Time-Series Modeling with Neural Networks at Uber

Nikolay Laptev

June 26, 2017

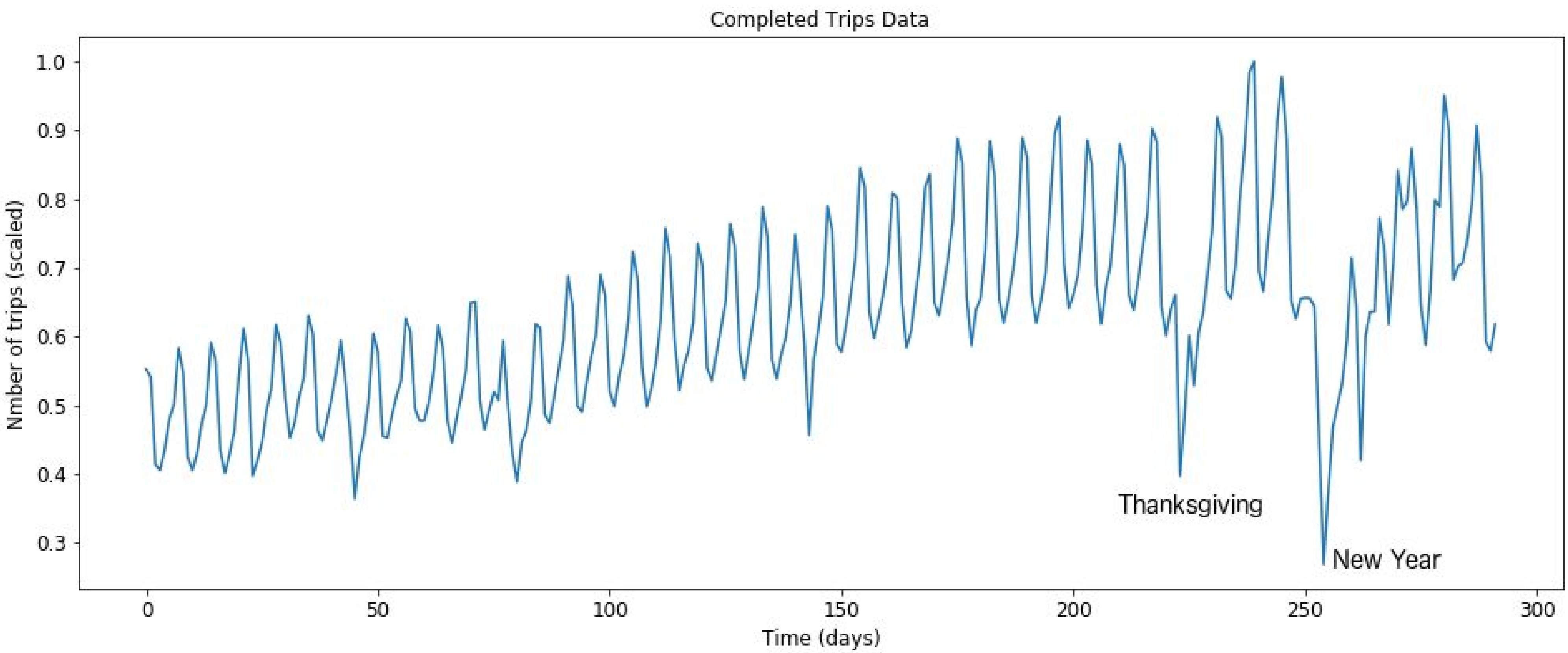
Outline

- **Motivation**
- **Modeling with Neural Nets**
- **Results & Discussion**

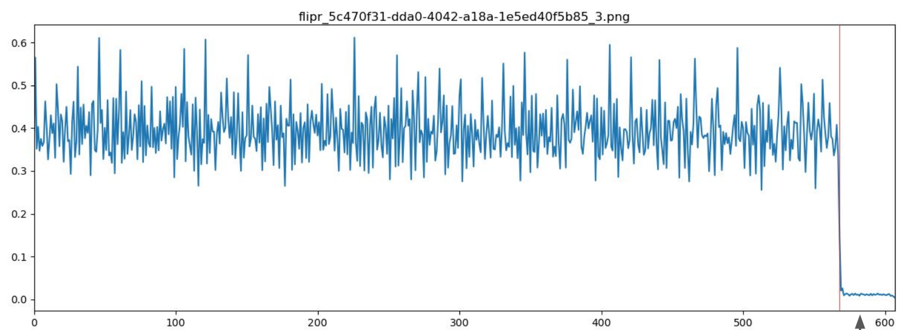
Outline

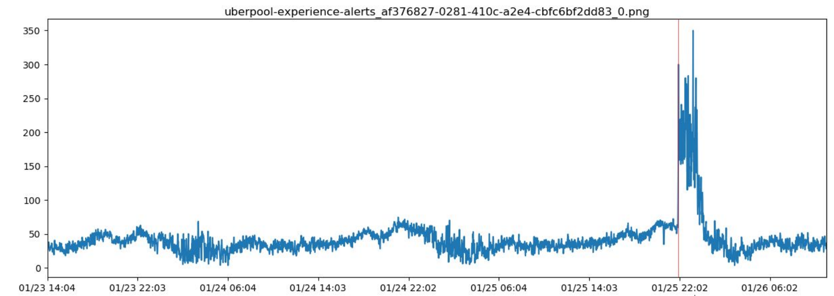
- **Motivation**
  - Special Event Prediction
  - Applications
  - Current solution
- **Modeling with Neural Nets**
- **Results & Discussion**

Motivation: Special Event Forecasting



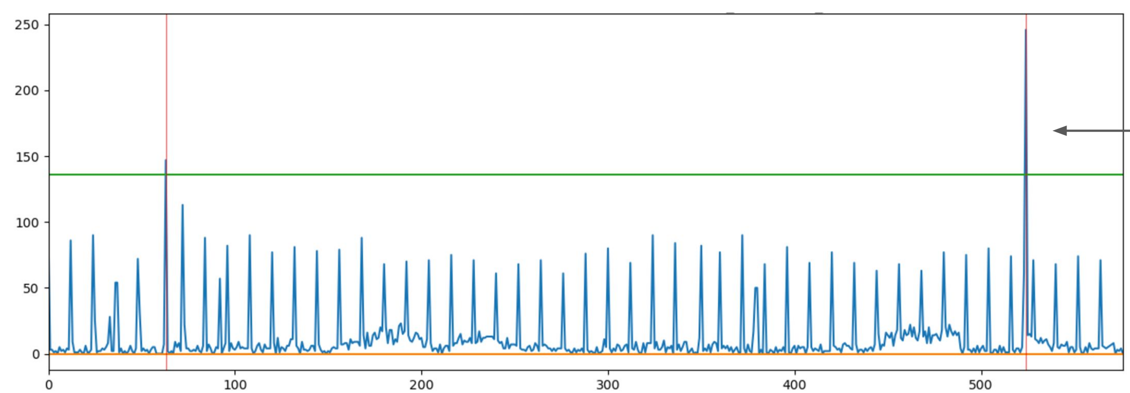
Application: Anomaly Detection





Internal dynamic **configuration down**.

High Uber Pool latency caused millions of users to drop

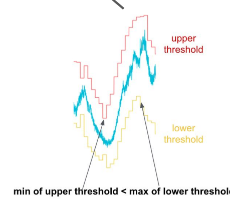


Intermittent fraud activity causes millions lost in revenue.

