Intro To PDE-Based Neural Networks

Scientific Computing Seminar

Derek Onken[1]

Simion Novikov[2] Eran Treister[2] Eldad Haber[3] Lars Ruthotto[1]

[1]Emory University

[2]Ben-Gurion University of the Negev

[3]University of British-Columbia

Apr 26, 2019

Overview

- **Continuous ResNet**
  - Discrete Neural Networks viewed in continuous framework
- **Discretize-Optimize in network layers**
  - Develop state and control layers (borrowing from optimal control)
  - Contrast with typical approach of 1-to-1 layers and weights (each layer has its own weights)

Goal: Decouple the time discretizations of the features and the weights. Does increasing parameters or layers improve convergence?

Motivation: Reduce network size (number of parameters) and complexity to simplify training and hyperparameter tuning

Use Case: Image Classification

Brief History

- **Modelling Nervous Systems via Temporal Propositional Functions**
  - Logical Calculus (McCulloch and Pitts, 1943)
- **Introduction of the Perceptron**
  - US Navy Press Conference (Rosenblatt, 1958)
- **However, single-layer perceptrons could not learn a simple XOR**
  - *Perceptrons* (Minsky and Papert, 1969)
- **Backpropogation**
  - Adjoint Method (more general) (Bliss, 1919)
  - *Applied Optimal Control* (Bryson and Ho, 1969)
  - Learning representation by back-propogating errors (Rumelhart, Hinton, and Williams, 1986)

A Neural Network is a Discrete Universal Approximator

Consider this approximating model as a function:

$$\mathbf{c} = f(\mathbf{y}, \boldsymbol{\theta})$$

where

$\mathbf{y} \in \mathbb{R}^{n_f}$ is an input item (e.g., an image of a dog)

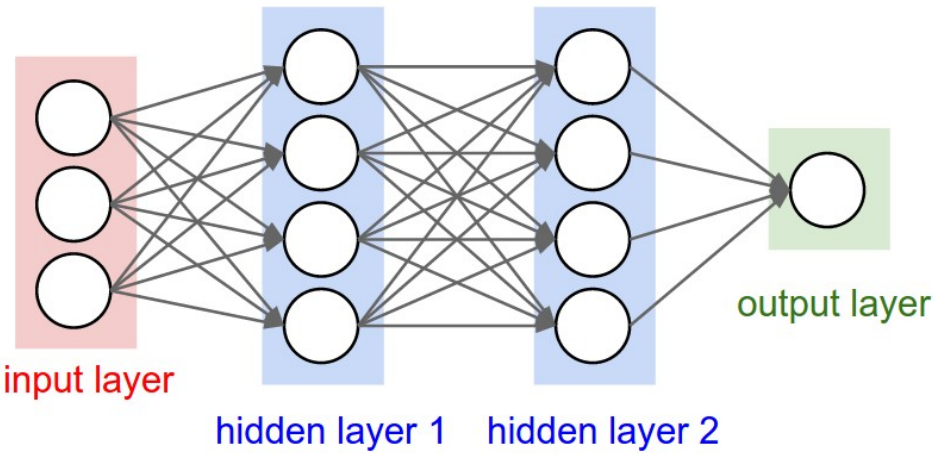
$\mathbf{c} \in \mathbb{R}^{n_c}$ is the corresponding output (e.g., a class/label "dog")

$\boldsymbol{\theta} \in \mathbb{R}^{n_p}$ are the parameters/weights of the model f

n_f - number of features

n_c - number of classes

n_p - number of parameters



Example: $f\left(\mathbf{y}, \begin{bmatrix} \mathrm{vec}(K_1) \\ \mathrm{vec}(K_2) \end{bmatrix}\right) = \sigma_2 \circ K_2 \circ \sigma_1 \circ K_1 \mathbf{y}$,

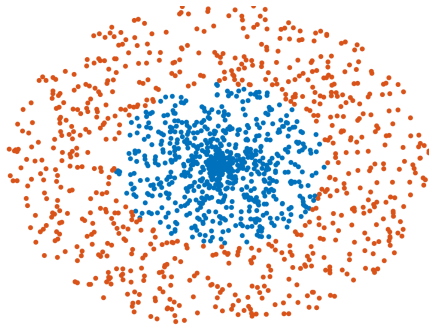
where $\sigma_1 = \sigma_2 = \tanh$

Vertically concatenate all n inputs and classes together:

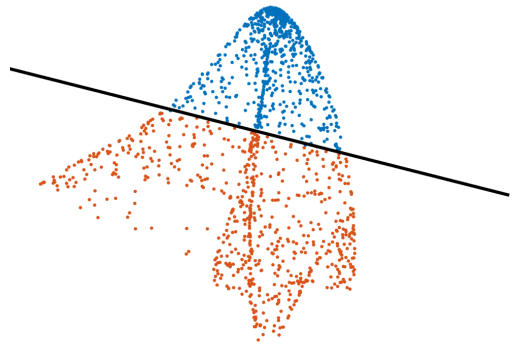
$$\left[\begin{array}{ccccc} 0 & & 255 & 255 & 255 \\ 0 & | & 255 & 128 & 255 \\ 255 & \mathbf{y} & 128 & 128 & 0 \\ 255 & | & 128 & 0 & 0 \\ 255 & & 0 & 0 & 0 \end{array}\right] \qquad \left[\begin{array}{ccccc} 0.1 & | & 0.4 & 0.2 & 1 \\ 0.8 & \mathbf{c} & 0.3 & 0.3 & 0 \\ 0.1 & | & 0.3 & 0.5 & 0 \end{array}\right]$$

$$Y \in \mathbb{R}^{n_f \times n} \qquad C \in \mathbb{R}^{n_c \times n}$$

Neural network as a projection onto a manifold:



original features Y



transformed features

Images from Ruthotto. *Deep Neural Networks motivated by PDEs*. 2018.

Single-Layer Example

n , # examples

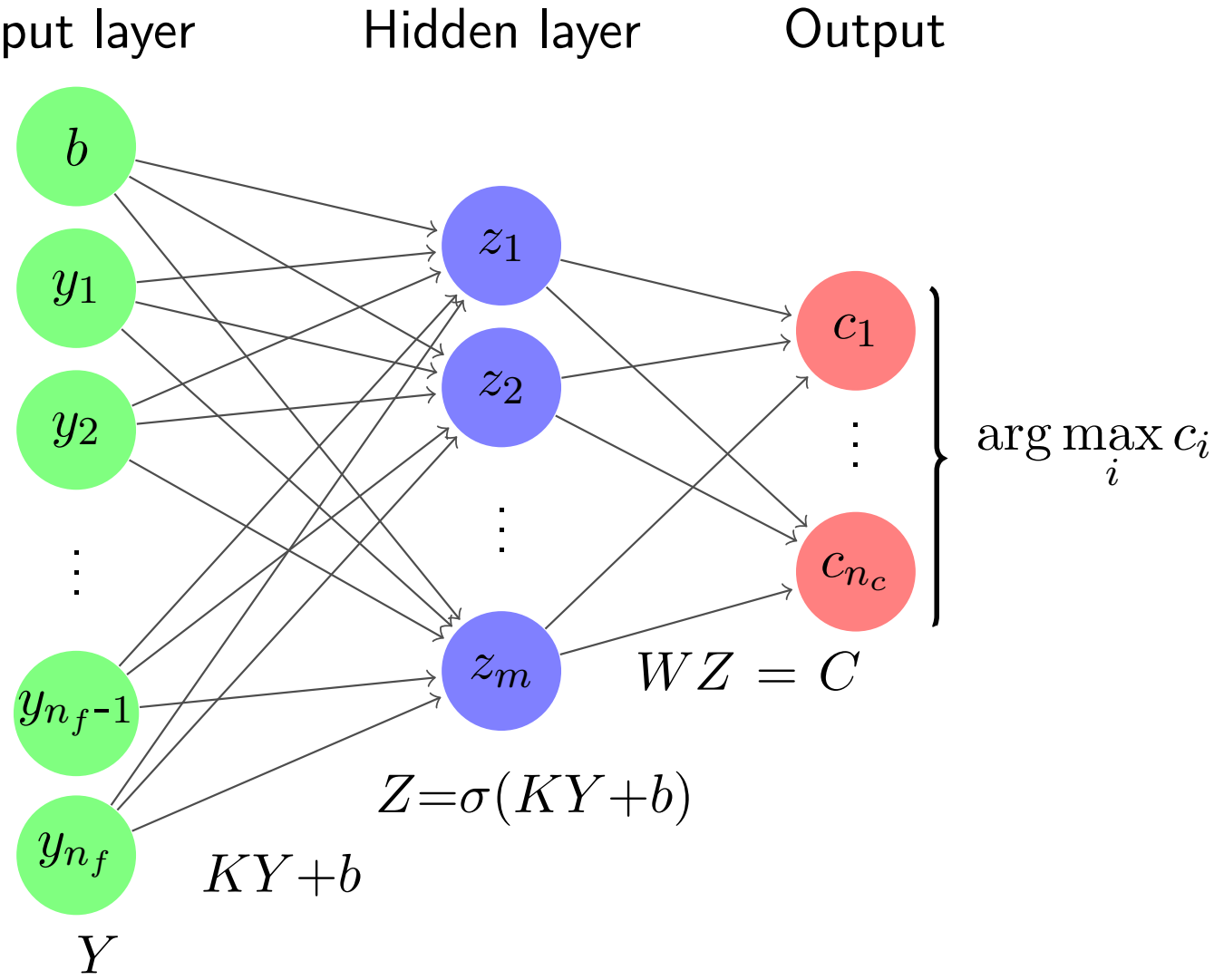
n_f, # features

n_c, # classes

Features
$Y \in \mathbb{R}^{n_f \times n}$

Weights ($\boldsymbol{\theta}$)
$K \in \mathbb{R}^{m \times n_f}$
$W \in \mathbb{R}^{n_c \times m}$
bias $b \in \mathbb{R}$

