cw2_solution

October 1, 2019

1 COMP3222/6246 Machine Learning Technologies (2019/20)

2 Partial Solution of Coursework 2

1. Motivated by the use of CNN for MNIST dataset, now you are to try CNN on the Sign Language MNIST next. In this task, you are given a dataset of hand gestures that are labelled with an American Sign Language letter (except for J and Z which need hand motion). The dataset can be retrieved from either https://drive.google.com/file/d/1zkX8oQ74JFcJ7Gli6M_qnnGo41tMeZQI/view?usp=sharing or the original source on Kaggle: https://www.kaggle.com/datamunge/sign-language-mnist/download. Its format is very similar to the classic MNIST (28 x 28 pixels and each label is a number representing a letter from A to Z -- but excluding 9=J and 25=Z). It is already partitioned into a training set and a testing set.

1.(a) Implement a CNN and train it on the training set with the Gradient Descent optimiser. You are free to set the structure and the hyperparameters by your own. However, please write a short description of them and a justification of your choices.

1.(b) Now replace the Gradient Descent with a Stochastic Gradient Descent (SGD) optimiser. Demonstrate how much does the CNN improve. Also justify your demonstration technique --- why did you demonstrate in such a way? (You must also explain if there is any change to the CNN's structure).

1.(c) Finally, replace SGD with the Adam optimiser and redo the previous subtask. Is it better to use Adam?

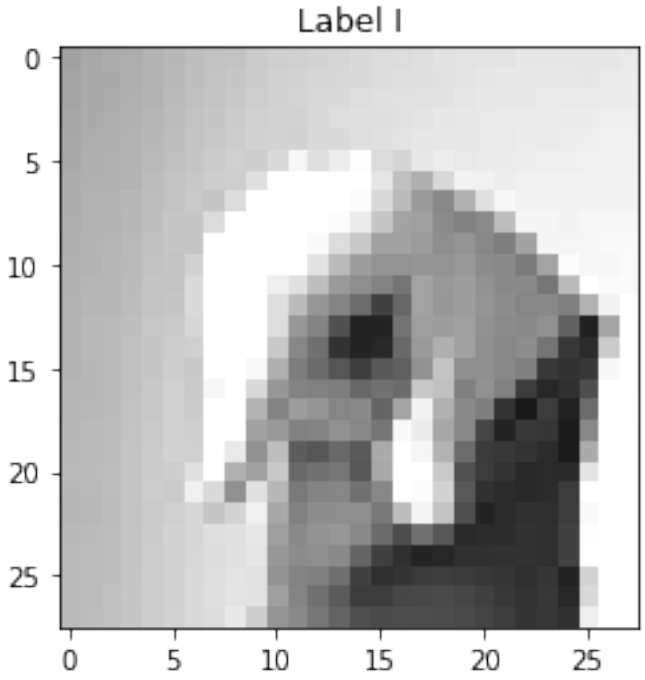
[1]:
```
sign_letters = [chr(x) for x in range(65,91)]
print(sign_letters)
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',
'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
```

[2]:
```
# Import and prepare dataset
import numpy as np
import pandas as pd

test = pd.read_csv("./test/sign_mnist_test.csv")
train = pd.read_csv("./train/sign_mnist_train.csv")
total_pixels = 28 * 28
```

```
X_train = train.loc[:, "pixel1":].to_numpy().astype(np.float32) / 255.0
y_train = train.loc[:, "label"].to_numpy().astype(np.int32)

X_test = test.loc[:, "pixel1":].to_numpy().astype(np.float32) / 255.0
y_test = test.loc[:, "label"].to_numpy().astype(np.int32)

X_train = X_train.reshape(X_train.shape[0],28,28,1)
X_test = X_test.reshape(X_test.shape[0],28,28,1)
```

