| | |
|---|---|
| Com S 573 | Spring Semester 2021 |
| Machine Learning | |
| Homework 5 | |
| Instructor: Hongyang Gao | Submitted by: Ibne Farabi Shihab |

# 1 no Question

**Solution** (a)

Number of trainable parameter for layer 1:

$C_{in} = 128$

$C_{in} = 256$

$k = 3$

Number of trainable parameter $=3 \times 3 \times 128 \times 256 = 294912$

Number of trainable parameter for For layer 2:

$C_{in} = 256$

$C_{in} = 512$

$k = 3$

Number of trainable parameter $=3 \times 3 \times 256 \times 512 = 1179648$

Number of total trainable parameter $= 294912 + 1179648 = 1474560$

**Solution** (b)

For layer 1:

$W = 64$

$D = 64$

$P = 1$

$k = 3$

$S = 2$

$M = 128$

$N = 256$

Output dimension $W^{'} = (64 - 3 + 2 * 1)/(2 + 1) = 21$

Output dimension $D^{'} = (64 - 3 + 2 * 1)/(2 + 1) = 21$

Number of number of multi-add operations $=(3 \times 3 \times 128 \times 256) \times 21 \times 21 = 130056192$

For layer 2:

$W = 21$

$D = 21$

$P = 1$

$k = 3$

$S = 1$

$M = 256$

$N = 512$ Output dimension $W^{'} = (21 - 3 + 2 * 1)/(1 + 1) = 10$

Output dimension $D^{'} = (21 - 3 + 2 * 1)/(1 + 1) = 10$

Number of number of multi-add operations $=(3 \times 3 \times 256 \times 512) \times 10 \times 10 = 117964800$

Number of total Multi add operations $= 117964800 + 130056192 = 248020992$

**Solution** (c)

For input features = 64*64*128 = 524288

For output features = 10*10*512 = 51200

Change in Memory = $\frac{524288-51200}{524288} \times 100 = 90.23\%$

**or**

Layer 1: 64*64*128=524288

Layer 2: 21*21*256=112896

Change in memory:

(524288-112896)/524288=78.5 %

## 2 No Question

**Solution (a)**

Step 1 : Computing mini-batch mean using the formula: $\mu = \frac{1}{m}\sum_{i=1}^{m} z_i$

| Data | Elements | | | | Minibatch Mean |
|---|---|---|---|---|---|
| M1 | 12 | 14 | 14 | 12 | 13 |
| M2 | 0 | 10 | 10 | 0 | 5 |
| M3 | -5 | 5 | 5 | -5 | 0 |

Step 2 : Computing mini-batch variance using the following equation: $\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(z_i - \mu)^2$.

| Data | Each Item | | | | Minibatch Variance |
|---|---|---|---|---|---|
| M1 | 1 | 1 | 1 | 1 | |
| M2 | 25 | 25 | 25 | 25 | |
| M3 | 25 | 25 | 25 | 25 | |

Step 3 : Computing normalized output according to

$$\hat{z}_i = \frac{z_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

and expressed as a matrix in the following:

$$\begin{bmatrix} -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 \end{bmatrix}$$

**solution (b)** : $\tilde{z}_i$ is calculated using the following equation:

$$\tilde{z}_i = \gamma\hat{z}_i + \beta.$$

and expressed as a matrix in the following:

$$\begin{bmatrix} -1 & 1 & 1 & -1 \\ -11 & -9 & -9 & -11 \\ 9 & 11 & 1 & 9 \end{bmatrix}$$

**solution c:**

In training, batch normalization is conducted using the following steps:

1. mini batch mean calculation
2. mini batch variance calculation
3. Normalization
4. Scaling and shifting

In test phase the mean/std are not computed based on the batch. Instead, empirical mean and std from training is being used

**Solution (d)**

Large batch size means the model makes very large gradient updates and very small gradient updates. The size of the update depends heavily on which particular samples are drawn from the data-set.For the same average Euclidean norm distance from the initial weights of the model, larger batch sizes have larger variance in the distance.Therefore, higher batch sizes leads to lower asymptotic test accuracy. we can recover the lost test accuracy from a larger batch size by increasing the learning rate.

