Using OpenMP and Pthreads to optimize speedup for labeling training data for RNN's

By: Sebastian Matiz

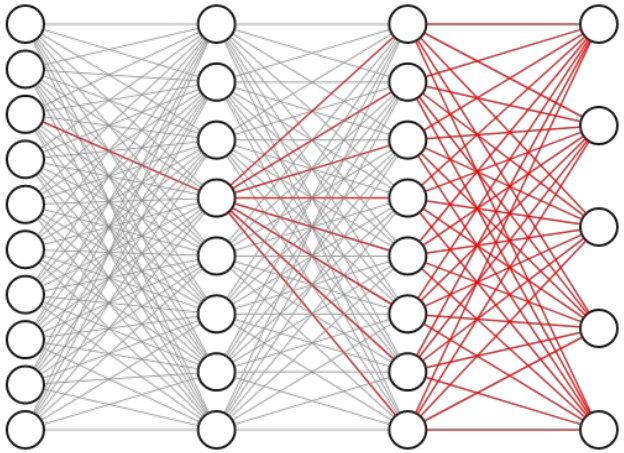
pthreads

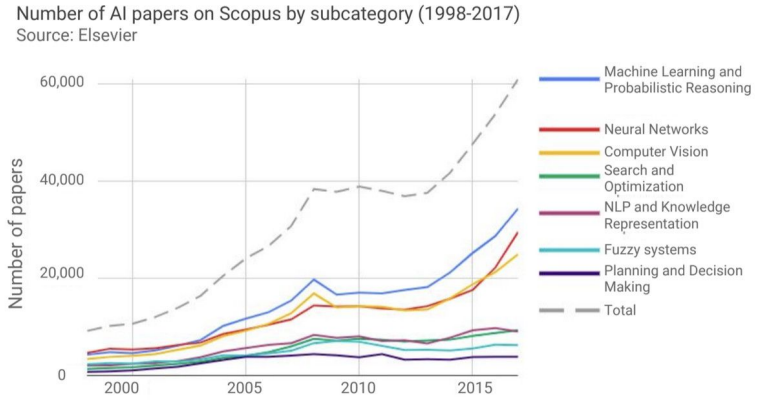
TensorFlow

Keras

Motivation

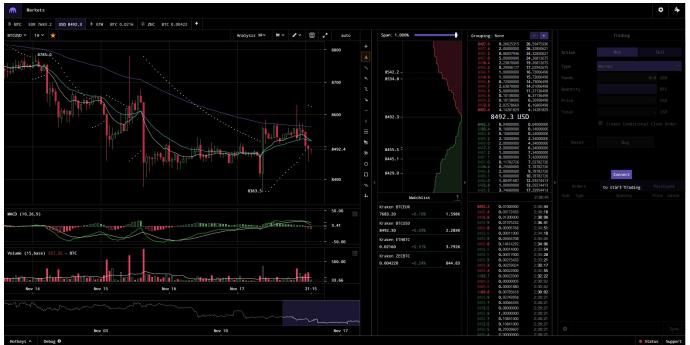
- Labeling training data is a very important part of training deep learning models
- Reliable models require A LOT of data to train and validate neural networks
- Efficient sourcing and distributing the work of labeling data is necessary to reduce the time it takes to start training the neural network





Project Background / Goals

- For our purposes, a single transaction is the buying and selling of a single security.
- Given price action data, I would like to label the best K transactions per day to achieve maximum profit.
- We will use this labeled data as training data for our neural network.
- The neural network will hopefully identify signals in the data that regular algorithms / humans may not be able to catch
- These signals will allow us to make accurate predictions



Data Collection

- Kraken is an exchange that deals with Crypto Currencies, BTC, ETH, etc
- They provide price data for each quarter for cryptocurrencies via CSV files
- In our case we will be labeling price data from the last quarter for Ethereum
- Columns:
  - 1: Date, UNIX EPOCH
  - 2: Price, USD
  - 3: Size

