lab6

December 12, 2018

1 COMP3222/COMP6246 Machine Learning Technologies (2018/19)

1.1 Week 11 - Recurrent Neural Networks (Chapter 14)

Follow each section at your own pace, you can have a look at the book or ask questions to demonstrators if you find something confusing.

2 1. Basic Theory

Until now, we looked into basic preceptrons, convolutional neural network (CNN) and how to implement them in TensorFlow. In practice these techniques are used in tasks such as: searching images, self-driving cars, automatic video classification and many more. Surely, there are different network architectures that are used in Deep Learning. In the previous lab, we showed that CNNs are essentially for `"processing a grid of values"`. However, the Deep Learning community has also generated another architecture specifically for `"processing a sequence of values"`, which are called **Recurrent Neural Networks (RNN)** [Goodfellow 2016]. In practice, recurrent neural networks are used for analyzing time series: stock prices, car trajectories, sentiment analysis and more.

Get Motivated: Have a look at this interactive example, which generates new strokes in your handwriting style using RNNs. The model is explained in this paper.

2.1 Bare-bones RNN

Let's implement an RNN with five recurrent neurons without using TensorFlow's RNN implementation/utilities.

```
In [0]: import tensorflow as tf
        import numpy as np
        import matplotlib.pyplot as plt

        # to make this notebook's output stable across runs
        def reset_graph(seed=42):
            tf.reset_default_graph()
            tf.set_random_seed(seed)
            np.random.seed(seed)

        reset_graph()
```

```
        # Let's assume some artificial data with three input (if our objective is to predict w
        n_inputs = 3  # then for instance: first word, second word, third word can be the inpu
        n_neurons = 5 # number of neurons

        X0 = tf.placeholder(tf.float32, [None, n_inputs]) # t=0 batch
        X1 = tf.placeholder(tf.float32, [None, n_inputs]) # t=1 batch

        # Weights on inputs (all steps share this), initialy they are set random
        Wx = tf.Variable(tf.random_normal(shape=[n_inputs, n_neurons],dtype=tf.float32))

        # Connection weights for the outputs of the previous timestep (all steps share this),
        Wy = tf.Variable(tf.random_normal(shape=[n_neurons,n_neurons],dtype=tf.float32))

        # bias vector, all zeros for now
        b = tf.Variable(tf.zeros([1, n_neurons], dtype=tf.float32))

        # outputs of timestep 0
        Y0 = tf.tanh(tf.matmul(X0, Wx) + b)

        # outputs of timestep 1
        Y1 = tf.tanh(tf.matmul(Y0, Wy) + tf.matmul(X1, Wx) + b)
        # Y1 = activation_function(dot_product(Y0, Wy) + dot_product(X1, Wx) + bias_vector)

        init = tf.global_variables_initializer()

        # Mini-batch:        instance1  instance2   instance3 instance4
        X0_batch = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 0, 1]]) # t = 0 (e.g. instan
        X1_batch = np.array([[9, 8, 7], [0, 0, 0], [6, 5, 4], [3, 2, 1]]) # t = 1 (e.g. instan

        # within the session
        with tf.Session() as sess:
            init.run()
            # get the outputs of each step
            Y0_val, Y1_val = sess.run([Y0, Y1], feed_dict={X0: X0_batch, X1: X1_batch})
```

```
In [0]: print(Y0_val) # layers output at t=0
```

```
[[-0.0664006   0.9625767   0.68105793  0.7091854  -0.898216  ]
 [ 0.9977755  -0.719789   -0.9965761   0.9673924  -0.9998972 ]
 [ 0.99999774 -0.99898803 -0.9999989   0.9967762  -0.9999999 ]
 [ 1.         -1.         -1.         -0.99818915  0.9995087 ]]
```

```
In [0]: print(Y1_val) # layers output at t=1
```

```
[[ 1.         -1.         -1.          0.4020025  -0.9999998 ]
 [-0.12210419  0.62805265  0.9671843  -0.9937122  -0.2583937 ]
 [ 0.9999983  -0.9999994  -0.9999975  -0.85943305 -0.9999881 ]
```

```
 [ 0.99928284 -0.99999815 -0.9999058   0.9857963  -0.92205757]]
```

For the given example above, from the comments in the code:
Exercise 1.2.: How would you define the outputs?
Exercise 1.3.: Why are there five columns?
Exercise 1.4.: Why are there four rows?
Exercise 1.5.: What would be the difference between `instance1` at $t = 0$ and `instance1` at $t = 1$?
Exercise 1.6.: What is the difference between `instance1` and `instance2` at $t = 1$?

