predicode

Release 0.0.0.9000

Samuel Lippl

Aug 24, 2019

CONTENTS:

CHAPTER

ONE

INSTALLATION

Install 'predicode' using pip and the command

```
pip install predicode
```

If you would like the newest development version, use pip to install the package from the GitHub repository using the command

```
pip install git+https://github.com/sflippl/predicode
```

CHAPTER

TWO

DATASETS

[1]:
```
try:
    import predicode as pc
except:
    !pip install git+https://github.com/sflippl/predicode
    import predicode as pc
```

```
Using TensorFlow backend.
```

2.1 Artificial Datasets

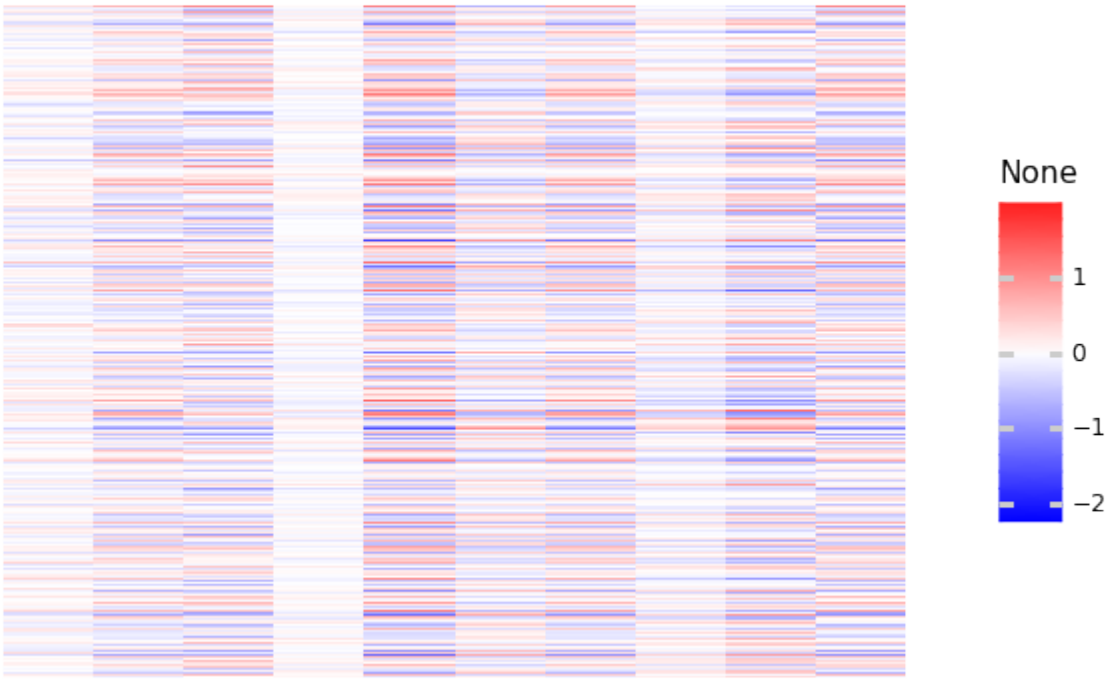
Artificial datasets provide a simple example for how the algorithm works and an opportunity to study its analytical solutions.

2.1.1 Decaying Multinormal Distribution

The closed-form solution of a linear predictive coding model is given by a principal components analysis. A multinormal distribution allows for an easy model for such a solution. The class 'DecayingMultiNormal' models a high-dimensional input with decaying importance. Namely, the variance of the different principal components is specified using the decay constant 'alpha'. Dimensionality of the input data is specified using 'dimensions' and sample size is specified by 'samples'.

