Feature Importance Identification through Bottleneck Reconstruction

Linsong Chu, Ramya Raghavendra, Mudhakar Srivatsa
IBM T.J. Watson Research Center
{*lchu, rraghav, msrivats*}*@us.ibm.com*

Alun Preece, Daniel Harborne
Cardiff University
{*preecead, harborned*}*@cardiff.ac.uk*

***Abstract*—We address the problem of feature importance. Often, when working with classification or regression problems, the results of black-box deep learning techniques are held to scrutiny in an effort to interpret which and to what extent various features affect outcome. We address this issue specifically when the model has a bottleneck which we will be used to infer feature importance. In this paper, we apply this technique to weather data and study which weather features affect traffic most. To this end, we introduce convolutional spatial embedding to convert data with spatial information into spatial images that are suitable for convolutional neural networks. An advantage of our approach is in dealing with input that has highly correlated features, where removing even an important feature will not increase loss.**

I. Introduction

We address the problem of identifying feature importance in a neural network. At a high level, we study feature importance by adding a reconstruction decoder to a bottleneck layer in an existing neural network and computing the reconstruction error. When the model goes through an information bottleneck (e.g., an encoder-decoder model), implicitly, the best-encoded representation of the input for the target is retained. Then, when we train the reconstruction decoder that reconstructs the bottleneck back to the input, we can infer feature importance by checking how well each feature is reconstructed. The key idea behind this is as follows: with a model that has a bottleneck, only important features get preserved in the bottleneck so these features will be reconstructed better than features that are less important.

In this paper, we consider a specific example of which weather features most affect vehicular traffic. The premise we have is that adverse weather conditions (rain, snow, etc.) impact driving conditions, and hence create traffic bottlenecks on the road. We apply our model structure to this scenario to study the problem of feature importance identification and, in order to do so, we first introduce spatial embedding to use spatial data in a deep encoder-decoder pipeline.

Spatial embedding is a process of converting data with spatial information into spatial images. A spatial image is created by dividing the region of interest into grids where each grid is defined using a geohash. Each pixel in the spatial image corresponds to a geohash of a certain bit depth, and non-spatial features are assigned as pixel values to different pixels based on their locations. With spatial embedding, we convert all the information into a simple image — spatial information is encoded as the image layout while non-spatial information is encoded as pixel values (i.e., image channels). With such spatial images, we can leverage CNN models to fully capture the underlying spatial information within the data. In fact, our embedding can not only capture spatial information, but also temporal information because a sequence of images can capture temporal information, e.g., weather data over a year can be represented by a sequence of 365 daily images. With such spatio-temporal embedding, we can leverage a pipeline with a Recurrent Neural Network (RNN) followed by a Convolutional Neural Network (CNN) to fully capture both temporal information and spatial information [**?**]. However, our focus here is feature importance so we want to keep our network structure minimal for the major structure by not introducing extra complexity to it; therefore, we decided to use handcrafted weather features to capture temporal information in the experimental work.

In our example of weather affecting traffic, this convolutional spatial embedding is very meaningful. Traffic in one neighborhood is not only affected by the weather condition in the same neighborhood, but also highly affected by neighboring area. With spatial embedding, we are able to capture such spatial dependencies through the convolutional operations, thus improving the accuracy of our predictions over methods that only look at local data. We did the same spatial embedding for target data as well: traffic data is encoded as a single channel image where each pixel value corresponds to the traffic congestion value in that pixel, i.e., a geohash.

Our initial results are promising, and largely coincide with both our intuition and feature importance extracted from other models (e.g., a decision tree model). Comparing this technique with a more straightforward approach of extracting feature importance (removing each feature and rerunning the model to check loss change), our method is shown to have advantages. One obvious advantage is the capability of dealing with input that has highly correlated features, in which case removing one of them will not increase the loss even though it can be very important. As such, examining all features in unison and comparing their reconstruction errors has enabled us to make more accurate feature selection decisions.

There are some challenges before we can generalize this technique, which we plan to address in the immediate future. We will discuss this in the last part of this paper.

