SUPPLEMENTARY MATERIAL

A EXPERIMENTAL SETTINGS

We are using Flax (`https://github.com/google/flax`) which is based on JAX (Bradbury et al., 2018). The implementations are based on the Flax examples and the repository `https://github.com/google-research/google-research/tree/master/flax_models/cifar` (Foret et al., 2020). We include a simple Flax implementation of ABEL with the submission and the corresponding minor modifications to run ABEL on the Flax examples and flax_models/cifar.All datasets are downloaded for tensorflow-datasets and we used their training/test split.

All experiments use the same seed for the weights at initialization and we consider only one such initialization unless otherwise stated. We have not seen much variance in the phenomena described, see Table $1, 2$ description. All experiments use cross-entropy loss.

We run all experiments in v3-8 TPUs. The WideResnet 28-10 CIFAR experiments take roughly 1h per run, the ResNet-50 Imagenet experiments take roughly 5h per run, the PyramidNet CIFAR experiments take roughly 50h per run, the Transformer experiments take roughly 24h per run, the ALBERT fine-tuning experiments take around $10 - 20$minutes per task.

Here we describe the particularities of each experiment in the figure/tables.

Table 1. All models use momentum with a momentum parameter of 0.9 and none of them uses dropout. ABEL considerations: learning rate decay 0.1 for ImageNet and SVHN and 0.2 for CIFAR-100. We force ABEL to decay at 85% of the total epoch budget by the same decay factor. We run the CIFAR-10 and CIFAR-100 WideResnet experiments with three different seeds and report the average accuracy, the standard deviation across experiments for all schedules and CIFAR-10, CIFAR-100 is $\leq 0.1\%$, except for ABEL and CIFAR-100 which has a standard deviation of 0.2% (the standard error of the mean is 0.1%), from these observations we only consider a single run for the other experiments. For the CIFAR and WRN experiments, the gradient norm is clipped to 5.

- Resnet-50 on Imagenet: learning rate of 0.8, L_2 regularization of 0.0001 (we do not decay the batch-norm parameters here), label smoothing of 0.1 and 90 epochs. All experiments include a linear warmup of 5 epochs. Standard data augmentation: horizontal flips and random crops. Step-wise decay: multiply learning rate by $\alpha = 0.1$ at $30, 60$ epochs. These experiements take around 4 hours on a v3-8 TPU.
- Wide Resnet 16-8 on SVHN: learning rate of 0.01, L_2 regularization of 0.0005, batch size of 128, no dropout and evolved for 160 epochs. Training data includes the "extra" training data and no data augmentation. Step-wise decay: multiply learning rate by $\alpha = 0.1$ at $80, 120$ epochs. These experiments take around 7 hours/run on a v3-8 TPU.
- Wide Resnet 28-10 on CIFAR10/100: learning rate of 0.1, L_2 regularization of 0.0005, batch size of 128 and evolved for 200 epochs. Standard data augmentation: horizontal flips and random crops. Step-wise decay: multiply learning rate by $\alpha = 0.2$ at $60, 120, 160$ epochs. These experiments take around 50 minutes/run on a v3-8 TPU.
- Shake-Drop PyramidNet on CIFAR100: learning rate of 0.1, L_2 regularization of 0.0005, batch size 256, evolved for 1800 epochs. Uses AutoAugment and cutout as data augmentation. Because training this model takes a long time and weight bounces generally take longer at smaller learning rates, we found it convenient to average the weight norm every five epochs when using ABEL in order to avoid the decays from being noise dominated (without this, the third decay happens too early and the test error increases by 0.2%). These experiments take around 58 hours/run on a v3-8 TPU.
- VGG-16 on CIFAR10: learning rate of 0.05, L_2 regularization of 0.0005, batch size 128, evolved for 200 epochs. Basic data augmentation and no batch norm. These experiments take around 30 minutes/run on a v3-8 TPU.

Table 2 NLP, RL. All these experiments use ADAM, no dropout nor weight decay.

- Base Transformer trained for translation: learning rate: $1.98 \cdot 10^{-3}$, learning rate warmup of 10k steps, evolved for 100k steps, batch size 256 uses reverse translation. Simple decay:

Task	MNLI(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Average
Linear Decay	82.6/83.4	88.1	92.1	92.8	58.5	91.0	88.7	70.8	83.1
Simple Decay	82.9/83.5	87.6	91.1	92.1	57.0	91.2	88.2	73.3	82.9

Table S1: Evaluation metric is accuracy except for CoLA (Matthew correlation), MRPC (F1) and STS-B (Pearson correlation).

decay by 0.1 at 90k steps. The translation tasks correspond to WMT'17. These experiments take around 22 hours/run on a v3-8 TPU.

