Crypto-currency Price Prediction with Machine Learning

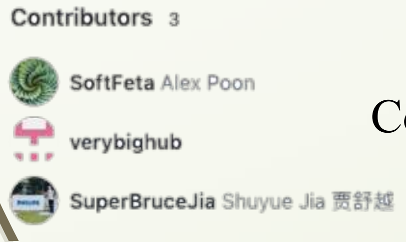
Big Data Parallel Processing by PySpark and Horovod **Distributed Deep Learning**

Project Team 5

Tso Yiu Chuen, Poon Bing Chun, Li Ka Faat, Shuyue Jia

EID: 90011533, 56200477, 55628042, 56846018

yctso4-c@my.cityu.edu.hk, bcpoon3@my-cityu.edu.hk, verybighub@gmail.com, shuyuej@ieee.org



Codes are publicly accessible: https://github.com/verybighub/CS5488_Project

Data Preprocessing by PySpark

```
oldtime = time()
from pyspark.sql.functions import col, log
cdf = cdf.withColumn('Log Price', log(10.0, col("Price USD")))
cdf = cdf.withColumn('Log Trading Volume Last 24h', log(10.0, col("Trading Volume Last 24h")))
cdf = cdf.withColumn('Log Market Cap', log(10.0, col("Market Cap")))
cdf.show()
print(f'Time needed: {time()-oldtime} s')
```

```
21/11/18 13:44:41 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can c
ause serious performance degradation.
[Stage 5:===============================================>             (4 + 1) / 5]
```

```
+-------------------+-------------------+-----------------------+-----------------+-------------------+-----------------------
----+-----------------+
|           DateTime|          Price USD|Trading Volume Last 24h|       Market Cap|          Log Price|Log Trading Volume Last
24h|   Log Market Cap|
+-------------------+-------------------+-----------------------+-----------------+-------------------+-----------------------
----+-----------------+
|2019-01-01 08:04:04|       0.1146302991|       90503046.2253028|2196405505.5719624|-0.9407005744041936|         7.956663197288
9015|9.341712523653495|
```

```
Time needed: 14.219001293182373 s
```

- **Data scaling**

→ Use **PySpark** to process data

≈ **twice speed** compared with Pandas

→ Apply **Log scale** to raw data

original value is too **large**

Data Preprocessing by Pyspark

Data scaling

→ Use **Zero Mean and Unit Variance Normalization**

```
# # Data scaling - Normalization
oldtime = time()
from pyspark.sql.functions import mean as _mean
from pyspark.sql.functions import stddev as _std
# Data scaling
def column_statistics(df, name=""):
    df_stats = df.select(
        _mean(col(name)).alias('mean'),
        _std(col(name)).alias('std')
    ).collect()

    return df_stats[0]['mean'], df_stats[0]['std']

data_p_mean, data_p_std = column_statistics(cdf, "Log Price")
data_v_mean, data_v_std = column_statistics(cdf, "Log Trading Volume Last 24h")
data_m_mean, data_m_std = column_statistics(cdf, "Log Market Cap")

cdf = cdf.withColumn("Price_Mean", f.lit(data_p_mean))
cdf = cdf.withColumn("Price_Std", f.lit(data_p_std))
cdf = cdf.withColumn("Price_Normalized", (f.col("Log Price") - f.col("Price_Mean")) / f.col("Price_Std"))

cdf = cdf.withColumn("Volume_Mean", f.lit(data_v_mean))
cdf = cdf.withColumn("Volume_Std", f.lit(data_v_std))
cdf = cdf.withColumn("Volume_Normalize", (f.col("Log Trading Volume Last 24h") - f.col("Volume_Mean")) / f.col("Volume_Std"))

cdf = cdf.withColumn("Market_Mean", f.lit(data_m_mean))
cdf = cdf.withColumn("Market_Std", f.lit(data_m_std))
cdf = cdf.withColumn("Market_Normalize", (f.col("Log Market Cap") - f.col("Market_Mean")) / f.col("Market_Std"))
cdf.show()
print(f'Time needed: {time()-oldtime} s')
```

Model selection: MLP and LSTM

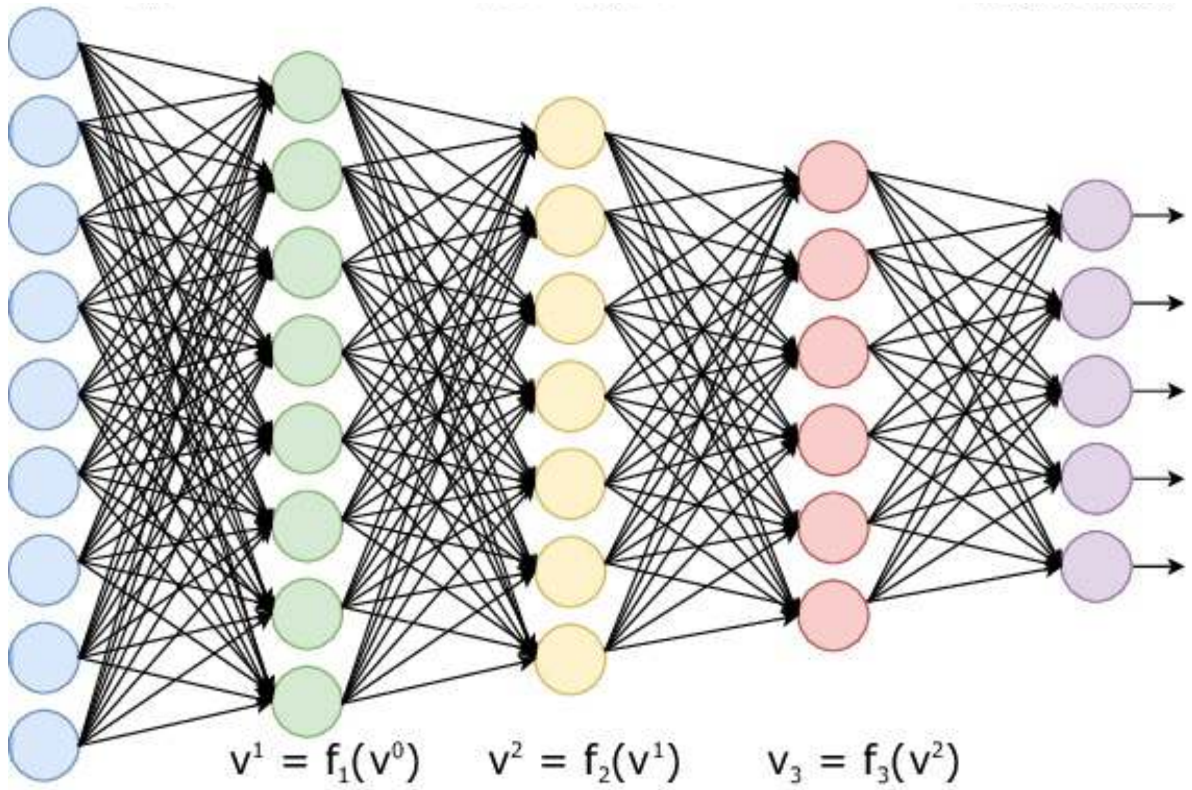


Figure 1: A Fully-connected Neural Network

Reference: Figure 1 is from open-sourced Google Images

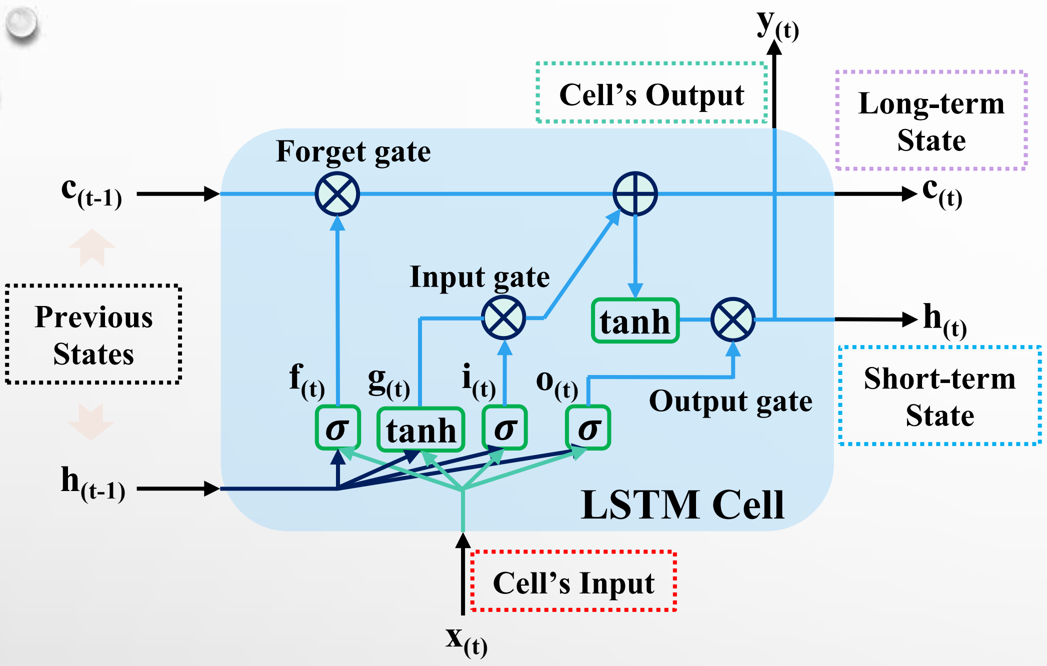


Figure 1: A Long-short Term Memory

Figure 2: Currency Price through time

Reference:

Figure 1 is from Shuyue Jia's under review paper.

"Deep feature mining via attention-based BiLSTM-GCN for human motor imagery recognition."

Currently Under review at *Frontier in Neuroscience*

https://arxiv.org/abs/2005.00777

Distributed Training by **Horovod**

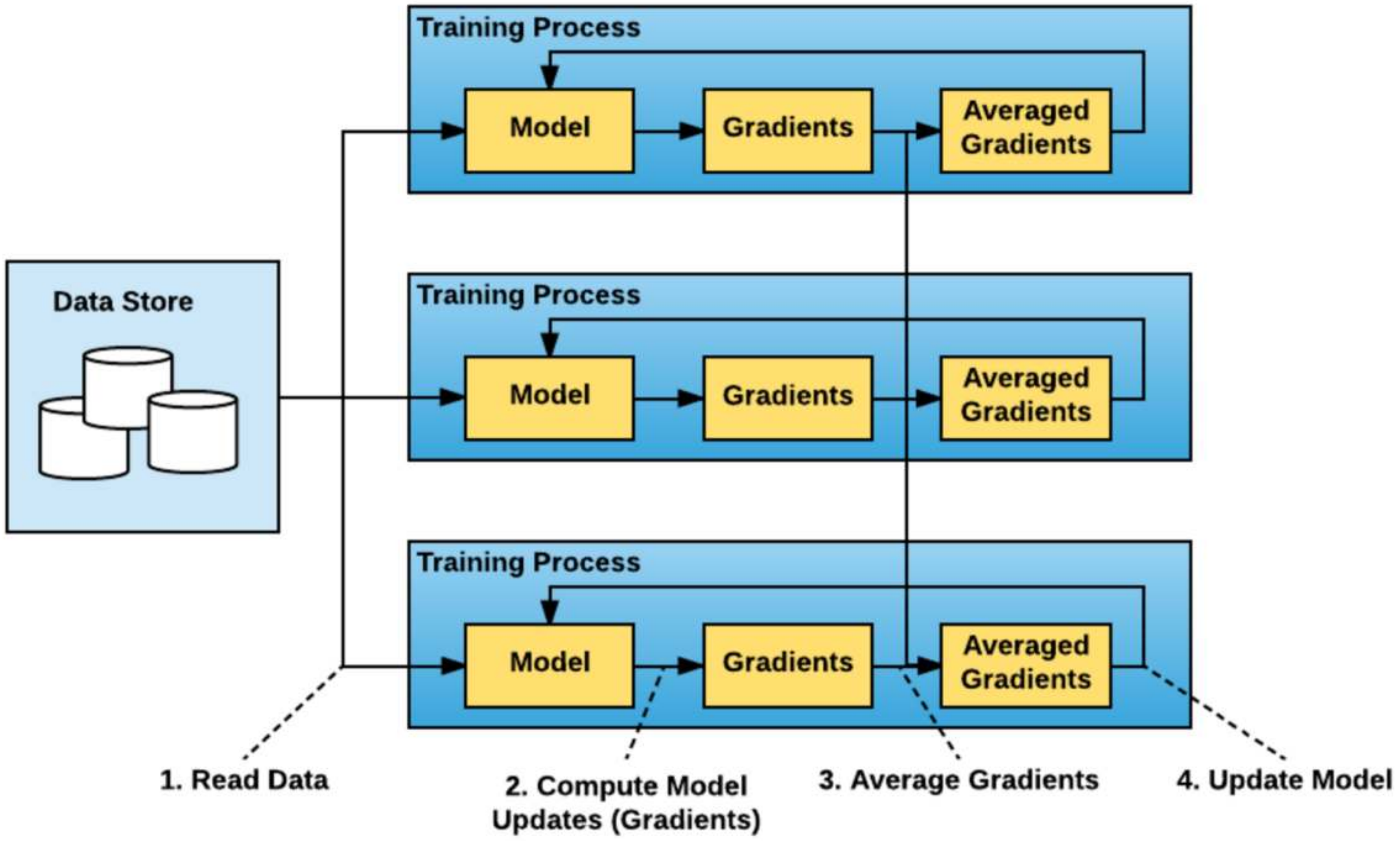


Figure 1: Data Parallel Distributed Training Paradigm

exchange all the gradients after trained with data

Data Parallelism:

