

딥러닝 Final Project

RNN (SimpleRNN / LSTM / GRU)을
이용한 출생아 수 예측

2016122032 권혁

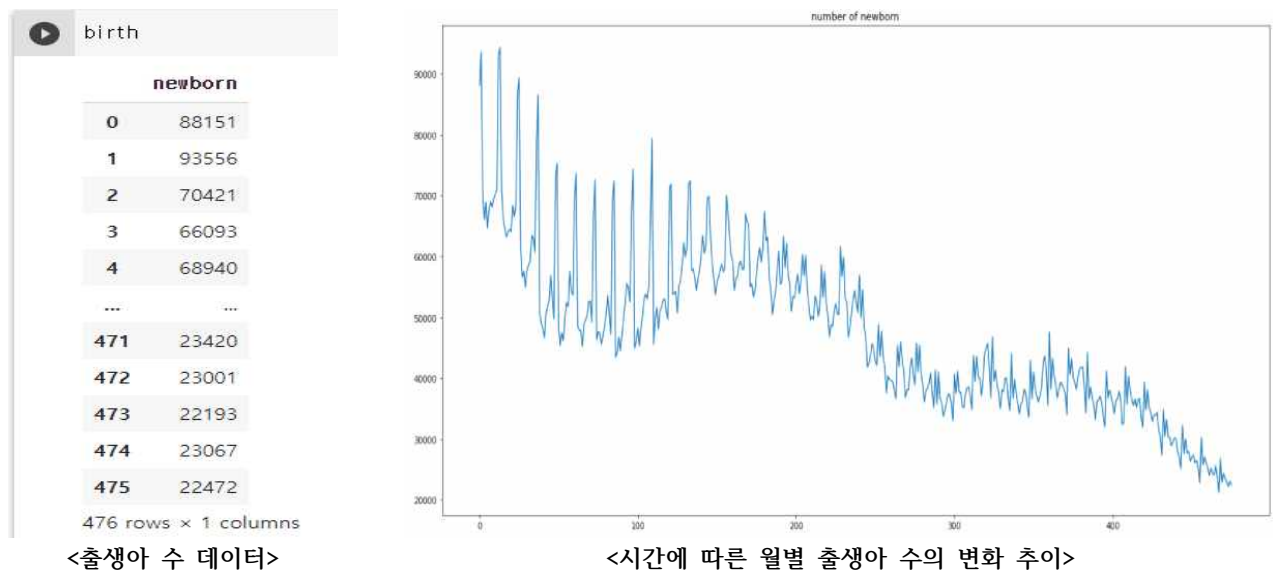
분석 배경

: 그동안 시계열 자료의 예측은 (S)ARIMA 등 기존의 통계적 모델을 사용해왔지만, 최근 들어 딥러닝 모델의 발달로 시계열 자료의 예측에서 딥러닝 모델을 활용하는 것을 종종 볼 수 있다. 딥러닝의 여러 가지 모델들 중에서도 sequence data에 적합한 RNN(recurrent neural network)이 시계열 자료를 다룰 때 많이 쓰이는 것으로 알려졌다. 따라서 이 (simple)RNN과 RNN에서 발생하는 gradient vanishing 문제를 보완한 LSTM이나 GRU를 이용해 시계열 자료를 얼마나 정확하게 예측할 수 있는지 확인해보도록 하겠다.

분석 데이터 소개

: 분석할 데이터는 ‘월별 출생아 수 데이터’이다. 최근 몇 년 동안 출산율이 심각한 수준으로까지 떨어졌다는 뉴스를 보고 이 데이터를 선정하게 되었다.

이 데이터는 통계청에서 발표한 월간 인구동향조사에서 출생 부분을 발췌한 것이다. 월간 데이터로, 1981년 1월부터 2020년 8월까지 월별 출생아 수를 나타낸 데이터로, 총 476개의 데이터로 구성되어 있다. 파이썬에서 잘 읽어들이도록 변수명을 영어로 변경하는 등 약간의 사전작업을 하였다.