- ALBERT and Glue fine-tuning. fine-tuned from the base ALBERT model of https://github.com/google-research/albert. Tasks: All tasks were trained for 10k steps (including 1k linear warmup steps) for four learning rates: $\{10^{-5}, 2\cdot 10^{-5}, 5\cdot 10^{-5}, 10^{-6}\}$, reported best learning rate, batch size 16. See table S1 for the specific scores. Simple decay: decay by 0.1 at 0.8 of training. The individual scores are summarized in S1. These fine-tuning experiments take around 30 minutes/run per task on a v3-8 TPU.
- PPO: learning rate $2.5\cdot 10^{-4}$, batch size 256, 8 agents. Evolved for 40M frames. Simple decay: decay by 0.1 at 0.9 of training. Rest of configuration is the default configuration of https://github.com/google/flax/examples. Reported score is the average score over the last 100 episodes. We have three runs per task and schedule. The scores with a 95% confidence interval are: Seaquest: Linear: 1750 ± 260, Simple: 1850 ± 40; Pong: Linear 21.0 ± 0.01, Simple 20.7 ± 0.5; Qbert: Linear 22300 ± 2950, Simple 23000 ± 1800. We use one Nvidia P100 for these experiments, and each run takes around 10 hours.

Table 2 Vision.

These experiments are the same as table 1 but without L_2. The simple decay is defined by evolving with a fixed learning rate during 85% of the time and then decay it (with decay factor 0.1 for ImageNet and 0.2 for CIFAR). In the case of ImageNet+Adam, the learning rate was reduced to 0.0008. For simple decay with $L_2 \neq 0$, the test error would often increase after decaying, in such case, we choose the test error at the minimum of the training loss, see fig S10a for an experiment with this behaviour.

Fig S6a: VGG-5 trained with $\eta = 0.25, \lambda = 0.005$ and decayed by a factor of 10 at the respective time.

Rest of figures. Small modifications or further plots of previous experiments, changes are specified explicitly.

B EXPERIMENTAL DETAILS

B.1 THE MILD DEPENDENCE ON THE LAST DECAY EPOCH

In figure S1 we run again the CIFAR experiments with ABEL for different values of ‘last_decay_epoch‘. We see how the final performance does not depend too much on this value.

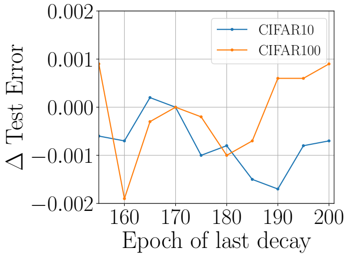
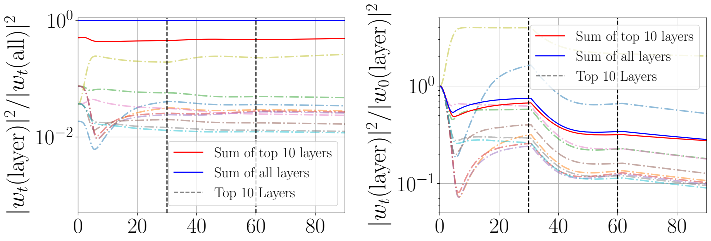


Figure S1: Models trained with ABEL with different values for ‘last_decay_epoch‘. Difference of test error with respect to 85% of training time (170 epochs). We see that the test error does not change too much, most changes are less than 0.2%.

B.2 LAYER-WISE DEPENDENCE DYNAMICS OF WEIGHT NORM.

In this section, we study the dynamics of the layer-wise weight norm. We consider the Resnet-50 Imagenet setup of table 1 with step-wise decay. For better visualization, we focuse on the top 10 layers, we rank layers by their maximum weight norm during training. These top 10 layers are mostly intermediate but there are also early and late layers. In figure S2a, we plot the contribution the quotient between the weight norm of each layer (or the sum of the top 10 layers) and the total weight norm. We see these top layers account for $\sim 50\%$ of the total weight norm and they exhibit the same dynamics: the red line is straight. At the individual layer level, we see that several layers become constant really fast while there are ~ 3 which have a deeper bounce but then have the same dynamics as the total weight norm. We can illustrate this in fig S2b, where we compare the evolution of the different weights (normalized by their value at initialization). Most layers have the behaviour of the total weight norm: a bounce and slow down of growth before decaying.



(a) Resnet-50 on Imagenet (b) Resnet-50 on Imagenet

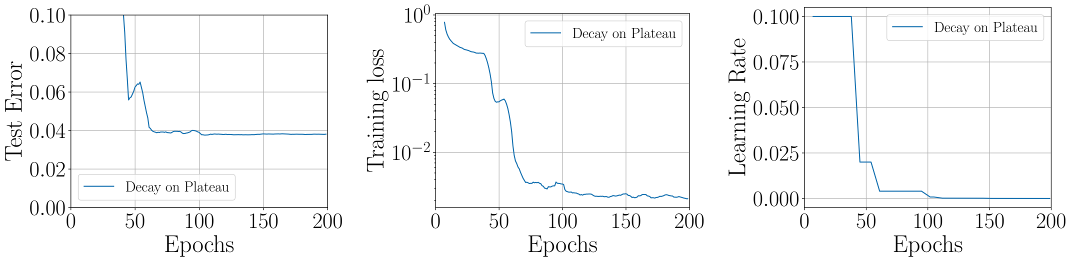
Figure S2: a) Evolution of the quotient between the weight norm of different layers and the total weight norm. b) Evolution of the weight norm of different layers, normalized by its value at initialization.

