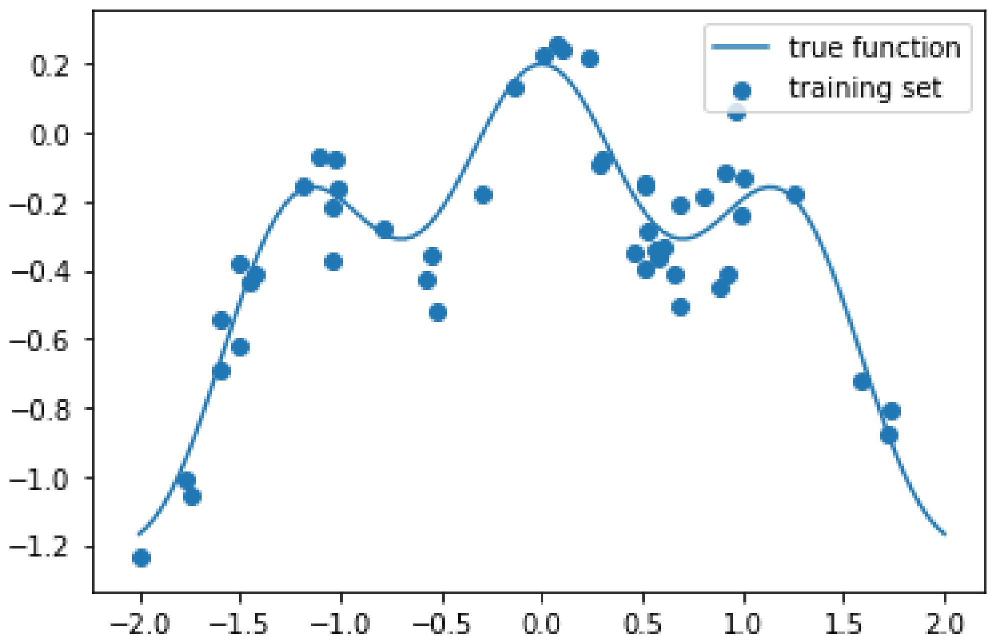
20200217_examples

February 16, 2020

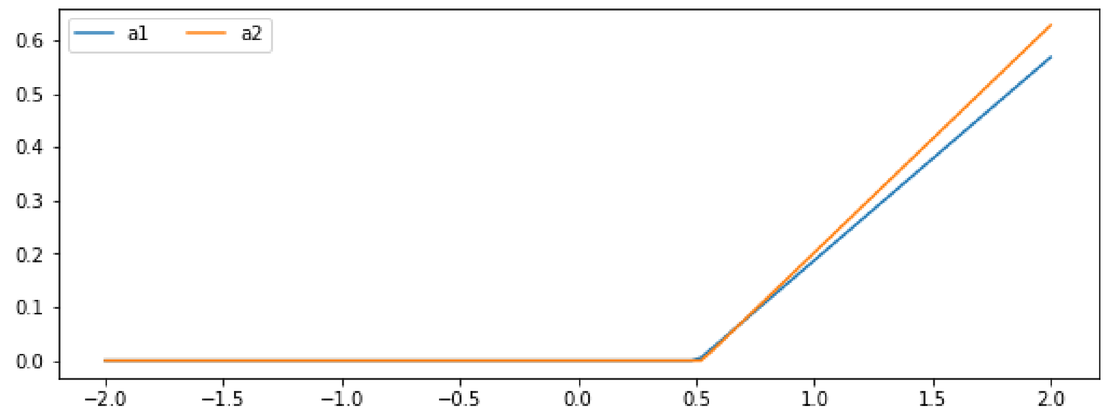
0.0.1 Data generation

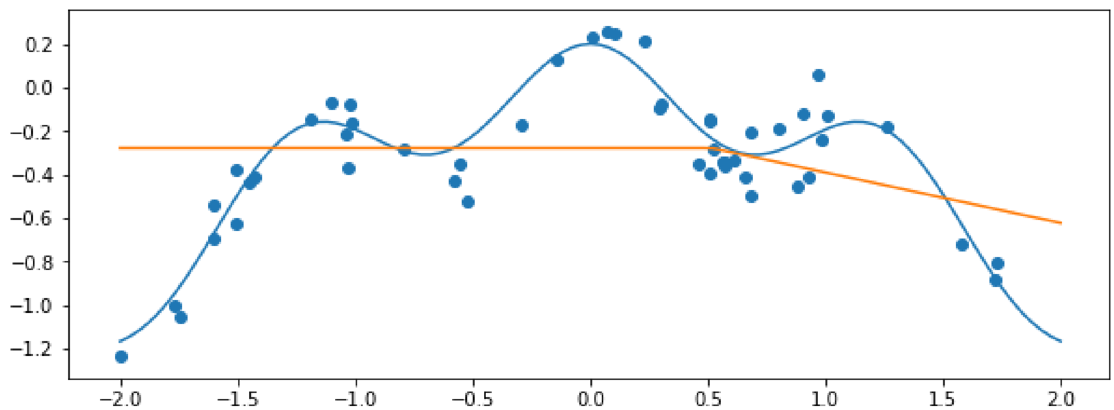


0.0.2 1 Hidden layer, varying number of units

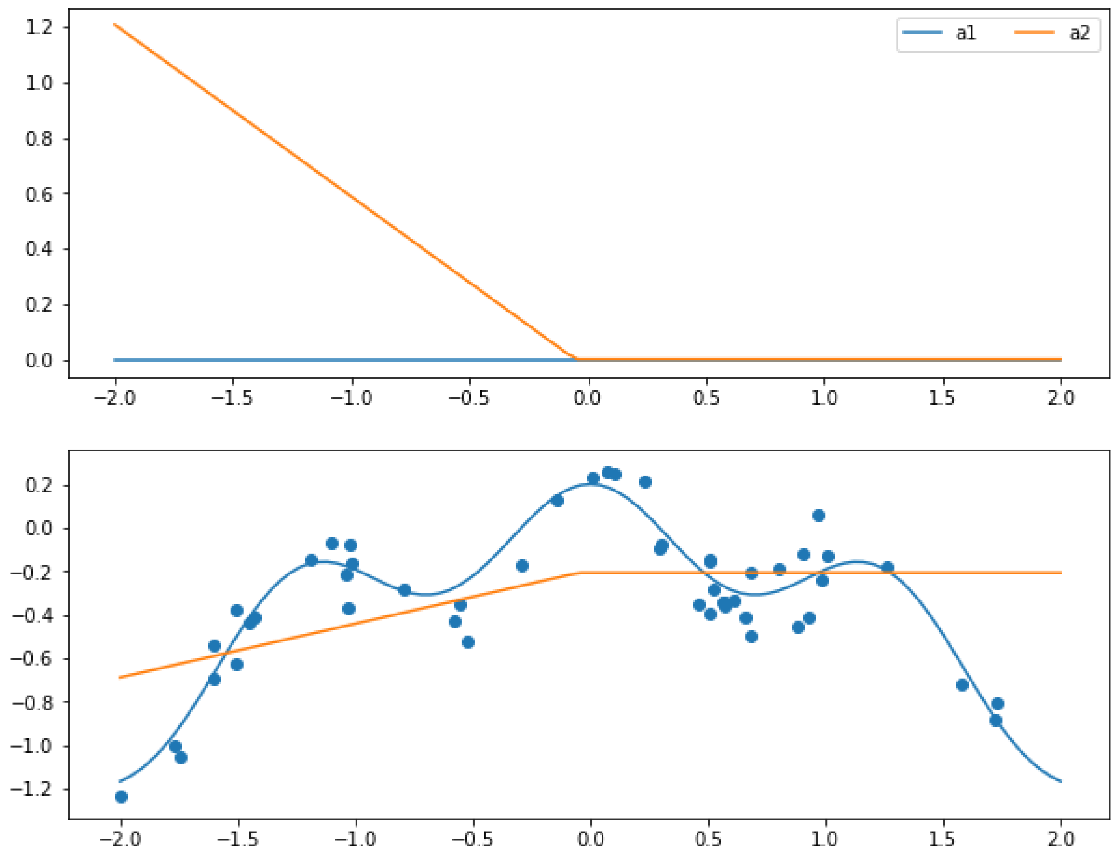
2 Units The following fits the same model 3 times with different seeds (basically, different randomly selected initial values for the parameters).

[301]:
```
hidden_units = [2]
np.random.seed(87462)
b_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
w_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
model_2units_a = fit_model_ex1(hidden_units, b_init_seeds, w_init_seeds)
plot_layers(model_2units_a, hidden_legend=True)
```


a1
a2
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

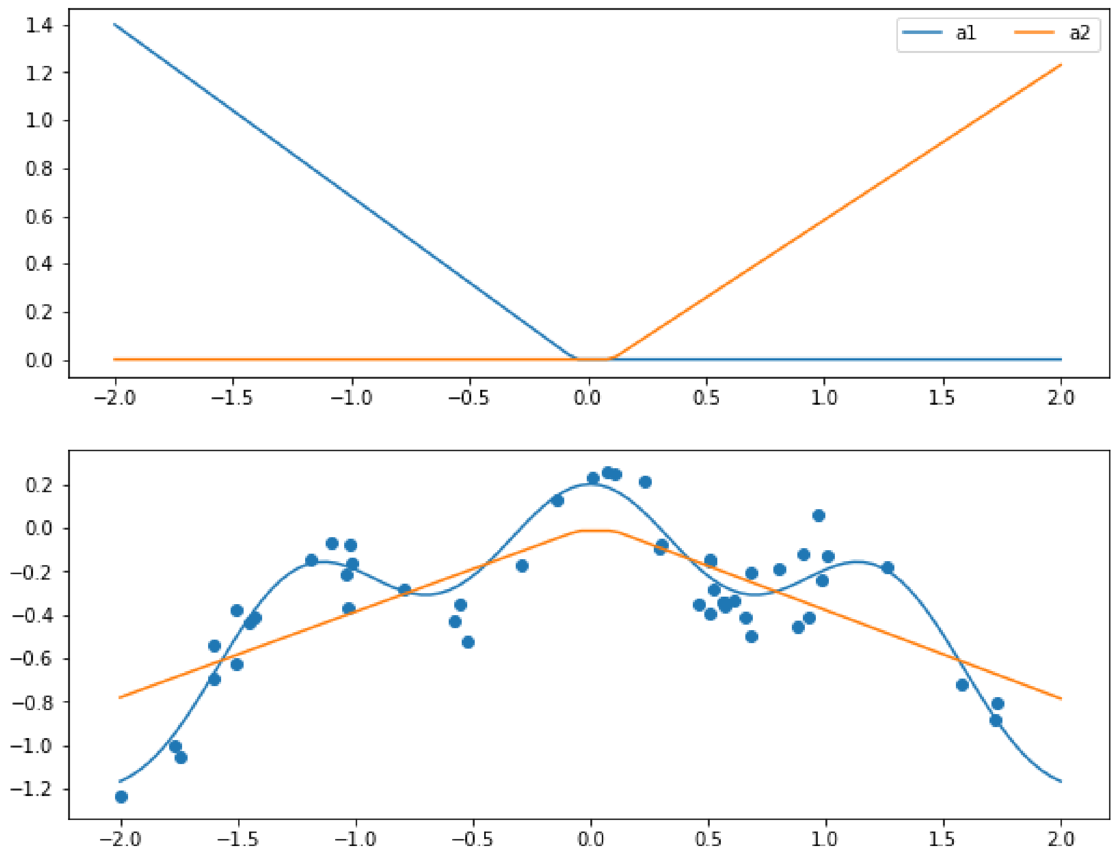

0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

[0]:
```
hidden_units = [2]
np.random.seed(874625)
b_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
w_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
model_2units_b = fit_model_ex1(hidden_units, b_init_seeds, w_init_seeds)
plot_layers(model_2units_b, hidden_legend=True)
```



