Deep Equilibrium Models and Implicit Layers

An introduction by a noob...

2021-11-18

Rémi Emonet

Data Intelligence Meeting

Outline

- Quick review of Deep Networks and ResNETs
- Fixed point iteration layer, Deep Equilibrium (DEQ)
- Other Implicit Layers

Figure 2. System considering one (f), two (f ◦ f) and three (f ◦ f ◦ f) instances of the network. In all three cases, the architecture produces labels (1 × 1 output planes corresponding to the pixel at the center of the input patch. Each network instance is fed with the previous label predictions, as well as a RGB patch surrounding the pixel of interest. For space constraints, we do not show the label maps of the first instances, as they are zero maps. Adding network instances increases the context patch size seen by the architecture (both RGB pixels and previous predicted labels).

A quick story

Recurrent Convolutional Neural Networks for Scene Labeling

Pedro O. Pinheiro[1,2]
Ronan Collobert[2]
[1]Ecole Polytechnique Fédérale de Lausanne (EPFL)
[2]Idiap Research Institute, Martigny, Switzerland

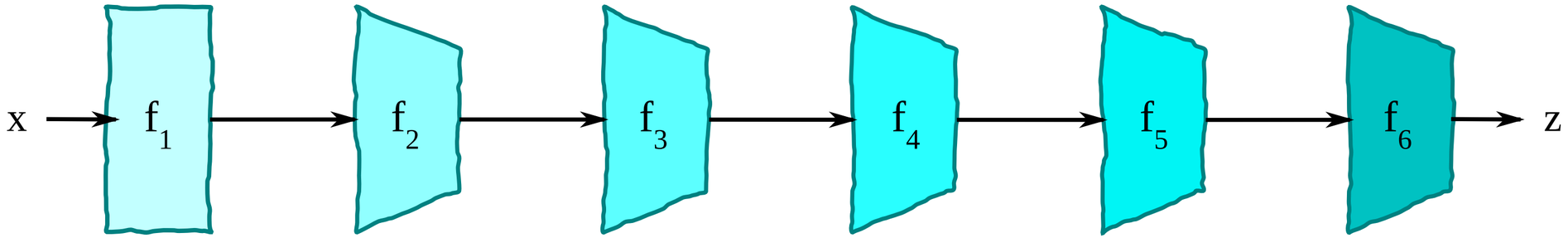
Proceedings of the 31st International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32.

Outline

- Quick review of Deep Networks and ResNETs
- Fixed point iteration layer, Deep Equilibrium (DEQ)
- Other Implicit Layers

Quick review of Deep Networks and ResNETs

A simple deep network



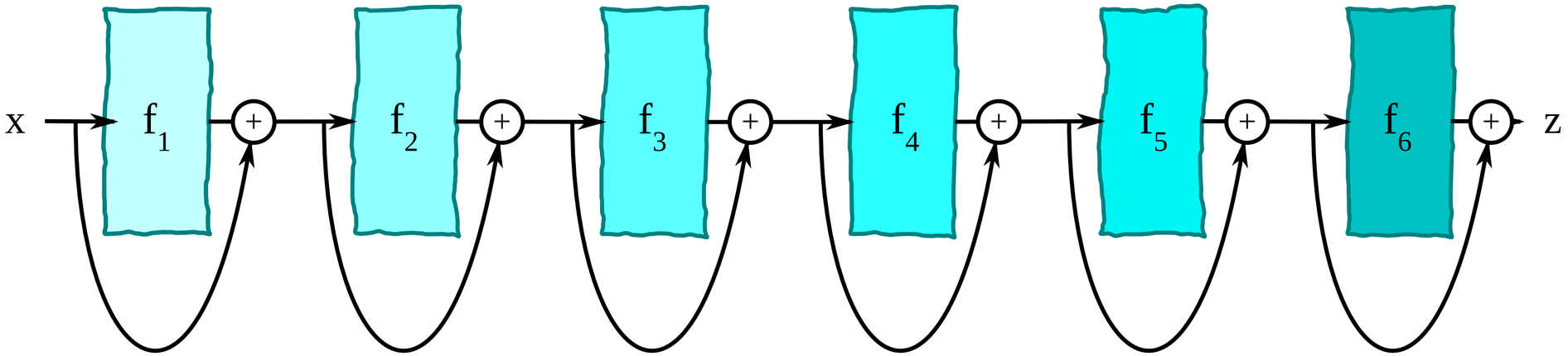
When going deeper, issue with "vanishing gradient"

- ReLU helps a lot
- but still...

