ECE 661: Homework #2

Elisa Chen; Duke NetID: eyc11

2023-10-02

1 True / False Questions

Problem 1.1: **True** - While there are other mathematical operations involved in normalizing the batch input (such as dividing the inputs by the variance), by subtracting the mean from the batch input, we are able to obtain zero mean.

Problem 1.2: **True** - PyTorch is a machine learning library for Python that provides a flexible and dynamic framework for building and training machine learning models, particularly neural networks.

Problem 1.3: **False** - Data augmentation CAN be an effective technique for generating diversified training data and to prevent the model from overfitting to the original training dataset. However, it is untrue that these techniques will ALWAYS be beneficial for CNNs. Data augmentation may hurt small models as they may underfit. Additionally, the user must be aware whether augmenting the data will corrupt the dataset.

Problem 1.4: **False** - CNNs can still be trained successfully without batch normalization or dropout, especially on simpler tasks or when using appropriate initialization methods, proper learning rates, and other regularization techniques.

Problem 1.5: **False** - While it is true that Dropout can be a common technique for combatting overfitting, it is untrue that if both L-normalization and Dropout are incorporated at the same time, the performance will be even better. At times, Dropout does not cooperate well with L-norm regularizations.

Problem 1.6: **True** - L1 regularization has a diamond loss contour whereas L2 regularization has a circular loss countour. As a result, L1 regularization is more likely to obtain sparser weights compared to L2 regularization.

Problem 1.7: **True** - Contrary to the vanilla ReLU, the leaky ReLU has a slope for negative inputs which solves the dead neurons problem. However, the inconsistent slope makes the training process for some neural network architectures unstable. Typically, it will take more time for leaky ReLU to reach convergence.

Problem 1.8: **True** - Depthwise separable convolution gives ~9x reduction in MAC. A 3x3 convolution layer has $3x3xMxNxD_FxD_F$ MACs whereas the Depthwise separable convolution has $3x3xMxD_FxD_F + D_FxD_FxMxN$ MACs.

Problem 1.9: **True** - SqueezeNet puts a significant portion of the computations in the later stages of the CNN design. This design choice allows SqueezeNet to reduce the number of parameters in the network's squeeze layers. The bulk of the computations and information flow happen in the "expand" layers which follow the squeeze layers.

Problem 1.10: **True** - The shortcut connections in ResNet result in a smoother loss surface making the optimization process easier.

2 Lab 1

a) Two dummy inputs of size 32 x 32 with 3 channels (one for each RGB color) were created resulting in a data shape of (2, 3, 32, 32). After passing through the neural net, we'd expect 2 predictions where each

prediction is a probability distribution over the 10 categories. As a result, we'd expect the output shape to be (2, 10). We can pass the dummy data through the net and assert that the output has a shape of (2, 10) with the following line of code:

```
assert(out.detach().cpu().numpy().shape == (2,10))
```

Additionally, after inspecting the parameters of the neural network at each layer, we can see that the first layer expects 3 channels as desired, and the last linear layer outputs 10 values, which also aligns with our expectations. The hidden layer shapes are mathematically correct and as our code didn't produce any errors when forward passing the data through the net, we can have comfort over the fact that they are correctly implemented.

b) The following two pre-processing steps were conducted to facilitate training and inference of the neural net: 1) Data Normalization and 2) Converting PIL images to Tensors. Data Normalization improves the learning process as the loss will be less sensitive to small changes in weights making it easier to optimize the model. We also need to convert the data type from PIL images to Tensors for model training purposes (training accepts tensors only).

c) Please see `simplenn-cifar10.ipynb` for details of the code implementation.

d) After running `nvidia-smi` command in the terminal, I see the following information:

```
!nvidia-smi
```

```
Fri Sep 22 20:13:54 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   64C    P0    31W /  70W |   1057MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

Figure 1: Nvidia-Smi output

From the above image, we can denote that we're using a GPU (CUDA version 12) to run the model.

e) Please see `simplenn-cifar10.ipynb` for details of the code implementation.

f) Please see `simplenn-cifar10.ipynb` for details of the code implementation.

g) The initial training loss value during Epoch 0 (prior to conducting any training steps) is 1.8561 and the validation loss is 1.5668. In other words, the initial predictions are about 2 numbers away from the correct prediction on average. Assuming that the model is more or less uniformly predicting the classification prior to the first weight update, an initial training loss of ~2 seems fairly reasonable to me. While we see an

improvement in the training loss and accuracy over time, we see no significant improvements in the validation loss / accuracy as we increase the epoch number. This might suggest the model is overfitting to the training data.

h) After we incorporate a learning rate scheduler, we can see that we reach the results obtained in part g) a lot sooner. Specifically, we obtain large significant gains in both training and validation accuracy each time the scheduler reduces the learning rate by 50% which is what the `DECAY` parameter was set.

My final best validation accuracy: 0.6656

```
Epoch 29:
Training loss: 0.1111, Training accuracy: 0.9684
Validation loss: 2.1459, Validation accuracy: 0.6472

==================================================
==> Optimization finished! Best validation accuracy: 0.6656
```

Figure 2: Best Validation Accuracy With Learning Rate Scheduler

3 Lab 2

a) As observed from the below results, the best validation accuracy has improved by ~8% after applying data augmentation techniques on the data. The best validation accuracy was 0.71.

```
Epoch 28:
Training loss: 0.8585, Training accuracy: 0.6970
Validation loss: 0.8504, Validation accuracy: 0.7050

Epoch 29:
Training loss: 0.8595, Training accuracy: 0.7000
Validation loss: 0.8369, Validation accuracy: 0.7120
Saving ...

