AI6103 Deep Learning & Applications Assignment

Kaiyu Li

School of Computer Science and Engineering
G2202164A

Introduction

In this paper, we will conduct a series of experiments with ResNet-18 network[2] and the CIFAR-10[3] dataset to study the effects of various kinds of hyperparameters. We will investigate the effects of initial learning rate, learning rate schedule, weight decay, and data augmentation methods in order, in the mean time, at each stage selecting the most plausible hyperparameter according to the performance on the validation set. At the end, we will examine the performance of our model on the hold-out test set and justify our selection.

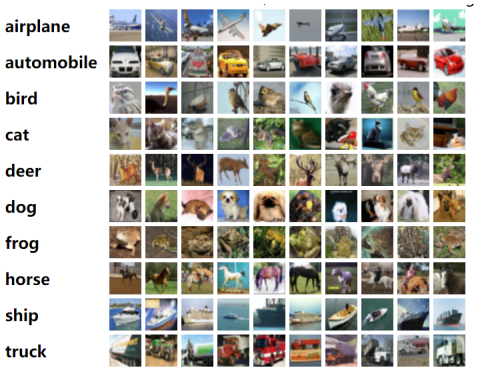


Figure 1: CIFAR-10 Overview

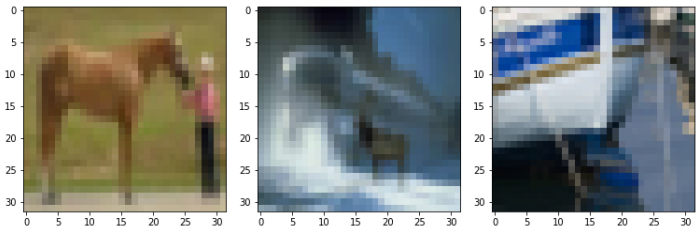


Figure 2: Image labels from left to right are: horse(left), deer(middle), ship(right)

Data Characteristics of CIFAR-10

The CIFAR-10 dataset consists of 60000 32x32 colored images in 10 classes as shown in figure 1, with exact 6000 images per class. All classes are mutually exclusive. Images within the CIFAR-10 dataset are taken in the natural world. Although most images are not occluded and placed in the center of image, this dataset still has some challenging characteristics. Observing the images shown in figure 2, from left to right, the first images contains not only the labeled object(a horse), but also contains another object - the person on the right side. Objects in the second and the third images have very different proportion against the whole image, and the third image only shows the front of the ship. In conclusion, the real-world images in CIFAR-10 have considerable background noises, the scale of the objects changes significantly from sample to sample, and some images only capture part of the object.

Training/Validation Set Split

The CIFAR-10 dataset has already been separated to a training set of 50000 images and a test set of 10000 images, so all we need to do is to split a validation set. Because we have relatively sufficient amount of samples for training, we can split the original training set to a new training set and a validation set, making the size of validation set equal to test set.

First, we can simply apply a **random split**, the code for partition is attached in the appendices. The proportion of each class in the new training set is shown in firgure 3.

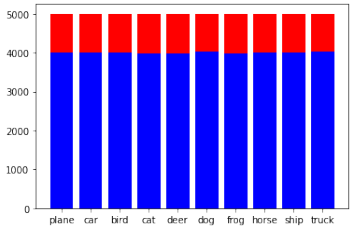


Figure 3: CIFAR-10 Overview

An alternative method would be using a **stratified split**(see code in appendices).This will guarantee that the train set and validation set have **exactly same distributions** in terms of classes(so we omit the figure here). This method will be very helpful if the dataset is small or the sizes of different classes are very biased, because we may have all

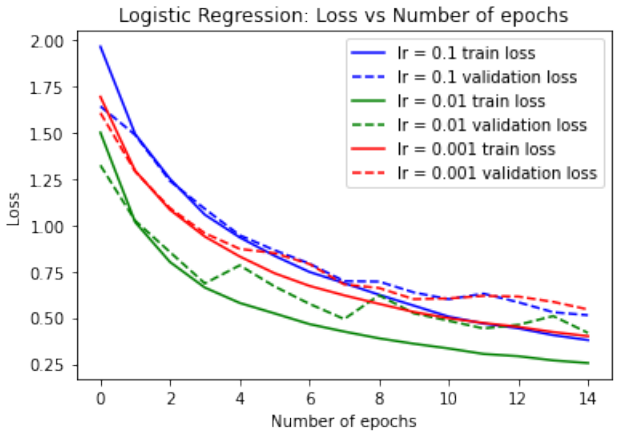
the sample of a particular class all goes to the train set or validation set. In this paper, we still apply the **random split** method.

