A Novel Framework and According Training Method Beyond Back-propagation

Jiebo Song, Tsinghua University
songjiebo2012@163.com
songjb@iiisct.com

April 19, 2019

1. Introduction & Motivation
   - Problems and limitations of back-propagation
   - A novel framework and according training method
2. Details in HDNN
   - A simple starting point based on FCN
   - Novel framework and training method
   - Non-linear HDNN architectures
3. Conclusion & Future Work
   - Connection and difference between HDNN and DNN
   - To be continued...

Problems and Limitations of Back-propagation

- Long term dependency: vanishing gradient
- Local minimums / saddle points
- Need a large number of training data

A Novel Framework and According Training Method

- Key point:
  gradually add bases and use linear equations techniques to approximate the target function.
- Based on:
  1. universal approximation theorem
  2. gradually adding bases
  3. solvers for linear equations
- Benefits:
  1. layer-wise training, no need of gradient
  2. global optimal in each layer
  3. be able to predict in each layer
  4. need much less data

A Simple Starting Point Based on FCN

- Convolution approximation[1]:

$$s * y \approx a_0 y + a_1 \frac{dy}{dx} + a_2 \frac{d^2 y}{dx^2}. \tag{1}$$

$$[s_1 \; s_2 \; s_3] * y = \frac{\alpha_1}{4}[1 \; 2 \; 1] * y + \frac{\alpha_2}{2\triangle}[-1 \; 0 \; 1] * y + \frac{\alpha_3}{\triangle^2}[-1 \; 2 \; -1] * y. \tag{2}$$

transformation

$$\begin{pmatrix} s_1 \\ s_2 \\ s_3 \end{pmatrix} = \begin{pmatrix} \frac{1}{4} & -\frac{1}{2\triangle} & -\frac{1}{2\triangle^2} \\ \frac{1}{2} & 0 & \frac{2}{\triangle^2} \\ \frac{1}{4} & \frac{1}{2\triangle} & -\frac{1}{\triangle^2} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix}.$$

$$\lim_{\triangle \to 0} \frac{1}{4}(y_{j-1} + 2y_j + y_{j+1}) = y(x_j),$$

$$\lim_{\triangle \to 0} \frac{1}{2\triangle}(-y_{j-1} + y_{j+1}) = \frac{dy}{dx}(x_j),$$

$$\lim_{\triangle \to 0} \frac{1}{\triangle^2}(y_{j-1} - 2y_j + y_{j+1}) = \frac{d^2 y}{dx^2}(x_j).$$

[1] Haber, Eldad, et al. "Learning Across Scales—Multiscale Methods for Convolution Neural Networks." Thirty-Second AAAI Conference on Artificial Intelligence. 2018.

A Simple Starting Point Based on FCN

- Convolution approximation:

$$s * y \approx a_0 y + a_1 \frac{dy}{dx} + a_2 \frac{d^2 y}{dx^2}. \tag{3}$$

- Fully convolutional network (FCN):

$$F^n(y) = w_n * \ldots * (w_2 * (w_1 * y)). \tag{4}$$

- Our model:

$$f^n(y) = s_n * \ldots * (s_2 * (s_1 * y)). \tag{5}$$

Given input data y and target Y, we have to find optimal $s_i (i = 1, \ldots, n)$ s.t. $f^n(y) \to Y$.

A Simple Starting Point Based on FCN

- 1-layer:

$$s_1 * y = a_0 y + a_1 \frac{dy}{dx} + a_2 \frac{d^2 y}{dx^2} = \begin{pmatrix} y & \frac{dy}{dx} & \frac{d^2 y}{dx^2} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}, \quad (6)$$

- 2-layer:

$$\begin{aligned} & s_2 * (s_1 * y) \\ &= b_0 (a_0 y + a_1 \frac{dy}{dx} + a_2 \frac{d^2 y}{dx^2}) + b_1 (a_0 \frac{dy}{dx} + a_1 \frac{d^2 y}{dx^2} + a_2 \frac{d^3 y}{dx^3}) \\ & \quad + b_2 (a_0 \frac{d^2 y}{dx^2} + a_1 \frac{d^3 y}{dx^3} + a_2 \frac{d^4 y}{dx^4}) \\ &= \begin{pmatrix} y & \frac{dy}{dx} & \frac{d^2 y}{dx^2} & \frac{d^3 y}{dx^3} & \frac{d^4 y}{dx^4} \end{pmatrix} \begin{pmatrix} a_0 & 0 & 0 \\ a_1 & a_0 & 0 \\ a_2 & a_1 & a_0 \\ 0 & a_2 & a_1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}. \end{aligned}$$

