Lookahead Optimizer: *k* steps forward, 1 step back

https://arxiv.org/abs/1907.08610

2017B4PS0559P, **Akash Palrecha**

2017A3PS0242P, **Dhananjay Vijayraghavan**

2017A3PS0244P, **Derek Shawn Mathias**

Paper Summary

- Project repository : https://github.com/akashpalrecha/Lookahead.git
- A Conda environment file has been provided in the repository to recreate the exact conditions of the experiments.
- The paper introduces a new optimizer for deep neural networks called “Lookahead”.
- The proposed method acts as a thin wrapper over existing state of the art optimizers like SGD, AdamW, RMSProp, etc.
- The paper claims that Lookahead performs better than existing techniques on tasks such as image classification, language modelling and neural machine translation.

Paper Summary

- Lookahead Algorithm (Simplified for the presentation):
  - Choose inner optimizer (SGD, AdamW, etc.)
  - Perform *K* iterations in the usual way
  - Linearly interpolate between weights of 1st and Kth iteration using a parameter *alpha* like so:

$$\phi_t \leftarrow \phi_{t-1} + \alpha(\theta_{t,k} - \phi_{t-1})$$

  - Repeat till terminating condition (convergence, epochs, etc.)

Intuition behind Lookahead

- We believe that the Lookahead update is analogous to using the Wisdom of the crowd where the average judgement of a group of individuals is more accurate than any individual judgement.
- Instead of relying on single updates to the weights by particular batches of training data, this method encourages an inner optimizer to "look ahead" for *K* iterations, and then cancels out mistakes made by those individual updates by performing an interpolation update.
- The update also prevents the network from overfitting to any particular sample of data by repeatedly interpolating between each set of *K* iterations. This can be seen as a form of regularization and this is evident from our experiments as we show further in this presentation

A word about resetting states when using Lookahead

- The lookahead algorithm advocates interpolating between two sets of weights that are essentially *k-iterations* away from each other.
- For stateful optimizers (momentum, AdamW, RMSProp), this means that the state terms for each parameter: momentum, exponential moving averages and squared averages, etc. also need to be taken care of.
- We tried two strategies:

1. Not modifying the state variables after lookahead updates
2. Resetting the state variables after each Lookahead update.(as suggested in the paper)

- The results that follow spell out the details of both of these strategies.

Hyperparameters and data Augmentation

- For all experiments, we used the suggested data augmentation in the paper:
  - 4 px zero padding for all images in CIFAR10, CIFAR100
  - Random 32x32 crop from padded images in CIFAR10, CIFAR100
  - 50% Horizontal flip transformation
  - No data augmentation for Imagenette (Imagenet)
- Learning Rates, weight decay values, momentum, alphas and betas: for each optimizer, we have used one of the values over which a gird search was made in the original paper. The Jupyter notebooks attached with the project detail out every hyperparameter used.

Results: CIFAR10 without State Reset

	Training Loss	Validation Loss	Accuracy
Lookahead/SGD	0.2717	**0.6554**	82.33%
SGD	0.003209	0.7999	**82.59%**
AdamW	**0.0001259**	1.6	80.57%
RMSProp	0.000256	1.876	80.23%
Lookahead/AdamW	0.2717	0.9423	74.71

Observations:

1. Lookahead/SGD generalizes the best with least validation loss.
2. Accuracy for Lookahead is not the highest, but is only very marginally lower than SGD
3. AdamW clearly overfits the data
4. Lookahead/AdamW is the worst performer.

Results: CIFAR100 without State Reset

	Training Loss	Validation Loss	Accuracy
Lookahead/SGD	0.07356	**2.441**	**52.61%**
SGD	0.07333	2.462	52.15%
AdamW	**0.007684**	3.772	48.89%
RMSProp	0.3934	40541.424	46.21%
Lookahead/AdamW	1.722	12.094	44.01%

Observations:

1. Lookahead/SGD generalizes the best with least validation loss.
2. Accuracy for Lookahead is the highest, despite a higher training loss.
3. AdamW clearly overfits the data
4. Lookahead/AdamW and RMSProp are the worst performers.

Results: CIFAR10 with State Reset

	Training Loss	Validation Loss	Accuracy
Lookahead/SGD	0.01315	**0.6195**	**85.53%**
SGD	0.004024	0.6882	85.24%
AdamW	**0.0005191**	1.296	83.88%
RMSProp	0.0007434	1.396	83.6%
Lookahead/AdamW	0.5555	164851.2	77.1%

Observations:

1. Lookahead/SGD generalizes the best with least validation loss.
2. Accuracy for Lookahead is the highest, despite a higher training loss.
3. AdamW clearly overfits the data
4. Lookahead/AdamW is the worst performer.

