CmpE 597: Homework 1-Report

B. Can Koban
Student No: 2021700096
Department of Computer Engineering
Bogazici University
Istanbul, Turkey, 34342
burakcan.koban@boun.edu.tr

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

1 Equations for Forward and Backward Passes

1.1 Convolution Layer

Forward pass equation for the convolution layer is the following. $\star$ is a special character to represent convolution operation between two matrices.

$$Y_i = B_i + \sum_{j=1}^{n} X_j \star K_{ij},\ i = 1...d \tag{1}$$

where,
Y_i : Output Matrix
B_i : Bias Matrix
X_j : Input Matrix
K_{ij} : Kernels

Table 1: Dimension Conventions

X	Depth x Height x Width
K	Channels x Depth x Height x Width
Y	Depth x Height x Width
B	Depth x Height x Width

Table 2: Dimensions of Convolution Layers for Forward

Layer	**X**	**K**	**B**	**Y**
Convolution Layer 1	(1x28x28)	(4x1x5x5)	(4x24x24)	(4x24x24)
Convolution Layer 2	(4x12x12)	(8x4x5x5)	(8x8x8)	(8x8x8)

Following equations are the backward calculations of convolution layer. E stands for output gradient coming from next layer. There are three calculation which are stands for gradient of kernel, bias and input. Kernel and bias gradients will be used in updating step. On the other hand input gradient which is the third one is the output gradient of previous layer. $R^1 80()$ is a function to rotate K 180^o as it should be done in real convolution operation. However, note that, I don't rotate filters in other convolution calculations.

$$\frac{\partial E}{\partial K_{ij}} = X_j \ \star \ \frac{\partial E}{\partial Y_i} \tag{2}$$

$$\frac{\partial E}{\partial Bi} = \frac{\partial E}{\partial Y_i} \tag{3}$$

$$\frac{\partial E}{\partial Xj} = \sum_{i=1}^{n} \frac{\partial E}{\partial Y_i} \ \star R^{180}(K_{ij}) \tag{4}$$

where,
$\frac{\partial E}{\partial K_{ij}}$: Kernel Gradient
$\frac{\partial E}{\partial Bi}$: Bias Gradient
$\frac{\partial E}{\partial Xj}$: Input Gradient
$\frac{\partial E}{\partial Yi}$: Output Gradient

Although following equations defines the calculations in the project, there is something wrong about them. K, B and X are actually matrices and derivative with respect to matrix is not defined in mathematics. To make it clear, I want to write one of them open. Following matrix and equation shows the open some part of the previous equation. Of course, this is not the correct sizes of this matrix and it is just a representation. However, it can be generalized to other equations too.

$$\frac{\partial E}{\partial K_{ij}} = \begin{array}{c|c} \frac{\partial E}{\partial k_{11}} & \frac{\partial E}{\partial k_{22}} \\ \hline \frac{\partial E}{\partial k_{21}} & \frac{\partial E}{\partial k_{22}} \end{array}$$

$$\frac{\partial E}{\partial k_{11}} = \frac{\partial E}{\partial y_{11}} x_{11} + \frac{\partial E}{\partial y_{12}} x_{12} + \frac{\partial E}{\partial y_{21}} x_{21} + \frac{\partial E}{\partial y_{22}} x_{22}$$

Table 3: Dimensions of Convolution Layers for Backward

Layer	$\frac{\partial E}{\partial K_{ij}}$	$\frac{\partial E}{\partial Bi}$	$\frac{\partial E}{\partial Xj}$
Convolution Layer 1	(4x1x5x5)	(4x24x24)	(1x28x28)
Convolution Layer 2	(8x4x5x5)	(8x8x8)	(4x12x12)

1.2 Pooling Layer

In pooling layer, maximum values are returned while pooling window is sliding on input matrix with stride of 2. Locations of the maximum values are also cached to use them in backward pass. Following equation means that it takes matrix called I_{slide} then returns maximum value of it. In the coding part, I basically did this but pooling window is slide over input and new output matrix is formed. Following two matrices are summarized this operation. Since writing a formal mathematical equation for pooling is not easy, I choose to explain it with an example.

