Q2)

Apply a GLM model (without considering the HRF function) to test the effects of eyes open, eyes close, and (eyes open - eyes close) on a random voxel signal. Interpret and discuss your results.

In [ ]:
```
# Import the requred Packages
import os
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import nibabel as nib


fmri_file = '../../datasets/fMRI/HW4/sub-001_ses-001_task-eoec_bold.nii.gz' # Get the nifti file
img = nib.load(fmri_file) # Load in the nifti file
```

The Predictors

The numbers used in the predictiors came from the content in the tsv file.

In [2]:
```
# TSV File content

# onset duration        trial_type
# 0     20      EC
# 20    20      EO
# 40    20      EC
# 60    20      EO
# 80    20      EC
# 100   20      EO
# 120   20      EC
# 140   20      EO
# 160   20      EC
# 180   20      EO
# 200   20      EC
# 220   20      EO

n_timepoints = 240  # Total timepoints
time = np.arange(n_timepoints) # vector of timepoint values of length n_timepoints

# Create binary regressors
design_EC = np.zeros(n_timepoints) # eyes closed vector
design_EO = np.zeros(n_timepoints) # eyes open vector

# Populate ranges based on TSV file
design_EC[0:20] = -1 # Set the value of first occurance of Eye Closed
design_EC[40:60] = -1 # Set the value of second occurance of Eye Closed
design_EC[80:100] = -1 # Set the value of third occurance of Eye Closed
design_EC[120:140] = -10 # Set the value of fourth occurance of Eye Closed
design_EC[160:180] = -15 # Set the value of fifth occurance of Eye Closed
design_EC[200:220] = -10 # Set the value of sixth occurance of Eye Closed

design_EO[20:40] = 20 # Set the value of first occurance of Eye Open
design_EO[60:80] = 15 # Set the value of second occurance of Eye Open
design_EO[100:120] = 20 # Set the value of third occurance of Eye Open
design_EO[140:160] = 10 # Set the value of fourth occurance of Eye Open
design_EO[180:200] = 10 # Set the value of fifth occurance of Eye Open
design_EO[220:240] = 10 # Set the value of sixth occurance of Eye Open
```

GLM (no HRF) Details

The GLM here was made with the knowledge given in the tsv file. That being that each interval of eyes open and eyes closed lasted around 20 second and therefore sets those ranges to a particular value (so instead of 0 being the only value set, 0, 1, ..., 19 are being set as well). This does make the model more **blocky** looking, but it makes better generalizations. For example, "This First Eyes Open Section has higher general activation compared to the average". Another thing is that the eyes closed values are negative and the eyes open values are positive. This is because my general hypothesis that this model tests is that if the eyes are closed, there is less brain activity going on there and therefore will be lower activation than average. The values being set are of a ratio of numbers that generally fits the random voxel signal, but is not so specific where it overfits with the signal to get an R^2 of like 1 for example, but still could overfit to the random voxel signal if that random voxel signal is much different from other voxel signals. For now we will make the assumption that this voxel signal is not that unique because it would simply make little sense if only one voxel of a 64x64x35 resolution was unique vs a general area. The same can also be said for setting events (eyes open and eyes closed occurance number). I also want to mention even though it's obvious for this assignment that this data is not preprocessed, leading to more noise and therefore a higher error term (a lower R^2 value). Regardless we have set the stage for the model and will now examine it.

