# Main

December 4, 2020

## 1 Signal Filtering With Neural Networks

Problem statement: Assume that you have a known bandpass filter, now using this known filter generate 200 input-output pairs. Now, assume that somehow you forget all things about this filter but you still have 200 input-output pairs.

Now, built a DNN to approximate the lost band-pass filter. Please assume it is a causal symmetric filter.

Questions to answer

**Data preparation:** * How do I produce meaningful data? * What should the data look like? * What filter to use?

**Neural Network Creation:** * Which neural **network architecture** would be suitable? * What should be the hidden neurons **activation function**? * How many **hidden neurons and layers**? * What should be the **error criteria**? * How should I judge the **Accuracy** of my result?

Remove this later

Things to update: – Add a plt.savefig() to every line, make a folder to store and fetch the results from. – Make a net accuracy list containing the test and train accuracies of each model. – Put in references for all the things related to Libraries, Basic deeplearning stuff, Dropout, regularization – Put comments in the FFNN model 1

## 2 Index

12. Annexure 2: Regularization

# 3 Importing Libraries

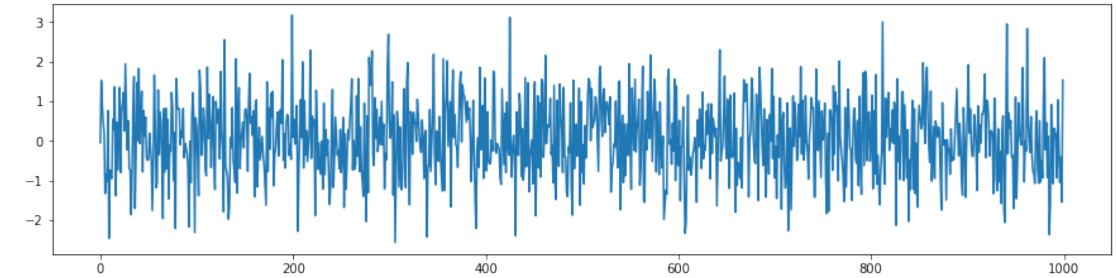
[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import json


%matplotlib inline
```

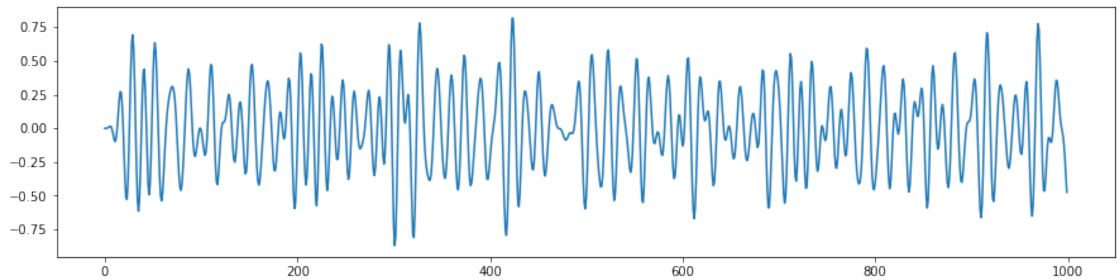
[2]:
```python
import tqdm

import torch
import torch.nn as nn
import torch.fft
print(torch.__version__)
```

```
1.7.0
```

# 4 Data Loading

[3]:
```python
data_file = open('./data.json')
data_dict = json.load(data_file)

plt.figure(figsize=(15,8))

plt.subplot(211)
plt.plot(data_dict[0]['x'])

plt.subplot(212)
plt.plot(data_dict[0]['y'])
plt.show()
```





[4]:
```
train_dict = data_dict[:int(len(data_dict)*2/3)]
test_dict = data_dict[-int(len(data_dict)*1/3):]
print(f'Training data:{len(train_dict)}\nTesting data:{len(test_dict)}')
```

```
Training data:200
Testing data:100
```

[5]:
```
#np.corrcoef(train_set[0]['x'],test_set[-1]['x'])
```

## 5 Preprocessing

[6]:
```
x_train = [train_dict[i]['x'] for i in range(len(train_dict))]
x_train = torch.Tensor(x_train)
x_train.shape
```

[6]:
```
torch.Size([200, 1000])
```

[7]:
```
y_train = [train_dict[i]['y'] for i in range(len(train_dict))]
y_train = torch.Tensor(y_train)
y_train.shape
```

[7]:
```
torch.Size([200, 1000])
```

[8]:
```
train = torch.utils.data.TensorDataset(x_train,y_train)
```

[9]:
```
x_test = [test_dict[i]['x'] for i in range(len(test_dict))]
x_test = torch.Tensor(x_test)
```

```
x_test.shape
```

[9]: torch.Size([100, 1000])

[10]:
```
y_test = [test_dict[i]['y'] for i in range(len(test_dict))]
y_test = torch.Tensor(y_test)
y_test.shape
```

[10]: torch.Size([100, 1000])

[11]:
```
test = torch.utils.data.TensorDataset(x_test,y_test)
```

[12]:
```
train_loader = torch.utils.data.DataLoader(train, batch_size = 1)
test_loader = torch.utils.data.DataLoader(test, batch_size = 1)
```

[13]:
```
x,y = next(iter(train_loader))
print(f'x: {x.shape}\ny: {y.shape}')
```

```
x: torch.Size([1, 1000])
y: torch.Size([1, 1000])
```

### 5.1 Evaluation utilities

[14]:
```
def evaluate(model, path):
    res = []

    model.eval()

    running_avg=0
    running_avg_fft=0
    i=0
    for x,y in train_loader:

        y_hat = model.forward(x)
        Y_hat = abs(np.fft.fft(y_hat.detach().numpy()))
        Y = abs(np.fft.fft(y.numpy()))


        corr = np.corrcoef(y.numpy(),y_hat.detach().numpy())[0,1]
        corr_fft = np.corrcoef(Y,Y_hat)[0,1]

        running_avg += (corr - running_avg)/(i+1)
        running_avg_fft += (corr_fft - running_avg_fft)/(i+1)
        i+=1

    print(f"Train Accuracy: {running_avg: .3f} || Train Accuracy (FFT):␣
↪{running_avg_fft: .3f}")
```

```
    res.append([running_avg,running_avg_fft])

    running_avg=0
    running_avg_fft=0
    i=0
    for x,y in test_loader:

        y_hat = model.forward(x)
        Y_hat = abs(np.fft.fft(y_hat.detach().numpy()))
        Y = abs(np.fft.fft(y.numpy()))


        corr = np.corrcoef(y.numpy(),y_hat.detach().numpy())[0,1]
        corr_fft = np.corrcoef(Y,Y_hat)[0,1]

        running_avg += (corr - running_avg)/(i+1)
        running_avg_fft += (corr_fft - running_avg_fft)/(i+1)
        i+=1


    print(f"Test Accuracy: {running_avg: .3f} || Test Accuracy (FFT):␣
↪{running_avg_fft: .3f}")

    res.append([running_avg,running_avg_fft])

    y_hat_test = model.forward(x_test[0])

    plt.figure(figsize=(15,8))

    plt.subplot(311)
    plt.plot(y_hat_test.detach().numpy() / y_hat_test.detach().numpy().max() ,␣
↪label='y_hat')
    plt.plot(y_test[0] / y_test[0].max(), label='y')
    plt.legend()

    plt.subplot(312)
    plt.plot(abs(np.fft.fft(y_hat_test.detach().numpy())))

    plt.subplot(313)
    plt.plot(abs(np.fft.fft(y_test[0])))

    plt.savefig('./eval/'+path)
    plt.show()

    return res
```

[15]:
```
Model_results = []
```

# 6 Simple Linear Regression

## 6.1 LR Model 1

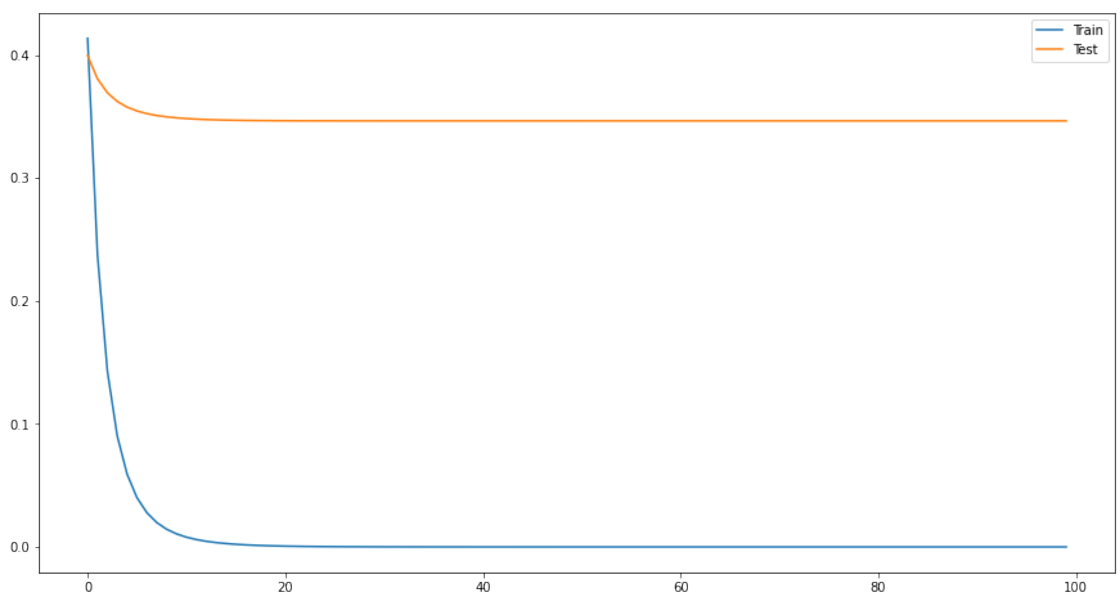
[16]:
```
class LR1(nn.Module):
    def __init__(self):

