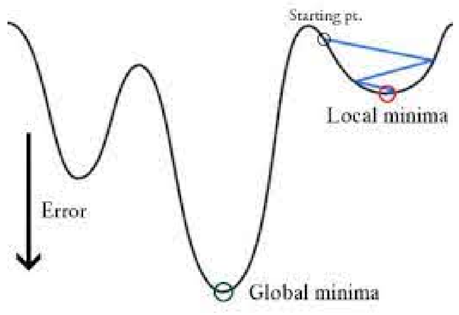
DL Assignment

Singh Arjita

21264

1. Cross Entropy Loss is preferred for classification tasks, especially in neural network classification with SoftMax output node activation, due to its ability to handle the nature of classification problems and ensure convex optimization during training. Unlike Mean Squared Error (MSE), which is commonly used for regression tasks, Cross Entropy Loss guarantees a more suitable optimization landscape for classification problems.

   In logistic regression, the predicted output $\hat{Y}$i is a nonlinear function of the input, represented by $\hat{Y} = 1/(1 + e\ \hat{}{-}z)$. When MSE is applied to this nonlinear function, it results in a non-convex loss function as follows:

   This non-convexity introduces complications during optimization using gradient descent, making it challenging to find the global minima efficiently. The Mean Squared Error Loss is given as:

$$J = \frac{\sum_{i=1}^{n} (\hat{Y}_i - Y_i)^2}{n}$$

   In classification problems, where the target values are typically binary (0 or 1) and the predicted output is also between 0-1, squaring the difference in MSE can lead to values that are difficult to interpret and manage, especially in terms of precision and storage.

   Binary cross entropy loss is given by:

$$-\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

   Binary Cross-Entropy / Log Loss

   Here yi represents the actual class and log(p(yi)is the probability of that class. The benefits of using logarithm here are that when the predicted probability (x-axis) is close to the actual class (0 or 1), the loss is less and when the predicted probability is close to the other class (1 or 0 respectively), loss approaches infinity. Using cross entropy loss thus will lead to a convex problem where you might find the optimum solution.

2. (b) MSE (Mean Squared Error)

   A deep neural network with only linear activation functions throughout all layers acts as a single, large linear transformation. This linear transformation maps the input to the output with a simple scaling and bias adjustment. Consequently, the output range of such a network will be the same as the range of possible values after the first linear layer, which is negative infinity to positive infinity.

   With an output range of negative infinity to positive infinity, CE (Cross-Entropy) might not always guarantee convexity. The challenge arises when the predicted probability (t) approaches 0 or 1 (due to the unbounded output from linear activations). Taking the logarithm of values very close to 0 or 1 approaches negative or positive infinity, respectively. Multiplying these infinite values by y and (1-y) can lead to undefined or non-numeric outputs, breaking the continuity and convexity of the loss function.

   MSE Remains Convex:

   MSE (Mean Squared Error) still maintains convexity even with linear activations. Squaring a linear function always results in a quadratic function. As mentioned earlier, quadratic functions are convex when the leading coefficient is non-negative. In MSE, the leading coefficient is always 1 (positive), ensuring convexity regardless of the unbounded output range.

   The Hessian matrix is a square matrix of second-order partial derivatives of a scalar-valued function. In the context of optimization, the Hessian matrix is a key tool for determining convexity.

   For a function to be convex, its Hessian matrix must be positive semidefinite for all points in its domain. This condition ensures that the function curves upward and doesn't have any local maxima.

   In the case of MSE (Mean Squared Error), the loss function is a quadratic function. Quadratic functions are always convex, and their Hessian matrices are constant and positive definite. Therefore, the Hessian matrix of MSE is positive definite, indicating convexity.

   MSE guarantees convex optimization because its Hessian matrix is positive definite, ensuring convexity and absence of local minima.

3. Preprocessing:

   Normalization is crucial for scaling pixel values to a specific range, typically between 0 and 1, ensuring each feature contributes proportionately to the learning process in neural networks. It aids in stabilizing and accelerating training.
   Standardization, on the other hand, achieves zero mean and unit variance for pixel values, beneficial when input data exhibits varying scales or units, though normalization is commonly preferred for neural networks.
   Resizing becomes imperative when input images exhibit size variations, particularly in convolutional neural networks

(CNNs) requiring fixed input sizes. Cropping selectively removes irrelevant background or isolates specific objects, commonly employed in object detection tasks.
Augmentation diversifies training data by applying random transformations, valuable for small datasets or models susceptible to overfitting.
Histogram equalization enhances contrast in images with poor lighting conditions. Noise reduction methods like Gaussian blurring eliminate unwanted variations caused by noise.
Data augmentation introduces domain-specific transformations, bolstering model robustness, especially pertinent for small datasets or models prone to overfitting.
The selection of preprocessing techniques hinges on dataset attributes, network requisites, and task specifics, necessitating experimentation for optimal performance.

In this implementation, normalization is utilized to preprocess input images. Each pixel value is divided by 255 to scale them between 0 and 1, fostering stabilized training and potentially expediting convergence.

Hyperparameter Tuning Strategies:

Number of Hidden Layers and Neurons per Layer: We can use techniques like grid search or random search to explore different combinations and select the one that yields the best performance on a validation set.

Activation Functions: ReLU is preferred in hidden layers for its computational efficiency and ability to alleviate the vanishing gradient problem. However, it may lead to the "dying ReLU" issue when inputs are persistently negative, causing neurons to remain inactive.
Sigmoid is suitable for binary classification outputs due to its ability to squash values into the (0, 1) range, representing probabilities. Nonetheless, sigmoid can suffer from vanishing gradients and lack of zero-centering in hidden layers.
Tanh, akin to sigmoid, maps inputs to (-1, 1) range, overcoming the latter's non-zero-centeredness. Despite this, tanh shares similar drawbacks such as vanishing gradients, albeit to a lesser extent than sigmoid. Softmax, primarily used in output layers for multi-class classification, normalizes outputs to a probability distribution summing to 1.
Lastly, Leaky ReLU addresses the "dying ReLU" issue by allowing a small gradient for negative inputs, though it introduces additional hyperparameters.
In summary, the choice of activation function hinges on the specific requirements and characteristics of the task at hand.Experimentation with different activation functions can be done using grid search or random search.

Learning Rate and Optimizer: We can use techniques like learning rate schedules or adaptive learning rate methods like Adam optimizer to automatically adjust the learning rate during training. Learning rate scheduling involves dynamically adjusting the learning rate during training based on predefined schedules, such as exponential decay, step decay, or performance-based schedules. Learning rate scheduling can help improve convergence and prevent overshooting or oscillations in the optimization process.

Initialization of Weights: Weight initialization methods determine the initial values of the neural network weights. Proper initialization can help stabilize training and prevent vanishing or exploding gradients. Experimenting with different weight initialization methods can help improve convergence and performance.

Regularization: Regularization techniques, such as L1 regularization, L2 regularization, and dropout, help prevent overfitting by penalizing large weights or randomly dropping units during training. Tuning regularization parameters involves experimenting with different regularization strengths to find the optimal level of regularization for the given task and dataset.

Batch Size and Number of Epochs: These hyperparameters control how many samples are used in each iteration and how many times the entire dataset is passed through the network during training, respectively. These can be tuned to find a balance between convergence speed, memory efficiency and generalization performance by grid search.

By systematically tuning these hyperparameters and monitoring the model's performance on a separate validation set, we can effectively optimize the neural network for the given task.