Deep Learning Systems: Algorithms and Implementation

Implicit Layers

Fall 2022

J. Zico Kolter (this time) and Tianqi Chen
Carnegie Mellon University

Outline

Complex functions as layers (operators) in deep networks

Differentiation of implicit layers

Examples of implicit layers

Outline

Complex functions as layers (operators) in deep networks

Differentiation of implicit layers

Examples of implicit layers

“Layers” in deep networks

The term “layer” is a overloaded in deep learning … we need to distinguish between two things, where we’ll use needle conventions:

- *Operators* define “atomic” operations in the compute graph, along with it’s explicit gradient pass: i.e. matrix multiplication, convolution, elementwise nonlinearities, etc
- *Modules* define collections of operators, where the backward pass is created implicitly via the construction of the computational graph

What is the advantage to defining functions as operators rather than modules?

Example: convolution

Recall from our previous lecture that convolution could be specified in terms of a number of matrix multiplication operations:

```python
def conv_matrix_mult(Z, weight):
    N,H,W,C_in = Z.shape
    K,_,_,C_out = weight.shape
    out = np.zeros((N,H-K+1,W-K+1,C_out))

    for i in range(K):
        for j in range(K):
            out += Z[:,i:i+H-K+1,j:j+W-K+1,:] @ weight[i,j]
    return out
```

What is advantage of making convolution an atomic operator?

- Possible to optimize forward pass in a modular fashion
- No need to keep all intermediate computational terms (e.g., im2col matrix) in memory in the compute graph

More complex layers?

What if we want to define a “layer” in a deep network that performs some more complex operation: solve an optimization problem, compute a fixed point, solve an ODE, etc?

We *could* simply implement the “solver” in our automatic differentiation tool itself (i.e., as a Module), allows us to embed it in any larger computational graph

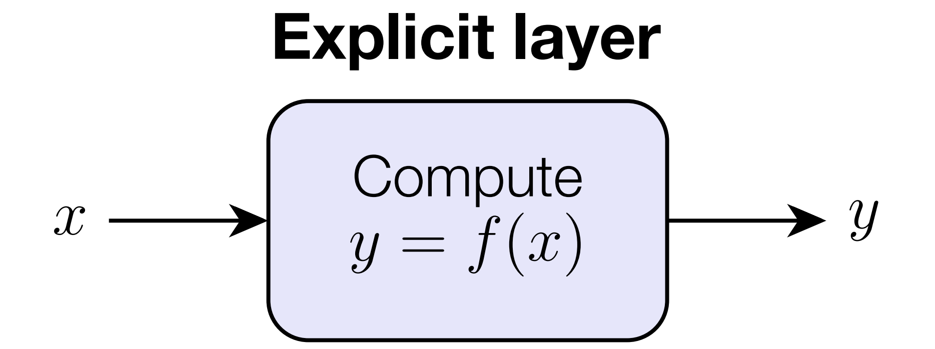
… but, as we will see, there are also some notable advantages to implementing these functions as atomic operations (i.e., as Operators)

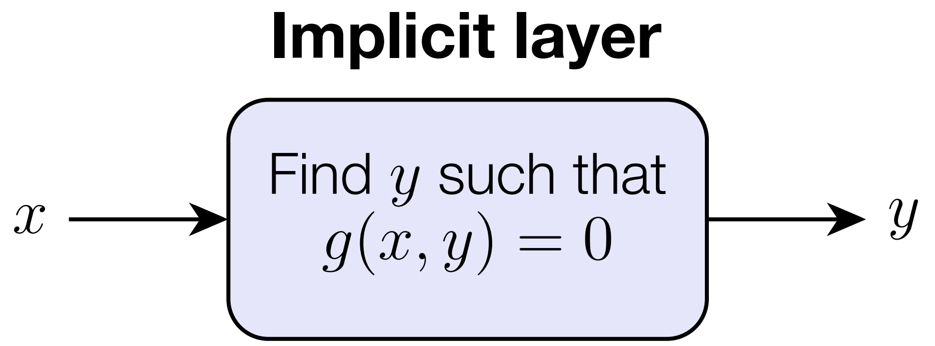
Explicit vs. Implicit layers

Virtually all layers (operators) we have used so far are ***explicit***, in that they themselves involve a “fixed” transformer between input and output

Implicit layers (operators), in contrast, just define a more complex operation, in terms of ***satisfying some joint condition of the input and output***

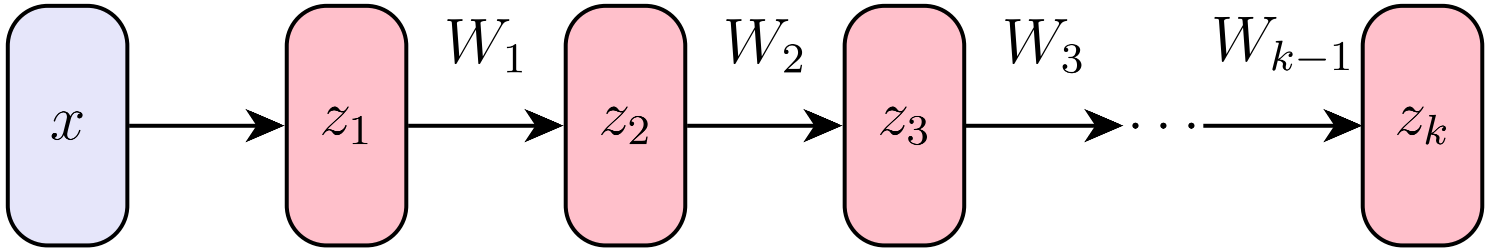
- Examples of differential equations, fixed point iteration, optimization solutions, etc, all can fit this mold





