DeepOBS Documentation

Release 1.1.0

Frank Schneider

Sep 25, 2019

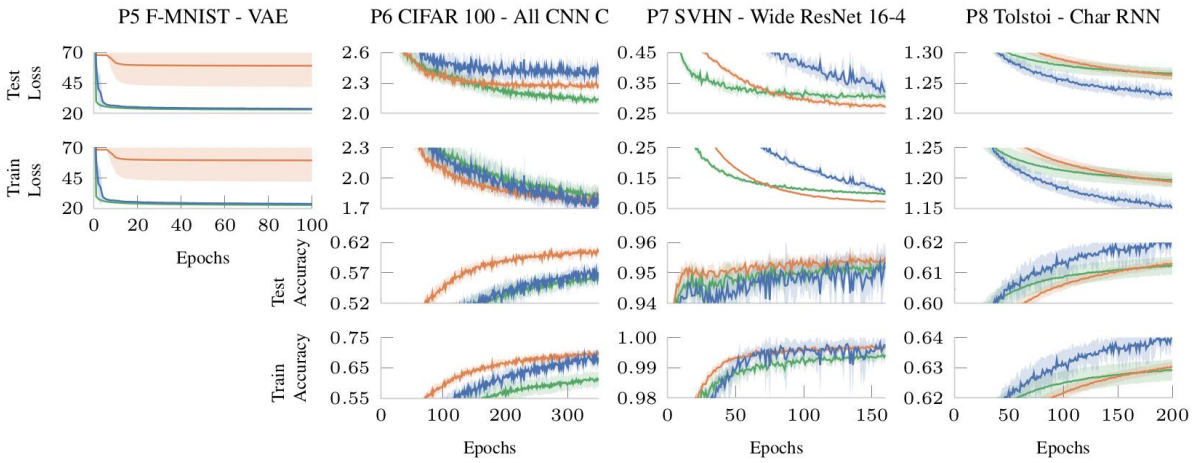
User Guide

DeepOBS is a benchmarking suite that drastically simplifies, automates and improves the evaluation of deep learning optimizers.

It can evaluate the performance of new optimizers on a variety of **real-world test problems** and automatically compare them with **realistic baselines**.

DeepOBS automates several steps when benchmarking deep learning optimizers:

- Downloading and preparing data sets.
- Setting up test problems consisting of contemporary data sets and realistic deep learning architectures.
- Running the optimizers on multiple test problems and logging relevant metrics.
- Reporting and visualization the results of the optimizer benchmark.



The code for the current implementation working with **TensorFlow** can be found on GitHub.

We are actively working on a **PyTorch** version and will be releasing it in the next months. In the meantime, PyTorch users can still use parts of DeepOBS such as the data preprocessing scripts or the visualization features.

CHAPTER 1

Quick Start

DeepOBS is a Python package to benchmark deep learning optimizers. It currently supports TensorFlow but a PyTorch version is currently in development.

We tested the package with Python 3.6 and TensorFlow version 1.12. Other versions of Python and TensorFlow (>= 1.4.0) might work, and we plan to expand compatibility in the future.

1.1 Installation

You can install the latest stable release of DeepOBS using *pip*:

```
pip install git+https://github.com/fsschneider/DeepOBS.git
```

Note: The package requires the following packages:

- argparse
- numpy
- pandas
- matplotlib
- matplotlib2tikz
- seaborn

TensorFlow is not a required package to allow for both the CPU and GPU version. Make sure that one of those is installed.

Hint: We do not specify the exact version of the required package. However, if any problems occur while using DeepOBS, it might be a good idea to upgrade those packages to the newest release (especially matplotlib and numpy).

1.2 Set-Up Data Sets

After installing DeepOBS, you have to download the data sets for the test problems. This can be done by simply running the *Prepare Data* script:

```
deepobs_prepare_data.sh
```

This will automatically download, sort and prepare all the data sets (except ImageNet) in a folder called `data_deepobs` in the current directory. It can take a while, as it will download roughly 1 GB.

Note: The ImageNet data set is currently excluded from this automatic downloading and preprocessing. ImageNet requires a registration to do this and has a total size of hundreds of GBs. You can download it and add it to the `imagenet` folder by yourself if you wish to use the ImageNet data set.

Hint: If you already have some of the data sets on your computer, you can only download the rest. If you have all data sets, you can skip this step, and always tell DeepOBS where the data sets are located. However, the DeepOBS package requires the data sets to be organized in a specific way.

You are now ready to run different optimizers on various test problems. We provide a *Simple Example* for this, as well as our *Suggested Protocol* for benchmarking deep learning optimizers.

1.3 Contributing to DeepOBS

If you want to see a certain data set or test problem added to DeepOBS, you can just fork DeepOBS, and implemented following the structure of the existing modules and create a pull-request. We are very happy to expand DeepOBS with more data sets and models.

We also invite the authors of other optimization algorithms to add their own method to the benchmark. Just edit a run script to include the new optimization method and create a pull-request.

Provided that this new optimizer produces competitive results, we will add the results to the set of provided baselines.

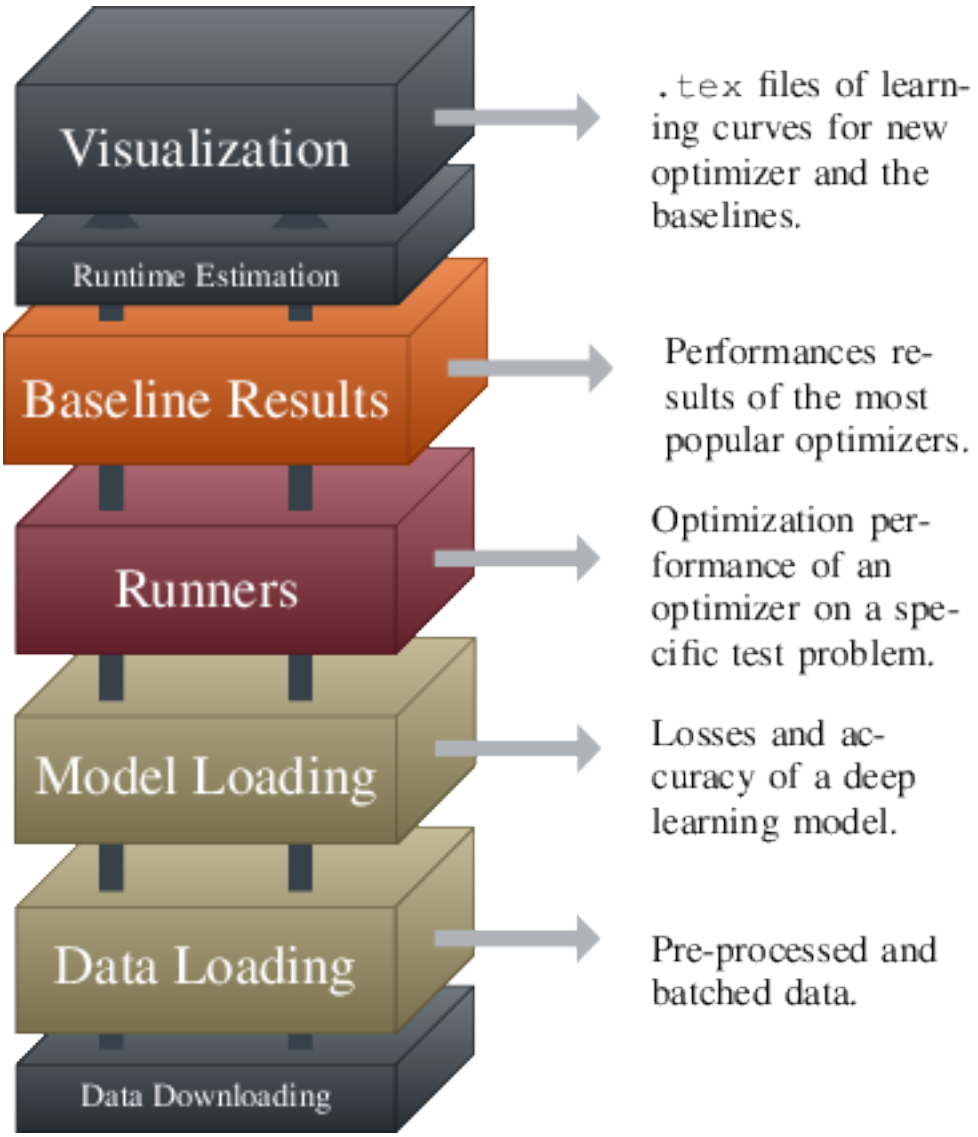
CHAPTER 2

Overview

DeepOBS provides modules and scripts for the full stack required to rapidly, reliably and reproducibly benchmark deep learning optimizers.

Here we briefly described the different levels of automation that DeepOBS provides. While, they are built hierarchically, they can be used separately. For example, one can use just the data loading capabilities of DeepOBS and built a new test problem on top of it.

A more detailed description of the modules and scripts can be found in the API reference section.



2.1 Data Downloading

DeepOBS can automatically download and pre-process all necessary data sets. This includes

- MNIST
- Fashion-MNIST (FMNIST)
- CIFAR-10
- CIFAR-100
- Street View House Numbers (SVHN)
- Leo Tolstoi's War and Peace

Note: While ImageNet is part of DeepOBS, it is currently not part of the automatic data downloading pipeline mechanic. Downloading the *ImageNet* data set requires an account and can take a lot of time to

download. Additonally, it requires quite a large amount of memory. The best way currently is to download and preprocess the *ImageNet* data set separately if needed and move it into the DeepOBS data folder.

The automatic data preparation script can be run using

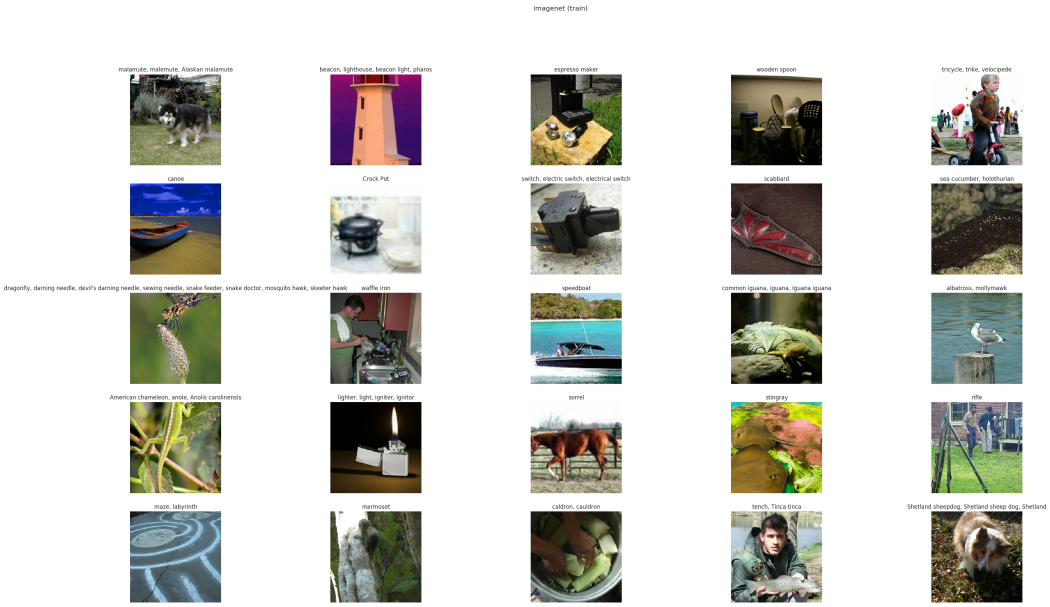
```
deepobs_prepare_data.sh
```

and is described in the API section under *Prepare Data*.

