

# Deep Learning for Forecasting at Amazon: Problems and Methods

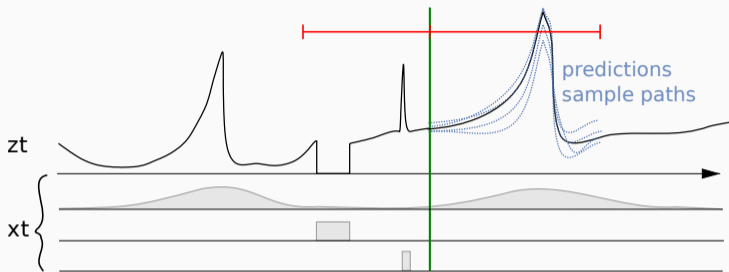
David Salinas

Amazon Research, Berlin

**CAMF London, October 2018**



- Examples of forecasting problems
- A coarse classification: not one canonical forecasting problem!
- One deep-learning approach for operational forecasting DeepAR [FSG17]

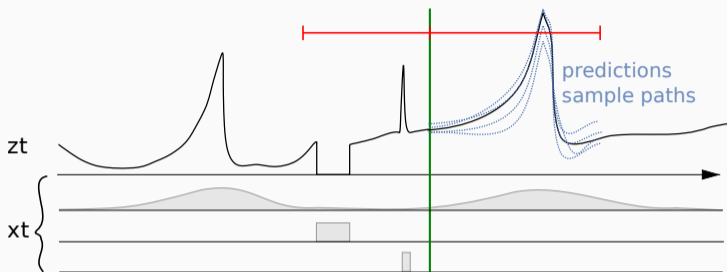


- Predict the future behavior of a time series given its past

$$\dots, z_{t_0-3}, z_{t_0-2}, z_{t_0-1} \implies P(z_{t_0}, z_{t_0+1}, \dots, z_T)$$

- Make optimal decisions

$$\text{best action} = \underset{a}{\operatorname{argmin}} \mathbb{E}_P[\text{cost}(a, z_{t_0}, z_{t_0+1}, \dots, z_T)]$$



- Predict the future behavior of a time series given its past

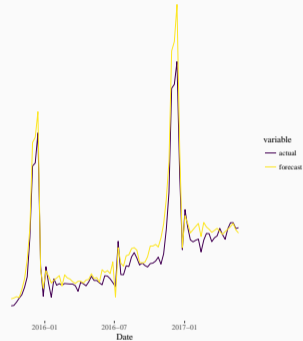
$$\dots, z_{t_0-3}, z_{t_0-2}, z_{t_0-1} \implies P(z_{t_0}, z_{t_0+1}, \dots, z_T)$$

- Make optimal decisions

$$\text{best action} = \underset{a}{\operatorname{argmin}} \mathbb{E}_P[\text{cost}(a, z_{t_0}, z_{t_0+1}, \dots, z_T)]$$

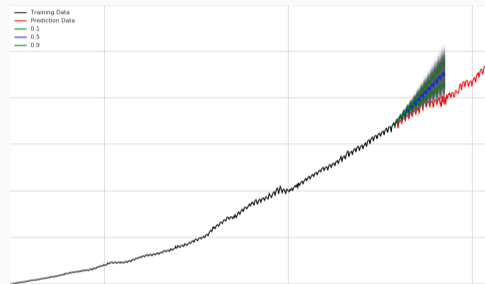
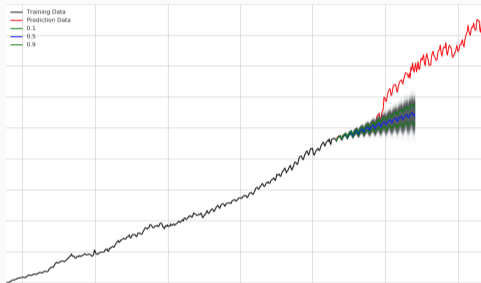
# Forecasting Problem I: Retail Demand

Weekly shipped units and forecast



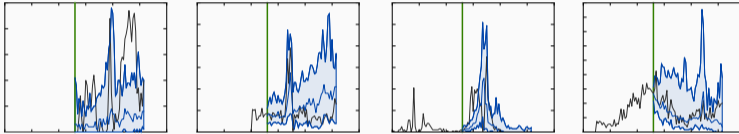
- Problem: predict overall Amazon retail demand years into the future.
- Decision Problems: topology planning, market entry/segment analyses

## Forecasting Problem II: AWS Compute Capacity



- Problem: predict AWS compute capacity demand
- Decision Problem: how many servers to order when and where

## Forecasting Problem IV: Retail product forecasting

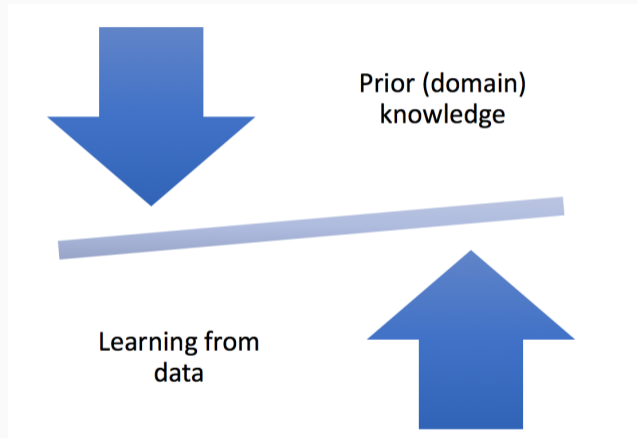


- Problem: predict the demand for a each product available at Amazon
- Decision Problems: how many units to order when, when to mark products down

