ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH

MÔ HÌNH HOÁ TOÁN HỌC

Học phần mở rộng

Mô hình SIR trong dự báo COVID-19

GVHD: Nguyễn An Khương
SV thực hiện: Võ Hoàng Hải Nam – 1810340
Thuộc nhóm HNPUS: Tô Duy Hưng – 1810198
Võ Thanh Phong – 1712633
Huỳnh Thị Uyên – 1810648
Lê Thành Sơn – 1810481

Tp. Hồ Chí Minh, Tháng 6/2020

Mục lục

Danh sách hình vẽ

1 Sử dụng học máy trong mô hình SIR

Ngoài sử dụng ước lượng bằng phương pháp Bayes, ta có thể ứng dụng Neuron Network trong việc dự đoán các hệ số $\alpha_0 = [\beta_0, \gamma_0, \mu_0]$ trong mô hình SIRD. Mã code trong báo cáo được viết bằng Python 3.8 và có sử dụng thư viện Keras hỗ trợ mô hình Neuron Network.

1.1 Chuẩn bị dữ liệu

Input
Trong báo cáo này, ta sẽ sử dụng tập dữ liệu mẫu[1] với tỉnh Hồ Bắc - Trung Quốc trong 60 ngày đầu tiên khi có dịch. Dân số Hồ Bắc tính đến năm 2020 là 58,5 triệu dân ($N_0 = 58.500.000$), các hệ số ban đầu được lấy là:

$$U_0 = [S_0, I_0, R_0, D_0] = [N_0 - I_0 - R_0 - D_0, I_0, R_0, D_0] = [58.499.511, 444, 28, 17]$$

Output
Thay vì sử dụng đầu ra là tập hợp các giá trị $[q(0), q(1), ..., q(N_t)]^T$ ta sẽ sử dụng trực tiếp các giá trị $\alpha_t = [\beta_t, \gamma_t, \mu_t]$. Các giá trị này được tính dựa vào mô hình SIRD theo phương pháp Euler với dt = 1 ngày. Cứ mỗi 1 cặp $[U_{t+1}, U_t]$ ta sẽ tính được 1 bộ $\alpha_t = [\beta_t, \gamma_t, \mu_t]$ theo công thức:

$$\begin{aligned} \mu_t &= \frac{D_{t+1} - D_t}{I_t} \\ \gamma_t &= \frac{R_{t+1} - R_t}{I_t} \\ \alpha_t &= \frac{S_t - S_{t+1}}{S_t I_t} \end{aligned} \tag{1}$$

1.2 Xây dựng model

Model được xây dựng dựa trên thư viện Keras với các thông số đã được tự điều chỉnh để cho ra kết quả phù hợp nhất.

Các thông số của Model bao gồm:

- Input shape: 60x3
- Output shape: 60x3
- Hidden Layer: 3 layer với số neuron lần lượt là `64/32/16`
- Hàm kích hoạt: `Sigmoid` theo như bài báo. Khi sử dụng các hàm kích hoạt khác như ReLU, tanh thì kết quả về sau bị phân kỳ so với số liệu thực tế.
- Optimizer: `Adam`
- Loss funcion: `Mean squared error (mse)`
- Learning rate: 0.01 là tốc độ học phù hợp trong dataset này, nếu chọn 0.001 thì kết quả hội tụ rất chậm.

[1] https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data/csse_covid_19_time_series

```
# Preprocess Data
scaler_x = MinMaxScaler()
scaler_y = MinMaxScaler()
scaler_x.fit(self.x)
xscale=scaler_x.transform(self.x)
scaler_y.fit(self.out)
yscale=scaler_y.transform(self.out)

# Config Model
model = Sequential()
model.add(Dense(64,activation='sigmoid',input_dim=3))
model.add(Dense(32,activation='sigmoid'))
model.add(Dense(16,activation='sigmoid'))
model.add(Dense(3,activation='sigmoid'))
opt = Adam(lr=0.01)
model.compile(loss='mse', optimizer=opt, metrics=['mse','mae'])

# Training Model
model.fit(xscale, yscale, epochs=500,batch_size=1,validation_split=0.2,verbose=0)
print (model.evaluate(self.x, self.out))

# Predict input
ynew= model.predict(xscale)
ynew = scaler_y.inverse_transform(ynew)
```

1.3 Giải hệ SIRD

Sau khi thu được các hệ số $\alpha_t = [\beta_t, \gamma_t, \mu_t]$ từ Model vừa train, ta sẽ kết hợp dữ liệu $U_0 = [58.499.511, 444, 28, 17]$ để giải hệ SIRD bằng phương pháp Euler với dt = 1 ngày. Cuối cùng ta so sánh kết quả vừa tìm được với số liệu thực tế.

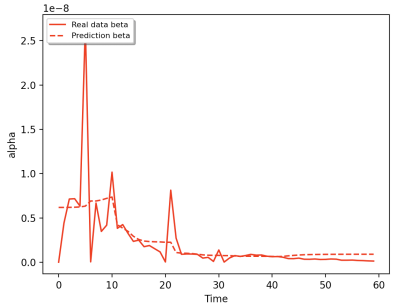
```
class EulerSIRD:
    def __init__(self, beta, gamma, mu, U_0):

    def ode_FE(self):
        S = self.U_0[0]
        I = self.U_0[1]
        R = self.U_0[2]
        D = self.U_0[3]
        N = S + I + R + D
        result = [S-self.beta*S*I, I + self.beta*S*I -
                (self.gamma+self.mu)*I, R + self.gamma*I, D + self.mu*I]
        return result
```

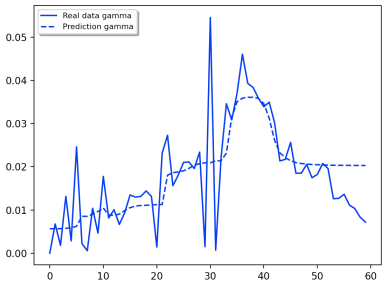
1.4 Kết quả

Ta lần lượt thử với số lần lặp khác nhau để đánh giá kết quả:

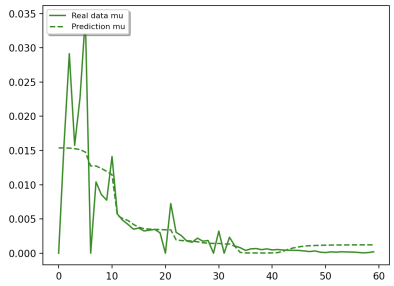
1. Với 250 lần lặp
   Đường cong thu được vẫn chưa sát với thực tế do số lần lặp còn ít