[4]:
```
# Verify the data
import matplotlib.pyplot as plt

image_index = 10
one_image = X_train[5]

one_image = one_image.reshape(28, 28)
plt.title("Label {}".format(sign_letters[test.loc[image_index, "label"]]))
plt.imshow(one_image.astype(np.float32), cmap="gray", vmin=0, vmax=1.0)
plt.show()
```



[5]:
```
# Build a CNN
import tensorflow as tf
from tensorflow.keras import layers, models
```

```
height = 28
width = 28
channels = 1

conv1_fmaps = 32
conv1_ksize = 3
conv1_stride = 1
conv1_pad = 'same'

conv2_fmaps = 64
conv2_ksize = 3
conv2_stride = 2
conv2_pad = 'same'

n_fc = 64
n_outputs = 25

model = models.Sequential()
model.add(layers.Conv2D(filters=conv1_fmaps, kernel_size=conv1_ksize,␣
 ↪strides=conv1_stride,
                         padding=conv1_pad, activation='relu',␣
 ↪input_shape=(height, width, channels)))
model.add(layers.Conv2D(filters=conv2_fmaps, kernel_size=conv2_ksize,␣
 ↪strides=conv2_stride,
                         padding=conv2_pad, activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2),␣
 ↪padding='valid'))
model.add(layers.Flatten())
model.add(layers.Dense(units=n_fc, activation='relu'))
model.add(layers.Dense(units=n_outputs))
model.add(layers.Softmax())

model.summary()
```

```
WARNING:tensorflow:From C:\Local\anaconda3\envs\MLTech\lib\site-
packages\tensorflow\python\ops\init_ops.py:1251: calling
VarianceScaling.__init__ (from tensorflow.python.ops.init_ops) with dtype is
deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the
constructor
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        320
```

```
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 7, 7, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
dense (Dense)                (None, 64)                200768
_________________________________________________________________
dense_1 (Dense)              (None, 25)                1625
_________________________________________________________________
softmax (Softmax)            (None, 25)                0
=================================================================
Total params: 221,209
Trainable params: 221,209
Non-trainable params: 0
_________________________________________________________________
```

[6]:
```
# Compile and train CNN
from tensorflow.keras import losses

model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
27455/27455 [==============================] - 18s 656us/sample - loss: 3.0166 -
acc: 0.1153
Epoch 2/10
27455/27455 [==============================] - 18s 659us/sample - loss: 1.7996 -
acc: 0.4378
Epoch 3/10
27455/27455 [==============================] - 18s 656us/sample - loss: 1.1269 -
acc: 0.6415
Epoch 4/10
27455/27455 [==============================] - 18s 657us/sample - loss: 0.7930 -
acc: 0.7485
Epoch 5/10
27455/27455 [==============================] - 18s 655us/sample - loss: 0.5490 -
acc: 0.8256 - loss: 0.5552
Epoch 6/10
27455/27455 [==============================] - 18s 655us/sample - loss: 0.3590 -
acc: 0.8900
Epoch 7/10
27455/27455 [==============================] - 18s 656us/sample - loss: 0.2153 -
acc: 0.9382
Epoch 8/10
27455/27455 [==============================] - 18s 656us/sample - loss: 0.1341 -
```

```
acc: 0.9630
Epoch 9/10
27455/27455 [==============================] - 18s 654us/sample - loss: 0.0836 -
acc: 0.9814
Epoch 10/10
27455/27455 [==============================] - 18s 656us/sample - loss: 0.0639 -
acc: 0.9884
```

[6]:
```
<tensorflow.python.keras.callbacks.History at 0x2abf8104248>
```

[7]:
```
# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
```

```
7172/7172 [==============================] - 1s 163us/sample - loss: 0.7324 -
acc: 0.8252
```

Accuracy on testing set of CNN trained with SGD optimiser is reported above. You can easily change the optimizer during the compilation with optimizer='adam'.

