Deep Learning Report

Devashish Tripathi 21093

26-03-2024

Question 1: Which loss function, out of Cross-Entropy and Mean Squared Error, works best with logistic regression because it guarantees a single best answer (no room for confusion)? Explain why this is important and maybe even show how it affects the model's training process.

Answer: Cross-entropy loss is used typically for classification problems, while MSE is used typically for regression problems in the supervised learning paradigm. Logistic Regression is used in binary classification problems. For logistic regression, the output is a probability value in the range [0,1], representing the chances for the input belonging to a particular class.
As such, cross-entropy loss would provide a better representation of the difference between the true and the predicted class. If we use MSE, we may only end up getting values like 0,1 and -1 as the losses, which won't tell us how much a particular example is getting misclassified.
For example, for a problem, we have the probability of an example i belonging to class 0 as p_i and belonging to class 1 as $1 - p_i$. Then

$$CELoss = -1/N \sum (y_i log(p_i) + (1 - y_i) log(1 - p_i))$$

And

$$MSE = 1/N \sum (y_i - z_i)^2$$

where N is the total number of examples, z_i is the predicted label and y_i is the correct label, for example i.
For an example, say y_i= 1, z_i=1 and p_i=0.8. Then $CELoss_i = -log(0.8) = 0.321$ and $MSE = (1-1)^2 = 0$.
For another example, say y_i= 1, z_i=1 and p_i=0.6. Then $CELoss_i = -log(0.6) = 0.737$ and $MSE = (1-1)^2 = 0$.
The second example needed more tuning, but if we used MSE, we would have no idea how the model performs. We would just know that it classifies correctly.
Thus, CE Loss would allow better visualization of how much wrong the model is.

Question 2: For a binary classification task with a deep neural network (containing at least one hidden layer) equipped with linear activation functions, which of the following loss functions guarantees a convex optimization problem? Justify your answer with a formal proof or a clear argument. (a) CE (b) MSE (c) Both (A) and (B) (d) None

Answer: As the activation is linear, the network is essentially a linear classifier. Thus CE Loss should provide a convex optimization problem. Following the logic in Question 1, as using CE Loss would allow us to see how much wrong the model is per example and take its average, we are guaranteed a total distance from true optima.
On the other hand, using MSE, we would only know if the example was correctly classified or not, thus for wrongly classified examples, there will not be a unified goal to reach towards to optimize the problem and as such, there would be several local minima.

Question 3: Dense Neural Network: Implement a feedforward neural network with dense layers only. Specify the number of hidden layers, neurons per layer, and activation functions. How will you preprocess the input images? Consider hyperparameter tuning strategies.

Answer:
Number of hidden layers= 2
Neurons per layer= Input= 1024, Hidden Layers= 256, Output= 10

Activation Function= ReLU
Preprocessing images:

- Using augmentation techniques like Cropping, Random Affine, Jittering etc.
- Transforming to grayscale, using normalization, flattening the images etc.

For this problem, the images were transformed to grayscale and were flattened.
Hyperparameter tuning strategies:

- Using GridSearch to select hyperparameters such as learning rate, optimizers, loss function, epochs, etc.
- Using techniques such as early stopping and scheduled learning rate
- Using regularization to limit errors

For this problem, the hyperparameters tuned were batch size, learning rate, number of neurons in the hidden layers and epochs. The code notebook can be found on the repository with the name Asg_3.

Question 4: Build a classifier for Street View House Numbers (SVHN) (Dataset) using pre-trained model weights from PyTorch. Try multiple models like LeNet-5, AlexNet, VGG, or ResNet(18, 50, 101). Compare performance and comment on why a particular model is well suited for the SVHN dataset.

Answer: The training was done on Google Colab GPUs, and the entire dataset was used. The training was done for 10 epochs. The training dataset was split into train and validation following a 75-25 ratio.
All the results can be found in the notebook Asg_4. The following results were obtained on each model:

- LeNet-5: As pre-trained weights were unavailable, the entire model was implemented by me. Images were converted to grayscale due to the model architecture.
  It achieved a training accuracy of 90%, a validation accuracy of 85% and a testing accuracy of 86%.
- AlexNet: Images were resized to 63x63 due to the model architecture. Only the final layer was trained.
  It achieved a training accuracy of 43%, a validation accuracy of 46% and a testing accuracy of 43%.
- VGG-16: No modifications to the images were made. Only the final layer was trained.
  It achieved a training accuracy of 34%, a validation accuracy of 41% and a testing accuracy of 42%.
- ResNet-18: No modifications to the images were made. Only the final layer was trained.
  All of its accuracies were 31%.

The reason for this performance may be due to how much the models scale to the comparatively smaller dataset. All these models except LeNet-5 were pre-trained on the huge ImageNet database, which means when using a very small SVHN dataset, there is a tendency of underfitting due to lack of data. Another reason why LeNEt-5 performs well maybe due to the fact that it is a small model and it was trained from scratch, allowing it to scale well to the SVHN dataset.
Additionally, all models were able to classify the digit '1' pretty well.