## Taxonomy of Forecasting Problems: Dimensions

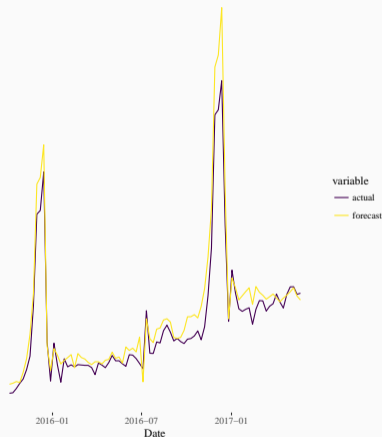
- number of time series/ratio of scientists per time series
- training of scientists: econometrics, statistics, machine learning, computer science
- forecast horizons: years to days
- time granularities: years, months, weeks, days
- aggregation granularity (for hierarchically organized time series)
- latency of forecast production/forecast computation frequency
- consumer of forecast/degree of automation/human interaction with forecast
- characteristics of time series
- *forecasting methods*: white vs black box (impose structure, parameter sharing, transparency)





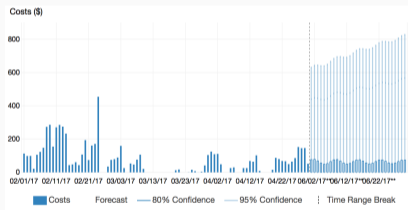
# Taxonomy of Forecasting Problems: Strategic Forecasting

Weekly shipped units and forecast



- Example: Overall demand for retail products on Amazon
- lots of econometricians for few time series
- forecast horizon: years, time granularity: weekly at most
- runs irregularly or a few times per month
- high degree of interaction with forecast
- models which estimate uncertainty correctly, allow to enforce structure, allow for careful modeling of effects
- high counts, relatively smooth, trend breaks possible, long history (in most cases)

# Taxonomy of Forecasting Problems: Tactical Forecasting



- Example: Ordering of compute racks for AWS
- 100s-1000s of time series per scientist
- forecast horizon: months, granularity: weekly at most
- runs irregularly or at most every week
- limited degree of interaction with forecast, but some constraints on stability of forecast over time and automated output checking
- models estimate uncertainty correctly, some transfer of information across time series necessary
- high counts, relatively smooth, trend breaks possible, short history & life cycles possible, burstiness

# Taxonomy of Forecasting Problems: Operational Forecasting



- Example: Demand forecast for retail products
- millions of time series per scientists (machine learning & software development engineers)
- forecast horizon: days, weeks, at most months
- runs at least daily/on-demand
- hands-off approach
- models can be more black box as long as they are robust
- low counts, bursty, short history and life cycles, intermittent

