Weekly Report()

Liu Junnan

Abstract

This week I finished assignment2 including the implementation of convolutional layer and {batch, layer, spatial batch, group}normalization layers, plus pytorch version of convnets.

1 Work Done

1.1 Convolutional Neural Network - Implementation

1.1.1 Conv Layer

Recall that a convolutional layer consists of several filters that convolves input images and produces a feature map.

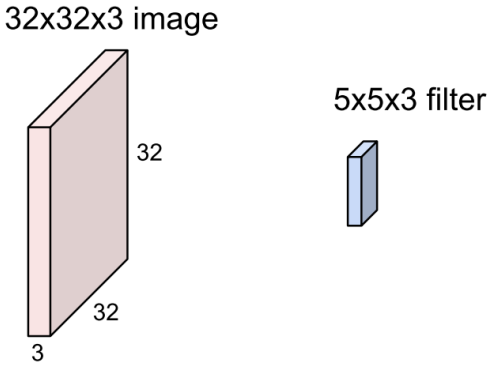

Figure 1: Convolutional Layer Example

Specifically, x is input data of shape (N, C, H, W), where N is the mini-batch size, C number of channels, H and W are height and width of each channel; w is filter weights of shape (F, C, HH, WW) where F is the number of filters, HH and WW are height and width of a filter. Then we can compute the shape of output by the formulation mentioned in the previous report:

$$\begin{aligned} H' &= \frac{1 + (H + 2 * \text{pad} - HH)}{\text{stride}} \\ W' &= \frac{1 + (W + 2 * \text{pad} - WW)}{\text{stride}} \end{aligned}$$

The naive version of forward pass of conv layer with 4 loops is as follows:

```
Ho = 1 + (H + 2 * pad - HH) // stride
```

```
Wo = 1 + (W + 2 * pad - WW) // stride

x_pad = np.pad(x, ((0, 0), (0, 0), (pad, pad), (pad,pad)), 'constant')
out = np.zeros((N, F, Ho, Wo))

for n in range(N):
   for i in range(Ho):
      for j in range(Wo):
         for k in range(F):
            out[n, k, i, j] = np.sum(
            x_pad[n, :, i*stride:i*stride+HH, j*stride:j*stride+WW] * w[k
                ]) + b[k]
```

It computes element-wise multiplication of blocks of input data and filter matrices over channels and batch samples.

A simple modification could help the code vectorized

```
for i in range(Ho):
   for j in range(Wo):
      x_pad_masked = x_pad[:, :, i*stride: i*stride+HH, j*stride: j*
          stride+WW]
      mul = x_pad_masked.reshape((N, -1, C, HH, WW)) * w.reshape((-1, F,
          C, HH, WW))
      out[:, :, i, j] = np.sum(mul, axis=(2, 3, 4))
out += b.reshape((1, F, 1, 1))
```

However, the backward pass of conv layer is quite tricky and it is not detailed in the cs231 course. This part is based on a blog on the Internet since it's quite mathematically difficult to derive. Recall that the forward activation of a convolutional layer is computed by

$$a^l = \sigma(z^l) = \sigma(a^{l-1} * W^l + b^l)$$

where the superscripts mean the variables in the l-th layer respectively, and σ denotes the activation function, z the input and a the activation/output value.

Then we can compute the gradient of l-th layer by

$$\delta^{l-1} = \frac{\partial J(W,b)}{\partial z^{l-1}} = \frac{\partial J(W,b)}{\partial z^l}\frac{\partial z^l}{\partial z^{l-1}} = \delta^l \frac{\partial z^l}{\partial z^{l-1}}$$

$$z^l = a^{l-1} * W^l + b^l \tag{1}$$

We can simplify the setting that, say, a is a 3×3 matrix, w is a 2×2 matrix, and b is a zero matrix. Then eqn1 can be written as

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} * \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} = \begin{pmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{pmatrix} \tag{2}$$

Stretch the convolution in eqn2 as

$$z_{11} = a_{11}w_{11} + a_{12}w_{12} + a_{21}w_{21} + a_{22}w_{22} \tag{3}$$
$$z_{12} = a_{12}w_{11} + a_{13}w_{12} + a_{22}w_{21} + a_{23}w_{22} \tag{4}$$
$$z_{21} = a_{21}w_{11} + a_{22}w_{12} + a_{31}w_{21} + a_{32}w_{22} \tag{5}$$
$$z_{22} = a_{22}w_{11} + a_{23}w_{12} + a_{32}w_{21} + a_{33}w_{22} \tag{6}$$

The gradient with respect to a^{l-1} is

$$\nabla a^{l-1} = \frac{\partial J(W,b)}{\partial a^{l-1}} = \frac{\partial J(W,b)}{\partial z^l}\frac{\partial z^l}{\partial a^{l-1}} = \delta^l \frac{\partial z^l}{\partial a^{l-1}}$$

