In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import tensorflow as tf
import tensorflow.keras.backend as K
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import LSTM, Dense, TimeDistributed
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

In [2]:
```
best_model = tf.keras.models.load_model("../best_model.h5")
```

In [3]:
```
best_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 20, 50)            11400
_________________________________________________________________
lstm_1 (LSTM)                (None, 20, 50)            20200
_________________________________________________________________
time_distributed (TimeDistri (None, 20, 6)             306
=================================================================
Total params: 31,906
Trainable params: 31,906
Non-trainable params: 0
_________________________________________________________________
```

In [4]:
```
# get the weights
weights_list = best_model.weights
```

In [5]:
```
len(weights_list)
```

Out[5]: 8

In [6]:
```
1 weights_list
```

Out[6]:
```
[<tf.Variable 'lstm/kernel:0' shape=(6, 200) dtype=float32, numpy=
 array([[-0.34732938,  0.02779082, -0.038332  , ...,  0.73004186,
         -0.03486849,  0.2486751 ],
        [ 0.03394089, -0.05550505, -0.08837266, ..., -0.61472344,
         -0.00978172, -0.2613679 ],
        [ 0.16310589,  0.12231547, -0.15671289, ..., -0.48992842,
          0.11285017, -0.20759155],
        [-0.00661713, -0.01501527,  0.07164232, ..., -0.1606346 ,
         -0.08436577,  0.05285156],
        [ 0.11169901, -0.05725458,  0.07910605, ...,  0.04379767,
         -0.14315079, -0.04795304],
        [-0.11409782, -0.07604562, -0.06896029, ..., -0.0701544 ,
         -0.13497025,  0.00825712]], dtype=float32)>,
 <tf.Variable 'lstm/recurrent_kernel:0' shape=(50, 200) dtype=float32, numpy=
 array([[ 0.15857396,  0.03394932,  0.4070957 , ...,  0.16254635,
         -0.0875672 , -0.04026914],
        [-0.27428126, -0.0012079 , -0.42910308, ..., -0.2849312 ,
         -0.03275565,  0.01087226],
        [-0.29858056, -0.04129125, -0.4602354 , ..., -0.13831714,
```

In [7]:
```
1 # get the weights array
2 kernel0 = K.get_value(weights_list[0])
3 kernel0
```

Out[7]:
```
array([[-0.34732938,  0.02779082, -0.038332  , ...,  0.73004186,
        -0.03486849,  0.2486751 ],
       [ 0.03394089, -0.05550505, -0.08837266, ..., -0.61472344,
        -0.00978172, -0.2613679 ],
       [ 0.16310589,  0.12231547, -0.15671289, ..., -0.48992842,
         0.11285017, -0.20759155],
       [-0.00661713, -0.01501527,  0.07164232, ..., -0.1606346 ,
        -0.08436577,  0.05285156],
       [ 0.11169901, -0.05725458,  0.07910605, ...,  0.04379767,
        -0.14315079, -0.04795304],
       [-0.11409782, -0.07604562, -0.06896029, ..., -0.0701544 ,
        -0.13497025,  0.00825712]], dtype=float32)
```

Model set up

In [8]:
```
1 from myLSTMclass import myLSTM
2 from myLSTMclass import RNN_VE
```

In [9]:
```
1 myModel=myLSTM(50,6,6,"../best_model.h5")
```

Check the layers of the RNNnet

In [10]:
```
1 myModel.myRNNet.layers
```

Out[10]:
```
[<tensorflow.python.keras.engine.input_layer.InputLayer at 0x17285730108>,
 <tensorflow.python.keras.layers.recurrent_v2.LSTM at 0x1727bfcbc88>,
 <tensorflow.python.keras.layers.recurrent_v2.LSTM at 0x17285725348>,
 <tensorflow.python.keras.layers.wrappers.TimeDistributed at 0x1728572f988>]
```

In [11]:
```
1 myModel.myRNNet.layers[1].states
```

Out[11]:
```
[None, None]
```

In [12]:
```
1 myModel.myRNNet.layers[2].states
```

Out[12]:
```
[None, None]
```

In [13]:
```
1 myModel.myRNNet.weights
```

Out[13]:
```
[<tf.Variable 'lstm_3/kernel:0' shape=(6, 200) dtype=float32, numpy=
 array([[-0.34732938,  0.02779082, -0.038332  , ...,  0.73004186,
         -0.03486849,  0.2486751 ],
        [ 0.03394089, -0.05550505, -0.08837266, ..., -0.61472344,
         -0.00978172, -0.2613679 ],
        [ 0.16310589,  0.12231547, -0.15671289, ..., -0.48992842,
          0.11285017, -0.20759155],
        [-0.00661713, -0.01501527,  0.07164232, ..., -0.1606346 ,
         -0.08436577,  0.05285156],
        [ 0.11169901, -0.05725458,  0.07910605, ...,  0.04379767,
         -0.14315079, -0.04795304],
        [-0.11409782, -0.07604562, -0.06896029, ..., -0.0701544 ,
         -0.13497025,  0.00825712]], dtype=float32)>,
 <tf.Variable 'lstm_3/recurrent_kernel:0' shape=(50, 200) dtype=float32, numpy=
 array([[ 0.15857396,  0.03394932,  0.4070957 , ...,  0.16254635,
         -0.0875672 , -0.04026914],
        [-0.27428126, -0.0012079 , -0.42910308, ..., -0.2849312 ,
         -0.03275565,  0.01087226],
        [-0.29858056, -0.04129125, -0.4602354 , ..., -0.13831714,
```

In [15]:
```
1 # get the weights array
2 kernel0 = K.get_value(myModel.myRNNet.weights[0])
3 kernel0
```

Out[15]:
```
array([[-0.34732938,  0.02779082, -0.038332  , ...,  0.73004186,
        -0.03486849,  0.2486751 ],
       [ 0.03394089, -0.05550505, -0.08837266, ..., -0.61472344,
        -0.00978172, -0.2613679 ],
       [ 0.16310589,  0.12231547, -0.15671289, ..., -0.48992842,
         0.11285017, -0.20759155],
       [-0.00661713, -0.01501527,  0.07164232, ..., -0.1606346 ,
        -0.08436577,  0.05285156],
       [ 0.11169901, -0.05725458,  0.07910605, ...,  0.04379767,
        -0.14315079, -0.04795304],
       [-0.11409782, -0.07604562, -0.06896029, ..., -0.0701544 ,
        -0.13497025,  0.00825712]], dtype=float32)
```

So that the set values of the weights are correct.

In [16]:
```
# import the strain stress
strains = np.load("extra_strains3d.npy")
stress_true = np.load("extra_stress3d_true.npy")
```

In [17]:
```
strains.shape, stress_true.shape
```

Out[17]: ((2, 40, 6), (2, 40, 6))

In [18]:
```
strains.dtype, stress_true.dtype
```

Out[18]: (dtype('float64'), dtype('float64'))

In [19]:
```
strain1 = strains[0, :, :]
strain2 = strains[1, :, :]
```

In [20]:
```
strain1 = strain1.astype(np.float32)
strain2 = strain2.astype(np.float32)
```

In [21]:
```
strain1.shape
```

Out[21]: (40, 6)

In [23]:
```
stress1 = RNN_VE(strain1, myModel, states=None)
stress1 = np.reshape(stress1, (40,6))
```

In [24]:
```
time_vec = np.linspace(0,2,40,endpoint=True)
```

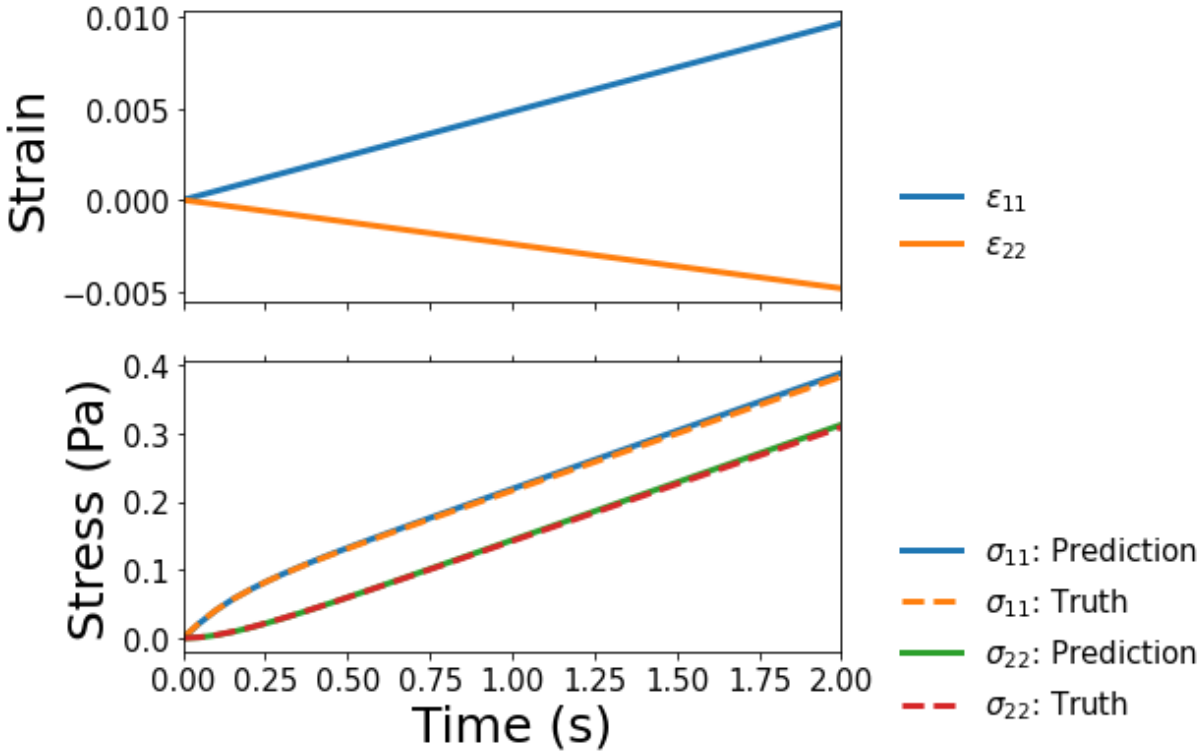
In [29]:

```
# plot
labelsize = 25
ticksize = 15

fig, axs = plt.subplots(2, figsize=(6, 6))
axs[0].plot(time_vec, strain1[:,0], lw=3.0, label="$\epsilon_{11}$")
axs[0].plot(time_vec, strain1[:,1], lw=3.0, label="$\epsilon_{22}$")
axs[0].legend(frameon=False, fontsize=ticksize, bbox_to_anchor=(1.05, 0.5))
axs[0].set_xlim(0,2)
axs[0].set_ylabel("Strain", fontsize=labelsize)
axs[0].tick_params(axis='x', labelbottom=False)
axs[0].tick_params(axis='y', labelsize=ticksize)

axs[1].plot(time_vec, stress1[:,0], lw=3.0, label="$\sigma_{11}$: Prediction")
axs[1].plot(time_vec, stress_true[0,:,0], lw=3.0, linestyle='dashed', label="$\sigma_
axs[1].plot(time_vec, stress1[:,1], lw=3.0, label="$\sigma_{22}$: Prediction")
axs[1].plot(time_vec, stress_true[0,:,1], lw=3.0, linestyle='dashed', label="$\sigma_
axs[1].legend(fontsize = ticksize, frameon=False, bbox_to_anchor=(1.05, 0.5))
axs[1].set_xlim(0,2)
axs[1].set_xlabel("Time (s)", fontsize=labelsize)
axs[1].set_ylabel("Stress (Pa)", fontsize=labelsize)
axs[1].tick_params(axis='x', top=True)
plt.xticks(fontsize=ticksize)
plt.yticks(fontsize=ticksize)

# save the plot as a file
fig.savefig('linear.png', dpi=800, bbox_inches='tight')
fig.savefig('linear.eps', format='eps', dpi=800, bbox_inches='tight')
```



In [26]:

```
stress2 = RNN_VE(strain2, myModel, states=None)
stress2 = np.reshape(stress2, (40,6))
```

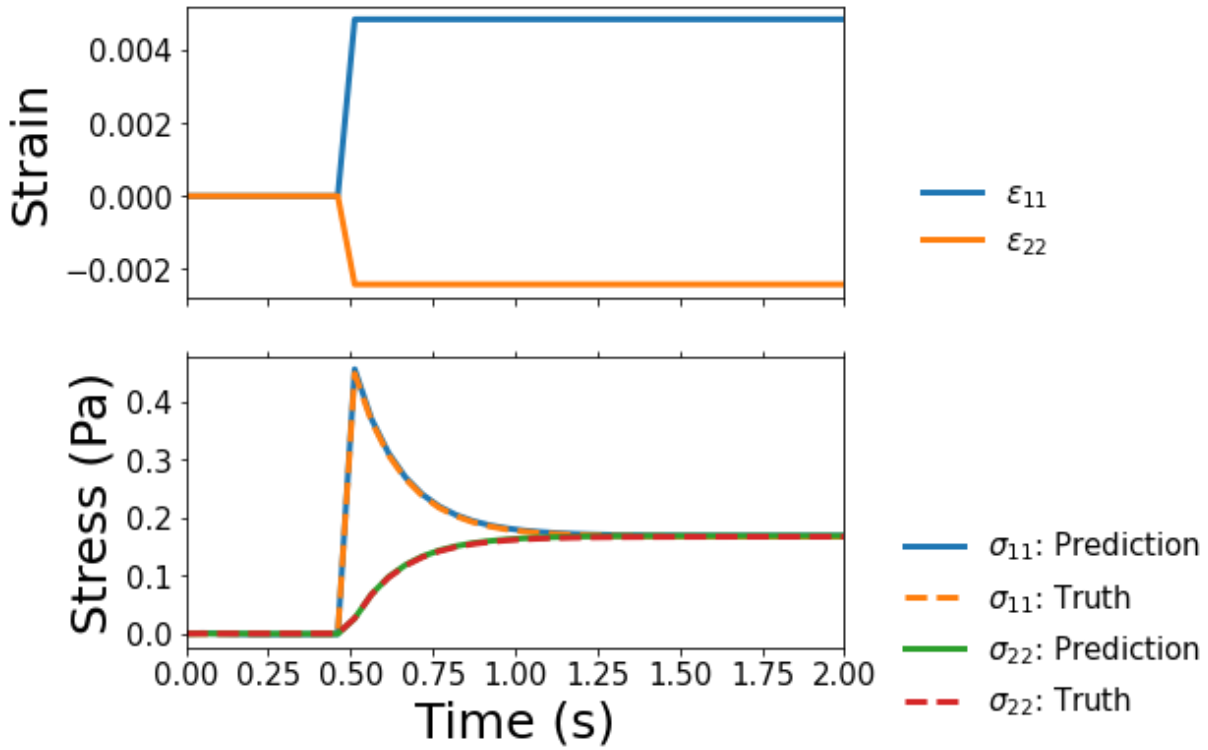
In [34]:

```
# plot
labelsize = 25
ticksize = 15

fig, axs = plt.subplots(2, figsize=(6, 6))
axs[0].plot(time_vec, strain2[:,0], lw=3.0, label="$\epsilon_{11}$")
axs[0].plot(time_vec, strain2[:,1], lw=3.0, label="$\epsilon_{22}$")
axs[0].set_xlim(0,2)
axs[0].set_ylabel("Strain", fontsize=labelsize)
axs[0].tick_params(axis='x', labelbottom=False)
axs[0].tick_params(axis='y', labelsize=ticksize)
axs[0].legend(frameon=False, fontsize=ticksize, bbox_to_anchor=(1.35, 0.5))

axs[1].plot(time_vec, stress2[:,0], lw=3.0, label="$\sigma_{11}$: Prediction")
axs[1].plot(time_vec, stress_true[1,:,0], lw=3.0, linestyle='dashed', label="$\sigma_
axs[1].plot(time_vec, stress2[:,1], lw=3.0, label="$\sigma_{22}$: Prediction")
axs[1].plot(time_vec, stress_true[1,:,1], lw=3.0, linestyle='dashed', label="$\sigma_
axs[1].legend(fontsize = ticksize, frameon=False, bbox_to_anchor=(1.05, 0.5))
axs[1].set_xlim(0,2)
axs[1].set_xlabel("Time (s)", fontsize=labelsize)
axs[1].set_ylabel("Stress (Pa)", fontsize=labelsize)
axs[1].tick_params(axis='x', top=True)
plt.xticks(fontsize=ticksize)
plt.yticks(fontsize=ticksize)

# save the plot as a file
fig.savefig('step.png', dpi=800, bbox_inches='tight')
fig.savefig('step.eps', format='eps', dpi=800, bbox_inches='tight')
```



In [ ]:

```
1
```