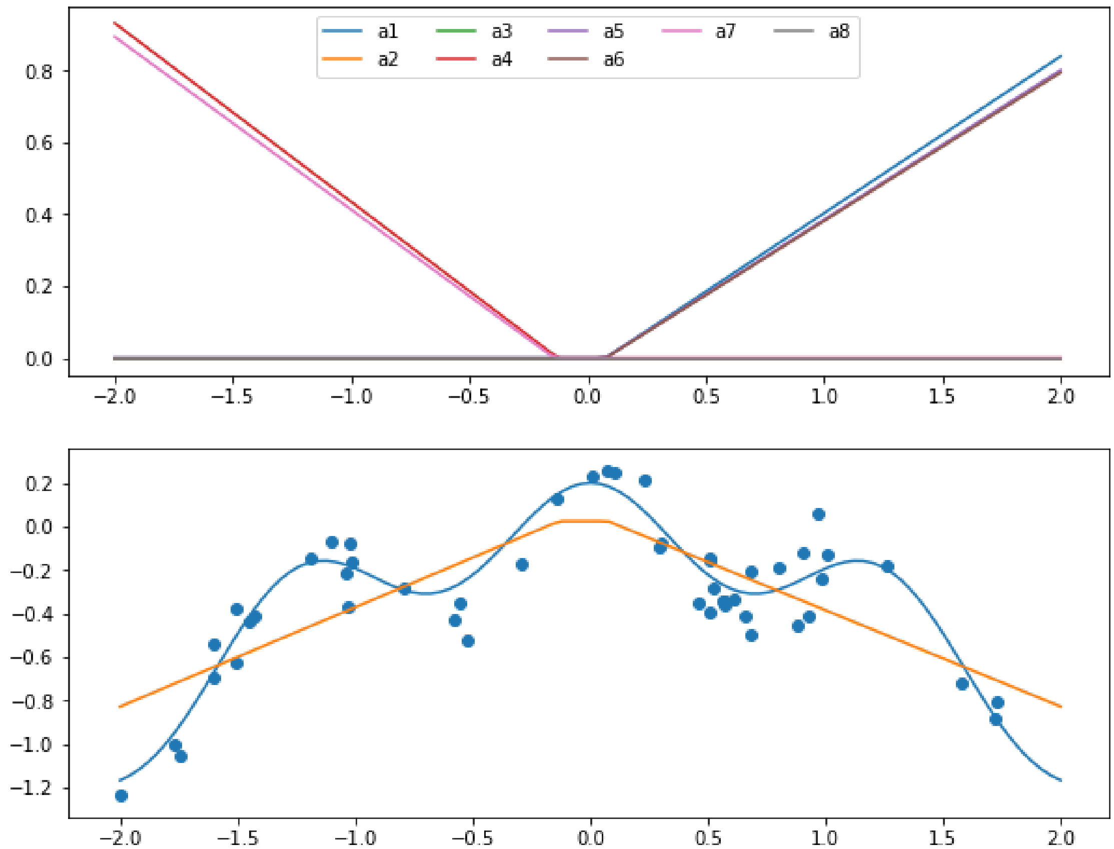
[0]:

```
hidden_units = [2]
np.random.seed(8746257)
b_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
w_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
model_2units_c = fit_model_ex1(hidden_units, b_init_seeds, w_init_seeds)
plot_layers(model_2units_c, hidden_legend=True)
```



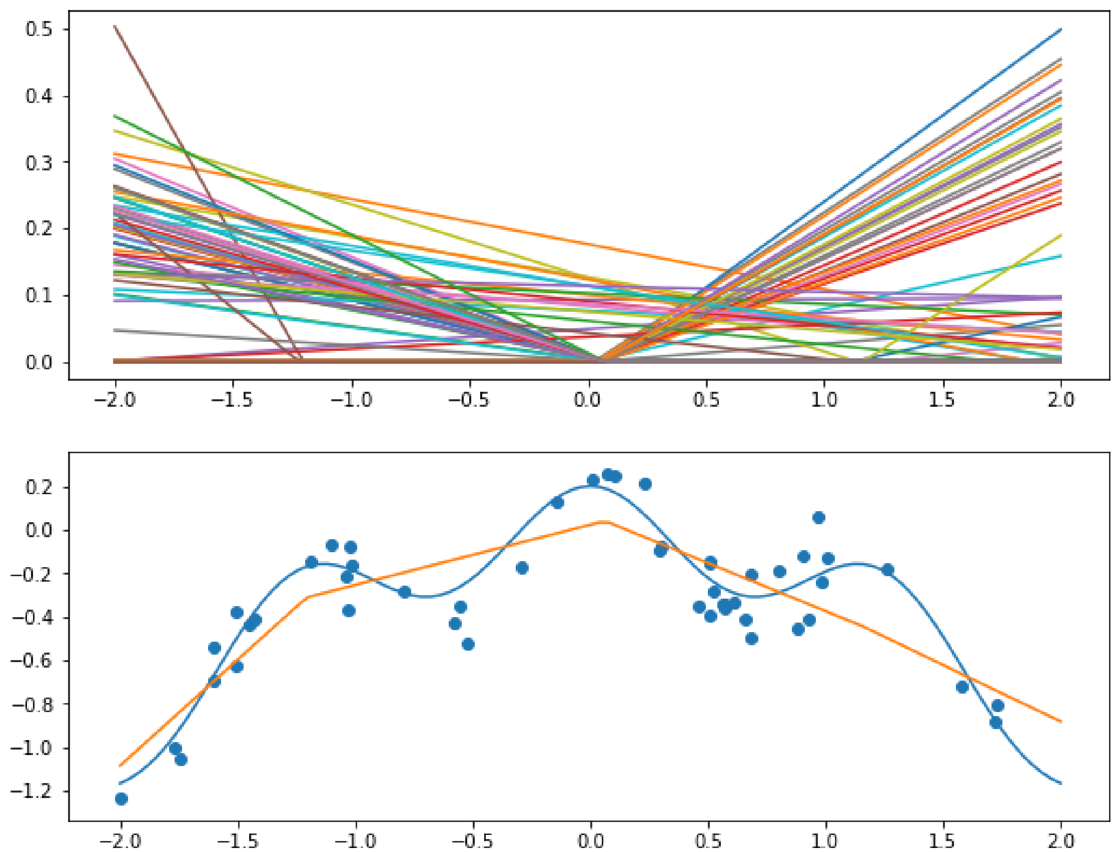
8 units

[0]:
```
hidden_units = [8]
np.random.seed(8746)
b_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
w_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
model_8units = fit_model_ex1(hidden_units, b_init_seeds, w_init_seeds)
plot_layers(model_8units, hidden_legend=True)
```



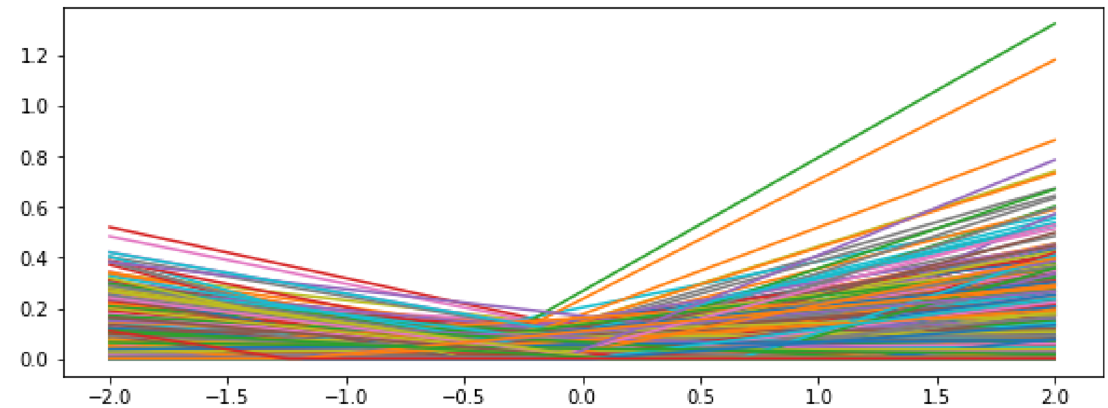
0.0.3 128 Units

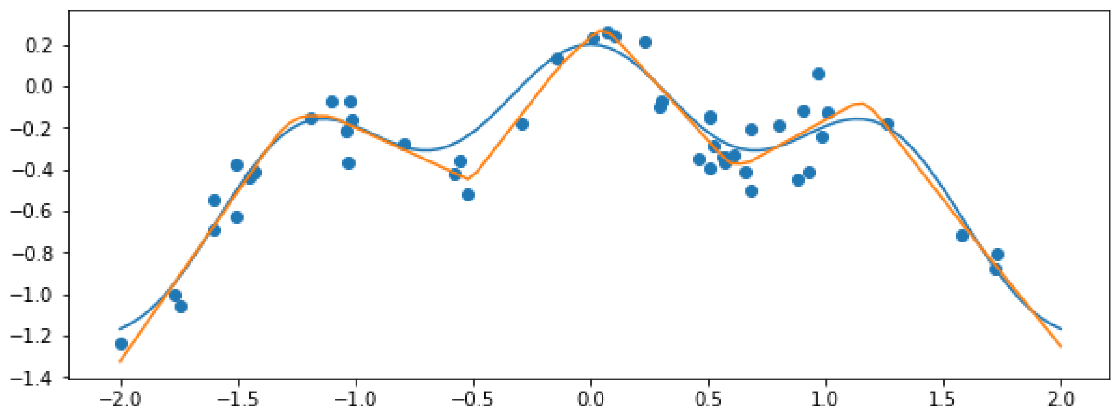
[0]:
```
hidden_units = [128]
np.random.seed(8746)
b_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
w_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
model_128units = fit_model_ex1(hidden_units, b_init_seeds, w_init_seeds)
plot_layers(model_128units, hidden_legend=False)
```