Then we can compute $\frac{\partial z^l}{\partial a^{l-1}}$ element-wise:

$$\begin{aligned}
\nabla a_{11} &= \delta_{11}w_{11} \\
\nabla a_{11} &= \delta_{11}w_{11} \\
\nabla a_{11} &= \delta_{11}w_{11} \\
\nabla a_{21} &= \delta_{11}w_{21} + \delta_{21}w_{11} \\
\nabla a_{22} &= \delta_{11}w_{22} + \delta_{12}w_{21} + \delta_{21}w_{12} + \delta_{22}w_{11} \\
\nabla a_{23} &= \delta_{12}w_{22} + \delta_{22}w_{12} \\
\nabla a_{31} &= \delta_{21}w_{21} \\
\nabla a_{32} &= \delta_{21}w_{22} + \delta_{22}w_{21} \\
\nabla a_{33} &= \delta_{22}w_{22}
\end{aligned}$$

Actually these equations can be represented as a convolution operation:

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \delta_{11} & \delta_{12} & 0 \\ 0 & \delta_{21} & \delta_{22} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} w_{22} & w_{21} \\ w_{12} & w_{11} \end{pmatrix} = \begin{pmatrix} \nabla a_{11} & \nabla a_{12} & \nabla a_{13} \\ \nabla a_{21} & \nabla a_{22} & \nabla a_{23} \\ \nabla a_{31} & \nabla a_{32} & \nabla a_{33} \end{pmatrix}$$

Notice that $\begin{pmatrix} w_{22} & w_{21} \\ w_{12} & w_{11} \end{pmatrix} = rot180(W)$

In summary, $\nabla a^{l-1} = \sigma^l * rot180(W^l)$

Similarly,

$$\nabla W^l = \frac{\partial J(W,b)}{\partial z^l}\frac{\partial z^l}{\partial W^l} = \delta^l * rot180(a^{l-1})$$

and

$$\frac{\partial J(W,b)}{\partial b^l} = \sum_{u,v}(\delta^l)_{u,v}$$

The formulations above still seem difficult to understand. But the intuition is that because of the weight sharing mechanism of convolution networks, each block of input data(of shape (HH, WW), i.e. the filter size) connects to a "replica" of the same filter and do element-wise multiplication in the forward pass. So in backward pass, you only have to compute the individual gradient for each block and replica, and sum over the gradients of replicas as the final gradient with respect to that filter.

The code is as follows:

```
padding = [(0,0), (0, 0), (pad, pad), (pad, pad)]
x_pad = np.pad(x, padding, 'constant')
dout_pad = np.pad(dout, padding, 'constant')

db = np.sum(dout, axis=(0,2,3))
for i in range(Ho):
   for j in range(Wo):
      x_pad_masked = x_pad[:, :, i*stride:i*stride+HH, j*stride:j*stride+
          WW]
      for k in range(F):
         dw[k] += np.sum(x_pad_masked * (dout[:, k, i, j])[:, None, None,
              None], axis=0)
      for n in range(N):
         dx_pad[n, :, i*stride:i*stride+HH, j*stride:j*stride+WW] +=\
            np.sum(w*(dout[n, :, i, j])[:, None, None, None], axis=0)

dx = dx_pad[:, :, pad:-pad, pad:-pad]
```

We do padding is due to the shape issue. In each element of the feature map/output(code line 6-8) we extract the responsible block of input x, and use it to compute gradient. Line 15 removes the padded area of dx.

Again, backpropagation of conv layer is quite tricky. Even though I referred to many blogs and materials, I still don't think I fully understand the math behind it. But at least I can implement the code if the formulations are given.

1.1.2 Normalization Layers - Implementation

I have already finished the naive version of batch normalization layer last week, but still remains improvement to do. Besides there are still several variants like layer normalization, spatial batch normalization and spatial group normalization.

Recall that batch normalization can be computed by the following equations: Forward:

$$\begin{aligned}
\mu &= \frac{1}{m}\sum_i x_i \\
\sigma^2 &= \frac{1}{m}\sum_i (x_i-\mu)^2 \\
\hat{x_i} &= \frac{x_i-\mu}{\sqrt{\sigma^2+\epsilon}} \\
y_i &= \gamma\hat{x_i}+\beta
\end{aligned}$$

Backward:

$$\begin{aligned}
\nabla\hat{x_i} &= \gamma\cdot\nabla y_i \\
\nabla\sigma^2 &= \sum_j \nabla\hat{x_i}\cdot(x_j-\mu)\cdot\frac{-1}{2}(\sigma^2+\epsilon)^{-\frac{3}{2}} \\
\nabla\mu &= \left(\sum_i \nabla\hat{x_i}\cdot\frac{-1}{\sqrt{\sigma^2+\epsilon}}\right)+\nabla\sigma^2\frac{\sum_i -2(x_i-\mu)}{m} \\
\nabla x_i &= \nabla\hat{x_i}\cdot\frac{1}{\sqrt{\sigma^2+\epsilon}}+\nabla\sigma^2\cdot\frac{2(x_i-\mu)}{m}+\frac{\nabla\mu}{m} \\
\nabla\gamma &= \sum_i \nabla y_i\cdot\hat{x_i} \\
\nabla\beta &= \sum_i \nabla y_i
\end{aligned}$$