2. In this task, you have to do time series prediction using RNNs. In particular, we aim to predict the internet traffic data (the dataset itself can be found here - https://secure.ecs.soton.ac.uk/noteswiki/images/internet-traffic-data-in-bits-fr.xlsx - or you can find the link to this data set from the module web site). It contains internet traffic data (in bits) of a academic backbone network in the UK. It was collected between 19 November 2004 and 27 January 2005. Data were collected at five minute intervals.

[8]:
```
# Get the data
import numpy as np
import pandas as pd

data = pd.read_excel("internet-traffic-data-in-bits-fr.xlsx", usecols=1)␣
 ↪#ignore the last null column
data = data.drop(data.index[(len(data)-1)]) # discard irrelevant rows at the end
data = data.drop(data.index[(len(data)-1)])
data.columns = ["time", "bits"] # rename columns
data.head(10)
```

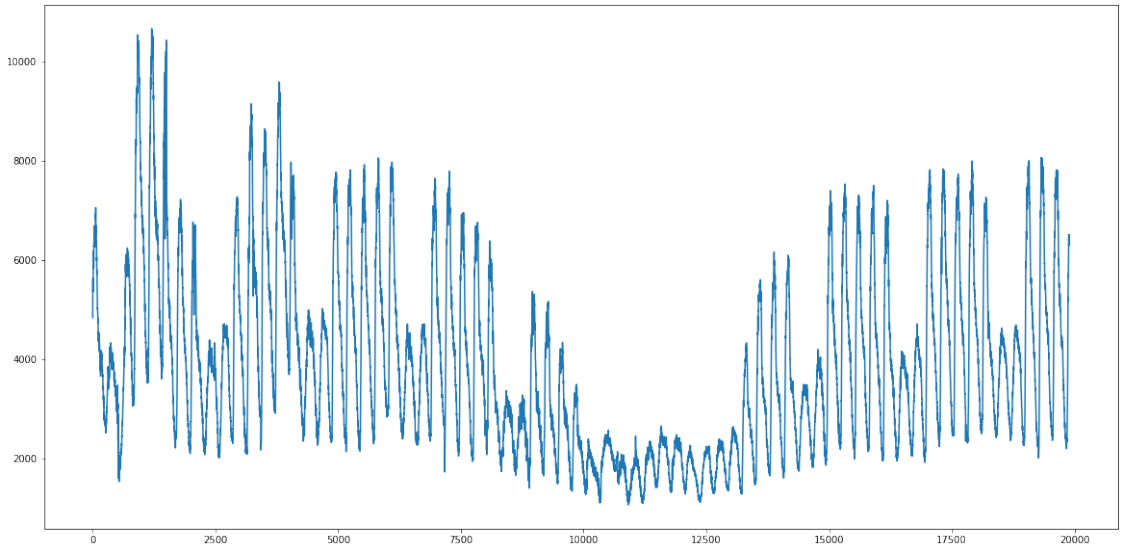
```
C:\Local\anaconda3\envs\MLTech\lib\site-packages\pandas\io\excel\_base.py:445:
FutureWarning: Passing in an integer for `usecols` has been deprecated. Please
pass in a list of int from 0 to `usecols` inclusive instead.
  usecols = _maybe_convert_usecols(usecols)
```

[8]:

	time	bits
0	2004-11-19 09:30:00	4838.66
1	2004-11-19 09:35:00	4845.18
2	2004-11-19 09:40:00	5158
3	2004-11-19 09:45:00	5637.88
4	2004-11-19 09:50:00	5520.69
5	2004-11-19 09:55:00	5626.34

```
6  2004-11-19 10:00:00  5350.55
7  2004-11-19 10:05:00  5356.98
8  2004-11-19 10:10:00  5385.81
9  2004-11-19 10:15:00  5403.91
```

[9]:
```
# Have a visual look on the data
import matplotlib.pyplot as plt

plt.figure(figsize=(20, 10))
plt.plot(data["bits"])
plt.show()
```