0.0.4 1024 Units

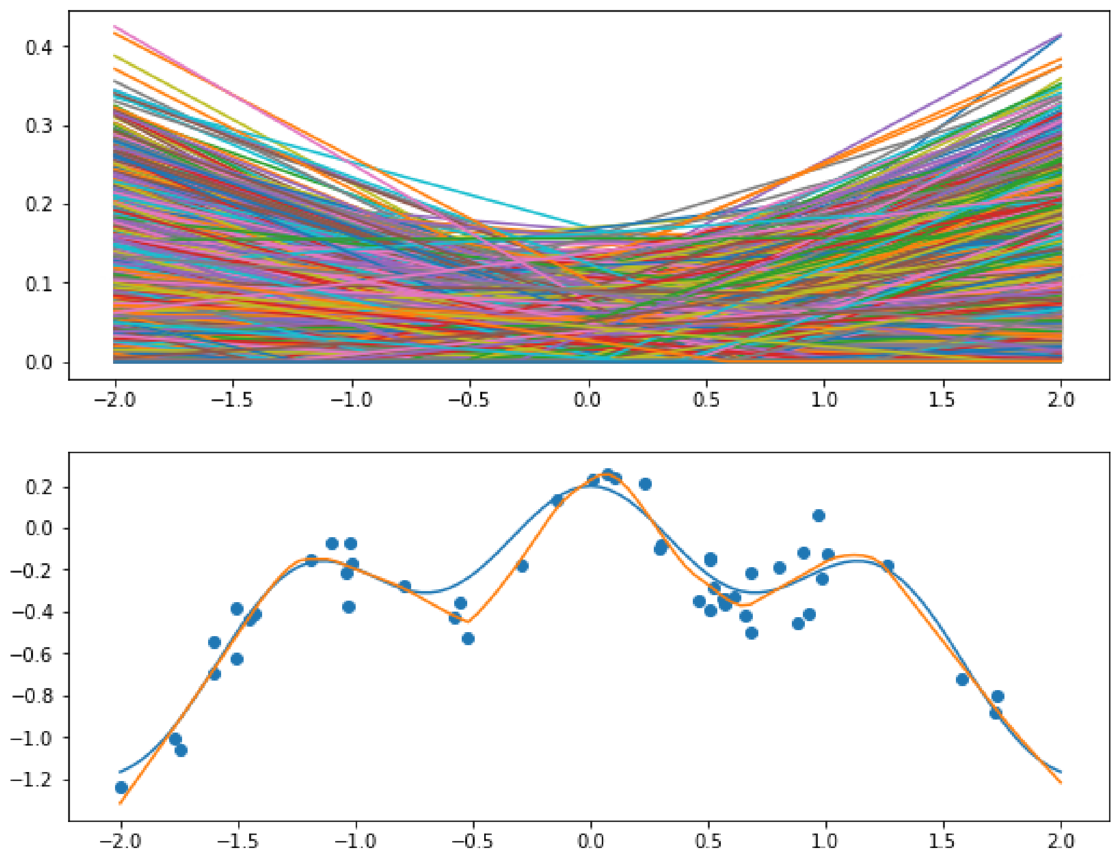
[0]:
```
hidden_units = [1024]
np.random.seed(8746)
b_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
w_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
model_1024units = fit_model_ex1(hidden_units, b_init_seeds, w_init_seeds)
plot_layers(model_1024units, hidden_legend=False)
```





0.0.5 8192 Units

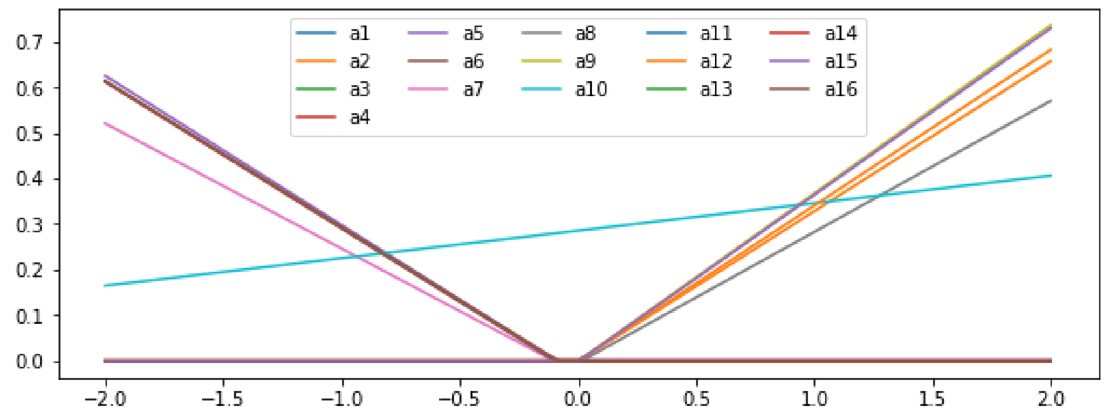
[0]:
```
hidden_units = [8192]
np.random.seed(8746)
b_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
w_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
model_8192units = fit_model_ex1(hidden_units, b_init_seeds, w_init_seeds)
plot_layers(model_8192units, hidden_legend=False)
```

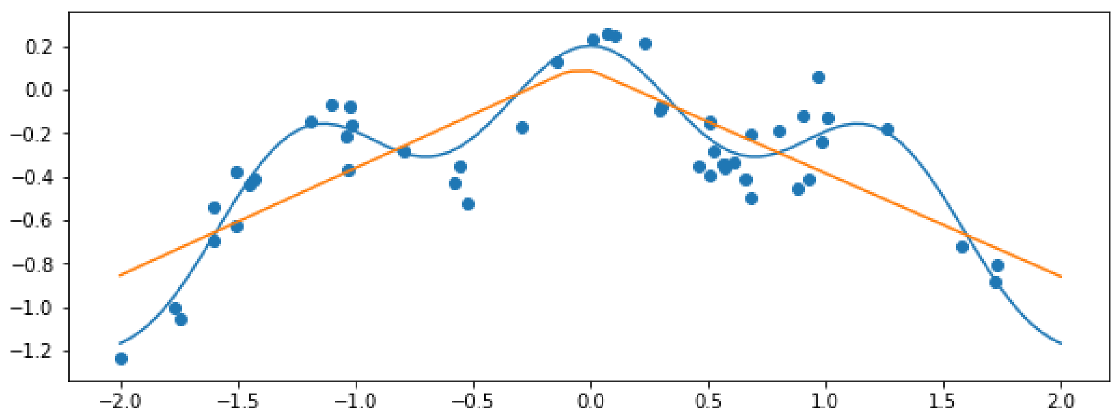


0.0.6 Models with Multiple Hidden Layers, 16 units per layer, relu activation

1 hidden layer

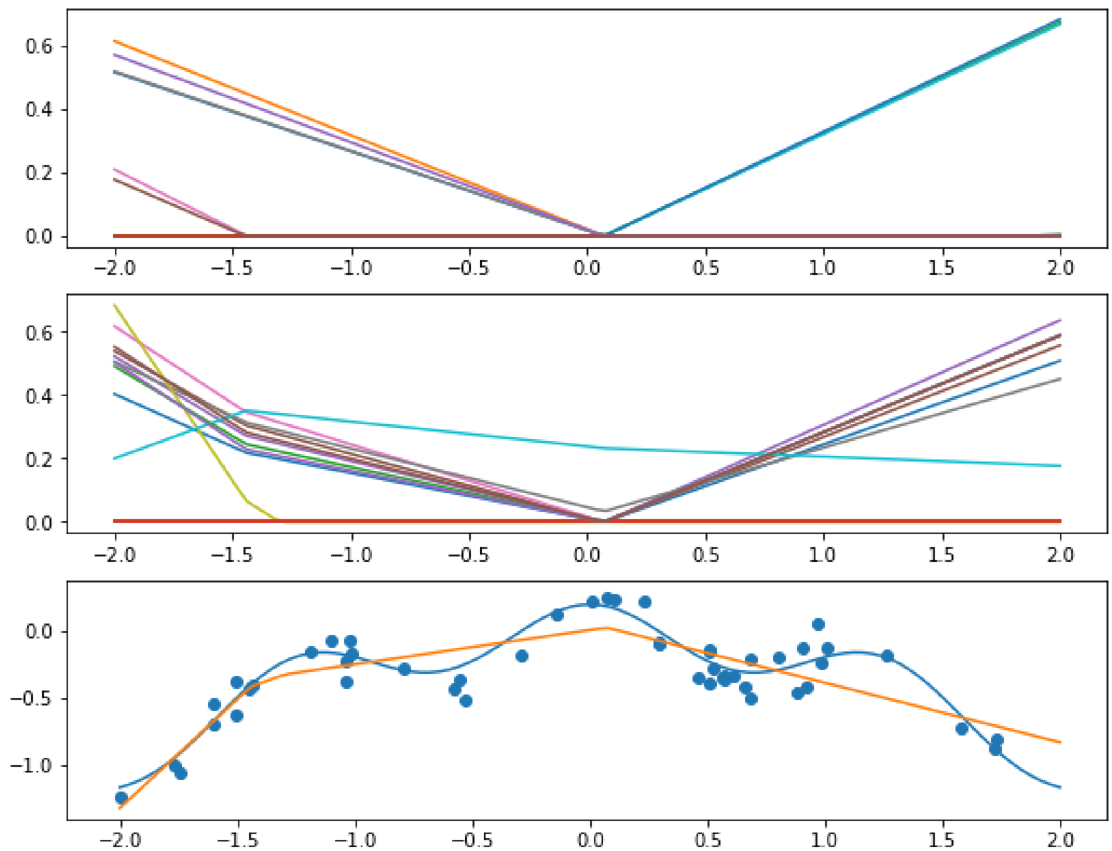
[0]:
```
hidden_units = [16]
np.random.seed(8746)
b_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
w_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
model_1layer = fit_model_ex1(hidden_units, b_init_seeds, w_init_seeds)
plot_layers(model_1layer, hidden_legend=True)
```





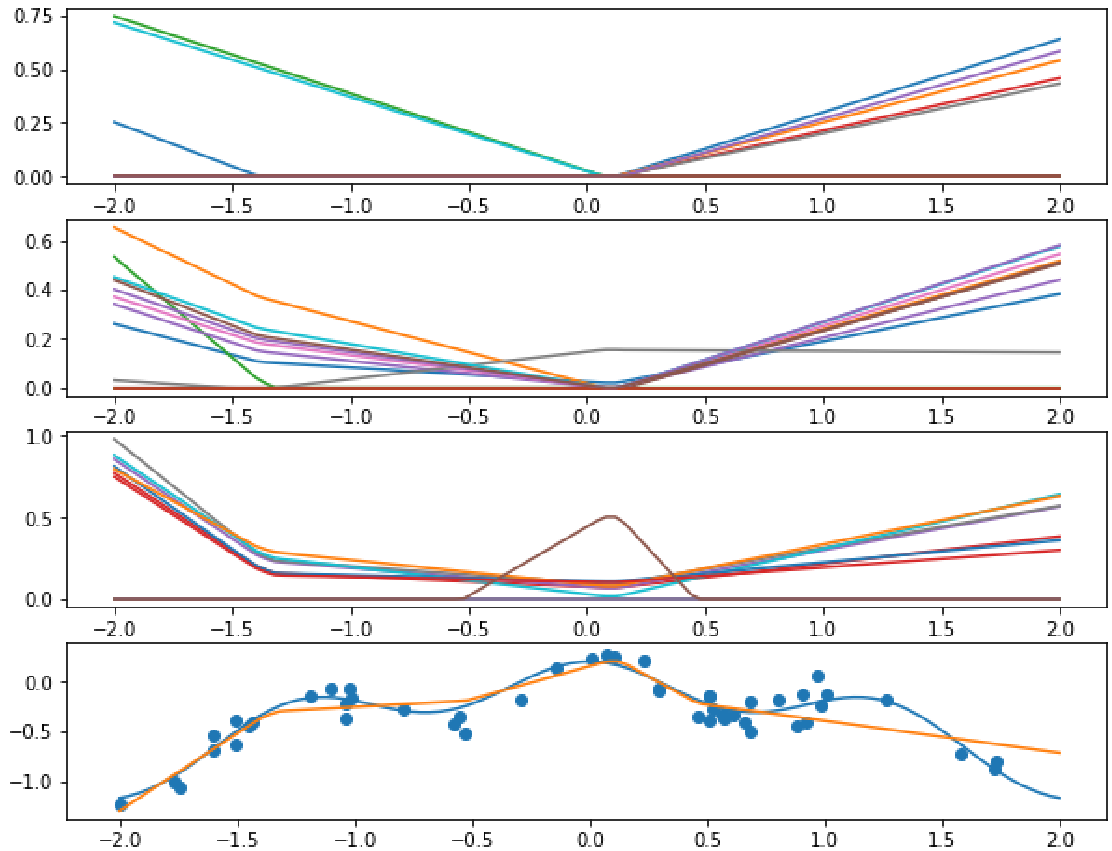
2 hidden layers

[0]:
```
hidden_units = [16, 16]
np.random.seed(8746)
b_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
w_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
model_2layers = fit_model_ex1(hidden_units, b_init_seeds, w_init_seeds)
plot_layers(model_2layers, hidden_legend=False)
```



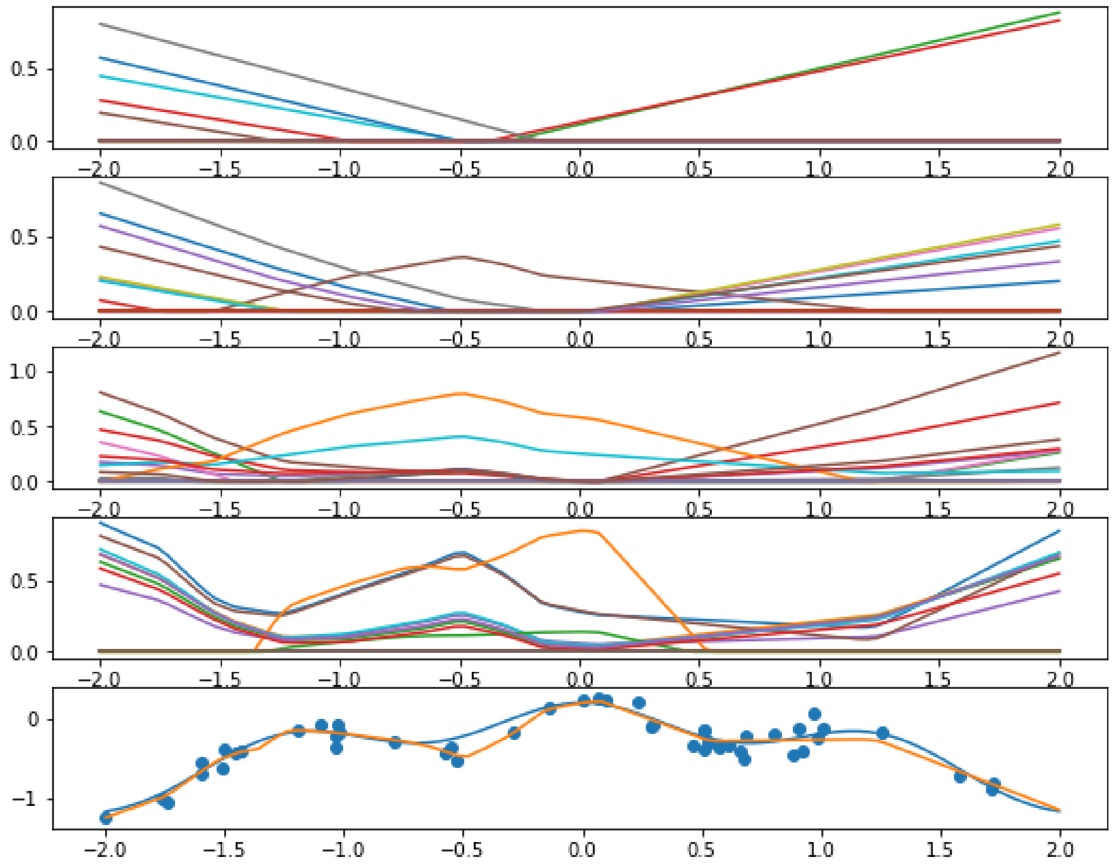
3 hidden layers

[0]:
```
hidden_units = [16, 16, 16]
np.random.seed(8746)
b_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
w_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
model_3layers = fit_model_ex1(hidden_units, b_init_seeds, w_init_seeds)
plot_layers(model_3layers, hidden_legend=False)
```



4 hidden layers

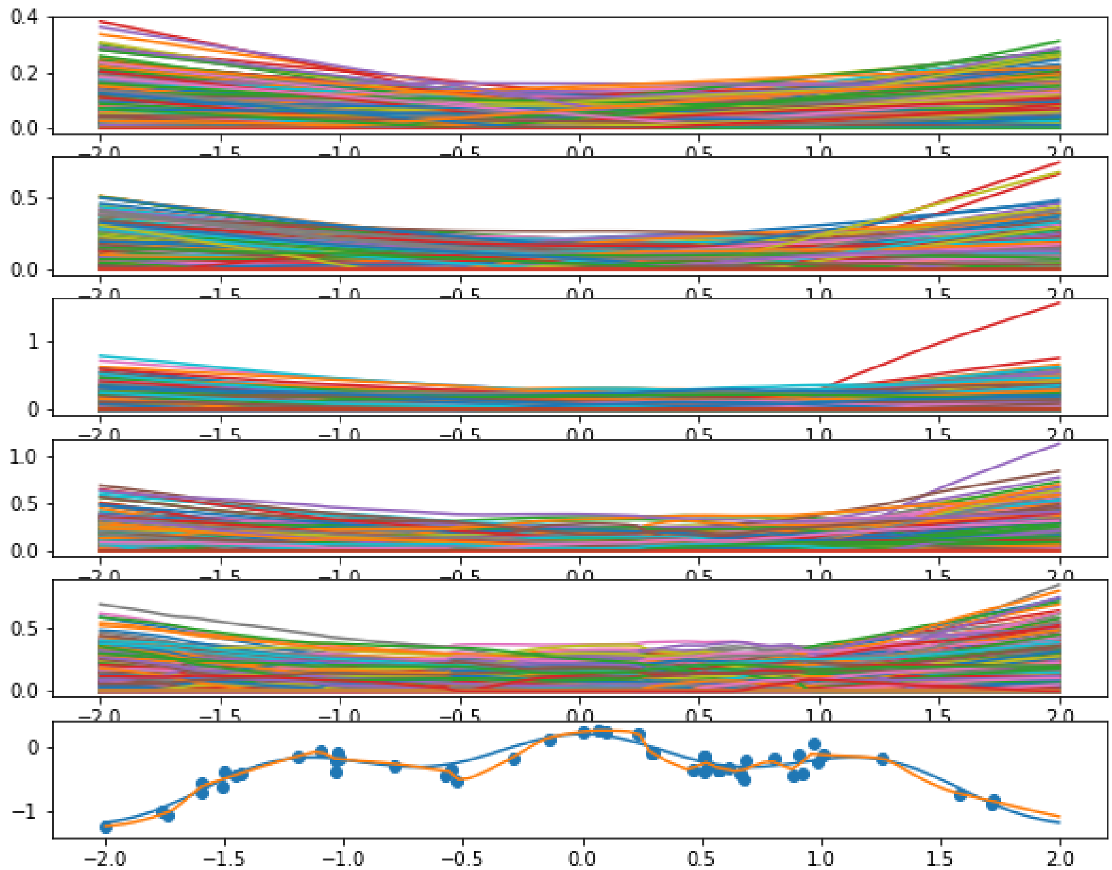
[0]:
```
hidden_units = [16, 16, 16, 16]
np.random.seed(8746)
b_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
w_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
model_4layers = fit_model_ex1(hidden_units, b_init_seeds, w_init_seeds)
plot_layers(model_4layers, hidden_legend=False)
```



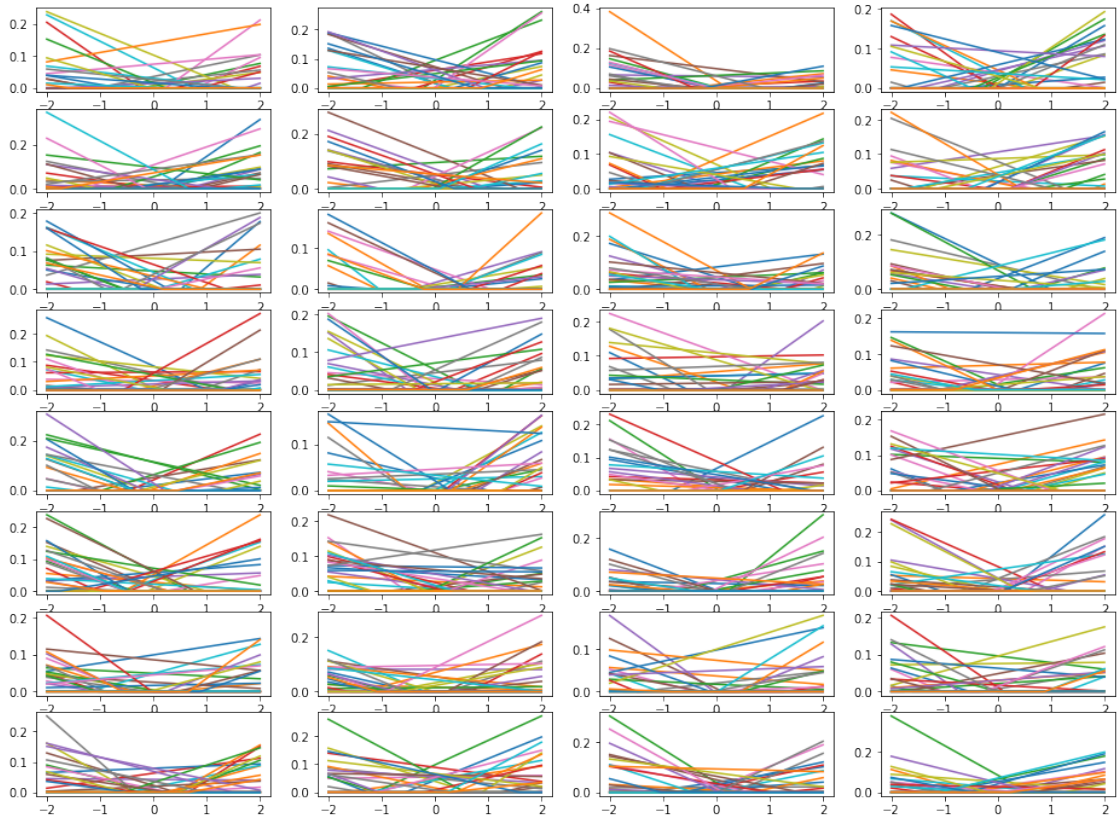
0.0.7 Model with so many hidden layers and units

5 hidden layers, each with 1024 units.