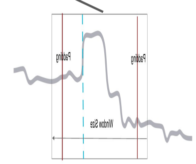
Application: Anomaly Detection - Argos

Narnia
Real-time rollout monitoring for business metrics

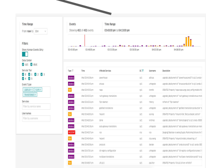
MeRL
Model Selection / Parameter tuning



F3
Seasonal Anomaly detection



JainCP
Change point detection



P3
Event data store → Root Cause tool

Rollout — Post rollout — Root cause

While we have a sophisticated anomaly detection system currently …

Application: Anomaly Detection

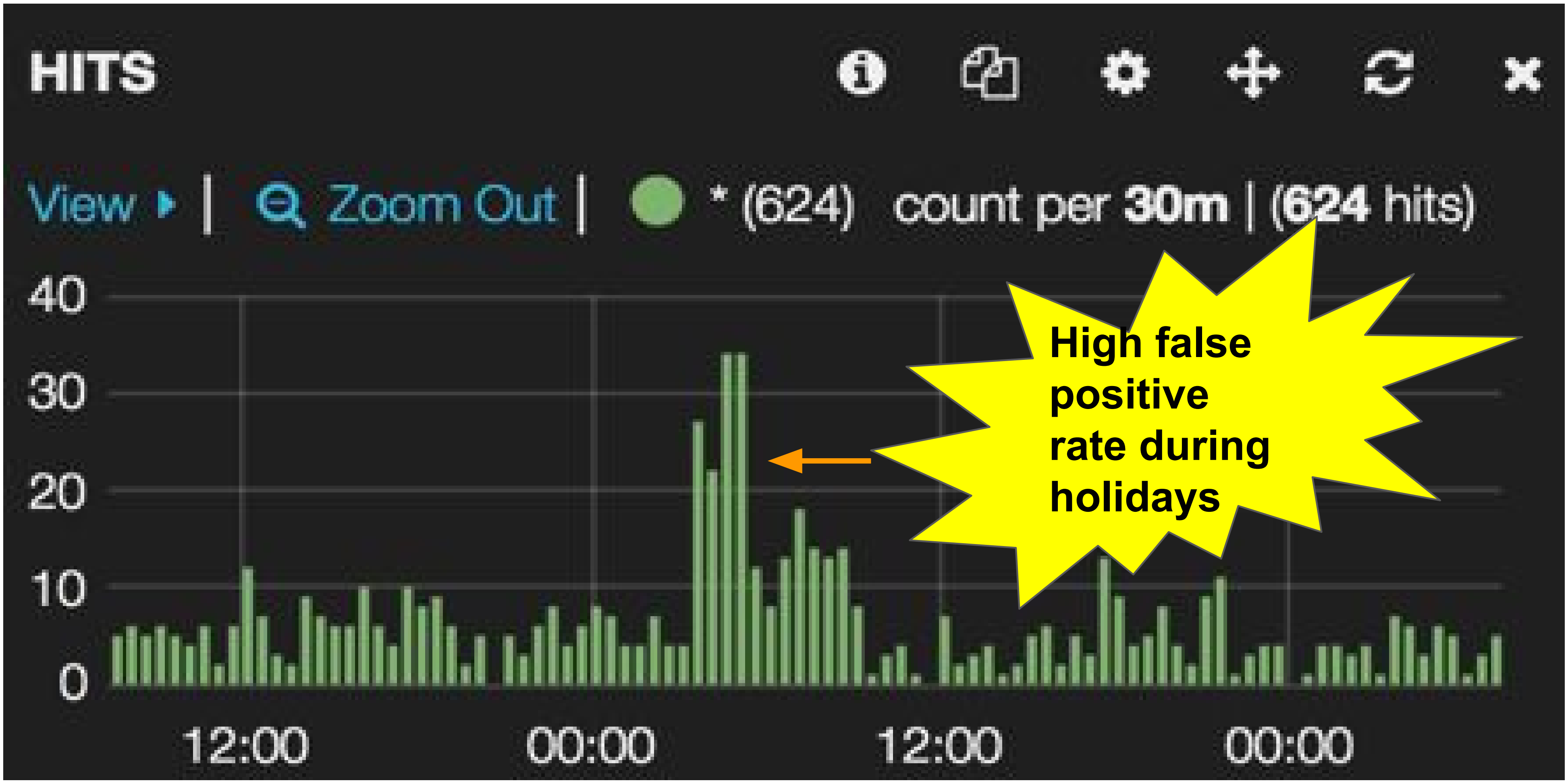
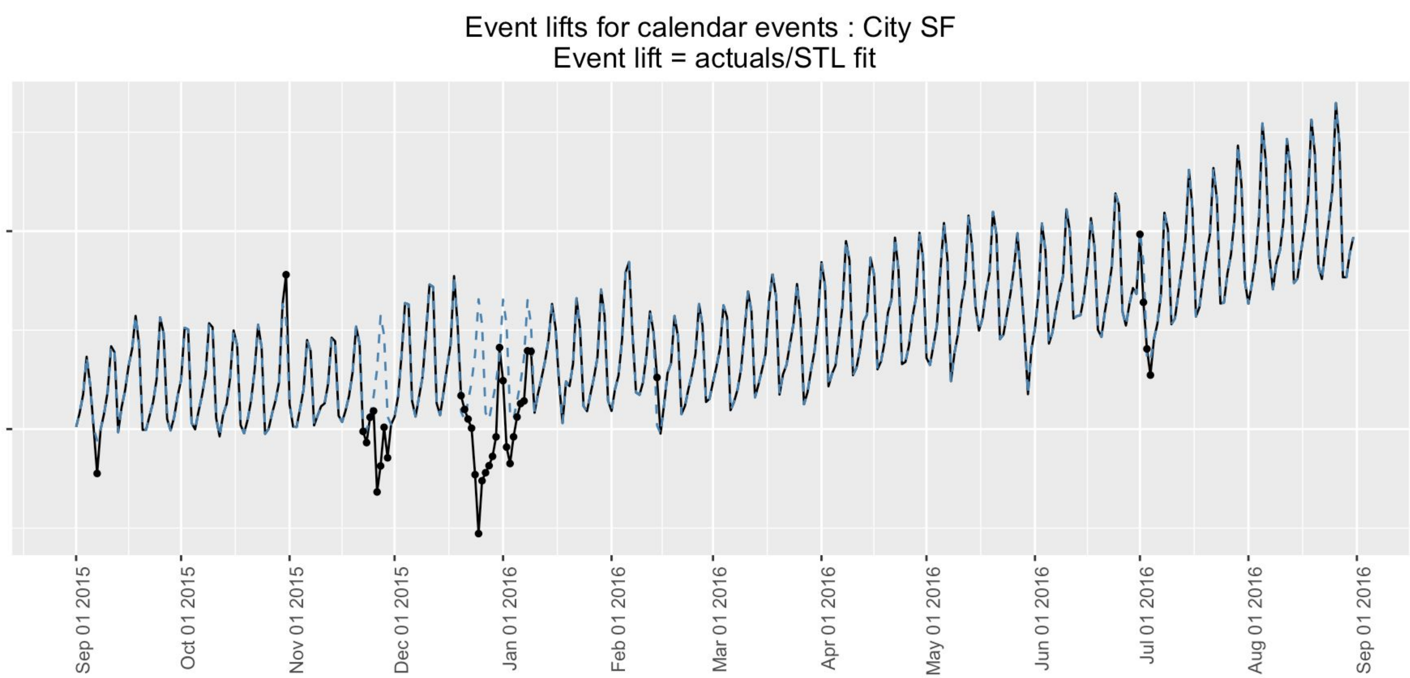


Figure: Histogram of alerts sent per 30m. Current system has a large number of false positive during holidays.

Current Solution

Two step process:

a) Backfill using a classical time-series model
b) Training a machine learning model on the residuals



Current Solution: Cons

- **System design cons**
  - Not scalable
  - Not end-to-end
- **Classical forecasting methods**
  - ARIMA: Cyclicality, Exogeneous variables
  - GARACH
  - Exponential smoothing (Holt-Winters)
  - Generalized autoregressive scoring
  - ...
- **Cons of classical approaches**
  - Low flexibility: They have difficulty adapting
  - Phase fluctuations, accelerating trends, repeated irregular patterns
  - Can happen in: Sensor data for dynamic systems, metrics, asset time-series
  - Require frequent retraining

Outline

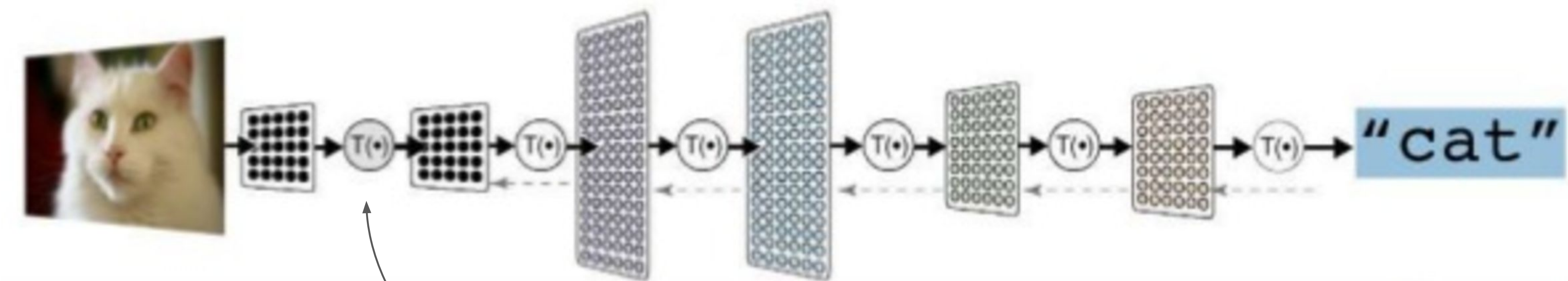
- Motivation
- **Modeling with Neural Nets**
  - Background
  - Data
  - Model
  - Inference
- Results & Discussion

Background: Neural Nets

- Easy to incorporate exogenous variables
  - External context variables
  - Other time-series (e.g., other sensor data)
  - Summary statistics (mean, max, min, std) for semi-regular telemetry
  - Less prone to errors from infrequent retraining.

Background: Neural Nets

- A powerful class of machine learning models
- Collection of simple, trainable mathematical functions
- Model reincarnation of Artificial Neural Networks



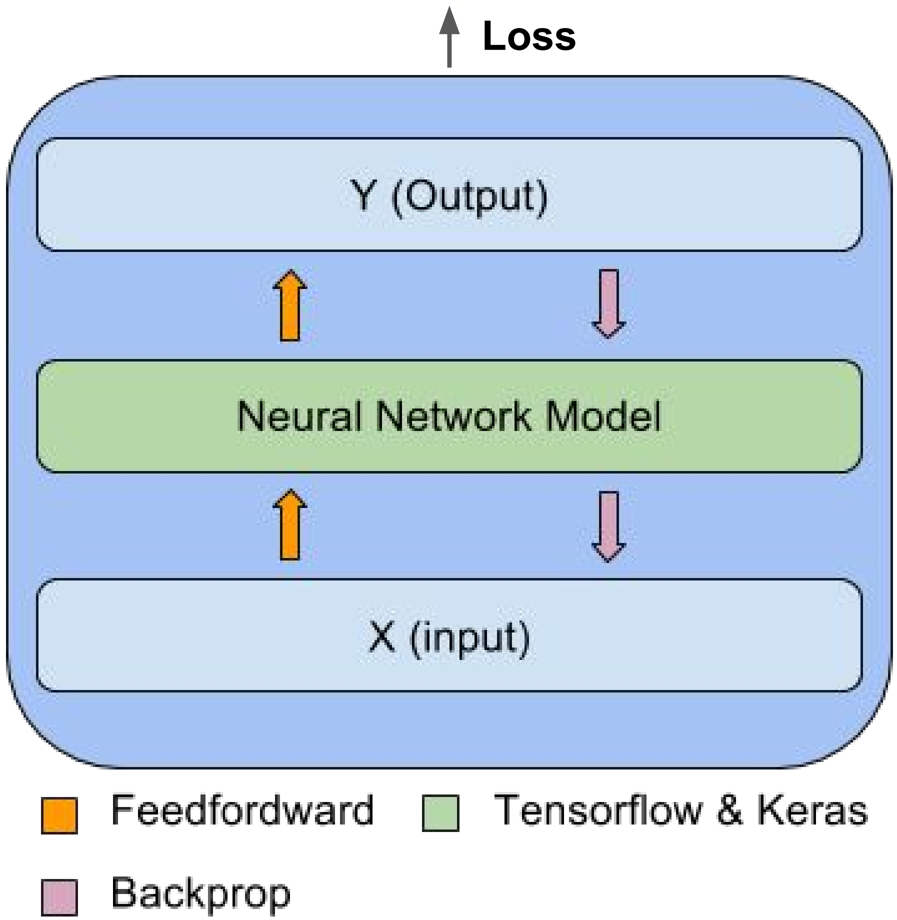
$$y = g(\vec{w} \cdot \vec{x} + b)$$

Each neuron implements a simple mathematical function. A composition of 10^6 - 10^9 of them is surprisingly powerful

