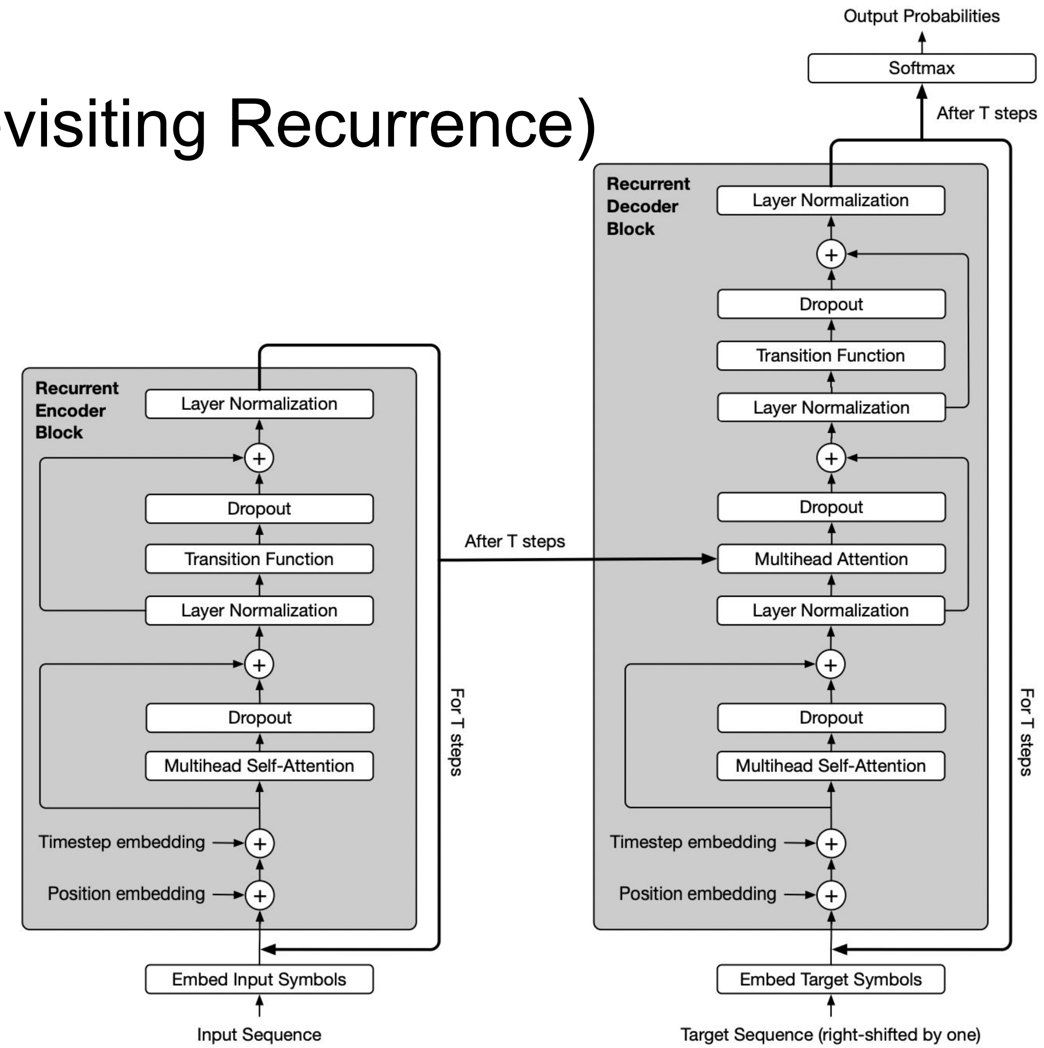
Deep Equilibrium Models

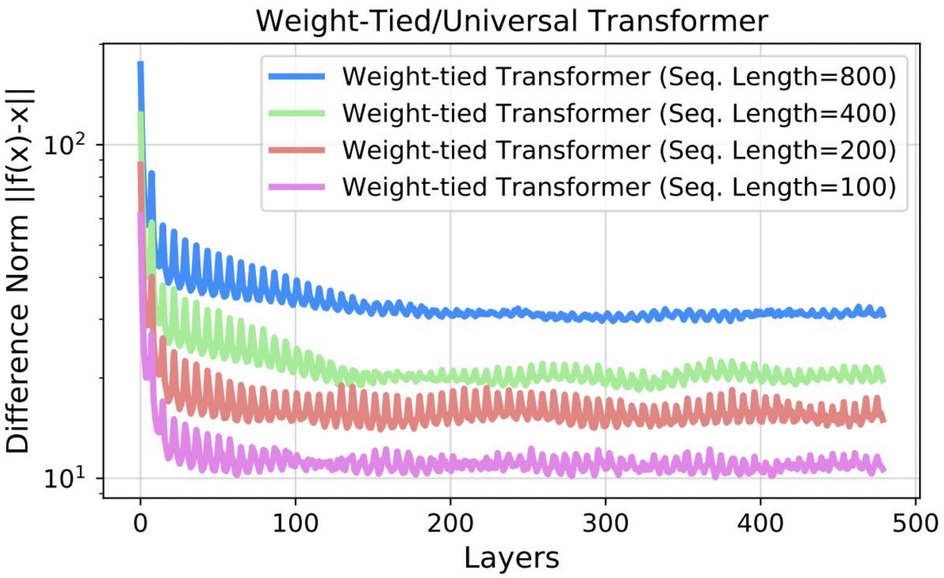
Deep Networks via Root Finding
Neurips 2019

