**Project #1: MLP Implementation**

2019170540 Wonjun, Oh (오원준)

**1. Description of your code**

**Implementation** Based on the assignments during class, the code was constructed in consideration of ReLU activation, Batch dimension, and L2 Regularization. A detailed description of the dimension and code of each step matrix is annotated in code.

**Hyperparameter Tuning** In order to explore the appropriate parameters for learning, we conducted a total of 32 random searches in the following ranges for four parameters: Learning rate, Learning rate decay, Weight decay(reg), and Hidden dim. Each trial ran for 16 Epoch with Batch 32, and while having a high Batch result in good learning, we limited the number small for quick exploration.

|  |  |  |
| --- | --- | --- |
| **Params** | **Min** | **Max** |
| **Learning rate** | 1e-5 | 1e-3 |
| **Learning rate decay** | 0.6 | 1 |
| **Weight decay** | 0.0001 | 0.1 |
| **Hidden dim** | 128 | 512 |

(Table 1) Settings for Hyperparameter tuning.

In the case of Learning rate and Weight decay, the following log scale sampling function was defined and used to aim to sample normally with log scale.

|  |
| --- |
| **Algorithm 1: Log Uniform Sampling** |
| ***def***log\_uniform\_sampling(a: float = 0., b: float = 1.) -> float:  high, low = (b, a) if b > a else (a, b)  ***return*** np.power(10, np.random.uniform(np.log10(low), np.log10(high))) |

**2. Results**

We took the highest of 32 sampling results and finally achieved 50.5% accuracy.

A screen shot of a computer

Description automatically generated

(Figure 1) Results

**3. Discussions**

**Analysis of Hyperparameters** We have summarized the results (see Appendix 1) of the analysis of the results for four parameters. For the **Learning Rate**, values ranging from 0.0001 to 0.001 produced generally good results. For **Learning Rate Decay**, values 0.6~0.8 were excellent. Lowering the learning rate is helpful for learning when it is 'close to convergence' in the current situation of iteration (considering that the number of terms is fixed). In the case of **Weight Decay**, values around 0.02 resulted in the best results, but no appropriate indicator was found for comparison. In the case of **Hidden Dim**, when the accuracy of 0.47 or more was recorded for Validation, they all had Hidden Dim's of 280 or more. This means that if the hidden dim of the model is reduced (model complexity is lowered), even if the data is sufficient, it can cause high bias problems due to Underfitting.

**Analysis of Visualized Weights** Looking at the results of visualizing the final layer, we often see that each weight has a form like the object we want to classify (horse, frog, etc.). It can be analyzed that the deep learning model is considering the semantic information of the image.

Based on the above point of view, the higher the hidden size, the higher the accuracy is because it is possible to consider various semantical information, composition, and characteristics of photographs.

A close-up of a mosaic of different colors

Description automatically generated

(Figure 2) Visualization of Weights

**Analysis of Train History** Because of learning with a relatively small batch size, the loss graph shows a very oscillating pattern. If you learn with a larger batch size, you will observe a more stable form than this. Also, it is difficult to accurately judge because the loss for the Validation dataset is not logged, but the trend of validation accuracy continues to increase, so it is likely that better results can be achieved when learning with more epoch.

A graph of loss and loss history

Description automatically generated

(Figure 3) History of loss and accuracy

**Analysis of prediction accuracy** Based on the results; the accuracy of each class was analyzed. Although the distribution of the resulting datasets is very evenly spread, it is observed that a particular class is not well judged. I thought this meant that certain classes may be more difficult to distinguish than others.

A graph with blue lines

Description automatically generated

(Figure 4) Distribution of data by class

A graph of blue bars

Description automatically generated

(Figure 5) Accuracy of data by class

**4. Further Research (Optional)**

1. Since the hyperparameter setting of the model changes, the degree of learning may vary depending on the number of epochs. Therefore, fixing the Epoch and comparing it can be dangerous. To overcome this, Early Stopping can be used by logging Validation Loss, but it was not implemented because it did not know whether the model code could be modified.

2. To accurately measure the impact of Hyperparameters, it is necessary to measure the impact independently after controlling the variable for each Hyperparameter. In this task, due to the code composed of NumPy, it was not possible to try because there was not enough time to learn with CPU rather than CUDA.

**End.**

**(Appendix 1)** History of Hyperparameter Tuning

#Trial 1 | val\_acc: 0.4780 | lr: 0.0002729 | lr\_decay: 0.8089 | reg: 0.0134 | hidden\_dim: 283 -> Model saved!

#Trial 2 | val\_acc: 0.3980 | lr: 0.0000431 | lr\_decay: 0.8588 | reg: 0.0001 | hidden\_dim: 142

#Trial 3 | val\_acc: 0.4770 | lr: 0.0005172 | lr\_decay: 0.8663 | reg: 0.0087 | hidden\_dim: 481

#Trial 4 | val\_acc: 0.4810 | lr: 0.0001361 | lr\_decay: 0.8292 | reg: 0.0081 | hidden\_dim: 407 -> Model saved!

