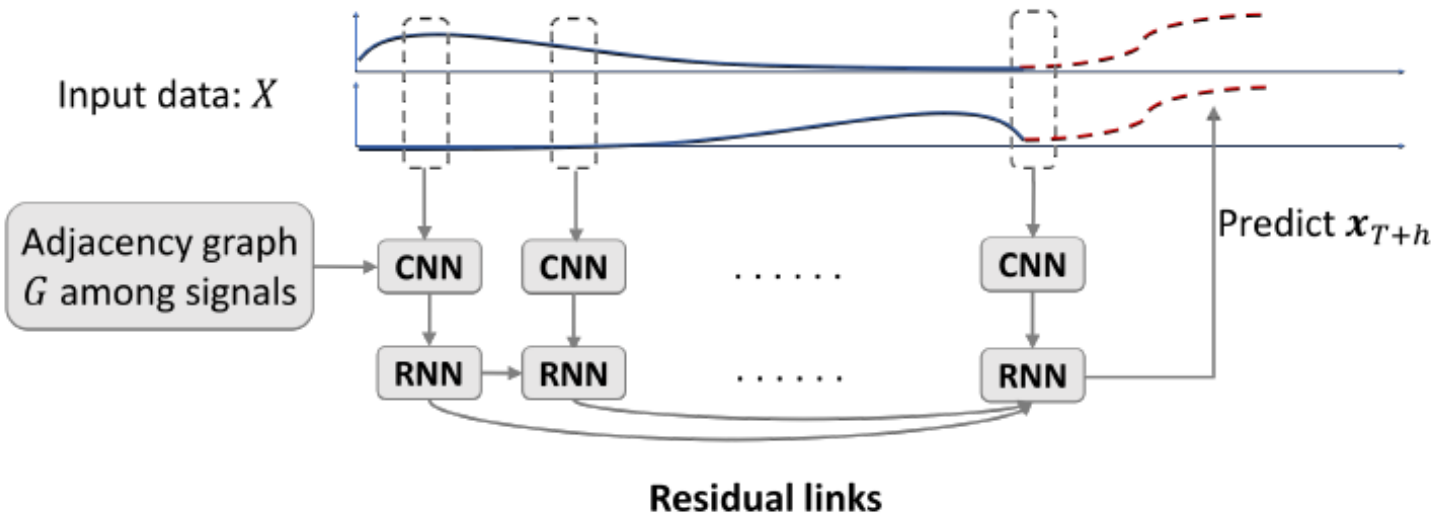
A Deep Learning Network for German State-level Influenza Forecasting

Casey Graham Gibson, Hachem Saddiki, Zhengfan Wang

May 1, 2019

Overview of the Proposed Method

1. Convolutional Neural Network (CNN) to capture spatial correlations between signals.
2. Recurrent Neural Network (RNN) for linking up the dependencies in the temporal dimension.
3. Residual links for fast training and overfitting prevention.



Proposed method (cont'd)

- CNN gives more weight to nearest neighbors in the multivariate time series (regardless of time ordering).
- CNN is used as a filter to condense the information from multiple time series into a single time series.
- This is passed to the RNN to then capture the time dependent structure of the condensed data.
- The deep learning model is trained using backpropagation through time (gradient descent).