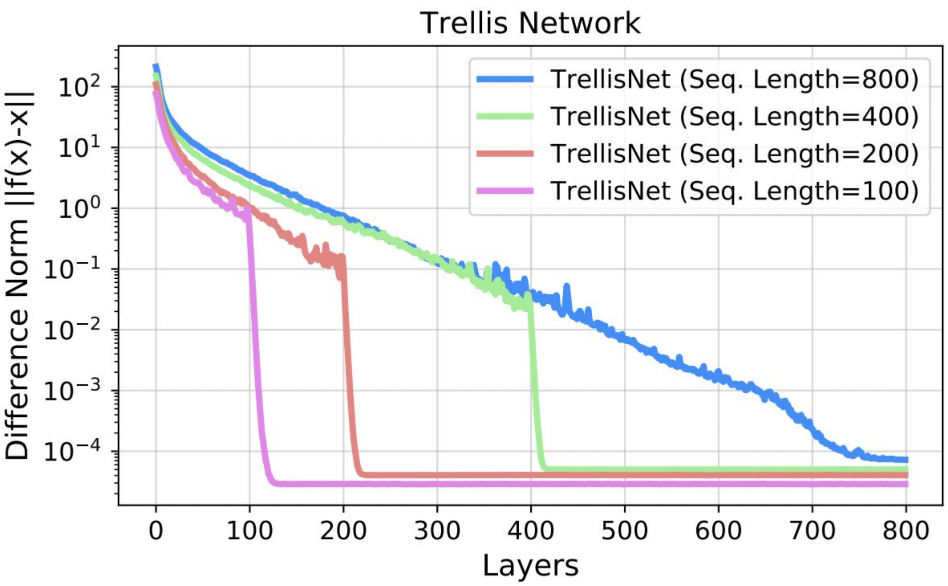
Universal Transformer (Revisiting Recurrence)

- Encoder-decoder architecture
- Recurrent over revised representation of each position (depth)
- Two Steps:
  - Self attention between all positions in sequence
  - Transition function (shared across position and time) to update each position
- Advantages of weight sharing:
  - Regularizes/stabilizes training
  - Reduces model size
  - Any deep network can be written as a weight-tied deep network (appendix)



The Sequence of Intermediate Representations (Empirically) Converges





Can we solve directly for the fixed point?

Why do we have a predetermined, fixed number of layers?

- Because of device memory constraints
- We essentially choose a number of layers that fits in device memory.

Suppose we had infinite layers.

- What is the limit of this model's intermediate representations? The eq point.

$$\lim_{i \to \infty} \mathbf{z}_{1:T}^{[i]} = \lim_{i \to \infty} f_\theta\left(\mathbf{z}_{1:T}^{[i]}; \mathbf{x}_{1:T}\right) \equiv f_\theta\left(\mathbf{z}_{1:T}^{\star}; \mathbf{x}_{1:T}\right) = \mathbf{z}_{1:T}^{\star}$$

The Deep Equilibrium Approach

Instead of iteratively stacking layers, solve for and differentiate through the equilibrium.

Conventional weight-tied networks do this via fixed-point iteration,

$$\mathbf{z}_{1:T}^{[i+1]} = f_\theta\left(\mathbf{z}_{1:T}^{[i]}; \mathbf{x}_{1:T}\right) \quad \text{for } i = 0, 1, 2, \ldots$$

