Frank Wolfe for NN training

Goals

1. Test wether SFW is suitable as a replacement for other popular optimizers like Adam or SGD.
    - 3 optimizers: SFW, Adam, SGD
2. Compare various constraint options for SFW:
    - 5 constraints: L_1, L_2, L_∞, K-sparse polytope, K-norm ball
3. See if SFW can be applied to the Chemistry problem to improve the result or insights gained.

Bonus:

1. Test parameters of constraints (other L_p norms, other K values), possibly plots of performance over p/ K.

Testing methodology

Goal 1. requires training multiple models with different optimizers.

Options:

- Apply automatic hyperparameter tuning to enable a fair comparison between optimizers as they might need different settings to work well.
- Train models on various datasets to ensure the results are transferable.
    - MNIST
    - fashion MNIST
    - ChemReg/ ChemClass

Implementation

2 options:

1. use existing code from paper, add automated model training of multiple models and hyperparameter tuning
   pros:

    - better automatic hyperparameter tuning with many options
    - likely faster training due to more efficient hyperparameter search
    - more flexible since most code is new

   cons:

    - potentially more work
    - fewer result output options already implemented

2. adapt code from Bachelor thesis to work with pytorch models **pros:**

    - hyperparameter search already implemented (although slow and not very flexible)
    - win ratio tables already implemented as useful result output with very detailed information

- training time already measured and output

cons:

- getting result ouput may be difficult
- plots need to be implemented from scratch

Adapting Bachelor thesis code

The program is structured into three steps the user needs to perform:

1. create a config file and use it to train models with all combinations of the given hyperparameters
2. summarize the tracked metrics into more useful statistical values (extract best/ last accuracy value, summarize multiple runs, etc.)
3. create visualizations of the results. Current code generates two kinds of tables:
    - win ratio tables: compare the impact each hyperparameter has on the tracked metrics (training time, accuaracy, loss, etc.)
    - best/worst tables: list the best few and worst few models for each tracked metric with their hyperparameter values

Python file overview

Here, I list which files are used in the three steps described above with a very brief description of their purpose.

1. Set parameters for training runs
    - `dense_parameter_study_[…].py` - config files to set hyperparameters for studies
    - `dense_parameter_study.py` - controls parameter study
    - `model_builder.py` - creates models (currently uses Tensorflow)
    - `model_trainer.py` - trains models (currently uses Tensorflow)
    - `model_tester.py` - evaluates models on test data (currently uses Tensorflow)
    - `pickleable_history.py` - stores information about training runs
    - `file_management.py` - handles saving and loading of most files
    - `helper_functions.py` - generic helper functions
    - `my_adam.py` - custom Adam optimizer
    - `c_adam.py` - custom C-Adam optimizer
    - `c_adam_hat.py` - custom C-Adam optimizer
    - `fast_c_adam.py` - custom C-Adam optimizer
2. Summarize information of multiple runs
    - `data_analysis.py` - controls data analysis
    - `batch_summary.py` - extracts useful information from history files
    - `load_md_files.py` - loads information about model parameters
    - `top_n_list.py` - simplifies storing data for best/worst tables
    - `file_management.py` - handles saving and loading of most files
    - `helper_functions.py` - generic helper functions
3. Create result visualizations

- `result_output.py` - controls result output, creates win ratio tables and best/worst tables
- `vertical_best_worst_output.py` - creates best/worst tables with latex formatting
- `loss_conversion.py` - converts loss values between different loss functions
- `file_management.py` - handles saving and loading of most files
- `helper_functions.py` - generic helper functions

Changes to adapt code to pytorch

Adjusting this program to work with the new pytorch models would require:

- update `model_trainer.py`, `model_builder.py` and `pickleable_history.py` to work with pytorch
- add constraint settings to `dense_parameter_study.py`
- add new result output options that produce graphs. Some graphs may need access to information not currently saved in step 2. (e.g. accuracy over time during training)
    - may require changes to `batch_summary.py`