B.3 DECAY LR ON LOSS PLATEAU

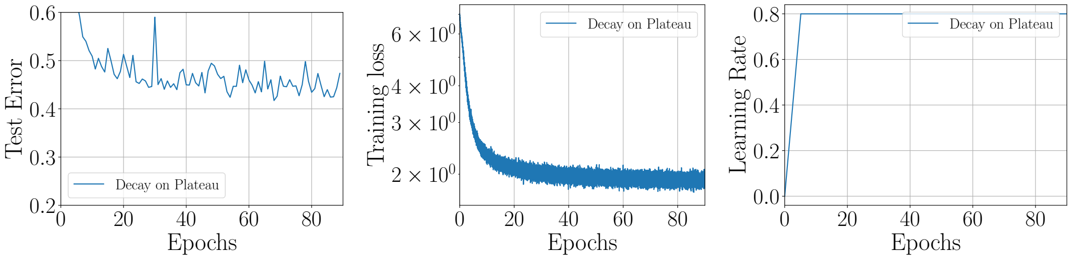
We implemented a simple version of this scheduler in FLAX with the default values for the hyperparameters from the pytorch implementation , except for the decay factor (called factor) which we set to 0.1 for ImageNet and 0.2 for CIFAR. The other relevant hyperparameters which we did not change are patience= 10 (Number of epochs with no improvement after which learning rate will be reduced), threshold= 0.0001 (Relative Threshold for measuring the new optimum, to only focus on significant changes.). See the `https://pytorch.org/docs/stable/optim.html` for more details. The test errors for WRN 28-10 on CIFAR-10 are 3.9 and 47.4 for Resnet-50 on Imagenet. For Imagenet, the loss still decreases slightly during training so it does not seem like the logic behind this schedule could yield good results.

For CIFAR-10, the first decay is caused by a real plateau, but after that the loss increases for more than 10 epochs, so the learning rate decays again. One decay after that, the loss basically stays in a plateau and the learning rate decays without bound. It is surprising that this schedule does so well on CIFAR-10 despite of the learning rate becoming so low so early.

These two examples exhibit the opposite issues: too much vs too few decay. It seems unrealistic that proper hyperparameter tuning can fix both problems, among other things because the ImageNet Resnet loss is slowly decaying.



(a) Wide Resnet on CIFAR-10 (b) Wide Resnet on CIFAR-10 (c) Wide Resnet on CIFAR-10

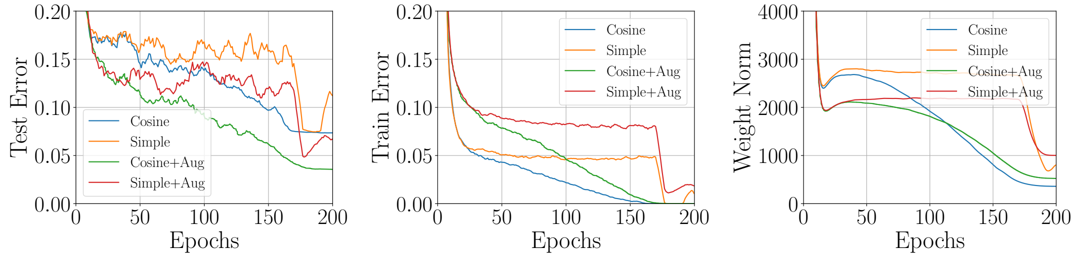


(d) Resnet-50 on Imagenet (e) Resnet-50 on Imagenet (f) Resnet-50 on Imagenet

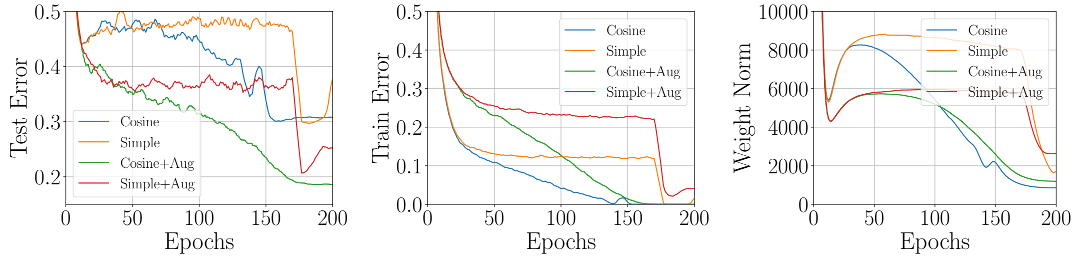
Figure S3: Experiments from the main section with a `ReduceLROnPlateau` schedule.

B.4 NO ADVANTAGE FOR EASY TASKS

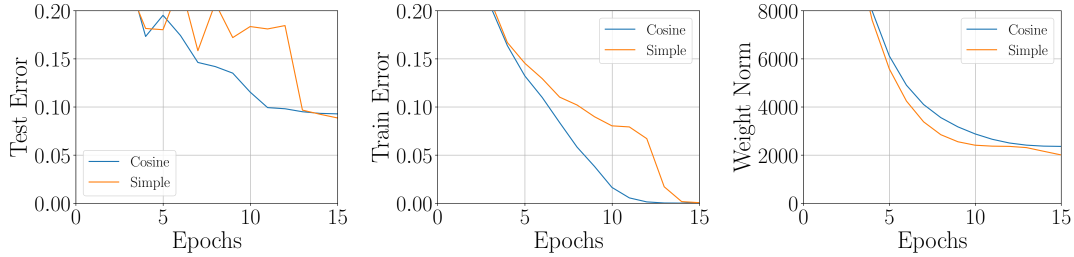
We expand on the discussion of the main text. We train the same Wide Resnet models of table 1 but without data augmentation. We see that in this case, simple decay reaches training error 0 at the end of training and there is no advantage of cosine decay. The test error without data augmentation for CIFAR100 and 29.7 (cosine) and 29.4 (simple) and for CIFAR10 it is 7.3 for both schedules. We attribute this to the fact that when evolved for a small number of epochs, before the weight norm has time to bounce, the simple decay schedule can already reach training error 0 and thus it can not benefit from the bounce.



(a) Wide Resnet on CIFAR-10 (b) Wide Resnet on CIFAR-10 (c) Wide Resnet on CIFAR-10



(d) Wide Resnet on CIFAR-100 (e) Wide Resnet on CIFAR-100 (f) Wide Resnet on CIFAR-100



(g) Wide Resnet on CIFAR-10 (h) Wide Resnet on CIFAR-10 (i) Wide Resnet on CIFAR-10

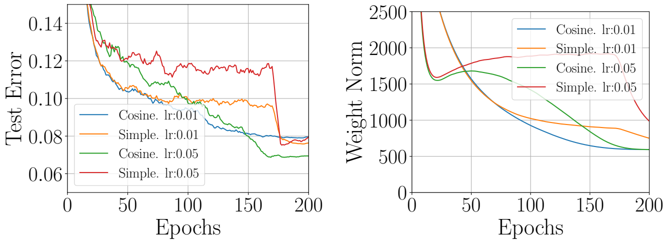
Figure S4: Wide Resnets on CIFAR without data augmentation can reach zero training error with simple decay.

B.5 LEARNING RATE DEPENDENCE OF BOUNCING

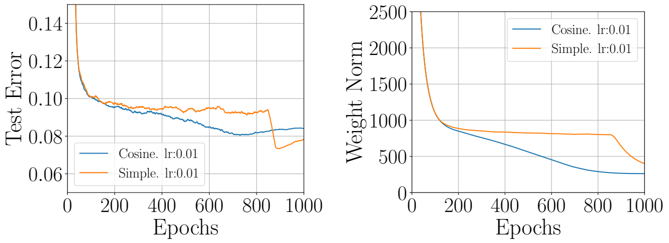
We can study the effect of the learning rate. Large learning rates seem important for weight norm bouncing as we can see in figure S5, the small learning rate of 0.01 does not have a bouncing norm even if we evolve for longer (since smaller learning often to take a longer time to train). We see that only when there is a bouncing norm are cosine schedules benefitial, see table S2.

Setup	Simple Decay	Cosine Decay
lr$= 0.05$	7.4	6.7
lr$= 0.01$	7.4	7.8
lr$= 0.01$ 1000 epochs	7.2	7.6

Table S2: Test error for VGG-16 and CIFAR-10 for different small learning rate setups. Cosine decay is only superior at large learning rates when there is a bouncing weight norm.



(a) VGG-16 on CIFAR-10 (b) VGG-16 on CIFAR-10



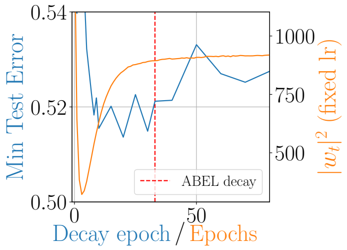
(c) VGG-16 on CIFAR-10 1000epochs (d) VGG-16 on CIFAR-10 1000epochs

Figure S5: In order for the weight norm to bounce in our training run, the learning rate has to be big enough. We see how for learning rate 0.01 weight norm does not bounce. This is true even if we evolve for five times longer.

C EXTRA EXPERIMENTS

Here we describe the extra couple of experiments mentioned in the main text.