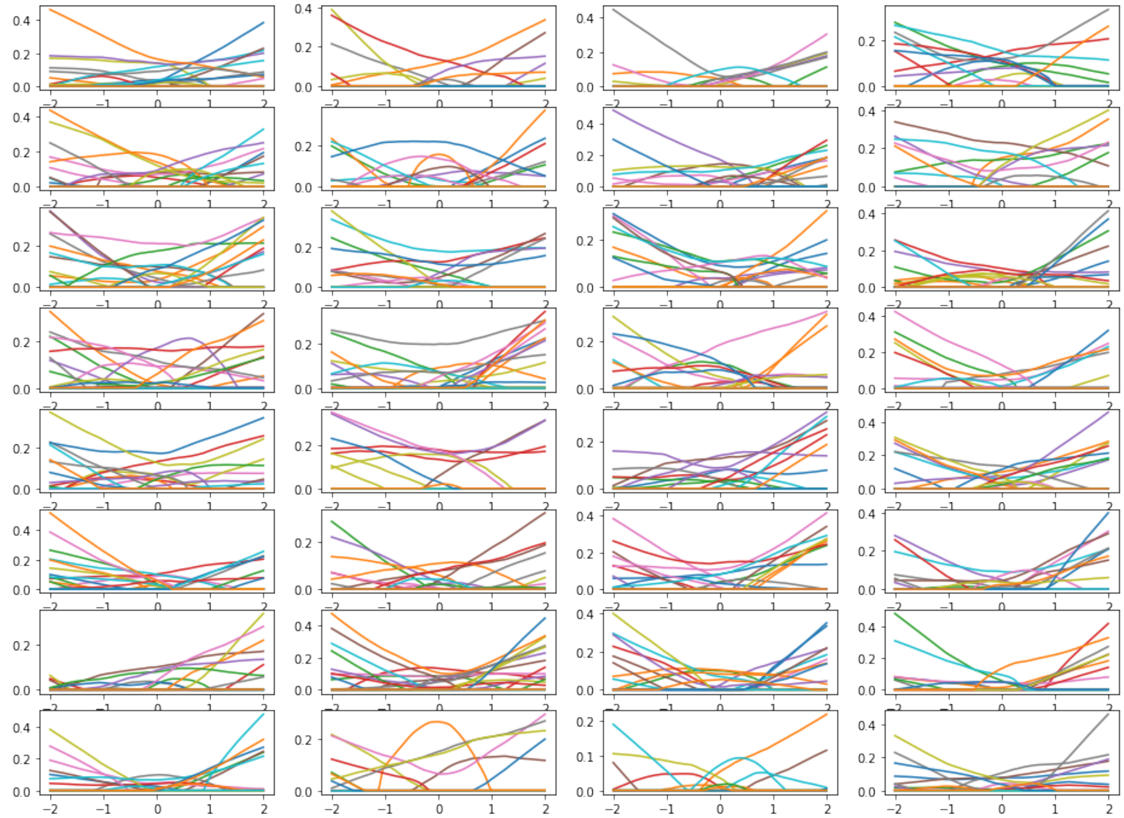
[17]:
```
hidden_units = [1024, 1024, 1024, 1024, 1024]
np.random.seed(8746)
b_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
w_init_seeds = np.random.randint(1, 1e6, size = sum(hidden_units)+1)
model_somany = fit_model_ex1(hidden_units, b_init_seeds, w_init_seeds)
plot_layers(model_somany, hidden_legend=False)
```



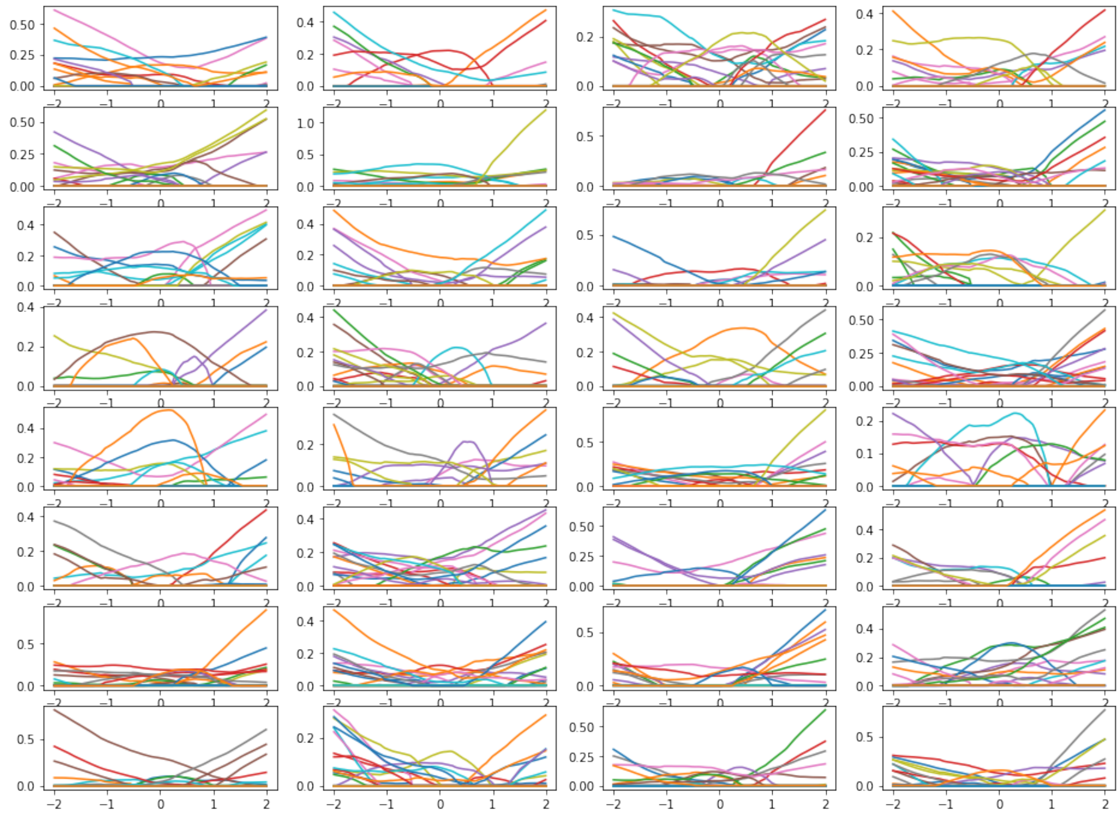
[0]:
```
plot_layer_activations_facetted(model_somany, 0)
```



