In [4]:

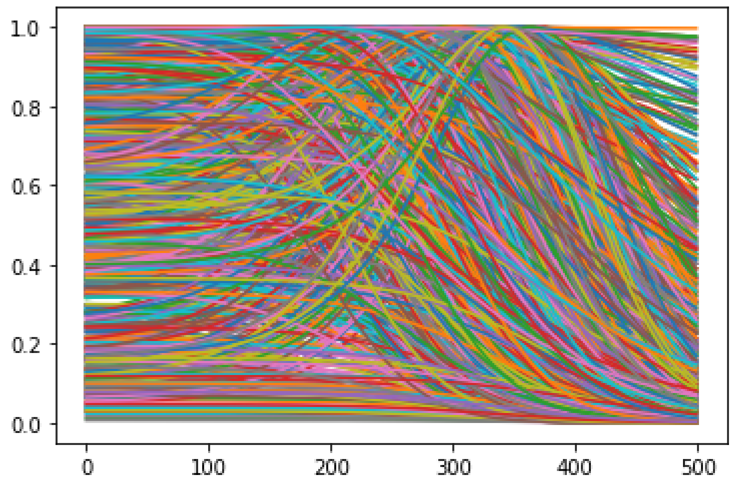
```
# Normal imports for everybody
import keras
#from context import * # imports the MDN layer
import mdn
import numpy as np
import random
import matplotlib.pyplot as plt
%matplotlib inline
```

```
Using TensorFlow backend.

WARNING: The TensorFlow contrib module will not be included in Tenso
rFlow 2.0.
For more information, please see:
  * https://github.com/tensorflow/community/blob/master/rfcs/2018090
7-contrib-sunset.md
  * https://github.com/tensorflow/addons
If you depend on functionality not listed there, please file an issu
e.
```

In [5]:

```
X=np.loadtxt('X.csv',delimiter=',')
Y=np.loadtxt('Y.csv',delimiter=',')

X=np.reshape(X,(-1,1000,3))

for i in range(3):
    mean=np.mean(X[:,:,i])
    std=np.std(X[:,:,i])
    X[:,:,i]=(X[:,:,i]-mean)/std
'''
plt.plot(np.transpose(X[:,:,0]))
plt.show()
plt.plot(np.transpose(X[:,:,1]))
plt.show()
plt.plot(np.transpose(X[:,:,2]))
plt.show()
'''
X=np.reshape(X,(-1,1000*3)) #Flatten

Y=Y*1e20

plt.plot(np.transpose(Y))

x_data=X
y_data=Y
print(np.shape(x_data),np.shape(y_data))
```

```
(1000, 3000) (1000, 500)
```



In [6]:

```
N_INPUT=3*1000
N_MIXES = 3
N_OUTPUT = 500

model = keras.Sequential()
model.add(keras.layers.Dense(1000, batch_input_shape=(None, N_INPUT), activation
='relu'))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Dense(500, activation='relu'))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Dense(1000, activation='relu'))
model.add(keras.layers.BatchNormalization())
model.add(mdn.MDN(N_OUTPUT, N_MIXES)) #output dimensions, no. of mixes.
model.compile(loss=mdn.get_mixture_loss_func(N_OUTPUT,N_MIXES), optimizer=keras.
optimizers.Adam()) #, metrics=[mdn.get_mixture_mse_accuracy(1,N_MIXES)])
model.summary()
```

```
WARNING:tensorflow:From /home/vinit/anaconda3/envs/tensorflow/lib/py
thon3.6/site-packages/tensorflow/python/framework/op_def_library.py:
263: colocate_with (from tensorflow.python.framework.ops) is depreca
ted and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 1000)              3001000   
_________________________________________________________________
batch_normalization_1 (Batch (None, 1000)              4000      
_________________________________________________________________
dense_2 (Dense)              (None, 500)               500500    
_________________________________________________________________
batch_normalization_2 (Batch (None, 500)               2000      
_________________________________________________________________
dense_3 (Dense)              (None, 1000)              501000    
_________________________________________________________________
batch_normalization_3 (Batch (None, 1000)              4000      
_________________________________________________________________
mdn_1 (MDN)                  (None, 3003)              3006003   
=================================================================
Total params: 7,018,503
Trainable params: 7,013,503
Non-trainable params: 5,000
_________________________________________________________________
```

In [7]:

```
history = model.fit(x=x_data, y=y_data, batch_size=100, epochs=20, validation_sp
lit=0.15,callbacks=[keras.callbacks.TerminateOnNaN()])
```

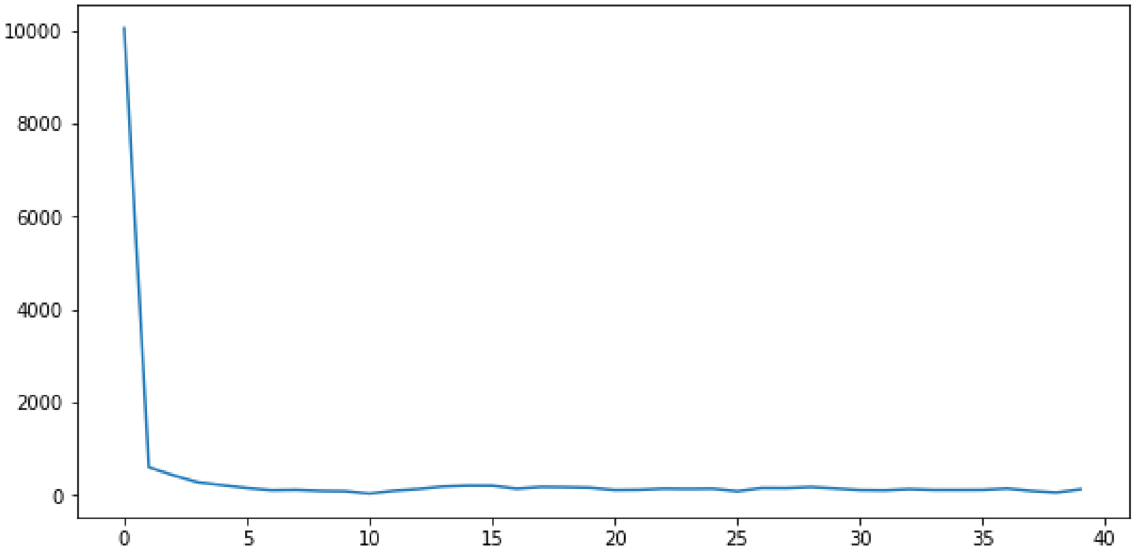
```
WARNING:tensorflow:From /home/vinit/anaconda3/envs/tensorflow/lib/py
thon3.6/site-packages/tensorflow/python/ops/math_ops.py:3066: to_int
32 (from tensorflow.python.ops.math_ops) is deprecated and will be r
emoved in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 850 samples, validate on 150 samples
Epoch 1/40
850/850 [==============================] - 5s 6ms/step - loss: 1005
8.3347 - val_loss: 3565322.0000
Epoch 2/40
850/850 [==============================] - 1s 1ms/step - loss: 607.2
239 - val_loss: 668631.7917
Epoch 3/40
850/850 [==============================] - 1s 1ms/step - loss: 429.1
245 - val_loss: 52106.2135
Epoch 4/40
850/850 [==============================] - 1s 1ms/step - loss: 278.5
510 - val_loss: 10071.4971
Epoch 5/40
850/850 [==============================] - 1s 1ms/step - loss: 217.0
308 - val_loss: 3650.2513
Epoch 6/40
850/850 [==============================] - 1s 1ms/step - loss: 155.2
594 - val_loss: 1981.3910
Epoch 7/40
850/850 [==============================] - 1s 1ms/step - loss: 109.3
593 - val_loss: 1540.0127
Epoch 8/40
850/850 [==============================] - 1s 1ms/step - loss: 117.3
692 - val_loss: 992.6818
Epoch 9/40
850/850 [==============================] - 1s 1ms/step - loss: 95.57
38 - val_loss: 970.1832
Epoch 10/40
850/850 [==============================] - 2s 2ms/step - loss: 89.45
31 - val_loss: 819.1007
Epoch 11/40
850/850 [==============================] - 2s 2ms/step - loss: 35.99
81 - val_loss: 749.7772
Epoch 12/40
850/850 [==============================] - 2s 2ms/step - loss: 98.20
76 - val_loss: 510.0120
Epoch 13/40
850/850 [==============================] - 1s 2ms/step - loss: 134.9
045 - val_loss: 422.0808
Epoch 14/40
850/850 [==============================] - 1s 1ms/step - loss: 187.6
087 - val_loss: 337.1822
Epoch 15/40
850/850 [==============================] - 1s 2ms/step - loss: 208.5
413 - val_loss: 329.6429
Epoch 16/40
850/850 [==============================] - 1s 2ms/step - loss: 207.4
195 - val_loss: 294.6440
Epoch 17/40
850/850 [==============================] - 1s 2ms/step - loss: 140.7
284 - val_loss: 320.6318
Epoch 18/40
850/850 [==============================] - 2s 2ms/step - loss: 178.7
937 - val_loss: 305.2497
```

```
Epoch 19/40
850/850 [==============================] - 1s 2ms/step - loss: 173.1
423 - val_loss: 215.5386
Epoch 20/40
850/850 [==============================] - 1s 2ms/step - loss: 161.9
509 - val_loss: 224.9626
Epoch 21/40
850/850 [==============================] - 1s 2ms/step - loss: 113.4
608 - val_loss: 205.9448
Epoch 22/40
850/850 [==============================] - 2s 2ms/step - loss: 118.8
771 - val_loss: 202.5706
Epoch 23/40
850/850 [==============================] - 2s 2ms/step - loss: 141.9
971 - val_loss: 167.3862
Epoch 24/40
850/850 [==============================] - 2s 3ms/step - loss: 136.7
694 - val_loss: 227.1336
Epoch 25/40
850/850 [==============================] - 2s 3ms/step - loss: 142.2
237 - val_loss: 167.9479
Epoch 26/40
850/850 [==============================] - 2s 2ms/step - loss: 88.77
79 - val_loss: 236.4647
Epoch 27/40
850/850 [==============================] - 2s 2ms/step - loss: 155.7
368 - val_loss: 242.6887
Epoch 28/40
850/850 [==============================] - 2s 3ms/step - loss: 153.5
379 - val_loss: 287.1839
Epoch 29/40
850/850 [==============================] - 2s 3ms/step - loss: 175.9
610 - val_loss: 235.7759
Epoch 30/40
850/850 [==============================] - 2s 3ms/step - loss: 145.3
422 - val_loss: 207.2799
Epoch 31/40
850/850 [==============================] - 2s 3ms/step - loss: 112.8
202 - val_loss: 153.4831
Epoch 32/40
850/850 [==============================] - 2s 3ms/step - loss: 106.9
465 - val_loss: 209.1587
Epoch 33/40
850/850 [==============================] - 2s 3ms/step - loss: 132.4
878 - val_loss: 230.8074
Epoch 34/40
850/850 [==============================] - 2s 2ms/step - loss: 115.5
442 - val_loss: 205.8430
Epoch 35/40
850/850 [==============================] - 2s 2ms/step - loss: 115.1
103 - val_loss: 164.2642
Epoch 36/40
850/850 [==============================] - 1s 2ms/step - loss: 116.8
120 - val_loss: 313.3868
Epoch 37/40
850/850 [==============================] - 1s 1ms/step - loss: 145.1
081 - val_loss: 195.4933
Epoch 38/40
850/850 [==============================] - 1s 1ms/step - loss: 95.15
51 - val_loss: 112.5865
Epoch 39/40
```

```
850/850 [==============================] - 1s 1ms/step - loss: 58.83
62 - val_loss: 170.3546
Epoch 40/40
850/850 [==============================] - 1s 1ms/step - loss: 130.1
065 - val_loss: 287.4069
```