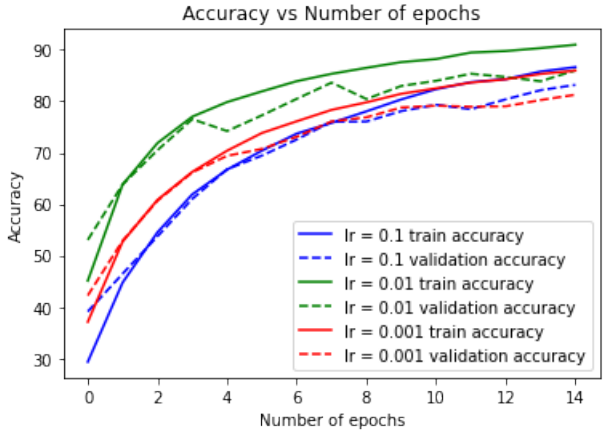
Learning Rate

First We conduct experiments to figure out the influence of learning rate on training neural network. Before we dig in, we should clarify our configuration of other hyperparameters, which is fixed and is applied through the entire series of experiments:

- batch size = 128
- Optimization algorithm: SGD, with moment = 0.9
- Data Augmentation: Random Cropping and Horizontal Flipping
- Normalized: True

Now we train the ResNet with learning rate as 0.001, 0.01, and 0.1, running 15 epochs for each setting. The result curves of loss and accuracy against number of epochs are shown in figures below, attached with a table reporting the result of the final epoch.





Learning Rate η	Train Loss	Validation Loss	Train Accuracy	Validation Accuracy
0.1	0.38	0.51	86.64	83.19
0.01*	0.26	0.42	91.0	86.02
0.001	0.40	0.55	85.96	81.28

(We use '*' to mark the parameter regarded as the best choice)

Recall that if the loss function is $L(w)$, the model parameter w is updated iteratively complying with the formula, where η represents the learning rate:

$$w_t = w_{t-1} - \eta \frac{\partial L}{\partial w}\Big|_{w_{t-1}} \tag{1}$$

Equation (1) explains the reason why η = 0.001 accounts for the slowest decreasing rate and increasing rate in terms for loss and accuracy respectively. Also, from epoch 8 to 15, the validation loss curve for η = 0.001 fluctuates but barely decreases in total. This phenomenon indicates a risk of being trapped in local minimum.

For learning rate as 0.1, the loss curve has a better decreasing rate, but large learning rate may suffer from oscillation and over shooting. We argue that we don't have enough epoch to detect such kind of phenomenon.

For learning rate as 0.01, although the curves seems to fluctuate more violently, the validation loss and accuracy beats the other two settings over the whole plot. Also, η = 0.01 is in the middle of the other two learning rate values, so we decide to choose this value as our appropriate learning rate for the rest of experiments.

Learning Rate Schedule

Cosine Annealing Schedule

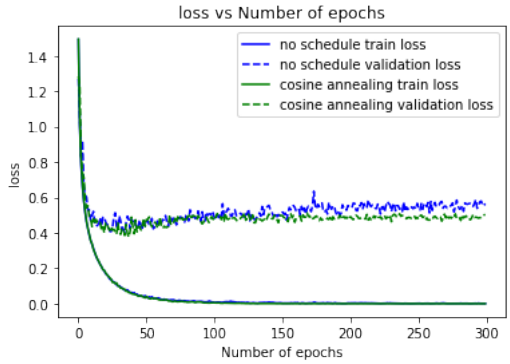
A cosine annealing scheduler[4] sets the learning rate η_t of t-th epoch using the following equation:

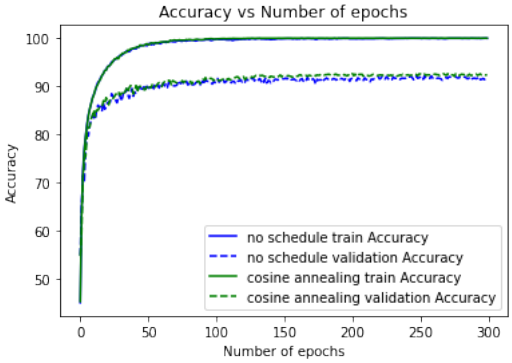
$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})(1 + cos(\frac{T_{cur}}{T_{max}}\pi)) \tag{2}$$

where η_{min} and η_{max} are the boundaries for the learning rate, T_{cur} represents the number of epochs since the last restart, and T_{max} accounts for the maximum number of iterations. According to equation 2, when $T_{cur} = 0$, the initial learning rate equals to η_{max}, and when $T_{cur} = T_{max}$, the learning rate finially decreases to η_{min}. Within the first and the last epoch, the learning rate against epoch decreases in the shape of the cosine function.

Result of Experiments

Now we investigate the effect of Cosine Annealing schedule by comparing training with and without the schedule. The ResNet is trained for 300 epochs this time, with learning rate set to 0.1. Correspondingly, the T_{max} parameter of the scheduler is set to 300, so that the learning rate reduces to zero over the entirety of the training session.The result curves of loss and accuracy against number of epochs are shown in figures below.





