Programming and Mathematics for AI Coursework

Task 3 and 4

By Muhammad Faaiz Ansari

Table of Contents

-Task 3 description

-task 3 parts

-task 3 results

-Task 4 description

-task 4 parts

-task 4 results

-References

-Link to Github repo: https://github.com/lmagFaaiz/IN3063

Task3 Description:

The third task is about classifying Fashion-MNIST dataset. You can use other API's/libraries for loading the dataset, but not for the neural network implementation. The point of this task is to develop a multi-layer neural network for classification using mostly the numpy.

-Implement sigmoid and Relu layers (with forward and backward pass), Implement a softmax output layer:

> -I first defined each of the 3 layer types as methods in the DeepNeuralNetwork class. I then made them all work with both forward and backward propagation in my forward pass method and backward pass methods in my code.

-Implement a fully parameterizable neural network (number and types of layers, number of units can be changed):

> -I did this a little more statically than ideal (because I faced some value errors otherwise) as to change up parameters of the neural network you have to make changes to the parameters in the initialiser (__init__ method) of the DeepNeuralNetwork class. However, the number and types of layer as well as other number of units of all the parameters can be changed there and take effect without issues. This can be seen in code.

-Implement an optimizer (e.g. SGD or Adam) and a stopping criterion of your choosing:

> -I used Stochastic Gradient Descent (SGD) optimizer in my update_network_parameters method in the DeepNeuralNetwork class as the maths seemed more straight-forward for it. I used information from a website I referenced below to make my own.

-Train your Neural Network using backpropagation:

> -I coded this in my backpass method. I needed to adjust my layer code by adding if statements to be compatible backwards and forwards. And get changes from layers on the backpass aswell. I used some of my referenced sites to guide the way I coded this as well.

Results:

-Evaluate different neural network architectures/parameters, present and discuss your results

> -1 of each layer (sigmoid, relu, softmax):

```
Epoch: 1, Time Spent: 28.01s, Accuracy: 69.62%
Epoch: 2, Time Spent: 56.10s, Accuracy: 76.40%
Epoch: 3, Time Spent: 82.75s, Accuracy: 78.76%
Epoch: 4, Time Spent: 110.76s, Accuracy: 80.10%
Epoch: 5, Time Spent: 139.88s, Accuracy: 81.11%
Epoch: 6, Time Spent: 168.95s, Accuracy: 81.69%
Epoch: 7, Time Spent: 197.60s, Accuracy: 82.39%
Epoch: 8, Time Spent: 225.10s, Accuracy: 83.11%
Epoch: 9, Time Spent: 254.01s, Accuracy: 83.34%
Epoch: 10, Time Spent: 281.94s, Accuracy: 83.50%
```

-2 sigmoid, 1 softmax:

```
Epoch: 1, Time Spent: 26.77s, Accuracy: 53.79%
Epoch: 2, Time Spent: 53.48s, Accuracy: 57.79%
Epoch: 3, Time Spent: 82.42s, Accuracy: 59.79%
Epoch: 4, Time Spent: 109.07s, Accuracy: 62.07%
Epoch: 5, Time Spent: 136.40s, Accuracy: 63.84%
Epoch: 6, Time Spent: 162.41s, Accuracy: 65.36%
Epoch: 7, Time Spent: 187.66s, Accuracy: 75.76%
Epoch: 8, Time Spent: 216.44s, Accuracy: 77.58%
Epoch: 9, Time Spent: 246.96s, Accuracy: 78.68%
Epoch: 10, Time Spent: 276.11s, Accuracy: 79.01%
```

-2 relu, 1 softmax:

```
Epoch: 1, Time Spent: 26.71s, Accuracy: 79.29%
Epoch: 2, Time Spent: 53.72s, Accuracy: 10.00%
Epoch: 3, Time Spent: 82.01s, Accuracy: 10.00%
Epoch: 4, Time Spent: 110.22s, Accuracy: 10.00%
Epoch: 5, Time Spent: 136.55s, Accuracy: 10.00%
Epoch: 6, Time Spent: 164.02s, Accuracy: 10.00%
Epoch: 7, Time Spent: 192.33s, Accuracy: 10.00%
Epoch: 8, Time Spent: 219.96s, Accuracy: 10.00%
Epoch: 9, Time Spent: 248.54s, Accuracy: 10.00%
Epoch: 10, Time Spent: 275.45s, Accuracy: 10.00%
```

Results: I will do all tests with 10 epochs, 0.01 learning rate to have models be more comparable.

Overall, the neural network with 1 of each sigmoid, relu and softmax layer was best. It reached the highest level of accuracy without overfitting.

With the 2 sigmoid, no relu layer model, the accuracy increased without going over into overfitting, but it started lowest at 53% and increased over epochs. By the tenth epoch it was only about 5% lower accuracy by the 10th epoch than the model with 1 of all 3 layers.

Finally, the model with 2 relu and no sigmoid layers was tested. It started with the highest accuracy, this makes sense as relu function is the fastest learning, but with 2 relu layers and 1 softmax layer only, the neural network goes too quickly and overfits harshly already by the 2nd epoch lowering accuracy to 10%. It the maintains this till the 10th epoch.

Task 4 Description:

The fourth task is about implementing a neural network using PyTorch. You will use

CIFAR-10 dataset for this task.

