Things to propose:

- Rewrite the args code to accept a list of arguments, instead of a single args argument

Steps to properly install cellbox on the cluster:

- Create conda with python 3.8: `conda create -n "cellbox" python==3.8.0`
- In this environment, write `pip --version` to make sure it's using conda pip
- `git clone https://github.com/Mustardburger/CellBox.git <folder_name>`
- `cd /<folder_name>/cellbox`
- `pip install -e .`
- On the command line, try opening python and import cellbox. When I did this, it threw `module not found 'packaging'`. I then `pip install packaging` and everything works now

Problems of replicating training model results:

- Loss function in pytorch did not have l1_lambda and l2_lambda (**resolved**)
- Why does the train loss increase during training?? (**resolved**, have to pass in **model.named_parameters()** to the loss function)
- In the original code, which loss is the gradient computed with respect to? (**resolved,** the total loss)
- Add a piece of code in **def train_model()** in **train.py** to load the trained model for later substages (**resolved**)
- The code at line 70 in **train_torch.py** is currently very cumbersome. Same for other places where there's an **args.loss_fn**.
- The code for checking **args.pert_form** to be either **by u** or **fix x** is also cumbersome.
- The ODE solver has some locations where the two versions greatly differ.
- The softplus function is defined differently in two versions.

How the data is created:

Drug concentration:

- In single agent perturbations, each drug is applied at two different concentrations, IC40 and 2 × IC40.
- In validation experiments, (+)JQ1 (Cayman Chemicals, Ann Arbor, MI) and the U.S. Food and Drug Administration (FDA)-approved RAFi PLX4032 (Selleckchem, Houston, TX) are used.

Proteomic nodes

- Use RPPA to create the matrix
- Antibody staining intensities are quantified using the MicroVigene automated RPPA module (VigeneTech, Inc., Carlisle, MA) and the standard RPPA protein concentration normalization procedure (Neeley et al., 2009) is followed.

- The proteomic readouts are log normalized with respect to the corresponding untreated condition readouts. We have eliminated those readouts with intra- or inter-slide coefficient of variation >0.15 (i.e., low reproducibility) and low degree of staining by antibodies. 100 proteomic entities are chosen for further analysis.

- From these 100 proteomic entities, entities that do not respond to at least a single perturbation condition from the network models are excluded using signal-to-noise detection. This ends up with the final 82 proteomic entities.

Phenotypic nodes:

- Cell viability and cell cycle progression are measured using the resazurin assay (72 hr after drug treatment) and flow cytometry analysis (24 hr after drug treatment), respectively. The percentage of cells in the G1, G2/M, and S phases and sub-G1 fraction are recorded based on the respective distribution of DNA content in each phase.

- The percentage of cells in the G1, G2/M, and S phases and sub-G1 fraction are recorded.

- The final value seems to be log-normalized, which takes on negative values

The sparse data is used for single-cell analysis

Things to add in the issue:

- Line 71 and 72 in **train.py**, whether **loss_valid_i** and **loss_valid_mse_i** is evaluated on only one batch of the validation, or the whole validation dataset
- The **eval_model** function returns different values with different calls. At line 95 it returns both the total and mse loss for the validation set. At line 102 it returns only the mse loss for the test set. And at line 222 it returns the predictions (**y_hat**) for the test set.
- The **pert_form** option at line 89 and 95 in **model.py**:
    - If “by u”, then the input to the ODE solver will be a zero-th vector
    - If “fix x”, then the input to the ODE solver will be a mini-batch of the perturbation data
- The **record_eval.csv** file generated after training has **test_mse** column to be None
- **random_pos.csv** is a file to store the index of the perturbation condition. Does it indicate how the data is split?
- **y_hat.loss.csv** contains the prediction for the perturbation conditions for all nodes (molecular and phenotypic); but it does not indicate which perturbation condition maps to the actual condition index
-

Details about the models:

- The **PertBio** class is just an abstraction over the tensorflow operators. It does not inherit anything from tensorflow’s objects.
- When **model.factory** is called, the corresponding model class is instantiated, then **.build()** is immediately called on that instance. The way the computational graph is set up is the following:

