)
Homework 3 (Calculus) Problem 2

Solution: Complete ■

CS 6301.503 Spring 2019 Scott C. Waggener (scw180000)

Homework 3 (Calculus) Problem 3

Solution: Complete ■

CS 6301.503 Spring 2019 Scott C. Waggener (scw180000)

Homework 3 (Calculus) Problem 4

Let $\boldsymbol{x}$ be the $K \times 1$ vector output of the last layer of a xNN and $e = \text{crossEntropy}(\boldsymbol{p}^*, \text{softMax}(\boldsymbol{x}))$ be the error where $\boldsymbol{p}^*$ is a $K \times 1$ vector with a 1 in position k^* representing the correct class and 0s elsewhere. Derive $\partial e / \partial \boldsymbol{x}$

Solution:

$$\frac{\partial e}{\partial \boldsymbol{x}} = \begin{pmatrix} p(x_0) \\ \vdots \\ p(x_k) - 1 \\ \vdots \\ p(x_n) \end{pmatrix} \leftarrow k \tag{1}$$

■

Proof: First, note that error function e is given by the cross entropy of a softmax, the typical error function chosen for **classification** networks. We can tell by the given error function that we will need to apply the chain rule. Specifically, we will need to compute the following for use in the chain rule.

$$\frac{d}{d\boldsymbol{x}} \text{softMax}(\boldsymbol{x}) \qquad\qquad \frac{\partial}{\partial \boldsymbol{p}} \text{crossEntropy}(\boldsymbol{p}^*, \boldsymbol{p}) \tag{2}$$

We will start with cross entropy. Recall that cross entropy is given by

$$\text{crossEntropy}(\boldsymbol{p}^*, \boldsymbol{p}) \equiv c(\boldsymbol{p}^*, \boldsymbol{p}) = -\sum_{x_i \in \boldsymbol{x}} p^*(x_i) \log p(x_i) \tag{3}$$

Differentiating cross entropy for each of $x_i \in \boldsymbol{x}$ produces a gradient vector. We know that $\boldsymbol{p}^*$ is a one hot vector at position k meaning that our gradient will be nonzero only at position k upon differentiation. Applying the log derivative identity gives

$$\nabla_{\boldsymbol{x}}\Big(-\sum_{x_i\in\boldsymbol{x}} p^*(x_i)\log p(x_i)\Big) = \begin{pmatrix} -\frac{\partial}{\partial x_1}p^*(x_1)\log p(x_1) \\ -\frac{\partial}{\partial x_2}p^*(x_2)\log p(x_2) \\ \vdots \\ -\frac{\partial}{\partial x_k}p^*(x_k)\log p(x_k) \\ \vdots \\ -\frac{\partial}{\partial x_N}p^*(x_N)\log p(x_N) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ \underbrace{-\frac{\partial}{\partial x}\log p(x_k)}_{-1/p(x_k)} \\ \vdots \\ 0 \end{pmatrix} \tag{4}$$

$$= \begin{pmatrix} 0 \\ 0 \\ \vdots \\ -\frac{1}{p(x_k)} \\ \vdots \\ 0 \end{pmatrix} \leftarrow k \tag{5}$$

So our cross entropy gradient for a one hot vector $\boldsymbol{p}^*$ with nonzero position k is 0 everywhere except at position k.

Now we compute our next required derivative: softmax. The softmax function is given by

$$\text{softMax}(\boldsymbol{x}) = \frac{1}{\sum_{x_i\in\boldsymbol{x}} e^{x_i}} \begin{pmatrix} e^{x_0} \\ e^{x_1} \\ \vdots \\ e^{x_K} \end{pmatrix} \tag{6}$$