unlabeled_price_data_csv > ETHUSD.csv

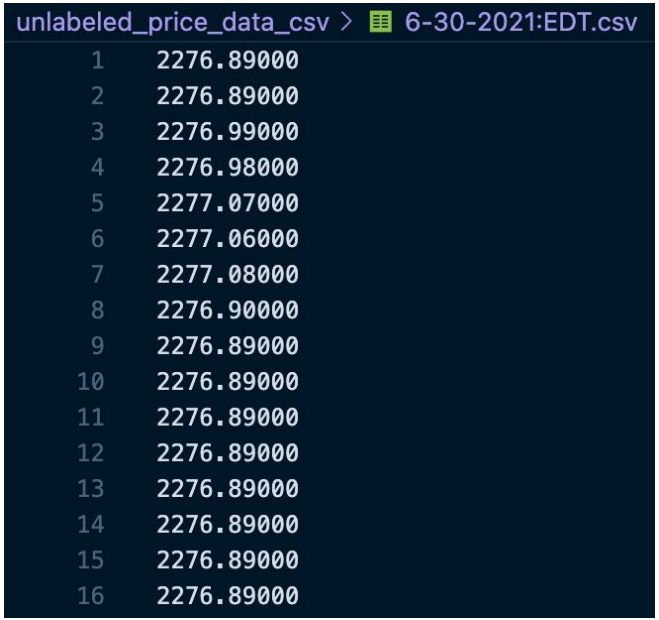
```
1625097602,2276.89000,0.06254287
1625097602,2276.89000,0.20000000
1625097606,2276.99000,0.33000000
1625097606,2276.98000,0.67000000
1625097608,2277.07000,0.33000000
1625097608,2277.06000,0.46232631
1625097609,2277.08000,0.05990412
1625097618,2276.90000,0.31000000
1625097618,2276.89000,0.48232631
1625097628,2276.89000,0.79232631
1625097635,2276.89000,15.38052259
1625097635,2276.89000,4.54866522
1625097635,2276.89000,6.59187090
1625097635,2276.89000,0.44257670
```

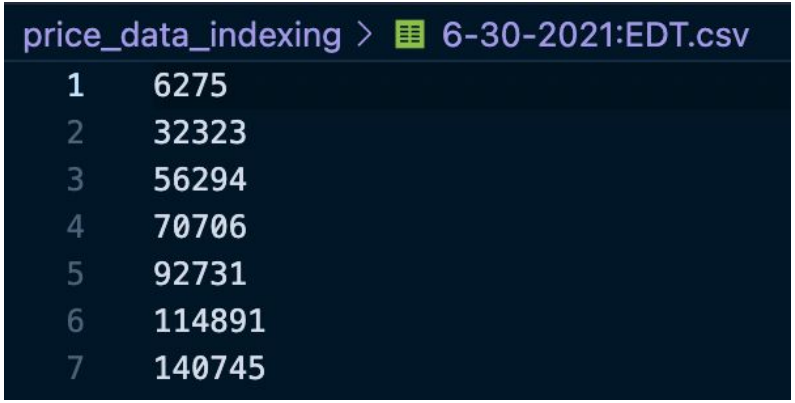
kraken

Pre Processing

- 1: Partition quarterly price data into weekly price data
- 2: Keep a dictionary of the indices of daily price data per week (for Threads to use)

Pre Processing Results





Parallelism / Distribution

- Each pthread will be given a specific week to label.
  - Each pthread will use OpenMP to label the best K transactions per day in the pthreads week
  - Each OpenMP thread will use a partition (1 day of the week) to call the labeling algorithm
  - The algorithm will label the best K transactions per day
  - Once all threads have labeled the shared memory. We will write to
  - All pthreads will join and labeling of data will be done

Labeling Scheme:

1 = Buy, 0 = Do Nothing, -1 = Sell

Example: Sell Signal Label

```
140742    2340.79,0
140743    2340.79,0
140744    2341.88,-1
140745    2340.83,0
```

Images of Pthread / OpenMP Usage

- Thread Args

```
24  struct thread_week_args {
25      int thread_id;
26      vector<int> partitions;
27      string week;
28  };
```

- Launching Threads

```
48      // launch 1 thread for each week
49      while (getline(date, currDate)) {
50          twa[i].thread_id = i;
51          twa[i].week = currDate;
52          index.open(indexFolder + "/" + currDate + ".csv");
53          string length;
54          vector<int> partitions;
55          // give indices of days in the week.. so we know which openMP thread gets what partition
56          while (getline(index, currIndexLength)) {
57              partitions.push_back(stoi(currIndexLength));
58          }
59          twa[i].partitions = partitions;
60          index.close();
61          // launch thread
62          thread_create = pthread_create(&threads[i], NULL, launchThreadsPerWeek, (void *) &twa[i]);
63
64          if (thread_create) {
65              cout << "Error: unable to create thread, " << thread_create << endl;
66              exit(-1);
67          }
68          i++;
69      }
70      // join threads
71      for (int i = 0; i < NUM_WEEKS; i++) {
72          pthread_join(threads[i], NULL);
73      }
74      date.close();
75      exit(EXIT_SUCCESS);
```

Images of Pthread / OpenMP Usage cont.