Class Probabilities
$C \in \mathbb{R}^{n_c \times n}$



Compute the Loss

For some loss / error function E, compute the difference between the predicted output $C = f(Y, \boldsymbol{\theta})$ and ground truth C_{gt},

$$\text{loss} = E(\, S(C)\,,\, C_{gt}\,)$$

where

- softmax activation $S(C) = \text{diag}\left(\frac{1}{\mathbf{e}_{n_c}\, e^{\,C}}\right) e^{\,C}$
- cross entropy $E(C, C_{gt}) = -\text{trace}(\, C_{gt}\, \log(C^{\top})\,)$

e^{C} and $\log(C)$ are element-wise and $\mathbf{e}_i \in \mathbb{R}^i$ contains all 1s.

Loss Details

Matrix versions

Softmax:

Scale the output values $\mathbf{c}$ so they represent percentages

$$S(c_i) = \frac{e^{\,c_i}}{\sum_{j=1}^{n_c} e^{\,c_j}} \qquad S(C) = \mathrm{diag}\left(\frac{1}{\mathbf{e}_{n_c}\, e^{C}}\right) e^{\,C}$$

Example:

$$\mathbf{c} = \begin{bmatrix} 0.6 \\ 1.0 \\ 2.0 \end{bmatrix}, \quad S(\mathbf{c}) \approx \begin{bmatrix} 0.153 \\ 0.228 \\ 0.619 \end{bmatrix}$$

Cross Entropy:

Comparing probability distributions

$$E(\mathbf{c}, \mathbf{c}_{gt}) = -\mathbf{c}_{gt} \log(\mathbf{c}^\top) \qquad E(C, C_{gt}) = -\mathrm{trace}(\, C_{gt}\, \log(C^\top)\,)$$

Optimization Problem

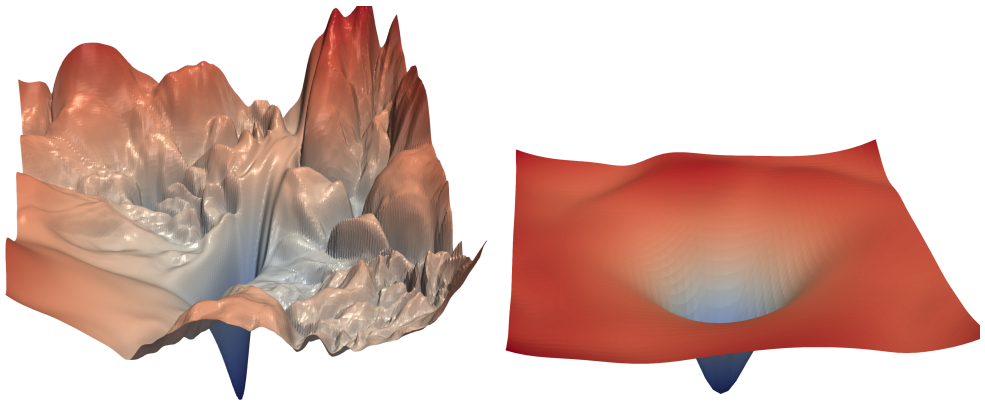
Given some training set Y with corresponding ground truth outputs C_{gt} and some "similar" testing set $\hat{Y}$ with its ground truth outputs $\hat{C}_{gt}$,

$$\min_{\boldsymbol{\theta}} \quad E(\, S(\, f(\hat{Y}, \boldsymbol{\theta})\,)\,, \hat{C}_{gt}\,)\,,$$

but only by using the training set to tune $\boldsymbol{\theta}$:

$$\arg\min_{\boldsymbol{\theta}} \quad E(\, S(\, f(Y, \boldsymbol{\theta})\,)\,, C_{gt}\,) + R(Y, \boldsymbol{\theta})$$

with some regularizer R.



(a) without skip connections

(b) with skip connections

Obstacles:

- high-dimensional
- non-convex
- not necessarily smooth
- f is stochastic in Y
- Y doesn't fit $\Rightarrow$ batches

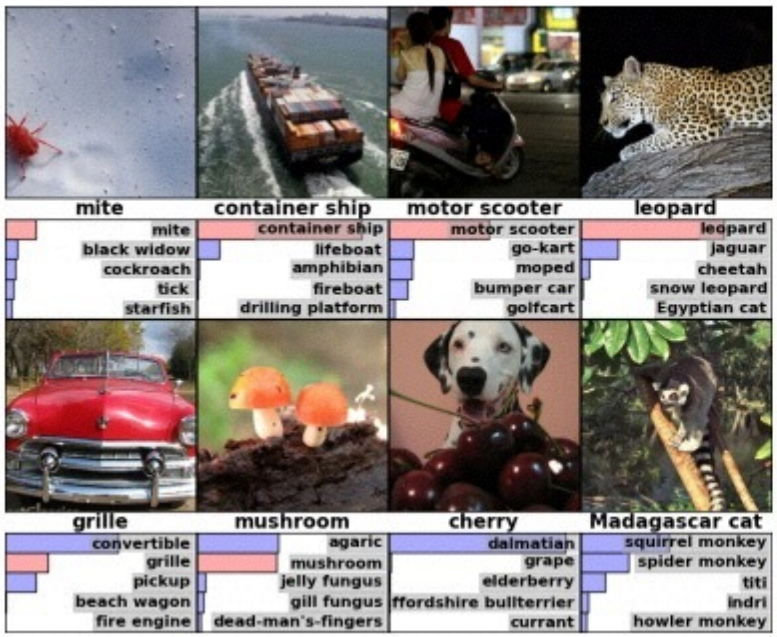
Result:

Stochastic Gradient Descent

Images from Li et al. "Visualizing the loss landscape of neural nets". 2018.

Image Classification

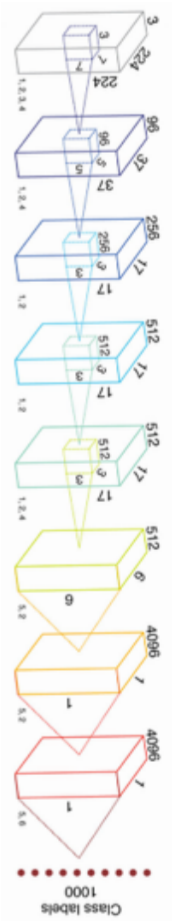
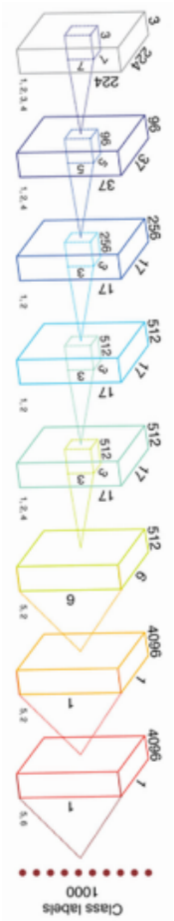
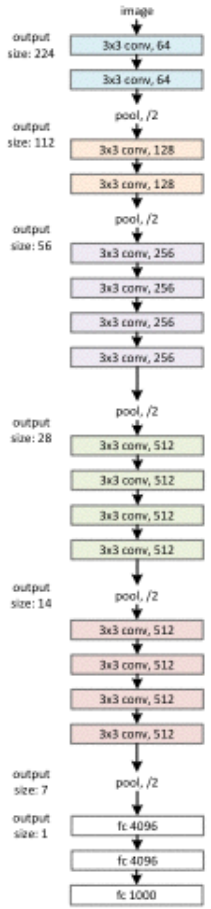
IMAGENET



1000 classes

Slide by Ilya Kuzovkin.

More Layers and "Deep Learning"

2012	2013	2014	2015
			?
8 layers	9 layers, 2x params	19 layers	∞ layers
15.31% error	11.74% error	7.41% error	0% error

Slide by Ilya Kuzovkin.

The Degradation Problem

Motivation for the Residual Neural Network (ResNet)[1]

"with the network depth increasing, accuracy gets saturated"

Not caused by overfitting:

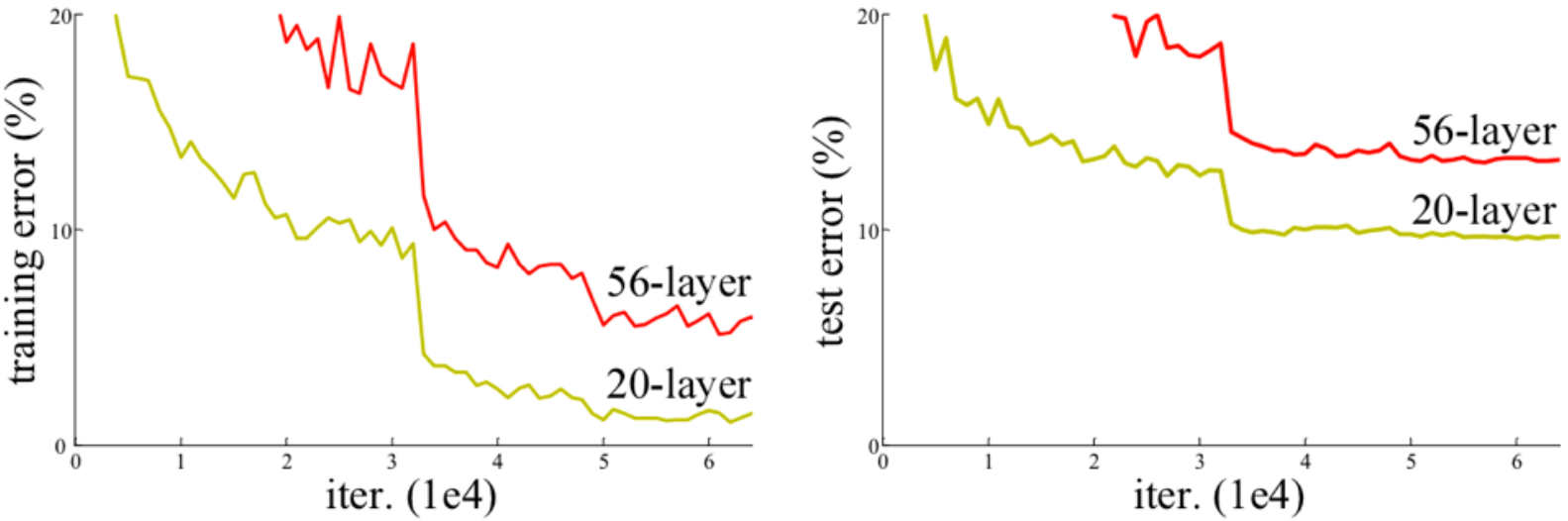


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network

Slide by Ilya Kuzovkin.

[1] He et al. "Deep residual learning for image recognition". 2016.

Looking at Residual

Add explicit **identity connections** and *"solvers may simply drive the **weights** of the multiple nonlinear layers **toward zero**"*

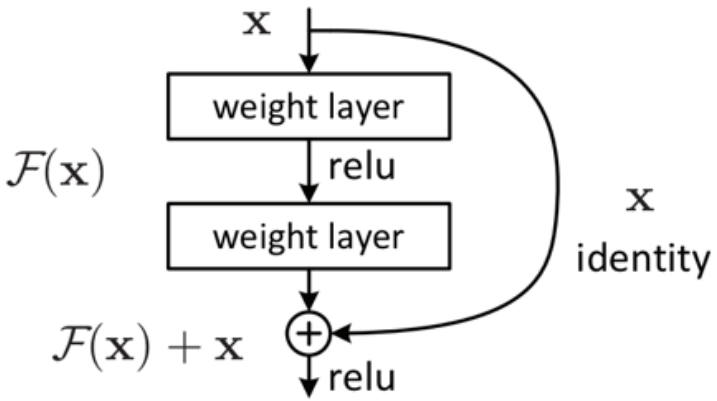


Figure 2. Residual learning: a building block.

$\mathcal{H}(\mathbf{x})$ is the true function we want to learn

Let's pretend we want to learn

$$\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$$

instead.

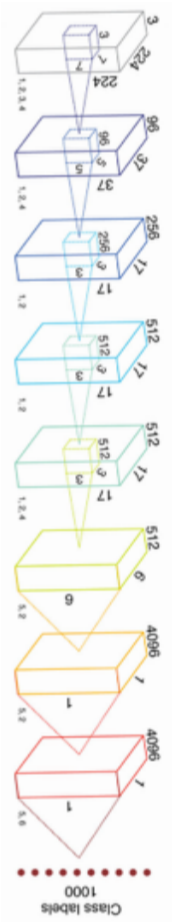
The original function is then

$$\mathcal{F}(\mathbf{x}) + \mathbf{x}$$

Slide by Ilya Kuzovkin.

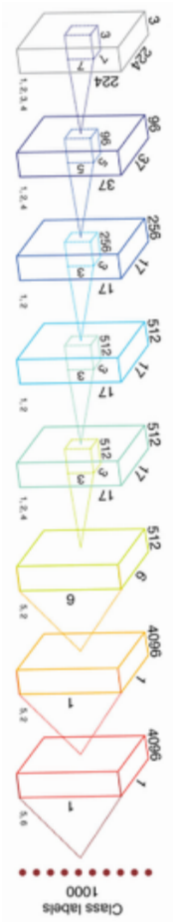
ResNet

2012



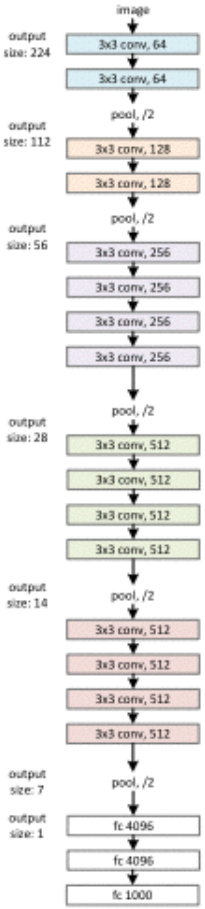
8 layers
15.31% error

2013



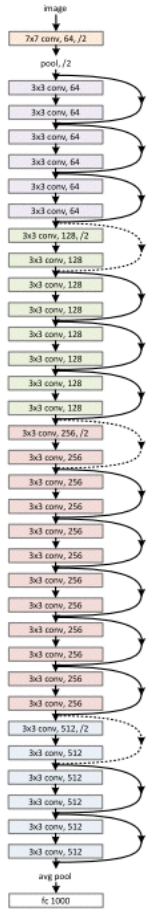
9 layers, 2x params
11.74% error

2014



19 layers
7.41% error

2015



152 layers
3.57% error

Slide by Ilya Kuzovkin.

No degradation & wins COCO and ImageNet awards

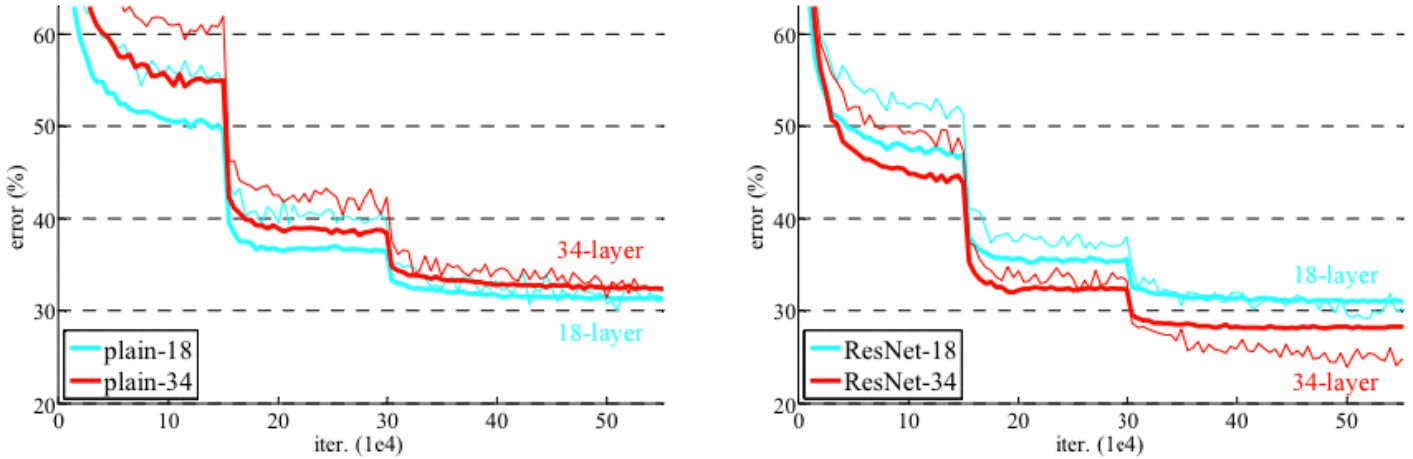


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Table 2. Top-1 error (%, 10-crop testing) on ImageNet validation. Here the ResNets have no extra parameter compared to their plain counterparts. Fig. 4 shows the training procedures.

- 34-layer ResNet has **lower training error**. This indicates that the **degradation problem is well addressed** and we manage to obtain accuracy **gains from increased depth**.
- 34-layer-ResNet **reduces the top-1 error** by 3.5%
- 18-layer ResNet **converges faster** and thus ResNet eases the optimization by providing faster convergence at the early stage.

Slide by Ilya Kuzovkin.

From Discrete to Continuous

These ResNets are just discrete forward Euler.[1,2]

View them continuously as the ordinary differential equation (ODE):

$$\partial_t \mathbf{u}(t) = f(\mathbf{u}(t), \boldsymbol{\theta}(t)), \quad \mathbf{u}(0) = \mathbf{y}.$$

for state variable (features) $\mathbf{u}$ and control variable (weights) $\boldsymbol{\theta}$ depending on some artificial time $t \in [0, T]$.

Now, we have ∞ layers!

We'll say t_i is a "layer" and $u(t_i)$ are the features output from the layer t_i.