A Simple Starting Point Based on FCN

- n-layer:

$$f^{(n+1)}(y) = s_{n+1} * f^{n}(y)$$

$$= D_0(d_0 y + \sum_{k=1}^{n+1} d_k \frac{d^k y}{dx^k}) + D_1(d_0 \frac{dy}{dx} + \sum_{k=1}^{n+1} d_k \frac{d^{k+1} y}{dx^{k+1}}) + D_2(d_0 \frac{d^2 y}{dx^2} + \sum_{k=1}^{n+1} d_k \frac{d^{k+2} y}{dx^{k+2}})$$

$$= \begin{pmatrix} y & \frac{dy}{dx} & \frac{d^2 y}{dx^2} & \cdots & \frac{d^{n+1} y}{dx^{n+1}} & \frac{d^{n+2} y}{dx^{n+2}} & \frac{d^{n+3} y}{dx^{n+3}} \end{pmatrix} \begin{pmatrix} d_0 & 0 & 0 & 0 \\ d_0 & d_1 & 0 & 0 \\ d_0 & d_1 & d_2 & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & d_{n-2} & d_{n-1} & d_n \\ 0 & 0 & d_{n-1} & d_n \\ 0 & 0 & 0 & d_n \end{pmatrix} \begin{pmatrix} D_0 \\ D_1 \\ D_2 \end{pmatrix}. \tag{8}$$

Novel Framework: High-order Differential Neural Network

- High-order Differential Neural Network (HDNN) framework:

$$Y^0 = y, \tag{9}$$

$$Y^n = s_n * Y^{n-1}, \tag{10}$$

where we use the following convolutional approximation:

$$s * y \approx a_0 y + a_1 \frac{dy}{dx} + a_2 \frac{d^2 y}{dx^2}. \tag{11}$$

And we use a new according Training Method!

New Training Method

- 1-layer:

$$s_1 * y = a_0 y + a_1 \frac{dy}{dx} + a_2 \frac{d^2y}{dx^2} = \begin{pmatrix} y & \frac{dy}{dx} & \frac{d^2y}{dx^2} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}. \tag{12}$$

Let $A = (y \ \frac{dy}{dx} \ \frac{d^2y}{dx^2}), \alpha = (a_0, a_1, a_2)^T$, we have to solve

$$\min_{\alpha} ||A\alpha - Y||^2. \tag{13}$$

New Training Method

- 2-layer: let $Y^1 = s_1 * y$, then

$$f^2(y) = b_0 Y^1 + b_1 \frac{dY^1}{dx} + b_2 \frac{d^2 Y^1}{dx^2} = \begin{pmatrix} Y^1 & \frac{dY^1}{dx} & \frac{d^2 Y^1}{dx^2} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}. \quad (14)$$

Let $A_2 = (Y^1 \; \frac{dY^1}{dx} \; \frac{d^2 Y^1}{dx^2}), \alpha_2 = (b_0, b_1, b_2)^T$, we have to solve

$$\min_{\alpha_2} ||A_2 \alpha_2 - Y||^2. \quad (15)$$

New Training Method

- n-layer: let $Y^{n-1} = s_{n-1} * Y^{n-2}$, then

$$f^n(y) = \begin{pmatrix} Y^{n-1} & \frac{dY^{n-1}}{dx} & \frac{d^2Y^{n-1}}{dx^2} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}. \tag{16}$$

Let $A_n = (Y^{n-1}\ \frac{dY^{n-1}}{dx}\ \frac{d^2Y^{n-1}}{dx^2}), \alpha_n = (D_0, D_1, D_2)^T$, we have to solve

$$\min_{\alpha_n} ||A_n\alpha_n - Y||^2. \tag{17}$$

New Training Method

- Training algorithm
  1. caculate $Y^n = A_{n-1}\alpha_{n-1}$, where $Y^0 = y$.
  2. create $A_n = D_0 Y^n + D_1 \frac{dY^n}{dx} + D_2 \frac{d^2 Y^n}{dx^2}$.
  3. solve a_n, s.t. $A_n\alpha_n \to Y$.