Calculating this derivative will yield a Jacobian matrix

$$\frac{\partial s}{\partial \boldsymbol{x}} = \Big(\sum_{x_i\in\boldsymbol{x}} e^{x_i}\Big)^{-1} \frac{\partial}{\partial \boldsymbol{x}} \begin{pmatrix} e^{x_0} \\ e^{x_1} \\ \vdots \\ e^{x_K} \end{pmatrix} + \frac{\partial}{\partial \boldsymbol{x}}\Big(\sum_{x_i\in\boldsymbol{x}} e^{x_i}\Big)^{-1} \begin{pmatrix} e^{x_0} \\ e^{x_1} \\ \vdots \\ e^{x_K} \end{pmatrix} \tag{7}$$

$$= \underbrace{\Big(\sum_{x_i\in\boldsymbol{x}} e^{x_i}\Big)^{-1} \begin{pmatrix} e^{x_0} & 0 & \cdots & 0 \\ 0 & e^{x_1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & e^{x_n} \end{pmatrix}}_{\begin{matrix} s(\boldsymbol{x})|i & \forall i = j \\ 0 & \forall i \neq j \end{matrix}} - \underbrace{\Big(\sum_{x_i\in\boldsymbol{x}} e^{x_i}\Big)^{-2} \begin{pmatrix} e^{2x_0} & e^{x_1+x_0} & \cdots & e^{x_n+x_0} \\ e^{x_0+x_1} & e^{2x_1} & \cdots & e^{x_n+x_1} \\ \vdots & \vdots & \ddots & \vdots \\ e^{x_0+x_n} & e^{x_1+x_n} & \cdots & e^{2x_n} \end{pmatrix}}_{\begin{matrix} s(\boldsymbol{x})^2|i & \forall i = j \\ s(\boldsymbol{x})|i \cdot s(\boldsymbol{x})|j & \forall i \neq j \end{matrix}} \tag{8}$$

(9)

In this Jacobian matrix the entries come from two possible cases that can both be expressed in terms of the original function. To express this more cleanly let us denote the softmax at a

given $x_i \in \boldsymbol{x}$ with the same $p(x_i)$ notation used for handling cross entropy. Then the entry of the ith row and jth column in the Jacobian matrix for softmax of vector $\boldsymbol{x}$ will be given by

$$J_s(\boldsymbol{x})_{ij} = \begin{cases} p(x_i) - p(x_i)^2 & \forall i = j \\ -p(x_i) \cdot p(x_j) & \forall i \neq j \end{cases} \tag{10}$$

Now we have found what we need to apply the chain rule

$$\frac{\partial e}{\partial \boldsymbol{x}} = \boldsymbol{J}_s(\boldsymbol{x}) \cdot \nabla_{\boldsymbol{x}} c(\boldsymbol{p}^*, \boldsymbol{p}) \tag{11}$$

$$= \begin{pmatrix} p(x_0) - p(x_0)^2 & -p(x_0) \cdot p(x_1) & \cdots & -p(x_0) \cdot p(x_n) \\ -p(x_1) \cdot p(x_0) & p(x_1) - p(x_1)^2 & \cdots & -p(x_1) \cdot p(x_n) \\ \vdots & \vdots & \ddots & \vdots \\ -p(x_n) \cdot p(x_0) & -p(x_n) \cdot p(x_1) & \cdots & p(x_n) - p(x_n)^2 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ \vdots \\ -\frac{1}{p(x_k)} \\ \vdots \\ 0 \end{pmatrix} \tag{12}$$

$$= \begin{pmatrix} \frac{p(x_0) \cdot p(x_k)}{p(x_k)} \\ \vdots \\ \frac{p(x_k)^2 - p(x_k)}{p(x_k)} \\ \vdots \\ \frac{p(x_n) \cdot p(x_k)}{p(x_k)} \end{pmatrix} \tag{13}$$

$$\frac{\partial e}{\partial \boldsymbol{x}} = \begin{pmatrix} p(x_0) \\ \vdots \\ p(x_k) - 1 \\ \vdots \\ p(x_n) \end{pmatrix} \leftarrow k \tag{14}$$

This result is very clean and intuitive. It shows that increasing the probabilities for incorrect labels increases error, and that decreasing probability for the correct label increases error. We were able to get this result by treating softmax and cross entropy derivatives together via the chain rule. This property can only be exploited in implementation by using procedures in the library that treat these layers together.