$$Y = Max(I_{slide}) \tag{5}$$

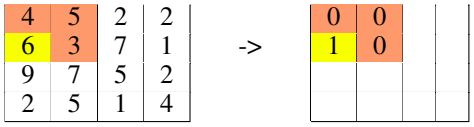

Table 4: Dimensions of Pooling Layers for Forward

Layer	**X**	**K**	**Y**
Pooling Layer 1	(4x24x24)	(1x2x2x2)	(4x24x24)
Pooling Layer 2	(8x8x8)	(1x8x2x2)	(8x4x4)

For the backward pass of pooling layer, there is nothing to update. Instead, output gradient is calculated to pass previous layer by element-wise product of cache come from forward pass and gradient came from next layer. Sliding operation is again in action.

$$\frac{\partial E}{\partial X} = Cache * \frac{\partial E}{\partial Y} \tag{6}$$

Table 5: Dimensions of Pooling Layers for Backward

Layer	$\frac{\partial E}{\partial X}$	**Cache**	$\frac{\partial E}{\partial Y}$
Pooling Layer 1	(4x24x24)	(4x24x24)	(4x12x12)
Pooling Layer 2	(8x8x8)	(8x8x8)	(8x4x4)

1.3 Fully Connected Layer

Forward pass equation for the fully connected layer is the following.

$$Y = W \times X + B \tag{7}$$

where,
Y : Output Matrix
W : Weight Matrix
X : Input Matrix
B : Bias Matrix

Table 6: Dimensions of Fully Connected Layers for Forward

Layer	X	W	B	Y
Fully Connected Layer 1	(128x1)	(128x128)	(128x1)	(128x1)
Fully Connected Layer 2	(128x1)	(10x128)	(10x1)	(10x1)

Backward propagation equations for the fully connected layer is the followings. Calculated parameters in equations 8 and 9 is used in updating state, on the other hand, parameter in equation in the 10 is passed to previous layer for further backward calculations.

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial Y} . X^T \tag{8}$$

$$\frac{\partial E}{\partial B} = \frac{\partial E}{\partial Y} \tag{9}$$

$$\frac{\partial E}{\partial X} = W^T . \frac{\partial E}{\partial Y} \tag{10}$$

where,
$\frac{\partial E}{\partial W}$: Weight Gradient Matrix
$\frac{\partial E}{\partial B}$: Bias Gradient Matrix
$\frac{\partial E}{\partial X}$: Input Gradient Matrix
$\frac{\partial E}{\partial Y}$: Output Gradient Matrix

Table 7: Dimensions of Fully Connected Layers for Backward

Layer	$\frac{\partial E}{\partial X}$	$\frac{\partial E}{\partial W}$	$\frac{\partial E}{\partial B}$	$\frac{\partial E}{\partial Y}$
Fully Connected Layer 1	(128x1)	(128x128)	(128x1)	(128x1)
Fully Connected Layer 2	(128x1)	(10x128)	(10x1)	(10x1)

1.4 Activation Layers

Dimensions are not important for activation layer since they are applied to matrices element-wise.

1.4.1 Rectified Linear Unit (ReLU)

Forward pass equation for ReLU is the following.

$$y(x) = max(0, x) \tag{11}$$

Backward pass equation for ReLU is the following.

$$\frac{dy}{dx} = \begin{cases} 0 & x < 0 \\ 1 & x > 0 \end{cases} \tag{12}$$

1.4.2 Linear

Forward pass equation for Linear function is the following.

$$y(x) = x \tag{13}$$

Backward pass equation for Linear function is the following.

$$\frac{dy}{dx} = 1 \tag{14}$$

1.4.3 Softmax and Cross Entropy Loss Function)

Softmax activation function and cross entropy loss function are implemented in one layer according to my implementation. After fully connected layer 2 matrix that is sized 10x1 come to this layer and outputs loss for forward pass. Backward pass starts from loss and outputs again 10x1 matrix.