==================================================
==> Optimization finished! Best validation accuracy: 0.7120
```

Figure 3: Best Validation Accuracy With Data Augmentation Techniques

b) The best validation accuracy improves marginally from 0.71 to 0.72 after implementing batch normalization. However, we can empirically tell that batch normalization will result in better training outcomes when the learning rate is larger. This was illustrated by using a learning rate of 0.1 (which was 10x larger than the initial learning rate) for neural nets with and without batch normalization. From Figure 4 we can tell that the training loss is consistently smaller when we use batch normalization:

Please refer to `simplenn-cifar10-dev.ipynb` notebook for more details on the implementation of the Swish function. The custom class is called `Swish`. When we train the model with Swish we obtain a best validation accuracy of 0.75, which is better than what we'd obtain with ReLU. Therefore, moving forward, we'll be applying Swish activation function in subsequent parts of this lab.

c)

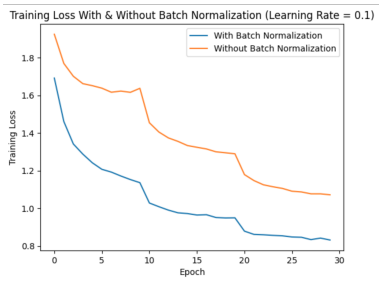

Figure 4: Training Loss for With and Without Batch Normalization

- Observations from tuning the learning rate: apart from a learning rate of 1 and 0.001 (the extreme ends of the range), all other values of learning rates seemed to result in similar gains in the validation loss as the epoch number increases (see figure 6). We can observe that a learning rate of 0.05 resulted in the greatest validation loss for the model. The model was not able to learn at all when the learning rate was set to 1. Please see the below table and figures for more details about each learning rate.

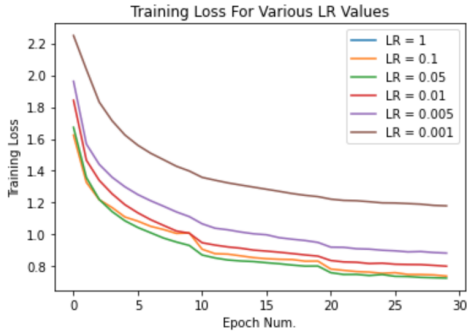

Figure 5: Training Loss For Various Learning Rates

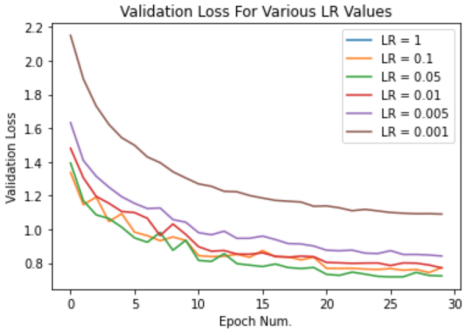

Figure 6: Validation Loss For Various Learning Rates

	Learning Rate	Best Validation Accuracy
0	1	0.1028
1	0.1	0.7386
2	0.05	0.7502
3	0.01	0.7278
4	0.005	0.7046
5	0.001	0.6110

Figure 7: Best Validation Accuracy For Each Learning Rate

- Observations from tuning the regularization rate: The model performed the poorest when the regularization rate was set to `1e-2` and the best when the regularization rate was set to `0`.We observe significant improvements in the validation loss when the regularization rate is anything other than `1e-2`. All other regularization rates resulted in fairly similar performance. Based on the experiment, it would suggest that regularization has little to no impact on the performance of the model. The Best validation accuracy was `0.7192`, which was obtained when no regularization was applied. Therefore, one could argue that this model might not require regularization as the benefits are marginal and probably not worth the additional computational cost that comes with regularization. Please see the below table and figures for more details about each regularization rate.

Figure 8: Training Loss For Various Reg Values When Learning Rate = 0.01

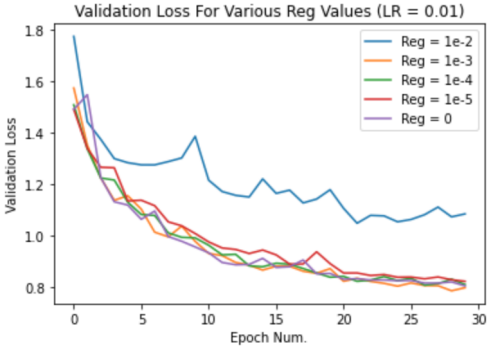

Figure 9: Validation Loss For Various Reg Values When Learning Rate = 0.01

	Reg Rate	Best Validation Accuracy
0	1e-2	0.6310
1	1e-3	0.7190
2	1e-4	0.7188
3	1e-5	0.7060
4	0	0.7192

Figure 10: Best Validation Accuracy For Each Regularization Rate

4 Lab 3

a) Please refer to `resnet_cifar10.ipynb` for more details about the implementation of the ResNet20. The class is called `ResNet20`.

b) The ResNet-20 model was trained using the following hyperparameters: > Learning rate: 0.1; Momentum: 0.9; Training Batch Size: 128, Validation Batch Size: 100; Weight_decay: 0.0001, # of Epochs: 120

An SGD optimizer and Cross-Entropy Loss were used to train the model. In addition to input normalization, a random horizontal flip and crop of size 32 with padding of 4 were implemented as part of data augmentation procedures. A best validation accuracy of 90.58% was achieved using the described method. Please see below for the training and validation loss for the model training with 120 epochs:

```
Epoch 118:
Training loss: 0.0710, Training accuracy: 0.9758
Validation loss: 0.3685, Validation accuracy: 0.9028

Epoch 119:
Training loss: 0.0686, Training accuracy: 0.9772
Validation loss: 0.3609, Validation accuracy: 0.9028

================================================
==> Optimization finished! Best validation accuracy: 0.9058
```

Figure 11: Best Validation Accuracy For ResNet-20

Figure 12: Training and Validation Loss For ResNet-20