CNN Filter

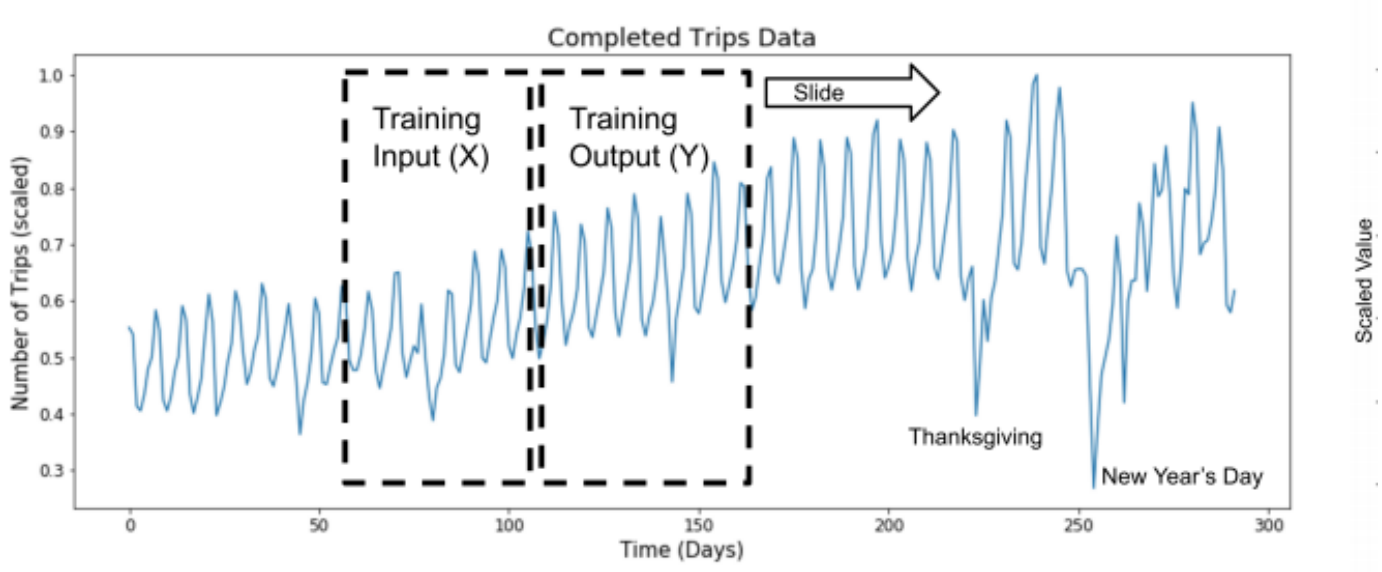
- **Input** $X_1 ... X_t$
- **Output** $h_t = \sigma(\Phi_G x_t)$

where Φ_G is a binary 16-by-16 adjacency matrix.

$\Phi_{G\,ij} = 1$ *if state* i *and* j *share a border, and* 0 *otherwise.*

RNN for Time Series

Sliding Window



Residual Links

- Residuals links were used to overcome overfitting issues in deep learning models.
- The last layer is connected to all the previous layers to allow the training process to bypass some intermediate layers.
- This design alleviates gradient vanishing issues during training and may potentially introduce highly relevant "long-jump" patterns yielding better predictive accruacy.

Implementation

- Running Python scripts in R using the package **reticulate**:
  1. First, make sure you have Python installed with the required packages to run the desired scripts.
  2. Then, import the main Python script into R as follows,

```
library(reticulate)

# tell reticulate where to find the python install folder
use_python("/Users/user/anaconda/bin/python2.7")

# source main python script into R
source_python("./models/main.py")
```

- Congratulations! now you can call python functions from R.