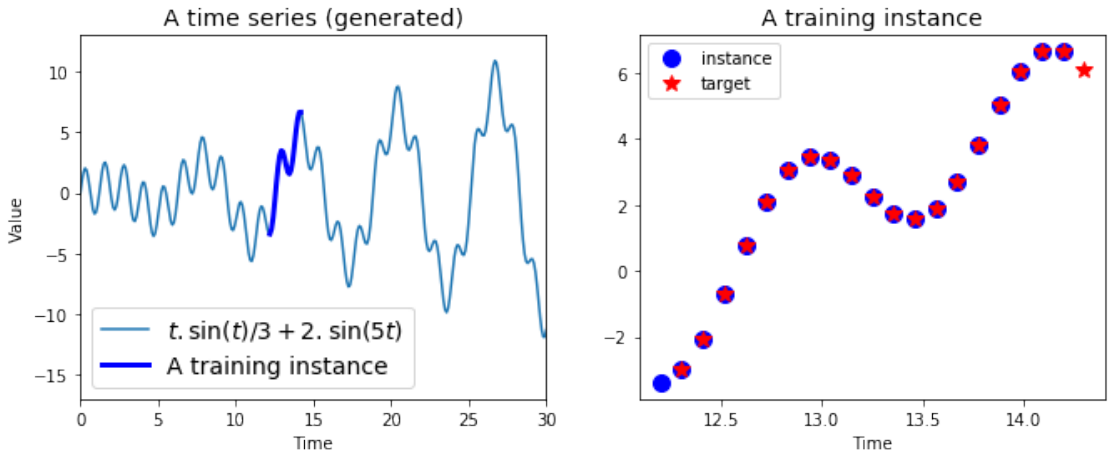
3 2. Predicting Time Series

Let's look at a simple use of RNNs with time series, these time series could be stock prices, brain wave patterns and so on. Our objective could be predicting the future stock price, given the available data that we have.

Let's define an arbitary sine function for stock prices `time_series(t)` to make our predictions.

```
In [0]: reset_graph()
        # time starts from 0 to 30
        t_min, t_max = 0, 30
        # we sample time_series function for every 0.1
        resolution = 0.1

        def time_series(t):
            return t * np.sin(t) / 3 + 2 * np.sin(t*5)

        def next_batch(batch_size, n_steps):
            """
            Returns a batch with `n_steps`: number of instances
            """
            # randomly get a starting number between a range
            t0 = np.random.rand(batch_size, 1) * (t_max - t_min - n_steps * resolution)
            # make a list until of number with n_steps until the next batch
            Ts = t0 + np.arange(0., n_steps + 1) * resolution
            # get the outputs of time_series functio given the input Ts (time points)
            ys = time_series(Ts)

            # return X's and Y's
            return ys[:, :-1].reshape(-1, n_steps, 1), ys[:, 1:].reshape(-1, n_steps, 1)

        # inputs to the time_series function
        t = np.linspace(t_min, t_max, int((t_max - t_min) / resolution))

        n_steps = 20
        # a training instance
        t_instance = np.linspace(12.2, 12.2 + resolution * (n_steps + 1), n_steps + 1)
```

```
plt.figure(figsize=(11,4))
plt.subplot(121)
plt.title("A time series (generated)", fontsize=14)
# plot all the data
plt.plot(t, time_series(t), label=r"$t . \sin(t) / 3 + 2 . \sin(5t)$")

# plot only the training set
plt.plot(t_instance[:-1], time_series(t_instance[:-1]), "b-", linewidth=3, label="A tra
plt.legend(loc="lower left", fontsize=14)
plt.axis([0, 30, -17, 13])
plt.xlabel("Time")
plt.ylabel("Value")

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

plt.show()
```



In [0]:
```
X_batch, y_batch = next_batch(1, n_steps)

# combining X_batch and y_batch for better printing, first_column=X, second_column=Y
print(np.c_[X_batch[0], y_batch[0]])
# Did you notice the shift in y values?
```

```
[[-6.38589336 -7.65675325]
 [-7.65675325 -8.79972348]
```

```
 [-8.79972348 -9.58553732]
 [-9.58553732 -9.85471151]
 [-9.85471151 -9.55610976]
 [-9.55610976 -8.7591591 ]
 [-8.7591591  -7.63673247]
 [-7.63673247 -6.42289149]
 [-6.42289149 -5.35583605]
 [-5.35583605 -4.62002782]
 [-4.62002782 -4.30165474]
 [-4.30165474 -4.3683356 ]
 [-4.3683356  -4.67802692]
 [-4.67802692 -5.01494284]
 [-5.01494284 -5.1436834 ]
 [-5.1436834  -4.86830673]
 [-4.86830673 -4.0818687 ]
 [-4.0818687  -2.79428621]
 [-2.79428621 -1.13168597]
 [-1.13168597  0.69261902]]
```

