University of Stuttgart
Institute for Parallel and Distributed Systems
Analytic Computing

March 1, 2023

ML in the Science
Deep Equilibrium Models

Daniel Frank

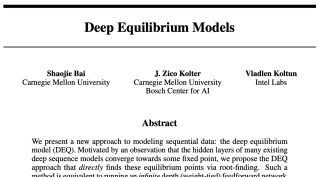
Outline

1. What are deep equilibrium network?
   - Forward Pass
   - Backward Pass
   - Example
2. Monotone equilibrium networks
   - Motivation
   - Formal Definition
   - Existence of a unique solution
3. Connection to System Theory
   - Nonlinear system identification
   - Recurrent Neural Networks as linear, time-invariant (LTI) system
   - Recurrent equilibrium network

What are deep equilibrium network?

Deep equilibrium network in Machine Learning

Where do they come from?



Deep Equilibrium Models

Shaojie Bai
Carnegie Mellon University

J. Zico Kolter
Carnegie Mellon University
Bosch Center for AI

Vladlen Koltun
Intel Labs

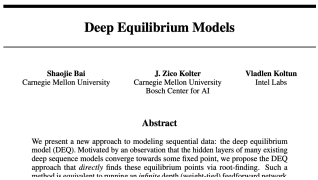
Abstract

We present a new approach to modeling sequential data: the deep equilibrium model (DEQ). Motivated by an observation that the hidden layers of many existing deep sequence models converge towards some fixed point, we propose the DEQ approach that *directly* finds these equilibrium points via root-finding. Such a

Figure 1: From NeurIPS 2019, *about 400 citations* [Bai et al., 2019]

Deep equilibrium network in Machine Learning

Where do they come from?



Deep Equilibrium Models

Shaojie Bai
Carnegie Mellon University

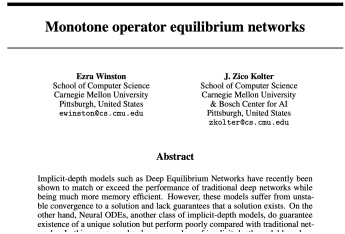
J. Zico Kolter
Carnegie Mellon University
Bosch Center for AI

Vladlen Koltun
Intel Labs

Abstract

We present a new approach to modeling sequential data: the deep equilibrium model (DEQ). Motivated by an observation that the hidden layers of many existing deep sequence models converge towards some fixed point, we propose the DEQ approach that *directly* finds these equilibrium points via root-finding. Such a

Figure 1: From NeurIPS 2019, *about 400 citations* [Bai et al., 2019]



Monotone operator equilibrium networks

Ezra Winston
School of Computer Science
Carnegie Mellon University
Pittsburgh, United States
ewinston@cs.cmu.edu

J. Zico Kolter
School of Computer Science
Carnegie Mellon University
& Bosch Center for AI
Pittsburgh, United States
zkolter@cs.cmu.edu

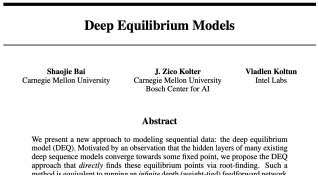
Abstract

Implicit-depth models such as Deep Equilibrium Networks have recently been shown to match or exceed the performance of traditional deep networks while being much more memory efficient. However, these models suffer from unstable convergence to a solution and lack guarantees that a solution exists. On the other hand, Neural ODEs, another class of implicit-depth models, do guarantee existence of a unique solution but perform poorly compared with traditional net-

Figure 2: From NeurIPS 2020, *about 70 citations* [Winston and Kolter, 2020]

Deep equilibrium network in Machine Learning

Where do they come from?



Deep Equilibrium Models

Shaojie Bai
Carnegie Mellon University

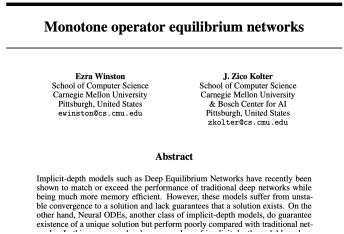
J. Zico Kolter
Carnegie Mellon University
Bosch Center for AI

Vladlen Koltun
Intel Labs

Abstract

We present a new approach to modeling sequential data: the deep equilibrium model (DEQ). Motivated by an observation that the hidden layers of many existing deep sequence models converge towards some fixed point, we propose the DEQ approach that *directly* finds these equilibrium points via root-finding. Such a

Figure 1: From NeurIPS 2019, *about 400 citations* [Bai et al., 2019]



Monotone operator equilibrium networks

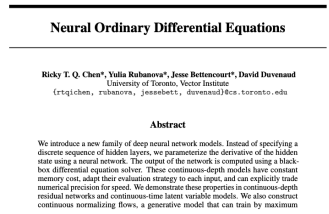
Ezra Winston
School of Computer Science
Carnegie Mellon University
Pittsburgh, United States
ewinston@cs.cmu.edu

J. Zico Kolter
School of Computer Science
Carnegie Mellon University
& Bosch Center for AI
Pittsburgh, United States
zkolter@cs.cmu.edu

Abstract

Implicit-depth models such as Deep Equilibrium Networks have recently been shown to match or exceed the performance of traditional deep networks while being much more memory efficient. However, these models suffer from unsta- ble convergence to a solution and lack guarantees that a solution exists. On the other hand, Neural ODEs, another class of implicit-depth models, do guarantee existence of a unique solution but perform poorly compared with traditional net-

Figure 2: From NeurIPS 2020, *about 70 citations* [Winston and Kolter, 2020]