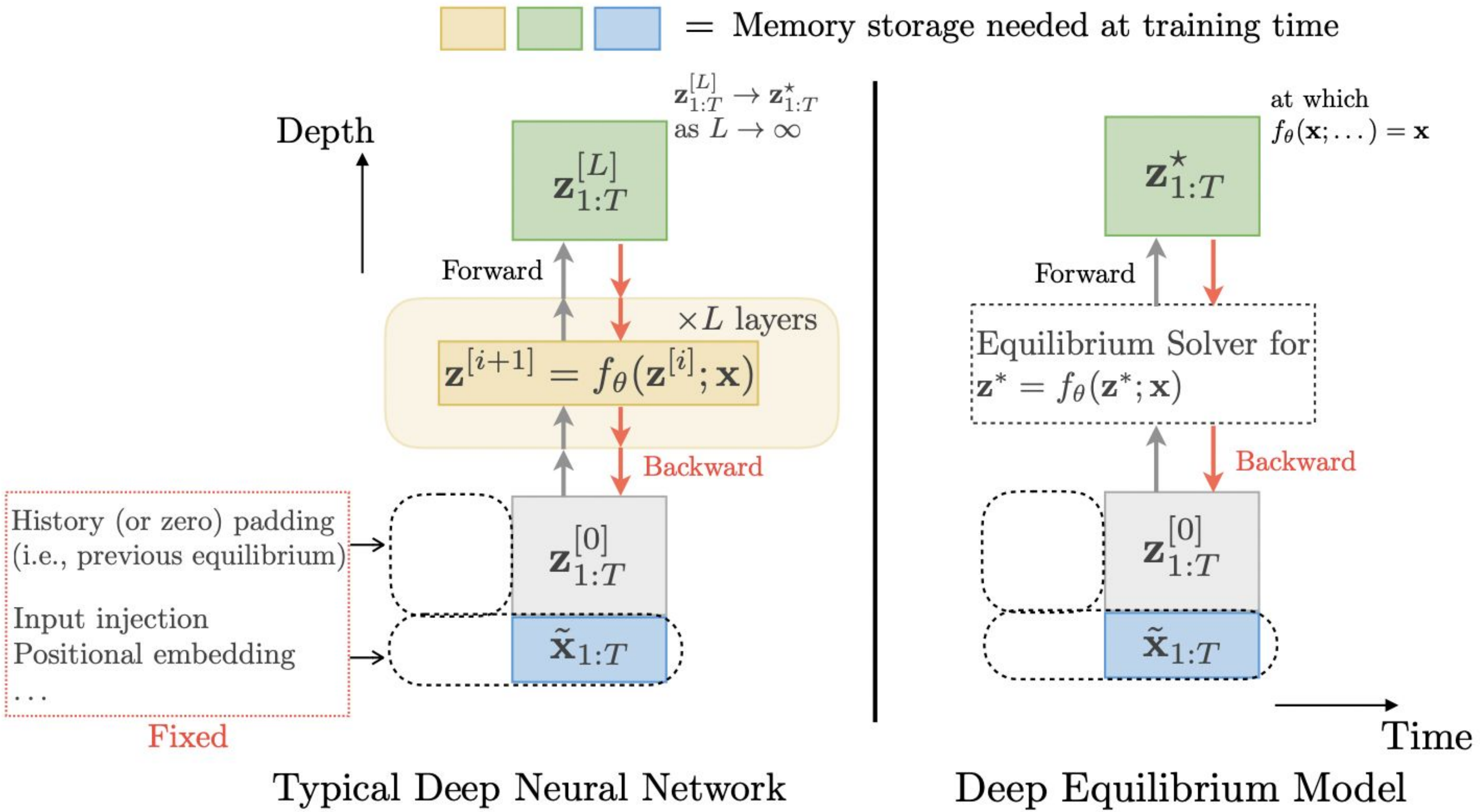
Is there a smarter way? Root finding algorithms,

$$g_\theta(\mathbf{z}_{1:T}^\star; \mathbf{x}_{1:T}) = f_\theta\left(\mathbf{z}_{1:T}^\star; \mathbf{x}_{1:T}\right) - \mathbf{z}_{1:T}^\star \to 0$$

$$\mathbf{z}_{1:T}^{[i+1]} = \mathbf{z}_{1:T}^{[i]} - \alpha B g_\theta(\mathbf{z}_{1:T}^{[i]}; \mathbf{x}_{1:T}) \quad \text{for } i = 0, 1, 2, \ldots$$

Newton’s method or quasi-newton (Broyden), B is inverse Jacobian
Or alternatively any root-finding algorithm

The Deep Equilibrium Approach



But how do we backprop?

In general we want to backprop through,

$$\ell = \mathcal{L}(h(\mathbf{z}^\star_{1:T}), \mathbf{y}_{1:T}) = \mathcal{L}(h(\mathsf{RootFind}(g_\theta; \mathbf{x}_{1:T})), \mathbf{y}_{1:T})$$

Rather than backprop through all Newton iterations, we can find the gradient in constant memory without any knowledge of RootFind,

$$\frac{\partial \ell}{\partial (\cdot)} = -\frac{\partial \ell}{\partial \mathbf{z}^\star_{1:T}} \left(J^{-1}_{g_\theta}\big|_{\mathbf{z}^\star_{1:T}}\right) \frac{\partial f_\theta(\mathbf{z}^\star_{1:T}; \mathbf{x}_{1:T})}{\partial (\cdot)} = -\frac{\partial \ell}{\partial h} \frac{\partial h}{\partial \mathbf{z}^\star_{1:T}} \left(J^{-1}_{g_\theta}\big|_{\mathbf{z}^\star_{1:T}}\right) \frac{\partial f_\theta(\mathbf{z}^\star_{1:T}; \mathbf{x}_{1:T})}{\partial (\cdot)}$$

where $J^{-1}_{g_\theta}\big|_{\mathbf{x}}$ is the inverse Jacobian of g_θ evaluated at $\mathbf{x}$.

