Abstract

This research trained an MLP model to predict missing pixel values in altered photos using the FashionMNIST dataset. The model was improved using enhancing methods such batch normalisation layers, dropout layers, and various activation functions. The model's performance showed a significant improvement, with the maximum accuracy being attained by combining various enhancing strategies.

Introduction

The FashionMNIST dataset was used to produce a modified dataset for this project, in which the centre column of each image's pixels was set to zero and its actual values were retrieved and recorded in a CSV file for each picture. To forecast the missing values in the photos, we divided the data into training and testing sets and used a Multilayer Perceptron (MLP) model. Following that, we tested our model's accuracy on the test set and visualised the final images it predicted.

We used a variety of boosting strategies, including batch normalisation layers, dropout layers, multiple activation functions, and learning rate scheduling, to enhance the performance of our model. To further optimise our model, we additionally used L1 and L2 regularization, various weight initialization strategies, and early stopping techniques. We also used other optimizers and evaluated their effectiveness.

Our findings showed that applying these boosting methods considerably enhanced the functionality of our MLP model. With a learning rate of 0.001, an amalgam of batch normalisation layers, dropout layers, ReLU activation functions, and Adam optimizer produced the best accuracy. These results demonstrate how enhancing approaches can enhance the functionality of neural network models.

1 LeakyReLU MLP

This code creates a neural network to forecast the values of missing pixels in FashionMNIST photos. Two hidden layers that each use the LeakyReLU activation function make up the neural network design, which is followed by an output layer. The model is trained on a train set, and mean squared error loss is used to assess the model's performance on a test set.

Stochastic gradient descent (SGD), with a learning rate of 0.01 and momentum of 0.9, is the optimizer utilised for training. The best performing model is saved based on the test loss after the model has been trained for 10 epochs.

Following training, a sample image from the test set is used to evaluate the model, and for visual comparison, the original image and the image with the projected missing pixel values are presented side by side.

In conclusion, using the FashionMNIST dataset, this code demonstrates how to create a basic neural network to forecast missing pixel values in photos. By utilising various enhancement methods, such as batch normalization, dropout layers, L1/L2 regularization, different weight initialization, and early stopping, the performance of the model can be further enhanced.

2 normalization Batch

Deep learning uses batch normalization as a method to enhance neural network performance and training stability. The main idea is to prevent the "internal co-variate shift" problem by normalizing the inputs to each layer of a network to have a zero mean and unit variance. As a result, each layer's inputs have a constant distribution, which makes it simpler for the network to learn.

The mean and standard deviation of the inputs to the batch normalization layers are calculated for each mini-batch of data during training. The inputs to the layer are then normalized using these statistics. The network can learn to undo the normalization if necessary by learning how to change the normalised inputs using a learned scaling and shifting operation.

The running mean and standard deviation calculated during training are used by the batch normalization layers to normalize the inputs at test time. This indicates that the network consistently generates outputs from inputs with various distributions.

The training loop of the code shows the effects of batch normalization. As the number of epochs rises, it is evident that the network is able to achieve lower training and testing losses. This shows that batch normalization improves the network's ability to learn compared to no normalization. The image generated at the end of the code also exhibits the network's output on a sample image, proving the network's capacity to generate visually compelling results.

3 Dropout

A regularization method called dropout is employed in neural networks to avoid overfitting. During each training iteration, a random fraction of the nodes in a layer are dropped out (set to zero). This pushes the nodes to acquire increasingly robust and practical capabilities while preventing the network from becoming overly dependent on any one node.

We may change the training loop to train two different versions of the model, one with a droprate of 0.1 and another with a droprate of 0.5, to compare how well the model performs with and without dropout.

4 activation functions (Sigmoid)

A common activation function used in neural networks is the sigmoid. For binary classification issues, it takes any real-valued number and compresses it into a range between 0 and 1. The sigmoid function is defined as:

$$f(x) = \frac{1}{1 + e^{-x}}$$

The vanishing gradient problem, in which the gradient gets very small when the input value is either large or very little, is one of the disadvantages of utilising sigmoid activation. Due to this, the neural network may find it challenging to draw conclusions from the data, which may slow down training or possibly cause convergence problems.

