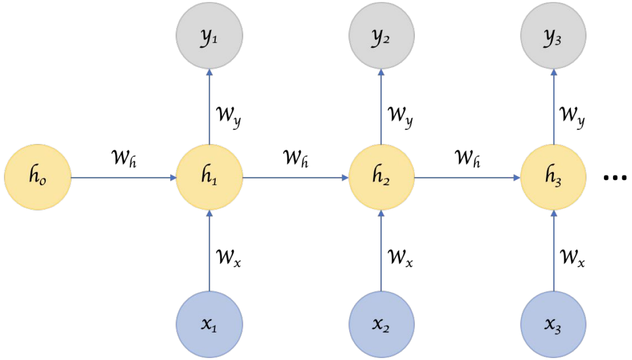
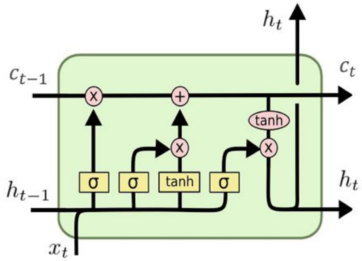
RECURRENT NEURAL NETWORKS

CONTENTS

- Recurrent Neural Networks
  - Definition
  - Types of RNNs
  - Backpropagation in RNNs
- Time Series Forecasting
  - Naive Forecasting
  - MLP
  - Simple RNN
  - Deep RNNs
- Forecasting several times ahead
  - One step at a time
  - Predicting several values
  - Training for all steps
  - Confidence intervals
- Adding memory
  - LSTMs
  - GRUs
- Preview of Convolutions

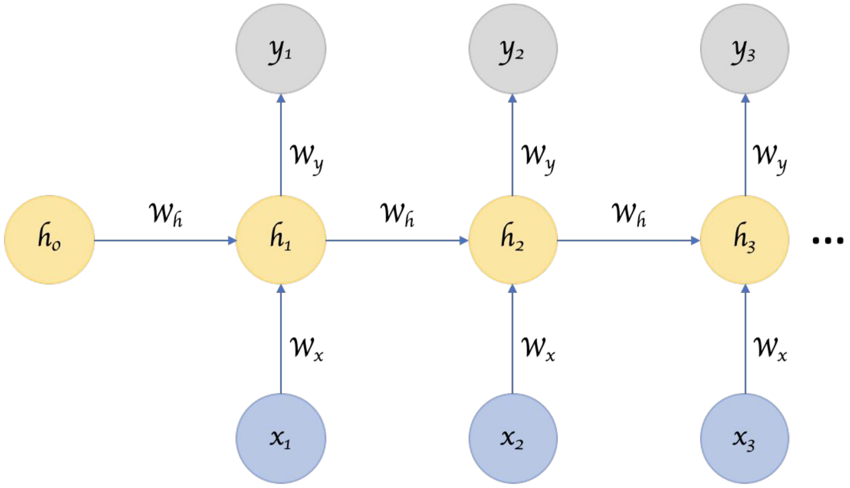




LSTM
(Long-Short Term Memory)

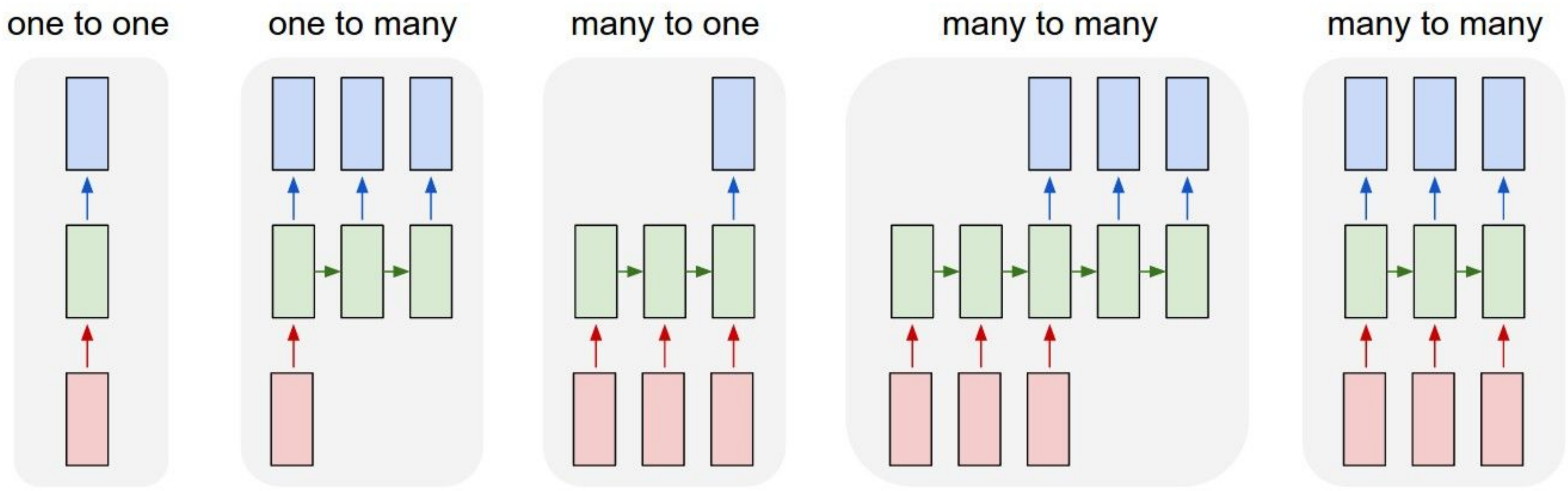
Recurrent Neural Networks

Definition

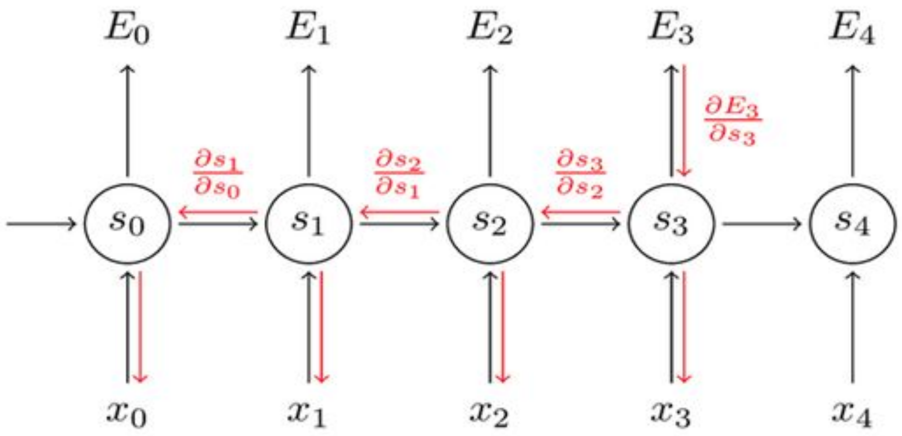


$$y_t = \mathbf{W}_y \mathbf{h}_t = \mathbf{W}_y f(\mathbf{W}_x \mathbf{x}_t + \mathbf{W}_h \mathbf{h}_{t-1})$$

