Journal of Machine Learning Research 1 (2021) 1-48 Submitted 5/17; Published 00/00

Improving L_2 regularization based on AutoL_2

Steven Fang STEVENFANG@NYU.EDU
Courant Institute of Mathematical Science
New York University
New York, NY 10012, USA

Evan Lin CYLIN@NYU.EDU
Courant Institute of Mathematical Science
New York University
New York, NY 10012, USA

Abstract

This paper describes our modification on AutoL_2, a mechanism for decreasing the regularization parameter lambda automatically in L_2 regularization. According to our referenced paper: On the training dynamics of deep networks with L_2 regularization, regularization parameter lambda is decreased by dividing a decay factor when certain criteria are met. In our project, we focus on the Convolutional Neural Network part and present three different methods to improve the AutoL_2 mechanism by utilizing the original AutoL_2 algorithm along with our own modifications. Experimental results demonstrates the performance of the model and the increase/decrease of the regularization parameter lambda as the number of training epochs grows. We also discuss the potential reasons for the results of our experiments.

Keywords: Regularization, Weight Decay, AutoL_2, Convolutional Neural Network

1. Introduction to AutoL_2

Commonly, L_2 regularization is used to reduce the variance, making our model less likely to overfit the training data. The role of L_2 regularization in deep learning has been shown to be effective on training speed and testing accuracy (Lewkowycz and Gur-Ari, 2020). When utilizing regularization, we need to spend some time taking care of the regularization parameter lambda. AutoL_2 is a mechanism to decrease the parameter lambda while training so that we can speed up the training process and even reach a better performance.

To begin with, we give the original AutoL_2 algorithm an initial value for the regularization parameter lambda, and this is the parameter that we are going to change as the training moves on. Then we define a parameter k, which indicates that we make a measurement of loss and error of our model every k steps. Whenever the training loss and training error are bigger than its minimum value for two measurements in a row, we update the lambda by multiplying it with a decay factor. In the following example, the decay factor is 10.

After decaying, notice that there is also a "cool down" time. We force the regularization parameter lambda to stay constant for a period of time, that is $decay\_factor/\lambda$ steps.

Algorithm 1: AUTOL2

MINLOSS, MINERROR = ∞, ∞
MIN_STEP, L2, $decay\_factor$ = 0, 0.1, 0.1
for *t in steps* **do**
  UPDATE_WEIGHTS;
  if *t mod k = 0* **then**
    MAKE_MEASUREMENTS;
    if *ERROR_OR_LOSS_INCREASES* AND *t>MIN_STEP* **then**
      L2=L2 * $decay\_factor$;
      MIN_STEP=$decay\_factor$/L2+t;
    else
      MINLOSS,MINERROR=min($LOSS_{t-k}$,MINLOSS),min($ERROR_{t-k}$,MINERROR);

Function `ERROR_OR_LOSS_INCREASES`(*LOSS,ERROR,MINLOSS,MINERROR*)**:**
  if $LOSS_t$ >*MINLOSS* AND $LOSS_{t-k}$ >*MINLOSS* **then**
    return True;
  if $ERROR_t$ >*MINERROR* AND $ERROR_{t-k}$ >*MINERROR* **then**
    return True;
  return False;

2. Background

Regularization techniques has been proved to be an efficient way to improve the generalization of neural networks (Krogh and Hertz, 1992). Often we want to find a regularization parameter that can achieve the best accuracy and also increase the training speed. However, these two are sometimes contradictory to each other. Imposing strong regularization with stochastic gradient descent on deep neural networks can fail easily (Park et al., 2018). The AutoL_2 (Lewkowycz and Gur-Ari, 2020) method strike a balance between increasing training speed and test accuracy by actively adjusting the regularization parameter during iterations. There isn't other previous work showing a way to improve or optimize the AutoL_2 application. Hence, we proposed the following methods to achieve better performance.

3. Our Modification on Auto L_2

In the original AutoL_2 algorithm, we compare loss and error with its minimum value directly, and we set the regularization parameter lambda to a large value which then decreases while training. We would like to see how the performance would change if we set a percentage threshold for training loss and training error, and moreover, what would happen if we can make lambda increase or decrease when certain criteria are met.
We proposed three different methods to improve the AutoL_2: Our first attempt is that we set a fixed parameter α and use the multiplication of α with $MINLOSS$ and $MINERROR$

respectively as a new threshold to determine whether to update lambda or not. Our second attempt is that, instead of only decreasing lambda, we add a feature to increase lambda in the opposite condition in AutoL_2, and we name it AdvanceL_2. An increase of regularization parameter sounds strange as we are approaching to the local minimum, but we would like to see if this method can help us to jump out of the local minimum. Finally, our third attempt is to combine the first and the second method, which is using AdvanceL_2 (method 2) along with a percentage threshold for training loss and training error.

Algorithm 2: Method 1: Add Percentage Threshold

```
// Main function is exactly the same as Algorithm 1
Function ERROR_OR_LOSS_INCREASES(LOSS,ERROR,MINLOSS,MINERROR,α):
    if LOSS_t >MINLOSS(1+α/100) AND LOSS_{t-k} >MINLOSS(1+α/100) then
        return True;
    if ERROR_t >MINERROR(1+α/100) AND ERROR_{t-k} >MINERROR(1+α/100)
     then
        return True;
    return False;
```

Algorithm 3: Method 2(Advance L_2): Add Increase λ Mechanism

```
MINLOSS, MINERROR = ∞, ∞
MIN_STEP, L2, decay_factor = 0, 0.1, 0.1
for t in steps do
    UPDATE_WEIGHTS;
    if t mod k = 0 then
        MAKE_MEASUREMENTS;
        if ERROR_OR_LOSS_INCREASES AND t>MIN_STEP then
            L2=L2*decay_factor;
            MIN_STEP=decay_factor/L2+t;
        else if ERROR_OR_LOSS_DECREASES AND t>MIN_STEP then
            L2=L2/decay_factor;
            MIN_STEP=decay_factor/L2+t;
        else
            MINLOSS,MINERROR=min(LOSS_{t-k},MINLOSS),min(ERROR_{t-k},MINERROR);
// ERROR_OR_LOSS_INCREASES function is exactly the same as Algorithm 1
Function ERROR_OR_LOSS_DECREASES(LOSS,ERROR,MINLOSS,MINERROR):
    if LOSS_t <MINLOSS AND LOSS_{t-k} <MINLOSS then
        return True;
    if ERROR_t <MINERROR AND ERROR_{t-k} <MINERROR then
        return True;
    return False;
```

Algorithm 4: Method 3: Combination of Method 1 and Method 2

```
MINLOSS, MINERROR = ∞, ∞
MIN_STEP, L2, decay_factor = 0, 0.1, 0.1
for t in steps do
    UPDATE_WEIGHTS;
    if t mod k = 0 then
        MAKE_MEASUREMENTS;
        if ERROR_OR_LOSS_INCREASES AND t>MIN_STEP then
            L2=L2*decay_factor;
            MIN_STEP=decay_factor/L2+t;
        else if ERROR_OR_LOSS_DECREASES AND t>MIN_STEP then
            L2=L2/decay_factor;
            MIN_STEP=decay_factor/L2+t;
        else
            MINLOSS,MINERROR=min(LOSS_{t-k},MINLOSS),min(ERROR_{t-k},MINERROR);
Function ERROR_OR_LOSS_INCREASES(LOSS,ERROR,MINLOSS,MINERROR,α):
    if LOSS_t >MINLOSS(1+α/100) AND LOSS_{t-k} >MINLOSS(1+α/100) then
        return True;
    if ERROR_t >MINERROR(1+α/100) AND ERROR_{t-k} >MINERROR(1+α/100)
     then
        return True;
    return False;
Function ERROR_OR_LOSS_DECREASES(LOSS,ERROR,MINLOSS,MINERROR,α):
    if LOSS_t(1 + α/100) <MINLOSS AND LOSS_{t-k}(1 + α/100) <MINLOSS then
        return True;
    if ERROR_t(1 + α/100) <MINERROR AND ERROR_{t-k}(1 + α/100) <MINERROR
     then
        return True;
    return False;
```

4. Experiments

We use the following architecture to experiment our modification on AutoL_2. $\text{Conv}_1(300) \rightarrow$ Act $\rightarrow$ MaxPool((2,2), 'VALID') $\rightarrow$ $\text{Conv}_1(300)$ $\rightarrow$ Act $\rightarrow$ MaxPool((2,2), 'VALID') $\rightarrow$ Flatten() $\rightarrow$ Dense(500) $\rightarrow$ ReLU $\rightarrow$ Dense(10). Dense(n) denotes a fully-connected layer with output dimension n. $\text{Conv}_1(300)$, $\text{Conv}_2(300)$ denote convolutional layers layers with n filters, respectively; MaxPool((2,2), 'VALID') performs max pooling with 'VALID' padding and a (2,2) window size. Act denotes the activation: '(Batch-Norm $\rightarrow$) ReLU' depending on whether we use Batch-Normalization or not. We use batch size 128, LeCun initialization $W \sim \mathcal{N}(0, \frac{\sigma_w^2}{N_{in}}), b \sim \mathcal{N}(0, \sigma_b^2),\ \sigma_w = \sqrt{2}, \sigma_b = 0$. All experiments are conducted with

learning rate = 0.01, and we make measurements every 5 steps (k=5). Trained on CIFAR-10 without data augmentation.