2.2 Data Loading

The DeepOBS data loading module then performs all necessary processing of the data sets to return inputs and outputs for the deep learning model (e.g. images and labels for image classification). This processing includes splitting, shuffling, batching and data augmentation. The data loading module can also be used to build new deep learning models that are not (yet) part of DeepOBS.

The outputs of the data loading module is illustrated in the figure below and is further described in the API section under *Data Sets*.



2.3 Model Loading

Together, data set and model define a loss function and thus an optimization problem. We selected problems for diversity of task as well as the difficulty of the optimization problem itself. The list of test problems of DeepOBS includes popular image classification models on data sets like `MNIST`, `CIFAR-10` or `ImageNet`, but also models for natural language processing and generative models.

Additionally, three two-dimensional problems and an ill-conditioned quadratic problem are included. These simple tests can be used as illustrative toy problems to highlight properties of an algorithm and perform sanity-checks.

Over time, we plan to expand this list when hardware and research progress renders small problems out of date, and introduces new research directions and more challenging problems.

The implementation of the models is described in the API section under *Test Problems*.

2.4 Runners

The runners of the DeepOBS package handle training and the logging of statistics measuring the optimizer's performance. For optimizers following the standard TensorFlow optimizer API it is enough to provide the runners with a list of the optimizer's hyperparameters. We provide a template for this, as well as an example of including a more sophisticated optimizer that can't be described as a subclass of the TensorFlow optimizer API.

In the API section, we described the *Standard Runner* and in the *Simple Example* we show an example of creating a run script for a new optimizer.

2.5 Baseline Results

DeepOBS also provides realistic baselines results for, currently, the three most popular optimizers in deep learning, `SGD`, `Momentum`, and `Adam`. These allow comparing a newly developed algorithm to the competition without computational overhead, and without risk of conscious or unconscious bias against the competition.

Baselines for further optimizers will be added when authors provide the optimizer's code, assuming the method perform competitively. Currently, baselines are available for all test problems in the small and large benchmark set.

2.6 Runtime Estimation

DeepOBS provides an option to quickly estimate the runtime overhead of a new optimization method compared to `SGD`. It measures the ratio of wall-clock time between the new optimizer and `SGD`.

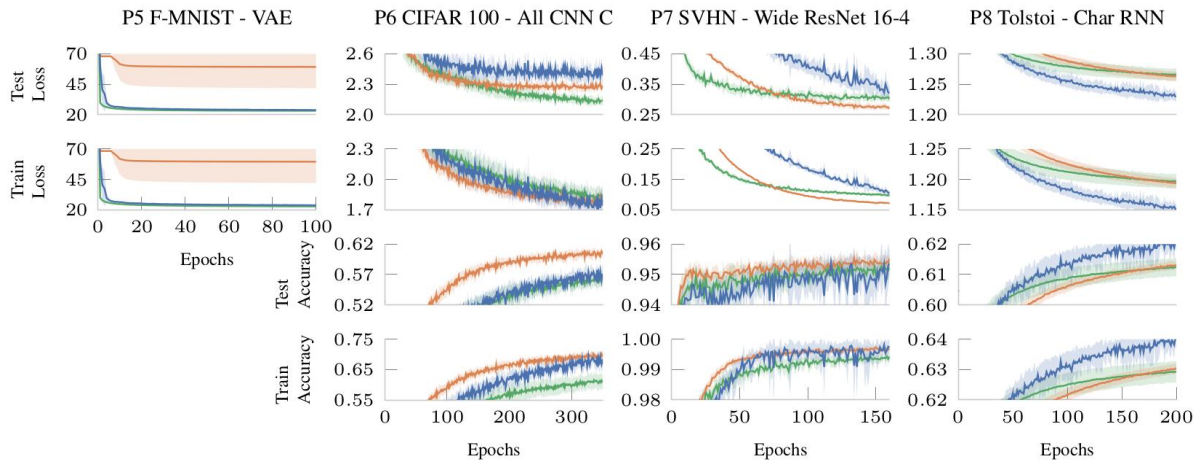
By default this ratio is measured on five runs each, for three epochs, on a fully connected network on `MNIST`. However, this can be adapted to a setting which fairly evaluates the new optimizer, as some optimizers might have a high initial cost that amortizes over many epochs.

The *Estimate Runtime* script is described in the API section.

2.7 Visualization

The DeepOBS visualization module reduces the overhead for the preparation of results, and simultaneously standardizes the presentation, making it possible to include a comparably large amount of information in limited space.

The module produces .tex files with pgfplots-code for all learning curves for the proposed optimizer as well as the most relevant baselines. This also includes a plot showing the `learning rate` sensitivity. An example plot is shown below, a more comprehensive example can be seen in section 4 of the DeepOBS paper.



The *Plot Results* script is described in the API section, as well as the lower-level functions it is relying on.

CHAPTER 3

Simple Example

This tutorial will show you an example of how DeepOBS can be used to benchmark the performance of a new optimization method for deep learning.

This simple example aims to show you some basic functions of DeepOBS, by creating a run script for a new optimizer (we will use the Momentum optimizer as an example here) and running it on a very simple test problem.

3.1 Create new Run Script

The easiest way to use DeepOBS with a new optimizer is to write a run script for it. This run script will import the optimizer and list its hyperparameters (other than the `learning rate`). For the Momentum optimizer this is simply

Listing 1: momentum_runner.py

```
1  import tensorflow as tf
2  import deepobs.tensorflow as tfobs
3
4  optimizer_class = tf.train.MomentumOptimizer
5  hyperparams = [{"name": "momentum", "type": float},
6                 {"name": "use_nesterov", "type": bool, "default": False }]
7  runner = tfobs.runners.StandardRunner(optimizer_class, hyperparams)
8
9  # The run method accepts all the relevant inputs, all arguments that are not
10 # provided will automatically be grabbed from the command line.
11 runner.run(train_log_interval=10)
```

You can download this `example run script` and use it as a template.

The DeepOBS runner (Line 7) needs access to an optimizer class with the same API as the TensorFlow optimizers and a list of additional hyperparameters for this new optimizers.

This run script is now fully command line based and is able to access all the test problems (and other options) of DeepOBS while also allowing to specify the new optimizers hyperparameters.

3.2 Run new Optimizer

Assuming the run script (from the previous section) is called `momentum_runner.py` we can use it to run the Momentum optimizer on one of the test problems on DeepOBS:

```
python momentum_runner.py quadratic_deep --bs 128 --lr 1e-2 --momentum 0.99 --num_
↪epochs 10
```

We will run it a couple times more this time with different `learning_rates`

```
python momentum_runner.py quadratic_deep --bs 128 --lr 1e-3 --momentum 0.99 --num_
↪epochs 10
python momentum_runner.py quadratic_deep --bs 128 --lr 1e-4 --momentum 0.99 --num_
↪epochs 10
python momentum_runner.py quadratic_deep --bs 128 --lr 1e-5 --momentum 0.99 --num_
↪epochs 10
```

3.3 Get best Run

We can use DeepOBS to automatically find the best of the hyperparameter settings.

In this example we will directly access the (lower-level) functions of DeepOBS. In the section *Suggested Protocol* we show you how to use the convenience scripts to do the following steps automatically.

```
import deepobs

analyzer = deepobs.analyzer.analyze_utils.Analyzer("results")
deepobs.analyzer.analyze.get_best_run(analyzer)
```

Since all of our results from the previous section are stored in the `results` folder, we pass this path to the DeepOBS *Analyzer*. Next, we can call the `get_best_run` function with this analyzer and get an output like this

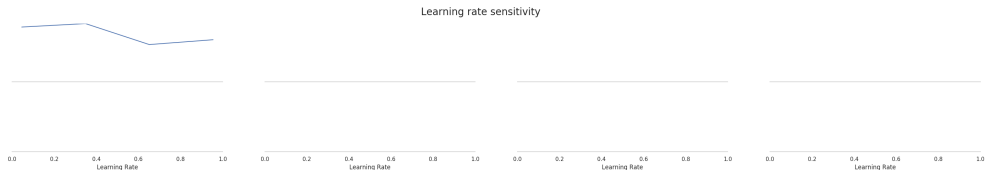
```
**********************
Analyzing quadratic_deep
**********************
Checked 4 settings for MomentumOptimizer and found the following
Best Setting (Final Value) num_epochs__10__batch_size__10__momentum__9.9e-01__use_
↪nesterov__False__lr__1.e-04 with final performance of 115.23509434291294
Best Setting (Best Value) num_epochs__10__batch_size__10__momentum__9.9e-01__use_
↪nesterov__False__lr__1.e-03 with best performance of 111.36394282749721
```

3.4 Plot Results

Similarly, we can plot the sensitivity of the (final) performance with regard to the `learning rate` by calling the appropriate DeepOBS function

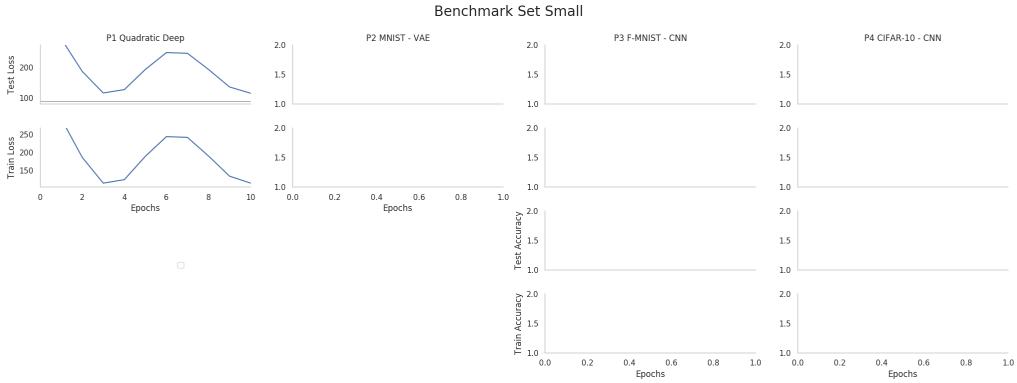
```
deepobs.analyzer.analyze.plot_lr_sensitivity(analyzer)
```

and getting a plot like this



And most importantly, a performance plot of the best performing hyperparameter setting (when looking at the final performance)

```
deepobs.analyzer.analyze.plot_performance(analyzer, mode='final')
```



CHAPTER 4

Suggested Protocol

Here we provide a suggested protocol for more rigorously benchmarking deep learning optimizer. It follows the same steps as the baseline results presented in the DeepOBS paper

4.1 Create new Run Script

In order to benchmark a new optimization method a new run script has to be written. A more detailed description can be found in the *Simple Example* and the the API section for the *Standard Runner*, but all is needed is the optimizer itself and a list of its hyperparameters. For example for the Momentum optimizer this will be.

```
import tensorflow as tf
import deepobs.tensorflow as tfobs

optimizer_class = tf.train.MomentumOptimizer
hyperparams = [{"name": "momentum", "type": float},
               {"name": "use_nesterov", "type": bool, "default": False }]
runner = tfobs.runners.StandardRunner(optimizer_class, hyperparams)

runner.run(train_log_interval=10)
```

4.2 Hyperparameter Search

Once the optimizer has been defined it is recommended to do a hyperparameter search for each test problem. For optimizers with only the `learning rate` as a free parameter a simple grid search can be done.

