Jean-Marc Rugomboka-Mahoro

MNIST Character Recognition

Important findings from the implementation

An important finding is actually the difference between iterations and epochs. I didn't understand what this code was doing with iterations but after doing more research I found that epochs go over all the training examples while iterations just go over a batch size of the training examples. Another important finding from the implementation is that increasing the iterations is not the best way to go if you don't also add more data, like in this case adding some diagonal, some shifted down, left, right, and up digits would have had much more effect if I then increased iterations rather than only increasing iterations. **[2]**

Why I choose an idea to implement

I choose to implement an idea of increasing the iterations. As a side note, I used iterations instead of epochs like we did in labs. One epoch would be the forward and backward pass of all the training examples. An iteration in contrary is to do one forward and backward pass over a batch size (a batch size simply is dividing your data into smaller samples instead of going over all your training examples). **[3]** I increased the iterations by twice as many, from 500 iterations to 1000 iterations. The reason I choose to do this is because I wanted to increase the accuracy on the training dataset, this solution gave me a guarantee to succeed.

I learned that increasing the iterations has disadvantages and is actually inefficient compared to another method that I discovered while working on this problem. Two of the disadvantages that I noticed included the fact that increasing iterations on a bigger problem with a bigger dataset and bigger batch size would be time consuming. Another disadvantage is that increasing iterations alone may not be effective enough, it could be that the dataset requires more variety for the accuracy to increase on weirdly drawn digits. **[2]** There are also advantages to increasing the iterations, in this case the accuracy did increase by 3% which in the end means that my mission was accomplished. **[4]** Another advantage is that from the source code **[1]** to my code you will be able to see that it was a simple solution to come up with.

Problem explained to a 7th-grade student

Draw a number from 0 to 9 on a piece of paper, now imagine dataset of thousands of such drawn numbers. A computer cannot read a drawing so I have a program that can guess what digit is drawn. All we have to do is show the program many of the examples in the dataset. This improves the amount of time the program will guess right. This program is good, but I want it to guess the right answer more often than it currently does. To do this I simply showed the program more examples of the dataset.

Idea explained to my friend

My program can predict what number between 0 to 9 has been drawn. We show the program examples of the dataset in order for it to be trained. After training my program, it was able to guess digits between 0 to 9 at an accuracy of 84%. **[4]** I was not satisfied with this number and wanted to increase it, so my idea was to show it more examples of the dataset to the program to train it more. This should increase the accuracy of the.

Overall performance

The overall performance of my program is simply that it works. I wanted to increase the accuracy and that's what I did. The program had an accuracy of 84% which was great, but after the iteration increase it was 87% which is better. **[4]** I am not satisfied with the increase of roughly a whole minute for just 3% accuracy, but there is another idea that may change this. This idea may increase the accuracy by much more for the same amount of time increase, I will talk about this idea later in this paper.

One of the most striking success is the following picture:

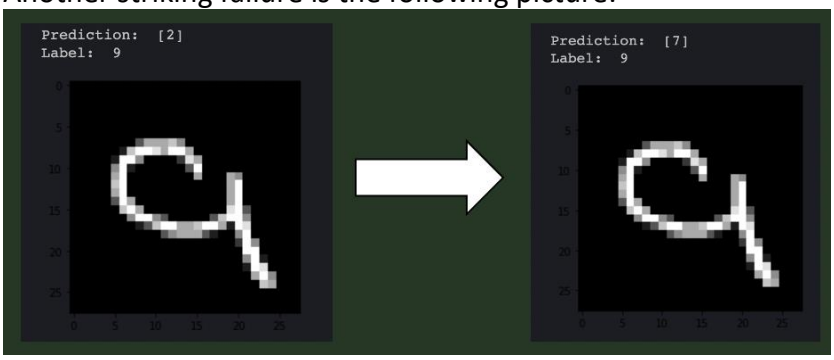
The reason this is one of the most striking success is because the label 2 has an image of a drawn 2, but the 2 is not drawn well. For all I know this could be the letter a to the human eye, but obviously our perception options are only numbers from 0 to 9. At 84% accuracy based on the training data, we predict for the 2 to be a 0 but the 87% accuracy surprisingly predicts the correct answer.

Another striking success is the following picture:

The label 4 has an image of a drawn 4. This 4, like the 2 in the picture before this, is also not drawn well. The first line on the 4 is a bit longer. The 84% accuracy predicts a 9 which in my opinion is not too far off as a guess. The 87% accuracy correctly predicts a 4. The reason this is a striking success is because just like in the image before these badly drawn images were wrongly predicted at an 84% accuracy and then correctly predicted after we added more iterations to get an accuracy of 87%.
Based on the fact that 2/8 predictions were corrected into a right answer makes this a successful idea.