Image Credit: Jeff Dean

Background: Neural Nets

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i^p - Y_i^r)^2$$



Background: Base Model

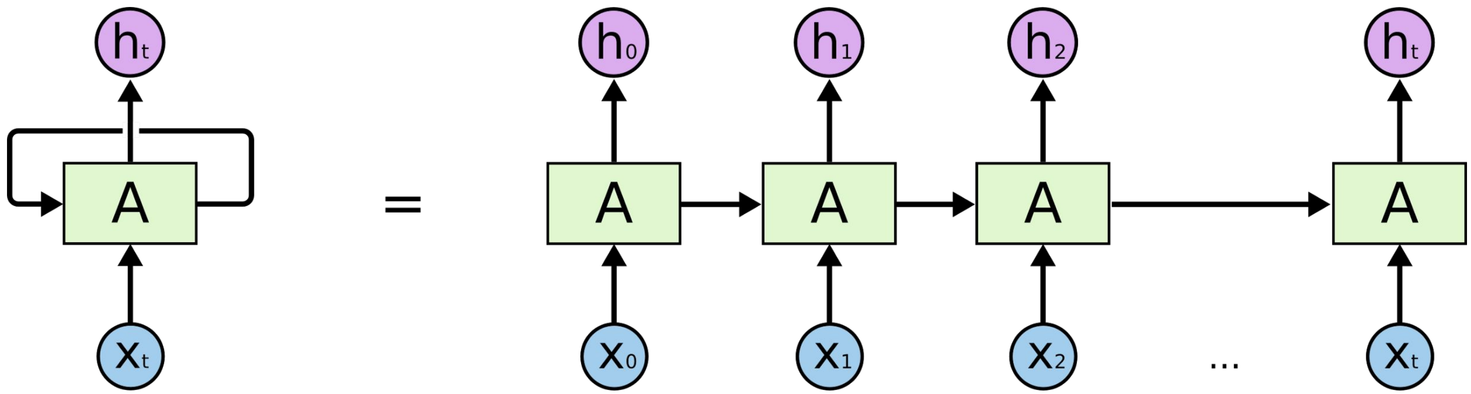
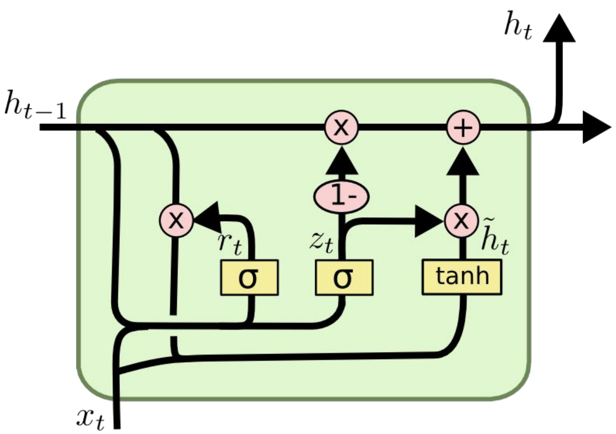


Figure: Base Recurrent Neural Network Model



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure: Variation of an LSTM Cell

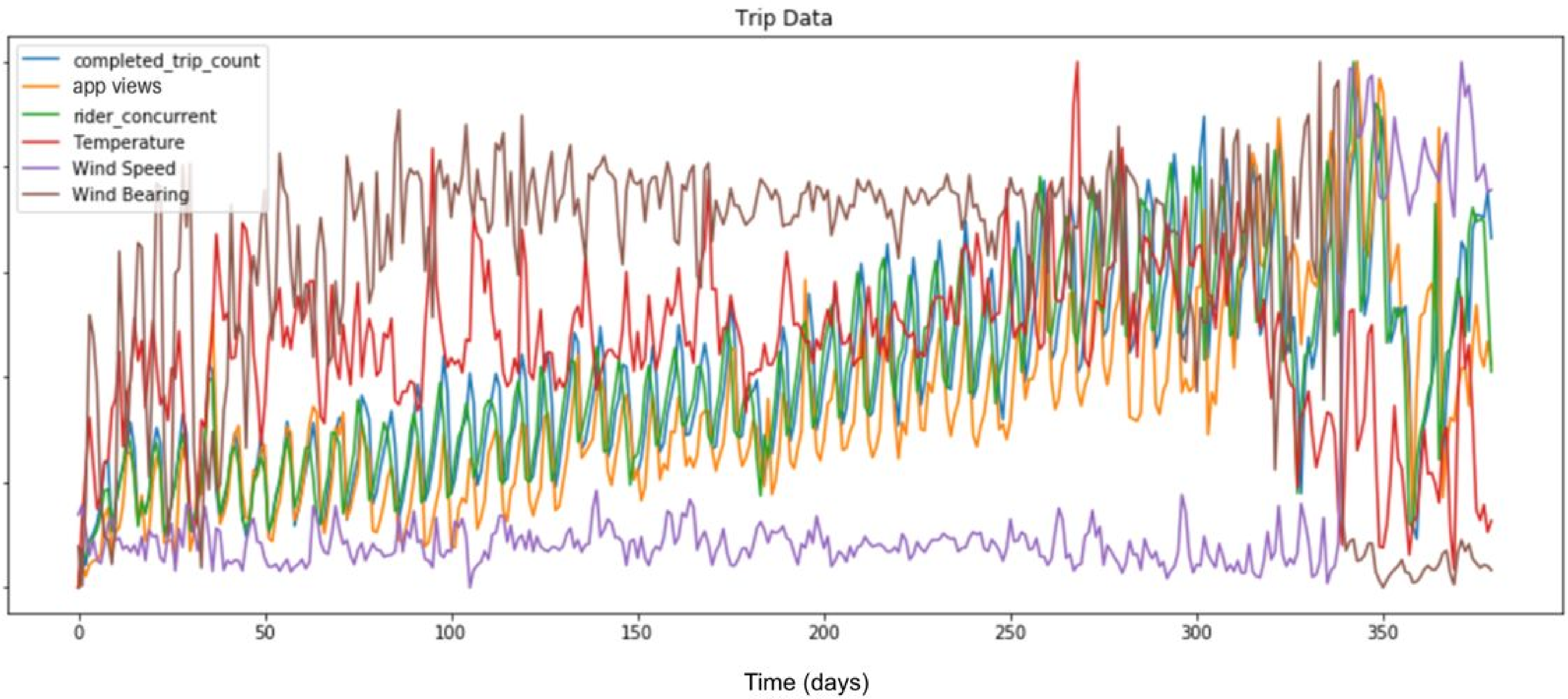
This is how we update the model with every example

Image Credit: http://colah.github.io/

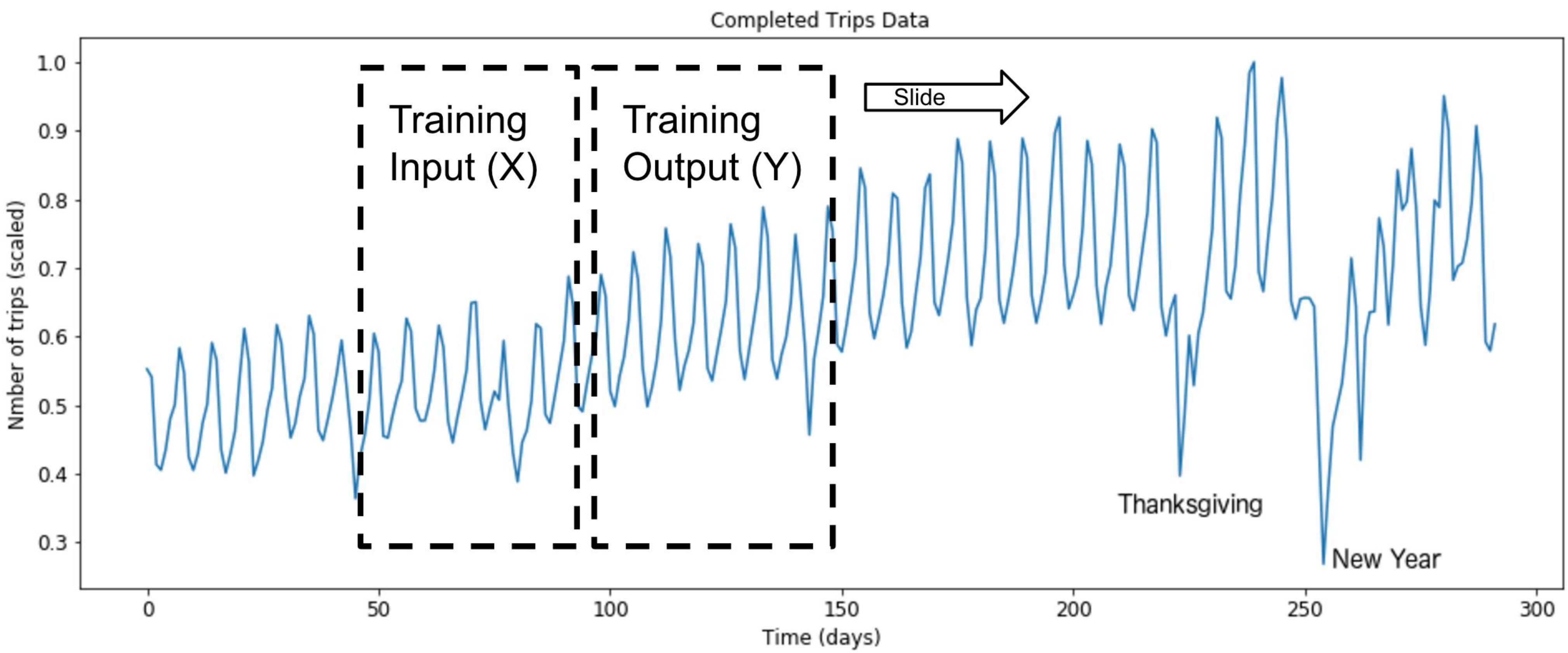
Data: External Features

- **Available data**
  - Weather
  - Trip data
  - City data
- **Challenges**
  - City to city holiday behavior is different
  - Little data for holidays
  - Sparse data for new Uber cities