□

CS 6301.503 Spring 2019 Scott C. Waggener (scw180000)

Homework 3 (Calculus) Problem 5

Solution:

$$\frac{\partial e}{\partial \boldsymbol{x}} = \left(\boldsymbol{H}^T \cdot \boldsymbol{I}_n \cdot \boldsymbol{I}_n\{0,1\} \cdot \boldsymbol{I}_n \cdot \frac{\partial e}{\partial \boldsymbol{y}} \right) + \boldsymbol{I}_n \tag{15}$$

■

Proof: First we decompose the residual block representing the flow of information in backpropagation: residual input, pointwise nonlinearity, bias vector, and dense layer.

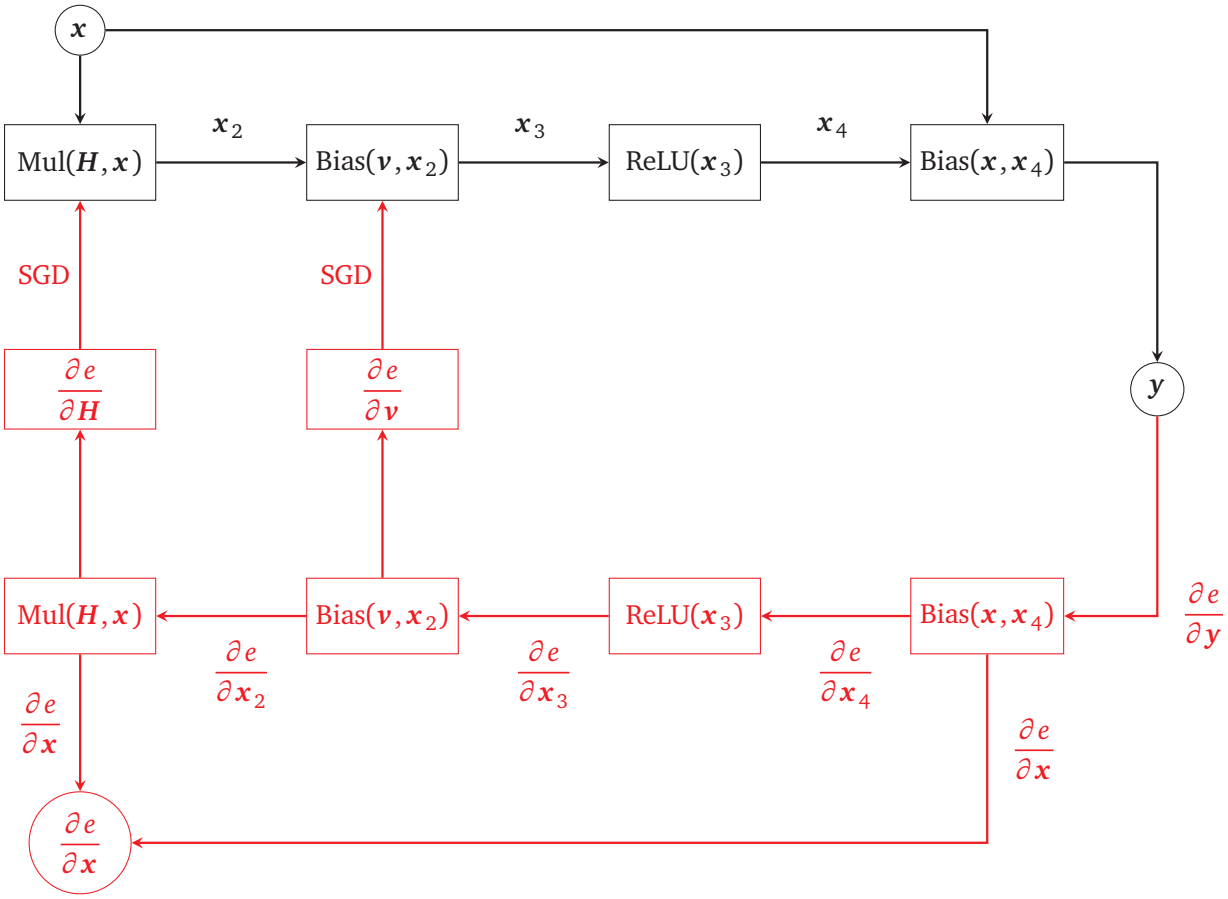
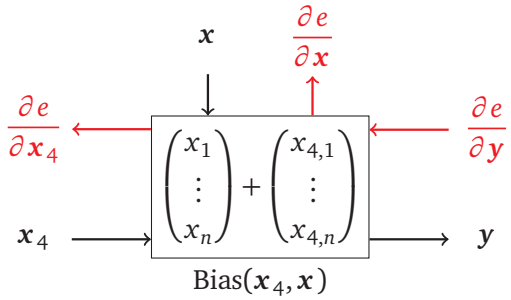


Figure 1. Decomposition of a residual layer

In order to obtain $\partial e / \partial \boldsymbol{x}$ we need to apply the chain rule to move along blocks in the backwards pass. Starting with residual bias block, we have



By definition the bias block is composed of a linear combination of independent vectors, meaning that our Jacobian reduces to an identity matrix before application of the chain rule.

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial x_4} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 1 \end{pmatrix} = I_n \tag{16}$$

And completing the chain rule we have

$$\frac{\partial e}{\partial x} = \frac{\partial y}{\partial x} \cdot \frac{\partial e}{\partial y} \qquad \frac{\partial e}{\partial x_4} = \frac{\partial y}{\partial x_4} \cdot \frac{\partial e}{\partial y} \tag{17}$$

$$= I_n \cdot \frac{\partial e}{\partial y} \qquad = I_n \cdot \frac{\partial e}{\partial y} \tag{18}$$

Next we have the ReLU block. This block has no weights to update, and we treat its Jacobian as an identity matrix

$$\frac{\partial e}{\partial x_3} = \frac{\partial x_4}{\partial x_3} \cdot \frac{\partial e}{\partial x_4} \tag{19}$$

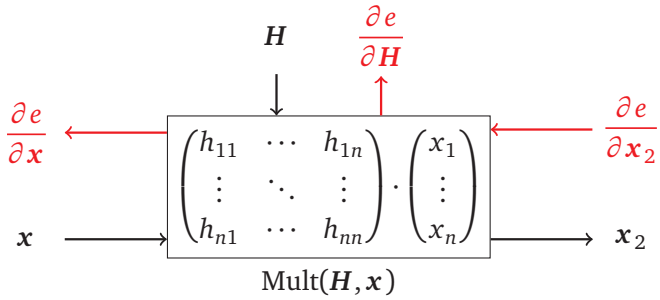
$$= I_n\{0,1\} \cdot \left(I_n \cdot \frac{\partial e}{\partial y} \right) \tag{20}$$

We pass through another trivial bias block

$$\frac{\partial e}{\partial x_2} = \frac{\partial x_3}{\partial x_2} \cdot \frac{\partial e}{\partial x_3} \tag{21}$$

$$= I_n \cdot \left(I_n\{0,1\} \cdot I_n \cdot \frac{\partial e}{\partial y} \right) \tag{22}$$

Lastly we have a multiplication block



The Jacobian for a multiplication block is given in the notes as the transpose of the term that we are not differentiating with respect to. This can always be verified manually if needed.

$$\frac{\partial e}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{x}_2}{\partial \boldsymbol{x}} \cdot \frac{\partial e}{\partial \boldsymbol{x}_2} \tag{23}$$

$$= \boldsymbol{H}^T \cdot \left(\boldsymbol{I}_n \cdot \boldsymbol{I}_n\{0,1\} \cdot \boldsymbol{I}_n \cdot \frac{\partial e}{\partial \boldsymbol{y}} \right) \tag{24}$$

Finally, we must remember that as a residual layer the backwards pass graph has a merge of two gradients into $\partial e/\partial \boldsymbol{x}$. One we finished computing through the dense layer, and the other comes from the residual block we computed first. This gives the final result

$$\frac{\partial e}{\partial \boldsymbol{x}} = \frac{\partial e}{\partial \boldsymbol{x}_2} \cdot \frac{\partial \boldsymbol{x}_2}{\partial \boldsymbol{x}} \tag{25}$$

$$= \left(\boldsymbol{H}^T \cdot \boldsymbol{I}_n \cdot \boldsymbol{I}_n\{0,1\} \cdot \boldsymbol{I}_n \cdot \frac{\partial e}{\partial \boldsymbol{y}} \right) + \boldsymbol{I}_n \tag{26}$$

Shown in the original figure, we have a flow of gradients that looks like

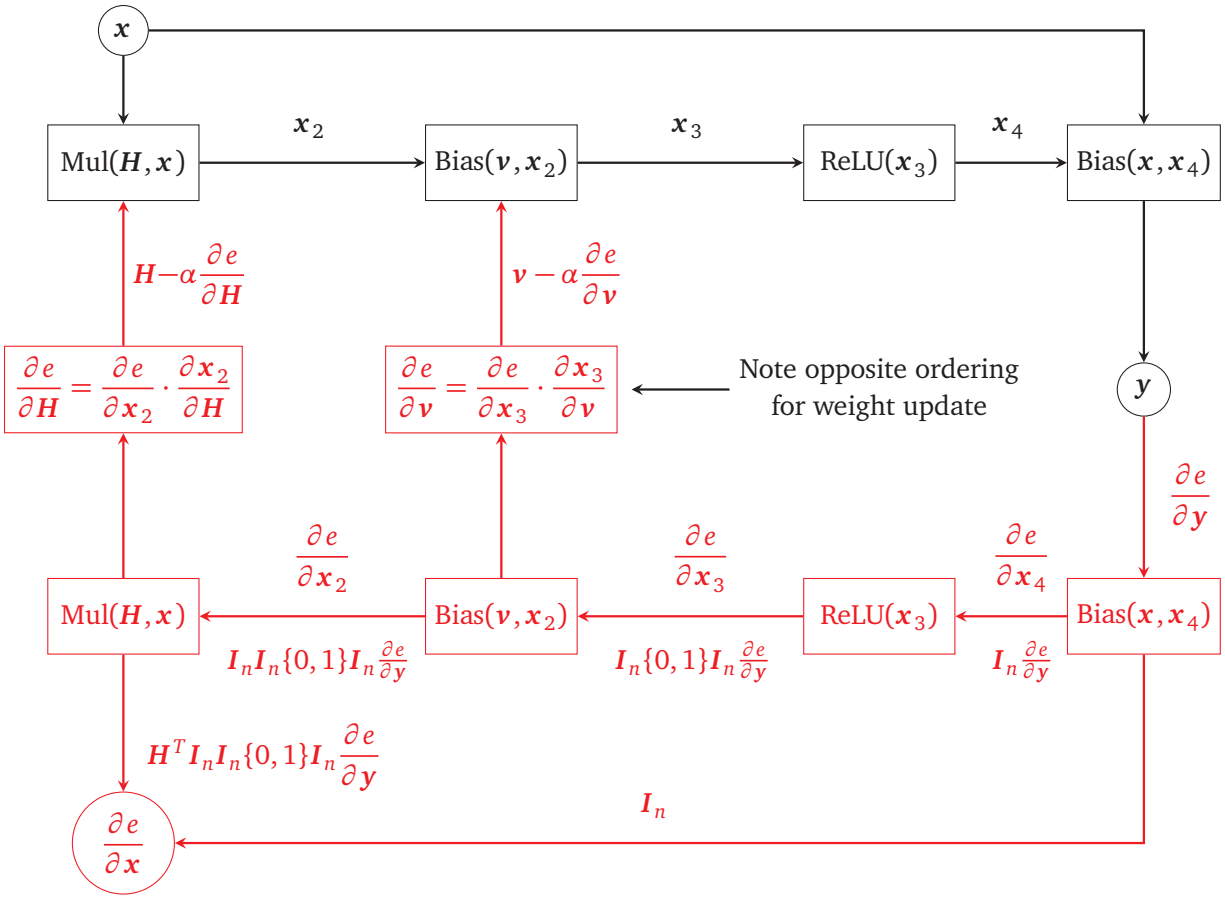


Figure 2. Chain rule applied over backwards pass

□

CS 6301.503 Spring 2019 Scott C. Waggener (scw180000)
Homework 3 (Calculus) Problem 6