Results: CIFAR100 with State Reset

	Training Loss	Validation Loss	Accuracy
Lookahead/SGD	0.1046	**2.02**	57.33%
SGD	0.06247	2.163	**57.34%**
AdamW	**0.006513**	3.44	54.27%
RMSProp	0.4939	13208.707	51.78%
Lookahead/AdamW	2.473	24347.662	43.58%

Observations:

1. Lookahead/SGD generalizes the best with least validation loss.
2. Accuracy for Lookahead is not the highest, but is only very marginally lower (0.01%) than SGD
3. AdamW clearly overfits the data
4. Lookahead/AdamW and RMSProp are the worst performers.

For an interactive demo of all the results with graphs, visit:
Lookahead Experiment

Imagenette, and a few other details.

- Imagenette is a smaller subset of 10 classes of Imagenet which allows for rapid experimentation with results that have been shown to carry over to Imagenet. *(https://github.com/fastai/imagenette)*
- Number of images: 120,000. Classes: 10.
- Mixup is a state-of-the-art data augmentation technique that linearly combines training samples and is shown to improve results over all image recognition tasks : https://arxiv.org/abs/1710.09412
- OneCycle : Cosine Annealing training schedule for superconvergence : https://arxiv.org/abs/1803.09820

Results: Imagenette with State Reset, *OneCycle*, *Mixup* and *heavy data augmentation*

	Training Loss	Validation Loss	Accuracy	Top-5-Accuracy
Lookahead/SGD	0.899	**0.3903**	**88.45%**	**98.78%**
SGD	0.9493	0.5358	87.64%	98.68%
AdamW	**0.8814**	0.5463	87.74%	98.53%

Observations:

1. Lookahead/SGD generalizes the best with least validation loss.
2. Accuracy for Lookahead is the highest by a good margin. Top-5-Accuracy is also the highest for Lookahead/SGD.
3. AdamW clearly overfits the data

Conclusions from Experiments conducted

- AdamW almost always overfits the data with a very low training loss, high validation loss, and lower accuracy.
- Lookahead consistently gets the least validation accuracy showing the best generalizability, which is more important than having a lower training loss.
- Lookahead almost always gets the highest accuracy and only sometimes loses to SGD with a very small margin.
- Lookahead/SGD offers very small, marginal advantages over vanilla SGD.
- Lookahead/AdamW consistently performs badly. Possible reason is the behavior of the exponential average and squared average states stored for each parameter after doing lookahead updates.

Conclusions from Experiments conducted

- RMSProp does not seem like a good choice as an optimizer for computer vision problems.
- The experiments reinforce the fact: lower training loss isn't always better.
- Again, Lookahead/SGD generalizes the best, and very consistently so.
- Resetting the state in both Lookahead/SGD and Lookahead/AdamW increased accuracies by about 3%. But the tables show the same increase in all other optimizers too. Hence, we cannot surely say if resetting the state helped here. It can simply be related to using a different random seed for the particular experiment.

Issues with getting it to work right: Stateful Optimizers

- The "inner optimizer", let's call it IO, can sometimes have other hyperparameters which may not all behave very well when working under Lookahead.
- For example, in SGD + Momentum, each parameter of the model will have an individual state with it's momentum. The paper suggests resetting the momentum after each Lookahead update. We experimented with a version that does not reset momentum. The results follow in the presentation.
- For Adam based optimizers, states like the exponential averages, exponential moving averages, etc need to be dealt with carefully too. Unfortunately, the paper does not discuss Lookahead + Adam and neither does it suggest a strategy for this specific case. We tried both resetting the state and not altering it after lookahead updates, but either approach did not give good results.

Issues with getting it to work right: Computational Graph

- Current implementations of popular deep learning libraries such as PyTorch, TensorFlow, MXNet, etc. all involve computational graphs being built as calculations are carried out so as to support features like automatic differentiation and network optimization by compilers.
- When storing the slow-weights at each 1st step in the lookahead algorithm, we need to make sure to not just clone the parameter weights that we're copying but also to detach them from the graph. This can be simply achieved in PyTorch by a statement such as: x.detach()
- This is a very important detail to get right for two reasons:
  - The network will still train to an acceptable accuracy even if this weren't taken care of, but it won't be as accurate.
  - This makes it hard to debug this issue, as there are no errors during training.

Remarks

- Our final accuracies do not match up to the the paper's results for the following reasons:
  - We did not do a hyperparameter grid search over multiple parameters for each optimizer as that would have taken days of continuous training.
  - The paper has not mentioned the final parameters used
  - As a result, our training is probably a little less efficient as compared to Geoffrey Hinton's.

Things we didn't do

- Polyak Averaging: The paper does not mention any details about the exact version of the Polyak algorithm being used (SWA, Rupert, etc.)
- Training on ImageNet : Running multiple experiments on the ImageNet dataset was not feasible on Collab and would have been too costly on GCP/AWS instances.
- Language Modelling: The paper requires training a huge language model for 750 + 700 = 1450 epochs on multiple optimizers. Such experiments cannot be run on Collab and we simply did not have the compute to do so otherwise.
- Neural Machine Translation: The paper trained a transformer based architecture on a single TPU Core with 8 workers each and a batch size of 2048. For the same reasons stated above, we did not perform this experiment too.

Thank you!