Data: External Features



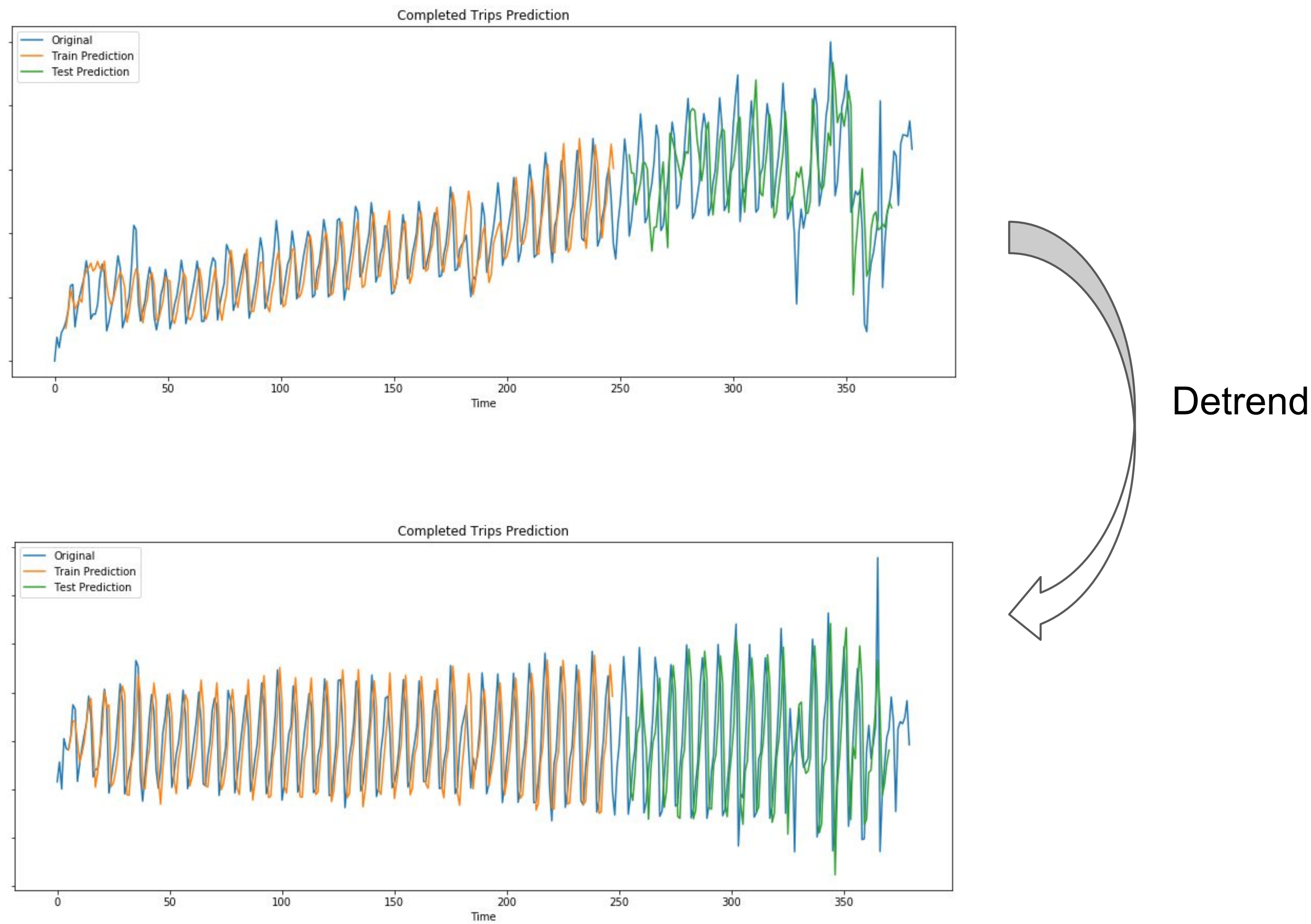
Data: Input Creation



Data: Input Creation

- **NNs learn faster and give better performance if the input variables are pre-processed before being used to train the network**
  - Exactly the same pre-processing should be done to the test set
  - `Log`
  - `scaledX = (X - minX)/(maxX - minX)`
  - Detrending, de-seasoning (using STL)

Data: Input Creation



Modeling: Base Model (Keras + Tensorflow)

```
def create_dataset(dataset, look_back = 1, forecast_horizon = 1):
    dataX, dataY = [], []
    for i in range(0, ... ):
        dataX.append(dataset[i:(i + look_back), ])
        dataY.append(dataset[i + look_back:i + look_back +
                                           forecast_horizon, 0])
    return np.array(dataX), np.array(dataY)

trainX, trainY = create_dataset(train, look_back, forecast_horizon)
testX, testY = create_dataset(test, look_back, forecast_horizon)

model = Sequential()
model.add(LSTM(64, input_dim = features, input_length = look_back ... ))
model.add(LSTM(32, ... ))
model.add(Dense(forecast_horizon))
model.compile(loss = 'mean_squared_error', optimizer = 'sgd')
model.fit(trainX, trainY, validation_data=(testX, testY))
```

Modeling: Base model pros & cons

- **Pros**
  - End-to-end
  - Infrequent retraining
  - Good performance
- **Cons**
  - Does not measure uncertainty
  - Does not scale to millions of time-series

Modeling: Uncertainty of a forecast

Decompose uncertainty into two parts:

- Model uncertainty
- Forecast Uncertainty

Modeling: Uncertainty of a forecast

$$u_y = \sqrt{(y - \hat{y})^2}$$

Uncertainty of an input given the model

$$u_y = \sqrt{\varepsilon_y^2}$$

$$y = \hat{y} + \varepsilon_{X|M,\theta} + \varepsilon_{Y|X,M,\theta}$$

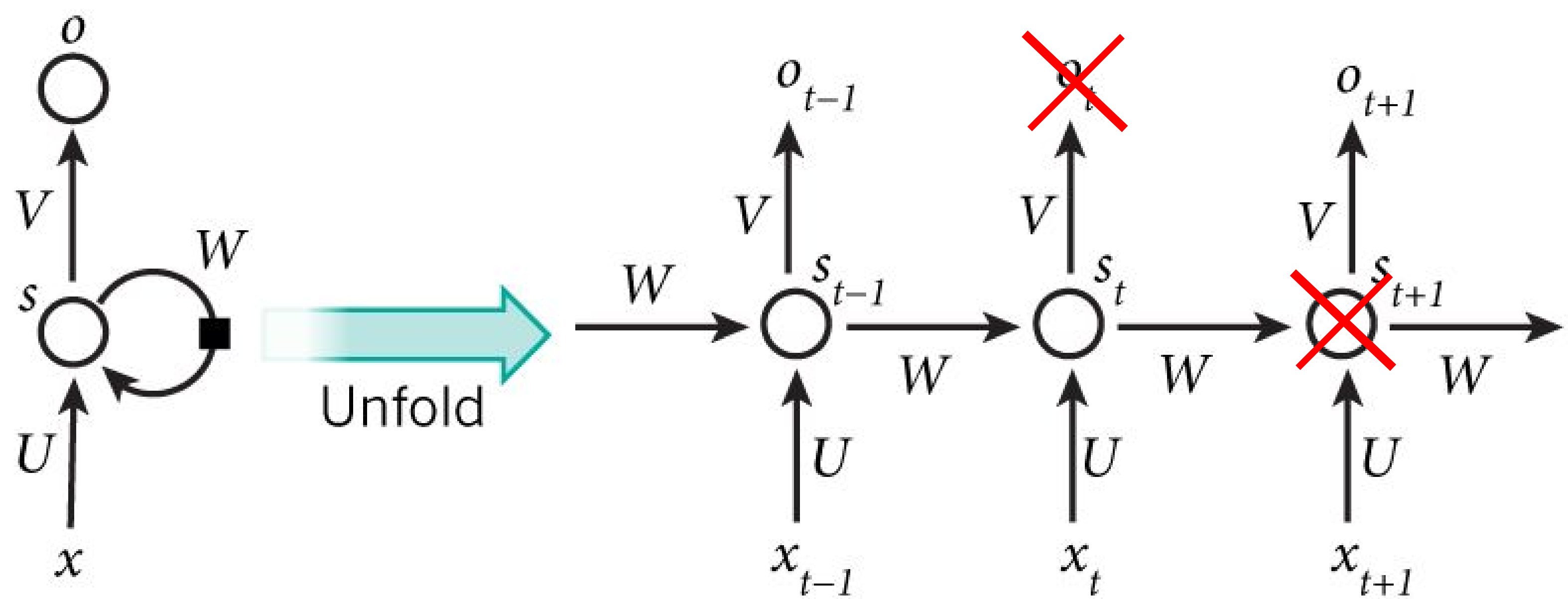
Uncertainty of the forecast given model

$$u_y^2 = \left(\varepsilon_{X|M,\theta} + \varepsilon_{Y|X,M,\theta}\right)^2$$

$$u_y^2 = \left(\varepsilon_{X|M,\theta}^2 + \varepsilon_{Y|X,M,\theta}^2 + 2(\varepsilon_{X|M,\theta}\varepsilon_{Y|X,M,\theta})\right)$$

$$u_y = \sqrt{\varepsilon_{X|M,\theta}^2 + \varepsilon_{Y|X,M,\theta}^2 + 2(\varepsilon_{X|M,\theta}\varepsilon_{Y|X,M,\theta})}$$

Modeling: Empirical Uncertainty Using Dropout



- The longer the time-series, the more useful dropout is.
- Dropout on the weights doesn’t work. Use dropout on activations.
- Regularization on the activation did not show improvement. Use regularization on the weights.

