REPORT AND ANALYSIS OF LAMBDANETWORK PERFORMANCE ON SMALL DATA SETS

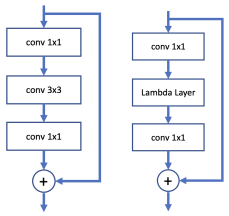
Xuan Li, Xiao Feng , Changran Huang, Xiangying Wei
Electronics and Computer Science
University of Southampton
{xl1u21,xf3u21,ch2u21,xw11u21}@soton.ac.uk

ABSTRACT

This report implements Lambda-Resnet[1] and analyzes its performance on small datasets namely ImageNette, the subset of ImageNet. With the same training configurations, we compare the differences of both training and validation loss on this dataset, with different levels of noisy labels and model architecture.

1 IMPLEMENTATION DETAILS

The lambda layer is proposed to reduce the memory cost while modelling long-range interactions between a query and a set of contexts in the self-attention mechanism. To replace the 3x3 convolution in one of the four bottleneck blocks of ResNet architecture with a lambda layer, as shown in figure 1.



(a) Conv (b) Lambda

Figure 1: Basic block of model

The original self-attention mechanism is shown as formula 1, where H is the output attention map, Q is the query, K is the key and V is the values. This computing method need huge amount of memory since its computational complexity (including spatial complexity) of the entire attention is $O\left(n^2\right)$, where n is the number of input elements.

$$H = \text{softmax}\left(QK^{\top}\right)V \tag{1}$$

The self-attention mechanism of proposed lambda layer is shown as formula 2, where keys are normalized via a softmax operation yielding normalized keys $\overline{\boldsymbol{K}}$, and $\boldsymbol{e}_{nm}$ is the position embedding. This method greatly reduce the memory usage by abstracting the keys K and values V.

$$\boldsymbol{\lambda}_n = \sum_m \left(\overline{\boldsymbol{k}}_m + \boldsymbol{e}_{nm}\right)\boldsymbol{v}_m^T = \underbrace{\overline{\boldsymbol{K}}^T\boldsymbol{V}}_{\text{position lambda}} + \underbrace{\boldsymbol{E}_n^T\boldsymbol{V}}_{\text{content lambda}} \in \mathbb{R}^{|k|\times|v|} \tag{2}$$

$$\boldsymbol{H} = \boldsymbol{\lambda}_n^T\boldsymbol{q}_n \tag{3}$$

There are two ways for computing position lambda in the paper, we implement the one that involves relative positional embedding, using einops package to do the matrix multiplication, and use lucidrains's codes[2] for lambda convolution methods.

[1] our code: https://github.com/eziaowonder/Lambda-; the original paper: https://arxiv.org/abs/2102.08602
[2] avaliable in https://github.com/lucidrains/lambda-networks

2 MODEL CONFIGURATIONS

The lambda layers involve relative positional embedding using query depth $|k| = 16$, $|h| = 4$ heads and scope size $|r| = None$, the one involving lambda convolution use the same two configurations but scope size $|r| = 23$ since these are the optimal hyperparameters with the consideration of the accuracy and the number of parameters. We use the similar training strategies, Adam optimizer with default hyperparameters, CosineAnnealingWarmRestarts with $T_0 = 5, T\_mult = 2, eta\_min = 0$, batch size is set to 64, initial learning rate is set to 1.0×10^-5, using CrossentropyLoss. As for image augmentation, we use the same configuration described in this paper.

We replace only one layer of ResNet in the second and third layer since the original paper does not has such experiments but shows that replacing the first and last layer would have the same performance, despite the difference in terms of the number of parameters and total flops. We notice that any LambdaResNet with more than two lambda layers would lead to gradient explosion after several epochs in this small dataset, which is indicated by the Nan validation loss.

The models we use f]] are ResNet18(R18), ResNet34(R34), R18 and R34 that replace the second or third basic block with lambda, using relative positional embedding(rpe) and lambda convolution(lc), denote as LR18_L2(rpe), LR18_L3(rpe), LR18_L3(lc), LR18_L3(lc), LR34_L2(rpe), LR34_L3(rpe), LR34_L3(lc), and LR34_L3(lc) respectively.

All LambdaResNet are training with Kaggle and Colab GPU, the Nvidia P100. The ResNet baselines are training with our computers for the best usage of the limited GPU quota.

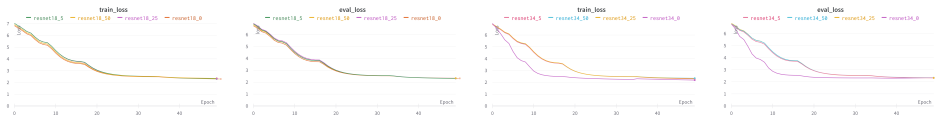
3 EXPERIMENTS IN IMAGENETTE

ImageNette is a subset of ImageNet with ten classes. We use four different levels(0%,5%,25%,50%) of noisy labels, which means a certain number of training labels are mislabeled, to see whether LambdaResNet can overcome such challenge and exceed the performance of their counterparts or not. Raw training data are availbel in this site[3].