[1] Weinan. "A Proposal on Machine Learning via Dynamical Systems". 2017.
[2] Haber and Ruthotto. "Stable Architectures for Deep Neural Networks". 2017.

Solving the ODE

$$\arg\min_{\boldsymbol{\theta}} \quad E(\, S(\, \partial_t \mathbf{u}(t\,; \boldsymbol{\theta}(t))\,)\,,\, C_{gt}\,) + R(Y, \boldsymbol{\theta}), \quad \mathbf{u}(0) = \mathbf{y}$$

We choose Discretize-Optimize

- Discretize $\mathbf{u}$ and $\boldsymbol{\theta}$. (ResNet uses same discretization for both).
- Solve the optimization problem.

This Discretize-Optimize camp includes ANODE.[3]

Other camp: Optimize-Discretize (includes Neural ODEs [4])

[3] Gholami, Keutzer, and Biros. "ANODE: Unconditionally Accurate Memory-Efficient Gradients for Neural ODEs". 2019.

[4] Chen et al. "Neural ordinary differential equations". 2018.

Generalized framework inspired by ResNet

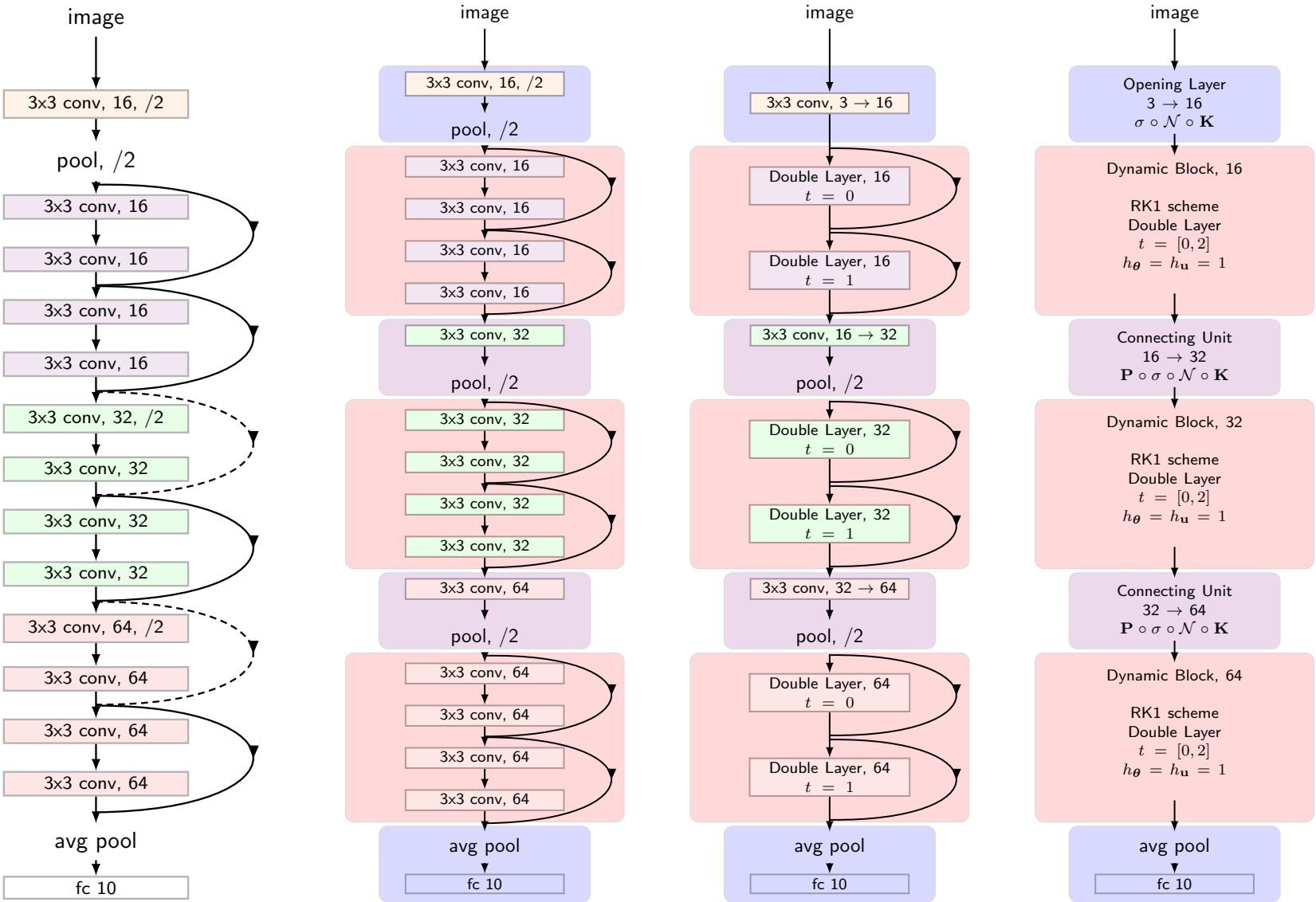


Figure: Different Representations for ResNet14

References I

Bliss, Go A (1919). "The use of adjoint systems in the problem of differential corrections for trajectories". In: *JUS Artillery* 51, pp. 296–311.