Neural Ordinary Differential Equations

Ricky T. Q. Chen*, Yulia Rubanova*, Jesse Bettencourt*, David Duvenaud
University of Toronto, Vector Institute
{rtqichen, rubanova, jessebett, duvenaud}@cs.toronto.edu

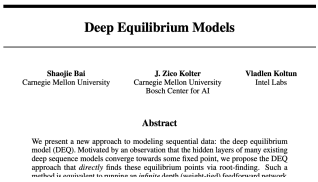
Abstract

We introduce a new family of deep neural network models. Instead of specifying a discrete sequence of hidden layers, we parameterize the derivative of the hidden state using a neural network. The output of the network is computed using a black-box differential equation solver. These continuous-depth models have constant memory cost, adapt their evaluation strategy to each input, and can explicitly trade numerical precision for speed. We demonstrate these properties in continuous-depth residual networks and continuous-time latent variable models. We also construct continuous normalizing flows, a generative model that can train by maximum

Figure 3: From NeurIPS 2018, *about 2700 citations* [Chen et al., 2018]

Deep equilibrium network in Machine Learning

Where do they come from?



Deep Equilibrium Models

Shaojie Bai
Carnegie Mellon University

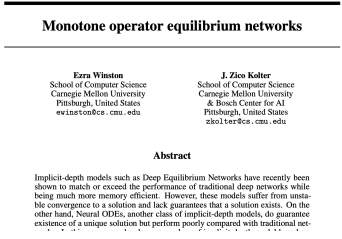
J. Zico Kolter
Carnegie Mellon University
Bosch Center for AI

Vladlen Koltun
Intel Labs

Abstract

We present a new approach to modeling sequential data: the deep equilibrium model (DEQ). Motivated by an observation that the hidden layers of many existing deep sequence models converge towards some fixed point, we propose the DEQ approach that *directly* finds these equilibrium points via root-finding. Such a

Figure 1: From NeurIPS 2019, *about 400 citations* [Bai et al., 2019]



Monotone operator equilibrium networks

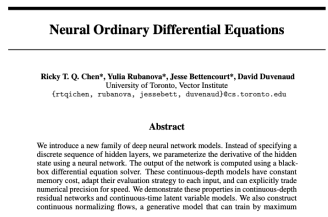
Ezra Winston
School of Computer Science
Carnegie Mellon University
Pittsburgh, United States
ewinston@cs.cmu.edu

J. Zico Kolter
School of Computer Science
Carnegie Mellon University
& Bosch Center for AI
Pittsburgh, United States
zkolter@cs.cmu.edu

Abstract

Implicit-depth models such as Deep Equilibrium Networks have recently been shown to match or exceed the performance of traditional deep networks while being much more memory efficient. However, these models suffer from unstable convergence to a solution and lack guarantees that a solution exists. On the other hand, Neural ODEs, another class of implicit-depth models, do guarantee existence of a unique solution but perform poorly compared with traditional net-

Figure 2: From NeurIPS 2020, *about 70 citations* [Winston and Kolter, 2020]



Neural Ordinary Differential Equations

Ricky T. Q. Chen*, Yulia Rubanova*, Jesse Bettencourt*, David Duvenaud
University of Toronto, Vector Institute
{rtqichen, rubanova, jessebett, duvenaud}@cs.toronto.edu

Abstract

We introduce a new family of deep neural network models. Instead of specifying a discrete sequence of hidden layers, we parameterize the derivative of the hidden state using a neural network. The output of the network is computed using a black-box differential equation solver. These continuous-depth models have constant memory cost, adapt their evaluation strategy to each input, and can explicitly trade numerical precision for speed. We demonstrate these properties in continuous-depth residual networks and continuous-time latent variable models. We also construct continuous normalizing flows, a generative model that can train by maximum

Figure 3: From NeurIPS 2018, *about 2700 citations* [Chen et al., 2018]

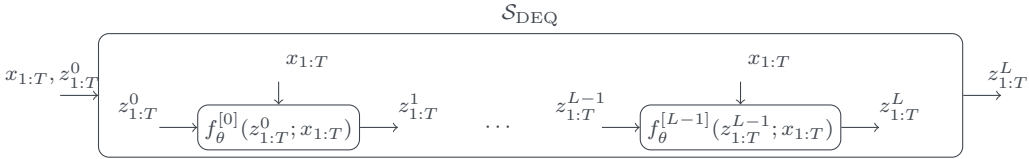
Why are they interesting?

- Training of DEQ only requires constant memory
- Achieve state-of-the-art performance compared to standard deep learning approaches
- DEQ have a strong relation to system theory

Deep equilibrium networks - Forward Pass

[Bai et al., 2019]

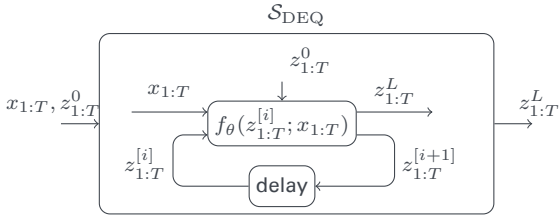
- $\mathcal{S}_{\text{DEQ}}$ maps an input sequence $x_{1:T}$ to an output sequence $z^L_{1:T}$. The input is *injected* at each layer, the number of layer is denoted by L.