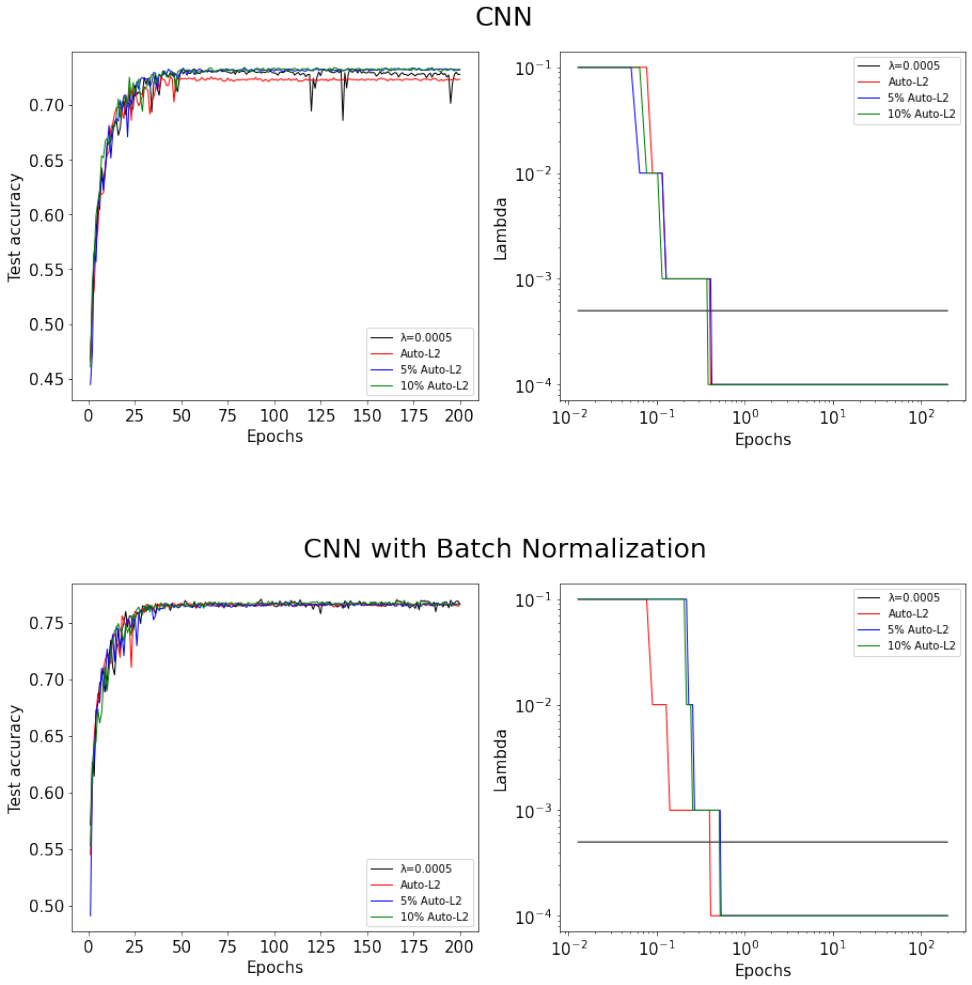

Figure 1: Compare percentage threshold AutoL_2 with original AutoL_2

We tested three fix λ [0.0001, 0.0005, 0.00001] and $\lambda = 0.0005$ achieves the highest test accuracy. Both the original AutoL_2 and adding percentage threshold couldn't have the same accuracy as the optimal λ we found. Also, we didn't observe an increase in training speed. Interestingly, all change in λ happens in the first epoch, and actually adding a percentage threshold even changes its λ earlier than original AutoL_2 without Batch-Normalization.

In figure 2 the AdvanceL_2 method achieves the highest test accuracy on the CNN with Batch-Norm setup. The test accuracy of AdvanceL_2 always start very low, we think it is because the large λ value at the beginning causes a strong regularization which makes the algorithm hard to learn. Although adding a percentage threshold can not increase the final test accuracy but it converges faster than the AdvanceL_2 method. Note that λ could change twice in a row in AdvanceL_2 method when λ is large because the factor affecting the cool down time, MIN_STEP, would not be increased.

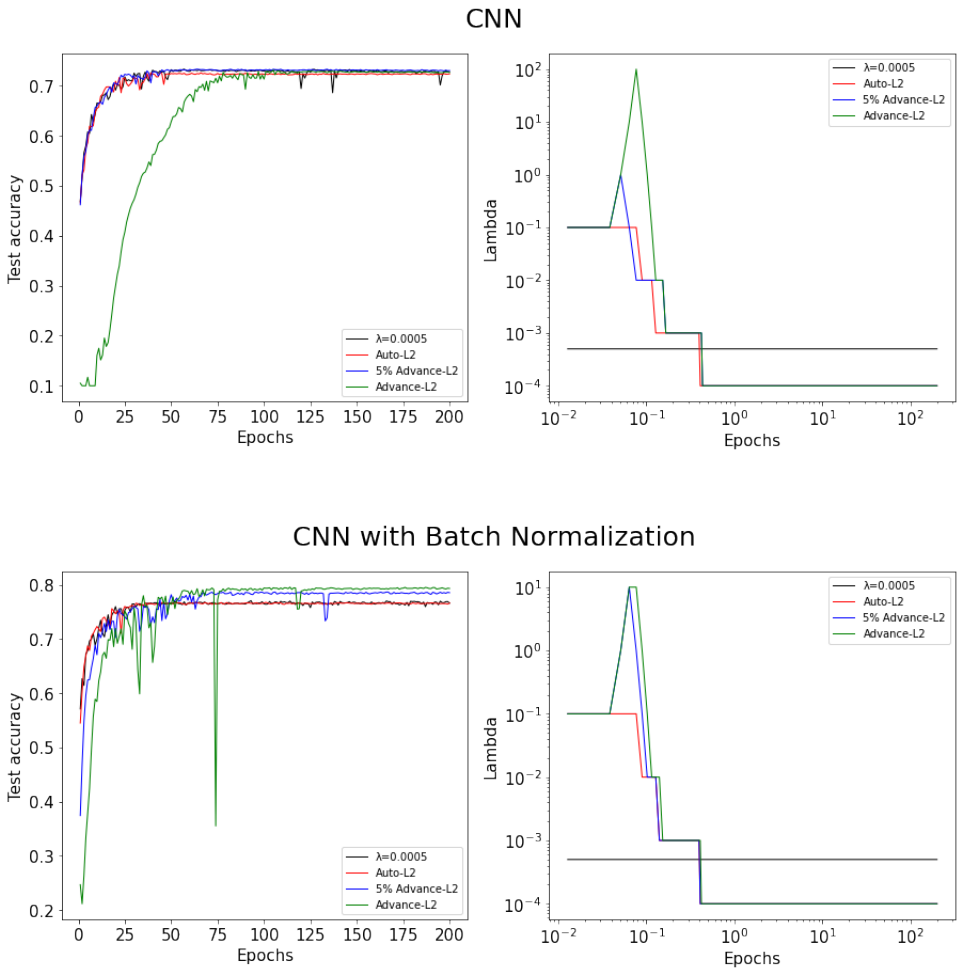

Figure 2: Add increase λ mechanism and combine with percentage threshold

5. Conclusion

In this work we proposed practical applications that can improve AutoL_2 in terms of the performance. By adding an increasing mechanism, it achieves better performance when compared against both AutoL_2 and optimal λ we found. This method works well when we add a percentage threshold in making change to the value of λ.