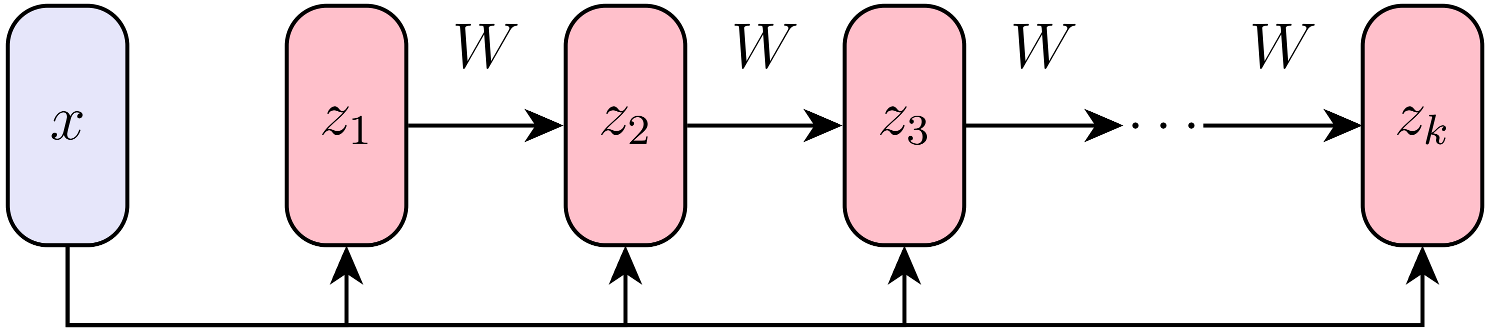
Motivating example: Deep Equilibrium Models

Consider a traditional MLP applied to an input x



$$z_{i+1} = \sigma(W_i z_i + b_i)$$

We now modify this network in two ways: by re-injecting the input at each step, and by applying the *same* weight matrix at each iteration (weight tying)



$$z_{i+1} = \sigma(W z_i + x)$$

Iterations of deep weight-tied models

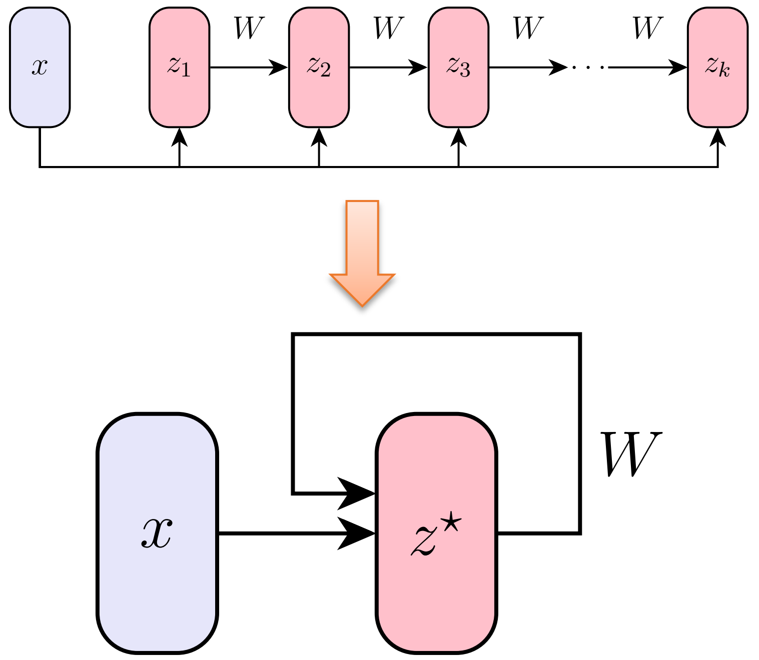
With a weight-tied model of this form, we are applying the *same* function repeatedly to the hidden units

$$z_{i+1} = \sigma(W z_i + x)$$

In many situations, we can design the network such that this iteration will converge to some *fixed point*, or *equilibrium point*

$$z^\star = \sigma(W z^\star + x)$$

Can define a layer that *directly* finds this equilibrium point (via fixed point iteration or other solution method)



Deep Equilibrium Models

In practice we want to find an equilibrium point of a more complex "cell", and use this as our *entire* model (plus one additional linear layer)

$$z^\star = \sigma(W z^\star + x) \quad \Longrightarrow \quad z^\star = \boxed{f}(z^\star, x, \theta)$$

