Homework I

Li Hantao, G2101725H, MSAI, hli038@e.ntu.edu.sg

I. QUESTION I

The input has a shape of 1x32x32. The output shape of each layer is provided as [<ignore>, output channels, height, width]. For instance, at layer 'Conv2d-1', the output shape is [6, 28, 28], i.e., six feature maps of spatial size 28x28. Each conv filter and neuron of linear layer has a bias term and stride = 1.

Layer (type)	Output Shape
Conv2d-1	[-1, 6, 28, 28]
ReLU-2	[-1, 6, 28, 28]
MaxPool2d-3	[-1, 6, 14, 14]
Conv2d-4	[-1, 16, 10, 10]
ReLU-5	[-1, 16, 10, 10]
MaxPool2d-6	[-1, 16, 5, 5]
Conv2d-7	[-1, 120, 1, 1]
ReLU-8	[-1, 120, 1, 1]
Linear-9	[-1, 84]
ReLU-10	[-1, 84]
Linear-11	[-1, 10]
LogSoftmax-12	[-1, 10]

Calculate the number of parameters for each layer and finally the total number of parameters of this network.

Answer:

a) For the relationship between the input size and output size of Convolutional layer, we have:

$$Output\ Size = \frac{Input\ Size - Filter\ Size + 2 \times Padding}{Stride} + 1$$

We have the **Stride = 1** and **Padding = 0**. Thus, in this question, we can calculate each **Kernel size** with the following formula:

$$Output\ Size = Input\ Size - Filter\ Size + 1$$

$$Filter\ Size = Input\ Size - Output\ Size + 1$$

b) For the **Activation function layer** (ReLU and Softmax), there is no parameters needed. Also, for the **downsampling Max-pooling layer**, there is no parameters needed. Thus, we only need to consider the convolutional layer and linear layer.

For the **Convolutional layer**, we have the following formula about the number of parameters *N*:

$$N = Channel_{Output} \times \left(Channel_{Input} \times Filter\ Size^2 + 1\right)$$

For the **Linear layer**, we have the following formula:

$$N = Channel_{Output} \times \left(Channel_{Input} + 1\right)$$

c) Then, we go through **each layer** one by one:

1) 1*32*32 -> Convolutional layer -> 6*28*28:
Input Size, Output Size = 32, 28
Kernel Size = 5
Input Channel, Output Channel = 1, 6
Number of Parameters: $\mathbf{6 \times (1 \times 5^2 + 1) = 156}$

2) 6*28*28 -> ReLU -> 6*28*28:
Number of Parameters: 0

3) 6*28*28 -> Max-Pooling -> 6*14*14:
Number of Parameters: 0

4) 6*14*14 -> Convolutional layer ->: 16*10*10
Input Size, Output Size = 14, 10
Kernel Size = 5
Input Channel, Output Channel = 6, 16
Number of Parameters: $\mathbf{16 \times (6 \times 5^2 + 1) = 2416}$

5) 16*10*10 -> ReLU -> 16*10*10:
Number of Parameters: 0

6) 16*10*10 -> Max-Pooling -> 16*5*5:
Number of Parameters: 0

7) 16*5*5 -> Convolutional layer ->: 120*1*1
Input Size, Output Size = 5, 1
Kernel Size = 5
Input Channel, Output Channel = 16, 120
Number of Parameters: $\mathbf{120 \times (16 \times 5^2 + 1) = 48120}$

8) 120*1*1 -> ReLU -> 120*1*1:
Number of Parameters: 0

9) 120 -> Linear layer -> 84
Input Channel, Output Channel = 120, 84
Number of Parameters: $\mathbf{84 \times (120 + 1) = 10164}$

10) 84*1*1 -> ReLU -> 84*1*1:
Number of Parameters: 0

11) 84 -> Linear layer -> 10
Input Channel, Output Channel = 84, 10
Number of Parameters: $\mathbf{10 \times (84 + 1) = 850}$

12) 10*1*1 -> LogSoftmax -> 10*1*1:
Number of Parameters: 0

d) Thus, the **total number of parameters** are:

$$\sum N_i = 156 + 0 + 0 + 2416 + 0 + 0 + 48120 + 0 + 10164 + 0 + 850 + 0 = 61706$$

II. QUESTION II

Let us consider the convolution of single-channel tensors $\mathbf{x} \in \mathbb{R}^{4\times4}$ and $\mathbf{w} \in \mathbb{R}^{3\times3}$

$$\boldsymbol{w} \star \boldsymbol{x} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 10 & 10 & 0 & 0 \\ 10 & 10 & 0 & 0 \\ 10 & 10 & 0 & 0 \\ 10 & 10 & 0 & 0 \end{pmatrix}$$

Perform convolution as matrix multiplication by converting the kernel into sparse Toeplitz circulant matrix. Show your steps.

