Sideways: Depth-Parallel Training of Video Models
Supplementary Material

Mateusz Malinowski
mateuszm@google.com

Grzegorz Świrszcz
swirszcz@google.com

João Carreira
joaoluis@google.com

Viorica Pătrăucean
viorica@google.com

DeepMind
London, U.K.

Abstract

We propose Sideways, an approximate backpropagation scheme for training video models. In standard backpropagation, the gradients and activations at every computation step through the model are temporally synchronized. The forward activations need to be stored until the backward pass is executed, preventing inter-layer (depth) parallelization. However, can we leverage smooth, redundant input streams such as videos to develop a more efficient training scheme? Here, we explore an alternative to backpropagation; we overwrite network activations whenever new ones, i.e.*, from new frames, become available. Such a more gradual accumulation of information from both passes breaks the precise correspondence between gradients and activations, leading to theoretically more noisy weight updates. Counter-intuitively, we show that Sideways training of deep convolutional video networks not only still converges, but can also potentially exhibit better generalization compared to standard synchronized backpropagation.*

1. Introduction

In the supplementary material, we complement the main paper with extra details. In Section 2, we provide more details about the datasets that we use in the experiments. In Section 3, we give a more detailed exposition on the hyperparameters such as learning rate or resolution that we use in our experiments. In Section 4, we show a few reconstructions of the auto-encoding models trained either with *Sideways* or *BP*. Finally, in Section 5, we show additional results by tracking various properties of the models during their training.

2. Datasets

We have conducted experiments on the following three datasets. The first two are action recognition datasets consisting of natural video clips. The last one is a synthetic video dataset.

HMDB51 is a widely used dataset for action recognition that has 6770 video clips representing 51 actions [6]. Video clips run at 30fps. In our experiments, we use the first train and test splits, similar to ablation studies in [4, 8]. This is the smallest real-world dataset in our experiments, and since training is fast, we therefore use mainly this setting to study our models in details.

UCF101 is another popular dataset for action recognition [9]. It has 13320 videos clips and 101 human actions. Actions include pizza tossing, typing, playing cello, skiing, etc. Default frame-rate is 25fps, with the duration of 7.21sec on average. Each frame has resolution 320-by-240. We use train and test splits in our studies. In our experiments, we find this dataset to be of particular interest due to its size, complexity, and realism.

CATER dataset [2] provides synthetically generated videos of moving simple 3D objects. We use only the video frames and we set up an unsupervised auto-encoding task. These videos have two desired properties – i) they are visually simple, and ii) they have diverse motion patterns of various objects – making it an excellent benchmark for *Sideways*. We use the pre-generated video sequences provided by the authors in the *all_actions* subset, consisting of 3861 training video sequences and 1639 test sequences, each with 300 frames, having 320-by-240 pixels resolution at 24fps.

3. Training Details

By default, we use gradient clipping by value 1.0. We use a bias term only on the last linear layer used to produce logits. We considered three different popular optimizers: SGD, SGD with momentum, and Adam [5]. We use default hyperparameters for all of them and our observation is that *Sideways* is stable under all the considered training algorithms so we report results with Adam only. We use a warm-up scheme by linearly interpolating learning rate in each iteration between 0 at the beginning and the initial value at the 5-th epoch. Later we drop the learning rate by dividing it by 10 at 100-th and 200-th epochs. We use Tensorflow 2 [1] to implement our framework.

In some experiments we use batchnorm [3] and dropout [10] in linear layers (0.9 as [8]). In most training experiments, we use a single GPU (V100) – but measure parallelization speedup later in the section using multiple GPUs. We decouple the weight decay term from the loss [7] and find that some amount of weight decay is beneficial for generalization of the VGG-net models. We do a hyper-parameter search over the weight decay coefficient (possible values are: $0.0, 10^{-4}, 10^{-3}, 10^{-2}$), and over the initial learning rate (either 10^{-4} or 10^{-5}), fairly for both types of training.

We train on videos with the per-frame resolution 112-by-112 and a sequence length 64 randomly cropped from the whole video clip as a contiguous subsequence. We also use random flipping during training, and this augmentation is performed consistently over all the frames in every clip. At test time we simply take a center crop of a video frame. In all cases, we use torus-padding (video frames are repeated) whenever a sampled video clip is shorter than 64 and at test time we evaluate over the whole video sequence. In practice we use a fixed batch size of 8 videos for training.

For the auto-encoding experiments with CATER, we use square crops of 240×240 pixels, extracted randomly during training and central crops of the same size during testing.

4. Auto-encoding Task

Here, we complement our quantitative results on the auto-encoding task with CATER [2] from the main paper with a few concrete reconstructions.

We show qualitative results of the *BP* and *Sideways* outputs in Figure 1 and Figure 2. Shapes, colors, as well as many fine-grained attributes such as metallic materials are properly decoded. We have found that the model trained with *BP* is successful at decoding coarser objects attributes, but is slightly less accurate with metallic materials. As mentioned above, because of the blocking mechanism of the *BP* algorithm and the high frame rate of the input, the method needs to discard input frames to not accumulate latency. In these cases, the last produced output is copied to compensate for the low output rate, resulting in identical output frames at different time steps over some time interval – and different ones at the beginning of the next computation cycle (this is best visible in the 1st row of *BP* or *Sideways* in the figures).

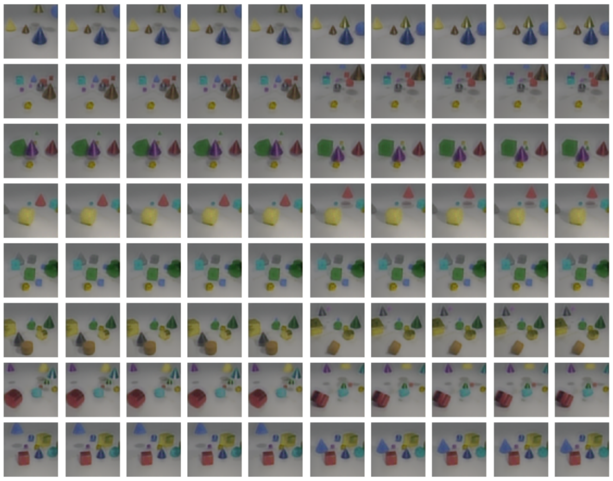

Figure 1: Auto-encoding results for *BP*. Each row shows an individual video sequence consisting of, in our case, 64 frames. For the sake of visualization, we sub-sample 10 consecutive middle frames from the outcome.

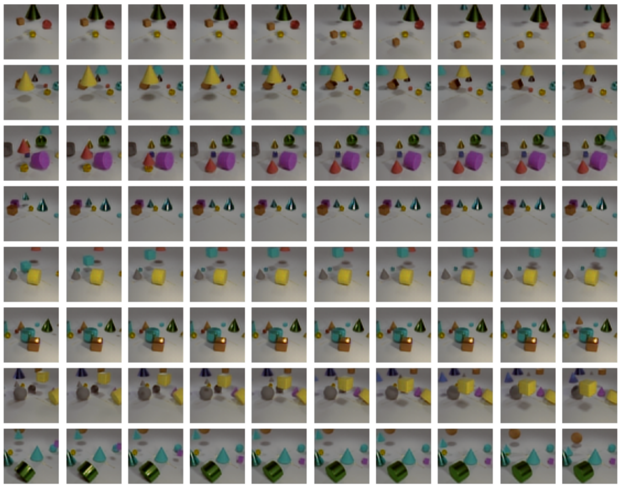

Figure 2: Auto-encoding results for *Sideways*. Each row shows an individual video sequence consisting of, in our case, 64 frames. For the sake of visualization, we sub-sample 10 consecutive middle frames from the outcome.

5. Training Dynamics

We also augment the main paper with further results showing training dynamics of the models trained either with *Sideways* or *BP*. We report these results on the HMDB51 dataset in Figure 3. As we explain in the main paper, we observe a relatively high training accuracy of the models trained with *Sideways* with unfavourable hyper-parameters, even though losses appear to be quite unstable. We conclude that this behavior is due to the over-confidence of the network. Figure 4 shows the training accuracy results.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Rohit Girdhar and Deva Ramanan. Cater: A diagnostic dataset for compositional actions and temporal reasoning. *International Conference on Learning Representations (ICLR)*, 2020.

[3] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015.

[4] Longlong Jing and Yingli Tian. Self-supervised spatiotemporal feature learning by video geometric transformations. *arXiv preprint arXiv:1811.11387*, 2018.

[5] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 12 2014.

[6] Hildegard Kuehne, Hueihan Jhuang, Estíbaliz Garrote, Tomaso Poggio, and Thomas Serre. Hmdb: a large video database for human motion recognition. In *2011 International Conference on Computer Vision (ICCV)*, 2011.