Larger batch size can be resulted in a large gradient update or small one as it it totally depends on from which particular distribution the samples are drawn. Larger batch sizes will have larger variance. For that, higher batch sizes will lead to the test accuracy to the lower side asymptotically.

On the contrary, for small batch size, the update of model will be the same almost. The size of the update only weakly depends on which particular samples are drawn from the data-set. There is a particular space where the batch size makes the model performs best.

For this we need to keep the batch size as low as possible. A smaller mini-batch size which is not too small, usually leads not only to a smaller number of iterations of our training algorithm, but also to a higher accuracy overall then larger batch size.Having a good batch size helps our model to to generalization much better which helps in the testing phase.

# 3 No question

**Solution (a)**

$$\begin{bmatrix} y_{11} \\ y_{12} \\ y_{21} \\ y_{22} \end{bmatrix} = \begin{bmatrix} w_1 & w_2 & w_3 & 0 & 0 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 \end{bmatrix} \tilde{[}x_{11}, x_{12}, x_{13}, x_{14}, x_{21}, x_{22}, x_{23}, x_{24}]^T$$

**Solution (b)**

Solution: In this question we need to find $\frac{\partial L}{\partial \tilde{X}}$. For that using chain rule,we can express it as follow: $\frac{\partial L}{\partial \tilde{X}} = \frac{\partial Y}{\partial \tilde{X}} \frac{\partial L}{\partial \tilde{Y}}$.

Using 3(a), we can get, $\frac{\partial \tilde{Y}}{\partial \tilde{X}} = \begin{bmatrix} w_1 & w_2 & w_3 & 0 & 0 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 \end{bmatrix}$

So,

$$\frac{\partial L}{\partial \tilde{X}} = \begin{bmatrix} w_1 & w_2 & w_3 & 0 & 0 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 \end{bmatrix}^\top \begin{bmatrix} \frac{\partial L}{\partial y_{11}} \\ \frac{\partial L}{\partial y_{12}} \\ \frac{\partial L}{\partial y_{21}} \\ \frac{\partial L}{\partial y_{22}} \end{bmatrix}^\top$$

Here A and B are the same apart from the fact that B is a transposed version of A.

**Solution (c)**

Solution: Yes, it is possible. We can use transposed version of A as our new kernel for this. The kernel will be:

$$\begin{bmatrix} w_1 & w_2 & w_3 & 0 & 0 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 \end{bmatrix}^\top$$

# 4 no Question

**Solution (a)**

The testing accuracy is 64.14% while the training accuracy is 76.62%. If we look at each epochs, we can see that the training accuracy is increasing for all of the epochs. Till the last epochs, the training accuracy was on the increasing side. However, the difference between training and testing accuracy is on the higher side(around 12%). From here, we can conclude that, the model is overfitting. The amount of time the model took to run is 8 mintues and 41 seconds in my pc.

**Solution (b)**

For this part, the model took a bit longer to run than the previous which is obvious as we know batch normalization do take more time. The model took 9 minutes and 36 seconds to run. In terms of accuracy, the testing accuracy improved slightly to 65.04% while the training accuracy went till 80.08 %. Here, though the batch normalization did improve the accuracy in terms of both testing and training, the overfitting issue is still here.

**Solution (c)**

After adding a dropout layer, both training and testing accuracy fall back to 75.258% and 65.73% respectively. Here the margin between the training and testing accuracy is still here which indicates that the model is still overfitting. The time also increased a bit to 9 minutes and 41 seconds.

After all this 3 improvement, we can come the conclusion that batch normalization do help to improve accuracy while dropout help to reduce the overfitting problem.