In [8]:

```
plt.figure(figsize=(10, 5))
#plt.ylim([0,9])
plt.plot(history.history['loss'])
#plt.loglog(history.history['val_loss'])
plt.show()
```



# Sampling Functions

The MDN model outputs parameters of a mixture model---a list of means (mu), variances (sigma), and weights (pi).

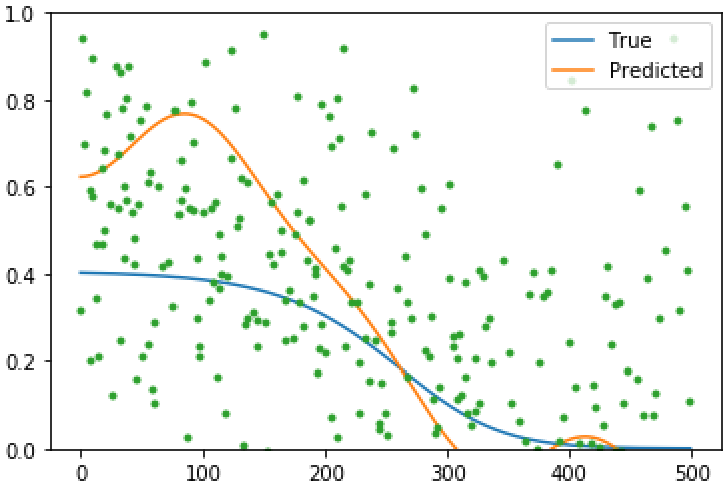
The `mdn` module provides a function to sample from these parameters as follows. First the parameters are split up into `mu` s, `sigma` s and `pi` s, then the categorical distribution formed by the `pi` s is sampled to choose which mixture component should be sampled, then that component's `mu` s and `sigma` s is used to sample from a multivariate normal model, here's the code:

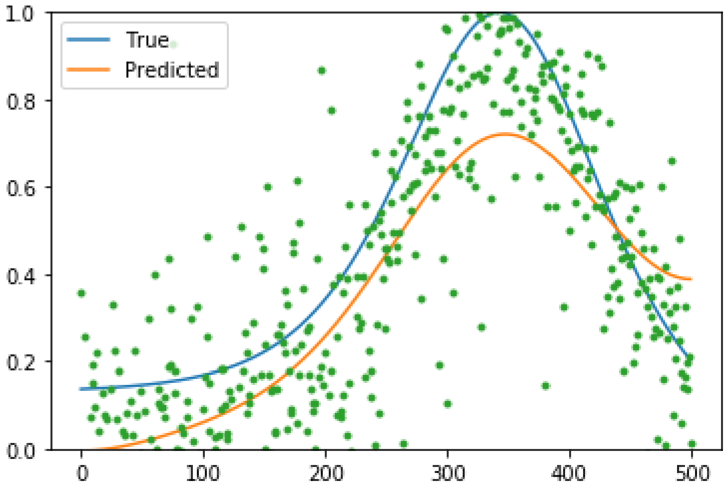
```
def sample_from_output(params, output_dim, num_mixes, temp=1.0):
    """Sample from an MDN output with temperature adjustment."""
    mus = params[:num_mixes*output_dim]
    sigs = params[num_mixes*output_dim:2*num_mixes*output_dim]
    pis = softmax(params[-num_mixes:], t=temp)
    m = sample_from_categorical(pis)
    # Alternative way to sample from categorical:
    # m = np.random.choice(range(len(pis)), p=pis)
    mus_vector = mus[m*output_dim:(m+1)*output_dim]
    sig_vector = sigs[m*output_dim:(m+1)*output_dim] * temp  # adjust for
temperature
    cov_matrix = np.identity(output_dim) * sig_vector
    sample = np.random.multivariate_normal(mus_vector, cov_matrix, 1)
    return sample
```

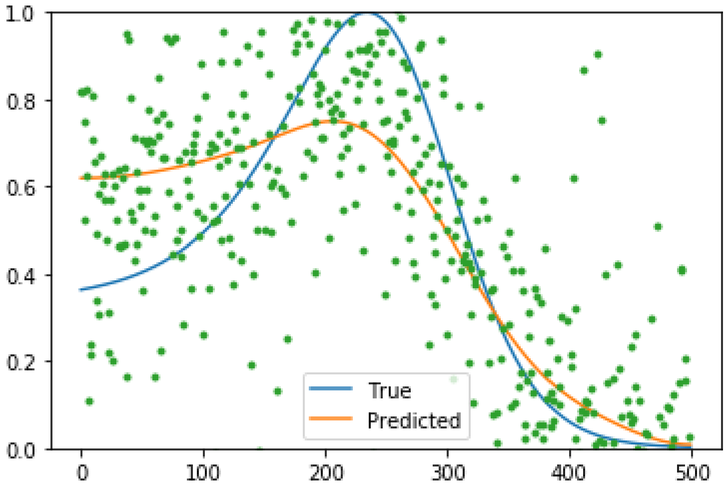
If you only have one prediction to sample from, you can use the function as is; but if you need to sample from a lot of predictions at once (as in the following sections), you can use `np.apply_along_axis` to apply it to a whole numpy array of predicted parameters.

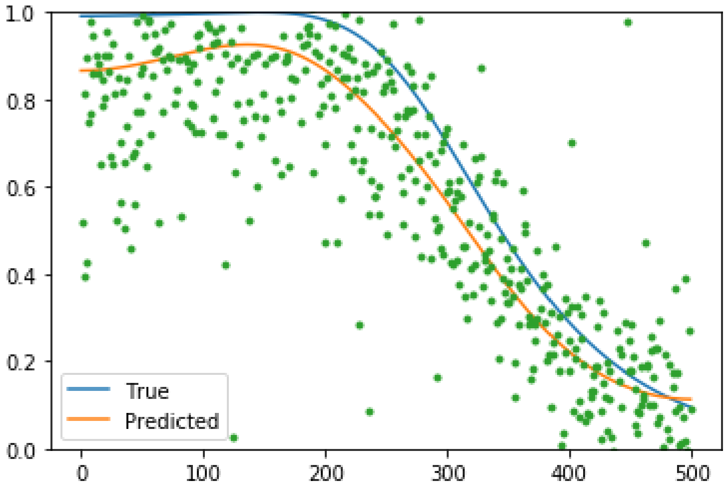
In [155]:

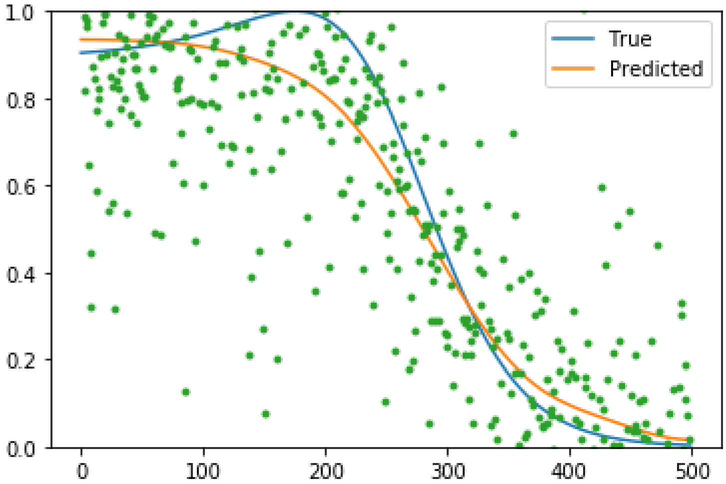
```
import scipy.ndimage.filters as sp
from scipy.special import softmax
y_test = model.predict(x_data)
for i in range(20):
    pdf=0
    pis=softmax(y_test[i,-N_MIXES:])
    for j in range(N_MIXES):
        mus=y_test[i,j*N_OUTPUT:N_OUTPUT*(j+1)]
        sigs=y_test[i,N_OUTPUT*(N_MIXES+j):N_OUTPUT*(N_MIXES+j+1)]
        pdf+=mus*pis[j]
    plt.plot(y_data[i,:],label='True')
    plt.plot(sp.gaussian_filter1d(pdf,50),label='Predicted')
    plt.plot(pdf,'.')
    plt.ylim((0,1))
    plt.legend()
    plt.show()
```