Following the above equations step by step we can write the naive version of batch normalization:

```
1 m = x_hat.shape[0]
2
3 dx_hat = dout * gamma
4 dvar_inner = dx_hat * (x - sample_mean) * var_inv**3
5 dvar = -0.5 * np.sum(dvar_inner, axis=0)
6 dmean = -np.sum(dx_hat * var_inv, axis=0) + -2 * dvar * np.mean(x -
      sample_mean, axis=0)
7 dx = dx_hat * var_inv + 2 / m * dvar * (x - sample_mean) + dmean / m
8 dgamma = np.sum(x_hat*dout, axis=0)
9 dbeta = np.sum(dout, axis=0)
```

where variable var means σ^2, and var_inv stands for $\frac{1}{\sqrt{\sigma^2+\epsilon}}$ for simplification.

Actually the computation of intermediate variables $\nabla\mu$ and $\nabla\sigma^2$ is unnecessary and code line 4-7 can be re-written in one line of less than 80 characters if spaces are omitted, which is the requirement of the assignment.

$$\nabla\sigma^2 = \sum_i \nabla\hat{x}_i \cdot (x_i - \mu) \cdot \frac{-1}{2}(\sigma^2+\epsilon)^{-\frac{3}{2}} = -\frac{1}{2} \cdot \sum_i \nabla\hat{x}_i \frac{x_i - \mu}{\sqrt{\sigma^2+\epsilon}} \cdot \frac{1}{\sqrt{\sigma^2+\epsilon}}$$
$$= -\frac{1}{2(\sigma^2+\epsilon)} \sum_i \nabla\hat{x}_i \hat{x}_i$$

$$\nabla\mu = \left(\sum_i \nabla\hat{x}_i \cdot \frac{-1}{\sqrt{\sigma^2+\epsilon}}\right) + \nabla\sigma^2 \frac{\sum_i -2(x_i-\mu)}{m}$$

In the above equation

$$\sum_i x_i - \mu = \sum_i x_i - \sum_i \mu = m * \mu - m * \mu = 0$$

Therefore $\nabla\mu = \frac{-1}{\sqrt{\sigma^2+\epsilon}} \cdot \sum_i \nabla\hat{x}_i$

Assemble them together we can get ∇x_i as:

$$\nabla\hat{x}_i = \left(\nabla\hat{x}_i \cdot \frac{1}{\sqrt{\sigma^2+\epsilon}}\right) + \left(\nabla\sigma^2 \cdot \frac{2(x_i-\mu)}{m}\right) + \left(\frac{\nabla\mu}{m}\right)$$
$$= \left(\nabla\hat{x}_i \cdot \frac{1}{\sqrt{\sigma^2+\epsilon}}\right) - \left(\frac{1}{2(\sigma^2+\epsilon)} \sum_j \nabla\hat{x}_j \hat{x}_j \cdot \frac{2(x_i-\mu)}{m}\right) + \left(\frac{-1}{m\sqrt{\sigma^2+\epsilon}} \cdot \sum_i \nabla\hat{x}_i\right)$$
$$= \left(\nabla\hat{x}_i \cdot \frac{1}{\sqrt{\sigma^2+\epsilon}}\right) - \left(\frac{1}{m\sqrt{\sigma^2+\epsilon}} \cdot \frac{(x_i-\mu)}{\sqrt{\sigma^2+\epsilon}} \cdot \sum_j \nabla\hat{x}_j \hat{x}_j\right) + \frac{-1}{m\sqrt{\sigma^2+\epsilon}} \cdot \sum_i \nabla\hat{x}_i$$
$$= \left(\nabla\hat{x}_i \cdot \frac{1}{\sqrt{\sigma^2+\epsilon}}\right) - \left(\frac{1}{m\sqrt{\sigma^2+\epsilon}} \cdot \nabla\hat{x}_i \sum_j \nabla\hat{x}_j \hat{x}_j\right) + \frac{-1}{m\sqrt{\sigma^2+\epsilon}} \cdot \sum_i \nabla\hat{x}_i$$
$$= \frac{1}{m\sqrt{\sigma^2+\epsilon}} \left(m\nabla\hat{x}_i - \nabla\hat{x}_i \sum_j \nabla\hat{x}_j \hat{x}_j - \sum_i \hat{x}_i\right)$$

The simplified version of backprop of batchnorm is given as:

```
dbeta = np.sum(dout, axis=0)
dx_hat = dout * gamma
dx = var_inv / m * (dx_hat * m - dx_hat.sum(0) - x_hat * (dx_hat * x_hat)
    .sum(0))
```

In the experiment we can see the difference between naive version and simplified version is very close to zero, but simplified version is 2.17 times faster than naive one, which is a significant improvement because we insert batchnorm layers after each conv layer to accelerate the training so we want the cost to be minimal.

2 Plans