Answer:

a) Toeplitz circulant matrix:

$$\mathbf{W} = \begin{pmatrix} -1 & 0 & 1 & 0 & -2 & 0 & 2 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & -2 & 0 & 2 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & -2 & 0 & 2 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & -2 & 0 & 2 & 0 & -1 & 0 & 1 \end{pmatrix}$$

$$v(\mathbf{x}) = (10 \quad 10 \quad 0 \quad 0 \quad 10 \quad 10 \quad 0 \quad 0 \quad 10 \quad 10 \quad 0 \quad 0 \quad 10 \quad 10 \quad 0 \quad 0)^{\mathrm{T}}$$

$$\mathbf{W}v(\mathbf{x}) = \begin{pmatrix} (-1\times10)+(-2\times10)+(-1\times10) \\ (-1\times10)+(-2\times10)+(-1\times10) \\ (-1\times10)+(-2\times10)+(-1\times10) \\ (-1\times10)+(-2\times10)+(-1\times10) \end{pmatrix} = \begin{pmatrix} -40 \\ -40 \\ -40 \\ -40 \end{pmatrix}$$

b) Reshape it:

$$\boldsymbol{w} \star \boldsymbol{x} = \begin{pmatrix} -40 & -40 \\ -40 & -40 \end{pmatrix}$$

III. QUESTION III

Many people in Singapore like to eat durian. Many customers believe that a perfectly oval and rounded durian is not always the best. An odd-shaped fruit that comes in slightly curved and crescent shape may taste better. You decide to train an image classifier to predict whether a durian is with rounded shape (label=0) or odd shape (label=1).

i) You've collected your own labeled dataset, chosen a neural network architecture, and are thinking about using the mean squared error (MSE) loss to optimize model parameters. Give one reason why MSE might not be a good choice for your loss function.

Answer:

Firstly, in the assumption of MSE when utilizing it as the loss function, we assume that the error follows the Gaussian distribution and use the maximum likelihood estimation (MLE) to derive the MSE to optimize the model. However, the durian shape dataset we collected follows the Bernoulli distribution.

Secondly, we usually need to use the activation function layer, such as Softmax (or in this task, it is the same as Sigmoid) before computing the loss. When we are computing the gradient, we will get $\partial L/\partial w = 0.5(y-\hat{y})\sigma'(wx+b)x$. If the variable of Sigmoid is very large or very small, the value of the partial derivative is close to zero, which will lead to the gradient vanishing. When BCE is used, the gradient of w and b will not be like MSE. Therefore, MSE is not applicable to classification problems like this durian task.

ii) You decide to use the binary cross-entropy (BCE) loss to optimize your network. Write down the formula for this loss (for a single example) in terms of the label y and prediction $\hat{y}$.

Answer:

$$BCE(\hat{y}, y) = -(y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y}))$$

iii) Compute the total cost, J, of the network averaged across the following dataset of three examples using the binary cross entropy loss. $\mathrm{Y} = (1, 0, 0)^{\mathrm{T}}$, and $\hat{\mathrm{Y}} = (0.2, 0.5, 0.1)^{\mathrm{T}}$. There is no penalty on the weights.

Answer:

$$J = -\frac{1}{n}\sum_{i} Y_i \cdot \log(\hat{Y}_i) = -\frac{1}{3}[1 \times \log(0.2) + 1 \times \log(0.5) + 1 \times \log(0.9)] = 0.802$$

iv) You decide to train one model with L2 regularization (model A) and one without (model B). How would you expect model A's weights to compare to model B's weights?

Answer:

The weights of A should be smaller than the weights of B. L2 regularization, a method of reducing the model complexity and preventing overfitting, aims to constrain the weight of the model to train a robust model with high performance on the unseen test data.

IV. QUESTION IV

Why might we prefer to minimize the sum of absolute residuals (L1 loss) instead of the residual sum of squares for some data sets (L2 loss)? (Hint: What is one of the flaws of least-squares regression?).