Fit Function

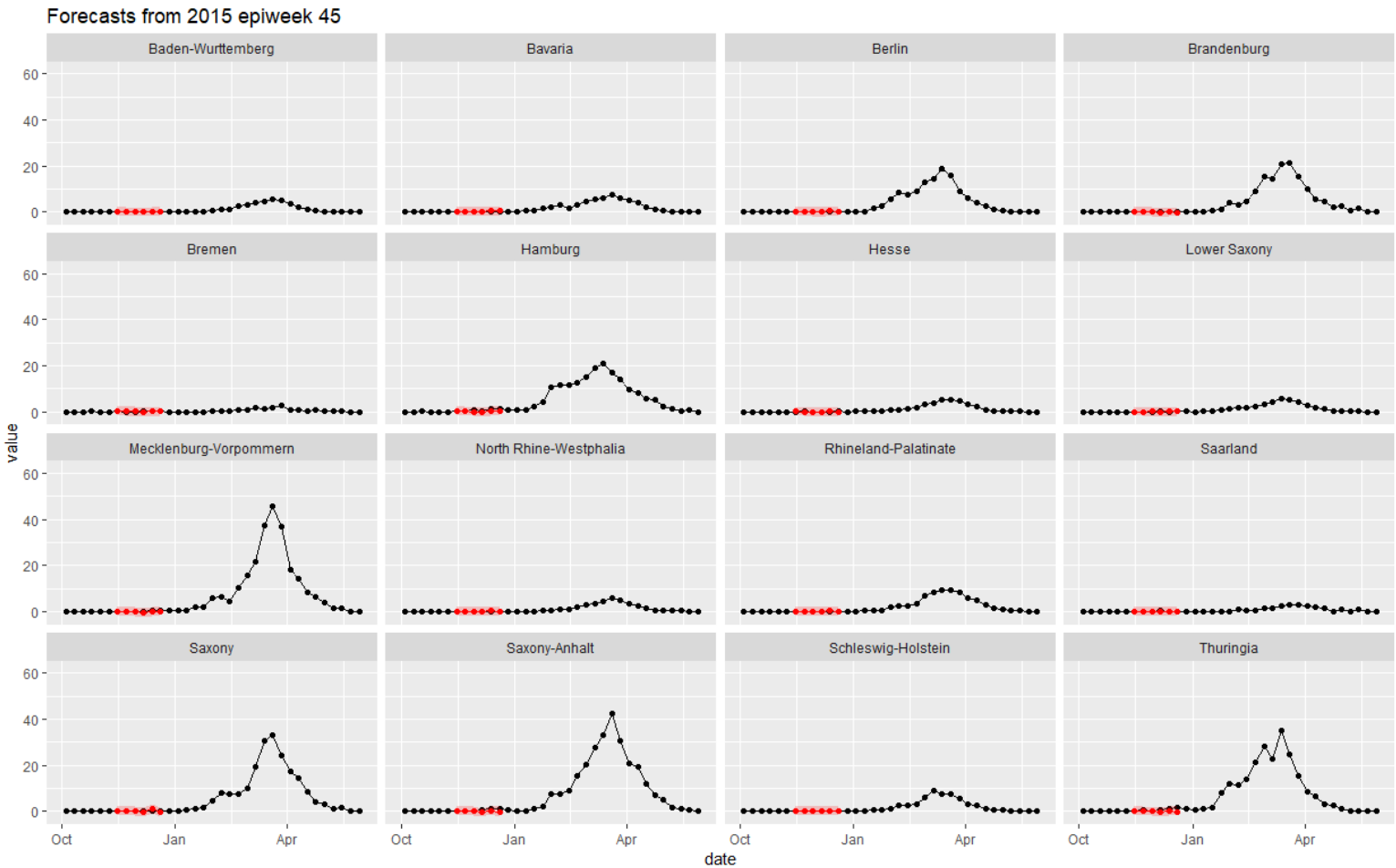
- Input data:
  1. N-by-16 matrix of weekly flu incidence for each German state (train data).
  2. 16-by-16 state adjacency matrix.
- Adjustable input parameters:
  1. **Epochs**: number of training iterations.
  2. **Window size**: number of previous observations to be used for each prediction.
  3. **Horizon**: sets the number of steps ahead forecast.
- The best performing model (lowest training error) across all epochs is saved in a file.

Forecast Function

- Takes as input an M-by-16 matrix of weekly flu incidence for each German state (test data).
- Assumes the observations from the train data **precede** those from the test data.
- Loads the best performing model from file (generated internally by the fit function).
- Uses the part of the train data corresponding to the window size from the fit function.
- Generates forecasts for the horizon specified in the fit function.