For the baselines, we tuned the `learning rate` for each optimizer and test problem individually, by evaluating on a logarithmic grid from `10e5` to `10e2` with `36` samples. If the same tuning method is used for a new optimizer no re-running of the baselines is needed saving valuable computational budget.

4.3 Repeated Runs with best Setting

In order to get a sense of the optimziers consistency, we suggest repeating runs with the best hyperparameter setting multiple times. This allows an assessment of the variance of the optimizer's performance.

For the baselines we determined the best learning rate looking at the final performance of each run, which can be done using

```
deepobs_plot_results results/ --get_best_run
```

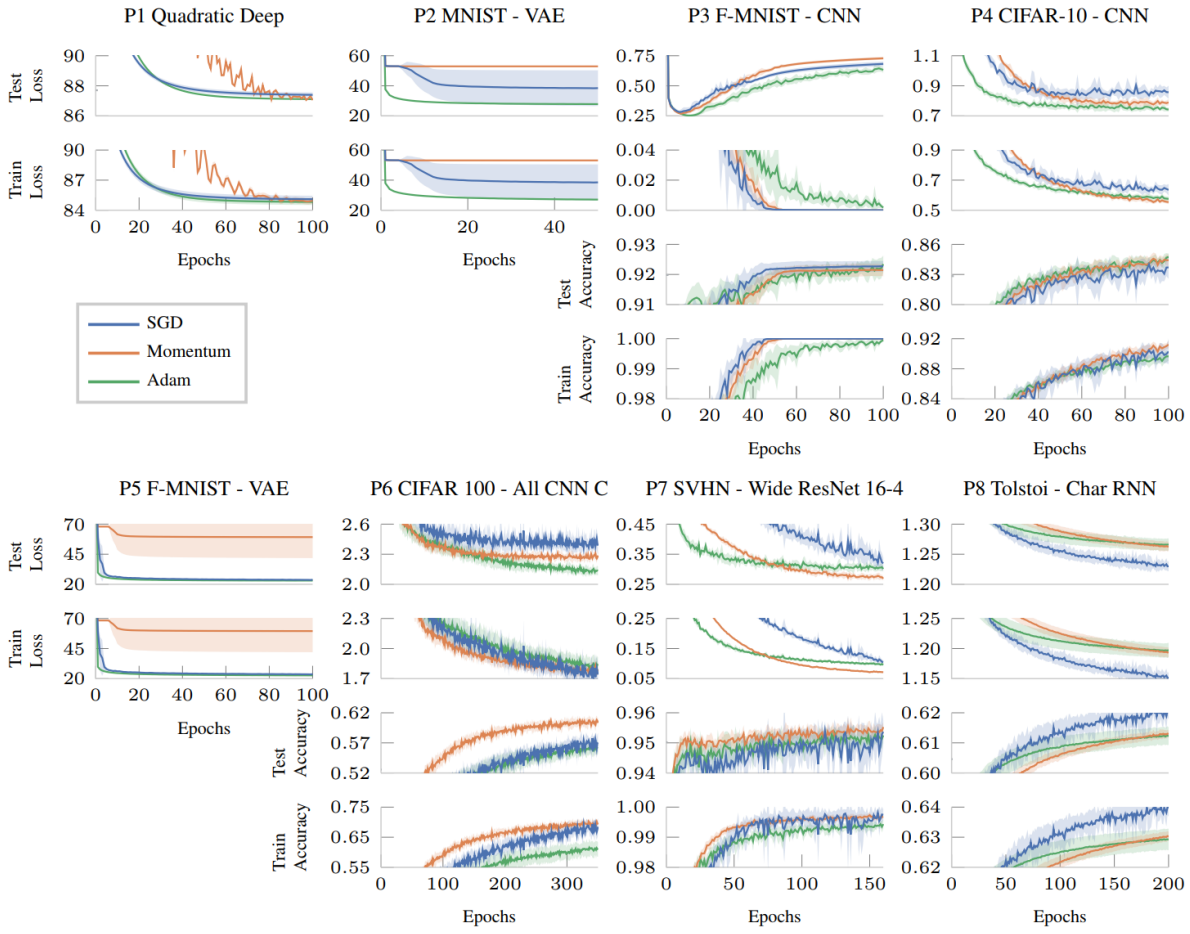
and then running the best performing setting again using ten different random seeds.

4.4 Plot Results

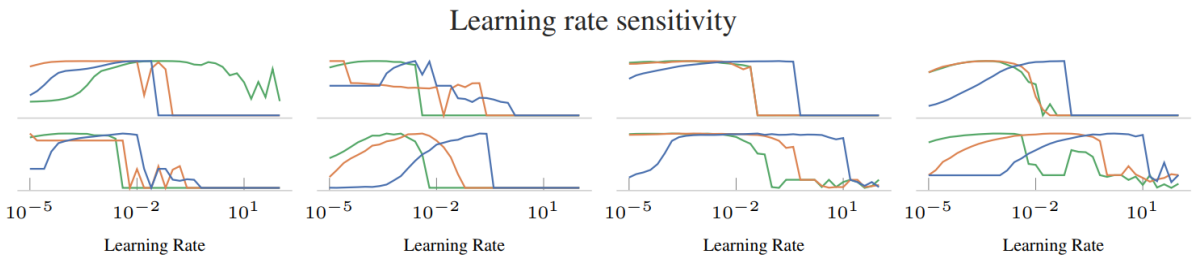
To visualize the final results it is sufficient to run

```
deepobs_plot_results results/ --full
```

This will show the performance plots for the `small` and `large` benchmark set



as well as the learning rate sensitivity plot



and the overall performance table

Test Problem		SGD	Momentum	Adam
P1 **Quadratic Deep**	Performance	87.40	87.05	87.11
	Speed	51.1	70.5	39.9
	Tuneability	α: 1.58e-02	α: 2.51e-03 μ: 0.99	α: 3.98e-02 β_1: 0.9 β_2: 0.999 ϵ: 1e-08
P2 **MNIST VAE**	Performance	38.46	52.93	27.83
	Speed	1.0	1.0	1.0
	Tuneability	α: 3.98e-03	α: 2.51e-05 μ: 0.99	α: 1.58e-04 β_1: 0.9 β_2: 0.999 ϵ: 1e-08
P3 **F-MNIST CNN**	Performance	92.27 %	92.14 %	92.34 %
	Speed	40.6	59.1	40.1
	Tuneability	α: 1.58e-01	α: 2.51e-03 μ: 0.99	α: 2.51e-04 β_1: 0.9 β_2: 0.999 ϵ: 1e-08
P4 **CIFAR-10 CNN**	Performance	83.71 %	84.41 %	84.75 %
	Speed	42.5	40.7	36.0
	Tuneability	α: 6.31e-02	α: 3.98e-04 μ: 0.99	α: 3.98e-04 β_1: 0.9 β_2: 0.999 ϵ: 1e-08

Test Problem		SGD	Momentum	Adam
P5 **F-MNIST VAE**	Performance	23.80	59.23	23.07
	Speed	1.0	1.0	1.0
	Tuneability	α: 3.98e-03	α: 1.00e-05 μ: 0.99	α: 1.58e-04 β_1: 0.9 β_2: 0.999 ϵ: 1e-08
P6 **CIFAR-100 All CNN C**	Performance	57.06 %	60.33 %	56.15 %
	Speed	128.7	72.8	152.6
	Tuneability	α: 1.58e-01	α: 3.98e-03 μ: 0.99	α: 1.00e-03 β_1: 0.9 β_2: 0.999 ϵ: 1e-08
P7 **SVHN Wide ResNet**	Performance	95.37 %	95.53 %	95.25 %
	Speed	28.3	10.8	12.1
	Tuneability	α: 2.51e-02	α: 6.31e-04 μ: 0.99	α: 1.58e-04 β_1: 0.9 β_2: 0.999 ϵ: 1e-08
P8 **TOLSTOI Char RNN**	Performance	62.07 %	61.30 %	61.23 %
	Speed	47.7	88.0	62.8
	Tuneability	α: 1.58e+00	α: 3.98e-02 μ: 0.99	α: 2.51e-03 β_1: 0.9 β_2: 0.999 ϵ: 1e-08

For all plots, `.tex` files will be generated with pgfplots-code for direct inclusion in academic publications.

CHAPTER 5

Data Sets

Currently DeepOBS includes nine different data sets. Each data set inherits from the same base class with the following signature.

class deepobs.tensorflow.datasets.dataset.**DataSet**(*batch_size*)

Base class for DeepOBS data sets.

Parameters **batch_size** (*int*) – The mini-batch size to use.

batch

A tuple of tensors, yielding batches of data from the dataset. Executing these tensors raises a `tf.errors.OutOfRangeError` after one epoch.

train_init_op

A tensorflow operation initializing the dataset for the training phase.

train_eval_init_op

A tensorflow operation initializing the testproblem for evaluating on training data.

test_init_op

A tensorflow operation initializing the testproblem for evaluating on test data.

phase

A string-value tf.Variable that is set to `train`, `train_eval` or `test`, depending on the current phase. This can be used by testproblems to adapt their behavior to this phase.

5.1 2D Data Set

class deepobs.tensorflow.datasets.two_d.**two_d**(*batch_size*, *train_size=10000*, *noise_level=1.0*)

DeepOBS data set class to create two dimensional stochastic testproblems.

This toy data set consists of a fixed number (`train_size`) of iid draws from two scalar zero-mean normal distributions with standard deviation specified by the `noise_level`.

Parameters

- **batch_size** (*int*) – The mini-batch size to use. Note that, if `batch_size` is not a divider of the dataset size (`1000` for train and test) the remainder is dropped in each epoch (after shuffling).
- **train_size** (*int*) – Size of the training data set. This will also be used as the train_eval and test set size. Defaults to `1000`.
- **noise_level** (*float*) – Standard deviation of the data points around the mean. The data points are drawn from a Gaussian distribution. Defaults to `1.0`.

batch
A tuple `(x, y)` of tensors with random x and y that can be used to create a noisy two dimensional testproblem. Executing these tensors raises a `tf.errors.OutOfRangeError` after one epoch.

train_init_op
A tensorflow operation initializing the dataset for the training phase.

train_eval_init_op
A tensorflow operation initializing the testproblem for evaluating on training data.

test_init_op
A tensorflow operation initializing the testproblem for evaluating on test data.

phase
A string-value tf.Variable that is set to "train", "train_eval" or "test", depending on the current phase. This can be used by testproblems to adapt their behavior to this phase.

5.2 Quadratic Data Set

class deepobs.tensorflow.datasets.quadratic.**quadratic**(*batch_size*, *dim=100*, *train_size=1000*, *noise_level=0.6*)

DeepOBS data set class to create an n dimensional stochastic quadratic testproblem.

This toy data set consists of a fixed number (`train_size`) of iid draws from a zero-mean normal distribution in `dim` dimensions with isotropic covariance specified by `noise_level`.

Parameters

- **batch_size** (*int*) – The mini-batch size to use. Note that, if `batch_size` is not a divider of the dataset size (`1000` for train and test) the remainder is dropped in each epoch (after shuffling).
- **dim** (*int*) – Dimensionality of the quadratic. Defaults to `100`.
- **train_size** (*int*) – Size of the dataset; will be used for train, train eval and test datasets. Defaults to `1000`.
- **noise_level** (*float*) – Standard deviation of the data points around the mean. The data points are drawn from a Gaussian distribution. Defaults to `0.6`.

batch
A tensor `X` of shape `(batch_size, dim)` yielding elements from the dataset. Executing these tensors raises a `tf.errors.OutOfRangeError` after one epoch.

train_init_op
A tensorflow operation initializing the dataset for the training phase.

train_eval_init_op
A tensorflow operation initializing the testproblem for evaluating on training data.

test_init_op
A tensorflow operation initializing the testproblem for evaluating on test data.

phase
A string-value tf.Variable that is set to `train`, `train_eval` or `test`, depending on the current phase. This can be used by testproblems to adapt their behavior to this phase.

5.3 MNIST Data Set

class deepobs.tensorflow.datasets.mnist.**mnist**(*batch_size*, *train_eval_size=10000*)
DeepOBS data set class for the MNIST data set.

Parameters

- **batch_size** (*int*) – The mini-batch size to use. Note that, if `batch_size` is not a divider of the dataset size (`60 000` for train, `10 000` for test) the remainder is dropped in each epoch (after shuffling).
- **train_eval_size** (*int*) – Size of the train eval data set. Defaults to `10 000` the size of the test set.

batch
A tuple `(x, y)` of tensors, yielding batches of MNIST images (`x` with shape `(batch_size, 28, 28, 1)`) and corresponding one-hot label vectors (`y` with shape `(batch_size, 10)`). Executing these tensors raises a `tf.errors.OutOfRangeError` after one epoch.

train_init_op
A tensorflow operation initializing the dataset for the training phase.

train_eval_init_op
A tensorflow operation initializing the testproblem for evaluating on training data.

test_init_op
A tensorflow operation initializing the testproblem for evaluating on test data.

phase
A string-value tf.Variable that is set to `train`, `train_eval` or `test`, depending on the current phase. This can be used by testproblems to adapt their behavior to this phase.

5.4 FMNIST Data Set

class deepobs.tensorflow.datasets.fmnist.**fmnist**(*batch_size*, *train_eval_size=10000*)
DeepOBS data set class for the Fashion-MNIST (FMNIST) data set.

Parameters

- **batch_size** (*int*) – The mini-batch size to use. Note that, if `batch_size` is not a divider of the dataset size (`60 000` for train, `10 000` for test) the remainder is dropped in each epoch (after shuffling).
- **train_eval_size** (*int*) – Size of the train eval data set. Defaults to `10 000` the size of the test set.

batch
A tuple `(x, y)` of tensors, yielding batches of MNIST images (`x` with shape `(batch_size, 28, 28, 1)`) and corresponding one-hot label vectors (`y` with shape `(batch_size, 10)`). Executing these tensors raises a `tf.errors.OutOfRangeError` after one epoch.

train_init_op
A tensorflow operation initializing the dataset for the training phase.

train_eval_init_op
A tensorflow operation initializing the testproblem for evaluating on training data.

test_init_op
A tensorflow operation initializing the testproblem for evaluating on test data.

phase
A string-value tf.Variable that is set to `train`, `train_eval` or `test`, depending on the current phase. This can be used by testproblems to adapt their behavior to this phase.

5.5 CIFAR-10 Data Set

class deepobs.tensorflow.datasets.cifar10.**cifar10**(*batch_size*, *data_augmentation=True*, *train_eval_size=10000*)

DeepOBS data set class for the CIFAR-10 data set.

Parameters

- **batch_size** (*int*) – The mini-batch size to use. Note that, if `batch_size` is not a divider of the dataset size (`50 000` for train, `10 000` for test) the remainder is dropped in each epoch (after shuffling).
- **data_augmentation** (*bool*) – If `True` some data augmentation operations (random crop window, horizontal flipping, lighting augmentation) are applied to the training data (but not the test data).
- **train_eval_size** (*int*) – Size of the train eval data set. Defaults to `10 000` the size of the test set.

batch
A tuple `(x, y)` of tensors, yielding batches of CIFAR-10 images (`x` with shape `(batch_size, 32, 32, 3)`) and corresponding one-hot label vectors (`y` with shape `(batch_size, 10)`). Executing these tensors raises a `tf.errors.OutOfRangeError` after one epoch.

train_init_op
A tensorflow operation initializing the dataset for the training phase.

train_eval_init_op
A tensorflow operation initializing the testproblem for evaluating on training data.

test_init_op
A tensorflow operation initializing the testproblem for evaluating on test data.

phase
A string-value tf.Variable that is set to `train`, `train_eval` or `test`, depending on the current phase. This can be used by testproblems to adapt their behavior to this phase.

5.6 CIFAR-100 Data Set

class deepobs.tensorflow.datasets.cifar100.**cifar100**(*batch_size*, *data_augmentation=True*, *train_eval_size=10000*)

DeepOBS data set class for the CIFAR-100 data set.

Parameters

- **batch_size** (*int*) – The mini-batch size to use. Note that, if `batch_size` is not a divider of the dataset size (`50 000` for train, `10 000` for test) the remainder is dropped in each epoch (after shuffling).
- **data_augmentation** (*bool*) – If `True` some data augmentation operations (random crop window, horizontal flipping, lighting augmentation) are applied to the training data (but not the test data).
- **train_eval_size** (*int*) – Size of the train eval data set. Defaults to `10 000` the size of the test set.

batch
A tuple `(x, y)` of tensors, yielding batches of CIFAR-100 images (`x` with shape `(batch_size, 32, 32, 3)`) and corresponding one-hot label vectors (`y` with shape `(batch_size, 100)`). Executing these tensors raises a `tf.errors.OutOfRangeError` after one epoch.

train_init_op
A tensorflow operation initializing the dataset for the training phase.

train_eval_init_op
A tensorflow operation initializing the testproblem for evaluating on training data.

test_init_op
A tensorflow operation initializing the testproblem for evaluating on test data.

phase
A string-value tf.Variable that is set to `train`, `train_eval` or `test`, depending on the current phase. This can be used by testproblems to adapt their behavior to this phase.

5.7 SVHN Data Set

class deepobs.tensorflow.datasets.svhn.**svhn**(*batch_size*, *data_augmentation=True*, *train_eval_size=26032*)

DeepOBS data set class for the Street View House Numbers (SVHN) data set.

Parameters

- **batch_size** (*int*) – The mini-batch size to use. Note that, if `batch_size` is not a divider of the dataset size (`73 000` for train, `26 000` for test) the remainder is dropped in each epoch (after shuffling).
- **data_augmentation** (*bool*) – If `True` some data augmentation operations (random crop window, lighting augmentation) are applied to the training data (but not the test data).
- **train_eval_size** (*int*) – Size of the train eval dataset. Defaults to `26 000` the size of the test set.

batch
A tuple `(x, y)` of tensors, yielding batches of SVHN images (`x` with shape `(batch_size, 32,`

`32, 3)`) and corresponding one-hot label vectors (`y` with shape `(batch_size, 10)`). Executing these tensors raises a `tf.errors.OutOfRangeError` after one epoch.

train_init_op
A tensorflow operation initializing the dataset for the training phase.

train_eval_init_op
A tensorflow operation initializing the testproblem for evaluating on training data.

test_init_op
A tensorflow operation initializing the testproblem for evaluating on test data.

phase
A string-value tf.Variable that is set to `train`, `train_eval` or `test`, depending on the current phase. This can be used by testproblems to adapt their behavior to this phase.

5.8 ImageNet Data Set

class deepobs.tensorflow.datasets.imagenet.**imagenet**(*batch_size*, *data_augmentation=True*, *train_eval_size=50000*)

DeepOBS data set class for the ImageNet data set.

Note: We use `1001` classes which includes an additional *background* class, as it is used for example by the inception net.

Parameters

- **batch_size** (*int*) – The mini-batch size to use. Note that, if `batch_size` is not a divider of the dataset size the remainder is dropped in each epoch (after shuffling).
- **data_augmentation** (*bool*) – If `True` some data augmentation operations (random crop window, horizontal flipping, lighting augmentation) are applied to the training data (but not the test data).
- **train_eval_size** (*int*) – Size of the train eval dataset. Defaults to `10 000`.

batch
A tuple `(x, y)` of tensors, yielding batches of ImageNet images (`x` with shape `(batch_size, 224, 224, 3)`) and corresponding one-hot label vectors (`y` with shape `(batch_size, 1001)`). Executing these tensors raises a `tf.errors.OutOfRangeError` after one epoch.

train_init_op
A tensorflow operation initializing the dataset for the training phase.

train_eval_init_op
A tensorflow operation initializing the testproblem for evaluating on training data.

test_init_op
A tensorflow operation initializing the testproblem for evaluating on test data.

phase
A string-value tf.Variable that is set to `train`, `train_eval` or `test`, depending on the current phase. This can be used by testproblems to adapt their behavior to this phase.

5.9 Tolstoi Data Set

class deepobs.tensorflow.datasets.tolstoi.**tolstoi**(*batch_size*, *seq_length=50*, *train_eval_size=653237*)

DeepOBS data set class for character prediction on *War and Peace* by Leo Tolstoi.

Parameters

- **batch_size** (*int*) – The mini-batch size to use. Note that, if `batch_size` is not a divider of the dataset size the remainder is dropped in each epoch (after shuffling).
- **seq_length** (*int*) – Sequence length to be modeled in each step. Defaults to `50`.
- **train_eval_size** (*int*) – Size of the train eval dataset. Defaults to `653 237`, the size of the test set.

batch
A tuple `(x, y)` of tensors, yielding batches of tolstoi data (`x` with shape `(batch_size, seq_length)`) and (`y` with shape `(batch_size, seq_length)` which is `x` shifted by one). Executing these tensors raises a `tf.errors.OutOfRangeError` after one epoch.

train_init_op
A tensorflow operation initializing the dataset for the training phase.

train_eval_init_op
A tensorflow operation initializing the testproblem for evaluating on training data.

test_init_op
A tensorflow operation initializing the testproblem for evaluating on test data.

phase
A string-value tf.Variable that is set to `train`, `train_eval` or `test`, depending on the current phase. This can be used by testproblems to adapt their behavior to this phase.

CHAPTER 6

Test Problems

Currently DeepOBS includes twenty-six different test problems. A test problem is given by a combination of a data set and a model and is characterized by its loss function.

Each test problem inherits from the same base class with the following signature.

class deepobs.tensorflow.testproblems.testproblem.**TestProblem**(*batch_size*, *weight_decay=None*)

Base class for DeepOBS test problems.

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – Weight decay (L2-regularization) factor to use. If not specified, the test problems revert to their respective defaults. Note: Some test problems do not use regularization and this value will be ignored in such a case.

dataset
The dataset used by the test problem (datasets.DataSet instance).