Answer:
The formula of L1 loss and L2 loss are shown below:

$$L1\ Loss = \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

$$L2\ Loss = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

L2 loss performs the square operation. Thus, when the difference between the real label and the predicted label is greater than 1, the error will be amplified; while when the difference is less than 1, the error will be reduced, which is determined by the square operation. Therefore, L2 loss gives a more significant penalty for larger errors (> 1) and a minor penalty for smaller errors (< 1). In other words, it is sensitive to outliers' data and is greatly affected by them. If there are outliers in the training dataset, L2 loss will give higher weight to the outliers, which will 'overlap' the results of other regular data, reducing the overall model performance.

For example, we have 100 regular points and 1 outlier in a regression task. The 100 regular points all have a distance of 1, compared with the target, and the outlier has 10 as the distance. L1 loss of them is 100*1 / 1*10, while L2 loss is 100*1 / 1*100. When we do the backpropagation, the regression result of L2 loss will be biased towards the outlier significantly.

All in all, if outliers can be detected and processed before the training, L2 loss can be selected as the loss function; however, if the outliers are only regarded as regular data and will enter the iterative step, the L1 loss should be selected as the loss function, which is not sensitive to the outlier.

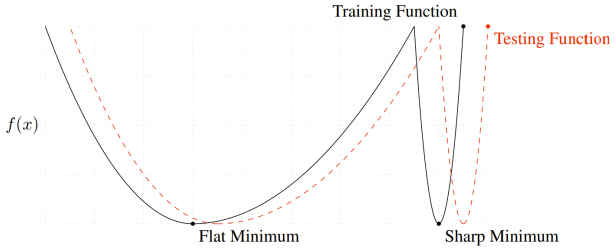
V. QUESTION V

You want to apply batch normalization in your network. Explain why you shouldn't choose a very small mini-batch size during your training.

Answer:

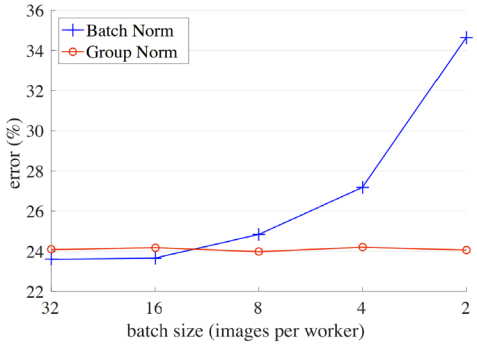
For the training process:

1. In a platform without parallel computing, a small dataset requires less time for a single iteration gradient computing. However, when we use a modern GPU to train the network (Tesla V100, for instance), a larger batch size does not require a longer time to compute the gradient for one batch. (Unless the batch size is too large. For batch size from 1 to 8000, the time for one update in one iteration is almost the same.) In this circumstance, a very small batch size requires a longer time for one epoch training, for they need more updates in one epoch while the single updating time is the same.
2. Theoretically, the batch size determines the gradient smoothness between adjacent iterations. If the batch size is too small and the difference between adjacent mini-batches is relatively significant, the gradient oscillation of the two adjacent iterations will be more serious, which is not conducive to convergence. (However, practice shows that the performance on the training set has no significant differences between small batch size and large batch size. Moreover, a small batch size network usually performs better on the test data. That is maybe because the randomness of the loss plane of large batch size in the different iterations is not enough, resulting in the higher possibility of falling into an unsatisfactory sharp minimum, as shown in the figure below. [1])



For the Batch Normalization process:

3. Batch Normalization normalizes the features by the mean and variance computed within a mini-batch. In other words, the normalized data should have the ability to reflect the distribution and characteristics of the original data. However, when we use small batch sizes, such as 4 and 2, the capability of the mean and variance of the batch data to represent the original data is poor, so it has a significant impact on the batch normalization results. A small batch leads to inaccurate estimation of the batch statistics, and reducing Batch Normalization's batch size increases the model error dramatically, as the figure below. [2] To solve such problem, some researchers redesigned the network training process, such as Fast R-CNN, they 'froze' the BN parameters [3]; others used Group Normalization as a simple alternative to Batch Normalization.



[1] Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. arXiv preprint arXiv:1609.04836.
[2] Wu, Y., & He, K. (2018). Group normalization. In Proceedings of the European conference on computer vision (ECCV) (pp. 3-19).
[3] Girshick, R. (2015). Fast r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 1440-1448)