ADACOV: MORE EFFICIENT LEARNING BY ESTIMATING GRADIENT COVARIANCES.

Cliff Stroganoff

ABSTRACT

We conceptualize the Adam algorithm as using a diagonal estimate of the covariance matrix to scale the optimization step. We then choose to expand the estimate of the covariance matrix to include some off-diagonal elements. An experiment training a neural network on the CIFAR-100 dataset is performed. Including covariances between the elements inside convolutional filters yields quicker learning than the regular Adam algorithm.

1 Introduction

Popular optimizers like RMSProp and Adam[1] use an adaptive learning-rate feature to calculate individul learning rates for each parameter. We can think of scenarios where learning rate would benefit from being boosted in a certain direction that does not coincide with one of the axes of our coordinate system. In these cases, we might risk that this boost is cancelled by a need to dampen the learning-rate in a different direction.

We thus modify the popular Adam algorithm to account for these types of correlations and see if faster learning can be achieved.

2 Theory

2.1 The Adam update step

Both RMSProp and Adam optimizers calculate individual adaptive learning-rates, based on estimates of the uncentered variance. If we have a vector containing the second moment estimates, $\vec{v}$, the individual learning rate,α_i, for an element $\hat{v}_i$ is:

$$\alpha_i = \alpha \frac{1}{\sqrt{v_i} + \epsilon} \tag{1}$$

We choose to slightly alter this expression to:

$$\alpha_i = \alpha \frac{1}{\sqrt{v_i + \epsilon^2}} \tag{2}$$

The two expressions are equivalent when $\epsilon \ll \sqrt{v_i}$ or $\epsilon \gg \sqrt{v_i}$. But behave slightly different in the area where $\sqrt{v_i}$ and ϵ are the same order of magnitude.

Conceptualizing the individual learning rates in this way, allows us to write the update step in Adam as:

$$\vec{\theta} \leftarrow \vec{\theta} - \alpha(\hat{C} + \epsilon^2 I)^{-\frac{1}{2}} \vec{\hat{m}}, \tag{3}$$

where $\hat{C}$ is a diagonal matrix containing the variance estimate $\vec{\hat{v}}$. $\vec{\hat{m}}$ is the first moment estimate of the gradient.

Seeing the matrix $\hat{C}$ as a diagonal estimate of the covariance matrix, we can now choose to swap it out with a different estimate of the covariance matrix which do contain (some of the) covariances between parameters.

This view of "Preconditioning" the gradient with the inverse square root of the covariance matrix was explored by Ida[2] .

3 Methods

A neural network with 6 convolutional layers with 3x3 kernels was created. It was trained on the CIFAR-100 dataset using both AdaCov optimizer and Adam optimizer. It was trained for 20 epochs. The batch size was 128.

Training was done on a machine with a nVidia 1080Ti GPU and a Intel i9-7900X CPU.

We use Adam with the standard parameters $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$. We use the same parameters for the AdaCov model, except for ϵ, which was is discussed below.

The same random seed is used in both cases when initializing the model.

3.1 Choosing what covariances to include

We choose to include covariances internally in the convolution kernels. We do this since these kernels typically end up highly correlated after learning is complete. And we hypothize that the gradients also will be correlated. Also, the kernels are typically small. With 3x3 being a popular size.

To store the covariance matrix estimate for a convolutional "weight", we need a tensor of size $[n_{out}, n_{in}, W\text{x}H, W\text{x}H]$

3.2 Setting eps. Condition number

We used $\epsilon = 10^{-8}$ for everything except the convolutional weights.

The estimates of the covariance matrices needs some iterations before they get a good condition number. We chose to solve this by setting ϵ slightly higher for the convolutional weights: $\epsilon = 10^{-5}$. Also, we zeroed out the off-diagonal elements of the covariance matrices at the start of training, and brought them in slowly. The off-diagonal elements where scaled by:$1.0 - 10.0^{-t/390}$. 390 is about one epoch. This way, at the start of training, the optimizer behaves like Adam.

4 Results

The learning curve of AdaCov versus Adam is shown in Figure 1. Increase in training time in Table 1. Although this is highly sensitive to computer hardware, batch size, image size, model configuration etc.

Table 1: Increase in training time for AdaCov vs Adam

Filter size	Train Time Increase
3x3	28 %
5x5	169 %

5 Further Work

5.1 Increase Performance

More efficient ways to calculate $(\hat{C} + \epsilon^2)^{-\frac{1}{2}}\vec{\hat{m}}$.

Find ways where one doesnt have to calculate the inverse square root at every iteration.