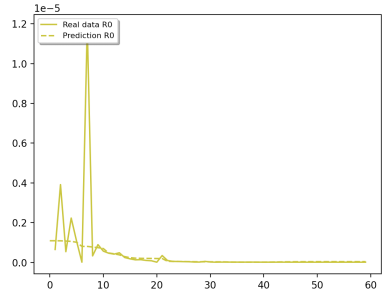
(a) *Chỉ số* β



(b) *Chỉ số* γ

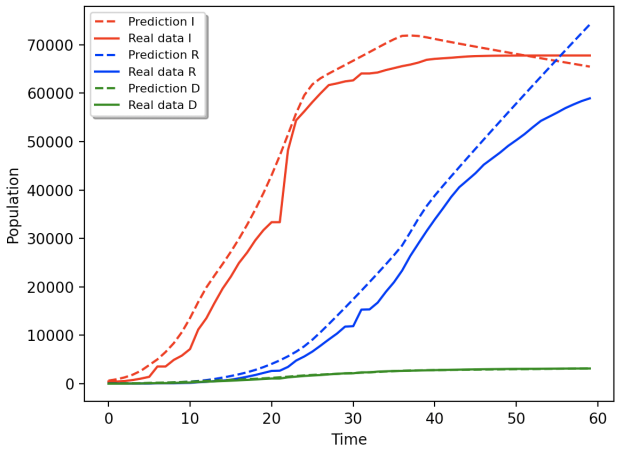


(c) *Chỉ số* μ



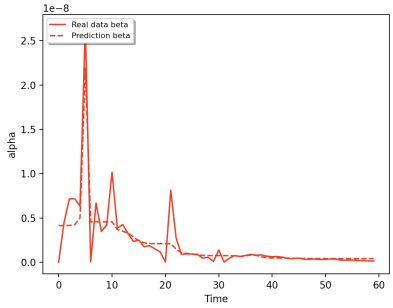
(d) *Chỉ số* $\frac{R_0}{N}$

Hình 1: *Các chỉ số của hệ SIRD sau 250 lần lặp*

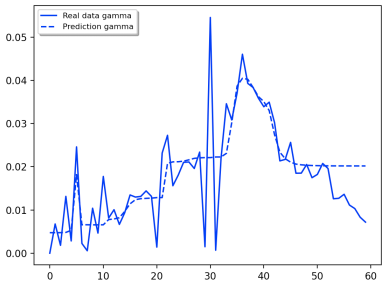


Hình 2: *Kết quả dân số ở Hồ Bắc sau 250 lần lặp*

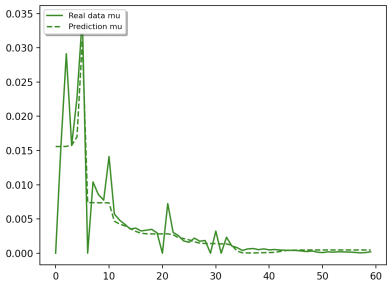
2. Với 500 lần lặp
   Đường cong thu được phần lớn đã hướng theo đường cong thực tế



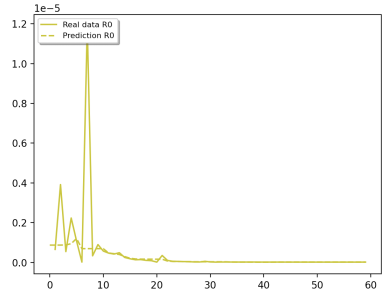
(a) *Chỉ số* β



(b) *Chỉ số* γ

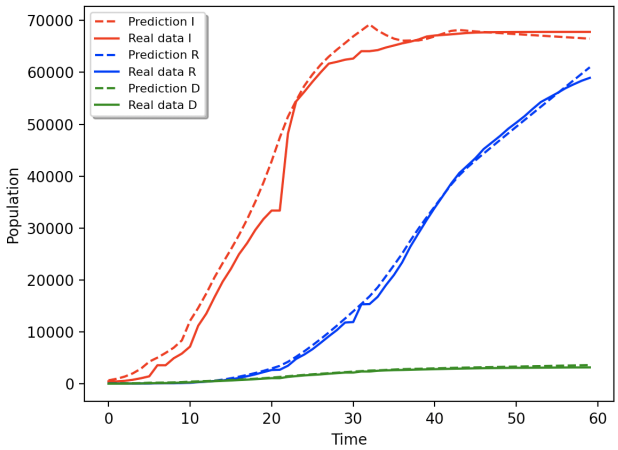


(c) *Chỉ số* μ



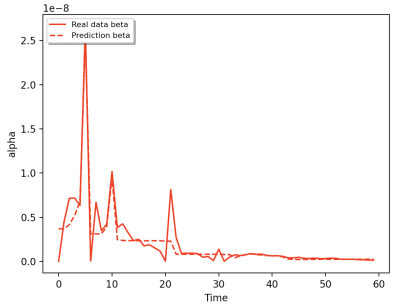
(d) *Chỉ số* $\frac{R_0}{N}$

Hình 3: *Các chỉ số của hệ SIRD sau 500 lần lặp*

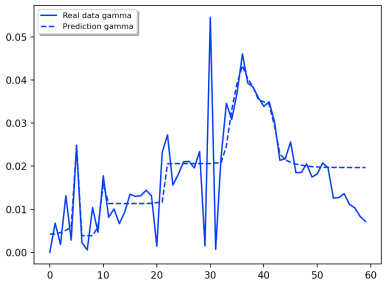


Hình 4: *Kết quả dân số ở Hồ Bắc sau 500 lần lặp*

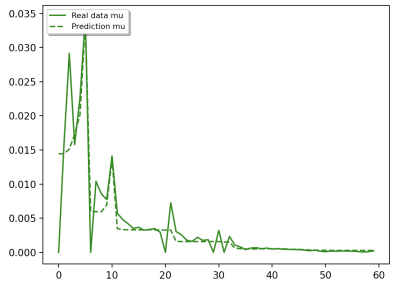
3. Với 1000 lần lặp
   Đường cong thu được phần lớn đã hướng theo đường cong thực tế, tuy nhiên không khác biệt đáng kể so với 500 lần lặp.



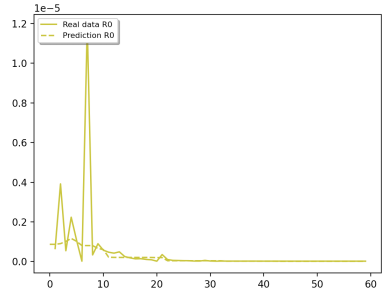
(a) *Chỉ số* β



(b) *Chỉ số* γ

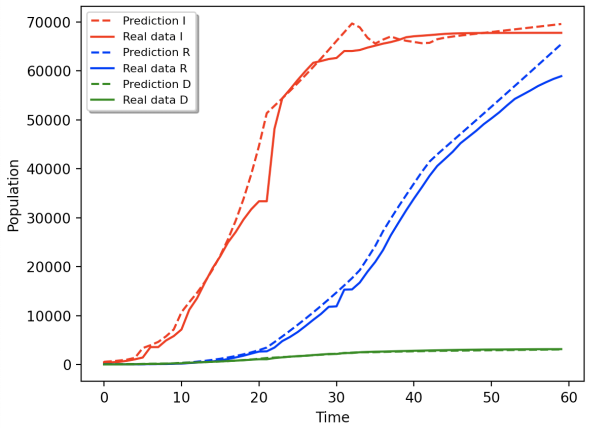


(c) *Chỉ số* μ



(d) *Chỉ số* $\frac{R_0}{N}$

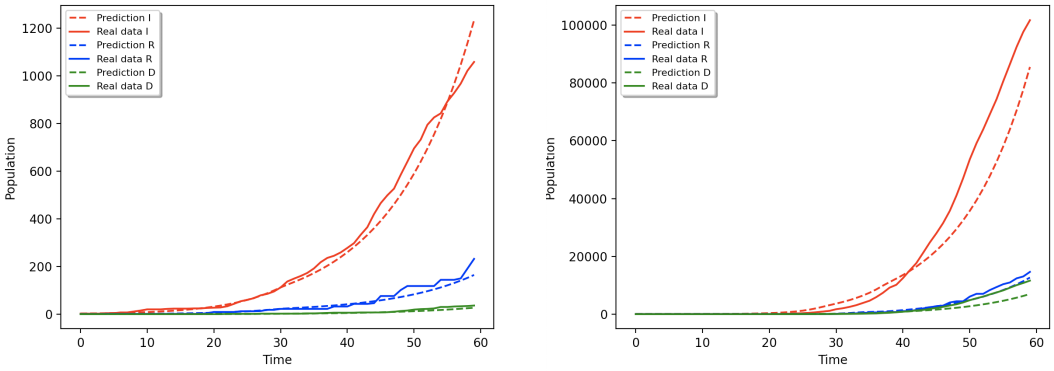
Hình 5: *Các chỉ số của hệ SIRD sau 1000 lần lặp*



