TTIC 31230 Fundamentals of Deep Learning
Problem set 3
Due Thursday 11:59 pm, January 26

- Zip all your ipynb&pdf file with name PS3_yourfullname to: ttic.dl.win.2017@gmail.com.
- Late Submission: submitting late work will be penalized 10% per day, maximum three days delay allowed, no submission allowed after that.

This problem sets involves writing convolution and maxpool layers in EDF. There are no analytical problems — just the coding. Your solutions should be entered into the appropriate cells of the jupyter notebook provided and you should receive an accuracy of around 40% on CIFAR 10.

Part 1. We want you to write a class MaxPool such that for

$$y = \mathrm{MaxPool}(x, s)$$

the instance y is specified as follows.

Here x.value is the input image and is assumed to have shape (B, H, W, C) where B is the batch size, H is the image height, W is the image width, and C is the number of channels at each pixel position. The stride s is assumed to be an integer.

The shape of y.value should be

$$(B, \lfloor H/s \rfloor,\ \lfloor W/s \rfloor, C).$$

We want y.forward() to set y.value to the following tensor where u and v range over 0 to $s - 1$.

$$y.\mathrm{value}[b, i, j, c] = \max_{u,v}\ x.\mathrm{value}[b, si + u, sj + v, c]$$

If the maximum is tied at several spatial locations you can assume that the gradient with respect to each such maximal value equals the corresponding pooled value of y.grad. This assumption simplifies the backward method.

Part 2. This is the same as part 1 except that you are to build AvePool such that for

$$y = \mathrm{AvePool}(x, s)$$

the instance y is specified as follows.

$$y.\mathrm{value}[b, i, j, c] = \frac{1}{s^2} \sum_{u,v}\ x.\mathrm{value}[b, si + u, sj + v, c]$$

Part 3. We want you to write a class Conv such that for

$$y = Conv(x, f, s, p)$$

the instance y is specified as follows.

Here x.value is the input image and is assumed to have shape (B, H, W, C) where B is the batch size, H is the image height, W is the image width, and C is the number of channels at each pixel position. Here f is assumed to be a square filter parameter with shape (K, K, C, C') where K is the spatial dimension of the filter. There is no batch dimension for parameters. The stride s and the padding p are assumed to be integers.

In the forward method one must first pad the x.value with padding width p. Let x' be the result of padding. x' should have shape $(B, H+2p, W+2p, C)$ where we have

$$x'[:, p : p+H, p : p+W, :] = x.\text{value}$$

and all other values of x' are zero.

For a square filter $K \times K$ filter the shape of y.value should be

$$(B, \lfloor (H+2p-K)/s \rfloor + 1, \lfloor (W+2p-K)/s \rfloor + 1, C').$$

We want y.forward() to set y.value to the following tensor where u and v range over 0 to $K-1$ (the spatial coordinates of the filter).

$$y.\text{value}[b, i, j, c'] = \left(\sum_{u,v,c} x'[b, si+u, sj+v, c] \; f.\text{value}[u, v, c, c'] \right)$$

Hint: you can write a Python loop over i and j (but not b) and for i, j fixed use np.matmul to do the summation over u, v and c.

In this problem set each batch component of ℓ.grad is initialized to $1/B$ where B is the batch size rather than being initialized to 1. Each backward method adding to the gradient of a paramter then sums over the batch rather than average.

Part 4. Once you have constructed the Conv and MaxPool layers construct and test the following model.

1. A Conv Layer with a 3×3 filter mapping 3 color channels to 32 feature channels and with stride 1, padding 1, and a ReLU activation function resulting in a 32×32 image.

2. A MaxPool layer with stride 4 resulting in image dimensions 8×8 with 32 channels.

3. A Conv layer with a 3×3 filter mapping 32 channels to 64 channels and with padding 0 and stride 1 and a ReLU activation function resulting in a 6×6 image with 64 channels.

4. An AvePool layer with stride 6 resulting in a 1×1 image with 64 channels.

5. A Conv layer on the 1×1 image with a 1×1 filter mapping 64 channels to 10 channels with a ReLu nonlinearity. This is functionally equivalent to a perceptron layer but you can use your implementation of the Conv layer.

6. Reshape the 1×1 image with 10 channels to a 10 dimensional vector using the Reshape class in edf.py.

7. Softmax, LogLoss etc, which are standard in classification task.

Filter parameters should be constructed using edf.Param(edf.xavier(shape)) which provides a well-initialized random value for a filter tensor with the given shape. After Convolution, you should add a bias vector β using edf.Add before each ReLU activation. Bias vector parameters should be initialized to zero. For

$$z = \text{Add}(y, \beta)$$

with y haveing shape (B, H, W, C) we have

$$z.\text{value}[b, i, j, c] = y.\text{value}[b, i, j, c] + \beta[c]$$

The notebook provides softmax and logloss layers to define the loss and code to run the model.

Partial credit will be give for code that is “almost right”.