Dilated Neural Networks for Time Series Forecasting

Chenhui Hu

Data Scientist
Microsoft Cloud & Enterprise

Microsoft

Agenda

- Overview of Time Series Forecasting Methods
- Foundations of Dilated Convolutional Neural Networks
- Application to Retail Demand Forecasting

Agenda

- Overview of Time Series Forecasting Methods
- Foundations of Dilated Convolutional Neural Networks
- Application to Retail Demand Forecasting

Overview

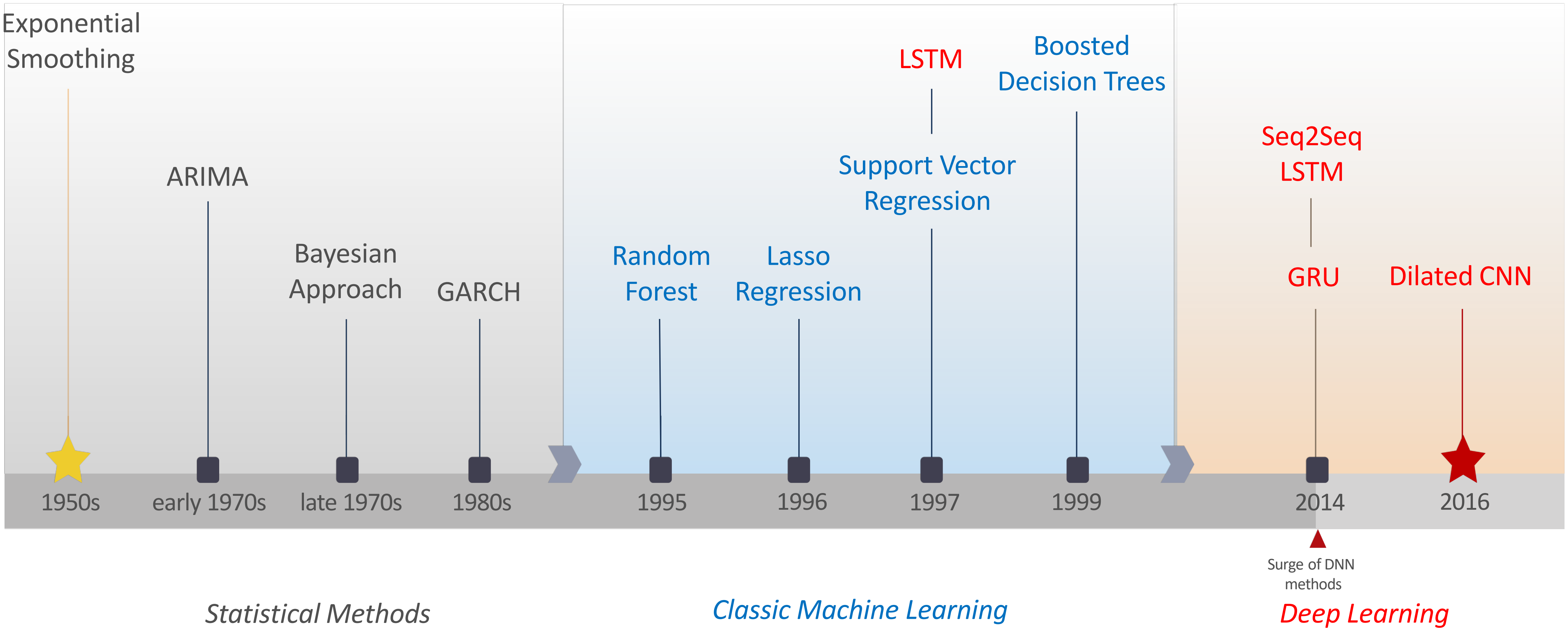
Time Series Forecasting Applications

retail sales forecasting

financial forecasting

web traffic forecasting

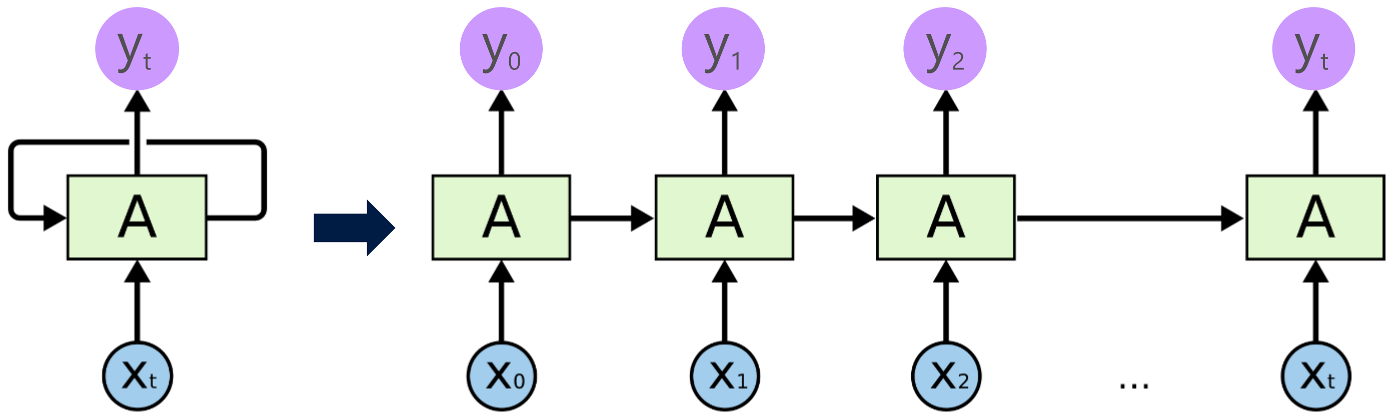
Representative Methods



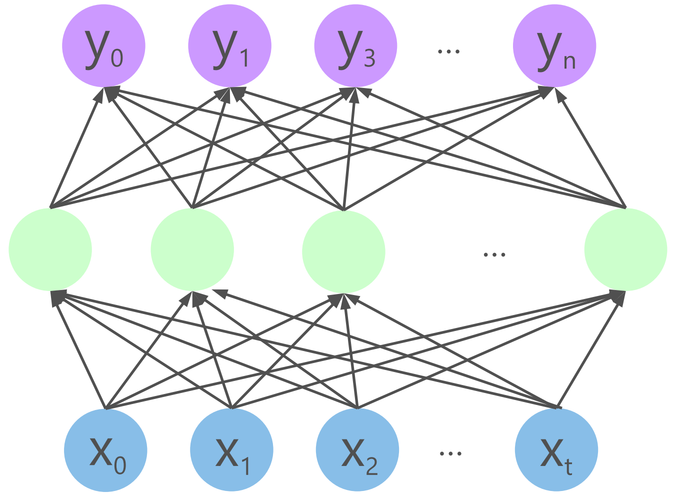
J. Gooijer and R. Hyndman. 25 Years of Time Series Forecasting.

