KerasNMRImproved-PaulGiesting

November 4, 2020

1 Keras - simulated NMR data

1.0.1 Paul Giesting

Summary of my work this week

1. I explored the baseline PyTorch implementation and calculated a version of the custom error metric we'll be graded on, about 0.17 per row of input data.
2. I tried to implement a Keras version of the PyTorch baseline model, but the PyTorch syntax is completely unfamiliar to me and I cannot be sure I'm really getting all that close. The error by the same metric was 0.24.
3. I did some EDA on the dataset and remarked how unrelated the xi, p, and d properties seemed to be to the values of the M curve at the 180 and echo points.
4. I used the autosklearn library and manual exploration to see if non-neural-network solutions could get any traction. The answer was "not really, not in the time alloted" (and I alloted a lot of Tuesday, all of Wednesday, and Thursday until 5 pm, more than made sense). At one point I encountered a k-nearest-neighbors regression via autosklearn that could reproduce some of the alpha behavior but even that was remarkably easy to lose. I cannot reproduce it manually. I noted that random forest and extra tree models at close to baseline configurations would horrendously overtrain (stuff like 0.08 training error and 0.52 test error).
5. I deployed autokeras. I tried implementing the custom error function as the loss and/or metric function in autokeras and while the model would train, it would then refuse to export the structure or compute predictions. I went back to the default loss and metric settings after training with the custom metric, trained again hoping that some of the learned weights would influence the outcome, but I doubt they did. The best I was able to find was a simple-ish model that got down to 0.22 error.
6. Today I produced the contents of this notebook. I saved the results of model 5, the best model in terms of performance, only insofar as they are seen in the output below; I redid model 5 as model 13, with a different draw of train / test data and different results on the stochastic descent, etc. Model 13 was used to produce my output submission; the cell where the data was exported is near the top, before model construction begins.

Description of another team's approach "Our solution, called VisEcho, was a fully convolutional network, with depthwise separable convolutions replacing the need for fully connected layers. If anyone is interested, they can check it out at my GitHub account here: https://github.com/stephenhgregory/VisEcho" [actually there is no information there]

[1]:
```
import keras.backend as K
import keras
from keras.models import Model
from keras.layers import Dense, Activation, Input, Concatenate
from keras.utils import np_utils
from keras.utils.data_utils import get_file
from keras.utils.vis_utils import model_to_dot, plot_model
from keras.preprocessing import sequence
from keras.optimizers import SGD, RMSprop
import kerastuner
```

[2]:
```
from keras.losses import MeanSquaredError
import sklearn.metrics
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
```

[3]:
```
from tensorflow import math as tfmath
```

[4]:
```
# mean version of gross error metric

def weight_mse(truth,predict):
    erf = 0.0
    weight = [0.2, 20, 2, 3]
    for col in range(4):
        erf += sklearn.metrics.mean_squared_error(truth[:,col],predict[:,col]) /
 ↪ (weight[col]**2)
    return erf
```

[5]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

mat_file = "mat_info.txt"
M_file_r = "echos_r.txt" # real part of echos
M_file_i = "echos_i.txt" # imaginary part of echos
print("Loading into numpy arrays...")
# settings of each simulated material:
# format:  |      |      |   p   |   d   |
mat_info = np.loadtxt(mat_file, comments="#", delimiter='\t', unpack=False);
# M(t) curve for each simulation:
M_r = np.loadtxt(M_file_r, comments="#", delimiter='\t', unpack=False);
M_i = np.loadtxt(M_file_i, comments="#", delimiter='\t', unpack=False);
M = M_r + 1j*M_i;
print("Done with numpy loads")
```

```
Loading into numpy arrays…
Done with numpy loads
```

[6]:
```
# partition data into a training and testing set using a random partition
# number of M(t) curves
N_data = np.shape(M)[0]

# pick a 90%/10% split for training/testing
test_frac = 0.10                                # fraction of data to save for␣
 ↪testing data
div_idx = int(np.floor((1.0-test_frac)*N_data)) # integer number of curves to␣
 ↪use for training
order_seed = np.random.permutation(N_data)      # random ordering for all curves

train_idx = order_seed[0:div_idx]               # first 90% of random order
test_idx = order_seed[div_idx:N_data]           # last 10% of random order
```

[7]:
```
# truncate time points from 210 to 410 in example
# centered roughly at the echo
# let's use whole experiment now
time_keep = range(0,157*3);

# concatenate the real and imaginary parts together, to make a real-vector of␣
 ↪double the length
M_train = M[train_idx[:,None],time_keep]                        # time truncation␣
 ↪of input
mat_train = mat_info[train_idx,:]                               # get the output
M_train = np.hstack( (np.real(M_train), np.imag(M_train)) ) # real part, then␣
 ↪imaginary part
# is this equivalent to
# M_train = np.hstack(M_r[train_idx],M_i[train_idx])
# ?

# same as above, but for test
M_test = M[test_idx[:,None],time_keep]
mat_test = mat_info[test_idx,:];
M_test = np.hstack( (np.real(M_test), np.imag(M_test)) )
```

[8]:
```
N = np.shape(M_train[0])[0]
```

[9]:
```
# scale the material properties consistently
# so that mean squared error is the appropriate metric
scaler = MinMaxScaler()
scaler.fit(mat_info)
```

[9]:
```
MinMaxScaler()
```

[10]:
```
# data for submission of final model
sub_file_r = "submit_echos_r.txt"
sub_file_i = "submit_echos_i.txt"
```

[86]:
```
# for competition submission only
import requests

r = requests.get("https://docs.google.com/uc?
↪export=download&id=14-oz_3OGsTFziJI1FUgOEenMcdQDf2F_",allow_redirects=True)
open(sub_file_r, "wb").write(r.content)
r = requests.get("https://docs.google.com/uc?
↪export=download&id=1Add2V9cYOBbOCvr1Dj-g4yMlx3LdJY8i",allow_redirects=True)
open(sub_file_i, "wb").write(r.content)

print("Done with file downloads")
```

```
Done with file downloads
```

[11]:
```
# for competition submission only
# M(t) curve for each simulation:
M_r_sub = np.loadtxt(sub_file_r, comments="#", delimiter='\t', unpack=False);
M_i_sub = np.loadtxt(sub_file_i, comments="#", delimiter='\t', unpack=False);
M_cmplx_sub = M_r_sub + 1j*M_i_sub;
M_sub = np.hstack( (np.real(M_cmplx_sub), np.imag(M_cmplx_sub)) ) # real part,␣
↪then imaginary part
print("Done with numpy loads")
```

```
Done with numpy loads
```

[16]:
```
# for competition submission only
mat_sub = scaler.inverse_transform(model13.predict(M_sub))
print(mat_sub[:5,:])
sub_file = "submitted_mat_info.txt"
np.savetxt(sub_file,mat_sub,fmt='%10.8f',delimiter='\t',header='Paul Giesting␣
↪NMR submission')
```

```
[[1.9445598e-01 9.7248974e+00 2.4649928e+00 4.1607680e+00]
 [2.6868028e-03 1.0338113e+01 2.9087524e+00 4.9438171e+00]
 [5.7800967e-02 1.0118806e+01 3.2239895e+00 4.5901608e+00]
 [5.3764884e-03 9.9226151e+00 3.1163795e+00 3.7858155e+00]
 [1.3918748e-01 1.0627731e+01 3.5263150e+00 4.6521125e+00]]
```

[17]:
```
# for competition submission only
check = np.loadtxt(sub_file, comments="#", delimiter='\t', unpack=False)
len(check)
```

[17]: 500

1.1 Preface

This is the description of the best autokeras model, exported to keras. I have no idea why autokeras inserted a category encoding layer. Since the input data are restricted to -0.5 to 0.5, I don't think a normalization layer is likely to be needed, but it's an option that could be tried later on. If I do that, it looks like I should `adapt` the layer to the training data, or possibly the whole training+validation set, before fitting the whole model.

Ok, step 1: see if I can reproduce the autokeras model without the encoding layer.

1.2 Model 1

[ ]:
```
model_description = \
'''Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 900)]             0
_________________________________________________________________
multi_category_encoding (Mul (None, 900)               0
_________________________________________________________________
dense (Dense)                (None, 32)                28832
_________________________________________________________________
re_lu (ReLU)                 (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                1056
_________________________________________________________________
re_lu_1 (ReLU)               (None, 32)                0
_________________________________________________________________
regression_head_1 (Dense)    (None, 4)                 132
=================================================================
Total params: 30,020
Trainable params: 30,020
Non-trainable params: 0
_________________________________________________________________'''
```

[14]:
```
input_layer = Input(shape=(N,))
layer1 = Dense(32,activation='relu')(input_layer)
layer2 = Dense(32,activation='relu')(layer1)
layer3 = Dense(4)(layer2)

model = Model(inputs=input_layer, outputs=layer3)
model.summary()
```

```
Model: "functional_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 900)]             0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                28832
_________________________________________________________________
```

```
dense_4 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_5 (Dense)              (None, 4)                 132
=================================================================
Total params: 30,020
Trainable params: 30,020
Non-trainable params: 0
_________________________________________________________________
```

[34]:

```
# 'huber_loss' is what I was using to construct the keras analog to PyTorch
# it is not autokeras' default, which was val_loss for one or both
# of the loss function and metric (mse was reported too)
# can't even find that here? yet it should be the default?
# vanilla keras won't take the custom loss metric either
opt = keras.optimizers.SGD(learning_rate=0.005,momentum=0.9)
model.
 ↪compile(loss='mean_squared_error',optimizer=opt,metrics=['mean_squared_error'])
```

[35]:

```
# number of epochs and batches
# 20 epochs was still dropping the metrics on autokeras
# can't tell from PyTorch OR autokeras what their what the batch size was
# batch size 10 worked a little better than 20, which was somewhat better
# than 50 in my earlier keras exploration
n_epochs = 40
n_batch = 10
# transform the target so that mse is equivalent to the appropriate metric
print('Starting Training')
model.fit(M_train,scaler.
 ↪transform(mat_train),epochs=n_epochs,batch_size=n_batch)
print('Finished Training')
```

```
Starting Training
Epoch 1/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0506 -
mean_squared_error: 0.0506
Epoch 2/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0501 -
mean_squared_error: 0.0501
Epoch 3/40
900/900 [==============================] - 1s 1ms/step - loss: 0.0495 -
mean_squared_error: 0.0495
Epoch 4/40
900/900 [==============================] - 1s 1ms/step - loss: 0.0491 -
mean_squared_error: 0.0491
Epoch 5/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0484 -
mean_squared_error: 0.0484
Epoch 6/40
```

```
900/900 [==============================] - 1s 1ms/step - loss: 0.0480 -
mean_squared_error: 0.0480
Epoch 7/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0474 -
mean_squared_error: 0.0474
Epoch 8/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0467 -
mean_squared_error: 0.0467
Epoch 9/40
900/900 [==============================] - 1s 1ms/step - loss: 0.0462 -
mean_squared_error: 0.0462
Epoch 10/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0455 -
mean_squared_error: 0.0455
Epoch 11/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0450 -
mean_squared_error: 0.0450
Epoch 12/40
900/900 [==============================] - 1s 1ms/step - loss: 0.0443 -
mean_squared_error: 0.0443
Epoch 13/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0437 -
mean_squared_error: 0.0437
Epoch 14/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0430 -
mean_squared_error: 0.0430
Epoch 15/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0423 -
mean_squared_error: 0.0423
Epoch 16/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0420 -
mean_squared_error: 0.0420
Epoch 17/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0416 -
mean_squared_error: 0.0416
Epoch 18/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0410 -
mean_squared_error: 0.0410
Epoch 19/40
900/900 [==============================] - 1s 1ms/step - loss: 0.0406 -
mean_squared_error: 0.0406
Epoch 20/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0404 -
mean_squared_error: 0.0404
Epoch 21/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0400 -
mean_squared_error: 0.0400
Epoch 22/40
```

```
900/900 [==============================] - 2s 2ms/step - loss: 0.0396 -
mean_squared_error: 0.0396
Epoch 23/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0393 -
mean_squared_error: 0.0393
Epoch 24/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0393 -
mean_squared_error: 0.0393
Epoch 25/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0390 -
mean_squared_error: 0.0390
Epoch 26/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0386 -
mean_squared_error: 0.0386
Epoch 27/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0384 -
mean_squared_error: 0.0384
Epoch 28/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0384 -
mean_squared_error: 0.0384
Epoch 29/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0380 -
mean_squared_error: 0.0380
Epoch 30/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0377 -
mean_squared_error: 0.0377
Epoch 31/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0382 -
mean_squared_error: 0.0382
Epoch 32/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0376 -
mean_squared_error: 0.0376
Epoch 33/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0377 -
mean_squared_error: 0.0377
Epoch 34/40
900/900 [==============================] - 1s 1ms/step - loss: 0.0376 -
mean_squared_error: 0.0376
Epoch 35/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0372 -
mean_squared_error: 0.0372
Epoch 36/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0370 -
mean_squared_error: 0.0370
Epoch 37/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0368 -
mean_squared_error: 0.0368
Epoch 38/40
```

```
900/900 [==============================] - 1s 2ms/step - loss: 0.0368 -
mean_squared_error: 0.0368
Epoch 39/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0370 -
mean_squared_error: 0.0370
Epoch 40/40
900/900 [==============================] - 1s 2ms/step - loss: 0.0366 -
mean_squared_error: 0.0366
Finished Training
```

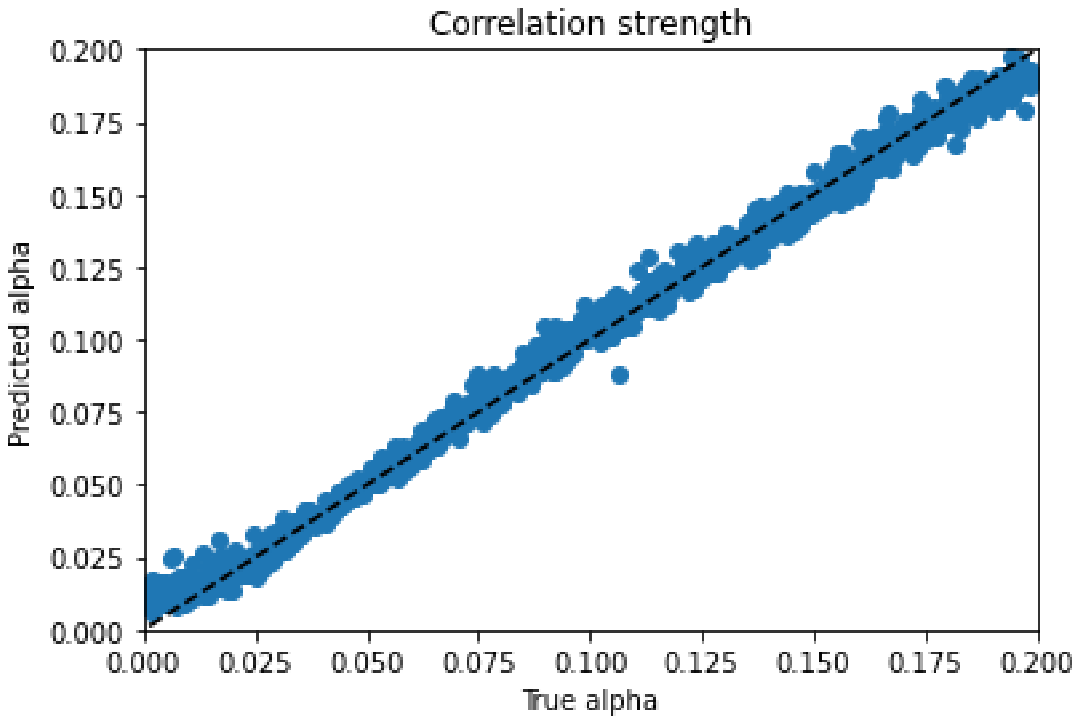
[36]:
```python
mat_train_predict = scaler.inverse_transform(model.
 ↪predict(M_train,batch_size=n_batch))
mat_predict = scaler.inverse_transform(model.predict(M_test,batch_size=n_batch))
print('Training score for model ',weight_mse(mat_train,mat_train_predict))
print('Test score for model ',weight_mse(mat_test,mat_predict))
```

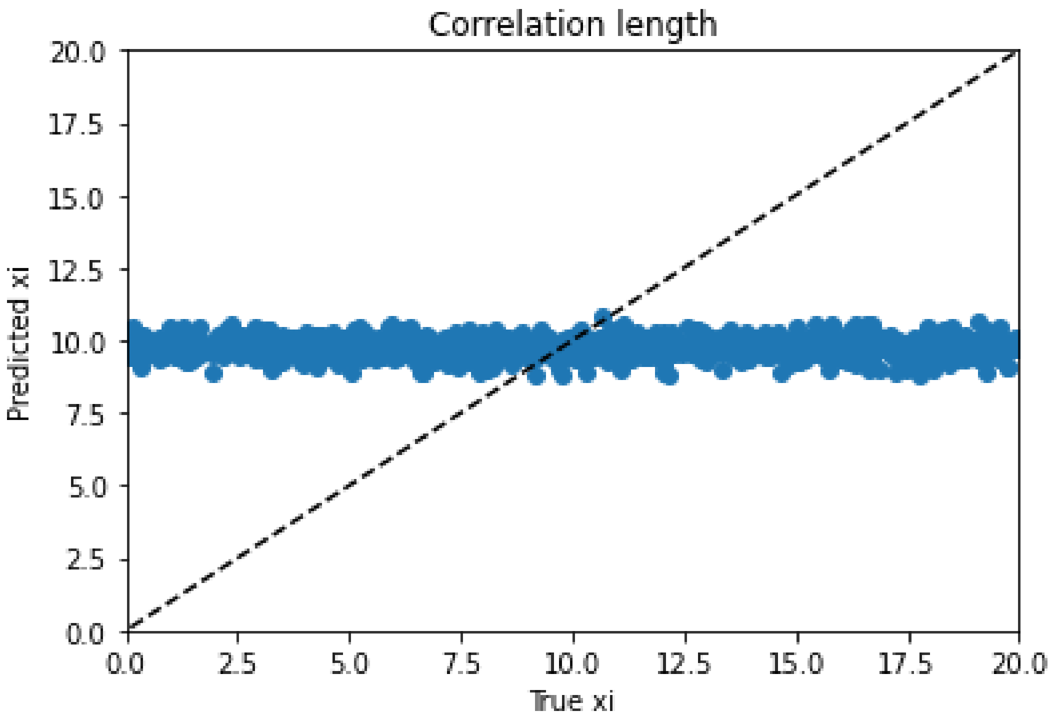
```
Training score for model  0.14181771689758704
Test score for model  0.13789912161340212
```

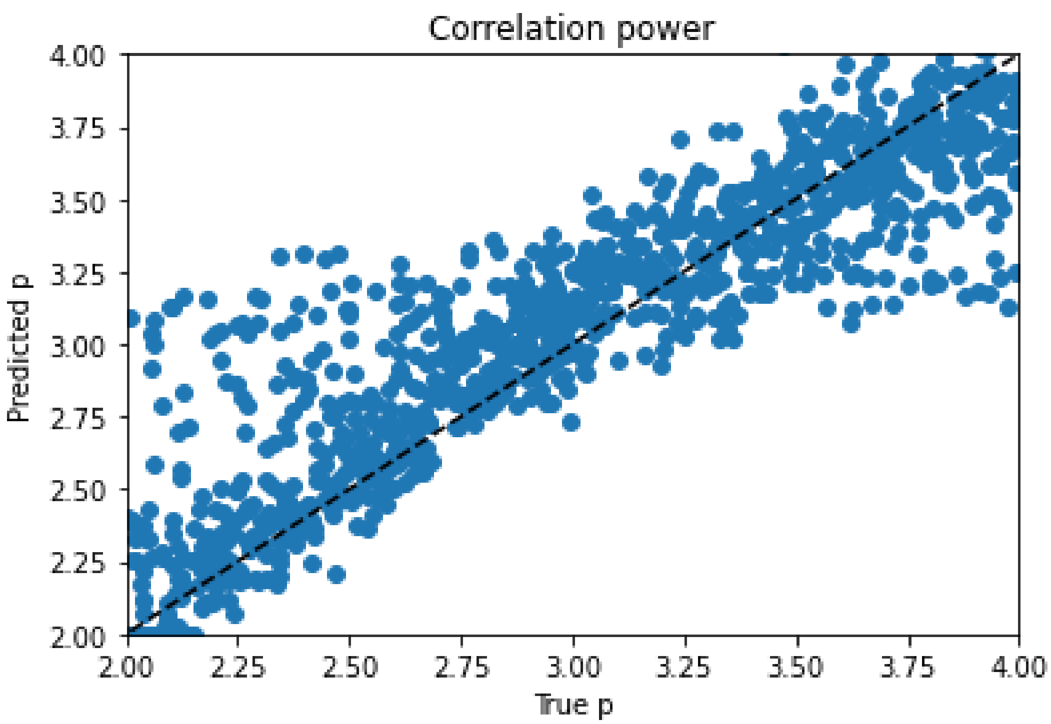
That seems to have done something. Egads. Let's plot that.

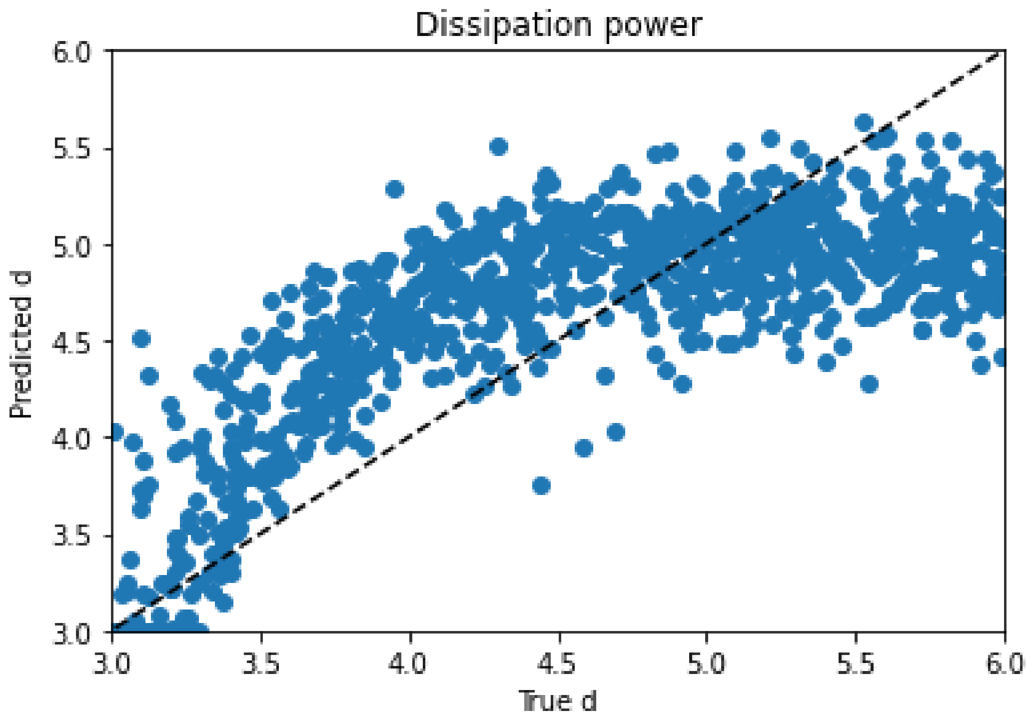
[37]:
```python
plt.scatter(mat_test[:,0],mat_predict[:,0]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True alpha");
plt.ylabel("Predicted alpha");
plt.axis([0, .2, 0, .2])
plt.title("Correlation strength")

plt.figure()
plt.scatter(mat_test[:,1],mat_predict[:,1]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True xi");
plt.ylabel("Predicted xi");
plt.axis([0, 20, 0, 20])
plt.title("Correlation length")

plt.figure()
plt.scatter(mat_test[:,2],mat_predict[:,2]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True p");
plt.ylabel("Predicted p");
plt.axis([2, 4, 2, 4])
plt.title("Correlation power")

plt.figure()
plt.scatter(mat_test[:,3],mat_predict[:,3]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True d");
```

```
plt.ylabel("Predicted d");
plt.axis([3, 6, 3, 6]);
plt.title("Dissipation power");
```




Correlation length
Predicted xi
True xi
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Correlation power
Predicted p
True p
2.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75
4.00



It's late Friday morning and my model is ALMOST to the point of the baseline. Whatever. We barely grazed deep learning in my bootcamp and I didn't engage with it seriously prior to this week.

(I then went back and increased learning rate from 0.001 to 0.005 and ran for 40 epochs. Now it's an actual improvement!)

Then let's try normalizing the targets instead of compressing them to [0,1].

1.3 Model 2

[30]:
```
input2_layer = Input(shape=(N,))
layer21 = Dense(32,activation='relu')(input2_layer)
layer22 = Dense(32,activation='relu')(layer21)
layer23 = Dense(4)(layer22)

model2 = Model(inputs=input2_layer, outputs=layer23)
model2.summary()
```

```
Model: "functional_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 900)]             0
_________________________________________________________________
```

```
dense_12 (Dense)             (None, 32)                28832
_________________________________________________________________
dense_13 (Dense)             (None, 32)                1056
_________________________________________________________________
dense_14 (Dense)             (None, 4)                 132
=================================================================
Total params: 30,020
Trainable params: 30,020
Non-trainable params: 0
_________________________________________________________________
```

[31]:
```python
# scale the material properties consistently
normer = StandardScaler()
normer.fit(mat_info)
```

[31]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

[38]:
```python
opt = keras.optimizers.SGD(learning_rate=0.005,momentum=0.9)
model2.
 ↪compile(loss='mean_squared_error',optimizer=opt,metrics=['mean_squared_error'])
```

[39]:
```python
# number of epochs and batches
n_epochs = 40
n_batch = 10
# transform the target; mse is not quite the appropriate metric now but still␣
 ↪close
print('Starting Training')
model2.fit(M_train,normer.
 ↪transform(mat_train),epochs=n_epochs,batch_size=n_batch)
print('Finished Training')
```

```
Starting Training
Epoch 1/40
900/900 [==============================] - 1s 2ms/step - loss: 0.5640 -
mean_squared_error: 0.5640
Epoch 2/40
900/900 [==============================] - 1s 1ms/step - loss: 0.5415 -
mean_squared_error: 0.5415
Epoch 3/40
900/900 [==============================] - 1s 1ms/step - loss: 0.5211 -
mean_squared_error: 0.5211
Epoch 4/40
900/900 [==============================] - 1s 2ms/step - loss: 0.5076 -
mean_squared_error: 0.5076
Epoch 5/40
900/900 [==============================] - 1s 2ms/step - loss: 0.5008 -
mean_squared_error: 0.5008
Epoch 6/40
```

```
900/900 [==============================] - 1s 1ms/step - loss: 0.4965 -
mean_squared_error: 0.4965
Epoch 7/40
900/900 [==============================] - 1s 2ms/step - loss: 0.4850 -
mean_squared_error: 0.4850
Epoch 8/40
900/900 [==============================] - 1s 2ms/step - loss: 0.4772 -
mean_squared_error: 0.4772
Epoch 9/40
900/900 [==============================] - 1s 2ms/step - loss: 0.4732 -
mean_squared_error: 0.4732
Epoch 10/40
900/900 [==============================] - 2s 2ms/step - loss: 0.4648 -
mean_squared_error: 0.4648
Epoch 11/40
900/900 [==============================] - 1s 2ms/step - loss: 0.4639 -
mean_squared_error: 0.4639
Epoch 12/40
900/900 [==============================] - 1s 2ms/step - loss: 0.4659 -
mean_squared_error: 0.4659
Epoch 13/40
900/900 [==============================] - 1s 1ms/step - loss: 0.4568 -
mean_squared_error: 0.4568
Epoch 14/40
900/900 [==============================] - 2s 2ms/step - loss: 0.4487 -
mean_squared_error: 0.4487
Epoch 15/40
900/900 [==============================] - 1s 2ms/step - loss: 0.4532 -
mean_squared_error: 0.4532
Epoch 16/40
900/900 [==============================] - 1s 1ms/step - loss: 0.4483 -
mean_squared_error: 0.4483
Epoch 17/40
900/900 [==============================] - 1s 2ms/step - loss: 0.4454 -
mean_squared_error: 0.4454
Epoch 18/40
900/900 [==============================] - 2s 2ms/step - loss: 0.4451 -
mean_squared_error: 0.4451
Epoch 19/40
900/900 [==============================] - 2s 2ms/step - loss: 0.4418 -
mean_squared_error: 0.4418
Epoch 20/40
900/900 [==============================] - 2s 2ms/step - loss: 0.4422 -
mean_squared_error: 0.4422
Epoch 21/40
900/900 [==============================] - 2s 2ms/step - loss: 0.4392 -
mean_squared_error: 0.4392
Epoch 22/40
```