You can train many neural networks with various activation functions and compare their accuracy and loss on a validation set to assess how well they perform. It's crucial to remember that an activation function's effectiveness can vary based on the precise task and dataset, so it's a good idea to try out many approaches to find the optimal one for your particular issue. Here are the results:

5 activation functions (ReLU)

Rectified Linear Unit, sometimes known as ReLU, is a well-liked activation function employed in deep learning. It is defined as:

$$f(x) = max(0, x)$$

The ReLU function has been demonstrated to be successful in numerous deep learning models, is computationally economical, and is simple to optimise. The vanishing gradient problem, which can happen when training very deep neural networks, is also known to be helped by it.

Due to its capacity to prevent the vanishing gradient issue and generate sparse activations, ReLU is anticipated to perform better than sigmoid.

After training is complete, we may assess the model on the test set and visually analyse the model's performance by plotting a sample output. It's crucial to analyse the performance parameters of several models, such as loss and accuracy, to choose which activation function would solve the particular problem the best.

6 Learning Rate

Using the LeakyReLU activation function, the code used in this section creates a neural network that is trained on the MNIST dataset. The loss function is Mean Squared Error (MSE), and the optimizer is Stochastic Gradient Descent (SGD) with a learning rate of 0.01 and momentum of 0.9. A learning rate scheduler is used to change the learning rate by multiplying it by a factor of 0.1 after every 5 epochs during the training, which lasts for 10 epochs.

The neural network is taught using training data, and the test loss is calculated using validation data, during training. To enhance the model's performance, the scheduler is used to modify the learning rate. If the validation loss lowers with each epoch, the model is saved.

The model is assessed using a set of test data at the conclusion of the training phase, and the results are shown next to the original image. This is a graphic illustration of how effectively the model can recreate the supplied data.

The learning rate is a crucial hyperparameter that has a big impact on how well the neural network performs. The optimizer might exceed the ideal weights if the learning rate is too high, which would cause the model to diverge. The model might converge too slowly or become stuck in a local minimum, however, if the learning rate is too low.

A learning rate scheduler is employed in this code to modify the learning rate as training progresses. After every 5 epochs, the scheduler cuts the learning rate by a factor of 0.1. This enhances the model's performance by allowing the model to converge more gradually as it approaches the ideal weights.

Overall, this code shows how the learning rate can affect a neural network's performance and how a learning rate scheduler can be used to change the learning rate to enhance the model's performance.

7 L1, L2 Regularization

7.1 L1

L1 regularisation is a machine learning approach used to lessen model overfitting. When a model learns training data too thoroughly and becomes extremely complicated, overfitting occurs, which results in poor generalisation to new data. The model is prompted to learn more straightforward representations of the data by the addition of a penalty term via L1 regularisation to the loss function.

The penalty term in L1 regularisation is the model's weight matrix's L1 norm times a hyperparameter. The total absolute values of a matrix's elements make up its L1 norm. The amount of features utilised in the model is effectively decreased by encouraging the model to learn weight matrices with lots of zero-valued members by including this penalty term in the loss function.

You were able to enhance the generalisation performance of your neural network by employing L1 regularization, as seen by the decline in test loss throughout training.

7.2 L2

Overfitting is a frequent issue in machine learning models, particularly neural networks. When a model is taught to match training data so well that it becomes overly specialized to that data and performs badly on new, untainted data, overfitting has taken place. L2 regularization is one of various methods that can be used to avoid overfitting. The use of L2 regularization in a neural network to avoid overfitting will be discussed in this study.

In this part, we trained a straightforward neural network with two hidden layers and one output layer. A hidden layer with 256 neurons, a second hidden layer with 64 neurons, and an output layer with 28 neurons made up the neural network design. With a learning rate of 0.01 and momentum of 0.9, we employed the mean squared error (MSE) loss function and Stochastic Gradient Descent (SGD) optimizer. The neural network was trained with a batch size of 128 over the course of 10 epochs. With a value of 1e-5, we employed L2 regularization (weight decay).