Is Scheduled	Train Loss	Validation Loss	Train Accuracy	Validation Accuracy
No	0.0011	0.5557	99.96	91.89
Yes*	0.0009	0.5030	100.0	92.40

First, for both models, no matter trained with the scheduler or not, there are huge gaps between their training results and validation results, and their training loss both reaches to almost 0. This outcome indicates that both models are overfitting, and applying a cosine annealing schedule does not help mitigate overfitting.

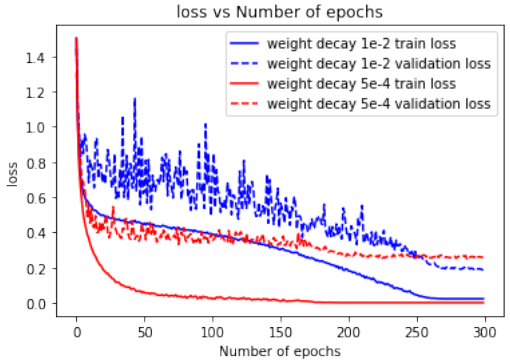
Second, for train loss curve of the model trained without cosine annealing, the loss keeps oscillating during the entire training process, and even slowly increase after reaching a minimum at about 40 epochs. We argue that before around the 40-th epoch, the learning rate as 0.1 is still an appropriate value, but then the learning rate becomes too large to find the minimum, and causes overshooting at every step. On the contrary, the model with cosine annealing converges swiftly and stably, and gives a better result in terms of both loss and accuracy.

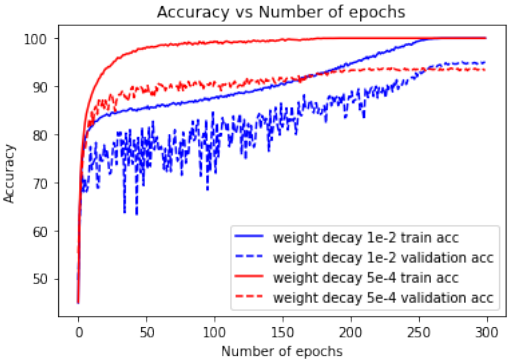
Weight Decay

From last section we have already acquired a easy-to-train model with relatively good accuracy, but our model is highly overfitting. Now we will explore the weight decay technique to add regularization to our model. The weight decay adds the regularization term $\frac{1}{2}\lambda||w||^2$ to the original loss function, which works similarly to L2 regularization. Consequently, the new update rule of model parameter w is given by:

$$w_{t+1} = w_t - \eta\frac{\partial L(w_t)}{\partial w_t} - \eta\lambda w_t \tag{3}$$

λ is called the weight decay coefficient. We intend to compare two settings - $\lambda = 1 \times 10^{-2}$ and $\lambda = 5 \times 10^{-4}$ respectively. Note that the following experiments are conducted using a cosine annealing schedule with initial learning rate equals to 0.1.





Weight Decay λ	Train Loss	Validation Loss	Train Accuracy	Validation Accuracy
1e-2	0.0234	0.1851	99.99	95.01
5e-4*	0.0012	0.2587	100.0	93.31

First, compared with the result of $\lambda = 5 \times 10^{-4}$, curves of$\lambda = 1 \times 10^{-2}$ exhibits a clear underfitting condition from the beginning to around epoch 260, where both validation loss curve and validation accuracy curve fluctuate dramatically. Nevertheless, recall the equation (3), in terms of regularization, reducing the learning rate η by half is effectively the same as reducing the weight decay coefficient λ by half and remaining the same learning rate. Therefore, the curves finally converges after about 260 epochs, due to the learning rate schedule. Yet, 1×10^{-2} is still too large for the weight decay setting.

Then, for $\lambda = 5 \times 10^{-4}$, the curves converges smoothly at around 170-th epoch, however, compare the final validation loss and accuracy with $\lambda = 1 \times 10^{-2}$, $\lambda = 5 \times 10^{-4}$

obviously makes the model overfitting. We still choose $\lambda = 5 \times 10^{-4}$ for our model at the next stage, because it gives more stable performance and relatively good accuracy. Although it is overfitting, yet we can add more regularization through data augmentation in the next section.