Deep equilibrium networks - Forward Pass

[Bai et al., 2019]

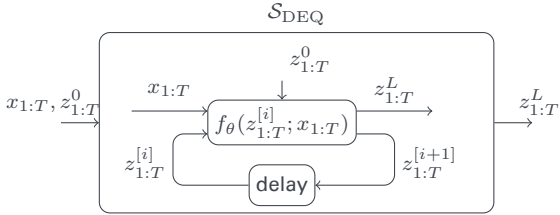
- $\mathcal{S}_{\mathrm{DEQ}}$ maps an input sequence $x_{1:T}$ to an output sequence $z^L_{1:T}$. The input is *injected* at each layer, the number of layer is denoted by L.
- Layers are weight tied $f_\theta(z^0_{1:T}; x) = f^{[i]}_\theta(z^0_{1:T}; x_{1:T})$ for all $i = 0, \ldots, L-1$



Deep equilibrium networks - Forward Pass

[Bai et al., 2019]

- $\mathcal{S}_{\mathrm{DEQ}}$ maps an input sequence $x_{1:T}$ to an output sequence $z^L_{1:T}$. The input is *injected* at each layer, the number of layer is denoted by L.
- Layers are weight tied $f_\theta(z^0_{1:T}; x) = f^{[i]}_\theta(z^0_{1:T}; x_{1:T})$ for all $i = 0, \ldots, L-1$



Core idea of DEQs

- What happens for $L \to \infty$?

Deep equilibrium networks - Forward Pass

[Bai et al., 2019]

- $\mathcal{S}_{\mathrm{DEQ}}$ maps an input sequence $x_{1:T}$ to an output sequence $z^L_{1:T}$. The input is *injected* at each layer, the number of layer is denoted by L.
- Layers are weight tied $f_\theta(z^0_{1:T}; x) = f^{[i]}_\theta(z^0_{1:T}; x_{1:T})$ for all $i = 0, \ldots, L-1$

$$\mathcal{S}_{\mathrm{DEQ}}: \quad x_{1:T}, z^0_{1:T} \longrightarrow \boxed{\mathrm{RootFind}\left(\underbrace{f_\theta(z_{1:T}; x_{1:T}) - z_{1:T}}_{g_\theta}\right)} \longrightarrow z^*_{1:T}$$

Core idea of DEQs

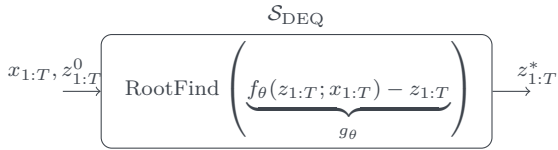
- What happens for $L \to \infty$?

$$z^*_{1:T} = f_\theta(z^*_{1:T}; x_{1:T})$$

Deep equilibrium networks - Forward Pass

[Bai et al., 2019]

- $\mathcal{S}_{\mathrm{DEQ}}$ maps an input sequence $x_{1:T}$ to an output sequence $z^L_{1:T}$. The input is *injected* at each layer, the number of layer is denoted by L.
- Layers are weight tied $f_\theta(z^0_{1:T}; x) = f^{[i]}_\theta(z^0_{1:T}; x_{1:T})$ for all $i = 0, \ldots, L-1$



Core idea of DEQs

- What happens for $L \to \infty$?

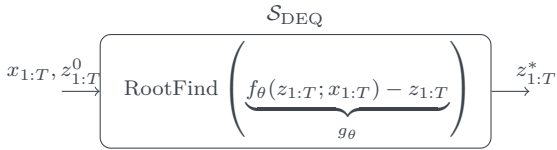
$$z^*_{1:T} = f_\theta(z^*_{1:T}; x_{1:T})$$

- How restrictive is weight tying?

Deep equilibrium networks - Forward Pass

[Bai et al., 2019]

- $\mathcal{S}_{\mathrm{DEQ}}$ maps an input sequence $x_{1:T}$ to an output sequence $z^L_{1:T}$. The input is *injected* at each layer, the number of layer is denoted by L.
- Layers are weight tied $f_\theta(z^0_{1:T}; x) = f^{[i]}_\theta(z^0_{1:T}; x_{1:T})$ for all $i = 0, \ldots, L-1$



Core idea of DEQs

- What happens for $L \to \infty$?

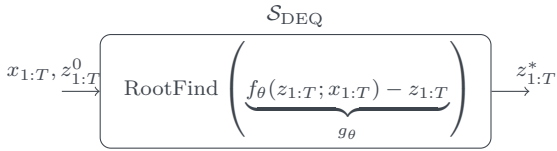
$$z^*_{1:T} = f_\theta(z^*_{1:T}; x_{1:T})$$

- How restrictive is weight tying?
  - *... Any traditional L-layer deep network (in "standard" format) can be represented by a weight-tied, input-injected network of equivalent depth, with linear increase in width ...*

Deep equilibrium networks - Forward Pass

[Bai et al., 2019]

- $\mathcal{S}_{\mathrm{DEQ}}$ maps an input sequence $x_{1:T}$ to an output sequence $z^L_{1:T}$. The input is *injected* at each layer, the number of layer is denoted by L.
- Layers are weight tied $f_\theta(z^0_{1:T}; x) = f^{[i]}_\theta(z^0_{1:T}; x_{1:T})$ for all $i = 0, \ldots, L-1$



Core idea of DEQs

- What happens for $L \to \infty$?

$$z^*_{1:T} = f_\theta(z^*_{1:T}; x_{1:T})$$

- How restrictive is weight tying?
  - *... Any traditional L-layer deep network (in "standard" format) can be represented by a weight-tied, input-injected network of equivalent depth, with linear increase in width ...*
- Any *black-box-root-finding* algorithm can be applied to solve the forward pass

Deep equilibrium network - Backward Pass

[Bai et al., 2019]

Traditional backpropagation is not suitable

- Would depend on the root finding method
- Intermediate results are not stored during the forward pass

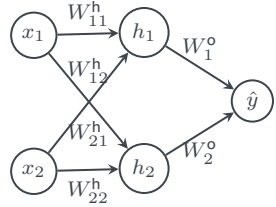


Figure 4: Backpropagation usually takes the same route as the forward pass.