At each iteration the covariance matrix estimate is scaled and then subjected to a (small) rank-1 update. There exists methods for calculating the inverse square root when a matrix is subjected to these updates[3]. Also, there are iterative methods like the Denman-Beavers iteration that calculates the matrix square root and its inverse[4]. Allen has some numerical methods for calculating the product of a matrix square root and a vector.[5]

Refinement of eigenvectors is a possible avenue. Ogita has an usable algorithm[6]

6 Conclusion

Method yields more efficient learning and higher memory consumption. Time-concumption scales linearly with number of conv. layers and number of channels in these layers. It scales horribly with filter size. Memory concumption does not scale favorably with filter size either ...

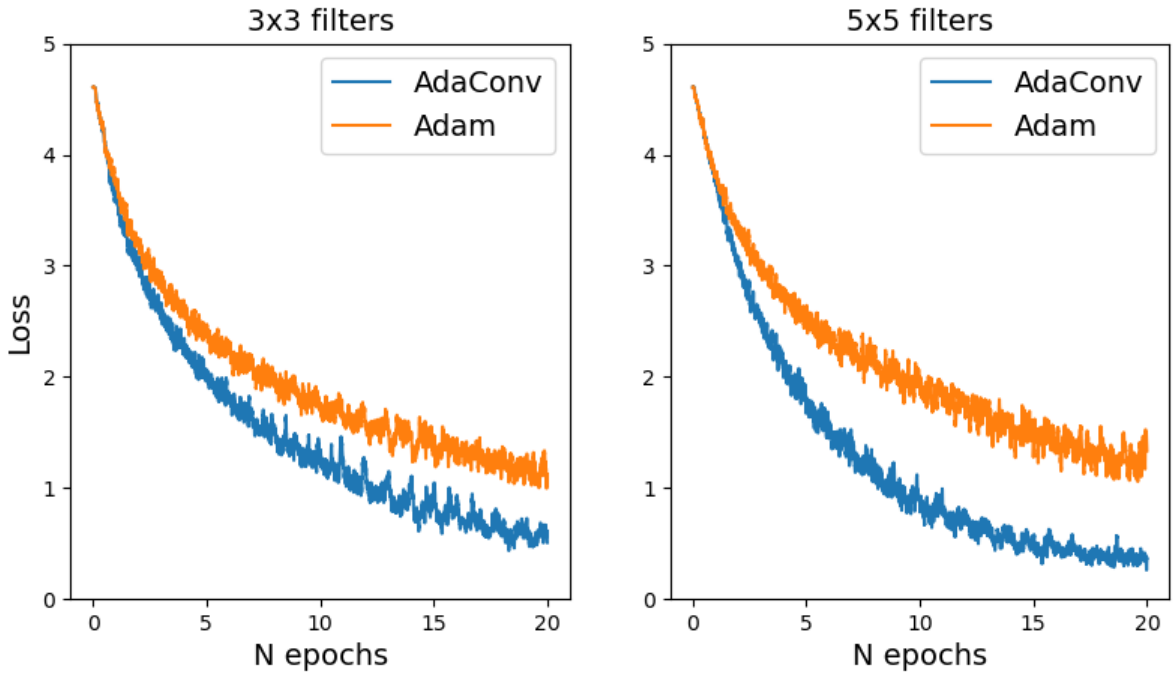

Figure 1: Learning curve of AdaCov vs Adam. CategoricalCrossEntropy loss vs number of epochs. Smoothed by a moving average of 10 iterations.

The method becomes increasingly attractive the longer the forward/backward pass takes. Could be useful in models that contains convolutional layers, but where most of the time consumption of the forward/backward pass is taken by other parts of the model.

References

[1] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[2] Yasutoshi Ida, Yasuhiro Fujiwara, and Sotetsu Iwamura. Adaptive learning rate via covariance matrix based preconditioning for deep neural networks. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, aug 2017.

[3] Shany Shumeli, Petros Drineas, and Haim Avron. Low-rank updates of matrix square roots, 2023.

[4] Eugene D. Denman and Alex N. Beavers. The matrix sign function and computations in systems. *Applied Mathematics and Computation*, 2(1):63–94, 1976.

[5] E.J. Allen, J. Baglama, and S.K. Boyd. Numerical approximation of the product of the square root of a matrix with a vector. *Linear Algebra and its Applications*, 310(1):167–181, 2000.

[6] Takeshi Ogita and Kensuke Aishima. Iterative refinement for symmetric eigenvalue decomposition. *Japan Journal of Industrial and Applied Mathematics*, 35(3):1007–1035, Nov 2018.