MFDNN HW7

MinGyu Shin

April 26, 2024

Problem 1 : *The true softmax.*

(a) Let $max = \max\{x_1, \dots, x_n\}$ for given x.

$$\begin{aligned}\nu_\beta(x) - max &= \frac{1}{\beta}\log\sum_{i=1}^{n}\exp(\beta x_i) - \frac{1}{\beta}\log\exp(\beta max) \\ &= \frac{1}{\beta}\log\frac{\sum_{i=1}^{n}\exp(\beta x_i)}{\exp(\beta max)} \leq \frac{1}{\beta}\log n.\end{aligned}$$

Since $0 \leq \nu_\beta(x) - max \leq \frac{1}{\beta}\log n, \nu_\beta(x) - max \to 0$ as $\beta \to \infty$.

(b)

$$\frac{\partial \nu_\beta}{\partial x_k} = \frac{1}{\beta}\frac{\partial \log\sum_{i=1}^{n}\exp(\beta x_i)}{\partial x_k} = \frac{1}{\beta}\frac{\beta\exp(\beta x_k)}{\sum_{i=1}^{n}\exp(\beta x_i)} = \frac{\exp(\beta x_k)}{\sum_{i=1}^{n}\exp(\beta x_i)}$$

$$\Rightarrow \nabla\nu_\beta(x) = \left[\left(\frac{\exp(\beta x_k)}{\sum_{i=1}^{n}\exp(\beta x_i)}\right)_{k=1,\dots,n}\right]^\intercal$$

(c) Note that, for x that $i_{\max} = \mathrm{argmax}_{1\leq i\leq n} x_i$ is uniquely defined,

$$\frac{\sum_{i=1}^{n}\exp(\beta x_i)}{\exp(\beta x_k)} = 1 + \sum_{\substack{i=1 \\ i\neq k}}^{n}\frac{\exp(\beta x_i)}{\exp(\beta x_k)} \to \begin{cases}\infty & x_k \neq max \\ 1 & x_k = max\end{cases} \quad \text{as } \beta \to \infty.$$

Thus,

$$\nabla\nu_\beta(x) \to e_{i_{\max}} \quad \text{as } \beta \to \infty$$

Problem 2 : *Are linear layers compute-heavy?*

The given AlexNet has shape flow below.

$(3\times227\times227) \underset{\text{conv}}{\rightarrow} (64\times55\times55) \underset{\text{maxpool}}{\rightarrow} (64\times27\times27) \underset{\text{conv}}{\rightarrow} (192\times27\times27) \underset{\text{maxpool}}{\rightarrow} (192\times13\times13) \underset{\text{conv}}{\rightarrow} (384\times13\times13) \underset{\text{conv}}{\rightarrow} (256\times13\times13) \underset{\text{conv}}{\rightarrow} (256\times13\times13) \underset{\text{maxpool}}{\rightarrow} (256\times6\times6) \underset{\text{avgpool}}{\rightarrow} (256\times6\times6) \underset{\text{linear}}{\rightarrow} (4096) \underset{\text{linear}}{\rightarrow} (4096) \underset{\text{linear}}{\rightarrow} (1000)$

Now count the operations with 'the number of convolution layer addition and multiplication formula' ($\underbrace{pk^2}_{\text{per a position}} l^2 n$) where $p = \#$ of input channels, $k =$ kernel size, $l =$ output layer size and $n = \#$ of output channels.

$$\# \text{ of the operations in conv} : (3 \times 11^2 \times 55^2 \times 64) + (64 \times 5^2 \times 27^2 \times 192)$$
$$+ (192 \times 3^2 \times 13^2 \times 384) + (384 \times 3^2 \times 13^2 \times 256) + (256 \times 3^2 \times 13^2 \times 256) = 655,566,528$$

Now count the operations with 'the number of linear layer addition and multiplication formula' pq where $p =$ input layer size and $q =$ output layer size.

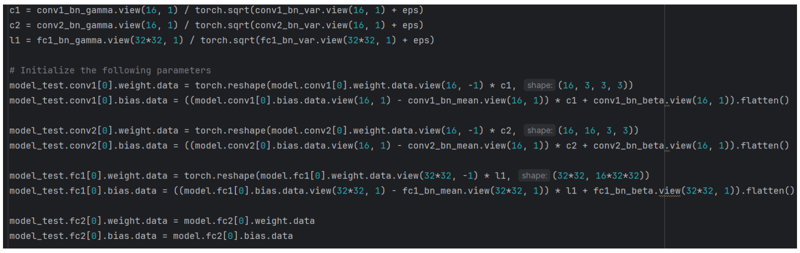
$$\# \text{ of the operations in linear} : (9216 \times 4096) + (4096 \times 4096) + (4096 \times 1000) = 58,621,952$$

Problem 3 : *Removing BN after training.*

Roughly, original convolution with batch norm can be represented like

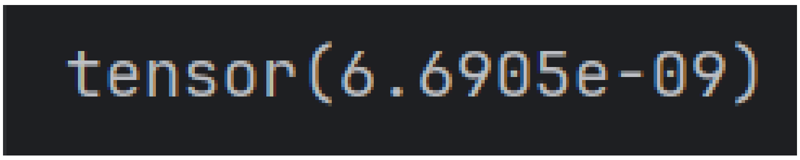
$$\gamma \frac{\sum wx + b - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta = \sum \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}} wx + \frac{(b - \mu)\gamma}{\sqrt{\sigma^2 + \epsilon}} + \beta.$$

The implemented code and the result(difference) are below.



```
c1 = conv1_bn_gamma.view(16, 1) / torch.sqrt(conv1_bn_var.view(16, 1) + eps)
c2 = conv2_bn_gamma.view(16, 1) / torch.sqrt(conv2_bn_var.view(16, 1) + eps)
l1 = fc1_bn_gamma.view(32*32, 1) / torch.sqrt(fc1_bn_var.view(32*32, 1) + eps)

# Initialize the following parameters
model_test.conv1[0].weight.data = torch.reshape(model.conv1[0].weight.data.view(16, -1) * c1, shape: (16, 3, 3, 3))
model_test.conv1[0].bias.data = ((model.conv1[0].bias.data.view(16, 1) - conv1_bn_mean.view(16, 1)) * c1 + conv1_bn_beta.view(16, 1)).flatten()

model_test.conv2[0].weight.data = torch.reshape(model.conv2[0].weight.data.view(16, -1) * c2, shape: (16, 16, 3, 3))
model_test.conv2[0].bias.data = ((model.conv2[0].bias.data.view(16, 1) - conv2_bn_mean.view(16, 1)) * c2 + conv2_bn_beta.view(16, 1)).flatten()

model_test.fc1[0].weight.data = torch.reshape(model.fc1[0].weight.data.view(32*32, -1) * l1, shape: (32*32, 16*32*32))
model_test.fc1[0].bias.data = ((model.fc1[0].bias.data.view(32*32, 1) - fc1_bn_mean.view(32*32, 1)) * l1 + fc1_bn_beta.view(32*32, 1)).flatten()

model_test.fc2[0].weight.data = model.fc2[0].weight.data
model_test.fc2[0].bias.data = model.fc2[0].bias.data
```

(a) Implemented code



```
tensor(6.6905e-09)
```

(b) Result(Difference between with/without batchnorm)

Problem 4 : *Backprop with convolutions.*

(a)

$$\frac{\partial y_L}{\partial y_{L-1}} = \frac{\partial (A_{w_L} y_{L-1} + b_L \mathbf{1}_{n_L})}{\partial y_{L-1}} = A_{w_L}$$

$$\frac{\partial (y_\ell)_i}{\partial y_{\ell-1}} = \frac{\partial \sigma((A_{w_\ell})_{i:} y_{\ell-1} + b_\ell)}{\partial y_{\ell-1}} = \sigma'((A_{w_\ell})_{i:} y_{\ell-1} + b_\ell)(A_{w_\ell})_{i:}$$

$$\Rightarrow \frac{\partial y_\ell}{\partial y_{\ell-1}} = \mathrm{diag}(\sigma'(A_{w_\ell} y_{\ell-1} + b_\ell \mathbf{1}_{n_\ell}))A_{w_\ell} \qquad \text{for } \ell = 2, \ldots, L-1$$

where $A_{i:} =$ i-th row of A.

