Meta-learning methods for runtime reduction in Deep Equilibrium Models

Tanya Dixit

Australian National University

tanya.dixit@anu.edu.au

August 24, 2020

Overview

1. DEQ Origins
   - Universal Transformer
   - TrellisNet
2. Deep Equilibrium Models - Overview
   - Forward Pass
   - Backward Pass
   - Convert any model to DEQ
3. Meta-learning

DEQ Origins - Universal Transformer

Motivation

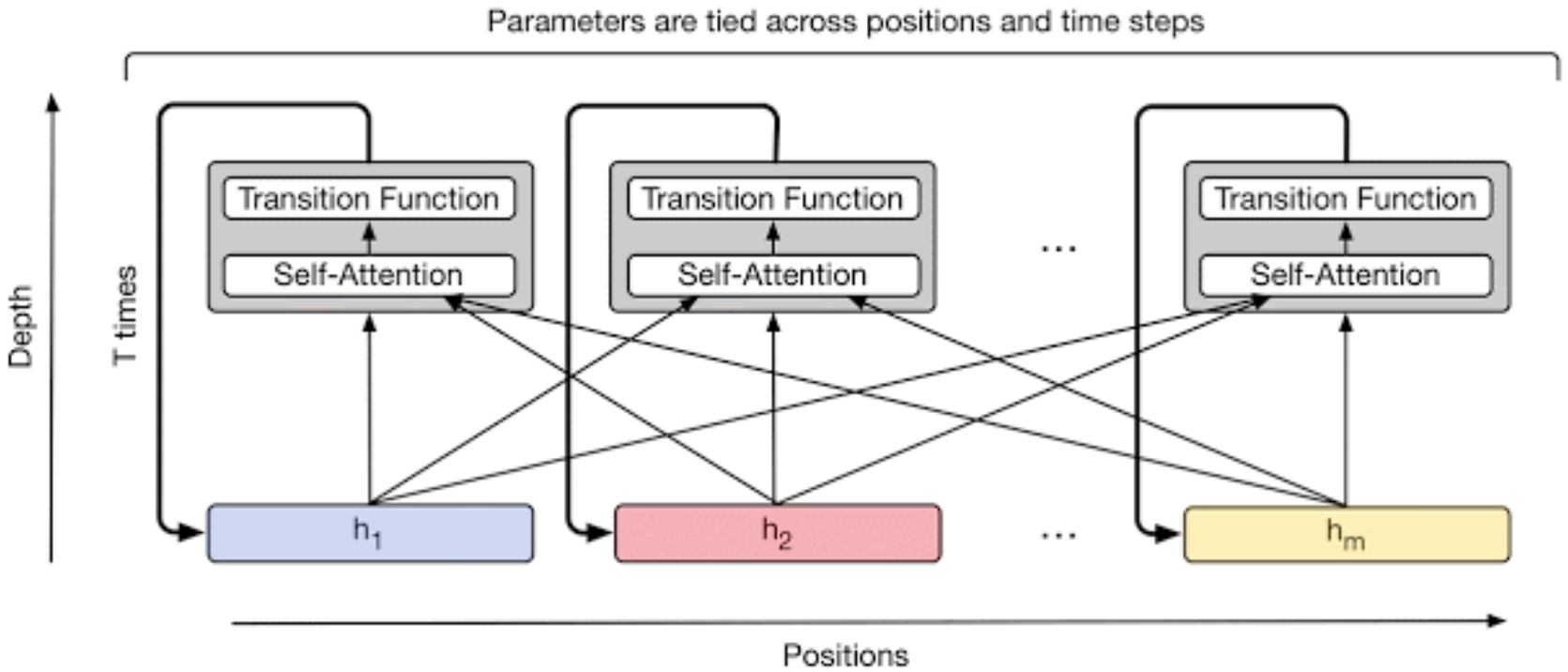
- Recurrence important for some tasks
- Transformer is not computationally universal (limits on seq len on num layers)
- Same computation to all inputs in transformer (complexity)