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term (might be a constant 0.0 for test problems that do not use regularization).

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Sets up the test problem.

This includes setting up the data loading pipeline for the data set and creating the tensorflow computation graph for this test problem (e.g. creating the neural network).

Note: Some of the test problems described here are based on more general implementations. For example the Wide ResNet 40-4 network on Cifar-100 is based on the general Wide ResNet architecture which is also implemented. Therefore, it is very easy to include new Wide ResNets if necessary.

6.1 2D Test Problems

Three two-dimensional test problems are included in DeepOBS. They are mainly included for illustrative purposes as these explicit loss functions can be plotted.

They are all stochastic variants of classical deterministic optimization test functions.

6.1.1 2D Beale

class deepobs.tensorflow.testproblems.two_d_beale.**two_d_beale**(*batch_size*, *weight_decay=None*)
DeepOBS test problem class for a stochastic version of thetwo-dimensional Beale function as the loss function.

Using the deterministic Beale function and adding stochastic noise of the form

$u \cdot x + v \cdot y$

where `x` and `y` are normally distributed with mean `0.0` and standard deviation `1.0` we get a loss function of the form

$((1.5 - u + u \cdot v)^2 + (2.25 - u + u \cdot v^2)^2 + (2.625 - \ u + u \cdot v^3)^2) + u \cdot x + v \cdot y.$

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – No weight decay (L2-regularization) is used in this test problem. Defaults to `None` and any input here is ignored.

dataset
The DeepOBS data set class for the two_d stochastic test problem.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term. Will always be `0.0` since no regularizer is used.

set_up()
Sets up the stochastic two-dimensional Beale test problem. Using `-4.5` and `4.5` as a starting point for the weights `u` and `v`.

6.1.2 2D Branin

class deepobs.tensorflow.testproblems.two_d_branin.**two_d_branin**(*batch_size*, *weight_decay=None*)
DeepOBS test problem class for a stochastic version of thetwo-dimensional Branin function as the loss function.

Using the deterministic Branin function and adding stochastic noise of the form

$u \cdot x + v \cdot y$

where `x` and `y` are normally distributed with mean `0.0` and standard deviation `1.0` we get a loss function of the form

$(v - 5.1/(4 \cdot \pi^2)u^2 + 5/\pi u - 6)^2 + \ 10 \cdot (1 - 1/(8 \cdot \pi)) \cdot \cos(u) + 10 + u \cdot x + v \cdot y.$

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – No weight decay (L2-regularization) is used in this test problem. Defaults to `None` and any input here is ignored.

dataset
The DeepOBS data set class for the two_d stochastic test problem.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term. Will always be `0.0` since no regularizer is used.

set_up()
Sets up the stochastic two-dimensional Branin test problem. Using `2.5` and `12.5` as a starting point for the weights `u` and `v`.

6.1.3 2D Rosenbrock

class deepobs.tensorflow.testproblems.two_d_rosenbrock.**two_d_rosenbrock**(*batch_size*, *weight_decay=None*)
DeepOBS test problem class for a stochastic version of thetwo-dimensional Rosenbrock function as the loss function.

Using the deterministic Rosenbrock function and adding stochastic noise of the form

$u \cdot x + v \cdot y$

where `x` and `y` are normally distributed with mean `0.0` and standard deviation `1.0` we get a loss function of the form

$(1-u)^2 + 100 \cdot (v-u^2)^2 + u \cdot x + v \cdot y$

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – No weight decay (L2-regularization) is used in this test problem. Defaults to `None` and any input here is ignored.

dataset
The DeepOBS data set class for the two_d stochastic test problem.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term. Will always be `0.0` since no regularizer is used.

set_up()
Sets up the stochastic two-dimensional Rosenbrock test problem. Using `-0.5` and `1.5` as a starting point for the weights `u` and `v`.

6.2 Quadratic Test Problems

DeepOBS includes a stochastic quadratic problem with an eigenspectrum similar to what has been reported for neural networks.

Other stochastic quadratic problems (of different dimensionality or with a different Hessian structure) can be created easily using the `quadratic_base` class.

class deepobs.tensorflow.testproblems._quadratic.**_quadratic_base**(*batch_size, weight_decay=None, hessian=array([[1., 0., 0., ..., 0., 0., 0.], [0., 1., 0., ..., 0., 0., 0.], [0., 0., 1., ..., 0., 0., 0.], ..., [0., 0., 0., ..., 1., 0., 0.], [0., 0., 0., ..., 0., 1., 0.], [0., 0., 0., ..., 0., 0., 1.]])*)

DeepOBS base class for a stochastic quadratic test problems creating lossfunctions of the form

$0.5 * (\theta - x)^T * Q * (\theta - x)$

with Hessian `Q` and "data" `x` coming from the quadratic data set, i.e., zero-mean normal.

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – No weight decay (L2-regularization) is used in this test problem. Defaults to `None` and any input here is ignored.
- **hessian** (*np.array*) – Hessian of the quadratic problem. Defaults to the `100` dimensional identity.

dataset
The DeepOBS data set class for the quadratic test problem.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term. Will always be `0.0` since no regularizer is used.

set_up()
Sets up the stochastic quadratic test problem. The parameter `Theta` will be initialized to (a vector of) `1.0`.

6.2.1 Quadratic Deep

class deepobs.tensorflow.testproblems.quadratic_deep.**quadratic_deep**(*batch_size*, *weight_decay=None*)

DeepOBS test problem class for a stochastic quadratic test problem `100`dimensions. The 90 % of the eigenvalues of the Hessian are drawn from theinterval $(0.0, 1.0)$ and the other 10 % are from $(30.0, 60.0)$ simulating an eigenspectrum which has been reported for Deep Learning https://arxiv.org/abs/1611.01838.

This creatis a loss functions of the form

$0.5 * (\theta - x)^T * Q * (\theta - x)$

with Hessian `Q` and "data" `x` coming from the quadratic data set, i.e., zero-mean normal.

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – No weight decay (L2-regularization) is used in this test problem. Defaults to `None` and any input here is ignored.

dataset
The DeepOBS data set class for the quadratic test problem.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term. Will always be `0.0` since no regularizer is used.

6.3 MNIST Test Problems

6.3.1 MNIST LogReg

class deepobs.tensorflow.testproblems.mnist_logreg.**mnist_logreg**(*batch_size*, *weight_decay=None*)

DeepOBS test problem class for multinomial logistic regression on MNIST.

No regularization is used and the weights and biases are initialized to `0.0`.

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – No weight decay (L2-regularization) is used in this test problem. Defaults to `None` and any input here is ignored.

dataset
The DeepOBS data set class for MNIST.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term. Will always be `0.0` since no regularizer is used.

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Sets up the logistic regression test problem on MNIST.

6.3.2 MNIST MLP

class deepobs.tensorflow.testproblems.mnist_mlp.**mnist_mlp**(*batch_size*, *weight_decay=None*)

DeepOBS test problem class for a multi-layer perceptron neural network on MNIST.

The network is build as follows:

- Four fully-connected layers with `1000`, `500`, `100` and `10` units per layer.

- The first three layers use ReLU activation, and the last one a softmax activation.
- The biases are initialized to `0.0` and the weight matrices with truncated normal (standard deviation of `3e-2`)
- The model uses a cross entropy loss.
- No regularization is used.

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – No weight decay (L2-regularization) is used in this test problem. Defaults to `None` and any input here is ignored.

dataset
The DeepOBS data set class for MNIST.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term. Will always be `0.0` since no regularizer is used.

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Set up the multi-layer perceptron test problem instance on MNIST.

6.3.3 MNIST 2c2d

class deepobs.tensorflow.testproblems.mnist_2c2d.**mnist_2c2d**(*batch_size*, *weight_decay=None*)

DeepOBS test problem class for a two convolutional and two dense layered neural network on MNIST.

The network has been adapted from the TensorFlow tutorial and consists of

- two conv layers with ReLUs, each followed by max-pooling
- one fully-connected layers with ReLUs
- 10-unit output layer with softmax
- cross-entropy loss
- No regularization

The weight matrices are initialized with truncated normal (standard deviation of `0.05`) and the biases are initialized to `0.05`.

Parameters

- **batch_size** (*int*) – Batch size to use.

- **weight_decay** (*float*) – No weight decay (L2-regularization) is used in this test problem. Defaults to `None` and any input here is ignored.

dataset
The DeepOBS data set class for MNIST.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term. Will always be `0.0` since no regularizer is used.

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Sets up the vanilla CNN test problem on MNIST.

6.3.4 MNIST VAE

class deepobs.tensorflow.testproblems.mnist_vae.**mnist_vae**(*batch_size*, *weight_decay=None*)

DeepOBS test problem class for a variational autoencoder (VAE) on MNIST.

The network has been adapted from the here and consists of an encoder:

- With three convolutional layers with each `64` filters.
- Using a leaky ReLU activation function with $\alpha = 0.3$
- Dropout layers after each convolutional layer with a rate of `0.2`.

and an decoder:

- With two dense layers with `24` and `49` units and leaky ReLU activation.
- With three deconvolutional layers with each `64` filters.
- Dropout layers after the first two deconvolutional layer with a rate of `0.2`.
- A final dense layer with `28 x 28` units and sigmoid activation.

No regularization is used.

Parameters

- **batch_size** (*type*) – Batch size to use.
- **weight_decay** (*type*) – No weight decay (L2-regularization) is used in this test problem. Defaults to `None` and any input here is ignored.

dataset
The DeepOBS data set class for MNIST.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term. Will always be `0.0` since no regularizer is used.

set_up()
Sets up the VAE test problem on MNIST.

6.4 Fashion-MNIST Test Problems

6.4.1 Fashion-MNIST LogReg

class deepobs.tensorflow.testproblems.fmnist_logreg.**fmnist_logreg**(*batch_size*, *weight_decay=None*)

DeepOBS test problem class for multinomial logistic regression on Fasion-MNIST.

No regularization is used and the weights and biases are initialized to `0.0`.

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – No weight decay (L2-regularization) is used in this test problem. Defaults to `None` and any input here is ignored.

dataset
The DeepOBS data set class for Fashion-MNIST.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term. Will always be `0.0` since no regularizer is used.

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Set up the logistic regression test problem on Fashion-MNIST.

6.4.2 Fashion-MNIST MLP

class deepobs.tensorflow.testproblems.fmnist_mlp.**fmnist_mlp**(*batch_size*, *weight_decay=None*)

DeepOBS test problem class for a multi-layer perceptron neural network on Fashion-MNIST.

The network is build as follows:

- Four fully-connected layers with `1000`, `500`, `100` and `10` units per layer.
- The first three layers use ReLU activation, and the last one a softmax activation.
- The biases are initialized to `0.0` and the weight matrices with truncated normal (standard deviation of `3e-2`)
- The model uses a cross entropy loss.
- No regularization is used.

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – No weight decay (L2-regularization) is used in this test problem. Defaults to `None` and any input here is ignored.

dataset
The DeepOBS data set class for Fashion-MNIST.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term. Will always be `0.0` since no regularizer is used.

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Set up the multi-layer perceptron test problem instance on Fashion-MNIST.