Residual Network to go deeper

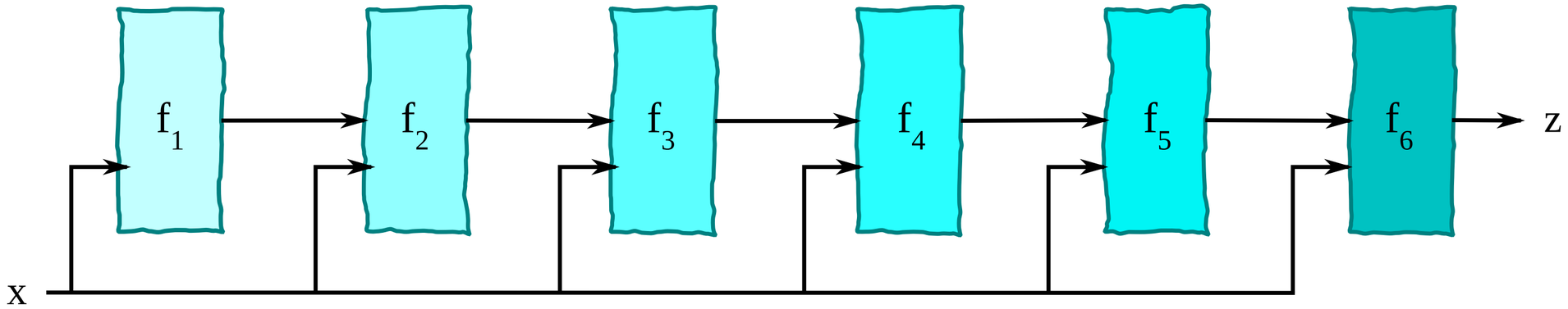
Principle:

- add skip connections
- i.e., for each layer: output = input + a value computed by a residual block



- NB: input and output have the same dimension (to make the sum).
- often, f_i is made of a few Conv/linear, with BatchNorm, ReLU

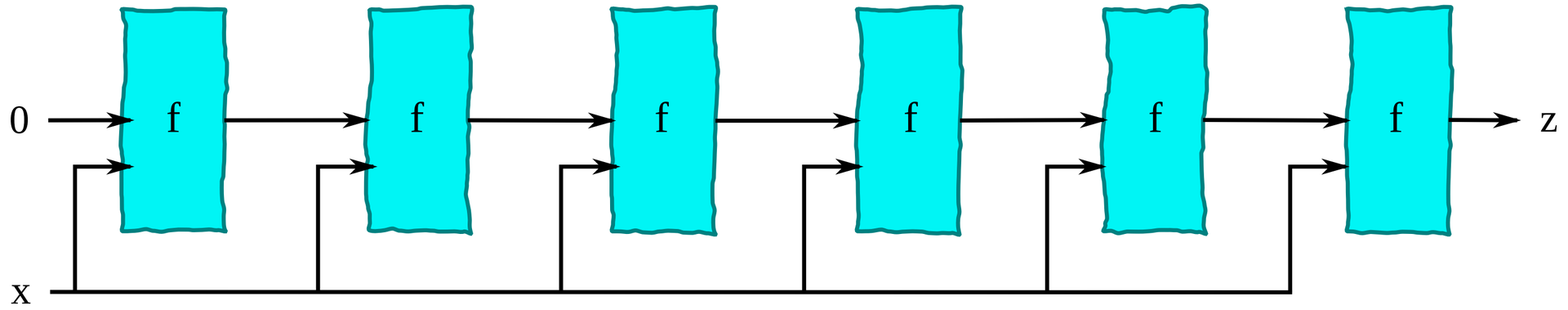
Input injection



Principle:

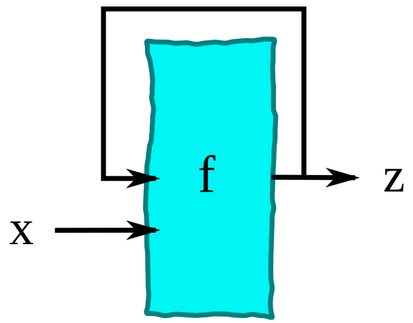
- pass the (original) input to every layer
- NB: actually used in other models like Deep Gaussian Processes

Shared parameters to allow recurrent formulation

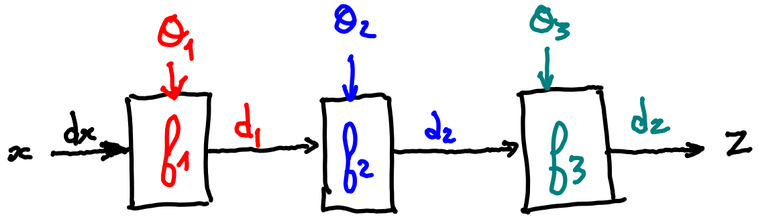


Principle:

- use the same f
  - can be as expressive as the original network
  - see next slide
- initial condition, e.g., $z_0 = 0$
- making the network recurrent
- more recurrent iterations = deeper network



Proof: a 3 layer network as a big(ger) reccurrent network



$$f(x;z) = f(x;[z_1, z_2, z_3]) \qquad z = [z_1, z_2, z_3] \in \mathbb{R}^{d_1+d_2+d_3}$$

$$f\left(x; \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix}\right) = \begin{pmatrix} f_1(x) \\ f_2(z_1) \\ f_3(z_2) \end{pmatrix}$$

$$f(x, f(x, f(x, \underbrace{0}_{\text{0}})))$$

$$\begin{pmatrix} f_1(x) \\ f_2(0) \\ f_3(0) \end{pmatrix}$$

$$\begin{pmatrix} f_1(x) \\ f_2(f_1(x)) \\ f_3(f_2(0)) \end{pmatrix}$$

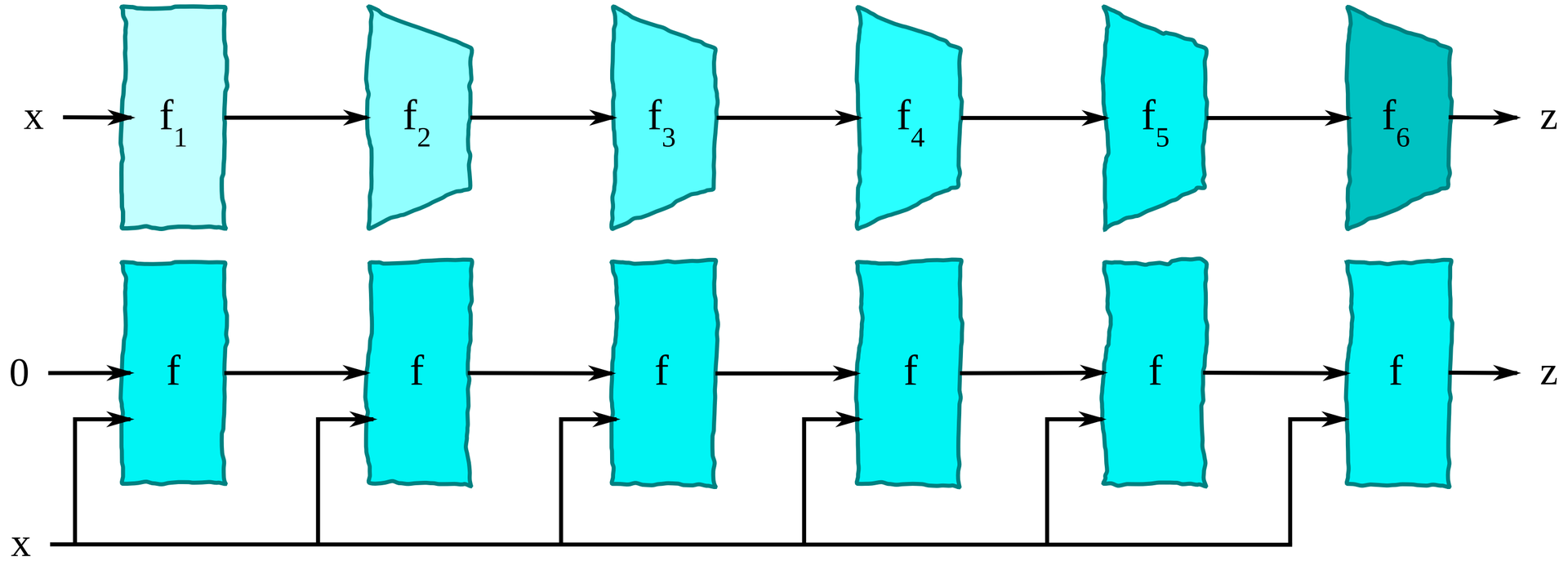
$$\begin{pmatrix} f_1(x) \\ f_2(f_1(x)) \\ f_3(f_2(f_1(x))) \end{pmatrix}$$