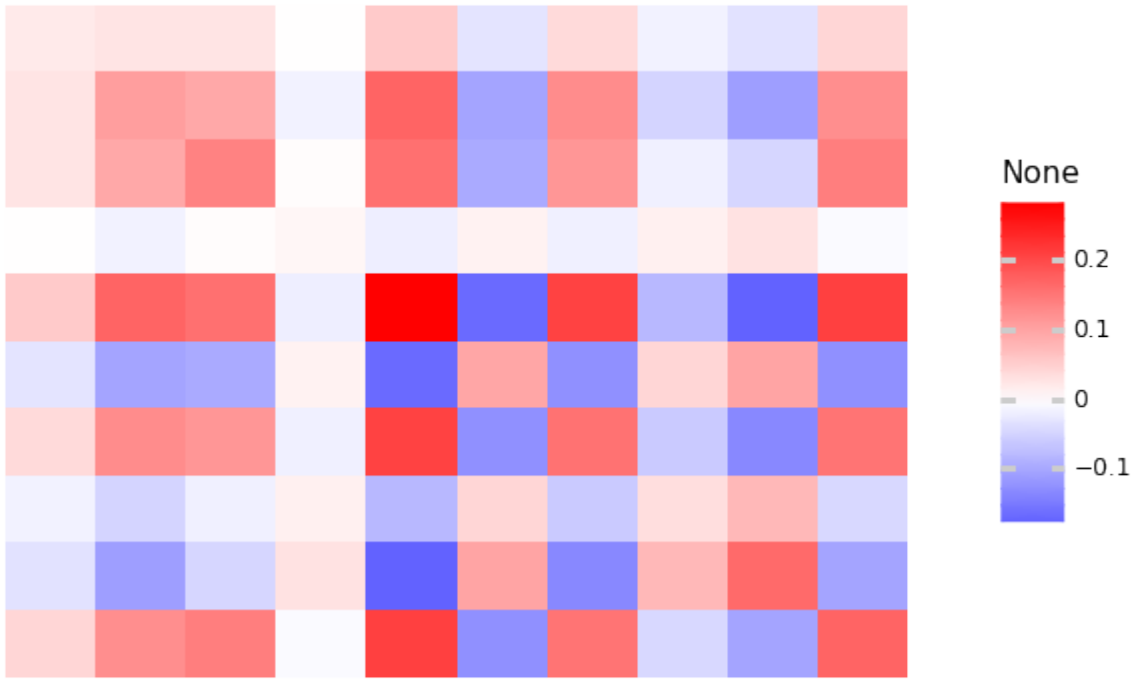
[1]:
```
art_data = pc.DecayingMultiNormal(dimensions = 10,
                                  samples = 10000,
                                  alpha = 1)
import lazytools
import numpy as np
```

[2]:
```
lazytools.matrix_heatmap(art_data.data, pole = 0)
```



[2]:
```
<ggplot: (8735735730103)>
```

[3]:
```
lazytools.matrix_heatmap(np.cov(art_data.data.T), pole = 0)
```



[3]:
```
<ggplot: (8735682449065)>
```

2.2 Image Datasets

Image datasets are predominantly included as examples for the predictive coding algorithms under the 'datasets' module. Whereas their main purpose is being incorporated by the respective algorithms, 'predicode' allows for some functionality in exploring the datasets on their own. In particular, a number of images may be visualized using the pictures method (see Cifar-10 below).

2.3 Cifar-10

Cifar-10 serves as a simple example dataset for basic predictive coding algorithms demonstrating static extraclassical effects.