        super(LR1,self).__init__()
        self.fc_o = nn.Linear(in_features=1000,out_features=1000)

    def forward(self,x):

        y = self.fc_o(x)

        return y
```

[17]:
```
model = LR1()
criteria = nn.MSELoss()
optim = torch.optim.SGD(model.parameters(), lr=0.1, momentum=0.1)
```

[18]:
```
model.train()
epoch_nums = 100

running_loss_train = []
running_loss_test = []

epoch_t = tqdm.trange(epoch_nums,desc = 'Running Loss',leave = True)

for epoch in epoch_t:

    running_loss = 0
    i=0

    model.train()
    for x,y in train_loader:

        optim.zero_grad()

        y_hat = model.forward(x)

        loss = criteria(y_hat,y)
        loss.backward()

        optim.step()
```

```
        running_loss += (loss.item()-running_loss)/(i+1)
        i+=1

    epoch_t.set_description(f'Running Loss: {running_loss: .5f}')
    epoch_t.refresh()
    running_loss_train.append(running_loss)

    running_loss = 0
    i=0
    model.eval()
    for x,y in test_loader:

        y_hat = model.forward(x)

        loss = criteria(y_hat, y)

        running_loss += (loss.item()-running_loss)/(i+1)
        i+=1

    running_loss_test.append(running_loss)

plt.figure(figsize=(15,8))
plt.plot(running_loss_train, label='Train')
plt.plot(running_loss_test, label='Test')
plt.savefig('./loss/lr_1.png')
plt.legend()
plt.show()
```

```
Running Loss:  0.00000: 100%|      | 100/100 [02:58<00:00,  1.78s/it]
```



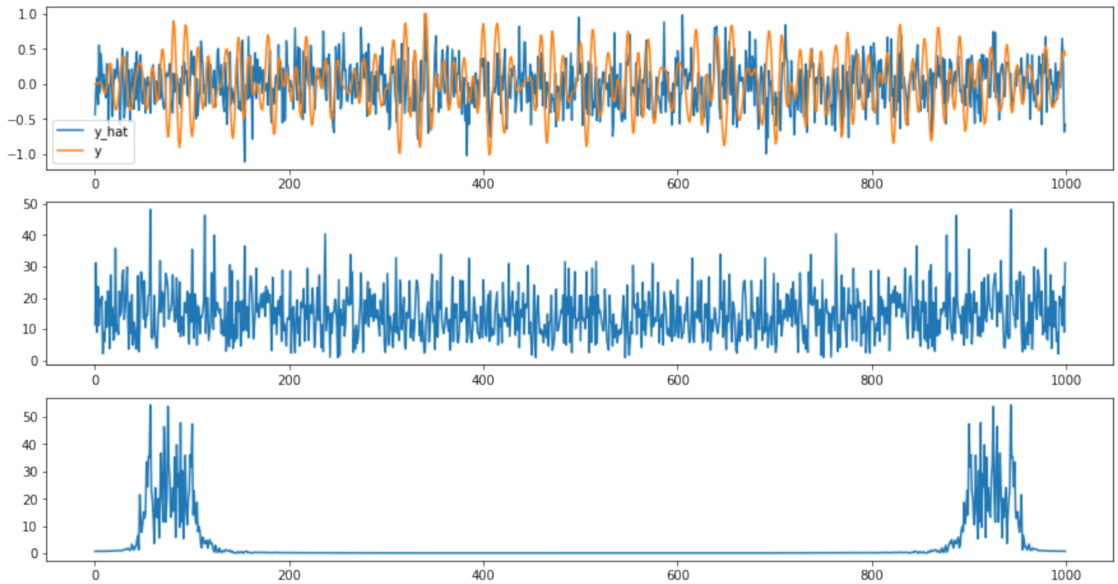
[19]:
```
#torch.save(model.state_dict(),'./models/lr_1.pth')
```

[21]:
```
#model.load_state_dict(torch.load('./models/lr_1.pth'))
```

[21]: <All keys matched successfully>

[22]:
```
Model_results.append(evaluate(model,'lr_1.png'))
```

```
Train Accuracy:  1.000 || Train Accuracy (FFT):  1.000
Test Accuracy:  0.116 || Test Accuracy (FFT):  0.157
```




## 6.2 LR Model 2

[23]:
```
class LR2(nn.Module):
    def __init__(self):

        super(LR2,self).__init__()
        self.fc_o = nn.Linear(in_features=1000,out_features=1000)

    def forward(self,x):

        y = self.fc_o(x)

        return y
```

[24]:
```
model = LR2()
criteria = nn.MSELoss()
optim = torch.optim.SGD(model.parameters(), lr=0.1, weight_decay=0.005)
```

[25]:
```
model.train()
epoch_nums = 100

running_loss_train = []
running_loss_test = []

epoch_t = tqdm.trange(epoch_nums,desc = 'Running Loss',leave = True)

for epoch in epoch_t:

    running_loss = 0
    i=0

    model.train()
    for x,y in train_loader:

        optim.zero_grad()

        y_hat = model.forward(x)

        loss = criteria(y_hat,y)
        loss.backward()

        optim.step()


        running_loss += (loss.item()-running_loss)/(i+1)
        i+=1

    epoch_t.set_description(f'Running Loss: {running_loss: .5f}')
    epoch_t.refresh()
    running_loss_train.append(running_loss)

    running_loss = 0
    i=0
    model.eval()
    for x,y in test_loader:

        y_hat = model.forward(x)

        loss = criteria(y_hat, y)

        running_loss += (loss.item()-running_loss)/(i+1)
```

```
        i+=1

    running_loss_test.append(running_loss)

plt.figure(figsize=(15,8))
plt.plot(running_loss_train, label='Train')
plt.plot(running_loss_test, label='Test')
plt.savefig('./loss/lr_2.png')
plt.legend()
plt.show()
```

```
Running Loss:  0.02585:   7%|          | 7/100 [00:12<02:52,  1.85s/it]
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-25-533320bf56a5> in <module>
     20
     21         loss = criteria(y_hat,y)
---> 22         loss.backward()
     23
     24         optim.step()

~/.local/lib/python3.8/site-packages/torch/tensor.py in backward(self, gradient ␣
 ↪retain_graph, create_graph)
    219                 retain_graph=retain_graph,
    220                 create_graph=create_graph)
--> 221         torch.autograd.backward(self, gradient, retain_graph,␣
 ↪create_graph)
    222
    223     def register_hook(self, hook):

~/.local/lib/python3.8/site-packages/torch/autograd/__init__.py in␣
 ↪backward(tensors, grad_tensors, retain_graph, create_graph, grad_variables)
    128         retain_graph = create_graph
    129
--> 130     Variable._execution_engine.run_backward(
    131         tensors, grad_tensors_, retain_graph, create_graph,
    132         allow_unreachable=True)  # allow_unreachable flag

KeyboardInterrupt:
```

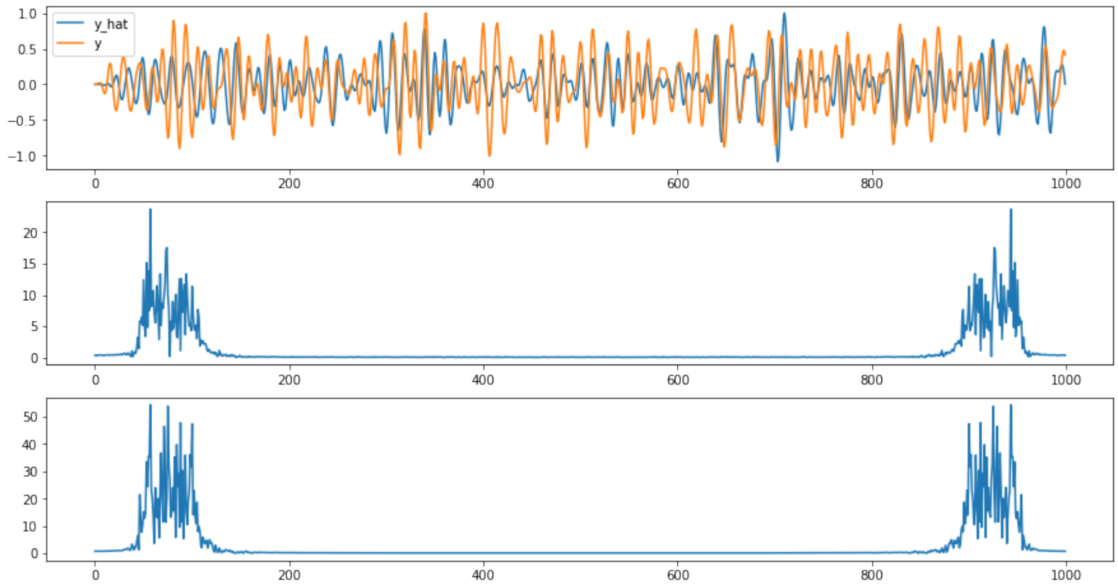
[81]:
```
#torch.save(model.state_dict(),'./models/lr_2.pth')
```

[26]:
```
model.load_state_dict(torch.load('./models/lr_2.pth'))
```

[26]: <All keys matched successfully>

[27]:
```
Model_results.append(evaluate(model,'lr_2.png'))
```

```
Train Accuracy:  0.996 || Train Accuracy (FFT):  0.998
Test Accuracy:  0.440 || Test Accuracy (FFT):  0.802
```




# 7 FFNN

## 7.1 Model 1

Simplest possible model. Let's see how this works against this data. Technically should not be be able to learn complex funcions.

[16]:
```
class ffnn_1(nn.Module):

    def __init__ (self):
        super(ffnn_1, self).__init__()

        self.fc1 = nn.Linear(1000, 1000)
        self.relu1 = nn.ReLU()
        self.fc_o = nn.Linear(1000, 1000)

    def forward(self, x):

        x = self.fc1(x)
        x = self.relu1(x)
        y = self.fc_o(x)
        return y
```

[17]:

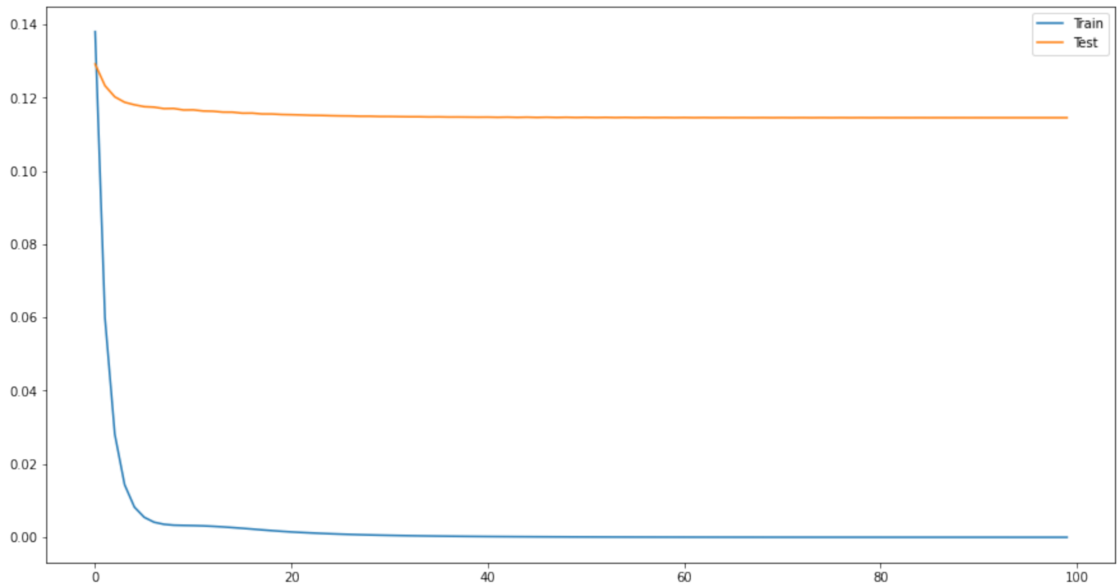
```
model = ffnn_1()
criteria = nn.MSELoss()
optim = torch.optim.SGD(model.parameters(), lr=1, momentum=0.1)
```

[18]:

```
model.train()
epoch_nums = 100

running_loss_train = []
running_loss_test = []

epoch_t = tqdm.trange(epoch_nums,desc = 'Running Loss',leave = True)

for epoch in epoch_t:

    running_loss = 0
    i=0

    model.train()
    for x,y in train_loader:

        optim.zero_grad()

        y_hat = model.forward(x)

        loss = criteria(y_hat,y)
        loss.backward()

        optim.step()


        running_loss += (loss.item()-running_loss)/(i+1)
        i+=1

    epoch_t.set_description(f'Running Loss: {running_loss: .5f}')
    epoch_t.refresh()
    running_loss_train.append(running_loss)

    running_loss = 0
    i=0
    model.eval()
    for x,y in test_loader:

        y_hat = model.forward(x)

        loss = criteria(y_hat, y)

        running_loss += (loss.item()-running_loss)/(i+1)
```

```
        i+=1

    running_loss_test.append(running_loss)

plt.figure(figsize=(15,8))
plt.plot(running_loss_train, label='Train')
plt.plot(running_loss_test, label='Test')
plt.savefig('./loss/ffnn_1.png')
plt.legend()
plt.show()
```

```
Running Loss:  0.00000: 100%|      | 100/100 [04:51<00:00,  2.91s/it]
```



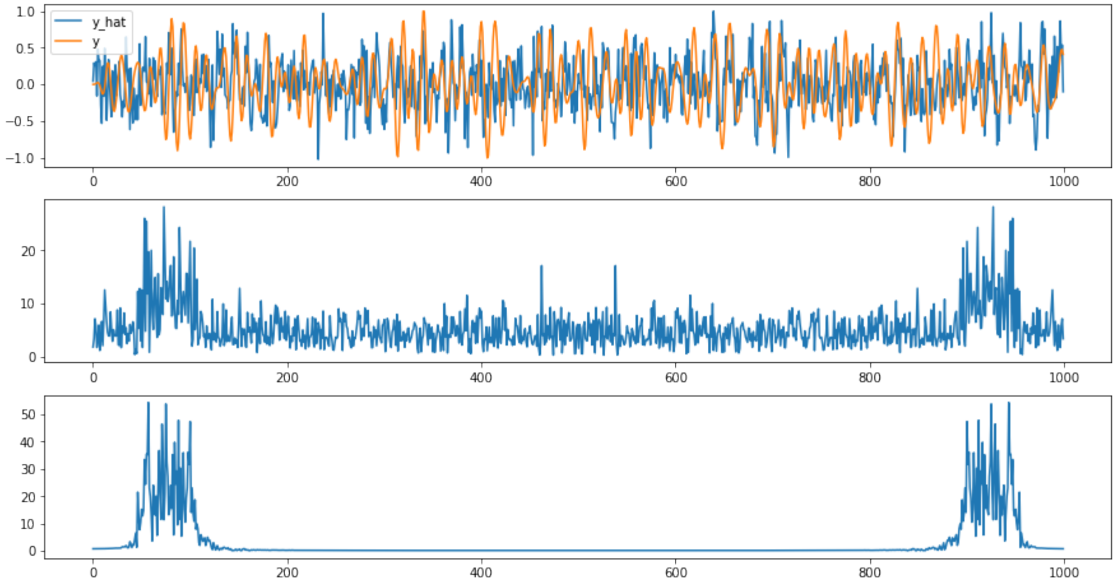

[20]:
```
#torch.save(model.state_dict(),'./models/ffnn_1.pth')
```

[32]:
```
#model.load_state_dict(torch.load('./models/ffnn_1.pth'))
```

[32]: <All keys matched successfully>

[21]:
```
Model_results.append(evaluate(model,'ffnn_1.png'))
```

```
Train Accuracy:  1.000 || Train Accuracy (FFT):  1.000
Test Accuracy:  0.211 || Test Accuracy (FFT):  0.543
```




## 7.2 Model 2

More power than the first one. Let's see how this does.

[34]:
```
class ffnn_2(nn.Module):

    def __init__ (self):
        super(ffnn_2, self).__init__()

        self.fc1 = nn.Linear(1000, 1000)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(1000, 1000)
        self.relu2 = nn.ReLU()
        self.fc_o = nn.Linear(1000, 1000)

    def forward(self, x):

        x = self.fc1(x)
        x = self.relu1(x)
        x = self.fc2(x)
        x = self.relu2(x)
        y = self.fc_o(x)
        return y
```

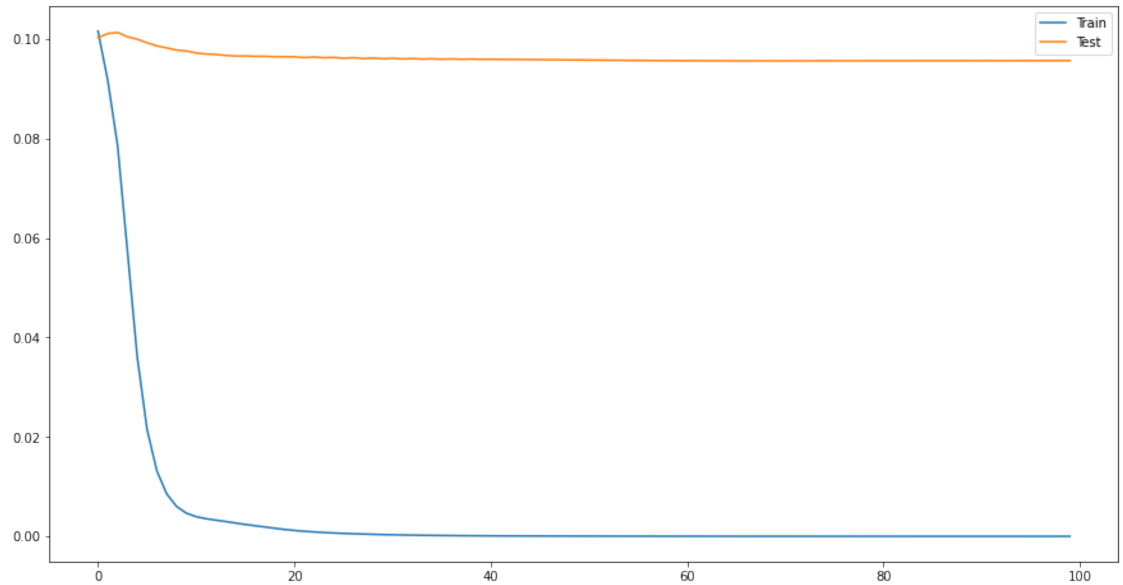
[35]:
```
model = ffnn_2()
criteria = nn.MSELoss()
optim = torch.optim.SGD(model.parameters(), lr=1, momentum=0.1)
```

[177]:

```python
# model.train()
# epoch_nums = 100

# running_loss_train = []
# running_loss_test = []

# epoch_t = tqdm.trange(epoch_nums,desc = 'Running Loss',leave = True)

# for epoch in epoch_t:

#     running_loss = 0
#     i=0

#     model.train()
#     for x,y in train_loader:

#         optim.zero_grad()

#         y_hat = model.forward(x)

#         loss = criteria(y_hat,y)
#         loss.backward()

#         optim.step()


#         running_loss += (loss.item()-running_loss)/(i+1)
#         i+=1

#     epoch_t.set_description(f'Running Loss: {running_loss: .5f}')
#     epoch_t.refresh()
#     running_loss_train.append(running_loss)

#     running_loss = 0
#     i=0
#     model.eval()
#     for x,y in test_loader:

#         y_hat = model.forward(x)

#         loss = criteria(y_hat, y)

#         running_loss += (loss.item()-running_loss)/(i+1)
#         i+=1

#     running_loss_test.append(running_loss)
```

```
# plt.figure(figsize=(15,8))
# plt.plot(running_loss_train, label='Train')
# plt.plot(running_loss_test, label='Test')
# plt.savefig('./loss/ffnn_2.png')
# plt.legend()
# plt.show()
```

```
Running Loss:  0.00000: 100%|          | 100/100 [07:01<00:00,  4.21s/it]
```



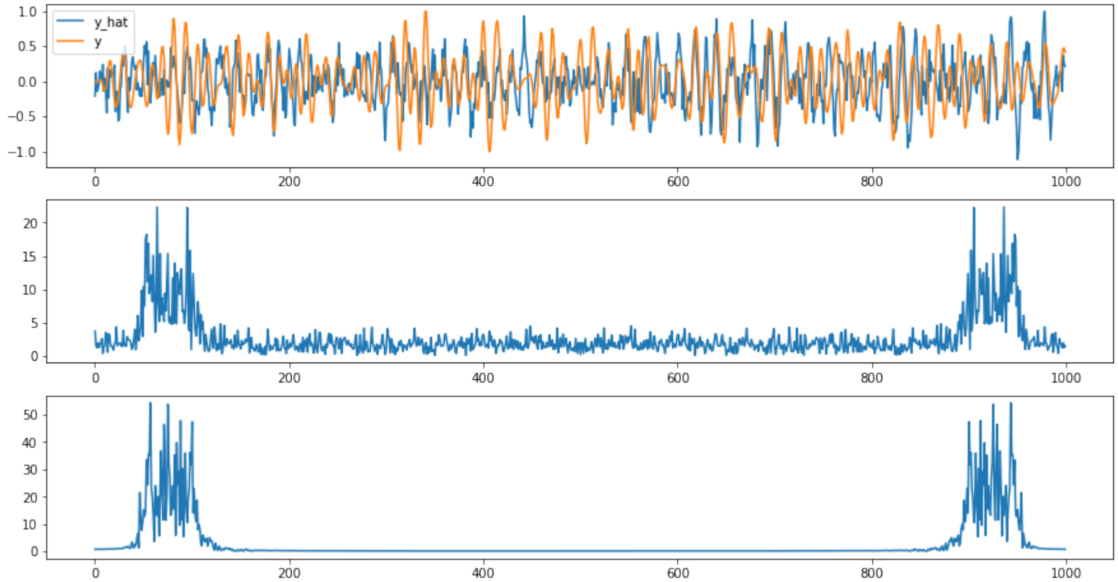

[178]:
```
#torch.save(model.state_dict(),'./models/ffnn_2.pth')
```

[37]:
```
model.load_state_dict(torch.load('./models/ffnn_2.pth'))
```

[37]:
```
<All keys matched successfully>
```

[38]:
```
Model_results.append(evaluate(model,'ffnn_2.png'))
```

```
Train Accuracy:  0.999 || Train Accuracy (FFT):  1.000
Test Accuracy:  0.251 || Test Accuracy (FFT):  0.697
```




### 7.3 Model 3

Using the first model, but this time with dropout and reguraliation

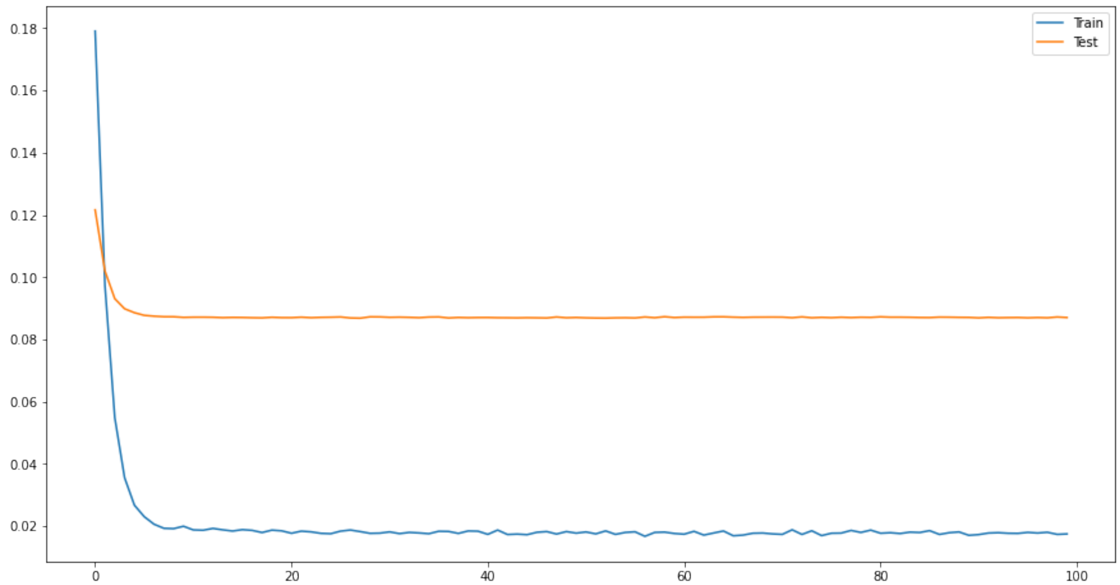
[39]:
```
class ffnn_3(nn.Module):

    def __init__ (self):
        super(ffnn_3, self).__init__()

        self.fc1 = nn.Linear(1000, 2000)
        self.relu1 = nn.ReLU()
        self.dropout = nn.Dropout(p=0.5)
        self.fc_o = nn.Linear(2000, 1000)

    def forward(self, x):

        x = self.fc1(x)
        x = self.relu1(x)
        x = self.dropout(x)
        y = self.fc_o(x)
        return y
```

[40]:
```
model = ffnn_3()
criteria = nn.MSELoss()
optim = torch.optim.SGD(model.parameters(), lr=1, momentum=0.1, weight_decay=0.
 ↪001)
```

[188]:
```
# model.train()
# epoch_nums = 100

# running_loss_train = []
# running_loss_test = []

# epoch_t = tqdm.trange(epoch_nums,desc = 'Running Loss',leave = True)

# for epoch in epoch_t:

#     running_loss = 0
#     i=0

#     model.train()
#     for x,y in train_loader:

#         optim.zero_grad()

#         y_hat = model.forward(x)

#         loss = criteria(y_hat,y)
#         loss.backward()

#         optim.step()


#         running_loss += (loss.item()-running_loss)/(i+1)
#         i+=1

#     epoch_t.set_description(f'Running Loss: {running_loss: .5f}')
#     epoch_t.refresh()
#     running_loss_train.append(running_loss)

#     running_loss = 0
#     i=0
#     model.eval()
#     for x,y in test_loader:

#         y_hat = model.forward(x)

#         loss = criteria(y_hat, y)

#         running_loss += (loss.item()-running_loss)/(i+1)
#         i+=1

#     running_loss_test.append(running_loss)
```

```
# plt.figure(figsize=(15,8))
# plt.plot(running_loss_train, label='Train')
# plt.plot(running_loss_test, label='Test')
# plt.savefig('./loss/ffnn_3.png')
# plt.legend()
# plt.show()
```

```
Running Loss:  0.01754: 100%|      | 100/100 [10:56<00:00,  6.56s/it]
```



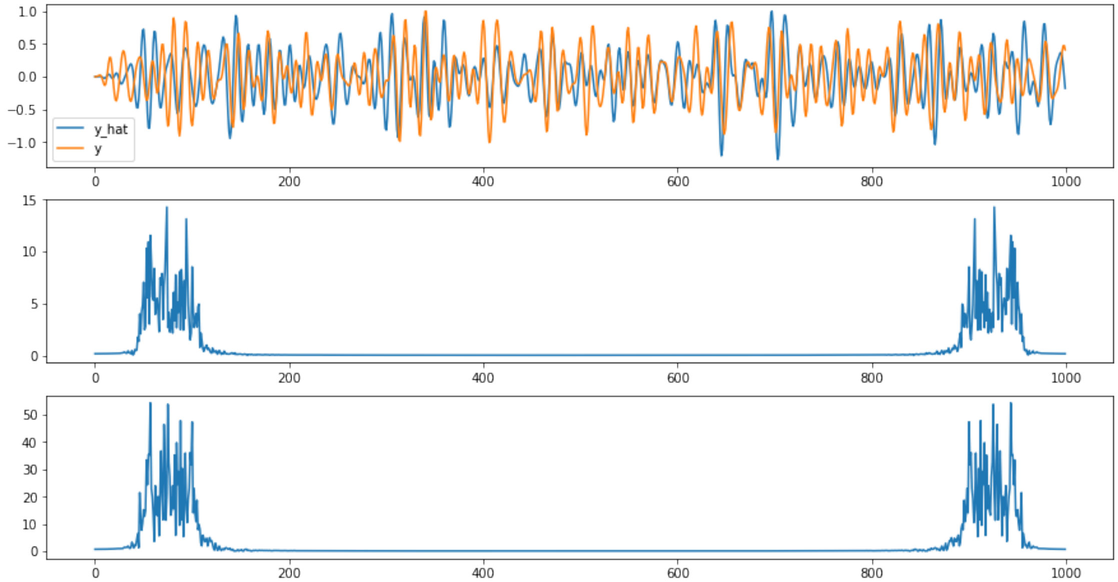

[190]:
```
#torch.save(model.state_dict(),'./models/ffnn_3.pth')
```

[41]:
```
model.load_state_dict(torch.load('./models/ffnn_3.pth'))
```

[41]:
```
<All keys matched successfully>
```

[42]:
```
Model_results.append(evaluate(model,'ffnn_3.png'))
```

```
Train Accuracy:  0.969 || Train Accuracy (FFT):  0.983
Test Accuracy:  0.391 || Test Accuracy (FFT):  0.796
```




## 7.4 Model 4

Using the Second model, but this time with dropout and reguraliation

[43]:
```
class ffnn_4(nn.Module):

    def __init__ (self):
        super(ffnn_4, self).__init__()

        self.fc1 = nn.Linear(1000, 2000)
        self.relu1 = nn.ReLU()
        self.dropout1 = nn.Dropout(p=0.4)

        self.fc2 = nn.Linear(2000, 2000)
        self.relu2 = nn.ReLU()
        self.dropout2 = nn.Dropout(p=0.4)

        self.fc_o = nn.Linear(2000, 1000)

    def forward(self, x):

        x = self.fc1(x)
        x = self.relu1(x)
        x = self.dropout1(x)

        x = self.fc2(x)
        x = self.relu2(x)
        x = self.dropout2(x)
```

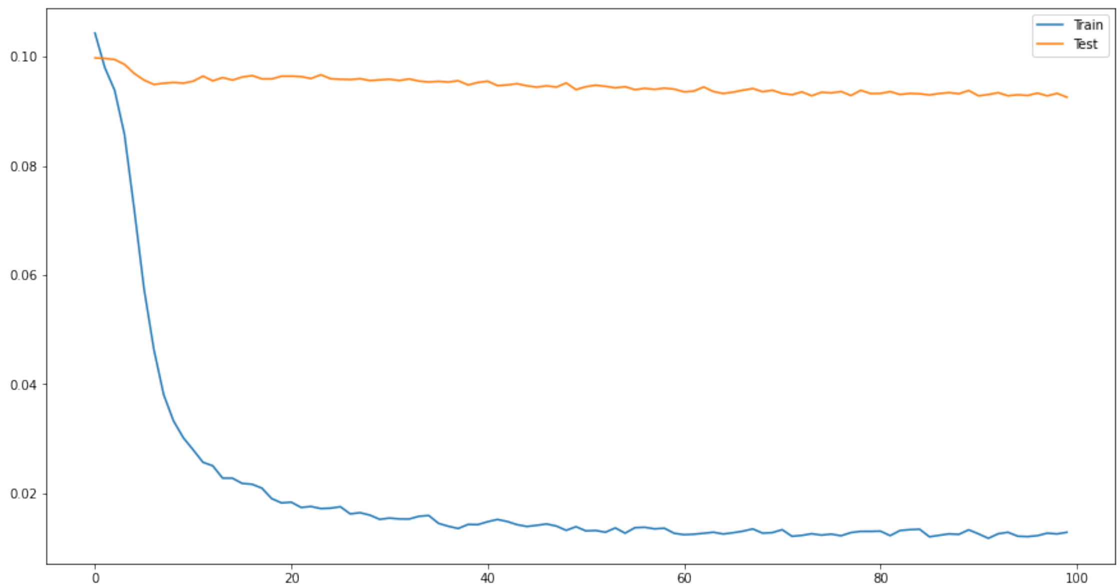
```
        y = self.fc_o(x)
        return y
```

[44]:
```
model = ffnn_4()
criteria = nn.MSELoss()
optim = torch.optim.SGD(model.parameters(), lr=1, momentum=0.1, weight_decay=0.
 ↪0001)
```

[63]:
```
# model.train()
# epoch_nums = 100

# running_loss_train = []
# running_loss_test = []

# epoch_t = tqdm.trange(epoch_nums,desc = 'Running Loss',leave = True)

# for epoch in epoch_t:

#     running_loss = 0
#     i=0

#     model.train()
#     for x,y in train_loader:

#         optim.zero_grad()

#         y_hat = model.forward(x)

#         loss = criteria(y_hat,y)
#         loss.backward()

#         optim.step()


#         running_loss += (loss.item()-running_loss)/(i+1)
#         i+=1

#     epoch_t.set_description(f'Running Loss: {running_loss: .5f}')
#     epoch_t.refresh()
#     running_loss_train.append(running_loss)

#     running_loss = 0
#     i=0
#     model.eval()
#     for x,y in test_loader:
```

```
#          y_hat = model.forward(x)

#          loss = criteria(y_hat, y)

#          running_loss += (loss.item()-running_loss)/(i+1)
#          i+=1

#      running_loss_test.append(running_loss)

# plt.figure(figsize=(15,8))
# plt.plot(running_loss_train, label='Train')
# plt.plot(running_loss_test, label='Test')
# plt.savefig('./loss/ffnn_4.png')
# plt.legend()
# plt.show()
```

```
Running Loss:  0.01284: 100%|      | 100/100 [21:11<00:00, 12.71s/it]
```



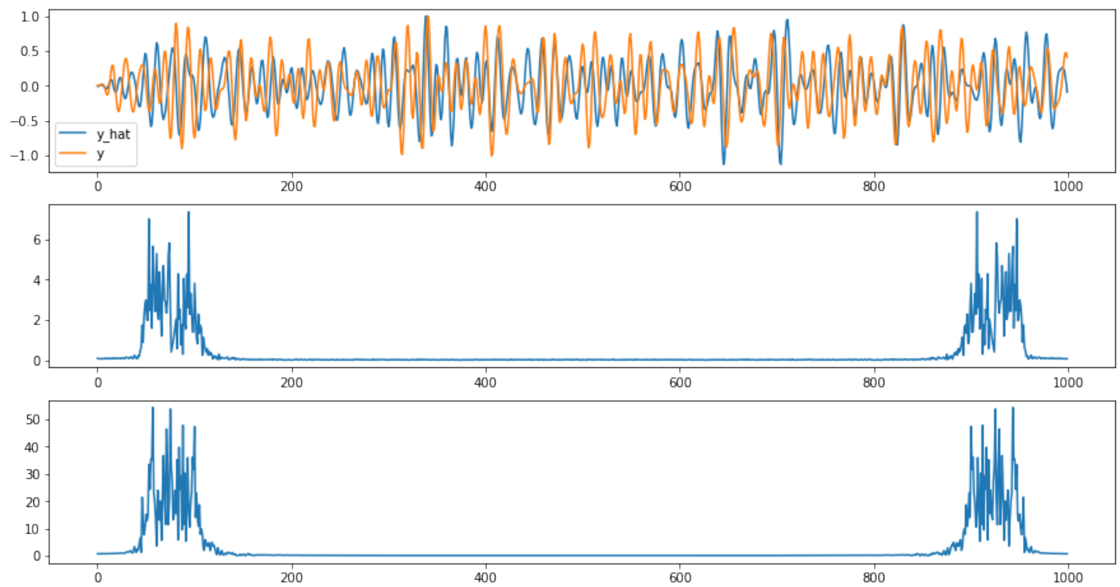

[67]:
```
#torch.save(model.state_dict(),'./models/ffnn_4.pth')
```

[45]:
```
model.load_state_dict(torch.load('./ffnn_4.pth'))
```

[45]: <All keys matched successfully>

[46]:
```
Model_results.append(evaluate(model,'ffnn_4.png'))
```

```
Train Accuracy:  0.986 || Train Accuracy (FFT):  0.992
Test Accuracy:  0.337 || Test Accuracy (FFT):  0.789
```




### 7.5 Model 5

Using the first model, but with larger hidden layer

[47]:
```
class ffnn_5(nn.Module):

    def __init__ (self):
        super(ffnn_5, self).__init__()

        self.fc1 = nn.Linear(1000, 500)
        self.relu1 = nn.ReLU()
        self.fc_o = nn.Linear(500, 1000)

    def forward(self, x):

        x = self.fc1(x)
        x = self.relu1(x)
        y = self.fc_o(x)
        return y
```

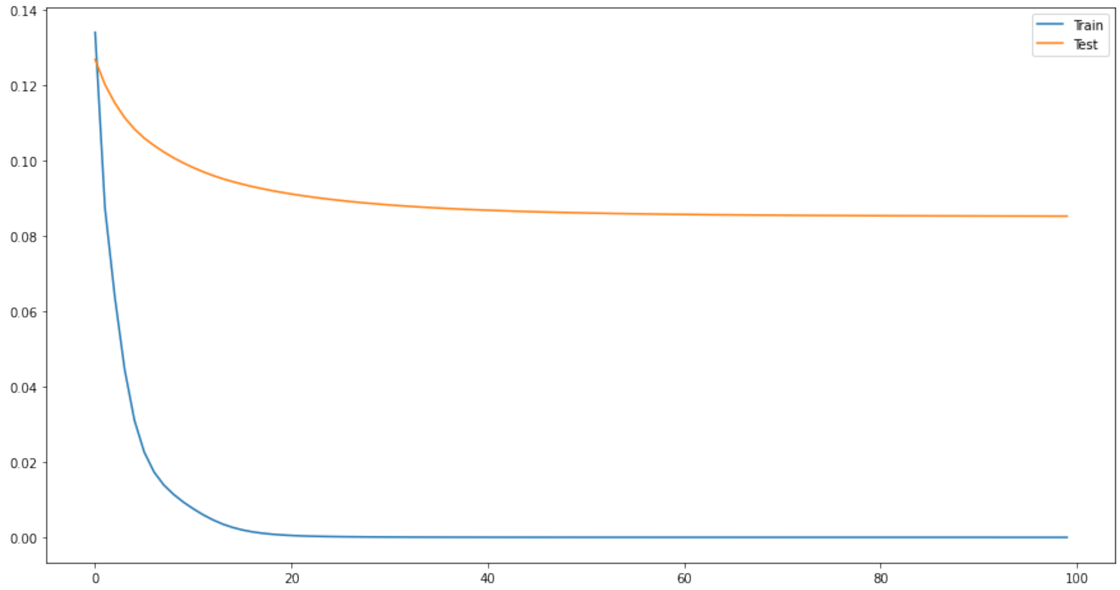
[48]:
```
model = ffnn_5()
criteria = nn.MSELoss()
optim = torch.optim.SGD(model.parameters(), lr=1, momentum=0.1, weight_decay=0.
 ↪0001)
```

[32]:
```
# model.train()
# epoch_nums = 100
```

```python
# running_loss_train = []
# running_loss_test = []

# epoch_t = tqdm.trange(epoch_nums,desc = 'Running Loss',leave = True)

# for epoch in epoch_t:

#     running_loss = 0
#     i=0

#     model.train()
#     for x,y in train_loader:

#         optim.zero_grad()

#         y_hat = model.forward(x)

#         loss = criteria(y_hat,y)
#         loss.backward()

#         optim.step()


#         running_loss += (loss.item()-running_loss)/(i+1)
#         i+=1

#     epoch_t.set_description(f'Running Loss: {running_loss: .5f}')
#     epoch_t.refresh()
#     running_loss_train.append(running_loss)

#     running_loss = 0
#     i=0
#     model.eval()
#     for x,y in test_loader:

#         y_hat = model.forward(x)

#         loss = criteria(y_hat, y)

#         running_loss += (loss.item()-running_loss)/(i+1)
#         i+=1

#     running_loss_test.append(running_loss)

# plt.figure(figsize=(15,8))
# plt.plot(running_loss_train, label='Train')
# plt.plot(running_loss_test, label='Test')
```

```
# plt.savefig('./loss/ffnn_5.png')
# plt.legend()
# plt.show()
```

```
Running Loss:  0.00011: 100%|      | 100/100 [02:54<00:00,  1.74s/it]
```



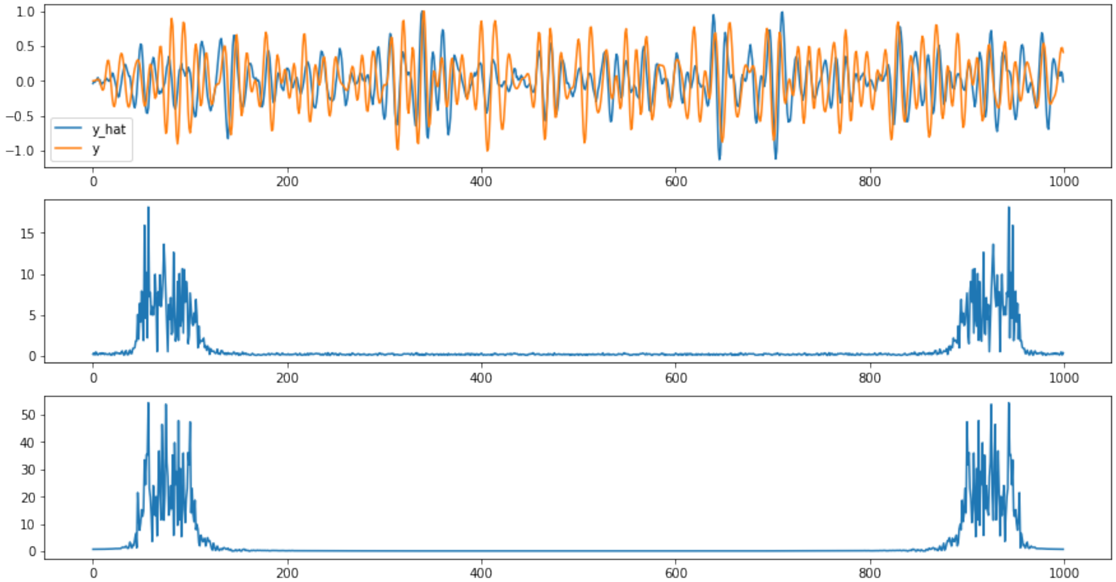

[37]:
```
#torch.save(model.state_dict(),'./models/ffnn_5.pth')
```

[49]:
```
model.load_state_dict(torch.load('./ffnn_5.pth'))
```

[49]:
```
<All keys matched successfully>
```

[50]:
```
Model_results.append(evaluate(model,'ffnn_5.png'))
```

```
Train Accuracy:  1.000 || Train Accuracy (FFT):  1.000
Test Accuracy:  0.402 || Test Accuracy (FFT):  0.794
```




## 7.6 Model 6

Simplest possible model. Let's see how this works against this data. Technically should not be be able to learn complex funcions.

[51]:
```
class ffnn_6(nn.Module):

    def __init__ (self):
        super(ffnn_6, self).__init__()

        self.fc1 = nn.Linear(1000, 1000)
        self.relu1 = nn.ReLU()
        self.fc_o = nn.Linear(1000, 1000)

    def forward(self, x):

        x = self.fc1(x)
        x = self.relu1(x)
        y = self.fc_o(x)
        return y
```

[52]:
```
model = ffnn_6()
criteria = nn.MSELoss()
optim = torch.optim.SGD(model.parameters(), lr=1, momentum=0.1, weight_decay=0.
↪0003)
```

[53]:
```
# model.train()
# epoch_nums = 100

# running_loss_train = []
# running_loss_test = []

# epoch_t = tqdm.trange(epoch_nums,desc = 'Running Loss',leave = True)

# for epoch in epoch_t:

#     running_loss = 0
#     i=0

#     model.train()
#     for x,y in train_loader:

#         optim.zero_grad()

#         y_hat = model.forward(x)

#         loss = criteria(y_hat,y)
#         loss.backward()

#         optim.step()


#         running_loss += (loss.item()-running_loss)/(i+1)
#         i+=1

#     epoch_t.set_description(f'Running Loss: {running_loss: .5f}')
#     epoch_t.refresh()
#     running_loss_train.append(running_loss)

#     running_loss = 0
#     i=0
#     model.eval()
#     for x,y in test_loader:

#         y_hat = model.forward(x)

#         loss = criteria(y_hat, y)

#         running_loss += (loss.item()-running_loss)/(i+1)
#         i+=1

#     running_loss_test.append(running_loss)
```

```
# plt.figure(figsize=(15,8))
# plt.plot(running_loss_train, label='Train')
# plt.plot(running_loss_test, label='Test')
# plt.savefig('./loss/ffnn_6.png')
# plt.legend()
# plt.show()
```

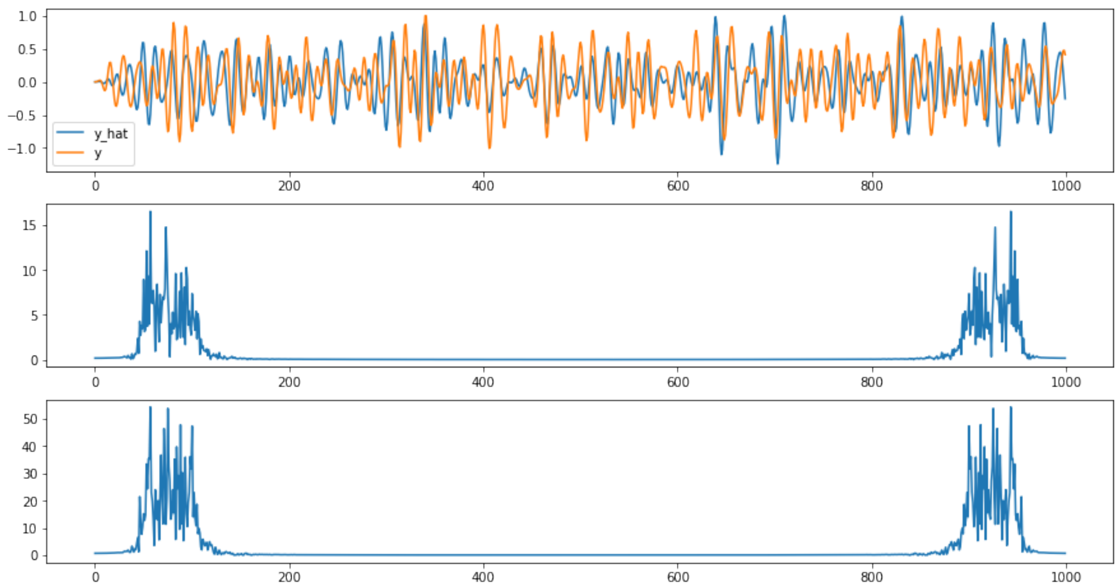
[46]:
```
#torch.save(model.state_dict(),'./models/ffnn_6.pth')
```

[54]:
```
model.load_state_dict(torch.load('./models/ffnn_6.pth'))
```

[54]: <All keys matched successfully>

[55]:
```
Model_results.append(evaluate(model,'ffnn_6.png'))
```

```
Train Accuracy:  0.998 || Train Accuracy (FFT):  0.999
Test Accuracy:  0.415 || Test Accuracy (FFT):  0.797
```




### 7.7 Model 7

[22]:
```
class ffnn_7(nn.Module):

    def __init__ (self):
        super(ffnn_7, self).__init__()

        self.fc1 = nn.Linear(1000, 1000)
        self.relu1 = nn.ReLU()
```

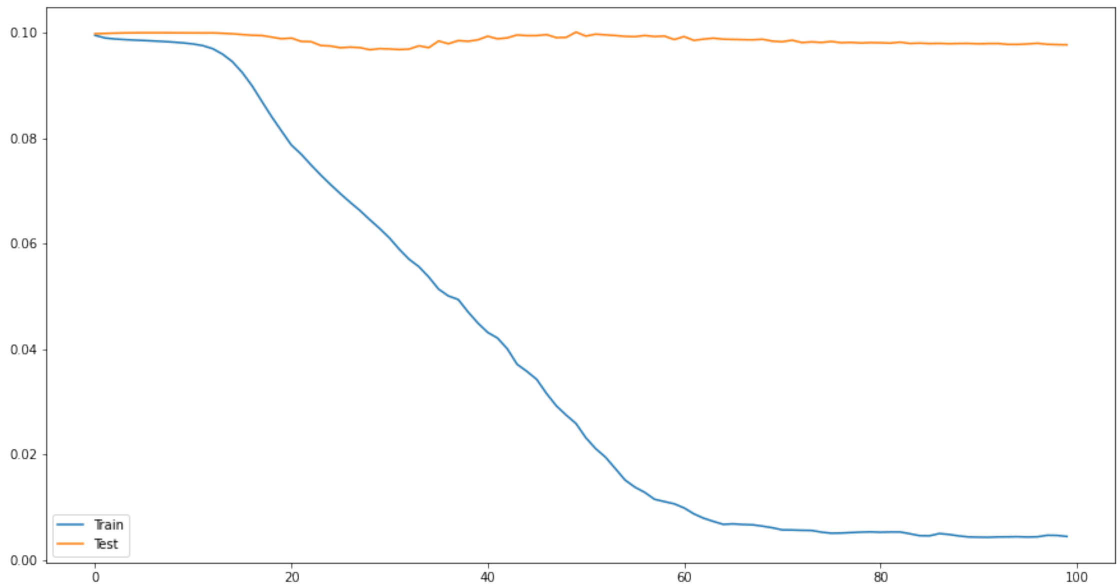
```
        self.dropout1 = nn.Dropout(p=0)

        self.fc2 = nn.Linear(1000, 1000)
        self.relu2 = nn.ReLU()
        self.dropout2 = nn.Dropout(p=0)

        self.fc3 = nn.Linear(1000, 1000)
        self.relu3 = nn.ReLU()
        self.dropout3 = nn.Dropout(p=0)

        self.fc4 = nn.Linear(1000, 1000)
        self.relu4 = nn.ReLU()
        self.dropout4 = nn.Dropout(p=0)

        self.fc5 = nn.Linear(1000, 1000)
        self.relu5 = nn.ReLU()
        self.dropout5 = nn.Dropout(p=0)

        self.fc_o = nn.Linear(1000, 1000)

    def forward(self, x):

        x = self.fc1(x)
        x = self.relu1(x)
        x = self.dropout1(x)

        x = self.fc2(x)
        x = self.relu2(x)
        x = self.dropout2(x)

        x = self.fc3(x)
        x = self.relu3(x)
        x = self.dropout3(x)

        x = self.fc4(x)
        x = self.relu4(x)
        x = self.dropout4(x)

        x = self.fc5(x)
        x = self.relu5(x)
        x = self.dropout5(x)

        y = self.fc_o(x)

        return y
```

[23]:
```
model = ffnn_7()
criteria = nn.MSELoss()
optim = torch.optim.SGD(model.parameters(), lr=1, momentum=0.1, weight_decay=0.
 ↪0001)
```

[24]:
```
model.train()
epoch_nums = 100

running_loss_train = []
running_loss_test = []

epoch_t = tqdm.trange(epoch_nums,desc = 'Running Loss',leave = True)

for epoch in epoch_t:

    running_loss = 0
    i=0

    model.train()
    for x,y in train_loader:

        optim.zero_grad()

        y_hat = model.forward(x)

        loss = criteria(y_hat,y)
        loss.backward()

        optim.step()


        running_loss += (loss.item()-running_loss)/(i+1)
        i+=1

    epoch_t.set_description(f'Running Loss: {running_loss: .5f}')
    epoch_t.refresh()
    running_loss_train.append(running_loss)

    running_loss = 0
    i=0
    model.eval()
    for x,y in test_loader:

        y_hat = model.forward(x)

        loss = criteria(y_hat, y)
```

```
        running_loss += (loss.item()-running_loss)/(i+1)
        i+=1

    running_loss_test.append(running_loss)

plt.figure(figsize=(15,8))
plt.plot(running_loss_train, label='Train')
plt.plot(running_loss_test, label='Test')
plt.savefig('./loss/ffnn_7.png')
plt.legend()
plt.show()
```

```
Running Loss:  0.00442: 100%|          | 100/100 [17:03<00:00, 10.23s/it]
```



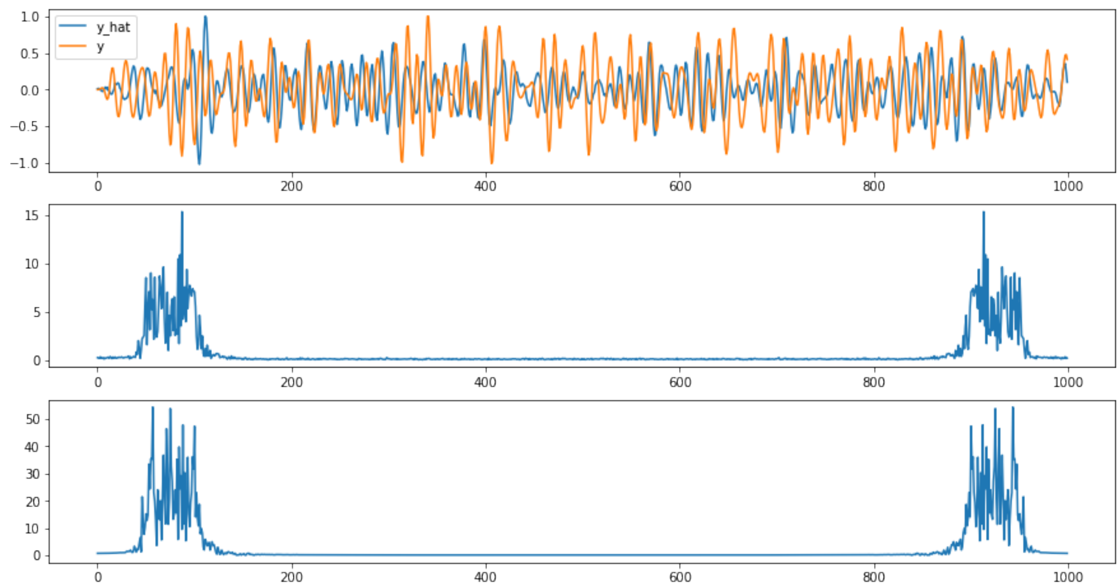

[25]:
```
#torch.save(model.state_dict(),'./models/ffnn_7.pth')
```

[26]:
```
model.load_state_dict(torch.load('./models/ffnn_7.pth'))
```

[26]: <All keys matched successfully>

[27]:
```
Model_results.append(evaluate(model,'ffnn_7.png'))
```

```
Train Accuracy:  0.976 || Train Accuracy (FFT):  0.987
Test Accuracy:  0.146 || Test Accuracy (FFT):  0.764
```




# 8 RNN

# 9 RNN model 1

[28]:
```
class rnn(nn.Module):


    def __init__(self, input_dim, hidden_dim, output_dim):
        super(rnn, self).__init__()

        self.hidden_dim = hidden_dim

        self.rnn = nn.RNN(input_size = input_dim, hidden_size = hidden_dim,␣
↪nonlinearity='relu')
        self.fc = nn.Linear(in_features=hidden_dim*1000, out_features=1000)

    def forward(self, x):

        h_0 = torch.zeros(1,1,self.hidden_dim)

        out, h_n = self.rnn(x, h_0)
        out = self.fc(out.view(-1))

        return out
```

[31]:
```
model = rnn(1,1,1)
optimizer = torch.optim.SGD(model.parameters(), lr=1, momentum=0.1)
criteria = nn.MSELoss()
```

[ ]:
```
model.train()
epoch_nums = 100

running_loss_train = []
running_loss_test = []

epoch_t = tqdm.trange(epoch_nums,desc = 'Running Loss',leave = True)

for epoch in epoch_t:

    running_loss = 0
    i=0

    model.train()
    for x,y in train_loader:
        x = x.view(-1,1,1)
        y = y.view(-1)

        optim.zero_grad()

        y_hat = model.forward(x)

        loss = criteria(y_hat,y)
        loss.backward()

        optim.step()


        running_loss += (loss.item()-running_loss)/(i+1)
        i+=1

    epoch_t.set_description(f'Running Loss: {running_loss: .5f}')
    epoch_t.refresh()
    running_loss_train.append(running_loss)

    running_loss = 0
    i=0
    model.eval()
    for x,y in test_loader:
        x = x.view(-1,1,1)
        y = y.view(-1)

        y_hat = model.forward(x)
```

```
        loss = criteria(y_hat, y)

        running_loss += (loss.item()-running_loss)/(i+1)
        i+=1

    running_loss_test.append(running_loss)

plt.figure(figsize=(15,8))
plt.plot(running_loss_train, label='Train')
plt.plot(running_loss_test, label='Test')
plt.savefig('./loss/rnn_1.png')
plt.legend()
plt.show()
```

```
Running Loss:   0%|          | 0/100 [00:00<?, ?it/s]
```

[46]:
```
#torch.save(model.state_dict(),'./models/rnn_1.pth')
```

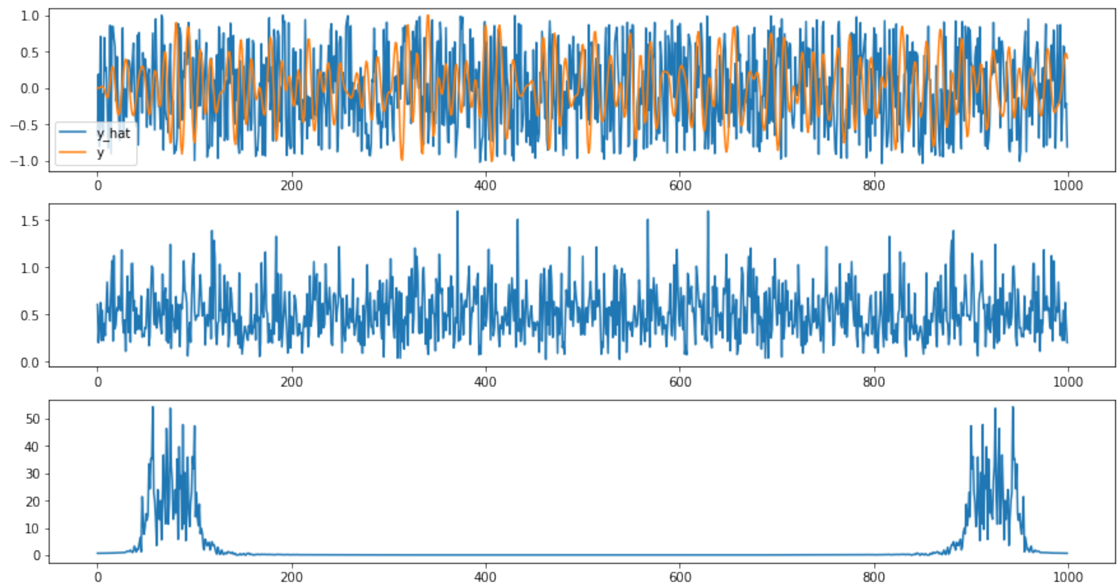
[156]:
```
#model.load_state_dict(torch.load('./models/rnn_1.pth'))
```

[88]:
```
res = []

model.eval()

running_avg=0
running_avg_fft=0
i=0
for x,y in train_loader:
    x = x.view(-1,1,1)
    y = y.view(-1)

    y_hat = model.forward(x)
    Y_hat = abs(np.fft.fft(y_hat.detach().numpy()))
    Y = abs(np.fft.fft(y.numpy()))


    corr = np.corrcoef(y.numpy(),y_hat.detach().numpy())[0,1]
    corr_fft = np.corrcoef(Y,Y_hat)[0,1]

    running_avg += (corr - running_avg)/(i+1)
    running_avg_fft += (corr_fft - running_avg_fft)/(i+1)
    i+=1

print(f"Train Accuracy: {running_avg: .3f} || Train Accuracy (FFT):␣
 ↪{running_avg_fft: .3f}")
```

```
res.append([running_avg,running_avg_fft])

running_avg=0
running_avg_fft=0
i=0
for x,y in test_loader:

    x = x.view(-1,1,1)
    y = y.view(-1)

    y_hat = model.forward(x)
    Y_hat = abs(np.fft.fft(y_hat.detach().numpy()))
    Y = abs(np.fft.fft(y.numpy()))


    corr = np.corrcoef(y.numpy(),y_hat.detach().numpy())[0,1]
    corr_fft = np.corrcoef(Y,Y_hat)[0,1]

    running_avg += (corr - running_avg)/(i+1)
    running_avg_fft += (corr_fft - running_avg_fft)/(i+1)
    i+=1


print(f"Test Accuracy: {running_avg: .3f} || Test Accuracy (FFT):␣
 ↪{running_avg_fft: .3f}")

res.append([running_avg,running_avg_fft])

y_hat_test = model.forward(x_test[0].view(-1,1,1))

plt.figure(figsize=(15,8))

plt.subplot(311)
plt.plot(y_hat_test.detach().numpy() / y_hat_test.detach().numpy().max() ,␣
 ↪label='y_hat')
plt.plot(y_test[0] / y_test[0].max(), label='y')
plt.legend()

plt.subplot(312)
plt.plot(abs(np.fft.fft(y_hat_test.detach().numpy())))

plt.subplot(313)
plt.plot(abs(np.fft.fft(y_test[0])))

plt.savefig('./eval/rnn_1.png')
plt.show()
```

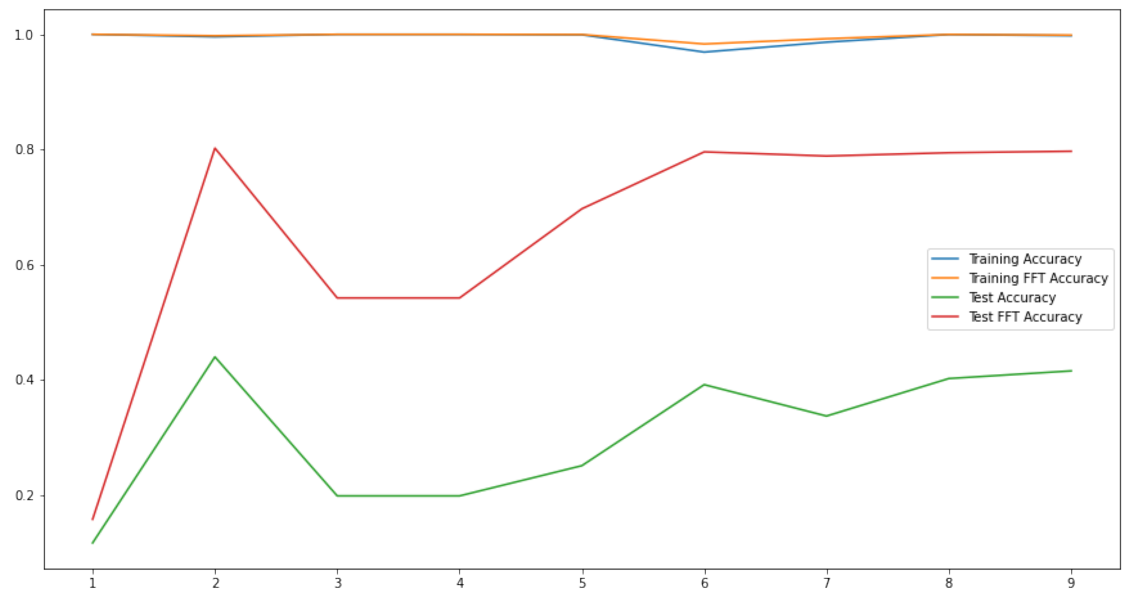
```
Model_results.append(res)
```

```
Train Accuracy: -0.001 || Train Accuracy (FFT):  0.019
Test Accuracy: -0.004 || Test Accuracy (FFT):  0.019
```




# 10 Conclusion

[60]:
```
train_acc = [Model_results[i][0][0] for i in range(len(Model_results))]
train_fft_acc = [Model_results[i][0][1] for i in range(len(Model_results))]
test_acc = [Model_results[i][1][0] for i in range(len(Model_results))]
test_fft_acc = [Model_results[i][1][1] for i in range(len(Model_results))]



plt.figure(figsize=(15,8))

plt.plot(train_acc, label='Training Accuracy')
plt.plot(train_fft_acc, label='Training FFT Accuracy')
plt.plot(test_acc, label='Test Accuracy')
plt.plot(test_fft_acc, label='Test FFT Accuracy')
plt.xticks(np.arange(0,len(Model_results)),np.arange(1,len(Model_results)+1))

plt.legend(loc='right')
plt.show()
```




## 11 References

Put stuff here...

## 12 Annex 1

Neural Networks for Noobs (Trying to give a intuitive idea)

Basically...

A Neural Network is just a bunch of numbers that iteratively come close to predicting what the output would look like through steps of gradient descent.

Some math...

So, it's pretty simple right?

Some terminologies...

FFNN : Called a Feed Forward Neural Network. Essentially a simple network of fully connected neurons stacked on top of each other to form a layer, with multiple layers to make the final network. Parameters: The weights and biases used in the neural network. Hyperparameters: Values that change how the neural network learns, like the learning rate, the architechture of the neural net etc. Activation Function: Essentially a non-linearity added for the sake of getting approximations that can be non-linear in nature. Can be of multiple types; I primarily used ReLU though.

Optimizing Algorithm: The algorithm that takes a tiny step in the direction of the gradient to iteratively arrive at the minimum. There are many types. I've used SGD with Momentum primarily.

Loss criteria:The function of the prediction and the acutal output that converts the difference between them into a convex function, for which we can find the point of minimum loss.

Black Box Intuition

Most neural networks work like black boxes of millions of numbers that give out the output. It's very hard to explain what a neuron in a Artificial Neural Network learns; infact it's an entire field in AI called Explainable AI.

## 13 Annex 2

Regularization

Why?

L2 regularization

Dropout

Basically...