Hình 6: *Kết quả dân số ở Hồ Bắc sau 1000 lần lặp*

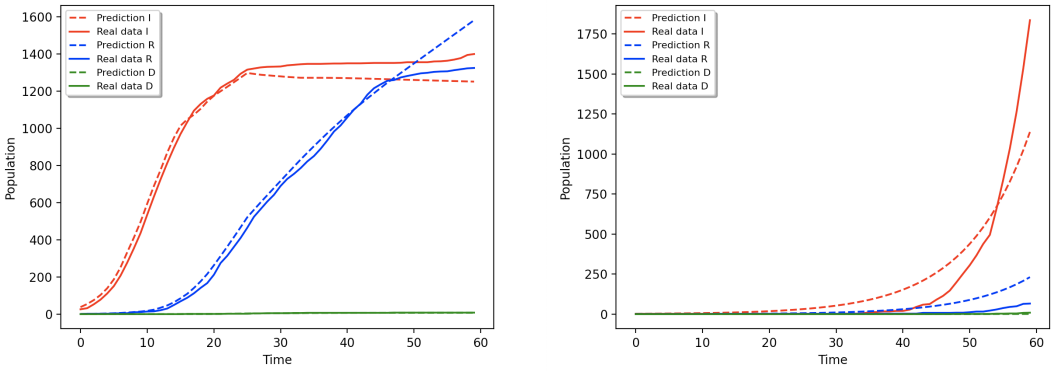
1.5 Dự đoán các khu vực khác

Ta sẽ sử dụng phương pháp tương tự ở các khu vực khác (đều sử dụng 500 lần lặp):



(a) *Nhật Bản* **(b)** *Italy*

Hình 7: *Kết quả dân số ở Nhật và Italy*



(a) *Quảng Đông - Trung Quốc* **(b)** *Nga*

Hình 8: *Kết quả dân số ở Quảng Đông và Nga*

1.6 Sử dụng hàm mất mát từ bài báo

1.6.1 Cơ sở lý thuyết

Thay vì sử dụng hàm mất mát "mean square error" trong thư viện Keras, ta sẽ sử dụng hàm mất mát trong bài báo[2].

$$L = Ed_1 + Ed_2$$

$$Ed_1 = \sum_{t=0}^{N_t}((\log_c \beta(t) - \log \widehat{\beta}(t))^2 + (\log_c \gamma(t) - \log \widehat{\gamma(t)})^2 + (\log_c \mu(t) - \log \widehat{\mu(t)})^2$$

$$Ed_2 = 0.01\frac{\log max(I_c)}{max(I_c)} \sum_{t=0}^{N_t}((\beta_c(t) - \widehat{\beta(t)})^2 + (\gamma_c(t) - \widehat{\gamma}(t))^2 + (\mu_c(t) - \widehat{\mu}(t))^2$$

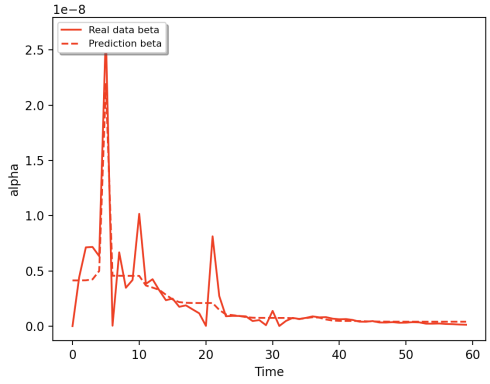
Ta sẽ điều chỉnh I_c, D_c thành β_c, γ_c, μ_c và lược bỏ hai số hạng Ed_3, Ed_4.

Ta sẽ thêm vào chương trình cũ đoạn code sau:

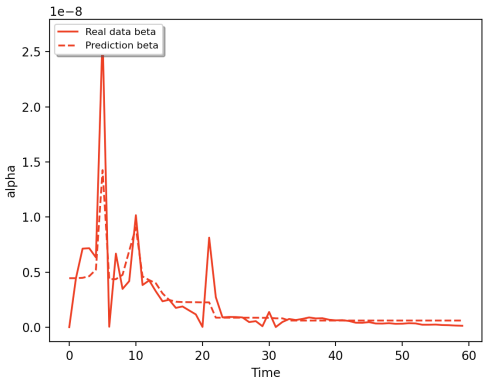
```
def customLoss(self, yTrue,yPred):
    Ed1 = K.sum(K.square(K.log(yTrue + 1) - K.log(yPred + 1)))
    Ed2 = 0.01*math.log(self.maxIc)/self.maxIc* K.sum(K.square(yTrue - yPred))
    return Ed1 + Ed2
```

1.6.2 So sánh kết quả với 1.4

Ta sẽ so sánh ở 500 lần lặp ở cả hai phương pháp.

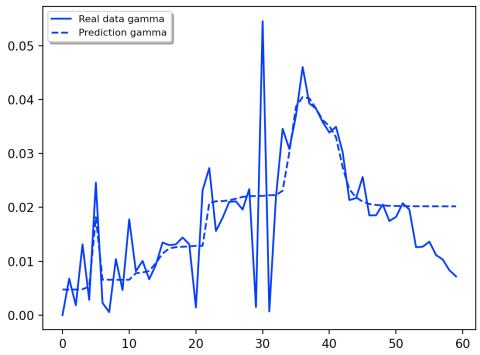


(a) *1.4* β

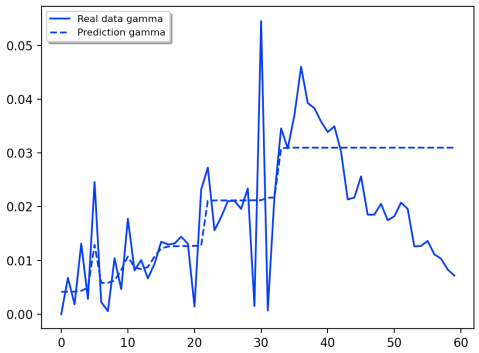


(b) *Custom loss function* β

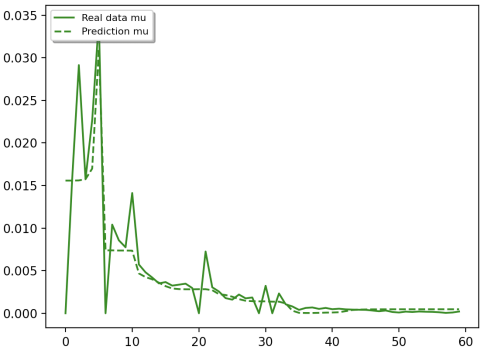
[2]Luca Magri and Nguyen Anh Khoa Doan. "First-principles Machine Learning for COVID-19 Modeling". In: arXiv preprint arXiv:2004.09478 (2020).