Residual block, Transformer block, LSTM cell, etc ($\theta \equiv$ parameters of layers)

If we implement this operation as an Operator in needle, then it doesn't matter *how* we compute the fixed point $z^\star$ (e.g., using advanced nonlinear solvers), and we don't need to store intermediate states in compute graph

But in order to make it an Operator, we need to be able to differentiate through (i.e., implement compute_gradient for this operation)

Outline

Complex functions as layers (operators) in deep networks

Differentiation of implicit layers

Examples of implicit layers

Differentiating through an implicit layer

We define an implicit layer that finds the solution to the the nonlinear equation

$z^\star = f(z^\star, x)$, where we call solution $z^\star(x)$

How do we differentiate through this layer? Implicit differentiation…

$$z^\star(x) = f(z^\star(x), x)$$

$$\frac{\partial z^\star(x)}{\partial x} = \frac{\partial f(z^\star(x), x)}{\partial x}$$

$$\frac{\partial z^\star(x)}{\partial x} = \frac{\partial f(z^\star(x), x)}{\partial z^\star(x)} \frac{\partial z^\star(x)}{\partial x} + \frac{\partial f(z^\star(x), x)}{\partial x}$$

$$\frac{\partial z^\star(x)}{\partial x} = \left(I - \frac{\partial f(z^\star(x), x)}{\partial z^\star(x)}\right)^{-1} \frac{\partial f(z^\star(x), x)}{\partial x}$$

Integration within automatic differentiation

Recall that in our automatic differentiation framework, we really just wanted to be able to compute *product* of adjoint and the layer gradient

$$\bar{v}\frac{\partial z^\star(x)}{\partial x} = \underbrace{\bar{v}\left(I - \frac{\partial f(z^\star(x), x)}{\partial z^\star(x)}\right)^{-1}}_{\text{“Special” adjoint computation for implicit layer}} \frac{\partial f(z^\star(x), x)}{\partial x} = \underbrace{\bar{v}' \frac{\partial f(z^\star(x), x)}{\partial x}}_{\text{“Normal” AD adjoint}}$$

In other words, to implement backprop for implicit layer, we just need to add this “special” adjoint computation into the backward pass

In detail: computing the backward pass

Let’s look at the solution to this backward pass more closely

$$\bar{v}' = \bar{v}\left(I - \frac{\partial f(z^\star(x), x)}{\partial z^\star(x)}\right)^{-1}$$

$$\bar{v}' = \bar{v} + \bar{v}' \frac{\partial f(z^\star(x), x)}{\partial z^\star(x)}$$

Which looks a *lot* like the solution to the fixed point equation

$$z_{i+1} = \bar{v} + z_i \frac{\partial f(z^\star(x), x)}{\partial z^\star(x)}$$

Backward pass of fixed point operator can be implemented as fixed point operator of AD adjoint computation

Summary of the above

The final implementation is indeed slightly involved, so don’t worry if you didn’t follow all of the above

Main takeaway is that the backward pass of a fixed point operator involves solving *another* fixed point operation that in terms of various adjoint operators

All of this can be implemented within Operator class of needle: i.e., create FixedPointOp calls itself in its compute_gradient() function

Outline

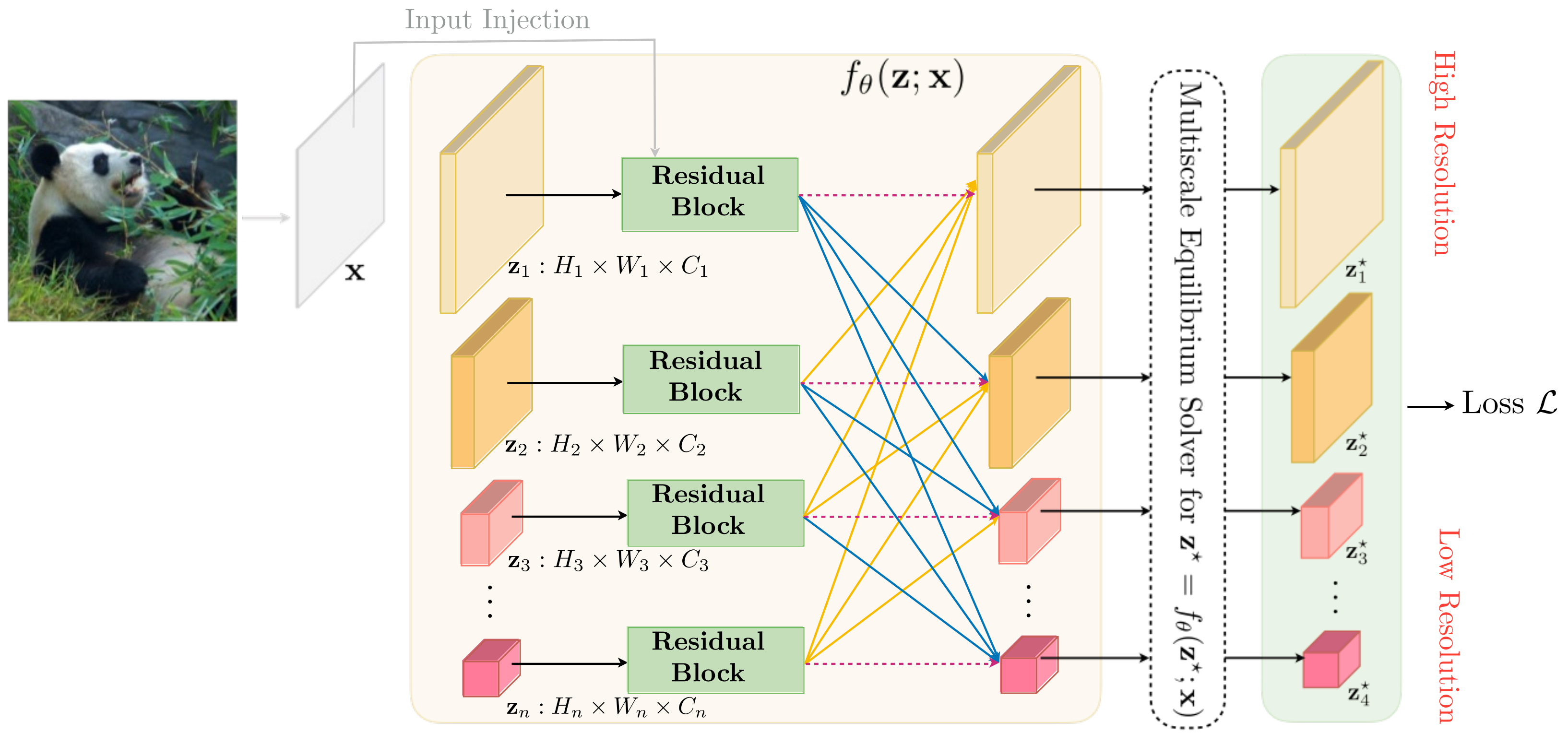
Complex functions as layers (operators) in deep networks

Differentiation of implicit layers

Examples of implicit layers

Multiscale deep equilibrium models

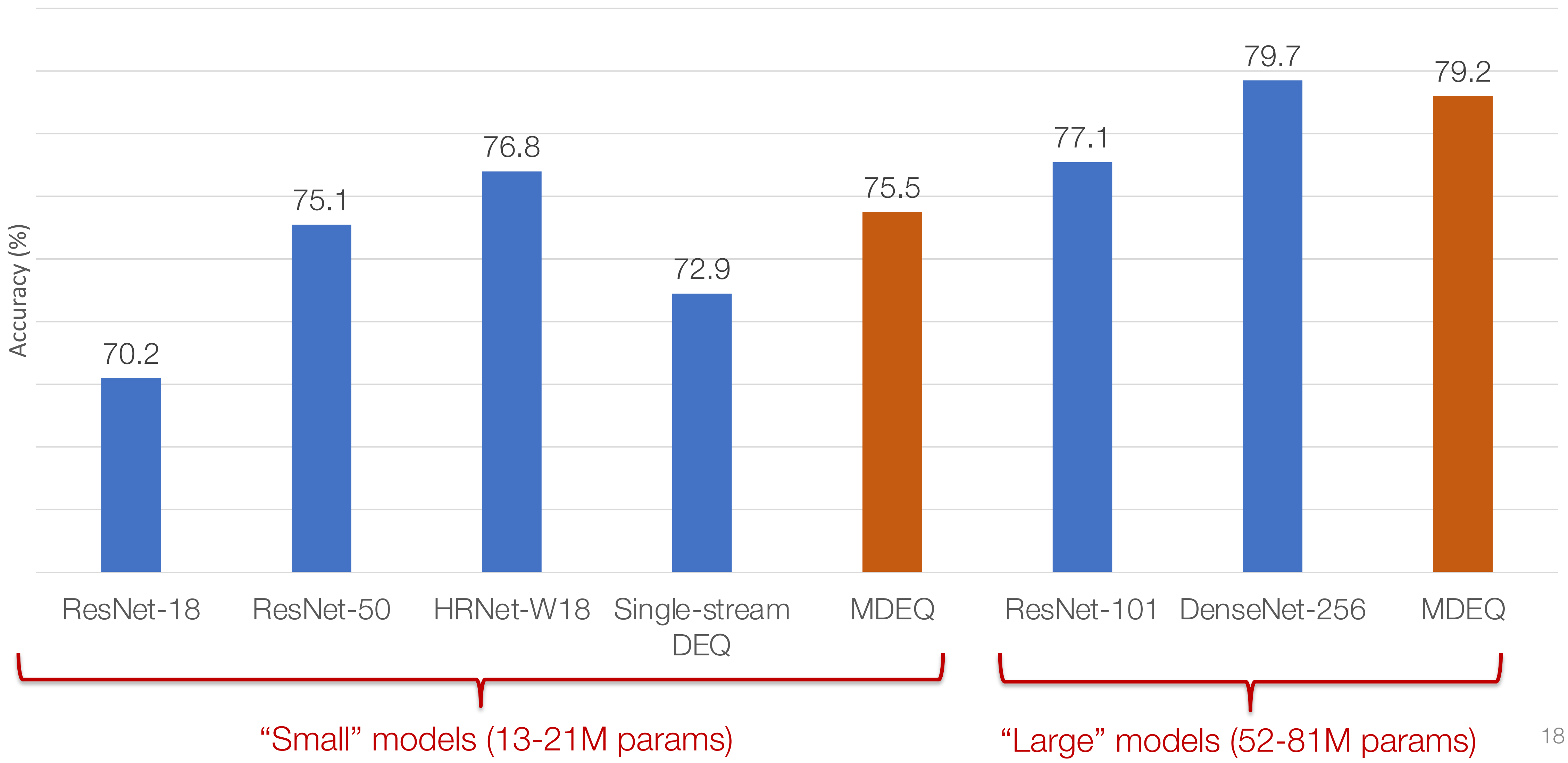
Key idea: maintain multiple spatial scales within the hidden unit of a DEQ model, and *simultaneously* find equilibrium point for all of them