6.4.3 Fashion-MNIST 2c2d

class deepobs.tensorflow.testproblems.fmnist_2c2d.**fmnist_2c2d**(*batch_size*, *weight_decay=None*)

DeepOBS test problem class for a two convolutional and two dense layered neural network on Fashion-MNIST.

The network has been adapted from the TensorFlow tutorial and consists of

- two conv layers with ReLUs, each followed by max-pooling
- one fully-connected layers with ReLUs
- 10-unit output layer with softmax

- cross-entropy loss
- No regularization

The weight matrices are initialized with truncated normal (standard deviation of `0.05`) and the biases are initialized to `0.05`.

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – No weight decay (L2-regularization) is used in this test problem. Defaults to `None` and any input here is ignored.

dataset
The DeepOBS data set class for Fashion-MNIST.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term. Will always be `0.0` since no regularizer is used.

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Set up the vanilla CNN test problem on Fashion-MNIST.

6.4.4 Fashion-MNIST VAE

class deepobs.tensorflow.testproblems.fmnist_vae.**fmnist_vae**(*batch_size*, *weight_decay=None*)

DeepOBS test problem class for a variational autoencoder (VAE) on Fashion-MNIST.

The network has been adapted from the here and consists of an encoder:

- With three convolutional layers with each `64` filters.
- Using a leaky ReLU activation function with $\alpha = 0.3$
- Dropout layers after each convolutional layer with a rate of `0.2`.

and an decoder:

- With two dense layers with `24` and `49` units and leaky ReLU activation.
- With three deconvolutional layers with each `64` filters.
- Dropout layers after the first two deconvolutional layer with a rate of `0.2`.
- A final dense layer with `28 x 28` units and sigmoid activation.

No regularization is used.

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – No weight decay (L2-regularization) is used in this test problem. Defaults to `None` and any input here is ignored.

dataset
The DeepOBS data set class for Fashion-MNIST.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term. Will always be `0.0` since no regularizer is used.

set_up()
Set up the VAE test problem on MNIST.

6.5 CIFAR-10 Test Problems

6.5.1 CIFAR-10 3c3d

class deepobs.tensorflow.testproblems.cifar10_3c3d.**cifar10_3c3d**(*batch_size*, *weight_decay=0.002*)
DeepOBS test problem class for a three convolutional and three dense layered neural network on Cifar-10.

The network consists of

- thre conv layers with ReLUs, each followed by max-pooling
- two fully-connected layers with `512` and `256` units and ReLU activation
- 10-unit output layer with softmax
- cross-entropy loss
- L2 regularization on the weights (but not the biases) with a default factor of 0.002

The weight matrices are initialized using Xavier initialization and the biases are initialized to `0.0`.

A working training setting is `batch size = 128`, `num_epochs = 100` and SGD with learning rate of `0.01`.

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – Weight decay factor. Weight decay (L2-regularization) is used on the weights but not the biases. Defaults to `0.002`.

dataset
The DeepOBS data set class for Cifar-10.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term.

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Set up the vanilla CNN test problem on Cifar-10.

6.5.2 CIFAR-10 VGG16

class deepobs.tensorflow.testproblems.cifar10_vgg16.**cifar10_vgg16**(*batch_size*, *weight_decay=0.0005*)

DeepOBS test problem class for the VGG 16 network on Cifar-10.

The CIFAR-10 images are resized to `224` by `224` to fit the input dimension of the original VGG network, which was designed for ImageNet.

Details about the architecture can be found in the original paper. VGG 16 consists of 16 weight layers, of mostly convolutions. The model uses cross-entroy loss. A weight decay is used on the weights (but not the biases) which defaults to `5e-4`.

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – Weight decay factor. Weight decay (L2-regularization) is used on the weights but not the biases. Defaults to `5e-4`.

dataset
The DeepOBS data set class for Cifar-10.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term.

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Set up the VGG 16 test problem on Cifar-10.

6.5.3 CIFAR-10 VGG19

class deepobs.tensorflow.testproblems.cifar10_vgg19.**cifar10_vgg19**(*batch_size*, *weight_decay=0.0005*)

DeepOBS test problem class for the VGG 19 network on Cifar-10.

The CIFAR-10 images are resized to `224` by `224` to fit the input dimension of the original VGG network, which was designed for ImageNet.

Details about the architecture can be found in the original paper. VGG 19 consists of 19 weight layers, of mostly convolutions. The model uses cross-entroy loss. A weight decay is used on the weights (but not the biases) which defaults to `5e-4`.

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – Weight decay factor. Weight decay (L2-regularization) is used on the weights but not the biases. Defaults to `5e-4`.

dataset
The DeepOBS data set class for Cifar-10.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term.

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Set up the VGG 19 test problem on Cifar-10.

6.6 CIFAR-100 Test Problems

6.6.1 CIFAR-100 3c3d

class deepobs.tensorflow.testproblems.cifar100_3c3d.**cifar100_3c3d**(*batch_size*, *weight_decay=0.002*)

DeepOBS test problem class for a three convolutional and three dense layered neural network on Cifar-100.

The network consists of

- thre conv layers with ReLUs, each followed by max-pooling
- two fully-connected layers with `512` and `256` units and ReLU activation
- 100-unit output layer with softmax
- cross-entropy loss

- L2 regularization on the weights (but not the biases) with a default factor of 0.002

The weight matrices are initialized using Xavier initialization and the biases are initialized to `0.0`.

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – Weight decay factor. Weight decay (L2-regularization) is used on the weights but not the biases. Defaults to `0.002`.

dataset
The DeepOBS data set class for Cifar-100.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term.

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Set up the vanilla CNN test problem on Cifar-100.

6.6.2 CIFAR-100 VGG16

class deepobs.tensorflow.testproblems.cifar100_vgg16.**cifar100_vgg16**(*batch_size*, *weight_decay=0.0005*)

DeepOBS test problem class for the VGG 16 network on Cifar-100.

The CIFAR-100 images are resized to `224` by `224` to fit the input dimension of the original VGG network, which was designed for ImageNet.

Details about the architecture can be found in the original paper. VGG 16 consists of 16 weight layers, of mostly convolutions. The model uses cross-entroy loss. A weight decay is used on the weights (but not the biases) which defaults to `5e-4`.

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – Weight decay factor. Weight decay (L2-regularization) is used on the weights but not the biases. Defaults to `5e-4`.

dataset
The DeepOBS data set class for Cifar-100.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term.

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Set up the VGG 16 test problem on Cifar-100.

6.6.3 CIFAR-100 VGG19

class deepobs.tensorflow.testproblems.cifar100_vgg19.**cifar100_vgg19**(*batch_size*, *weight_decay=0.0005*)

DeepOBS test problem class for the VGG 19 network on Cifar-100.

The CIFAR-100 images are resized to `224` by `224` to fit the input dimension of the original VGG network, which was designed for ImageNet.

Details about the architecture can be found in the original paper. VGG 19 consists of 19 weight layers, of mostly convolutions. The model uses cross-entroy loss. A weight decay is used on the weights (but not the biases) which defaults to `5e-4`.

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – Weight decay factor. Weight decay (L2-regularization) is used on the weights but not the biases. Defaults to `5e-4`.

dataset
The DeepOBS data set class for Cifar-100.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term.

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Set up the VGG 19 test problem on Cifar-100.

6.6.4 CIFAR-100 All-CNN-C

class deepobs.tensorflow.testproblems.cifar100_allcnnc.**cifar100_allcnnc**(*batch_size*, *weight_decay=0.0005*)

DeepOBS test problem class for the All Convolutional Neural Network C on Cifar-100.

Details about the architecture can be found in the original paper.

The paper does not comment on initialization; here we use Xavier for conv filters and constant 0.1 for biases.

A weight decay is used on the weights (but not the biases) which defaults to `5e-4`.

The reference training parameters from the paper are `batch size = 256`, `num_epochs = 350` using the Momentum optimizer with $\mu = 0.9$ and an initial learning rate of $\alpha = 0.05$ and decrease by a factor of `10` after `200`, `250` and `300` epochs.

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – Weight decay factor. Weight decay (L2-regularization) is used on the weights but not the biases. Defaults to `5e-4`.

dataset
The DeepOBS data set class for Cifar-100.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term.

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Set up the All CNN C test problem on Cifar-100.

6.6.5 CIFAR-100 WideResNet 40-4

class deepobs.tensorflow.testproblems.cifar100_wrn404.**cifar100_wrn404**(*batch_size*, *weight_decay=0.0005*)

DeepOBS test problem class for the Wide Residual Network 40-4 architecture for CIFAR-100.

Details about the architecture can be found in the original paper. A weight decay is used on the weights (but not the biases) which defaults to `5e-4`.

Training settings recommenden in the original paper: `batch size = 128`, `num_epochs = 200` using the Momentum optimizer with $\mu = 0.9$ and an initial learning rate of `0.1` with a decrease by `0.2` after `60`, `120` and `160` epochs.

Parameters

- **batch_size** (*int*) – Batch size to use.

- **weight_decay** (*float*) – Weight decay factor. Weight decay (L2-regularization) is used on the weights but not the biases. Defaults to `5e-4`.

dataset
The DeepOBS data set class for Cifar-100.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term.

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Set up the Wide ResNet 40-4 test problem on Cifar-100.

6.7 SVHN Test Problems

6.7.1 SVHN 3c3d

class deepobs.tensorflow.testproblems.svhn_3c3d.**svhn_3c3d**(*batch_size*, *weight_decay=0.002*)
DeepOBS test problem class for a three convolutional and three dense layered neural network on SVHN.

The network consists of

- thre conv layers with ReLUs, each followed by max-pooling
- two fully-connected layers with `512` and `256` units and ReLU activation
- 10-unit output layer with softmax
- cross-entropy loss
- L2 regularization on the weights (but not the biases) with a default factor of 0.002

The weight matrices are initialized using Xavier initialization and the biases are initialized to `0.0`.

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – Weight decay factor. Weight decay (L2-regularization) is used on the weights but not the biases. Defaults to `0.002`.

dataset
The DeepOBS data set class for SVHN.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term.

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Set up the vanilla CNN test problem on SVHN.

6.7.2 SVHN WideResNet 16-4

class deepobs.tensorflow.testproblems.svhn_wrn164.**svhn_wrn164**(*batch_size*, *weight_decay=0.0005*)
DeepOBS test problem class for the Wide Residual Network 16-4 architecture for SVHN.

Details about the architecture can be found in the original paper. A weight decay is used on the weights (but not the biases) which defaults to `5e-4`.

Training settings recommenden in the original paper: `batch size = 128`, `num_epochs = 160` using the Momentum optimizer with $\mu = 0.9$ and an initial learning rate of `0.01` with a decrease by `0.1` after `80` and `120` epochs.

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – Weight decay factor. Weight decay (L2-regularization) is used on the weights but not the biases. Defaults to `5e-4`.

dataset
The DeepOBS data set class for SVHN.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term.

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Set up the Wide ResNet 16-4 test problem on SVHN.

6.8 ImageNet Test Problems

6.8.1 ImageNet VGG16

class deepobs.tensorflow.testproblems.imagenet_vgg16.**imagenet_vgg16**(*batch_size*, *weight_decay=0.0005*)

DeepOBS test problem class for the VGG 16 network on ImageNet.