Deep equilibrium network - Backward Pass

[Bai et al., 2019]

Traditional backpropagation is not suitable

- Would depend on the root finding method
- Intermediate results are not stored during the forward pass

Gradient of the Equilibrium Model

Assume the loss function

$$\ell = \mathcal{L}\left(h\left(\text{RootFind}\left(g_0; x_{1:T}\right)\right), y_{1:T}\right)$$

- $h : \mathbb{R}^{n_z} \mapsto \mathbb{R}^{n_y}$ can be any differentiable function (e.g. linear)
- $y_{1:T}$ is the ground-truth sequence
- $\mathcal{L} : \mathbb{R}^{n_y} \times \mathbb{R}^{n_y} \mapsto \mathbb{R}$ is the loss function

Deep equilibrium network - Backward Pass

[Bai et al., 2019]

Traditional backpropagation is not suitable

- Would depend on the root finding method
- Intermediate results are not stored during the forward pass

Gradient of the Equilibrium Model

Assume the loss function

$$\ell = \mathcal{L}\left(h\left(\text{RootFind}\left(g_0; x_{1:T}\right)\right), y_{1:T}\right)$$

- $h : \mathbb{R}^{n_z} \mapsto \mathbb{R}^{n_y}$ can be any differentiable function (e.g. linear)
- $y_{1:T}$ is the ground-truth sequence
- $\mathcal{L} : \mathbb{R}^{n_y} \times \mathbb{R}^{n_y} \mapsto \mathbb{R}$ is the loss function

The loss gradient with respect to $(\cdot)$ is

$$\frac{\partial \ell}{\partial (\cdot)} = -\frac{\partial \ell}{\partial h}\frac{\partial h^\star}{\partial z}_{1:T}\left(J_{g_\theta}^{-1}\Big|_{x^*_{1:T}}\right)\frac{\partial f_\theta\left(z^\star_{1:T}; x_{1:T}\right)}{\partial (\cdot)}$$

- $J_{g_\theta}^{-1}\Big|_{x^*_{1:T}}$ si the inverse Jacobian of g_θ evaluated at x

Simple DEQ - Example

Assume: **Sequence length $T = 3$, size of hidden state $n_z = 10$, input and output size $n_y = n_x = 1$:**

- Weights are randomly initialized
- Initial hidden state $z^0_{1:T} = 0$
- $W_z \in \mathbb{R}^{n_z \times n_z}$, $U_z \in \mathbb{R}^{n_z \times T}$
- Linear output layer $W_y \in \mathbb{R}^{n_y \times n_z}$

Simple DEQ - Example

Assume: **Sequence length $T = 3$, size of hidden state $n_z = 10$, input and output size $n_y = n_x = 1$:**

- Weights are randomly initialized
- Initial hidden state $z^0_{1:T} = 0$
- $W_z \in \mathbb{R}^{n_z \times n_z}$, $U_z \in \mathbb{R}^{n_z \times T}$
- Linear output layer $W_y \in \mathbb{R}^{n_y \times n_z}$

Network with fixed layer size:

$$z^{l+1}_{1:T} = \sigma\left(W_z z^l_{1:T} + U_z x_{1:T} + b_z\right) \qquad l = 1, \ldots, L-1$$
$$\hat{y}_{1:T} = W_y z^*_{1:T} + b_z$$

Listing 1: Traditional forward pass

```
# forward pass for fixed number of layers
z = torch.zeros(size=(1, n_z))
x = torch.tensor(u).reshape(1, n_x)
for l in range(L):
    z = nl(W_z(z) + U_z(x))
y_hat = W_y(z)
```

Simple DEQ - Example

Assume: **Sequence length $T = 3$, size of hidden state $n_z = 10$, input and output size $n_y = n_x = 1$:**

- Weights are randomly initialized
- Initial hidden state $z_{1:T}^0 = 0$
- $W_z \in \mathbb{R}^{n_z \times n_z}$, $U_z \in \mathbb{R}^{n_z \times T}$
- Linear output layer $W_y \in \mathbb{R}^{n_y \times n_z}$

DEQ:

$$z_{1:T}^* = \text{RootFind}\left(\sigma\left(W_z z_{1:T} + U_z x_{1:T} + b_z\right) - z_{1:T}\right)$$
$$\hat{y}_{1:T} = W_y z_{1:T}^L + b_z$$

Listing 2: Forward pass of DEQ

```
# DEQ
def g_theta(z):
    z = z.reshape(n_z,1)
    return np.squeeze(np.tanh(W_z_numpy @ z + U_z_numpy @ x + b_z_numpy) - z)

z_star, infodict, ier, mesg = fsolve(g_theta, x0=z_0, full_output=True)
z_star = z_star.reshape(n_z, 1)
y_hat_eq = W_y_numpy @ z_star + b_y_numpy
```

Simple DEQ - Example

Assume: **Sequence length $T = 3$, size of hidden state $n_z = 10$, input and output size $n_y = n_x = 1$:**

- Weights are randomly initialized
- Initial hidden state $z^0_{1:T} = 0$
- $W_z \in \mathbb{R}^{n_z \times n_z}$, $U_z \in \mathbb{R}^{n_z \times T}$
- Linear output layer $W_y \in \mathbb{R}^{n_y \times n_z}$

Network with fixed layer size:

$$z^{l+1}_{1:T} = \sigma\left(W_z z^l_{1:T} + U_z x_{1:T} + b_z\right) \qquad l = 1, \dots, L-1$$
$$\hat{y}_{1:T} = W_y z^*_{1:T} + b_z$$