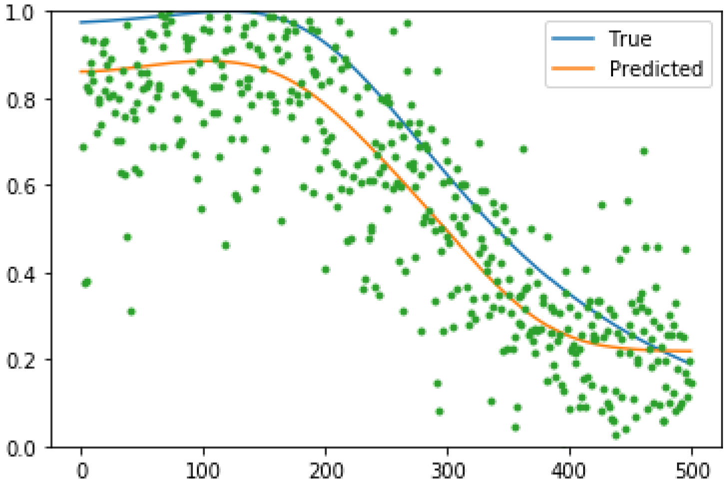

True
Predicted

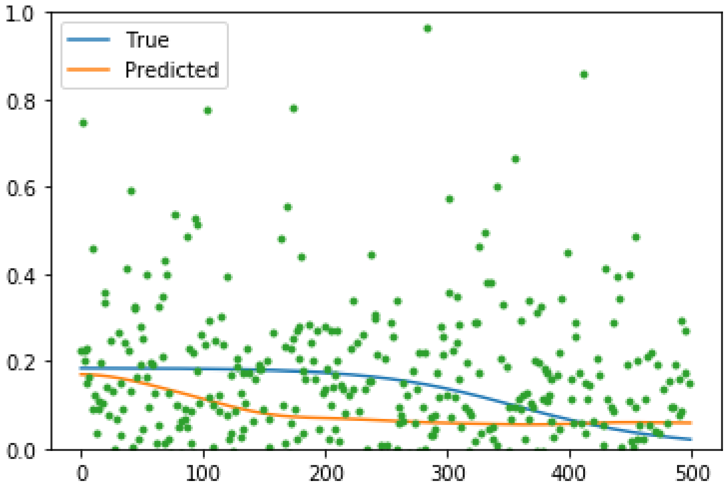

True
Predicted

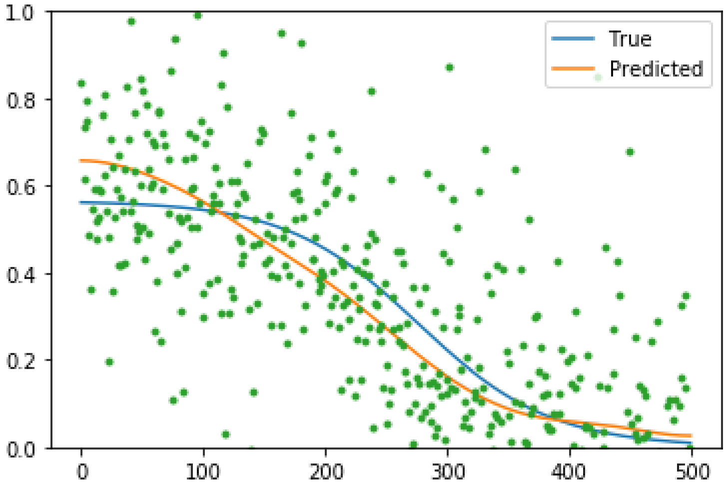

True
Predicted

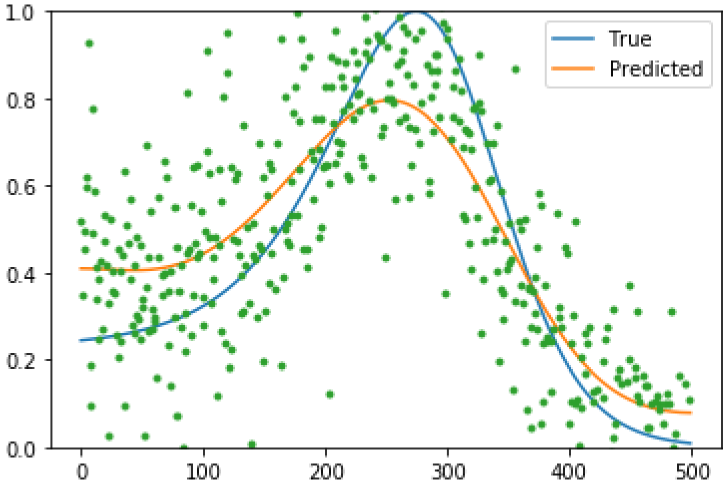

True
Predicted

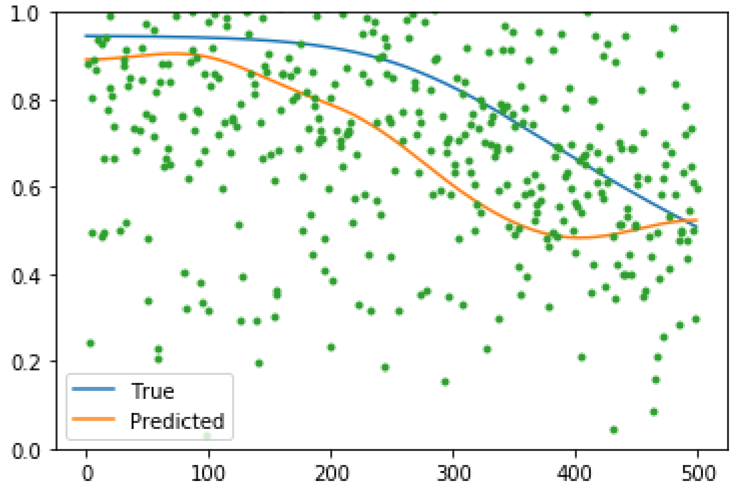

True
Predicted

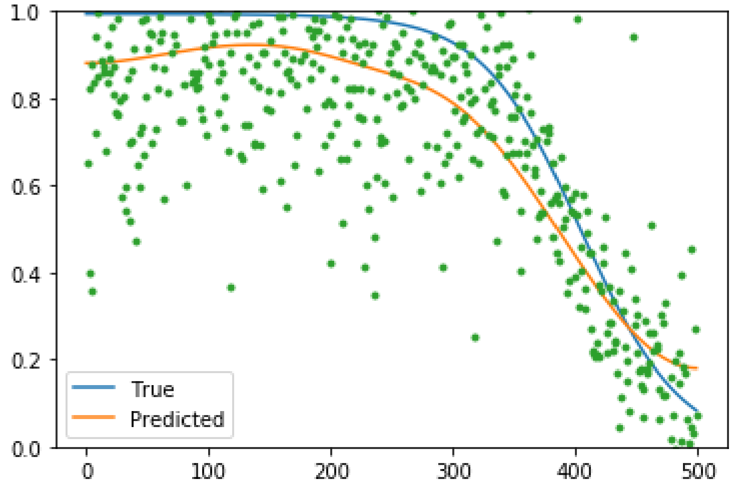

True
Predicted

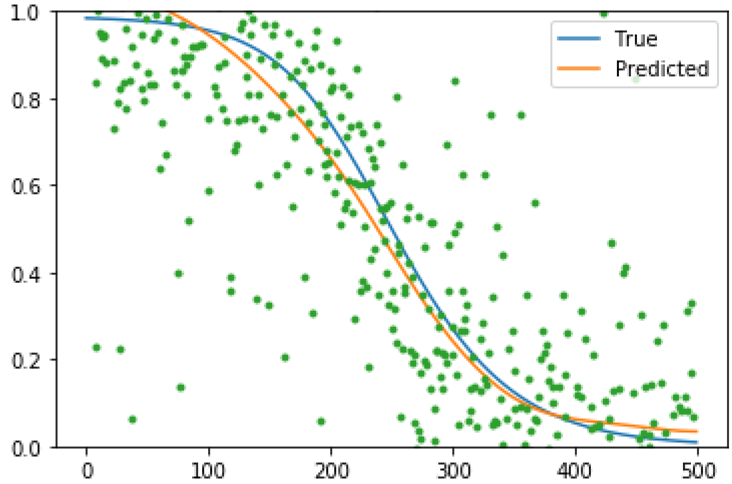

True
Predicted
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500

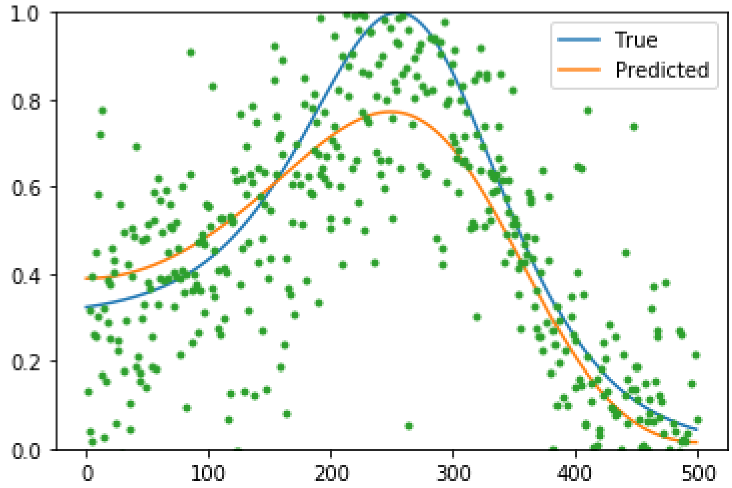

True
Predicted
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500


True
Predicted
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500

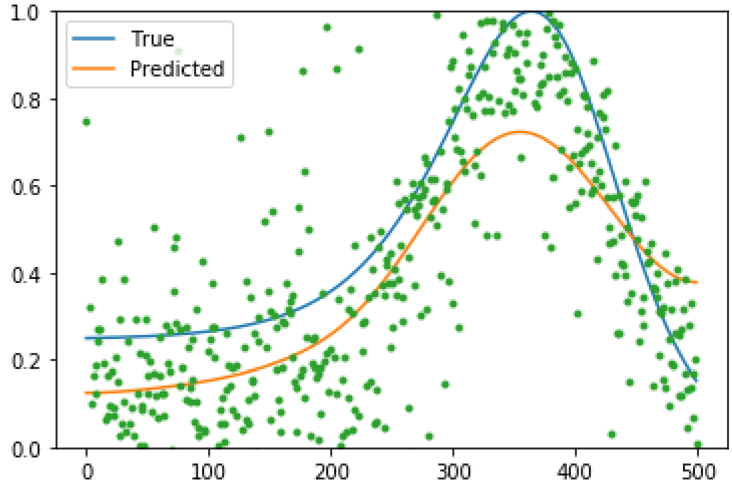

1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500
True
Predicted


1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500
True
Predicted

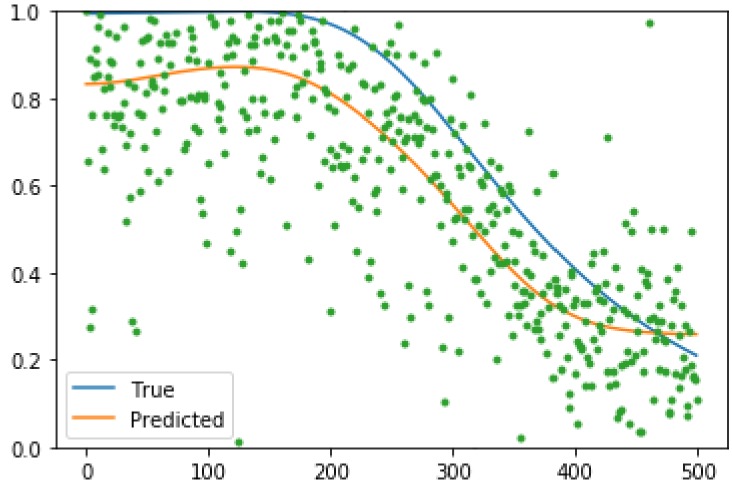

1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500
True
Predicted



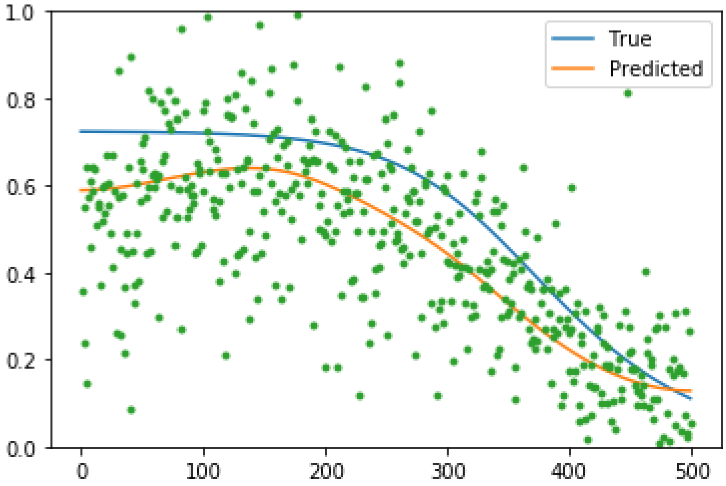
True
Predicted
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500

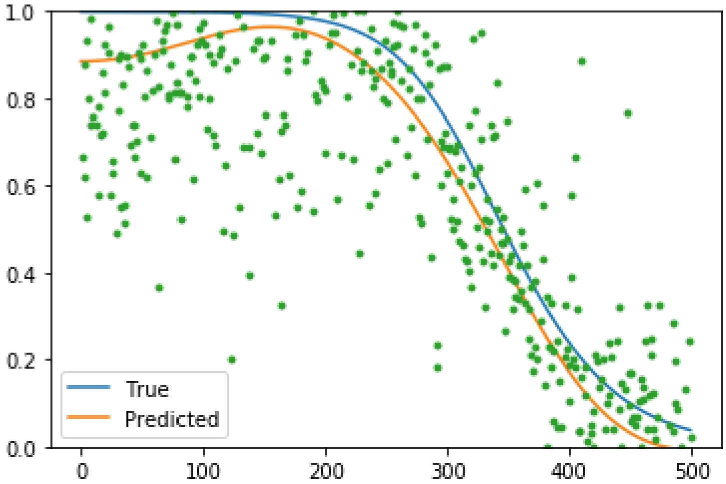



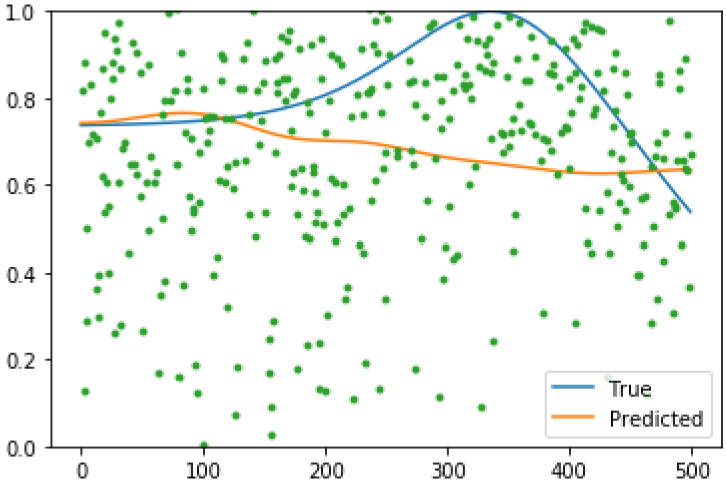
True
Predicted
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500

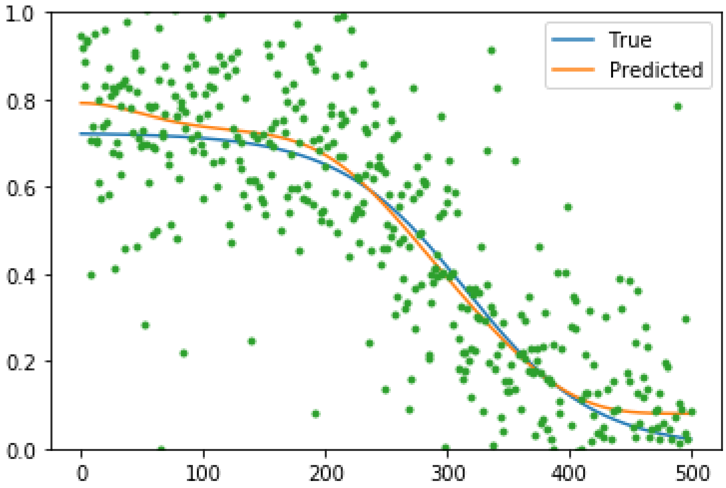



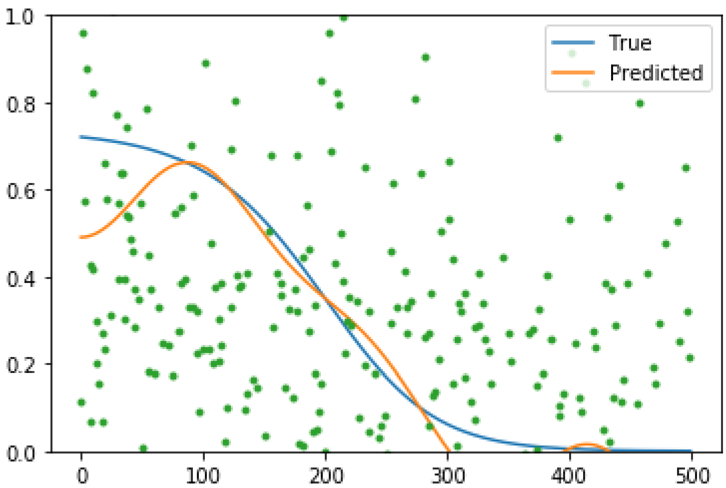
True
Predicted
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500


