Deep Learning (616) Assignment 1

Prachi Chachondhia 2311003

27th March 2024

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. [3 Marks] [Theory]

In logistic regression, which is widely used for binary and extendable to multi-class classification tasks, Cross Entropy Loss is more effective than Mean Squared Error (MSE). This preference stems from logistic regression's reliance on a logistic or sigmoid function to bound outputs between 0 and 1, making it ideal for predicting probabilities of classification.

Cross Entropy Loss excels in evaluating classification models where outputs are probabilities. As the predicted probability deviates from the actual label, the performance impact of Cross Entropy Loss becomes more pronounced, thereby enhancing its suitability for probabilistic output models.

On the other hand, Mean Squared Error, while a staple in regression problems, is less optimal in probabilistic contexts like logistic regression. Its application may lead to suboptimal results due to its different optimization objectives compared to Cross Entropy Loss.

Significance of Cross Entropy Loss

1. It adeptly manages probabilities by quantifying the divergence between predicted probabilities and the actual label distribution, aligning perfectly with logistic regression's probabilistic nature.
2. The gradient of Cross Entropy Loss relative to the model inputs remains stable, reducing the risk of vanishing or exploding gradients, especially with incorrect predictions.
3. It facilitates a more straightforward trajectory towards the optimal training solution, making the model training process more efficient.

Training Process Implications

Choosing Cross Entropy Loss influences the logistic regression model's training process in several critical ways:

1. **Faster Convergence:** Models trained with Cross Entropy Loss typically reach optimal solutions more quickly due to the loss function's alignment with the model's probabilistic outputs.
2. **Enhanced Learning:** This loss function ensures the model learns effectively from challenging examples, bolstering overall performance.
3. **Prevention of Learning Plateaus:** Cross Entropy Loss mitigates learning slowdowns that can occur with MSE when predictions are close to 0 or 1 but incorrect, ensuring consistent learning progress.

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 [3 Marks] [Theory]

In the context of a binary classification task with linear activation functions in a deep neural network, neither Cross Entropy Loss nor Mean Squared Error can guarantee a convex optimization problem.

While MSE is convex with respect to its inputs, the combination of MSE with the outputs of a deep neural network, which result from complex transformations of inputs and weights, does not ensure convexity with respect to the network's parameters.

Similarly, although Cross Entropy Loss is commonly used for classification tasks and is typically paired with non-linear outputs such as the sigmoid function for binary classification, it does not necessarily guarantee convexity when applied to neural network outputs with linear activation functions.

In conclusion, neither Cross Entropy Loss nor Mean Squared Error guarantees a convex optimization problem for a deep neural network with linear activation functions in the context of a binary classification task.

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. [2 for implementation and 3 for explanation]

Report: Hyperparameter Tuning for Convolutional Neural Network on MNIST Dataset

1. ### 3.1 Objective

   The main goal of this project was to create and optimize a convolutional neural network (CNN) to classify handwritten digits from the MNIST dataset. We focused on experimenting with hyperparameter tuning techniques and preprocessing methods to enhance the model's performance.

   ### 3.2 Dataset

   The MNIST dataset consists of grayscale images representing handwritten digits ranging from 0 to 9. It's commonly used for digit classification tasks and comprises 60,000 training images and 10,000 test images.

   ### 3.3 Preprocessing

   (a) **Normalization**: Pixel values of input images were scaled to fit within the range [0, 1].

   (b) **Reshaping**: Input images were reshaped to include a channel dimension, making them suitable for CNN processing.

3.4 Convolutional Neural Network Architecture

The CNN architecture included multiple convolutional and pooling layers followed by fully connected layers for classification. The architecture was defined as follows:

- Convolutional layer: 32 filters, kernel size (3, 3), ReLU activation.
- MaxPooling layer: Pool size (2, 2).
- Convolutional layer: 64 filters, kernel size (3, 3), ReLU activation.
- MaxPooling layer: Pool size (2, 2).
- Flatten layer.
- Dense layer: 128 neurons
- Output layer: 10 neurons (softmax activation).

3.5 Hyperparameters Tuned

(a) Number of neurons in the dense layer: 32, 64, 128.
(b) Activation function for the dense layer: ReLU, tanh, sigmoid.

