A LOSS FLATTENING

Consider a binary classification problem. Figure 3 depicts the logistic loss as a function of logit. Notice that loss becomes extremely small once the logit value crosses a certain threshold (say 2.5). A small value of loss yields small gradients, making it difficult for neural network training to be adaptive. We define this as *loss flattening*. Multiclass classification is slightly more complex. The difference between the score of the target class and the largest score of the remaining classes plays the role of the logit and, when it becomes large enough, loss flattening occurs.

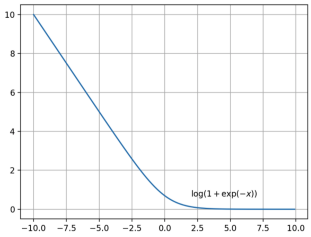


Figure 3: Logistic loss becomes small and flat if the logit crosses a threshold of 2.5 or more. Training becomes less adaptive when the losses of most examples fall in the flat zone.

B EXPERIMENTS

We now describe the detailed experimental setup used in §4.

B.1 CIFAR EXPERIMENTAL SETUP

Dataset Details Both CIFAR-10 and CIFAR-100 contain the same set of 50k images for training and 10k images for the test set. CIFAR-10 uses 10 labels, whereas CIFAR-100 uses 100 labels.

Dataset Pre-Processing For both CIFAR-10 and CIFAR-100, we employ data augmentation techniques for training. We use $RandomCrop(32, padding = 4)$ and $RandomHorizontalFlip$. Additionally, for CIFAR-100, we also use $RandomRotation(15)$. We mean center all the training and test samples.

Network Architecture We used VGG-19 network with a single fully connected layers for all CIFAR experiments. The network is almost homogeneous except for the batch normalization layer. Link to the code used will be released once the paper is accepted.

Hyperparameters E_{total} is fixed at 300 epochs. E_{warmup} is searched over [5, 20, 40, 60]. Weight decay is swept over [1e-5, 1e-4, 1e-3]. E_{free} is searched over [1, 5, 20, 40]. Gradient clipping was used for all optimizers with the gradient norm clipped to a maximum value of 1.0.

Results with standard error are shown in Table 7. Figure 4(a) shows the effect of LAWN switch point E_{free} on CIFAR-10 when Adam-LAWN is used. Small values for E_{free} usually result in good generalization performance.

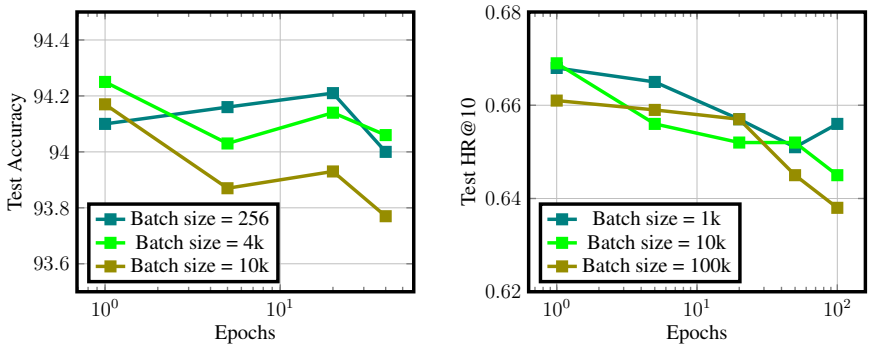
Method	**CIFAR-10**			**CIFAR-100**		
	256	4k	10k	256	4k	10k
SGD	93.99 (0.05)	93.48 (0.09)	92.99 (0.20)	73.49 (0.43)	71.68 (0.26)	71.07 (0.14)
Adam	93.48 (0.11)	92.93 (0.01)	92.63 (0.06)	70.84 (0.23)	68.91 (0.05)	68.61 (0.28)
Adam-L	**93.91** (0.04)	**93.74** (0.04)	**93.84** (0.08)	**72.99** (0.02)	**73.12** (0.11)	**72.97** (0.10)
LAMB	**93.76** (0.07)	**93.27** (0.08)	92.91 (0.03)	**71.29** (0.11)	69.39 (0.06)	67.76 (0.22)
LAMB-L	93.67 (0.04)	93.22 (0.05)	**92.92** (0.03)	71.25 (0.17)	**69.68** (0.16)	**69.16** (0.09)

Table 7: Test accuracy on CIFAR-10 and CIFAR-100 with standard error. Highlighted values indicate the better performing method between x and x-L.

