Stock Market Trading with LSTM Analysis

Franklin Doane
dept. of Computer Science
Tennessee Technological University
Cookeville, United States
fgdoane42@tntech.edu

I. Dataset Creation

II. Training Setup

A. Load Config

1) Load: Load config as json object with path from positional arg

2) Unpack: Unpack config keys into variables

B. Setup regulator

1) Init regulator: Pass starting learning rate and other kwargs from config

C. Load dataset

1) Init dataset object: The dataset object is initialized and given config. Then the load dataset method is called which does the following.

2) Verify files: Loads all csvs in dataset and verifies a minimum length. A legacy feature at this point.

3) Split files into task groups: List of valid files is split into groups to have each group loaded with threads.

4) Open dataset thread pool: Open thread pool to load files.

5) Load csv files: Open every file from directory listed in config. Load as pandas dataframe with date index column and put them all in a list.

6) Calculate indicators: Calculate all additional features listed in configs. Cut out any data at the front of the dataframes that is missing these features (e.g. for rolling avgs).

7) Create date feature: 1hot encode day of the week and add the new weekday features to all dataframes.

8) Slice data into training examples: Use rolling window to cut each dataframe into training example dataframes with n day history. All examples get added to a list and labels are made into a dataframe of their own.

9) Normalize examples: Each dataframe training example is normalized using norm function from config.

10) Create tensor input: Each dataframe is converted to numpy then a tensor with the datatype and device listed in the config. All labels are converted a single tensor as well.

D. Load model

1) Dynamically load model class: The Python file containing the model class is loaded by name using importlib.

2) Call init: Init custom model class. The custom model class is passed a name, starting LR, and input feature count from dataset. The model inherits from torch.nn.Module which is also inited.

3) Setup pytorch objects: Model class init sets up LSTM object and Sequential object for forward passes, a default optimizer, a null loss func and an empty history.

4) Set optimizer: Call model set optim method to initialize and save optimizer to model class.

5) Send to device: Set model to device specified in config.

E. Setup scheduler

1) Init scheduler: Pass optimizer from model. Give it kwargs from config

F. Setup dataloader

1) Check for test set: Check's to see if the model save config contains test set indices. For loading a partially trained model. Indices get set after init if none are there. Pretty much a vestigial feature.

2) Call init: Init object and pass in dataset object (and test set indices if applicable)

3) Pick test indices: If not test set indices were passed, create list of indices and either random sample indices (legacy) or slice chunk off the front to create test set indices.

4) Seperate test set: Use indices to make a mask and remove test examples from all indices to get training set indices

5) Shuffle train set: random.shuffle train set indices

6) Split batches: Split train set indices into batches using a list comprehension and slicing with option to drop last partial batch. Transforms list[int] -¿ list[list[int]]. Casts result to tensor.

7) Setup first inputs: Create empty list for inputs and labels. Prefetch first n batches by indexing them out of dataset and adding them to input and label sets. Remove these indices from batch indices.

8) Load test set: Index test set and labels out of data and Store

III. Training Loop (per epoch)

A. Setup

1) Get regulator coefs: Pass regulator lr and get the coefficient for each logit. Coefficients will be stepped down if LR has changed

2) Update loss: Call function to init custom MSE loss and pass it the regularization coefficients. Call method to set new loss for model class.

B. Train batches

1) Unpack dataloader: Each iteration grabs a batch from the loaded set of batches. If the loaded set is empty, the dataloader loads indexes more batches out of the dataset. Continues until all batch indices are popped.

2) Forward pass: Call model forward with batch inputs

3) Call backward pass: Pass model backpass method the logits and batch labels (with added time series dimension). Starts by putting model in train mode.

4) Calc loss: Zero optimizer gradient then run model's built-in loss function. Backwards' the loss.

5) Optimize: Step optimizer.

C. Save

1) Checkpoint model: Save epoch copy of model as pkl to model save dir. Includes model params as well as model training history.

D. Train-time eval

1) Eval setup: Set model in eval mode. Create data structures for saving outcomes and losses.

2) Update scheduler: Send average example loss to scheduler.

IV. Evaluation