[0]:
```
plot_layer_activations_facetted(model_somany, 1)
```

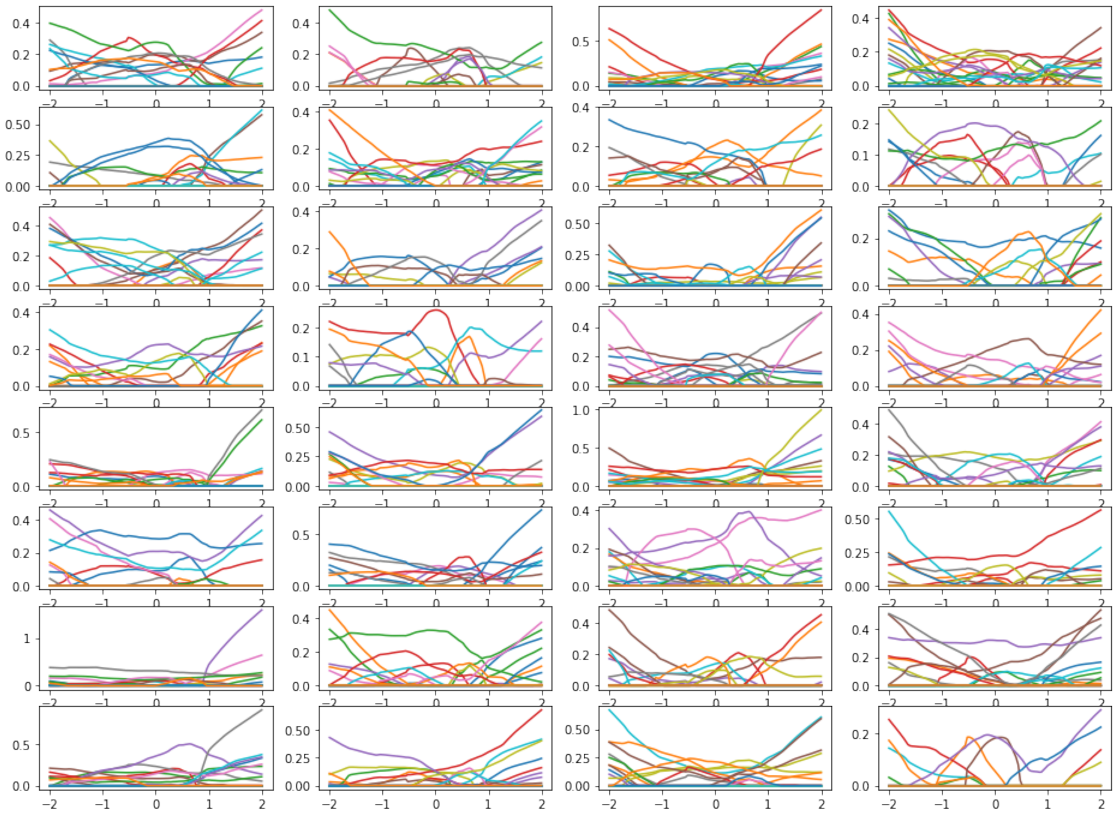


[0]:
```
plot_layer_activations_facetted(model_somany, 2)
```

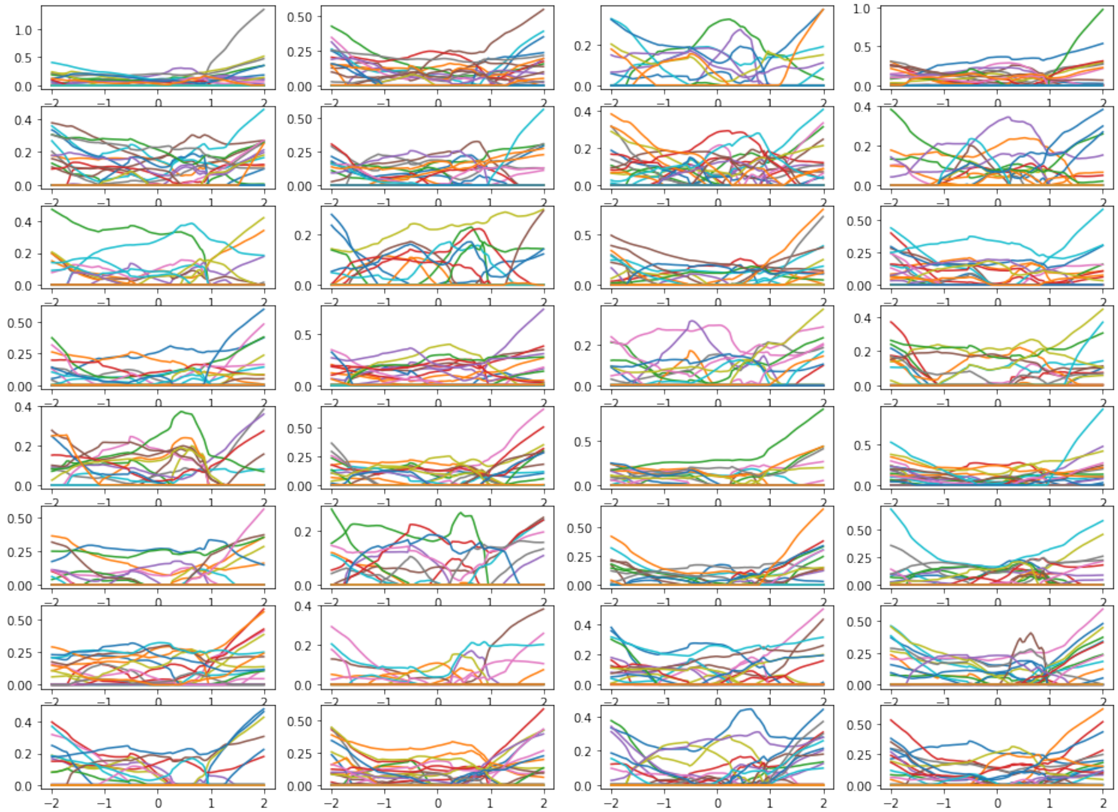


[0]:

```
plot_layer_activations_facetted(model_somany, 3)
```



[309]:
```
plot_layer_activations_facetted(model_somany, 4)
```



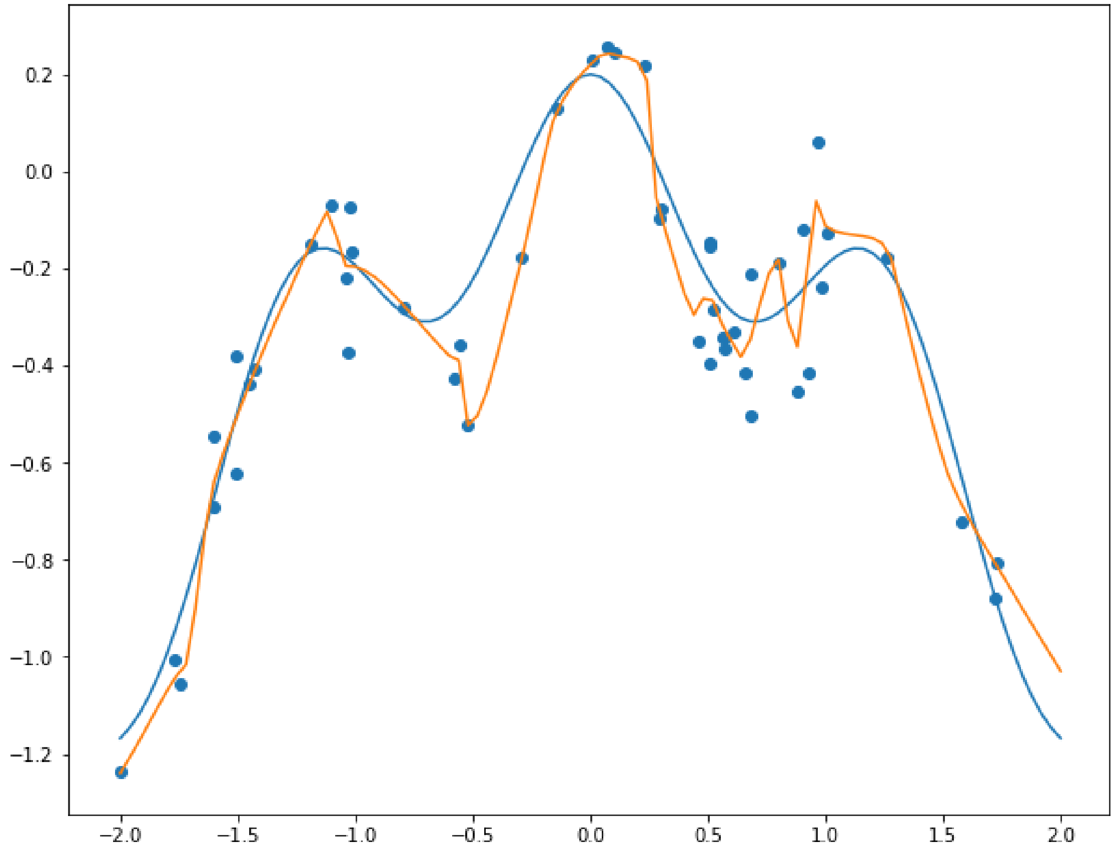
0.0.8 How are these models doing in terms of training and validation set performance?

[310]:

	train_mse	val_mse
model_2units_c	0.0444497	0.0575317
model_8units	0.0427819	0.0535399
model_128units	0.033027	0.0429847
model_1024units	0.00960591	0.0246087
model_8192units	0.00923209	0.0246102
model_1layer	0.0421595	0.0513442
model_2layers	0.0310402	0.0433764
model_3layers	0.0239409	0.0433056
model_4layers	0.0110532	0.0273052
model_somany	0.00613459	0.0296568

0.0.9 How can we fix up that last model?

[311]:
```
plot_layers(model_somany, hidden_legend=False, include_hidden = False)
```



[0]:

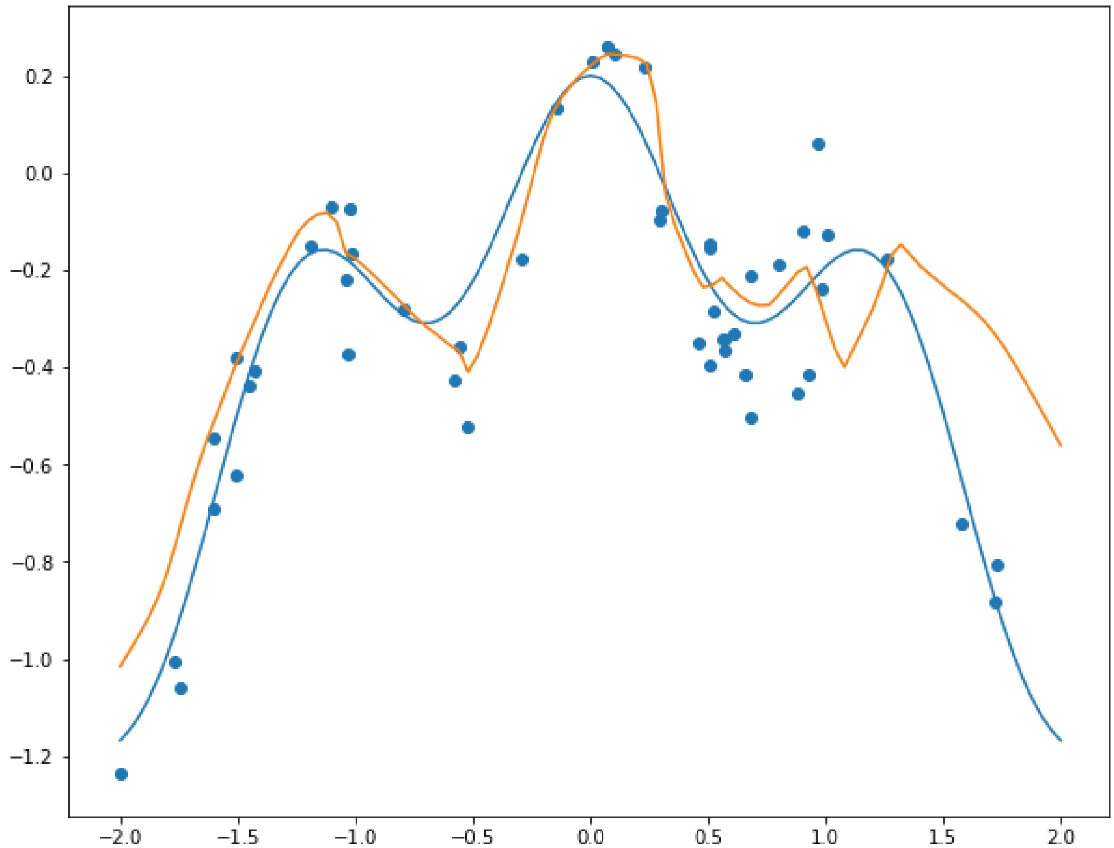
```
(w, b) = model_somany.layers[0].get_weights()
orig_w = w.copy()
```

[0]:

```
change_pt = -b / orig_w
inds_to_fix = np.where(np.logical_and(change_pt >= 0.5, change_pt <= 1.0))
w[inds_to_fix] = orig_w[inds_to_fix] * 0.1

model_somany.layers[0].set_weights((w, b))
```

[314]:

```
plot_layers(model_somany, hidden_legend=False, include_hidden = False)
```



0.0.10 Regularizing the model from the start:

We modify the cost function to be the sum of the negative log likelihood and a penalty on the size of the (squared) weights:

$$J(b, w) = -\ell(b, w) + \sum_{l=1}^{L} \lambda_l \left\{ \sum_{i,j} \left(w_{i,j}^{[l]} \right)^2 \right\}$$

- We want to minimize $J(b, w)$. Think of this in 2 parts:
  - Minimize negative log-likelihood: want a good fit to the data
  - Minimize penalty terms like $\lambda_l \sum_{i,j} \left(w_{i,j}^{[l]} \right)^2$: weight parameter estimates should not be large.

Note that there is a separate penalty parameter for each layer l.

Names for this:

- L_2 Regularization (L_2 norm is another name for Euclidean distance, which is based on sums of squares)
- Penalized estimation
- Ridge regression
- Weight decay

0.0.11 Penalty 0.01 on all layers

Two main changes:

- `kernel_regularizer=regularizers.l2(penalty)` for each dense layer
- `metrics = ['mean_squared_error']` when compiling – note that our cost function now includes penalties and is not a direct measure of validation or test set performance.