DEQ Origins - Universal Transformer

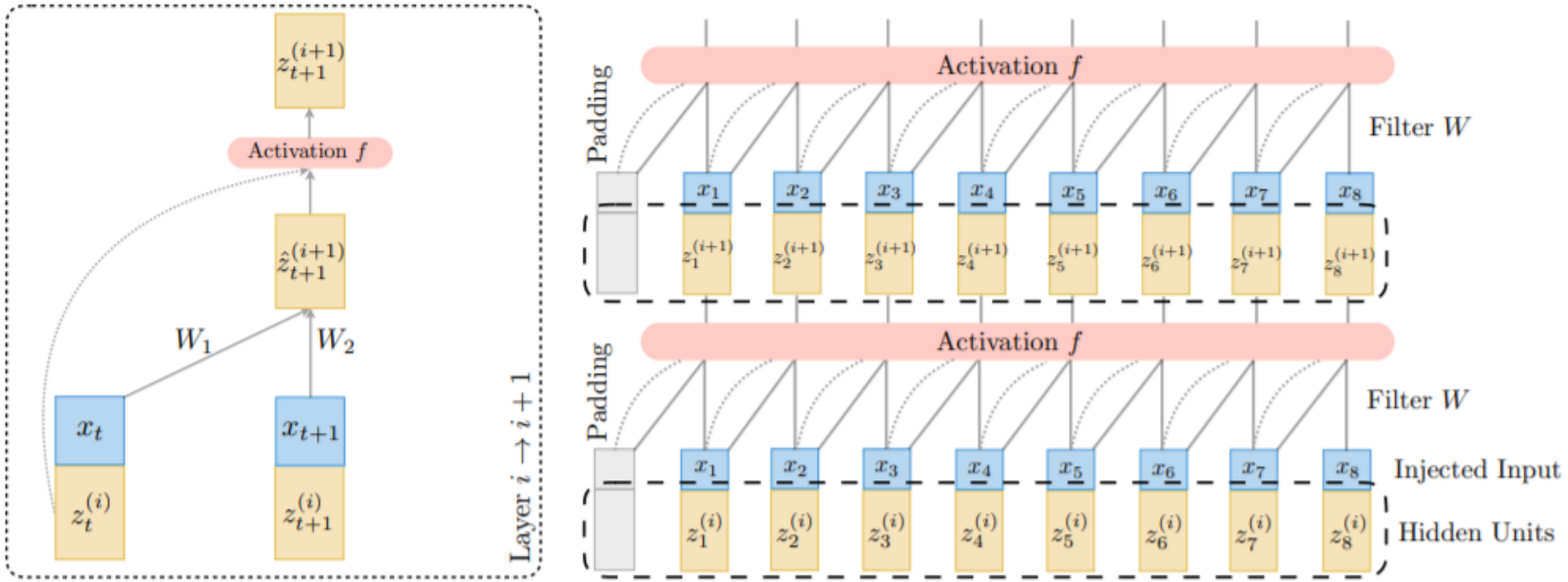
How is Universal Transformer different and similar to Transformer?

- Parallel-in-time
- Self-attention
- Weight-tied
- Global receptive field is intact
- Each recurrent step refines representations

DEQ Origins - Universal Transformer



DEQ Origins - TrellisNet



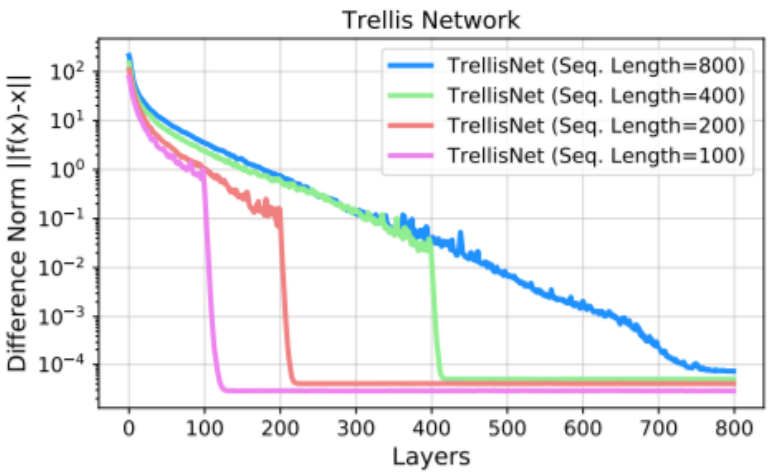
(a) TrellisNet at an atomic level

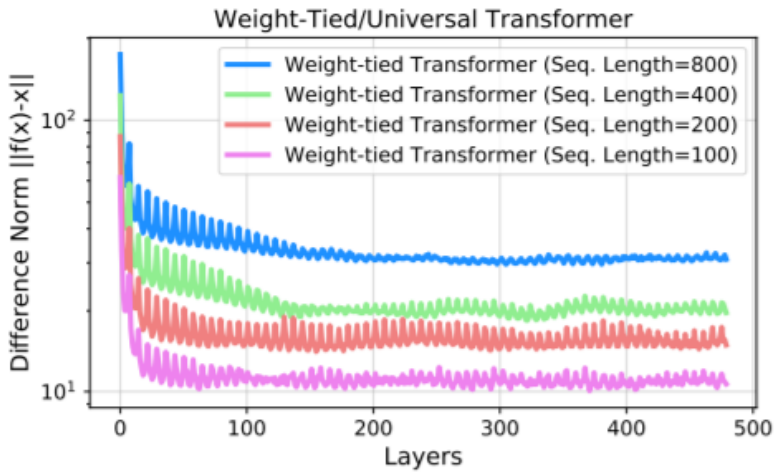
(b) TrellisNet on a sequence of units

Figure 1: The interlayer transformation of TrellisNet, at an atomic level (time steps t and $t+1$, layers i and $i+1$) and on a longer sequence (time steps 1 to 8, layers i and $i+1$).

DEQ Origins

The activations of different layers in TrellisNet and Universal Transformer converge, while oscillating around a fixed point/value.





Deep Equilibrium Models - Forward Pass

As seen before, we apply a set of non-linear transformations T times and see the activations converge. We hypothesize and observe that such weight-tied models tend to converge to a fixed point as depth increases towards infinity

Take a network as a set of non-linear transformations on an input.

$$z_{1:T}^{[i+1]} = f_\theta(z_{1:T}^{[i]}; x_{1:T}), i = 0, ..., L-1, z_{1:T}^{[0]} = 0 \tag{1}$$

If considering equilibrium conditions,

$$\lim_{i \to \infty} z_{1:T}^{[i]} = \lim_{i \to \infty} f_\theta(z_{1:T}^{[i]}; x_{1:T}) \equiv f_\theta(z_{1:T}^{*}; x_{1:T}) = z_{1:T}^{*} \tag{2}$$

Deep Equilibrium Models - Convert any model to DEQ

A traditional deep learning network is written as:

$$z^{[i+1]} = \sigma^{[i]}(W^{[i]}z^{[i]} + b^{[i]}), i = 0, .., L-1, z^{[0]} = x \tag{3}$$

can be written as a weight-tied, input-injected network like this. Note the repeat application of this equation will result in the same network as in equation (3)

$$\widetilde{z}^{[i+1]} = \sigma(W_z\widetilde{z}^{[i]} + W_x x + \widetilde{b}), i = 0, ..., L-1 \tag{4}$$

$$W_z = \begin{bmatrix} 0 & 0 & \dots & 0 & 0 \\ W^{[1]} & 0 & \dots & 0 & 0 \\ 0 & W^{[2]} & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & W^{[L-1]} & 0 \end{bmatrix} \tag{5}$$

Deep Equilibrium Models - Convert any model to DEQ

$$W_x = \begin{bmatrix} W^{[0]} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{6}$$

$$\widetilde{b} = \begin{bmatrix} b^{[0]} \\ b^{[1]} \\ \vdots \\ b^{[L-1]} \end{bmatrix} \tag{7}$$

$$\sigma = \begin{bmatrix} \sigma^{[0]} \\ \sigma^{[1]} \\ \vdots \\ \sigma^{[L-1]} \end{bmatrix} \tag{8}$$

Deep Equilibrium Models - Convert any model to DEQ

$$\widetilde{z}^{[1]} = \begin{bmatrix} \sigma^{[0]} \\ \sigma^{[1]} \\ \sigma^{[2]} \\ \sigma^{[3]} \end{bmatrix} \left(\begin{bmatrix} 0 & 0 & 0 & 0 \\ W^{[1]} & 0 & 0 & 0 \\ 0 & W^{[2]} & 0 & 0 \\ 0 & 0 & 0 & W^{[3]} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} W^{[0]} \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} x \\ 0 \\ 0 \\ 0 \end{bmatrix} + \widetilde{b} \right) \quad (9)$$

$$\widetilde{z}^{[2]} = \begin{bmatrix} \sigma^{[0]} \\ \sigma^{[1]} \\ \sigma^{[2]} \\ \sigma^{[3]} \end{bmatrix} \left(\begin{bmatrix} 0 & 0 & 0 & 0 \\ W^{[1]} & 0 & 0 & 0 \\ 0 & W^{[2]} & 0 & 0 \\ 0 & 0 & 0 & W^{[3]} \end{bmatrix} \begin{bmatrix} \widetilde{z}^{[1]} \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} W^{[0]} \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} x \\ 0 \\ 0 \\ 0 \end{bmatrix} + \widetilde{b} \right) \quad (10)$$