B.2 RECOMMENDATION SYSTEMS EXPERIMENTAL SETUP

Dataset Details Table 8 contains statistics about the three datasets used.

Dataset Pre-Processing Following the experimental setup used in He et al He et al. (2017), we first remove all users in the 3 datasets that have less than 20 interactions. We then binarize the data by treating any interactions as 1s (positives) and the absence of an interaction as 0s (negatives). While training, we sample four negatives for every positive. The negatives are refreshed every epoch.



(a) CIFAR-10 (b) MovieLens-100k

Figure 4: Effect of E_{free} on Adam-LAWN.

Dataset	n(samples)	n(users)	n(items)
MovieLens-100k	∼100k	943	1682
MovieLens-1M	∼1M	6040	3706
Pinterest	∼1.5M	54906	9909

Table 8: Datasets for item recommendation.

Network Architecture We adopt the multi-layer perceptron (MLP) architecture used by He et al He et al. (2017). The MLP is used in conjunction with user and item embeddings. The embeddings are concatenated and pushed through the MLP. The MLP follows a tower pattern, and results in a scalar output prediction y_{ui} for a user u and item i, which can then be compared to the ground truth label y using the binary cross entropy loss for training and evaluation purposes. The number of units in each layer of the MLP are 256-128-64-1, where the first layer takes a 256-dimensional vector as input. Consequently, user and item embeddings are of 128 dimensions each.

Evaluation Details For each user in the datasets, we consider the latest interaction with an item for test performance evaluation. Specifically, we use the *hit ratio@10* metric, where the test item is part of 100 randomly sampled items that the user has not interacted with in the past. The metrics records whether the test item occurs in the top 10 of the predicted ranking of the 100 items.

Hyperparameters

E_{total} was fixed at 300 epochs, and E_{warmup} was fixed at 30 epochs. Weight decay is swept over [1e-4, 1e-3, 1e-2, 1e-1, 0]. E_{free} is searched over [0.1, 1, 5, 20, 50, 100].

Results with standard error can be found in Tables 9, 10 and 11. Figure 4(b) shows the effect of E_{free} on MovieLens-100k when Adam-LAWN is used.

Method	MovieLens-100k			
	1k	10k	100k	400k
SGD	66.33 (0.24)	65.58 (0.17)	Fail	Fail
Adam	66.01 (0.25)	66.03 (0.15)	63.20 (0.24)	63.98 (0.25)
Adam-L	**66.81** (0.17)	**66.91** (0.15)	**66.24** (0.16)	**66.14** (0.22)
LAMB	65.45 (0.25)	65.34 (0.16)	64.23 (0.16)	62.57 (0.17)
LAMB-L	**66.56** (0.22)	**66.54** (0.20)	**66.52** (0.25)	**66.14** (0.23)

Table 9: Test HR@10 on MovieLens-100k with standard error. Highlighted values indicate the better performing method between x and x-L.

Method	MovieLens-1M			
	1k	10k	100k	1M
SGD	70.91 (0.18)	69.31 (0.17)	Fail	Fail
Adam	69.87 (0.07)	70.12 (0.11)	69.28 (0.10)	68.99 (0.21)
Adam-L	**70.80** (0.09)	**70.41** (0.20)	**70.77** (0.04)	**70.66** (0.22)
LAMB	69.91 (0.12)	69.77 (0.09)	69.44 (0.10)	68.95 (0.12)
LAMB-L	**70.86** (0.21)	**70.86** (0.04)	**70.68** (0.08)	**70.34** (0.19)

Table 10: Test HR@10 on MovieLens-1M with standard error. Highlighted values indicate the better performing method between x and x-L.

Method	Pinterest			
	1k	10k	100k	1M
SGD	86.62 (0.05)	85.57 (0.20)	Fail	Fail
Adam	**87.27** (0.07)	85.97 (0.01)	85.81 (0.01)	85.30 (0.20)
Adam-L	86.85 (0.10)	**86.61** (0.10)	**86.04** (0.10)	**86.06** (0.20)
LAMB	86.63 (0.10)	85.91 (0.10)	85.80 (0.10)	85.65 (0.20)
LAMB-L	**86.83** (0.10)	**86.25** (0.10)	**85.99** (0.10)	**86.07** (0.10)

Table 11: Test HR@10 on Pinterest with standard error. Highlighted values indicate the better performing method between x and x-L.

B.3 IMAGENET EXPERIMENTAL SETUP

Dataset Details The ImageNet dataset consists of about 1.2 million training images and 50,000 validation images.

B.3.1 DATA PRE-PROCESSING

This model uses the following data augmentation:

For training:

- Normalization (0 mean, 1 std. dev.)
- Random resized crop to 224×224
- Scale from 8% to 100%
- Aspect ratio from $3/4$ to $4/3$
- Random horizontal flip

For inference:

- Normalization (0 mean, 1 std. dev.)
- Scale to 256×256
- Center crop to 224×224

B.3.2 MODEL

We used the ResNet-50-v1.5 model for ImageNet classification, which differs slightly from the original ResNet50 model He et al. (2015). For more details, the reader can refer to this open source code repository `https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/Classification/ConvNets/resnet50v1.5`.

B.3.3 THE LAMB+ ALGORITHM

The LAMB algorithm uses a ratio consisting of a transformation of the network weight norm in the numerator and the update (including contribution of weight decay) in the denominator. This is known

as the **trust ratio**. Without any or small weight decay, the trust ratio can grow to a large value and cause the training to become unstable. To counter this, we cap the trust ratio to have a max value of 1.0. We found this to make training more stable, and to have better generalization performance. We refer to this algorithm as LAMB+.

B.3.4 HYPERPARAMETER TUNING

All optimizers were tuned with a fixed epoch budget of 90 epochs regardless of batch size. For optimizers that use weight decay, we did not apply weight decay to batch normalization parameters.

Momentum for SGD was fixed at 0.875. Gradient clipping was used for all optimizers except SGD, with the gradient norm clipped to a maximum value of 1.0.

Small batch size

For SGD at a batch size of 256, we fixed weight decay to 10^{-4}. Following Goyal et al. (2018), we allow 0.5 epochs for E_{warmup}.

For Adam, we did not use any L2 regularization or weight decay. We used 0.16 epochs for E_{warmup}. For Adam-LAWN, we tuned E_{free} and E_{warmup} over a small grid of $\{1, 5, 10\}$.

For LAMB and LAMB+, we bootstrapped using suggestions in literature and fixed weight decay to 0.01 and E_{warmup} to 0.16 epochs. For LAMB-LAWN, we fixed E_{free} to match learning rate warmup of LAMB to 0.16 epochs. We tuned E_{warmup} over a grid of $\{1, 5, 10\}$.

Large batch size

For SGD for large batch size, we retain the weight decay value to be 10^{-4}. Since learning rate warmup is important for large batch sizes (Goyal et al., 2018), we tune E_{warmup} over a grid of $\{0.16, 1, 5, 10\}$ epochs.

For Adam at a batch size of 16k, we tuned weight decay over $\{0, 1e-3, 1e-2\}$ and duration of learning rate warmup over $\{5, 10, 20\}$. Similar to Adam-LAWN at small batch sizes, we tuned E_{free} and E_{warmup} over a small grid of $\{1, 5, 10\}$.

For LAMB and LAMB+, we again fixed weight decay to 0.01. We grid searched E_{warmup} over $\{5, 10, 20\}$.

For LAMB-LAWN, we retained the best settings of learning rate and weight decay from LAMB and tuned E_{free} and E_{warmup} over a small grid of $\{5, 10, 20\}$.

C EXPLAINING ESCAPE

Recent theoretical and empirical analysis (Jastrzebski et al., 2020; Wu et al.; Zhou et al., 2020) have pointed out the following. (a) For deep nets to generalize well, escaping regions of attraction of "sharp" minima with sub-optimal generalization performance is important. (b) The escape mechanism is influenced by the training paradigm (the type of formulation, regularization, normalizations employed, etc.) and the training hyperparameters (η, the learning rate, and B, the batch size are the two crucial ones). The proposed theories have differences in explaining the nature of escape, but they broadly align in terms of how the various mentioned parameters influence the escape. Here we will use the theory of Wu et al (Wu et al.).

Two matrices - Σ, the covariance of the noise associated with minibatch gradient, and H, the Hessian of the training loss - combine with η and B to dictate what happens in the basin of one minimum of the loss. To give a rough guidance, Wu et al (Wu et al.) conduct a rough theoretical analysis around the minimum and require that the following condition is met for escape from that minimum:

$$\lambda_{\max}\left\{(I-\eta H)^2+\frac{\eta^2(m-B)}{B(m-1)}\Sigma\right\}>1 \tag{4}$$

where $\lambda_{\max}(A)$ denotes the largest eigenvalue of A, m is the total number of training examples and, Σ and H are evaluated at the minimum. The following are useful to note.

- As we move to large batch sizes and hence get B closer to m, the effect of the second term diminishes significantly, causing the escaping to become difficult. This is a key reason why

normal training methods suffer from a loss of generalization performance with large batch sizes.

- When the target class logit values become very large and hence training loss starts moving very close to zero, H and Σ themselves become very small, thus severely hampering the escape. A careful scheduling of η to very large values to cause the escape and then come back to smaller η values to continue the training to a better minimum are needed. But designing such a schedule is hard.

LAWN, by keeping weights (and hence logits) well bounded and hence keeping most examples away from regions where loss flattens to zero, Σ and H are kept large and hence escape from sub-optimal minima is made easier. In fact, even with large batch sizes where B comes close to m, the largeness of H can make escape still possible.

D GRADIENT FLOW WITH WEIGHT NORM CONSTRAINTS

Let $g = \nabla_w L$, and $\dot{w}$ denote $\frac{dw}{dt}$. Unconstrained gradient flow corresponds to the ordinary differential equation (ODE), $\dot{w} = -g$. We know that $\frac{dL(w(t))}{dt} \leq 0$, i.e., the loss descends along any trajectory, and, a w is an equilibrium point of the gradient flow ODE iff $g = 0$ at w.

Let us now consider the constrained phase of LAWN, which solves

$$\min_w \; L(w) \text{ s.t } \|w^\ell\|^2 = (c^\ell)^2 \; \forall \ell \tag{5}$$

At a given weight vector, w let g be the gradient vector. The gradient of the ℓ-th constraint function, $\|w^\ell\|^2$ is simply $2w^\ell$. Let us define the projected gradient g_p as the projection of the gradient g to the linear space defined by $\{d : (2w^\ell)^T d^\ell = 0 \; \forall \ell\}$. Given the decoupling over ℓ, this corresponds to individually projecting, for each ℓ, g^ℓ to $\{d^\ell : (2w^\ell)^T d^\ell = 0\}$. The projection is easy to compute as:

$$g_p^\ell = g^\ell - \frac{(w^\ell)^T g^\ell}{\|w^\ell\|^2} w^\ell \;\; \forall \ell \tag{6}$$

The g_p^ℓ, all combined together, yield the full projected gradient, g_p. Let θ^ℓ be the angle between g^ℓ and w^ℓ, i.e.,

$$(w^\ell)^T g^\ell = \|w^\ell\| \; \|g^\ell\| \; \cos(\theta^\ell) \tag{7}$$

The gradient flow of the constrained phase of LAWN is given by the ODE,

$$\dot{w} = -g_p \tag{8}$$

Along any trajectory, $w(t)$ of (8), we have, using (6) and (7),

$$\frac{dL(w(t))}{dt} = \sum_\ell (g^\ell)^T \dot{w}_\ell = -\sum_\ell (g^\ell)^T (g_p^\ell) = -\sum_\ell (1 - \cos^2\theta^\ell) \|g^\ell\|^2 \leq 0 \tag{9}$$

Also,

$$\frac{d\|w^\ell(t)\|^2}{dt} = 2(w^\ell)^T \dot{w}_\ell = -2(w^\ell)^T (g_p^\ell) = 0 \;\; \forall \ell \tag{10}$$

Thus, the trajectory stays on the constraint set and descends along any trajectory of (8). Further, an equilibrium point of (8) corresponds to $\cos\theta^\ell = 0 \; \forall \ell$, which is same as $(w^\ell)^T g^\ell = 0 \; \forall \ell$, which is equivalent to saying that w is a KKT point of (5). In fact, using (6), the term, $\frac{(w^\ell)^T g^\ell}{2\|w^\ell\|^2}$ can be seen as the lagrangian multiplier corresponding to the constraint, $\|w^\ell\|^2 = (c^\ell)^2$.

Therefore, for training the constrained phase of LAWN, any gradient based optimizer simply has to use the projected gradient, g_p instead of the gradient, g. The only additional point to note is that, when an optimizer takes a discrete step that leads to a w violating the constraints, we need to rescale that w to satisfy the constraints of (5).