神经网络求 ODE 和 PDE 数值解

NN for solve ODE and PDE

学生：唐国鑫
导师：孙峪怀

SICNU　数学科学学院

2021 年 5 月 5 日

目录

微分方程的一般数值解法

差分方法

- Euler 方法
- Taylor 级数法
- Runge-Kutta 方法
- 线性多步法
- Adams 格式
- Gear 格式
- ...

其它方法

- 有限元法

微分方程的神经网络解法

神经网络可以认为是一个强大的函数逼近器，通过最小化损失函数来训练整个网络，下图展示了 CNN 的拓扑结构 [1]:

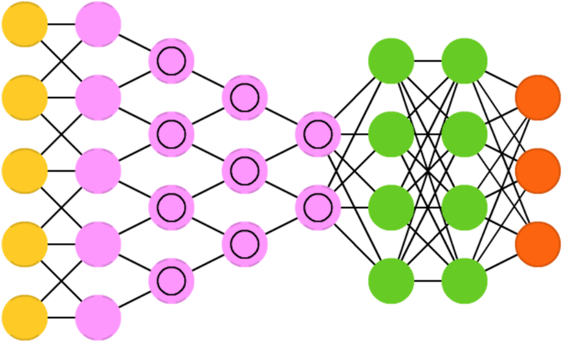

图 1: 卷积神经网络：CNN

微分方程的一般形式

连续形式

$$G\left(\vec{x}, \Psi(\vec{x}), \nabla\Psi(\vec{x}), \nabla^2\Psi(\vec{x}), ...\right) = 0, \vec{x} \in D \tag{1}$$

其中，$\vec{x} = (x_1, \ldots, x_n) \in R^n$， $D \subset R^n$.

离散形式

$$G\left(\overrightarrow{x_i}, \Psi\left(\overrightarrow{x_i}\right), \nabla\Psi\left(\overrightarrow{x_i}\right), \nabla^2\Psi\left(\overrightarrow{x_i}\right), ...\right) = 0, \forall\overrightarrow{x_i} \in \hat{D} \tag{2}$$

寻找损失函数

如果 $\Psi_t(\vec{x},\vec{p})$ 是微分方程(2)的一个试解，那么求解微分方程数值解转化为下面的问题 [2]：

Loss Function

$$\min_{\vec{p}} \sum_{\vec{x_i}\in\hat{D}} G\left(\vec{x_i}, \Psi_t\left(\vec{x_i},\vec{p}\right), \nabla\Psi\left(\vec{x_i},\vec{p}\right), \nabla^2\Psi\left(\vec{x_i},\vec{p}\right), \ldots\right)^2 \tag{3}$$

其中，$\vec{p}$ 是神经网络的参数

试解的构造

$$\Psi_t(\vec{x}) = A(\vec{x}) + F(\vec{x}, N(\vec{x},\vec{p})) \tag{4}$$

其中，$N(\vec{x},\vec{p})$ 表示我们的神经网络 (NN)，$\vec{x}$ 为输入，$\vec{p}$ 为网络参数。A 和 F 为满足边界的辅助函数。

试解的构造方法

试解的构造形式

$$\Psi_t(\vec{x}) = A(\vec{x}) + F(\vec{x}, N(\vec{x}, \vec{p})) \tag{5}$$

例如，对于下式常微分方程：

$$\frac{d^2}{dx^2}\Psi + \frac{1}{5}\frac{d}{dx}\Psi + \Psi = -\frac{1}{5}e^{-\frac{x}{5}}\cos x \tag{6}$$

边界和初值为：$\Psi(0) = 0$，$\frac{d}{dx}\Psi(0) = 1$ 且 $x \in [0, 2]$。我们构造的试解如下：

$$\Psi_t(x) = x + x^2 N(x, \vec{p}) \tag{7}$$

解析解为：

$$\Psi(x) = e^{-\frac{x}{5}}\sin(x) \tag{8}$$

目录

模型说明

为了用神经网络求解微分方程，我们构造了如下的多层感知机 (MLP) 模型：

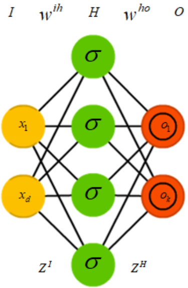

图 2: MLP 模型

参数说明

1. 输入/隐含/输出层：H/I/O
2. 权重：w^{ih}/w^{ho}
3. I 层输入：$I = [x_1, x_2, ..., x_d]$
4. H 层输入：$Z^I = Xw^{ih}$
5. H 层输出：$H = [h_1, h_2, ..., h_h]$
6. O 层输入：$Z^H = Hw^{ho}$
7. O 层输出：$O = [o_1, o_2, ..., o_k]$
8. 激活函数：$\delta(X) = 1/\left(1 + e^{-X}\right)$

训练方法

神经网络的训练方法有很多，包括：

常见训练方法

1. BP
2. SGD
3. ADAM
4. BFGS
5. LBFGS
6. ...

接下来我们主要推导 BP 算法，事实上，BP 算法就是梯度下降法。

BP 算法

Algorithm 1 BP 算法

Require: X,η,ε,$MaxIter$ and (4)
Ensure: w^{ih} and w^{ho}
1: **repeat**
2: 计算 $I = [x_1, x_2, ..., x_d]$
3: 计算 H 层输入：$Z^I = Xw^{ih}$
4: 计算 H 层输出：$H = [h_1, h_2, ..., h_h]$
5: 计算 O 层输入：$Z^H = Hw^{ho}$
6: 计算 O 层输出：$O = [o_1, o_2, ..., o_k]$
7: 更新 O 层与 H 层之间的梯度 $w^{ho} = w^{ho} + \eta\nabla w^{ho}$
8: 更新 H 层与 I 层之间的梯度 $w^{ih} = w^{ih} + \eta\nabla w^{ih}$
9: **until** 是否满足 Max_Iter 或 ε
10: **return** w^{ih} and w^{ho}

BP 算法的推导

在整个算法中，我们需要求解的参数有：

隐含层与输出层之间的权重 w^{ih}

$$w^{ih} = \begin{bmatrix} w^{ih}_{11} & \cdots & w^{ih}_{1h} \\ \vdots & \ddots & \vdots \\ w^{ih}_{d1} & \cdots & w^{ih}_{dh} \end{bmatrix}_{d\times h} \tag{9}$$

输入层与隐含层之间的权重 w^{ho}

$$w^{ho} = \begin{bmatrix} w^{ho}_{11} & \cdots & w^{ho}_{1k} \\ \vdots & \ddots & \vdots \\ w^{ho}_{h1} & \cdots & w^{ho}_{hk} \end{bmatrix}_{h\times k} \tag{10}$$

输入层输出

$$X = \begin{bmatrix} x_1 & \cdots & x_d \end{bmatrix} \tag{11}$$

隐含层输入

$$Z_H^{in} = Xw^{ih} \tag{12}$$

隐含层输出

$$H = \delta\left(Z_H^{in}\right) \tag{13}$$

输出层输入

$$Z_O^{in} = Hw^{ho} \tag{14}$$

输出层输出

$$O = g\left(Z_O^{in}\right) \tag{15}$$

损失函数

$$L = \frac{1}{2}(O - Y)(O - Y)^T \tag{16}$$

其中，Y 为实际输出。

w^{ho} 的更新

对 w^{ho} 求偏导:

$$\frac{\partial L}{\partial w_{hk}^{ho}} = \frac{\partial L}{\partial O_k}\frac{\partial O_k}{\partial g}\frac{\partial g}{\partial Z_O^{in}}\frac{\partial Z_O^{in}}{\partial w_{hk}^{ho}} = L_k g' H_h \tag{17}$$

引入学习率 η:

$$\Delta w_{hk}^{ho} = -\eta L_k g' H_h \tag{18}$$

引入局部梯度的定义:

定理 1

局部梯度

$$\gamma_O^k = \frac{\partial L}{\partial O_k}\frac{\partial O_k}{\partial g}\frac{\partial g}{\partial Z_O^{in}} = L_k g' \tag{19}$$

权值修正量:

$$\Delta w_{hk}^{ho} = -\eta\gamma_O H_h \tag{20}$$

w^{ih} 的更新

同 w^{ho} 的计算，我们应有：

$$\frac{\partial L}{\partial w_{dh}^{ih}} = \gamma_H^h X_d = \frac{\partial L}{\partial H_h}\frac{\partial H_h}{\partial \sigma}\frac{\partial \sigma}{\partial Z_H^{in}} X_d = \frac{\partial L}{\partial H_h}\sigma'\left(Z_H^{in}\right) X_d \tag{21}$$