# The Classical Approach



Image (c) User:Colin / Wikimedia Commons / CC BY-SA 3.0

# The Classical Approach

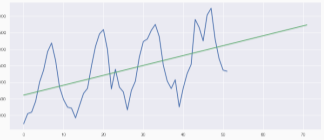


Image (c) User:Colin / Wikimedia Commons / CC BY-SA 3.0

# The Classical Approach

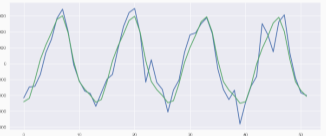


Image (c) User:Colin / Wikimedia Commons / CC BY-SA 3.0

# The Classical Approach

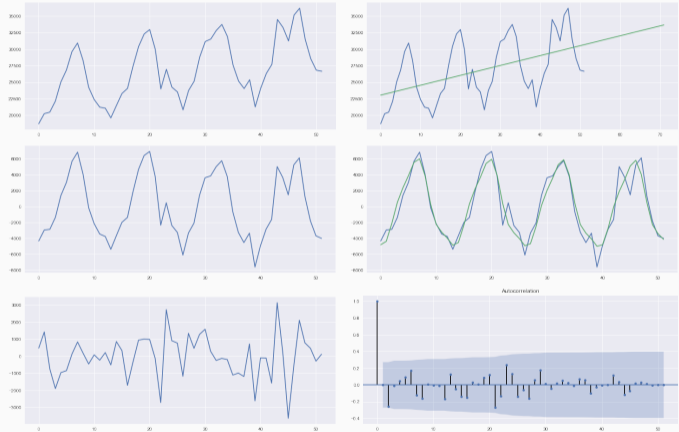


Image (c) User:Colin / Wikimedia Commons / CC BY-SA 3.0



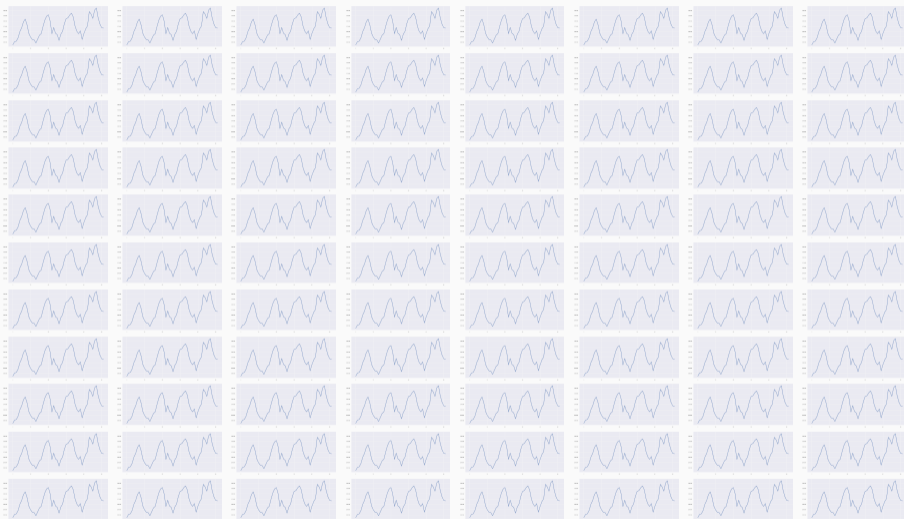
### PROS

- Well understood
- Decomposition  $\rightarrow$  decoupling
- White box: explicitly model-based
- Embarrassingly parallel

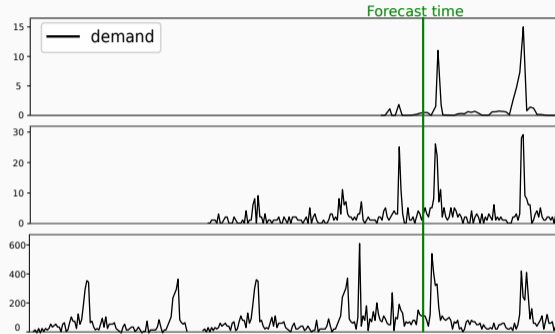
### CONS

- Model-based: all effects need to be explicitly modeled
- Rarely support additional time-features
- Gaussian noise often assumed
- Cannot learn patterns across time series

## Local vs. Global Models

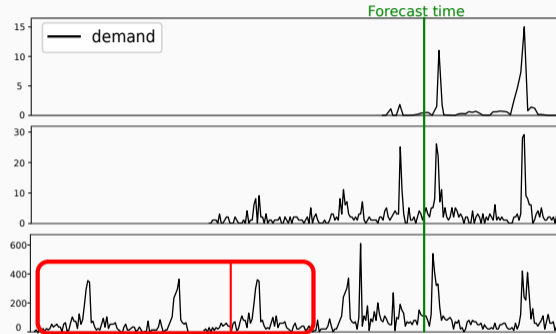


## Training with sliding windows



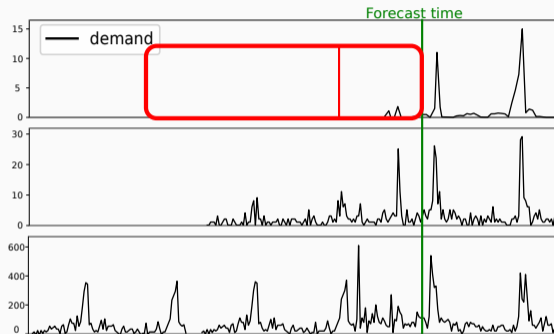
- Trained by maximizing likelihood on past windows

## Training with sliding windows



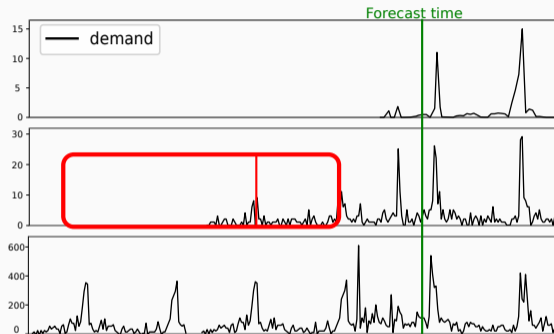
- Trained by maximizing likelihood on past windows

## Training with sliding windows



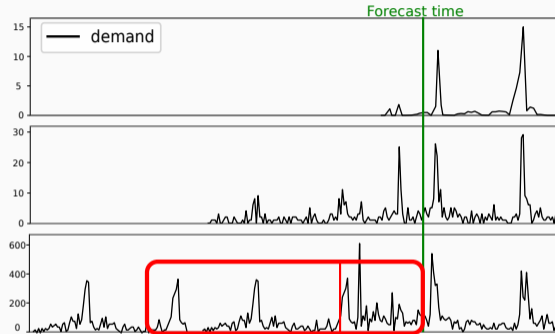
- Trained by maximizing likelihood on past windows