978-1-7281-2711-8/19/$31.00 ©2019 IEEE
DOI 10.1109/ICCC.2019.00022

II. Network Architecture

The high level network architecture is illustrated in Figure 1. It consists of two paths: i) a *prediction path* which is a typical encoder-decoder structure [?] that goes from input data to target data, which is trained to predict the target using input features; ii) a *feature reconstruction* path which is a decoder that goes from trained encoded data (i.e., output from bottleneck layer) to reconstructed input data. The prediction and feature reconstruction paths run sequentially. First, a typical encoder-decoder is trained using given input data and target data. After the encoder-decoder is fully trained, the encoded data is extracted and fed into another decoder (feature reconstruction decoder) to reconstruct back to the original input. Once the reconstruction decoder is also fully trained, we examine the reconstruction error for each feature and use that as a proxy for feature importance.

III. Experiments

We apply this network structure to study the importance of weather features when predicting traffic. We have three steps at a high level. The first step is to transform our dataset into spatial images using spatial embedding in order to make it fit for a convolutional encoder-decoder, which will be discussed in the Dataset section. The second step is to train an encoder-decoder by using weather data as input and traffic data as target; network details will be discussed in Prediction Path section. The final step is to train the reconstruction decoder, which goes from the last layer of previous encoder to the original input, which will be discussed in Feature Importance Path section.

A. Dataset

We use two internal datasets in our evaluations: weather data as input, and traffic data as target, in the area of greater Chicago. Our purpose is to study which weather features are more important when predicting traffic.

The weather data comprises 24 features such as temperature, humidity, rain, snow etc., with each feature being a historical hourly average. Each weather feature is normalized by z-score and values are clipped to [-3, 3] in order to avoid outliers. Extra indicator (binary) features are generated for 6 rain and snow related features. The traffic data consists of road segments and the corresponding maximum and observed vehicle speeds. For traffic data, we use normalized traffic data from major roads, namely FRC (Functional Road Class) 0-2. And traffic congestion, which is defined as road speed normalized by road reference speed, is computed and used as our target.

We then transform both weather data and traffic data into spatial images (essentially, a tensor) by using spatial embedding with details as follows: 1) Use geohash as image pixels. We use 25 bitdepth (5 digit base-32 precision) geohashes [?] as pixels when constructing spatial images for both weather and traffic data. For weather data, we use weighted average to compute weather features for each geohash. For traffic data, we project each road segment to the corresponding

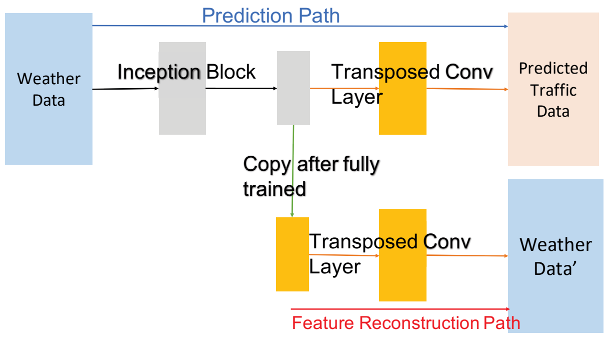

Figure 1. Architecture of experiment. For prediction path, two inception blocks are used as encoder while two transposed conv layers are used as decoder. For feature reconstruction path, two transposed conv layers are used as decoder.

geohash(es) and get a weighted congestion value for each geohash. There are some geohashes that do not have any major roads thus have no traffic data available; we fill those pixels with value -1 and assign 0 weights to them during network training to prevent the neural network from learning on these regions. Our area of interest is divided into 17*18 geohashes, thus the shape of spatial images are 17*18. 2) Use features as image channels. Each local feature value is converted to one dimension of the pixel value, so each feature becomes one channel of the spatial image. We have 24 weather features, and thus 24 channels for weather images. For traffic data, we use traffic congestion as a single channel for the traffic images.

Using such spatial embedding, we generated 8760 (365 days * 24 hours) weather images and traffic images for the year 2016 for the greater Chicago area. And we created two tensors for our neural network: one with shape (8760, 17, 18, 24) for weather data and another with shape (8760, 17, 18, 1) for traffic data.

Note that, we only use weather features as our input in the prediction path because the main goal here is not prediction but feature importance. We believe that y(t-1) is the most important feature in predicting y(t), especially in case of traffic, but our goal here is to study weather feature importance when affecting traffic. We do not want to distort the importance of weather features by introducing traffic itself as an input feature.