In [3]:
```python
data = img.get_fdata() # Get the 4 dimentional data from the fMRI
voxel_signal = data[0, 0, 0, :] # Look at the first voxel's time signal
print(voxel_signal) # See the values of the voxel
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

The voxel of (0, 0, 0) doesn't have any signal because it's not in the area of the brain, now let's pick better location.

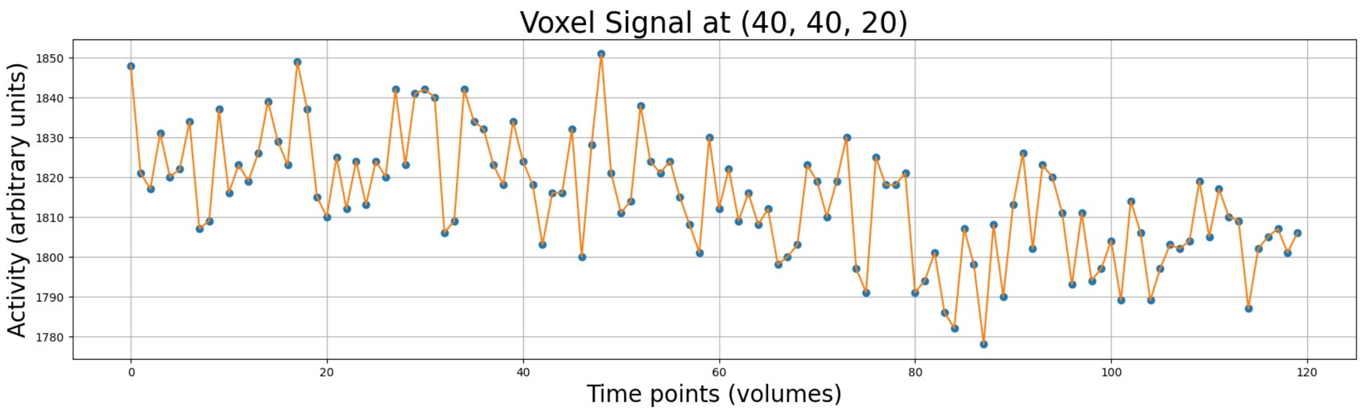
In [4]:
```python
voxel_signal = data[40, 40, 20, :] # Look at a random voxel's time signal
print(voxel_signal) # Look at the random voxel's time signal
```

```
[1848. 1821. 1817. 1831. 1820. 1822. 1834. 1807. 1809. 1837. 1816. 1823.
 1819. 1826. 1839. 1829. 1823. 1849. 1837. 1815. 1810. 1825. 1812. 1824.
 1813. 1824. 1820. 1842. 1823. 1841. 1842. 1840. 1806. 1809. 1842. 1834.
 1832. 1823. 1818. 1834. 1824. 1818. 1803. 1816. 1816. 1832. 1800. 1828.
 1851. 1821. 1811. 1814. 1838. 1824. 1821. 1824. 1815. 1808. 1801. 1830.
 1812. 1822. 1809. 1816. 1808. 1812. 1798. 1800. 1803. 1823. 1819. 1810.
 1819. 1830. 1797. 1791. 1825. 1818. 1818. 1821. 1791. 1794. 1801. 1786.
 1782. 1807. 1798. 1778. 1808. 1790. 1813. 1826. 1802. 1823. 1820. 1811.
 1793. 1811. 1794. 1797. 1804. 1789. 1814. 1806. 1789. 1797. 1803. 1802.
 1804. 1819. 1805. 1817. 1810. 1809. 1787. 1802. 1805. 1807. 1801. 1806.]
```

See Content of Voxel Signal at (40, 40, 20)

In [5]:
```python
# Generates a plot of a voxel's signal.

plt.figure(figsize=(20, 5)) # Make the figure size look presentable
plt.plot(voxel_signal, 'o') # Plot the voxel signal's numerical values with an 'o'
plt.plot(voxel_signal) # Plot the whole voxel signal
plt.xlabel('Time points (volumes)', fontsize=20) # Provide an understandable x label
plt.ylabel('Activity (arbitrary units)', fontsize=20) # Provide an understandable y label
plt.title('Voxel Signal at (40, 40, 20)', fontsize=25) # Provide an understandable title
plt.grid() # Display grid lines
plt.show() # Show the plot in the output cell
```



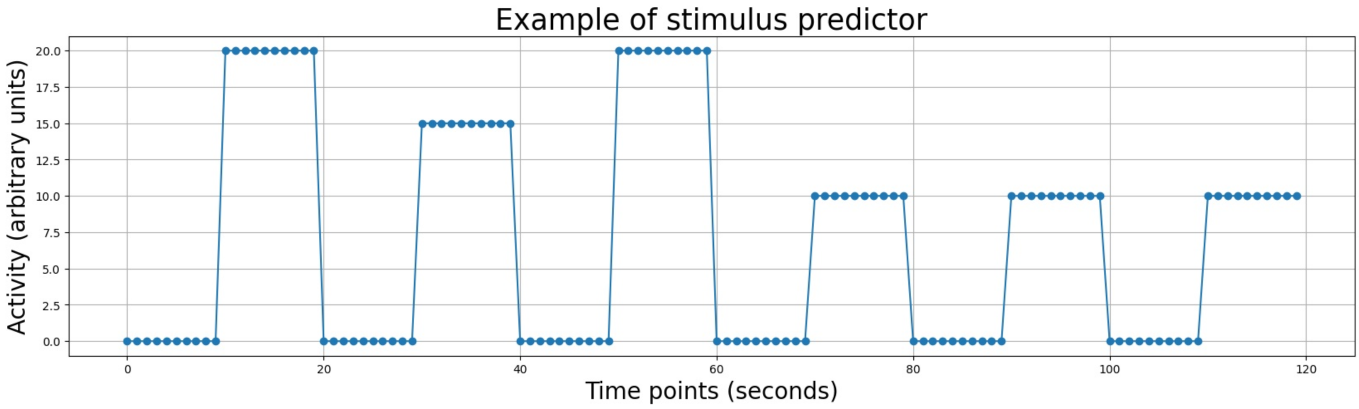
The signal looks valid and has good values, now we will apply the GLM to this.

See Preditor Model (eyes open)

In [6]:
```python
# Create the predictor varaible.