Recurrent NN

- RNN models *the order* of the data explicitly
- Long Short-Term Memory (LSTM) network models long-range dependencies



unrolled Recurrent Neural Network (RNN)

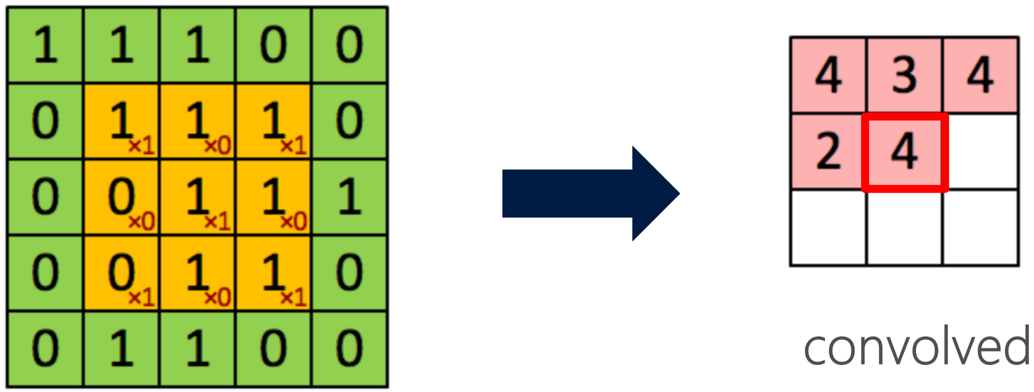


multi-layer perceptron

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Convolutional NN

- Success of CNN in CV and recently in NLP
  - Outperforms human in many computer vision tasks
  - Achieves state-of-the-art accuracy in audio synthesis & machine translation
- MLP with convolution layers



image

convolved feature

2D convolution as a linear combination of pixel values in a neighborhood

RNN vs. CNN

- RNN
  - Has been default DL choice for sequence modeling
  - Can learn long-range dependencies effectively
  - But training is usually slower compared with CNN

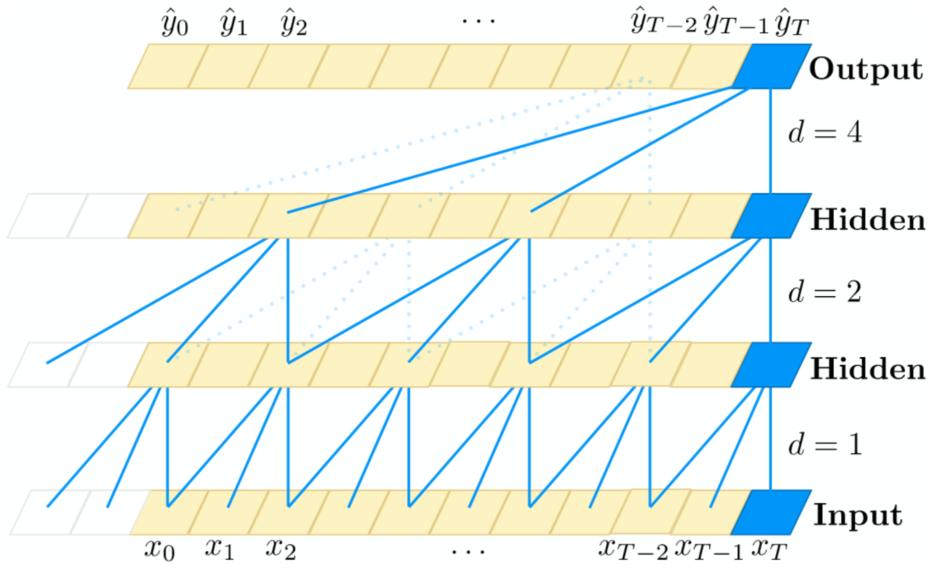
- CNN
  - Lower level of model complexity
  - Easy to parallelize the computation
  - Dilated CNN can outperform RNN in sequence modeling

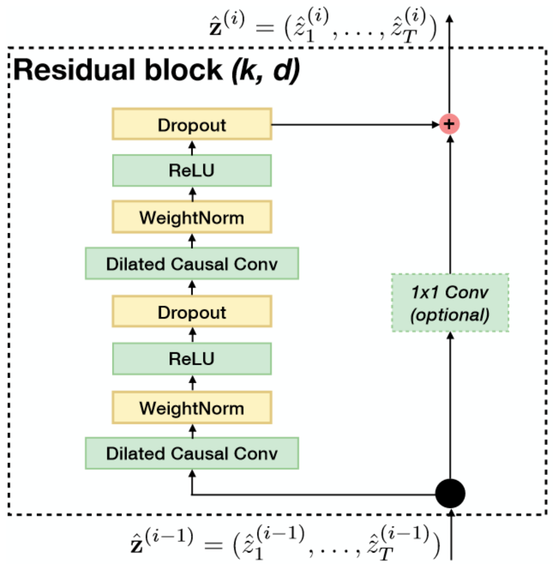
Agenda

- Overview of Time Series Forecasting Methods
- Foundations of Dilated Convolutional Neural Networks
- Application to Retail Demand Forecasting

Dilated CNN Overview

- CNN with dilated causal conv. layers
  - Basic setup: dilated conv. layers + output layer
  - Advanced: skip connections, dropout, batch normalization, gated activation units, ...
- Examples
  - WaveNet for audio synthesis
  - Temporal CNN for sequence modeling





S. Bai, J. Kolter, and V. Koltun. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling.

1-D Convolution for Time Series

	Feature 1	Feature 2	Feature 3	Feature 4
Timestamp 1	0.4	0.8	0.5	0.7
Timestamp 2	0.7	0.3	0.6	0.7
Timestamp 3	0.8	0.4	0.9	0.2
Timestamp 4	0.1	0.5	0.8	0.5
Timestamp 5	0.4	0.8	0.7	0.1
Timestamp 6	0.3	0.0	0.2	0.6
Timestamp 7	0.2	0.5	0.8	0.2

Filter

1	2	-3	1
-1	1	-2	2
2	2	1	3

kernel size = 3

$$4.9 = 0.4\times1 + 0.8\times2 + 0.5\times(-3) + 0.7\times1 + 0.7\times(-1) + 0.3\times1 + 0.6\times(-2) + 0.7\times2 + 0.8\times2 + 0.4\times2 + 0.9\times1 + 0.2\times3$$

Timestamp 1, 2, 3	4.9
Timestamp 2, 3, 4	1.9
Timestamp 3, 4, 5	2.3
Timestamp 4, 5, 6	1.0
Timestamp 5, 6, 7	3.3