We used the dataset to train the neural network, and a test dataset to assess its performance. The outcomes demonstrate that L2 regularization enhanced the model's ability to perform on unobserved data and helped prevent overfitting. After 10 epochs, the training loss dropped from 0.1163 to 0.0501 and the test loss from 0.1099 to 0.0491. The fact that the validation loss dropped from infinite to 0.0491 shows that L2 regularization successfully suppressed overfitting.

L2 regularization is a straightforward but efficient method for guarding against overfitting in neural networks. The loss function is given a penalty term that promotes the model to have low weights. This penalty term promotes the model to learn more generalizable features that may be applied to fresh, unexplored data and helps prevent the model from being overly particular to the training data. You can alter the weight decay regularization term's value to strike a compromise between minimising overfitting and avoiding underfitting the training set of data.

8 weight initialization

Numerous machine learning issues have been successfully solved using neural networks. Initializing the weights is a crucial step in the training of neural networks because it can significantly affect the network's performance. In this section, we'll talk about how to use a neural network trained on the dataset with specific weight initialization.

The benefit of using unique weight initialization is that it can enhance the neural network's convergence during training. Poor initialization can cause gradients to vanish or explode, which can make network convergence challenging or impossible. We may make sure that the weights are initialized in a way that encourages stable training by employing a bespoke weight initialization function.

The mean squared error (MSE) loss function and the stochastic gradient descent (SGD) optimizer are used to train the neural network. The average training and test losses are computed and printed after each of the training loop's 10 iterations.

The validation loss in this code continually drops as training progresses, showing that the network is effectively learning to generalize to new data. Additionally, if the validation loss has decreased from the previous epoch, the model is preserved after each one. Finally, a small sample of test data is predicted using the saved model, and the original and anticipated images are shown side by side.

This code highlights the significance of accurate weight initialization and how it can lead to more effective training and generalization. .

9 Early stopping

Deep learning uses early stopping as a regularisation method to avoid overfitting. Early stopping's main tenet is to stop training the model as soon as the model's performance on a validation set begins to deteriorate. This indicates that the model has learned everything it can from the training data and that continuing to train it could result in overfitting.

Early stopping is implemented in the provided code using a counter and a patience parameter. When the validation loss does not decrease, the counter variable is initialized to 0 and increased. The model should wait a certain amount of epochs if the validation loss does not decrease, according to the patience parameter. The best model (i.e., the one with the lowest validation loss) is saved if the validation loss does not improve after the predetermined number of epochs.

A straightforward feedforward neural network with two hidden layers makes up the neural network in the code. Stochastic gradient descent (SGD) is the optimizer employed, with a learning rate of 0.01 and momentum of 0.9. The mean squared error (MSE) loss function is employed.

Ten training epochs are completed, and following each epoch, the training and test losses are printed. Based on the test loss, the best model is saved, and the saved model is then utilised to create an output image by sending a test image over the network.

The outcomes demonstrate that as the number of epochs rises, the model performs better, with both training and test losses declining over time. Up to epoch 9, when there is no more improvement, the model's performance on the test set gets better every time. The best model is then saved after the training is terminated.

10 Utilizing different optimizers

The training and validation loss for three different optimisation methods, namely SGD, Adam, and RMSprop, are shown in this section's results. For each optimizer, the training and validation losses are assessed over ten epochs.

The outcomes demonstrate that the loss across the ten epochs has been successfully decreased by all three optimisation strategies. But compared to SGD and RMSprop, Adam has produced the greatest results because it has the lowest loss values for both the training and validation sets. Adam also has the quickest convergence rate, which means the best result was attained in a smaller number of epochs.

SGD got good results, although it took longer epochs than Adam to get to the lowest loss. Although it generated variations in the loss curve, the momentum factor in the SGD optimizer helped to reduce the loss.

Out of the three optimizers, RMSprop had the worst outcomes. Both Adam and SGD excelled it in terms of loss reduction and convergence rate.

In comparison to SGD and RMSprop, Adam has, in our opinion, obtained the finest outcomes and the quickest pace of convergence. The optimal optimizer for your task will depend on the particular problem at hand, therefore choosing one depends on experimentation with several optimizers.