[7] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

[8] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[9] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[10] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research (JMLR)*, 15(1):1929–1958, 2014.

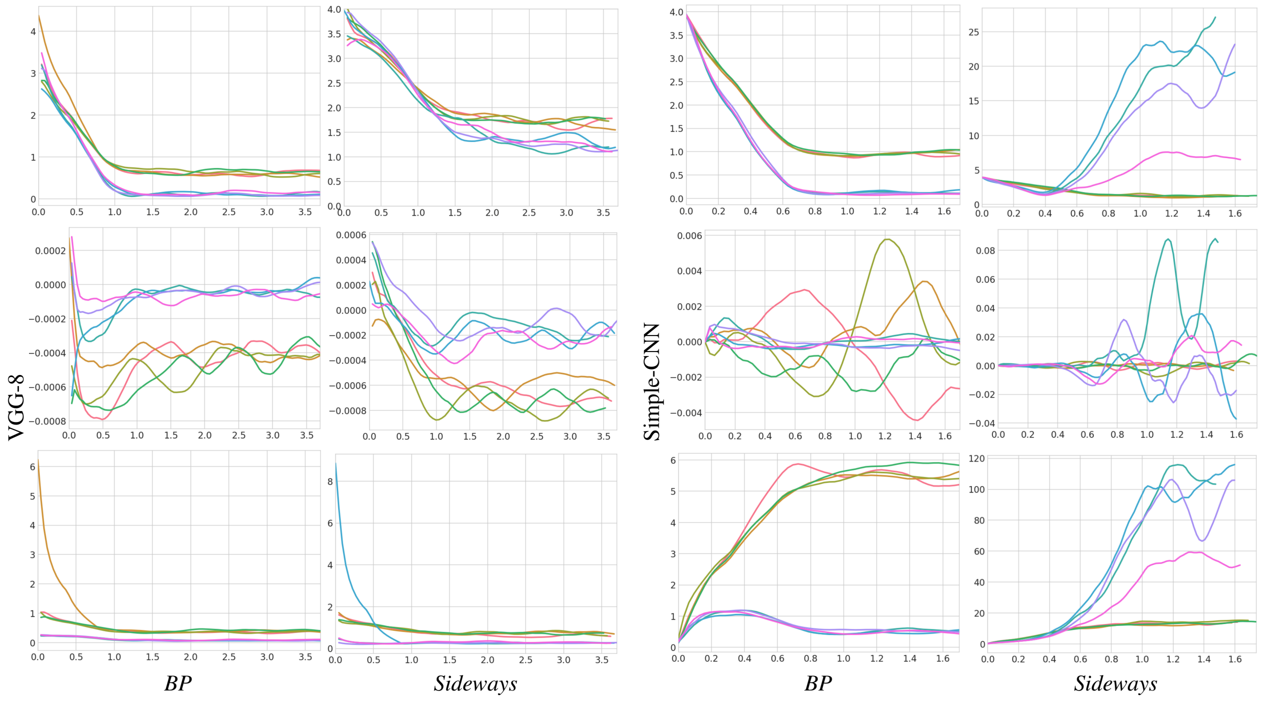

Figure 3: Training dynamics of Simple-CNN and VGG-8 with different models of computations. Experiments are conducted on the HMDB51 dataset. Different colors denote different hyper-parameters (red, green, olive, orange refer to the initial learning rate 10^{-5} and teal, pink, violet, blue to 10^{-4}, all with various weight decay). On the x-axis, we report number of iteration steps, in 10^5 scale. On the y-axis, we report, from the top to bottom: loss, mean of the gradients, and average gradient magnitude (l2-norm). Note that the figures have different y-limits to allow a detailed visualization of the training dynamics as training progresses.

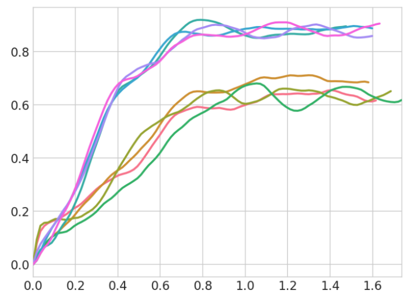

Figure 4: Training of Simple-CNN with the *Sideways* algorithm on HMDB51. Different colors denote different hyper-parameters (the same as Figure 3). On the x-axis, we report number of iteration steps, in 10^5 scale. On the y-axis, we report, accuracy numbers.