Bryson, Arthur E. and Yu-Chi Ho (1969). *Applied Optimal Control: Optimization, Estimation, and Control*. Waltham, MA: Blaisdell, pp. 148–176.

Chen, Tian Qi et al. (2018). "Neural ordinary differential equations". In: *Advances in Neural Information Processing Systems*, pp. 6571–6583.

Gholami, Amir, Kurt Keutzer, and George Biros (2019). "ANODE: Unconditionally Accurate Memory-Efficient Gradients for Neural ODEs". In: *arXiv preprint arXiv:1902.10298*.

Haber, Eldad and Lars Ruthotto (2017). "Stable Architectures for Deep Neural Networks". In: *Inverse Probl.* 34 (1).

He, Kaiming et al. (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.

References II

Li, Hao et al. (2018). "Visualizing the loss landscape of neural nets". In: *Advances in Neural Information Processing Systems*, pp. 6389–6399.

McCulloch, Warren S. and Walter H. Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133.

Minsky, Marvin and Seymour Papert (1969). *Perceptrons: An Introduction to Computational Geometry*. Cambridge, Massachusetts: MITPress.

Rosenblatt, Frank (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, p. 386.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). "Learning representations by back-propagating errors". In: *Nature* 323.9, pp. 533–536.

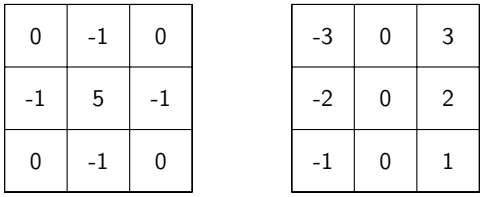
Ruthotto, Lars (2018). *Deep Neural Networks motivated by PDEs*. URL: `www.math.emory.edu/{~}lruthot/talks/2018-DeepLearning-handout.pdf`.

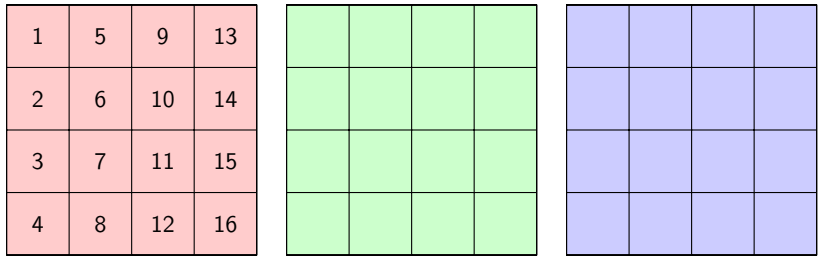
References III

Weinan, E. (2017). "A Proposal on Machine Learning via Dynamical Systems". In: *Comm. Math. Statist.* 5.1, pp. 1–11.

Convolution Formalism

Assume input image $\mathbf{y}$ is a 4×4 pixel RGB image, and convolutional operator K will convert those 3 channels to 2 channels (no padding).





$$K$$

$$\mathbf{y} = \begin{bmatrix} \mathbf{y}_R \\ \mathbf{y}_G \\ \mathbf{y}_B \end{bmatrix}$$

$$\left[\begin{array}{cccccccccccc|c|c} 0 & -1 & 0 & -1 & 5 & -1 & 0 & -1 & 0 & & & & & \\ & 0 & -1 & 0 & -1 & 5 & -1 & 0 & -1 & 0 & & & & \\ & & & 0 & -1 & 0 & -1 & 5 & -1 & 0 & -1 & 0 & K_{G1} & K_{B1} \\ & & & & 0 & -1 & 0 & -1 & 5 & -1 & 0 & -1 & & \\ \hline -3 & -2 & -1 & 0 & 0 & 0 & 3 & 2 & 1 & & & & & \\ & -3 & -2 & -1 & 0 & 0 & 0 & 3 & 2 & 1 & & & & \\ & & & -3 & -2 & -1 & 0 & 0 & 0 & 3 & 2 & 1 & K_{G2} & K_{B2} \end{array}\right] \left[\begin{array}{c} 1 \\ \vdots \\ 16 \\ \hline \mathbf{y}_G \\ \hline \mathbf{y}_B \end{array}\right]$$

$$= K\mathbf{y}$$