Deep Learning DSE316
Assignment 1

Suhas Bhat
19310

March 2024

Question 1

The Cross Entropy loss function works better than the Mean Squared Error loss function for logistic regression problems as it aligns with the probabilistic nature of the model's output and ensures an optimization landscape that is more conducive to efficient learning.

Logistic regression models the probability that an input belongs to a particular class. Cross Entropy is a measure of how well the predicted probability distribution aligns with the actual distribution. It directly measures the difference between two probability distributions, making it more appropriate for tasks like classification where predictions are probabilistic. When the predicted probability value is far from the actual class label, the value of the loss function is very high and the gradient is steep, which results in quicker training.

MSE as a loss function will also work for logistic regression in the sense that it tries to ensure the prediction aligns with the actual class label. However, it does not penalise wrong predictions as much as CE does, and the gradient is also less steep, resulting in slower training.

Question 2

For a deep neural network (DNN) used for a binary classification task, if all activation functions are linear, the network effectively becomes a single linear model, because the composition of linear functions is itself a linear function. In this context, we shall examine if the CE and MSE loss functions give a convex optimization problem. We optimize the loss function with respect to the neural network parameters $w = (b, w_1, w_2 \ldots, w_n)$, assuming the input vectors are of the augmented form $x = (1, x_1, x_2, \ldots, x_n)$. For a function to be convex, its second derivative must be non-negative. So we shall examine the second derivative of CE and MSE loss functions.

For CE:

$$\begin{aligned}
L &= -\left(y \log(wx) + (1-y)\log(1-wx)\right) \\
\frac{\partial L}{\partial w} &= -\left[\frac{yx}{wx} + \frac{(1-y)(-x)}{(1-wx)}\right] \\
&= \frac{-y(1-wx)}{w(1-wx)} + \frac{w(1-y)}{w(1-wx)} \\
&= \frac{-y + wyx + wx - wxy}{w(1-wx)} \\
&= \frac{-y + wx}{w - w^2x}.
\end{aligned}$$

$$\begin{aligned}\frac{\partial^2 L}{\partial w^2} &= \frac{x(w-w^2x)-(-y+wx)(1-2wx).}{(w-w^2x)^2} \\ &= \frac{(y-wx)(1-2wx)+wx-w^2x^2}{(w-w^2x)^2} \\ &= \frac{y-2ywx-wx+2w^2x^2+wx-w^2x^2}{(w-w^2x)^2} \\ &= \frac{y-2wyx+w^2x^2}{(w-w^2x)^2}\end{aligned}$$

The second derivative w.r.t weights is not necessarily non-negative and hence it does not guarantee a convex optimization problem.

For MSE:

$$L = (y-wx)^2$$
$$\frac{\partial L}{\partial w} = -2x(y-wx)$$
$$\frac{\partial^2 L}{\partial w^2} = 2x^2 \geq 0$$

Therefore the MSE loss guarantees a convex optimization problem as its second derivative w.r.t DNN parameters is non-negative.

Question 3

The neural network architecture consists of an input layer, one or more hidden layers, and an output layer. The number of hidden layers and activation functions can be customized using hyperparameters. The input layer has 784 neurons corresponding to a 28x28 grayscale image, and the output layer has 3 neurons assuming a multi-class classification problem.

Hyperparameters

The following hyperparameters are considered in the implementation:

- **Number of Hidden Layers (*n_hidden_layers*):** Determines the depth of the neural network.
- **Activation Function (*activation*):** Specifies the activation function for each layer.
- **Learning Rate (*learning_rate*):** Rate at which the model parameters are updated during training.
- **Batch Size (*batch_size*):** Number of samples processed in each iteration.
- **Number of Epochs (*epochs*):** Total number of passes through the entire dataset during training.
- **Weight Decay (*weight_decay*):** Regularization parameter to prevent overfitting.
- **Optimizer (*optimizer*):** Optimization algorithm used during training.
- **Data Augmentation (*augmentation*):** Techniques to augment the training dataset.

Model Creation and Training

The model creation involves defining the neural network architecture based on the specified number of hidden layers and activation functions. The model is then trained using the selected optimizer, loss function, and hyperparameters such as learning rate and weight decay. The training process iterates through multiple epochs, updating the model parameters using backpropagation.

Preprocessing and Inference

Before feeding the input images to the model, preprocessing steps are applied, including resizing, converting to grayscale, normalization, and flattening. Additionally, data augmentation techniques such as flipping, rotation, or cropping can be used to increase the diversity of the training dataset. After preprocessing, the model performs inference to predict the probabilities of each class.

The provided code demonstrates the implementation of a neural network model with customizable hyperparameters and training options using PyTorch. By adjusting the architecture, optimization algorithm, and preprocessing techniques, the model can be tailored for different classification tasks. Further experimentation with hyperparameters and training strategies may lead to improved performance on specific datasets.

Question 4

The percentage accuracy rates for different models are as below:

- ShuffleNet - 77.5
- LeNet - 89
- AlexNet - 88.7
- ResNet18 - 95.3
- ResNet50 - 94
- ResNet101 - 96
- VGG16 - 95.4

  ResNet101 performs the best because its size is the highest, and its skip connections allow the model to backpropagate faster.