Student Name: LI JIAYI
Student ID: 1155188890

Deep Learning for Time Series
Prediction

---

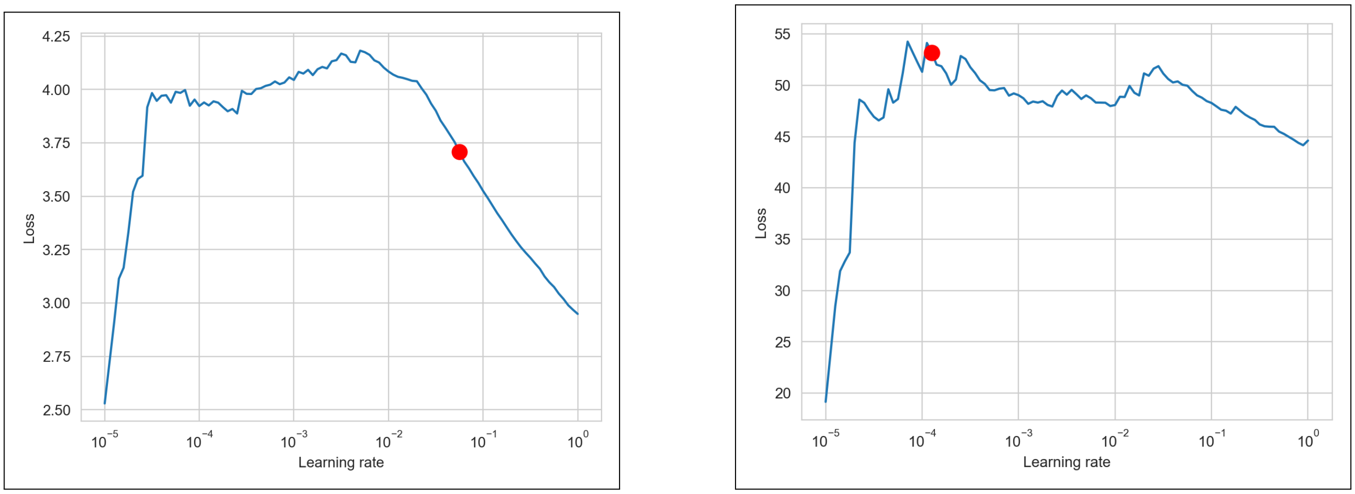
1. Amazon's DeepAR Model and DeepVAR Model

Table 1: Data preview

Date	0002.HK	0003.HK	0005.HK	0006.HK	0011.HK	0012.HK	0016.HK	0017.HK
2019-01-02	70.96	11.55	52.86	40.92	143.97	25.94	85.21	30.70
2019-01-03	70.96	11.49	52.82	41.12	139.89	26.25	87.71	30.82
2019-01-04	72.31	11.57	53.74	41.96	139.64	27.57	91.94	32.34
2019-01-07	72.27	11.64	54.20	41.96	140.39	27.94	92.57	32.82
2019-01-08	73.12	11.68	54.07	42.07	141.97	28.21	93.59	33.85

Table 2: Change data from wide to long format

Date	Ticker	Price	Time_idx
2019-01-02	0002.HK	70.96	0
2019-01-03	0002.HK	70.96	1
2019-01-04	0002.HK	72.31	2
2019-01-07	0002.HK	72.27	3
2019-01-08	0002.HK	73.12	4

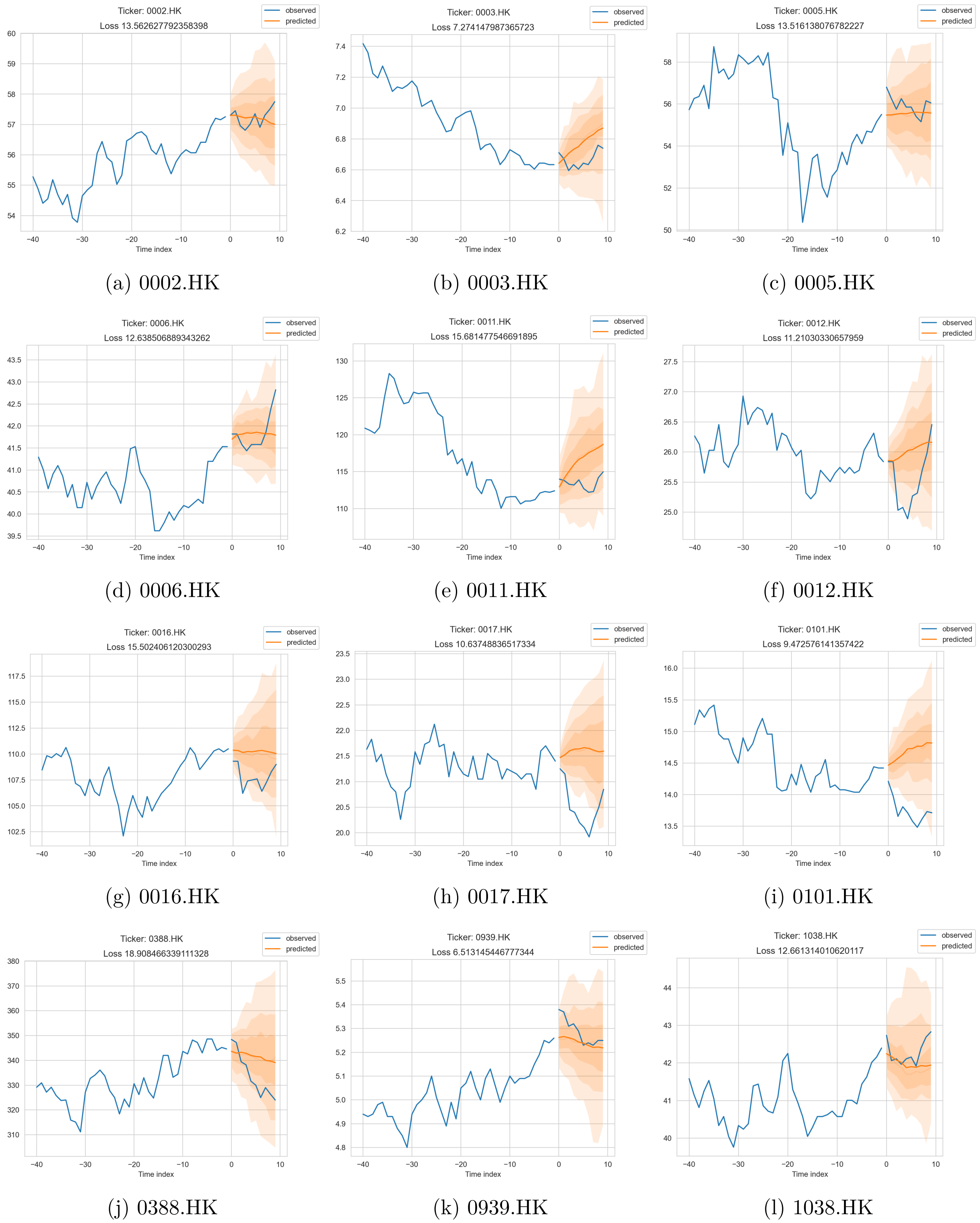


(a) DeepAR model

(b) DeepVAR model

Figure 1: Suggested learning rate for training the DeepAR Model and DeepVAR Model

```
Suggested learning rate: 0.056234(DeepAR) 0.000126(DeepVAR)
```