## Training with sliding windows



- Trained by maximizing likelihood on past windows

## Training with sliding windows



- Trained by maximizing likelihood on past windows

## Autoregressive Recurrent Networks

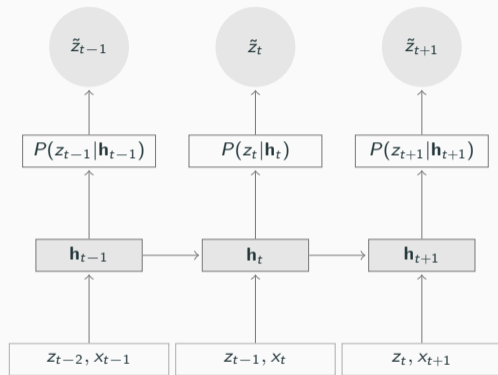
$$\mathbf{h}_t = \psi_{\mathbf{w}}(\mathbf{h}_{t-1}, z_{t-1}, \mathbf{x}_t)$$

$$z_t \sim P(z_t | \mathbf{w}_{\text{proj}}^T \mathbf{h}_t)$$

- The recurrent network  $\psi_{\mathbf{w}}(\cdot)$  is typically a stack of LSTM cells parametrized by  $\mathbf{w}$
- Any likelihood can be used, for instance, for a Gaussian likelihood:

$$P(z_t | \mathbf{h}_t) = \mathcal{N}(\mathbf{w}_{\mu}^T \mathbf{h}_t, \text{softplus}(\mathbf{w}_{\sigma}^T \mathbf{h}_t))$$

- Parameters  $\mathbf{w}_{\text{proj}}$  and  $\mathbf{w}$  are shared for all time-series and learned by backpropagation





## Autoregressive Recurrent Networks

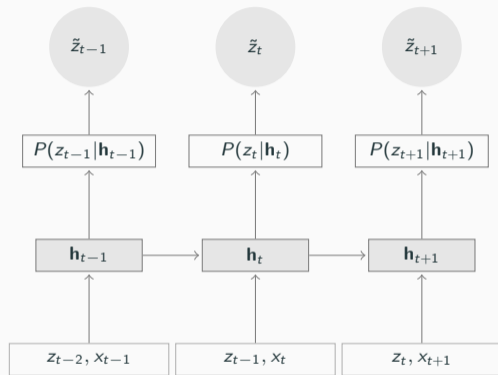
$$\mathbf{h}_t = \psi_{\mathbf{w}}(\mathbf{h}_{t-1}, z_{t-1}, \mathbf{x}_t)$$

$$z_t \sim P(z_t | \mathbf{w}_{\text{proj}}^T \mathbf{h}_t)$$

- The recurrent network  $\psi_{\mathbf{w}}(\cdot)$  is typically a stack of LSTM cells parametrized by  $\mathbf{w}$
- Any likelihood can be used, for instance, for a Gaussian likelihood:

$$P(z_t | \mathbf{h}_t) = \mathcal{N}(\mathbf{w}_{\mu}^T \mathbf{h}_t, \text{softplus}(\mathbf{w}_{\sigma}^T \mathbf{h}_t))$$

- Parameters  $\mathbf{w}_{\text{proj}}$  and  $\mathbf{w}$  are shared for all time-series and learned by backpropagation



## Autoregressive Recurrent Networks

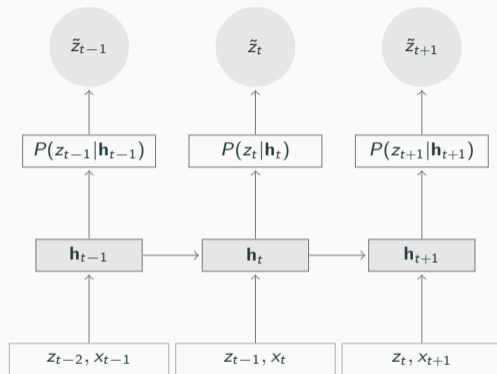
$$\mathbf{h}_t = \psi_{\mathbf{w}}(\mathbf{h}_{t-1}, z_{t-1}, \mathbf{x}_t)$$

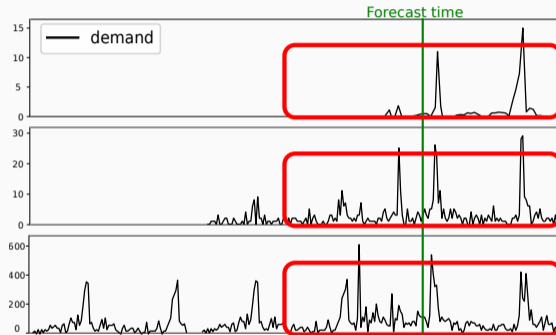
$$z_t \sim P(z_t | \mathbf{w}_{\text{proj}}^T \mathbf{h}_t)$$

- The recurrent network  $\psi_{\mathbf{w}}(\cdot)$  is typically a stack of LSTM cells parametrized by  $\mathbf{w}$
- Any likelihood can be used, for instance, for a Gaussian likelihood:

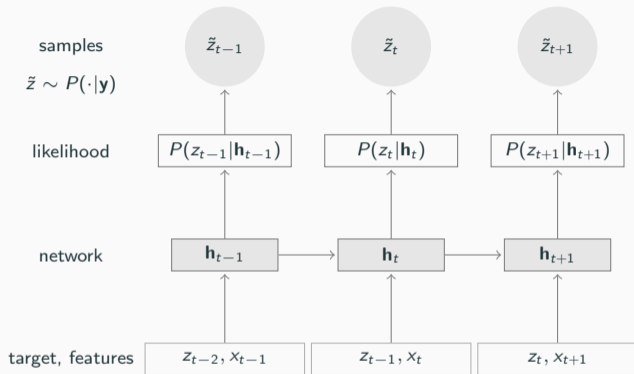
$$P(z_t | \mathbf{h}_t) = \mathcal{N}(\mathbf{w}_{\mu}^T \mathbf{h}_t, \text{softplus}(\mathbf{w}_{\sigma}^T \mathbf{h}_t))$$