Modeling: Uncertainty

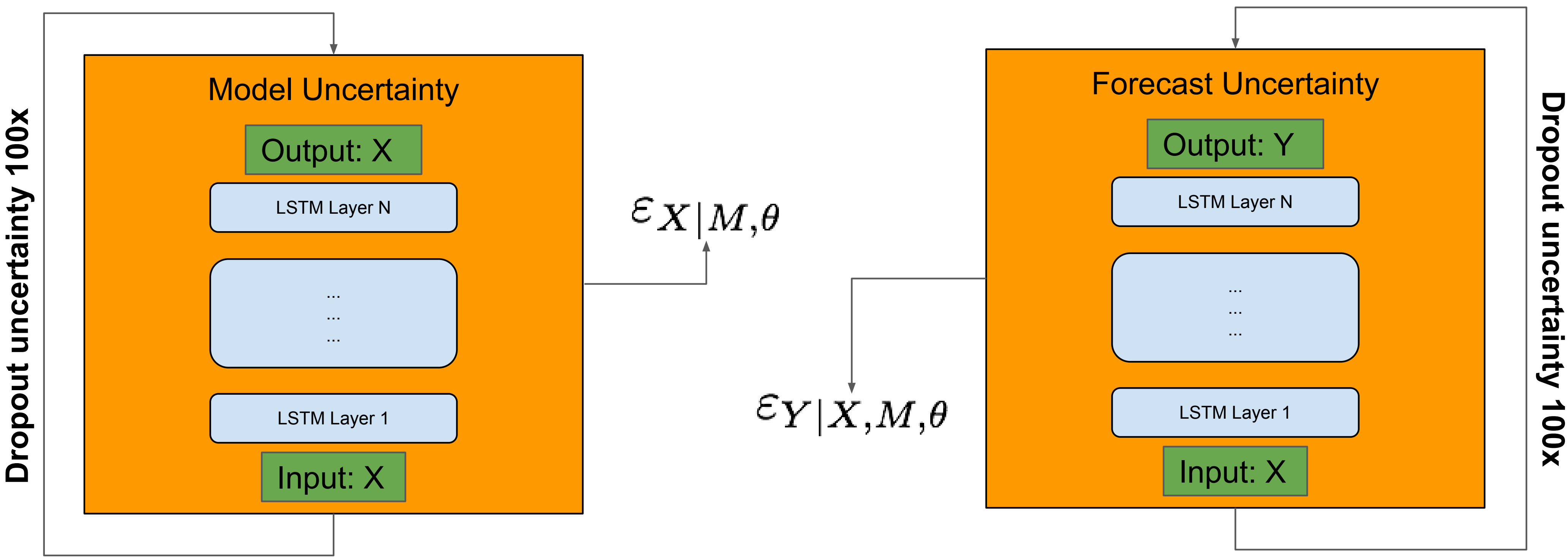
Bayesian uncertainty

```
vals = []
for r in range(100):
  vals.append(model.eval(x, dropout = normal(0,1)))
mean = np.mean(vals)
var = np.var(vals)
```

Dropout is normally done during training.
To make it work during testing, add this:

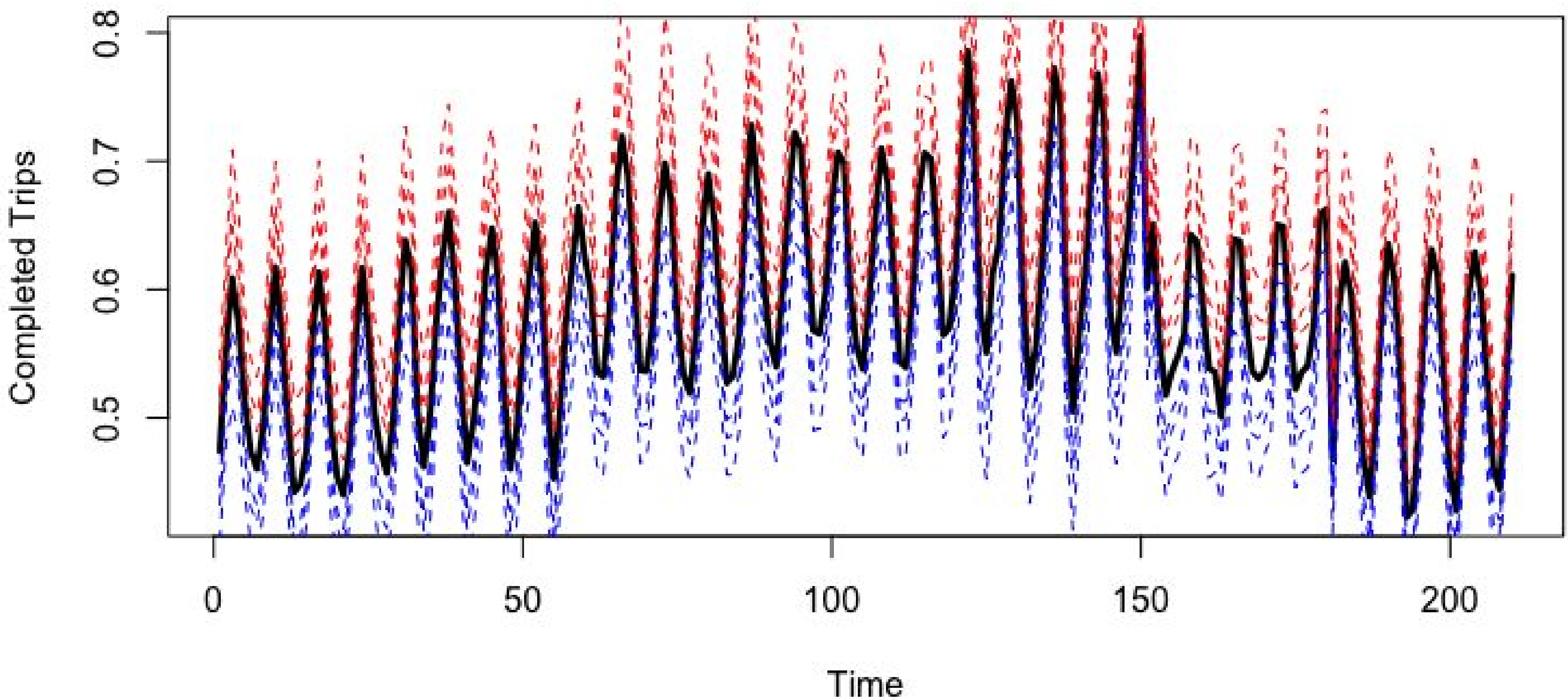
```
model.add(Lambda(lambda x: K.dropout(x, level = normal(0,1))))
```

Modeling: Model and Forecast Uncertainty



Note: Term $2(\varepsilon_{X|M,\theta}\varepsilon_{Y|X,M,\theta})$ goes away assuming model and forecast error correlation is very high.

Modeling: Uncertainty Example



Forecast with uncertainty. Lag = 7 days, Forecast Horizon = 7 days

Modeling: Scaling to millions of time-series

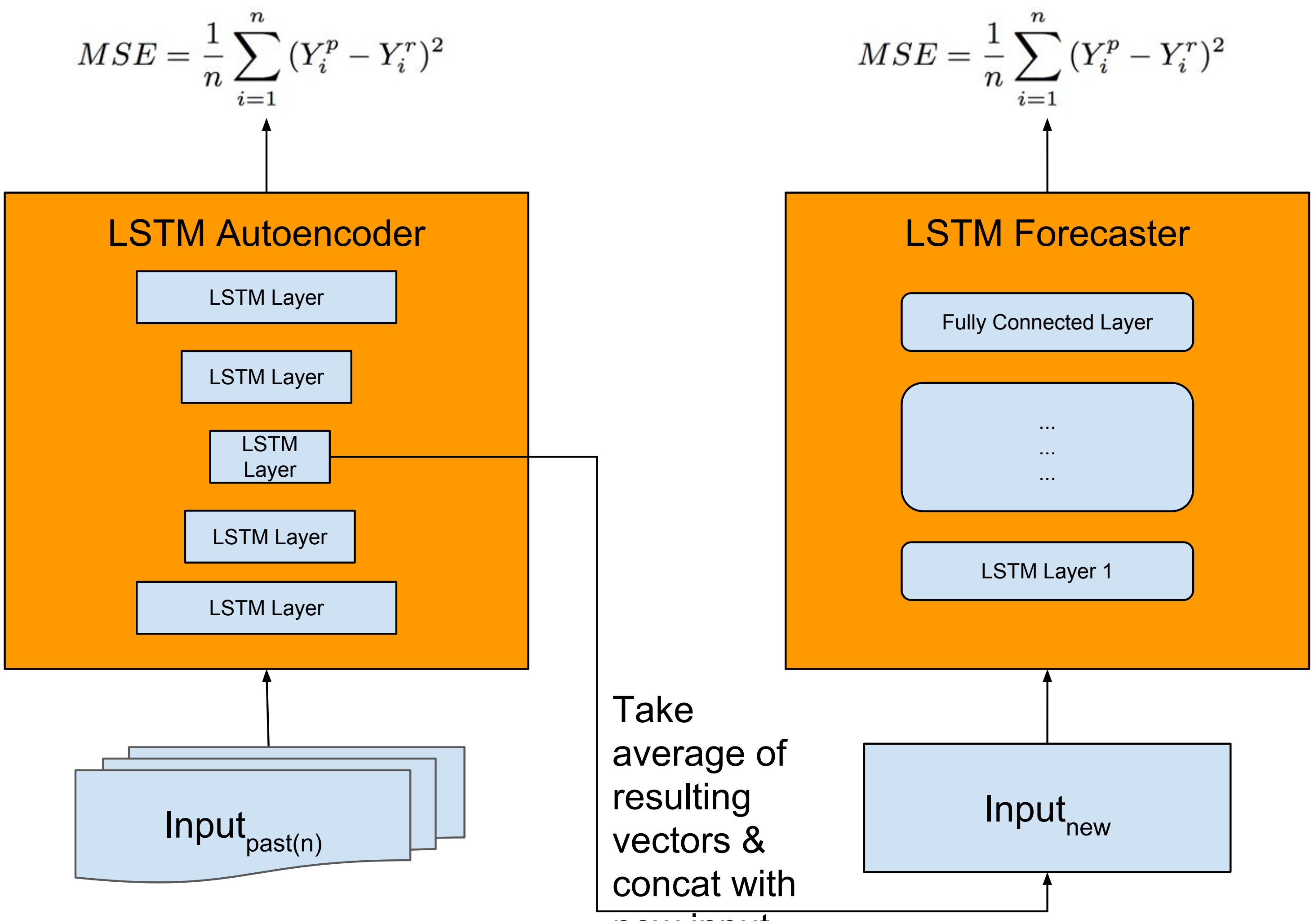
- Training a separate neural network for every city is infeasible
- Manually encoding city level features is prone to error
- How can we automate feature extraction to support a single model?