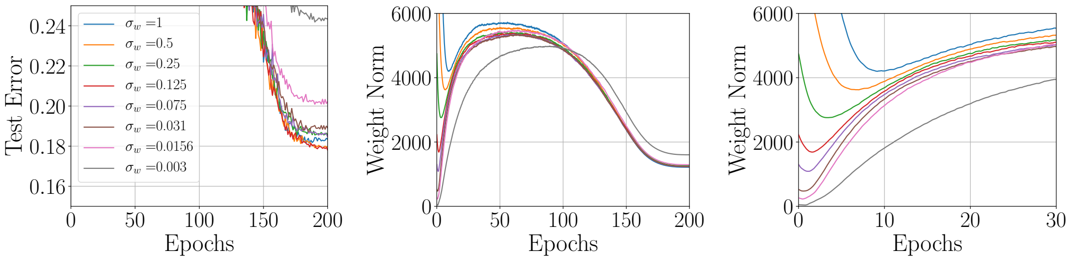
The disadvantage of decaying too early or too late. Waiting for the weight norm to bounce seems key to get good performance. Decaying too late might be harmful because the weight norm does not have enough time to bounce again, but it is not clear if it is bad by itself. In this section, we run a VGG-5 experiment on CIFAR-100 for 80 epochs and we decay the learning rate once (by a factor of 10) at different times and compare the minimum test error, see fig S6a. We see that decaying too early significantly hurts performance and the best time to decay is after the weight norm has started slowing down its growth, before it is fully equilibrated. We can compare this sweep over decay times with ABEL: ABEL would be equivalent to a simple schedule decayed at the time marked by the red line (here there is no benefit from decaying it again at the end of training). Given the limitation of the experiment, we can not conclude that decaying to late is hurtful and from the success of cosine decay we expect it is not bad.



(a)

Figure S6: Minimum test error for VGG5 models decayed at different times (all are evolved for 80 epochs) and weight norm for fixed learning rate. Models with the best performance are decayed after the bounce, soon before the weight norm dramatically slows down its growth.

Dependence on initialization scale. The bouncing behaviour suggests that maybe one can avoid bouncing at all by starting with the minimum weight norm to begin with. In order to check this, we run our CIFAR-100 WRN28-10 experiments with cosine decay for different initializations for all weights (including the batchnorm weights), which we denote σ_w, with $\sigma_w = 1$ the standard experiments that we have run. We see how, even if we start with a weight norm which is below the minimum weight norm for $\sigma_w = 1$, there is still bouncing. As we keep decreasing σ_w, it gets to a point where performance gets significantly degraded (and bouncing dissappears too). These small σ_w are probably pathological, but it is interesting to see again the correlation between degraded performance and no bouncing. Before reaching this small weight initialization the final weight norm (and error rate) is very similar for the different initializations.



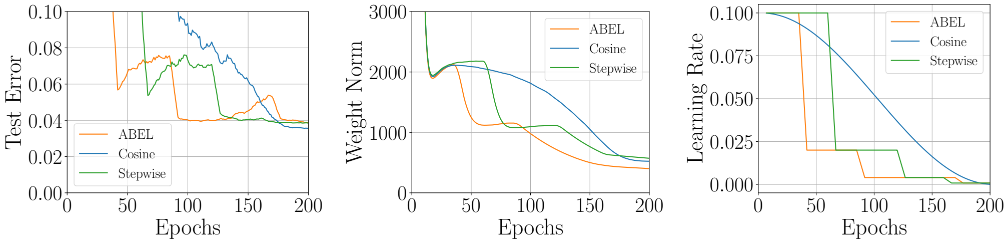
(a) WRN 28-10 on CIFAR-100 (b) WRN 28-10 on CIFAR-100 (c) WRN 28-10 on CIFAR-100

Figure S7: Wide Resnet 28-10 on CIFAR100 expeirents with different initialization scale. a) Test error, we see that the two smallest initialization present substantial degradation of performance. b) Despite of the weight norm being so different at initialization, the final weight norm is the same for the different models. c) Zoomed version of b) we see that all the initializations but the smallest two exhibit clear bouncing.

