Intro To PDE-Based Neural Networks
Scientific Computing Seminar

Derek Onken\textsuperscript{1}  
Simion Novikov\textsuperscript{2}  
Eran Treister\textsuperscript{2}  
Eldad Haber\textsuperscript{3}  
Lars Ruthotto\textsuperscript{1}

\textsuperscript{1}Emory University  
\textsuperscript{2}Ben-Gurion University of the Negev  
\textsuperscript{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:

\[ c = f(y, \theta) \]

where

- \( y \in \mathbb{R}^{n_f} \) is an input item (e.g., an image of a dog)
- \( c \in \mathbb{R}^{n_c} \) is the corresponding output (e.g., a class/label "dog")
- \( \theta \in \mathbb{R}^{n_p} \) are the parameters/weights of the model \( f \)

- \( n_f \) - number of features
- \( n_c \) - number of classes
- \( n_p \) - number of parameters

Example:

\[
f \left( y, \begin{bmatrix} \text{vec}(K_1) \\ \text{vec}(K_2) \end{bmatrix} \right) = \sigma_2 \circ K_2 \circ \sigma_1 \circ K_1 y,
\]

where \( \sigma_1 = \sigma_2 = \tanh \)
Vertically concatenate all $n$ inputs and classes together:

\[
\begin{bmatrix}
0 & 255 & 255 & 255 \\
0 & 255 & 128 & 255 \\
255 & y & 128 & 128 & 0 \\
255 & 128 & 0 & 0 \\
255 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[
Y \in \mathbb{R}^{n_f \times n}
\]

\[
C \in \mathbb{R}^{n_c \times n}
\]

Neural network as a projection onto a manifold:

Images from Ruthotto. *Deep Neural Networks motivated by PDEs.* 2018.
Single-Layer Example

\[ Y \in \mathbb{R}^{n_f \times n} \]

Features

Weights (\( \theta \))

Weights \( K \) in \( \mathbb{R}^{m \times n_f} \)

Weights \( W \) in \( \mathbb{R}^{n_c \times m} \)

Bias \( b \) in \( \mathbb{R} \)

Class Probabilities

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

\[ WZ = C \]

\[ Z = \sigma(KY + b) \]

\[ \arg \max_i c_i \]

Input layer

Hidden layer

Output

\( n \), # examples

\( n_f \), # features

\( n_c \), # classes
Compute the Loss

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

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

where

- softmax activation \( S(C) = \text{diag} \left( \frac{1}{\mathbf{e}_{n_c}^\top e^C} \right) e^C \)
- 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

**Softmax:**
Scale the output values $c$ so they represent percentages

$$S(c_i) = \frac{e^{c_i}}{\sum_{j=1}^{n_c} e^{c_j}}$$

Example:

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

**Cross Entropy:**
Comparing probability distributions

$$E(c, c_{gt}) = -c_{gt} \log(c^\top)$$

$$E(C, C_{gt}) = -\text{trace}(C_{gt} \log(C^\top))$$
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 $\hat{C}_{gt}$,

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

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

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

with some regularizer $R$.

Obstacles:

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

Result:

Stochastic Gradient Descent

Image Classification

ImageNet

1000 classes

Slide by Ilya Kuzovkin.
More Layers and "Deep Learning"

2012
8 layers
15.31% error

2013
9 layers, 2x params
11.74% error

2014
19 layers
7.41% error

2015
∞ layers
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.

Looking at Residual

Add explicit **identity connections** and “**solvers may simply drive the weights of the multiple nonlinear layers toward zero**” is the true function we want to learn. Let’s pretend we want to learn instead.

The original function is then

\[ \mathcal{F}(x) + x \]

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

Let’s pretend we want to learn
\[ \mathcal{F}(x) := \mathcal{H}(x) - x \]

instead.

Figure 2. Residual learning: a building block.

Slide by Ilya Kuzovkin.
**ResNet**

<table>
<thead>
<tr>
<th>Year</th>
<th>Layers</th>
<th>Parameters</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012</td>
<td>8</td>
<td></td>
<td>15.31%</td>
</tr>
<tr>
<td>2013</td>
<td>9</td>
<td>2x</td>
<td>11.74%</td>
</tr>
<tr>
<td>2014</td>
<td>19</td>
<td></td>
<td>7.41%</td>
</tr>
<tr>
<td>2015</td>
<td>152</td>
<td></td>
<td>3.57%</td>
</tr>
</tbody>
</table>

Slide by Ilya Kuzovkin.
No degradation & wins COCO and ImageNet awards

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.

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.

<table>
<thead>
<tr>
<th></th>
<th>plain</th>
<th>ResNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>18 layers</td>
<td>27.94</td>
<td>27.88</td>
</tr>
<tr>
<td>34 layers</td>
<td>28.54</td>
<td>25.03</td>
</tr>
</tbody>
</table>

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

Slide by Ilya Kuzovkin.
From Discrete to Continuous

These ResNets are just discrete forward Euler.\textsuperscript{1,2}

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

$$\partial_t u(t) = f(u(t), \theta(t)), \quad u(0) = y.$$ 

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

Now, we have $\infty$ layers!

We’ll say $t_i$ is a “layer” and $u(t_i)$ are the features output from the layer $t_i$.

\textsuperscript{2}Haber and Ruthotto. “Stable Architectures for Deep Neural Networks”. 2017.
Solving the ODE

\[ \arg \min_{\theta} \quad E\left( S\left( \partial_t u(t ; \theta(t)) \right) \right), \ C_{gt} \right) + R(Y, \theta), \quad u(0) = y \]

**We choose Discretize-Optimize**

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

This Discretize-Optimize camp includes ANODE.\(^3\)

Other camp: Optimize-Discretize (includes Neural ODEs \(^4\))


Generalized framework inspired by ResNet

**Figure:** Different Representations for ResNet14
References I


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.


Convolution Formalism

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

$$K = \begin{bmatrix}
0 & -1 & 0 \\
-1 & 5 & -1 \\
0 & -1 & 0
\end{bmatrix}$$

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

$$K = \begin{bmatrix}
1 & 5 & 9 & 13 \\
2 & 6 & 10 & 14 \\
3 & 7 & 11 & 15 \\
4 & 8 & 12 & 16
\end{bmatrix}$$

$$K \mathbf{y} = \mathbf{y}$$