In [0]:
```
reset_graph()
n_steps = 20
n_inputs = 1
n_neurons = 100
n_outputs = 1
learning_rate = 0.04
n_iterations = 50
batch_size = 50

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

# We use `dynamic_rnn` and `BasicRNNCell` utilities in this case, with tf.nn.relu
cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons, activation=tf.nn.relu)
rnn_outputs, states = tf.nn.dynamic_rnn(cell, X, dtype=tf.float32)

# This part is visually shown in the book Figure 14-10.
stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])

# What do you think line below will be doing? (Tip: https://www.tensorflow.org/api_doc
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)

outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])

loss = tf.reduce_mean(tf.square(outputs - y))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)
```

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

        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 iteration % 100 == 0:
                    mse = loss.eval(feed_dict={X: X_batch, y: y_batch})
                    print(iteration, "\tMSE:", mse)

            X_new = time_series(np.array(t_instance[:-1].reshape(-1, n_steps, n_inputs)))
            y_pred = sess.run(outputs, feed_dict={X: X_new})
            saver.save(sess, "./my_time_series_model")
```

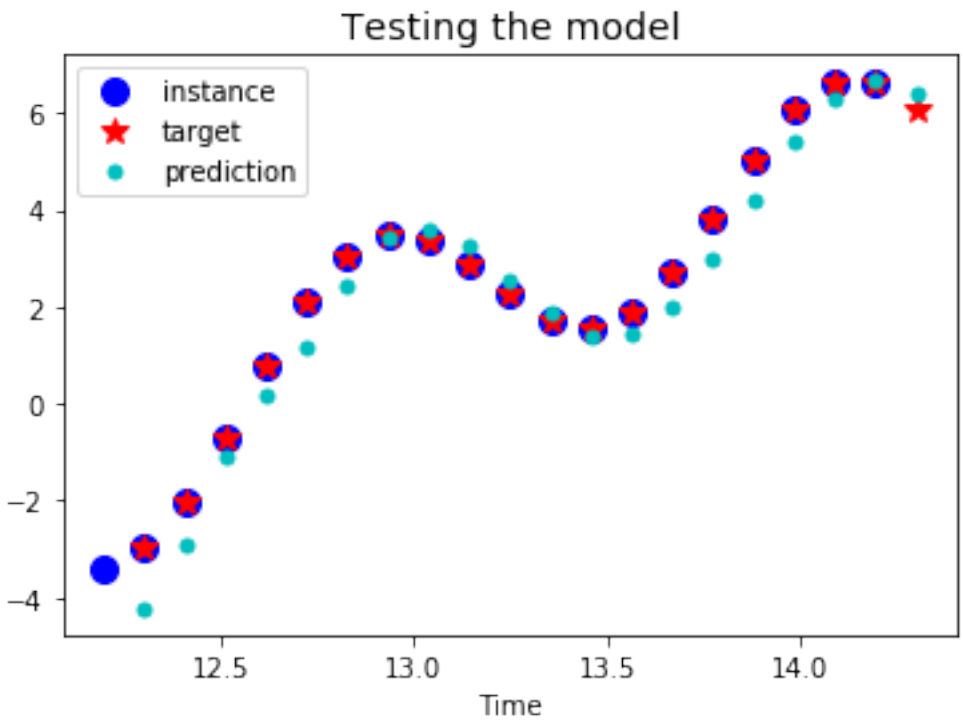
```
0          MSE: 4.325624
```

```
In [0]: y_pred
```

```
Out[0]: array([[[-4.2340837 ],
                [-2.8911543 ],
                [-1.070605  ],
                [ 0.19214775],
                [ 1.1697538 ],
                [ 2.4484897 ],
                [ 3.4318795 ],
                [ 3.5786705 ],
                [ 3.251478  ],
                [ 2.5415256 ],
                [ 1.8611032 ],
                [ 1.411264  ],
                [ 1.4580247 ],
                [ 1.9701571 ],
                [ 2.9672565 ],
                [ 4.206966  ],
                [ 5.4253616 ],
                [ 6.314535  ],
                [ 6.6684027 ],
                [ 6.400406  ]]], dtype=float32)
```