Details about the architecture can be found in the original paper. VGG 16 consists of 16 weight layers, of mostly convolutions. The model uses cross-entroy loss. A weight decay is used on the weights (but not the biases) which defaults to `5e-4`.

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – Weight decay factor. Weight decay (L2-regularization) is used on the weights but not the biases. Defaults to `5e-4`.

dataset
The DeepOBS data set class for ImageNet.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term.

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Set up the VGG 16 test problem on ImageNet.

6.8.2 ImageNet VGG19

class deepobs.tensorflow.testproblems.imagenet_vgg19.**imagenet_vgg19**(*batch_size*, *weight_decay=0.0005*)

DeepOBS test problem class for the VGG 19 network on ImageNet.

Details about the architecture can be found in the original paper. VGG 19 consists of 19 weight layers, of mostly convolutions. The model uses cross-entroy loss. A weight decay is used on the weights (but not the biases) which defaults to `5e-4`.

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – Weight decay factor. Weight decay (L2-regularization) is used on the weights but not the biases. Defaults to `5e-4`.

dataset
The DeepOBS data set class for ImageNet.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term.

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Set up the VGG 19 test problem on ImageNet.

6.8.3 ImageNet Inception v3

class deepobs.tensorflow.testproblems.imagenet_inception_v3.**imagenet_inception_v3**(*batch_size*, *weight_deca*

DeepOBS test problem class for the Inception version 3 architecture on ImageNet.

Details about the architecture can be found in the original paper.

There are many changes from the paper to the official Tensorflow implementation as well as the model.txt that can be found in the sources of the original paper. We chose to implement the version from Tensorflow (with possibly some minor changes)

In the original paper they trained the network using:

- `100` Epochs.
- Batch size `32`.
- RMSProp with a decay of `0.9` and $\epsilon = 1.0$.
- Initial learning rate `0.045`.
- Learning rate decay every two epochs with exponential rate of `0.94`.
- Gradient clipping with threshold 2.0

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – Weight decay factor. Weight decay (L2-regularization) is used on the weights but not the biases. Defaults to `5e-4`.

dataset
The DeepOBS data set class for ImageNet.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term.

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Set up the Inception v3 test problem on ImageNet.

6.9 Tolstoi Test Problems

6.9.1 Tolstoi Char RNN

class deepobs.tensorflow.testproblems.tolstoi_char_rnn.**tolstoi_char_rnn**(*batch_size*, *weight_decay=None*)
DeepOBS test problem class for a two-layer LSTM for character-level language modelling (Char RNN) on Tolstoi's War and Peace.

Some network characteristics:

- 128 hidden units per LSTM cell
- sequence length 50
- cell state is automatically stored in variables between subsequent steps
- when the phase placeholder swithches its value from one step to the next, the cell state is set to its zero value (meaning that we set to zero state after each round of evaluation, it is therefore important to set the evaluation interval such that we evaluate after a full epoch.)

Working training parameters are:

- batch size 50
- 200 epochs
- SGD with a learning rate of ≈ 0.1 works

Parameters

- **batch_size** (*int*) – Batch size to use.
- **weight_decay** (*float*) – No weight decay (L2-regularization) is used in this test problem. Defaults to None and any input here is ignored.

dataset
The DeepOBS data set class for Tolstoi.

train_init_op
A tensorflow operation initializing the test problem for the training phase.

train_eval_init_op
A tensorflow operation initializing the test problem for evaluating on training data.

test_init_op
A tensorflow operation initializing the test problem for evaluating on test data.

losses
A tf.Tensor of shape (batch_size,) containing the per-example loss values.

regularizer
A scalar tf.Tensor containing a regularization term.

accuracy
A scalar tf.Tensor containing the mini-batch mean accuracy.

set_up()
Set up the Char RNN test problem instance on Tolstoi.

CHAPTER 7

Runner

Runner take care of the actual training process in DeepOBS. They also log performance statistics such as the loss and accuracy on the test and training data set.

The output of those runners is saved into `JSON` files and optionally also TensorFlow output files that can be plotted in real-time using *Tensorboard*.

7.1 Standard Runner

class deepobs.tensorflow.runners.standard_runner.**StandardRunner**(*optimizer_class*, *hyperparams*)

Provides functionality to run optimizers on DeepOBS testproblems including the logging of important performance metrics.

Parameters

- **optimizer_class** – Optimizer class, which should inherit from tf.train.Optimizer and/or obey the same interface for `.minimize()`.
- **hyperparams** – A list describing the optimizer's hyperparameters other than learning rate. Each entry of the list is a dictionary describing one of the hyperparameters. This dictionary is expected to have the following two fields:
  - hyperparams["name"] must contain the name of the parameter (i.e., the exact name of the corresponding keyword argument to the optimizer class' init function.
  - hyperparams["type"] specifies the type of the parameter (e.g., `int`, `float`, `bool`).

  Optionally, the dictionary can have a third field indexed by the key "default", which specifies a default value for the hyperparameter.

Example

```
>>> optimizer_class = tf.train.MomentumOptimizer
>>> hyperparams = [
        {"name": "momentum", "type": float},
        {"name": "use_nesterov", "type": bool, "default": False}]
>>> runner = StandardRunner(optimizer_class, hyperparms)
```

run(*testproblem=None*, *weight_decay=None*, *batch_size=None*, *num_epochs=None*, *learning_rate=None*, *lr_sched_epochs=None*, *lr_sched_factors=None*, *random_seed=None*, *data_dir=None*, *output_dir=None*, *train_log_interval=None*, *print_train_iter=None*, *tf_logging=None*, *no_logs=None*, ***optimizer_hyperparams*)

Runs a given optimizer on a DeepOBS testproblem.

This method receives all relevant options to run the optimizer on a DeepOBS testproblem, including the hyperparameters of the optimizers, which can be passed as keyword arguments (based on the names provided via `hyperparams` in the init function).

Options which are *not* passed here will automatically be added as command line arguments. (Some of those will be required, others will have defaults; run the script with the `--help` flag to see a description of the command line interface.)

Training statistics (train/test loss/accuracy) are collected and will be saved to a `JSON` output file, together with metadata. The training statistics can optionally also be saved in TensorFlow output files and read during training using *Tensorboard*.

Parameters

- **testproblem** (*str*) – Name of a DeepOBS test problem.
- **weight_decay** (*float*) – The weight decay factor to use.
- **batch_size** (*int*) – The mini-batch size to use.
- **num_epochs** (*int*) – The number of epochs to train.
- **learning_rate** (*float*) – The learning rate to use. This will function as the base learning rate when implementing a schedule using `lr_sched_epochs` and `lr_sched_factors` (see below).
- **lr_sched_epochs** (*list*) – A list of epoch numbers (positive integers) that mark learning rate changes. The base learning rate is passed via `learning_rate` and the factors by which to change are passed via `lr_sched_factors`. Example: `learning_rate=0.3, lr_sched_epochs=[50, 100], lr_sched_factors=[0.1 0.01]` will start with a learning rate of `0.3`, then decrease to `0.1*0.3=0.03` after training for `50` epochs, and decrease to `0.01*0.3=0.003` after training for `100` epochs.
- **lr_sched_factors** (*list*) – A list of factors (floats) by which to change the learning rate. The base learning rate has to be passed via `learing_rate` and the epochs at which to change the learning rate have to be passed via `lr_sched_factors`. Example: `learning_rate=0.3, lr_sched_epochs=[50, 100], lr_sched_factors=[0.1 0.01]` will start with a learning rate of `0.3`, then decrease to `0.1*0.3=0.03` after training for `50` epochs, and decrease to `0.01*0.3=0.003` after training for `100` epochs.
- **random_seed** (*int*) – Random seed to use. If unspecified, it defaults to `42`.
- **data_dir** (*str*) – Path to the DeepOBS data directory. If unspecified, DeepOBS uses its default */data_deepobs*.

- **output_dir** (*str*) – Path to the output directory. Within this directory, subfolders for the testproblem and the optimizer are automatically created. If unspecified, defaults to '/results'.
- **train_log_interval** (*int*) – Interval of steps at which to log training loss. If unspecified it defaults to `10`.
- **print_train_iter** (*bool*) – If `True`, training loss is printed to screen. If unspecified it defaults to `False`.
- **tf_logging** (*bool*) – If `True` log all statistics with tensorflow summaries, which can be viewed in real time with tensorboard. If unspecified it defaults to `False`.
- **no_logs** (*bool*) – If `True` no `JSON` files are created. If unspecified it defaults to `False`.
- **optimizer_hyperparams** (*dict*) – Keyword arguments for the hyperparameters of the optimizer. These are the ones specified in the `hyperparams` dictionary passed to the `__init__`.

CHAPTER 8

Analyzer

DeepOBS uses the analyzer class to get meaning full outputs from the results created by the runners. This includes:

- Getting the best settings (e.g. best `learning rate`) for an optimizer on a specific test problem.
- Plotting the `learning_rate` sensitivity for multiple optimizers on a test problem.
- Plotting all performance metrics of the whole benchmark set.
- Returning the overall performance table for multiple optimizers.

The analyzer can return those outputs as matplotlib plots or `.tex` files for direct inclusion in academic publications.

DeepOBS also includes a convenience script using this analyzer class for these most used cases, see *Plot Results*

8.1 Analyzer

class deepobs.analyzer.analyze_utils.**Analyzer**(*path*)

DeepOBS analyzer class to generate result plots or get other summaries.

Parameters **path** (*str*) – Path to the results folder. This folder should contain one or multiple testproblem folders.

testproblems

Dictionary of test problems where the key is the name of a test problem (e.g. `cifar10_3c3d`) and the value is an instance of the TestProblemAnalyzer class (see below).

8.2 Test Problem Analyzer

class deepobs.analyzer.analyze_utils.**TestProblemAnalyzer**(*path*, *tp*)

DeepOBS analyzer class for a specific test problem.

This class will store all relevant information regarding a test problem, such as the convergence performance of this problem.

Parameters

- **path** (*str*) – Path to the parent folder of the test problem (i.e. the results folder).
- **tp** (*str*) – Name of the test problem (same as the folder name).

name
Name of the test problem in DeepOBS format (e.g. `cifar10_3c3d`).

conv_perf
Convergence performance for this test problem.

metric
Metric to use for this test problem. If available this will be `test_accuracies`, otherwise `test_losses`.

optimizer
Dictionary of optimizers for this test problem where the key is the name of the optimizer (e.g. `GradientDescentOptimizer`) and the value is an instance of the OptimizerAnalyzer class (see below).

8.3 Optimizer Analyzer

class deepobs.analyzer.analyze_utils.**OptimizerAnalyzer**(*path*, *opt*, *metric*, *testproblem*, *conv_perf*)

DeepOBS analyzer class for an optimizer (and a specific test problem).

This class will give access to all relevant information regarding this optimizer such as the best performing hyperparameter setting or the number of settings.