Modeling: Scaling to millions of time-series

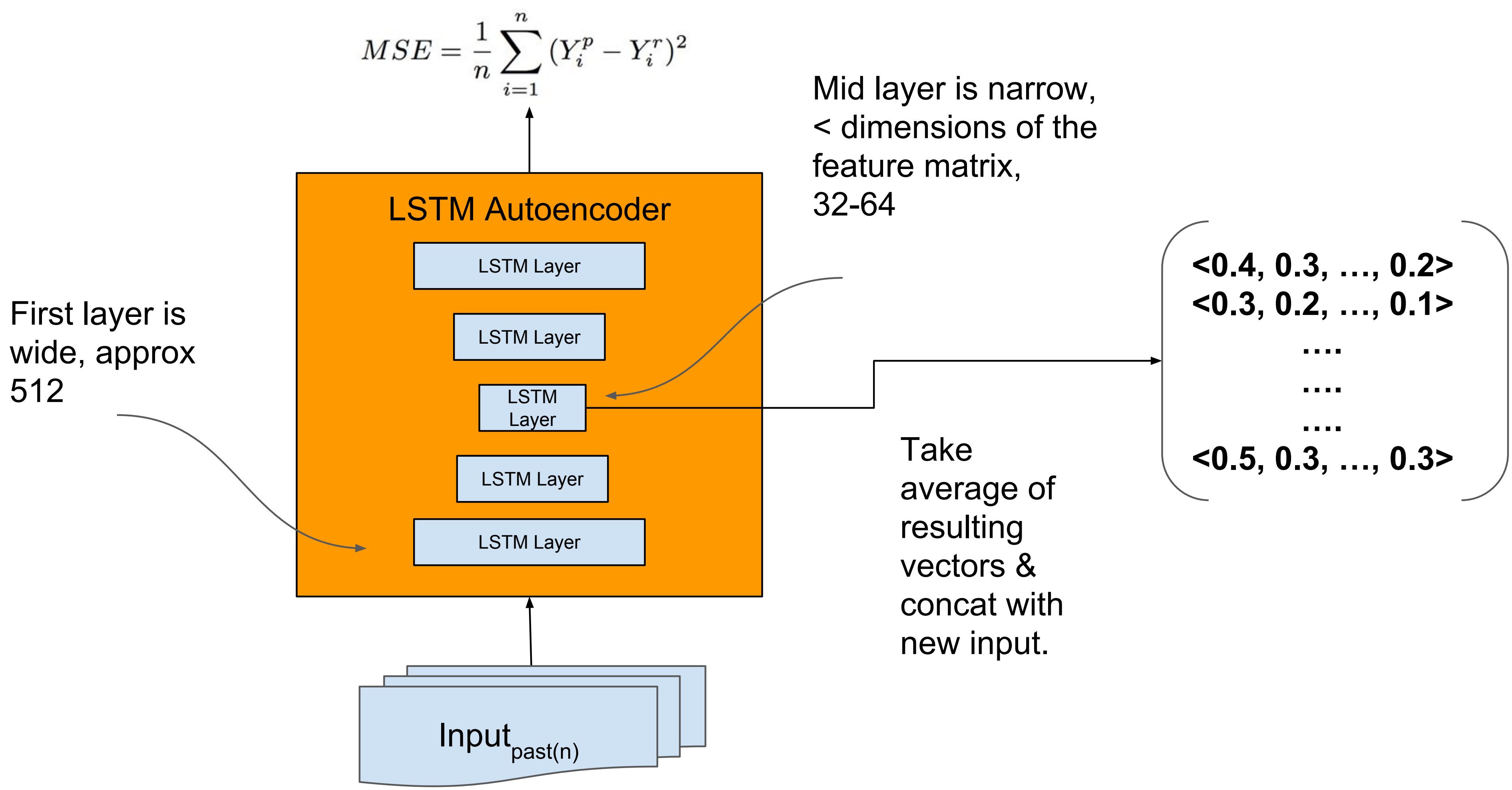
Feature	Description
Mean	Mean.
Var	Variance.
ACF1	First order of autocorrelation.
Trend	Strength of trend.
Linearity	Strength of linearity.
Curvature	Strength of curvature
Season	Strength of seasonality.
Peak	Strength of peaks.
Trough	Strength of trough.
Entropy	Spectral entropy.
Lumpiness	Changing variance in remainder.
Spikiness	Strength of spikiness
Lshift	Level shift using rolling window.
Vchange	Variance change.
Fspots	Flat spots using disretization.
Cpoints	The number of crossing points.
KLscore	Kullback-Leibler score.
Change.idx	Index of the maximum KL score.

Figure: Some manually developed time-series features we extracted in our prior work.

Modeling: Scaling to millions of time-series

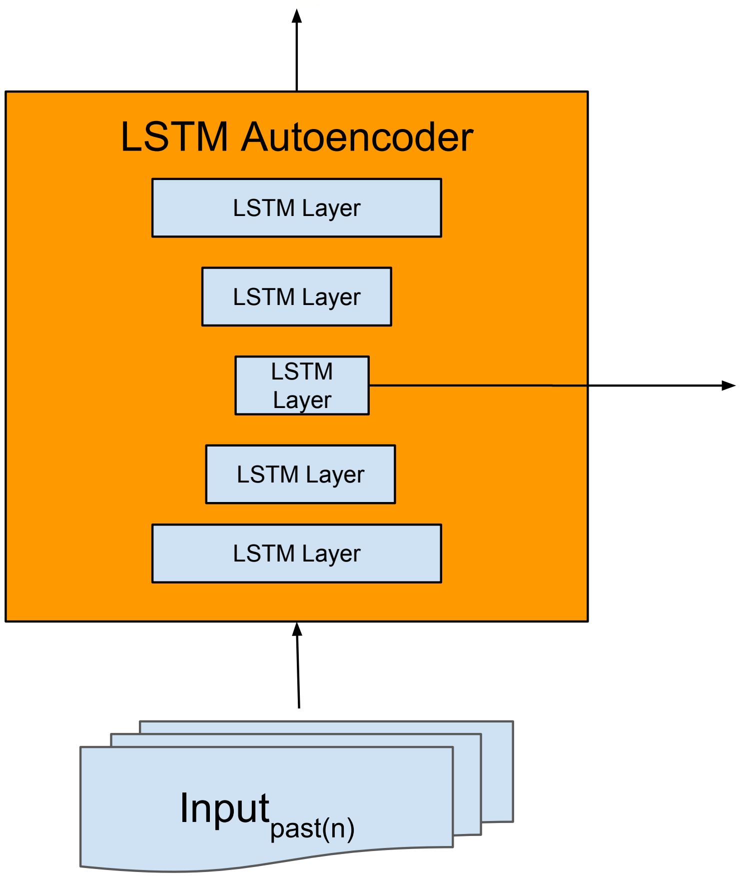


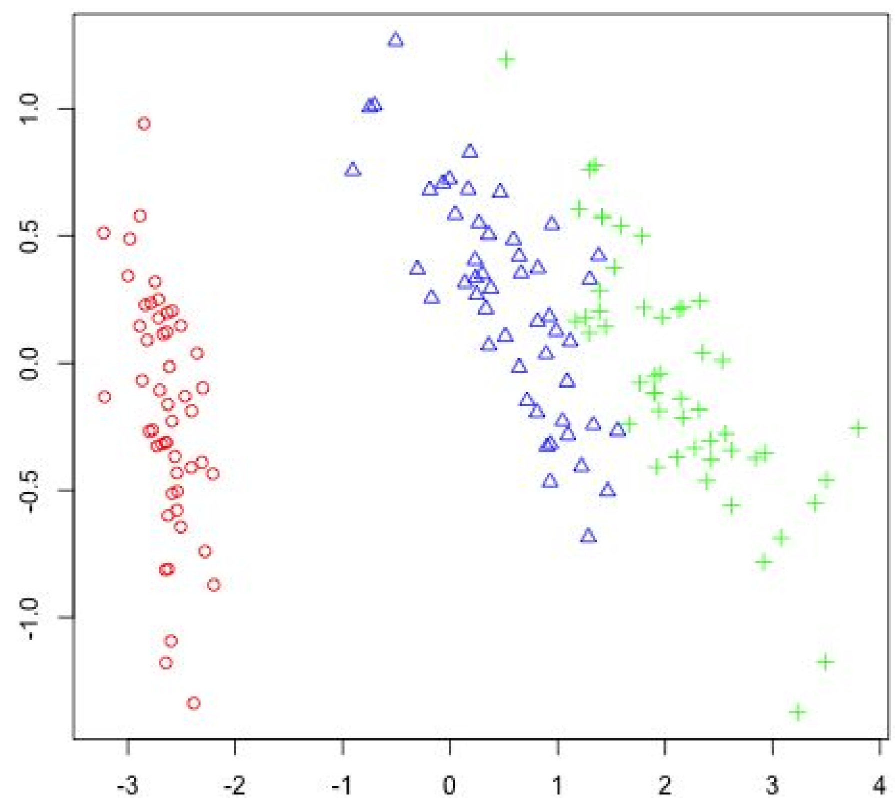
Modeling: Scaling to millions of time-series



Modeling: Scaling to millions of time-series

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i^p - Y_i^r)^2$$

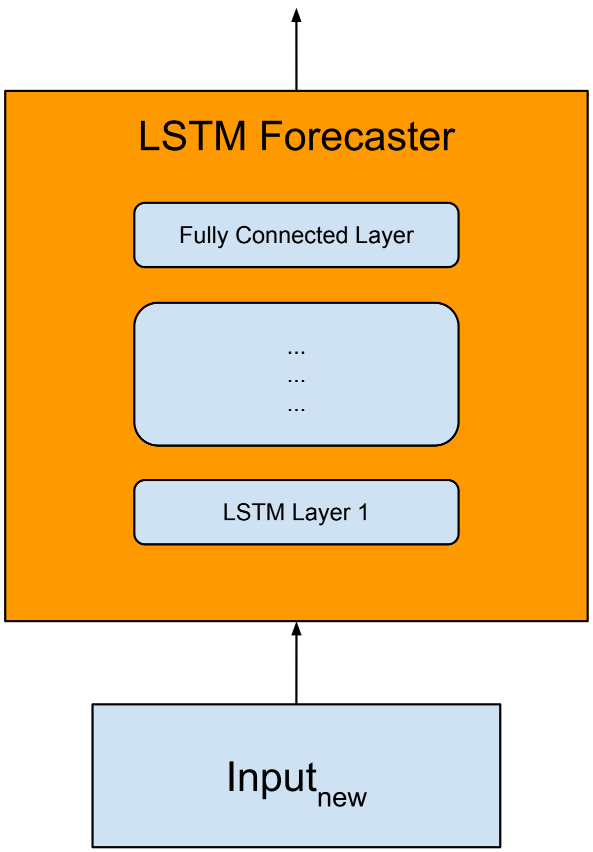




One can plot the extracted features in a 2D space to visualize the time-series. A deeper study of this is part of our future work