```
In [0]: plt.title("Testing the model", fontsize=14)
        plt.plot(t_instance[:-1], time_series(t_instance[:-1]), "bo", markersize=10, label="ins
        plt.plot(t_instance[1:], time_series(t_instance[1:]), "r*", markersize=10, label="targe
        plt.plot(t_instance[1:], y_pred[0,:,0], "c.", markersize=10, label="prediction")
        plt.legend(loc="upper left")
        plt.xlabel("Time")
```

```
plt.show()
```



Exercise 2.1. Add comments to the code blocks above. Do you understand the purpose of each line?

Exercise 2.2. How can you improve the `MSE`? *(Tip: Remember Lab 4: Gradient Descent)*

Exercise 2.3. Implement the `RMSE` instead of the `MSE`, compare the test plots.

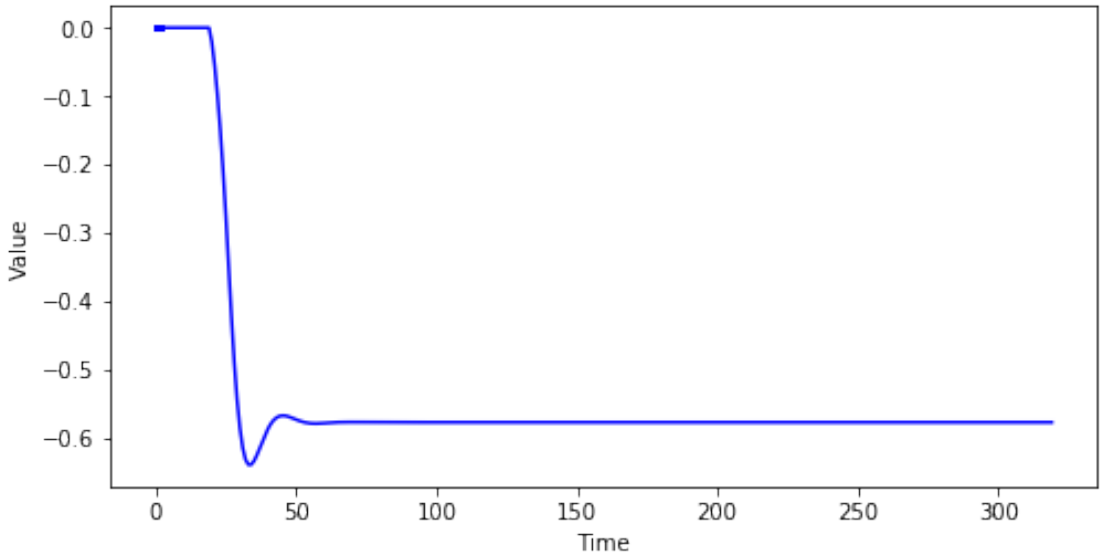
4 3. Generative RNNs

We can use RNNs to generate sequences, below you are going to use the model we trained. You should expect some resemblance to the original time series.

```
In [0]: with tf.Session() as sess:
            saver.restore(sess, "./my_time_series_model")

            sequence = [0.] * n_steps
            for iteration in range(300):
                X_batch = np.array(sequence[-n_steps:]).reshape(1, n_steps, 1)
                y_pred = sess.run(outputs, feed_dict={X: X_batch})
                sequence.append(y_pred[0, -1, 0])

        plt.figure(figsize=(8,4))
        plt.plot(np.arange(len(sequence)), sequence, "b-")
```

```
        plt.plot(t[:n_steps], sequence[:n_steps], "b-", linewidth=3)
        plt.xlabel("Time")
        plt.ylabel("Value")
        plt.show()
```

```
INFO:tensorflow:Restoring parameters from ./my_time_series_model
```



Exercise 3.1. Does your plot resemble the actual time series? Why do you think so?
Exercise 3.2. Change your optimizer to `AdamOptimizer`, what do you think has changed?
Exercise 3.3. Try different activation functions. (e.g. logit, tanh, ...)

5 Recap

In this lab, we demonstrated these concepts:

- from theory to implementation, how a simple RNN is works
- how to predict a time series with RNN
- which parameters to look out for in order to improve the predictions
- generation of sequences with a RNN

As in the previous labs, there is some material that we have not been able to cover. In your free time, you can have a look at:

- LTSM Cells and GRU Cells
- NLP Applications with RNNs
- Encoding and Decoding with RNNs

5.0.1 References

[Goodfellow, 2016] : https://www.deeplearningbook.org/

In [0]: