Deep Learning

Mohammad Saifullah Khan - 21169

March 26, 2024

The codes for the questions can be accessed in the Github repository.

Question 1:

a. Cross-Entropy Loss (CE) is employed in logistic regression to guarantee a precise and definitive single correct outcome.

b. Logistic regression aims to sort data points into two separate categories, with the selection of a single optimal solution enhancing the clarity of the model's forecasts.

c. i. Mean Squared Error (MSE) aims to reduce the squared difference between the predicted probabilities and the actual values (0 or 1). Nonetheless, in the context of logistic regression, even a slight deviation in the predicted probability from the true class (0 or 1) can indicate a major classification mistake.
ii. Cross entropy addresses this by applying more severe penalties for errors in predicted probabilities, especially when there is a significant discrepancy between the predicted probability and the actual class (0 or 1).

Question 2: Both Cross-Entropy (CE) loss and Mean Squared Error (MSE) loss do not ensure a convex optimization landscape.

(a) **CE Loss:** For binary classification, the CE loss function is defined as:

$$\mathrm{CE}(y,\hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i\log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]$$

where y is the true label, $\hat{y}$ is the predicted probability, and N is the number of samples. The CE loss is not convex because of the presence of the logarithmic terms, which lead to non-convexity.

(b) **MSE Loss:** The MSE loss formula for binary classification is:

$$\mathrm{MSE}(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i-\hat{y}_i)^2$$

In this formula, the squared differences $(y_i - \hat{y}_i)^2$ introduce non-linear characteristics, rendering the loss function non-convex.

Since neither CE loss nor MSE loss guarantees convexity, the correct answer is **None**.

Question 3:

Consider a model designed for classifying images of digits, such as those found in the **MNIST** [2] dataset available in Pytorch. This dataset comprises a total of 70000, 28x28 images of digits.

a. **Architecture:** The model has 1 or 2 hidden layers
• The hidden layers have Dropout regularization [3] applied to them (during training)
• The hidden layers have "relu" or "tanh" activations
• The number of neurons in each hidden layer is a tunable as given in Table 1

b. The pre-processing strategies employed are:
• Reshaping: The images in the MNIST dataset are originally in a 28x28 pixel format. For the neural network to process them, they were reshaped into a single dimensional array of 784 elements (28 * 28).
• Scaling: Pixel values in the images range from 0 to 255. To normalize these values into a range suitable for neural network inputs, they were scaled to a range between 0 and 1 by dividing each pixel value by 255.

• One-hot encoding: The labels of the MNIST dataset, which range from 0 to 9 representing the digit images, were one-hot encoded. This means that each label was converted into a 10-dimensional vector where the index corresponding to the digit value is set to 1, and all other indices are set to 0. This is necessary because the model outputs a probability distribution across the 10 classes for each image.

Hyper-parameter	**Range of Values**
Hidden Layer 1 size	[128, 256, 512]
Hidden Layer 2 size	[128, 256, 512]
Learning rate in Adam	[0.001, 0.003]
Dropout rate	[0.0, 0.2]
Activation function	[”relu”, ”tanh”]

Table 1: The table displays the hyperparameters and their respective tuning ranges. The same learning rate, dropout rate, and activation function were applied consistently across all layers.

Hyper-parameter	**Optimal Value**
Number of neurons in Layer 1	512
Number of neurons in Layer 2	None
Learning rate	0.003
Dropout rate	0.0
Activation function	relu

Table 2: The table lists the hyperparameters and their optimal values. The same learning rate, dropout rate, and activation function have been uniformly applied to all layers.

Question 4:

Due to a shortage of computational resources, we have only trained the models for 10 epochs.

LeNet-5[1] :

It shows a steady improvement in accuracy from 17% to 84% over 10 epochs, which is impressive for such an early and simple convolutional network architecture. However, its relatively simpler architecture might limit its ability to capture the more complex features in the SVHN dataset compared to deeper models.

AlexNet [1] :

AlexNet’s performance remained relatively unchanged at 18% accuracy throughout the training period. This stagnation suggests that the model, despite being deeper than LeNet-5, may not be as effective at handling the complexity or variability of the SVHN dataset, possibly due to the way its architecture processes image features.

VGG-16[2] :

VGG-16 showed a very erratic performance with loss values spiking to extremely high numbers, indicating potential issues with overfitting, learning rate settings, or data preprocessing mismatches for the SVHN dataset. Its deeper architecture, which is known for success in large-scale image recognition, might not translate well to the specific characteristics of SVHN

[1] `https://shorturl.at/opPV0`
[2] `https://shorturl.at/ckrt8`

without significant tuning.

ResNet-18[3] :

ResNet-18 stands out with a strong start at 71% accuracy in the first epoch and a steady climb to 91% by the 10th epoch. The use of residual connections likely helps it effectively learn from the SVHN dataset without the vanishing gradient problem affecting its performance, making it well-suited for this task.

ResNet-50[4] and ResNet-101[5] :

While both models show significant improvement over time, their performances do not surpass that of ResNet-18. This might be due to the increased complexity of these models, which could lead to difficulties in training effectively on the relatively limited subset of the SVHN dataset used, suggesting that beyond a certain depth, additional layers may not yield proportional benefits for this specific task.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[2] Y. LeCun, C. Cortes, C. Burges, et al. Mnist handwritten digit database, 2010.

[3] S. Nitish. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15:1, 2014.

[3] https://shorturl.at/fmwX1
[4] https://shorturl.at/bkJQW
[5] https://shorturl.at/mKL27