For now, only the training dataset can be read in using the class `Cifar10`

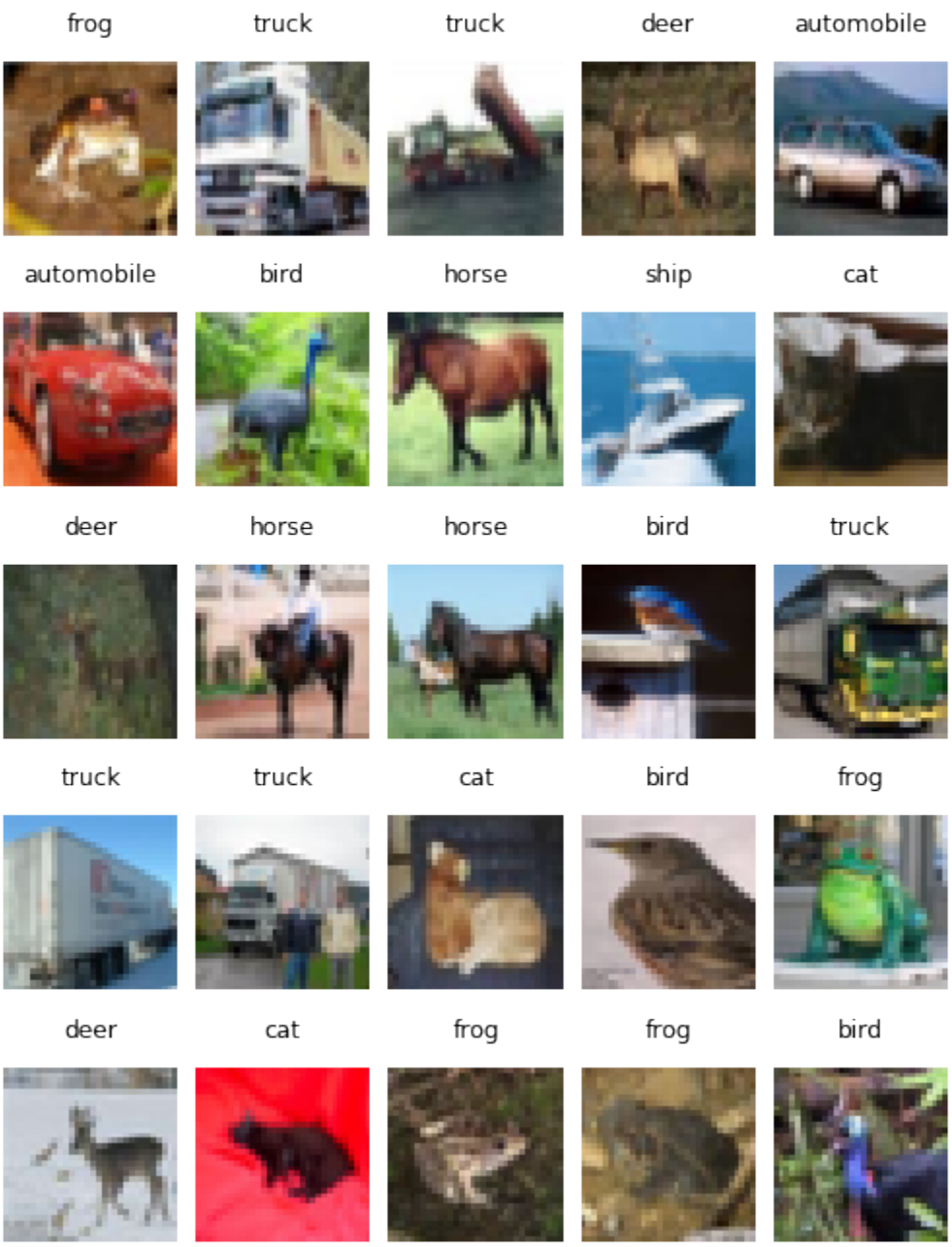
[1]:
```
import predicode as pc
cifar = pc.Cifar10()
```

This dataset may be explored by looking at the pictures along with their labels. This is possible in black-white and color.

[2]:
```
cifar.pictures(subset = range(25), mode = 'bw')
```

[2]:
```
<ggplot: (-9223363293114942036)>
```

[3]:
```
cifar.pictures(subset = range(25), mode = 'color')
```



[3]:
```
<ggplot: (8743719730980)>
```

This builds upon the underlying data frame that contains the RGB values for the color and the black-white pictures:

[4]:
```
cifar.rgb_dataframe(subset = range(1)).head()
```

[4]:

	image_id	x	y	r	g	b	bw	rgb	rgb_bw
0	0	0	0	59	62	63	61.333333	#3b3e3f	#3d3d3d
1	0	1	0	43	46	45	44.666667	#2b2e2d	#2c2c2c
2	0	2	0	50	48	43	47.000000	#32302b	#2f2f2f
3	0	3	0	68	54	42	54.666667	#44362a	#363636
4	0	4	0	98	73	52	74.333333	#624934	#4a4a4a

CHAPTER

THREE

MINIMAL MODEL

[10]:
```
try:
    import predicode as pc
except:
    !pip install git+https://github.com/sflippl/predicode
    import predicode as pc
```

'predicode' contains several[1] high-level interfaces to the more general hierarchical model[2]. We will use the minimal model, consisting of an input layer and one latent layer with a specified number of dimensions as an example. As an example, we will use an artificial dataset as presented in the previous chapter.

[3]:
```
art = pc.DecayingMultiNormal(dimensions = 10,
                             samples = 100)
```

The minimal model can be fitted by providing input data and the number of latent dimensions to the class 'pc.MinimalHierarchicalModel':

[4]:
```
hpc = pc.MinimalHierarchicalModel(input_data = art.data,
                                  latent_dimensions = 4)
```

```
WARNING: Logging before flag parsing goes to stderr.
W0824 15:59:17.338732 139821115447104 estimator.py:1811] Using temporary folder as␣
↪model directory: /tmp/tmp8kmjnddb
```

3.1 State Estimation

By default the weights of the minimal model are initialized as the first PCA components. This is the optimal solution for the minimal model and can be used to study state estimation. Since a predictive coding model can improve either by adapting its states or its weights, we first need to specify which of the two can currently be modified:

[5]:
```
hpc.activate('state')
```

(Technically, this would not have been necessary.) States are activated by default.

The minimal model can then be trained in order to extract the latent values.

[6]:
```
hpc.train()
```

[1] so far one
[2] not yet a thing

```
W0824 15:59:18.510554 139821115447104 deprecation.py:323] From /home/sflippl/.local/
↪lib/python3.7/site-packages/tensorflow/python/training/training_util.py:236:␣
↪Variable.initialized_value (from tensorflow.python.ops.variables) is deprecated and␣
↪will be removed in a future version.
Instructions for updating:
Use Variable.read_value. Variables in 2.X are initialized automatically both in eager␣
↪and graph (inside tf.defun) contexts.
W0824 15:59:18.523109 139821115447104 deprecation_wrapper.py:119] From /home/sflippl/.
↪local/lib/python3.7/site-packages/predicode/hierarchical/interfaces/minimal_model.
↪py:24: The name tf.feature_column.input_layer is deprecated. Please use tf.compat.
↪v1.feature_column.input_layer instead.

W0824 15:59:18.524461 139821115447104 deprecation.py:323] From /home/sflippl/.local/
↪lib/python3.7/site-packages/tensorflow/python/feature_column/feature_column.py:205:␣
↪NumericColumn._get_dense_tensor (from tensorflow.python.feature_column.
↪feature_column_v2) is deprecated and will be removed in a future version.
Instructions for updating:
The old _FeatureColumn APIs are being deprecated. Please use the new FeatureColumn␣
↪APIs instead.
W0824 15:59:18.525506 139821115447104 deprecation.py:323] From /home/sflippl/.local/
↪lib/python3.7/site-packages/tensorflow/python/feature_column/feature_column.
↪py:2115: NumericColumn._transform_feature (from tensorflow.python.feature_column.
↪feature_column_v2) is deprecated and will be removed in a future version.
Instructions for updating:
The old _FeatureColumn APIs are being deprecated. Please use the new FeatureColumn␣
↪APIs instead.
W0824 15:59:18.532030 139821115447104 deprecation.py:323] From /home/sflippl/.local/
↪lib/python3.7/site-packages/tensorflow/python/feature_column/feature_column.py:206:␣
↪NumericColumn._variable_shape (from tensorflow.python.feature_column.
↪feature_column_v2) is deprecated and will be removed in a future version.
Instructions for updating:
The old _FeatureColumn APIs are being deprecated. Please use the new FeatureColumn␣
↪APIs instead.
W0824 15:59:18.574176 139821115447104 deprecation.py:323] From /home/sflippl/.local/
↪lib/python3.7/site-packages/predicode/hierarchical/interfaces/minimal_model.py:27:␣
↪dense (from tensorflow.python.layers.core) is deprecated and will be removed in a␣
↪future version.
Instructions for updating:
Use keras.layers.dense instead.
W0824 15:59:18.576800 139821115447104 deprecation.py:506] From /home/sflippl/.local/
↪lib/python3.7/site-packages/tensorflow/python/ops/init_ops.py:1251: calling␣
↪VarianceScaling.__init__ (from tensorflow.python.ops.init_ops) with dtype is␣
↪deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the␣
↪constructor
W0824 15:59:18.836281 139821115447104 deprecation_wrapper.py:119] From /home/sflippl/.
↪local/lib/python3.7/site-packages/predicode/hierarchical/interfaces/minimal_model.
↪py:10: The name tf.losses.mean_squared_error is deprecated. Please use tf.compat.v1.
↪losses.mean_squared_error instead.

W0824 15:59:18.848142 139821115447104 deprecation.py:323] From /home/sflippl/.local/
↪lib/python3.7/site-packages/tensorflow/python/ops/losses/losses_impl.py:121:␣
↪add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is␣
↪deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

(continues on next page)

(continued from previous page)

```
W0824 15:59:18.858359 139821115447104 deprecation_wrapper.py:119] From /home/sflippl/.
↪local/lib/python3.7/site-packages/predicode/hierarchical/interfaces/minimal_model.
↪py:19: The name tf.train.GradientDescentOptimizer is deprecated. Please use tf.
↪compat.v1.train.GradientDescentOptimizer instead.

W0824 15:59:18.859064 139821115447104 deprecation_wrapper.py:119] From /home/sflippl/.
↪local/lib/python3.7/site-packages/predicode/hierarchical/interfaces/minimal_model.
↪py:20: The name tf.train.get_global_step is deprecated. Please use tf.compat.v1.
↪train.get_global_step instead.

W0824 15:59:19.296718 139821115447104 basic_session_run_hooks.py:724] It seems that␣
↪global step (tf.train.get_global_step) has not been increased. Current value (could␣
↪be stable): 6 vs previous value: 6. You could increase the global step by passing␣
↪tf.train.get_global_step() to Optimizer.apply_gradients or Optimizer.minimize.
W0824 15:59:19.312877 139821115447104 basic_session_run_hooks.py:724] It seems that␣
↪global step (tf.train.get_global_step) has not been increased. Current value (could␣
↪be stable): 27 vs previous value: 27. You could increase the global step by passing␣
↪tf.train.get_global_step() to Optimizer.apply_gradients or Optimizer.minimize.
W0824 15:59:19.332931 139821115447104 basic_session_run_hooks.py:724] It seems that␣
↪global step (tf.train.get_global_step) has not been increased. Current value (could␣
↪be stable): 62 vs previous value: 62. You could increase the global step by passing␣
↪tf.train.get_global_step() to Optimizer.apply_gradients or Optimizer.minimize.
W0824 15:59:19.340380 139821115447104 basic_session_run_hooks.py:724] It seems that␣
↪global step (tf.train.get_global_step) has not been increased. Current value (could␣
↪be stable): 74 vs previous value: 74. You could increase the global step by passing␣
↪tf.train.get_global_step() to Optimizer.apply_gradients or Optimizer.minimize.
W0824 15:59:19.349026 139821115447104 basic_session_run_hooks.py:724] It seems that␣
↪global step (tf.train.get_global_step) has not been increased. Current value (could␣
↪be stable): 86 vs previous value: 86. You could increase the global step by passing␣
↪tf.train.get_global_step() to Optimizer.apply_gradients or Optimizer.minimize.
```

[6]:
```
<tensorflow_estimator.python.estimator.estimator.Estimator at 0x7f2a56b03b38>
```

The usual Tensorflow framework can be used in this context:

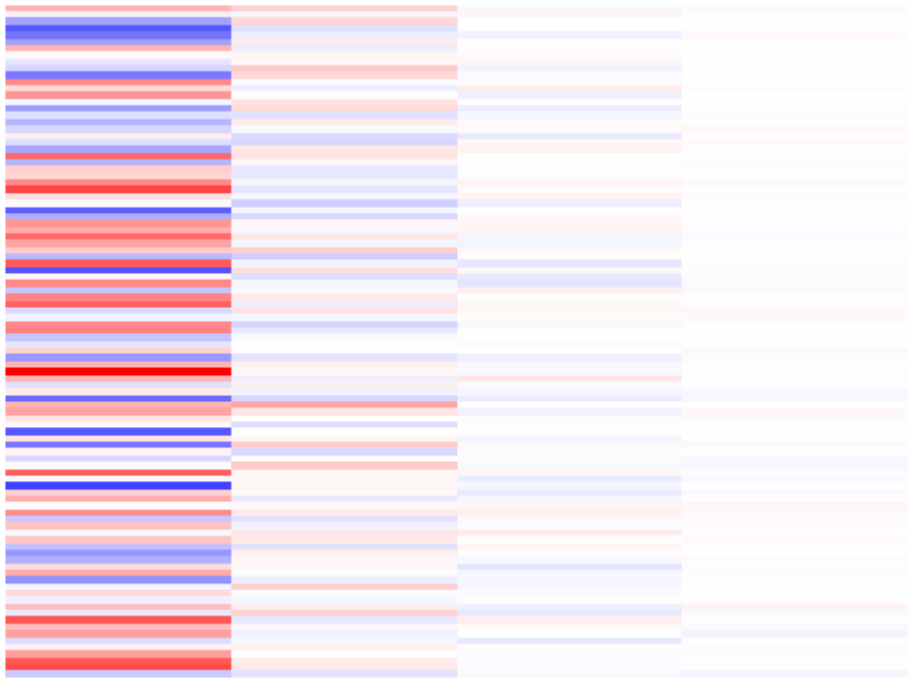
[7]:
```
hpc.evaluate()
```

```
W0824 15:59:23.045202 139821115447104 deprecation.py:323] From /home/sflippl/.local/
↪lib/python3.7/site-packages/tensorflow/python/training/saver.py:1276:␣
↪checkpoint_exists (from tensorflow.python.training.checkpoint_management) is␣
↪deprecated and will be removed in a future version.
Instructions for updating:
Use standard file APIs to check for files with this prefix.
```

[7]:
```
{'loss': 3.4082903e-05, 'global_step': 10000}
```

The method ‘latent_values’ allows for the extraction of the latent values, which we here visualize using a heatmap:

[8]:
```
latent_values = hpc.latent_values()
import lazytools
lazytools.matrix_heatmap(latent_values, pole = 0)
```

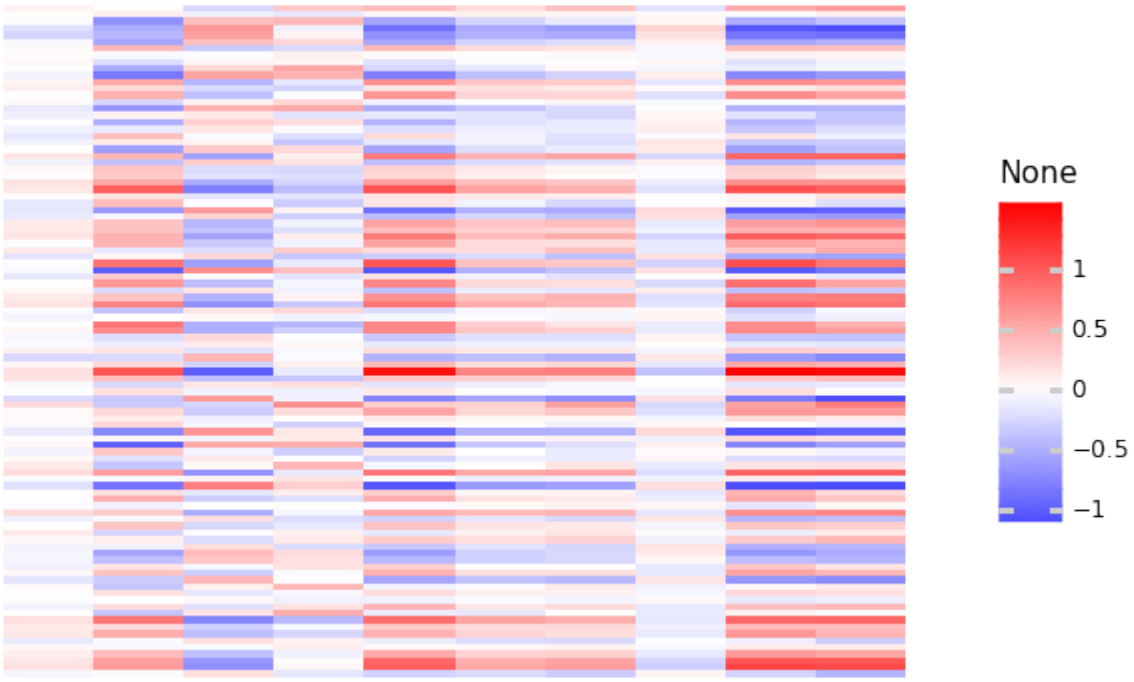


[8]:
```
<ggplot: (8738711660976)>
```

Similarly the predictions generated by the latent values can be studied:

[9]:
```
predictions = hpc.predict()
lazytools.matrix_heatmap(predictions, pole = 0)
```

```
W0824 15:59:23.500001 139821115447104 estimator.py:1000] Input graph does not use tf.
↪data.Dataset or contain a QueueRunner. That means predict yields forever. This is␣
↪probably a mistake.
```



[9]: <ggplot: (8738711440090)>

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search