B. Prediction And Feature Reconstruction Paths

We build a convolutional [?], [?] encoder-decoder for the prediction path, as shown in Figure 1. We use two inception blocks as our encoder, the intuition is that the kernel size defines weather condition in how large of the area should attribute to the traffic condition in the current area, and we want the neural network to learn this hyper-parameter, instead of using our own knowledge to define the convolution kernel size. We have kernel sizes 1, 3 and 5 for the first inception block and kernel sizes 3 and 7 for the second inception block, both with proper padding to adjust the image height and width. After two inception blocks, our "encode" image (bottleneck layer) has both height and width of 8. For decoder, we use two transposed convolutional layers with the first layer using 2*2 kernel with stride 2 and second layer using 2*3 kernel with stride 1.

We build a convolutional decoder for feature importance path, as shown in Figure 1. We use the same decoder structure as in the prediction path, except that we change the number of neurons in each layer to make it reconstruct to correct number of channels.

C. Feature Importance

We use per-channel normalized reconstruction error as proxy for feature importance. First, after feature reconstruction path is trained, we pass all the data once again through the feature reconstruction path to get the reconstructed weather images. Second, we calculate per-channel RMSE (root mean square error) between reconstructed weather images and raw weather images. Then, we calculate per-channel RMSE baseline for each weather feature as if we are reconstructing each channel to its mean. This step is important as the raw distribution of input features might be different. Finally, we calculate the normalized per-channel reconstruction error by normalizing the per-channel RMSE by baseline RMSE. And we say that the feature tend to be more important if it is reconstructed with smaller loss because more information is preserved in the encoded data during prediction path.

D. Result

We picked several features of interest and rank them based on the normalized reconstruction error; results are shown in Table I. Note that since all these features are normalized with z-score during data preprocessing, the baseline per-channel RMSE is 1. Therefore, the raw reconstruction error is the same as normalized reconstruction error, and we display raw error only in the following table.

Feature	Reconstruction Error
Temperature (zscore)	0.00462615
Temperature Feels Like (zscore)	0.00538474
Temperature Change 24 Hour (zscore)	0.01004373
Wind Speed (zscore)	0.01443422
Pressure Change (zscore)	0.03341724

Table I
RECONSTRUCTION ERROR FOR SEVERAL FEATURES .

The results shows that temperature is the most important feature while temperature-feels-like is almost equally important, followed by temperature change and wind speed, and the least important feature out of these five is pressure change. We did the same experiments on the greater Portland area as well and got very similar results. Results from both areas coincide with our knowledge about this problem and the reconstruction error acts as a good proxy for feature importance.

Another thing we can observed is that, since temperature and temperature-feels-like are highly correlated features, they also have a very similar reconstruction error. Therefore, for highly correlated features, we tend to get similar feature importance through this approach, which is as desired. This is an advantage over other approaches like dropping/noising features. For example, if we drop or add noise to temperature only, the model performance won't be affected even if temperature is the most important weather feature, simply because there exists an alternative feature, temperature-feels-like, which will act as temperature.

IV. DISCUSSION AND NEXT STEPS

We believe this to be a very promising direction, opening research problems in both theoretical and empirical directions. From a model structure perspective, this approach is generalizable to any neural network that has a bottleneck layer. Also, as we introduce spatial embedding, we provide a way to transform data with spatial information into spatial images which can be processed using a CNN and spatial dependencies are well captured by convolutional operations. Another obvious advantage of this approach is that it allows for spatial dependencies in the input data to be captured.

There are a few challenges ahead with this particular approach. As the first step, we plan to extend the network to include the capability of dealing with features that have piecewise impact on the target, because the network alone is not sufficient enough to draw robust feature importance from these features. Take precipitation for example: let's assume it affects traffic at constant levels in different intervals, i.e., a simple step function. In this case, the actual value/variance within each interval might be dropped during training and those information might be gone in the bottleneck layer. As a result, when we try to reconstruct the input back to its original value, the reconstruction error might be large, even though these features should be important. While this is work in progress, we have a few preliminary experiments to address this issue. For instance, we have run the input data through the entire prediction and reconstruction path, and then we study the reconstruction error in different intervals for each feature and use that intuition to do feature-bucketization and re-do our approach again with the new bucketized features. However, building such a wide model is brute-force and requires manual check on the first run, so we are exploring other possible solutions.

REFERENCES

[1] G Niemeyer. Geohash. https://en.wikipedia.org/wiki/Geohash.

[2] V. Badrinarayanan et al. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *CoRR*, 2015.

[3] A. Krizhevsky et al. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*. 2012.

[4] C. Streiffer et al. Darnet: A deep learning solution for distracted driving detection. In *ACM/IFIP/USENIX Middleware Conference: Industrial Track*, Middleware '17, 2017.