1-D Convolution for Time Series

	Feature 1	Feature 2	Feature 3	Feature 4
Timestamp 1	0.4	0.8	0.5	0.7
Timestamp 2	0.7	0.3	0.6	0.7
Timestamp 3	0.8	0.4	0.9	0.2
Timestamp 4	0.1	0.5	0.8	0.5
Timestamp 5	0.4	0.8	0.7	0.1
Timestamp 6	0.3	0.0	0.2	0.6
Timestamp 7	0.2	0.5	0.8	0.2

Filter

1	2	-3	1
-1	1	-2	2
2	2	1	3

kernel size = 3

Timestamp 1, 2, 3	4.9
Timestamp 2, 3, 4	1.9
Timestamp 3, 4, 5	2.3
Timestamp 4, 5, 6	1.0
Timestamp 5, 6, 7	3.3

1-D Convolution with Multiple Filters

	Feature 1	Feature 2	Feature 3	Feature 4
Timestamp 1	0.4	0.8	0.5	0.7
Timestamp 2	0.7	0.3	0.6	0.7
Timestamp 3	0.8	0.4	0.9	0.2
Timestamp 4	0.1	0.5	0.8	0.5
Timestamp 5	0.4	0.8	0.7	0.1
Timestamp 6	0.3	0.0	0.2	0.6
Timestamp 7	0.2	0.5	0.8	0.2

Filters

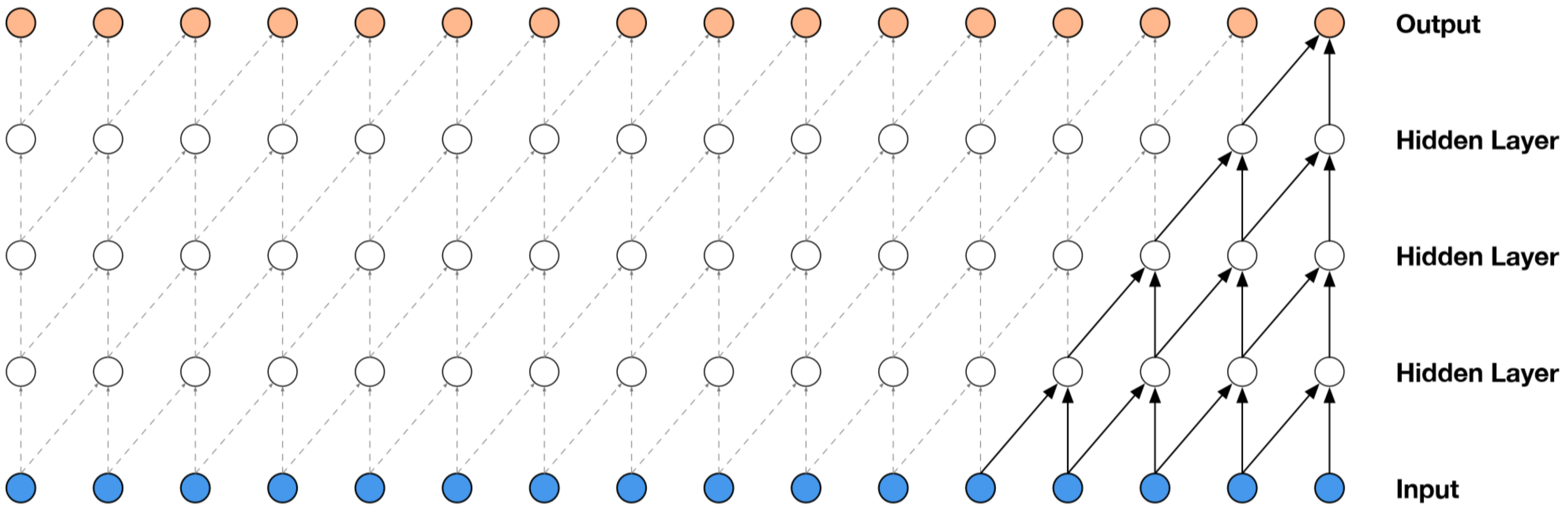
1	2	-3	1
-1	1	-2	2
2	2	1	3

1	1	2	1
1	0	1	0
1	1	2	1

	Filter 1	Filter 2
Timestamp 1, 2, 3	4.9	7.4
Timestamp 2, 3, 4	1.9	7.3
Timestamp 3, 4, 5	2.3	6.8
Timestamp 4, 5, 6	1.0	5.1
Timestamp 5, 6, 7	3.3	5.7

Causal Convolution

Only existing (no future) data is used in each neuron



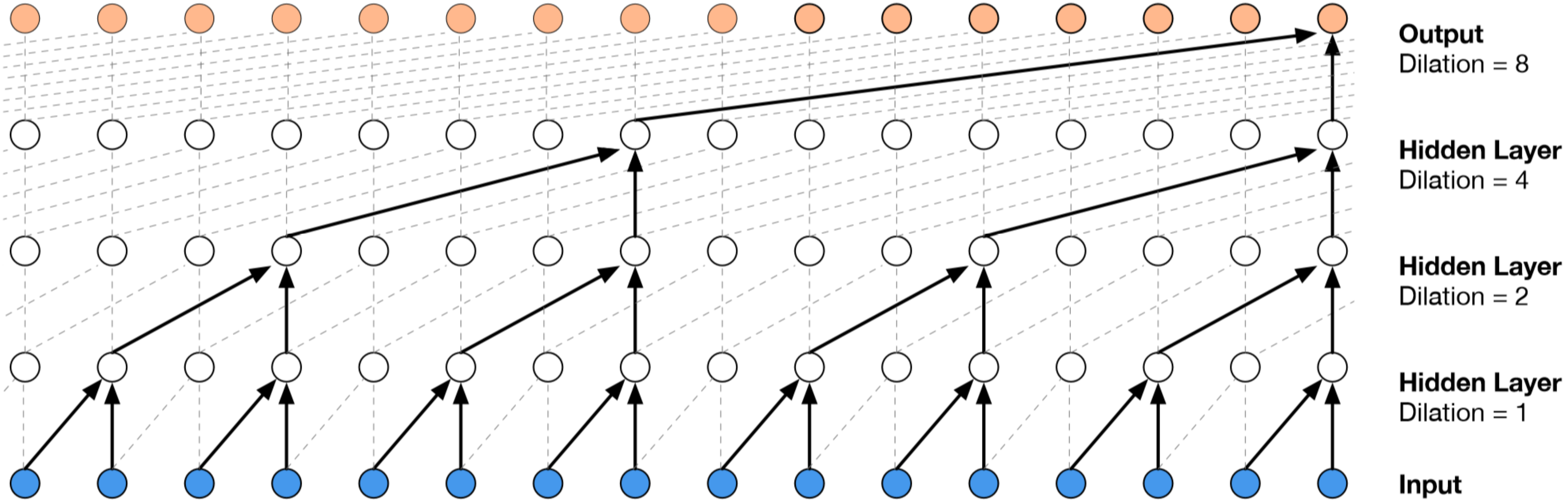
a stack of causal conv. layers

Oord et al. WaveNet: A Generative Model for Raw Audio.

