Assignment 4

Danilo Vladicic - 250861339 - dvladici@uwo.ca

April 4, 2020

Question 1: Convolutional layer

Let $x = x_{in}$ of size Nx1
Let $\hat{x} = x_{out}$ of size Nx1
Let W be the weight kernel of size Kx1
Let $f(x, w)$ denote the operation at the layer

Part A: Forward Propagation

Combining the above notation the convolutional layer operation is

$$\hat{x} = f(x, w) = x \star w$$

We note that both x and $\hat{x}$ have the same dimension N. A convolution operation of size K on size N without any padding results in the output of size (N - K + 1)x1. The simplest solution to maintain the dimensionality through the operation is to pad the input with $\frac{k-1}{2}$ zeros. Suppose $K = 3$, the output of the top 3 neurons would look as so

$$\hat{x}_1 = w_1 * 0 + w_2 * x_1 + w_3 * x_2$$

$$\hat{x}_2 = w_1 * x_1 + w_2 * x_2 + w_3 * x_3$$

$$\hat{x}_3 = w_1 * x_2 + w_2 * x_3 + w_3 * x_4$$

I notice that the output of the convolution layer is the same as multiplying a (NxN) weights matrix **W** (K=3).

$$\begin{bmatrix} 0 & w_2 & w_3 & 0 & \dots & 0 & 0 \\ w_1 & w_2 & w_3 & 0 & \dots & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 & \dots & 0 \\ \vdots & 0 & \dots & 0 & w_1 & w_2 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \\ \vdots \\ \hat{x}_N \end{bmatrix} \tag{1}$$

Then for this layer, the forward propagation equation can be written an

$$f(x, w) = w \star x = \mathbf{w}x$$

Part B: Backward Propagation

This layer recieves $\frac{\delta C}{\delta \hat{x}}$ as input during back propagation. If we suppose that the $C(\hat{x}_i, y_i)$ is scalar for training instance (x_i, y_i), then since $\hat{x}$ is a (Nx1) vector,

$$\frac{\delta C}{\delta \hat{x}} = \begin{bmatrix} \frac{\delta C}{\delta \hat{x}_1} & \dots & \frac{\delta C}{\delta \hat{x}_N} \end{bmatrix}$$

is a vector of size 1xN.
This layer then needs to compute the following
$\frac{\delta C}{\delta x}$ as its output and $\frac{\delta C}{\delta w}$ to update weights.

For $\frac{\delta C}{\delta x}$:

$$\frac{\delta C}{\delta x} = \frac{\delta C}{\delta \hat{x}} \frac{\delta \hat{x}}{\delta x}$$

Since both x and $\hat{x}$ are (Nx1) vectors, their gradient will be a (NxN) matrix as so,

$$\frac{\delta \hat{x}}{\delta x} = \begin{bmatrix} \frac{\delta \hat{x}_1}{\delta x_1} & \frac{\delta \hat{x}_1}{\delta x_2} & \cdots & \frac{\delta \hat{x}_1}{\delta x_N} \\ \frac{\delta \hat{x}_2}{\delta x_1} & \frac{\delta \hat{x}_2}{\delta x_2} & \cdots & \frac{\delta \hat{x}_2}{\delta x_N} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\delta \hat{x}_N}{\delta x_1} & \frac{\delta \hat{x}_N}{\delta x_2} & \cdots & \frac{\delta \hat{x}_N}{\delta x_N} \end{bmatrix}$$

The output of $\frac{\delta C}{\delta x}$ will be a (1xN)x(NxN) = (1xN) vector. This output of the i-th item can be written as

$$(\frac{\delta C}{\delta x})_i = \Sigma_{k=1}^{N} (\frac{\delta C}{\delta \hat{x}_k} \frac{\delta \hat{x}_k}{\delta x_i})$$

The total output would be

$$\frac{\delta C}{\delta x} = \begin{bmatrix} \Sigma_{k=1}^{N} (\frac{\delta C}{\delta \hat{x}_k} \frac{\delta \hat{x}_k}{\delta x_1}) & \cdots & \Sigma_{k=1}^{N} (\frac{\delta C}{\delta \hat{x}_k} \frac{\delta \hat{x}_k}{\delta x_N}) \end{bmatrix}$$

which is propagated to the layer behind this one.

For $\frac{\delta C}{\delta w}$: This value will be used in the SGD $w_2 = w_1 + n(\frac{\delta J}{\delta w})$. Assuming this is the first training instance, then $\frac{\delta J}{\delta w} = \frac{\delta C}{\delta w}$. To calculate,

$$\frac{\delta C}{\delta w} = \frac{\delta C}{\delta \hat{x}} \frac{\delta \hat{x}}{\delta x} \frac{\delta x}{\delta w}$$

The first two terms result in the (1xN) vector found above to be propagated backwards. So,

$$\frac{\delta x}{\delta w} = \begin{bmatrix} \frac{\delta x_1}{\delta w_1} & \frac{\delta x_1}{\delta w_2} & \cdots & \frac{\delta x_1}{\delta w_k} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\delta x_N}{\delta w_1} & \frac{\delta x_N}{\delta w_2} & \cdots & \frac{\delta x_N}{\delta w_k} \end{bmatrix}$$

which is a (NxK) matrix. Multiplying this and the (1xN) matrix results in (1xN)x(NxK) = (1xK). Similarly to above

$$(\frac{\delta C}{\delta w})_i = \Sigma_{j=1}^{N} (\Sigma_{k=1}^{N} (\frac{\delta C}{\delta \hat{x}_k} \frac{\delta \hat{x}_k}{\delta x_j})) \frac{\delta x_j}{w_i}$$

The update after the first training instance will be

$$w_2 = \begin{bmatrix} w_1 \\ \vdots \\ w_k \end{bmatrix} + \begin{bmatrix} n\frac{\delta C}{\delta w_1} \\ \vdots \\ n\frac{\delta C}{\delta w_N} \end{bmatrix}$$

Part C: Padding Discussion

For handling the boundries, I padded the input with $\frac{K-1}{2}$ zeros. The reason for this was that the output and input size had the same dimensionality. The underlying reason for doing any padding however, is for information preservation/expression in the output. By padding the input on both sides with zeros, then the values of the input at the boundries gets affected by more weights in the kernel. If there is no padding, the first input that provides to some outputs relative to each variable in the kernel would occur for the input that is at position K (size of kernel). For the first output it contributes the product $x_k * w_k$, for the second output it contributes $x_k * w_{k-1}$ and so on until the (k-1) output where it contributes $x_k * w_1$. This is the first value then that is affected by all the variables. With padding this occurs earlier, and so the data near the boundries contributes much more to the output than just those values in the middle. This helps in perserving information along the boundries as these values will contribute to more weights in the weight updates, where before they only could have contributed to some smaller subset of the weights.

Question 2: Pooling Layer

The pooling discussed in this question is a max pooling operator with stride 1.

Part A: Forward/Backward Propagation

Let $x = x_{in}$ with size N
Let $\hat{x} = x_{out}$ with size Z
Let $f(x) = \hat{x}$ denote the pooling operation For Forward Pass:
During the forward pass, consider what the output would be for a max pooling operator of size 2 with stride 1. Then

$$\hat{x}_1 = max(x_1, x_2)$$

$$\hat{x}_2 = max(x_2, x_3)$$

and so on. The equation for forward propagation can be generalized to any stride and size.
Let S denote the stride of the pooling operator
Let K denote the size of the pooling operator
Then,

$$\hat{x} = f(x) = \{max_{0 \leq j \leq k-1}(x_{i+j})\}_{i=1,\, i+=S}^{N-K+1}$$

Where

$$max_{0 \leq j \leq k-1}(x_{i+j}) = max(x_i, x_{i+1}, \ldots, x_{i+k-1})$$

For Backward Pass:
This layer recieves $\frac{\delta C}{\delta \hat{x}}$ which will be a (1xK) vector assuming that $C(x, y)$ is scalar. This layer only needs to compute $\frac{\delta C}{\delta x}$ to propagate back, as pooling layers have no learned parameters.

$$\frac{\delta C}{\delta x} = \frac{\delta C}{\delta \hat{x}} \frac{\delta \hat{x}}{\delta x}$$

Since both $\hat{x}$ and x are vectors, then

$$\frac{\delta \hat{x}}{\delta x} = \begin{bmatrix} \frac{\delta \hat{x}_1}{\delta x_1} & \cdots & \frac{\delta \hat{x}_1}{\delta x_N} \\ \vdots & \vdots & \vdots \\ \frac{\delta \hat{x}_K}{\delta x_1} & \cdots & \frac{\delta \hat{x}_K}{\delta x_N} \end{bmatrix}$$

is a (KxN) matrix. The multiplacation of (1xK)x(KxN) will result in a vector (1xN) that is propagated backwards. Consider the output $\hat{x}_1$ with a pooling operator of size 2. Then either $\frac{\delta \hat{x}_1}{\delta x_1} = 1$ when $x_1 > x_2$ and 0 otherwise. If this is the case the column of $\frac{\delta \hat{x}_i}{\delta x_1}$ will have 0's whenever $\hat{x} \neq x$. Then this equation can be formulated as

$$(\frac{\delta C}{\delta x})_i = \begin{cases} 1 & x_i > x_{i+1} > ... > x_{i+K-1} \\ 0 & otherwise \end{cases}$$

Then in the output of this layer, only K of the N values will have 1's with 0's in between those values.

Part B: Padding Discussion

The purpose of pooling is to down sample feature maps and highlight the most expressive features in the patch of size K. Thus adding padding to this layers boundries, wouldn't improve feature detection, since regardless of the pooling operator (max or min), the padding will always be in the output, and doesn't provide valuable information about a specific reason. The only time padding might be done in a pooling layer is when the pooling size and stride don't perfectly fit in the input size. Suppose you have a max pooling operator with a size of 2 and a stride of 2. Suppose that the input x has an odd number of values (1-D). Then padding may be done on the endpoint so that the pooling operator fits nicely on the input.

Question 3: AlexNet

The features of last layer of the AlexNet:
Last layer has 4096000 input features and 1000 output features
The image we are classifying is:

The top 5 predictions from a randomly initialized AlexNet are:
Prediction: wombat Percentage 0.10257013142108917
Prediction: mud turtle Percentage 0.10256970673799515
Prediction: lifeboat Percentage 0.10254091024398804
Prediction: macaw Percentage 0.10235338658094406
Prediction: corkscrew, bottle screw Percentage 0.10224508494138718
Which if run multiple times, the predictions will change but the confidence levels are usually always around the same mark.

Top 5 predictions from a AlexNet pretrained on Imagenet data:
Prediction: Pembroke, Pembroke Welsh corgi Percentage 89.18708801269531
Prediction: Cardigan, Cardigan Welsh corgi Percentage 10.649338722229004
Prediction: kelpie Percentage 0.04727327823638916
Prediction: Eskimo dog, husky Percentage 0.04288318008184433
Prediction: Shetland sheepdog, Shetland sheep dog, Shetland Percentage 0.017419634386897087
Which correctly identified the dog! Its good to note that although the AI is pretty confident its a corgi, its even more confident that the image is a dog, as all the top 5 predictions are various types of dogs!.