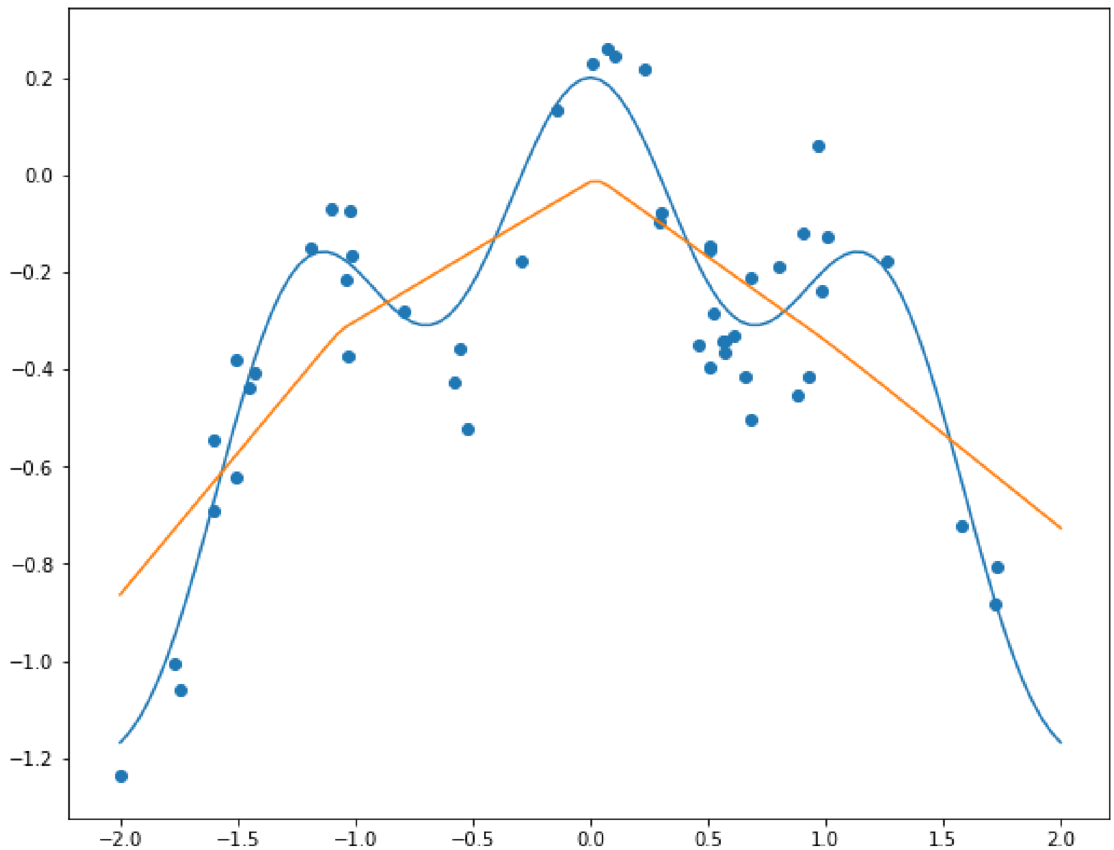
[26]:
```
# import regularizers
from keras import regularizers

penalty = 0.01

np.random.seed(8746)
b_init_seeds = np.random.randint(1, 1e6, size = 6)
w_init_seeds = np.random.randint(1, 1e6, size = 6)

regularized_model = models.Sequential()

# add hidden layers
b_initializer = initializers.RandomNormal(seed=b_init_seeds[0])
w_initializer = initializers.RandomNormal(seed=w_init_seeds[0])
regularized_model.add(layers.Dense(1024,
    activation = 'relu',
    input_shape = (1,),
```

```
    bias_initializer = b_initializer,
    kernel_initializer = w_initializer,
    kernel_regularizer=regularizers.l2(penalty)))

b_initializer = initializers.RandomNormal(seed=b_init_seeds[1])
w_initializer = initializers.RandomNormal(seed=w_init_seeds[1])
regularized_model.add(layers.Dense(1024,
    activation = 'relu',
    bias_initializer = b_initializer,
    kernel_initializer = w_initializer,
    kernel_regularizer=regularizers.l2(penalty)))

b_initializer = initializers.RandomNormal(seed=b_init_seeds[2])
w_initializer = initializers.RandomNormal(seed=w_init_seeds[2])
regularized_model.add(layers.Dense(1024,
    activation = 'relu',
    bias_initializer = b_initializer,
    kernel_initializer = w_initializer,
    kernel_regularizer=regularizers.l2(penalty)))

b_initializer = initializers.RandomNormal(seed=b_init_seeds[3])
w_initializer = initializers.RandomNormal(seed=w_init_seeds[3])
regularized_model.add(layers.Dense(1024,
    activation = 'relu',
    bias_initializer = b_initializer,
    kernel_initializer = w_initializer,
    kernel_regularizer=regularizers.l2(penalty)))

b_initializer = initializers.RandomNormal(seed=b_init_seeds[4])
w_initializer = initializers.RandomNormal(seed=w_init_seeds[4])
regularized_model.add(layers.Dense(1024,
    activation = 'relu',
    bias_initializer = b_initializer,
    kernel_initializer = w_initializer,
    kernel_regularizer=regularizers.l2(penalty)))

# add output layer
b_initializer = initializers.RandomNormal(seed=b_init_seeds[5])
w_initializer = initializers.RandomNormal(seed=w_init_seeds[5])
regularized_model.add(layers.Dense(1,
    activation = 'linear',
    bias_initializer = b_initializer,
    kernel_initializer = w_initializer,
    kernel_regularizer=regularizers.l2(penalty)))

# compile and fit model
```

```
regularized_model.compile(optimizer = 'adam', loss = 'mean_squared_error',␣
 ↪metrics = ['mean_squared_error'])

regularized_model.fit(train_x, train_y,
  validation_data = (val_x, val_y),
  epochs = 1000,
  batch_size = train_x.shape[0],
  verbose = 0)
```

[26]: <keras.callbacks.History at 0x7f70d34c7400>

[27]:
```
plot_layers(regularized_model, hidden_legend=False, include_hidden = False)
```



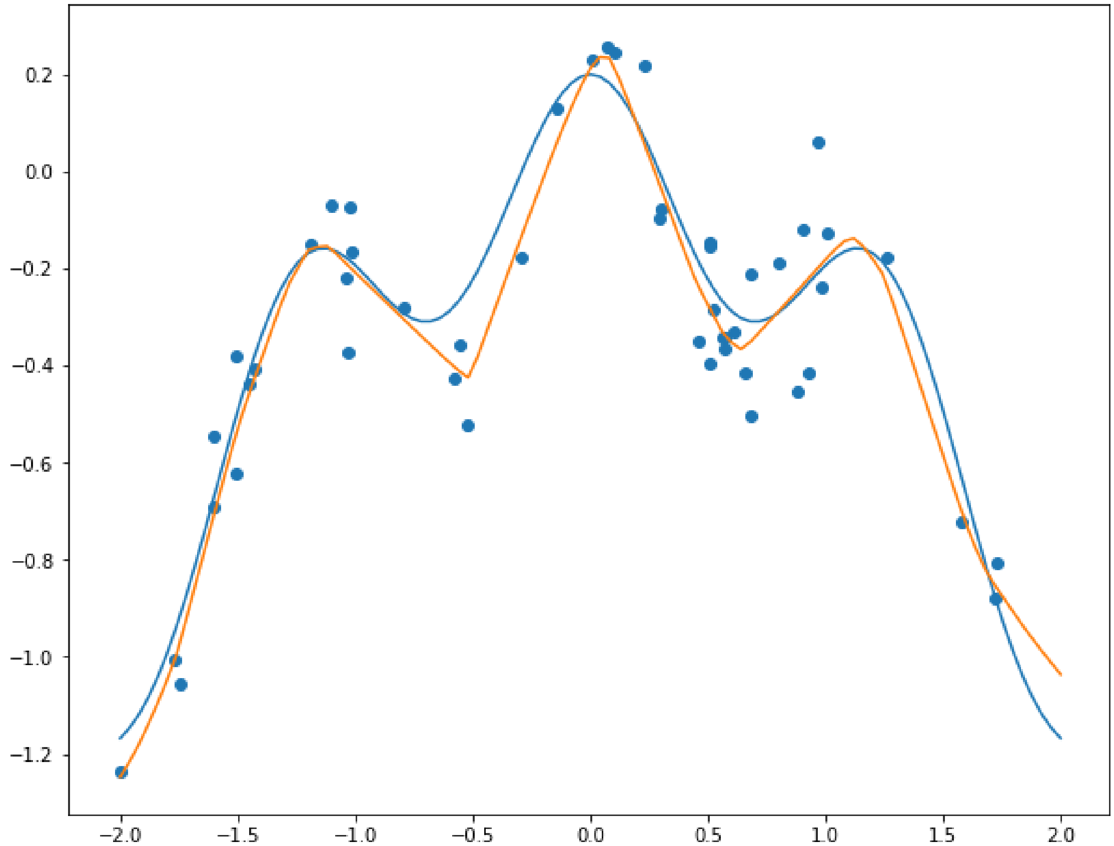
[28]:
```
print(regularized_model.evaluate(train_x, train_y))
print(regularized_model.evaluate(val_x, val_y))
```

```
50/50 [==============================] - 0s 156us/step
[0.10061833500862122, 0.03954936936497688]
1000/1000 [==============================] - 0s 67us/step
[0.11445685869455338, 0.05338789260387421]
```

0.0.12 Penalty 0.001 on all layers

Code same as above, but with penalty 0.001.

[24]:
```
plot_layers(regularized_model, hidden_legend=False, include_hidden = False)
```



[25]:
```
print(regularized_model.evaluate(train_x, train_y))
print(regularized_model.evaluate(val_x, val_y))
```

```
50/50 [==============================] - 0s 164us/step
[0.0928116300702095, 0.009383224323391915]
1000/1000 [==============================] - 0s 54us/step
[0.10905261212587357, 0.02562420552968979]
```

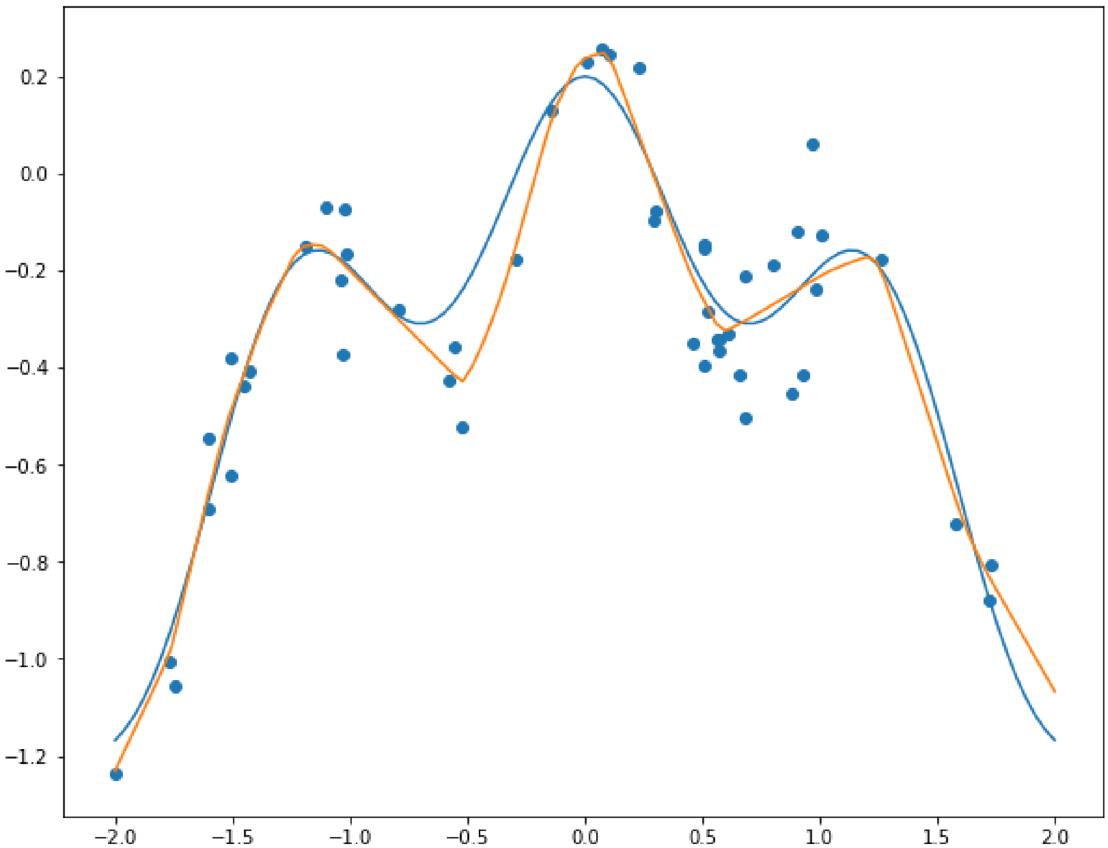
0.1 L1 Regularization

We could also use a penalty based on the absolute values of the weights:

$$J(b,w) = -\ell(b,w) + \sum_{l=1}^{L} \lambda_l \left\{ \sum_{i,j} \left| w_{i,j}^{[l]} \right| \right\}$$

- Code identical to above except for using `regularizers.l1(penalty)`

[32]:
```
plot_layers(regularized_model, hidden_legend=False, include_hidden = False)
```