Solution: At the end of the last problem we created a graph depicting the backward pass and how gradients flow across blocks. This graph also shows the generic form of the gradient descent updates. All we have to do is express these generic forms in terms of known variables

$$\boldsymbol{H} - \alpha \frac{\partial e}{\partial \boldsymbol{H}} = \boldsymbol{H} - \alpha \left(\frac{\partial e}{\partial \boldsymbol{x}_2} \cdot \frac{\partial \boldsymbol{x}_2}{\partial \boldsymbol{x}} \right) \tag{27}$$

$$= \boldsymbol{H} - \alpha \left(\left(\boldsymbol{I}_n \boldsymbol{I}_n \{0,1\} \boldsymbol{I}_n \frac{\partial e}{\partial \boldsymbol{y}} \right) \cdot \left(\boldsymbol{x}^T \right) \right) \tag{28}$$

$$\boldsymbol{v} - \alpha \frac{\partial e}{\partial \boldsymbol{v}} = \boldsymbol{v} - \alpha \left(\frac{\partial e}{\partial \boldsymbol{x}_3} \cdot \frac{\partial \boldsymbol{x}_3}{\partial \boldsymbol{x}_2} \right) \tag{29}$$

$$= \boldsymbol{v} - \alpha \left(\left(\boldsymbol{I}_n \{0,1\} \boldsymbol{I}_n \frac{\partial e}{\partial \boldsymbol{y}} \right) \cdot \left(\boldsymbol{I}_n \right) \right) \tag{30}$$

For the multiplication block we see that the weight update depends in part on $\boldsymbol{x}$. As such, we will need to store $\boldsymbol{x}$ from the forward pass in order to perform gradient descent during the backward pass.

Furthermore, the diagonal matrix produced as we differentiate ReLU will depend on what inputs were given to ReLU in the forward pass. Thus we must store input $\boldsymbol{x}_3$ where we map $x_i \in \boldsymbol{x}_3 < 0$ to 0 in $\boldsymbol{I}\{0,1\}$.

■

CS 6301.503 Spring 2019 Scott C. Waggener (scw180000)

Homework 3 (Calculus) Problem 7

Solution: This problem expands on the summation proved in the last homework. Slide 76 of the linear algebra lecture provides a nice picture of what is going on. The yellow dense Toeplitz matrix for $\boldsymbol{X}$ shows concretely the role of edge effects in producing varying numbers of duplicated filter weights. Such edge effects are the result of elements in an input feature map being passed over by a varying number of filter kernel weights based on their location. For example, column edge elements will receive only one filter weight across a given row, while inner elements will be exposed to the filter kernel multiple times.

In order to avoid wasting memory on duplicated inputs we do not store the entire Toeplitz matrix. From the picture on slide 76 it is clear that no duplications occur along a row of the Toeplitz matrix. It is also clear from the indexing for $\boldsymbol{X}_{f_r f_c}$ given in the problem that a row of the Toeplitz matrix contains inputs of one feature map that are all passed over by the same filter coefficient. As such, skipping $F_r * F_c$ rows will take us between input feature maps.

With that background, we can start by drawing a rough approximation of the summation of matrix matrix multiplications as follows.

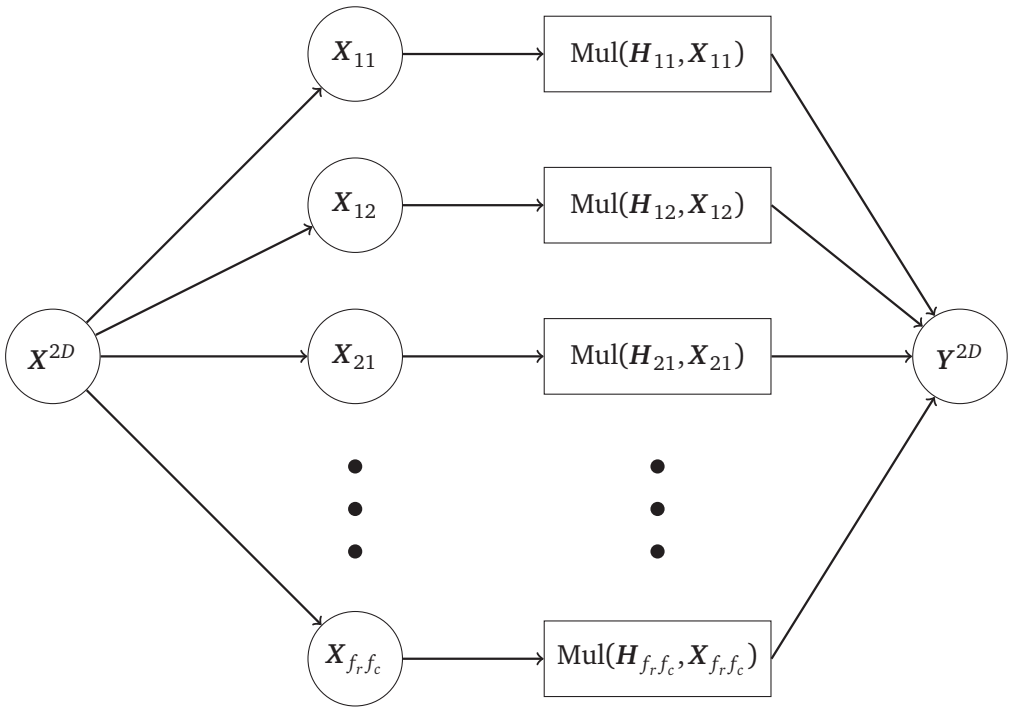


Figure 3. Outline of CNN style 2D convolution as sum of matrix multiplications

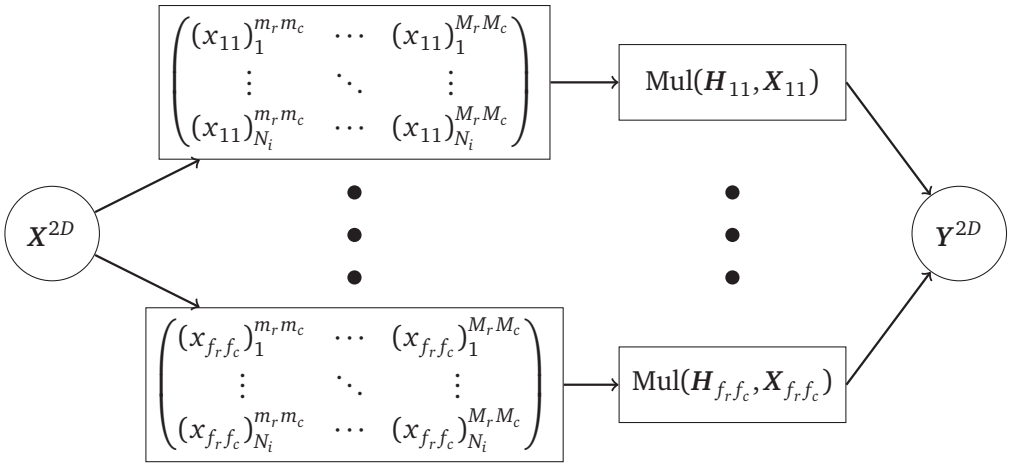


Figure 4. Attempt at showing matrices that contribute to the summation

We can then draw a **rough** outline of the backward pass

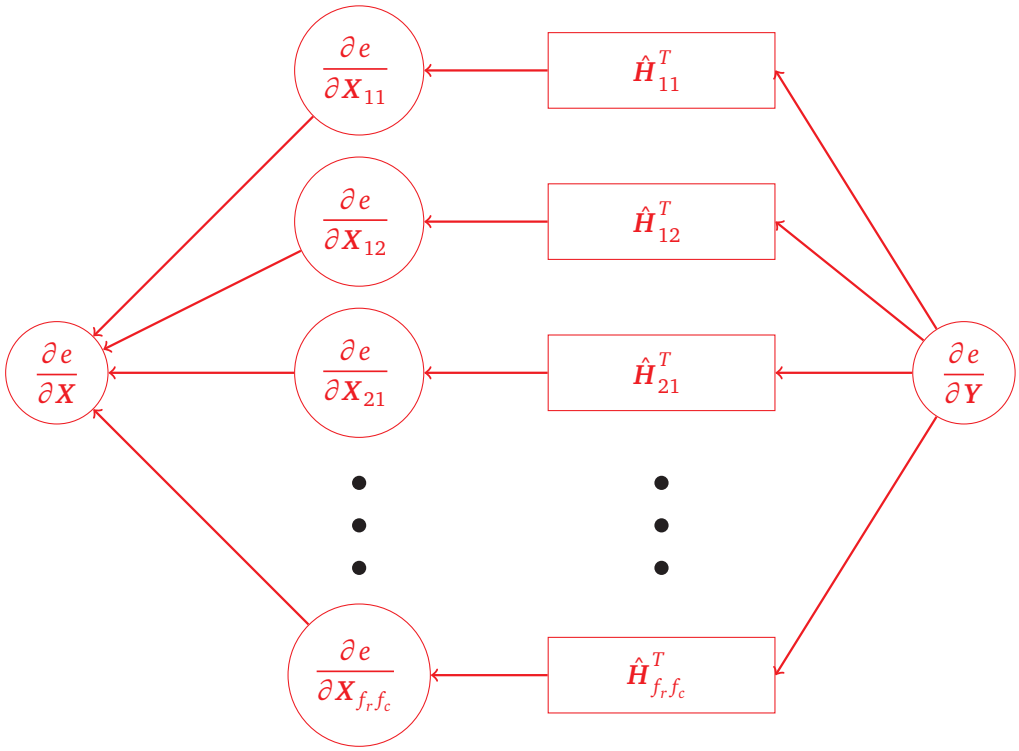


Figure 5. Backward pass outline of CNN style 2D convolution as sum of matrix multiplications

First, note that the backward pass strongly resembles a forward pass, but acting on differently shaped input with transposed block diagonal filters. As such, we should be able to express the backward pass as a CNN style 2D convolution or equivalent sum of lowered matrix matrix multiplications. However, to express the backpropagation step as a lowered CNN style 2D convolution we would need to modify $\partial e / \partial \boldsymbol{Y}$ in the same way we did $\boldsymbol{X}$ in the original lowering.

This would let us write something like

$$\frac{\partial e}{\partial \boldsymbol{X}^{2D}} = \sum_{f_r f_c} \left(\boldsymbol{H}^{2D}_{f_r f_c} \right)^T \frac{\partial e}{\partial \boldsymbol{Y}^{2D}_{f_r f_c}} \tag{31}$$

with associated CNN style 2D convolution

$$\frac{\partial e}{\partial \boldsymbol{X}} = \boldsymbol{H}^T \frac{\partial e}{\partial \boldsymbol{Y}} \tag{32}$$

My best guess on how to create $\boldsymbol{Y}^{2D}$ is that we would follow the process by which the Toeplitz matrix created $\boldsymbol{X}^{2D}$. We could express $(\boldsymbol{H}^{2D})^T$ as $N_i \times (N_o F_r F_c)$ and $\boldsymbol{Y}^{2D}$ as $N_o F_r F_c \times M_r M_c$. If we then pad $\boldsymbol{Y}^{2D}$ appropriately such that we convert M_r, M_c to $L_r L_c$ we would have a lowered matrix matrix expression that would produce the correct output. The padding would be probably be more easily visualized if it was applied using normal convolution padding/stride rules before lowering $\boldsymbol{Y}$.

■

CS 6301.503 Spring 2019 Scott C. Waggener (scw180000)
Homework 3 (Calculus) Problem 8

Solution: Complete

I tried doubling various parameters and for the most part the impact on training time and accuracy was negligible. However, doubling the number of output feature maps or adding additional layers did create a substantial impact. It was a nice reflection that network design is not arbitrary, and that guessing network architecture would involve a lot of trial and error. ■

CS 6301.503 Spring 2019 Scott C. Waggener (scw180000)

Homework 3 (Calculus) Problem 9

Solution: Complete ■