Outline

- Quick review of Deep Networks and ResNETs
- Fixed point iteration layer, Deep Equilibrium (DEQ)
- Other Implicit Layers

Fixed point iteration layer, Deep Equilibrium (DEQ)

Limitation of gradient descent and deep models?



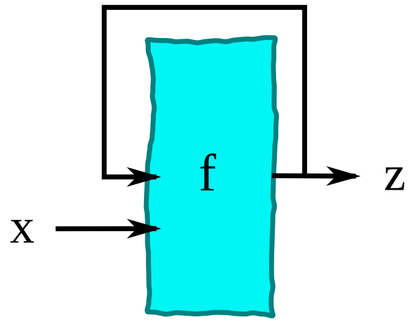
Do you see/know any problem with big/deep networks (incl. ResNETs)?

- Need to store intermediate activations
- … huge memory requirements (at training time)
- … can do "checkpointing" (not a perfect solution)

Fixed point iteration: an example of implicit layer

Implicit layer

- instead of telling how to compute the output from the input (explicit)
- we rather tell what the output should verify
- and how we actually find the output is secondary



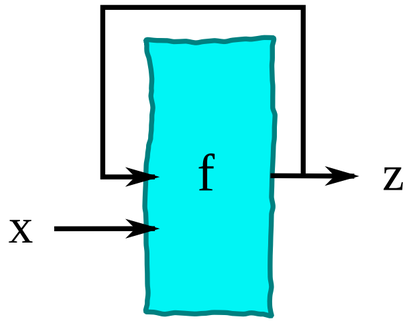
Fixed point layers are implicit layers:

- given a function f_θ parametrized by θ, and an input x
- find the output $z^\star$, which is a *fixed point* of $f_\theta(x, .)$, i.e.,
  $z^\star = f_\theta(x, z^\star)$

Fixed point iteration layers are (already) auto-differentiable

Implicit layer definition:

- given a function f_θ parametrized by θ, and an input x
- find the output $z^\star$, which is a *fixed point* of $f_\theta(x, .)$, i.e., $z^\star = f_\theta(x, z^\star)$



Possible solution:

- iteratively apply f_θ until convergence
- like an infinitely deep network
- reccurent iterations are unrolled: $z^\star = g(g(g(g(g(g(...g(g(g(0)))))))))$, with $g : z \mapsto f_\theta(x, z)$
- computing the gradient is done by the frameworks
- … but requires storing intermediate results

… let's call maths to the rescue

Differentiating an implicit layer?