DEQ:

$$z^*_{1:T} = \text{RootFind}\left(\sigma\left(W_z z_{1:T} + U_z x_{1:T} + b_z\right) - z_{1:T}\right)$$
$$\hat{y}_{1:T} = W_y z^L_{1:T} + b_z$$

Comparison:

```
Number of finite layers: 0       || z^L - z^* ||^2: 0.7032
Number of finite layers: 1       || z^L - z^* ||^2: 0.3898
Number of finite layers: 2       || z^L - z^* ||^2: 0.2898
Number of finite layers: 3       || z^L - z^* ||^2: 0.1621
Number of finite layers: 4       || z^L - z^* ||^2: 0.09451
Number of finite layers: 10      || z^L - z^* ||^2: 0.001685
Number of finite layers: 20      || z^L - z^* ||^2: 7.595e-06
Number of finite layers: 30      || z^L - z^* ||^2: 7.069e-08
```

Monotone equilibrium networks

2

Example continued

Do the solutions $z^*_{1:T}$ and $z^L_{1:T}$ always converge for sufficient large L?

- Let the weight of the hidden layer be initialized by a normal distribution

Listing 3: Added line 2

```
1 W_z = torch.nn.Linear(in_features=n_z, out_features=n_z, bias=True)
2 torch.nn.init.normal_(W_z.weight)
3 U_z = torch.nn.Linear(in_features=n_x, out_features=n_z, bias=False)
4 W_y = torch.nn.Linear(in_features=n_z, out_features=n_x, bias=True)
```

Example continued

Do the solutions $z^*_{1:T}$ and $z^L_{1:T}$ always converge for sufficient large L?

- Let the weight of the hidden layer be initialized by a normal distribution

Listing 3: Added line 2

```
W_z = torch.nn.Linear(in_features=n_z, out_features=n_z, bias=True)
torch.nn.init.normal_(W_z.weight)
U_z = torch.nn.Linear(in_features=n_x, out_features=n_z, bias=False)
W_y = torch.nn.Linear(in_features=n_z, out_features=n_x, bias=True)
```

Comparison:

```
    Number of finite layers: 0       || z^L - z^* ||^2: 0.4664
    Number of finite layers: 1       || z^L - z^* ||^2: 0.332
    Number of finite layers: 2       || z^L - z^* ||^2: 1.035
    Number of finite layers: 3       || z^L - z^* ||^2: 1.834
    Number of finite layers: 4       || z^L - z^* ||^2: 2.348
    Number of finite layers: 10      || z^L - z^* ||^2: 2.75
    Number of finite layers: 20      || z^L - z^* ||^2: 2.724
    Number of finite layers: 30      || z^L - z^* ||^2: 2.927
```

Core idea of monotone equilibrium network (monDEQ)

[Winston and Kolter, 2020]

Finding equilibrium point as operator splitting [Winston and Kolter, 2020, Theorem 1]

Consider (again) a weight-tied input-injected network

$$z^{k+1} = \sigma\left(W_z z^k + U_z x + b_z\right) \tag{1}$$

and an equilibrium point that remains constant after update

$$z^* = \sigma\left(W z_z^* + U_z x + b\right).$$

Finding an equilibrium point of (1) is equivalent of finding a zero of the operator splitting problem $0 \in (F + G)(z^*)$ with the operators

$$F(z) = (I - W_z)(z) - (U_z x + b), \qquad G = \partial f \tag{2}$$

and $\sigma(\cdot) = \text{prox}_f^1(\cdot)$ for some convex closed proper function f, where prox_f^α denotes the proximal operator

$$\text{prox}_f^\alpha(x) \equiv \text{argmin}_z \frac{1}{2}\|x - z\|_2^2 + \alpha f(z)$$

Core idea of monotone equilibrium network (monDEQ)

[Winston and Kolter, 2020]

Finding equilibrium point as operator splitting [Winston and Kolter, 2020, Theorem 1]

Consider (again) a weight-tied input-injected network

$$z^{k+1} = \sigma\left(W_z z^k + U_z x + b_z\right) \tag{1}$$

and an equilibrium point that remains constant after update

$$z^* = \sigma\left(W z_z^* + U_z x + b\right).$$

Finding an equilibrium point of (1) is equivalent of finding a zero of the operator splitting problem $0 \in (F + G)(z^*)$ with the operators

$$F(z) = (I - W_z)(z) - (U_z x + b), \qquad G = \partial f \tag{2}$$

and $\sigma(\cdot) = \text{prox}_f^1(\cdot)$ for some convex closed proper function f, where prox_f^α denotes the proximal operator

$$\text{prox}_f^\alpha(x) \equiv \text{argmin}_z \, \frac{1}{2}\|x - z\|_2^2 + \alpha f(z)$$

- An introduction to monotone operator theory would be required to fully appreciate the result, see [Winston and Kolter, 2020, Appendix A]
- Lets view the operator splitting as a *black-box-root-finding* algorithm

Monotone Operator

Monotone operator $F(z) = (I - W_z)(z) - (U_z x + b)$ from (2).

If

$$I - W_z \succeq mI \qquad m > 0$$

is strongly monotone, an unique equilibrium point z^* exists.

- For a scalar, linear function this connection is intuitive

$$F_{\text{scal}}(z) = \underbrace{(1 - w)}_{\text{slope}} z + \underbrace{(ux + b)}_{\text{constant}}$$

Monotone Operator

Monotone operator $F(z) = (I - W_z)(z) - (U_z x + b)$ from (2).

