Fundamentals of Neural Networks

Yuncheng Yuan(1778307)

January 22, 2023

1 Multi-Layer Perceptron

1.1 Implementation

1.2 Training and Testing

1. Train the MLP and report the training and test accuracies. Explain why there are differences in training and test performance?

   Ans:

   The training accuracy is 0.8741 and test accuracy is 0.9231.

   Normally,

2. Calculate the number of trainable parameters of your model (you can implement it or calculate by hand) and explain how you have estimated it?

   Ans:

   the trainable parameter in the first layer is (784 + 1) *10 = 7850. 1 is the bias. In the second layer, the trainable parameter is (10 + 1) *10. 10 is the output from the first neuron layer. 1 is the bias.

3. How would the number of parameters vary if you use a convolution layer with batch normalization layer instead of a fully connected layer (explain)? What is the relation between a convolution and fully-connected layer, when are they the same/different?

   Ans:

   the parameter of Convolution layer will be much smaller than the fully connected layer. A batch normalization layer typically has 4 parameters per input channel, so it would add a relatively small number of parameters to the model. Because there is only small number of kernels like 3*3 and channels are usually smaller than 64. Therefore, there is only 3*3*64 = 576 parameters in one convolutional layer with batch normalization layer.

   Convolutional layers are typically used for image. Fully connected layers are more general and can be used for any kind of data. But if a convolutional layer is followed by a global average pooling layer and then a fully connected layer, in that case, the convolutional layer can be replaced by a fully connected layer with the same number of parameters and the same performance.

4. Use the knowledge acquired in the course or sources online to improve the performance on the MNIST dataset. Try to achieve an accuracy of more than what a standard CNN can achieve (say more than 98.5-99.0). You can use PyTorch for this question.

   (a) Explain in detail the improvements and changes that you have added. Well reasoned answers will get more/full points.

   Ans:

   1, Add dropout layer. It randomly sets a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting.

   2, Add maxpolling layer. It reduce the spatial dimensions of the input data while retaining the most important information.

   3, Add iteration times. A

(b) Given that you have unlimited resources for computation and no extra data, how would you improve performance on the MNIST dataset?

Ans:

1, Use more advanced layers like SeparableConv2D to improve performance.

2, Use more advanced architectures like ResNet to improve the model's performance.

3, Use data augmentation to generate new training examples by applying various transformations to the original images can help the model generalize better.

2 Loss functions

There are several loss function that are often used in Convolutional Neural Networks (CNNs). Depending on the objective, different loss functions can be used. In this part of the project, you will derivative the gradients of these loss functions with respect to the outputs y or logits z.

The softmax function is given as

$$y = \sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Cross-Entropy Loss

$$-(y\log(p) + (1-y)\log(1-p))$$

1. Derive the derivative of the output y with respect to logits z

   Ans:

$$\frac{\partial}{\partial z_i}\frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}\left(1 - \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}\right)$$

2. Use the condition when i = j, and derive the derivative of the loss function Lce with respect to the logits z.

   Ans:

$$\frac{\partial L}{\partial z_i} = -y_i + y_i \frac{e^{z_i}}{e^{z_i} + \sum_{j\neq i} e^{z_j}} = -y_i + y_i * P(y = i|x)$$

3. Show that

$$\frac{\partial L_{dice}}{\partial z} = \frac{(y-1)}{2y}(1 - L_{dice})^2$$

4. Expand the generic to the binary version of focal loss.

   Ans:

$$L_{focal}(y,t) = -(1-y)^{\gamma} * log(y) - (1-(y)^{\gamma}) * log(1-y)$$

5. Show that

$$\frac{\partial L_{focal}}{\partial y} = \frac{-1}{y(1-y)}[t(1-y)^{\gamma} * (\gamma * \varepsilon_y + 1 - y) + (t-1) * y^{\gamma} * (\gamma * \varepsilon_{1-y} + y)]$$

   when $\varepsilon_p = -p * log(p)$

   Ans:

   because: $\varepsilon_p = -p * log(p)$,

   so we can get that:

$$\frac{\partial L_{focal}}{\partial y} = \frac{1}{y(y-1)} * [t(1-y)^{\gamma} * (\gamma * -ylog(y) - y + 1) + (t-1)y^{\gamma}(\gamma * ylog(1-y) - \gamma log(1-y) + y)]$$

Therefore,

$$\frac{\partial L_{focal}}{\partial y} = \frac{-1}{y(1-y)} * [t(1-y)^{\gamma} * (\gamma * -ylog(y) - y + 1) + (t-1)y^{\gamma}(\gamma * ylog(1-y) - \gamma log(1-y) + y)]$$

6. Under what condition is
$$\frac{\partial L_{focal}}{\partial y} = \frac{\partial L_{ce}}{\partial y}$$
What happens when γ value is too high or low in focal loss? Explain in detail how it would impact performance?

   Ans:

   When the γ value is too high, the weighting factor becomes very large for predictions that are close to 0, and the model will be heavily penalized for predictions that are not confident in the positive class. This can cause the model to become overly cautious and make fewer positive predictions overall, resulting in poor performance on the positive class.

   when the γ value is too low, the weighting factor will not be large enough to penalize predictions that are not confident in the positive class, which means the model might predict the positive class even when the input is not representative of the positive class. This can cause overfitting to the negative class, resulting in poor performance on the positive class.

7. List the applications where each loss function would be useful and explain why?

   Ans:

   Cross-entropy loss: Cross-entropy loss is used when adjusting model weights during training. The aim is to minimize the loss, i.e, the smaller the loss the better the model. A perfect model has a cross-entropy loss of 0. Cross-Entropy gives a good measure of how effective each model is.

   Dice Loss: Dice loss is often used in image segmentation tasks. By minimizing the Dice loss during training, the model is encouraged to produce segmentation maps that closely match the ground truth.

8. Discuss (in detail) the behavior of precision and recall for each of the listed loss functions above. Use the proofs from questions 2, 3 and 5 to support your answers.