Scale single-GPU training to **many GPUs or/and machines**

Reference: https://eng.uber.com/horovod/

Distributed Training by **Horovod**

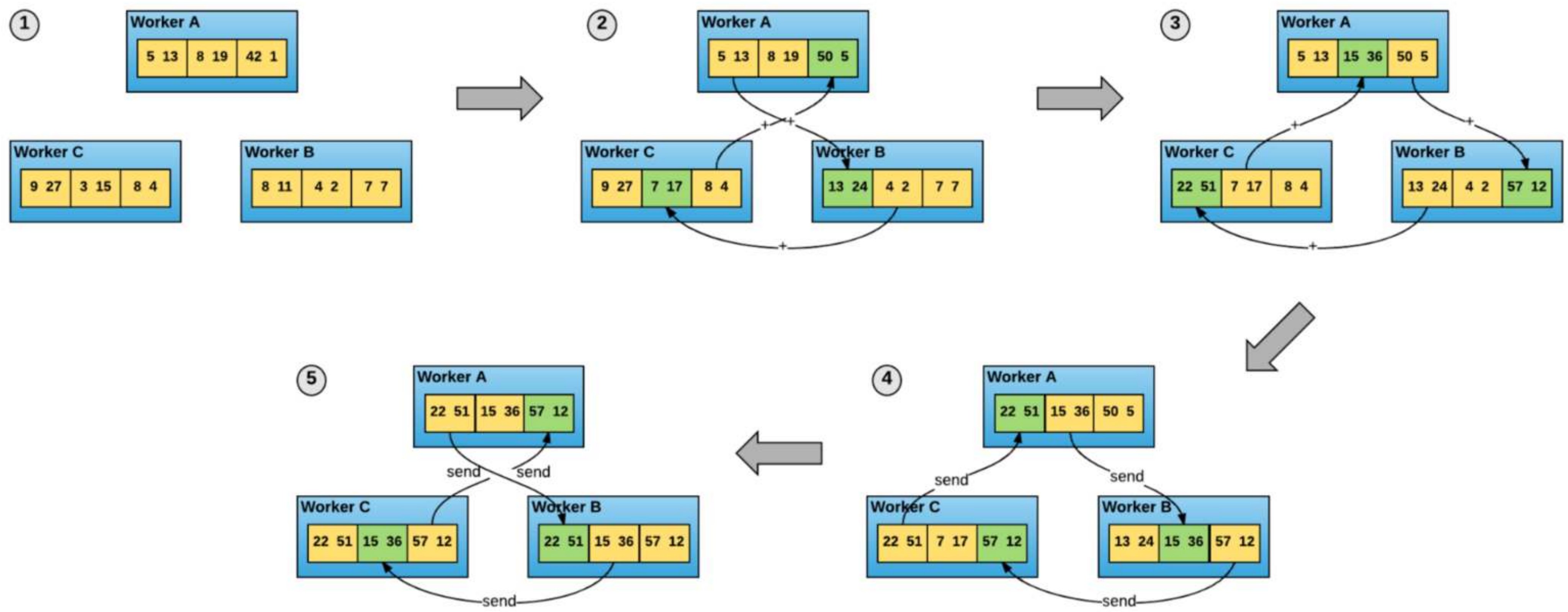


Figure 1: The ring-allreduce algorithm → **exchange a part of the gradients during training**