    - **.__init__()** is called: this creates tf placeholders for dataloaders (**self.iter_train** and **self.train_x, self_train_y**)
    - **.build()** is called. Inside **.build()**:
        - **.get_variables()** is called to instantiated the params of each model
        - **.forward()** is called to do one forward pass of the model
        - **.get_ops()** is called to calculate the loss value and to run the optimizer
    - The computational graph therefore starts with the operation **self.iter_train.get_next()** and ends up calculating the loss
    - The **feed_dict** argument has key: value of {tf placeholder for pert_in: tf placeholder for expr_out}. This is for **self.iter_train.get_next().**
    - In **CellBox**, the **.forward** takes in both **y0** and **mu**, whereas for other models, it only takes in **mu**. So **mu** here is **self.train_x**
- In **train.py**:
    - **eval_results** is the prediction of the model on the test set:
        - It returns a list equal to the length of the test data loader
        - Each element in the list is a matrix of shape (batch_size, 99). In the original code, **eval_results** either returns the predicted expression values or the loss using that data loader.
    - **Screenshot** object in the end is a dict with the following keys:
        - **"W"**: the weight matrix
        - **"b"**: the bias vector
        - **"yhat"**: a pd.DataFrame with num columns equal to 99, or number of observed nodes, and num rows I think equal to the num perturbed conditions in test dataloader
    - When
- In **kernel.py:**
    - The original ODE from the paper is as follows:

$$\frac{\partial {x_i}^{\mu}(t)}{\partial t} = \epsilon_i \varphi\left(\sum_{j \neq i} w_{ij} {x_j}^{\mu}(t) + {u_i}^{\mu}(t)\right) - \alpha_i {x_j}^{\mu}(t)$$

    in which:
        - phi() is referred to as the envelope in the code (or **args.envelope_fn**). There are several implementations of phi, including the tanh, linear, exponential, …
        - The multiplication of w times x inside the envelope is referred to as the weighted sum, defined in **def weighted_sum(x)**
        - The whole right hand side of the ODE is defined by setting **args.envelope** to either 0, 1, or 2, as it changes slightly the epsilon and alpha
    - The ODE solver is defined in **def get_ode_solver(args)** and **args.ode_solver**. There are several types, including Heun (used in the paper), Euler, Runge-Kutta-4, and midpoint.
- Output directory:

- **/results**
    - **/Example_RP_<hash string>**: a folder that contains a specific run
        - **/seed_000:**
            - **/record_eval**: The csv file that contains information about the model training, such as epoch, iter, train_loss, valid_loss, train_mse, valid_mse, test_mse, time_elapsed

          config.json: The config used for this run

Github Actions realizations:

- In **setup.py**, __version__ is not defined
- In the master branch of CellBox main repo, the return type for **get_ode_solver()** in **kernel.py** is ill-defined
- Why do we need linting test for Github Actions?

Details from paper:

- Here, we used a perturbation dataset for the melanoma cell line SK-Mel-133 (Korkut et al., 2015), which contains molecular and phenotypic response profiles of cells treated with 12 different drugs and their pairwise combinations (Figures 1A and S1). For each of the 89 perturbation conditions, levels of 82 selected proteins and phosphoproteins were measured …
- In order to test the prediction performance of this training scheme, we randomly selected 70% of the perturbation data (n = 62 conditions) for training and withheld the rest 30% (n = 27 conditions) for testing.
- In the single-to-combo analysis, all single-drug-treatment conditions were used for training, and predictions were made on all combinatorial drug conditions.
- In leave-one-drug-out cross-validation, all the combination conditions containing the treatment of a particular drug with or without the corresponding single-drug conditions were withheld and the rest of the conditions were used for training.
- **The resulting dataset has 89 perturbation conditions and 99 observed nodes (82 protein and phosphoproteins, 5 phenotypes, and 12 drug activity).**

Key-value things in cfg:

Key	Value datatype	Where?	Description
cfg.dataset	dict	dataset.py	Contains numpy arrays to later feed into feed_dicts
cfg.iter_train	tf.compat.v1.data.make_initializable_iterator()	dataset.py	Like a DataLoader for training data
cfg.iter_monitor	tf.compat.v1.data.make_initializable_iterator()	dataset.py	Like a DataLoader for training data

cfg.pert_in	placeholder tf variables with size [None, cfg.n_x]		
cfg.expr_out	placeholder tf variables with size [None, cfg.n_x]		
cfg.pert_file	str	config.py	Dir to perturbation matrix
cfg.expr_file			
cfg.loo	pd DataFrame	dataset.py	??? (can ask Bo on this)
cfg.drug_index	int	main.py	An index among 99 drugs to be left out
cfg.l1_lambda	float	dataset.py	
cfg.l2_lambda	float	dataset.py	
cfg.lr	float	dataset.py	Learning rate
cfg.stages	list of dictionary	main.py	Each dict contains another dict about each substage
cfg.sub_stages	list of dictionary	main.py	Each dict contains lr_val, l1_lamb and n_iter for each substage

Code specifics:

- **tf.sparse.SparseTensor**: a class of Tensorflow tensor that includes 3 submatrices: **indices, values,** and **dense_shape**:
    - **indices** specifies the coordinate in the sparse tensor with non-zero value
    - **values** specifies the value of the sparse tensor at each coordinate in **indices**
    - **dense_shape** specifies the shape of the sparse tensor
- **test.py:**
    - This file contains the tests for comparing Tensorflow and Pytorch dataloaders. It uses pytest.

    - To run it, first **pip install pytest** if pytest is not in env. Then simply write **source load cellbox-env** then **python test.py**
    - **test.py** also utilizes many utils functions in the **/test_utils** folder
- The val dataloader (**cfg.iter_monitor**) is set to repeat, meaning the data can be fetched from it indefinitely.

Test cases matrix:

Model	Data Partitioning	Hyperparams JSON file	Test properties
LinReg	Random Partition	configs_dev/Example.random_partition.json	Both code runs

DataLoader:

- Run dataloader for random partition, s2c, and loo, and do the same thing for Pytorch, at different seeds. If the generated random_pos.csv is similar, then the test passes

ODE:

- Simulate Tensorflow's ODE with different envelopes and ODE solver, using the same param matrices and input. Do the same for Pytorch. Simulate up to 100 time steps only. If the mean (or median) discrepancy is smaller than a threshold, then the test passes

Model:

- Check whether the mask has been applied correctly
- Using the same param weights and input, if the Tensorflow and Pytorch output are similar by a threshold, then the test passes

Miscellaneous

- Check if the output folder contains files of the correct structure
-

Edge cases:

- When **cfg.sparse_data** is set to True

    - Use weight decay instead of manually changing the learning rate
    - Dynamic building
    - Github Actions

More information:

- expr_index.txt: information about each perturbation condition. In the first column, the values on the far left (901, AK, HN, …) separated by "|" are the drugs and there are 12 of them. The total number of rows in this file is equal to 89. Each drug has its own condition + condition in combination of other drugs

- loo_label.csv: information about what row in expr.csv and pert.csv corresponds to what drug combination. For example, row 35 (5,0) means the condition of drug at index 5 only, whereas row 36 (5,3) means the condition of drug at both index 5 and 3.

Cancer Institute and Library
National Library of Medicine

Cancer patient data, data comes from cell lines tested with drugs, network analysis.
Work with clinicians at the NIH hospital to study rare cancers.

Ongoing challenge to get more data

```
git fetch origin
git checkout 51-update-cellbox-readme
```

```
    - name: Lint with ruff (only Python 3.7+)
      # Run if not on master
      if: github.ref != 'refs/heads/master'
      run: |
        # stop the build if there are Python syntax errors or undefined names
        ruff --format=github --select=E9,F63,F7,F82 --target-version=py37 .
        # default set of ruff rules with GitHub Annotations
        ruff --format=github --target-version=py37 .
```

Things needed in kernel file args:

- n_x
- envelope_form (1)
- envelope_fn (1)
- polynormial_k (1)
- ode_degree (2)
- envelope (2)
- ode_solver (3)
- dT
- n_T
- gradient_zero_from: either None or args.n_activity_nodes (defined in model_torch.py)

Notes, Jul 26, 2023:

- The way softplus is done in pytorch is currently unsound

Notes, Aug 17, 2023:

- The clean-code branch is ready to be pushed to the main branch eventually
    - Remove everything related to Tensorflow
    - Improve readability for train_torch.py
    - Remove comment sections for some pieces of code
    - Improve documentation for functions
- Another branch to finalize tests: