Accelerating Equilibrium Models by Stabilizing Their Jacobians
Supplementary Material

A. Dataset Information, Experimental Settings and Hyperparameters

We provide below a detailed description of all tasks and settings for experiments reported in Section 5, as well as some training specifics of the deep equilibrium network (DEQs) we use.

A.1. 1D Synthetic Dataset

To visualize the effect of the proposed Jacobian regularization on DEQ models (see Section 5), we generated a synthetic dataset with 5096 pairs (x, y) from the target function:

$$y = h(x) = \frac{3}{2}x^3 + x^2 + 5x + 2\sin(x) - 3 + \delta$$

where $\delta \in \mathcal{N}(0, 0.05)$ are i.i.d. noise variables added to each sample in the dataset. Specifically, we split the generated data into 4096 training samples and 1000 validation samples.

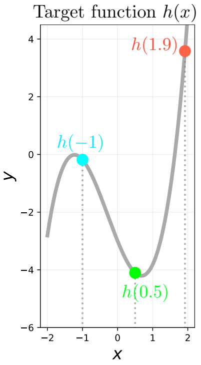


Figure 9. Target function $y = h(x)$.

Figure 9 shows the target function. In the context of deep equilibrium networks, we aim to learn a function $z^\star(x)$ such that $z^\star = f_\theta(z^\star; x)$ and $z^\star(x) \approx h(x)$. At a high level, we should expect the *intersection* between the $z_{\text{out}} = f_\theta(z; x)$ surface and the $z_{\text{out}} = z$ plane to be exactly like the gray curve in Figure 9.

The learned DEQ equilibria $z^\star(x)$ are empirically demonstrated in Figure 5 in red dashed lines for different choices of γ. As expected, all γ fit the target function perfectly, but the introduction of the Jacobian regularization makes the surface more flat around the fixed point.

A.2. WikiText-103 Word-level Language Modeling

Word-level language modeling tasks aim to predict the next word of a textual sequence by integrating the semantics and information of current and past tokens. Formally, given an input sequence $\mathbf{x}_{1:T} \in \mathbb{R}^{T \times p}$ (where $x_i \in \mathbb{R}^p$ and T is the sequence length), an autoregressive sequence model G produces output $G(\mathbf{x}_{1:T}) = \mathbf{y}_{1:T} \in \mathbb{R}^{T \times q}$ that satisfies the causality constraint: y_t depends only on $x_{1:t}$ and not on the future information $x_{t+1:T}$. When each x_i represents a word (i.e., a word embedding), the task is essentially a *word-level language modeling task*. This is a widely-studied problem in the NLP community (e.g., (Merity et al., 2017; 2018; Dai et al., 2019)), and has seen practical advancement in the last few years with development of GPT-3 (Brown et al., 2020; Radford et al., 2019).

A commonly used large-scale corpus for this task is the WikiText-103 (Merity et al., 2017) dataset, which contains 103M/217K/246K words at train/validation/test time, respectively. The entire corpus has a vocabulary size of 267K (i.e., the number of rows in the word embedding). Unlike other well-processed, much smaller datasets like Penn Treebank (Marcus et al., 1993), WikiText-103 is much more challenging as it contains many rare words and retains punctuations, numbers, upper- and lower-cases from the source Wikipedia articles; it has been the standard benchmark for many high-capacity language models in recent literature (Merity et al., 2018; Bradbury et al., 2017; Dai et al., 2019). We provide a shell script in our submitted code to download this dataset.[1]

A.3. CIFAR-10 & ImageNet Image Classification

The CIFAR-10 (Krizhevsky & Hinton, 2009) dataset contains 60,000 color images of resolution 32×32 that fall into 10 object classes (with uniformly 6,000 images per class). We use the standard setting where 50K of these images are used for training and the rest 10K for validation purpose.

The ImageNet (Krizhevsky et al., 2012) dataset, on the other hand, contains over 1.28M training images and 150K test images, distributed over 1,000 classes. All images are rescaled to 224×244 resolution before they are fed into the models (as the original images are of variable resolutions and scales). This is a frequently used dataset for evaluating large-scale vision networks, and has been used for also pretraining many image feature extractor for use on downstream tasks.

For both CIFAR-10 and ImageNet, each training image goes through a canonical data augmentation process before they are fed into the model, where we perform random cropping and random horizontal flipping.

[1] Officially, this dataset can be downloaded at this link.

Table 4. Hyperparameters, optimizer choices, and model details (at training time) for all tasks reported in Section 5. The arrows in the Jacobian regularization strength (e.g., $A \to B$) mean that we dynamically increase from A to B over the course of DEQ training.

	Synthetic Dataset	**WikiText-103 language modeling**	**CIFAR-10 classification**	**ImageNet classification**
Architecture of f_θ	2-Layer ReLU block (see Section 5)	Transformer layer (Pre- and Post-LN)	Multiscale DEQ layer (residual block + fusion)	Multiscale DEQ layer (residual block + fusion)
# of Epochs	50	23	200	120
Batch Size	64	60	96	112
Optimizer	Adam	Adam	Adam	SGD
Start Learning rate	0.001	0.00025	0.001	0.05
Learning rate warmup	No	Yes, 1 epoch	No	No
Learning rate schedule	Cosine	Cosine	Cosine	Cosine
Weight Decay	0	0	0	$5 \cdot 10^{-5}$
Hidden dimensionality	50	700 (embedding size)	[28,56,112,224] (4 scales)	[32,64,128,256]
Input Sequence Length	N/A	150	N/A	N/A
Input Image Size	N/A	N/A	32×32	224×224
Normalization	None	LayerNorm (Ba et al., 2016)	GroupNorm (Wu & He, 2018)	GroupNorm
Recurrent Dropout	N/A	0.06	0.25	0.02
Weight Normalization	No	Yes	Yes	Yes
# of Input Injection Downsamplings	N/A	N/A	N/A	2
Forward NFEs Threshold	6	12	7	14
Backward NFEs Threshold	6	12	8	14
Forward Threshold ε	10^{-3}	10^{-3}	10^{-3}	10^{-3}
Backward Threshold ε	10^{-4}	10^{-4}	10^{-4}	10^{-4}
Jacobian Reg. Strength γ	{0,1,2,4}	$1.6 \to 2.5$	0.5	$2.0 \to 3.0$
Jacobian Reg. Frequency p	0.4	0.35	0.05	0.1
M for Hutchinson Estimator	1	1 or 2	1	1 or 2

A.4. Training Setting and Hardware

Our experimental protocols are intentionally set to be maximally consistent with prior work (Bai et al., 2019; 2020). This includes hyperparameters (see the subsection below), other regularization methods (e.g., recurrent dropout (Gal & Ghahramani, 2016) & group normalization (Wu & He, 2018)), and initialization schemes (where all parameters are initialized at the start of training by sampling from $\mathcal{N}(0, 0.01)$). For the multiscale DEQs that were used in the image classification task, we used 4 resolutions, where each subsequent resolution is of exactly half the height and width of the previous resolution. Although Bai et al. (2020) highlighted the need to train a ReLU-based network with softplus for stability purposes, we found it not necessary in our experiments with regularized DEQs, most likely because of the role Jacobian regularization plays in stabilizing the network convergence.

One thing to note is that empirically, rather than applying the proposed Jacobian regularization on all training iterations, we only randomly and partially apply this auxiliary loss. For example, when we set the auxiliary loss frequency p to 0.5, only half of the training iterations (randomly selected) are trained with the Jacobian regularization term (see Table 4). This is motivated by the empirical observation that Jacobian-related regularizations usually hurt performance, e.g., as in its application in robust learning (Hoffman et al., 2019). Therefore, such partial/random supervision with the Jacobian regularization brings two benefits: 1) the rest $(1-p)$-portion of the training iterations can pick up a further speedup as we don't need to compute the Hutchinson estimator and backpropagate through it; and 2) it helps reduce the likelihood of the model overfitting on this auxiliary loss term (since, as we noted in Section 5.5, the model could be sensitive to γ, and M is small), which we generally observe to benefit the performance, though only slightly. Therefore, during training, the model would still proceed in the actual stochastic gradient direction, and only use the regularized direction occasionally.

Formally, the training objective we highlighted in Section 4.2 should be:

$$\mathcal{L}_{\text{total}}(\mathbf{z}^\star) = \mathcal{L}_{\text{orig}}(\mathbf{z}^\star) + \tau \cdot \gamma \frac{\sum_{m=1}^{M} \|\epsilon^\top J_{f_\theta}(\mathbf{z}^\star)\|_2^2}{Md}, \quad \epsilon_m \in \mathcal{N}(0, I_d)$$

where $\tau = \text{Bernoulli}(p)$ is a random variable and M is the number of samples used for Hutchinson estimator.

All experiments in this paper, including the speed and memory benchmarks we provide, were conducted on RTX 2080 Ti GPUs. WikiText-103 language modeling and ImageNet classification models (MDEQ-small) were trained with 4 GPUs in a data-parallel setting.

A.5. Hyperparameters

We report the hyperparameters used at training time in Table 4. Except for those used in the synthetic data and for Jacobian regularization, most of the other hyperparameters were essentially taken from the original DEQ-Transformer (Bai et al., 2019) and MDEQ (Bai et al., 2020) without major modifications. For both Anderson and Broyden fixed-point solvers, we use the relative residual $\frac{\|f_\theta(\mathbf{z};\mathbf{x})-\mathbf{z}\|}{\|f_\theta(\mathbf{z};\mathbf{x})\|}$ as a measure of convergence quality in forward

Table 5. A more complete version of Table 1 with more memory and efficiency comparison. Memory benchmarked on batch size 15 and excludes the embedding layer. † indicates unregularized model hard-stopped at inference time (while still trained with more NFEs). Overall, we find that Jacobian regularization allows us to train and predict with much fewer NFEs, at a relatively small cost in performance.

	Model Size	Perplexity	t_{train} (relative)	Train NFE	Valid. NFE	Training Memory
AWD-Quasi RNN (Bradbury et al., 2017)	159M	33.0	-	-	-	7.1GB
Relational Memory Core (Santoro et al., 2018)	195M	31.6	-	-	-	-
Megatron-LM (Shoeybi et al., 2019) [SOTA]	8300M	10.8	-	-	-	-
Transformer-XL (18-layer) (Dai et al., 2019)	110M	24.1	1×	-	-	9.0GB
DEQ-Transformer (Pre-LN) (Bai et al., 2019)	98M	[diverged]	N/A	30	N/A	N/A
DEQ-Transformer (Post-LN) (Bai et al., 2019)	98M	24.0	3.1×	30	30	3.9GB
DEQ-Transformer (Post-LN) *early stopped*	98M	29.2	3.1×	30	12	3.9GB
DEQ-Transformer (Post-LN) (Bai et al., 2019)	98M	26.0	2.2×	20	20	3.6GB
DEQ-Transformer (Post-LN) (Bai et al., 2019)	98M	[diverged]	N/A	15	N/A	3.6GB
DEQ-Transformer (Pre-LN) + JR (ours)	98M	24.5	1.5×	14	14	4.8GB
DEQ-Transformer (Post-LN) + JR (ours)	98M	24.9	1.4×	13	12	4.8GB
DEQ-Transformer (Post-LN) + JR (ours) (trained on seqlen=300)	98M	23.8	2.2×	13	13	6.5GB

and backward passes. At inference time, we generally reduce the number of NFEs (e.g., cf. Table 4 and Table 1), while the other hyperparameters (e.g., GroupNorm group sizes) are kept the same.

B. Additional Experimental Results

B.1. Memory Consumption

As we noted in Sections 4 and 5, using Jacobian regularization and thus the vector-Jacobian-product-based Hutchinson estimator introduces some extra memory cost at training time due to the need to differentiate w.r.t. the $\|J_{f_\theta}\|_F$ term. Overall, with the same batch size and sequence length, we observe a roughly 25% increase in training memory required (from about 3.9GB to 4.8GB, excluding embeddings). This is less than the memory consumption of a layer, because the reduction in NFEs needed on the other side saves the memory used by the solver (see Section 3.4). However, this memory footprint is still much better than the conventional explicit Transformer-XL model, which consumes about 2× as much GPU memory. With the Jacobian regularization, as we can see, the DEQ models are much more efficient in time complexity than before, while still staying competitive on the space complexity and the performance fronts.

B.2. DEQ's Backward Convergence with Jacobian Regularization (CIFAR-10)

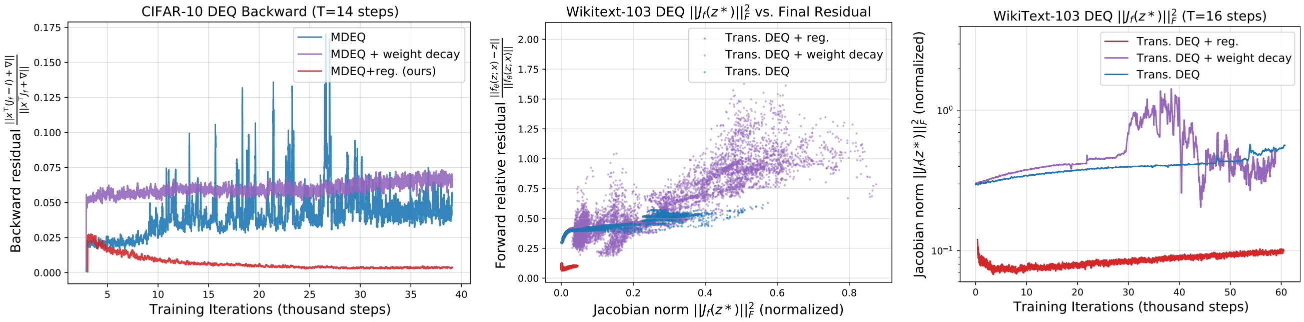
As we discussed in Section 4, the backward dynamics of a DEQ model is a *linear* fixed point system that depends directly on the Jacobian at equilibrium (i.e., $J_{f_\theta}(\mathbf{z}^\star)$). Therefore, the backward pass stability is directly influenced by the conditioning of the Jacobian that we regularize. The stabilizing effect of the proposed Jacobian regularization on the backward pass convergence was already shown for WikiText-103 language modeling in Figure 3b, where we empirically observe that the Jacobian-regularized DEQ-Transformer's backward pass stays at a consistent level, which indicates a relatively more accurate gradient produced by the implicit function theorem.

We further corroborate this finding via empirical evidence on the CIFAR-10 dataset with a multiscale DEQ (MDEQ) instance, shown in Figure 10a. Compared to the original MDEQ (blue line), the Jacobian-regularized version of the backward pass experiences much fewer fluctuations (and thus less stochastic gradients). We also compared to an alternative solution that uses the simple weight decay. Although it also alleviates the fluctuation problem, our empirical observations suggest that weight decay alone almost always adds more difficulty to the fixed point solving. This agrees with what we have observed in the forward pass in Section 5.5. Such comparison can be seen in Figure 10a in the purple line, which converged even more poorly than the original baseline after 14 backward solver iterations (with relative residual > 0.05 and increasing slowly over training). In contrast, the regularized backward pass is more smooth and stable (red line) throughout training (we used $\gamma = 0.5$).

B.3. Failure of Weight Decay to Fix the Problem

This overall inability of weight decay alone to fix the DEQ stability issue (e.g., see Figures 8 and 10a), we believe, exactly suggests that there is a deeper *implicitness* property of the model that should be regularized than just the value of individual weights. As DEQ networks typically rely on a single f_θ block, their complex non-linear structure makes their stability depend as much on the linear parts of f_θ (which weight decay does regularize) as the non-linear parts (which weight decay does not directly regularize; e.g., self-attention in f_θ if we use a Transformer layer). On the other hand, Jacobian regularization takes into account both parts as it tries to constrain the overall spectral radius of the matrix.

We also provide some additional analysis on how $\|J_{f_\theta}\|_F$ evolves during training in Figure 10b and 10c. Specifically, even with weight decay, the convergence of DEQ-



(a) Jacobian regularization improves both the fluctuation and quality of the backward convergence.

(b) Forward relative residual on WikiText-103 as a function of the Jacobian norm.

(c) Jacobian norm grows throughout training, even when we regularize for it.

Figure 10. Additional analysis on DEQ models' backward convergence (on CIFAR-10), Jacobian norm, etc.

Transformer models can be quite bad (see purple dots in Figure 10b), with a clear correlation between the larger relative residual and larger $\|J_{f_\theta}\|_F^2$. Indeed, with a non-linear structure as complex as the multi-head self-attention, simply constraining the weights to be small is not sufficient to ensure well-conditioned Jacobians. Moreover, while the Jacobian regularization helps significantly stabilize the forward and backward convergence (see Figure 1a, 10a and 3), we note that a regularized DEQ model still in fact gradually tends to "critical stability". This can be seen in Figure 10c, where the Jacobian norm grows slowly over training iterations (red line) for a fixed γ, though at a rate much slower than the unregularized and weight-decayed baselines. Therefore, as we indicated in Section 5.5, the proposed Jacobian regularization does not fundamentally *fix* the growing instability problem, but only *alleviates* it. This also calls for adaptive γ scheduling during training (which we adopt in a simple form in our implementation and leave more advanced schemes for future work).