```
900/900 [==============================] - 2s 2ms/step - loss: 0.4406 -
mean_squared_error: 0.4406
Epoch 23/40
900/900 [==============================] - 1s 2ms/step - loss: 0.4386 -
mean_squared_error: 0.4386
Epoch 24/40
900/900 [==============================] - 1s 2ms/step - loss: 0.4354 -
mean_squared_error: 0.4354
Epoch 25/40
900/900 [==============================] - 1s 2ms/step - loss: 0.4358 -
mean_squared_error: 0.4358
Epoch 26/40
900/900 [==============================] - 1s 2ms/step - loss: 0.4299 -
mean_squared_error: 0.4299
Epoch 27/40
900/900 [==============================] - 1s 2ms/step - loss: 0.4276 -
mean_squared_error: 0.4276
Epoch 28/40
900/900 [==============================] - 1s 2ms/step - loss: 0.4245 -
mean_squared_error: 0.4245
Epoch 29/40
900/900 [==============================] - 1s 2ms/step - loss: 0.4217 -
mean_squared_error: 0.4217
Epoch 30/40
900/900 [==============================] - 1s 2ms/step - loss: 0.4187 -
mean_squared_error: 0.4187
Epoch 31/40
900/900 [==============================] - 1s 1ms/step - loss: 0.4236 -
mean_squared_error: 0.4236
Epoch 32/40
900/900 [==============================] - 1s 2ms/step - loss: 0.4247 -
mean_squared_error: 0.4247
Epoch 33/40
900/900 [==============================] - 1s 2ms/step - loss: 0.4245 -
mean_squared_error: 0.4245
Epoch 34/40
900/900 [==============================] - 1s 2ms/step - loss: 0.4156 -
mean_squared_error: 0.4156
Epoch 35/40
900/900 [==============================] - 1s 1ms/step - loss: 0.4181 -
mean_squared_error: 0.4181
Epoch 36/40
900/900 [==============================] - 2s 2ms/step - loss: 0.4159 -
mean_squared_error: 0.4159
Epoch 37/40
900/900 [==============================] - 1s 2ms/step - loss: 0.4217 -
mean_squared_error: 0.4217
Epoch 38/40
```

```
900/900 [==============================] - 1s 1ms/step - loss: 0.4185 -
mean_squared_error: 0.4185
Epoch 39/40
900/900 [==============================] - 1s 1ms/step - loss: 0.4158 -
mean_squared_error: 0.4158
Epoch 40/40
900/900 [==============================] - 1s 2ms/step - loss: 0.4121 -
mean_squared_error: 0.4121
Finished Training
```

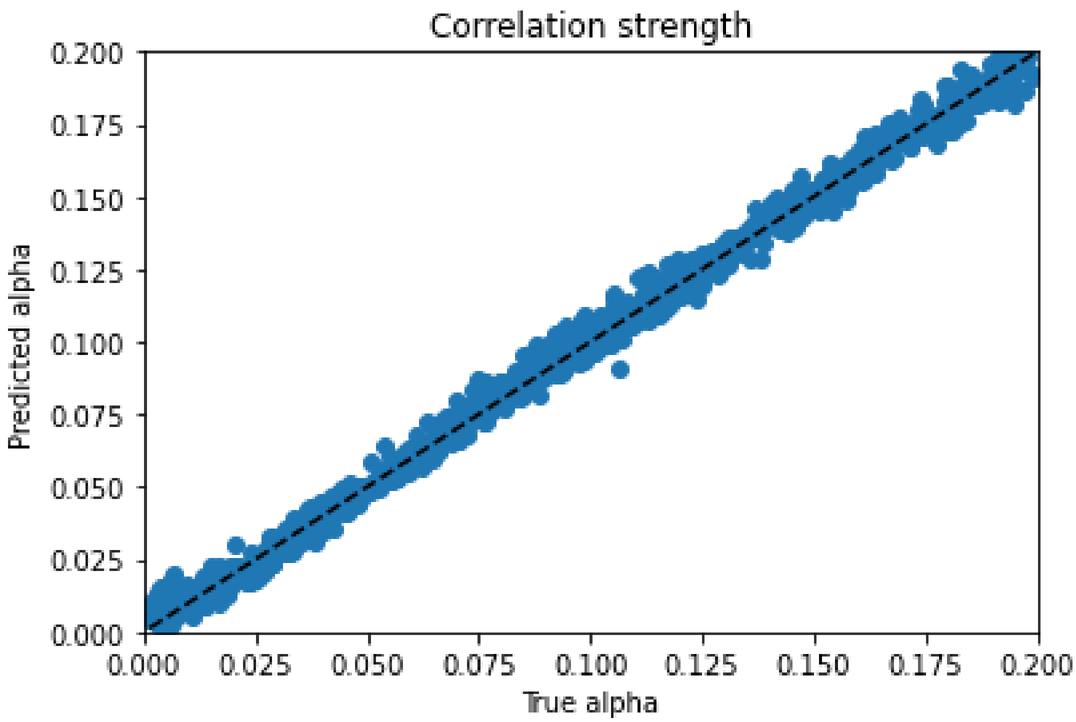
[40]:
```
mat2_train_predict = normer.inverse_transform(model2.
 ↪predict(M_train,batch_size=n_batch))
mat2_predict = normer.inverse_transform(model2.
 ↪predict(M_test,batch_size=n_batch))
print('Training score for model ',weight_mse(mat_train,mat2_train_predict))
print('Test score for model ',weight_mse(mat_test,mat2_predict))
```

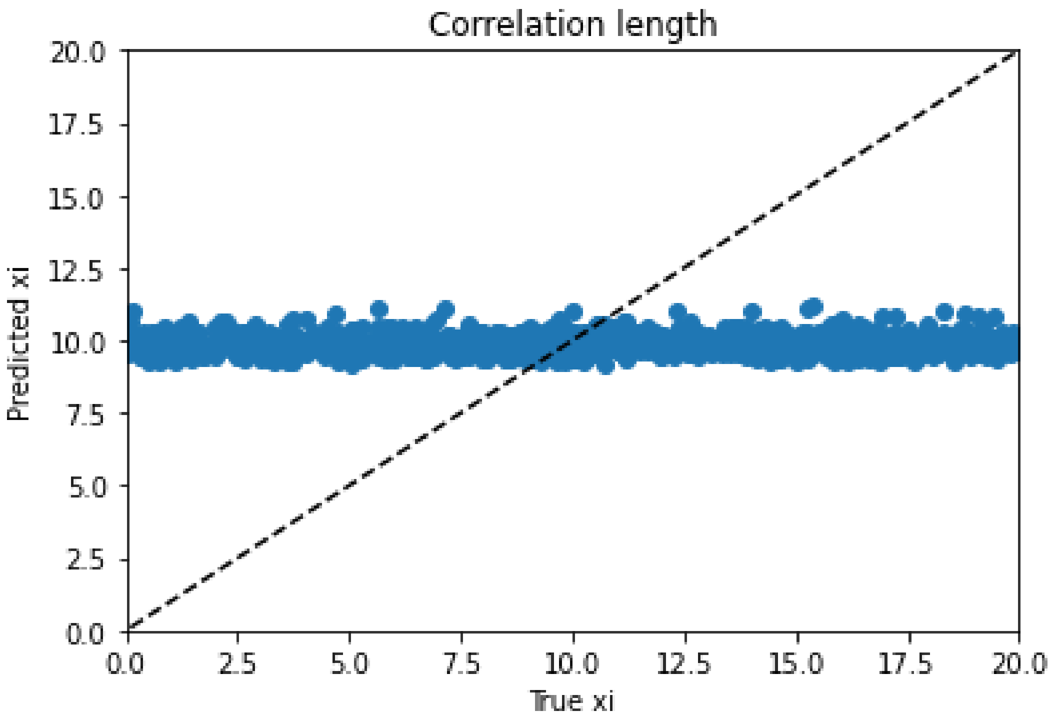
```
Training score for model  0.16133084616711393
Test score for model  0.15742218232789149
```

Let's plot that.

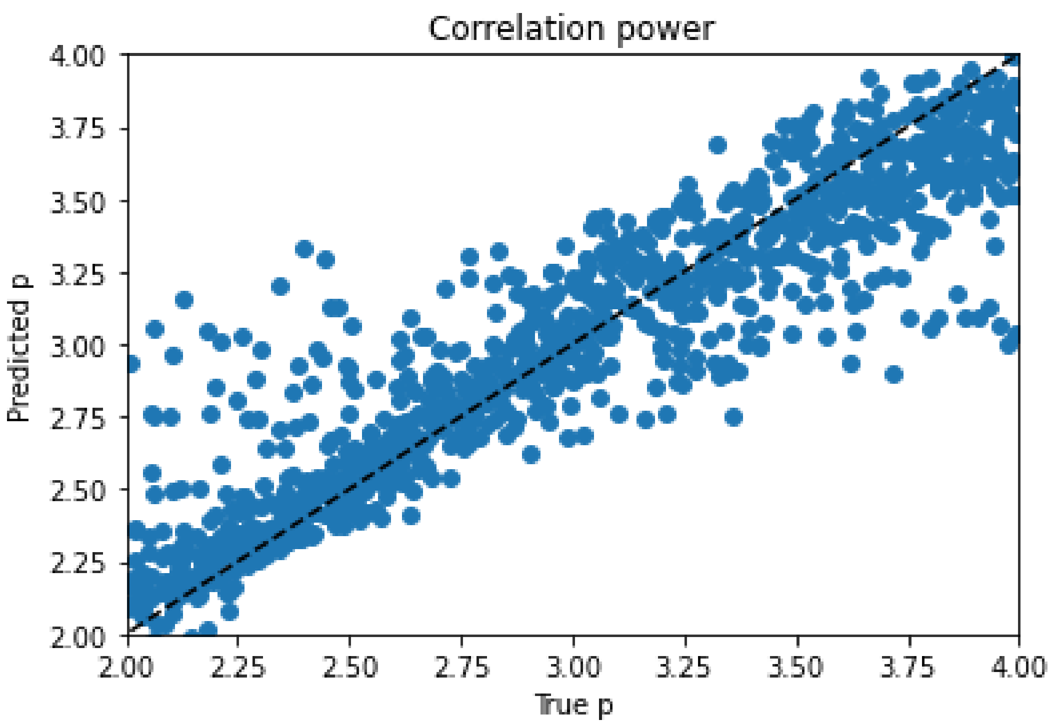
[41]:
```
plt.scatter(mat_test[:,0],mat2_predict[:,0]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True alpha");
plt.ylabel("Predicted alpha");
plt.axis([0, .2, 0, .2])
plt.title("Correlation strength")

plt.figure()
plt.scatter(mat_test[:,1],mat2_predict[:,1]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True xi");
plt.ylabel("Predicted xi");
plt.axis([0, 20, 0, 20])
plt.title("Correlation length")

plt.figure()
plt.scatter(mat_test[:,2],mat2_predict[:,2]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True p");
plt.ylabel("Predicted p");
plt.axis([2, 4, 2, 4])
plt.title("Correlation power")

plt.figure()
plt.scatter(mat_test[:,3],mat2_predict[:,3]);
plt.plot([-100, 100],[-100, 100],"--k")
```

```
plt.xlabel("True d");
plt.ylabel("Predicted d");
plt.axis([3, 6, 3, 6]);
plt.title("Dissipation power");
```

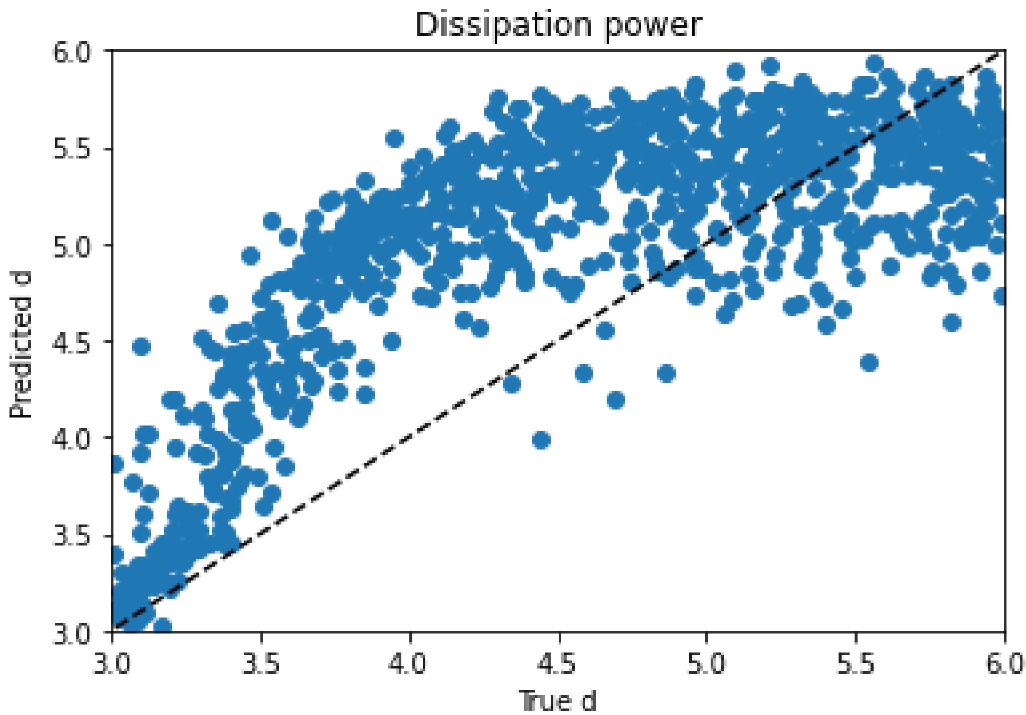




Correlation length
Predicted xi
True xi



Correlation power
Predicted p
True p



Fascinating and weird.

I could introduce some dropout to regularize features.

Another step would be to introduce an RNN layer as we saw at Metis.

Also worth checking using only Im(M) for input.

Obviously I could also tinker with the width of layers and number of layers.

Keras has the Tuner to fit structure and hyperparameters, too. Let's go back and follow that methodology.

1.4 Model 3

[98]:
```
def build_model(hp):
    inputs = Input(shape=(N,))
    x = Dense(
        units = hp.Int('units1',min_value=32,max_value=512,step=32),
        activation='relu'
    )(inputs)
    y = Dense(
        units = hp.Int('units2',min_value=16,max_value=128,step=16),
        activation='relu'
    )(x)
    outputs = Dense(4)(y)
```

```
    model = Model(inputs, outputs)
    opt = keras.optimizers.SGD(
        hp.Choice('learning_rate',
                  values=[0.01,0.005,0.001]),
        hp.Choice('momentum',
                  values=[0.67,0.9,0.95])
    )
    model.
↪compile(loss='mean_squared_error',optimizer=opt,metrics=['mean_squared_error'])
    return model
```

[96]:
```
tuner = kerastuner.tuners.Hyperband(
        build_model,
        objective='mean_squared_error',
        max_epochs=100,
        executions_per_trial=2,
        directory='keras_tune'
)
```

```
INFO:tensorflow:Reloading Oracle from existing project
keras_tune/untitled_project/oracle.json
INFO:tensorflow:Reloading Tuner from keras_tune/untitled_project/tuner0.json
```

[97]:
```
print('Starting Tuning')
tuner.search(M_train,scaler.transform(mat_train),
             validation_data=(M_test,mat_test))
print('Finished Tuning')
```

```
Trial 3 Complete [00h 00m 16s]
mean_squared_error: 0.052428992465138435

Best mean_squared_error So Far: 0.03339175321161747
Total elapsed time: 00h 00m 50s

Search: Running Trial #4

Hyperparameter    |Value             |Best Value So Far
units             |416               |512
learning_rate     |0.001             |0.01
momentum          |0.95              |0.95
tuner/epochs      |4                 |100
tuner/initial_e…|0                 |34
tuner/bracket     |3                 |4
tuner/round       |0                 |4

Epoch 1/4
282/282 [==============================] - 2s 8ms/step - loss: 0.0850 -
```

```
mean_squared_error: 0.0850 - val_loss: 37.1822 - val_mean_squared_error: 37.1822
Epoch 2/4
282/282 [==============================] - 2s 7ms/step - loss: 0.0625 -
mean_squared_error: 0.0625 - val_loss: 37.1349 - val_mean_squared_error: 37.1349
Epoch 3/4
282/282 [==============================] - 3s 10ms/step - loss: 0.0609 -
mean_squared_error: 0.0609 - val_loss: 37.2339 - val_mean_squared_error: 37.2339
Epoch 4/4
282/282 [==============================] - 3s 10ms/step - loss: 0.0597 -
mean_squared_error: 0.0597 - val_loss: 37.1982 - val_mean_squared_error: 37.1982
Epoch 1/4
282/282 [==============================] - 2s 9ms/step - loss: 0.0808 -
mean_squared_error: 0.0808 - val_loss: 37.1449 - val_mean_squared_error: 37.1449
Epoch 2/4
282/282 [==============================] - 2s 8ms/step - loss: 0.0626 -
mean_squared_error: 0.0626 - val_loss: 37.1644 - val_mean_squared_error: 37.1644
Epoch 3/4
176/282 [=================>…] - ETA: 0s - loss: 0.0616 -
mean_squared_error: 0.0616
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-97-064b5fdd9b26> in <module>
      1 print('Starting Tuning')
----> 2 tuner.search(M_train,scaler.transform(mat_train),

      3               validation_data=(M_test,mat_test))
      4 print('Finished Tuning')

~/.pyenv/versions/3.8.5/lib/python3.8/site-packages/kerastuner/engine/base_tuner.
↪py in search(self, *fit_args, **fit_kwargs)
    129
    130             self.on_trial_begin(trial)
--> 131             self.run_trial(trial, *fit_args, **fit_kwargs)
    132             self.on_trial_end(trial)
    133         self.on_search_end()

~/.pyenv/versions/3.8.5/lib/python3.8/site-packages/kerastuner/tuners/hyperband
↪py in run_trial(self, trial, *fit_args, **fit_kwargs)
    352             fit_kwargs['epochs'] = hp.values['tuner/epochs']
    353             fit_kwargs['initial_epoch'] = hp.values['tuner/
↪initial_epoch']
--> 354         super(Hyperband, self).run_trial(trial, *fit_args, **fit_kwargs
    355
    356     def _build_model(self, hp):

~/.pyenv/versions/3.8.5/lib/python3.8/site-packages/kerastuner/engine/
↪multi_execution_tuner.py in run_trial(self, trial, *fit_args, **fit_kwargs)
```

```
     93                 copied_fit_kwargs['callbacks'] = callbacks
     94 
---> 95                 history = self._build_and_fit_model(trial, fit_args,␣
 ↪copied_fit_kwargs)
     96                 for metric, epoch_values in history.history.items():
     97                     if self.oracle.objective.direction == 'min':

~/.pyenv/versions/3.8.5/lib/python3.8/site-packages/kerastuner/engine/tuner.py␣
 ↪in _build_and_fit_model(self, trial, fit_args, fit_kwargs)
    138         """
    139         model = self.hypermodel.build(trial.hyperparameters)
--> 140         return model.fit(*fit_args, **fit_kwargs)
    141 
    142     def run_trial(self, trial, *fit_args, **fit_kwargs):

~/.pyenv/versions/3.8.5/lib/python3.8/site-packages/tensorflow/python/keras/
 ↪engine/training.py in _method_wrapper(self, *args, **kwargs)
    106   def _method_wrapper(self, *args, **kwargs):
    107     if not self._in_multi_worker_mode():  # pylint:␣
 ↪disable=protected-access
--> 108       return method(self, *args, **kwargs)
    109 
    110     # Running inside `run_distribute_coordinator` already.

~/.pyenv/versions/3.8.5/lib/python3.8/site-packages/tensorflow/python/keras/
 ↪engine/training.py in fit(self, x, y, batch_size, epochs, verbose, callbacks,␣
 ↪validation_split, validation_data, shuffle, class_weight, sample_weight,␣
 ↪initial_epoch, steps_per_epoch, validation_steps, validation_batch_size,␣
 ↪validation_freq, max_queue_size, workers, use_multiprocessing)
   1096                 batch_size=batch_size):
   1097               callbacks.on_train_batch_begin(step)
-> 1098               tmp_logs = train_function(iterator)
   1099               if data_handler.should_sync:
   1100                 context.async_wait()

~/.pyenv/versions/3.8.5/lib/python3.8/site-packages/tensorflow/python/eager/
 ↪def_function.py in __call__(self, *args, **kwds)
    778       else:
    779         compiler = "nonXla"
--> 780         result = self._call(*args, **kwds)
    781 
    782       new_tracing_count = self._get_tracing_count()

~/.pyenv/versions/3.8.5/lib/python3.8/site-packages/tensorflow/python/eager/
 ↪def_function.py in _call(self, *args, **kwds)
    805       # In this case we have created variables on the first call, so we
 ↪run the
    806       # defunned version which is guaranteed to never create variables.
```

```
--> 807         return self._stateless_fn(*args, **kwds)  # pylint:␣
↪disable=not-callable
    808       elif self._stateful_fn is not None:
    809         # Release the lock early so that multiple threads can perform the␣
↪call

~/.pyenv/versions/3.8.5/lib/python3.8/site-packages/tensorflow/python/eager/
↪function.py in __call__(self, *args, **kwargs)
   2827     with self._lock:
   2828       graph_function, args, kwargs = self._maybe_define_function(args,␣
↪kwargs)
-> 2829     return graph_function._filtered_call(args, kwargs)  # pylint:␣
↪disable=protected-access
   2830 
   2831   @property

~/.pyenv/versions/3.8.5/lib/python3.8/site-packages/tensorflow/python/eager/
↪function.py in _filtered_call(self, args, kwargs, cancellation_manager)
   1841       `args` and `kwargs`.
   1842     """
-> 1843     return self._call_flat(

   1844         [t for t in nest.flatten((args, kwargs), expand_composites=True)
   1845          if isinstance(t, (ops.Tensor,

~/.pyenv/versions/3.8.5/lib/python3.8/site-packages/tensorflow/python/eager/
↪function.py in _call_flat(self, args, captured_inputs, cancellation_manager)
   1921         and executing_eagerly):
   1922       # No tape is watching; skip to running the function.
-> 1923       return self._build_call_outputs(self._inference_function.call(

   1924           ctx, args, cancellation_manager=cancellation_manager))
   1925     forward_backward = self._select_forward_and_backward_functions(

~/.pyenv/versions/3.8.5/lib/python3.8/site-packages/tensorflow/python/eager/
↪function.py in call(self, ctx, args, cancellation_manager)
    543       with _InterpolateFunctionError(self):
    544         if cancellation_manager is None:
--> 545           outputs = execute.execute(

    546               str(self.signature.name),
    547               num_outputs=self._num_outputs,

~/.pyenv/versions/3.8.5/lib/python3.8/site-packages/tensorflow/python/eager/
↪execute.py in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     57   try:
     58     ctx.ensure_initialized()
```

```
---> 59     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name,␣
↪op_name,
     60                                         inputs, attrs, num_outputs)
     61   except core._NotOkStatusException as e:

KeyboardInterrupt:
```

Tuner is using a much larger batch size than I was and probably that's why it's taking it so much effort to get the same loss as I got with a weakly optimized model from autokeras. I can redo that model manually, or dig into the tuner docs (such as any docs are these days) to see if I can specify batch size in the search method (since it gets specified in the fit method).

The best model (480 nodes on layer x) is now bouncing around in the 0.034 range. 100 epochs is probably more than helpful.

[99]:
```
tuner.results_summary()
```

```
Results summary
Results in keras_tune/untitled_project
Showing 10 best trials
Objective(name='mean_squared_error', direction='min')
Trial summary
Hyperparameters:
units: 512
learning_rate: 0.01
momentum: 0.95
tuner/epochs: 100
tuner/initial_epoch: 34
tuner/bracket: 4
tuner/round: 4
tuner/trial_id: 16d525137b29d2fb93bb6895b7584b14
Score: 0.03339175321161747
Trial summary
Hyperparameters:
units: 480
learning_rate: 0.01
momentum: 0.95
tuner/epochs: 100
tuner/initial_epoch: 34
tuner/bracket: 4
tuner/round: 4
tuner/trial_id: 9ae48accc072e172da3ee0269e5e1af3
Score: 0.03342658281326294
Trial summary
Hyperparameters:
units: 480
learning_rate: 0.01
momentum: 0.95
```

```
tuner/epochs: 34
tuner/initial_epoch: 12
tuner/bracket: 4
tuner/round: 3
tuner/trial_id: b2f2dabda9a3790faa3a0346ba88fbe8
Score: 0.03837982751429081
Trial summary
Hyperparameters:
units: 512
learning_rate: 0.01
momentum: 0.95
tuner/epochs: 34
tuner/initial_epoch: 12
tuner/bracket: 4
tuner/round: 3
tuner/trial_id: fd1fc8699ec897f87f15ff273202dde8
Score: 0.038449836894869804
Trial summary
Hyperparameters:
units: 384
learning_rate: 0.01
momentum: 0.95
tuner/epochs: 34
tuner/initial_epoch: 12
tuner/bracket: 4
tuner/round: 3
tuner/trial_id: a57b762b0eae9c371b8db25bab5c092d
Score: 0.0385188814252615
Trial summary
Hyperparameters:
units: 352
learning_rate: 0.01
momentum: 0.95
tuner/epochs: 34
tuner/initial_epoch: 12
tuner/bracket: 4
tuner/round: 3
tuner/trial_id: 39d3227177d2551846a23a3ed7408850
Score: 0.038651760667562485
Trial summary
Hyperparameters:
units: 512
learning_rate: 0.01
momentum: 0.95
tuner/epochs: 12
tuner/initial_epoch: 4
tuner/bracket: 4
tuner/round: 2
```

```
tuner/trial_id: c2c89310ee9cc2b440b62603a36137d1
Score: 0.046272074803709984
Trial summary
Hyperparameters:
units: 480
learning_rate: 0.01
momentum: 0.95
tuner/epochs: 12
tuner/initial_epoch: 4
tuner/bracket: 4
tuner/round: 2
tuner/trial_id: 7599318f7d3973ca92e0e52f66a6fc53
Score: 0.046395815908908844
Trial summary
Hyperparameters:
units: 384
learning_rate: 0.01
momentum: 0.95
tuner/epochs: 12
tuner/initial_epoch: 4
tuner/bracket: 4
tuner/round: 2
tuner/trial_id: 69cfc0b667d20583d0d028e9be59968f
Score: 0.04689629748463631
Trial summary
Hyperparameters:
units: 352
learning_rate: 0.01
momentum: 0.95
tuner/epochs: 12
tuner/initial_epoch: 4
tuner/bracket: 4
tuner/round: 2
tuner/trial_id: 7e17ffc992fbd73d97eaa098d8fcef91
Score: 0.047147538512945175
```

[100]:
```
tuned_models = tuner.get_best_models(num_models=2)
```

[101]:
```
print(tuned_models[0].summary(),tuned_models[1].summary())
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 900)]             0
_________________________________________________________________
dense (Dense)                (None, 512)               461312
_________________________________________________________________
```

```
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 2052
=================================================================
Total params: 726,020
Trainable params: 726,020
Non-trainable params: 0
_________________________________________________________________
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 900)]             0
_________________________________________________________________
dense (Dense)                (None, 480)               432480
_________________________________________________________________
dense_1 (Dense)              (None, 480)               230880
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 1924
=================================================================
Total params: 665,284
Trainable params: 665,284
Non-trainable params: 0
_________________________________________________________________
None None
```

[102]:

```
model3 = tuned_models[0]
#opt = keras.optimizers.SGD(learning_rate=0.01,momentum=0.95)
#model3.
 ↪compile(loss='mean_squared_error',optimizer=opt,metrics=['mean_squared_error'])
n_epochs = 40
n_batch = 10
# transform the target so that mse is equivalent to the appropriate metric
print('Starting Training')
model3.fit(M_train,scaler.
 ↪transform(mat_train),epochs=n_epochs,batch_size=n_batch)
print('Finished Training')
```

```
Starting Training
Epoch 1/40
900/900 [==============================] - 6s 6ms/step - loss: 0.0383 -
mean_squared_error: 0.0383
Epoch 2/40
900/900 [==============================] - 6s 7ms/step - loss: 0.0377 -
mean_squared_error: 0.0377
Epoch 3/40
900/900 [==============================] - 6s 6ms/step - loss: 0.0373 -
mean_squared_error: 0.0373
```

```
Epoch 4/40
900/900 [==============================] - 7s 7ms/step - loss: 0.0366 -
mean_squared_error: 0.0366
Epoch 5/40
900/900 [==============================] - 6s 7ms/step - loss: 0.0372 -
mean_squared_error: 0.0372
Epoch 6/40
900/900 [==============================] - 6s 7ms/step - loss: 0.0362 -
mean_squared_error: 0.0362
Epoch 7/40
900/900 [==============================] - 6s 7ms/step - loss: 0.0361 -
mean_squared_error: 0.0361
Epoch 8/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0362 -
mean_squared_error: 0.0362
Epoch 9/40
900/900 [==============================] - 6s 7ms/step - loss: 0.0347 -
mean_squared_error: 0.0347
Epoch 10/40
900/900 [==============================] - 6s 7ms/step - loss: 0.0352 -
mean_squared_error: 0.0352
Epoch 11/40
900/900 [==============================] - 6s 6ms/step - loss: 0.0358 -
mean_squared_error: 0.0358
Epoch 12/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0346 -
mean_squared_error: 0.0346
Epoch 13/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0351 -
mean_squared_error: 0.0351
Epoch 14/40
900/900 [==============================] - 6s 7ms/step - loss: 0.0351 -
mean_squared_error: 0.0351
Epoch 15/40
900/900 [==============================] - 7s 7ms/step - loss: 0.0353 -
mean_squared_error: 0.0353
Epoch 16/40
900/900 [==============================] - 7s 7ms/step - loss: 0.0347 -
mean_squared_error: 0.0347
Epoch 17/40
900/900 [==============================] - 6s 7ms/step - loss: 0.0345 -
mean_squared_error: 0.0345
Epoch 18/40
900/900 [==============================] - 6s 6ms/step - loss: 0.0348 -
mean_squared_error: 0.0348
Epoch 19/40
900/900 [==============================] - 7s 7ms/step - loss: 0.0349 -
mean_squared_error: 0.0349
```

```
Epoch 20/40
900/900 [==============================] - 6s 7ms/step - loss: 0.0343 -
mean_squared_error: 0.0343
Epoch 21/40
900/900 [==============================] - 7s 7ms/step - loss: 0.0341 -
mean_squared_error: 0.0341
Epoch 22/40
900/900 [==============================] - 7s 8ms/step - loss: 0.0343 -
mean_squared_error: 0.0343
Epoch 23/40
900/900 [==============================] - 7s 7ms/step - loss: 0.0346 -
mean_squared_error: 0.0346
Epoch 24/40
900/900 [==============================] - 6s 7ms/step - loss: 0.0337 -
mean_squared_error: 0.0337
Epoch 25/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0342 -
mean_squared_error: 0.0342
Epoch 26/40
900/900 [==============================] - 6s 7ms/step - loss: 0.0340 -
mean_squared_error: 0.0340
Epoch 27/40
900/900 [==============================] - 6s 7ms/step - loss: 0.0336 -
mean_squared_error: 0.0336
Epoch 28/40
900/900 [==============================] - 6s 7ms/step - loss: 0.0339 -
mean_squared_error: 0.0339
Epoch 29/40
900/900 [==============================] - 7s 8ms/step - loss: 0.0341 -
mean_squared_error: 0.0341
Epoch 30/40
900/900 [==============================] - 7s 8ms/step - loss: 0.0342 -
mean_squared_error: 0.0342
Epoch 31/40
900/900 [==============================] - 8s 9ms/step - loss: 0.0336 -
mean_squared_error: 0.0336
Epoch 32/40
900/900 [==============================] - 7s 8ms/step - loss: 0.0330 -
mean_squared_error: 0.0330
Epoch 33/40
900/900 [==============================] - 10s 11ms/step - loss: 0.0335 -
mean_squared_error: 0.0335
Epoch 34/40
900/900 [==============================] - 6s 7ms/step - loss: 0.0336 -
mean_squared_error: 0.0336
Epoch 35/40
900/900 [==============================] - 7s 7ms/step - loss: 0.0332 -
mean_squared_error: 0.0332
```

```
Epoch 36/40
900/900 [==============================] - 9s 10ms/step - loss: 0.0332 -
mean_squared_error: 0.0332
Epoch 37/40
900/900 [==============================] - 6s 7ms/step - loss: 0.0337 -
mean_squared_error: 0.0337
Epoch 38/40
900/900 [==============================] - 6s 6ms/step - loss: 0.0333 -
mean_squared_error: 0.0333
Epoch 39/40
900/900 [==============================] - 10s 11ms/step - loss: 0.0335 -
mean_squared_error: 0.0335
Epoch 40/40
900/900 [==============================] - 7s 7ms/step - loss: 0.0334 -
mean_squared_error: 0.0334
Finished Training
```

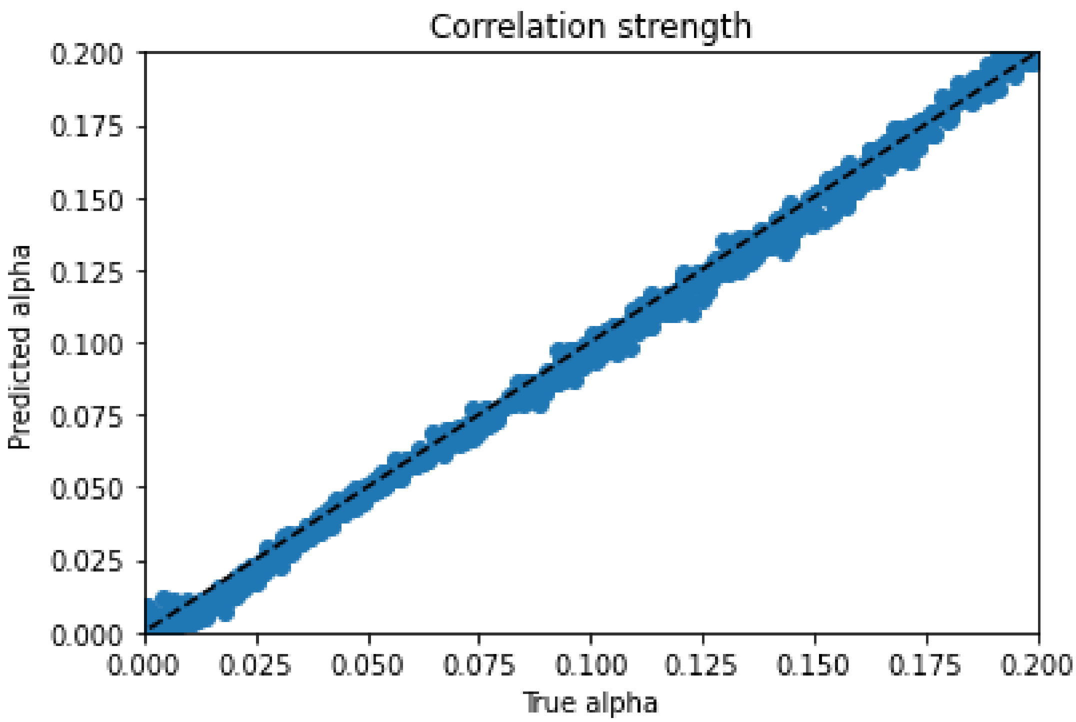
[115]:
```
mat3_train_predict = scaler.inverse_transform(model3.predict(M_train))
mat3_predict = scaler.inverse_transform(model3.predict(M_test))
print('Training score for model ',weight_mse(mat_train,mat3_train_predict))
print('Test score for model ',weight_mse(mat_test,mat3_predict))
```

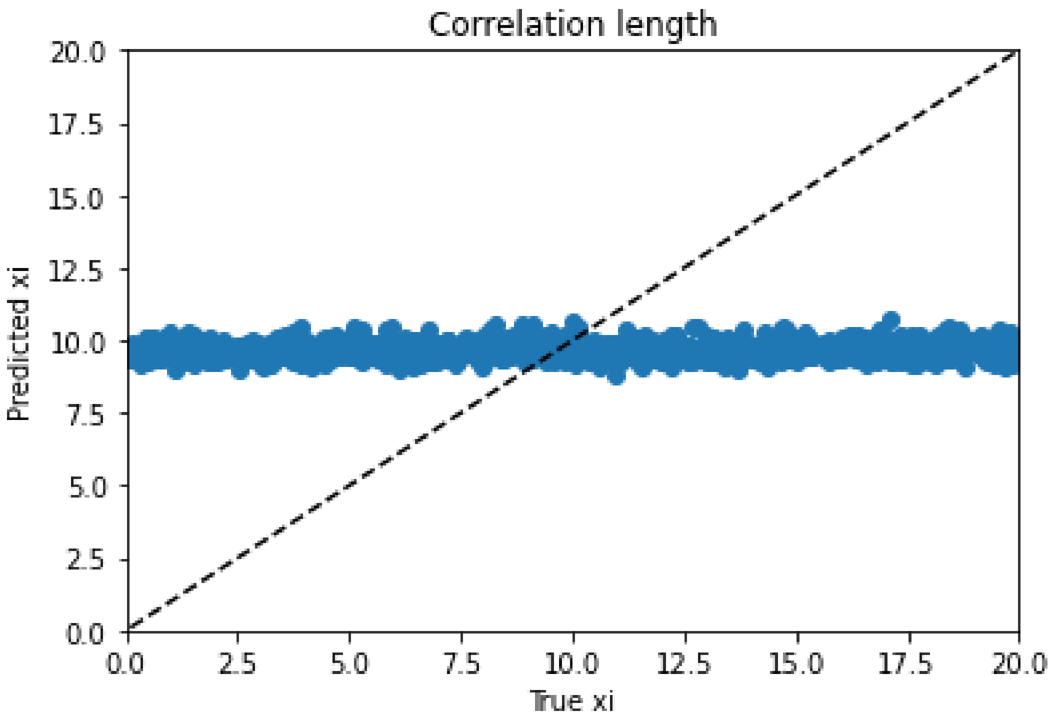
```
Training score for model  0.12945181578137516
Test score for model  0.1339498798977661
```

Let's plot that.

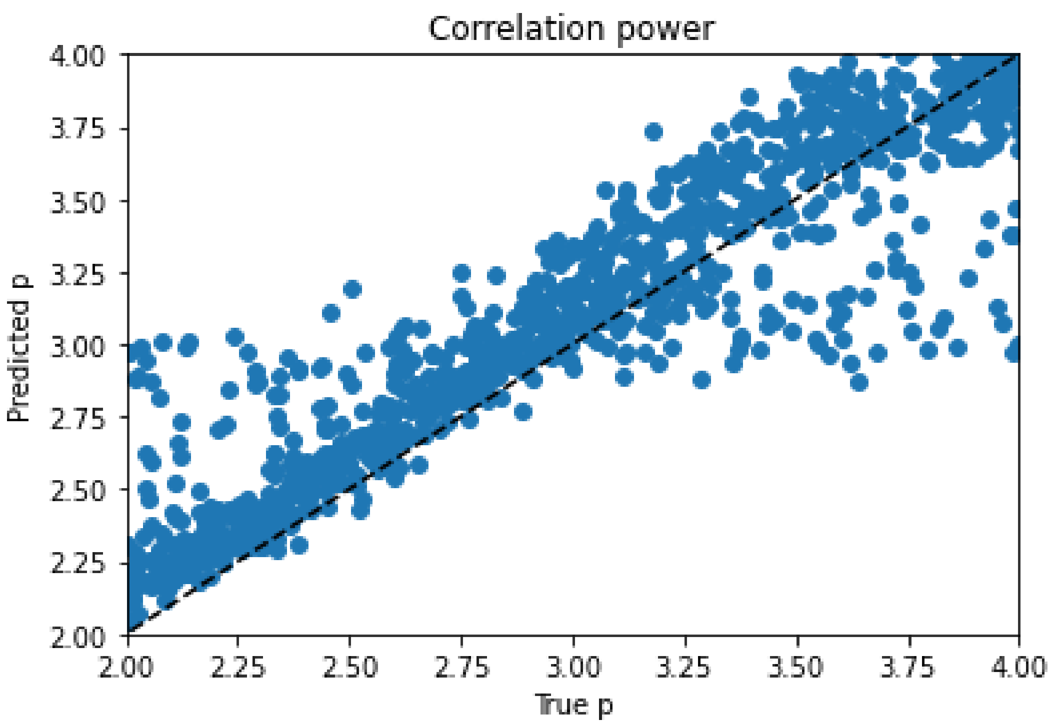
[116]:
```
plt.scatter(mat_test[:,0],mat3_predict[:,0]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True alpha");
plt.ylabel("Predicted alpha");
plt.axis([0, .2, 0, .2])
plt.title("Correlation strength")

plt.figure()
plt.scatter(mat_test[:,1],mat3_predict[:,1]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True xi");
plt.ylabel("Predicted xi");
plt.axis([0, 20, 0, 20])
plt.title("Correlation length")

plt.figure()
plt.scatter(mat_test[:,2],mat3_predict[:,2]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True p");
plt.ylabel("Predicted p");
plt.axis([2, 4, 2, 4])
```

```
plt.title("Correlation power")

plt.figure()
plt.scatter(mat_test[:,3],mat3_predict[:,3]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True d");
plt.ylabel("Predicted d");
plt.axis([3, 6, 3, 6]);
plt.title("Dissipation power");
```

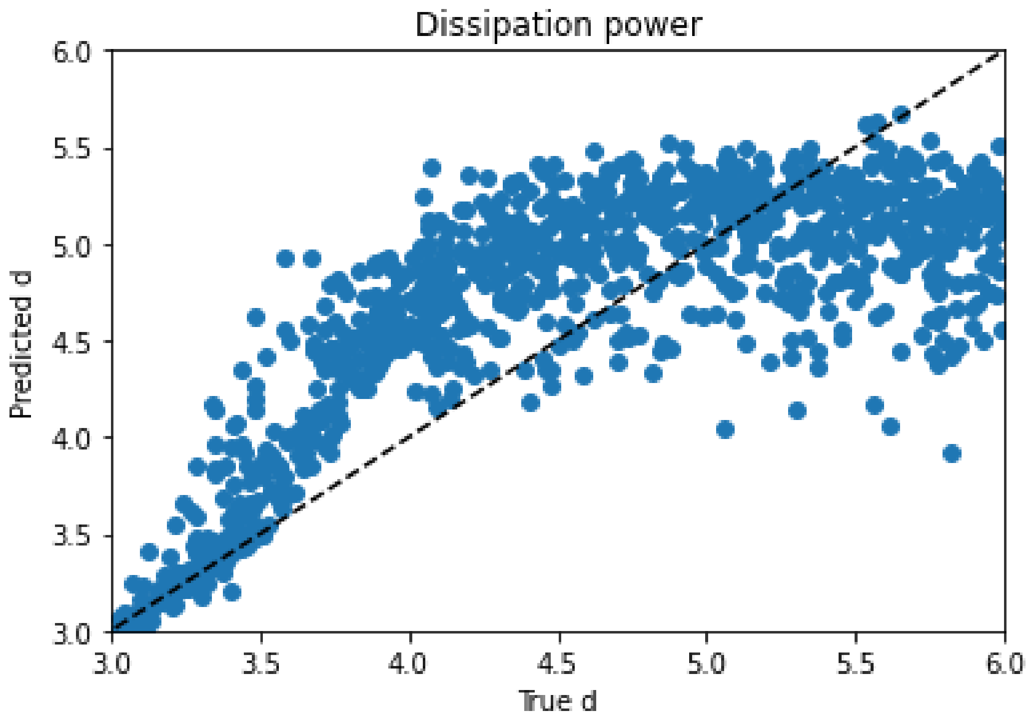




Correlation length
Predicted xi
True xi
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0

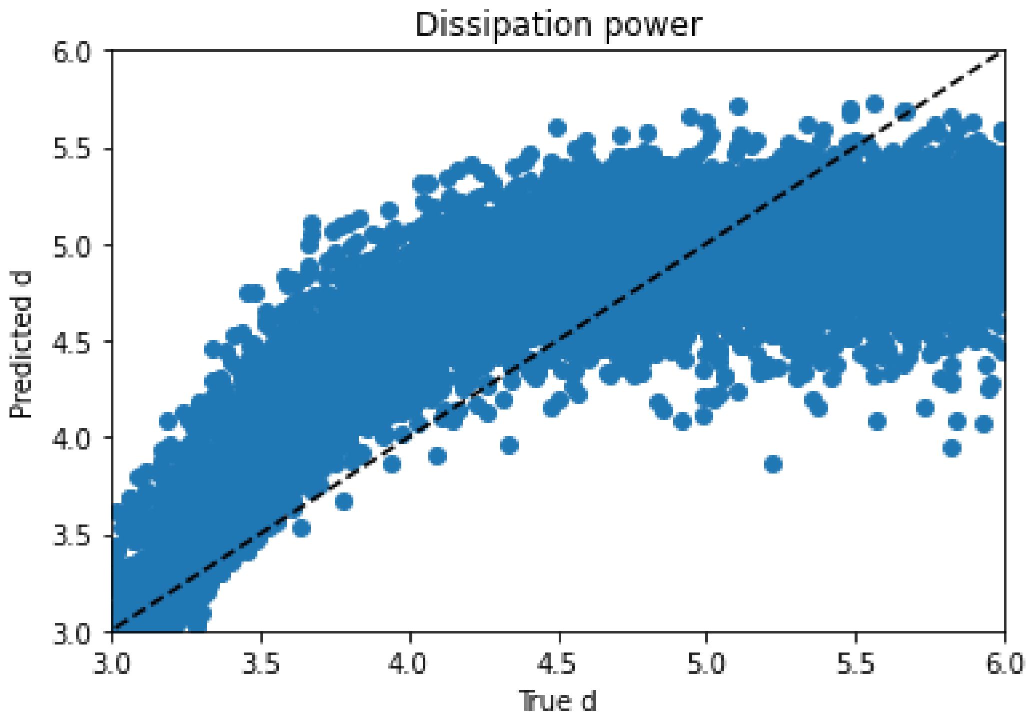


Correlation power
Predicted p
True p
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00



[56]:

```
plt.figure()
plt.scatter(mat_train[:,3],mat3_train_predict[:,3]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True d");
plt.ylabel("Predicted d");
plt.axis([3, 6, 3, 6]);
plt.title("Dissipation power");
```



This seems like the obvious place to look for improvement. I'm boggled why this parameter can't be trained better, but I don't know enough about neural networks to identify a solution. Fortunately there are six hours to go.

Dropout is a way to deal with overfitting, and that's not our problem here. Train and test errors are very close, this plot of the problem target looks very similar for both train and test data.

Well, first, let's see if I can sharpen up the model creation a bit.

1.5 Model 4

[66]:
```
def build_model2(hp):
    inputs = Input(shape=(N,))
    x = Dense(units=512,activation='relu')(inputs)
    y = Dense(
        units = hp.Int('units',min_value=16,max_value=512,step=124),
        activation='relu'
    )(x)
    outputs = Dense(4)(y)
    model = Model(inputs, outputs)
    opt = keras.optimizers.SGD(learning_rate=0.01,momentum=0.95)
    model.
 ↪compile(loss='mean_squared_error',optimizer=opt,metrics=['mean_squared_error'])
    return model
```

[67]:
```
tuner2 = kerastuner.tuners.Hyperband(
        build_model2,
        objective='mean_squared_error',
        max_epochs=100,
        executions_per_trial=2,
        directory='keras_tune2'
)
```

[68]:
```
print('Starting Tuning')
tuner2.search(M_train,scaler.transform(mat_train),
             validation_data=(M_test,mat_test))
print('Finished Tuning')
```

```
Trial 5 Complete [00h 00m 08s]
mean_squared_error: 0.0581966508179903

Best mean_squared_error So Far: 0.05451534874737263
Total elapsed time: 00h 00m 54s
INFO:tensorflow:Oracle triggered exit
Finished Tuning
```

[72]:
```
tuned_models2 = tuner2.get_best_models(num_models=5)
```

```
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.iter
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.decay
WARNING:tensorflow:Unresolved object in checkpoint:
(root).optimizer.learning_rate
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.momentum
WARNING:tensorflow:A checkpoint was restored (e.g. tf.train.Checkpoint.restore
or tf.keras.Model.load_weights) but not all checkpointed values were used. See
above for specific issues. Use expect_partial() on the load status object, e.g.
tf.train.Checkpoint.restore(…).expect_partial(), to silence these warnings, or
use assert_consumed() to make the check explicit. See
https://www.tensorflow.org/guide/checkpoint#loading_mechanics for details.
```

[73]:
```
for tm in tuned_models2:
    print(tm.summary())
```

```
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.iter
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.decay
WARNING:tensorflow:Unresolved object in checkpoint:
(root).optimizer.learning_rate
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.momentum
WARNING:tensorflow:A checkpoint was restored (e.g. tf.train.Checkpoint.restore
or tf.keras.Model.load_weights) but not all checkpointed values were used. See
above for specific issues. Use expect_partial() on the load status object, e.g.
tf.train.Checkpoint.restore(…).expect_partial(), to silence these warnings, or
use assert_consumed() to make the check explicit. See
```

```
https://www.tensorflow.org/guide/checkpoint#loading_mechanics for details.
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 900)]             0
_________________________________________________________________
dense (Dense)                (None, 512)               461312
_________________________________________________________________
dense_1 (Dense)              (None, 264)               135432
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 1060
=================================================================
Total params: 597,804
Trainable params: 597,804
Non-trainable params: 0
_________________________________________________________________
None
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 900)]             0
_________________________________________________________________
dense (Dense)                (None, 512)               461312
_________________________________________________________________
dense_1 (Dense)              (None, 388)               199044
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 1556
=================================================================
Total params: 661,912
Trainable params: 661,912
Non-trainable params: 0
_________________________________________________________________
None
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 900)]             0
_________________________________________________________________
dense (Dense)                (None, 512)               461312
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 2052
=================================================================
Total params: 726,020
```

```
Trainable params: 726,020
Non-trainable params: 0
_________________________________________________________________
None
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 900)]             0
_________________________________________________________________
dense (Dense)                (None, 512)               461312
_________________________________________________________________
dense_1 (Dense)              (None, 140)               71820
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 564
=================================================================
Total params: 533,696
Trainable params: 533,696
Non-trainable params: 0
_________________________________________________________________
None
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 900)]             0
_________________________________________________________________
dense (Dense)                (None, 512)               461312
_________________________________________________________________
dense_1 (Dense)              (None, 16)                8208
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 68
=================================================================
Total params: 469,588
Trainable params: 469,588
Non-trainable params: 0
_________________________________________________________________
None
```

[74]:

```
model4 = tuned_models2[0]
mat4_train_predict = scaler.inverse_transform(model4.predict(M_train))
mat4_predict = scaler.inverse_transform(model4.predict(M_test))
print('Training score for model ',weight_mse(mat_train,mat4_train_predict))
print('Test score for model ',weight_mse(mat_test,mat4_predict))
```

```
Training score for model  0.20658943634673155
Test score for model  0.20357652644143265
```

For whatever reason the tuner quit early, long before it had tried doing 100 epochs. Let's manually train a model with 512 and 256 neurons in the two hidden layers.

1.6 Model 5

[79]:
```
input5_layer = Input(shape=(N,))
layer51 = Dense(512,activation='relu')(input5_layer)
layer52 = Dense(256,activation='relu')(layer51)
layer53 = Dense(4)(layer52)

model5 = Model(name='Model_5',inputs=input5_layer, outputs=layer53)
model5.summary()
```

```
Model: "Model_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 900)]             0
_________________________________________________________________
dense_6 (Dense)              (None, 512)               461312
_________________________________________________________________
dense_7 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_8 (Dense)              (None, 4)                 1028
=================================================================
Total params: 593,668
Trainable params: 593,668
Non-trainable params: 0
_________________________________________________________________
```

[80]:
```
opt = keras.optimizers.SGD(learning_rate=0.01,momentum=0.95)
model5.
 ↪compile(loss='mean_squared_error',optimizer=opt,metrics=['mean_squared_error'])
```

[81]:
```
n_epochs = 40
n_batch = 10
# transform the target so that mse is equivalent to the appropriate metric
print('Starting Training')
model5.fit(M_train,scaler.
 ↪transform(mat_train),epochs=n_epochs,batch_size=n_batch)
print('Finished Training')
```

```
Starting Training
Epoch 1/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0600 -
mean_squared_error: 0.0600
Epoch 2/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0503 -
```

```
mean_squared_error: 0.0503
Epoch 3/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0486 -
mean_squared_error: 0.0486
Epoch 4/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0467 -
mean_squared_error: 0.0467
Epoch 5/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0453 -
mean_squared_error: 0.0453
Epoch 6/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0436 -
mean_squared_error: 0.0436
Epoch 7/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0429 -
mean_squared_error: 0.0429
Epoch 8/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0417 -
mean_squared_error: 0.0417
Epoch 9/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0413 -
mean_squared_error: 0.0413
Epoch 10/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0405 -
mean_squared_error: 0.0405
Epoch 11/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0396 -
mean_squared_error: 0.0396
Epoch 12/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0390 -
mean_squared_error: 0.0390
Epoch 13/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0393 -
mean_squared_error: 0.0393
Epoch 14/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0385 -
mean_squared_error: 0.0385
Epoch 15/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0382 -
mean_squared_error: 0.0382
Epoch 16/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0380 -
mean_squared_error: 0.0380
Epoch 17/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0377 -
mean_squared_error: 0.0377
Epoch 18/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0377 -
```

```
mean_squared_error: 0.0377
Epoch 19/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0375 -
mean_squared_error: 0.0375
Epoch 20/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0374 -
mean_squared_error: 0.0374
Epoch 21/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0369 -
mean_squared_error: 0.0369
Epoch 22/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0365 -
mean_squared_error: 0.0365
Epoch 23/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0373 -
mean_squared_error: 0.0373
Epoch 24/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0370 -
mean_squared_error: 0.0370
Epoch 25/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0364 -
mean_squared_error: 0.0364
Epoch 26/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0361 -
mean_squared_error: 0.0361
Epoch 27/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0356 -
mean_squared_error: 0.0356
Epoch 28/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0360 -
mean_squared_error: 0.0360
Epoch 29/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0359 -
mean_squared_error: 0.0359
Epoch 30/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0358 -
mean_squared_error: 0.0358
Epoch 31/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0349 -
mean_squared_error: 0.0349
Epoch 32/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0352 -
mean_squared_error: 0.0352
Epoch 33/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0353 -
mean_squared_error: 0.0353
Epoch 34/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0356 -
```

```
mean_squared_error: 0.0356
Epoch 35/40
900/900 [==============================] - 6s 6ms/step - loss: 0.0353 -
mean_squared_error: 0.0353
Epoch 36/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0353 -
mean_squared_error: 0.0353
Epoch 37/40
900/900 [==============================] - 6s 6ms/step - loss: 0.0351 -
mean_squared_error: 0.0351
Epoch 38/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0356 -
mean_squared_error: 0.0356
Epoch 39/40
900/900 [==============================] - 6s 6ms/step - loss: 0.0351 -
mean_squared_error: 0.0351
Epoch 40/40
900/900 [==============================] - 7s 7ms/step - loss: 0.0347 -
mean_squared_error: 0.0347
Finished Training
```

[82]:
```
mat5_train_predict = scaler.inverse_transform(model5.
 ↪predict(M_train,batch_size=n_batch))
mat5_predict = scaler.inverse_transform(model5.
 ↪predict(M_test,batch_size=n_batch))
print('Training score for model ',weight_mse(mat_train,mat5_train_predict))
print('Test score for model ',weight_mse(mat_test,mat5_predict))
```

```
Training score for model  0.129377103538l4633
Test score for model  0.12651953377293923
```

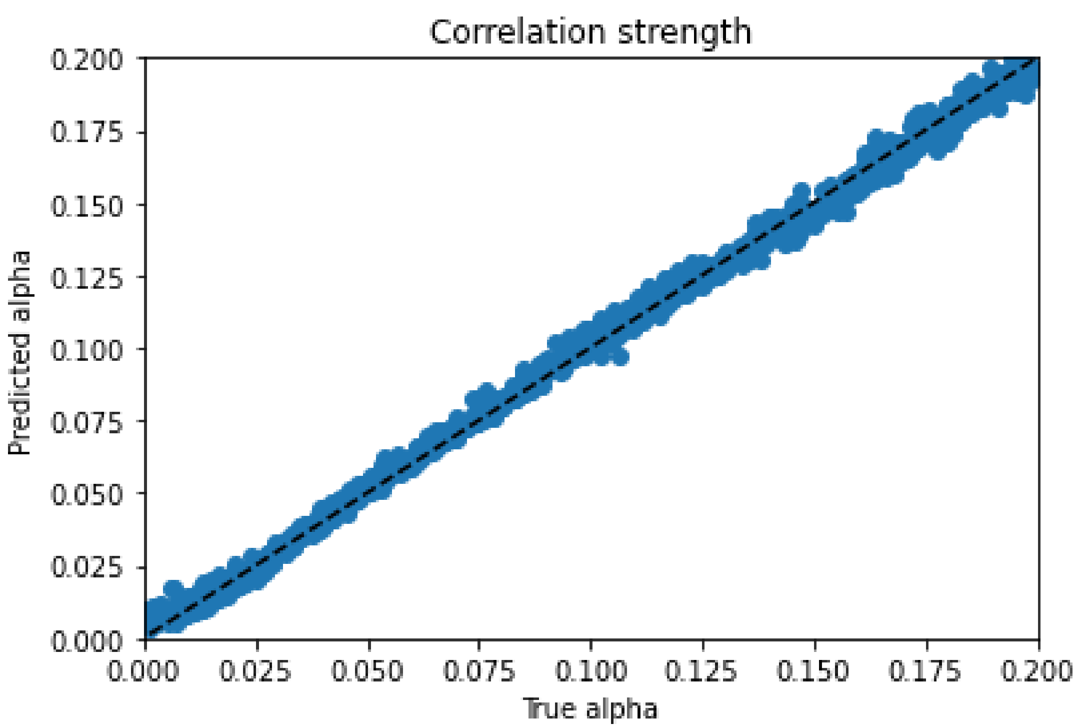
[121]:
```
mat_sub = scaler.inverse_transform(model5.predict(M_train))
```

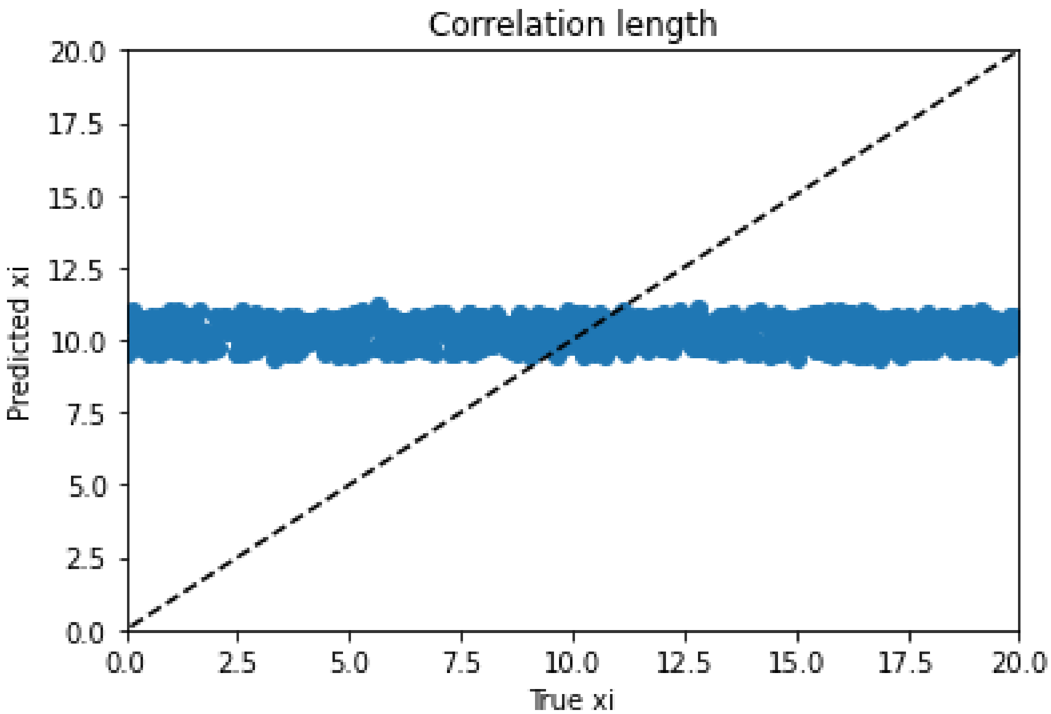
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-121-1ed6e5682e32> in <module>
----> 1 mat_sub = scaler.inverse_transform(model5.predict(M_train))

NameError: name 'model5' is not defined
```

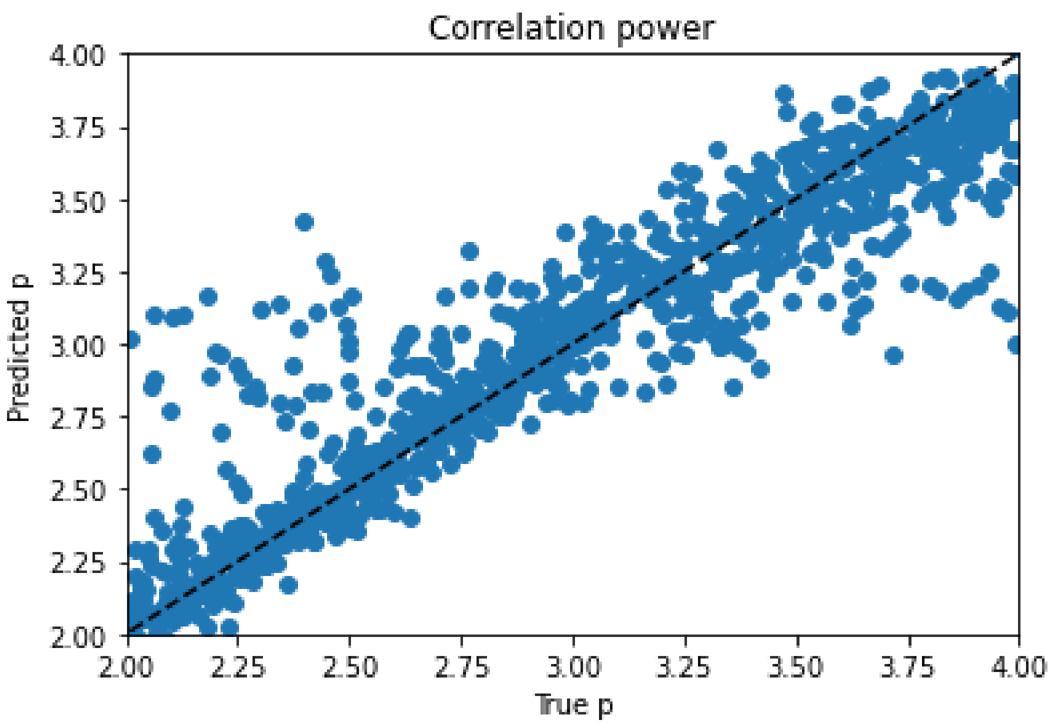
Let's plot that.

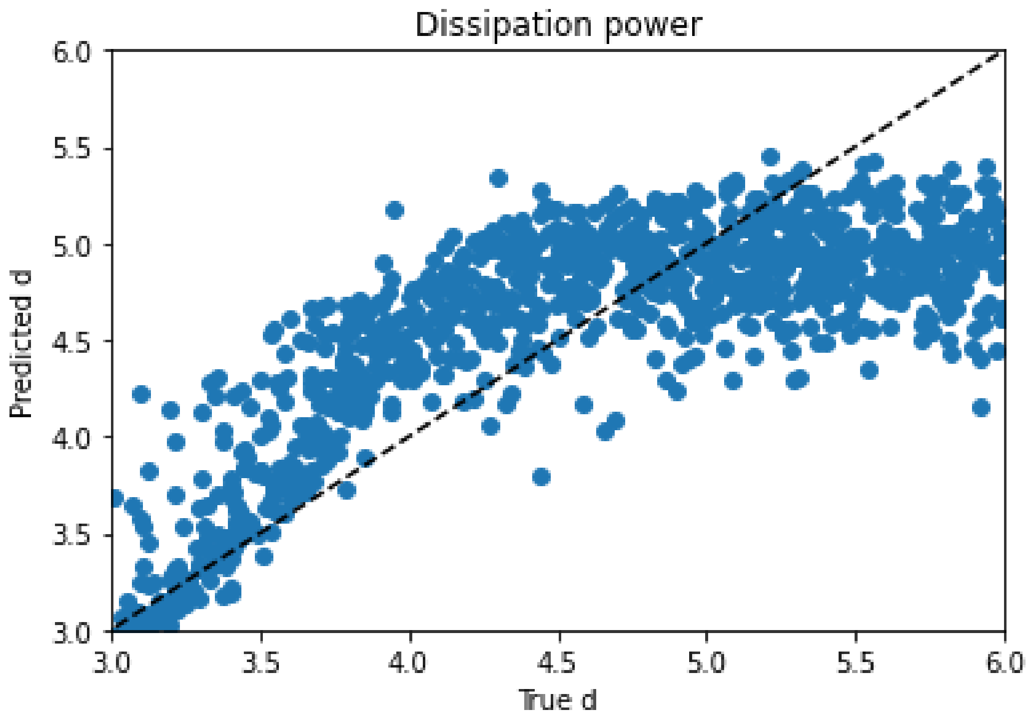
[122]:
```
plt.scatter(mat_test[:,0],mat5_predict[:,0]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True alpha");
plt.ylabel("Predicted alpha");
plt.axis([0, .2, 0, .2])
```

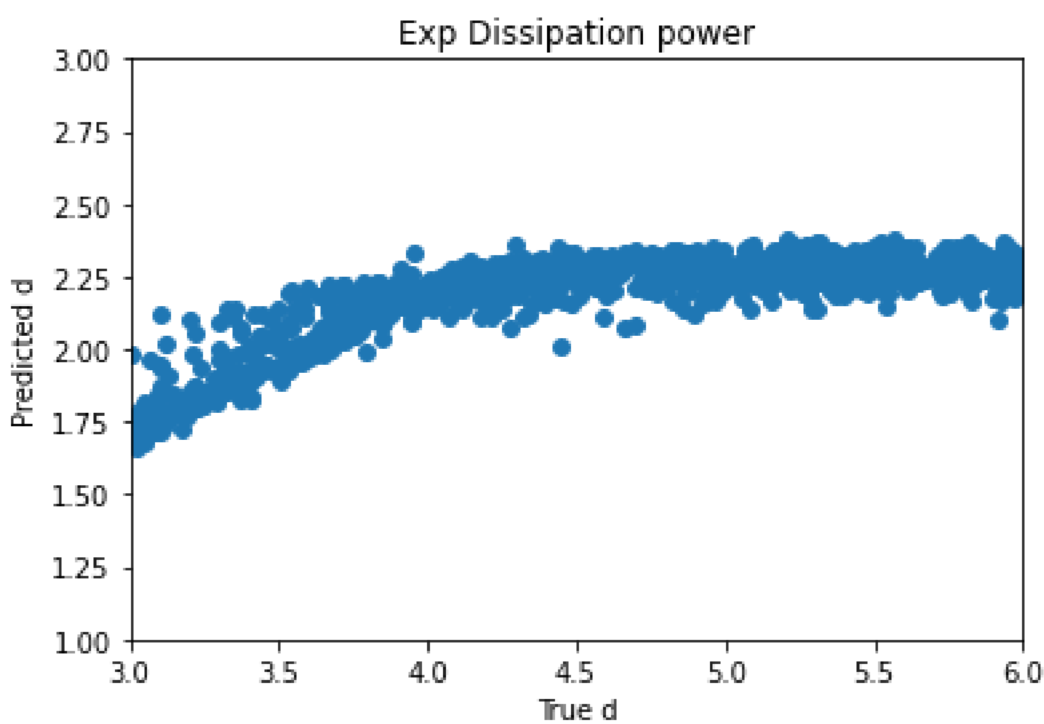
```
plt.title("Correlation strength")

plt.figure()
plt.scatter(mat_test[:,1],mat5_predict[:,1]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True xi");
plt.ylabel("Predicted xi");
plt.axis([0, 20, 0, 20])
plt.title("Correlation length")

plt.figure()
plt.scatter(mat_test[:,2],mat5_predict[:,2]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True p");
plt.ylabel("Predicted p");
plt.axis([2, 4, 2, 4])
plt.title("Correlation power")

plt.figure()
plt.scatter(mat_test[:,3],mat5_predict[:,3]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True d");
plt.ylabel("Predicted d");
plt.axis([3, 6, 3, 6]);
plt.title("Dissipation power");
```




Correlation length
Predicted xi
True xi
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Correlation power
Predicted p
True p
2.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75
4.00


Dissipation power
Predicted d
True d
6.0
5.5
5.0
4.5
4.0
3.5
3.0
3.0
3.5
4.0
4.5
5.0
5.5
6.0


Exp Dissipation power
Predicted d
True d
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
3.0
3.5
4.0
4.5
5.0
5.5
6.0

1.7 Model 6

I don't see an easier way to do this. I've worked damn hard this week and my brain is going numb.

Well, I mean I *do* see a better way, but not an easier way. This is a crude hack; I will go back and see if I can split the output neurons and put an exponential activation on the neuron that reports d.

[75]:
```
mat_train_tr = mat_train.copy()
```

[76]:
```
mat_train_tr is mat_train
```

[76]:
```
False
```

[77]:
```
mat_train_tr[:,3]=np.log(mat_train[:,3])
```

[78]:
```
mat_train[:3,:]
```

[78]:
```
array([[ 0.11501314,  8.01074766,  2.53666314,  4.05991278],
       [ 0.174987  , 11.64201446,  3.68769076,  4.24813948],
       [ 0.10317918,  1.88649881,  2.31492155,  4.6918782 ]])
```

[79]:
```
mat_train_tr[:3,:]
```

[79]:
```
array([[ 0.11501314,  8.01074766,  2.53666314,  1.40116149],
       [ 0.174987  , 11.64201446,  3.68769076,  1.44648112],
       [ 0.10317918,  1.88649881,  2.31492155,  1.54583297]])
```

[80]:
```
mat_test_tr = mat_test.copy()
mat_test_tr[:,3]=np.log(mat_test[:,3])
```

[81]:
```
trscaler=MinMaxScaler()
trscaler.fit(mat_train_tr)
```

[81]:
```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

[92]:
```
input6_layer = Input(shape=(N,))
layer61 = Dense(512,activation='relu')(input6_layer)
layer62 = Dense(256,activation='relu')(layer61)
layer63 = Dense(4)(layer62)

model6 = Model(name='Model_6',inputs=input6_layer, outputs=layer63)
model6.summary()
```

```
Model: "Model_6"
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 900)]             0

_________________________________________________________________
dense_12 (Dense)             (None, 512)               461312

_________________________________________________________________
dense_13 (Dense)             (None, 256)               131328

_________________________________________________________________
dense_14 (Dense)             (None, 4)                 1028
=================================================================
Total params: 593,668
Trainable params: 593,668
Non-trainable params: 0

_________________________________________________________________
```

[93]:

```
#opt = keras.optimizers.SGD(learning_rate=0.01,momentum=0.95)
model6.
 ↪compile(loss='mean_squared_error',optimizer=opt,metrics=['mean_squared_error'])
```

[95]:

```
n_epochs = 40
n_batch = 10
# transform the target to avoid
print('Starting Training')
model6.fit(M_train,trscaler.
 ↪transform(mat_train_tr),epochs=n_epochs,batch_size=n_batch)
print('Finished Training')
```

```
Starting Training
Epoch 1/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0586 -
mean_squared_error: 0.0586
Epoch 2/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0493 -
mean_squared_error: 0.0493
Epoch 3/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0470 -
mean_squared_error: 0.0470
Epoch 4/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0449 -
mean_squared_error: 0.0449
Epoch 5/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0436 -
mean_squared_error: 0.0436
Epoch 6/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0416 -
mean_squared_error: 0.0416
Epoch 7/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0401 -
```

```
mean_squared_error: 0.0401
Epoch 8/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0389 -
mean_squared_error: 0.0389
Epoch 9/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0392 -
mean_squared_error: 0.0392
Epoch 10/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0387 -
mean_squared_error: 0.0387
Epoch 11/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0384 -
mean_squared_error: 0.0384
Epoch 12/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0375 -
mean_squared_error: 0.0375
Epoch 13/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0378 -
mean_squared_error: 0.0378
Epoch 14/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0370 -
mean_squared_error: 0.0370
Epoch 15/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0366 -
mean_squared_error: 0.0366
Epoch 16/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0364 -
mean_squared_error: 0.0364
Epoch 17/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0363 -
mean_squared_error: 0.0363
Epoch 18/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0354 -
mean_squared_error: 0.0354
Epoch 19/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0355 -
mean_squared_error: 0.0355
Epoch 20/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0350 -
mean_squared_error: 0.0350
Epoch 21/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0354 -
mean_squared_error: 0.0354
Epoch 22/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0346 -
mean_squared_error: 0.0346
Epoch 23/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0357 -
```

```
mean_squared_error: 0.0357
Epoch 24/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0344 -
mean_squared_error: 0.0344
Epoch 25/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0342 -
mean_squared_error: 0.0342
Epoch 26/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0343 -
mean_squared_error: 0.0343
Epoch 27/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0343 -
mean_squared_error: 0.0343
Epoch 28/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0345 -
mean_squared_error: 0.0345
Epoch 29/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0340 -
mean_squared_error: 0.0340
Epoch 30/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0344 -
mean_squared_error: 0.0344
Epoch 31/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0336 -
mean_squared_error: 0.0336
Epoch 32/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0334 -
mean_squared_error: 0.0334
Epoch 33/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0335 -
mean_squared_error: 0.0335
Epoch 34/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0337 -
mean_squared_error: 0.0337
Epoch 35/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0331 -
mean_squared_error: 0.0331
Epoch 36/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0333 -
mean_squared_error: 0.0333
Epoch 37/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0331 -
mean_squared_error: 0.0331
Epoch 38/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0329 -
mean_squared_error: 0.0329
Epoch 39/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0328 -
```

```
mean_squared_error: 0.0328
Epoch 40/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0330 -
mean_squared_error: 0.0330
Finished Training
```

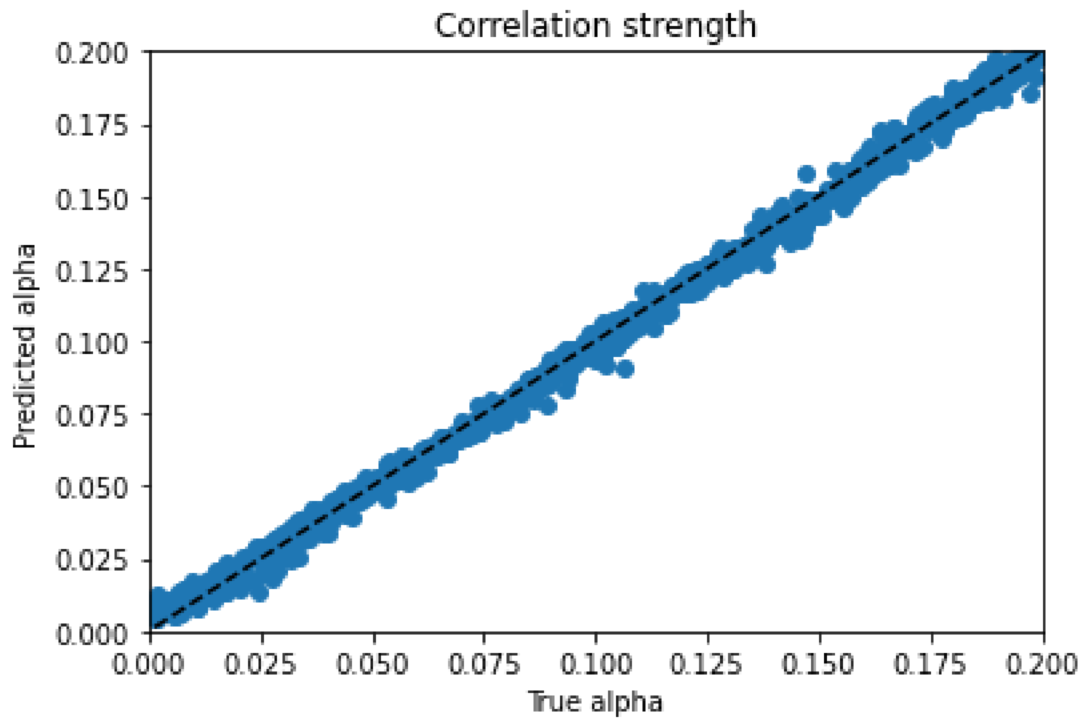
[96]:
```
mat6_train_predict = trscaler.inverse_transform(model6.
 ↪predict(M_train,batch_size=n_batch))
mat6_train_predict[:,3]=np.exp(mat6_train_predict[:,3])
mat6_predict = trscaler.inverse_transform(model6.
 ↪predict(M_test,batch_size=n_batch))
mat6_predict[:,3]=np.exp(mat6_predict[:,3])
print('Training score for model ',weight_mse(mat_train,mat6_train_predict))
print('Test score for model ',weight_mse(mat_test,mat6_predict))
```

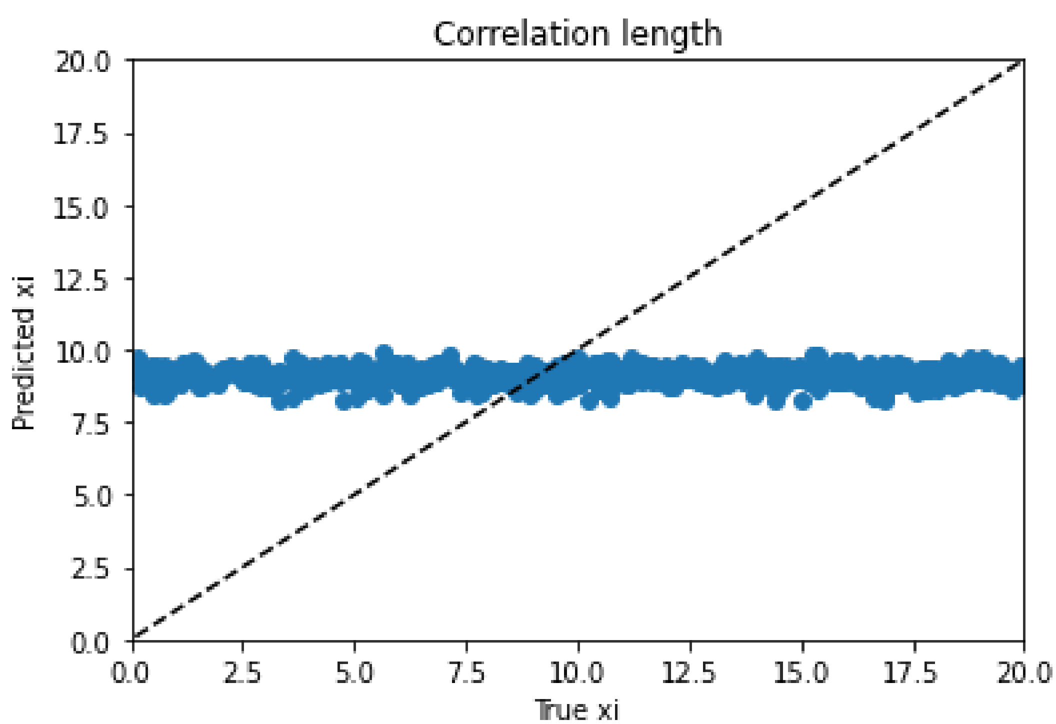
```
Training score for model  0.1545942497902711
Test score for model  0.15046690286594983
```

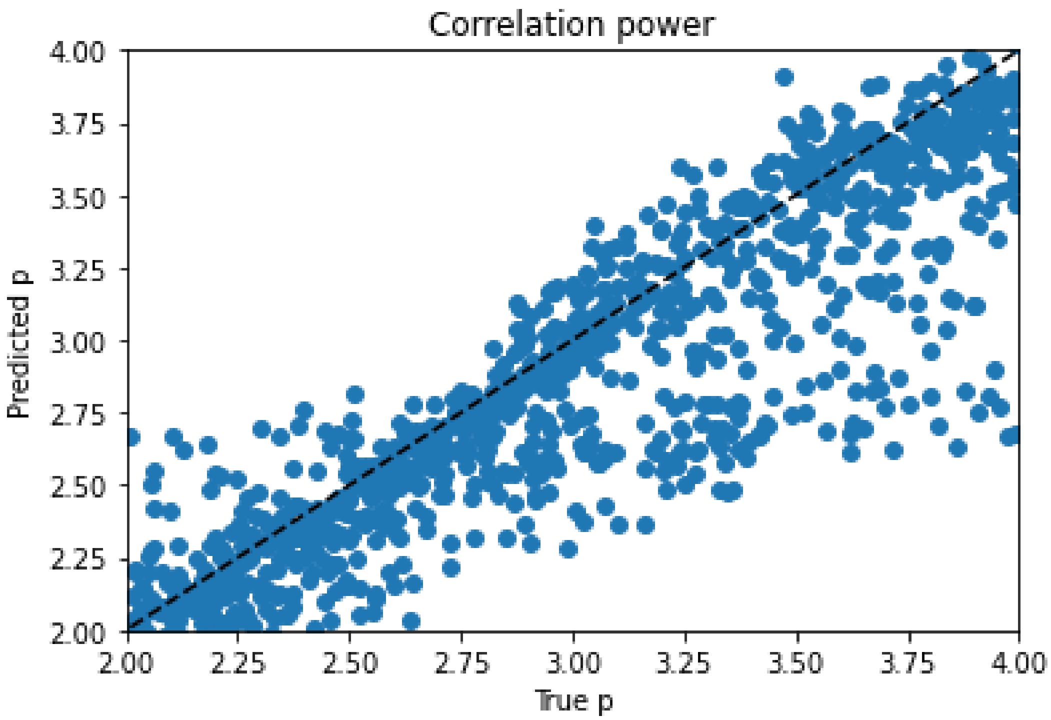
Let's plot that.

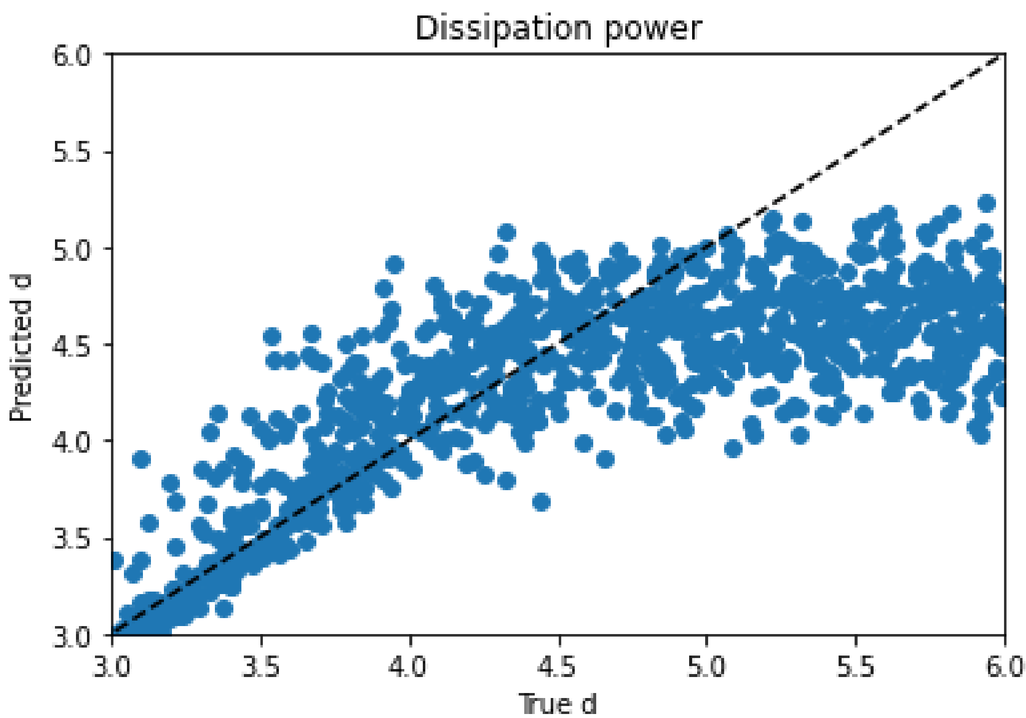
[97]:
```
plt.scatter(mat_test[:,0],mat6_predict[:,0]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True alpha");
plt.ylabel("Predicted alpha");
plt.axis([0, .2, 0, .2])
plt.title("Correlation strength")

plt.figure()
plt.scatter(mat_test[:,1],mat6_predict[:,1]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True xi");
plt.ylabel("Predicted xi");
plt.axis([0, 20, 0, 20])
plt.title("Correlation length")

plt.figure()
plt.scatter(mat_test[:,2],mat6_predict[:,2]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True p");
plt.ylabel("Predicted p");
plt.axis([2, 4, 2, 4])
plt.title("Correlation power")

plt.figure()
plt.scatter(mat_test[:,3],mat6_predict[:,3]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True d");
plt.ylabel("Predicted d");
```

```
plt.axis([3, 6, 3, 6]);
plt.title("Dissipation power");
```






Correlation power
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
Predicted p
2.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75
4.00
True p


Dissipation power
6.0
5.5
5.0
4.5
4.0
3.5
3.0
Predicted d
3.0
3.5
4.0
4.5
5.0
5.5
6.0
True d

1.8 Model 7

Wherein we first of all try to do things the right way, by splitting d from the other variables and giving it a distinct activation function, then track down what in God's name that function might be.

[24]:
```
input7_layer = Input(shape=(N,))
layer71 = Dense(512,activation='relu')(input7_layer)
layer72 = Dense(256,activation='relu')(layer71)
layer73a = Dense(3)(layer72)
layer73b = Dense(1,activation='tanh')(layer72)
layer74 = Concatenate()([layer73a,layer73b])

model7 = Model(name='Model_7',inputs=input7_layer, outputs=layer74)
model7.summary()
```

```
Model: "Model_7"
________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
==================
input_6 (InputLayer)            [(None, 900)]        0

________________________________________________________________________________
__________________
dense_20 (Dense)                (None, 512)          461312      input_6[0][0]

________________________________________________________________________________
__________________
dense_21 (Dense)                (None, 256)          131328      dense_20[0][0]

________________________________________________________________________________
__________________
dense_22 (Dense)                (None, 3)            771         dense_21[0][0]

________________________________________________________________________________
__________________
dense_23 (Dense)                (None, 1)            257         dense_21[0][0]

________________________________________________________________________________
__________________
concatenate_4 (Concatenate)     (None, 4)            0           dense_22[0][0]
                                                                 dense_23[0][0]
================================================================================
==================
Total params: 593,668
Trainable params: 593,668
Non-trainable params: 0

________________________________________________________________________________
```

----

[25]:
```
opt = keras.optimizers.SGD(learning_rate=0.01,momentum=0.95)
model7.
 ↪compile(loss='mean_squared_error',optimizer=opt,metrics=['mean_squared_error'])
```

[26]:
```
n_epochs = 40
n_batch = 10
# transform the target to scale features to [0,1]...not sure how this will␣
 ↪affect d
print('Starting Training')
model7.fit(M_train,scaler.
 ↪transform(mat_train),epochs=n_epochs,batch_size=n_batch)
print('Finished Training')
```

```
Starting Training
Epoch 1/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0598 -
mean_squared_error: 0.0598
Epoch 2/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0503 -
mean_squared_error: 0.0503
Epoch 3/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0485 -
mean_squared_error: 0.0485
Epoch 4/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0470 -
mean_squared_error: 0.0470
Epoch 5/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0451 -
mean_squared_error: 0.0451
Epoch 6/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0436 -
mean_squared_error: 0.0436
Epoch 7/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0423 -
mean_squared_error: 0.0423
Epoch 8/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0413 -
mean_squared_error: 0.0413
Epoch 9/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0409 -
mean_squared_error: 0.0409
Epoch 10/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0399 -
mean_squared_error: 0.0399
Epoch 11/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0390 -
```

```
mean_squared_error: 0.0390
Epoch 12/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0394 -
mean_squared_error: 0.0394
Epoch 13/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0391 -
mean_squared_error: 0.0391
Epoch 14/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0384 -
mean_squared_error: 0.0384
Epoch 15/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0380 -
mean_squared_error: 0.0380
Epoch 16/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0378 -
mean_squared_error: 0.0378
Epoch 17/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0378 -
mean_squared_error: 0.0378
Epoch 18/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0371 -
mean_squared_error: 0.0371
Epoch 19/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0370 -
mean_squared_error: 0.0370
Epoch 20/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0370 -
mean_squared_error: 0.0370
Epoch 21/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0363 -
mean_squared_error: 0.0363
Epoch 22/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0371 -
mean_squared_error: 0.0371
Epoch 23/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0359 -
mean_squared_error: 0.0359
Epoch 24/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0361 -
mean_squared_error: 0.0361
Epoch 25/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0362 -
mean_squared_error: 0.0362
Epoch 26/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0357 -
mean_squared_error: 0.0357
Epoch 27/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0357 -
```

```
mean_squared_error: 0.0357
Epoch 28/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0358 -
mean_squared_error: 0.0358
Epoch 29/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0359 -
mean_squared_error: 0.0359
Epoch 30/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0355 -
mean_squared_error: 0.0355
Epoch 31/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0348 -
mean_squared_error: 0.0348
Epoch 32/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0351 -
mean_squared_error: 0.0351
Epoch 33/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0354 -
mean_squared_error: 0.0354
Epoch 34/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0353 -
mean_squared_error: 0.0353
Epoch 35/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0345 -
mean_squared_error: 0.0345
Epoch 36/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0346 -
mean_squared_error: 0.0346
Epoch 37/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0344 -
mean_squared_error: 0.0344
Epoch 38/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0345 -
mean_squared_error: 0.0345
Epoch 39/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0343 -
mean_squared_error: 0.0343
Epoch 40/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0347 -
mean_squared_error: 0.0347
Finished Training
```

[27]:

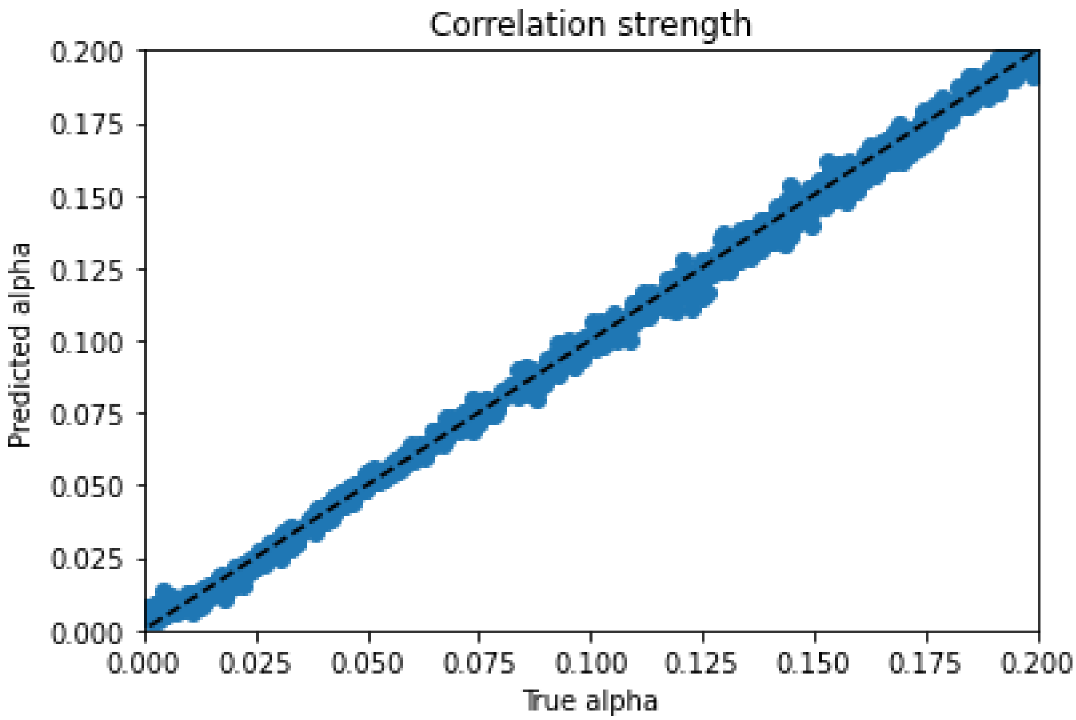
```
mat7_train_predict = scaler.inverse_transform(model7.
 ↪predict(M_train,batch_size=n_batch))
mat7_predict = scaler.inverse_transform(model7.
 ↪predict(M_test,batch_size=n_batch))
print('Training score for model ',weight_mse(mat_train,mat7_train_predict))
```

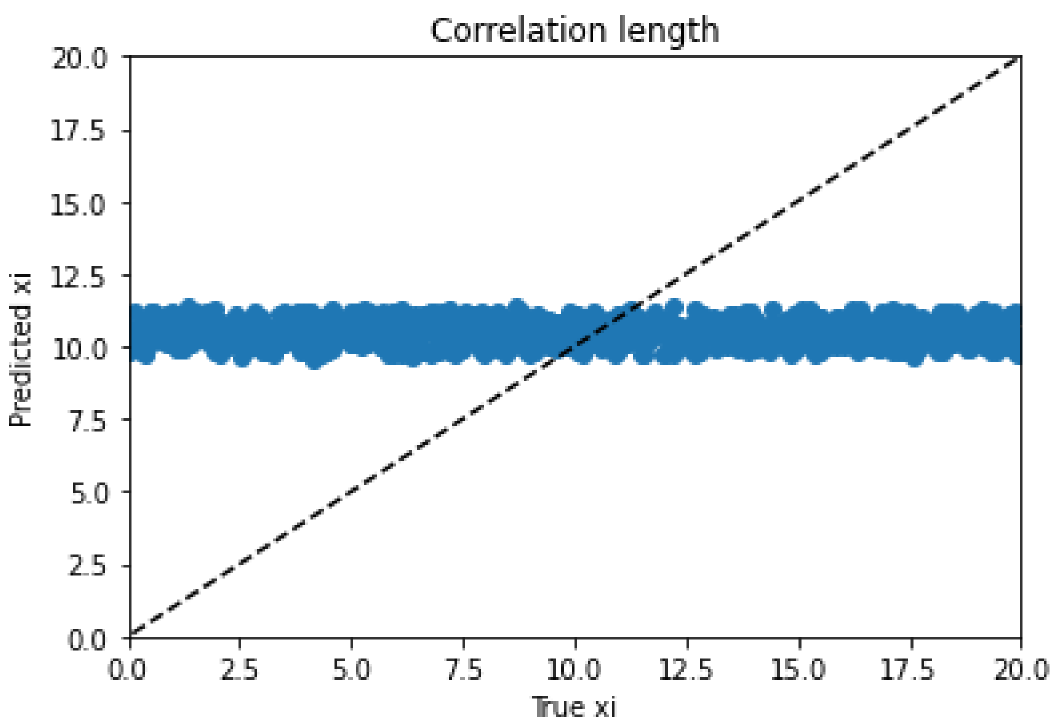
```
print('Test score for model ',weight_mse(mat_test,mat7_predict))
```

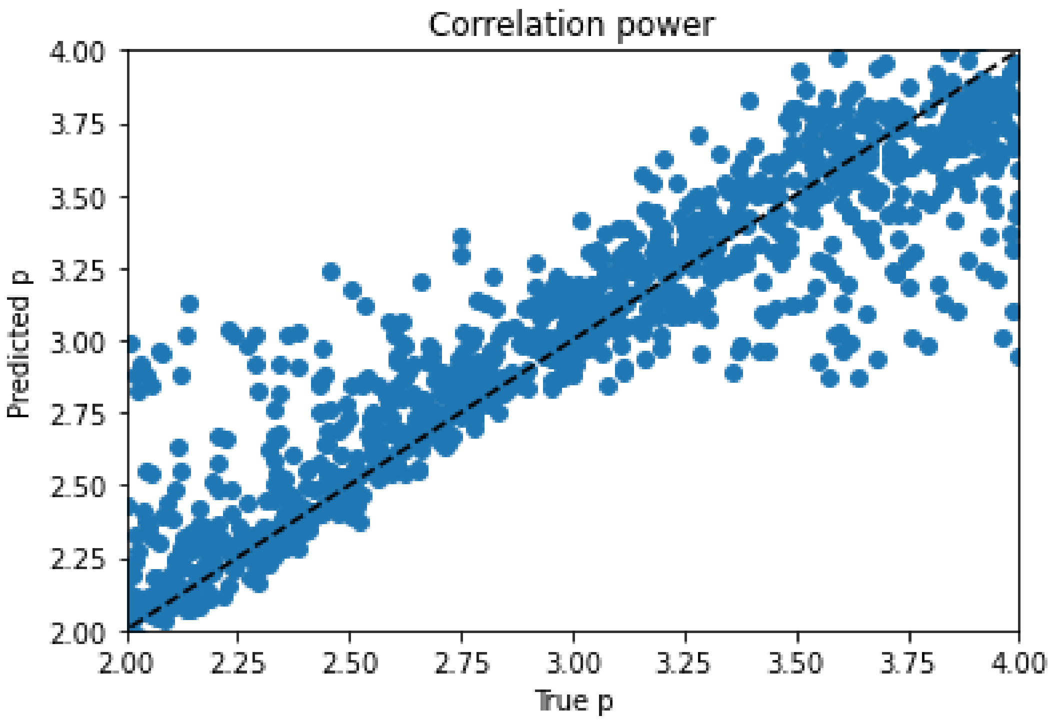
```
Training score for model  0.13344670118089058
Test score for model  0.1389222410674627
```

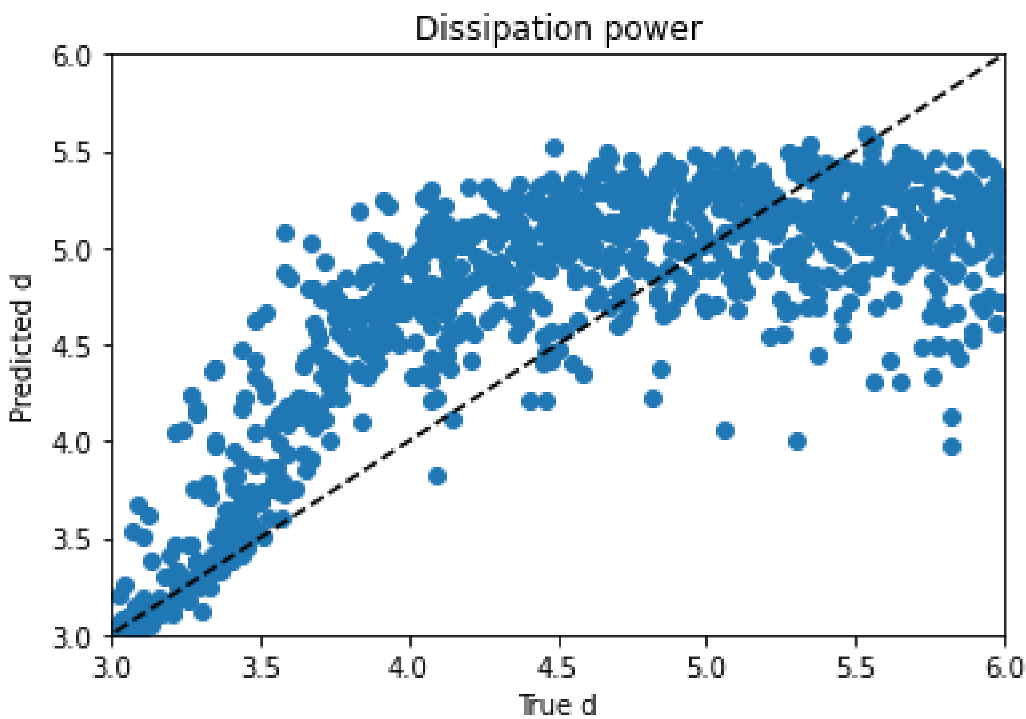
Let's plot that.

[28]:
```
plt.scatter(mat_test[:,0],mat7_predict[:,0]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True alpha");
plt.ylabel("Predicted alpha");
plt.axis([0, .2, 0, .2])
plt.title("Correlation strength")

plt.figure()
plt.scatter(mat_test[:,1],mat7_predict[:,1]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True xi");
plt.ylabel("Predicted xi");
plt.axis([0, 20, 0, 20])
plt.title("Correlation length")

plt.figure()
plt.scatter(mat_test[:,2],mat7_predict[:,2]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True p");
plt.ylabel("Predicted p");
plt.axis([2, 4, 2, 4])
plt.title("Correlation power")

plt.figure()
plt.scatter(mat_test[:,3],mat7_predict[:,3]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True d");
plt.ylabel("Predicted d");
plt.axis([3, 6, 3, 6]);
plt.title("Dissipation power");
```


Correlation strength
Predicted alpha
True alpha
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200


Correlation length
Predicted xi
True xi
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Correlation power
Predicted p
True p
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
2.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75
4.00


Dissipation power
Predicted d
True d
6.0
5.5
5.0
4.5
4.0
3.5
3.0
3.0
3.5
4.0
4.5
5.0
5.5
6.0

1.9 Model 8

You know what, let's just see if we can tune (a) model(s) specifically to deal with our problem child / children (yes, I'm looking at you, too, correlation length). They tell me that neural networks are universal function approximators; if I give each variable its own network I should get bloody somewhere.

Just building a new model specifically for d didn't do much, so I started playing with activation functions in the interior.

Post-competition strategy

Over the weekend I considered whether mean absolute error might result in a model that predicts the predictable part of the curve better and allows the unpredictable part to flare out.

[16]:
```
def build_model3(hp):
    inputs = Input(shape=(N,))
    x = Dense(
        units = hp.Choice('layer1',values=[64,256,1024]),
        activation='elu'
    )(inputs)
    y = Dense(
        units = hp.Choice('layer2',values=[16,64,256]),
        activation='elu'
    )(x)
    outputs = Dense(1)(y)
    model = Model(inputs, outputs)
    opt = keras.optimizers.SGD(learning_rate=0.01,momentum=0.95)
    model.
 ↪compile(loss='mean_absolute_error',optimizer=opt,metrics=['mean_absolute_error'])
    return model
```

[ ]:
```
opt = keras.optimizers.SGD(
        hp.Choice('learning_rate',
                  values=[0.01,0.005,0.001]),
        hp.Choice('momentum',
                  values=[0.67,0.9,0.95])
    )
```

[17]:
```
tuner3 = kerastuner.tuners.Hyperband(
        build_model3,
        objective='mean_absolute_error',
        max_epochs=100,
        executions_per_trial=2,
        directory='post_model_d'
)
```

```
INFO:tensorflow:Reloading Oracle from existing project
post_model_d/untitled_project/oracle.json
```

[18]:
```
scd = MinMaxScaler()
scd.fit(mat_info[:,3].reshape(-1,1))
```

[18]:
```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

[19]:
```
print('Starting Tuning')
tuner3.search(M_train,scd.transform(mat_train)[:,3])
print('Finished Tuning')
```

```
Trial 9 Complete [00h 00m 05s]
mean_absolute_error: 0.24674198031425476

Best mean_absolute_error So Far: 0.24350889027118683
Total elapsed time: 00h 01m 27s
INFO:tensorflow:Oracle triggered exit
Finished Tuning
```

[20]:
```
tuner3.results_summary()
```

```
Results summary
Results in post_model_d/untitled_project
Showing 10 best trials
Objective(name='mean_absolute_error', direction='min')
Trial summary
Hyperparameters:
layer1: 1024
layer2: 64
tuner/epochs: 2
tuner/initial_epoch: 0
tuner/bracket: 4
tuner/round: 0
Score: 0.24350889027118683
Trial summary
Hyperparameters:
layer1: 1024
layer2: 16
tuner/epochs: 2
tuner/initial_epoch: 0
tuner/bracket: 4
tuner/round: 0
Score: 0.24407228827476501
Trial summary
Hyperparameters:
layer1: 256
layer2: 64
tuner/epochs: 2
tuner/initial_epoch: 0
tuner/bracket: 4
```

```
tuner/round: 0
Score: 0.24511373043060303
Trial summary
Hyperparameters:
layer1: 1024
layer2: 256
tuner/epochs: 2
tuner/initial_epoch: 0
tuner/bracket: 4
tuner/round: 0
Score: 0.24663963913917542
Trial summary
Hyperparameters:
layer1: 64
layer2: 16
tuner/epochs: 2
tuner/initial_epoch: 0
tuner/bracket: 4
tuner/round: 0
Score: 0.24669626355171204
Trial summary
Hyperparameters:
layer1: 64
layer2: 256
tuner/epochs: 2
tuner/initial_epoch: 0
tuner/bracket: 4
tuner/round: 0
Score: 0.24674198031425476
Trial summary
Hyperparameters:
layer1: 64
layer2: 64
tuner/epochs: 2
tuner/initial_epoch: 0
tuner/bracket: 4
tuner/round: 0
Score: 0.24689961969852448
Trial summary
Hyperparameters:
layer1: 256
layer2: 256
tuner/epochs: 2
tuner/initial_epoch: 0
tuner/bracket: 4
tuner/round: 0
Score: 0.2471635490655899
Trial summary
```

```
Hyperparameters:
layer1: 256
layer2: 16
tuner/epochs: 2
tuner/initial_epoch: 0
tuner/bracket: 4
tuner/round: 0
Score: 0.24867098033428192
```

[22]:
```python
model8 = tuner3.get_best_models(num_models=5)[4]
```

```
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.iter
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.decay
WARNING:tensorflow:Unresolved object in checkpoint:
(root).optimizer.learning_rate
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.momentum
WARNING:tensorflow:A checkpoint was restored (e.g. tf.train.Checkpoint.restore
or tf.keras.Model.load_weights) but not all checkpointed values were used. See
above for specific issues. Use expect_partial() on the load status object, e.g.
tf.train.Checkpoint.restore(…).expect_partial(), to silence these warnings, or
use assert_consumed() to make the check explicit. See
https://www.tensorflow.org/guide/checkpoint#loading_mechanics for details.
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.iter
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.decay
WARNING:tensorflow:Unresolved object in checkpoint:
(root).optimizer.learning_rate
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.momentum
WARNING:tensorflow:A checkpoint was restored (e.g. tf.train.Checkpoint.restore
or tf.keras.Model.load_weights) but not all checkpointed values were used. See
above for specific issues. Use expect_partial() on the load status object, e.g.
tf.train.Checkpoint.restore(…).expect_partial(), to silence these warnings, or
use assert_consumed() to make the check explicit. See
https://www.tensorflow.org/guide/checkpoint#loading_mechanics for details.
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.iter
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.decay
WARNING:tensorflow:Unresolved object in checkpoint:
(root).optimizer.learning_rate
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.momentum
WARNING:tensorflow:A checkpoint was restored (e.g. tf.train.Checkpoint.restore
or tf.keras.Model.load_weights) but not all checkpointed values were used. See
above for specific issues. Use expect_partial() on the load status object, e.g.
tf.train.Checkpoint.restore(…).expect_partial(), to silence these warnings, or
use assert_consumed() to make the check explicit. See
https://www.tensorflow.org/guide/checkpoint#loading_mechanics for details.
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.iter
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.decay
WARNING:tensorflow:Unresolved object in checkpoint:
(root).optimizer.learning_rate
```

```
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.momentum
WARNING:tensorflow:A checkpoint was restored (e.g. tf.train.Checkpoint.restore
or tf.keras.Model.load_weights) but not all checkpointed values were used. See
above for specific issues. Use expect_partial() on the load status object, e.g.
tf.train.Checkpoint.restore(…).expect_partial(), to silence these warnings, or
use assert_consumed() to make the check explicit. See
https://www.tensorflow.org/guide/checkpoint#loading_mechanics for details.
```

[23]:
```
model8.summary()
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 942)]             0
_________________________________________________________________
dense (Dense)                (None, 64)                60352
_________________________________________________________________
dense_1 (Dense)              (None, 16)                1040
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 17
=================================================================
Total params: 61,409
Trainable params: 61,409
Non-trainable params: 0
_________________________________________________________________
```

[24]:
```
# we can make it stronger... we have the technology
# we are going to PUMP you UP
n_epochs = 40
n_batch = 10
# transform the target to scale features to [0,1]...not sure how this will␣
↪affect d
print('Starting Training')
model8.fit(M_train,scd.transform(mat_train[:,3].
↪reshape(-1,1)),epochs=n_epochs,batch_size=n_batch)
print('Finished Training')
```

```
Starting Training
Epoch 1/40
900/900 [==============================] - 2s 2ms/step - loss: 0.2465 -
mean_absolute_error: 0.2465
Epoch 2/40
900/900 [==============================] - 2s 2ms/step - loss: 0.2409 -
mean_absolute_error: 0.2409
Epoch 3/40
900/900 [==============================] - 2s 2ms/step - loss: 0.2306 -
mean_absolute_error: 0.2306
```

```
Epoch 4/40
900/900 [==============================] - 2s 2ms/step - loss: 0.2235 -
mean_absolute_error: 0.2235
Epoch 5/40
900/900 [==============================] - 2s 2ms/step - loss: 0.2197 -
mean_absolute_error: 0.2197
Epoch 6/40
900/900 [==============================] - 2s 2ms/step - loss: 0.2206 -
mean_absolute_error: 0.2206
Epoch 7/40
900/900 [==============================] - 2s 2ms/step - loss: 0.2080 -
mean_absolute_error: 0.2080
Epoch 8/40
900/900 [==============================] - 2s 2ms/step - loss: 0.2106 -
mean_absolute_error: 0.2106
Epoch 9/40
900/900 [==============================] - 2s 2ms/step - loss: 0.2061 -
mean_absolute_error: 0.2061
Epoch 10/40
900/900 [==============================] - 2s 2ms/step - loss: 0.2099 -
mean_absolute_error: 0.2099
Epoch 11/40
900/900 [==============================] - 2s 2ms/step - loss: 0.2039 -
mean_absolute_error: 0.2039
Epoch 12/40
900/900 [==============================] - 2s 2ms/step - loss: 0.2043 -
mean_absolute_error: 0.2043
Epoch 13/40
900/900 [==============================] - 2s 2ms/step - loss: 0.2039 -
mean_absolute_error: 0.2039
Epoch 14/40
900/900 [==============================] - 2s 2ms/step - loss: 0.2033 -
mean_absolute_error: 0.2033
Epoch 15/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1981 -
mean_absolute_error: 0.1981
Epoch 16/40
900/900 [==============================] - 2s 2ms/step - loss: 0.2035 -
mean_absolute_error: 0.2035
Epoch 17/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1977 -
mean_absolute_error: 0.1977
Epoch 18/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1991 -
mean_absolute_error: 0.1991
Epoch 19/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1950 -
mean_absolute_error: 0.1950
```

```
Epoch 20/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1966 -
mean_absolute_error: 0.1966
Epoch 21/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1908 -
mean_absolute_error: 0.1908
Epoch 22/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1945 -
mean_absolute_error: 0.1945
Epoch 23/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1879 -
mean_absolute_error: 0.1879
Epoch 24/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1898 -
mean_absolute_error: 0.1898
Epoch 25/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1948 -
mean_absolute_error: 0.1948
Epoch 26/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1876 -
mean_absolute_error: 0.1876
Epoch 27/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1882 -
mean_absolute_error: 0.1882
Epoch 28/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1922 -
mean_absolute_error: 0.1922
Epoch 29/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1961 -
mean_absolute_error: 0.1961
Epoch 30/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1911 -
mean_absolute_error: 0.1911
Epoch 31/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1855 -
mean_absolute_error: 0.1855
Epoch 32/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1848 -
mean_absolute_error: 0.1848
Epoch 33/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1856 -
mean_absolute_error: 0.1856
Epoch 34/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1860 -
mean_absolute_error: 0.1860
Epoch 35/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1810 -
mean_absolute_error: 0.1810
```

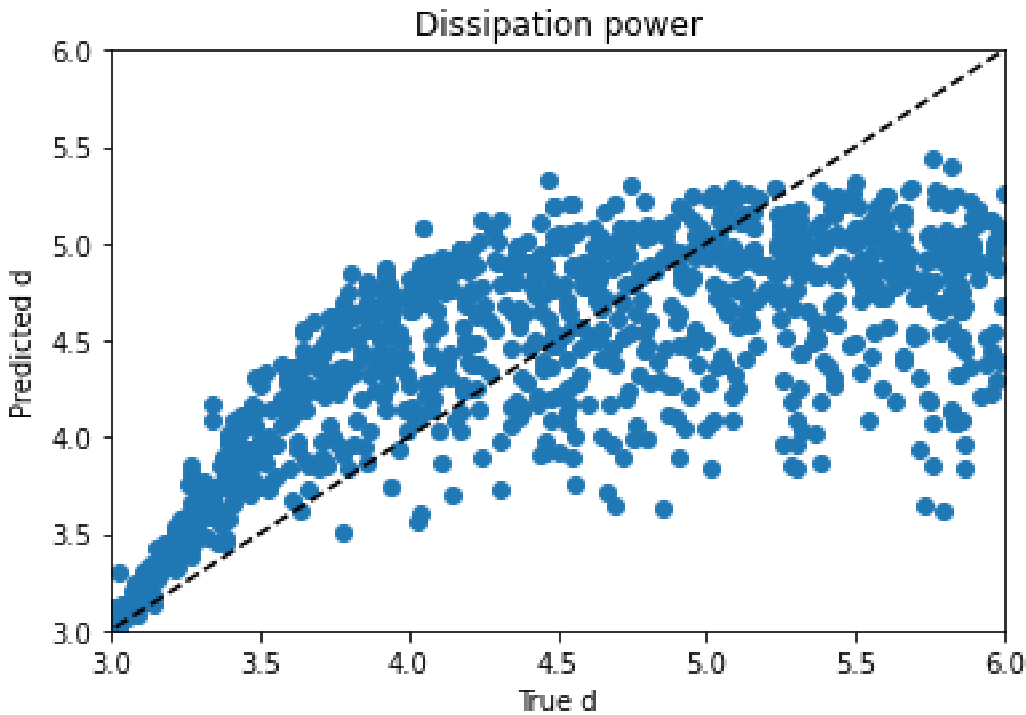
```
Epoch 36/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1852 -
mean_absolute_error: 0.1852
Epoch 37/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1817 -
mean_absolute_error: 0.1817
Epoch 38/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1823 -
mean_absolute_error: 0.1823
Epoch 39/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1781 -
mean_absolute_error: 0.1781
Epoch 40/40
900/900 [==============================] - 2s 2ms/step - loss: 0.1819 -
mean_absolute_error: 0.1819
Finished Training
```

[25]:
```
mat8_train_predict = scd.inverse_transform(model8.predict(M_train))
mat8_predict = scd.inverse_transform(model8.predict(M_test))
print('Training score for model ',sklearn.metrics.mean_squared_error(mat_train[:
 ↪,3],mat8_train_predict)/9)
print('Test score for model ',sklearn.metrics.mean_squared_error(mat_test[:
 ↪,3],mat8_predict)/9)
```

```
Training score for model  0.04498311937029458
Test score for model  0.04428550541659765
```

Let's plot that.