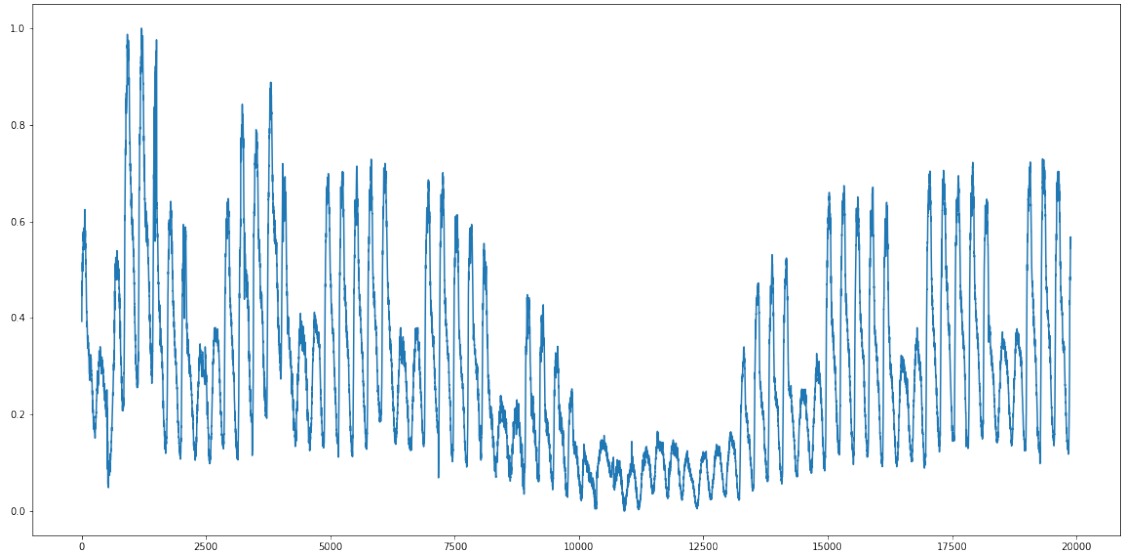


[10]:
```
t_min, t_max = 0, len(data)
t_min, t_max
```

[10]:
```
(0, 19888)
```

[11]:
```
# preprocessing
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
print(scaler.fit(data["bits"].values.reshape(-1, 1)))

normalized_data = scaler.transform(data["bits"].values.reshape(-1, 1)).flatten()
plt.figure(figsize=(20, 10))
plt.plot(normalized_data)
plt.show()
```

```
MinMaxScaler(copy=True, feature_range=(0, 1))
```



2.(a) Implement an RNN and train it on the provided data. You are free to set the hyperparameters by your own. However, please write a short description to justify your choices.

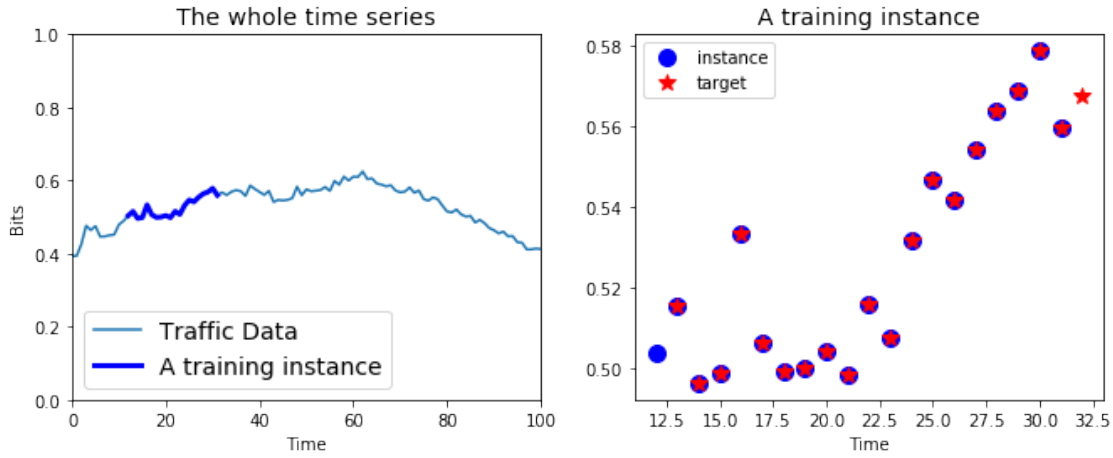
2.(b) Once you have trained the RNN, use the Stochastic Gradient Descent technique to improve the RMSE and plot the results to demonstrate how SGD works (again, you are free to choose the way you demonstrate - however, you need to explain why you did choose to do that way).