Inside of:

```
void * launchThreadsPerWeek(void * thread_week_args);
```

```
    omp_set_dynamic(0);     // Explicitly disable dynamic teams
    // Spawn numThreads threads for this parallel region only
    #pragma omp parallel num_threads(numThreads)
    {
        int start = omp_get_thread_num() == 0 ? 0 : partitions[omp_get_thread_num()-1];
        int end = partitions[omp_get_thread_num()];
        vector<double> currPartition;
        for (int i = start; i < end; i++) {
            currPartition.push_back(prices[i]);
        }
        int * labels = (int *) calloc(currPartition.size(), sizeof(int));
        int x = maxProfit(K, currPartition, labels);


        for (int i = start, j = 0; i < end; i++, j++) {
            buyOrSell[i] = labels[j];
        }
        #pragma omp critical
        {
            totalProfitWeek += x;
            // cout << "Thread: " << omp_get_thread_num() << ", Max Profit:  $" << x << endl;
        }
    }
    ofstream labeledData;
    labeledData.open(labeledDataFolder + "/" + week + ".csv");
    for (int i = 0; i < sizeOfFile; i++) {
        labeledData << prices[i] << "," << buyOrSell[i] << "\n";
    }
    labeledData.close();
    free(prices);
    free(buyOrSell);

    cout << "Total Profit for week, " << week << ": $" << totalProfitWeek << endl;
    return NULL;
}
```

Output when Running Labeling Code

```
smati@nbp-223-219 par_dist_prog_final_proj % g++-11 driver.cpp label.o -fopenmp -o run.o
smati@nbp-223-219 par_dist_prog_final_proj % ./run.o
[SOF: 140745
SOF: 149264
[SOF: 172375
SOF: 208426
SOF: 322628
SOF: 303518
SOF: 206214
SOF: 161407
SOF: 280234
SOF: 246180
SOF: 181587
SOF: 186952
SOF: 41277
Total Profit for week, 9-28-2021:EDT: $810
Total Profit for week, 6-30-2021:EDT: $1923
Total Profit for week, 7-7-2021:EDT: $1574
Total Profit for week, 8-21-2021:EDT: $2196
Total Profit for week, 9-21-2021:EDT: $3687
Total Profit for week, 7-14-2021:EDT: $1606
Total Profit for week, 9-14-2021:EDT: $2900
Total Profit for week, 7-21-2021:EDT: $2390
Total Profit for week, 8-14-2021:EDT: $2565
Total Profit for week, 8-28-2021:EDT: $3796
Total Profit for week, 8-7-2021:EDT: $3309
Total Profit for week, 7-28-2021:EDT: $3827
Total Profit for week, 9-7-2021:EDT: $4534
smati@nbp-223-219 par_dist_prog_final_proj %
```

Algorithm for Labeling

Inspiration for Labeling Algo:
https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iv/discuss/54118/C%2B%2B-Solution-with-O(n-%2B-klgn)-time-using-Max-Heap-and-Stack

Modified to label indices of when to buy and sell given a price vector.

We can find all adjacent valley/peak pairs and calculate the profits easily. Instead of accumulating all these profits like Buy&Sell Stock II, we need the highest k ones.

The key point is when there are two v/p pairs (v1, p1) and (v2, p2), satisfying v1 <= v2 and p1 <= p2, we can either make one transaction at [v1, p2], or make two at both [v1, p1] and [v2, p2]. The trick is to treat [v1, p2] as the first transaction, and [v2, p1] as the second. Then we can guarantee the right max profits in both situations, **p2 - v1** for one transaction and **p1 - v1 + p2 - v2** for two.

