MATH308: Neural Network and ODE

Siu Wun (Tony) Cheung   Jingyan Zhang

Department of Mathematics, Texas A&M University

Nov 26, 2019

ODE Background

Examples

Problem 1

Let $\alpha > 0$ be a constant. Solve the initial-value problem

$$\frac{dy}{dt} = \cos(\alpha y), \quad y(0) = 0.$$

Evaluate $y(1)$ in terms of α.

Examples

$$\frac{dy}{dt} = \cos(\alpha y)$$

$$\int \sec(\alpha y) dy = \int dt$$

$$\frac{1}{\alpha} \log(\sec(\alpha y) + \tan(\alpha y)) = t + C$$

$$\sec(\alpha y) + \tan(\alpha y) = e^{\alpha(t+C)}$$

Substituting $y(0) = 0$, we have

$$\sec(0) + \tan(0) = e^{\alpha(0+C)} \implies C = 0,$$

and therefore

$$\sec(\alpha y) + \tan(\alpha y) = e^{\alpha t}.$$

No close form for y.

Examples

Problem 2

Let $-1 \leq \alpha \leq 1$ be a constant and the matrix $\mathbf{A}_\alpha$ be defined as

$$\mathbf{A}_\alpha = \begin{bmatrix} 1 & \alpha \\ \alpha & 1 \end{bmatrix}.$$

Solve the initial-value problem

$$\frac{d\mathbf{y}}{dt} = \mathbf{A}_\alpha \mathbf{y}, \quad \mathbf{y}(0) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Evaluate $y(1)$ in terms of α.

Examples

- Characteristic polynomial

$$\det(\mathbf{A}_\alpha - \lambda \mathbf{I}) = (1-\lambda)^2 - \alpha^2 = (\lambda - 1 + \alpha)(\lambda - 1 - \alpha).$$

- Eigenvalues and eigenvectors

$$\lambda_1 = 1-\alpha, \qquad (\mathbf{A}_\alpha - \lambda_1 \mathbf{I})\mathbf{v}_1 = 0 \implies \mathbf{v}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix},$$

$$\lambda_2 = 1+\alpha, \qquad (\mathbf{A}_\alpha - \lambda_2 \mathbf{I})\mathbf{v}_2 = 0 \implies \mathbf{v}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

- General solution

$$\begin{aligned}\mathbf{y}(t) &= C_1 e^{\lambda_1 t}\mathbf{v}_1 + C_2 e^{\lambda_2 t}\mathbf{v}_2 \\ &= \begin{bmatrix} C_1 e^{(1-\alpha)t} + C_2 e^{(1+\alpha)t} \\ -C_1 e^{(1-\alpha)t} + C_2 e^{(1+\alpha)t} \end{bmatrix}.\end{aligned}$$

Examples

- Initial condition

$$\mathbf{y}(0) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \implies C_1 = -\frac{1}{2} \text{ and } C_2 = \frac{1}{2}.$$

- Final solution

$$\mathbf{y}(t) = \frac{1}{2} \begin{bmatrix} -e^{(1-\alpha)t} + e^{(1+\alpha)t} \\ e^{(1-\alpha)t} + e^{(1+\alpha)t}. \end{bmatrix}$$

$$\mathbf{y}(1) = \frac{1}{2} \begin{bmatrix} -e^{1-\alpha} + e^{1+\alpha} \\ e^{1-\alpha} + e^{1+\alpha}. \end{bmatrix}.$$

Diagonalization becomes potentially infeasible for large systems.

Finite difference

- Partition the time interval $[0, 1]$ into m equal subintervals. Take $h = 1/m$ and $t_n = nh$.

$$0 = t_0 < t_1 < \cdots < t_{n-1} < t_n < t_{n+1} < \cdots < t_m = 1.$$

- **Forward difference**

$$\frac{d\phi}{dt}(t_n) \approx \frac{\phi(t_{n+1}) - \phi(t_n)}{h} \text{ for small } h.$$

- **Backward difference**

$$\frac{d\phi}{dt}(t_n) \approx \frac{\phi(t_n) - \phi(t_{n-1})}{h} \text{ for small } h.$$

Euler methods

- Replace the derivative by forward difference in Problem 1, we have

$$\frac{y_{n+1} - y_n}{h} = \cos(\alpha y_n), \quad y_n = y(t_n) \text{ for } n = 0, 1, 2, \ldots, m-1.$$

- Result in the **explicit Euler method.**

$$\begin{aligned} y_0 &= 0, \\ y_{n+1} &= y_n + h\cos(\alpha y_n) \quad \text{for } n = 0, 1, 2, \ldots, m-1. \end{aligned}$$

- Terminal time $Y = y_m \approx y(1)$.

Euler methods

- Replace the derivative by backward difference in Problem 2, we have

$$\frac{\mathbf{y}_n - \mathbf{y}_{n-1}}{h} = \mathbf{A}_\alpha \mathbf{y}_n, \quad \mathbf{y}_n = \mathbf{y}(t_n) \text{ for } n = 1, 2, \ldots, m.$$

- Result in the **implicit Euler method.**

$$\mathbf{y}_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$
$$\mathbf{y}_n = (\mathbf{I} - h\mathbf{A}_\alpha)^{-1}\mathbf{y}_{n-1} \quad \text{for } n = 1, 2, \ldots, m.$$

- Terminal time $\mathbf{Y} = \mathbf{y}_m \approx \mathbf{y}(1)$.

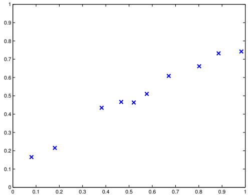
Euler methods

- Easy to understand.
- Convert the **continuous** dynamical process into a **discrete** one, i.e. piece-by-piece in time.
- Result in successive **iterations** in temporal progression.
- Easy to implement by computer programs (simple for-loop).
- Lose accuracy when replacing the derivative by finite difference.
- Trade-off between accuracy and computations cost.
- Serves as a method to find an **input-output relationship** $\alpha \mapsto \mathbf{Y} \approx \mathbf{y}(1)$.
- Have to start over another sequence of iterations for each choice of α.

Function approximation

Basic ideas

- A set of data points $\{(x_i, y_i)\}_{i=1}^{N}$ obtained by sampling or experimentation of an input-output mechanism (**target function**).
- Among a class of functions, find the function that **best approximates**, i.e. most closely matches, the target function.
- The best approximate, a.k.a. the **model**, is used to interpret the original target function or predict future observations.



Basic ideas

- **Loss function** $\mathcal{L}(f)$ determines how well a function f approximates the target function.
- Objective: Among a class of functions, find the function which minimizes the loss function.
- A commonly used loss function is the Euclidean error at the data points

$$\mathcal{L}(f) = \left(\frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i))^2 \right)^{\frac{1}{2}}.$$

Interpolation

- Among all polynomials of degree at most $N-1$, find the **interpolant** $p(x)$ which satisfies

$$p(x_i) = y_i \text{ for all } i = 1, 2, \ldots, N \iff \mathcal{L}(p) = 0.$$

- Equivalently, one has to seek the **coefficients** a_k of the interpolant

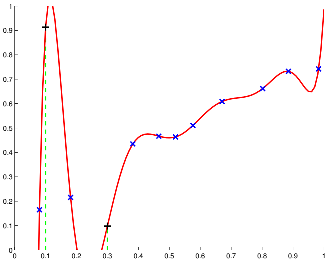
$$p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{N-1} x^{N-1}.$$

- An explicit formula of the interpolant is given by

$$p(x) = \sum_{i=1}^{N} p(x_i) \prod_{j \neq i} \frac{x - x_i}{x_j - x_i}.$$

Overfitting

- **Overfitting**: a model follows too closely to a particular set of data, but fail to fit additional data or predict future observations reliably.
- Interpolation follows exactly to the data points and suffers from overfitting.



Regression

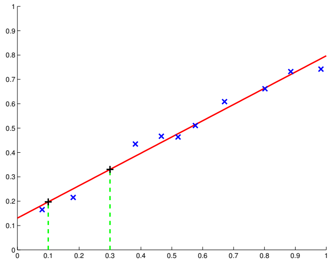
- Seek the model in a class of functions with higher interpretability.
- Relax the fitting of data points.
- Usually less prone to overfitting.

Linear regression

- Among all linear polynomials, find the **regressor** $p(x)$ which minimizes the loss function $\mathcal{L}(p)$.
- Equivalently, one has to seek the **coefficients** a_k of the regressor

$$p(x) = a_0 + a_1 x.$$

- The regressor can be explicitly found by solving a **least-squares problem.**



Summary

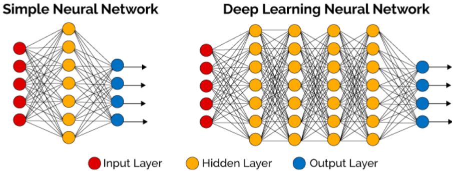
Methods	Interpolation	Linear regression	Neural network
Class of functions	Polynomials of deg. $\leq N-1$	Straight line	Composition of nonlinear activation
Loss	Min. $\to 0$	Minimized	Minimized locally
Model parameters θ	Coefficients $a_0, a_1, \ldots, a_{N-1}$	Slope a_1 intercept a_0	Weight matrices W_k bias vectors b_k

Deep Neural Network

Deep Neural Network (DNN)

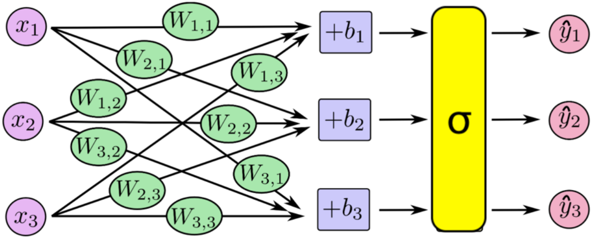
Theorem (universal approximation theorem)

A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$, under mild assumptions on the activation function.



- DNNs can model complex relationships.

One Layer Neural Network



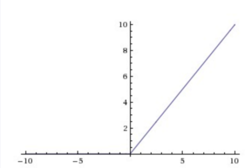
- We do not have control over data (x_i, y_i).
- We can tone the weights(e.g. $W_{2,3}$) and bias (e.g.b_2)such that the output of the network can produce results($\hat{y}_i$) consistent with ground truth label of training data(y_i)

Activation Function

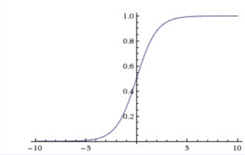
- Nonlinear
- Easy to find derivative

Examples

- ReLu $\sigma(x) = \max(0, x)$



- Sigmoid Function $\sigma(x) = \frac{1}{1+e^{-x}}$



Deep Neural Network

- A deep neural network (DNN) can be written as

$$\hat{y} = \mathcal{N}(x;\theta) = \sigma(W_n\sigma(\cdots\sigma(W_2\sigma(W_1 x + b_1) + b_2)\cdots) + b_n),$$
$$\theta := (W_1, W_2, \cdots, W_n, b_1, b_2, \cdots, b_n), \tag{1}$$

where W_i represents the weight matrix connecting two layers, b_i is bias, while σ is a simple point-wise nonlinear function.