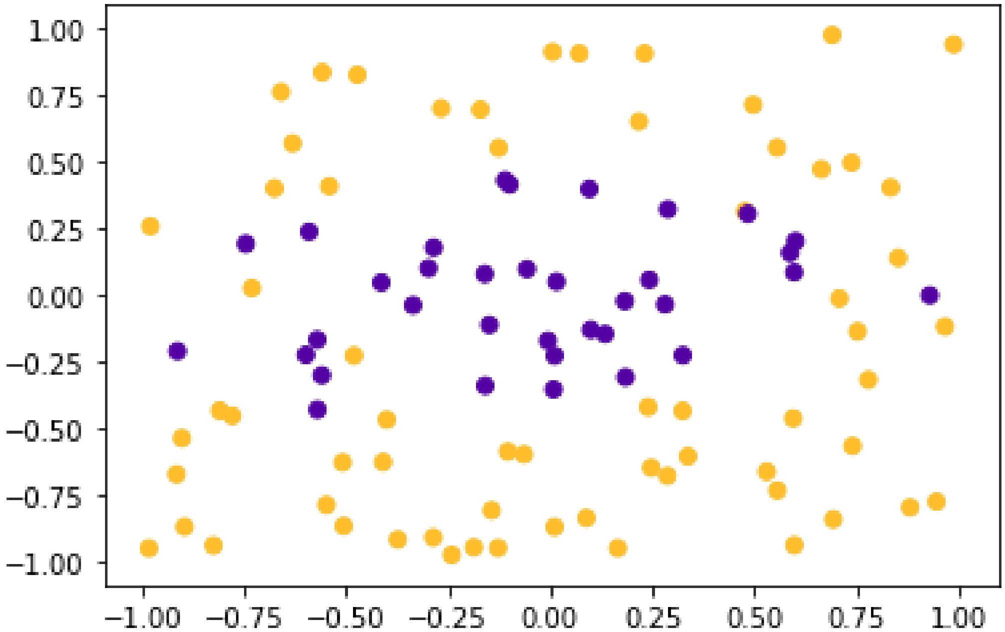


[33]:
```
print(regularized_model.evaluate(train_x, train_y))
print(regularized_model.evaluate(val_x, val_y))
```

```
50/50 [==============================] - 0s 137us/step
[0.3927127182483673, 0.009438994713127613]
1000/1000 [==============================] - 0s 61us/step
[0.40855745816230776, 0.02528374271094799]
```

0.2 Classification Example

0.2.1 Data Generation



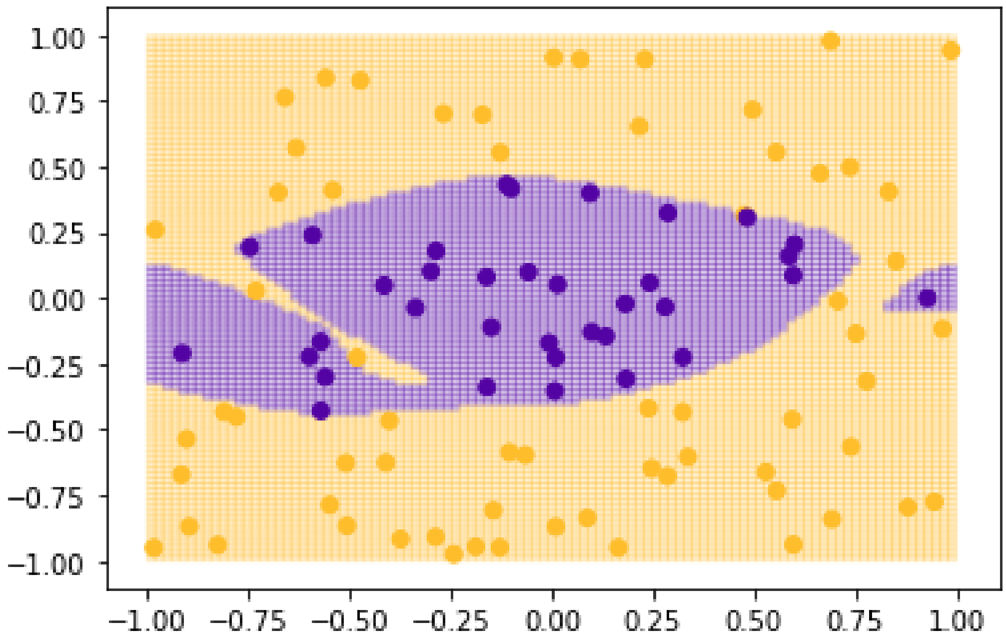
0.2.2 Unpenalized Model

5 hidden layers of 1024 units each.

[59]:
```
unregularized_model.summary()
plot_decision_boundary(unregularized_model, None, (-1, 1), (-1, 1), 101,␣
 ↪train_x, train_y)
print("Validation set evaluation:")
unregularized_model.evaluate(val_x, val_y)
```

```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_91 (Dense)             (None, 1024)              3072
_________________________________________________________________
dense_92 (Dense)             (None, 1024)              1049600
_________________________________________________________________
dense_93 (Dense)             (None, 1024)              1049600
_________________________________________________________________
dense_94 (Dense)             (None, 1024)              1049600
```

```
_________________________________________________________________
dense_95 (Dense)             (None, 1024)              1049600
_________________________________________________________________
dense_96 (Dense)             (None, 1)                 1025
=================================================================
Total params: 4,202,497
Trainable params: 4,202,497
Non-trainable params: 0
_________________________________________________________________
```




```
Validation set evaluation:
10000/10000 [==============================] - 1s 52us/step
```

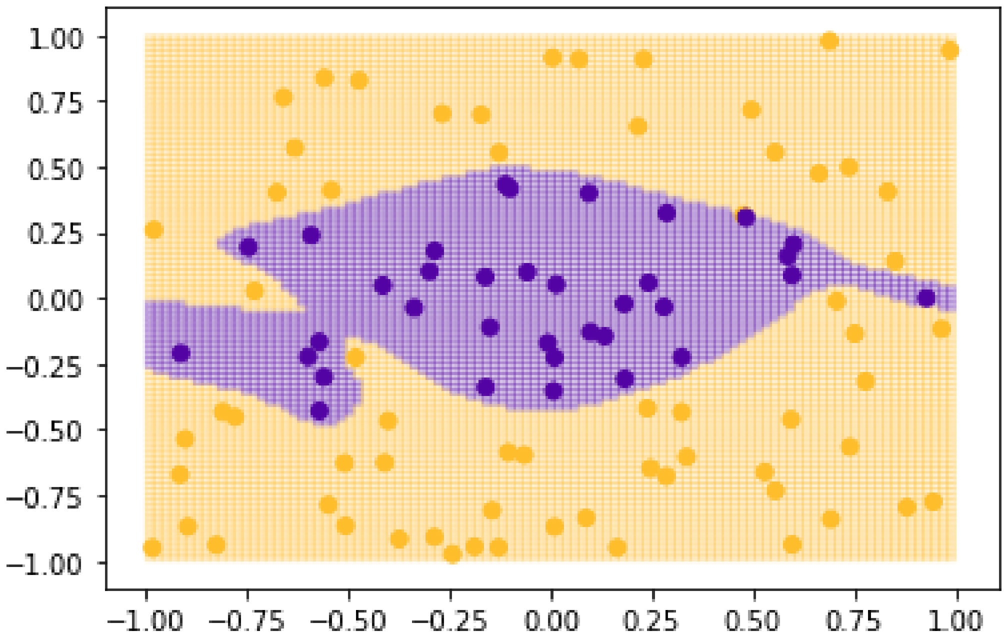
[59]: [1.2083770444512367, 0.88]

0.2.3 Penalty 0.001

Same as before, we just add a kernel_regularizer to each layer whose weights we want to regularize.

[62]:
```
regularized_model.summary()
plot_decision_boundary(regularized_model, None, (-1, 1), (-1, 1), 101, train_x,␣
 ↪train_y)
print("Validation set evaluation:")
regularized_model.evaluate(val_x, val_y)
```

```
Model: "sequential_22"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_109 (Dense)            (None, 1024)              3072
_________________________________________________________________
dense_110 (Dense)            (None, 1024)              1049600
_________________________________________________________________
dense_111 (Dense)            (None, 1024)              1049600
_________________________________________________________________
dense_112 (Dense)            (None, 1024)              1049600
_________________________________________________________________
dense_113 (Dense)            (None, 1024)              1049600
_________________________________________________________________
dense_114 (Dense)            (None, 1)                 1025
=================================================================
Total params: 4,202,497
Trainable params: 4,202,497
Non-trainable params: 0
_________________________________________________________________
```




```
Validation set evaluation:
10000/10000 [==============================] - 1s 63us/step
```

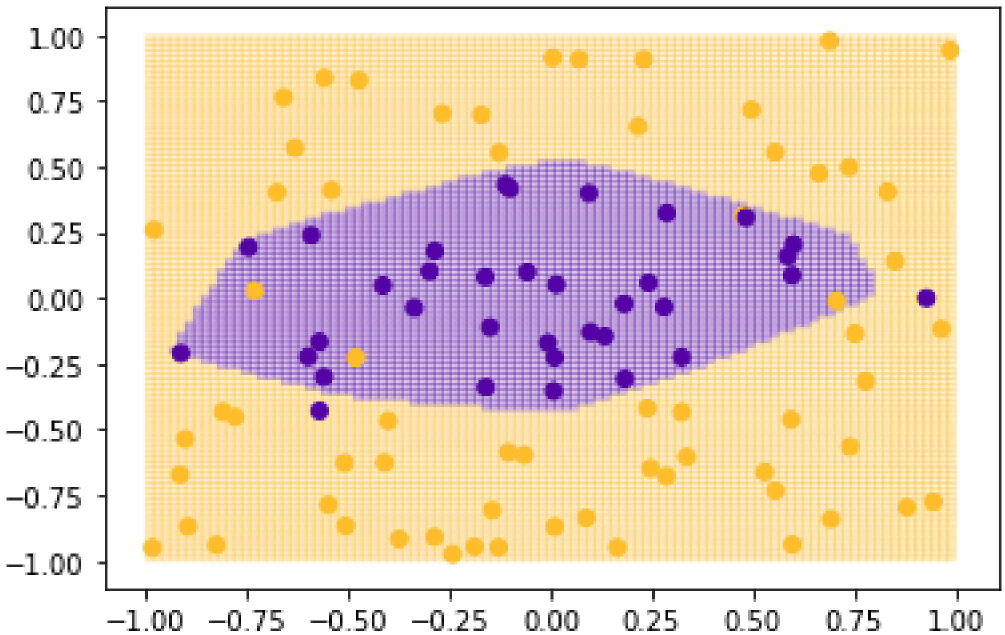
[62]: [0.5694817571163178, 0.8801]

0.2.4 Penalty 0.01

Code suppressed.

[64]:
```
regularized_model_01.summary()
plot_decision_boundary(regularized_model_01, None, (-1, 1), (-1, 1), 101,␣
 ↪train_x, train_y)
print("Validation set evaluation:")
regularized_model_01.evaluate(val_x, val_y)
```

```
Model: "sequential_23"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_115 (Dense)            (None, 1024)              3072
_________________________________________________________________
dense_116 (Dense)            (None, 1024)              1049600
_________________________________________________________________
dense_117 (Dense)            (None, 1024)              1049600
_________________________________________________________________
dense_118 (Dense)            (None, 1024)              1049600
_________________________________________________________________
dense_119 (Dense)            (None, 1024)              1049600
_________________________________________________________________
dense_120 (Dense)            (None, 1)                 1025
=================================================================
Total params: 4,202,497
Trainable params: 4,202,497
Non-trainable params: 0
_________________________________________________________________
```




```
Validation set evaluation:
10000/10000 [==============================] - 1s 62us/step
```

[64]: [0.4293009859085083, 0.9029]