True
Predicted
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500


True
Predicted
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500

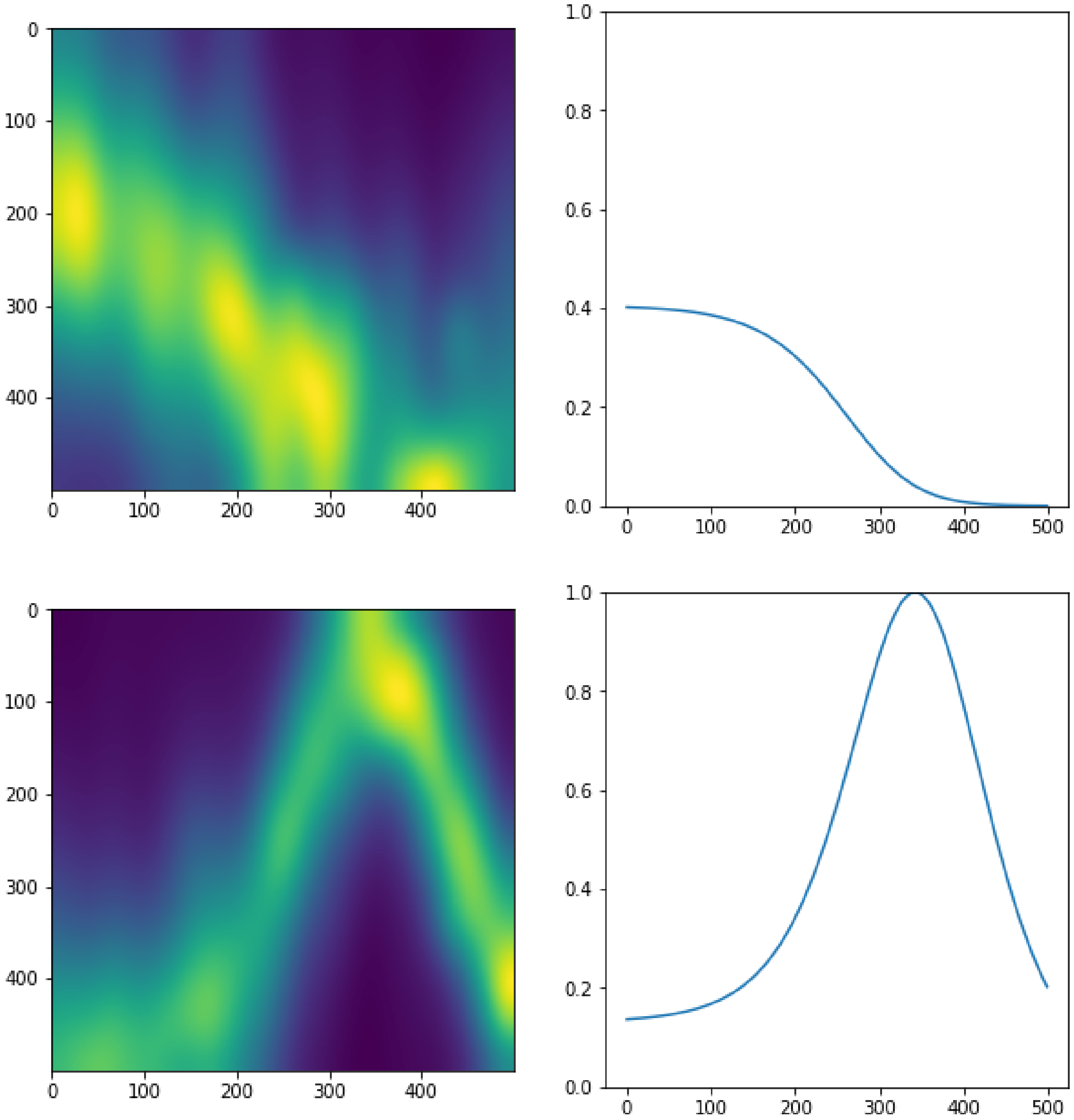

True
Predicted
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500

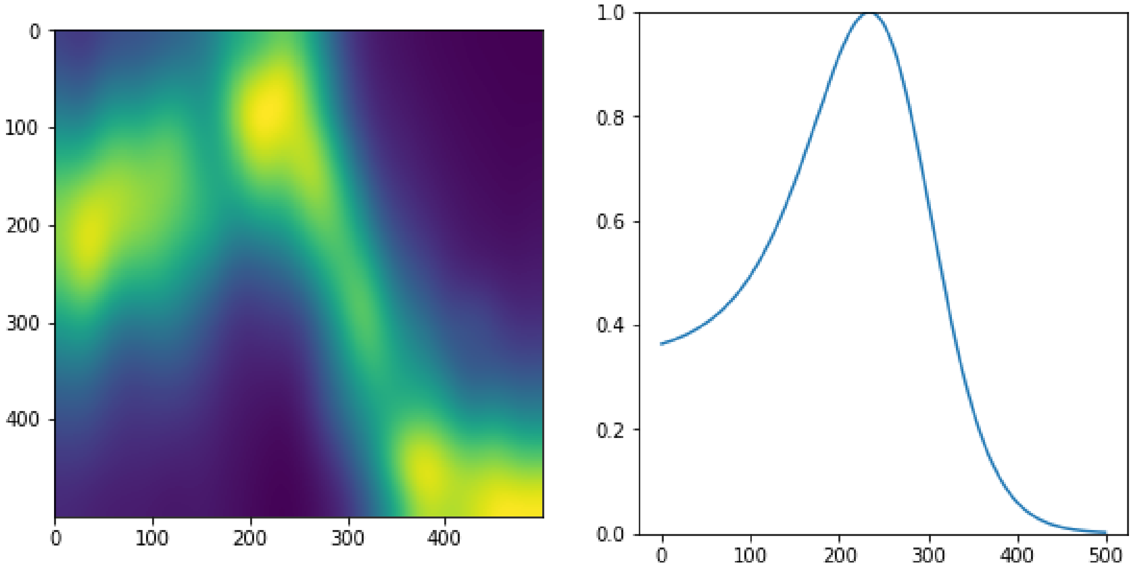

True
Predicted
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500


True
Predicted
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500

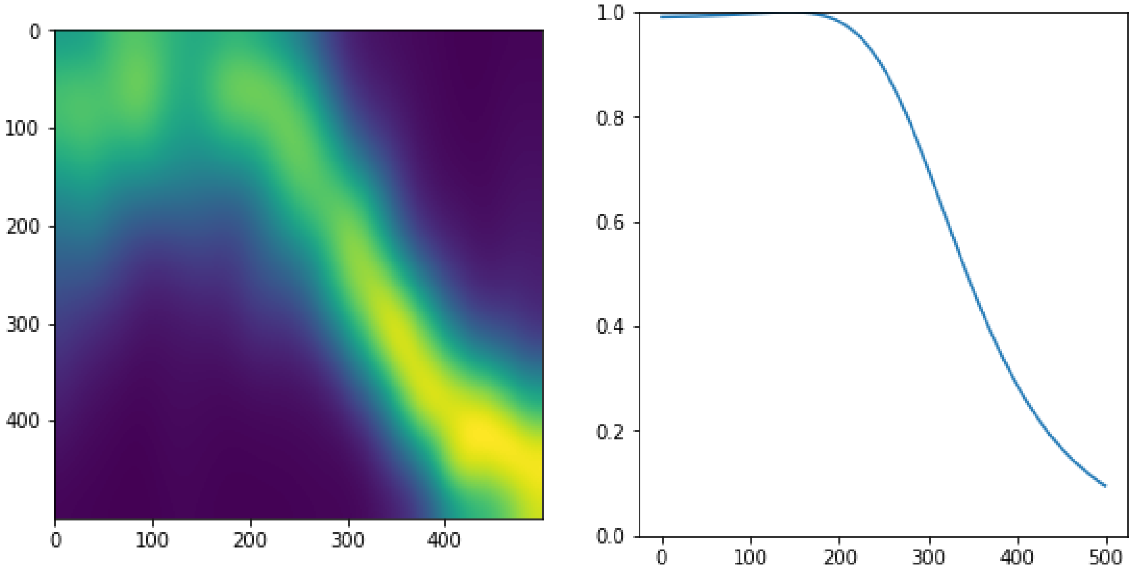
In [212]:

```
for i in range(20):
    #plt.plot(gaussian_filter1d(y_test[i,:500],40))
    x=range(N_OUTPUT)
    y=np.arange(0,1,0.002)
    c=np.zeros((len(y),N_OUTPUT))
    pis=softmax(y_test[i,-N_MIXES:])

    for j in range(N_MIXES):
        mus=y_test[i,j*N_OUTPUT:N_OUTPUT*(j+1)]
        sigs=y_test[i,N_OUTPUT*(N_MIXES+j):N_OUTPUT*(N_MIXES+j+1)]
        for k in range(N_OUTPUT):
            #c[:,k]+=pis[j]*np.exp(-(y-mus[k])**2/sigs[k]/2)/np.sqrt(2*np.pi*sig
s[k])
            c[:,k]+=pis[j]*np.exp(-(y-mus[k])**2/sigs[k]**2/2)/np.sqrt(2*np.pi)/
sigs[k]
    c=gaussian_filter(c,20)
    c=np.flipud(c)

    f=plt.figure(figsize=(10,5))
    ax1 = f.add_subplot(1,2,1, aspect=1)
    ax2 = f.add_subplot(1,2,2)

    ax1.imshow(c)
    ax2.plot(y_data[i,:])
    plt.ylim((0,1))
```

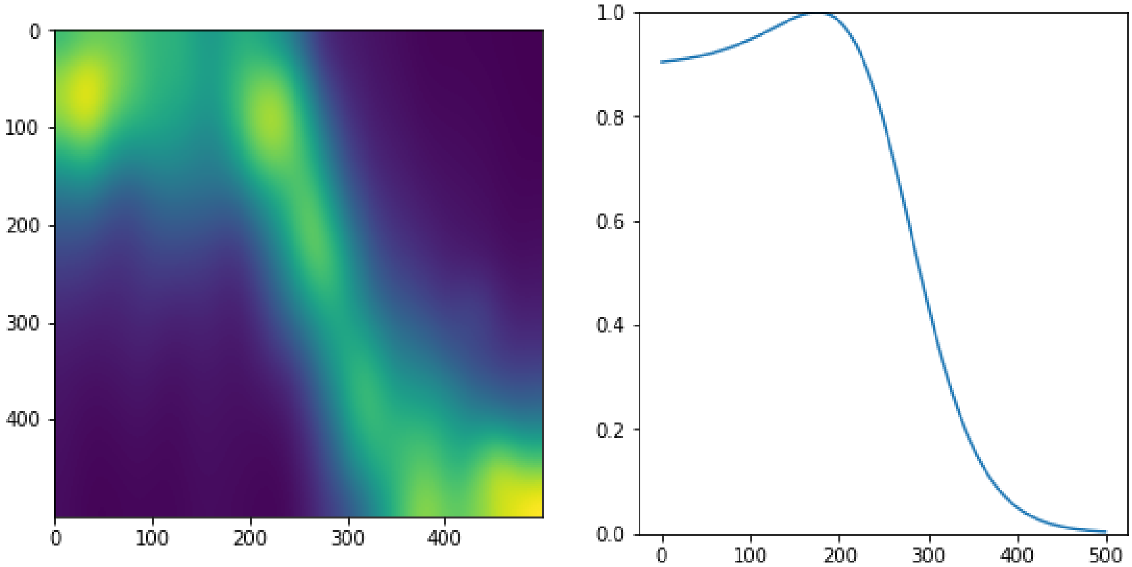

0 100 200 300 400
0 100 200 300 400
1.0 0.8 0.6 0.4 0.2 0.0
0 100 200 300 400 500
0 100 200 300 400
0 100 200 300 400
1.0 0.8 0.6 0.4 0.2 0.0
0 100 200 300 400 500

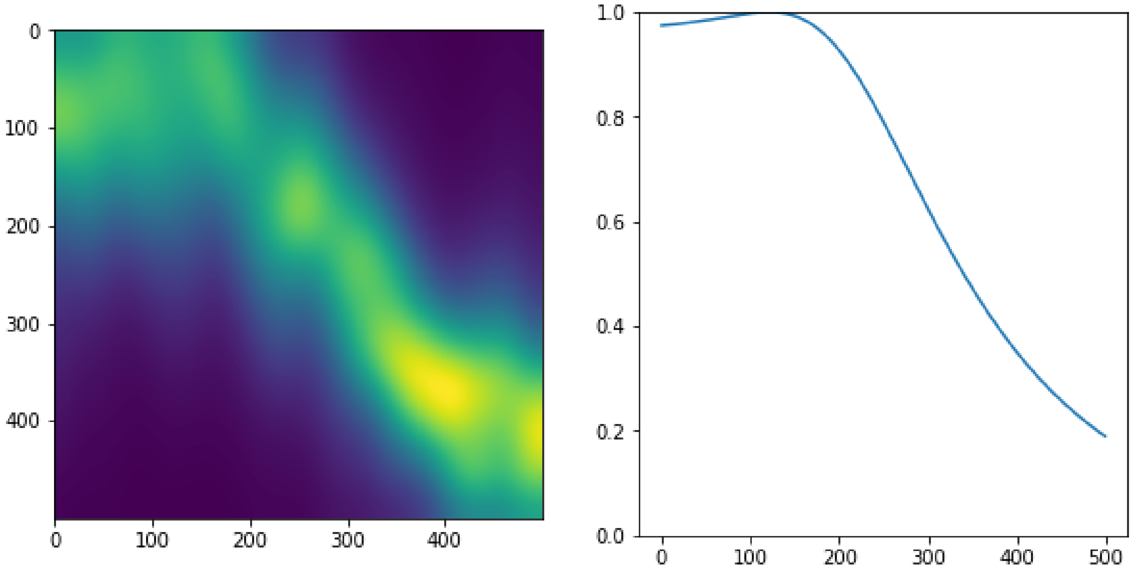


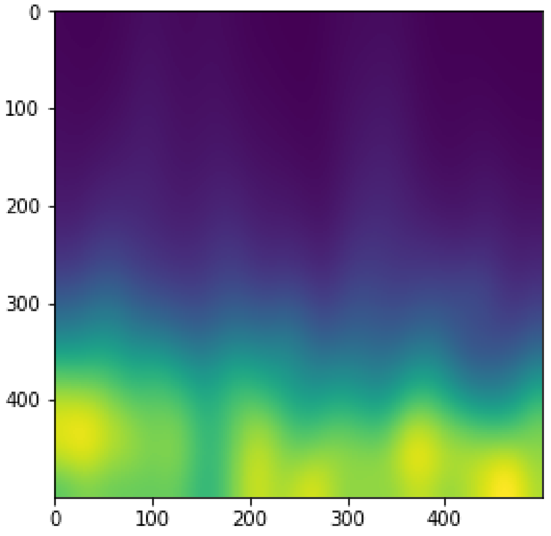
0
100
200
300
400
0 100 200 300 400
1.0
0.8
0.6
0.4
0.2
0.0
0 100 200 300 400 500

