Jasleen Kaur

Assignment 4

Report

Problem1: Convolutional Layer

Convolutional layer is the most popular module in computer vision tasks. In this question, you will drive the equations for its forward and backward equations.

a) Consider your input x_{in} and output x_{out} are both 1-D signals with the same dimension N, and your kernel W has size k. Find the equation for forward propagation.

AI-4

Problem 1.

(a) Input : x_{in} output x_{out}

kernel → W

A) For convolutional neural network input for node j in layer l is sum of activated outputs from previous $l-1$ layers

$$x_{in_j}^{l} = b_j^{l} + \sum_{k=0}^{N-1} \text{conv1D}\left(w_{jk}^{l}, x_{out_k}^{l-1}\right)$$

Activation output x_{out} for activation function g^L for node j, layer l is

$$x_{out}^{l} = g^{l}\left(x_{in}^{l}\right)$$ for forward pass

b) Consider the back-propagation process, with learning rate η, and the gradients from the last layer is $\frac{\partial C}{\partial x_{out}}$. Find the gradients of the input $\frac{\partial C}{\partial x_{in}}$, and the update rule for the kernel weights W.

(b) learning rate - η

Gradient from last layer $= \dfrac{\partial C}{\partial x_{out}}$

Loss is given as

$$C = \sum_{j=0}^{N-1} (x_{out} - y_j)^2 \text{, for node } j$$

∴ Loss for single node j in output layer l

is given as $C_{0j} = (x_{out}^{l} - y_j)^2$

C_{0j} is function of activation output of node j in layer l

$$C_{0j}(x_{out}^{l})$$

From previous derivation $x_{out} = g^{l}(x_{in}^{l})$

$$\therefore C_{0j} = C_{0j}(x_{out}^{l}(x_{in}^{l}(w_i^{l})))$$

$$C = \sum_{j=0}^{N-1} C_{0j}$$

$$\frac{\partial C}{\partial x_{in}} = \left(\frac{\partial C}{\partial x_{out}^{l}}\right)\left(\frac{\partial x_{out}^{l}}{\partial x_{in}^{l}}\right)$$

$$= \frac{\partial}{\partial x_{out}}\left(\left(\sum_{j=0}^{n-1}(x_{out\,j}^{l} - y_j)^2\right)\right.$$

$$= \frac{\partial}{\partial x_{out}}\left((x_{out}^{l}-y_0)^2 + (x_1^{l}-y_1)^2 + \dots \text{ losses}\right)$$

$$\frac{\partial C}{\partial x_{out}} = 2(x_{out}^{l} - y_1)$$

$$\frac{\partial x_{out}^{l}}{\partial x_{in}^{l}} = \frac{\partial}{\partial x_1}(g^{l}(x_{in}^{l}))$$

$$= g^{\prime(l)}(x_1^{(l)})$$

$$\therefore \frac{\partial C}{\partial x_{in}} = 2(x_{out}^{l} - y_1)(g^{\prime l}(x_{in}^{l}))$$

$$= \frac{\partial C}{\partial x_{out}}(g^{\prime l}(x_{in}^{l}))$$

Kernel weights are updated as W_{l+1}

$$W_{ik}^{l+1} = W_{ik}^{l-1} - \eta \frac{\partial C}{\partial W_{ik}^{l-1}}$$

and $$\frac{\partial C}{\partial W} = 2(x_{out}^{l} - y_n)\left(g'^{(l)}(x_{in}^{l})\right)\frac{\partial x_{in}}{\partial W}$$

$$= \left(\frac{\partial C}{\partial x_{out}}\right)\left(\frac{\partial x_{out}}{\partial x_{in}}\right)\left(\frac{\partial x_{in}}{\partial W}\right)$$

c) Discuss how you handle the boundaries and explain your choice.

Answer: To handle the boundaries, we can use padding where we add zero to the left or right of input and by default the value is 1. The input must be padded with zeros to resulting in output has the same length as the original input .In this case for 1-D signals, we can use 1-D zero padding around the input to get a larger output or tensor.

Problem2: Pooling Layers

Pooling layer is a popular layer without trainable parameters. In this question, the pooling is a max pooling operator with stride 1.

a) Consider your input x_{in} and output x_{out} are 1-D signals with the different size. Please find the equations or pseudo code for its forward and backward propagations.

// Pseudo code for forward propagation
input : x¬_input, pool¬_window
batch size, channels, rows and column for in
batch_size, channels, row, col = x_in
#pooling window
pool_ht = pool_window(height)
pool_wth = pool_window(width)
pool_len = pool_window(length)
# Evaluate Kernel dimension
Kernel_height = 1 + (row – pool_ht)/pool_len
Kernel_width = 1 + (column – pool_wth)/pool_len
# output
For each element in batch_size:
# max element in the window
Output[] = max(x_in)
Save(x_in,pool_window)
return output

//Pseudo code for backward propagation
input : derivative_x_out, saved_fw_propagation
Get saved values from forward propagation
x_in, pool_window = saved_values()
rows and column for input
row, col = x_in
#pooling window
pool_ht = pool_window(height)
pool_wth = pool_window(width)
pool_len = pool_window(length)
# batch size, channels and kernel dimension
batch_size, channels, Kernel_height, Kernel_width = shape(derivative_x_out)
Calculate derivative of x_in using mask * derivative_x_out
For each element in batch_size:
# window
Pool_xinput = x(params)
# mask

```
Mask = get_mask(Pool_xinput)
# mask * derivative_x_out
derivative_xin = Mask * derivative_x_out
return derivative_xin
```

b) Discuss how you handle the boundaries and explain your choice.

Answer: The pooling layer (POOL) is a down sampling operation, typically applied after a convolution layer, which does some spatial invariance. Padding can be used here to process the input into a size that fits the kernel dimension being used and the length of pooling window of sliding size. If the kernel size doesn't capture any information like of size 1*1, we would not use any padding to handle the boundary. Though bigger size kernel would require some padding to fit kernel size.