Our work can be further extended in several ways. Since adding 5%and10% threshold to AutoL_2 can not significantly limit the decay in λ, we can fine tune the α term to observe more interesting result. Besides, it will be worth trying to experiment with different values of decay factor in order to make the change in λ smoother. The slow start in AdvanceL_2 setup might be improved by setting a upper bound of λ, but further experiments is needed to conclude a more robust result. When training loss and error converge, we could try to increase the λ value to strengthen regularization. We left these directions for our future work.

Acknowledgments

We would like to acknowledge support for this project from the course staff and NYU HPC :D

Appendix

We use PyTorch to run all the experiments. Code can be found here: `https://github.com/powenfang/optimizing-lambda-of-regularization`.

Below are more experiments results in both without Batch-Norm and with Batch-Norm. We attempt to test AdvanceL_2 second time because of the extreme low test accuracy in the beginning, and the result shows that the relatively large value of λ might cause a poor performance in the beginning.

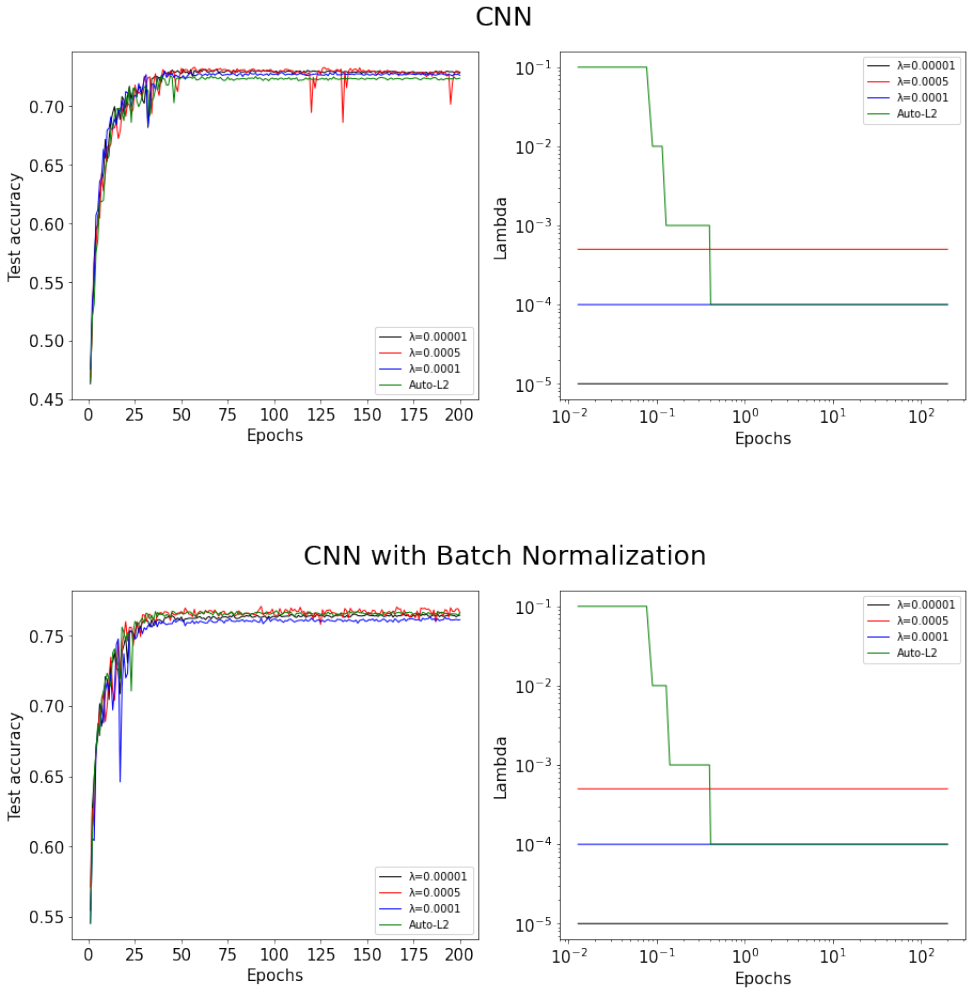

Figure 3: different values of λ and AutoL_2

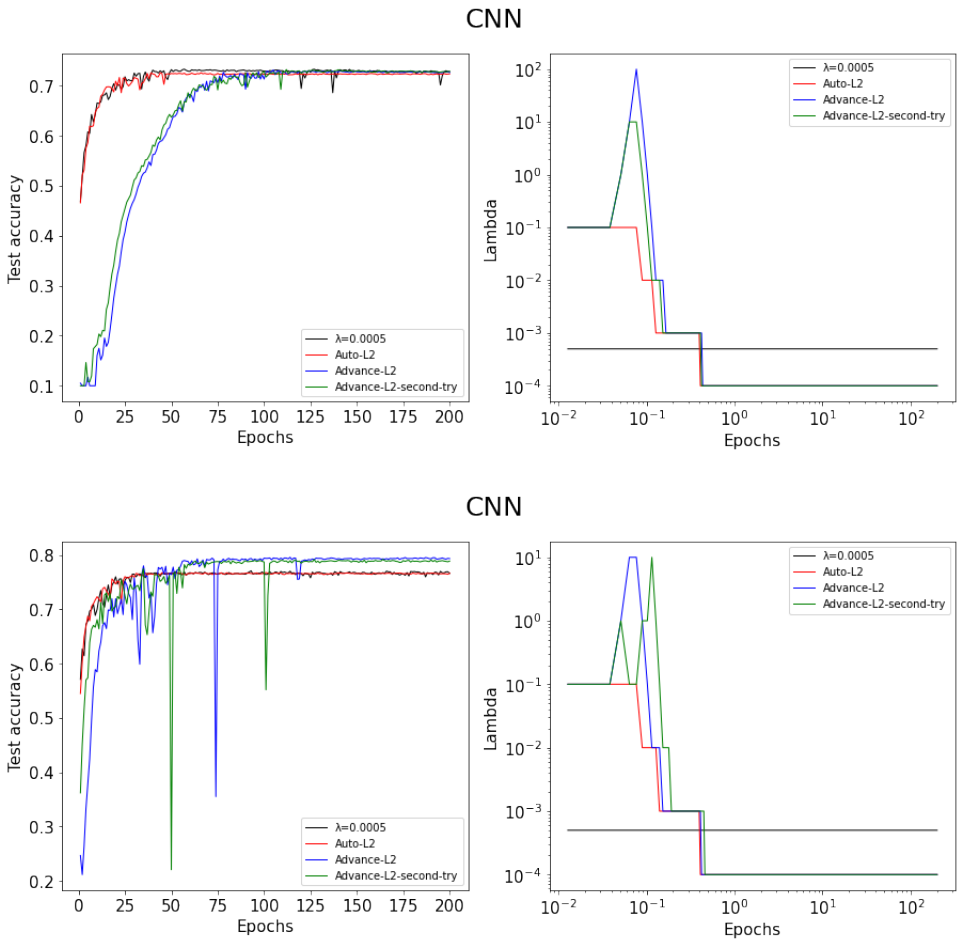

Figure 4: second attempt on AdvanceL_2

References

Anders Krogh and John A Hertz. A simple weight decay can improve generalization. *Advances in neural information processing systems*, IT-14(3):950–957, 1992.

Aitor Lewkowycz and Guy Gur-Ari. On the training dynamics of deep networks with l_2 regularization. *NeurIPS*, 2020.

Dae Hoon Park, Chiu Man Ho, Yi Chang, and Huaqing Zhang. Gradient-coherent strong regularization for deep neural networks. *CoRR*, abs/1811.08056, 2018. URL `http://arxiv.org/abs/1811.08056`.