For the forward pass, softmax function and cross entropy function are the followings, respectively.

$$\hat{y}_i = \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}} \tag{15}$$

where,
$\hat{y}_i$: Prediction value
x_i : Input value

$$L = \sum_{i=1}^{n} y_i \ln \hat{y}_i \tag{16}$$

where,
y_i : Actual Value
L : Loss

For the backward pass, there is combined form of softmax and cross entropy which is very elegant and small equation. However, I learned the derivation of this function which is referenced in the references.

$$\frac{\partial L}{\partial x_j} = \hat{y}_i - y_i \tag{17}$$

1.5 Conclusion of Question 1: The problems I faced, how I solved and could not solve them

First of all, this was my first project in both machine learning and deep learning area. When I started the project, I hadn't used any library in this area even numpy. Therefore, during making project, I learned a lot of things from using libraries to implementing CNN from scratch.

While implementing it, I wrote every layer separately and checked them for different matrices by using unit tests. I tracked every operation's inputs and outputs and compared them with handwritten calculations. After making sure every operation is working correctly, I carry one of the image data from the first layer to loss then carried back from loss to first layer again in debugging mode pf PyCharm. This two approach made debugging possible for me for this big structure.

After being sure every layer works properly, I started training with smaller data set which is 100 images from every class. However, another problem is occurred which is exploding gradient. I found out that this was happening because the exponential in the softmax layer. Firstly, I substracted maximum value from both nominator and denominator to decrease degree of exponent.

$$\hat{y}_i = \frac{e^{x_i - max(x_i)}}{\sum_{k=1}^{n} e^{x_k - max(x_i)}} \tag{18}$$

However, this didn't solve the problem again. Then, I tried to train shallower network. This worked but unfortunately, I couldn't train correct sized network for this homework. There is only one convolution layer, pooling layer and two fully connected layer. I submitted this homework in this form because I want to show layers actually work. The accuracy I get for 1000 (100 from every class) data points and 50 epochs is 0.686.

Lastly, I want to talk about what I could do to solve this problem. I want to propose two different approach. First one is gradient clipping. This is simply putting threshold for gradients to prevent exploding. Another approach could be mini batch gradient descent. However, although I tried both of them, I couldn't make them work.

2 Decision Boundary of a Neural Network with ReLU

Yes, it is possible to obtain non linear decision boundary with ReLU. To show that, I want create an example and then generalize it. ReLU is a function that returns 0 for negative values and itself for positive values.

$$ReLU(x) = max(0, x) \tag{19}$$

$$ReLU(x) = 0, \quad for \quad x \leq 0 \tag{20}$$

The ReLU function can be shifted as in the following equation.

$$ReLU(x - c) = 0, \quad for \quad x \leq c \tag{21}$$

Any function can be added to equation 21 from both sides which does not change anything.

$$g(x) + ReLU(x - c) = g(x), \quad for \quad x \leq c \tag{22}$$

We can abuse these property to obtain nonlinear decision boundaries. Also, by multiplying functions with different values we can bend the function at different boundaries as in the following equation.

$$f(x) = ReLU(x) + (-3)ReLU(x - 2) + 5ReLU(x - 4) + (-6)ReLU(x - 6) \tag{23}$$

For example previous equation can be written as this.

$$\begin{gathered} f(x) = x, \quad for \quad 0 \leq x < 2 \\ f(x) = x - 3x, \quad for \quad 2 \leq x < 4 \\ f(x) = x - 3x + 5x, \quad for \quad 4 \leq x < 6 \\ f(x) = x - 3x + 5x - 6x, \quad for \quad 6 \leq x \end{gathered} \tag{24}$$

As it can be seen from the example, by adding different ReLU functions to each other and multiplying them with different values we can bend them as much as it is needed. Lastly, by adding constant value to the function, it can be translated everywhere vertically. Now, last step is generalizing this approach.

$$f(x) = b + \sum_{i=1}^{n} c_i ReLU(x - i) \tag{25}$$

3 References

1. Lopin M. Why is ReLU non-linear? Medium. link

2. Ye A. Finally, an intuitive explanation of why ReLU works. Towards Data Science. link

3. Zhang S. Simple MNIST NN from scratch (numpy, no TF/Keras). Towards Data Science. link

4. Roelants P. Softmax classification with cross-entropy. Github.io. link

5. Sharma S. A Visual Introduction to Neural Networks. Towards Data Science. link

6. Does it make sense to use Relu activation on the output neuron for binary classification? If not, why? Quora. link

7. Backpropagation in a convolutional layer. Towards Data Science. link