=> 출생아 수의 변화 추이를 보니 일정한 주기를 가지고 증가 감소를 반복하고 있는 것을 보았다. 따라서 이 데이터에서 계절성이 있다고 판단하게 되었고, 엑셀에서 살펴본 결과 주로 연말이나 연초에 출생아 수가 많고, 여름에 출생아 수가 적은 패턴이 1년을 주기로 계속 반복하고 있었다.

Simple RNN, LSTM, GRU 모델링 결과

=> 본격적으로 딥러닝 모델을 돌리기 전에 모델이 잘 학습할 수 있도록 min-max scaler를 이용해 데이터들을 스케일링해주었다. 그리고 앞에서부터 전체 데이터의 80%를 training data로 정했고, 나머지 뒤의 20%를 test data로 정하였다. (train-380개, test-96개)

=> 1년을 주기로 반복되고 있었으므로 12개 시점의 데이터가 입력되면 그 다음 시점의 데이터를 예측하는 것을 목표로 하였다. 즉, 데이터는 예측하고자 하는 시점의 12개 이전 시점의 값들을 입력받아 모델은 이 값들을 바탕으로 해당 시점의 데이터를 예측하게 되고, 실제 값과 비교하면서 가중치를 업데이트 하게 된다.

=> test data에서는 전 시점에서 예측한 값을 이후 시점의 데이터를 예측하는 데에 사용하였다. 이에 기존의 train data / test data의 개수는 380개 / 96개 였던 것이, 367개 / 83개로 줄어들었다.

데이터의 개수가 많지 않아 간단한 모델을 사용하기로 하였다. 1개 층을 이용한 모델과 2개 층을 이용한 모델을 사용하기로 하였고, loss는 mse로, 모델의 평가기준은 rmse를 사용하기로 하였다. 모델마다 가장 좋은 결과를 찾은 것이므로 노드 수 등 parameter가 다를 수 있다.

1) SIMPLE RNN

	single layer RNN	double layer RNN
parameter	노드 수: 50 dropout: 0.2 optimizer: 'adam' 활성화함수: 'relu' batch_size = 10	노드 수: 50, 50 dropout: 0.2 optimizer: 'adam' 활성화함수: 'relu' batch_size = 10
model summary	Model: "sequential_23" <pre> Layer (type) Output Shape Param # ----- simple_rnn_19 (SimpleRNN) (None, 50) 3150 dropout_37 (Dropout) (None, 50) 0 dense_23 (Dense) (None, 1) 51 </pre> Total params: 3,201 Trainable params: 3,201 Non-trainable params: 0	Model: "sequential_5" <pre> Layer (type) Output Shape Param # ----- simple_rnn_6 (SimpleRNN) (None, 1, 50) 3150 dropout_6 (Dropout) (None, 1, 50) 0 simple_rnn_7 (SimpleRNN) (None, 50) 5050 dropout_7 (Dropout) (None, 50) 0 dense_5 (Dense) (None, 1) 51 </pre> Total params: 8,251 Trainable params: 8,251 Non-trainable params: 0
train loss		
test data 예측 결과		
test score (RMSE)	2796.69	2570.38

=> Simple RNN의 경우 single model과 double model의 차이는 크지 않다. double model에서는 첫 번째 층에서와 같은 노드 수의 RNN layer와 같은 비율의 dropout layer를 추가했을 뿐이다. 학습과정에서는 double model이 더 빠르게 loss가 수렴하는 것을 확인할 수 있었지만, test data에서의 예측 결과를 보니 두 모델의 성능 차이는 그리 커 보이지는 않는다. 두 모델 모두 하락 추세를 과소평가해 실제 값보다 높게 예측 하긴 했지만, 전체적으로 하락하는 추세뿐만 아니라 상승과 하락을 주기적으로 반복하는 패턴을 잘 잡아내어 기대보다 더 좋은 성능을 보여주었다.

2) LSTM

	single layer LSTM	double layer LSTM																														
parameter	노드 수: 55 dropout: 0.2 optimizer: 'rmsprop' 활성화함수: 'relu' batch_size = 10	노드 수: 50, 55 dropout: 0.2 optimizer: 'rmsprop' 활성화함수: 'relu' batch_size = 10																														
model summary	Model: "sequential_16" <table border="1"> <thead> <tr> <th>Layer (type)</th><th>Output Shape</th><th>Param #</th></tr> </thead> <tbody> <tr> <td>lstm_11 (LSTM)</td><td>(None, 55)</td><td>14960</td></tr> <tr> <td>dropout_19 (Dropout)</td><td>(None, 55)</td><td>0</td></tr> <tr> <td>dense_16 (Dense)</td><td>(None, 1)</td><td>56</td></tr> </tbody> </table> Total params: 15,016 Trainable params: 15,016 Non-trainable params: 0	Layer (type)	Output Shape	Param #	lstm_11 (LSTM)	(None, 55)	14960	dropout_19 (Dropout)	(None, 55)	0	dense_16 (Dense)	(None, 1)	56	Model: "sequential_63" <table border="1"> <thead> <tr> <th>Layer (type)</th><th>Output Shape</th><th>Param #</th></tr> </thead> <tbody> <tr> <td>lstm_104 (LSTM)</td><td>(None, 1, 50)</td><td>12600</td></tr> <tr> <td>dropout_112 (Dropout)</td><td>(None, 1, 50)</td><td>0</td></tr> <tr> <td>lstm_105 (LSTM)</td><td>(None, 55)</td><td>23320</td></tr> <tr> <td>dropout_113 (Dropout)</td><td>(None, 55)</td><td>0</td></tr> <tr> <td>dense_63 (Dense)</td><td>(None, 1)</td><td>56</td></tr> </tbody> </table> Total params: 35,976 Trainable params: 35,976 Non-trainable params: 0	Layer (type)	Output Shape	Param #	lstm_104 (LSTM)	(None, 1, 50)	12600	dropout_112 (Dropout)	(None, 1, 50)	0	lstm_105 (LSTM)	(None, 55)	23320	dropout_113 (Dropout)	(None, 55)	0	dense_63 (Dense)	(None, 1)	56
Layer (type)	Output Shape	Param #																														
lstm_11 (LSTM)	(None, 55)	14960																														
dropout_19 (Dropout)	(None, 55)	0																														
dense_16 (Dense)	(None, 1)	56																														
Layer (type)	Output Shape	Param #																														
lstm_104 (LSTM)	(None, 1, 50)	12600																														
dropout_112 (Dropout)	(None, 1, 50)	0																														
lstm_105 (LSTM)	(None, 55)	23320																														
dropout_113 (Dropout)	(None, 55)	0																														
dense_63 (Dense)	(None, 1)	56																														
train loss																																
test data 예측 결과																																
test score (RMSE)	2384.84	2797.68																														

=> LSTM은 시계열 예측에 가장 많이 쓰이는 모델이라고 한다. 실제로 RNN으로 시계열 자료를 예측한 것을 찾아보면 대부분이 LSTM을 이용한 것을 볼 수 있었다. LSTM의 경우 optimizer로 rmsprop을 사용했을 때 더 좋은 결과를 보여 optimizer로 adam 대신 rmsprop을 사용하였다.

=> LSTM의 경우에는 simple RNN 때와 달리 double model 보다 single model의 결과가 더 좋게 나왔는데, 자료 수가 적어 단순한 모형의 성능이 더 좋게 나온 것일 수도 있겠다고 생각했다. 이번에도 하락 추세를 과소평가해 실제보다 출생아 수를 높게 예측하였다. double model의 경우 하락하는 추세를 많이 과소평가해 80 근처에서 실제 값과 꽤 큰 격차를 보여 좀 아쉬운 모습을 보였지만 single model에서는 비교적 실제값에 가까운 예측값을 내놓았고 전체적 패턴을 잘 잡아낸 동시에 하락 추세도 꽤 반영한 것으로 보인다.

3) GRU

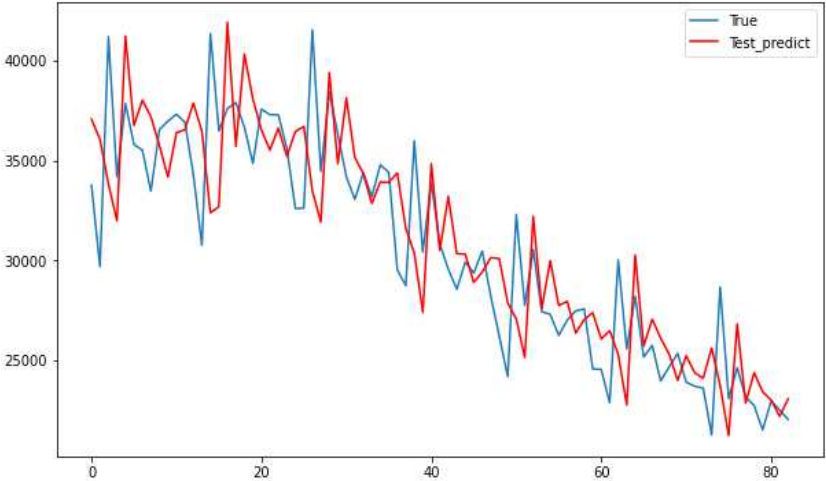
	single layer GRU	double layer GRU																														
parameter	노드 수: 50 dropout: 0.2 optimizer: 'adam' 활성화함수: 'relu' batch_size = 10	노드 수: 50, 50 dropout: 0.2 optimizer: 'adam' 활성화함수: 'relu' batch_size = 10																														
model summary	Model: "sequential_68" <table border="1"> <thead> <tr> <th>Layer (type)</th><th>Output Shape</th><th>Param #</th></tr> </thead> <tbody> <tr> <td>gru_6 (GRU)</td><td>(None, 50)</td><td>9600</td></tr> <tr> <td>dropout_120 (Dropout)</td><td>(None, 50)</td><td>0</td></tr> <tr> <td>dense_68 (Dense)</td><td>(None, 1)</td><td>51</td></tr> </tbody> </table> Total params: 9,651 Trainable params: 9,651 Non-trainable params: 0	Layer (type)	Output Shape	Param #	gru_6 (GRU)	(None, 50)	9600	dropout_120 (Dropout)	(None, 50)	0	dense_68 (Dense)	(None, 1)	51	Model: "sequential_80" <table border="1"> <thead> <tr> <th>Layer (type)</th><th>Output Shape</th><th>Param #</th></tr> </thead> <tbody> <tr> <td>gru_29 (GRU)</td><td>(None, 1, 50)</td><td>9600</td></tr> <tr> <td>dropout_143 (Dropout)</td><td>(None, 1, 50)</td><td>0</td></tr> <tr> <td>gru_30 (GRU)</td><td>(None, 50)</td><td>15300</td></tr> <tr> <td>dropout_144 (Dropout)</td><td>(None, 50)</td><td>0</td></tr> <tr> <td>dense_80 (Dense)</td><td>(None, 1)</td><td>51</td></tr> </tbody> </table> Total params: 24,951 Trainable params: 24,951 Non-trainable params: 0	Layer (type)	Output Shape	Param #	gru_29 (GRU)	(None, 1, 50)	9600	dropout_143 (Dropout)	(None, 1, 50)	0	gru_30 (GRU)	(None, 50)	15300	dropout_144 (Dropout)	(None, 50)	0	dense_80 (Dense)	(None, 1)	51
Layer (type)	Output Shape	Param #																														
gru_6 (GRU)	(None, 50)	9600																														
dropout_120 (Dropout)	(None, 50)	0																														
dense_68 (Dense)	(None, 1)	51																														
Layer (type)	Output Shape	Param #																														
gru_29 (GRU)	(None, 1, 50)	9600																														
dropout_143 (Dropout)	(None, 1, 50)	0																														
gru_30 (GRU)	(None, 50)	15300																														
dropout_144 (Dropout)	(None, 50)	0																														
dense_80 (Dense)	(None, 1)	51																														
train loss 변화 과정																																
예측 결과																																
test score (RMSE)	2463.81	2157.15																														

=> GRU는 LSTM과 거의 비슷하지만 좀 더 단순한 구조로, LSTM과 비슷한 성능을 보이지만, 학습속도가 더 빠르고 작은 데이터셋에서 더 좋은 성능을 보인다고 하여 기대를 해보았다. 우연의 일치인지, 아니면 위에서 언급한 이유때문인지는 모르겠지만, 전체적으로 LSTM보다 더 좋은 성능을 보였다.