Independent of internals of f (any NN), no storage of intermediate hidden state, one matrix multiply step

Approximating the Inverse Jacobian

Computing the exact Jacobian inverse is cubic.

Instead make a low-rank approximation using Broyden's method (quasi-Newton).

I will skip lots of details here, but essentially we can compute

$$-\frac{\partial \ell}{\partial \mathbf{z}^\star_{1:T}}\left(J^{-1}_{g_\theta}\Big|_{\mathbf{z}^\star_{1:T}}\right)$$

By instead solving for the vector-Jacobian product (which autograd gives you)

$$\left(J^\top_{g_\theta}\Big|_{\mathbf{z}^\star_{1:T}}\right)\mathbf{x}^\top + \left(\frac{\partial \ell}{\partial \mathbf{z}^\star_{1:T}}\right)^\top = \mathbf{0}$$

Properties of Deep Equilibrium Models

1. Memory

   Cheap: only store z* (equilibrium sequence) and f_\theta single layer

   vector-jacobian has dim N x Td

   Constant

2. Any choice of f_\theta that contracts (eg through layer norm, gated activation)
3. No benefit to stacking

Evals that are sort of easy

Table 1: DEQ achieves strong performance on the long-range copy-memory task.

	Models (Size)			
	DEQ-Transformer (ours) (14K)	TCN [7] (16K)	LSTM [26] (14K)	GRU [14] (14K)
Copy Memory T=400 Loss	**3.5e-6**	**2.7e-5**	0.0501	0.0491

Table 2: DEQ achieves competitive performance on word-level Penn Treebank language modeling (on par with SOTA results, without fine-tuning steps [34]). †The memory footprints are benchmarked (for fairness) on input sequence length 150 and batch size 15, which does not reflect the actual hyperparameters used; the values also do *not* include the memory for word embeddings.

Word-level Language Modeling w/ Penn Treebank (PTB)				
Model	# Params	Non-embedding model size	Test perplexity	Memory†
Variational LSTM [22]	66M	-	73.4	-
NAS Cell [55]	54M	-	62.4	-
NAS (w/ black-box hyperparameter tuner) [32]	24M	20M	59.7	-
AWD-LSTM [34]	24M	20M	58.8	-
DARTS architecture search (second order) [29]	23M	20M	**55.7**	-
60-layer TrellisNet (w/ auxiliary loss, w/o MoS) [8]	24M	20M	57.0	8.5GB
DEQ-TrellisNet (ours)	24M	20M	57.1	**1.2GB**

Evals that are sort of real

Word-level Language Modeling w/ WikiText-103 (WT103)				
Model	# Params	Non-Embedding Model Size	Test perplexity	Memory†
Generic TCN [7]	150M	34M	45.2	-
Gated Linear ConvNet [17]	230M	-	37.2	-
AWD-QRNN [33]	159M	51M	33.0	7.1GB
Relational Memory Core [40]	195M	60M	31.6	-
Transformer-XL (X-large, adaptive embed., on TPU) [16]	257M	224M	**18.7**	12.0GB
70-layer TrellisNet (+ auxiliary loss, etc.) [8]	180M	45M	29.2	24.7GB
70-layer TrellisNet with *gradient checkpointing*	180M	45M	29.2	5.2GB
DEQ-TrellisNet (ours)	180M	45M	**29.0**	**3.3GB**
Transformer-XL (medium, 16 layers)	165M	44M	24.3	8.5GB
DEQ-Transformer (medium, ours).	172M	43M	24.2	**2.7GB**
Transformer-XL (medium, 18 layers, adaptive embed.)	110M	72M	23.6	9.0GB
DEQ-Transformer (medium, adaptive embed., ours)	110M	70M	**23.2**	3.7GB
Transformer-XL (small, 4 layers)	139M	4.9M	35.8	4.8GB
Transformer-XL (small, weight-tied 16 layers)	138M	4.5M	34.9	6.8GB
DEQ-Transformer (small, ours).	138M	4.5M	**32.4**	**1.1GB**

Is Slow

Table 4: Runtime ratios between DEQs and corresponding deep networks at training and inference ($> 1\times$ implies DEQ is slower). The ratios are benchmarked on WikiText-103.

DEQ / 18-layer Transformer		DEQ / 70-layer TrellisNet	
Training	Inference	Training	Inference
2.82×	1.76×	2.40×	1.64×