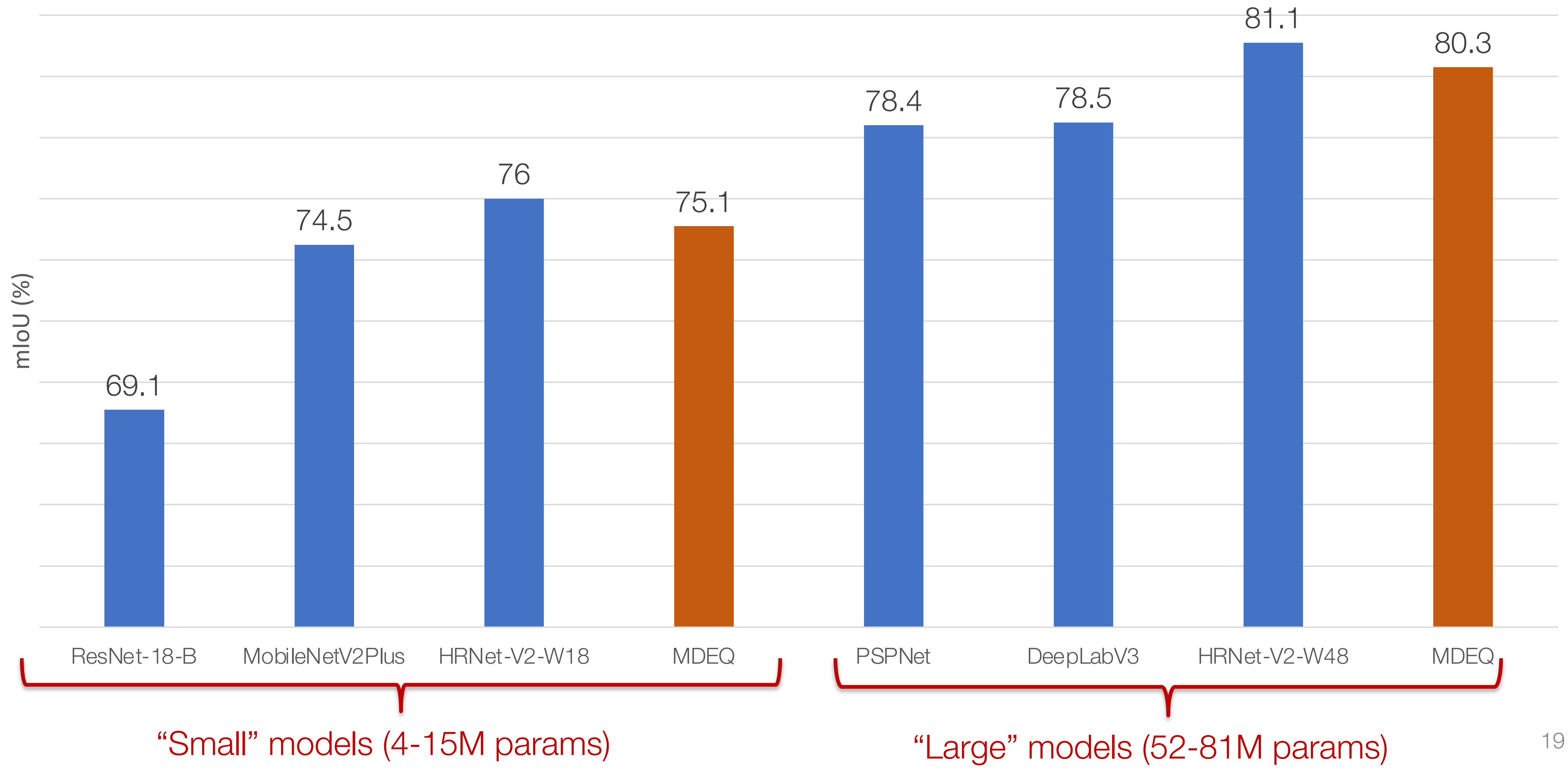
Input Injection
$\mathbf{x}$
$f_\theta(\mathbf{z};\mathbf{x})$
Residual Block
Residual Block
Residual Block
Residual Block
$\mathbf{z}_1 : H_1 \times W_1 \times C_1$
$\mathbf{z}_2 : H_2 \times W_2 \times C_2$
$\mathbf{z}_3 : H_3 \times W_3 \times C_3$
$\mathbf{z}_n : H_n \times W_n \times C_n$
Multiscale Equilibrium Solver for $\mathbf{z}^\star = f_\theta(\mathbf{z}^\star;\mathbf{x})$
$\mathbf{z}_1^\star$
$\mathbf{z}_2^\star$
$\mathbf{z}_3^\star$
$\mathbf{z}_4^\star$
High Resolution
Low Resolution
Loss $\mathcal{L}$

[Bai, Koltun, Kolter "Multiscale Deep Equilibrium Models", NeurIPS 2020]

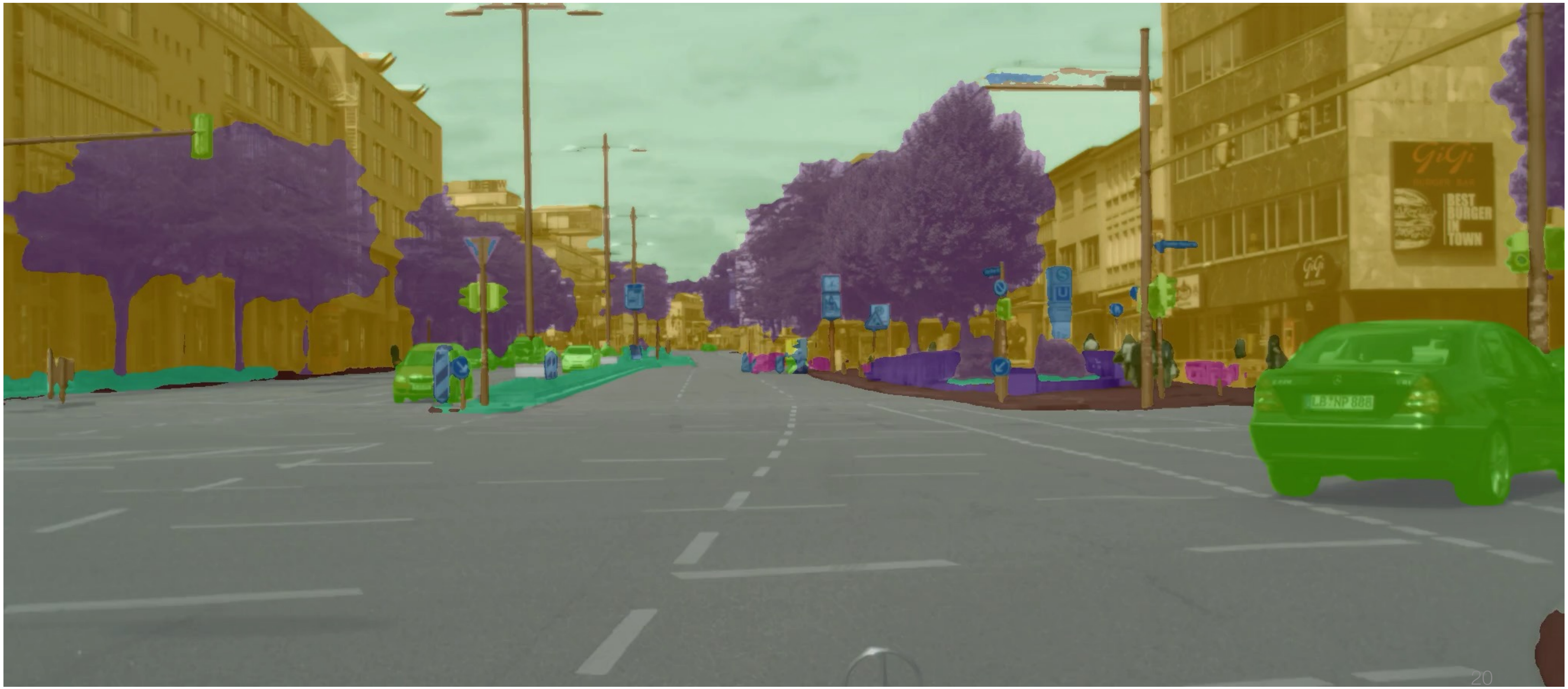
ImageNet Top-1 Accuracy



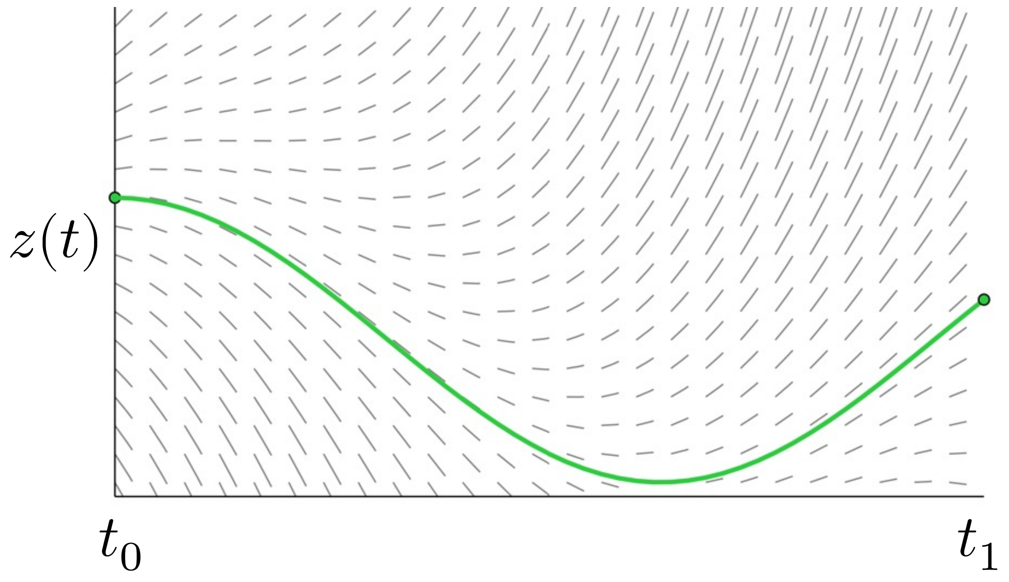
Citiscapes mIoU



Visualization of Segmentation



Neural ordinary differential equations



If a vector z follows dynamics f:

$$\frac{dz}{dt} = f(z(t), t, \theta)$$

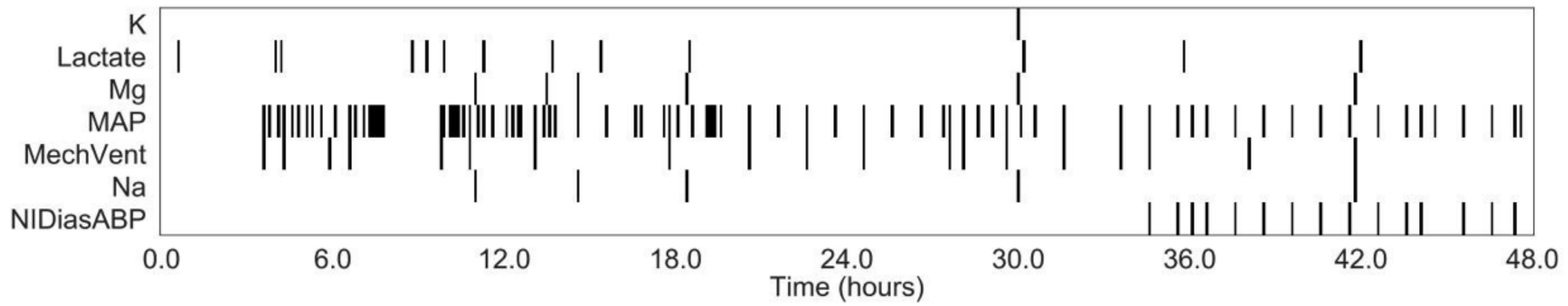
Can find $z(t_1)$ by starting at $z(t_0)$ and integrating until time t_1:

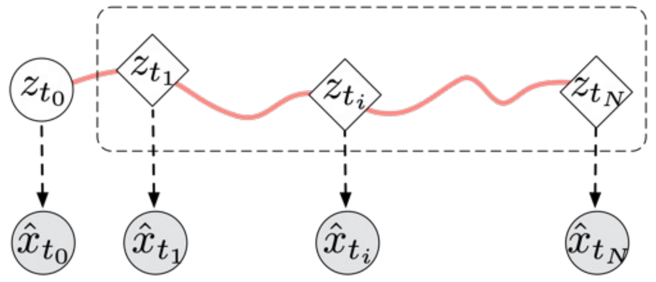
$$z(t_1) = z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta)dt$$

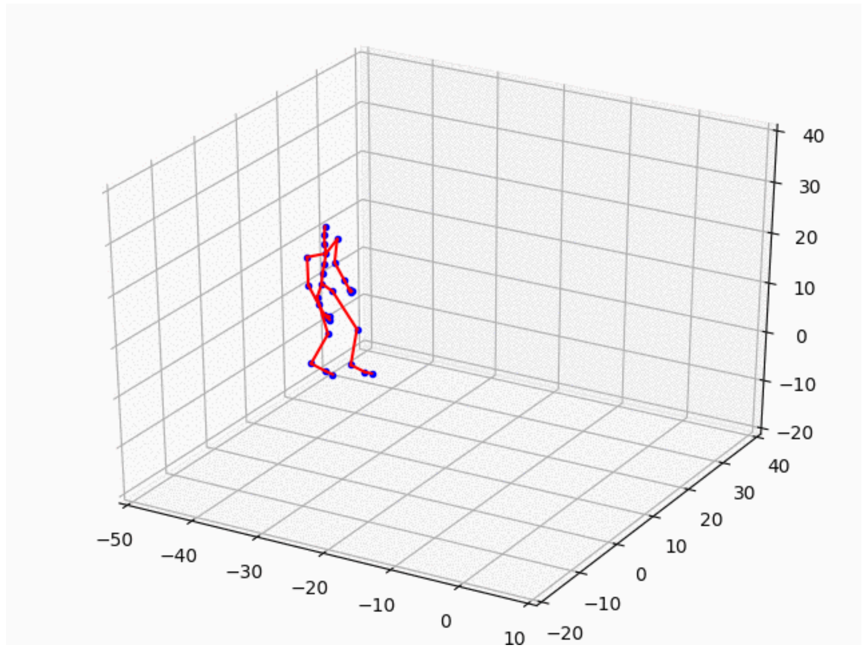
An implicit layer: $y = \mathrm{odeint}(f, x, t_0, t_1, \theta)$