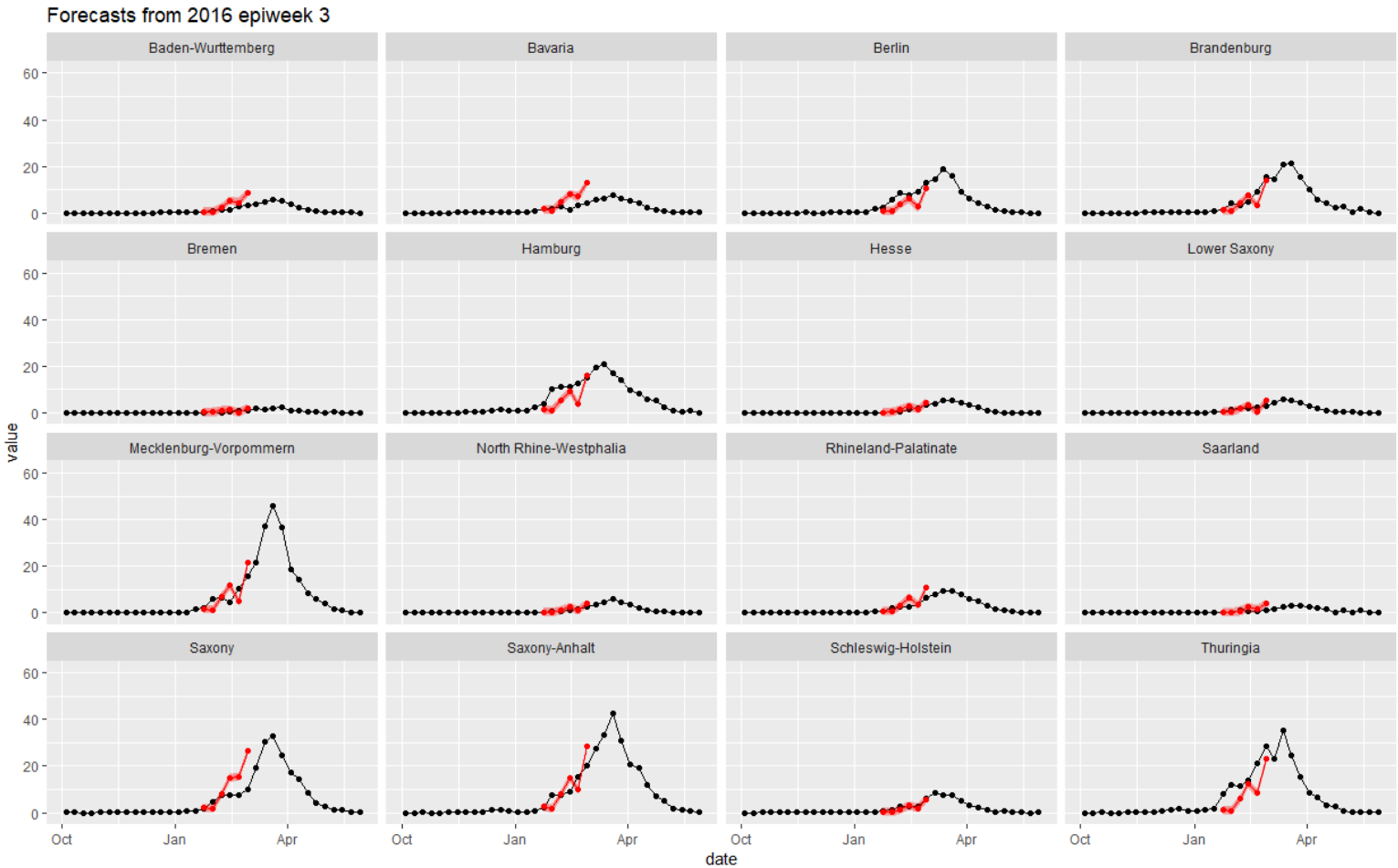
Forecasts for Epiweek 45, 2015

True vs. Predicted



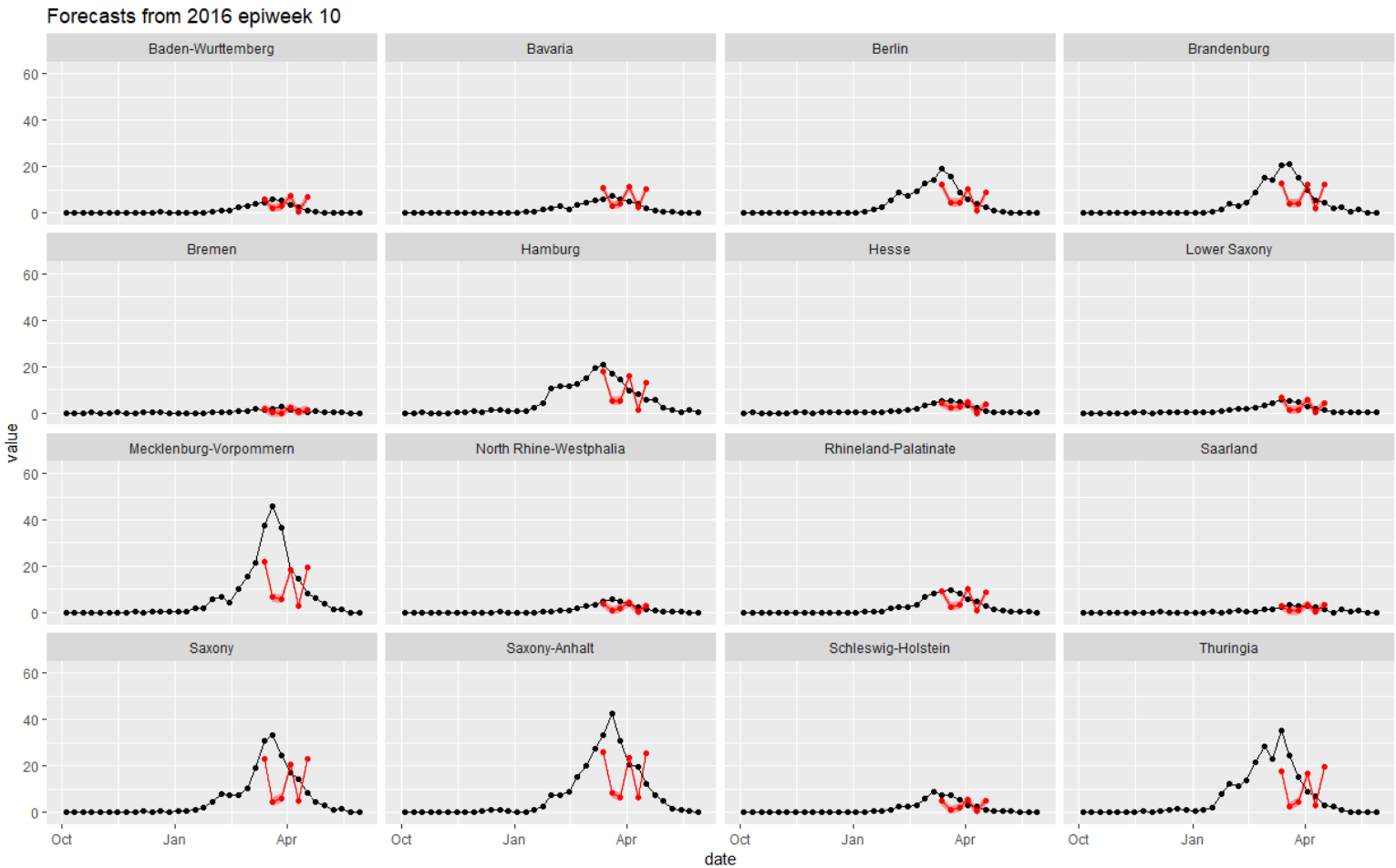
Forecasts for Epiweek 3, 2016

True vs. Predicted



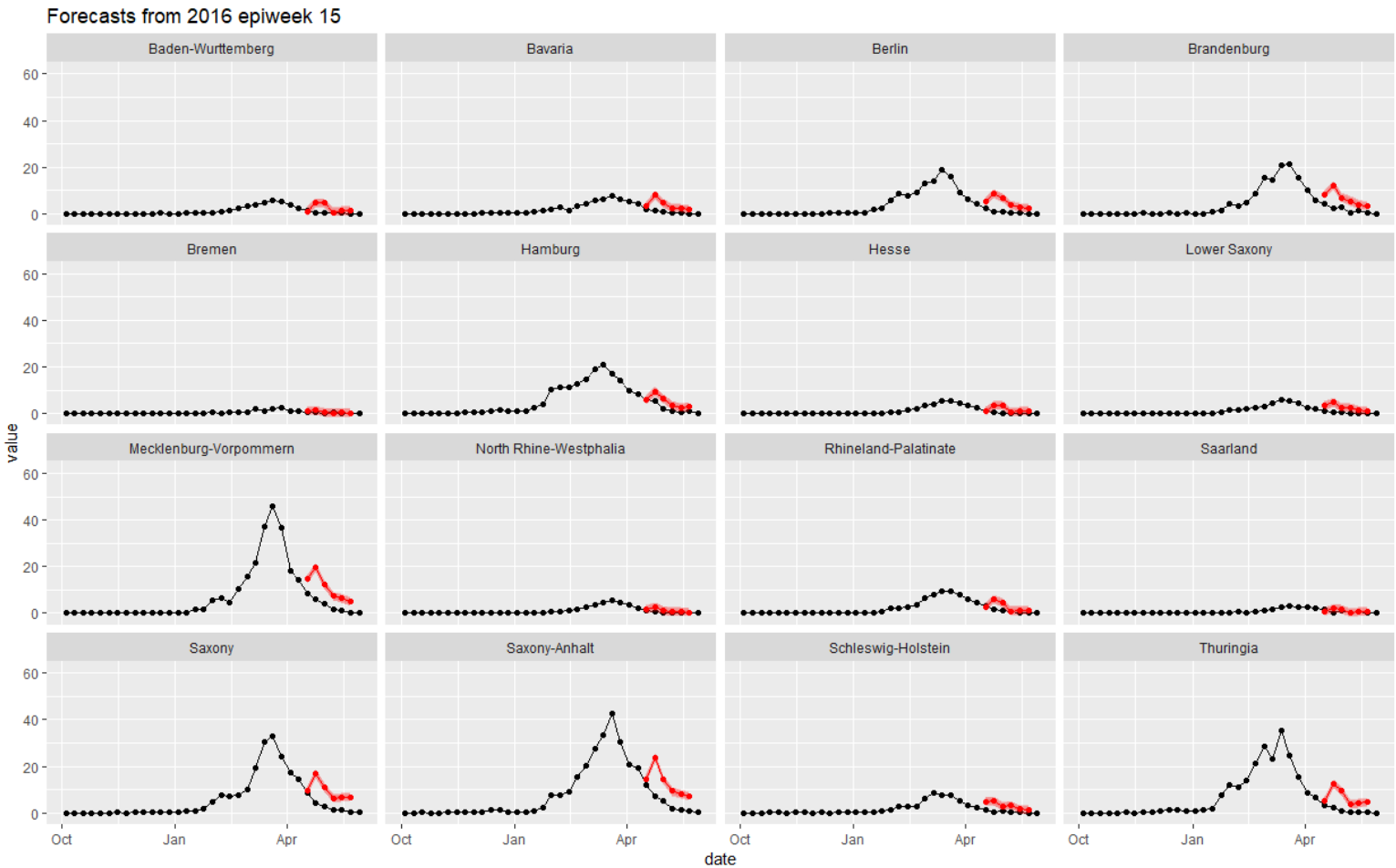
Forecasts for Epiweek 10, 2016

True vs. Predicted



Forecasts for Epiweek 15, 2016

True vs. Predicted



Bias, CIs and Coverage

	location	avg_bias	ci50_cov	ci80_cov	ci95_cov
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
1	Baden-Württemberg	-0.948	0.618	0.770	0.797
2	Bavaria	0.0561	0.498	0.666	0.696
3	Berlin	-0.150	0.531	0.682	0.720
4	Brandenburg	0.269	0.501	0.651	0.688
5	Bremen	-1.84	0.703	0.804	0.816
6	Hamburg	0.582	0.469	0.647	0.680