注意到隐含层的一个权值会影响到输出层的 k 个权值，那么：

$$\frac{\partial L}{\partial H_h} = \sum_{j=i}^{k} \gamma_O^j w_{hj} \tag{22}$$

权值修正量：

$$\Delta w_{dh}^{ih} = -\eta \sum_{j=i}^{k} \left(\gamma_O^j w_{hj}\right) \sigma' X_d \tag{23}$$

其中，$\sigma' = \sigma(1-\sigma)$.

目录

一个例子

Example

$$\frac{d}{dx}\Psi + \left(x + \frac{1+3x^2}{1+x+x^3}\right)\Psi = x^3 + 2x + x^2\frac{1+3x^2}{1+x+x^3} \tag{24}$$

condition

$$\begin{cases} \Psi(0) = 1\ (x \in [0,1]) \\ \Psi_a(x) = \frac{e^{-x^2/2}}{1+x+x^3} + x^2 \end{cases} \tag{25}$$

构造试解

$$\Psi_t(x) = 1 + xN(x, \vec{p}) \tag{26}$$

算法实现

```
W = [npr.randn(1, 10), npr.randn(10, 1)]
lmb = 0.001
for i in range(1000):
    loss_grad =  grad(loss_function)(W, x_space)
    W[0] = W[0] - lmb * loss_grad[0]
    W[1] = W[1] - lmb * loss_grad[1]
^^Iprint(loss_function(W, x_space))
res = [1 + xi * neural_network(W, xi)[0][0] for
       xi in x_space]
print(W)
plt.plot(x_space, y_space)
plt.plot(x_space, psy_fd)
plt.plot(x_space, res)
plt.show()
}
```

数值结果

我们将区间 10 等分，分别用欧拉方法和 NN 方法求解式(24)的数值解，结果如下图所示：

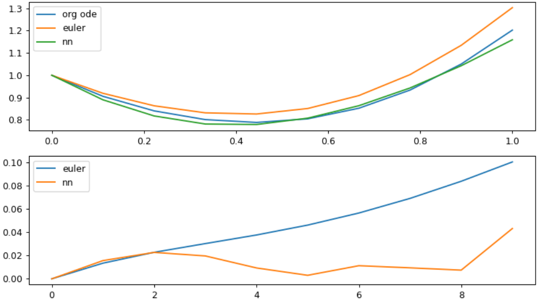

图 3: 真实解-Euler 解-NN 解

目录

结论

优势

1. 无需在意差分格式的构造
2. 可以达到极高的精度
3. 理论上随着神经元和隐含层的增多，损失函数的值趋于 0
4. 训练一个模型，可以解一类方程 [3]

劣势

1. 针对简单的微分方程仍需多次迭代，此时收敛速度无法与差分方法相提并论
2. 重点在于试解的构造
3. 不仅要求神经网络的导数，还要求试解的导数，这会导致第一点
4. 编程更为复杂

展望

神经网络在 PDE 上的应用

1. 分子蛋白运动模拟 (上一次中科院研究员卢本卓老师 201 讲座，未找到参考文献)
2. 求解薛定谔方程 [4, 5]
3. 神经网络不仅在数学工程，而且在生物分子、物理化学都在发挥着更重要的作用

致谢

Thanks!

参考文献 I

[1] I. Goodfellow, Y. Bengio, and A. Courville. Deep Learning. Deep Learning, 2016.

[2] Lagaris et al. "Artificial neural networks for solving ordinary and partial differential equations.". In: IEEE Transactions on Neural Networks (1998).

[3] Lu Lu, Pengzhan Jin, and George Em Karniadakis. "DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators". In: CoRR abs/1910.03193 (2019). arXiv: 1910.03193. URL: http://arxiv.org/abs/1910.03193.

[4] STXDETX Pfau et al. "Ab-Initio Solution of the Many-Electron Schr" odinger Equation with Deep Neural Networks". In: (2019).

参考文献 II

[5] A Jh, A Lz, and C Weab. “Solving many-electron Schrdinger equation using deep neural networks - ScienceDirect”. In: Journal of Computational Physics 399 ().