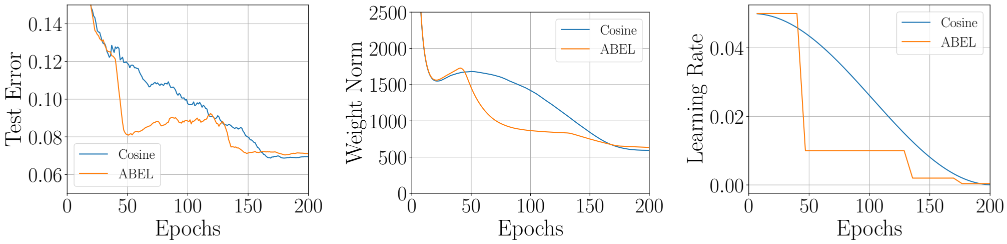
D MORE TRAINING CURVES

D.1 TRAINING CURVES FOR TABLE 1 EXPERIMENTS

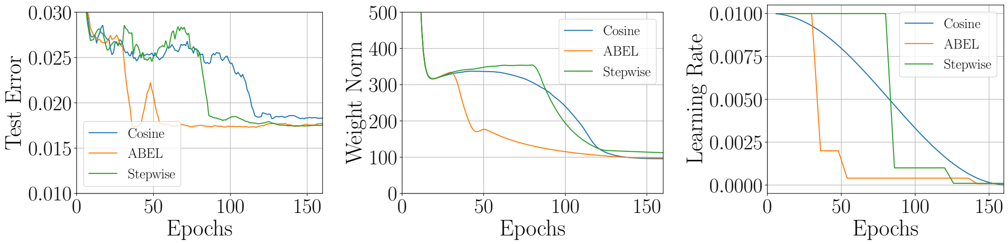
We show the remaining training curves for the Table 1 experiments.



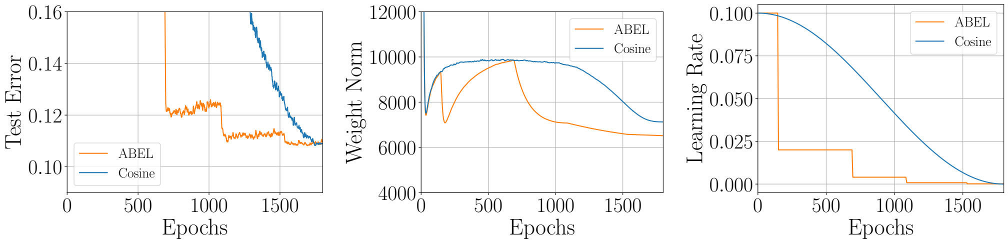
(a) Wide Resnet on CIFAR-10 (b) Wide Resnet on CIFAR-10 (c) Wide Resnet on CIFAR-10



(d) VGG on CIFAR-10 (e) VGG on CIFAR-10 (f) VGG on CIFAR-10



(g) Wide Resnet 16-8 on SVHN (h) Wide Resnet 16-8 on SVHN (i) Wide Resnet 16-8 on SVHN

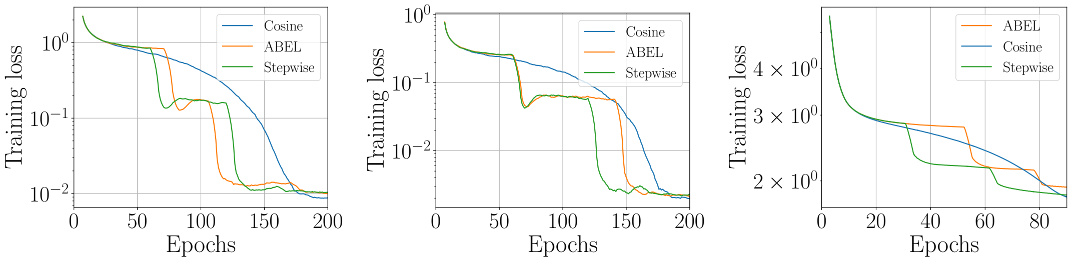


(j) PyramidNet on CIFAR100 (k) PyramidNet on CIFAR100 (l) PyramidNet on CIFAR100

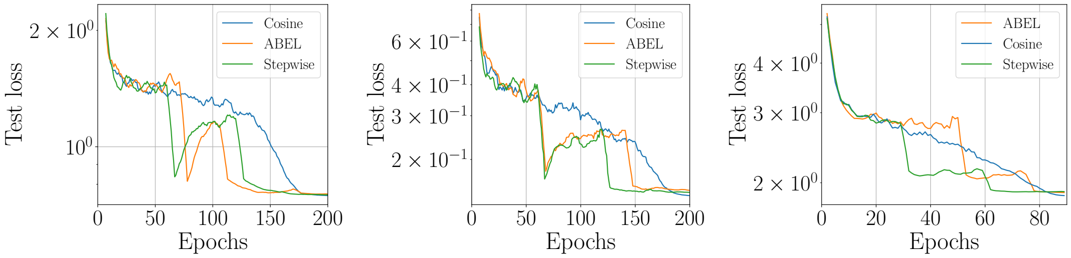
Figure S8: Remaining training curves for experiments of table 1.

D.2 LOSS CURVES FOR FIGURE 2 EXPERIMENTS

Here we show the loss curves for the experiments of figure 2.



(a) WRN 28-10 on CIFAR100 (b) WRN 28-10 on CIFAR10 (c) Resnet-50 on ImageNet

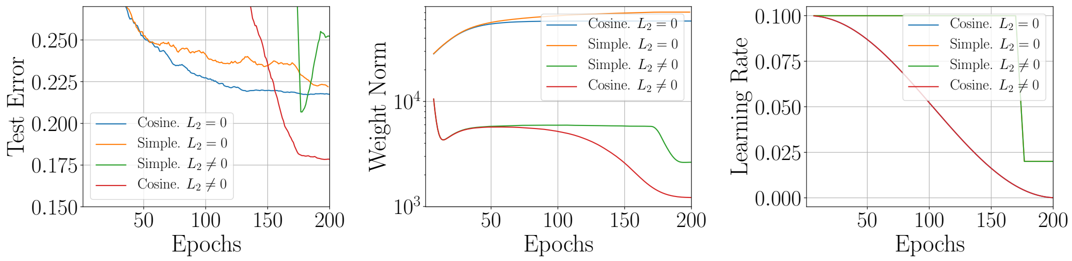


(d) WRN 28-10 on CIFAR100 (e) WRN 28-10 on CIFAR10 (f) Resnet-50 on ImageNet

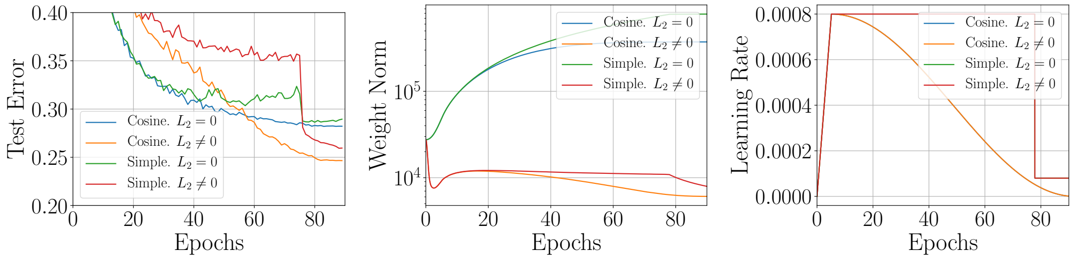
Figure S9: Training and test loss curves for the experiments displayed on figure 2.

D.3 TRAINING CURVES FOR TABLE 2 EXPERIMENTS

Here we show the training curves for some experiments in Table 2.

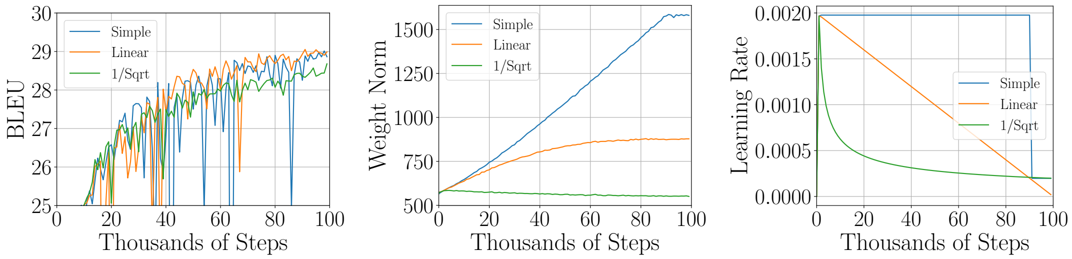


(a) WRN 28-10 on CIFAR100 (b) WRN 28-10 on CIFAR100 (c) WRN 28-10 on CIFAR100



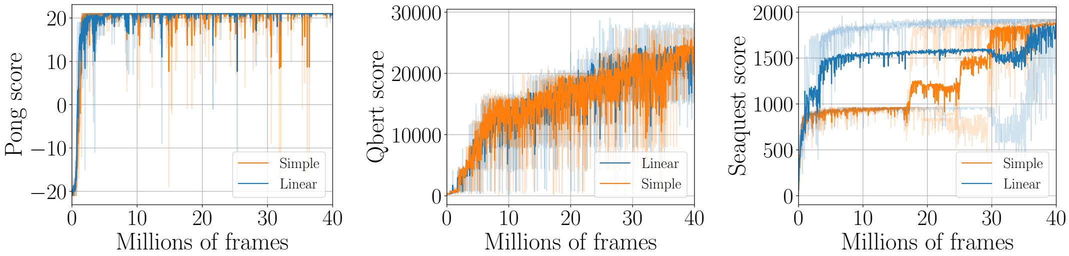
(d) Adam and Imagenet (e) Adam and Imagenet (f) Adam and Imagenet

Figure S10: A couple of vision tasks of table 2. We see how in the absence of L_2, a non-trivial schedule does not affect the performance and the weight norm is usually monotonically increasing. d,e,f) Resnet-50 with Adam on Imagenet. We also see how weight norm bouncing also occurs despite of Adam having an implicit schedule.