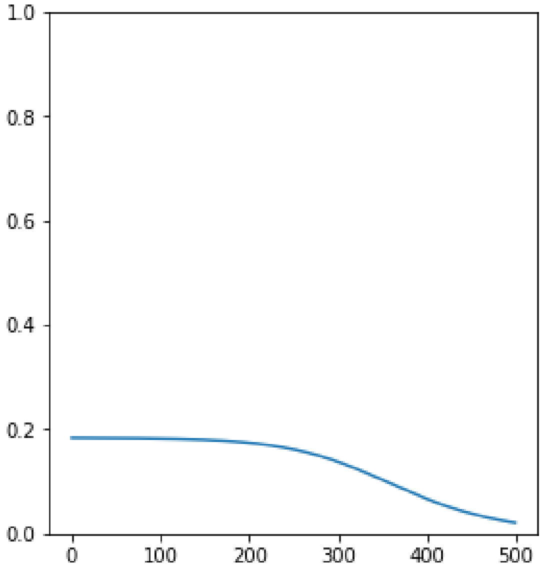



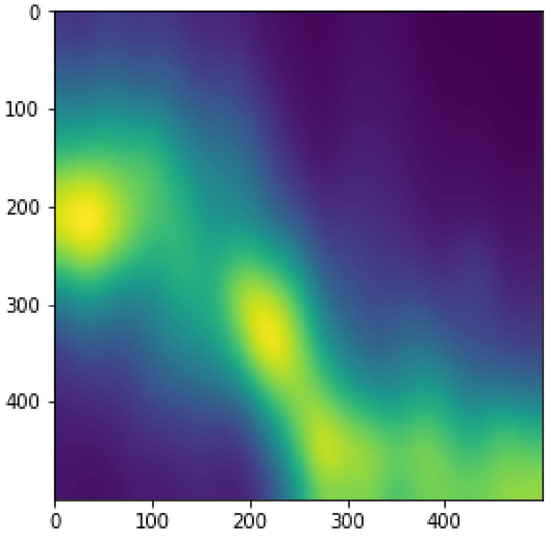
0
100
200
300
400
0 100 200 300 400
1.0
0.8
0.6
0.4
0.2
0.0
0 100 200 300 400 500

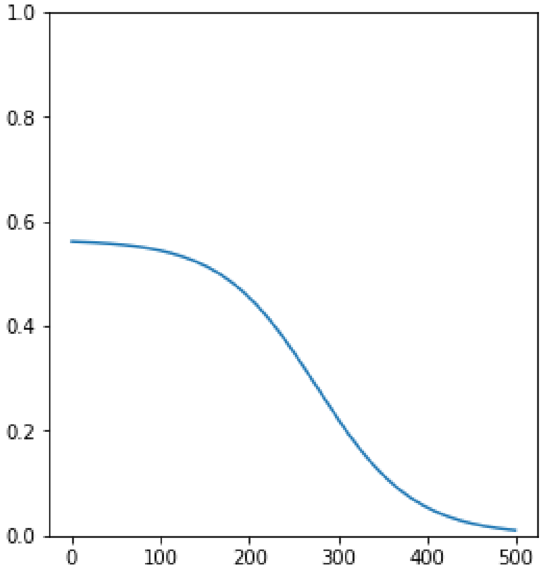

0
100
200
300
400
0 100 200 300 400
1.0
0.8
0.6
0.4
0.2
0.0
0 100 200 300 400 500

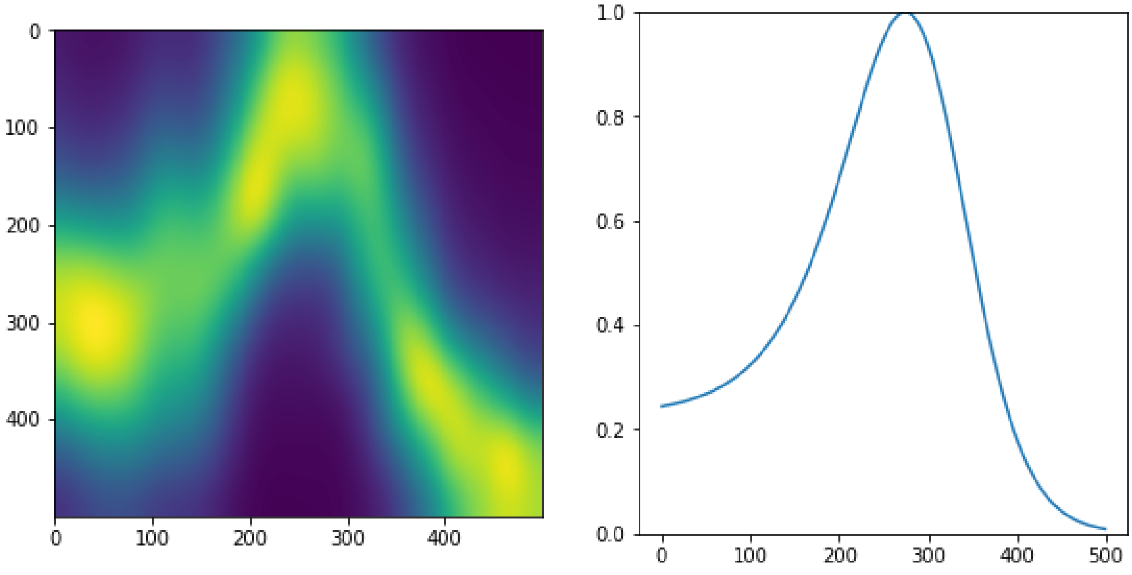

0
100
200
300
400
0 100 200 300 400
1.0
0.8
0.6
0.4
0.2
0.0
0 100 200 300 400 500

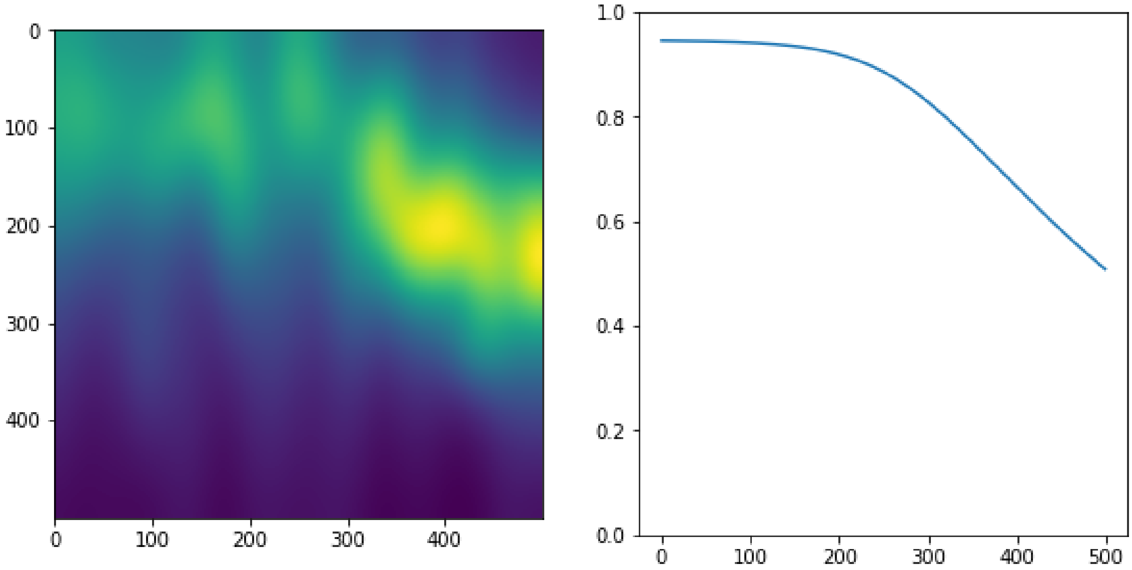

0
100
200
300
400
0
100
200
300
400

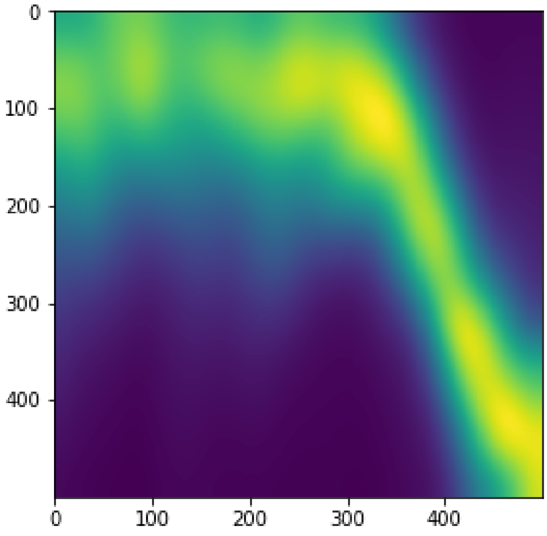

1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500

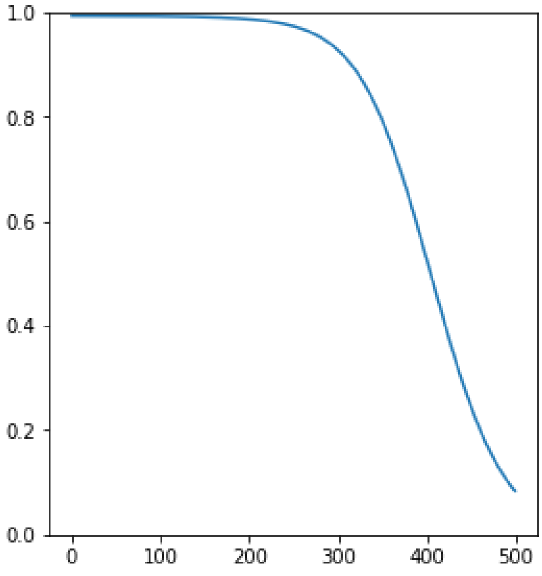

0
100
200
300
400
0
100
200
300
400

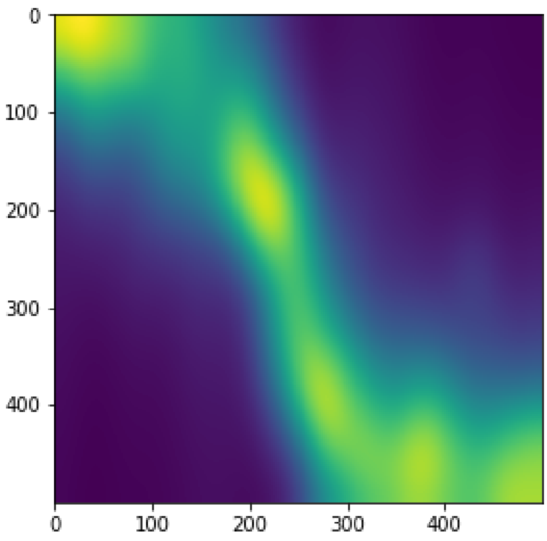

1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500

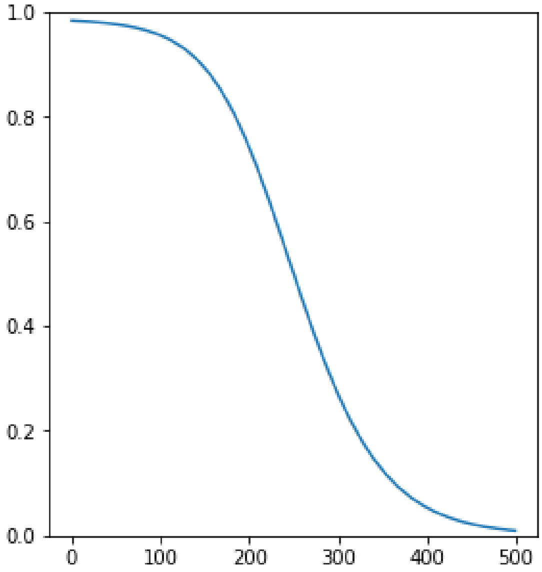

0
100
200
300
400
0
100
200
300
400
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500

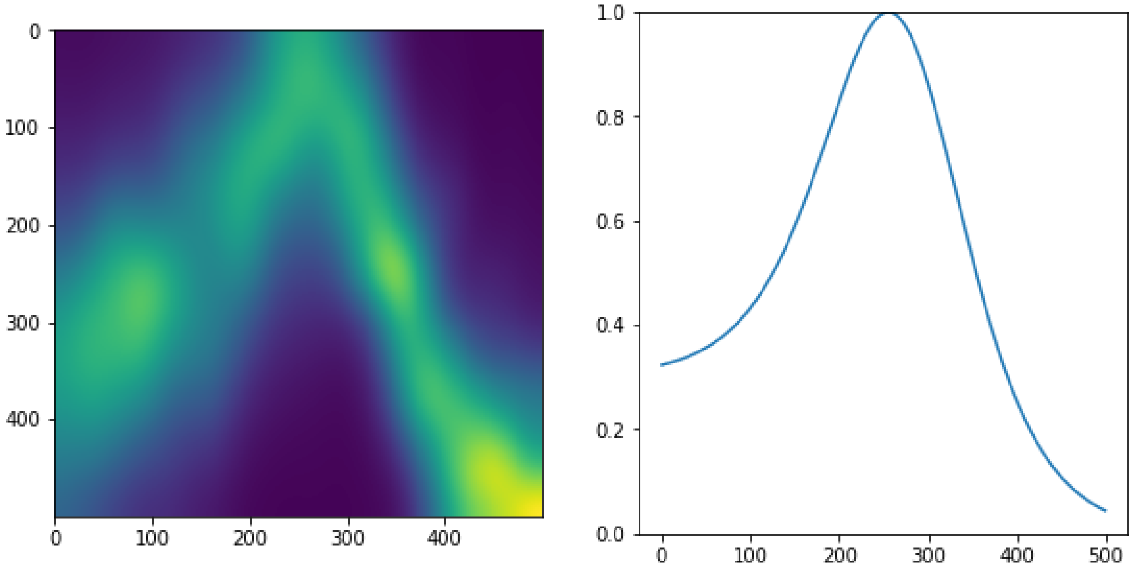

0
100
200
300
400
0
100
200
300
400
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500