- Parameters  $\mathbf{w}_{\text{proj}}$  and  $\mathbf{w}$  are **shared for all time-series** and learned by backpropagation

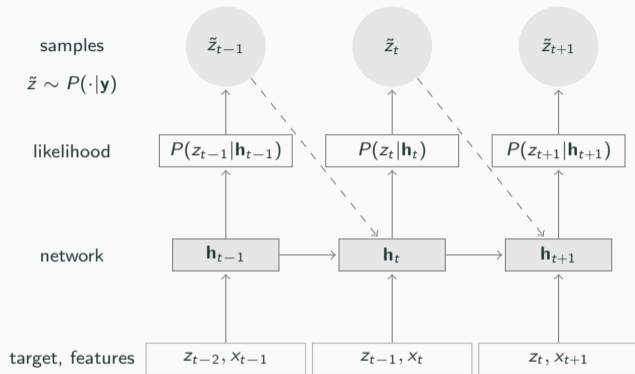




- target  $z_t$  is unobserved after the forecast time

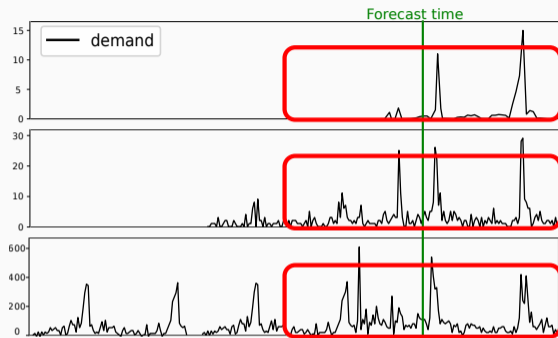


- $z_t$  target,  $x_t$  features,  $h_t$  LSTM state
- $P(z_t|y_t)$  likelihood: Gaussian, negative binomial



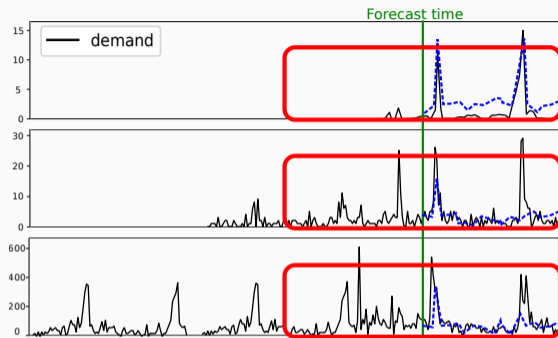
- $z_t$  target,  $x_t$  features,  $h_t$  LSTM state
- $P(z_t|y_t)$  likelihood: Gaussian, negative binomial
- Prediction: use sample  $\tilde{z} \sim P(\cdot|y)$  instead of true target for unknown (future) values

## Predicting with sample paths



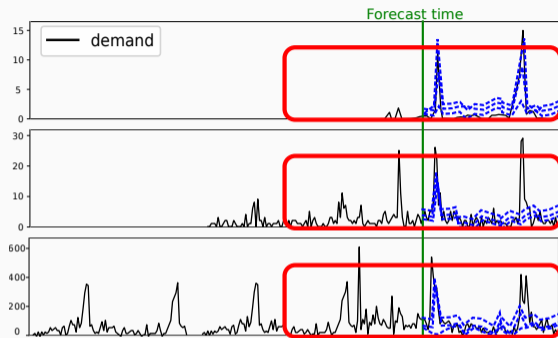
- Now we have a generative model!
  - The joint distribution is represented with *sample paths*
  - One can calculate confidence intervals, marginal distributions, ...

## Predicting with sample paths



- Now we have a generative model!
  - The joint distribution is represented with *sample paths*
  - One can calculate confidence intervals, marginal distributions, ...

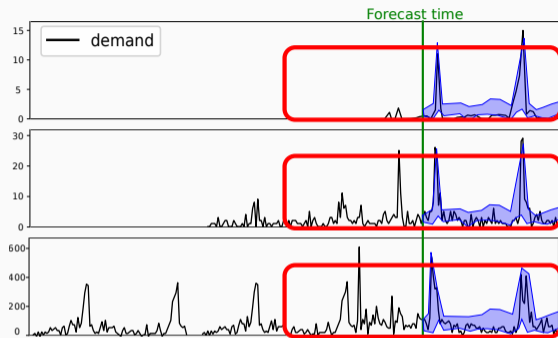
## Predicting with sample paths



- Now we have a generative model!
- The joint distribution is represented with *sample paths*
- One can calculate confidence intervals, marginal distributions, ...

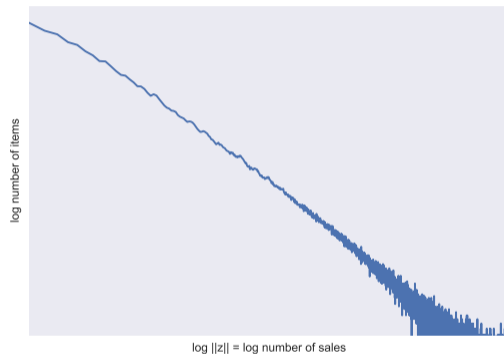


## Predicting with sample paths



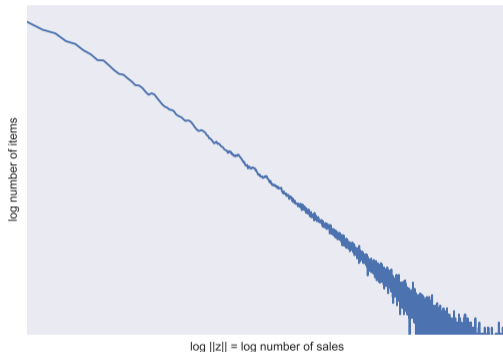
- Now we have a generative model!
- The joint distribution is represented with *sample paths*
- One can calculate confidence intervals, marginal distributions, ...

## Input scaling - what could go wrong



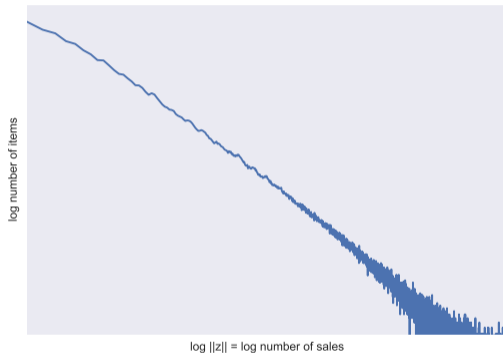
- Average number of sales follows a power-law across items
- Learning patterns across time series is difficult as amplitudes for  $z_t$  are drastically different
- Scale-free  $\Rightarrow$  no good bucket separation!
- Large amplitude items have larger signal to noise ratio but are sample very infrequently

## Input scaling - what could go wrong



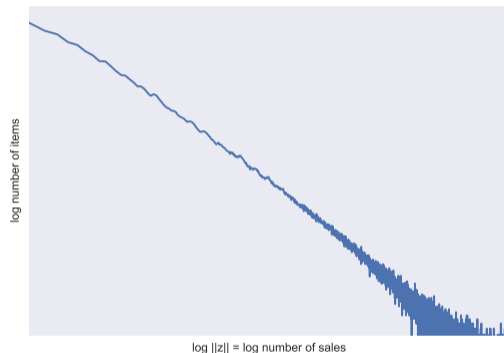
- Average number of sales follows a power-law across items
- Learning patterns across time series is difficult as amplitudes for  $z_t$  are drastically different
  - Scale-free  $\Rightarrow$  no good bucket separation!
  - Large amplitude items have larger signal to noise ratio but are sample very infrequently

## Input scaling - what could go wrong



- Average number of sales follows a power-law across items
- Learning patterns across time series is difficult as amplitudes for  $z_t$  are drastically different
- Scale-free  $\Rightarrow$  no good bucket separation!
- Large amplitude items have larger signal to noise ratio but are sample very infrequently

## Input scaling - what could go wrong



- Average number of sales follows a power-law across items
- Learning patterns across time series is difficult as amplitudes for  $z_t$  are drastically different
- Scale-free  $\Rightarrow$  no good bucket separation!
- Large amplitude items have larger signal to noise ratio but are sample very infrequently

### Reducing the scale amplitude variation

$$\nu_t = E[|z_t|] + 1$$

$$\mathbf{h}_t = \psi_{\mathbf{w}}(\mathbf{h}_{t-1}, z_{t-1}/\nu_t, \mathbf{x}_t)$$

$$P(z_t | h_t) = \mathcal{N}(w_{\mu}^T h_t * \nu_{t-1}, \text{softplus}(w_{\sigma}^T h_t) * \nu_{t-1})$$

- Inputs and outputs of the RNN are reparametrized as mean variations

### Weighed sampling to sample equally across different amplitudes

- Denote  $z_{it}$  the value of item  $i$  at time  $t$
- Sample item  $i$  with probability:

$$\frac{\sum_i |z_{it}|}{\sum_{i,j \neq i} |z_{jt}|}$$

### Reducing the scale amplitude variation

$$\nu_t = E[|z_t|] + 1$$

$$\mathbf{h}_t = \psi_{\mathbf{w}}(\mathbf{h}_{t-1}, z_{t-1}/\nu_t, \mathbf{x}_t)$$

$$P(z_t | h_t) = \mathcal{N}(w_{\mu}^T h_t * \nu_{t-1}, \text{softplus}(w_{\sigma}^T h_t) * \nu_{t-1})$$

- Inputs and outputs of the RNN are reparametrized as mean variations

### Weighed sampling to sample equally across different amplitudes

- Denote  $z_{it}$  the value of item  $i$  at time  $t$
- Sample item  $i$  with probability:

$$\sum_t |z_{it}| / \sum_{t,j \neq i} |z_{jt}|$$

- 5 Datasets including one with 500K weekly time series of sales of US products
- Baseline: Innovation State Space Model [SSF16], ETS [HKOS08], Croston [Cro72], Matfact [YRD16], Recurrent neural network without scaling/sampling
- On average 15% improvement for P50QL/P90QL
- < 5 features; little hyper-parameter tuning
- Training/predicting/evaluating 500K time-series takes less than 4 hours on a single AWS p2.xlarge instance (1 GPU & 4 CPU)



- 5 Datasets including one with 500K weekly time series of sales of US products
- Baseline: Innovation State Space Model [SSF16], ETS [HKOS08], Croston [Cro72], Matfact [YRD16], Recurrent neural network without scaling/sampling
- On average 15% improvement for P50QL/P90QL
- < 5 features; little hyper-parameter tuning
- Training/predicting/evaluating 500K time-series takes less than 4 hours on a single AWS p2.xlarge instance (1 GPU & 4 CPU)