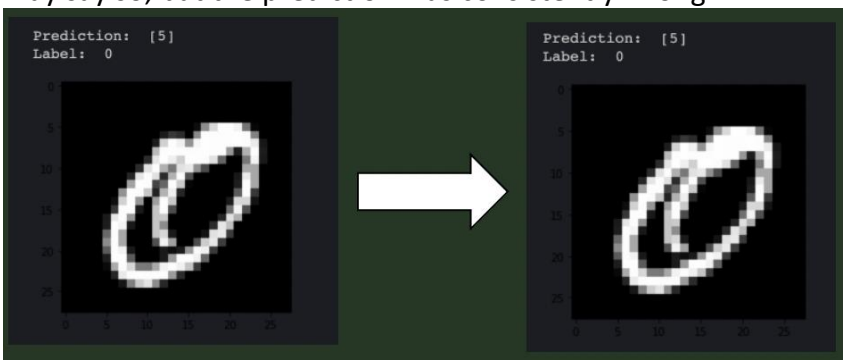
The following picture is a striking failure.

The reason for this striking failure is because at the 84% accuracy we predict a number of 4 on a label of 5. Looking at the image makes this to be my greatest failure in my opinion since the 5 doesn't look similar to any 4 I've ever seen drawn. After the increased iterations to get an accuracy of 87%, the prediction is 6 on the same label of 5. This makes sense because the 5 is nearly drawn like a 6 to my human eye. The answer is still wrong in the improved version making these one of my most striking failures.
Another striking failure is the following picture:



The label 9 for the 84% accuracy predicts a 2 which makes no sense at all to my human eye. The improvement to an 87% accuracy changed the prediction to a 7 which to me personally is a bit better than a prediction of 2 but still a definite wrong answer. The reason I picked these 2 images is because my idea to increase iterations in order to increase accuracy did not work on them. We ended up changing the prediction, possible to a digit closer that makes more sense to the human eye, but still a wrong prediction.

The following striking failure is a failure because to the human eye it's a clearly drawn digit if I may say so, but the prediction was consistently wrong.



The program predicts a 5 for a label of 0 at an 84% accuracy and after upgrading to the 87% accuracy, the program makes the same prediction as before, another prediction of 5. My theory on why the program is not even able to correctly predict a decently drawn digit of 0 is because it's drawn crooked/diagonally. I will go into more detail on this theory.

Other ideas and the reason I didn't choose them

My idea was simply to add more iterations to increase the accuracy which did work, but it possibly wasn't as efficient as another idea that I had. I thought about adding more variety to the dataset and including digits that are weirdly written, some shifted by a certain number of pixels at either side, and some diagonally drawn digits. This may have significantly increased the accuracy to even get the weirdly drawn numbers correct. A perfect example of this is the last picture of the striking failures, we predicted 5 on a label of 0 for both 84% and 87% accuracy. This may have happened because my dataset did not have enough example for diagonally drawn images, so just increasing iterations on a dataset that is flawed won't change much on the prediction results for those particular images. This may have been prevented if I had more variety in the dataset. **[2]** The reason I did not choose this idea is because I'd need to create a large dataset of shifted and crooked numbers and this may have taken too much time.

Summary

After increasing the iteration and only increasing the accuracy by 3% while taking a minute longer **[4],** I came to a conclusion that even though my solution worked and I succeeded in increasing the accuracy, it wasn't efficient enough. If my idea was mixed with an expanded dataset that had digits shifted and diagonal, then this method may have had more of an effect than a roughly 3% increase on the accuracy.

Something to remember for future students who would do work on a neural network and want to increase the accuracy of their training sets, if you are to increase iterations to improve the accuracy make sure to also improve the dataset so more varieties will be tested. This should result in a more efficient and better result.

In the future I'd love to work on more data sets than the MNIST and apply the knowledge that I've gained throughout this project.

References

[1] Zhang, S. (2020). Simple MNIST NN from scratch
https://www.kaggle.com/wwsalmon/simple-mnist-nn-from-scratch-numpy-no-tf-keras?select=test.csv
[2] Panchal, A. (2020). Machine Learning: Improving Classification accuracy on MNIST using Data Augmentation
https://towardsdatascience.com/improving-accuracy-on-mnist-using-data-augmentation-b5c38eb5a903
[3] Sharma, S. (2017). Epoch vs Batch Size vs Iterations
https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9
[4] Source code (separate file)
[5] Haddadin, T. (2018). Reading CSV Files into Jupyter Notebook using Pandas?
https://www.kaggle.com/questions-and-answers/67295
[6] Gupta, T. (2017). Deep Learning: Feedforward Neural Network
https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7
[7] Hebergementwebs. (2021). How to perform MNIST digit recognition with multilayer neural Network
https://www.hebergementwebs.com/news/how-to-perform-mnist-digit-recognition-with-multilayer-neural-network
[8] Brownlee, J. (2019). How to Develop a CNN for MNIST Handwritten Digit Classification
https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/