Data Augmentation

In the last section, we apply the Cutout augmentation technique [1]with the best experimental setup so far, that is:

- Learning rate: 0.01
- Learning rate schedule: cosine annealing
- Weight decay: 5×10^{-4}

Random Erasing algorithm randomly selects a rectangle shaped area in a input image for training, and wipes out its pixels with a preset value or random values. This technique alleviates model overfitting and makes the model more robust to occlusion. We use the implementation provided by $PyTorch$ and set the erasing value to be the mean pixel value of three color channels across the whole training set. Then, there are three parameters left to be tuned:

- **p** p - Probability that the random erasing operation will be performed.
- **scale** s_l, s_h - Range of proportion of erased area against input image. To simplify, we set the minimum proportion of erased area to be 0.02, then we only tune the upper bound of the range.
- **ratio** r_1, r_2 - Range of aspect ratio of erased area. To simplify, we set $r_1 = \frac{1}{r_2}$ and only evaluate r_2.

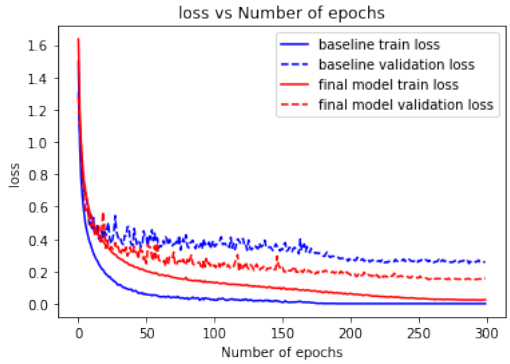
We firstly define a base setting to be $p = 0.5, s_h = 0.3, r_2 = 3$, and then we investigate each parameter independently. Note that the parameters are evaluate this way due to limited time, not because they are independent(e.g. Setting $s_h = 0$ is equivalent to $p = 0$). Therefore, if time allows, these parameters can be tuned jointly. Our results are shown in the tables below:

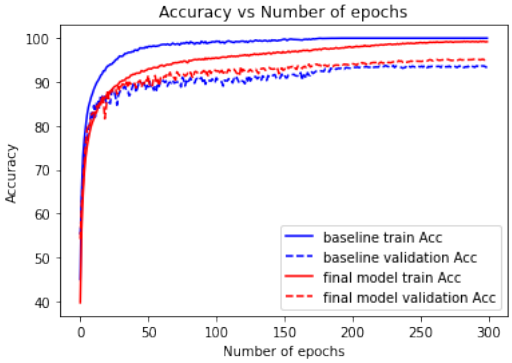
p	Train Loss	Validation Loss	Train Accuracy	Validation Accuracy
None	0.0012	0.2587	100.0	93.31
0.5*	0.0140	0.1921	99.56	95.05
0.75	0.0135	0.1923	99.63	95.06
1.0	0.0119	0.1940	99.68	95.1

scale	Train Loss	Validation Loss	Train Accuracy	Validation Accuracy
None	0.0012	0.2587	100.0	93.31
0.3	0.0140	0.1921	99.56	95.05
0.45*	0.0251	0.1770	99.2	95.08
0.6	0.0430	0.1823	99.56	95.21

ratio	Train Loss	Validation Loss	Train Accuracy	Validation Accuracy
None	0.0012	0.2587	100.0	93.31
1.5	0.0125	0.1965	99.65	94.78
3*	0.0140	0.1921	99.56	95.05
6	0.0118	0.1844	99.63	95.15

Note that the first item of each table shows the baseline result for comparison, which is the result of 5×10^{-4} weight decay from the last section, without Cutout augmentation. Each parameter is tuned within a reasonable range, therefore all the results except baseline have close validation loss and validation accuracy. We choose the setting with the lowest validation loss as our final setting, which is $p = 0.5$,$s_h = 0.45$, $r_2 = 3$. The accuracy of this setting on the hold-out test set is **95.16%**, which is the expected accuracy for the final model to perform at for all similar images in future. At last, we plot the curves of our final setting against the baseline setting:





The above figures shows that our final model has worse performance on training set, but better performance on validation set, which proves that our model reduces the overfitting problem.

References

[1] DeVries, T.; and Taylor, G. W. 2017. Improved Regularization of Convolutional Neural Networks with Cutout.

[2] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[3] Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images.

[4] Loshchilov, I.; and Hutter, F. 2016. SGDR: Stochastic Gradient Descent with Warm Restarts.

Appendices

Python Code for Train/Validation Set Split

Random Split

```
import torch.utils.data as data
import torchvision
#...
tempset = torchvision.datasets.CIFAR10(
root='./data', train=True, download=True, transform=transform_train)

trainset, valset = data.random_split(
tempset, [40000, 10000], generator= torch.Generator().manual_seed(0))
```

Stratified Split

```
import torch.utils.data as data
from sklearn.model_selection import train_test_split
#...
tempset = torchvision.datasets.CIFAR10(
root='./data', train=True, download=True, transform=transform_train)

train_indices, val_indices = train_test_split(list(range(len(tempset.targets))),
test_size=0.2, random_state = 0,stratify=tempset.targets)
trainset = torch.utils.data.Subset(tempset, train_indices)
valset = torch.utils.data.Subset(tempset, val_indices)
```