Reference: https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

Two-layer LSTM

Three-layer MLP

Horovod Distributed Training

```python
start = time.time()
# Horovod: initialize Horovod.
hvd.init()

# LSTM Model
model = Sequential()
model.add(LSTM(128, dropout=0.05, input_shape=(5 * 7, 3), return_sequences=True))
model.add(LSTM(64, dropout=0.05))
model.add(Dense(3, activation='sigmoid'))

# Horovod: adjust learning rate based on number of GPUs.
scaled_lr = 0.001 * hvd.size()
opt = tf.optimizers.Adam(scaled_lr)
opt = hvd.DistributedOptimizer(opt, backward_passes_per_step=1, average_aggregated_gradients=True)

# uses hvd.DistributedOptimizer() to compute gradients.
model.compile(loss='mean_squared_error', optimizer=opt, metrics=['mean_absolute_error', 'mean_squared_error', tf.keras.metrics.RootMeanSquaredError()],
    experimental_run_tf_function=False)

callbacks = [
    hvd.callbacks.BroadcastGlobalVariablesCallback(0),
    hvd.callbacks.MetricAverageCallback(),
    hvd.callbacks.LearningRateWarmupCallback(initial_lr=scaled_lr, warmup_epochs=3, verbose=1)
]

# Horovod: save checkpoints only on worker 0 to prevent other workers from corrupting them.
if hvd.rank() == 0:
    callbackEs = tf.keras.callbacks.EarlyStopping(monitor="val_loss", patience=10, verbose=1, restore_best_weights=True)
    callbackTb = tf.keras.callbacks.TensorBoard()
    callbacks.append(tf.keras.callbacks.ModelCheckpoint('./checkpoint-{epoch}.h5'))
verbose = 1 if hvd.rank() == 0 else 0

# Train the model.
model.fit(x_train, y_train, shuffle=True, steps_per_epoch=500 // hvd.size(), validation_data=(x_val, y_val), callbacks=[callbacks, callbackTb, callbackEs],
    epochs=50, verbose=verbose)
end = time.time()
print('Used training time: %f' %(end - start))
```

```python
# MLP Model
model = Sequential()
model.add(Flatten(input_shape=(5 * 7, 3)))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(3, activation='sigmoid'))
```

Early Stopping

Save Metrics to Tensorboard and save trained model

Contributions:

ML & DL Model building is achieved by **POON Bing Chun.**

Paper survey and **price movement prediction** are finished by **TSO Yiu Chuen.**

Part of Data pre-processing by Spark in Horovod and **Horovod distributed training** are performed by **LI Ka Faat** and **me.**

Reference:
https://github.com/verybighub/CS5488_Project/blob/main/Model%20Logs%20and%20Results/With_Horovod.py

Experimental Results (Stellar Currency)

21483 Training, 1193 Validation, 1194 Testing

Method	MAE	MSE	RMSE	Training Time (s)
LSTM w/o Horovod	0.009817	0.000137	0.011701	822.40
LSTM with Horovod (CPU)	0.012009	0.000265	0.016287	779.77 (-42.63)
MLP w/o Horovod	0.015290 (+0.005473)	0.000321 (+0.000184)	0.017924 (+0.0062223)	40.02
MLP with Horovod (CPU)	0.018781 (+0.006772)	0.000454 (+0.000189)	0.021310 (+0.005023)	25.62 (-14.4)

Figure 1: MLP without Horovod

Figure 2: LSTM without Horovod

Figure 3: MLP with Horovod

Figure 4: LSTM Model with Horovod

Future Work

Horovd On Spark
(Improved Horovod Performance)

→ Train Model on Spark Clusters

→ Directly Train Model
with PySpark DataFrames

→ Ease of Use

Future Work

- Combine the Prediction with Social Media
- Collect data from social media, e.g. Twitter,

Facebook, etc.

- Sentimental analysis
- Model: Transformer
- Based on public’s review on cryptocurrency

Conclusions

✓ Based on the **sliding window approach**, the **LSTM model** can effectively predict the price of crypto-currency with **superior performances** compared with MLP and other ML models.

✓ The time of **data pre-processing by Spark** can be as half as that by Pandas.

✓ **DL modes with Horovod support** can efficiently reduce training time.

Thanks a lot and any question?