2.(c) Finally, replace SGD with Adam and redo the previous subtask. Is it better to use Adam? Explain your answer (hint: compare the number of steps, how many times it takes to converge etc)

[12]:
```
import tensorflow as tf

def reset_graph(seed=42):
    tf.reset_default_graph()
    tf.set_random_seed(seed)
    np.random.seed(seed)
```

[13]:
```
def next_batch(batch_size, n_steps):
    """
    Returns a batch with `n_steps`: number of instances
    """

    t0 = np.random.randint(low=t_min, high=t_max - t_min - n_steps,␣
 ↪size=(batch_size, 1), dtype=np.int32)
    Ts = t0 + np.arange(0., n_steps + 1, dtype=np.int32)
    ys = normalized_data[Ts]
    # return X's and Y's
    return ys[:, :-1].reshape(-1, n_steps, 1), ys[:, 1:].reshape(-1, n_steps, 1)
```

[14]:
```
t = np.arange(t_min, t_max)
n_steps = 20

# a training instance
t_instance = np.arange(12, 12+n_steps+1)


plt.figure(figsize=(11,4))
plt.subplot(121)
plt.title("The whole time series", fontsize=14)
# plot all the data
plt.plot(t,normalized_data[t], label=r"Traffic Data")

# plot only the training set
plt.plot(t_instance[:-1], normalized_data[t_instance[:-1]], "b-", linewidth=3,␣
 ↪label="A training instance")
plt.legend(loc="lower left", fontsize=14)
plt.axis([0, 100, 0, 1])
plt.xlabel("Time")
plt.ylabel("Bits")

plt.subplot(122)
plt.title("A training instance", fontsize=14)
plt.plot(t_instance[:-1], normalized_data[t_instance[:-1]], "bo",␣
 ↪markersize=10, label="instance")
# notice that targets are shifted by one time step into the future
plt.plot(t_instance[1:], normalized_data[t_instance[1:]], "r*", markersize=10,␣
 ↪label="target")
plt.legend(loc="upper left")
plt.xlabel("Time")

plt.show()
```



[15]:
```
reset_graph()

n_steps = 50
n_inputs = 1
n_neurons = 100
n_outputs = 1
n_layers = 3
learning_rate = 0.001
n_iterations = 1000
batch_size = 50

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_steps, n_outputs])

layers = [tf.nn.rnn_cell.BasicRNNCell(num_units=n_neurons, activation=tf.nn.
 ↪relu)
          for layer in range(n_layers)]

multi_layer_cell = tf.nn.rnn_cell.MultiRNNCell(layers)

cell = tf.contrib.rnn.OutputProjectionWrapper(tf.nn.rnn_cell.
 ↪BasicRNNCell(num_units=n_neurons, activation=tf.nn.relu),␣
 ↪output_size=n_outputs)

outputs, states = tf.nn.dynamic_rnn(cell, X, dtype=tf.float32)

loss = tf.sqrt(tf.losses.mean_squared_error(outputs, y)) # RMSE
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

init = tf.global_variables_initializer()
saver = tf.train.Saver()


debug = False
with tf.Session() as sess:
    init.run()
    for iteration in range(n_iterations):
        X_batch, y_batch = next_batch(batch_size, n_steps)
        sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
        if debug or iteration % 100 == 0:
            mse = loss.eval(feed_dict={X: X_batch, y: y_batch})
            print(iteration, "\tRMSE:", mse)
    saver.save(sess, "./my_time_series_model")
```

```
WARNING:tensorflow:From <ipython-input-15-ae985672183c>:16:
```

```
BasicRNNCell.__init__ (from tensorflow.python.ops.rnn_cell_impl) is deprecated
and will be removed in a future version.
Instructions for updating:
This class is equivalent as tf.keras.layers.SimpleRNNCell, and will be replaced
by that in Tensorflow 2.0.
WARNING:tensorflow:From <ipython-input-15-ae985672183c>:18:
MultiRNNCell.__init__ (from tensorflow.python.ops.rnn_cell_impl) is deprecated
and will be removed in a future version.
Instructions for updating:
This class is equivalent as tf.keras.layers.StackedRNNCells, and will be
replaced by that in Tensorflow 2.0.
WARNING:tensorflow:
The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
  * https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-
sunset.md
  * https://github.com/tensorflow/addons
  * https://github.com/tensorflow/io (for I/O related ops)
If you depend on functionality not listed there, please file an issue.

WARNING:tensorflow:From <ipython-input-15-ae985672183c>:22: dynamic_rnn (from
tensorflow.python.ops.rnn) is deprecated and will be removed in a future
version.
Instructions for updating:
Please use `keras.layers.RNN(cell)`, which is equivalent to this API
WARNING:tensorflow:Entity <bound method OutputProjectionWrapper.call of
<tensorflow.contrib.rnn.python.ops.core_rnn_cell.OutputProjectionWrapper object
at 0x000002AB9C2C4448>> could not be transformed and will be executed as-is.
Please report this to the AutgoGraph team. When filing the bug, set the
verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full
output. Cause: converting <bound method OutputProjectionWrapper.call of
<tensorflow.contrib.rnn.python.ops.core_rnn_cell.OutputProjectionWrapper object
at 0x000002AB9C2C4448>>: AssertionError: Bad argument number for Name: 3,
expecting 4
WARNING: Entity <bound method OutputProjectionWrapper.call of
<tensorflow.contrib.rnn.python.ops.core_rnn_cell.OutputProjectionWrapper object
at 0x000002AB9C2C4448>> could not be transformed and will be executed as-is.
Please report this to the AutgoGraph team. When filing the bug, set the
verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full
output. Cause: converting <bound method OutputProjectionWrapper.call of
<tensorflow.contrib.rnn.python.ops.core_rnn_cell.OutputProjectionWrapper object
at 0x000002AB9C2C4448>>: AssertionError: Bad argument number for Name: 3,
expecting 4
WARNING:tensorflow:From C:\Local\anaconda3\envs\MLTech\lib\site-
packages\tensorflow\python\ops\rnn_cell_impl.py:459: calling Zeros.__init__
(from tensorflow.python.ops.init_ops) with dtype is deprecated and will be
removed in a future version.
Instructions for updating:
```

```
Call initializer instance with the dtype argument instead of passing it to the
constructor
WARNING:tensorflow:Entity <bound method BasicRNNCell.call of
<tensorflow.python.ops.rnn_cell_impl.BasicRNNCell object at 0x000002AB9C127548>>
could not be transformed and will be executed as-is. Please report this to the
AutgoGraph team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output. Cause: converting <bound
method BasicRNNCell.call of <tensorflow.python.ops.rnn_cell_impl.BasicRNNCell
object at 0x000002AB9C127548>>: AssertionError: Bad argument number for Name: 3,
expecting 4
WARNING: Entity <bound method BasicRNNCell.call of
<tensorflow.python.ops.rnn_cell_impl.BasicRNNCell object at 0x000002AB9C127548>>
could not be transformed and will be executed as-is. Please report this to the
AutgoGraph team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output. Cause: converting <bound
method BasicRNNCell.call of <tensorflow.python.ops.rnn_cell_impl.BasicRNNCell
object at 0x000002AB9C127548>>: AssertionError: Bad argument number for Name: 3,
expecting 4
WARNING:tensorflow:From C:\Local\anaconda3\envs\MLTech\lib\site-
packages\tensorflow\contrib\rnn\python\ops\core_rnn_cell.py:104: calling
Constant.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated
and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the
constructor
WARNING:tensorflow:From C:\Local\anaconda3\envs\MLTech\lib\site-
packages\tensorflow\python\ops\losses\losses_impl.py:121:
add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
0       RMSE: 0.23250437
100     RMSE: 0.013355941
200     RMSE: 0.009185492
300     RMSE: 0.008139848
400     RMSE: 0.008460406
500     RMSE: 0.008098823
600     RMSE: 0.009119303
700     RMSE: 0.0087545
800     RMSE: 0.008048211
900     RMSE: 0.011366932
```

[16]:
```python
t_instance = np.arange(12, 12+n_steps+1)
pred_all = [0 for i in range(n_steps)]
with tf.Session() as sess:
    saver.restore(sess, "./my_time_series_model")
    #init.run()
```

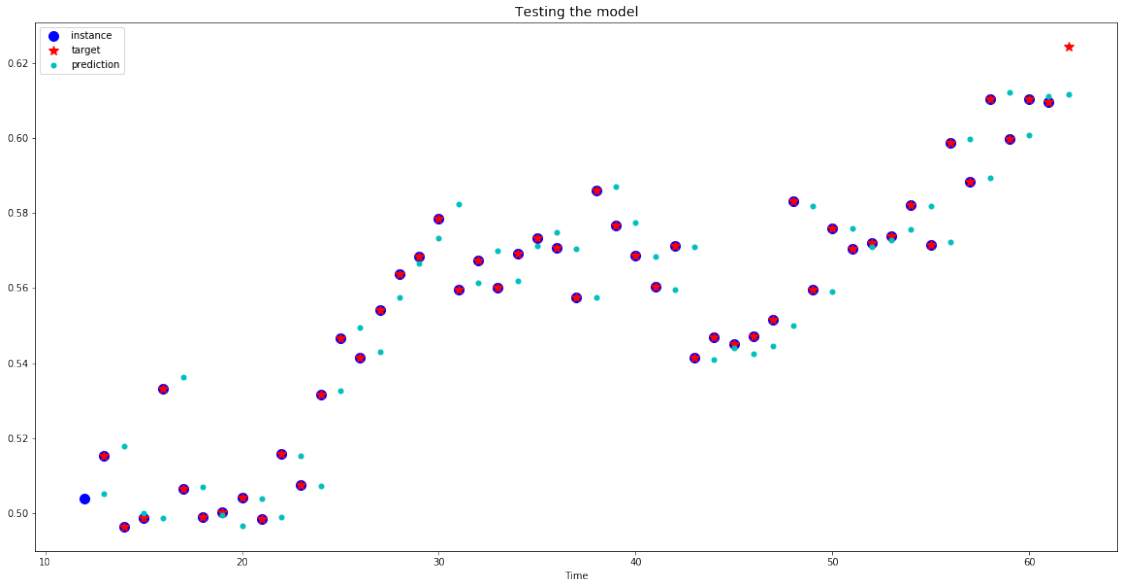
```
    X_new = normalized_data[np.array(t_instance[:-1].reshape(-1, n_steps,␣
 ↪n_inputs))]
    y_pred = sess.run(outputs, feed_dict={X: X_new})
    for i in range(0, len(normalized_data)-n_steps):
        if i % 2000 == 0:
            print(i)
        r = np.arange(i, i + n_steps)
#         print(np.array(r.reshape(-1, n_steps, n_inputs)))
        X_new = normalized_data[np.array(r.reshape(-1, n_steps, n_inputs))]
        pred = sess.run(outputs, feed_dict={X: X_new})
       # print(pred)
        new_inst = pred[0][0][-1]
        pred_all.append(new_inst)
```

```
WARNING:tensorflow:From C:\Local\anaconda3\envs\MLTech\lib\site-
packages\tensorflow\python\training\saver.py:1276: checkpoint_exists (from
tensorflow.python.training.checkpoint_management) is deprecated and will be
removed in a future version.
Instructions for updating:
Use standard file APIs to check for files with this prefix.
INFO:tensorflow:Restoring parameters from ./my_time_series_model
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
```

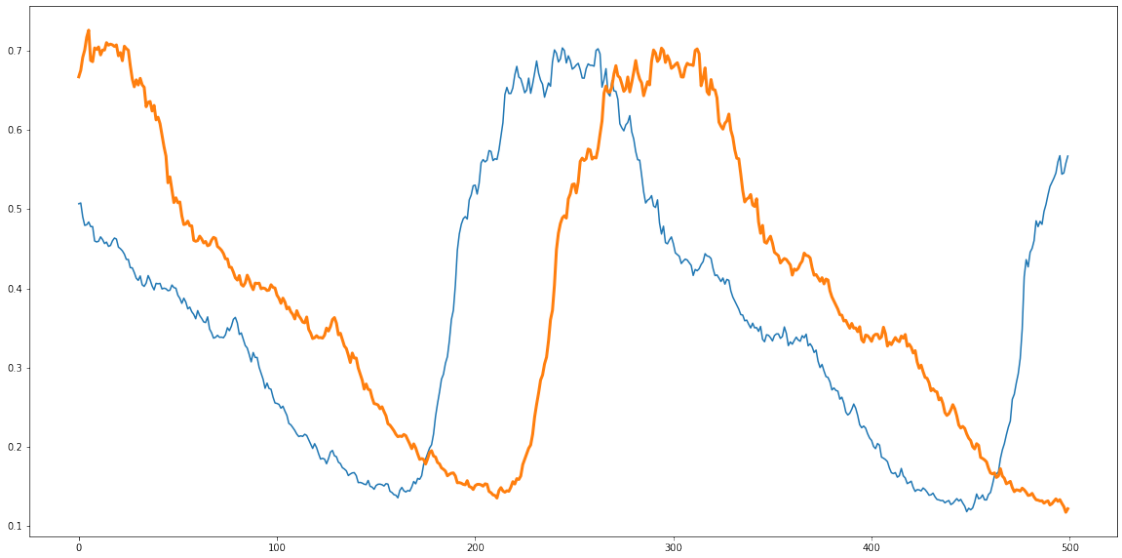
[17]:
```
len(pred_all), len(normalized_data)
```

[17]:
```
(19888, 19888)
```

[18]:
```
plt.figure(figsize=(20, 10))
plt.title("Testing the model", fontsize=14)
plt.plot(t_instance[:-1], normalized_data[t_instance[:-1]], "bo",␣
 ↪markersize=10, label="instance")
plt.plot(t_instance[1:], normalized_data[t_instance[1:]], "r*", markersize=10,␣
 ↪label="target")
plt.plot(t_instance[1:], y_pred[0,:,0], "c.", markersize=10, label="prediction")
plt.legend(loc="upper left")
plt.xlabel("Time")

plt.show()
```



[19]:
```
plt.figure(figsize=(20, 10))
plt.plot(normalized_data[-500:])
plt.plot(pred_all[-500:], linewidth=3)
plt.show()
```



[20]:
```
from sklearn.metrics import mean_squared_error
import math

print("RMSE", mean_squared_error(normalized_data[:(len(pred_all))], pred_all))
```

```
RMSE 0.02794209676046289
```