Types of RNNs

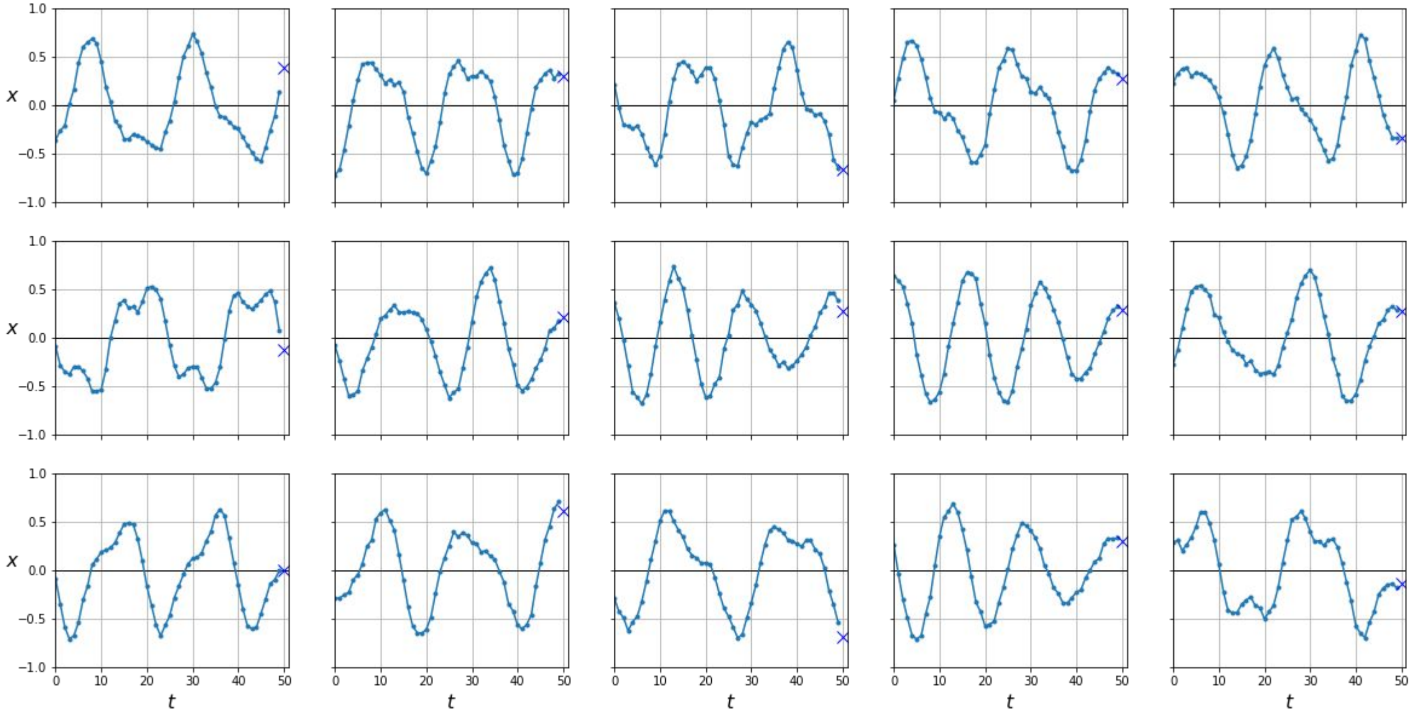


Backpropagation in RNNs



Backpropagation Through Time

Time Series Forecasting

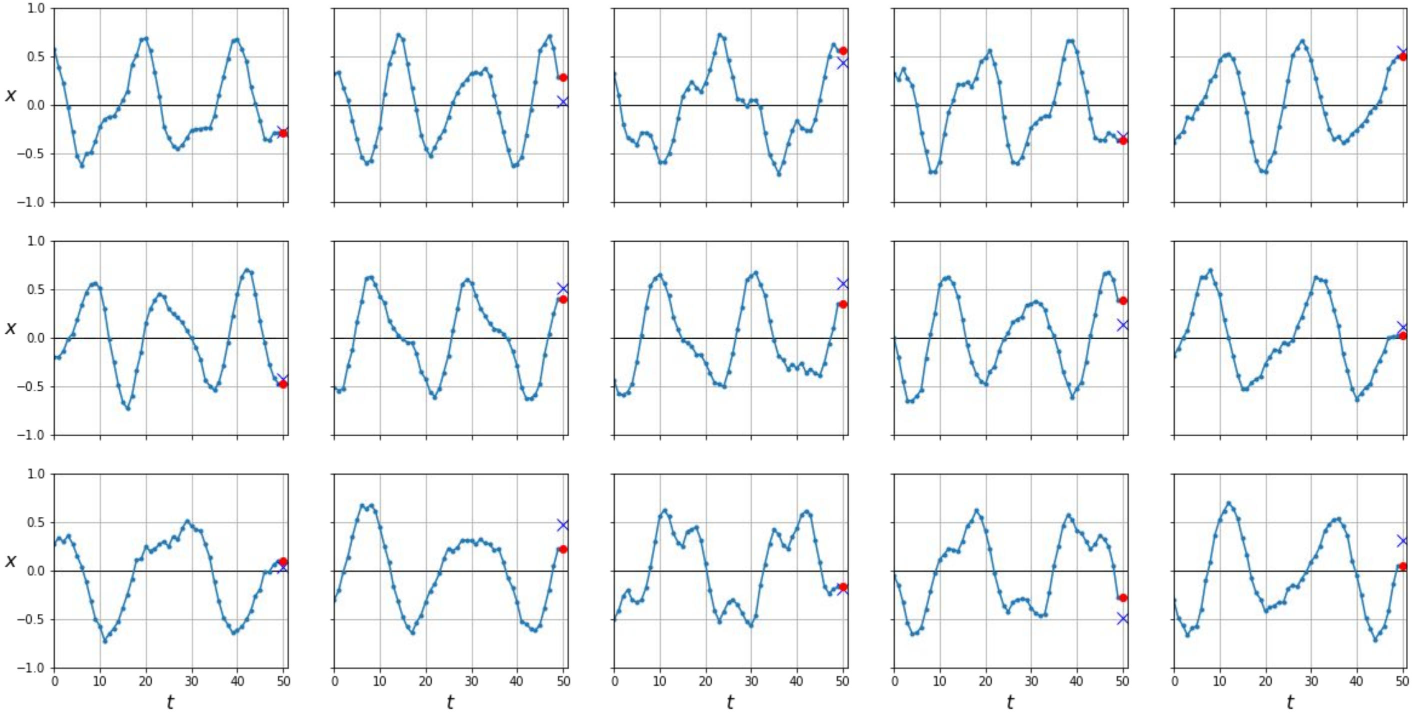


1.0
0.5
0.0
−0.5
−1.0
x
0
10
20
30
40
50
t

Naive forecasting

Just predict the last value.

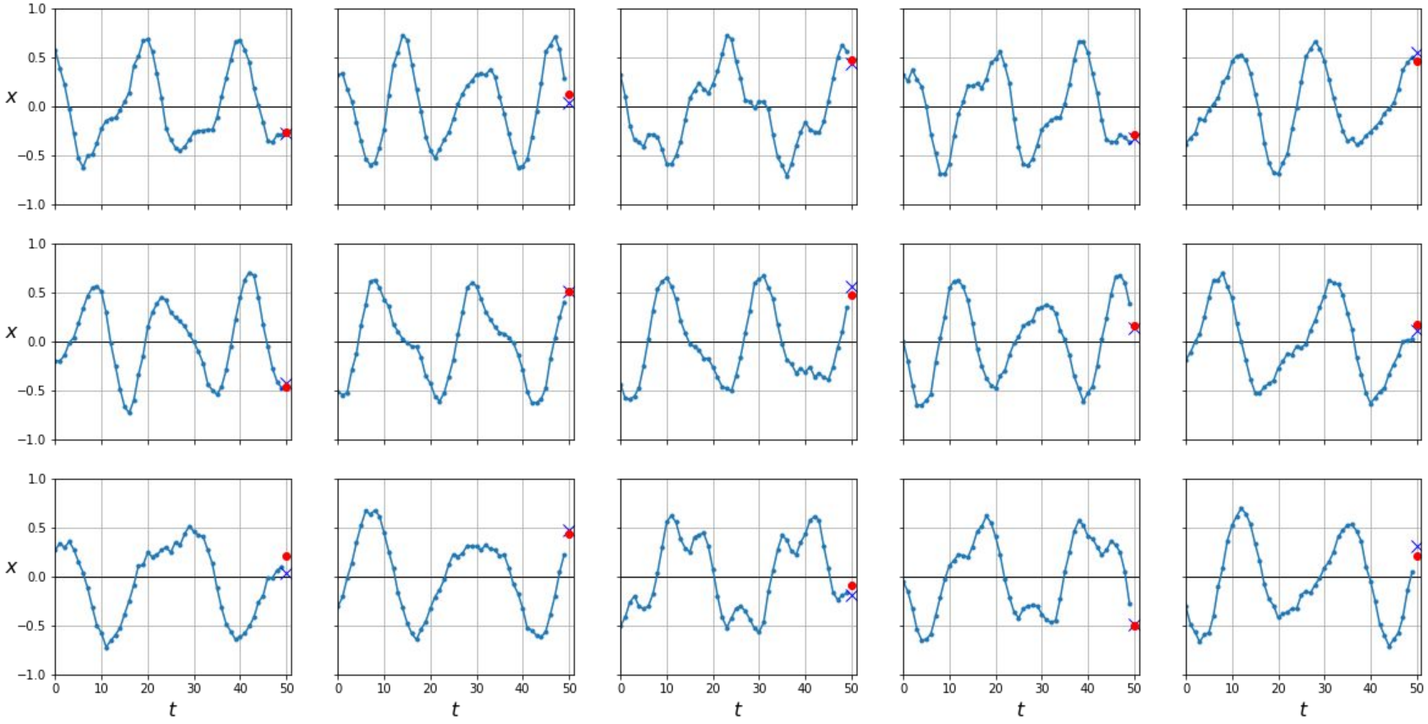
```
y_pred = X_test[:,-1]

mean_squared_error(y_test, y_pred)
> 0.02157
```



1.0
0.5
0.0
−0.5
−1.0
x
0
10
20
30
40
50
t

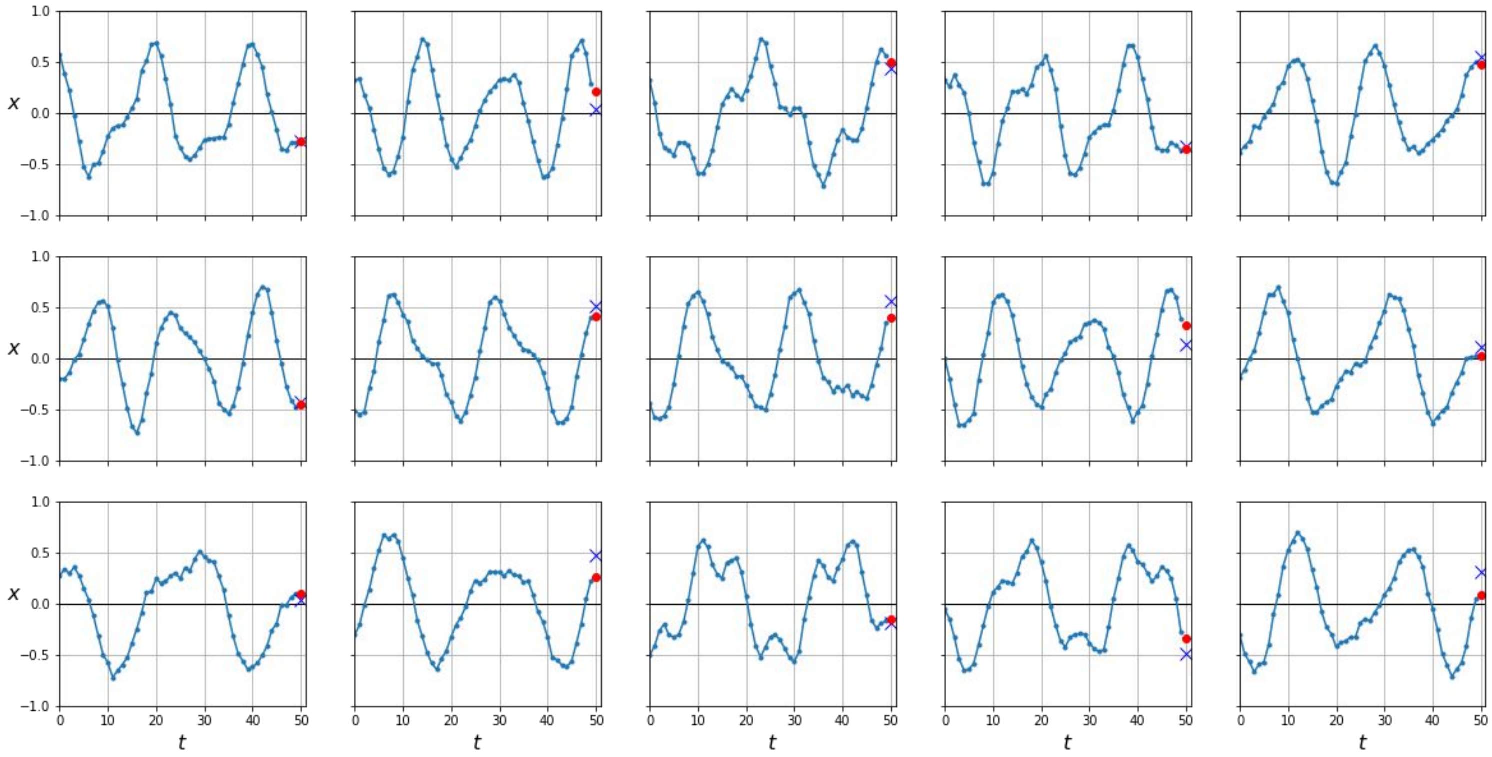
MLP

```
class MLP(torch.nn.Module):
  def __init__(self, n_in=50, n_out=1):
    super().__init__()
    self.fc = torch.nn.Linear(n_in, n_out)

  def forward(self, x):
    x = x.view(x.shape[0], -1)
    x = self.fc(x)
    return x

mean_squared_error(y_test, y_pred)
> 0.00481
```


x
t
1.0
0.5
0.0
−0.5
−1.0
0
10
20
30
40
50

Simple RNN

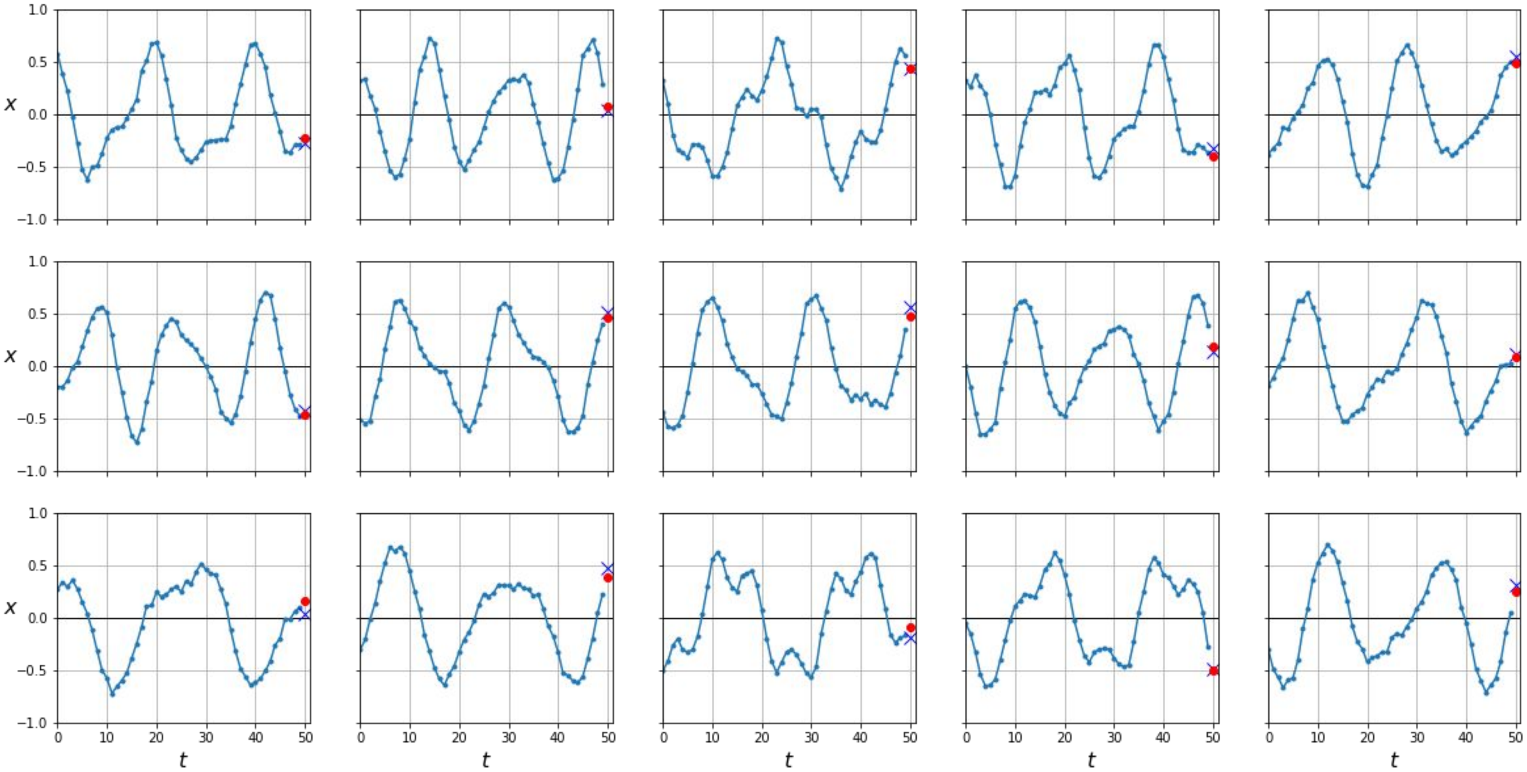
```
class SimpleRNN(torch.nn.Module):
  def __init__(self):
    super().__init__()
    self.rnn = torch.nn.RNN(input_size=1, hidden_size=1, num_layers=1, batch_first=True)

  def forward(self, x):
    x, h = self.rnn(x)
    # return the last output
    return x[:,-1]

mean_squared_error(y_test, y_pred)
> 0.02253
```



1.0
0.5
0.0
−0.5
−1.0
x
t
0
10
20
30
40
50

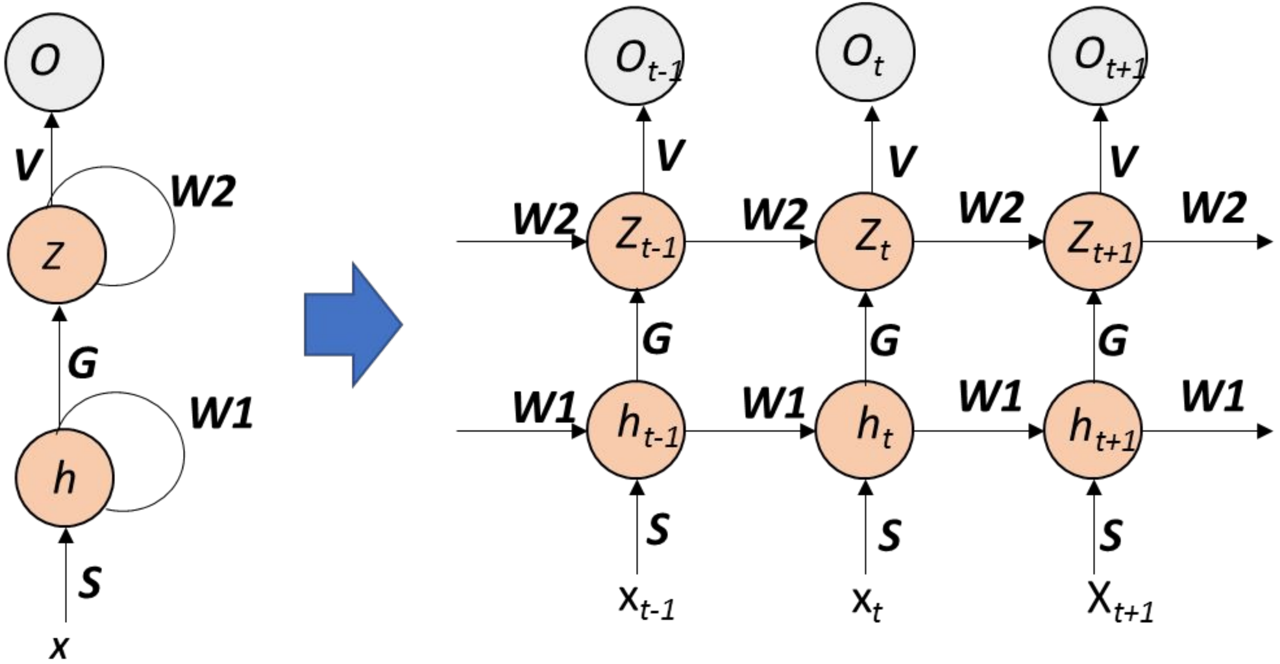
```python
class RNN(torch.nn.Module):
  def __init__(self):
    super().__init__()
    self.rnn = torch.nn.RNN(input_size=1, hidden_size=20, num_layers=1, batch_first=True)
    self.fc = torch.nn.Linear(20, 1)

  def forward(self, x):
    x, h = self.rnn(x)
    # get the last output and apply linear layer
    y = self.fc(x[:,-1])
    return y
```

```
mean_squared_error(y_test, y_pred)
> 0.00343
```



1.0
0.5
0.0
−0.5
−1.0
x
0
10
20
30
40
50
t

Deep RNNs

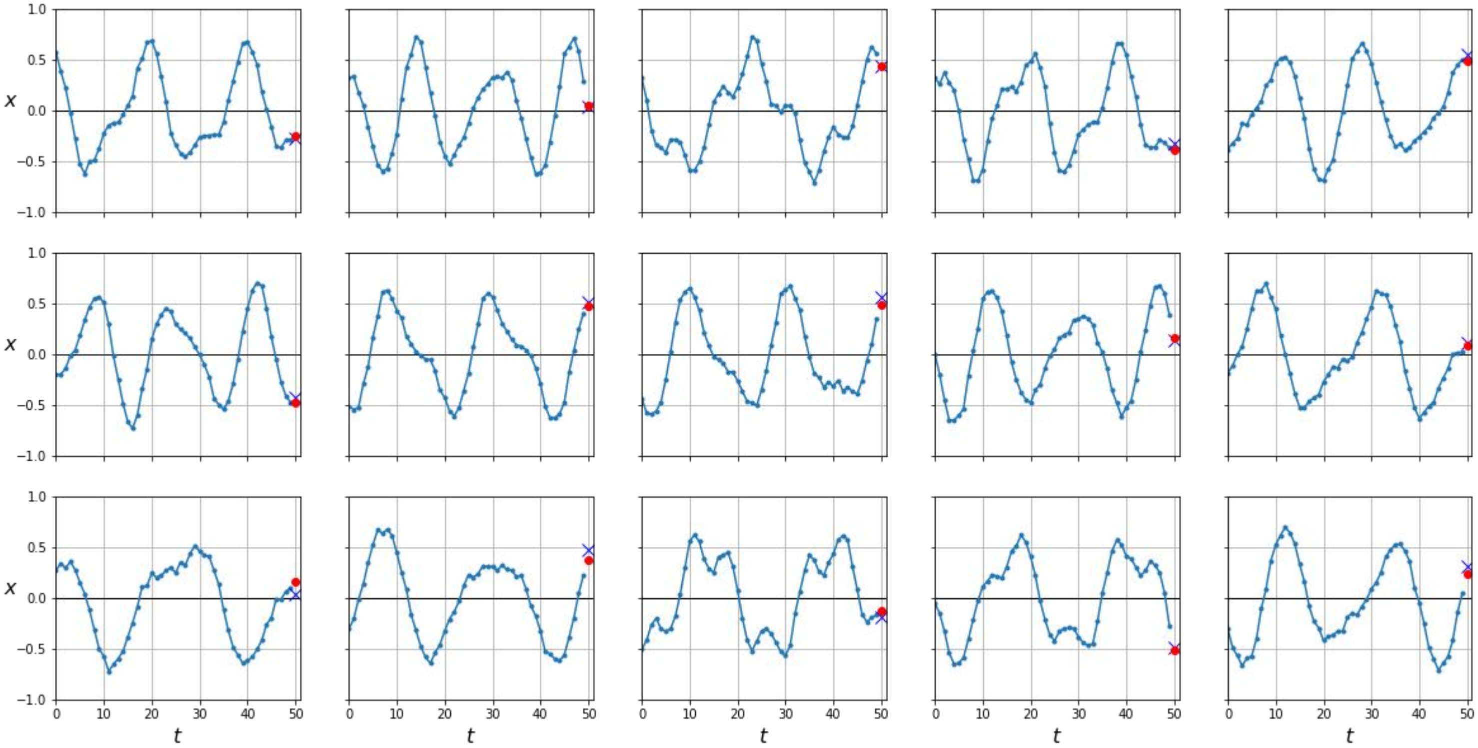


a) 2-layer Recurrent Neural Network (RNN)

b) Unfolded 2-layer Recurrent Neural Network (RNN)

```python
class DeepRNN(torch.nn.Module):
  def __init__(self, n_in=50, n_out=1):
    super().__init__()
    self.rnn = torch.nn.RNN(input_size=1, hidden_size=20, num_layers=2, batch_first=True)
    self.fc = torch.nn.Linear(20, 1)

  def forward(self, x):
    x, h = self.rnn(x)
    # get the last output
    x = self.fc(x[:,-1])
    return x
```

```
mean_squared_error(y_test, y_pred)
> 0.003229
```

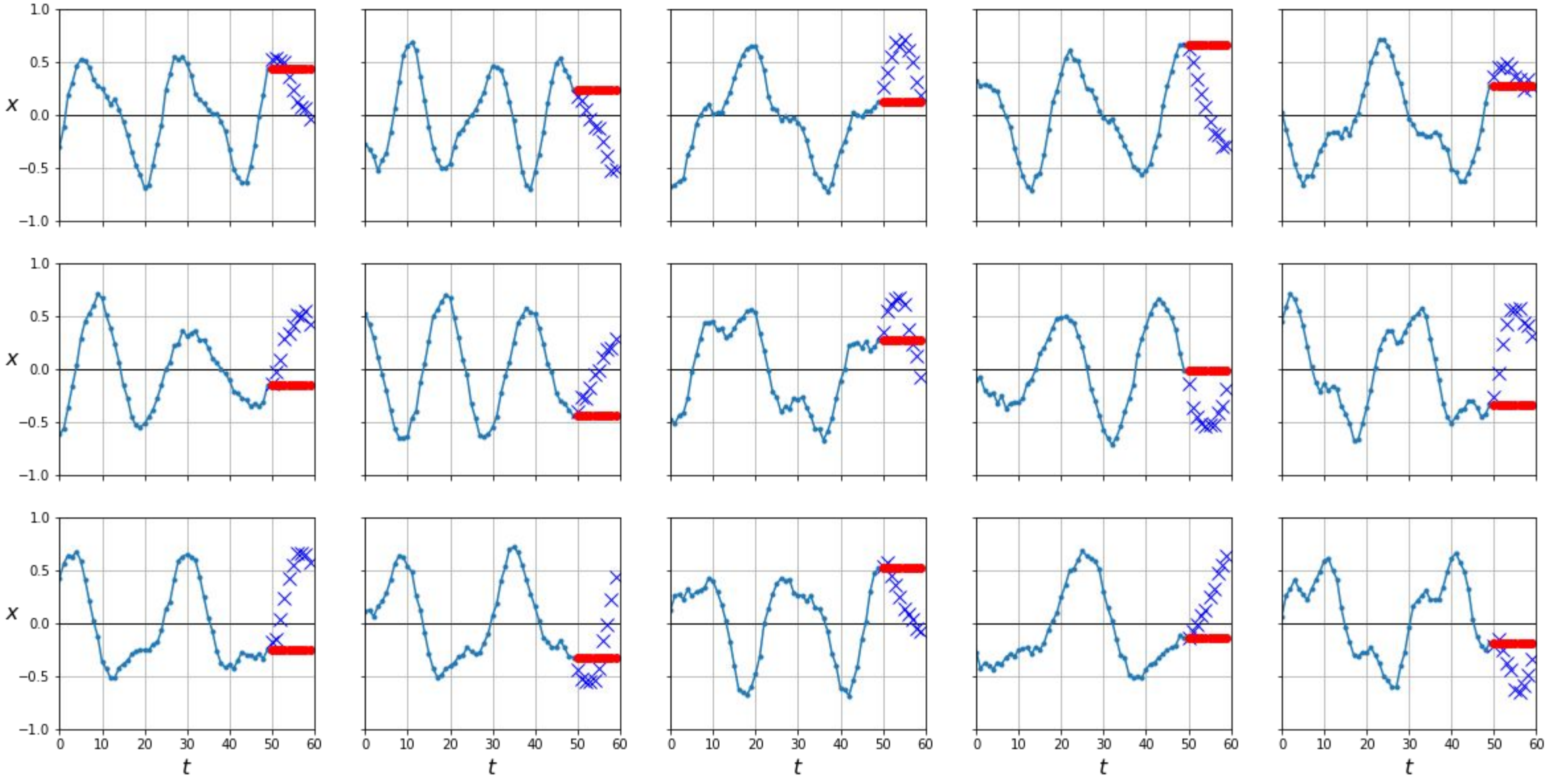


1.0
0.5
0.0
−0.5
−1.0
x
t
0
10
20
30
40
50

Forecasting several times ahead

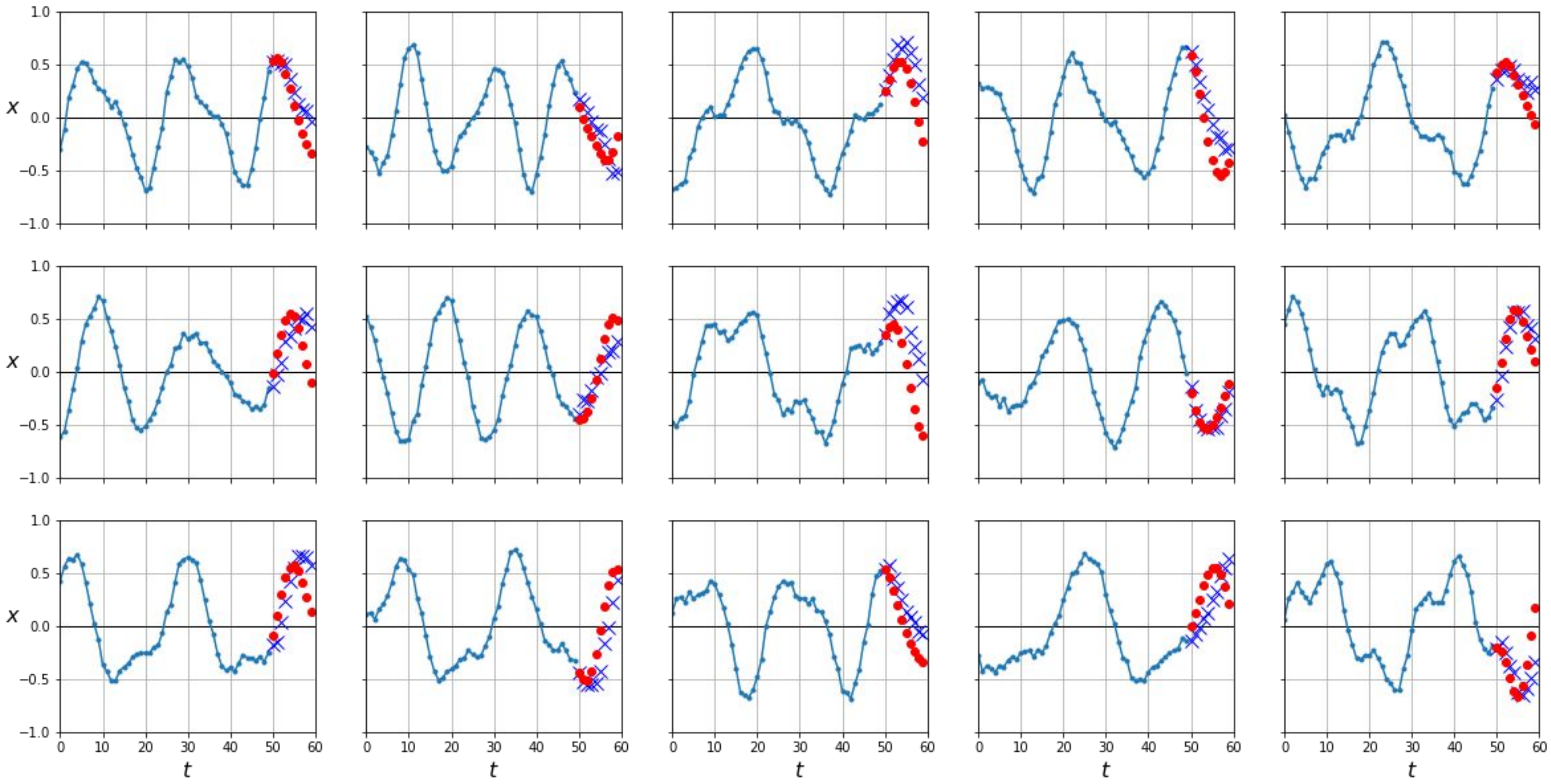
Naive forecasting

```
y_pred = X_test[:,-1]
for step_ahead in range(9):
    y_pred = np.concatenate([y_pred, X_test[:,-1]], axis=1)

mean_squared_error(Y_test, y_pred)
> 0.2627
```


1.0
0.5
0.0
−0.5
−1.0
x
0
10
20
30
40
50
60
t

Deep RNN (one step at a time)

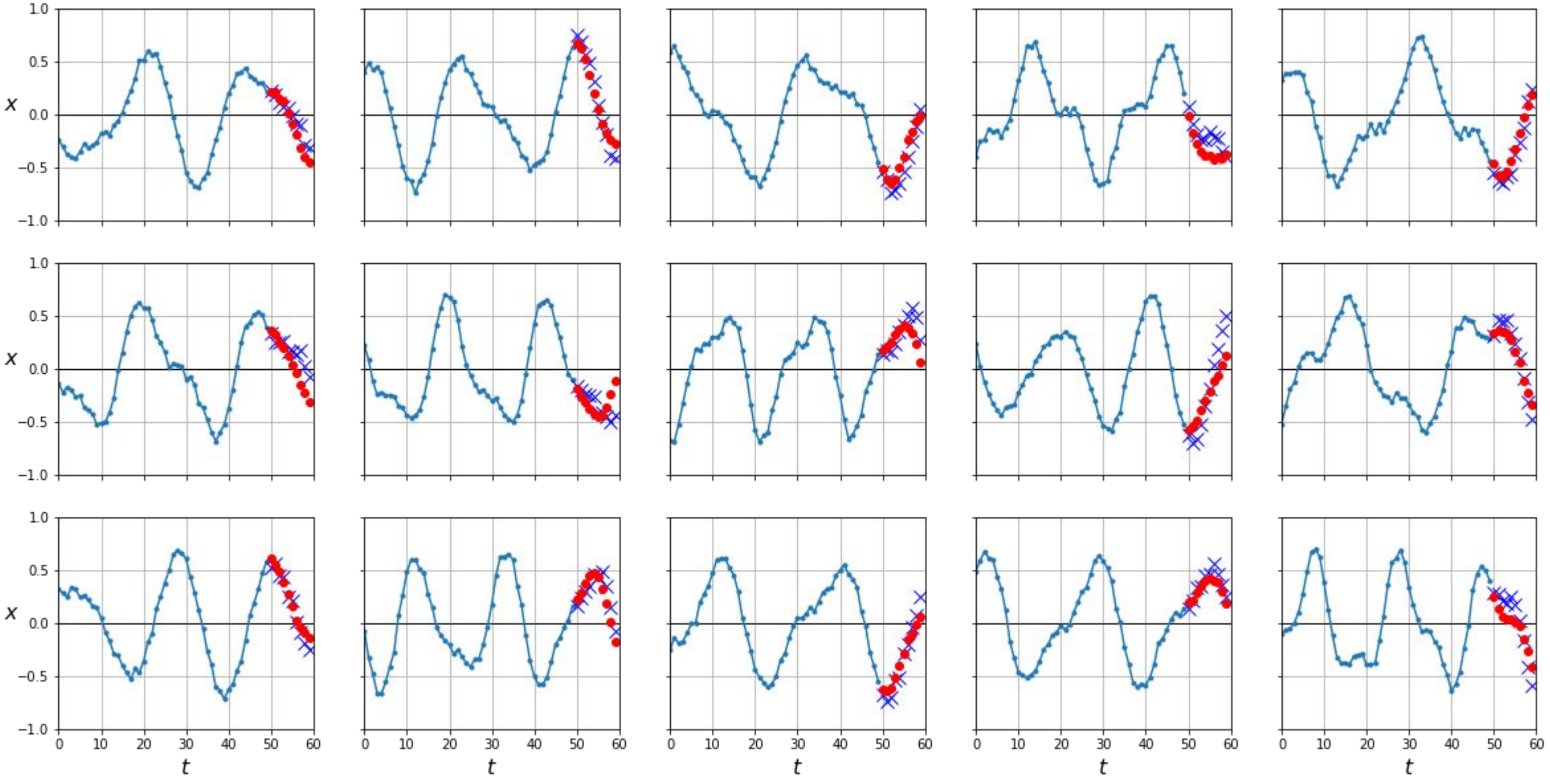
```
X = X_test
for step_ahead in range(10):
    inputs = torch.from_numpy(X[:, step_ahead:]).unsqueeze(0)
    y_pred_one = model.predict(inputs).cpu().numpy()
    X = np.concatenate([X, y_pred_one[:, np.newaxis, :]], axis=1)

mean_squared_error(Y_test, y_pred)
> 0.05195
```



1.0
0.5
0.0
−0.5
−1.0
x
t
0
10
20
30
40
50
60

Predicting several values

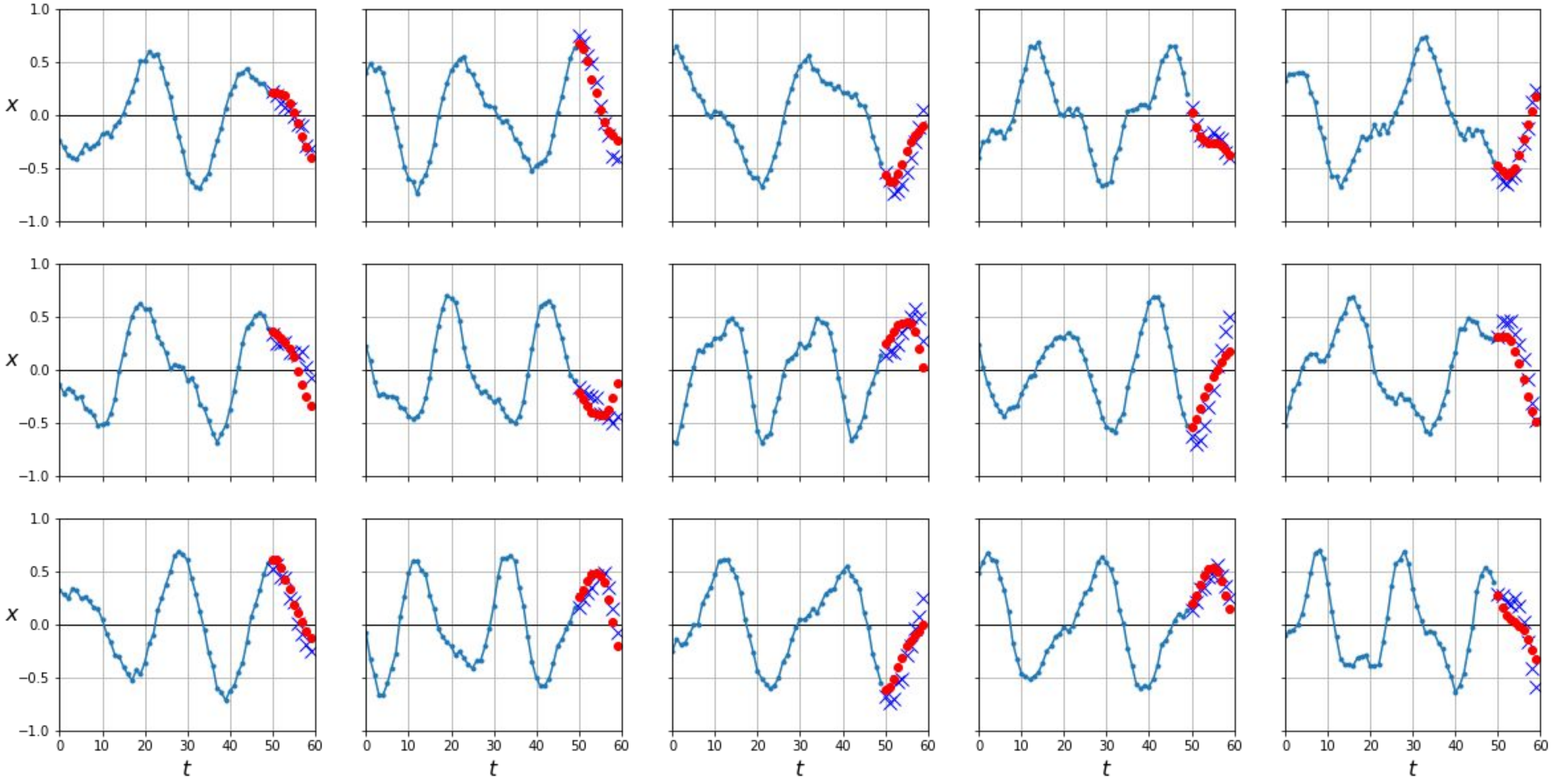
```
class DeepRNN(torch.nn.Module):
  def __init__(self, n_out=10):
    super().__init__()
    self.rnn = torch.nn.RNN(input_size=1, hidden_size=20, num_layers=2, batch_first=True)
    self.fc = torch.nn.Linear(20, n_out)

  def forward(self, x):
    x, h = self.rnn(x)
    x = self.fc(x[:,-1])
    return x

mean_squared_error(Y_test, y_pred)
> 0.011771
```



1.0
0.5
0.0
−0.5
−1.0
x
0
10
20
30
40
50
60
t

Training for all steps

```
class DeepRNN(torch.nn.Module):
  def __init__(self, n_out=10):
    super().__init__()
    self.rnn = torch.nn.RNN(input_size=1, hidden_size=20, num_layers=2, batch_first=True)
    self.fc = torch.nn.Linear(20, n_out)

  def forward(self, x):
    x, h = self.rnn(x)
    # reshape rnn output to feed fc
    # [ Batch, time steps, features ] --> [ Batch x time steps, features ]
    x_reshaped = x.contiguous().view(-1, x.size(-1))
    y = self.fc(x_reshaped)
    # reset to original shape
    # [ Batch x time steps, features ] --> [ Batch, time steps, features ]
    y = y.contiguous().view(x.size(0), -1, y.size(-1))
    return y

mean_squared_error(Y_test[:,-1], y_pred[:,-1])
> 0.015259
```

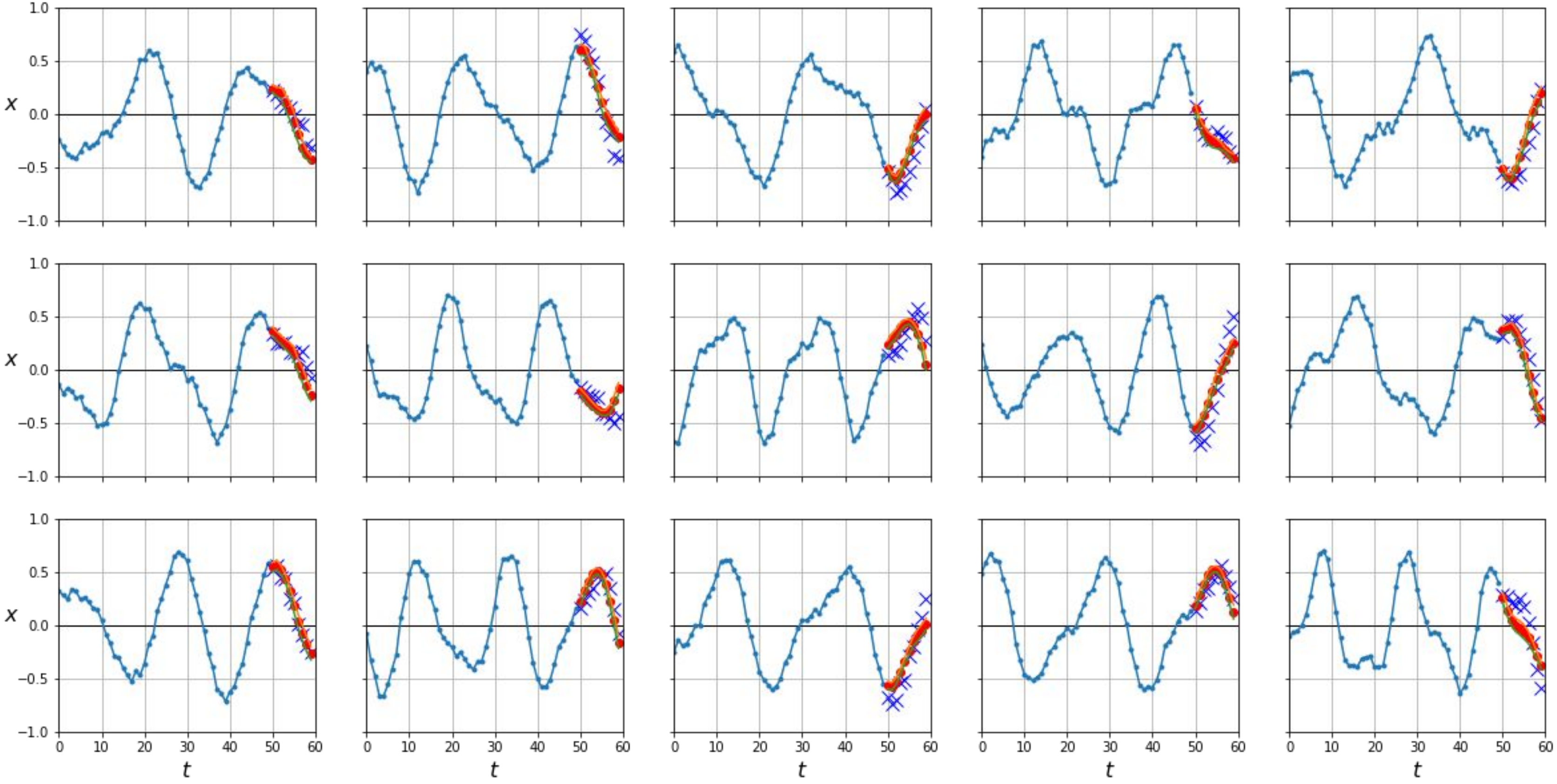


1.0
0.5
0.0
−0.5
−1.0
x
0
10
20
30
40
50
60
t

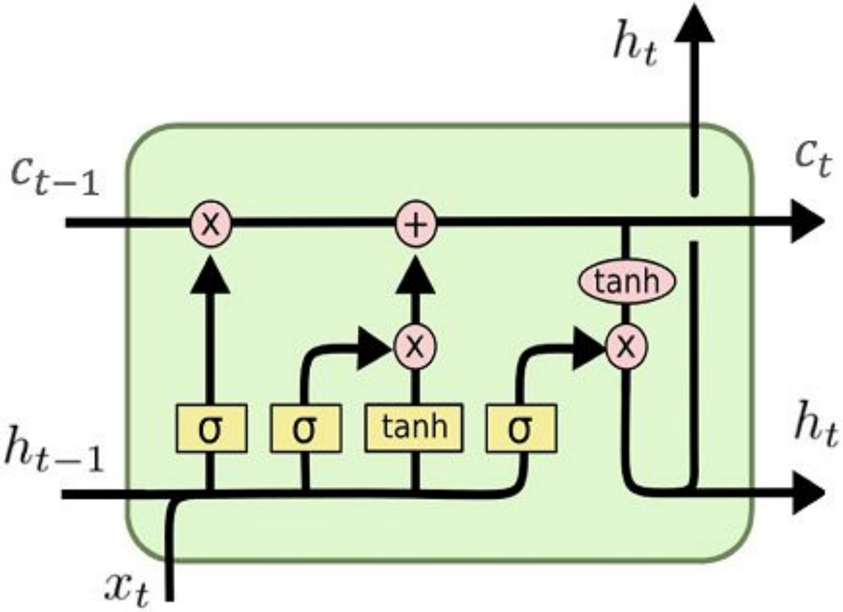
Confidence intervals

We can provide confidence intervals with *MC Dropout* (do predictions with dropout on and average)

```
class DeepRNN(torch.nn.Module):
  def __init__(self, n_out=10, dropout=0.5):
    super().__init__()
    self.rnn = torch.nn.RNN(input_size=1, hidden_size=20, num_layers=2, dropout=dropout,
batch_first=True)
    self.fc = torch.nn.Linear(20, n_out)

  def forward(self, x):
    x, h = self.rnn(x)
    # reshape rnn output to feed fc
    # [ Batch, time steps, features ] --> [ Batch x time steps, features ]
    x_reshaped = x.contiguous().view(-1, x.size(-1))
    y = self.fc(x_reshaped)
    # reset to original shape
    # [ Batch x time steps, features ] --> [ Batch, time steps, features ]
    y = y.contiguous().view(x.size(0), -1, y.size(-1))
    return y
```