3.6 Hyperparameter Tuning Strategy

We utilized grid search with 3-fold cross-validation using scikit-learn's GridSearchCV. This explored various combinations of neurons in the dense layer and activation functions, with model performance assessed based on mean validation accuracy across folds.

3.7 Results

- The best model configuration identified by grid search achieved an overall validation accuracy of 96.52%, utilizing ReLU activation with 128 neurons in the dense layer.
- ReLU consistently outperformed tanh and sigmoid across different numbers of dense layer neurons.
- Increasing the number of neurons generally improved performance, with the highest accuracy achieved using 128 neurons.
- The test accuracy of the best model on the unseen test set was 97.57%.

3.8 Conclusion

Preprocessing techniques like normalization and reshaping, coupled with hyperparameter tuning, significantly improved the CNN's performance on MNIST. The chosen model showed competitive accuracy, with a test accuracy of 97.57% demonstrating its effectiveness in classifying handwritten digits.

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 comments on why a particular model is well suited for the SVHN dataset. (You can use a subset of the dataset (25%) in case you do not have enough compute.) [4 Marks] [Code and Report]

Report: Comparison of Pretrained Models for SVHN Dataset Classification

1. ### 4.1 Introduction

The Street View House Numbers (SVHN) dataset comprises images of house numbers from Google Street View, requiring classification into their respective digit labels. This report evaluates the performance of several pre-trained models: LeNet-5, AlexNet, VGG, ResNet-18, ResNet-50, and ResNet-101 on the SVHN dataset.

4.2 Experimental Setup

Using PyTorch and pre-trained models from torchvision, we worked with the SVHN dataset. Due to computational constraints, we utilized a subset containing 25% of the data for training and testing.

4.3 Performance Metrics

We assessed the models' performance using the following metrics:

- Test Accuracy: Percentage of correctly classified images in the test set.
- Precision: Ability of the classifier not to label a negative sample as positive.
- Recall: Ability of the classifier to find all positive samples.
- F1-score: Harmonic mean of precision and recall, providing a balance between the two metrics.

4.4 Results

Model	Test Accuracy	Precision	Recall	F1-score
LeNet-5	73.89%	0.7044	0.7044	0.7081
VGG-16	19.59%	0.0196	0.1000	0.0328
ResNet-18	89.33%	0.8877	0.8837	0.8844
ResNet-50	87.86%	0.8766	0.8702	0.8714
ResNet-101	83.98%	0.8353	0.8254	0.8282

Table 1: Performance Metrics of Pretrained Models on SVHN Dataset

4.5 Analysis

- **LeNet-5**: Despite showing improvement over epochs, LeNet-5 achieved a lower test accuracy of 73.89%, indicating limited capacity for learning complex features.
- **VGG-16**:VGG-16 demonstrated a very low test accuracy of 19.59%, suggesting significant issues in learning and generalization.
- **ResNet-18 and ResNet-50**: With an impressive test accuracy of 89.33%, ResNet-18 outperformed both LeNet-5 and VGG-16. The model showed consistent improvement over epochs and effectively learned complex features.Similar to ResNet-18, ResNet-50 exhibited strong performance with a test accuracy of 87.86%. It showcased steady improvement over epochs and handled deeper architecture effectively.
- **ResNet-101**: Despite being a deeper architecture, ResNet-101 achieved a test accuracy of 83.98%, slightly lower than ResNet-18 and ResNet-50. This could be due to potential overfitting from increased complexity.

4.6 Conclusion

Among the tested models, ResNet-18 and ResNet-50 stood out for their exceptional performance on the SVHN dataset. Both models demonstrated robustness in learning complex features and showed steady improvement over epochs. VGG-16, on the other hand, performed poorly, indicating challenges in learning the dataset's features effectively. LeNet-5, while a classic architecture, lacked the capacity to achieve competitive performance on this dataset. Therefore, ResNet-18 emerges as a strong candidate considering both accuracy and model complexity, closely followed by ResNet-50. These models showcase the importance of architecture depth and effective feature propagation in achieving high performance on complex datasets like SVHN.