Dilated Convolution

With dilation width $\boldsymbol{d}$, the conv. window starts at location $\boldsymbol{i}$ of size $\boldsymbol{k}$ is

$$\begin{bmatrix} \mathbf{x}_i & \mathbf{x}_{i+d} & \mathbf{x}_{i+2d} & \cdots & \mathbf{x}_{i+(k-1)\cdot d} \end{bmatrix}$$



stack of *dilated* causal convolutional layers

Oord et al. WaveNet: A Generative Model for Raw Audio.

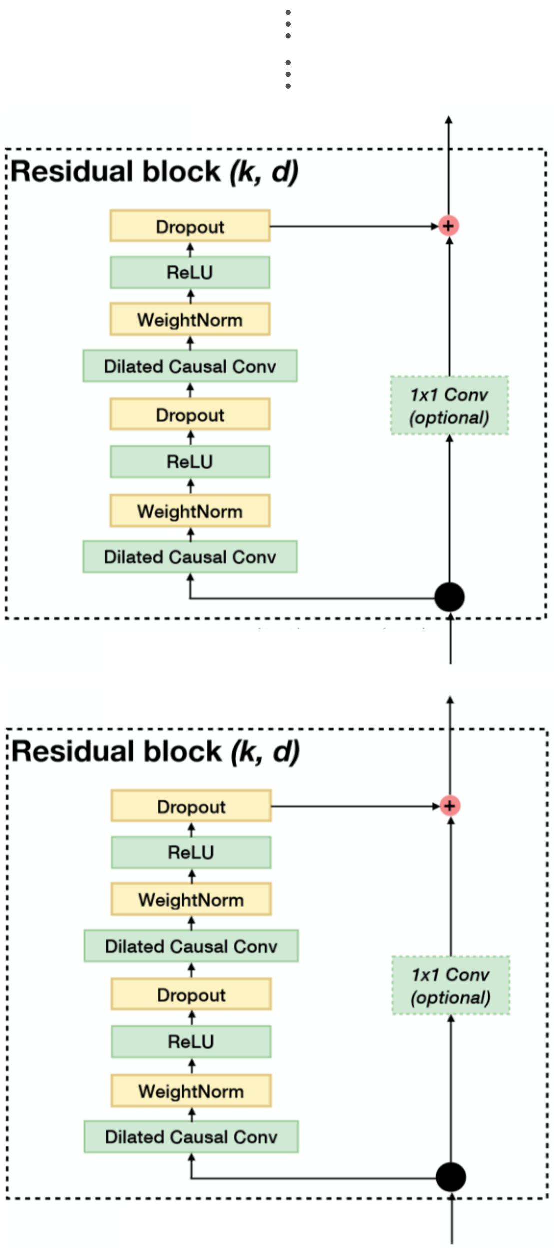
Dilated-CNN Advantages

- Increases receptive field
  - Receptive field is the part of the data visible to a neuron
  - Exponentially increasing the dilation factor results in exponential receptive field growth
- Captures global view of the input with less parameters
- Handles temporal flow with causal structure

K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition.

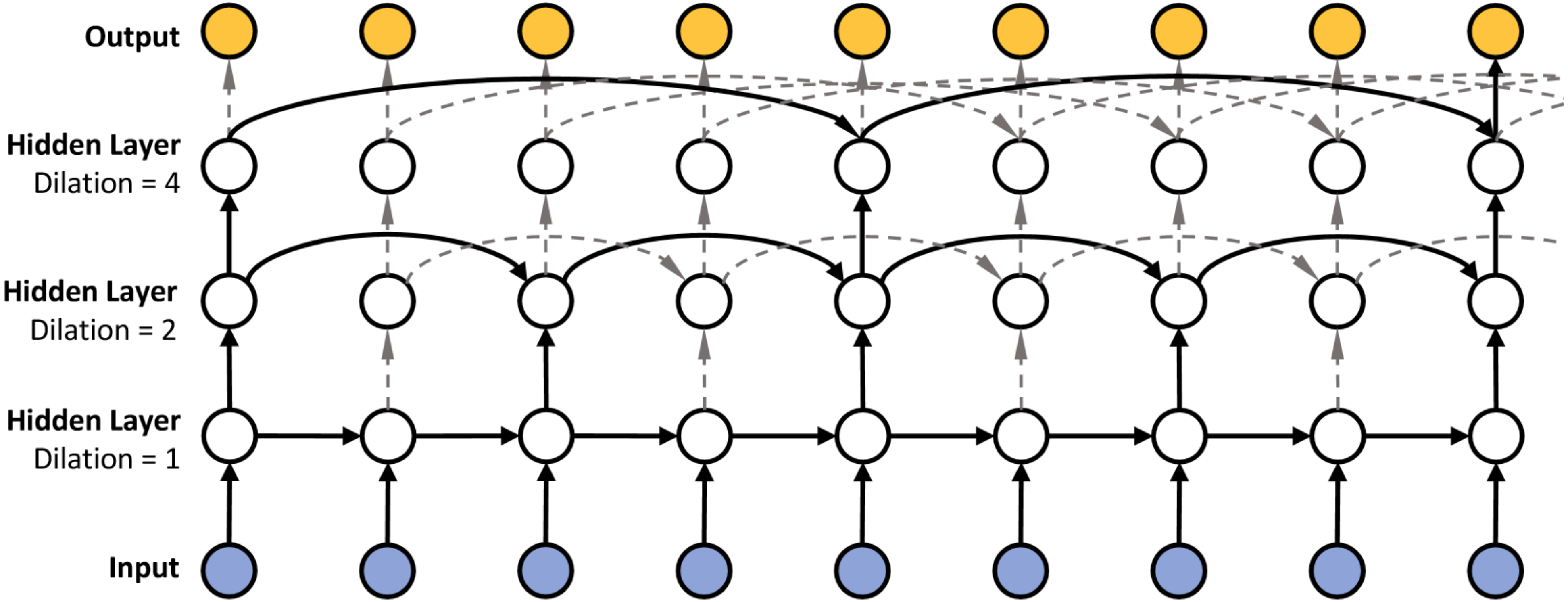
Dilated-CNN Design

- ReLU activation: $h = \max(0, a)$, where $a = Wx + b$.
- Dropout for regularization
- Use WeightNorm or BatchNorm to speed up convergence
- Use skip connection to avoid degradation of model performance when adding more layers
- Concatenate building blocks to increase model capacity



S. Bai, J. Kolter, and V. Koltun. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling.

Dilated-RNNs



example of a three-layer Dilated-RNN with dilation 1, 2, and 4

Chang et al. Dilated Recurrent Neural Networks.

Agenda

- Overview of Time Series Forecasting Methods
- Foundations of Dilated Convolutional Neural Networks
- Application to Retail Demand Forecasting

Application to Retail Demand Forecasting

OrangeJuice dataset in *bayesm* R package

store	brand	week	logmove	constant	price1	price2	price3
2	1	40	9.018695	1	0.060469	0.060497	0.042031
2	1	46	8.723231	1	0.060469	0.060312	0.045156
2	1	47	8.253228	1	0.060469	0.060312	0.045156
2	1	48	8.987197	1	0.060469	0.060312	0.049844

price4	price5	price6	price7	price8	price9	price10
0.029531	0.049531	0.053021	0.038906	0.041406	0.028906	0.024844
0.046719	0.049531	0.047813	0.045781	0.027969	0.042969	0.042031
0.046719	0.037344	0.053021	0.045781	0.041406	0.048125	0.032656
0.037344	0.049531	0.053021	0.045781	0.041406	0.042344	0.032656

price11	deal	feat	profit
0.038984	1	0.0	37.992326
0.038984	0	0.0	30.126667
0.038984	0	0.0	30.000000
0.038984	0	0.0	29.950000

83 stores
11 brands
121 weeks

https://cran.r-project.org/web/packages/bayesm/bayesm.pdf

Sample Data


Weekly sales of store 2 brand 1 (missing values are filled with zeros)
move
50000
40000
30000
20000
10000
0
40
80
120
160
week

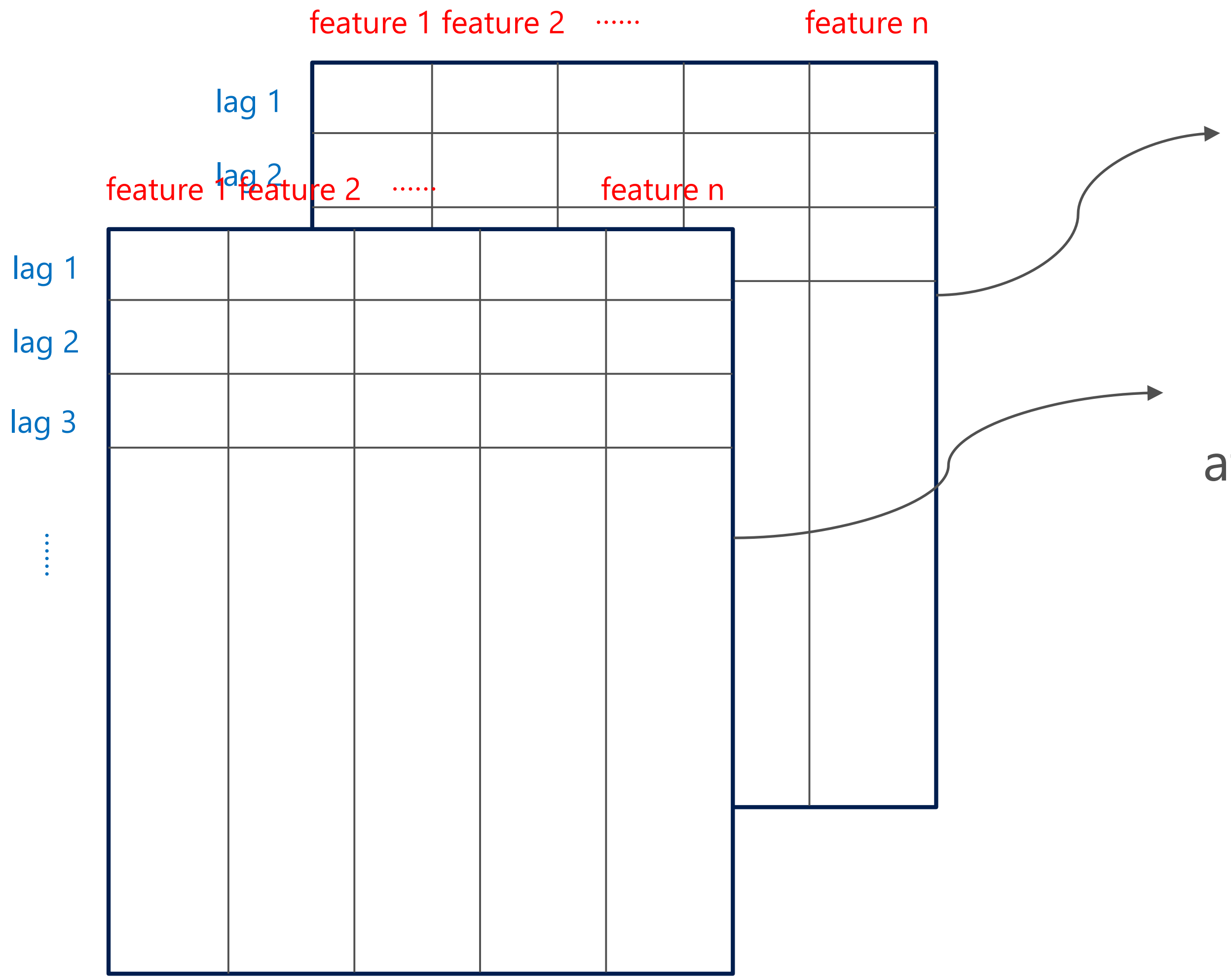
Feature Engineering

- *move* – historical unit sales
- *deal, feat* – if on sale or advertisement
- *price* – price of the current brand
- *price_ratio* – relative price to other brands