=> single model보다 double모델에서 더 좋은 성능을 보여줬는데, 그래프를 통해 예측값 추이를 보면, 그동안의 모델들보다 예측값과 실제값과의 간격이 좁다. 비록 이전모델들과 마찬가지로 고점은 잘 반영하지 못하고, 하락 추세를 과소평가해 전체적으로 출생아수를 높게 예측했지만, 전체적인 패턴을 잘 반영한 것으로 보아 괜찮은 결과로 보인다.

SARIMA의 결과

* p, d, q, P, D, Q의 범위를 넓게 잡으면 계산량이 많아 시간이 너무 오래 걸려 0, 1로 제한하였고, 그중에서 가장 AIC가 작은 모델을 선정하였다.

	SARIMA(0,1,1)*(0,1,1,12)		
parameter	De_Seasonal	Seasonal	AIC
	0	(0, 1, 1)	(0, 1, 1, 12) 8088.486149
예측 결과			
test score (RMSE)	3051.03		

=> 예측값의 그래프가 실제값의 그래프와 매우 유사한 모습을 보이고 있다. 하락 추세를 그동안의 RNN모델들 보다 더 잘 잡아내었고, 증가/감소의 패턴도 잘 나온 것 같지만 실제값들의 패턴보다 한두단계 좀 밀려있다는 느낌을 받았다. 아마 p,d,q의 범위를 늘려 더 많은 경우의 수에서 최적의 모델을 찾았다면 더 좋은 결과를 얻을 수 있을 것이라 예상한다.

결론

- RNN의 세 종류 모델 (simple RNN, LSTM, GRU)로 시계열 자료를 예측해보았는데, 기대했던 것보다 예측값과 실제값의 차이가 작았고, 전체적으로 하락하는 추세뿐만 아니라 연초에 출생아 수가 많고 여름에 출생아 수가 적은 것이 반복되어 증가와 감소가 반복되는 패턴을 잘 반영하였다..
- SARIMA 모델에 비해 실제값과 예측값과의 차이가 커 시계열 자료 예측에 있어 RNN모델이 기존의 통계학적 모델들 보다 우위에 있다고 할 수는 없지만, 전체적인 하락추세와 패턴학습이 잘 되었다는 점에서 RNN을 활용한 시계열 자료 예측은 꽤 성공적이라고 볼 수 있었다. 만약 좀 더 복잡한 모델을 사용했다라면 더 좋은 결과를 얻을 수 있을까에 대한 생각을 해보았지만, 데이터의 개수가 476개로 작은 편이라 확신은 들지 않았다.
- 이후에 RNN이 좀 더 발달한다면 지금보다 더 정확한 예측을 할 수 있지 않을까 하는 생각이 들었고, 만약 그렇게 된다면 시계열 자료 예측에서 RNN모델이 주류가 되어 지금보다 넓은 범위에서 활용될 수 있을 것이라는 생각을 하게 되었다.

부록. Reference

1. 데이터 출처

=> 통계청 - 국내통계 - 인구 - 인구동향조사 - 월,분기,연간 인구동향(출생, 사망, 혼인, 이혼통계) 중 출생
https://kosis.kr/statisticsList/statisticsListIndex.do?menuId=M_01_01&vwcd=MT_ZTITLE&parmTabId=M_01_01#SelectStatsBoxDiv

2. RNN 모형에 대한 정보

: <https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/>

: <https://excelsior-cjh.tistory.com/185>

3. 모델링

- SimpleRNN, LSTM, GRU parameter in keras : <https://keras.io/> 참고

- LSTM :

<https://machinelearningmastery.com/time-series-prediction-LSTM-recurrent-neural-networks-python-keras/>

<https://machinelearningmastery.com/how-to-develop-LSTM-models-for-time-series-forecasting/>

- GRU:

<https://towardsdatascience.com/predictive-analytics-time-series-forecasting-with-GRU-and-biLSTM-in-tensorflow-87588c852915>

4. SARIMA

<https://machinelearningmastery.com/SARIMA-for-time-series-forecasting-in-python/>

<https://towardsdatascience.com/how-to-forecast-sales-with-python-using-SARIMA-model-ba600992fa7d>