design_EO_downsampled = design_EO.reshape(-1, 2).mean(axis=1) # Downsample the eyes open vector
predictor_all = design_EO_downsampled # Assign the downsampled vector to the predictor_all variable
print("Shape of predictor: %s" % (predictor_all.shape,)) # Print out the shape of the predictor_all to make sure
plt.figure(figsize=(20, 5)) # Make the figure size look presentable
plt.plot(predictor_all, marker='o') # Plot the predictor's numerical values with an 'o' along with the whole pre
plt.xlabel('Time points (seconds)', fontsize=20) # Provide an understandable x label
plt.ylabel('Activity (arbitrary units)', fontsize=20) # Provide an understandable y label
plt.title('Example of stimulus predictor', fontsize=25) # Provide an understandable title
plt.grid() # Display grid lines
plt.show() # Show the plot in the output cell
```

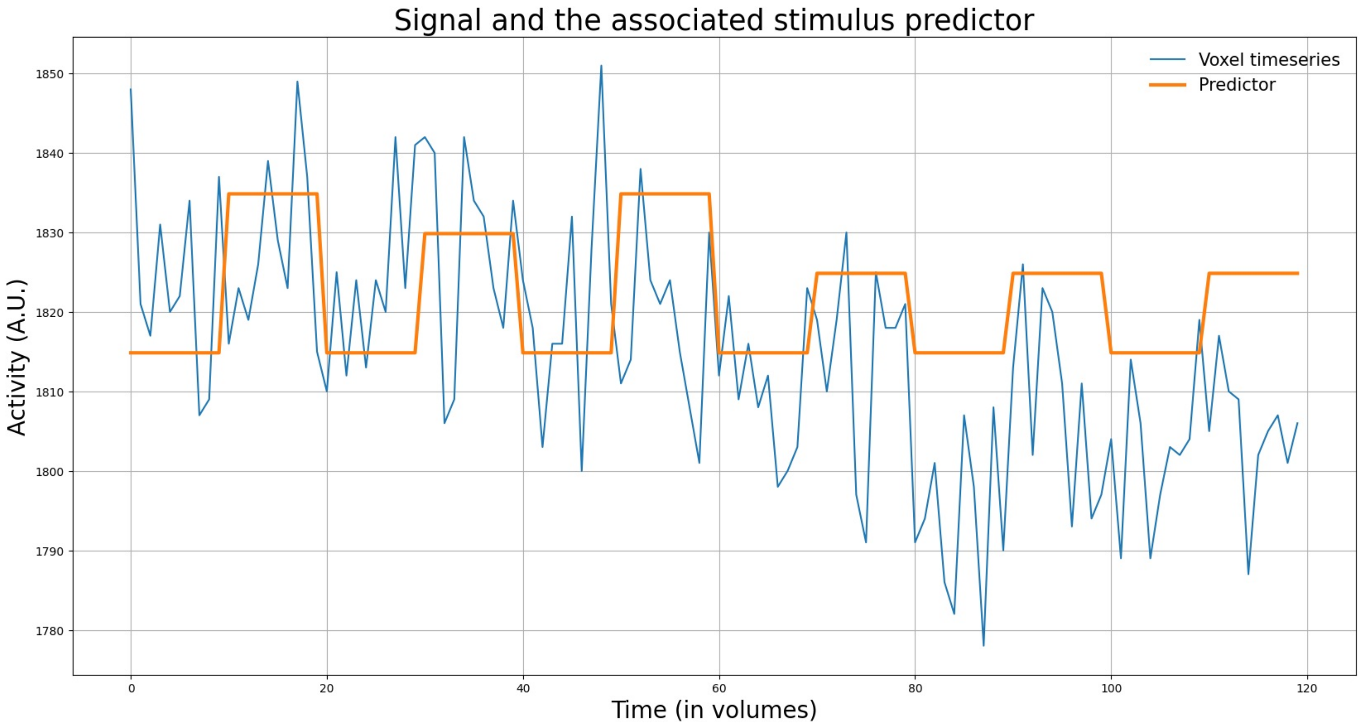
```
Shape of predictor: (120,)
```



This shows us a graphical look at the GLM model (eyes open) with our given input values at the beginning.

See Preditor Model (eyes open) with Voxel Signal

In [7]:
```
plt.figure(figsize=(20, 10)) # Make the figure size look presentable
plt.plot(voxel_signal) # Plot the whole voxel signal
plt.plot(predictor_all + voxel_signal.mean() , lw=3) # Plot the predicted signal
plt.xlabel('Time (in volumes)', fontsize=20) # Provide an understandable x label
plt.ylabel('Activity (A.U.)', fontsize=20) # Provide an understandable y label
plt.legend(['Voxel timeseries', 'Predictor'], fontsize=15, loc='upper right', frameon=False) # Display legend t
plt.title("Signal and the associated stimulus predictor", fontsize=25) # Provide an understandable title
plt.grid() # Display grid lines
plt.show() # Show the plot in the output cell
```



We can see that our model kind of get's the trend of the signal during the eyes open events, but it's definitally a rough estimate.

Test the Predictor Model (eyes open) with R^2

In [8]:
```
# Fit a regresison model to the predictor and target signal.

predictor_all_ds = predictor_all[:, np.newaxis] # Copy predictor and add an extra axis to it for R^2 calculatio
icept = np.ones((predictor_all_ds.size, 1)) # Generate template for intercepts
X_simple = np.hstack((icept, predictor_all_ds)) # Generate Simple Model with hstack b_0 and b_1
betas_simple = np.linalg.inv(X_simple.T @ X_simple) @ X_simple.T @ voxel_signal # Generate Optimal beta values
y_hat_simple = X_simple[:, 0] * betas_simple[0] + X_simple[:, 1] * betas_simple[1] # Generate the estimated y va
print(betas_simple) # Print the Optimal beta values
numerator = np.sum((voxel_signal - y_hat_simple) ** 2) # Get the MSE of y_hat
denominator = np.sum((voxel_signal - np.mean(voxel_signal)) ** 2) # Get the MSE of the voxel_signal mean
```

```
r_squared = 1 - numerator / denominator # Get R^2 value from y_hat
print('The R² value is: %.3f' % r_squared) # Print R^2 value
```

```
[1.81125216e+03 5.09106628e-01]
The R² value is: 0.070
```

Here we get an R^2 value of 0.070, this means 7.0% of the variance is explained through our model which is honestly pretty **bad**. However this result is expected with a regression model along with the data not being pre-processed at all.

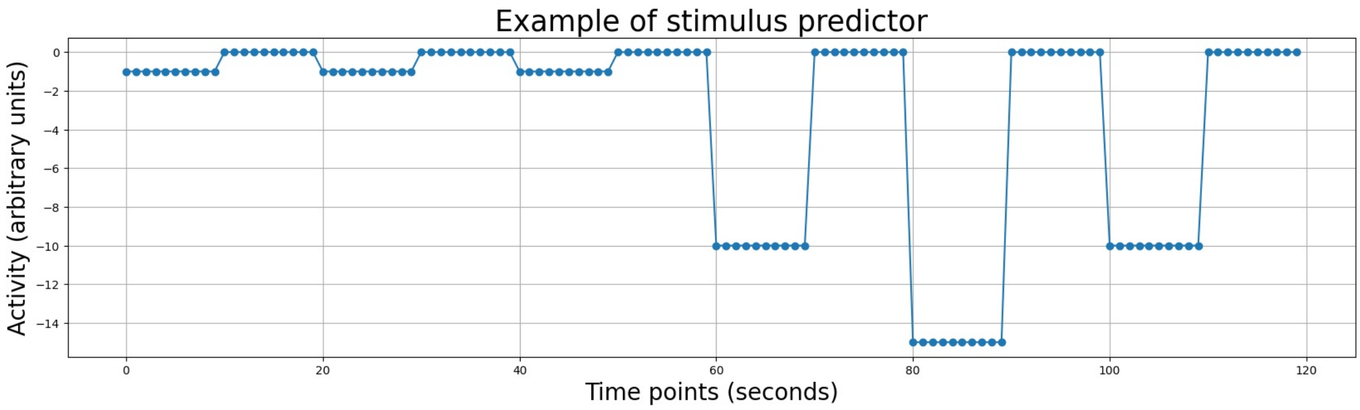
See Preditor Model (eyes closed)

In [9]:

```
# Create the predictor varaible.

design_EC_downsampled = design_EC.reshape(-1, 2).mean(axis=1) # Downsample the eyes closed vector
predictor_all = design_EC_downsampled # Assign the downsampled vector to the predictor_all variable
print("Shape of predictor: %s" % (predictor_all.shape,)) # Print out the shape of the predictor_all to make sur
plt.figure(figsize=(20, 5)) # Make the figure size look presentable
plt.plot(predictor_all, marker='o') # Plot the predictor's numerical values with an 'o' along with the whole pr
plt.xlabel('Time points (seconds)', fontsize=20) # Provide an understandable x label
plt.ylabel('Activity (arbitrary units)', fontsize=20) # Provide an understandable y label
plt.title('Example of stimulus predictor', fontsize=25) # Provide an understandable title
plt.grid() # Display grid lines
plt.show() # Show the plot in the output cell
```

```
Shape of predictor: (120,)
```

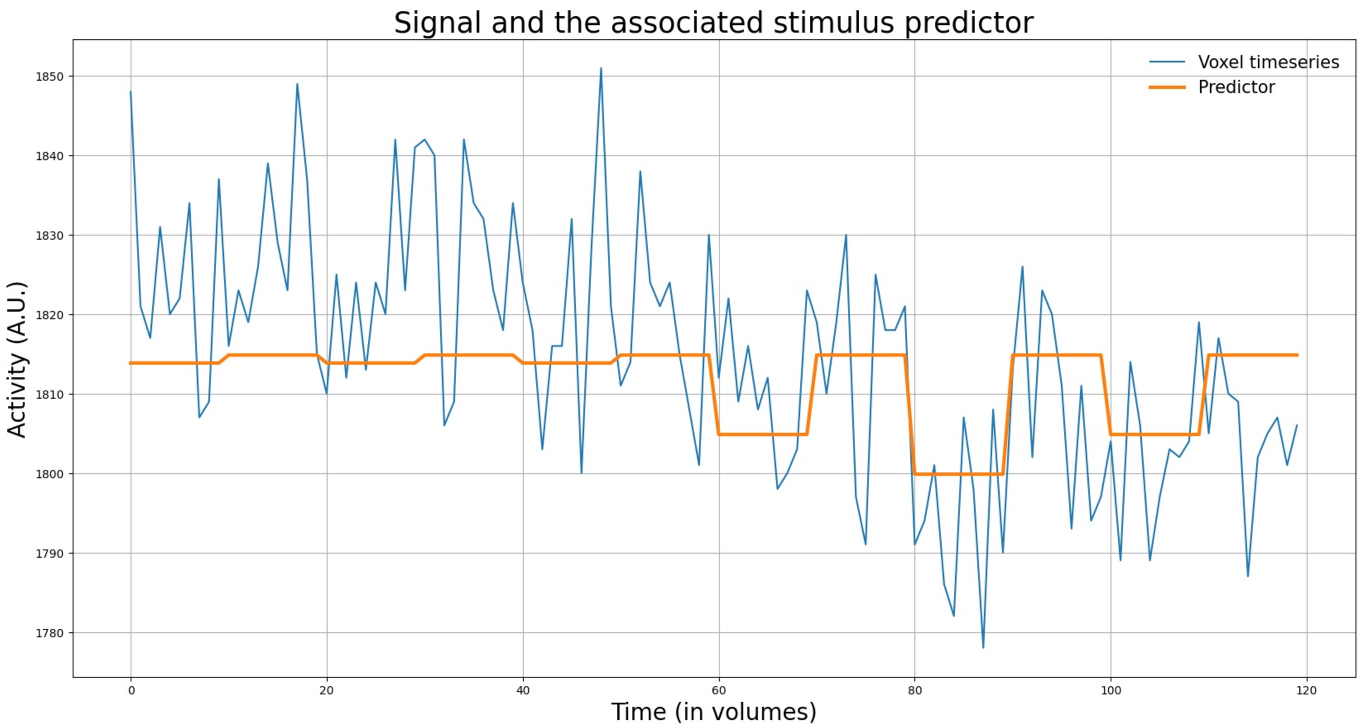


This shows us a graphical look at the GLM model (eyes closed) with our given input values at the beginning.

See Preditor Model (eyes closed) with Voxel Signal

In [10]:

```
plt.figure(figsize=(20, 10)) # Make the figure size look presentable
plt.plot(voxel_signal) # Plot the whole voxel signal
plt.plot(predictor_all + voxel_signal.mean() , lw=3) # Plot the predicted signal
plt.xlabel('Time (in volumes)', fontsize=20) # Provide an understandable x label
plt.ylabel('Activity (A.U.)', fontsize=20) # Provide an understandable y label
plt.legend(['Voxel timeseries', 'Predictor'], fontsize=15, loc='upper right', frameon=False) # Display legend t
plt.title("Signal and the associated stimulus predictor", fontsize=25) # Provide an understandable title
plt.grid() # Display grid lines
plt.show() # Show the plot in the output cell
```



We can see that our model kind of get's the trend of the signal during the eyes closed events, but it's definitally a rough estimate.

Test the Predictor Model (eyes closed) with R^2

In [11]:
```
# Fit a regresison model to the predictor and target signal.

predictor_all_ds = predictor_all[:, np.newaxis] # Copy predictor and add an extra axis to it for R^2 calculatio
icept = np.ones((predictor_all_ds.size, 1)) # Generate template for intercepts
X_simple = np.hstack((icept, predictor_all_ds)) # Generate Simple Model with hstack b_0 and b_1
betas_simple = np.linalg.inv(X_simple.T @ X_simple) @ X_simple.T @ voxel_signal # Generate Optimal beta values
y_hat_simple = X_simple[:, 0] * betas_simple[0] + X_simple[:, 1] * betas_simple[1] # Generate the estimated y v
print(betas_simple) # Print the Optimal beta values
numerator = np.sum((voxel_signal - y_hat_simple) ** 2) # Get the MSE of y_hat
denominator = np.sum((voxel_signal - np.mean(voxel_signal)) ** 2) # Get the MSE of the voxel_signal mean
r_squared = 1 - numerator / denominator # Get R^2 value from y_hat
print('The R² value is: %.3f' % r_squared) # Print R^2 value
```

```
[1.81962600e+03 1.50557963e+00]
The R² value is: 0.259
```

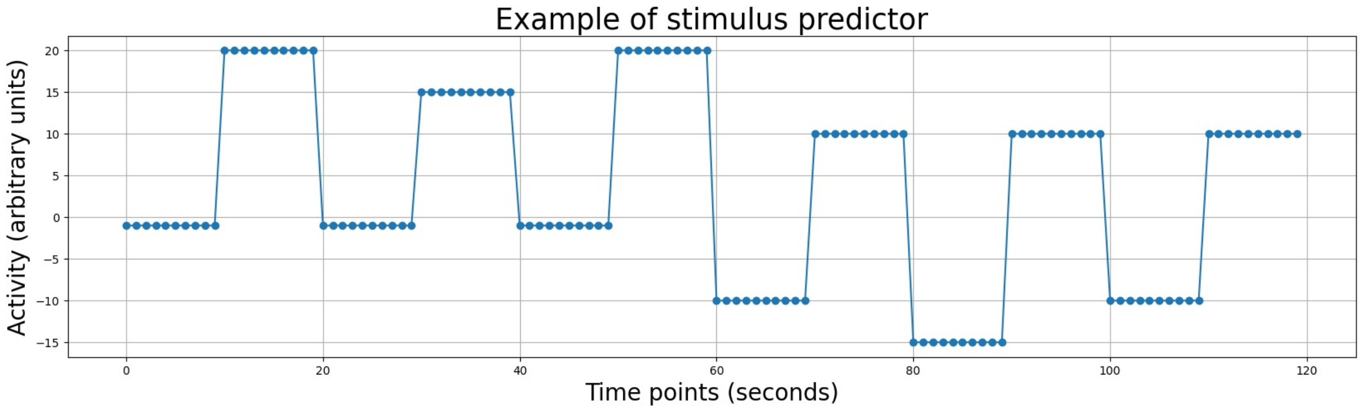
Here we get an R^2 value of 0.259, this means 25.9% of the variance is explained through our model which is honestly **relativly good** baised on the fact that our data is not pre-processed at all. This eyes closed only model is better than the eyes open model, by 18.9% points, which is way better. However due to us only analysing one signal it difference could be lengthend in other voxel signals.

See Preditor Model (eyes open - eyes closed)

In [12]:
```
# Create the predictor varaible.

combined_average = (design_EO_downsampled + design_EC_downsampled) # Add the downsampled vectors together (beca
predictor_all = combined_average # Assign the combined downsampled vector to the predictor_all variable
print("Shape of predictor: %s" % (predictor_all.shape,)) # Print out the shape of the predictor_all to make sur
plt.figure(figsize=(20, 5)) # Make the figure size look presentable
plt.plot(predictor_all, marker='o') # Plot the predictor's numerical values with an 'o' along with the whole pr
plt.xlabel('Time points (seconds)', fontsize=20) # Provide an understandable x label
plt.ylabel('Activity (arbitrary units)', fontsize=20) # Provide an understandable y label
plt.title('Example of stimulus predictor', fontsize=25) # Provide an understandable title
plt.grid() # Display grid lines
plt.show() # Show the plot in the output cell
```

```
Shape of predictor: (120,)
```

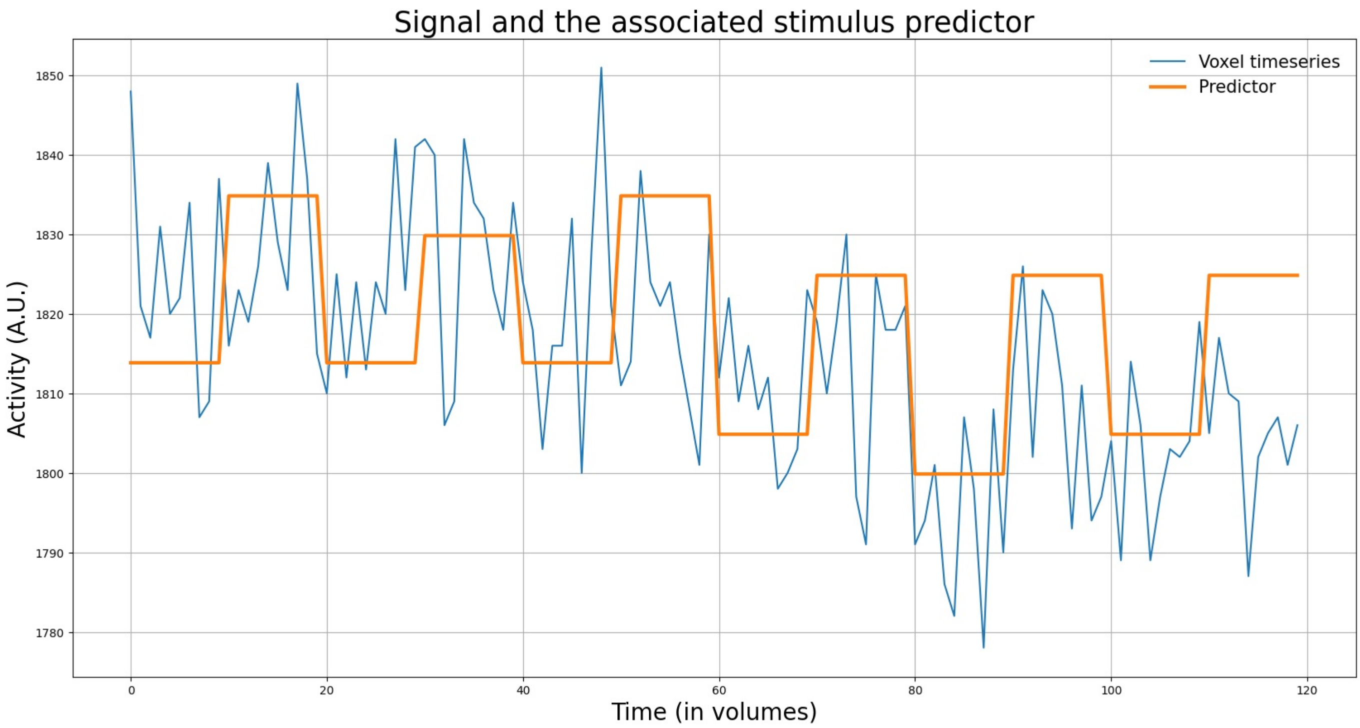


This shows us a graphical look at the GLM model (eyes open - eyes closed) with our given input values at the beginning.

See Preditor Model (eyes open - eyes closed) with Voxel Signal

In [13]:

```
plt.figure(figsize=(20, 10)) # Make the figure size look presentable
plt.plot(voxel_signal) # Plot the whole voxel signal
plt.plot(predictor_all + voxel_signal.mean() , lw=3) # Plot the predicted signal
plt.xlabel('Time (in volumes)', fontsize=20) # Provide an understandable x label
plt.ylabel('Activity (A.U.)', fontsize=20) # Provide an understandable y label
plt.legend(['Voxel timeseries', 'Predictor'], fontsize=15, loc='upper right', frameon=False) # Display legend to
plt.title("Signal and the associated stimulus predictor", fontsize=25) # Provide an understandable title
plt.grid() # Display grid lines
plt.show() # Show the plot in the output cell
```



We can see that our model kind of get's the trend of the general signal, but it's definitally a rough estimate.

Test the Predictor Model (eyes open - eyes closed) with R^2

In [14]:

```
# Fit a regresison model to the predictor and target signal.

predictor_all_ds = predictor_all[:, np.newaxis] # Copy predictor and add an extra axis to it for R^2 calculatio
icept = np.ones((predictor_all_ds.size, 1)) # Generate template for intercepts
X_simple = np.hstack((icept, predictor_all_ds)) # Generate Simple Model with hstack b_0 and b_1
betas_simple = np.linalg.inv(X_simple.T @ X_simple) @ X_simple.T @ voxel_signal # Generate Optimal beta values
y_hat_simple = X_simple[:, 0] * betas_simple[0] + X_simple[:, 1] * betas_simple[1] # Generate the estimated y va
```

```
print(betas_simple) # Print the Optimal beta values
numerator = np.sum((voxel_signal - y_hat_simple) ** 2) # Get the MSE of y_hat
denominator = np.sum((voxel_signal - np.mean(voxel_signal)) ** 2) # Get the MSE of the voxel_signal mean
r_squared = 1 - numerator / denominator # Get R^2 value from y_hat
print('The R² value is: %.3f' % r_squared) # Print R^2 value
```

```
[1.81278317e+03 5.29829500e-01]
The R² value is: 0.164
```

Here we get an R^2 value of 0.164, this means 16.4% of the variance is explained through our model which is honestly kinda **bad**. This (eyes open - eyes closed) model is better than both the eyes open model, but not the eyes closed model. Given that the data is not preprocessed at all and it's near 0.175 R^2 score I think this model is in the right direction, but clearly has it's issues which we must take into account.

Thoughts about all GLM (no HRF) Models

Overall I think that this GLMs made here were not the best, it's just mostly hard to tell because I have not messed around with other unpreprocessed data and other voxel signals to compare what makes a "good" R^2 score. General Rule of thumb is to be safe than sorry, so I will say that the models are not the best here.