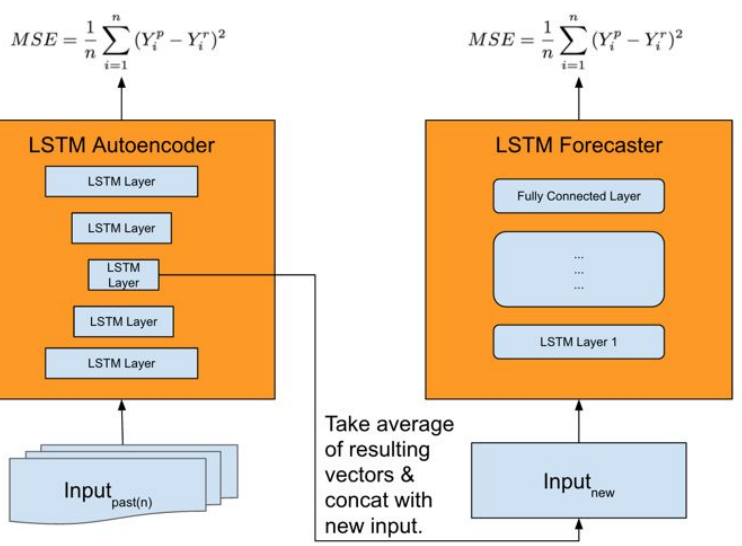
Modeling: Scaling to millions of time-series

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i^p - Y_i^r)^2$$



- First layer is wide, approx 512
- For mid-layers we use depth of 4 with polynomially decreasing widths
- Last layer is a fully connected layer with size = forecast
- No retraining is required to forecast any part of the time-series given the immediate past.

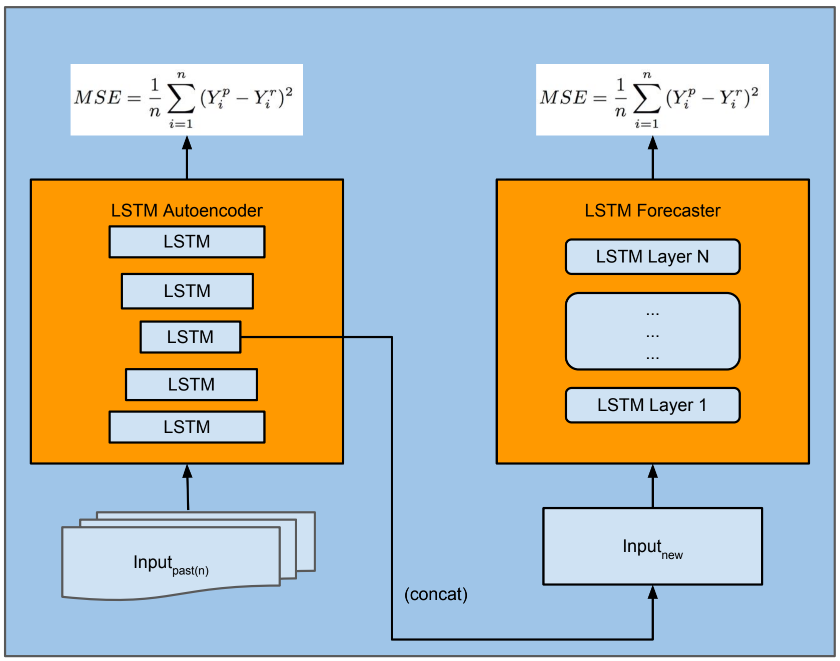
Modeling: Scaling to millions of time-series



- Different ways to combine feature extractor and forecaster:
  - Extend the input of forecasting module
  - Extend the depth of the forecasting module
  - Separate modules give best performance
- This architecture allows for a ‘generic forecasting machine’

Inference: at Uber Scale

- Want to avoid 3rd party dependencies (e.g., Tensorflow, Keras [**Python**])
- Export weights and model architecture and execute natively in **Go**
- Applicable to a generic time-series
  - Vision - Let other teams use the model and adapt to specialized use cases with add-on layers if necessary



Export weights and operations

```
func (tt timeTravel) nnForecast(x []float64) ([]float64, error) {
  layerWeights := tt.layerWeights
  layerBias := tt.layerBias
  var h1 []float64
  if layerWeights == nil || layerBias == nil {
    return h1, errNNForecast
  }
  prev := x

  for i := range layerWeights {
    h1, _ = stats.ElemSum(stats.DotProduct(prev,
                          layerWeights[i]), layerBias[i])
    prev = h1
  }
  return h1, nil
}
```

Train, infrequently, using Tensorflow, Keras, GPUs

Export weights and operations to native Go code

Outline

- Motivation
- Modeling with Neural Nets
- **Results & Discussion**
  - Forecasting and Special Event Performance
  - Lessons learned

Results: Experiments & Methodology

- Internal and public datasets
- Three years of data from 10 cities
- Target variable is completed trips.
- Records for holidays, weather, eyeballs.
- Forecast is done one week ahead.
- Measure SMAPE: $\text{SMAPE} = \frac{100\%}{n} \sum_{t=1}^{n} \frac{|F_t - A_t|}{(|A_t| + |F_t|)/2}$

Results: Forecasting (Uber Data)

Query	Previous Model	Described Neural Network
Query #1	10.60	13.05
Query #2	23.23	22.60
Query #3	48.57	18.23
Query #4	47.41	26.35
Query #5	19.40	16.87
Query #6	19.25	22.65
Query #7	21.35	19.78
Query #8	39.31	36.31
Query #9	22.11	21.01
Mean	**32.44**	**26.66**
Median	**25.42**	**22.62**

Table: SMAPE Comparison on sample queries

- Single neural network model is constructed compared to per query training requirement for the previous model.

Results: Forecasting (Public dataset - M3 Monthly)

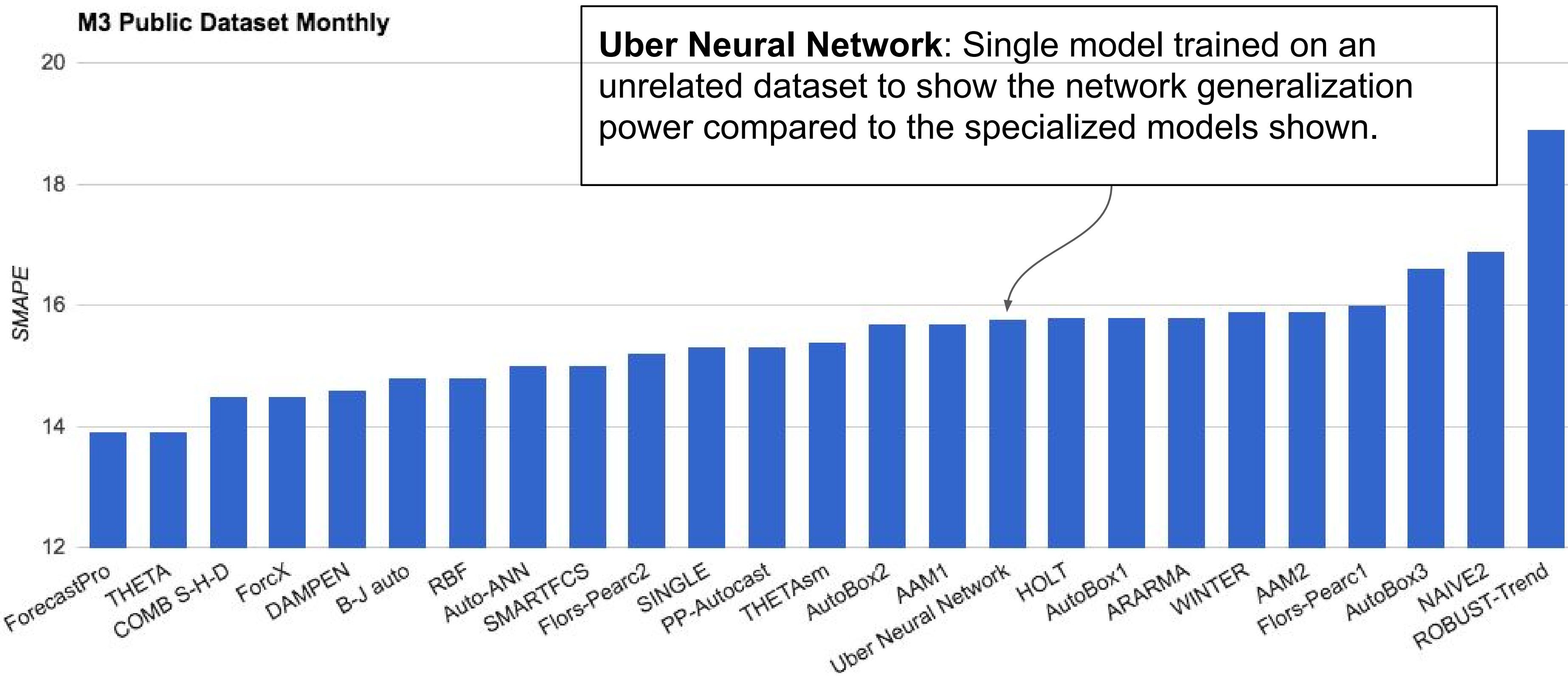


Table: Experiment on the public M3 Monthly Dataset showing the generalization power of the Uber Neural Network