HDNN Experiments

- Convolutional approximation: $Y = s_5 * \cdots * (s_2 * (s_1 * y))$.

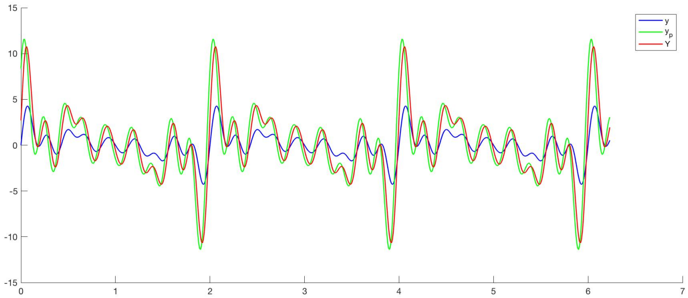
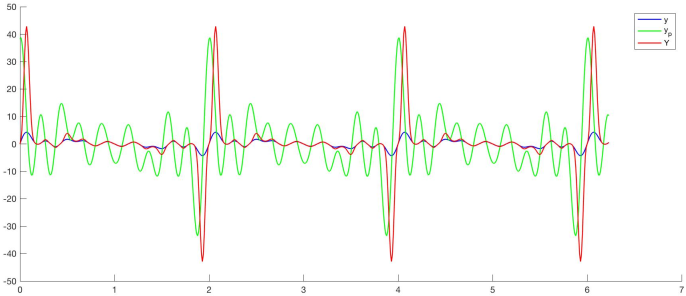


Figure: Single layer

HDNN Experiments

- Polynomial approximation: $Y = 0.5y + 0.5y^3$.



- Conclusion: good in case 1, bad in case 2.

We have to add non-linear item!

Non-linear HDNN: Architecture 1

- New operator

$$s_n \otimes f(y) = s_n * f(y) + d_n y^{n+1}, \tag{18}$$

- Architecture 1

$$Y^n = s_n \otimes Y^{n-1}, \tag{19}$$

where $Y^0 = y$.

Non-HDNN Experiments: Architecture 1

- Polynomial approximation: $Y = -0.5y + 0.5y^2 + 2.5y^3$.

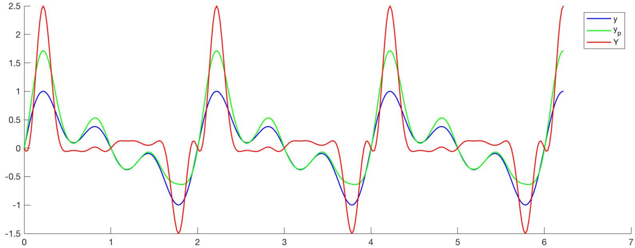


Figure: Single layer

Non-HDNN Experiments: Architecture 1

- Polynomial approximation: $Y = -0.5y + 0.5y^2 + 2.5y^3$.

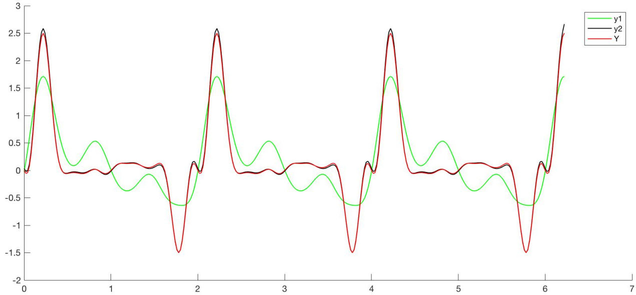


Figure: Two-layer

Good in lower order, but bad in higher order! such as $Y = 0.5y + 2y^7$.