7	Hesse	-1.46	0.661	0.806	0.822
8	Lower Saxony	-1.20	0.608	0.779	0.803
9	Mecklenburg-Vorpommern	1.77	0.441	0.582	0.604
10	North Rhine-Westphalia	-1.56	0.680	0.813	0.831
11	Rhineland-Palatinate	-0.596	0.601	0.732	0.773
12	Saarland	-1.72	0.702	0.807	0.824
13	Saxony	2.21	0.408	0.555	0.580
14	Saxony-Anhalt	2.85	0.399	0.534	0.559
15	Schleswig-Holstein	-1.21	0.623	0.776	0.798
16	Thuringia	0.994	0.470	0.632	0.654

```
> colMeans(dplyr:: .... [TRUNCATED]
      bias   ci50_cov   ci80_cov   ci95_cov
-0.1224307  0.5571037  0.7022256  0.7277975
```

- The point forecast is pretty close to the true value.
- We use a normal distribution over the forecast results to generate confidence intervals.

Challenges

- The proposed method provides good point estimates, but no standard errors.
- Need to re-train the entire model for each input parameter setting, and for each simulated trajectory.
- Cross-validating the neural network parameters is time consuming.
- R session must be restarted every time a python script is edited for the changes to take effect.
- Need to be EXTRA careful with data type compatibility between R and Python objects.

References

- **Paper**
  Wu Y., Yang Y., Nishiura H., Masaya S. Deep Learning for Epidemiological Predictions. SIGIR (2018).
- **Authors' code**
  https://github.com/CrickWu/DL4Epi
- **Our code**
  https://github.com/gcgibson/german-flu-forecasting