Continuous-time time series Models

Neural ODEs can deal with data collected at irregular intervals natively







Latent ODEs for Irregularly-Sampled Time Series. Rubanova, Chen, Duvenaud (2020)
Neural Controlled Differential Equations for Irregular Time Series.
Kidger, Morrill, Foster, Lyons (2020)
GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series. de Brouwer, Simm, Arany, Moreau. (2020)

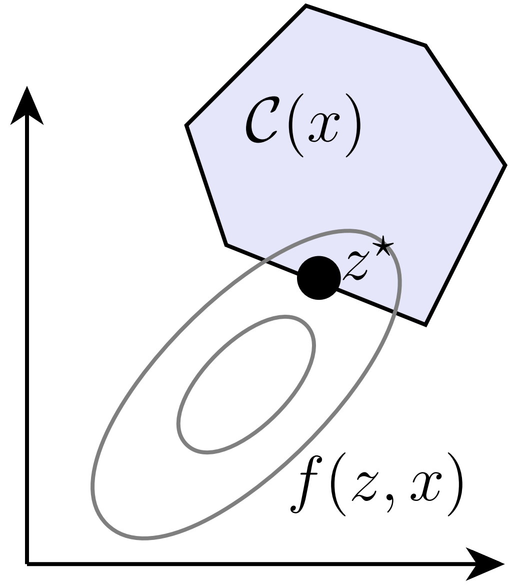
Differentiable optimization

DEQs and Neural ODEs both impose substantial structure on the nature of the layer, in order to gain substantial representational power

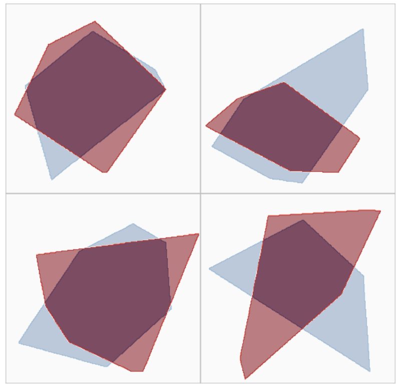
Other common strategy for imposing a different (but related) kind of structure is that of differentiable optimization

Layer of the form

$$z^\star = \operatorname*{argmin}_{z \in \mathcal{C}(x)} f(z, x)$$

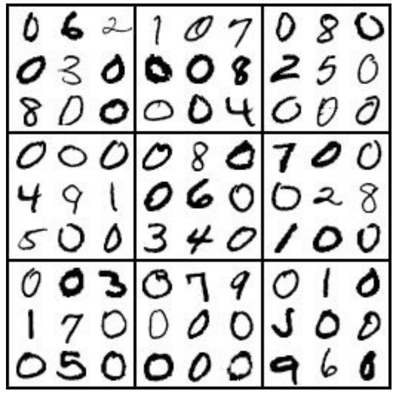


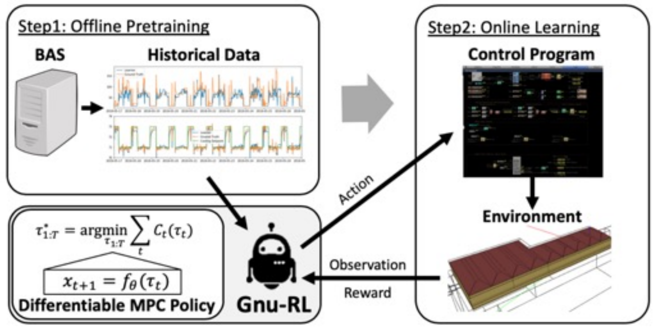
Some example applications



Learning a convex polytope from data [Amos and Kolter., 2018]

Solving Sudoku (w/ MNIST digits) using differentiable SDP solver [Wang et al., 2019]




Step1: Offline Pretraining
BAS
Historical Data
Step2: Online Learning
Control Program
Environment
Action
Observation
Reward
$\tau_{1:T}^* = \arg\min_{\tau_{1:T}} \sum_t C_t(\tau_t)$
$x_{t+1} = f_\theta(\tau_t)$
Differentiable MPC Policy
Gnu-RL

Controlling HVAC systems with differentiable MPC controllers [Chen et al., 2019]