x
t
1.0
0.5
0.0
−0.5
−1.0
0
10
20
30
40
50
60

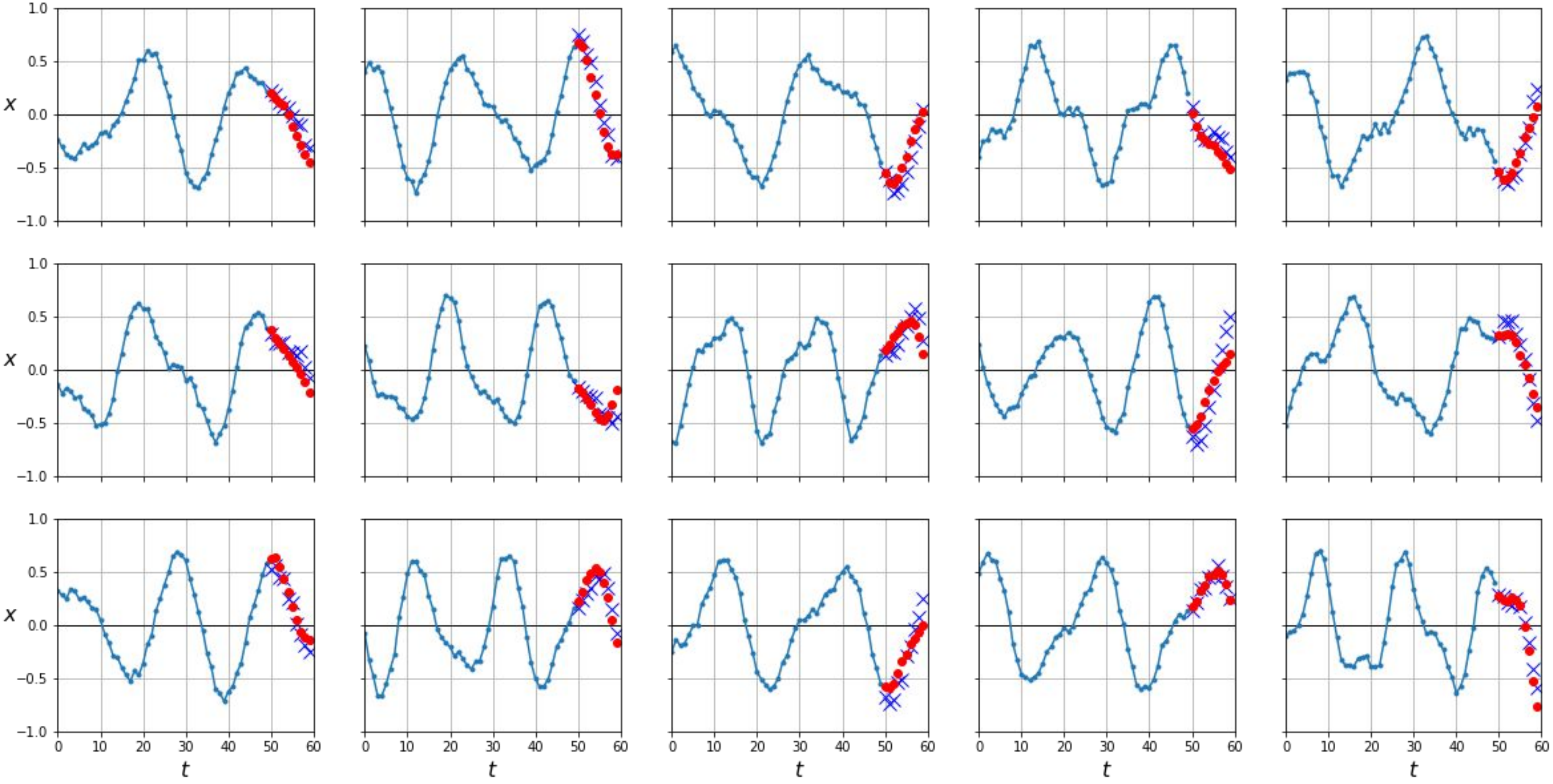
Adding memory

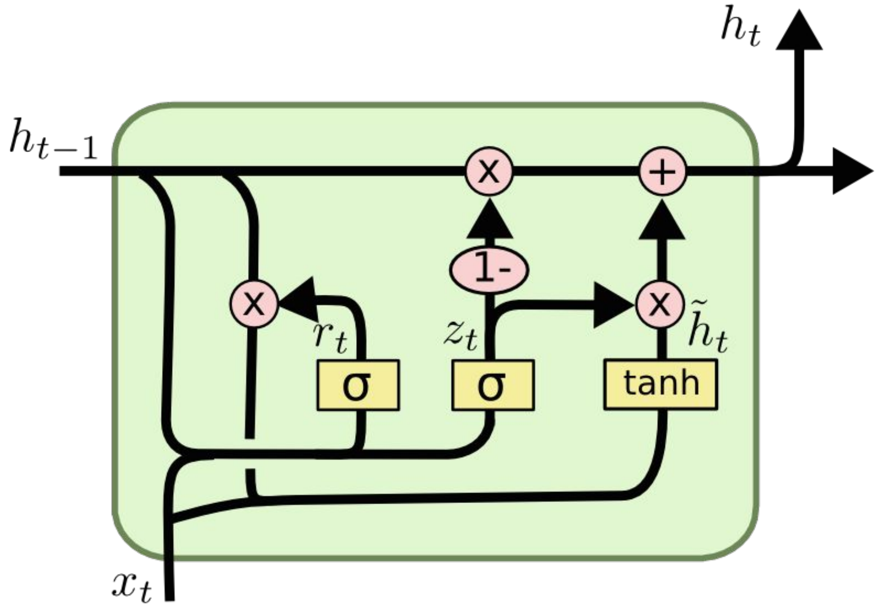


LSTM
(Long-Short Term Memory)

LSTM

```
class LSTM(DeepRNN):
  def __init__(self, n_out=10, dropout=0):
    super().__init__()
    self.rnn = torch.nn.LSTM(input_size=1, hidden_size=20, num_layers=2,
dropout=dropout, batch_first=True)

mean_squared_error(Y_test[:,-1], y_pred[:,-1])
> 0.00887
```

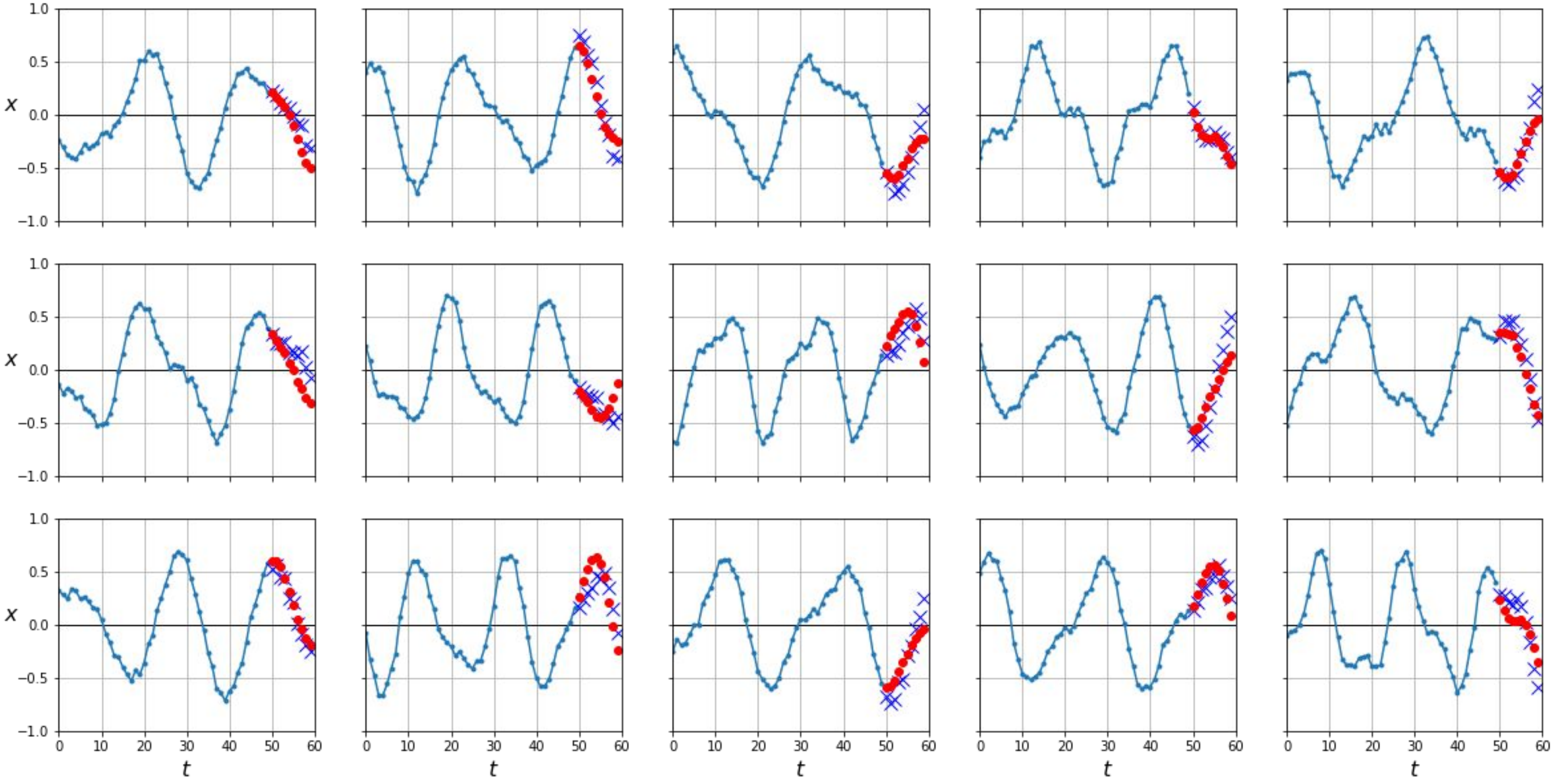

1.0
0.5
0.0
−0.5
−1.0
x
0
10
20
30
40
50
60
t



GRU
(Gated Recurrent Unit)

GRU

```
class GRU(DeepRNN):
  def __init__(self, n_out=10, dropout=0):
    super().__init__()
    self.rnn = torch.nn.GRU(input_size=1, hidden_size=20, num_layers=2, dropout=dropout,
batch_first=True)

mean_squared_error(Y_test[:,-1], y_pred[:,-1])
> 0.01165
```

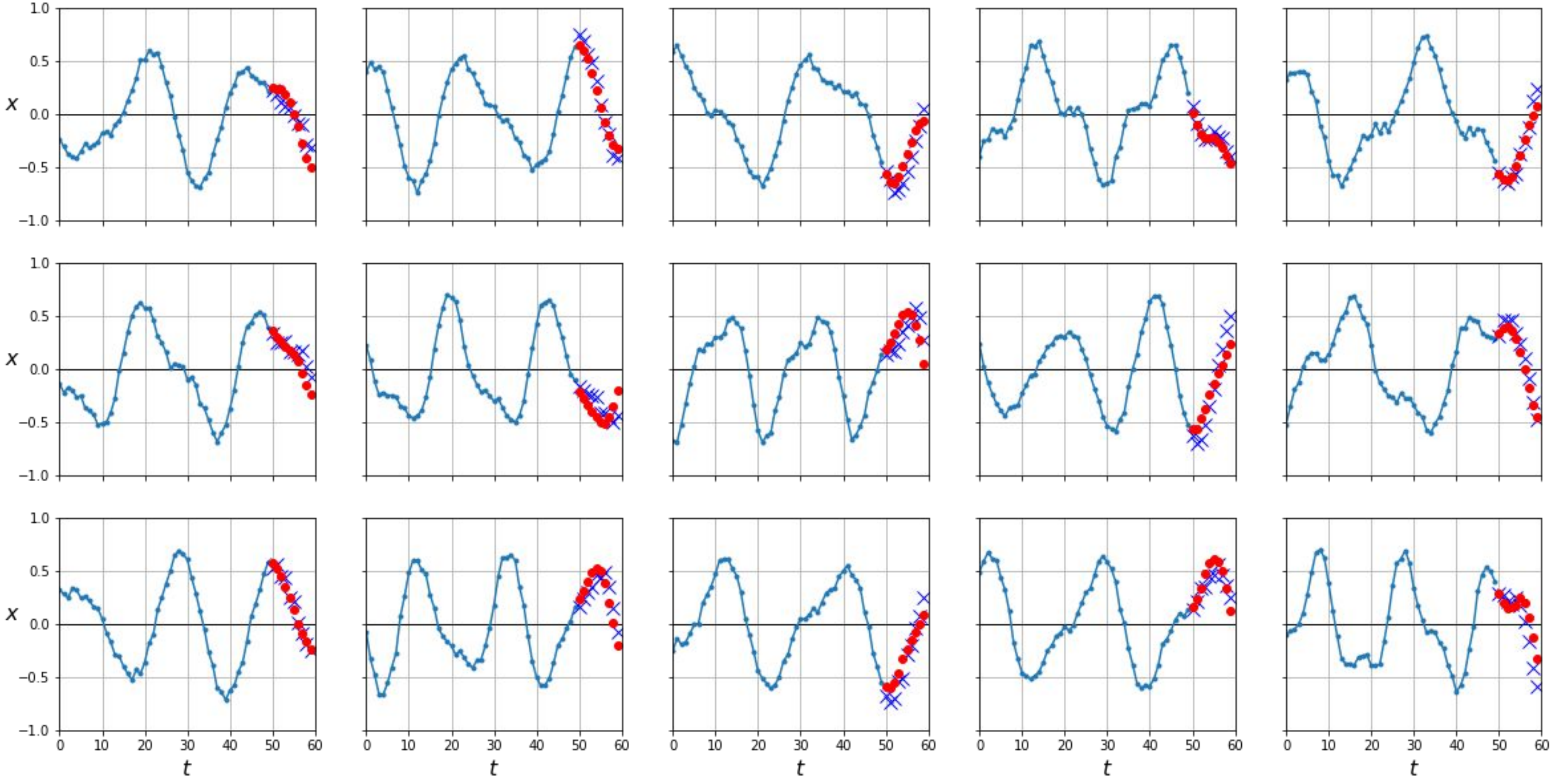


x
t

Preview of Convolutions

We will talk about CNNs later, but for now keep in mind that they can also be used to process sequence data, often achieving better results.

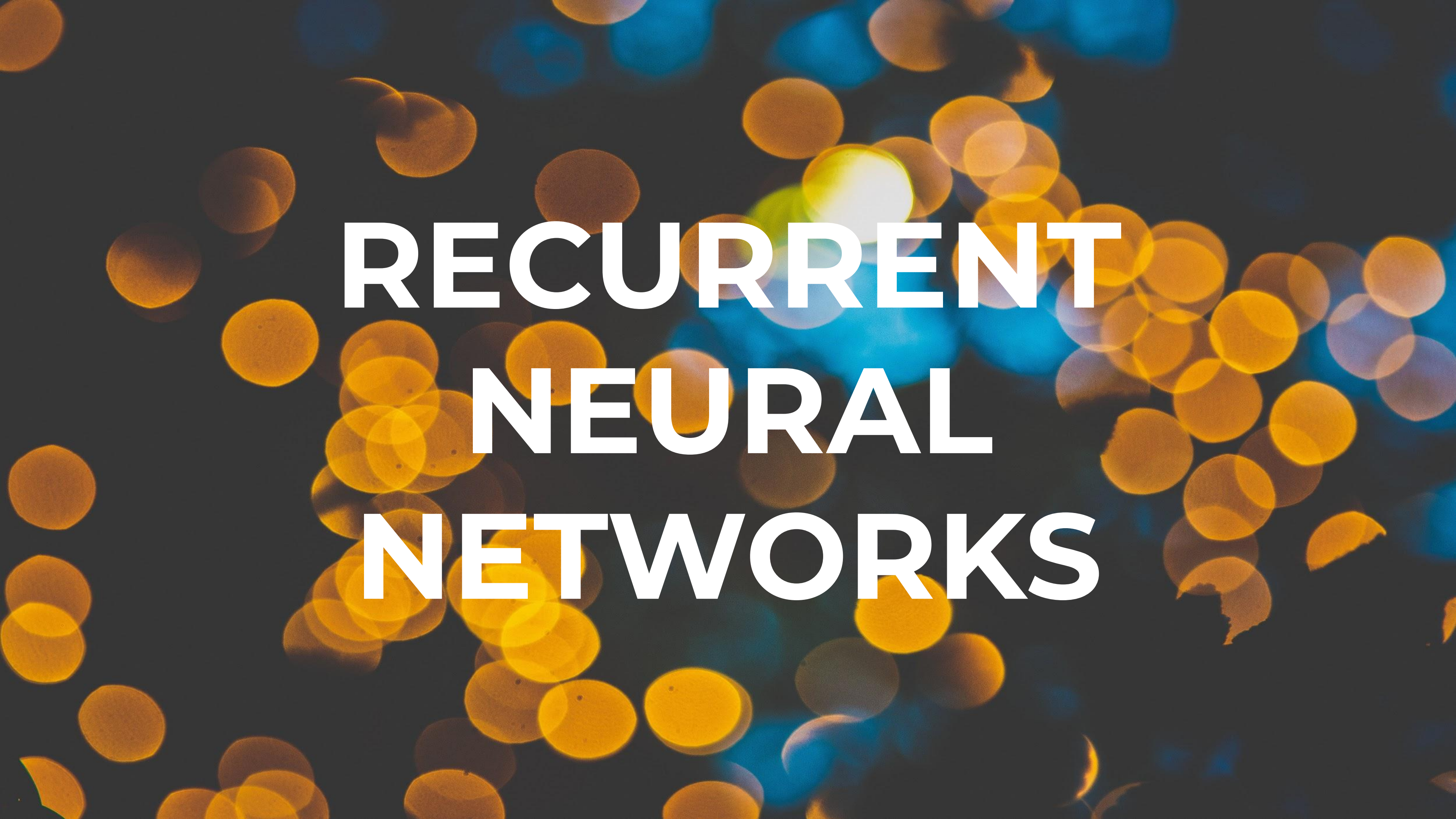
```
class ConvRNN(torch.nn.Module):
  def __init__(self, n_out=10, dropout=0):
    super().__init__()
    self.conv = torch.nn.Conv1d(1, 20, 4, stride=2, padding=0)
    self.rnn = torch.nn.GRU(input_size=20, hidden_size=20, num_layers=2,
dropout=dropout, batch_first=True)
    self.fc = torch.nn.Linear(20, n_out)

  def forward(self, x):
    x = self.conv(x.permute(0,2,1))
    x, h = self.rnn(x.permute(0,2,1))
    x_reshaped = x.contiguous().view(-1, x.size(-1))
    y = self.fc(x_reshaped)
    y = y.contiguous().view(x.size(0), -1, y.size(-1))
    return y

mean_squared_error(Y_test[:,-1], y_pred[:,-1])
> 0.009064
```



x
t
1.0
0.5
0.0
−0.5
−1.0
0
10
20
30
40
50
60

https://github.com/sensioai/dl/blob/master/rnns/rnns.ipynb

https://www.youtube.com/watch?v=_h66BW-xNgk


RECURRENT
NEURAL
NETWORKS