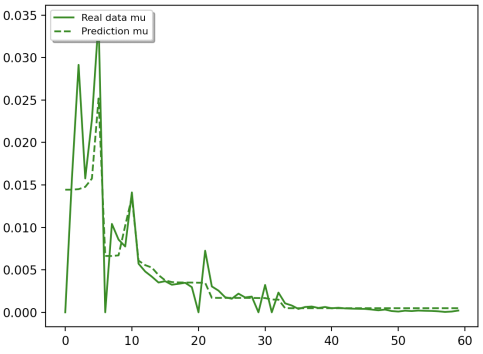
(a) *1.4* γ



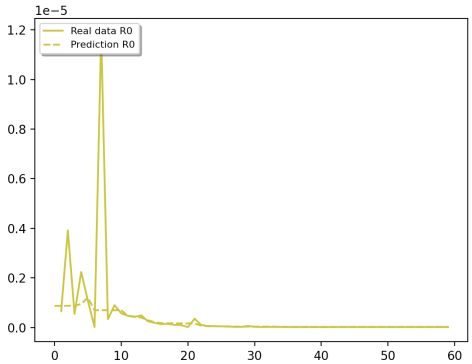
(b) *Custom loss function* γ



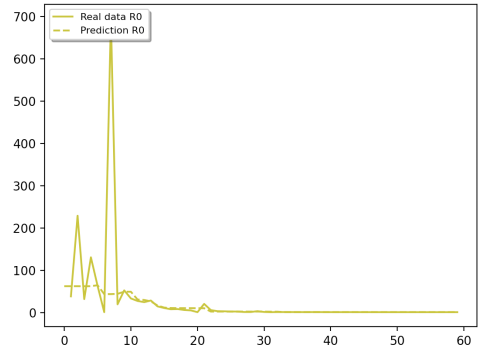
(c) *1.4* μ



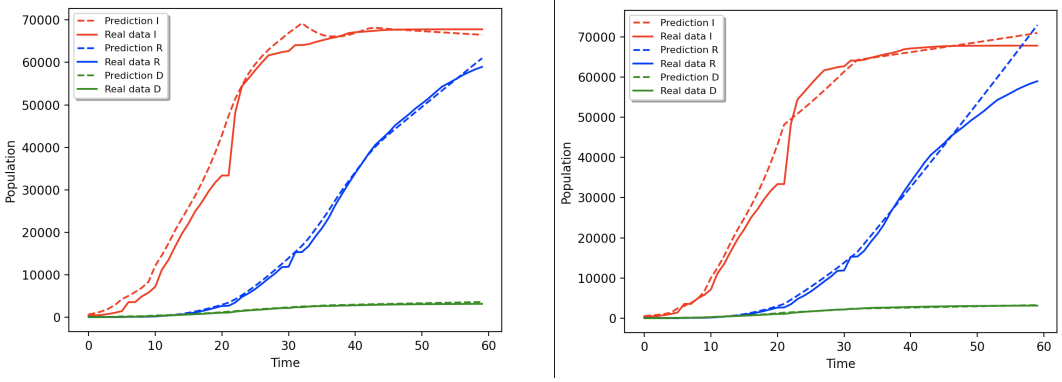
(d) *Custom loss function* μ



(e) *1.4* $\frac{R_0}{N}$



(f) *Custom loss function* $\frac{R_0}{N}$



(a) *1.4* **(b)** *Custom loss function*

Hình 9: *So sánh kết quả*

Nhận xét: Đối với hàm mất đã được tùy chỉnh lại, ở một số điểm nó tỏ ra ưu thế hơn khi đường cong dự đoán theo sát đường cong thực tế hơn. Tuy nhiên số lần lặp tăng cao (>1000) thì sự khác biệt không còn rõ nữa do hàm mất mát được tối ưu cùng cách và đạt đến mức bão hòa khó có thể giảm tiếp tục.

Tài liệu

[1] Jason Brownlee *Your First Deep Learning Project in Python with Keras Step By Step* https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/.

[2] hongthaiphi *Làm thế nào để xây dựng một mạng neural đơn giản trong 9 dòng mã Python* https://medium.com/@hongthaiphi/l%C3%A0m-th%E1%BA%BF-n%C3%A0o-%C4%91%E1%BB%83-x%C3%A2y-d%E1%BB%B1ng-m%E1%BB%99t-m%E1%BA%A1ng-neural-%C4%91%C6%A1n-gi%E1%BA%A3n-trong-9-d%C3%B2ng-m%C3%A3-python-90b528c9e0d4

[3] *Developer guides* https://keras.io/guides/

[4] JJafar Ali Habshee *On Custom Loss Functions in Keras* https://medium.com/@j.ali.hab/on-custom-loss-functions-in-keras-3af88cf59e48.