- Implement a neural network, Propose improvements (eg. Convolutional Neural Network, dropout, etc):

    -I built a convolutional neural network in my code as CIFAR-10 is image data and the type of neural networks that works best with images and videos is a convolutional neural network (CNN). This is because CNNs are specially designed to read pixel values from Images or videos and learn from them. This should be an improvement on a basic neural network in our case as a CNNs learning ability for image datasets is much better and we can therefore have a higher accuracy than we would otherwise for image recognition.

Results:

-Evaluate different parameters, Present and discuss the results in the report:

--Generally, what are the different measures of accuracy for my convolutional neural network on the CIFAR-10 images (figures used for 4 epocs):

```
[epoch 1,  2000] loss: 2.238
[epoch 1,  4000] loss: 1.856
[epoch 1,  6000] loss: 1.683
[epoch 1,  8000] loss: 1.569
[epoch 1, 10000] loss: 1.505
[epoch 2,  2000] loss: 1.459
[epoch 2,  4000] loss: 1.420
[epoch 2,  6000] loss: 1.392
[epoch 2,  8000] loss: 1.345
[epoch 2, 10000] loss: 1.321
[epoch 3,  2000] loss: 1.283
[epoch 3,  4000] loss: 1.245
[epoch 3,  6000] loss: 1.233
[epoch 3,  8000] loss: 1.214
[epoch 3, 10000] loss: 1.199
[epoch 4,  2000] loss: 1.139
[epoch 4,  4000] loss: 1.139
[epoch 4,  6000] loss: 1.142
[epoch 4,  8000] loss: 1.128
[epoch 4, 10000] loss: 1.088
Training complete
```

-loss: the mean square error in the prediction. Fig:

-accuracy: measure of how accurate your model's prediction is compared to the true data. fig:

```
Accuracy of the network on test image dataset: 59 %
```

-we can also explore the accuracy of the prediction by image class. Fig:

```
Accuracy for airplane is: 51.2 %
Accuracy for automobile is: 63.0 %
Accuracy for bird  is: 54.6 %
Accuracy for cat   is: 41.7 %
Accuracy for deer  is: 60.8 %
Accuracy for dog   is: 38.7 %
Accuracy for frog  is: 78.1 %
Accuracy for horse is: 50.4 %
Accuracy for ship  is: 80.1 %
Accuracy for truck is: 71.6 %
```

-Convolutional neural networks are a bit more specialised to read pixel values, but the main parameter to change and read its effects on accuracy and loss rate are epochs. That is to say how does the number of times we loop the training data effect the outcome of the overall accuracy and loss of the model.

-If we were to plot the line of loss vs epochs and the line of accuracy vs epochs, we could visualise the point overfitting occurs at the point the lines for loss and accuracy converge.

-Increasing the number of epochs used to train the neural network model will at that point make the model less accurate and less useful when confronting new testing data which defeats the purpose of our models as we want to be able to predict and classify new data, based on knowledge from training data.

-My machine takes a long time to run the computations, so I can't find the exact, more reliable point which overfitting occurs. However, using my testing with epochs from 1 to 10, I can find roughly where it occurs.

-from the following figure, It can be understood that somewhere around 6 epochs overfitting will occur as more epochs of training data after this reduces or at best stagnates the accuracy of our convolutional neural network model on testing data.

```
#how does changing epochs effects results
#1 epoch 47&
#2 epoc 55%
#3 epoc 57%
#4 epoc 59%
#5 epoc 60%
#6 epoc 63%
#7 epoc 62%
#8 epoc 63%
#9 epoc 61%
#10 epoc 63%
```

References:

-dataset: https://www.kaggle.com/zalando-research/fashionmnist

-didn't use code but helped decide how I would import data: https://www.tensorflow.org/tutorials/keras/classification#preprocess_the_data

-same as above: https://www.renom.jp/notebooks/tutorial/neuralnetwork/download_mnist/notebook.html

-I used a lot of similar conventions to make my own code for task 3. Due to similar maths and the way some code is specific if you do it in this convention, like maybe 25% of my numpy neural network class code is similar: https://mlfromscratch.com/neural-network-tutorial/#/

-didn't use code but learned some maths for layer functions from: https://towardsdatascience.com/lets-code-a-neural-network-in-plain-numpy-ae7e74410795

-again, helped with maths not used code: https://medium.com/ai%C2%B3-theory-practice-business/a-beginners-guide-to-numpy-with-sigmoid-relu-and-softmax-activation-functions-25b840a9a272

-not used code but learned sgd optimizer from this: https://ruder.io/optimizing-gradient-descent/index.html#stochasticgradientdescent

-task 4 more so below:

-dataset: https://www.cs.toronto.edu/~kriz/cifar.html

-used similar methodology to make my code. Some lines of code might be used as I used diff tutorials to learn how to code the convolutional neural network. Maybe 20-30% : https://pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html

-code not used but helped to learn to build my own convolutional neural network:
https://www.cs.toronto.edu/~lczhang/aps360_20191/lec/w03/convnet.html

-not code, explains overfitting threshold:
https://www.analyticsvidhya.com/blog/2020/09/overfitting-in-cnn-show-to-treat-overfitting-in-convolutional-neural-networks/#:~:text=Overfitting%20indicates%20that%20your%20model,of%20overall%20Deep%20Learning%20Models.