## Some Empirical Results

- 5 Datasets including one with 500K weekly time series of sales of US products
- Baseline: Innovation State Space Model [SSF16], ETS [HKOS08], Croston [Cro72], Matfact [YRD16], Recurrent neural network without scaling/sampling
- On average **15%** improvement for P50QL/P90QL
- < 5 features; little hyper-parameter tuning
- Training/predicting/evaluating 500K time-series takes less than 4 hours on a single AWS p2.xlarge instance (1 GPU & 4 CPU)

## Some Empirical Results

- 5 Datasets including one with 500K weekly time series of sales of US products
- Baseline: Innovation State Space Model [SSF16], ETS [HKOS08], Croston [Cro72], Matfact [YRD16], Recurrent neural network without scaling/sampling
- On average **15%** improvement for P50QL/P90QL
- < 5 features; little hyper-parameter tuning
- Training/predicting/evaluating 500K time-series takes less than 4 hours on a single AWS p2.xlarge instance (1 GPU & 4 CPU)

## Some Empirical Results

- 5 Datasets including one with 500K weekly time series of sales of US products
- Baseline: Innovation State Space Model [SSF16], ETS [HKOS08], Croston [Cro72], Matfact [YRD16], Recurrent neural network without scaling/sampling
- On average **15%** improvement for P50QL/P90QL
- < 5 features; little hyper-parameter tuning
- Training/predicting/evaluating 500K time-series takes less than 4 hours on a single AWS p2.xlarge instance (1 GPU & 4 CPU)

### Datasets:

- parts, ec-sub, ec contains 1K, 40K and 530K demand time-series (integer values)
- traffic, and elec contains 1K and 400 hourly time-series of road and households usage (real values)

### Baselines

- Snyder: negative-binomial autoregressive method [SOB12]
- Croston: intermittent demand forecasting method (R package) [Cro72]
- ETS: exponential time smoothing with automatic model selection (R package) [HKOS08]
- ISSM model with covariates features shown earlier [SSF16]
- Rnn-gaussian: autoregressive RNN model with Gaussian likelihood
- Rnn-negative-binomial: autoregressive RNN model with negative binomial likelihood
- Matfact: matrix factorization [YRD16]
- Ours: appropriate likelihood + scaling + weighted sampling [FSG17] (DeepAR)

## Quantitative results

dataset	Snyder	Croston	ETS	ISSM	Rnn-gaussian	Rnn-negative-binomial	Ours
parts	0.0	-97.0	-23.0	-8.0	-19.0	1.0	6.0
ec-sub	0.0	-3.4	7.8	13.8	-4.3	-0.9	33.6
ec	0.0	-27.6	-5.7	4.8	3.8	11.4	19.0

**Table 1:** Percent improvement versus [Snyder 2012] (integer time-series)

	elec		traffic	
	ND	RMSE	ND	RMSE
Matfact	0.0	0.0	0.0	0.0
ours	128.6	15.0	17.6	2.4

**Table 2:** Percent improvement versus Matfact [YRD16] (real-value time-series)

# Some Real-World Examples



## Amazon SageMaker

Developer Guide



Documentation - This Guide

Search

What Is Amazon SageMaker?

How It Works

Getting Started

Using Built-in Algorithms

Common Information

Linear Learner

Factorization Machines

XGBoost Algorithm

Image Classification Algorithm

Sequence to Sequence (seq2seq)

K-Means Algorithm

Principal Component Analysis (PCA)

Latent Dirichlet Allocation (LDA)

Neural Topic Model (NTM)

DeepAR Forecasting

Hyperparameters

Inference Formats

BlazingText

Automatically Scaling Amazon SageMaker Models

[AWS Documentation](#) » [Amazon SageMaker](#) » [Developer Guide](#) » [Using Built-in Algorithms with Amazon SageMaker](#) » [DeepAR Forecasting](#)

## DeepAR Forecasting

Amazon SageMaker DeepAR is a supervised learning algorithm for forecasting scalar time series using recurrent neural networks (RNN). Classical forecasting methods, such as Autoregressive Integrated Moving Average (ARIMA) or Exponential Smoothing (ETS), fit one model to each individual time series, and then use that model to extrapolate the time series into the future. In many applications, however, you might have many similar time series across a set of cross-sectional units (for example, demand for different products, load of servers, requests for web pages, and so on). In this case, it can be beneficial to train a single model jointly over all of these time series. DeepAR takes this approach, training a model for predicting a time series over a large set of (related) time series.

For the training phase, the dataset consists of one or preferably more than one time series, and an optional categorical grouping variable of which the time series is a member. The model learns entirely from these values. The DeepAR algorithm currently accepts no other external features. The model is then trained by randomly selecting time points from the provided time series and using them as training examples.

For inference, the trained model takes as input an individual time series which might or might not have been used during training, and generates a forecast for the time series. This forecast takes into account what typically happened for similar time series in the training set.

### Input/Output Interface

DeepAR supports two data channels. The train channel is used for training a model and is required. The test channel is optional. If the test channel is present, the algorithm uses it to calculate accuracy metrics for the model after training. You can provide datasets as JSON or Parquet files.

By default, the model determines the input format from the file extension (either `.json` or `.parquet`). If you provide input files with different extensions, you can specify the file type by setting the `contentType` parameter of the `Channel` data type.

If you use a JSON file, it must be in the [JSON Lines](#) format, where each record contains the following fields:

- "start" whose value is a string of the format `YYYY-MM-DD HH:MM:SS`.
- "target", whose value is an array of floats (or integers) that represent the time series variable's values.
- "cat" (optional), whose value is an integer that encodes the categorical grouping that record's time series is a member of. The categorical feature allows the model to learn typical behavior for that group. This can increase accuracy.

The following is an example of JSON data:



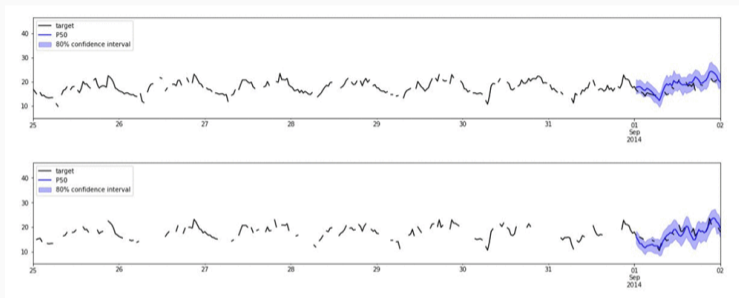
- data: json or parquet

```
{"start": "2012-01-03", "target": [1.9, 4.9, 6.3, 7.3, 7.8, ...], "cat": 2}  
{"start": "2012-04-05", "target": [2.3, 4.9, 6.2, 1.4, 4.3, ...], "cat": 0}  
{"start": "2012-04-05", "target": [5.0, 22.5, 23.1, 15.4, 34.0, ...], "cat": 1}  
...
```

- hyper parameters

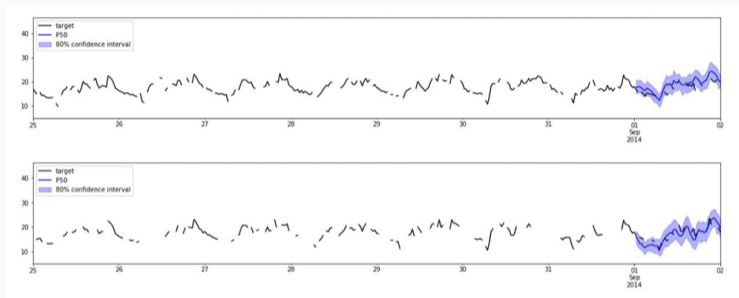
```
{  
  "time_freq": "W",  
  "prediction_length": 52,  
  "context_length": 52,  
  "likelihood": "gaussian",  
  "epochs": 100  
}
```

## Demo



- Try it yourself!
- Notebook at: [https://github.com/aws-labs/amazon-sagemaker-examples/blob/master/introduction\\_to\\_amazon\\_algorithms/deepar\\_electricity/DeepAR-Electricity.ipynb](https://github.com/aws-labs/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/deepar_electricity/DeepAR-Electricity.ipynb)
- Documentation: <https://docs.aws.amazon.com/sagemaker/latest/dg/deepar.html>

## Demo



- Try it yourself!
- Notebook at: [https://github.com/aws-labs/amazon-sagemaker-examples/blob/master/introduction\\_to\\_amazon\\_algorithms/deepar\\_electricity/DeepAR-Electricity.ipynb](https://github.com/aws-labs/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/deepar_electricity/DeepAR-Electricity.ipynb)
- Documentation: <https://docs.aws.amazon.com/sagemaker/latest/dg/deepar.html>

- Not one but many different forecasting problems
- Deep Learning methodology applied to forecasting yields flexible, accurate, and scalable forecasting systems
  - Providing sufficient data! (what is sufficient data?)
- Models can learn complex temporal patterns across time series
- “Model-free” black-box approaches trained end-to-end can replace complex model-based forecasting systems
- Handling scaling is key in reaching good accuracy (more generally keeping activations normalized)

J.D. Croston.

**Forecasting and stock control for intermittent demands.**

*Operational Research Quarterly*, 23:289–304, 1972.

Valentin Flunkert\*, David Salinas\*, and Jan Gasthaus.

**Deepar: Probabilistic forecasting with autoregressive recurrent networks.**

*CoRR*, abs/1704.04110, 2017.

R. Hyndman, A. B. Koehler, J. K. Ord, and R .D. Snyder.

**Forecasting with Exponential Smoothing: The State Space Approach.**

Springer Series in Statistics. Springer, 2008.

Ralph D Snyder, J Keith Ord, and Adrian Beaumont.

**Forecasting the intermittent demand for slow-moving inventories: A modelling approach.**

*International Journal of Forecasting*, 28(2):485–496, 2012.

Matthias W Seeger, David Salinas, and Valentin Flunkert.

**Bayesian intermittent demand forecasting for large inventories.**

In *Advances in Neural Information Processing Systems*, pages 4646–4654, 2016.

Hsiang-Fu Yu, Nikhil Rao, and Inderjit S Dhillon.

**Temporal regularized matrix factorization for high-dimensional time series prediction.**

In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 847–855. Curran Associates, Inc., 2016.

# Thank you!

PS: we are hiring :-)  
Ping me if you are interested!