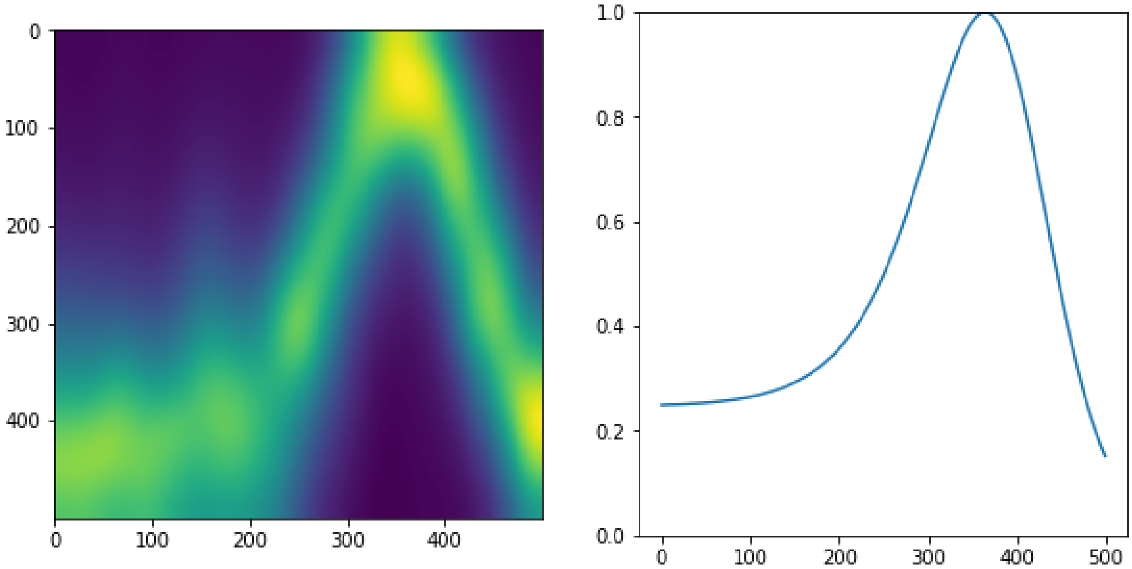

0
100
200
300
400
0
100
200
300
400

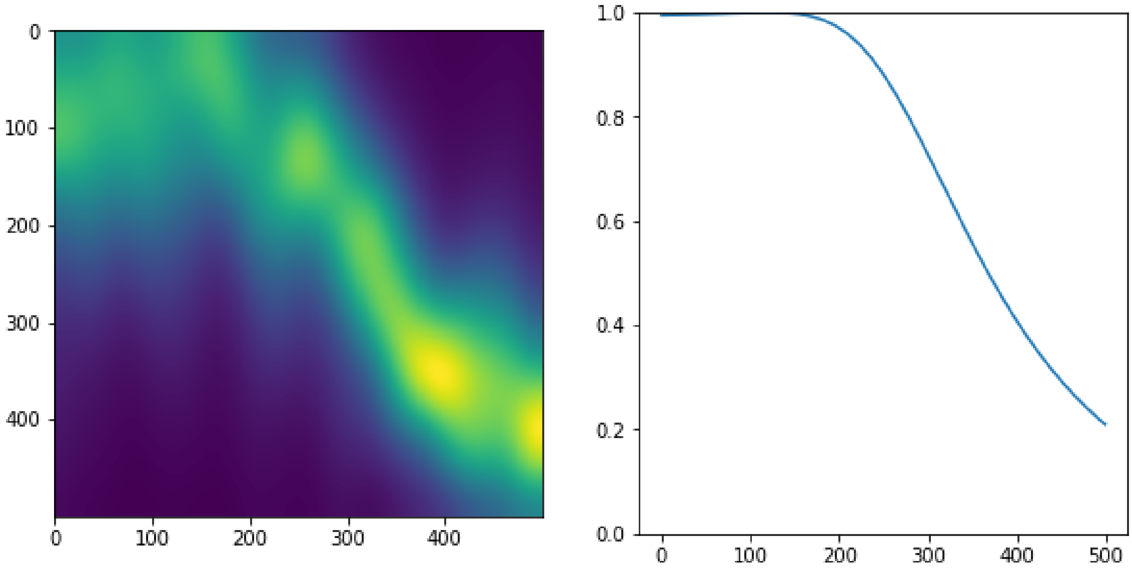

1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500

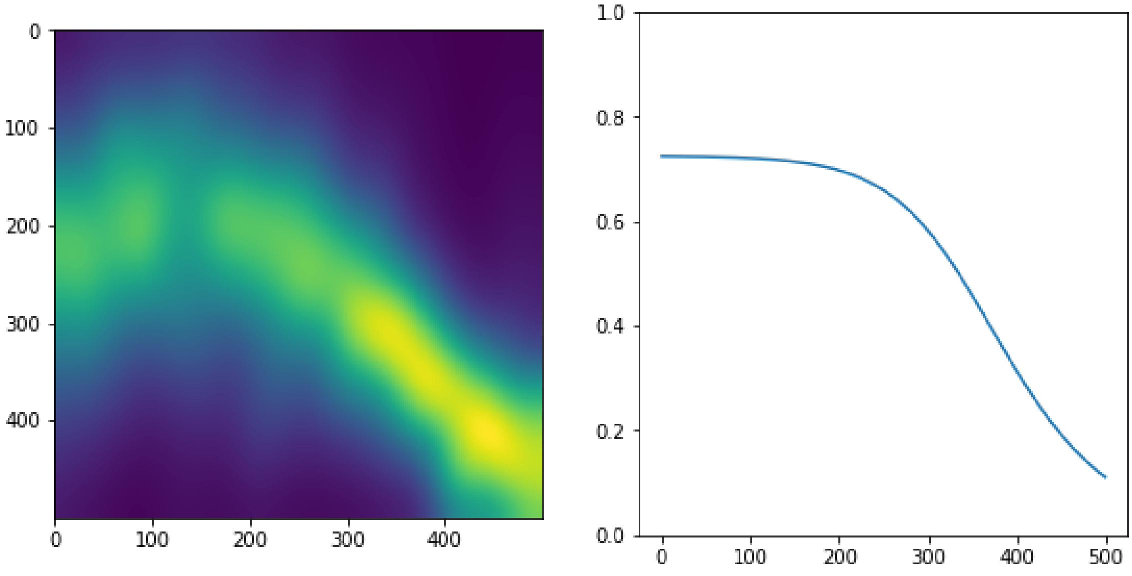

0
100
200
300
400
0
100
200
300
400

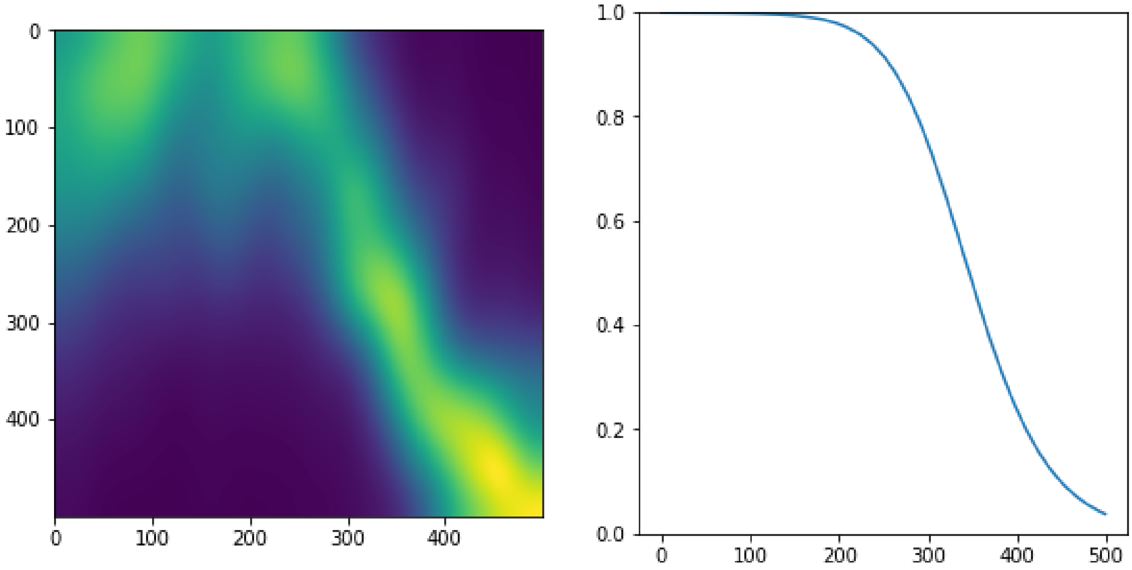

1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500

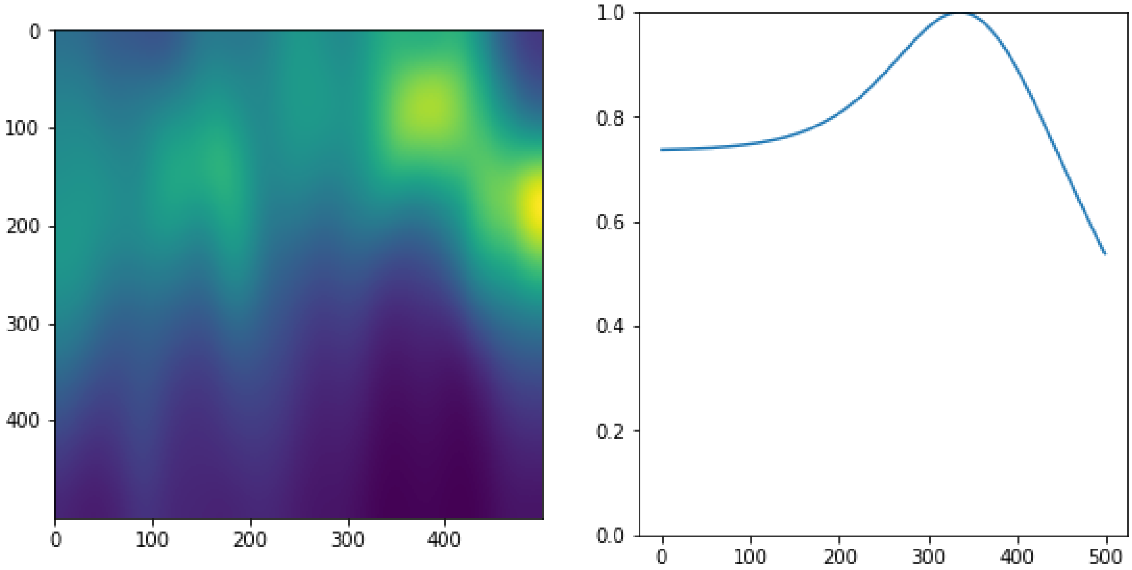

0
100
200
300
400
0 100 200 300 400
1.0
0.8
0.6
0.4
0.2
0.0
0 100 200 300 400 500

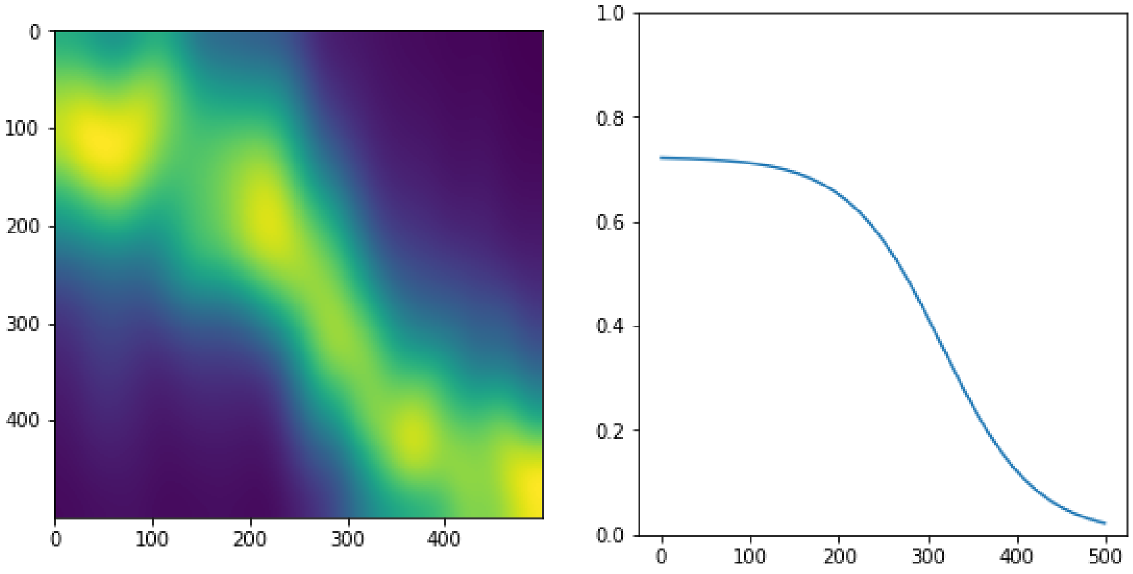

0
100
200
300
400
0 100 200 300 400
1.0
0.8
0.6
0.4
0.2
0.0
0 100 200 300 400 500

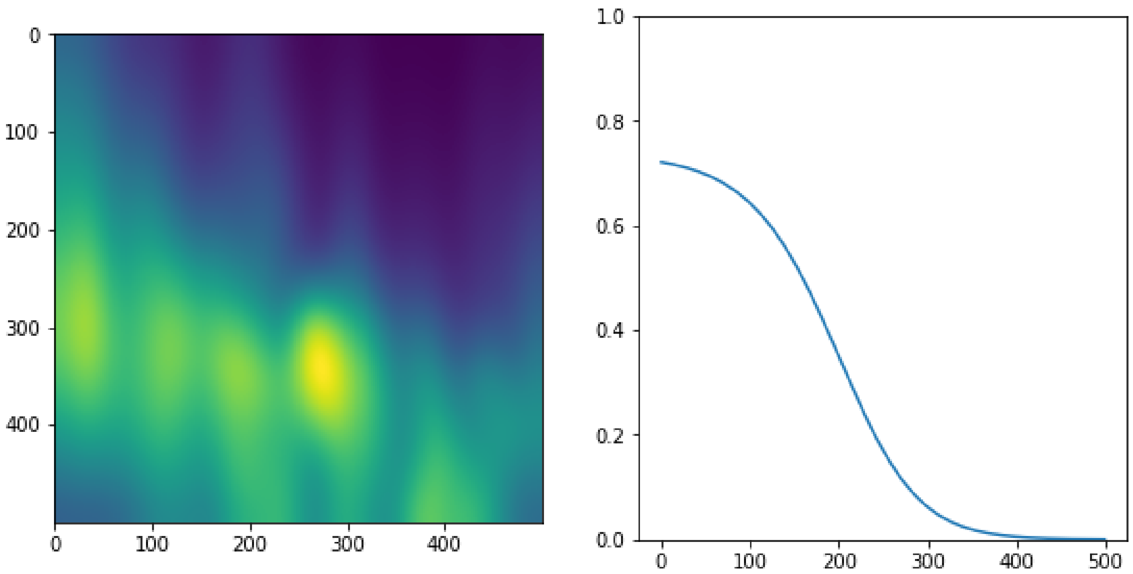

0
100
200
300
400
0
100
200
300
400
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500


0
100
200
300
400
0
100
200
300
400
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500


0
100
200
300
400
0 100 200 300 400
1.0
0.8
0.6
0.4
0.2
0.0
0 100 200 300 400 500


0
100
200
300
400
0 100 200 300 400
1.0
0.8
0.6
0.4
0.2
0.0
0 100 200 300 400 500


0
100
200
300
400
0 100 200 300 400
1.0
0.8
0.6
0.4
0.2
0.0
0 100 200 300 400 500


0
100
200
300
400
0 100 200 300 400
1.0
0.8
0.6
0.4
0.2
0.0
0 100 200 300 400 500

In [ ]:

```

```