If

$$I - W_z \succeq mI \qquad m > 0$$

is strongly monotone, an unique equilibrium point z^* exists.

- For a scalar, linear function this connection is intuitive

$$F_{\text{scal}}(z) = \underbrace{(1-w)}_{\text{slope}} z + \underbrace{(ux+b)}_{\text{constant}}$$

```
min EW of (I-W_z): 0.6272        L: 40   || z^L - z^* ||^2: 3.999e-08
min EW of (I-W_z): 0.3782        L: 40   || z^L - z^* ||^2: 4.184e-08
min EW of (I-W_z): 0.5671        L: 40   || z^L - z^* ||^2: 3.373e-08
min EW of (I-W_z): 0.6786        L: 40   || z^L - z^* ||^2: 6.231e-08
min EW of (I-W_z): 0.8057        L: 40   || z^L - z^* ||^2: 2.551e-08
min EW of (I-W_z): 0.662         L: 40   || z^L - z^* ||^2: 3.364e-08
min EW of (I-W_z): 0.3946        L: 40   || z^L - z^* ||^2: 4.522e-08
min EW of (I-W_z): 0.6532        L: 40   || z^L - z^* ||^2: 2.656e-08
min EW of (I-W_z): 0.4264        L: 40   || z^L - z^* ||^2: 3.059e-08
min EW of (I-W_z): 0.6787        L: 40   || z^L - z^* ||^2: 5.395e-08
```

Monotone Operator

Monotone operator $F(z) = (I - W_z)(z) - (U_z x + b)$ from (2).

If

$$I - W_z \succeq mI \qquad m > 0$$

is strongly monotone, an unique equilibrium point z^* exists.

- For a scalar, linear function this connection is intuitive

$$F_{\text{scal}}(z) = \underbrace{(1-w)}_{\text{slope}} z + \underbrace{(ux+b)}_{\text{constant}}$$

```
1  min EW of (I-W_z): -2.172        L: 40   || z^L - z^* ||^2: 3.301
2  min EW of (I-W_z): -2.353        L: 40   || z^L - z^* ||^2: 2.48
3  min EW of (I-W_z): -1.519        L: 40   || z^L - z^* ||^2: 2.585
4  min EW of (I-W_z): -1.657        L: 40   || z^L - z^* ||^2: 2.839
5  min EW of (I-W_z): -0.6791       L: 40   || z^L - z^* ||^2: 2.59
6  min EW of (I-W_z): -2.574        L: 40   || z^L - z^* ||^2: 3.095
7  min EW of (I-W_z): -1.609        L: 40   || z^L - z^* ||^2: 3.514
8  min EW of (I-W_z): -3.02         L: 40   || z^L - z^* ||^2: 2.832
9  min EW of (I-W_z): -1.922        L: 40   || z^L - z^* ||^2: 2.983
10 min EW of (I-W_z): -1.023        L: 40   || z^L - z^* ||^2: 2.722
```

Monotone Operator

Monotone operator $F(z) = (I - W_z)(z) - (U_z x + b)$ from (2).

If

$$I - W_z \succeq mI \qquad m > 0$$

is strongly monotone, an unique equilibrium point z^* exists.

- For a scalar, linear function this connection is intuitive

$$F_{\text{scal}}(z) = \underbrace{(1 - w)}_{\text{slope}} z + \underbrace{(ux + b)}_{\text{constant}}$$

Introduce constraints on the weight W_z to ensure montonicity

Monotonicity is ensured $I - W_z \succeq mI$ if there exists $A,\ B \in \mathbb{R}^{n_z \times n_z}$ such that

$$W_z = (1 - m)I - A^{\mathrm{T}} A + B - B^{\mathrm{T}}.$$

- The weight W_z is now directly parametrized by A and B

Connection to System Theory

3

Inverted pendulum with torque input

Classical toy example in control

System Identification Problem

If the equations of the pendulum are not available but we are given data $\mathcal{D} = \{(u, y)_i\}_i^N$, which contains input output measurements. Finding the parameter set θ can be seen as a deep learning problem.

- The system states $x_{\text{sys}}^k = [\phi,\ \dot{\phi}]^{\mathrm{T}}$ represents the *angle* and *angular velocity*
- Input u^k represents a torque
- Output y^k is the *angle* ϕ
- One stable and one unstable equilibrium point

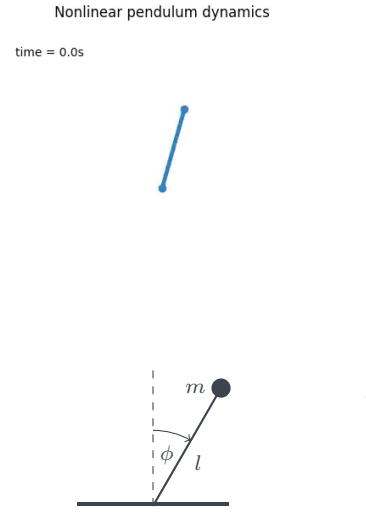


Figure 5: Parameters of single pendulum.

Generalization of Recurrent Neural Networks

LTI system with nonlinear disturbance

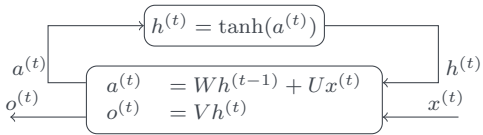


Figure 6: Recurrent neural network from [Goodfellow et al., 2016]

Generalization of Recurrent Neural Networks

LTI system with nonlinear disturbance

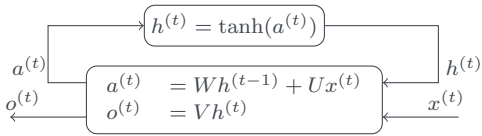


Figure 6: Recurrent neural network from [Goodfellow et al., 2016]

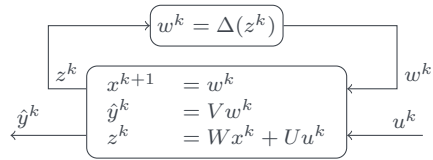


Figure 7: Using common notation from system theory.

Generalization of Recurrent Neural Networks

LTI system with nonlinear disturbance

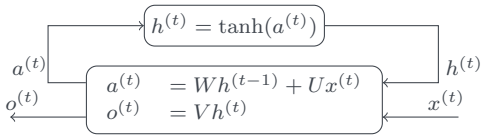


Figure 6: Recurrent neural network from [Goodfellow et al., 2016]

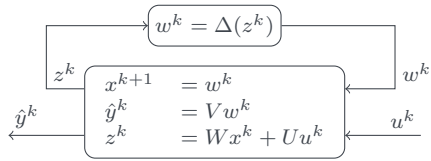


Figure 7: Using common notation from system theory.

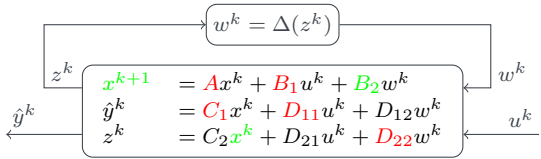


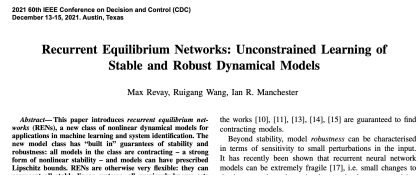
Figure 8: LTI with nonlinear disturbance.

- For $A = B_1 = C_1 = D_{11}, D_{22} = 0$, $B_2 = I$, $\Delta(\cdot) = \tanh(\cdot)$ and $h^{t-1} = x^k$ the networks in Figure 6 and Figure 8 are equivalent

Linear systems with disturbances can be analyzed with tools from robust control

Neural networks with guarantees

- Recurrent neural network (RNN) interpreted as linear systems with nonlinear disturbances can be analyzed with tools from robust control
- The neural network has system theoretic guarantees



2021 60th IEEE Conference on Decision and Control (CDC)
December 13-15, 2021. Austin, Texas

Recurrent Equilibrium Networks: Unconstrained Learning of Stable and Robust Dynamical Models

Max Revay, Ruigang Wang, Ian R. Manchester

***Abstract*— This paper introduces *recurrent equilibrium networks* (RENs), a new class of nonlinear dynamical models for applications in machine learning and system identification. The new model class has "built in" guarantees of stability and robustness: all models in the class are contracting – a strong form of nonlinear stability – and models can have prescribed Lipschitz bounds. RENs are otherwise very flexible: they can**

the works [10], [11], [13], [14], [15] are guaranteed to find contracting models.

Beyond stability, model *robustness* can be characterised in terms of sensitivity to small perturbations in the input. It has recently been shown that recurrent neural network models can be extremely fragile [17], i.e. small changes to

Figure 9: From CDC 2021, *about 8 citations*[Revay et al., 2021]

Linear systems with disturbances can be analyzed with tools from robust control

Figure 10: F16 nonlinear system identification benchmark [Noël and Schoukens, 2017]

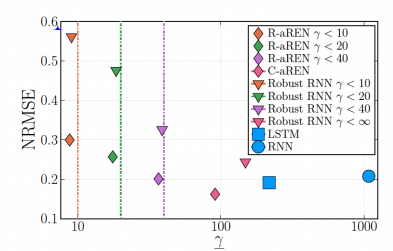


Figure 11: Robustness versus accuracy [Revay et al., 2021, Fig. 2]

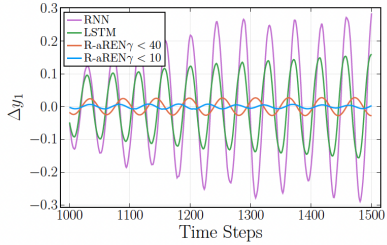


Figure 12: Stability [Revay et al., 2021, Fig. 5]

University of Stuttgart

Daniel Frank
Institute for Parallel and Distributed Systems
Analytic Computing

eMail daniel.frank@ipvs.uni-stuttgart.de
phone +49 711 685-88107
address Universitaetstraße 32
70569 Stuttgart
2.204

Literature I

[Bai et al., 2019] Bai, S., Kolter, J. Z., and Koltun, V. (2019).
Deep equilibrium models.
In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 688–699.

[Chen et al., 2018] Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. (2018).
Neural ordinary differential equations.
In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6572–6583.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016).
Deep learning.
MIT press.

[Lunze and Lunze, 1996] Lunze, J. and Lunze, J. (1996).
Regelungstechnik 1, volume 10.
Springer.

Literature II

[Noël and Schoukens, 2017] Noël, J.-P. and Schoukens, M. (2017).
F-16 aircraft benchmark based on ground vibration test data.
In *2017 Workshop on Nonlinear System Identification Benchmarks*, pages 19–23.

[Revay et al., 2021] Revay, M., Wang, R., and Manchester, I. R. (2021).
Recurrent equilibrium networks: Unconstrained learning of stable and robust dynamical models.
In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 2282–2287. IEEE.

[Willems, 1972] Willems, J. C. (1972).
Dissipative dynamical systems part i: General theory.
Archive for rational mechanics and analysis, 45(5):321–351.