Finding all v/p pairs and calculating the profits takes O(n) since there are up to n/2 such pairs. And extracting k maximums from the heap consumes another O(klgn).

label_k_transactions.cpp

Speed Up Multi-Thread vs Sequential

Hardware Overview:

Model Name:	MacBook Pro
Model Identifier:	MacBookPro18,3
Chip:	Apple M1 Pro
Total Number of Cores:	8 (6 performance and 2 efficiency)
Memory:	16 GB

- ~1.2x Speed Up
- Thread Count
  - Pthreads: 13 Weeks
    - 13 Pthreads
  - OpenMP: ~7 days * 13 Pthreads
    - 91 OpenMP Threads
- Total Threads: ~104
- Theoretical Speed Up:
  - ~100x Speed Up per Quarterly Data

Algo is meant for machines that can handle ~100+ total threads. Maybe GPU parallelization can be further studied!

```
smati@nbp-223-219 par_dist_prog_final_proj % g++-11 driver_seq.cpp label.o -o run_seq.o
smati@nbp-223-219 par_dist_prog_final_proj % ./run_seq.o
Total Profit for week, 6-30-2021:EDT: $1923
Total Profit for week, 7-7-2021:EDT: $1574
Total Profit for week, 7-14-2021:EDT: $1606
Total Profit for week, 7-21-2021:EDT: $2390
Total Profit for week, 7-28-2021:EDT: $3827
Total Profit for week, 8-7-2021:EDT: $3309
Total Profit for week, 8-14-2021:EDT: $2565
Total Profit for week, 8-21-2021:EDT: $2196
Total Profit for week, 8-28-2021:EDT: $3796
Total Profit for week, 9-7-2021:EDT: $4534
Total Profit for week, 9-14-2021:EDT: $2900
Total Profit for week, 9-21-2021:EDT: $3687
Total Profit for week, 9-28-2021:EDT: $810
Total Time Seq: 1188 milliseconds
smati@nbp-223-219 par_dist_prog_final_proj %
```

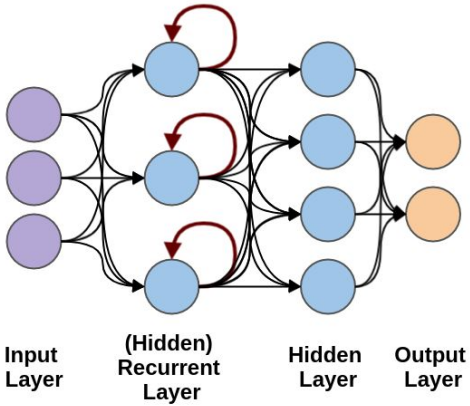
```
smati@nbp-223-219 par_dist_prog_final_proj % g++-11 driver.cpp -fopenmp label.o -o run.o
smati@nbp-223-219 par_dist_prog_final_proj % ./run.o
Total Profit for week, 9-28-2021:EDT: $810
Total Profit for week, 6-30-2021:EDT: $1923
Total Profit for week, 7-7-2021:EDT: $1574
Total Profit for week, 8-21-2021:EDT: $2196
Total Profit for week, 7-14-2021:EDT: $1606
Total Profit for week, 9-14-2021:EDT: $2900
Total Profit for week, 9-21-2021:EDT: $3687
Total Profit for week, 7-21-2021:EDT: $2390
Total Profit for week, 8-14-2021:EDT: $2565
Total Profit for week, 8-28-2021:EDT: $3796
Total Profit for week, 9-7-2021:EDT: $4534
Total Profit for week, 8-7-2021:EDT: $3309
Total Profit for week, 7-28-2021:EDT: $3827
Total Time Multi-Thread: 1010 milliseconds
smati@nbp-223-219 par_dist_prog_final_proj %
```

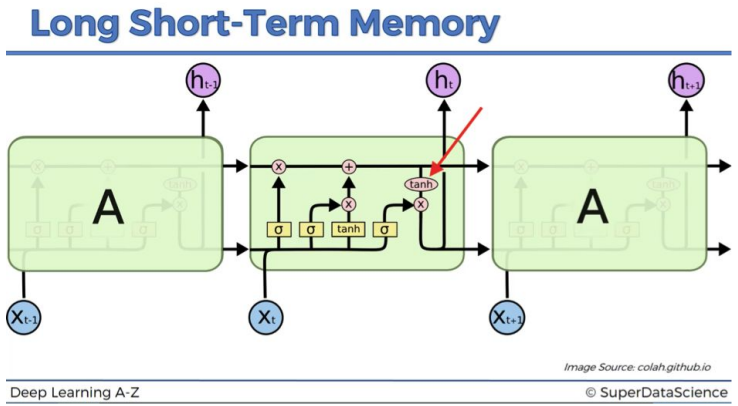
RNN / LSTM

Research:

The Long Short-Term Memory network, or LSTM network, is a recurrent neural network that is trained using Backpropagation Through Time and overcomes the vanishing gradient problem. As such, it can be used to create large recurrent networks that in turn can be used to address difficult sequence problems in machine learning and achieve state-of-the-art results. - https://machinelearningmastery.com

-





TensorFlow -> Keras -> Python

Why TensorFlow

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

About →

Easy model building

Build and train ML models easily using intuitive high-level APIs like Keras with eager execution, which makes for immediate model iteration and easy debugging.

Robust ML production anywhere

Easily train and deploy models in the cloud, on-prem, in the browser, or on-device no matter what language you use.

Powerful experimentation for research

A simple and flexible architecture to take new ideas from concept to code, to state-of-the-art models, and to publication faster.

Python Script

```
37    # get data
38    dataset = pd.read_csv('./labeled_price_data_csv/6-30-2021:EDT.csv')
39    dataset.columns = ['prices','action']
40    values = dataset.values
41
42    # ensure all data is float
43    values = values.astype('float32')
44    # normalize features
45    scaler = MinMaxScaler(feature_range=(0, 1))
46    scaled = scaler.fit_transform(values)
47    # frame as supervised learning
48    reframed = series_to_supervised(scaled, 1, 1)
49    print(reframed.head())
50
```

Python Script cont.

```
51  # split into train and test sets
52  values = reframed.values
53  n_train_hours = 365 * 24
54  train = values[:n_train_hours, :]
55  test = values[n_train_hours:, :]
56  # split into input and outputs
57  train_X, train_y = train[:, :-1], train[:, -1]
58  test_X, test_y = test[:, :-1], test[:, -1]
59  # reshape input to be 3D [samples, timesteps, features]
60  train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
61  test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
62  print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
63
64  # design network
65  model = Sequential()
66  model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
67  model.add(Dense(1))
68  model.compile(loss='mae', optimizer='adam')
69  # fit network
70  history = model.fit(train_X, train_y, epochs=50, batch_size=72, validation_data=(test_X, test_y), verbose=2, shuffle=False)
71
```

Python Script cont. pt2

```
71
72  # make a prediction
73  yhat = model.predict(test_X)
74  test_X = test_X.reshape((test_X.shape[0], test_X.shape[2]))
75  # invert scaling for forecast
76  inv_yhat = numpy.concatenate((yhat, test_X[:, 1:]), axis=1)
77  inv_yhat = scaler.inverse_transform(inv_yhat)
78  inv_yhat = inv_yhat[:,0]
79  # invert scaling for actual
80  test_y = test_y.reshape((len(test_y), 1))
81  inv_y = numpy.concatenate((test_y, test_X[:, 1:]), axis=1)
82  inv_y = scaler.inverse_transform(inv_y)
83  inv_y = inv_y[:,0]
84  # calculate RMSE
85  rmse = math.sqrt(mean_squared_error(inv_y, inv_yhat))
86  print('Test RMSE: %.3f' % rmse)
```

Results

The **root-mean-square deviation** (**RMSD**) or **root-mean-square error** (**RMSE**) is a frequently used measure of the differences between values (sample or population values) predicted by a model or an estimator and the values observed - Wikipedia

```
122/122 - 1s - loss: 0.0048 - val_loss: 0.0163 - 684ms/epoch - 6ms/step
Epoch 45/50
122/122 - 1s - loss: 0.0034 - val_loss: 0.0149 - 678ms/epoch - 6ms/step
Epoch 46/50
122/122 - 1s - loss: 0.0032 - val_loss: 0.0148 - 677ms/epoch - 6ms/step
Epoch 47/50
122/122 - 1s - loss: 0.0034 - val_loss: 0.0136 - 678ms/epoch - 6ms/step
Epoch 48/50
122/122 - 1s - loss: 0.0038 - val_loss: 0.0132 - 677ms/epoch - 6ms/step
Epoch 49/50
122/122 - 1s - loss: 0.0028 - val_loss: 0.0137 - 677ms/epoch - 6ms/step
Epoch 50/50
122/122 - 1s - loss: 0.0029 - val_loss: 0.0130 - 675ms/epoch - 6ms/step
Test RMSE: 6.693
(mlp) smati@nbp-223-219 par_dist_prog_final_proj %
```

Future

- Have this implementation be built / modified for less powerful machines
- Utilize GPU Parallelization
- Parallelize Training of RNN*
- Create a production ready model that can make transaction with real money via kraken API
- Thank you for listening!

REST API (1.0.0)