Note that for $w \in \mathbb{R}^f, y \in \mathbb{R}^n$,

$$(\mathcal{C}_w y)_i = \left(\begin{bmatrix} —w—0\dots 0 \\ 0—w—0\dots 0 \\ \vdots \\ 0\dots 0—w— \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \right)_i = w^\intercal \cdot \begin{bmatrix} y_i \\ \vdots \\ y_{i+f-1} \end{bmatrix} \quad \text{for } i = 1, \dots n-f+1$$

$$\frac{\partial y_\ell}{\partial (w_\ell)_i} = \frac{\partial(\sigma(A_{w_\ell} y_{\ell-1} + b_\ell \mathbf{1}_{n_\ell}))}{\partial (w_\ell)_i} = \sigma'(A_{w_\ell} y_{\ell-1} + b_\ell \mathbf{1}_{n_\ell}) \odot \begin{bmatrix} (y_{\ell-1})_i \\ (y_{\ell-1})_{i+1} \\ \vdots \\ (y_{\ell-1})_{i+n_\ell-1} \end{bmatrix}$$

$$\Rightarrow \frac{\partial y_\ell}{\partial w_\ell} = \operatorname{diag}(\sigma'(A_{w_\ell} y_{\ell-1} + b_\ell \mathbf{1}_{n_\ell})) \begin{bmatrix} (y_{\ell-1})_i \\ (y_{\ell-1})_{i+1} \\ \vdots \\ (y_{\ell-1})_{i+n_\ell-1} \end{bmatrix}_{i=1,\dots,f_\ell}$$

$$\Rightarrow (\frac{\partial y_L}{\partial w_\ell})_k = \left(\frac{\partial y_L}{\partial y_\ell} \operatorname{diag}(\sigma'(A_{w_\ell} y_{\ell-1} + b_\ell \mathbf{1}_{n_\ell})) \begin{bmatrix} (y_{\ell-1})_i \\ (y_{\ell-1})_{i+1} \\ \vdots \\ (y_{\ell-1})_{i+n_\ell-1} \end{bmatrix}_{i=1,\dots,f_\ell} \right)_k$$

$$= v_\ell \begin{bmatrix} (y_{\ell-1})_k \\ (y_{\ell-1})_{k+1} \\ \vdots \\ (y_{\ell-1})_{k+n_\ell-1} \end{bmatrix} = (\mathcal{C}_{v_\ell^\intercal} y_{\ell-1})_k$$

Since $\frac{\partial y_L}{\partial w_\ell} \in \mathbb{R}^{1\times f_\ell}, \mathcal{C}_{v_\ell^\intercal} y_{\ell-1} \in \mathbb{R}^{f_\ell \times 1}$,

$$\frac{\partial y_L}{\partial w_\ell} = (\mathcal{C}_{v_\ell^\intercal} y_{\ell-1})^\intercal$$

$$\frac{\partial y_L}{\partial b_\ell} = \frac{\partial y_L}{\partial y_\ell}\frac{\partial y_\ell}{\partial b_\ell} = \frac{\partial y_L}{\partial y_\ell}\frac{\partial(\sigma(A_{w_\ell} y_{\ell-1} + b_\ell \mathbf{1}_{n_\ell}))}{\partial b_\ell} = \frac{\partial y_L}{\partial y_\ell}\operatorname{diag}(\sigma'(A_{w_\ell} y_{\ell-1} + b_\ell \mathbf{1}_{n_\ell}))\mathbf{1}_{n_\ell} = v_\ell \mathbf{1}_{n_\ell}$$

(b) One should avoid matrix-matrix products, and for matrix-vector(reverse way can be represented by transpose), slicing vector with filter size are followed by vector-vector products.

Problem 5 : *Lager network in network.*

The implemented codes and the result are below.

```
def copy_weights_from(self, net1):
    with torch.no_grad():
        for i in range(0, len(self.features), 2):
            self.features[i].weight.copy_(net1.features[i].weight)
            self.features[i].bias.copy_(net1.features[i].bias)

        for i in range(0, len(self.classifier), 2):
            _sp = self.classifier[i].weight.shape
            self.classifier[i].weight.copy_(net1.classifier[i].weight.reshape(_sp))
            self.classifier[i].bias.copy_(net1.classifier[i].bias)
```

(a) (a) code

```
images, _ = next(iter(test_loader))
b, w, h = images.shape[0], images.shape[-1], images.shape[-2]
out1 = torch.empty((b, 10, h - 31, w - 31))
for i in range(h - 31):
    for j in range(w - 31):
        out1[:, :, i, j] = model1(images[:, :, i:i + 32, j:j + 32])
out2 = model2(images)
diff = torch.mean((out1 - out2) ** 2)
```

(b) (b) code

```
Files already downloaded and verified
Average Pixel Difference: 7.947929843257794e-17
Files already downloaded and verified
Average Pixel Diff: 7.111637009076882e-17
```

(c) Result (difference)