3.1 BASELINE

Table 1: Baseline performance

Architecture	Best train loss	Best val loss	Training Time
ResNet18(0%)	2.299	2.320	1h 32m 37s
ResNet18(5%)	2.307	2.317	1h 58m 53s
ResNet18(25%)	2.332	2.289	1h 32m 37s
ResNet18(50%)	2.348	2.348	1h 35m 42s
ResNet34(0%)	2.186	2.233	1h 57m 2s
ResNet34(5%)	2.287	2.320	1h 53m 24s
ResNet34(25%)	2.324	2.330	1h 57m 2s
ResNet34(50%)	2.344	2.337	1h 41m 4s



(a) R18_train. (b) R18_eval. (c) R34_train. (d) R34_eval.

Figure 2: Baseline losses.

From figure2(a) and 2(b), the R18 has similar loss curves for four different noisy labels and converges to similar loss, whereas R34 is more sensitive to noisy labels, as the ResNet34(0%) has the best train and val loss out of all the other ResNet34 counterparts.

[3]https://wandb.ai/lambdadl/LambdaResNet/overview

3.2 LAMBDA RESNET

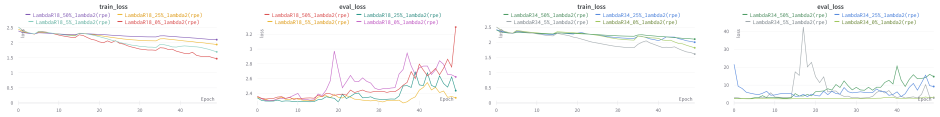
3.2.1 RELATIVE POSITION EMBEDDING(RPE)

Architecture	Best train loss	Best val loss	Training Time
LR18_L2(rpe)(0%)	1.472	2.312	2h 22m 41
LR18_L2(rpe)(5%)	1.697	2.271	2h 22m 48
LR18_L2(rpe)(25%)	1.942	2.284	2h 50m 5s
LR18_L2(rpe)(50%)	2.103	2.318	3h 11m 17
LR34_L2(rpe)(0%)	1.821	2.312	3h 4m 53s
LR34_L2(rpe)(5%)	1.614	2.444	2h 49m 37
LR34_L2(rpe)(25%)	2.005	3.533	2h 48m 43
LR34_L2(rpe)(50%)	2.107	2.410	2h 43m 2s

Table 2: LambdaResNet Layer2 rpe

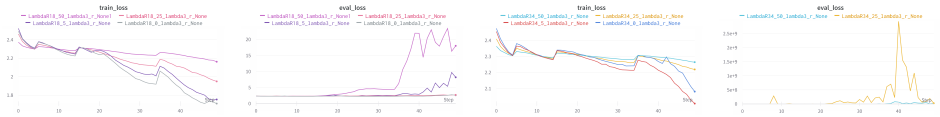
Architecture	Best train loss	Best val loss	Training Time
LR18_L3(lc)(0%)	1.313	2.367	3h 57m 33s
LR18_L3(lc)(5%)	1.467	2.297	3h 13m 3s
LR18_L3(lc)(25%)	1.904	2.363	4h 56m 32s
LR18_L3(lc)(50%)	2.038	2.434	4h 3m 59s
LR34_L3(lc)(0%)	1.744	1434.656	4h 40m 44s
LR34_L3(lc)(5%)	2.012	42.167	3h 20m 34s
LR34_L3(lc)(25%)	2.177	71622.614	3h 16m 36s
LR34_L3(lc)(50%)	2.197	3.815	4h 20m 57s

Table 3: LambdaResNet Layer3 lc



(a) LR18_l2_rpe_train. (b) LR18_l2_rpe_eval. (c) LR34_l2_rpe_train. (d) LR34_l2_rpe_eval.

Figure 3: Model outputs illustration.



(a) LR18_l3_rpe_train. (b) LR18_l3_rpe_eval. (c) LR34_l3_rpe_train. (d) LR34_l3_rpe_eval.

Figure 4: Model outputs illustration.

As can be seen from the figure 3 and 4, LambdaResNet over-fitted after about 10 epochs of validation.

For LambdaResNet18, its training and validation losses are slightly hinted at compared to the original ResNet18, but its training process is very unstable. While the training loss decreases steadily, the eval loss curve is very unstable, with a sharp irregular oscillation after about 10 epochs. An extral hour is needed for training a Lambda ResNet, and it is sensitive to the noisy labels, shown in table2 and 3.