#Trial 5 | val\_acc: 0.4890 | lr: 0.0004208 | lr\_decay: 0.6289 | reg: 0.0356 | hidden\_dim: 392 -> Model saved!

#Trial 6 | val\_acc: 0.4450 | lr: 0.0000893 | lr\_decay: 0.7727 | reg: 0.0065 | hidden\_dim: 292

#Trial 7 | val\_acc: 0.2870 | lr: 0.0000183 | lr\_decay: 0.6849 | reg: 0.0002 | hidden\_dim: 473

#Trial 8 | val\_acc: 0.4650 | lr: 0.0001239 | lr\_decay: 0.8593 | reg: 0.0002 | hidden\_dim: 244

#Trial 9 | val\_acc: 0.4730 | lr: 0.0001459 | lr\_decay: 0.8175 | reg: 0.0001 | hidden\_dim: 395

#Trial 10 | val\_acc: 0.4530 | lr: 0.0002298 | lr\_decay: 0.9568 | reg: 0.0009 | hidden\_dim: 378

#Trial 11 | val\_acc: 0.3820 | lr: 0.0000349 | lr\_decay: 0.7833 | reg: 0.0005 | hidden\_dim: 245

#Trial 12 | val\_acc: 0.3600 | lr: 0.0000169 | lr\_decay: 0.9529 | reg: 0.0001 | hidden\_dim: 176

#Trial 13 | val\_acc: 0.3960 | lr: 0.0000713 | lr\_decay: 0.6714 | reg: 0.0077 | hidden\_dim: 489

#Trial 14 | val\_acc: 0.4320 | lr: 0.0000585 | lr\_decay: 0.8215 | reg: 0.0008 | hidden\_dim: 439

#Trial 15 | val\_acc: 0.4590 | lr: 0.0001035 | lr\_decay: 0.8216 | reg: 0.0031 | hidden\_dim: 460

#Trial 16 | val\_acc: 0.4580 | lr: 0.0003805 | lr\_decay: 0.9467 | reg: 0.0001 | hidden\_dim: 457

#Trial 17 | val\_acc: 0.4440 | lr: 0.0000630 | lr\_decay: 0.8985 | reg: 0.0044 | hidden\_dim: 473

#Trial 18 | val\_acc: 0.4800 | lr: 0.0003819 | lr\_decay: 0.7525 | reg: 0.0373 | hidden\_dim: 380

#Trial 19 | val\_acc: 0.4330 | lr: 0.0000517 | lr\_decay: 0.9007 | reg: 0.0030 | hidden\_dim: 418

#Trial 20 | val\_acc: 0.3990 | lr: 0.0000491 | lr\_decay: 0.8179 | reg: 0.0160 | hidden\_dim: 396

#Trial 21 | val\_acc: 0.4480 | lr: 0.0002505 | lr\_decay: 0.8255 | reg: 0.0036 | hidden\_dim: 192

#Trial 22 | val\_acc: 0.4370 | lr: 0.0008602 | lr\_decay: 0.9203 | reg: 0.0003 | hidden\_dim: 270

#Trial 23 | val\_acc: 0.4670 | lr: 0.0001354 | lr\_decay: 0.7490 | reg: 0.0012 | hidden\_dim: 166

#Trial 24 | val\_acc: 0.3700 | lr: 0.0000197 | lr\_decay: 0.8898 | reg: 0.0125 | hidden\_dim: 369

#Trial 25 | val\_acc: 0.3840 | lr: 0.0000332 | lr\_decay: 0.9345 | reg: 0.0002 | hidden\_dim: 136

#Trial 26 | val\_acc: 0.4300 | lr: 0.0000959 | lr\_decay: 0.6998 | reg: 0.0560 | hidden\_dim: 243

#Trial 27 | val\_acc: 0.4710 | lr: 0.0002649 | lr\_decay: 0.9846 | reg: 0.0008 | hidden\_dim: 225

#Trial 28 | val\_acc: 0.4120 | lr: 0.0000544 | lr\_decay: 0.8290 | reg: 0.0415 | hidden\_dim: 347

#Trial 29 | val\_acc: 0.3910 | lr: 0.0000300 | lr\_decay: 0.9162 | reg: 0.0110 | hidden\_dim: 392

#Trial 30 | val\_acc: 0.4750 | lr: 0.0004019 | lr\_decay: 0.7878 | reg: 0.0104 | hidden\_dim: 380

#Trial 31 | val\_acc: 0.1870 | lr: 0.0000110 | lr\_decay: 0.6346 | reg: 0.0064 | hidden\_dim: 383

#Trial 32 | val\_acc: 0.4720 | lr: 0.0003888 | lr\_decay: 0.7268 | reg: 0.0087 | hidden\_dim: 366