Non-linear HDNN: DNA-Nets

- New operator

$$c \circ g(y) = c_0 g(y) + c_1 g(y) \cdot y + c_2 g(y) \cdot y^2, \tag{20}$$

- DNA-Nets architecture

$$Y^n = F^n(y) + G^n(y), \tag{21}$$

where,

$$F^n(y) = s_n * F^{n-1}(y), \tag{22}$$

$$G^n(y) = c_n \circ G^{n-1}(y), \tag{23}$$

and $F^0 = y, G^0 = y^2$.

Non-HDNN Experiments: DNA-Nets

- Polynomial approximation: $Y = -0.5y + 0.5y^2 + 2.5y^7$.

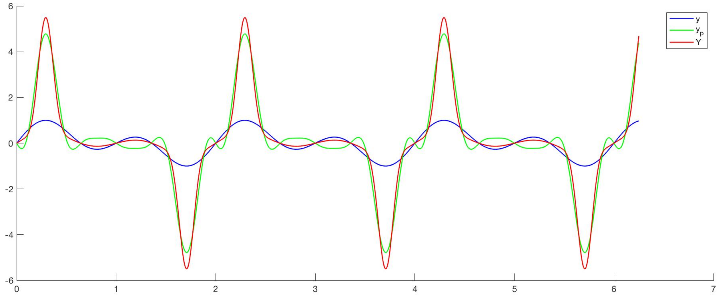


Figure: Single layer

Non-HDNN Experiments: DNA-Nets

- Polynomial approximation: $Y = -0.5y + 0.5y^2 + 2.5y^7$.

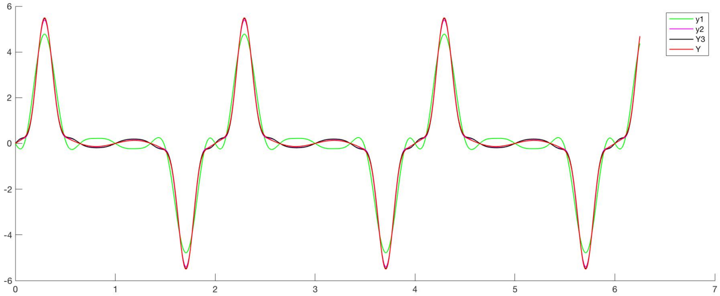
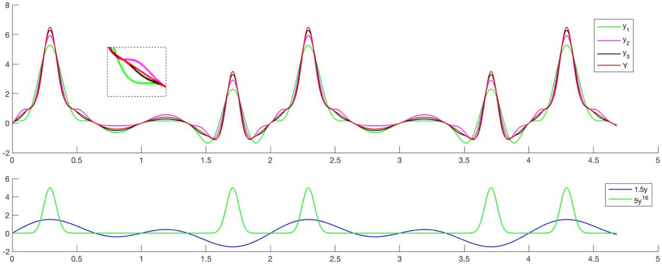


Figure: Multi-layer

Non-HDNN: performance analysis

- F-principle in DNN:
  - lower frequency $\rightarrow$ higher frequency.
- F-principle in HDNN:
  1. lower frequency $\rightarrow$ higher frequency
  2. higher coefficient $\rightarrow$ lower coefficient $(Y = 1.5y + 5y^{16})$



Connection and difference between HDNN and DNN

- Universal approximation:

$$\mathcal{R}^n \xrightarrow{f} \mathcal{R}^m. \tag{24}$$

$$f(y) \to Y. \tag{25}$$

- Neural Network:

$$\sigma(W_i y_i + b_i) \to Y^i. \tag{26}$$

- HDDN:

$$g_1(y) + g_2(y) + g_3(y) \to Y. \tag{27}$$

Connection and difference between HDNN and DNN

- $dim(y) = dim(Y)$
  – universal approximation theorem
  1. 2-layer approximation: MLP
  2. n-layer approximation: CNN
  3. n-order approximation: Taylor expansion
- $dim(y) \neq dim(Y)$
  – dimension reduction
  – multi-kernel

To be continued...

- How to find an appropriate bases to approximate universal function?
  - how to fit cross term, such as $y_i * y_{i+1}$?
  - how to keep feature invariants?
    (scaling, translation, rotation, illumination chages...)
  - how to introduce multi-kernel?
  - how to introduce dimension reduction?

Thank you!

Q & A