… more precisely, integrating into a deep/differentiation-based learning framework

Backprop in automatic differentiation frameworks

- Two differentiation modes
  - JVP, Jacobian-Vector-Product, need to implement $v \mapsto \partial_0 f(x).v$
  - **VJP, Vector-Jacobian-Product**, need to implement $w \mapsto w^T.\partial_0 f(x)$
  - NB: "backprop" is VJP
- Forward pass: use any solver to get $z^\star$ such that $z^\star = f_\theta(x, z^\star) \stackrel{\text{def}}{=} z(\theta, x)$
- Backward pass:
  - need to implement VJP for both θ and x, i.e.,
    - $w \mapsto w^T.\partial_0 z(\theta, x)$ for parameter updates
    - $w \mapsto w^T.\partial_1 z(\theta, x)$ to propagate gradient on the input (preceding layers)
  - spoiler alert (see next slide), the solution also involves a fixed point problem

Let's differentiate an implicit layer (VJP $w \mapsto w^T.\partial_1 z(\theta, x)$)

We consider the gradient w.r.t. x, and omit the fixed θ for conciseness, i.e. we want $w \mapsto VJP_0(z,(x),w^T) \stackrel{\text{def}}{=} w^T.\partial_0 z(x)$

We take the equilibrium equation $z^\star = f(x, z^\star)$, or $z(x) = f(x, z(x))$ and differentiate w.r.t. the input x

$$\frac{\partial}{\partial_x} z(x) = \frac{\partial}{\partial_x} f(x, z(x)) \tag{1}$$
$$\partial_0 z(x) = \partial_0 f(x, z(x)) + \partial_1 f(x, z(x)) \partial_0 z(x) \tag{2}$$
$$[I - \partial_1 f(x, z(x))] \partial_0 z(x) = \partial_0 f(x, z(x)) \tag{3}$$
$$\partial_0 z(x) = [I - \partial_1 f(x, z(x))]^{-1} \partial_0 f(x, z(x)) \tag{4}$$

for the VJP, we left-multiply by w^T then define a new variable u

$$w^T.\partial_0 z(x) = w^T [I - \partial_1 f(x, z(x))]^{-1} \partial_0 f(x, z(x)) \tag{5}$$
$$w^T.\partial_0 z(x) \stackrel{\text{def}}{=} u^T \partial_0 f(x, z(x)) = VJP_0(f, (x, z(x)), u^T) \tag{6}$$

the VJP is given by the framework, it only remains to find $u^T = w^T [I - \partial_1 f(x, z(x))]^{-1}$, that we can rewrite

$$u^T [I - \partial_1 f(x, z(x))] = w^T [I - \partial_1 f(x, z(x))]^{-1} [I - \partial_1 f(x, z(x))] \tag{7}$$
$$u^T - u^T \partial_1 f(x, z(x)) = w^T \tag{8}$$
$$u^T = w^T + u^T \partial_1 f(x, z(x)) \text{... a fixed point equation!} \tag{9}$$

(exactly the same with θ)

NB: No intermediate memory

- but strictly not faster
- discussion
  - size, number of parameters and expressive power

NB: Any solver can be used

- Iterative application of f_θ is a naive implementation (linear)
- Newton–Raphson's method[W] (quadratic, uses differentiation)
  - finding $z^\star$ such that $z^\star = f_\theta(x, z^\star)$
  - can be done by finding zeros of $z \mapsto f_\theta(x, z) - z$
- Anderson acceleration[W]
  - accelerates the convergence of the fixed-point sequence
  - (optimally) combine previous evaluations
  - links with GMRES[W]

Outline

- Quick review of Deep Networks and ResNETs
- Fixed point iteration layer, Deep Equilibrium (DEQ)
- Other Implicit Layers

Other Implicit Layers

Some stuff

Neural ODEs `odeint(f, x, t0, t1, θ)`

- when we care about intermediate values of t
- time reversible
- continuous normalizing flow

Differentiable optimization

- cvxpy as solver for implicitly specified layer
- differentiable thanks to cvxpylayers

Links and pointers

- Tutorial
  - http://implicit-layers-tutorial.org/ (companion website)
  - https://www.youtube.com/watch?v=MX1RJELWONc (video)
    - Neural ODEs, norm. flow
    - Neural SDE
    - DEQ vs Neural ODEs
    - DEQ or Neural ODEs?
    - Implicit function theorem etc
    - Coding it in jax
  - Results
    - https://youtu.be/MX1RJELWONc?t=3674 (WikiText-103)
    - https://youtu.be/MX1RJELWONc?t=3789 (image, multiscale)
- cvxpy layers https://github.com/cvxgrp/cvxpylayers

Discussion, Questions?