[Winston and Kolter, 2020] Winston, E. and Kolter, J. Z. (2020).
Monotone operator equilibrium networks.
Advances in neural information processing systems, 33:10718–10728.

Background

Pendulum is also in LTI structure

Same representation as RNNs

Difference equation of pendulum dynamics

$$
\mathcal{G}\begin{cases} x^{k+1} = \underbrace{\begin{pmatrix} 1 & \delta \\ \frac{g\delta}{l} & 1-\frac{\mu\delta}{ml^2} \end{pmatrix}}_{A} x^k + \underbrace{\begin{pmatrix} 0 \\ -\frac{g\delta}{l} \end{pmatrix}}_{B_1} u^k + \underbrace{\begin{pmatrix} 0 \\ \frac{\delta}{ml^2} \end{pmatrix}}_{B_2} w^k \\ y^k = \underbrace{\begin{pmatrix} 1 & 0 \end{pmatrix}}_{C_1} x^k \\ z^k = \underbrace{\begin{pmatrix} 1 & 0 \end{pmatrix}}_{C_2} x^k \end{cases}
$$

$$
w^k = \Delta(z^k) = z^k - \sin(z^k)
$$

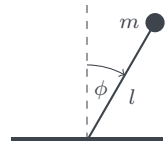


Figure 13: Parameters of single pendulum.

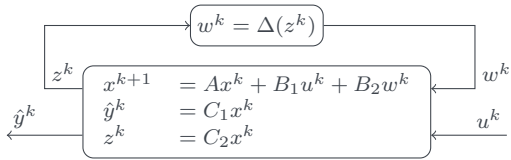


Figure 14: Pendulum dynamics in LTI structure with nonlinearity $\Delta(\cdot)$

Why is this connection interesting?

Linear systems are well understood

- Stability depends on eigenvalues of A matrix
- Well established controller design techniques

Figure 15: [Lunze and Lunze, 1996]

Why is this connection interesting?

Linear systems are well understood

- Stability depends on eigenvalues of A matrix
- Well established controller design techniques

Figure 15: [Lunze and Lunze, 1996]

Linear system with disturbances

- For known disturbances robust control methods can be applied
- Activation function of neural networks can be seen as disturbance

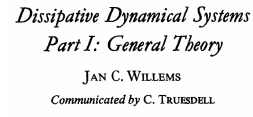


Figure 16: [Willems, 1972]

System Identification Problem

If the equations of the pendulum are not available but we are given data $\mathcal{D} = \{(u, y)_i\}_i^N$, which contains input output measurements. Finding the parameters A, B_1, B_2, C_1, C_2 can be seen as a deep learning problem.

Why is this connection interesting?

Linear systems are well understood

- Stability depends on eigenvalues of A matrix
- Well established controller design techniques

Figure 15: [Lunze and Lunze, 1996]

Linear system with disturbances

- For known disturbances robust control methods can be applied
- Activation function of neural networks can be seen as disturbance

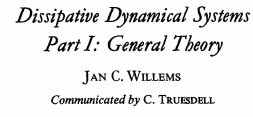


Figure 16: [Willems, 1972]

System Identification Problem

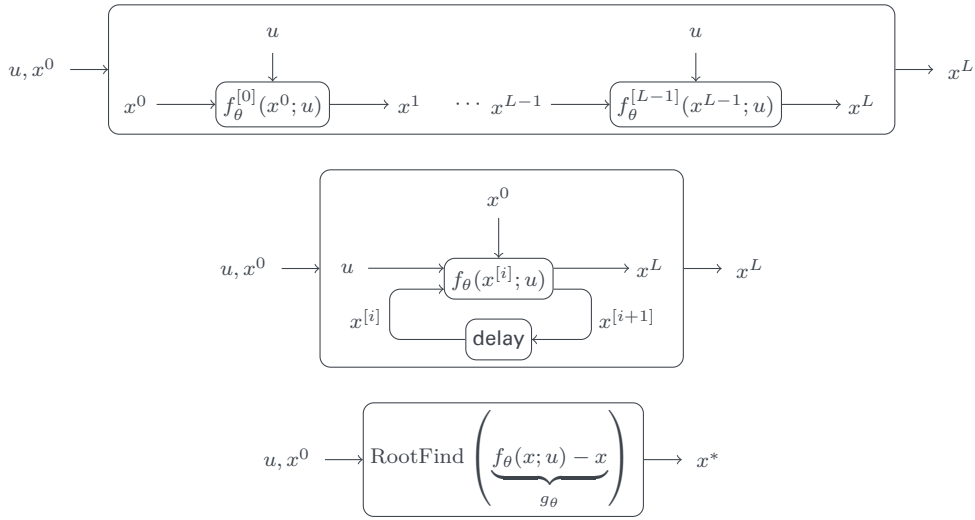
If the equations of the pendulum are not available but we are given data $\mathcal{D} = \{(u, y)_i\}_i^N$, which contains input output measurements. Finding the parameters A, B_1, B_2, C_1, C_2 can be seen as a deep learning problem.

Neural networks that can be represented in LTI structure with nonlinear disturbances can be analyzed with well established tools from robust control.

Figures

5

Figures I


u, x^0
u
x^0
$f_\theta^{[0]}(x^0; u)$
x^1
$\cdots$ x^{L-1}
u
$f_\theta^{[L-1]}(x^{L-1}; u)$
x^L
x^L
u, x^0
x^0
u
$f_\theta(x^{[i]}; u)$
x^L
x^L
$x^{[i]}$
delay
$x^{[i+1]}$
u, x^0
RootFind
$f_\theta(x; u) - x$
g_θ
x^*