Think of $\widetilde{z}^{L}$ as

$$\widetilde{z}^{[4]} = \begin{bmatrix} z^{[1]} \\ z^{[2]} \\ z^{[3]} \\ z^{[4]} \end{bmatrix} \quad (11)$$

Deep Equilibrium Models - Forward Pass

$$g_\theta(z^*_{1:T}; x_{1:T}) = f_\theta(z^*_{1:T}; x_{1:T}) - z^*_{1:T} \to 0 \tag{12}$$

$z^*_{1:T}$ is the root of g_θ

Deep Equilibrium Models - Forward Pass

Equation (4) is a set of non-linear equations. Solution methods include Newton's iterative method.

Newton iterations

$$z_{1:T}^{[i+1]} = z_{1:T}^{[i]} - \alpha B g_\theta(z_{1:T}^{[i]}; x_{1:T}) \tag{13}$$

But we don't want to calculate inverse jacobian (B) at every iteration of the method - expensive operation. Solution - **Broyden Method**

Deep Equilibrium Models - Broyden Method

Makes low-rank updates to approximate the Jacobian inverse:

$$J_{g\theta}^{-1}\,|_{z_{1:T}^{[i+1]}} \approx B_{g\theta}^{i+1} = B_{g\theta}^{i} + \frac{\Delta z^{[i+1]} - B_{g\theta}^{[i]} \Delta g_{\theta}^{[i+1]}}{\Delta z^{[i+1]T} B_{g\theta}^{[i]} \Delta g_{\theta}^{[i+1]}} \Delta z^{[i+1]T} B_{g\theta}^{[i]} \tag{14}$$

where

$$\Delta z^{[i+1]} = z_{1:T}^{[i+1]} - z_{1:T}^{[i]}, \Delta g_{\theta}^{[i+1]} = g_{\theta}(z_{1:T}^{[i+1]}; x_{1:T}) - g_{\theta}(z_{1:T}^{[i]}; x_{1:T})$$

Deep Equilibrium Models - The issue with Broyden

To quote the authors - "Unlike conventional deep networks that come with a fixed number L of layers, the runtime of DEQ depends strongly on the number of Broyden steps to reach the equilibrium."

Meta-Learning

As Broyden is an iterative algorithm like Gradient Descent, we model this project on the paper "Learning to learn by gradient descent by gradient descent", where

Meta-Learning Basic Idea

As Broyden is an iterative algorithm like Gradient Descent, we model this project on the paper "Learning to learn by gradient descent by gradient descent", where

$$\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t), \phi) \tag{15}$$

Meta-Learning

How does it learn?
We have an **optimizee** network (params θ) (the original neural network) and an **optimizer** network (params ϕ)
We update θ based on output from the Optimizer network - An LSTM meta-learner

Meta-Learning

Training the meta-learner by minimizing loss.
The expected loss is:

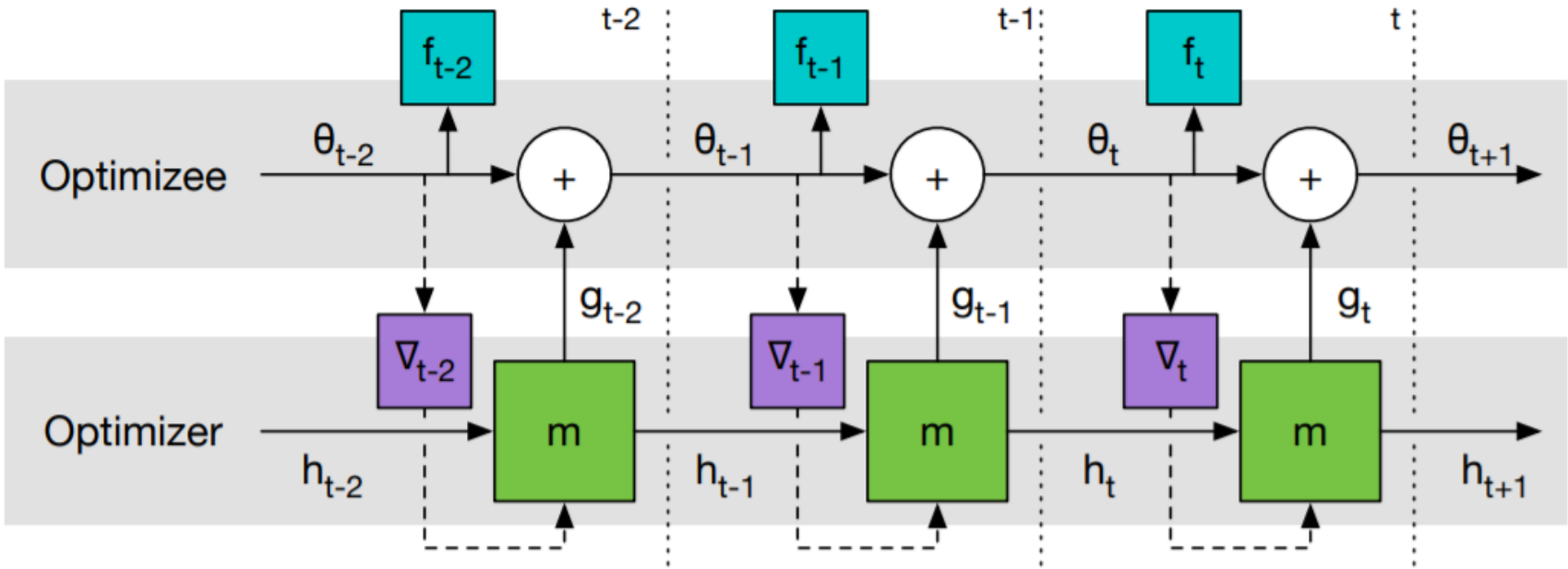
$$\mathcal{L}(\phi) = \mathbf{E_f}[f(\theta^*(f, \phi))] = \mathbf{E_f}[\Sigma_{t=1}^{T} w_t f(\theta_t)]$$

$$\theta_{t+1} = \theta_t + g_t,$$
$$\begin{bmatrix} g_t \\ h_{t+1} \end{bmatrix} = \mathrm{m}(\nabla f_t, h_t, \phi) \quad (16)$$

For using the above concept in this research, we formulate the LSTM meta-optimizer to take context information as inputs and try to reduce the number of iterations.

Meta-Learning

Computational Graph unrolled at every step



Meta-Learning Extension - Generalization

Algorithm 1 Train Meta-Learner

Input: Meta-training set $\mathscr{D}_{meta-train}$, Learner M with parameters θ, Meta-Learner R with parameters Θ.

$\Theta_0 \leftarrow$ random initialization

for $d = 1, n$ **do**
  $D_{train}, D_{test} \leftarrow$ random dataset from $\mathscr{D}_{meta-train}$
  $\theta_0 \leftarrow c_0$ ▷ Intialize learner parameters
  for $t = 1, T$ **do**
    $\mathbf{X}_t, \mathbf{Y}_t \leftarrow$ random batch from D_{train}
    $\mathcal{L}_t \leftarrow \mathcal{L}(M(\mathbf{X}_t; \theta_{t-1}), \mathbf{Y}_t)$ ▷ Get loss of learner on train batch
    $c_t \leftarrow R((\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t); \Theta_{d-1})$ ▷ Get output of meta-learner using Equation 2
    $\theta_t \leftarrow c_t$ ▷ Update learner parameters
  end for
  $\mathbf{X}, \mathbf{Y} \leftarrow D_{test}$
  $\mathcal{L}_{test} \leftarrow \mathcal{L}(M(\mathbf{X}; \theta_T), \mathbf{Y})$ ▷ Get loss of learner on test batch
  Update Θ_d using $\nabla_{\Theta_{d-1}} \mathcal{L}_{test}$ ▷ Update meta-learner parameters
end for

References

John Smith (2012)

Title of the publication

Journal Name 12(3), 45 – 678.

The End