Training by Optimization

- Given samples points $\{(x^i, y^i) | i = 1, 2, 3, \cdots, N\}$, use $\mathcal{N}(x; \theta)$ to find relationship between x^i and y^i by toning θ. We want to find θ^*

$$\hat{y}^i = \mathcal{N}(x^i; \theta^*) \approx y^i$$

- If $\mathcal{L}(\cdot)$ is a metric used to measure the mismatch between $\hat{y}$ and true value y. We can find such θ^* by solving

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta). \tag{2}$$

Loss Functions

- Measure mismatch between prediction $\hat{y}$ with true solution y with a loss function(cost function).
- Intuitively, the loss will be high if we're doing a poor job of prediction and it will be low if we're doing well

Examples

- Mean Square Error (MSE)

$$\mathcal{L}(y, \hat{y}) = \frac{\sum_i^n (y_i - \hat{y}_i)^2}{n}$$

Optimization

- Finding optimal weight matrix W and bias b to minimize the loss $\mathcal{L}(\theta)$ with given training samples (x^i, y^i)
- Gradient Decent: $W_i^{(n+1)} = W_i^{(n)} - \gamma \cdot \nabla_{W_i^{(n)}} \mathcal{L}(\theta)$
- Optimize by iteration

```
  initialize network weights (often small random values)
  do
     forEach training example named ex
        prediction = neural-net-output(network, ex)  // forward pass
        actual = teacher-output(ex)
        compute error (prediction - actual) at the output units
        compute Δw_h for all weights from hidden layer to output layer
// backward pass
        compute Δw_i for all weights from input layer to hidden layer
// backward pass continued
        update network weights // input layer not modified by error
estimate
  until all examples classified correctly or another stopping criterion
satisfied
  return the network
```

General Deep Learning Procedures

Training:
Use labeled samples(training set) $\{(x^i, y^i)\}_{i=1}^{N}$ to obtain optimized Neural Network $\mathcal{N}(x;\theta^*)$.
Evaluation:
Use a set of unseen labeled samples $\{(x^i, y^i)\}_{i=N+1}^{N+M}$ and the trained model $\mathcal{N}(x;\theta^*)$ to evaluate the current model performance. If not satisfied, one can modify hyperparameter and do training and evaluation again.
Prediction:
Use the model to make prediction for a set of sample never seen by $N(x;\theta^*)$.

What DNN is Really Doing

- Approximate an unknown relationship (can be very complicated ones) using a fixed network framework.
- Different weight might be used to capture abstract features. Stacking up layers of simple features gives complicated and abstract features.
- Nonlinear activation function is more or less an switch to filter out some irrelevant features.

Applications

Problems

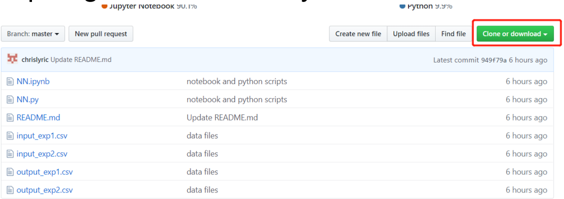
- Objective: For each of problems 1 & 2, find a neural network model for the input-output mechanism

$$\alpha \mapsto \mathbf{Y}.$$

- A set of data points $\{(\alpha_i, \mathbf{Y}_i)\}_{i=1}^{N}$ is obtained by the corresponding Euler method (independently for each α) and serves as training set.
- Train a neural network for future predictions of **Y** with other inputs α.

How to use the code: Step 0

Step 0: Download code and data at https://github.com/chrislyric/ODE_ML



How to use the code: Step 1

Step 1: You need something to run the code.

- Jupyter notebook
- Anaconda
- Editor online
- ...

How to use the code: Step 2

Step 2: Define the data set and output file names

```
# Parameters
LEARNING_RATE = 0.01
BATCH_SIZE = 50
EPOCHS = 100
scale = 1

input_size = 1
output_size = 2
nodes = 3
input_file = "input_exp2.csv"
output_file = "output_exp2.csv"

model_name = "./exp2_node_3"
model_file = model_name+'/model.hdf5'
weight_file = model_name+'/best_weights.h5'
```

How to use the code: Step 3

Step 3: Run! (trainig and predicting)

More to Play with

- Try different number of nodes, learning rates, number of training steps, number of layers and so on.
- Try different activation functions.
- Apply DNN to a different scenario (classification of photos, prediction of house price).