Results: Scalability

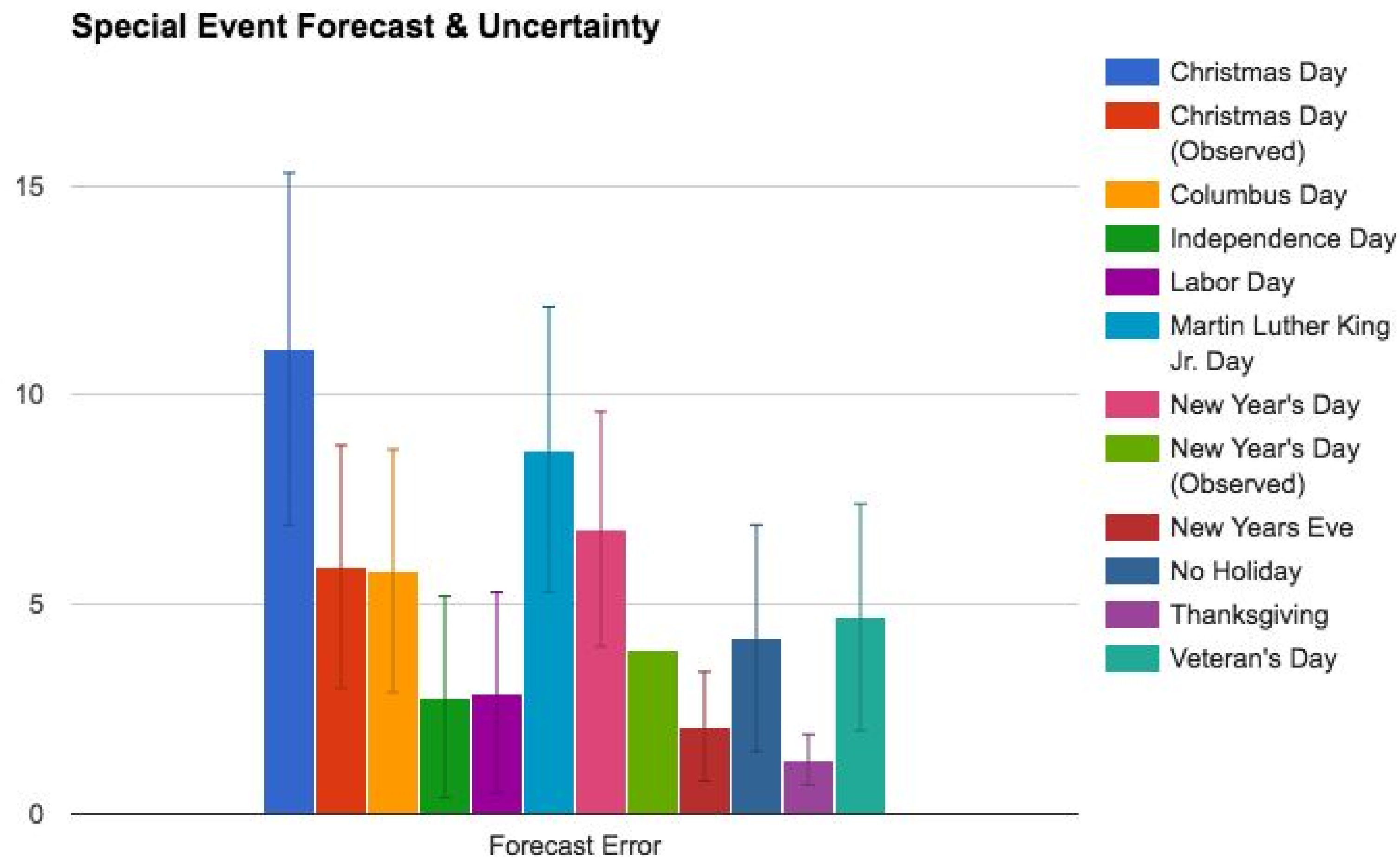
Model	**Training Time**	**Inference Time**
Neural Network	O(1)	O(M)
Prod Best	O(N*L)	O(1)

Table: Scalability comparison, where **N** is the number of time-series, **L** is their length and **M** is Neural Network's set up stage. **M << L**.

Results: Special Event Prediction Performance (SMAPE)

Feature	Described Neural Network	Previous Model
Christmas Day	11.1	29.2
MLK	8.7	20.2
Independence Day	2.8	17.6
Labor Day	2.9	6.9
New Year’s Day	6.8	7.8
Veteran’s Day	4.7	8.9

Results: Special Event Use Case Forecast Errors (SMAPE)



Results: Example of a forecast (Testing)

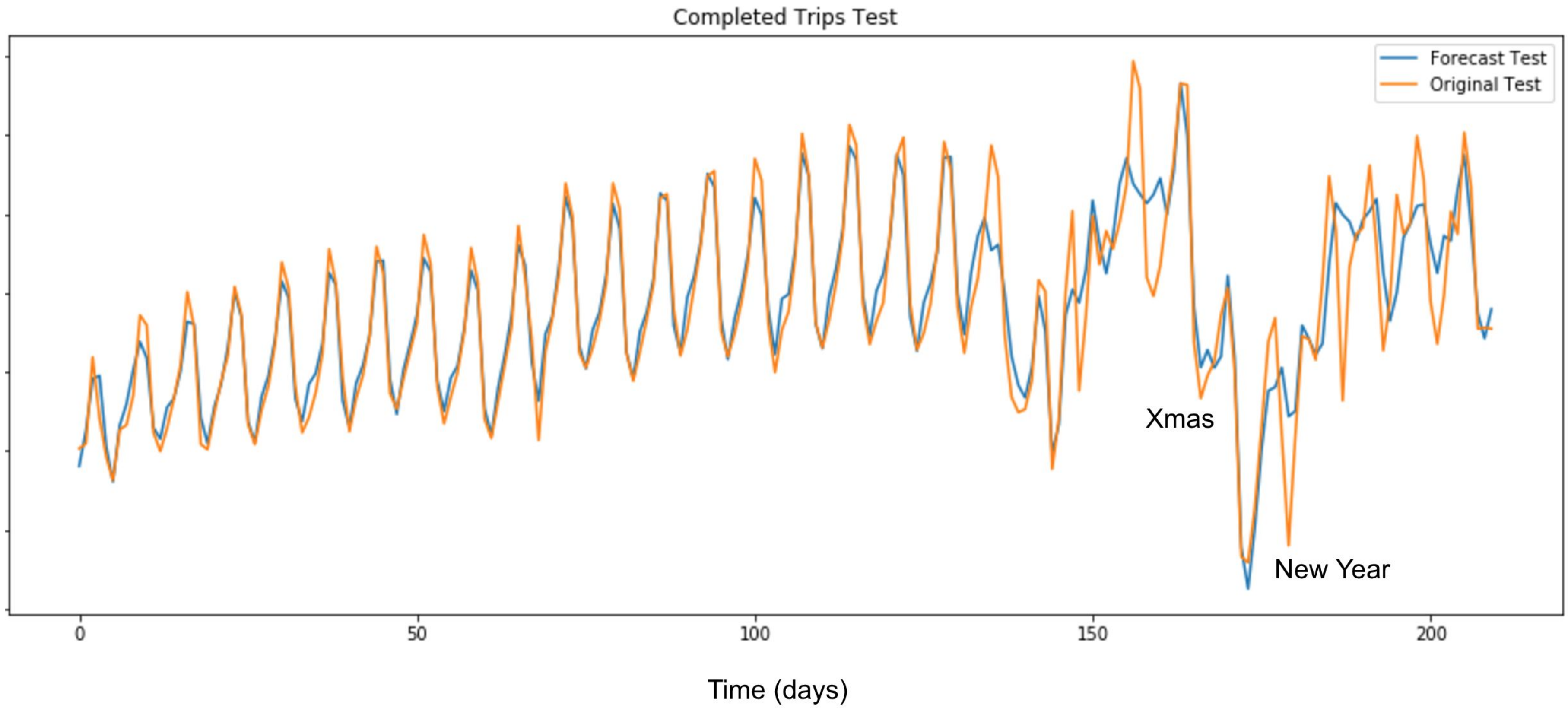


Figure: Testing time-series and forecast.

Results: Lessons learned

Time-Series Type	**RNN Performance**	**Classical Model Performance**
Short Time-Series	Not enough data to train.	Symbolic Regression, HMMs perform well.
Long Time-Series	Able to optimize.	Classical Model Performance is Equivalent to RNN.
Multivariate Short Time-Series	Not enough data. While RNNs able to represent any function, need a lot of data.	Multi-varaite regression, Symbolic regression, Hierarchical forecasting perform well.
Multivariate Long Time-Series	RNN is able to model nonlinear relationships among features. Computationally efficient. Automatic feature selection.	Computation efficiency may not be optimal. Feature selection challenging.

Results: Lessons learned

- **Classical models are best for:**
  - Short or unrelated time-series
  - Known state of world
- **Neural Network is best for:**
  - A lot of time-series
  - Long time-series
  - Hidden interactions
  - Explanation is not important
- **Future work**
  - Model debugging using uncertainty for special events.
  - Work towards a general forecasting machine
    - To be used as a building block in a larger forecasting model (e.g., similar to the ImageNet model)
- **See more**
  - Eng blog
  - ICML paper under review

Results: Anomaly Detection

Uber Neural Network

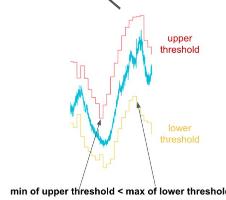
Live on Argos & uMonitor

Narnia

Real-time rollout monitoring for business metrics

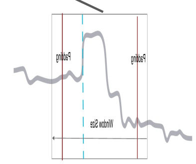
MeRL

Model Selection / Parameter tuning



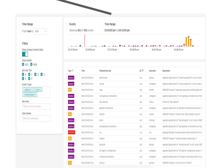
F3

Seasonal Anomaly detection



JainCP

Change point detection



P3

Event data store → Root Cause tool

Rollout // Post rollout // Root cause

We are pleased to introduce a Neural Network into Argos suite of models.

Acknowledgements

Jason Yosinksi, Uber AI Labs

Li Erran Li, ATC

Santhosh Shanmugam, IDS, Delphi

Eric McMillen, IDS, Delphi

Slawomir Smyl, IDS, Delphi

Calvin Worsnup, IDS, Delphi

Thank you

eng.uber.com
github.com/uber

UBER