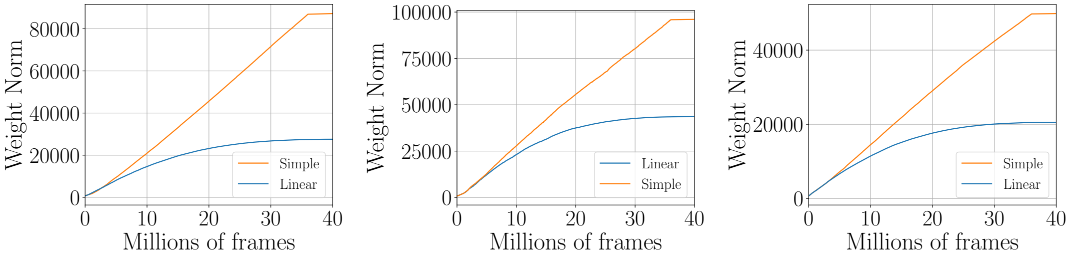


(a) Transformer on En-De WMT (b) Transformer on En-De WMT (c) WRN 28-10 on CIFAR100

Figure S11: Training curves for Transformer trained on the English-German translation task WMT'14. We see how there is no significant difference between differnet learning rate schedules.



(a) PPO Pong (b) PPO Qbert (c) PPO Seaquest

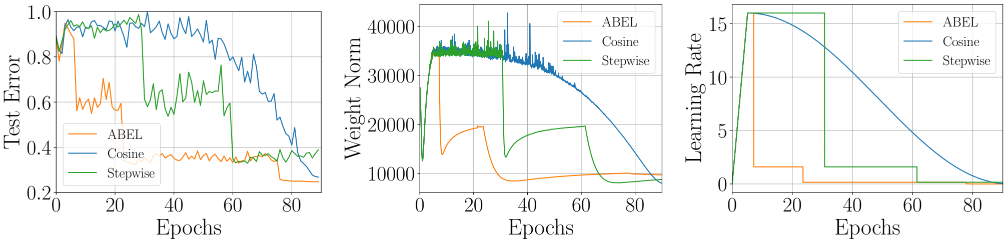


(d) PPO Pong (e) PPO Qbert (f) PPO Seaquest

Figure S12: Game score of different RL games for the two considered schedules. Mean score over three runs and individual scores in lighter color. Simple is just decaying the base learning rate by a factor of 10 at 90% of training and linear is linear decay.

D.4 TRAINING CURVES FOR LARGE LEARNING RATE EXPERIMENT

We show the training curves for the ImageNet experiment with a large learning rate of 16 of section 2, figure 3. We see that Cosine and Stepwise schedules are basically not learning until the learning rate decays significantly, however, ABEL adapts quickly to the large learning rate and decays it fast, yielding better performance.



(a) Resnet-50 on ImageNet (b) Resnet-50 on ImageNet (c) Resnet-50 on ImageNet

Figure S13: Training curves for the Imagenet experiment of section 2, figure 3 with learning rate 16.

E DERIVATION OF EQUATIONS OF SECTION 4

Weight dynamics equations. Equation 1 follows directly from the SGD update:

$$\Delta w_{t+1} = w_{t+1} - w_t = -\eta g_t - \eta\lambda w_t \tag{S1}$$

$$\begin{aligned}\Delta|w_{t+1}|^2 \equiv |w_{t+1}|^2 - |w_t|^2 = \Delta w_{t-1}.\Delta w_{t-1} + 2w_{t-1}\Delta w_{t-1} \quad &= \eta^2|g_t|^2 - (2-\eta\lambda)\eta\lambda|w_t|^2 - 2(1-\eta\lambda)g_t\cdot w_t \\ &\approx \eta^2|g_t|^2 - 2\eta\lambda|w_t|^2 - 2\eta g_t\cdot w_t \end{aligned} \tag{S2}$$

The terms that we are dropping are subleading for our purposes because $\eta\lambda \ll 1$ practically. We expect these terms to be only relevant if the dominant terms cancel each other: $\eta^2|g_t|^2 - 2\eta\lambda|w_t|^2 - 2\eta g_t\cdot w_t \sim 0$

$g\cdot w = 0$ **for scale invariant layers.** A scale invariant layer is such that its network function satisfies $f(\alpha w) = f(w)$, the bare loss only depends on the weights through the network function. If we divide the weight into its norm and its direction: $w_a = |w|\hat{w}_a$ and use $\frac{dL}{d|w|} = 0$, we get that:

$$0 = |w|\frac{dL}{d|w|} = |w|\sum_a \frac{dL}{dw_a}\frac{dw_a}{d|w|} = \sum_a \frac{dL}{dw_a}|w|\hat{w}_a = g\cdot w \tag{S3}$$

For scale invariant networks without L_2, the change in the weights becomes smaller as the weight norm equilibrates. Using the SGD equation we get that:

$$\begin{aligned}\cos\text{angle}(w_{t+1}, w_t) &= \frac{w_{t+1}\cdot w_t}{|w_{t+1}||w_t|} =_{\Delta w_t = -\eta g} \frac{|w_t|}{|w_{t+1}|} \\ \sin\text{angle}(w_{t+1}, w_t) &= \sqrt{1-\cos^2\text{angle}} = \sqrt{\frac{\Delta|w_t|^2}{|w_{t+1}|^2}}\end{aligned} \tag{S4}$$

As $\Delta|w|^2 \to 0$, angle$(w_{t+1}, w_t) \sim \sin\text{angle}(w_{t+1}, w_t) \to 0$.