For LambdaResNet34, the training processes are even more unstable. Especially for the LR34_L3, several losses in table3 shows that the this model has experienced gradient explosion, shown in the figure3(d) and 3(d).

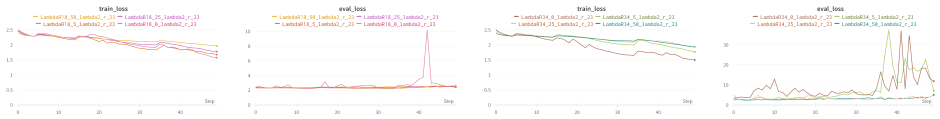
3.2.2 LAMBDA CONVOLUTION(LC)

Architecture	Best train loss	Best val loss	Training Time
LR18_L2(lc)(0%)	1.579	2.343	2h 49m 52s
LR18_L2(lc)(5%)	1.578	2.277	2h 53m 30s
LR18_L2(lc)(25%)	1.779	2.323	3h 44m 34s
LR18_L2(lc)(50%)	1.968	2.303	3h 16m 47s
LR34_L2(lc)(0%)	1.498	3.671	4h 5m 20s
LR34_L2(lc)(5%)	1.77	2.722	3h 46m 45s
LR34_L2(lc)(25%)	1.924	2.534	3h 24m 31s
LR34_L2(lc)(50%)	2.258	2.576	3h 17m 34s

Table 4: LambdaResNet Layer2 lc

Architecture	Best train loss	Best val loss	Training Time
LR18_L3(lc)(0%)	1.313	2.367	3h 57m 33s
LR18_L3(lc)(5%)	1.467	2.297	3h 13m 3s
LR18_L3(lc)(25%)	1.904	2.363	4h 56m 32s
LR18_L3(lc)(50%)	2.038	2.434	4h 3m 59s
LR34_L3(lc)(0%)	1.744	1434.656	4h 40m 44s
LR34_L3(lc)(5%)	2.012	42.167	3h 20m 34s
LR34_L3(lc)(25%)	2.177	71622.614	3h 16m 36s
LR34_L3(lc)(50%)	2.197	3.815	4h 20m 57s

Table 5: LambdaResNet Layer3 lc



(a) LR18_l2_lc_train. (b) LR18_l2_lc_eval. (c) LR34_l2_lc_train. (d) LR34_l2_lc_eval.

Figure 5: Model outputs illustration.

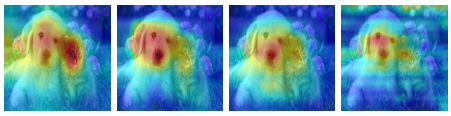
(a) LR18_l3_lc_train. (b) LR18_l3_lc_eval. (c) LR34_l3_lc_train. (d) LR34_l3_lc_eval.

Figure 6: Model outputs illustration.

Similar to the RPE models, the LC models have improved in terms of training loss compare to the RPE and baseline models, but still, have the problem of gradient explosion and unstable training process. We assume that the LambdaResNet needs a large dataset like ImageNet to prevent overfitting and also gradient explosion since it happens along with overfitting. We also tried the lambda layer on GitHub(lucidrains's code) because it has the most star and there is no official implementation. It also comes with a similar overfit and gradient explosion.

4 GradCAM of pre-trained models

Since the LambdaResNet is a kind of attention mechanism, we utilize the FullGrad[4] to show the models' attention, with the pre-trained ResNet18(R18), ResNet50(R50), lambda_resnet26t(LR26) and lambda_resnet50ts(lr50) from timm[5]. The redder the area in the figure, the higher the model's attention in that area. All the model are predicting the highest scoring category, the dog.



(a) R18. (b) LR26. (c) R50. (d) LR50.

Figure 7: Model attention visualisation.

As can be seen from figure7, the R18 model7(a) has shifted attention to the cat rather than focus on the dog and has a distributed attention over the left part of the figure. As the model gets deeper, R507(c) has attention on the head of the dog, as well as part of its body. whereas LR26, its focus on the major part of the dog as well as its body, indicate that the model learns how to model long-range interactions between a query and a structured set of context elements, i.e., associate the dog's head and its body. The deeper version LR507(d) learns to associate the areas surrounding the dog to help it determine the category.

5 Conclusion

In this project, we implement and explore the feasibility of LambdaResNet, a memory-saved attention mechanism that can easily plug into a regular deep neural network like ResNet, in a small dataset. We found that the LambdaResNet does have to improve in training loss, but also face the risk of overfitting and gradient explosion, we assume such challenge can be overcome in larger datasets like ImageNet. The gradCAM of pre-trained models shows that the LambdaResNet did learn the long-range interactions between query and context, which prove the statement in its paper.

[4]https://github.com/jacobgil/pytorch-grad-cam

[5]https://github.com/rwightman/pytorch-image-models