(a) 0002.HK

(b) 0003.HK

(c) 0005.HK

(d) 0006.HK

(e) 0011.HK

(f) 0012.HK

(g) 0016.HK

(h) 0017.HK

(i) 0101.HK

(j) 0388.HK

(k) 0939.HK

(l) 1038.HK

Figure 2: DeepAR's forecast for first 12 stocks in HengSeng Index

Code for Figure 2

```
import matplotlib.pyplot as plt
import pandas as pd
import torch
```

```python
import yfinance as yf
import seaborn as sns

import lightning.pytorch as pl
from lightning.pytorch.callbacks import EarlyStopping
from pytorch_forecasting import DeepAR, TimeSeriesDataSet
from lightning.pytorch.tuner.tuning import Tuner

#Read HSI index from the website
df = pd.read_html(
    "https://en.wikipedia.org/wiki/Hang_Seng_Index"
)
#Ticker is in 6th column and there exist &nbsp to be removed
df = df[6]["Ticker"].str.replace("SEHK:", "").str.lstrip()
df = df.str.zfill(4) + ".HK"

RISKY_ASSET = df.to_list()[0:20]
START_DATE = '2019-01-01'
END_DATE = '2023-05-01'

raw_df = yf.download(RISKY_ASSET,
                 start = START_DATE,
                 end = END_DATE,
                 progress = True)
df = raw_df["Adj Close"]

#Check if there exist missing values
df.isna().any()

#The drop=False argument is used to keep the current index as a column in the DataFrame.
df = df.reset_index(drop=False)
#pd.melt() transform a DataFrame from a wide format to a long format by unpivoting the data.
#id_vars: The column(s) to use as identifier variables remain columns in resulting DataFrame
#value_vars: The column(s) to unpivot, will be converted to a single column call "variable"
df = (
    pd.melt(df,
            id_vars=["Date"],
            value_vars=df.columns,
            var_name="Ticker",
            value_name="Price"
            )
)
df["Time_idx"] = df.groupby("Ticker").cumcount()

#The encoder is the part of the model that takes in historical data to make predictions
MAX_ENCODER_LENGTH = 40
#The prediction sequence is the part of the model that generates predictions for future
MAX_PRED_LENGTH = 10
#Number of time series that are included in each batch during training
BATCH_SIZE = 128
```

```python
#Maximum number of epochs that the model will be trained for
MAX_EPOCHS = 30
training_cutoff = df["Time_idx"].max() - MAX_PRED_LENGTH

#Define the training and validation datasets
train_set = TimeSeriesDataSet(
    df[lambda x: x["Time_idx"] <= training_cutoff],
    time_idx="Time_idx",
    target="Price",
    group_ids=["Ticker"],
    time_varying_unknown_reals=["Price"],
    max_encoder_length=MAX_ENCODER_LENGTH,
    max_prediction_length=MAX_PRED_LENGTH,
)

#Copies the properties of the training set to the validation set
valid_set = TimeSeriesDataSet.from_dataset(
    train_set, df, min_prediction_idx=training_cutoff+1
)

#TimeSeriesDataSet object can be used to train a time series model using PyTorch Lightning.
train_dataloader = train_set.to_dataloader(
    train=True, batch_size=BATCH_SIZE
)
valid_dataloader = valid_set.to_dataloader(
    train=False, batch_size=BATCH_SIZE
)

pl.seed_everything(7)

#Define the DeepAR model. By default, the DeepAR model uses the Gaussian loss function.
#Last two hyperparameters should be tuned using HPO framework (Hyperopt or Optuna)
deep_ar = DeepAR.from_dataset(
    train_set,
    learning_rate=1e-2,
    hidden_size=30,
    rnn_layers=4
)

#Find the suggested learning rate
trainer = pl.Trainer(gradient_clip_val=1e-1)
res = Tuner(trainer).lr_find(
    deep_ar,
    train_dataloaders=train_dataloader,
    val_dataloaders=valid_dataloader,
    min_lr=1e-5,
    max_lr=1e0,
    early_stop_threshold=100,
)
```

```
print(f"Suggested learning rate: {res.suggestion()}")
fig = res.plot(show=True, suggest=True)

pl.seed_everything(7)

#Train the DeepAR model using the identified learning rate
deep_ar.hparams.learning_rate = res.suggestion()

#Stop the training if no significant improvement in validation loss over 10 epochs
early_stop_callback = EarlyStopping(
    monitor="val_loss",
    min_delta=1e-4,
    patience=10
)

trainer = pl.Trainer(
    max_epochs=MAX_EPOCHS,
    gradient_clip_val=0.1,
    callbacks=[early_stop_callback]
)

trainer.fit(
    deep_ar,
    train_dataloaders=train_dataloader,
    val_dataloaders=valid_dataloader,
)

#Extract the best DeepAR model from a checkpoint
best_model = DeepAR.load_from_checkpoint(
    trainer.checkpoint_callback.best_model_path
)

#Create the predictions for the validation set and plot 12 of them
result = best_model.predict(
    valid_dataloader,
    mode="raw",
    return_x=True,
    n_samples=100
)

raw_predictions = result[0]
x = result[1]

tickers = valid_set.x_to_index(x)["Ticker"]

for idx in range(12):
    best_model.plot_prediction(
        x, raw_predictions, idx=idx, add_loss_to_title=True
    )
    plt.suptitle(f"Ticker: {tickers.iloc[idx]}")
```