$$\frac{price}{average\_price\_of\_all\_brands}$$

- *month* – month number
- *week_of_month* – week number of the month

Data Transformation



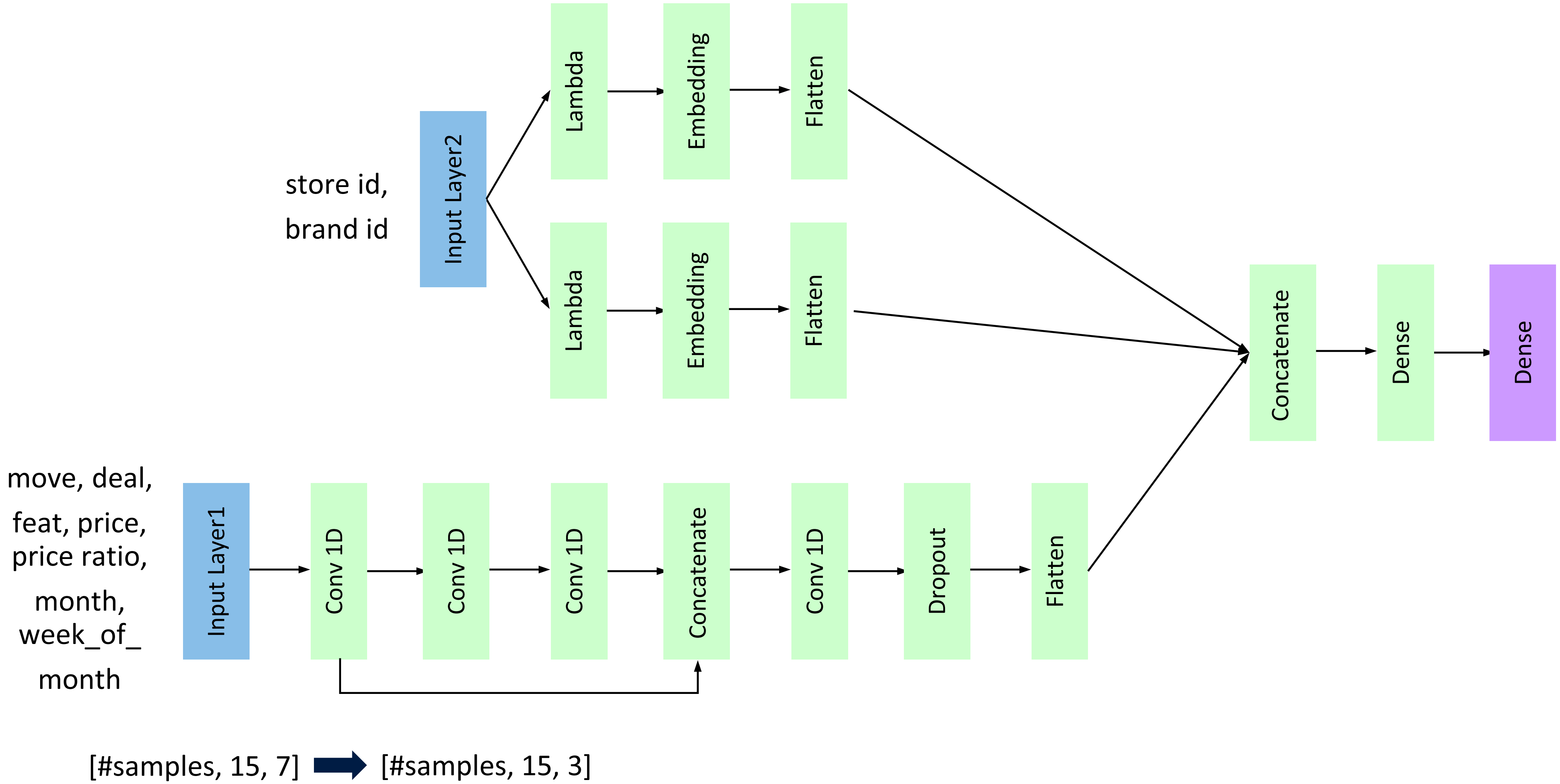
features created at time t

features created at time $t+1$

(# of lags, # of features)

(# of samples, # of lags, # of features)

Network Structure



Model Definition

- *Embedding:* encode categorical features into binary sequences
- *Conv1D*
  - n_filters
  - kernel_size
  - dilation_rate
  - padding
  - activation function
- *Skip connections*
- *Combine with categorial features and pass to a dense layer*

```
# Sequential input
seq_in = Input(shape=(seq_len, n_input_series))

# Categorical input
cat_fea_in = Input(shape=(2,), dtype='uint8')
store_id = Lambda(lambda x: x[:, 0, None])(cat_fea_in)
brand_id = Lambda(lambda x: x[:, 1, None])(cat_fea_in)
store_embed = Embedding(MAX_STORE_ID+1, 7, input_length=1)(store_id)
brand_embed = Embedding(MAX_BRAND_ID+1, 4, input_length=1)(brand_id)

# Dilated convolutional layers
c1 = Conv1D(filters=n_filters, kernel_size=kernel_size, dilation_rate=1,
            padding='causal', activation='relu')(seq_in)
c2 = Conv1D(filters=n_filters, kernel_size=kernel_size, dilation_rate=2,
            padding='causal', activation='relu')(c1)
c3 = Conv1D(filters=n_filters, kernel_size=kernel_size, dilation_rate=4,
            padding='causal', activation='relu')(c2)

# Skip connections
c4 = concatenate([c1, c3])

# Output of convolutional layers
conv_out = Conv1D(8, 1, activation='relu')(c4)
conv_out = Dropout(args.dropout_rate)(conv_out)
conv_out = Flatten()(conv_out)

# Concatenate with categorical features
x = concatenate([conv_out, Flatten()(store_embed), Flatten()(brand_embed)])
x = Dense(16, activation='relu')(x)
output = Dense(n_outputs, activation='linear')(x)

# Define model interface, loss function, and optimizer
model = Model(inputs=[seq_in, cat_fea_in], outputs=output)
```

Model Definition

- *Embedding:* encode categorical features into binary sequences
- *Conv1D*
  - n_filters
  - kernel_size
  - dilation_rate
  - padding
  - activation function
- *Skip connections*
- *Combine with categorial features and pass to a dense layer*

```
# Sequential input
seq_in = Input(shape=(seq_len, n_input_series))

# Categorical input
cat_fea_in = Input(shape=(2,), dtype='uint8')
store_id = Lambda(lambda x: x[:, 0, None])(cat_fea_in)
brand_id = Lambda(lambda x: x[:, 1, None])(cat_fea_in)
store_embed = Embedding(MAX_STORE_ID+1, 7, input_length=1)(store_id)
brand_embed = Embedding(MAX_BRAND_ID+1, 4, input_length=1)(brand_id)

# Dilated convolutional layers
c1 = Conv1D(filters=n_filters, kernel_size=kernel_size, dilation_rate=1,
            padding='causal', activation='relu')(seq_in)
c2 = Conv1D(filters=n_filters, kernel_size=kernel_size, dilation_rate=2,
            padding='causal', activation='relu')(c1)
c3 = Conv1D(filters=n_filters, kernel_size=kernel_size, dilation_rate=4,
            padding='causal', activation='relu')(c2)

# Skip connections
c4 = concatenate([c1, c3])

# Output of convolutional layers
conv_out = Conv1D(8, 1, activation='relu')(c4)
conv_out = Dropout(args.dropout_rate)(conv_out)
conv_out = Flatten()(conv_out)

# Concatenate with categorical features
x = concatenate([conv_out, Flatten()(store_embed), Flatten()(brand_embed)])
x = Dense(16, activation='relu')(x)
output = Dense(n_outputs, activation='linear')(x)

# Define model interface, loss function, and optimizer
model = Model(inputs=[seq_in, cat_fea_in], outputs=output)
```

Model Definition

- *Embedding:* encode categorical features into binary sequences
- *Conv1D*
  - n_filters
  - kernel_size
  - dilation_rate
  - padding
  - activation function
- *Skip connections*
- *Combine with categorial features and pass to a dense layer*

```
# Sequential input
seq_in = Input(shape=(seq_len, n_input_series))

# Categorical input
cat_fea_in = Input(shape=(2,), dtype='uint8')
store_id = Lambda(lambda x: x[:, 0, None])(cat_fea_in)
brand_id = Lambda(lambda x: x[:, 1, None])(cat_fea_in)
store_embed = Embedding(MAX_STORE_ID+1, 7, input_length=1)(store_id)
brand_embed = Embedding(MAX_BRAND_ID+1, 4, input_length=1)(brand_id)

# Dilated convolutional layers
c1 = Conv1D(filters=n_filters, kernel_size=kernel_size, dilation_rate=1,
            padding='causal', activation='relu')(seq_in)
c2 = Conv1D(filters=n_filters, kernel_size=kernel_size, dilation_rate=2,
            padding='causal', activation='relu')(c1)
c3 = Conv1D(filters=n_filters, kernel_size=kernel_size, dilation_rate=4,
            padding='causal', activation='relu')(c2)

# Skip connections
c4 = concatenate([c1, c3])

# Output of convolutional layers
conv_out = Conv1D(8, 1, activation='relu')(c4)
conv_out = Dropout(args.dropout_rate)(conv_out)
conv_out = Flatten()(conv_out)

# Concatenate with categorical features
x = concatenate([conv_out, Flatten()(store_embed), Flatten()(brand_embed)])
x = Dense(16, activation='relu')(x)
output = Dense(n_outputs, activation='linear')(x)

# Define model interface, loss function, and optimizer
model = Model(inputs=[seq_in, cat_fea_in], outputs=output)
```

Model Definition

- *Embedding:* encode categorical features into binary sequences
- *Conv1D*
  - n_filters
  - kernel_size
  - dilation_rate
  - padding
  - activation function
- *Skip connections*
- *Combine with categorial features and pass to a dense layer*

```
# Sequential input
seq_in = Input(shape=(seq_len, n_input_series))

# Categorical input
cat_fea_in = Input(shape=(2,), dtype='uint8')
store_id = Lambda(lambda x: x[:, 0, None])(cat_fea_in)
brand_id = Lambda(lambda x: x[:, 1, None])(cat_fea_in)
store_embed = Embedding(MAX_STORE_ID+1, 7, input_length=1)(store_id)
brand_embed = Embedding(MAX_BRAND_ID+1, 4, input_length=1)(brand_id)

# Dilated convolutional layers
c1 = Conv1D(filters=n_filters, kernel_size=kernel_size, dilation_rate=1,
            padding='causal', activation='relu')(seq_in)
c2 = Conv1D(filters=n_filters, kernel_size=kernel_size, dilation_rate=2,
            padding='causal', activation='relu')(c1)
c3 = Conv1D(filters=n_filters, kernel_size=kernel_size, dilation_rate=4,
            padding='causal', activation='relu')(c2)

# Skip connections
c4 = concatenate([c1, c3])

# Output of convolutional layers
conv_out = Conv1D(8, 1, activation='relu')(c4)
conv_out = Dropout(args.dropout_rate)(conv_out)
conv_out = Flatten()(conv_out)

# Concatenate with categorical features
x = concatenate([conv_out, Flatten()(store_embed), Flatten()(brand_embed)])
x = Dense(16, activation='relu')(x)
output = Dense(n_outputs, activation='linear')(x)

# Define model interface, loss function, and optimizer
model = Model(inputs=[seq_in, cat_fea_in], outputs=output)
```

Hyperparameter Tuning

HyperDrive via Azure Machine Learning SDK

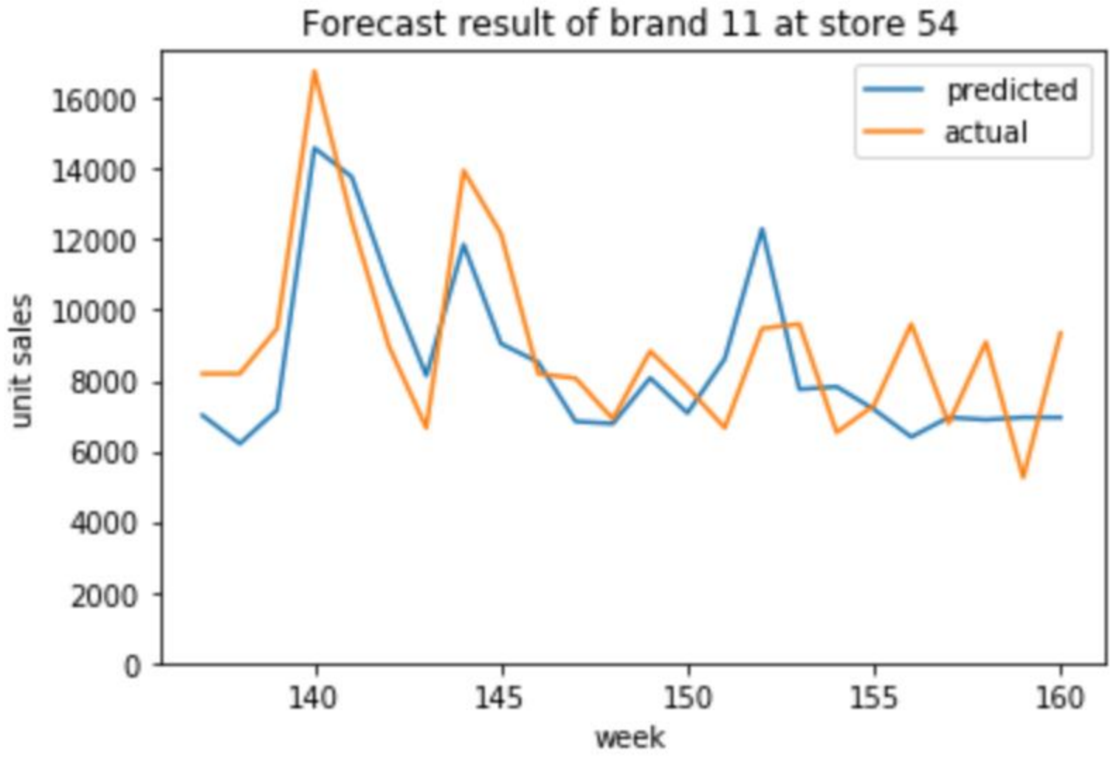
- Automates the hyperparameter sweeps on an elastic cluster
- Supports Bayesian sampling and early termination

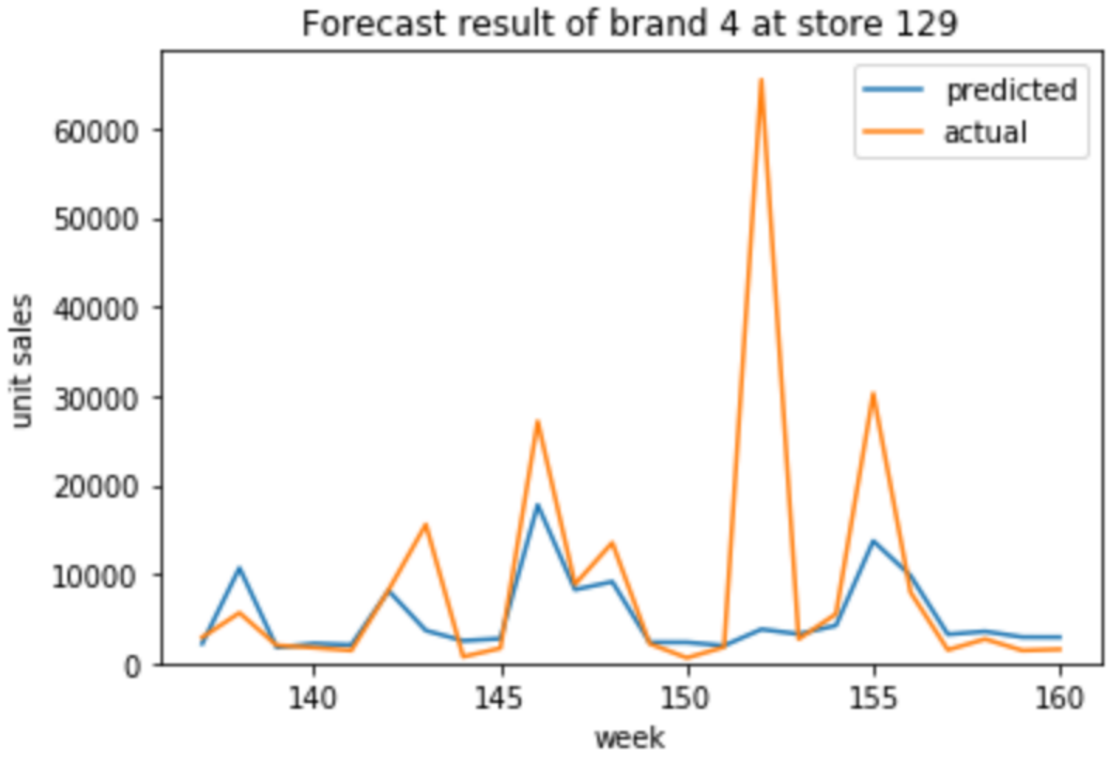
```
ps = BayesianParameterSampling({
    '--seq-len': quniform(5, 40, 1),
    '--dropout-rate': uniform(0, 0.4),
    '--batch-size': choice(32, 64),
    '--learning-rate': choice(1e-4, 1e-3, 5e-3, 1e-2, 1.5e-2, 2e-2, 3e-2, 5e-2, 1e-1),
    '--epochs': quniform(2, 80, 1)
})
htc = HyperDriveRunConfig(estimator=est,
                          hyperparameter_sampling=ps,
                          primary_metric_name='MAPE',
                          primary_metric_goal=PrimaryMetricGoal.MINIMIZE,
                          max_total_runs=200,
                          max_concurrent_runs=4)
htr = exp.submit(config=htc)
```

https://docs.microsoft.com/en-us/azure/machine-learning/service/how-to-tune-hyperparameters

Results

- Forecast unit sales between week 138 and 160 with model retrained every 2 weeks
- Azure Ubuntu Linux VM with one-half Tesla K80 GPU, 56 GB memory





Results

Mean absolute percentage error (MAPE)

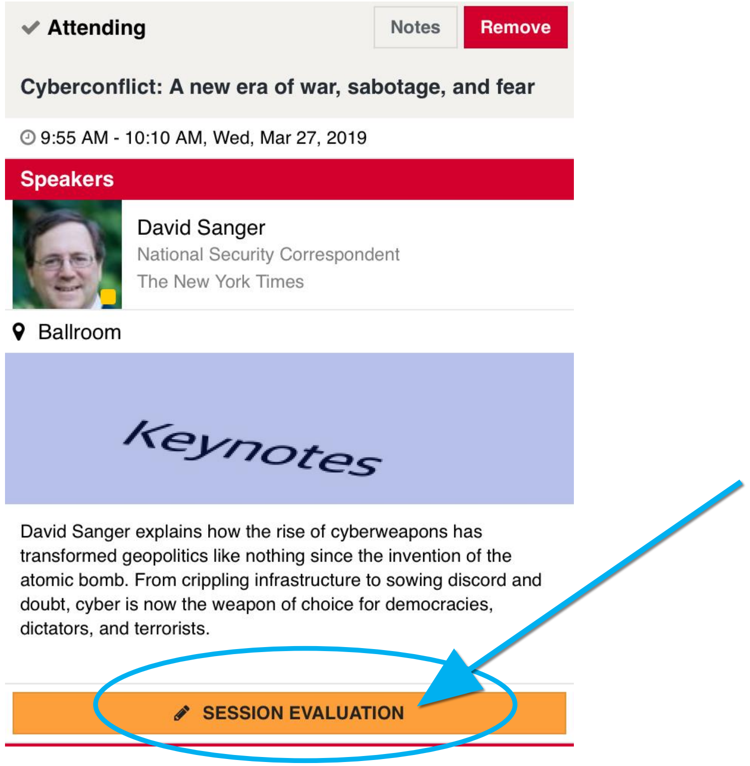
Method	MAPE	Running time	Machine
Dilated CNN	37.09 %	413 s	GPU Linux VM
Seq2Seq RNN	37.68 %	669 s	GPU Linux VM
Naive	109.67 %	114.06 s	CPU Linux VM
ETS	70.99 %	277.01 s	CPU Linux VM
ARIMA	70.80 %	265.94 s	CPU Linux VM

Results are collected based on the median of 5 run results

Thank you!

Rate today’s session

Session page on conference website



O’Reilly Events App