Parameters

- **path** (*str*) – Path to the parent folder of the optimizer folder (i.e. the test problem folder).
- **opt** (*str*) – Name of the optimizer (folder).
- **metric** (*str*) – Metric to use for this test problem. If available this will be `test_accuracies`, otherwise `test_losses`.
- **testproblem** (*str*) – Name of the test problem this optimizer (folder) belongs to.
- **conv_perf** (*float*) – Convergence performance of the test problem this optimizer (folder) belongs to.

name
Name of the optimizer (folder).

metric
Metric to use for this test problem. If available this will be `test_accuracies`, otherwise `test_losses`.

testproblem
Name of the test problem this optimizer (folder) belongs to.

conv_perf
Convergence performance for this test problem.

settings
Dictionary of hyperparameter settings for this optimizer (on this test problem) where the key is the name of the setting (folder) and the value is an instance of the SettingAnalyzer class (see below).

num_settings
Total number of settings for this optimizer (and test problem)

get_best_setting_best()
Returns the setting for this optimizer that has the best overall performance using the metric (`test_losses` or `test_accuracies`) defined for this test problem. In contrast to `get_best_setting_final` in not only looks at the final performance per setting, but the best performance per setting.

Returns Instance of the SettingAnalyzer class with the best overall performance

Return type *SettingAnalyzer*

get_best_setting_final()
Returns the setting for this optimizer that has the best final performance using the metric (`test_losses` or `test_accuracies`) defined for this test problem.

Returns Instance of the SettingAnalyzer class with the best final performance

Return type *SettingAnalyzer*

get_bm_table(*perf_table*, *mode='most'*)
Generates the overall performance table for this optimizer.

This includes metrics for the performance, speed and tuneability of this optimizer (on this test problem).

Parameters

- **perf_table** (*dict*) – A dictionary with three keys: `Performance`, `Speed` and `Tuneability`.
- **mode** (*str*) – Whether to use the setting with the best final (`final`) performance, the best overall (`best`) performance or the setting with the most runs (`most`). Defaults to `most`.

Returns Dictionary with holding the performance, speed and tuneability measure for this optimizer.

Return type dict

get_setting_most_runs()
Returns the setting with the most repeated runs (with the same setting, but probably different seeds).

Returns Instance of the SettingAnalyzer class with the most repeated runs.

Return type *SettingAnalyzer*

plot_lr_sensitivity(*ax*, *mode='final'*)
Generates the `learning rate` sensitivity plot for this optimizer. This plots the relative performance (relative to the best setting for this optimizer) against the `learning rate` used in this setting.

This assumes that all settings or otherwise equal and only different in the `learning rate`.

Parameters

- **ax** (*matplotlib.axes*) – Handle to a matplotlib axis to plot the `learning rate` sensitivity onto.
- **mode** (*str*) – Whether to use the final (`final`) performance or the best (`best`) when evaluating each setting. Defaults to `final`.

plot_performance(*ax*, *mode='most'*)
Generates a performance plot for this optimzer using one hyperparameter setting.

Can either use the setting with the best final performance, the best overall performance or the setting with the most runs.

This function will plot all four possible performance metrics (`test_losses`, `train_losses`, `test_accuracies` and `train_accuracies`).

Parameters

- **ax** (*list*) – List of four matplotlib axis to plot the performancs metrics onto.
- **mode** (*str*) – Whether to use the setting with the best final (`final`) performance, the best overall (`best`) performance or the setting with the most runs (`most`) when plotting. Defaults to `most`.

8.4 Setting Analyzer

class deepobs.analyzer.analyze_utils.**SettingAnalyzer**(*path*, *sett*, *metric*, *testproblem*, *conv_perf*)

DeepOBS analyzer class for a setting (a hyperparameter setting).

Parameters

- **path** (*str*) – Path to the parent folder of the setting folder (i.e. the optimizer folder).
- **sett** (*str*) – Name of the setting (folder).
- **metric** (*str*) – Metric to use for this test problem. If available this will be `test_accuracies`, otherwise `test_losses`.
- **testproblem** (*str*) – Name of the test problem this setting (folder) belongs to.
- **conv_perf** (*float*) – Convergence performance of the test problem this setting (folder) belongs to.

name
Name of the setting (folder).

Type str

metric
Metric to use for this test problem. If available this will be `test_accuracies`, otherwise `test_losses`.

Type str

testproblem
Name of the test problem this setting (folder) belongs to.

Type str

conv_perf
Convergence performance for this test problem.

Type float

aggregate
Instance of the AggregateRun class for all runs with this setting.

Type *AggregateRun*

8.5 Aggregate Run

class deepobs.analyzer.analyze_utils.**AggregateRun**(*path*, *runs*, *name*, *metric*, *testproblem*, *conv_perf*)

DeepOBS class for a group of runs witht the same settings (but possibly different seeds).

Parameters

- **path** (*str*) – Path to the parent folder of the aggregate run folder (i.e. the settings folder).
- **runs** (*list*) – List of run names all with the same setting.
- **name** (*str*) – Name of the aggregate run (folder).
- **metric** (*str*) – Metric to use for this test problem. If available this will be `test_accuracies`, otherwise `test_losses`.
- **testproblem** (*str*) – Name of the test problem this aggregate run (folder) belongs to.
- **conv_perf** (*float*) – Convergence performance of the test problem this aggregate run (folder) belongs to.

name
Name of the aggregate run (folder).

testproblem
Name of the test problem this aggregate run (folder) belongs to.

conv_perf
Convergence performance for this test problem.

runs
List of run names all with the same setting.

num_runs
Number of runs (with the same setting).

metric
Metric to use for this test problem. If available this will be `test_accuracies`, otherwise `test_losses`.

output
Dictionary including all aggregate information of the runs with this setting. All performance metrics have a mean and a standard deviation (can be zero if there is only one run with this setting).

final_value
Final (mean) value of the test problem's metric

best_value
Best (mean) value of the test problem's metric

CHAPTER 9

Scripts

DeepOBS includes a few convenience scripts that can be run directly from the command line

- **Prepare Data**: Takes care of downloading and preprocessing all data sets for DeepOBS.
- **Estimate Runtime**: Allows to estimate the runtime overhead of a new optimizer compared to SGD.
- **Plot Results**: Quickly plots the suggested outputs of a optimizer benchmark.

9.1 Prepare Data

A convenience script to download all data sets for DeepOBS and preprocess them so they are ready to be used with DeepOBS.

Note: Currently there is no data downloading and preprocessing mechanic implemented for *ImageNet*. Downloading the *ImageNet* data set requires an account and can take a lot of time to download. Additionally, it requires quite a large amount of memory. The best way currently is to download and preprocess the *ImageNet* data set separately if needed and move it into the DeepOBS data folder.

The file will create a set of folders of the following structure:

```
data_deepobs
├── cifar10
│   ├── data_batch_1.bin
│   ├── data_batch_2.bin
│   └── ...
├── cifar100
│   ├── train.bin
│   ├── test.bin
│   └── ...
```

```
├── fmnist
│   ├── t10k-images-idx3-ubyte.gz
│   ├── t10k-labels-idx1-ubyte.gz
│   └── ...
├── mnist
│   ├── t10k-images-idx3-ubyte.gz
│   ├── t10k-labels-idx1-ubyte.gz
│   └── ...
├── svhn
│   ├── data_batch_0.bin
│   ├── data_batch_1.bin
│   └── ...
├── tolstoi
│   ├── train.npy
│   ├── test.npy
│   └── ...
├── imagenet
│   ├── train-00000-of-01024
│   ├── ...
│   ├── validation-00000-of-00128
│   └── ...
```

DeepOBS expects a structure like this, so if you already have (most of the) the data sets already, you still need to bring it into this order.

Usage:

```
usage: deepobs_prepare_data.sh [--data_dir=DIR] [--skip SKIP] [--only ONLY]
```

9.1.1 Named Arguments

-d – data_dir	Path where the data sets should be saved. Defaults to the current folder.
-s – skip	Defines which data sets should be skipped. Argument needs to be one of the following `mnist`, `fmnist`, `cifar10`, `cifar100`, `svhn`, `imagenet`, `tolstoi`. You can use the `--skip` argument multiple times.
-o – only	Specify if only a single data set should be downloaded. Argument needs to be one of the following `mnist`, `fmnist`, `cifar10`, `cifar100`, `svhn`, `imagenet`, `tolstoi`. This overwrites the `--skip` argument and should can only be used once.

9.2 Estimate Runtime

A convenience script to estimate the run time overhead of a new optimization method compared to SGD.

By default this script runs SGD as well as the new optimizer `5` times for `3` epochs on the multi-layer perceptron on MNIST while measuring the time. It will output the mean run time overhead of the new optimizer for these runs.

Optionally the setup can be changed, by varying the test problem, the number of epochs, the number of runs, etc. if this allows for a fairer evaluation.

Usage:

Run a new run script and compare its runtime to SGD.

```
usage: deepobs_estimate_runtime.py [-h] [--test_problem TEST_PROBLEM]
                                   [--data_dir DATA_DIR] [--bs BS] [--lr LR]
                                   [-N NUM_EPOCHS] [--num_runs NUM_RUNS]
                                   [--saveto SAVETO]
                                   [--optimizer_args OPTIMIZER_ARGS]
                                   run_script
```

9.2.1 Positional Arguments

run_script Path to the new run_script.

9.2.2 Named Arguments

--test_problem Name of the test problem to run both scripts.

Default: "mnist_mlp"

--data_dir Path to the base data dir. If not set, deepobs uses its default.

Default: "data_deepobs"

--bs, --batch_size The batch size (positive integer).

Default: 128

--lr, --learning_rate The learning rate of both SGD and the new optimizer, defaults to 1e-5.

Default: 1e-05

-N, --num_epochs Total number of training epochs per run.

Default: 3

--num_runs Total number of runs for each optimizer.

Default: 5

--saveto Folder for saving a txt files with a summary.

--optimizer_args Additional arguments for the new optimizer

9.3 Plot Results

A convenience script to extract useful information out of the results create by the runners.

This script can return one or all of the below information:

- Get best run: Returns the best hyperparameter setting for each optimizer in each test problem.
- Plot learning rate sensitivity: Creates a plot for each test problem showing the relative performance of each optimizer against the learning rate to get a sense of how difficult the tuning process was.

- Plot performance: Creates a plot for the `small` and `large` benchmark set, plotting (if available) all four performance metric (`losses` and `accuracies` for both the test and the train data set) for each optimizer.
- Plot table: Creates the overall performance table for the `small` and `large` benchmark set including metrics for the performance, speed and tuneability of each optimizer on each test problem.

By default this script also plots the baseline results for *SGD*, *Momentum* and *Adam*, but this can be turned off.

Usage:

Plotting tool for DeepOBS.

```
usage: deepobs_plot_results.py [-h] [--get_best_run] [--plot_lr_sensitivity]
                               [--plot_performance] [--plot_table] [--full]
                               [--ignore_baselines]
                               path
```

9.3.1 Positional Arguments

path Path to the results folder

9.3.2 Named Arguments

--get_best_run Return best hyperparameter setting per optimizer and testproblem.

Default: False

--plot_lr_sensitivity Plot 'sensitivity' plot for the learning rates.

Default: False

--plot_performance Plot performance plot compared to the baselines.

Default: False

--plot_table Plot overall performance table including speed and hyperparameters.

Default: False

--full Run a full analysis and plot all figures.

Default: False

--ignore_baselines Ignore baselines and just plot from results folder.

Default: False

CHAPTER 10

Indices and tables

- genindex
- search

Index

Symbols

A

B

C

D

F

G

I

L

M

N

O

P

Q

R

S

T