[26]:
```
plt.figure()
plt.scatter(mat_test[:,3],mat8_predict);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True d");
plt.ylabel("Predicted d");
plt.axis([3, 6, 3, 6]);
plt.title("Dissipation power");
```



I can't believe this. What is forcing these models to behave this way that isn't addressed by the stuff I've tried? Basically it looks like the behavior below $d = 4$ can be predicted and the behavior above cannot.

1.10 Model 9

In which I draw sword against ξ.

[58]:
```
def build_model4(hp):
    inputs = Input(shape=(N,))
    x = Dense(
        units = hp.Choice('layer1',values=[64,256,1024]),
        activation='elu'
    )(inputs)
    y = Dense(
        units = hp.Choice('layer2',values=[16,64,256]),
        activation='elu'
    )(x)
    outputs = Dense(1)(y)
    model = Model(inputs, outputs)
    opt = keras.optimizers.SGD(learning_rate=0.01,momentum=0.95)
    model.
 ↪compile(loss='mean_squared_error',optimizer=opt,metrics=['mean_squared_error'])
    return model
```

[59]:
```
tuner4 = kerastuner.tuners.Hyperband(
        build_model4,
        objective='mean_squared_error',
        max_epochs=100,
        executions_per_trial=2,
        directory='keras_tune4'
)
```

[60]:
```
scxi = MinMaxScaler()
scxi.fit(mat_info[:,1].reshape(-1,1))
```

[60]:
```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

[61]:
```
print('Starting Tuning')
tuner4.search(M_train,scxi.transform(mat_train)[:,1])
print('Finished Tuning')
```

```
Trial 9 Complete [00h 00m 15s]
mean_squared_error: 0.08622241765260696

Best mean_squared_error So Far: 0.08592872321605682
Total elapsed time: 00h 01m 19s
INFO:tensorflow:Oracle triggered exit
Finished Tuning
```

[62]:
```
tuner4.results_summary()
```

```
Results summary
Results in keras_tune4/untitled_project
Showing 10 best trials
Objective(name='mean_squared_error', direction='min')
Trial summary
Hyperparameters:
layer1: 64
layer2: 256
tuner/epochs: 2
tuner/initial_epoch: 0
tuner/bracket: 4
tuner/round: 0
Score: 0.08592872321605682
Trial summary
Hyperparameters:
layer1: 1024
layer2: 256
tuner/epochs: 2
tuner/initial_epoch: 0
tuner/bracket: 4
tuner/round: 0
```

```
Score: 0.08620576187968254
Trial summary
Hyperparameters:
layer1: 1024
layer2: 64
tuner/epochs: 2
tuner/initial_epoch: 0
tuner/bracket: 4
tuner/round: 0
Score: 0.08622241765260696
Trial summary
Hyperparameters:
layer1: 256
layer2: 256
tuner/epochs: 2
tuner/initial_epoch: 0
tuner/bracket: 4
tuner/round: 0
Score: 0.08670295402407646
Trial summary
Hyperparameters:
layer1: 256
layer2: 64
tuner/epochs: 2
tuner/initial_epoch: 0
tuner/bracket: 4
tuner/round: 0
Score: 0.08680764585733414
Trial summary
Hyperparameters:
layer1: 64
layer2: 64
tuner/epochs: 2
tuner/initial_epoch: 0
tuner/bracket: 4
tuner/round: 0
Score: 0.08689809590578079
Trial summary
Hyperparameters:
layer1: 1024
layer2: 16
tuner/epochs: 2
tuner/initial_epoch: 0
tuner/bracket: 4
tuner/round: 0
Score: 0.08716437965631485
Trial summary
Hyperparameters:
```

```
layer1: 64
layer2: 16
tuner/epochs: 2
tuner/initial_epoch: 0
tuner/bracket: 4
tuner/round: 0
Score: 0.08743632212281227
Trial summary
Hyperparameters:
layer1: 256
layer2: 16
tuner/epochs: 2
tuner/initial_epoch: 0
tuner/bracket: 4
tuner/round: 0
Score: 0.08749530836939812
```

[63]:
```
model9 = tuner4.get_best_models(num_models=1)[0]
```

[64]:
```
model9.summary()
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 900)]             0
_________________________________________________________________
dense (Dense)                (None, 64)                57664
_________________________________________________________________
dense_1 (Dense)              (None, 256)               16640
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 74,561
Trainable params: 74,561
Non-trainable params: 0
_________________________________________________________________
```

[65]:
```
n_epochs = 40
n_batch = 10
# transform the target to scale features to [0,1]
print('Starting Training')
model8.fit(M_train,scxi.transform(mat_train[:,1].
 ↪reshape(-1,1)),epochs=n_epochs,batch_size=n_batch)
print('Finished Training')
```

```
Starting Training
Epoch 1/40
```

```
900/900 [==============================] - 2s 3ms/step - loss: 0.0887 -
mean_squared_error: 0.0887
Epoch 2/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0882 -
mean_squared_error: 0.0882
Epoch 3/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0874 -
mean_squared_error: 0.0874
Epoch 4/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0875 -
mean_squared_error: 0.0875
Epoch 5/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0874 -
mean_squared_error: 0.0874
Epoch 6/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0872 -
mean_squared_error: 0.0872
Epoch 7/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0875 -
mean_squared_error: 0.0875
Epoch 8/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0867 -
mean_squared_error: 0.0867
Epoch 9/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0878 -
mean_squared_error: 0.0878
Epoch 10/40
900/900 [==============================] - 3s 3ms/step - loss: 0.0877 -
mean_squared_error: 0.0877
Epoch 11/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0869 -
mean_squared_error: 0.0869
Epoch 12/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0868 -
mean_squared_error: 0.0868
Epoch 13/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0873 -
mean_squared_error: 0.0873
Epoch 14/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0875 -
mean_squared_error: 0.0875
Epoch 15/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0865 -
mean_squared_error: 0.0865
Epoch 16/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0870 -
mean_squared_error: 0.0870
Epoch 17/40
```

```
900/900 [==============================] - 2s 3ms/step - loss: 0.0873 -
mean_squared_error: 0.0873
Epoch 18/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0867 -
mean_squared_error: 0.0867
Epoch 19/40
900/900 [==============================] - 3s 3ms/step - loss: 0.0866 -
mean_squared_error: 0.0866
Epoch 20/40
900/900 [==============================] - 3s 3ms/step - loss: 0.0871 -
mean_squared_error: 0.0871
Epoch 21/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0870 -
mean_squared_error: 0.0870
Epoch 22/40
900/900 [==============================] - 3s 3ms/step - loss: 0.0876 -
mean_squared_error: 0.0876
Epoch 23/40
900/900 [==============================] - 3s 3ms/step - loss: 0.0879 -
mean_squared_error: 0.0879
Epoch 24/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0872 -
mean_squared_error: 0.0872
Epoch 25/40
900/900 [==============================] - 3s 3ms/step - loss: 0.0872 -
mean_squared_error: 0.0872
Epoch 26/40
900/900 [==============================] - 3s 3ms/step - loss: 0.0871 -
mean_squared_error: 0.0871
Epoch 27/40
900/900 [==============================] - 3s 3ms/step - loss: 0.0871 -
mean_squared_error: 0.0871
Epoch 28/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0867 -
mean_squared_error: 0.0867
Epoch 29/40
900/900 [==============================] - 3s 3ms/step - loss: 0.0870 -
mean_squared_error: 0.0870
Epoch 30/40
900/900 [==============================] - 3s 3ms/step - loss: 0.0875 -
mean_squared_error: 0.0875
Epoch 31/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0870 -
mean_squared_error: 0.0870
Epoch 32/40
900/900 [==============================] - 3s 3ms/step - loss: 0.0877 -
mean_squared_error: 0.0877
Epoch 33/40
```

```
900/900 [==============================] - 3s 3ms/step - loss: 0.0871 -
mean_squared_error: 0.0871
Epoch 34/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0865 -
mean_squared_error: 0.0865
Epoch 35/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0868 -
mean_squared_error: 0.0868
Epoch 36/40
900/900 [==============================] - 3s 3ms/step - loss: 0.0872 -
mean_squared_error: 0.0872
Epoch 37/40
900/900 [==============================] - 3s 3ms/step - loss: 0.0873 -
mean_squared_error: 0.0873
Epoch 38/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0872 -
mean_squared_error: 0.0872
Epoch 39/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0870 -
mean_squared_error: 0.0870
Epoch 40/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0868 -
mean_squared_error: 0.0868
Finished Training
```

[68]:

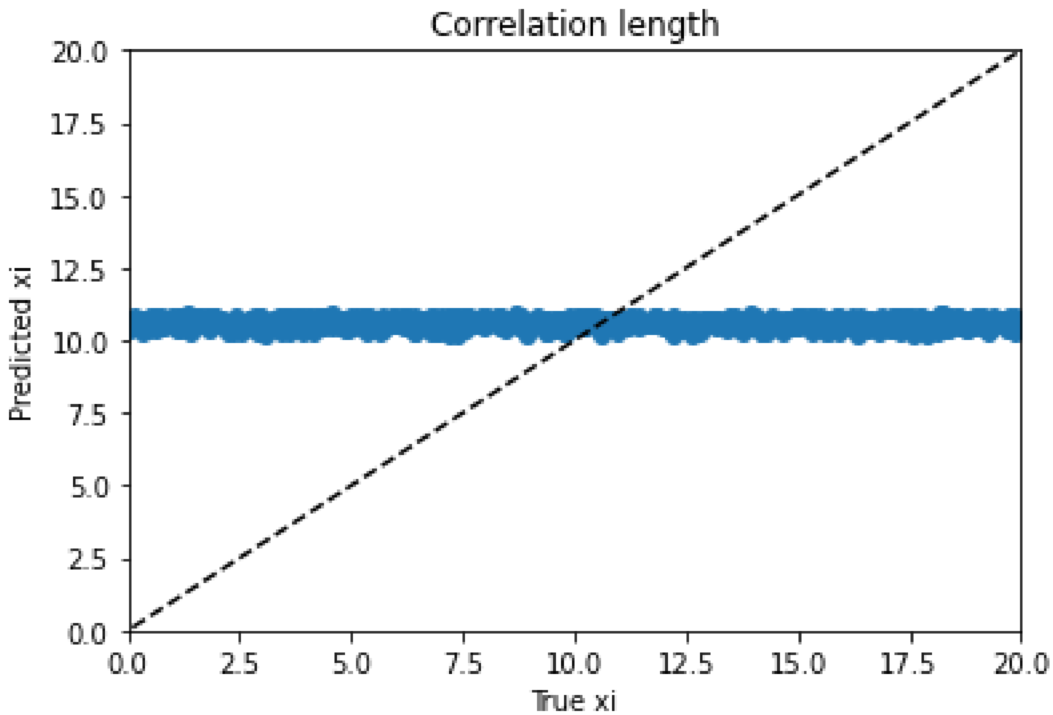
```
mat9_train_predict = scxi.inverse_transform(model9.predict(M_train))
mat9_predict = scxi.inverse_transform(model9.predict(M_test))
print('Training score for model ',sklearn.metrics.mean_squared_error(mat_train[:
 ↪,1],mat9_train_predict)/400)
print('Test score for model ',sklearn.metrics.mean_squared_error(mat_test[:
 ↪,1],mat9_predict)/400)
```

```
Training score for model  0.08570692378965893
Test score for model  0.0857066459535306
```

Let's plot that.

[69]:

```
plt.figure()
plt.scatter(mat_test[:,1],mat9_predict);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True xi");
plt.ylabel("Predicted xi");
plt.axis([0, 20, 0, 20]);
plt.title("Correlation length");
```



Training noise. It's a peck of fun.

So what else am I going to try in this last couple of hours?

Thinking: d is related to the time constant $T_d = k10^{-d/2}$. Once d gets above 4, there is barely any change in the plots and therefore barely any information in them that the neural nets can extract to predict higher values of d… that's where I stand conceptually at the moment, looking at the data. Increases in d are decreases in T_d. Are the low d cases the ones with significant residual magnetic activity at time τ and the high d ones the "standard" cases where the spins have spread out and the 180 pulse starts them back toward reassembling into the echo?

In that case, given that I only have two hours to tie this up and send you something, I could do something rather rash: train on just the entries with $d <= 4$.

The other possibility is to go back to my crude hack model 6 tactic where I bruted a log transform onto the data and apply that to a model with separated paths for d and the other variables. Maybe I also see what happens if I push ξ off into its own internal tree. I tried to apply a log function as an activation for d in model 7, but the loss functions could not be evaluated… bizarrely, the model tried to train itself, but the losses were all nan and the model could not be evaluated afterward.

1.11 Model 10

[71]:
```
def build_model10(hp):
    inputs = Input(shape=(N,))
    x = Dense(
        units = hp.Choice('layer1',values=[16,64,256]),
```

```
        activation='elu'
    )(inputs)
    yd = Dense(
        units = hp.Choice('layer2d',values=[8,32,128]),
        activation='elu'
    )(x)
    yxi = Dense(
        units = 4,
        activation='elu'
    )(x)
    yap = Dense(
        units = hp.Choice('layer2ap',values=[16,64,256]),
        activation='elu'
    )(x)
    zd = Dense(
        units = hp.Choice('layer3d',values=[4,32]),
        activation='elu'
    )(yd)
    zxi = Dense(
        units = 2,
        activation='elu'
    )(yxi)
    zap = Dense(
        units = hp.Choice('layer3ap',values=[8,64]),
        activation='elu'
    )(yap)
    outd = Dense(1)(zd)
    outxi = Dense(1)(zxi)
    outa = Dense(1)(zap)
    outp = Dense(1)(zap)
    outputs = Concatenate()([outa,outxi,outp,outd])
    model = Model(inputs, outputs)
    opt = keras.optimizers.SGD(learning_rate=0.01,momentum=0.95)
    model.
 ↪compile(loss='mean_squared_error',optimizer=opt,metrics=['mean_squared_error'])
    return model
```

[73]:
```
tuner10 = kerastuner.tuners.Hyperband(
        build_model10,
        objective='mean_squared_error',
        max_epochs=33,
        executions_per_trial=2,
        directory='keras_tune10'
)
```

I mistakenly used `scaler` instead of `trscaler` in the next cell. I will not have time to go back and rerun the tuning, but hopefully the model structure it discovers will work.

[82]:
```
print('Starting Tuning')
tuner10.search(M_train,scaler.transform(mat_train_tr))
print('Finished Tuning')
```

```
Trial 90 Complete [00h 01m 06s]
mean_squared_error: 0.030391693115234375

Best mean_squared_error So Far: 0.029421127401292324
Total elapsed time: 00h 30m 13s
INFO:tensorflow:Oracle triggered exit
Finished Tuning
```

[83]:
```
tuner10.results_summary()
```

```
Results summary
Results in keras_tune10/untitled_project
Showing 10 best trials
Objective(name='mean_squared_error', direction='min')
Trial summary
Hyperparameters:
layer1: 64
layer2d: 32
layer2ap: 16
layer3d: 4
layer3ap: 64
tuner/epochs: 33
tuner/initial_epoch: 0
tuner/bracket: 0
tuner/round: 0
Score: 0.029421127401292324
Trial summary
Hyperparameters:
layer1: 256
layer2d: 32
layer2ap: 256
layer3d: 32
layer3ap: 8
tuner/epochs: 33
tuner/initial_epoch: 11
tuner/bracket: 3
tuner/round: 3
tuner/trial_id: eff667eff5fff4be28e80c6686a3b2a3
Score: 0.029459443874657154
Trial summary
Hyperparameters:
layer1: 256
layer2d: 8
layer2ap: 64
```

```
layer3d: 32
layer3ap: 8
tuner/epochs: 33
tuner/initial_epoch: 11
tuner/bracket: 2
tuner/round: 2
tuner/trial_id: 83edd46611e2c066b074d6f79b5a74dc
Score: 0.029687143862247467
Trial summary
Hyperparameters:
layer1: 256
layer2d: 128
layer2ap: 16
layer3d: 4
layer3ap: 8
tuner/epochs: 33
tuner/initial_epoch: 11
tuner/bracket: 2
tuner/round: 2
tuner/trial_id: a28031d50c02a34b7b8eee21d88fe242
Score: 0.029981818050146103
Trial summary
Hyperparameters:
layer1: 64
layer2d: 32
layer2ap: 64
layer3d: 32
layer3ap: 64
tuner/epochs: 33
tuner/initial_epoch: 0
tuner/bracket: 0
tuner/round: 0
Score: 0.030391693115234375
Trial summary
Hyperparameters:
layer1: 64
layer2d: 128
layer2ap: 256
layer3d: 32
layer3ap: 8
tuner/epochs: 33
tuner/initial_epoch: 0
tuner/bracket: 0
tuner/round: 0
Score: 0.030469906516373158
Trial summary
Hyperparameters:
layer1: 256
```

```
layer2d: 32
layer2ap: 64
layer3d: 32
layer3ap: 64
tuner/epochs: 33
tuner/initial_epoch: 11
tuner/bracket: 1
tuner/round: 1
tuner/trial_id: 441b73793e465bea001d72f121955160
Score: 0.030869778245687485
Trial summary
Hyperparameters:
layer1: 16
layer2d: 128
layer2ap: 64
layer3d: 32
layer3ap: 8
tuner/epochs: 33
tuner/initial_epoch: 0
tuner/bracket: 0
tuner/round: 0
Score: 0.030882260762155056
Trial summary
Hyperparameters:
layer1: 256
layer2d: 128
layer2ap: 16
layer3d: 32
layer3ap: 8
tuner/epochs: 33
tuner/initial_epoch: 11
tuner/bracket: 1
tuner/round: 1
tuner/trial_id: 76c28804b1b307e37325338c135936f8
Score: 0.030923728831112385
Trial summary
Hyperparameters:
layer1: 16
layer2d: 8
layer2ap: 256
layer3d: 32
layer3ap: 8
tuner/epochs: 33
tuner/initial_epoch: 0
tuner/bracket: 0
tuner/round: 0
Score: 0.031109227798879147
```

[88]:
```
model10 = tuner10.get_best_models(num_models=1)[0]
model10.summary()
```

```
Model: "functional_1"
__________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
==================
input_1 (InputLayer)            [(None, 900)]        0

__________________________________________________________________________________
__________________
dense (Dense)                   (None, 64)           57664       input_1[0][0]

__________________________________________________________________________________
__________________
dense_3 (Dense)                 (None, 16)           1040        dense[0][0]

__________________________________________________________________________________
__________________
dense_2 (Dense)                 (None, 4)            260         dense[0][0]

__________________________________________________________________________________
__________________
dense_1 (Dense)                 (None, 32)           2080        dense[0][0]

__________________________________________________________________________________
__________________
dense_6 (Dense)                 (None, 64)           1088        dense_3[0][0]

__________________________________________________________________________________
__________________
dense_5 (Dense)                 (None, 2)            10          dense_2[0][0]

__________________________________________________________________________________
__________________
dense_4 (Dense)                 (None, 4)            132         dense_1[0][0]

__________________________________________________________________________________
__________________
dense_9 (Dense)                 (None, 1)            65          dense_6[0][0]

__________________________________________________________________________________
__________________
dense_8 (Dense)                 (None, 1)            3           dense_5[0][0]

__________________________________________________________________________________
__________________
dense_10 (Dense)                (None, 1)            65          dense_6[0][0]

__________________________________________________________________________________
__________________
dense_7 (Dense)                 (None, 1)            5           dense_4[0][0]

__________________________________________________________________________________
__________________
concatenate (Concatenate)       (None, 4)            0           dense_9[0][0]
                                                                 dense_8[0][0]
```

```
                                                                 dense_10[0][0]
                                                                 dense_7[0][0]
================================================================================
==================
Total params: 62,412
Trainable params: 62,412
Non-trainable params: 0
--------------------------------------------------------------------------------
------------------
```

[92]:

```
# we can make it stronger... we have the technology
# we are going to PUMP you UP
n_epochs = 40
n_batch = 10
# transform the target to scale features to [0,1]...with the mistake that I used
# scaler instead of trscaler above, probably no time to fix it now, but maybe,␣
↪we'll see
print('Starting Training')
model10.fit(M_train,scaler.
↪transform(mat_train_tr),epochs=n_epochs,batch_size=n_batch)
print('Finished Training')
```

```
Starting Training
Epoch 1/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0306 -
mean_squared_error: 0.0306
Epoch 2/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0302 -
mean_squared_error: 0.0302
Epoch 3/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0301 -
mean_squared_error: 0.0301
Epoch 4/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0296 -
mean_squared_error: 0.0296
Epoch 5/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0295 -
mean_squared_error: 0.0295
Epoch 6/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0294 -
mean_squared_error: 0.0294
Epoch 7/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0288 -
mean_squared_error: 0.0288
Epoch 8/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0294 -
mean_squared_error: 0.0294
Epoch 9/40
```

```
900/900 [==============================] - 2s 2ms/step - loss: 0.0292 -
mean_squared_error: 0.0292
Epoch 10/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0288 -
mean_squared_error: 0.0288
Epoch 11/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0287 -
mean_squared_error: 0.0287
Epoch 12/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0287 -
mean_squared_error: 0.0287
Epoch 13/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0286 -
mean_squared_error: 0.0286
Epoch 14/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0282 -
mean_squared_error: 0.0282
Epoch 15/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0283 -
mean_squared_error: 0.0283
Epoch 16/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0284 -
mean_squared_error: 0.0284
Epoch 17/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0281 -
mean_squared_error: 0.0281
Epoch 18/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0281 -
mean_squared_error: 0.0281
Epoch 19/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0280 -
mean_squared_error: 0.0280
Epoch 20/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0280 -
mean_squared_error: 0.0280
Epoch 21/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0280 -
mean_squared_error: 0.0280
Epoch 22/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0277 -
mean_squared_error: 0.0277
Epoch 23/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0278 -
mean_squared_error: 0.0278
Epoch 24/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0274 -
mean_squared_error: 0.0274
Epoch 25/40
```

```
900/900 [==============================] - 3s 3ms/step - loss: 0.0274 -
mean_squared_error: 0.0274
Epoch 26/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0278 -
mean_squared_error: 0.0278
Epoch 27/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0277 -
mean_squared_error: 0.0277
Epoch 28/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0274 -
mean_squared_error: 0.0274
Epoch 29/40
900/900 [==============================] - ETA: 0s - loss: 0.0274 -
mean_squared_error: 0.02 - 2s 2ms/step - loss: 0.0274 - mean_squared_error:
0.0274
Epoch 30/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0274 -
mean_squared_error: 0.0274
Epoch 31/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0274 -
mean_squared_error: 0.0274
Epoch 32/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0275 -
mean_squared_error: 0.0275
Epoch 33/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0272 -
mean_squared_error: 0.0272
Epoch 34/40
900/900 [==============================] - 3s 3ms/step - loss: 0.0272 -
mean_squared_error: 0.0272
Epoch 35/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0268 -
mean_squared_error: 0.0268
Epoch 36/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0272 -
mean_squared_error: 0.0272
Epoch 37/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0270 -
mean_squared_error: 0.0270
Epoch 38/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0268 -
mean_squared_error: 0.0268
Epoch 39/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0271 -
mean_squared_error: 0.0271
Epoch 40/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0270 -
mean_squared_error: 0.0270
```

```
Finished Training
```

[93]:

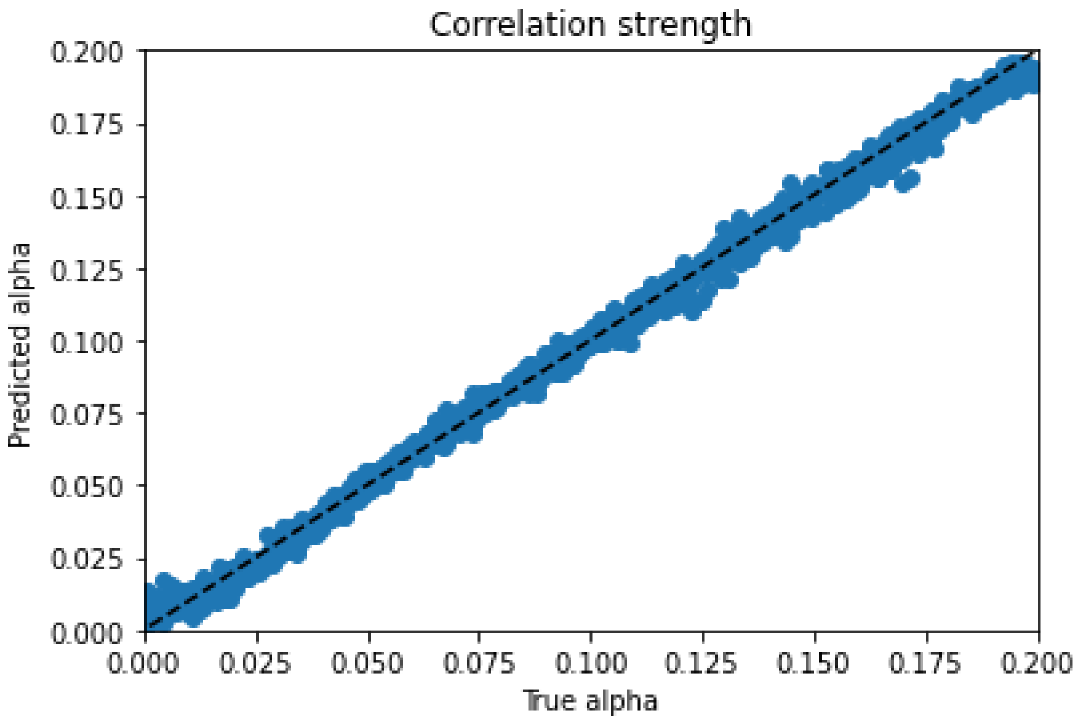
```
mat10_train_predict = scaler.inverse_transform(model10.
 ↪predict(M_train,batch_size=n_batch))
mat10_train_predict[:,3]=np.exp(mat10_train_predict[:,3])
mat10_predict = scaler.inverse_transform(model10.
 ↪predict(M_test,batch_size=n_batch))
mat10_predict[:,3]=np.exp(mat10_predict[:,3])
print('Training score for model ',weight_mse(mat_train,mat10_train_predict))
print('Test score for model ',weight_mse(mat_test,mat10_predict))
```

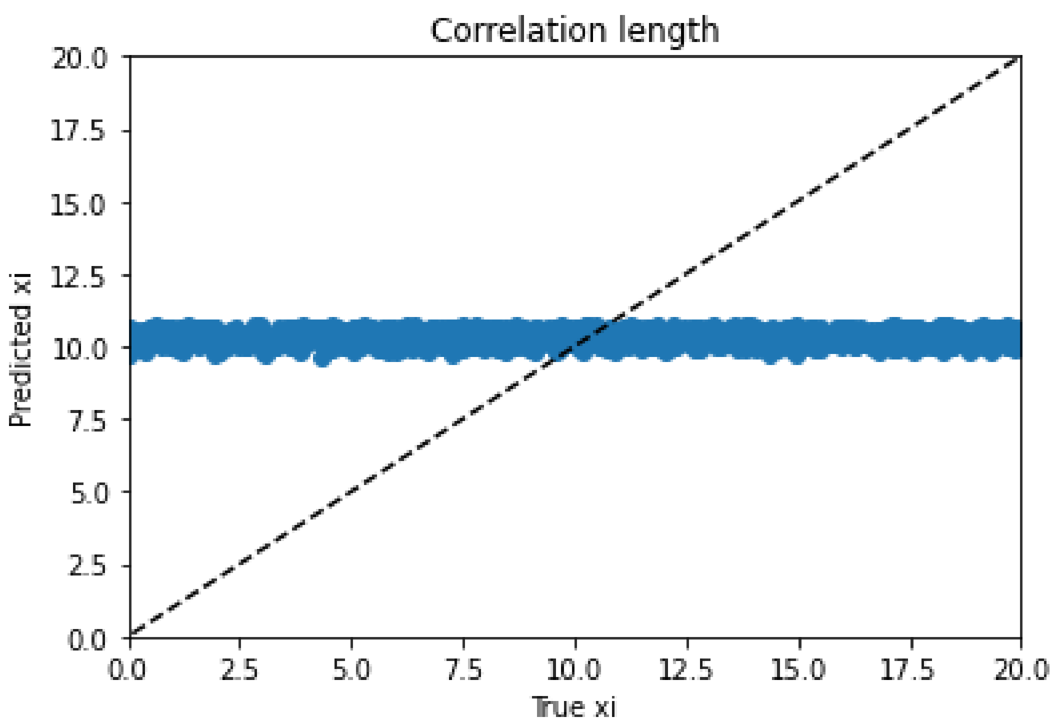
```
Training score for model  0.1518837705316063
Test score for model  0.15609987081435087
```

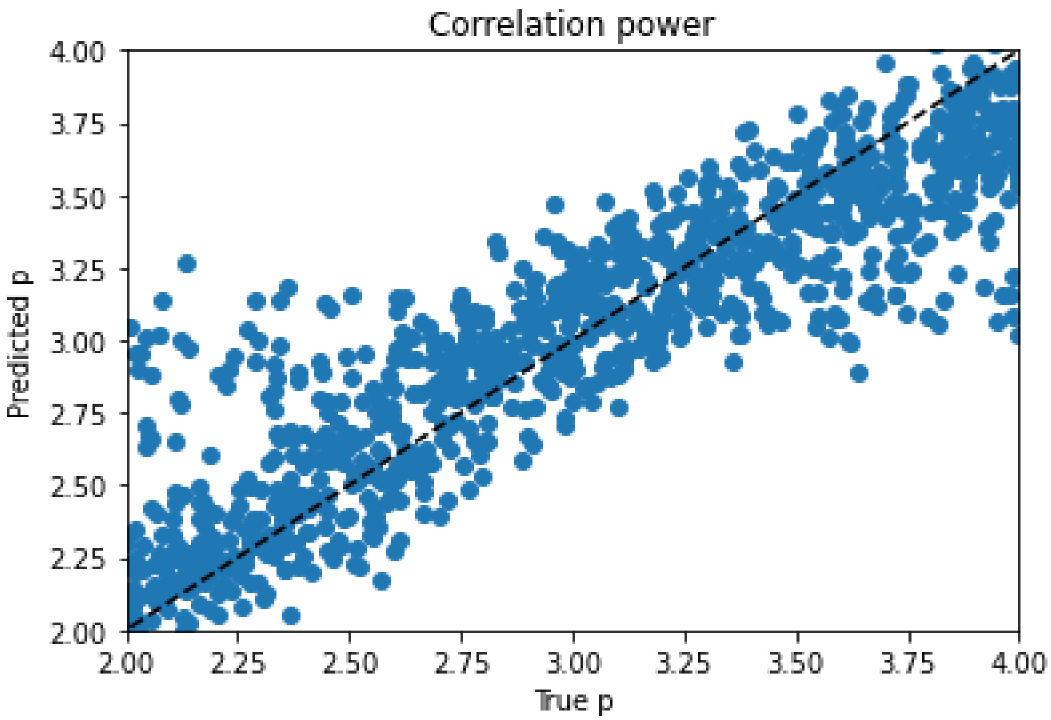
Let's plot that.

[94]:

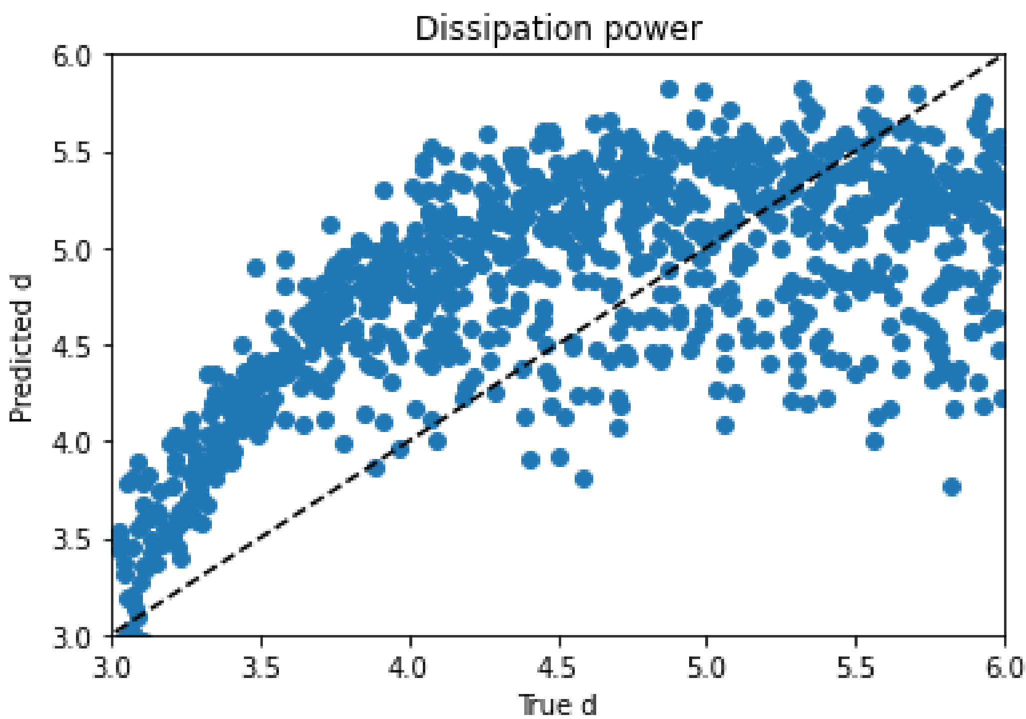
```
plt.scatter(mat_test[:,0],mat10_predict[:,0]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True alpha");
plt.ylabel("Predicted alpha");
plt.axis([0, .2, 0, .2])
plt.title("Correlation strength")

plt.figure()
plt.scatter(mat_test[:,1],mat10_predict[:,1]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True xi");
plt.ylabel("Predicted xi");
plt.axis([0, 20, 0, 20])
plt.title("Correlation length")

plt.figure()
plt.scatter(mat_test[:,2],mat10_predict[:,2]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True p");
plt.ylabel("Predicted p");
plt.axis([2, 4, 2, 4])
plt.title("Correlation power")

plt.figure()
plt.scatter(mat_test[:,3],mat10_predict[:,3]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True d");
plt.ylabel("Predicted d");
plt.axis([3, 6, 3, 6]);
plt.title("Dissipation power");
```


Correlation strength
Predicted alpha
True alpha
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200


Correlation length
Predicted xi
True xi
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



Correlation power
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
Predicted p
2.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75
4.00
True p



Dissipation power
6.0
5.5
5.0
4.5
4.0
3.5
3.0
Predicted d
3.0
3.5
4.0
4.5
5.0
5.5
6.0
True d

1.12 Model 12

in greater haste

[110]:
```
model12 = keras.models.clone_model(model10)
```

[111]:
```
model12.summary()
```

```
Model: "functional_1"
________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
==================
input_1 (InputLayer)            [(None, 900)]        0
________________________________________________________________________________
__________________
dense (Dense)                   (None, 64)           57664       input_1[0][0]
________________________________________________________________________________
__________________
dense_3 (Dense)                 (None, 16)           1040        dense[0][0]
________________________________________________________________________________
__________________
dense_2 (Dense)                 (None, 4)            260         dense[0][0]
________________________________________________________________________________
__________________
dense_1 (Dense)                 (None, 32)           2080        dense[0][0]
________________________________________________________________________________
__________________
dense_6 (Dense)                 (None, 64)           1088        dense_3[0][0]
________________________________________________________________________________
__________________
dense_5 (Dense)                 (None, 2)            10          dense_2[0][0]
________________________________________________________________________________
__________________
dense_4 (Dense)                 (None, 4)            132         dense_1[0][0]
________________________________________________________________________________
__________________
dense_9 (Dense)                 (None, 1)            65          dense_6[0][0]
________________________________________________________________________________
__________________
dense_8 (Dense)                 (None, 1)            3           dense_5[0][0]
________________________________________________________________________________
__________________
dense_10 (Dense)                (None, 1)            65          dense_6[0][0]
________________________________________________________________________________
__________________
dense_7 (Dense)                 (None, 1)            5           dense_4[0][0]
```

```
--------------------------------------------------------------------------------
------------------
concatenate (Concatenate)       (None, 4)            0           dense_9[0][0]
                                                                 dense_8[0][0]
                                                                 dense_10[0][0]
                                                                 dense_7[0][0]
================================================================================
==================
Total params: 62,412
Trainable params: 62,412
Non-trainable params: 0
--------------------------------------------------------------------------------
------------------
```

[112]:

```
opt = keras.optimizers.SGD(learning_rate=0.01,momentum=0.95)
model12.
 ↪compile(loss='mean_squared_error',optimizer=opt,metrics=['mean_squared_error'])
n_epochs = 40
n_batch = 10
# transform the target to scale features to [0,1]...not not effing with log␣
 ↪transform
# just using the bigger structure
print('Starting Training')
model12.fit(M_train,scaler.
 ↪transform(mat_train),epochs=n_epochs,batch_size=n_batch)
print('Finished Training')
```

```
Starting Training
Epoch 1/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0613 -
mean_squared_error: 0.0613
Epoch 2/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0533 -
mean_squared_error: 0.0533
Epoch 3/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0510 -
mean_squared_error: 0.0510
Epoch 4/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0503 -
mean_squared_error: 0.0503
Epoch 5/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0486 -
mean_squared_error: 0.0486
Epoch 6/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0479 -
mean_squared_error: 0.0479
Epoch 7/40
```

```
900/900 [==============================] - 2s 2ms/step - loss: 0.0473 -
mean_squared_error: 0.0473
Epoch 8/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0472 -
mean_squared_error: 0.0472
Epoch 9/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0467 -
mean_squared_error: 0.0467
Epoch 10/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0454 -
mean_squared_error: 0.0454
Epoch 11/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0453 -
mean_squared_error: 0.0453
Epoch 12/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0445 -
mean_squared_error: 0.0445
Epoch 13/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0445 -
mean_squared_error: 0.0445
Epoch 14/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0445 -
mean_squared_error: 0.0445
Epoch 15/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0432 -
mean_squared_error: 0.0432
Epoch 16/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0424 -
mean_squared_error: 0.0424
Epoch 17/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0420 -
mean_squared_error: 0.0420
Epoch 18/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0416 -
mean_squared_error: 0.0416
Epoch 19/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0412 -
mean_squared_error: 0.0412
Epoch 20/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0404 -
mean_squared_error: 0.0404
Epoch 21/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0410 -
mean_squared_error: 0.0410
Epoch 22/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0402 -
mean_squared_error: 0.0402
Epoch 23/40
```

```
900/900 [==============================] - 2s 2ms/step - loss: 0.0398 -
mean_squared_error: 0.0398
Epoch 24/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0395 -
mean_squared_error: 0.0395
Epoch 25/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0394 -
mean_squared_error: 0.0394
Epoch 26/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0395 -
mean_squared_error: 0.0395
Epoch 27/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0387 -
mean_squared_error: 0.0387
Epoch 28/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0387 -
mean_squared_error: 0.0387
Epoch 29/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0384 -
mean_squared_error: 0.0384
Epoch 30/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0381 -
mean_squared_error: 0.0381
Epoch 31/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0377 -
mean_squared_error: 0.0377
Epoch 32/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0379 -
mean_squared_error: 0.0379
Epoch 33/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0373 -
mean_squared_error: 0.0373
Epoch 34/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0375 -
mean_squared_error: 0.0375
Epoch 35/40
900/900 [==============================] - 2s 3ms/step - loss: 0.0373 -
mean_squared_error: 0.0373
Epoch 36/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0371 -
mean_squared_error: 0.0371
Epoch 37/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0375 -
mean_squared_error: 0.0375
Epoch 38/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0369 -
mean_squared_error: 0.0369
Epoch 39/40
```

```
900/900 [==============================] - 2s 2ms/step - loss: 0.0369 -
mean_squared_error: 0.0369
Epoch 40/40
900/900 [==============================] - 2s 2ms/step - loss: 0.0372 -
mean_squared_error: 0.0372
Finished Training
```

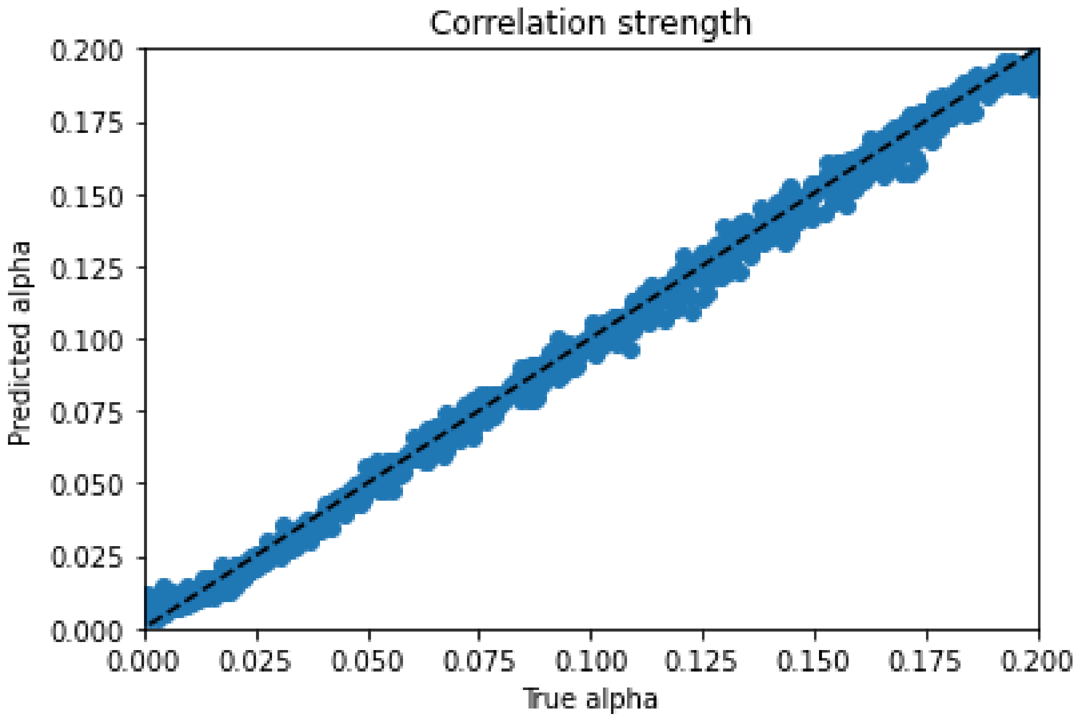
[113]:
```
mat12_train_predict = scaler.inverse_transform(model12.
 ↪predict(M_train,batch_size=n_batch))
mat12_predict = scaler.inverse_transform(model12.
 ↪predict(M_test,batch_size=n_batch))
print('Training score for model ',weight_mse(mat_train,mat12_train_predict))
print('Test score for model ',weight_mse(mat_test,mat12_predict))
```

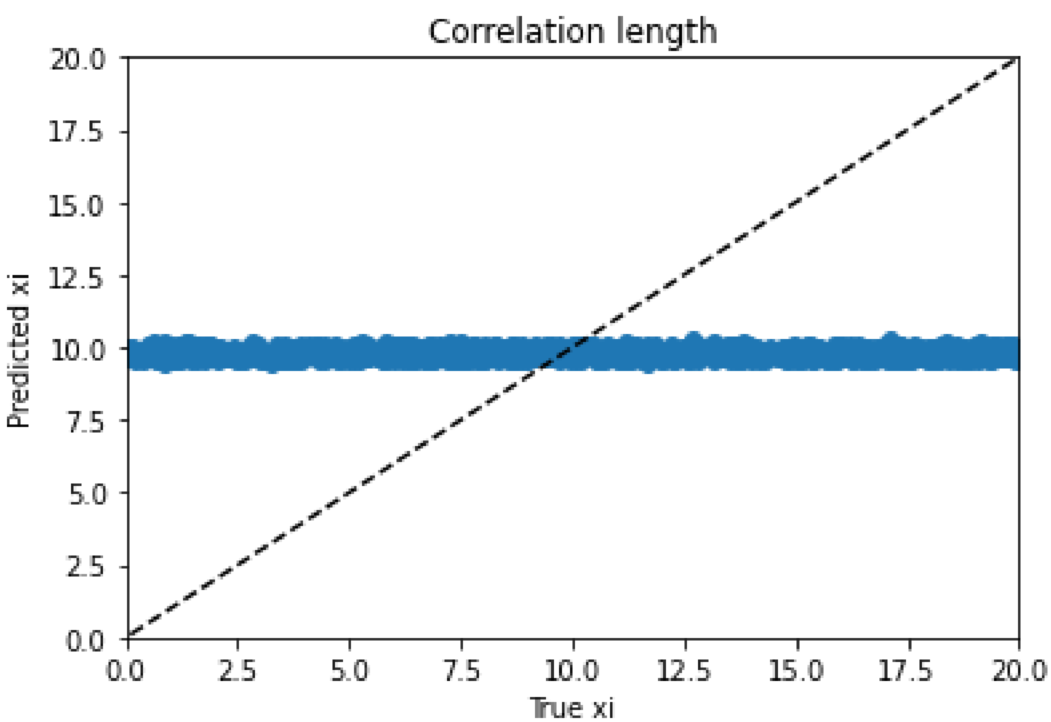
```
Training score for model  0.14520216390627058
Test score for model  0.149538385748348
```

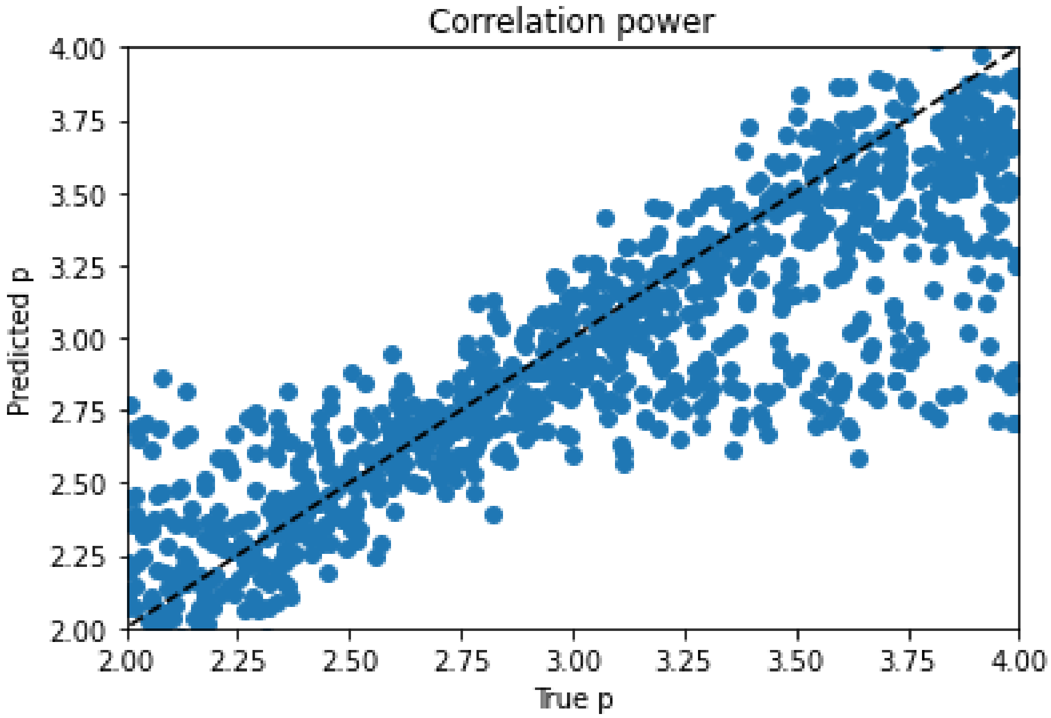
Let's plot that.

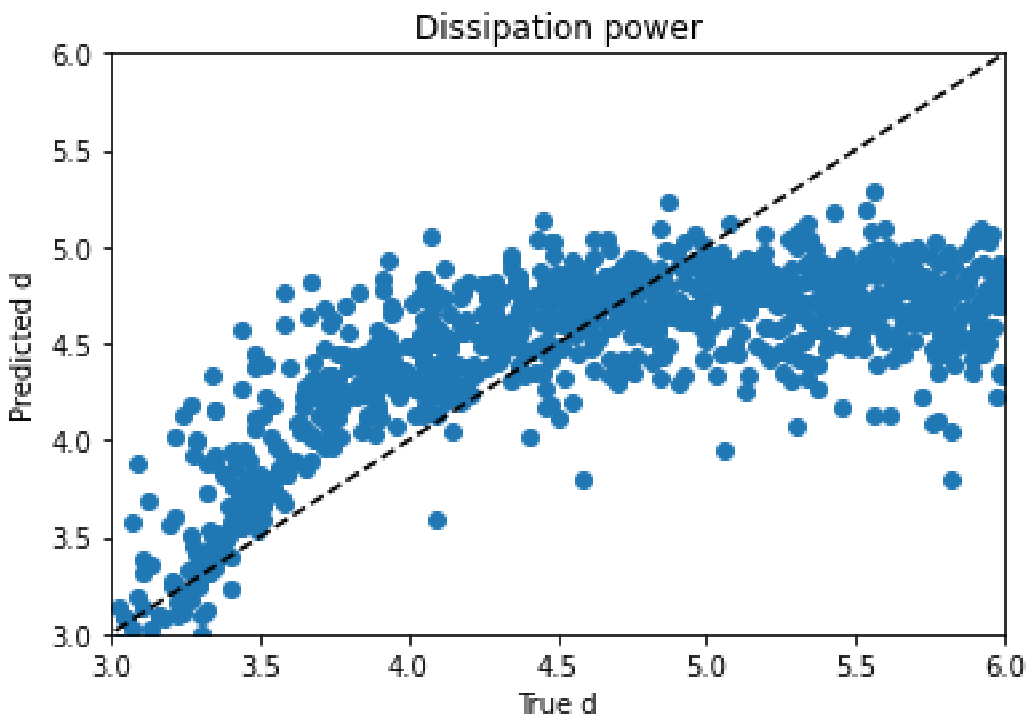
[114]:
```
plt.scatter(mat_test[:,0],mat12_predict[:,0]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True alpha");
plt.ylabel("Predicted alpha");
plt.axis([0, .2, 0, .2])
plt.title("Correlation strength")

plt.figure()
plt.scatter(mat_test[:,1],mat12_predict[:,1]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True xi");
plt.ylabel("Predicted xi");
plt.axis([0, 20, 0, 20])
plt.title("Correlation length")

plt.figure()
plt.scatter(mat_test[:,2],mat12_predict[:,2]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True p");
plt.ylabel("Predicted p");
plt.axis([2, 4, 2, 4])
plt.title("Correlation power")

plt.figure()
plt.scatter(mat_test[:,3],mat12_predict[:,3]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True d");
plt.ylabel("Predicted d");
plt.axis([3, 6, 3, 6]);
```

```
plt.title("Dissipation power");
```






Correlation power
Predicted p
True p
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
2.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75
4.00


Dissipation power
Predicted d
True d
6.0
5.5
5.0
4.5
4.0
3.5
3.0
3.0
3.5
4.0
4.5
5.0
5.5
6.0

1.13 Model 13

redo of model 5

[12]:
```
input13_layer = Input(shape=(N,))
layer131 = Dense(512,activation='relu')(input13_layer)
layer132 = Dense(256,activation='relu')(layer131)
layer133 = Dense(4)(layer132)

model13 = Model(name='Model_13',inputs=input13_layer, outputs=layer133)
model13.summary()
```

```
Model: "Model_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 942)]             0
_________________________________________________________________
dense (Dense)                (None, 512)               482816
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 1028
=================================================================
Total params: 615,172
Trainable params: 615,172
Non-trainable params: 0
_________________________________________________________________
```

[13]:
```
opt = keras.optimizers.SGD(learning_rate=0.01,momentum=0.95)
model13.
 ↪compile(loss='mean_squared_error',optimizer=opt,metrics=['mean_squared_error'])
```

[14]:
```
n_epochs = 40
n_batch = 10
# transform the target so that mse is equivalent to the appropriate metric
print('Starting Training')
model13.fit(M_train,scaler.
 ↪transform(mat_train),epochs=n_epochs,batch_size=n_batch)
print('Finished Training')
```

```
Starting Training
Epoch 1/40
900/900 [==============================] - 6s 7ms/step - loss: 0.0590 -
mean_squared_error: 0.0590
Epoch 2/40
```

```
900/900 [==============================] - 6s 6ms/step - loss: 0.0501 -
mean_squared_error: 0.0501
Epoch 3/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0480 -
mean_squared_error: 0.0480
Epoch 4/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0467 -
mean_squared_error: 0.0467
Epoch 5/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0446 -
mean_squared_error: 0.0446
Epoch 6/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0432 -
mean_squared_error: 0.0432
Epoch 7/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0420 -
mean_squared_error: 0.0420
Epoch 8/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0408 -
mean_squared_error: 0.0408
Epoch 9/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0409 -
mean_squared_error: 0.0409
Epoch 10/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0404 -
mean_squared_error: 0.0404
Epoch 11/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0393 -
mean_squared_error: 0.0393
Epoch 12/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0393 -
mean_squared_error: 0.0393
Epoch 13/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0393 -
mean_squared_error: 0.0393
Epoch 14/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0391 -
mean_squared_error: 0.0391
Epoch 15/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0378 -
mean_squared_error: 0.0378
Epoch 16/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0382 -
mean_squared_error: 0.0382
Epoch 17/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0375 -
mean_squared_error: 0.0375
Epoch 18/40
```

```
900/900 [==============================] - 4s 5ms/step - loss: 0.0383 -
mean_squared_error: 0.0383
Epoch 19/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0375 -
mean_squared_error: 0.0375
Epoch 20/40
900/900 [==============================] - 4s 5ms/step - loss: 0.0374 -
mean_squared_error: 0.0374
Epoch 21/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0367 -
mean_squared_error: 0.0367
Epoch 22/40
900/900 [==============================] - 6s 6ms/step - loss: 0.0369 -
mean_squared_error: 0.0369
Epoch 23/40
900/900 [==============================] - 6s 6ms/step - loss: 0.0369 -
mean_squared_error: 0.0369
Epoch 24/40
900/900 [==============================] - 7s 8ms/step - loss: 0.0364 -
mean_squared_error: 0.0364
Epoch 25/40
900/900 [==============================] - 9s 10ms/step - loss: 0.0358 -
mean_squared_error: 0.0358
Epoch 26/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0362 -
mean_squared_error: 0.0362
Epoch 27/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0357 -
mean_squared_error: 0.0357
Epoch 28/40
900/900 [==============================] - 9s 10ms/step - loss: 0.0360 -
mean_squared_error: 0.0360
Epoch 29/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0353 -
mean_squared_error: 0.0353
Epoch 30/40
900/900 [==============================] - 6s 7ms/step - loss: 0.0354 -
mean_squared_error: 0.0354
Epoch 31/40
900/900 [==============================] - 8s 9ms/step - loss: 0.0353 -
mean_squared_error: 0.0353
Epoch 32/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0354 -
mean_squared_error: 0.0354
Epoch 33/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0352 -
mean_squared_error: 0.0352
Epoch 34/40
```

```
900/900 [==============================] - 4s 5ms/step - loss: 0.0360 -
mean_squared_error: 0.0360
Epoch 35/40
900/900 [==============================] - 9s 10ms/step - loss: 0.0348 -
mean_squared_error: 0.0348
Epoch 36/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0357 -
mean_squared_error: 0.0357
Epoch 37/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0347 -
mean_squared_error: 0.0347
Epoch 38/40
900/900 [==============================] - 9s 9ms/step - loss: 0.0348 -
mean_squared_error: 0.0348
Epoch 39/40
900/900 [==============================] - 5s 5ms/step - loss: 0.0343 -
mean_squared_error: 0.0343
Epoch 40/40
900/900 [==============================] - 5s 6ms/step - loss: 0.0345 -
mean_squared_error: 0.0345
Finished Training
```

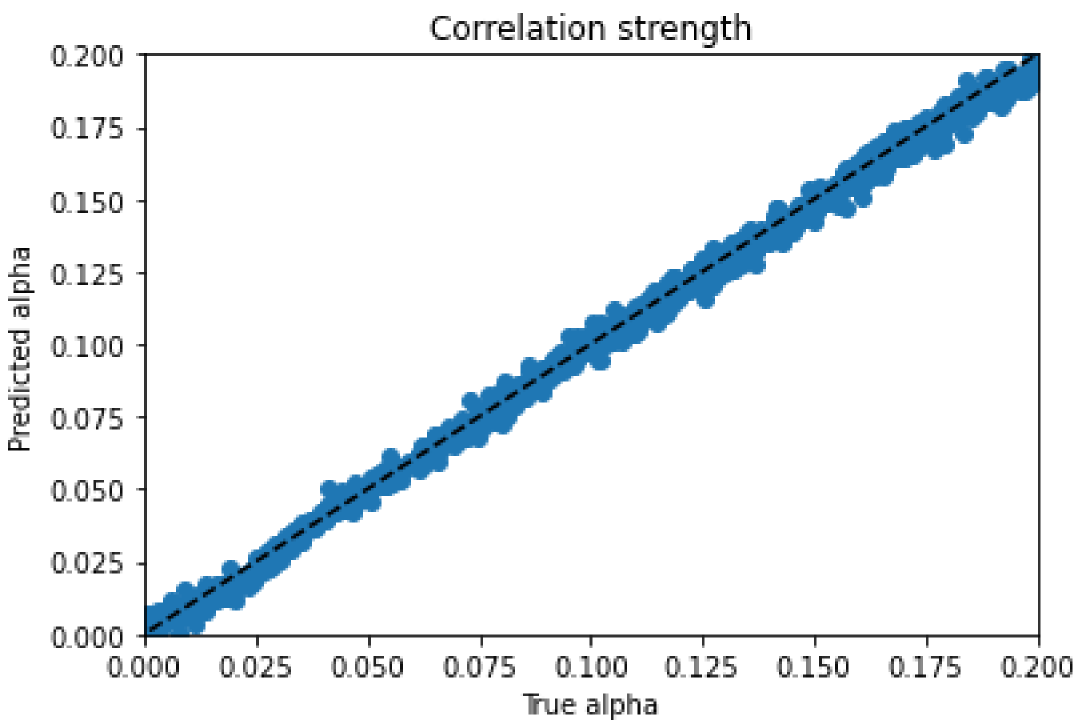
[15]:
```
mat13_train_predict = scaler.inverse_transform(model13.
 ↪predict(M_train,batch_size=n_batch))
mat13_predict = scaler.inverse_transform(model13.
 ↪predict(M_test,batch_size=n_batch))
print('Training score for model ',weight_mse(mat_train,mat13_train_predict))
print('Test score for model ',weight_mse(mat_test,mat13_predict))
```

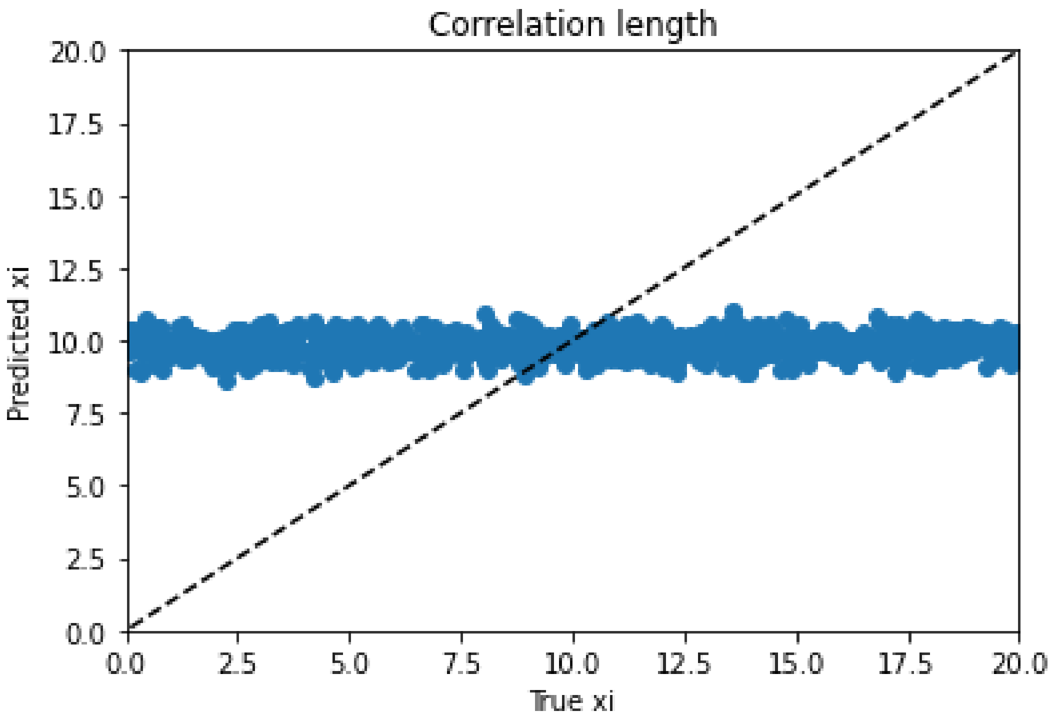
```
Training score for model  0.12976466189058466
Test score for model  0.1333459706907496
```

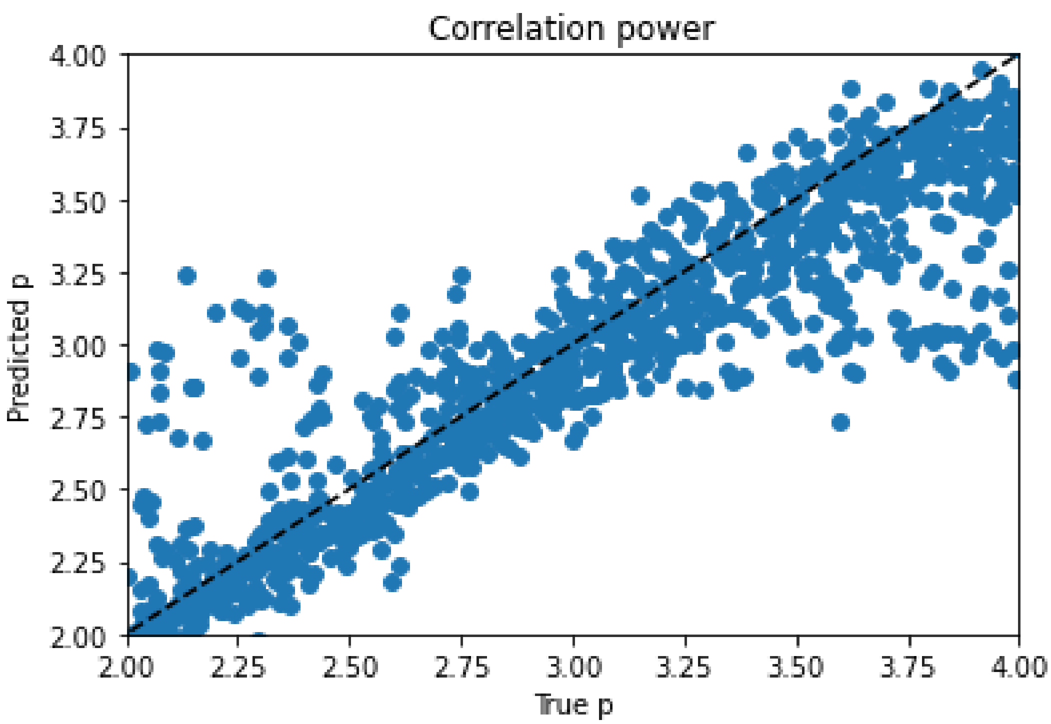
Let's plot that.

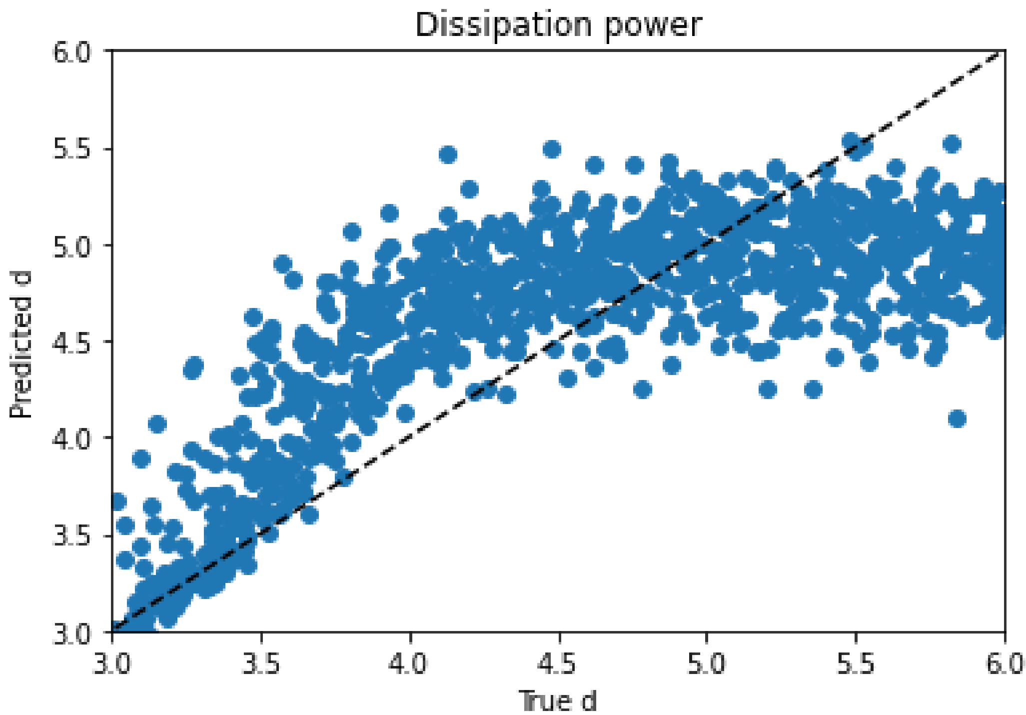
[17]:
```
plt.scatter(mat_test[:,0],mat13_predict[:,0]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True alpha");
plt.ylabel("Predicted alpha");
plt.axis([0, .2, 0, .2])
plt.title("Correlation strength")

plt.figure()
plt.scatter(mat_test[:,1],mat13_predict[:,1]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True xi");
plt.ylabel("Predicted xi");
plt.axis([0, 20, 0, 20])
plt.title("Correlation length")
```

```
plt.figure()
plt.scatter(mat_test[:,2],mat13_predict[:,2]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True p");
plt.ylabel("Predicted p");
plt.axis([2, 4, 2, 4])
plt.title("Correlation power")

plt.figure()
plt.scatter(mat_test[:,3],mat13_predict[:,3]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True d");
plt.ylabel("Predicted d");
plt.axis([3, 6, 3, 6]);
plt.title("Dissipation power");
```




Correlation length
Predicted xi
True xi
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0


Correlation power
Predicted p
True p
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00



1.14 Model 14

20 minutes to go and the frap ray cannon, I mean cells to produce submission, are all warmed up

[126]:
```
model14 = keras.models.clone_model(tuner10.get_best_models(num_models=2)[1])
model14.summary()
```

```
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.iter
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.decay
WARNING:tensorflow:Unresolved object in checkpoint:
(root).optimizer.learning_rate
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.momentum
WARNING:tensorflow:A checkpoint was restored (e.g. tf.train.Checkpoint.restore
or tf.keras.Model.load_weights) but not all checkpointed values were used. See
above for specific issues. Use expect_partial() on the load status object, e.g.
tf.train.Checkpoint.restore(…).expect_partial(), to silence these warnings, or
use assert_consumed() to make the check explicit. See
https://www.tensorflow.org/guide/checkpoint#loading_mechanics for details.
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.iter
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.decay
WARNING:tensorflow:Unresolved object in checkpoint:
(root).optimizer.learning_rate
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.momentum
WARNING:tensorflow:A checkpoint was restored (e.g. tf.train.Checkpoint.restore
```

```
or tf.keras.Model.load_weights) but not all checkpointed values were used. See
above for specific issues. Use expect_partial() on the load status object, e.g.
tf.train.Checkpoint.restore(…).expect_partial(), to silence these warnings, or
use assert_consumed() to make the check explicit. See
https://www.tensorflow.org/guide/checkpoint#loading_mechanics for details.
Model: "functional_1"
________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
==================
input_1 (InputLayer)            [(None, 900)]        0
________________________________________________________________________________
__________________
dense (Dense)                   (None, 256)          230656      input_1[0][0]
________________________________________________________________________________
__________________
dense_3 (Dense)                 (None, 256)          65792       dense[0][0]
________________________________________________________________________________
__________________
dense_2 (Dense)                 (None, 4)            1028        dense[0][0]
________________________________________________________________________________
__________________
dense_1 (Dense)                 (None, 32)           8224        dense[0][0]
________________________________________________________________________________
__________________
dense_6 (Dense)                 (None, 8)            2056        dense_3[0][0]
________________________________________________________________________________
__________________
dense_5 (Dense)                 (None, 2)            10          dense_2[0][0]
________________________________________________________________________________
__________________
dense_4 (Dense)                 (None, 32)           1056        dense_1[0][0]
________________________________________________________________________________
__________________
dense_9 (Dense)                 (None, 1)            9           dense_6[0][0]
________________________________________________________________________________
__________________
dense_8 (Dense)                 (None, 1)            3           dense_5[0][0]
________________________________________________________________________________
__________________
dense_10 (Dense)                (None, 1)            9           dense_6[0][0]
________________________________________________________________________________
__________________
dense_7 (Dense)                 (None, 1)            33          dense_4[0][0]
________________________________________________________________________________
__________________
concatenate (Concatenate)       (None, 4)            0           dense_9[0][0]
```

```
                                                                 dense_8[0][0]
                                                                 dense_10[0][0]
                                                                 dense_7[0][0]
================================================================================
==================
Total params: 308,876
Trainable params: 308,876
Non-trainable params: 0

--------------------------------------------------------------------------------
------------------
```

[127]:
```
opt = keras.optimizers.SGD(learning_rate=0.01,momentum=0.95)
model14.
 ↪compile(loss='mean_squared_error',optimizer=opt,metrics=['mean_squared_error'])
n_epochs = 40
n_batch = 10
# transform the target to scale features to [0,1]...not effing with log␣
 ↪transform
# just using the even bigger structure from tuner10 #2
print('Starting Training')
model14.fit(M_train,scaler.
 ↪transform(mat_train),epochs=n_epochs,batch_size=n_batch)
print('Finished Training')
```

```
Starting Training
Epoch 1/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0587 -
mean_squared_error: 0.0587
Epoch 2/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0516 -
mean_squared_error: 0.0516
Epoch 3/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0503 -
mean_squared_error: 0.0503
Epoch 4/40
900/900 [==============================] - 3s 4ms/step - loss: 0.0490 -
mean_squared_error: 0.0490
Epoch 5/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0474 -
mean_squared_error: 0.0474
Epoch 6/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0471 -
mean_squared_error: 0.0471
Epoch 7/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0458 -
mean_squared_error: 0.0458
Epoch 8/40
```

```
900/900 [==============================] - 4s 4ms/step - loss: 0.0442 -
mean_squared_error: 0.0442
Epoch 9/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0438 -
mean_squared_error: 0.0438
Epoch 10/40
900/900 [==============================] - 3s 4ms/step - loss: 0.0424 -
mean_squared_error: 0.0424
Epoch 11/40
900/900 [==============================] - 3s 4ms/step - loss: 0.0424 -
mean_squared_error: 0.0424
Epoch 12/40
900/900 [==============================] - 3s 4ms/step - loss: 0.0417 -
mean_squared_error: 0.0417
Epoch 13/40
900/900 [==============================] - 3s 4ms/step - loss: 0.0418 -
mean_squared_error: 0.0418
Epoch 14/40
900/900 [==============================] - 3s 4ms/step - loss: 0.0406 -
mean_squared_error: 0.0406
Epoch 15/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0409 -
mean_squared_error: 0.0409
Epoch 16/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0398 -
mean_squared_error: 0.0398
Epoch 17/40
900/900 [==============================] - 3s 4ms/step - loss: 0.0402 -
mean_squared_error: 0.0402
Epoch 18/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0395 -
mean_squared_error: 0.0395
Epoch 19/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0396 -
mean_squared_error: 0.0396
Epoch 20/40
900/900 [==============================] - 3s 4ms/step - loss: 0.0392 -
mean_squared_error: 0.0392
Epoch 21/40
900/900 [==============================] - 3s 4ms/step - loss: 0.0395 -
mean_squared_error: 0.0395
Epoch 22/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0392 -
mean_squared_error: 0.0392
Epoch 23/40
900/900 [==============================] - 3s 4ms/step - loss: 0.0391 -
mean_squared_error: 0.0391
Epoch 24/40
```

```
900/900 [==============================] - 4s 4ms/step - loss: 0.0386 -
mean_squared_error: 0.0386
Epoch 25/40
900/900 [==============================] - 3s 4ms/step - loss: 0.0380 -
mean_squared_error: 0.0380
Epoch 26/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0387 -
mean_squared_error: 0.0387
Epoch 27/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0380 -
mean_squared_error: 0.0380
Epoch 28/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0378 -
mean_squared_error: 0.0378
Epoch 29/40
900/900 [==============================] - 3s 4ms/step - loss: 0.0375 -
mean_squared_error: 0.0375
Epoch 30/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0378 -
mean_squared_error: 0.0378
Epoch 31/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0377 -
mean_squared_error: 0.0377
Epoch 32/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0373 -
mean_squared_error: 0.0373
Epoch 33/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0381 -
mean_squared_error: 0.0381
Epoch 34/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0370 -
mean_squared_error: 0.0370
Epoch 35/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0368 -
mean_squared_error: 0.0368
Epoch 36/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0376 -
mean_squared_error: 0.0376
Epoch 37/40
900/900 [==============================] - 3s 4ms/step - loss: 0.0368 -
mean_squared_error: 0.0368
Epoch 38/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0373 -
mean_squared_error: 0.0373
Epoch 39/40
900/900 [==============================] - 4s 4ms/step - loss: 0.0365 -
mean_squared_error: 0.0365
Epoch 40/40
```

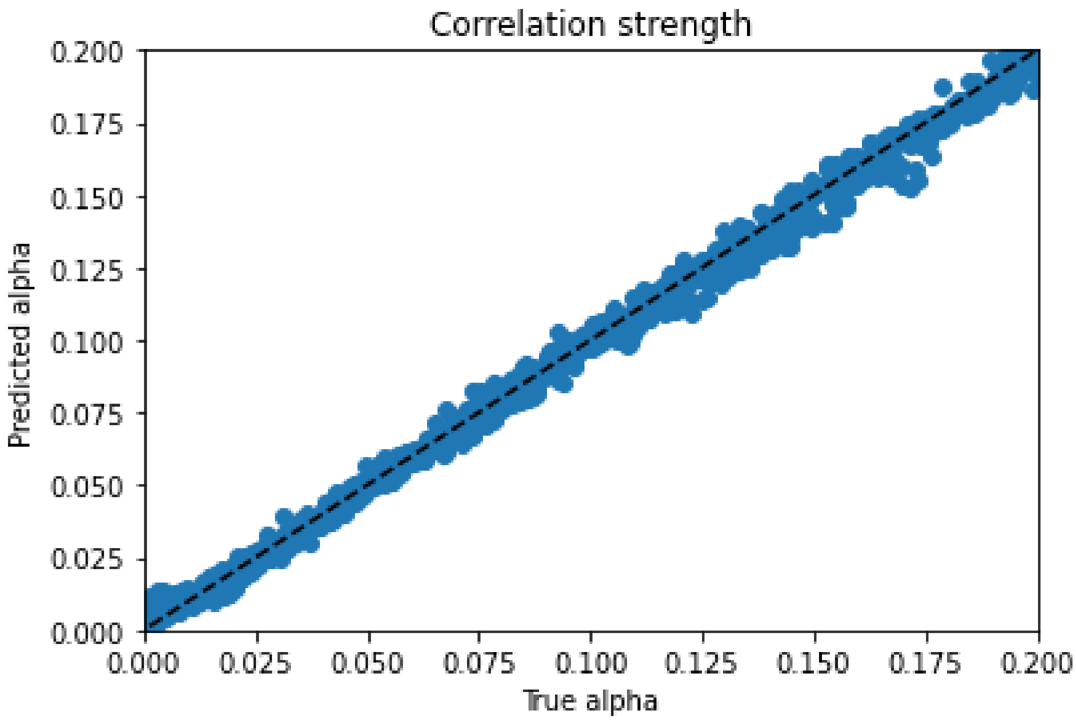
```
900/900 [==============================] - 4s 4ms/step - loss: 0.0365 -
mean_squared_error: 0.0365
Finished Training
```

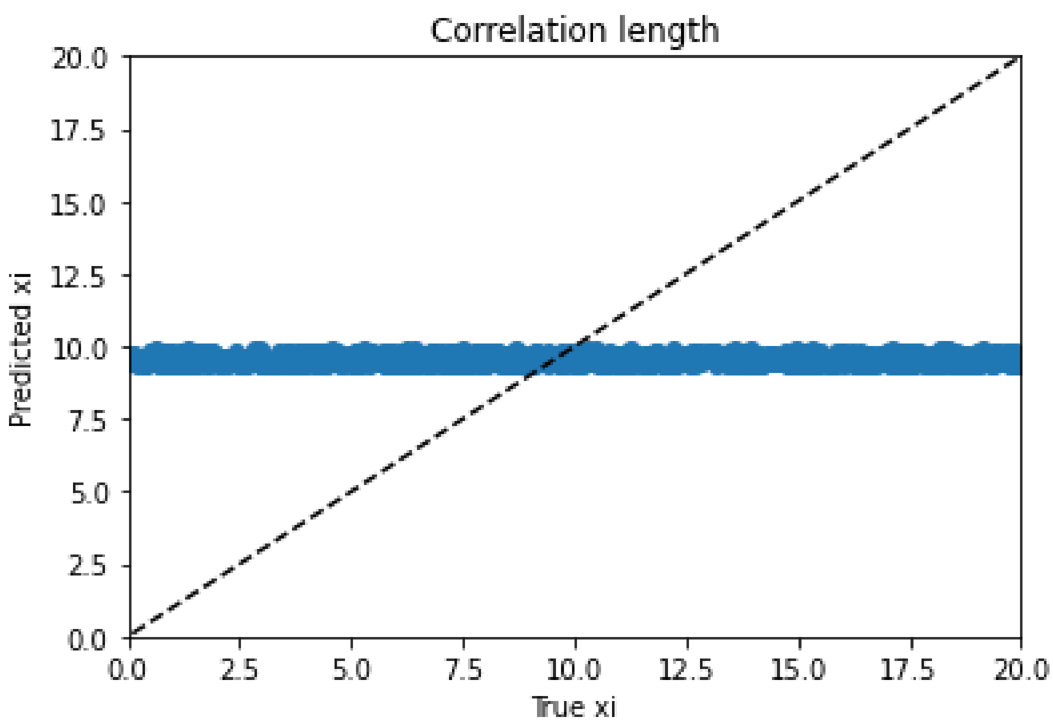
[128]:
```
mat14_train_predict = scaler.inverse_transform(model14.
 ↪predict(M_train,batch_size=n_batch))
mat14_predict = scaler.inverse_transform(model14.
 ↪predict(M_test,batch_size=n_batch))
print('Training score for model ',weight_mse(mat_train,mat14_train_predict))
print('Test score for model ',weight_mse(mat_test,mat14_predict))
```

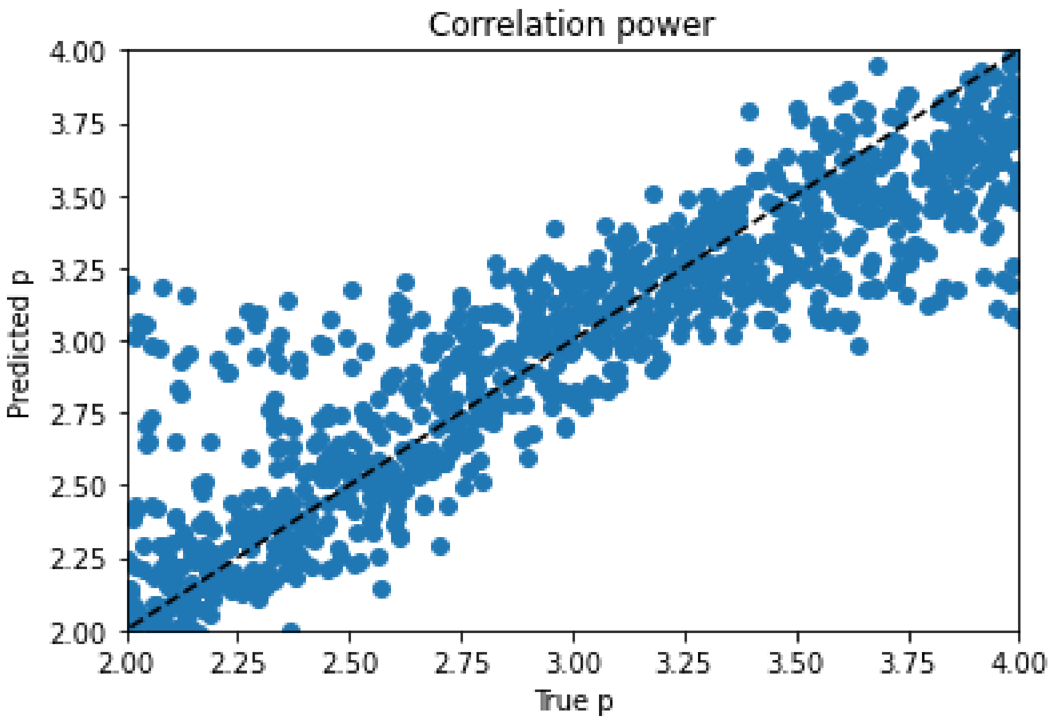
```
Training score for model  0.1416714822112891
Test score for model  0.1466672355807766
```

Let's plot that.

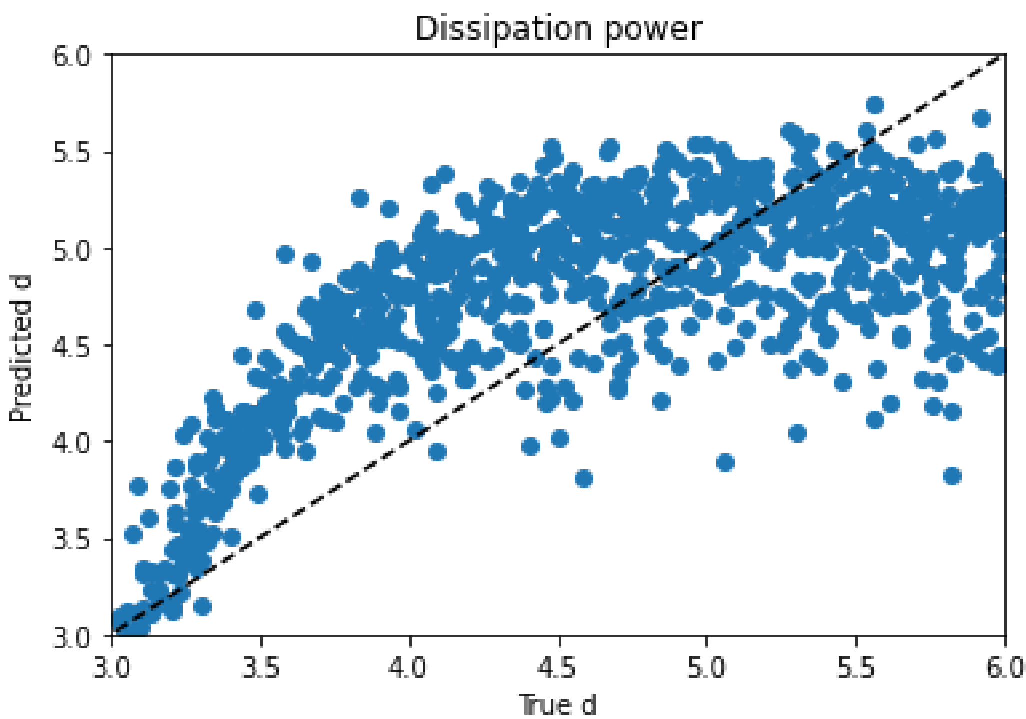
[129]:
```
plt.scatter(mat_test[:,0],mat14_predict[:,0]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True alpha");
plt.ylabel("Predicted alpha");
plt.axis([0, .2, 0, .2])
plt.title("Correlation strength")

plt.figure()
plt.scatter(mat_test[:,1],mat14_predict[:,1]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True xi");
plt.ylabel("Predicted xi");
plt.axis([0, 20, 0, 20])
plt.title("Correlation length")

plt.figure()
plt.scatter(mat_test[:,2],mat14_predict[:,2]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True p");
plt.ylabel("Predicted p");
plt.axis([2, 4, 2, 4])
plt.title("Correlation power")

plt.figure()
plt.scatter(mat_test[:,3],mat14_predict[:,3]);
plt.plot([-100, 100],[-100, 100],"--k")
plt.xlabel("True d");
plt.ylabel("Predicted d");
plt.axis([3, 6, 3, 6]);
plt.title("Dissipation power");
```


Correlation strength
Predicted alpha
True alpha
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200


Correlation length
Predicted xi
True xi
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



Correlation power
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
Predicted p
2.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75
4.00
True p



Dissipation power
6.0
5.5
5.0
4.5
4.0
3.5
3.0
Predicted d
3.0
3.5
4.0
4.5
5.0
5.5
6.0
True d

```
[ ]:
```