Artificial Neural Networks

Prof. Dr. Sen Cheng

Dec 16, 2019

Problem Set 12: Recurrent Neural Networks

Tutors: Filippos Panagiotou (filippos.panagiotou@rub.de), Sandhiya Vijayabaskaran(sandhiya.vijayabaskaran@rub.de)

1. **Predicting time series data.** Use Keras' *SimpleRNN* layer to predict data points of a ramping sine wave.

$$f(x) = sin(x) + 0.1x \tag{1}$$

   (a) In Keras, recurrent neural networks require a specific input shape (batch size, time steps, input length). Write a function *prepareData* that takes the number of time steps as a parameter, and prepares inputs x and labels $y = f(x)$ with appropriate shapes. Generate a dataset of size $N = 1000$ (with x linearly spaced in the range $[0, 100]$).

   (b) Build a sequential model with a hidden *SimpleRNN* layer (32 units, ReLU activation) and a fully connected output layer (1 unit, linear activation). Use mean squared error (MSE) as your loss function and Adam as your optimizer.

   (c) Prepare your data using one time step as input x. Split the data set $70/30$ into training and test sets. Train the model for 20 epochs. Plot the prediction for training and test sets.

   (d) Calculate and plot the squared errors for each data point. How does the squared error depend on the phase of the sine wave? Why does it show this behavior? Are there MSE differences between training and test set?

   (e) Repeat the previous two tasks using three time steps instead. Does the MSE differ? How does the squared error depend on the phase of the sine wave?

2. **Learning long-term dependencies.** In this exercise, we will compare a simple RNN to LSTMs in a simple text generation task. Train a network on the text file containing Goethe's *Faust* to predict the *next character* given a sequence of characters, i.e., a character-level RNN. The model is only predicting single characters and does not have any knowledge that words are units of language, or that words are combined into sentences.

   (a) Load the text file in lowercase and print out the length of the whole text. To get an idea of the dataset, also print the number of individual characters present in the text after sorting them in order (use the default python functions `sorted` and `set`).

   (b) To prepare the data, create two dictionaries that map characters to integers and vice-versa. Next, prepare the training data: split the data into sequences of length 50 that start at every third character of the text. The labels are the next character after each sequence.

   (c) To encode the characters use one-hot encoding. Use the dictionaries from the previous step to generate the one-hot encodings of the labels and also of the input sequences. That is, each input sequence is a sequence of the one-hot encodings of the individual characters in that sequence.

   (d) Build a Sequential model using an LSTM layer with 256 units as the hidden layer followed by an ouput Dense layer. How many units should the ouput layer have? What are the activation function and loss function that you would use, and why? Use Adam as the optimizer again and compile.

(e) Callbacks are a useful function in Keras that show relevant aspects of the model during training. The file `print_callback.py` provides a keras callback that prints the network output for a random sample after every epoch. Pass this callback to the fit function to see how the model does after every epoch. Train the model for 20 epochs. Save the weights after training. (Note : There is no train/test split. We train the model on the entire dataset to learn a probability distribution of characters in a sequence.)

(f) Test the model using a random input sample and look at the next 500 characters generated (approximately a paragraph). Look at the callback function provided for hints on how to do this. Analyze the features of the generated text.

(g) Repeat the training and test after replacing the LSTM layer with a SimpleRNN of 256 units. What are the qualitative differences you notice in the text generated by the two models?

Solutions

1. **Predicting time series data.**

   (a) Write prepareData and generate a dataset

```
# basic imports
import numpy as np
import matplotlib.pyplot as plt
# keras imports
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN


def prepareData(data, input_steps=1, output=1):
    '''
    This function prepares inputs and labels from raw data.

    | **Args**
    | data:          Raw data.
    | input_steps:   The number of steps to be used as input.
    | output:        The number of steps to be used as label.
    '''
    # prepare inputs
    X = []
    for example in range(data.shape[0] - input_steps):
        X += [np.reshape(data[example:example + input_steps], (input_steps, 1))]
    # prepare labels
    y = []
    for example in range(input_steps, data.shape[0] - output + 1):
        y += [data[example:example + output]]

    return np.array(X), np.array(y)

# define data size
N = 1000 # data set size
start = 0  # linspace start
end = 100  # linspace end

# generate raw data
X = np.linspace(start, end, N)
sine = np.sin(X) + X*0.1
```

   (b) Build a sequential model

```
# prepare model
model = Sequential()
model.add(SimpleRNN(units=32, input_shape=(input_steps, 1), activation="relu"))
model.add(Dense(output))
model.compile(loss='mean_squared_error', optimizer='adam')
```

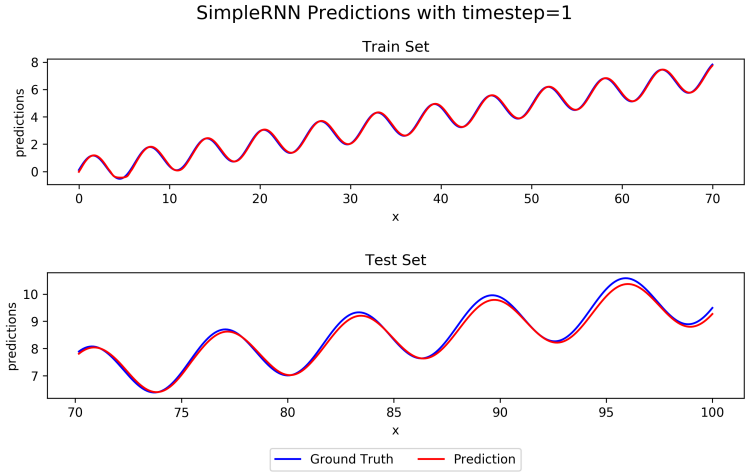
   (c) Prepare the data with one timestep.

```
# prepare data
X, y = prepareData(sine, input_steps=1, output=output)
```

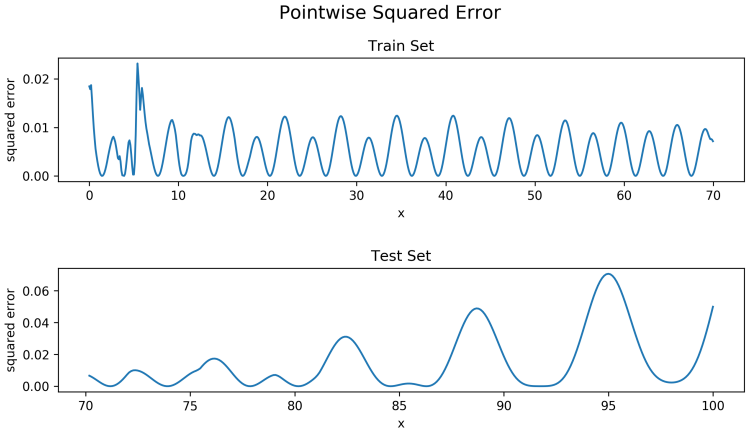
```
4 # split data
5 X_train, X_test, y_train, y_test = X[:N_split], X[N_split:], y[:N_split], y[N_split:]

7 # fit model
8 model.fit(X_train, y_train, epochs=20, batch_size=16, verbose=2)
```

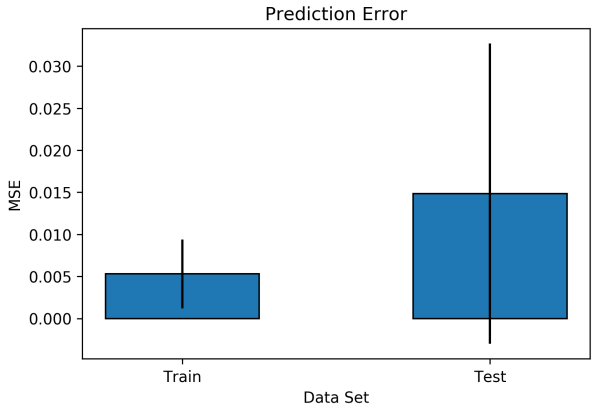
The RNN is reasonably effective in predicting the next values of the sinusoid.



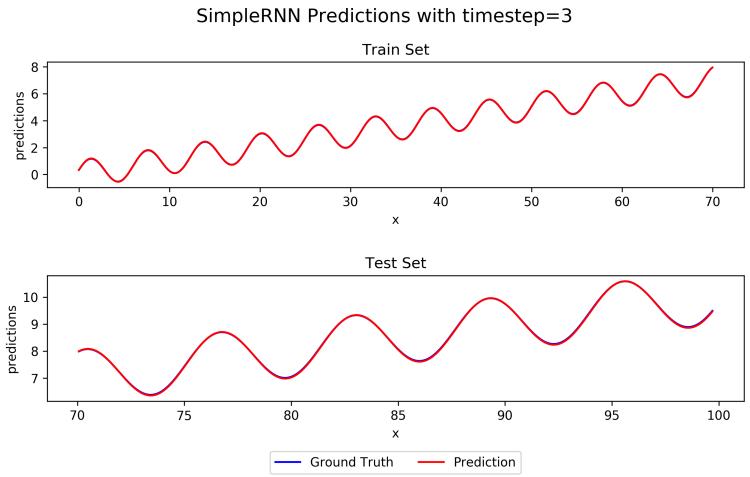
(d) Two things jump out in the pointwise squared errors of the model. Firstly, the error peaks right after the derivative of the function changes sign. Secondly, when the derivative changes from negative to positive, the error is larger that when the derivative changes from positive to negative. The reason for the latter is that the RNN has to combat both the change in the derivative of the sine component as well as the overall upwards trend.



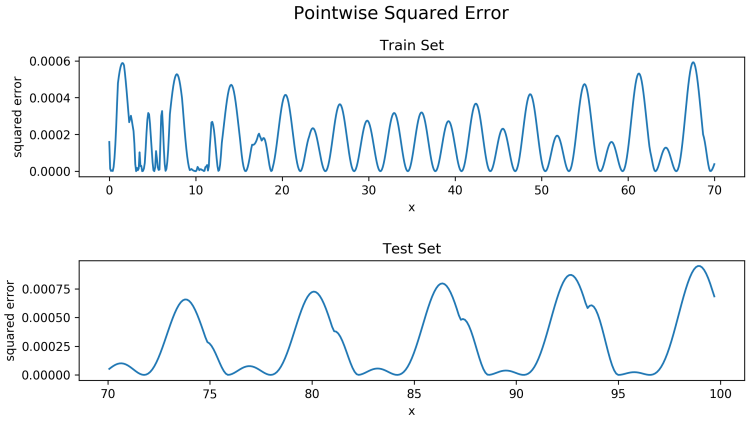
The MSE increases for the test set, as the RNN does not keep up with the general upwards trend.



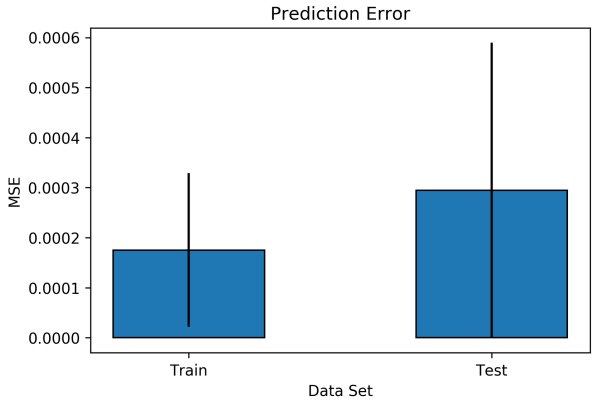
(e) On a first reading increasing the input length has produced better performance for the RNN. The network's generalization ability does not seem to deteriorate over time as much.



While the error is still dependent on the sign change of the derivative of the sinusoid, the squared errors are much smaller and don't increase overall.



The MSE is lower in general, and the test error is much closer to training error.



2. **Learning long-term dependencies.**

(a) Loading the text and looking at basic characteristics :

Text length : 523485

Number of unique characters in the text : 64

```
import io
path = 'faust.txt'
with io.open(path, encoding='utf-8') as f:
    text = f.read().lower()

print('Text length:', len(text))
chars = sorted(list(set(text)))
print('Number of unique characters in text:', len(chars))
```

(b) Create mapping dictionaries and split the data into sequences as follows :

```
char2int = dict((c, i) for i, c in enumerate(chars))
int2char = dict((i, c) for i, c in enumerate(chars))

len_seq = 50
step = 3
sequences = []
targets = []
for i in range(0, len(text) - len_seq, step):
    sequences.append(text[i: i + len_seq])
    targets.append(text[i + len_seq])
```

(c) Compute the one-hot encoding using the dictionaries created

```
x = np.zeros((len(sequences), len_seq, len(chars)))
y = np.zeros((len(sequences), len(chars)))

#compute one-hot encoding
for i, sentence in enumerate(sequences):
    for t, char in enumerate(sentence):
        x[i, t, char2int[char]] = 1
    y[i, char2int[targets[i]]] = 1
```

(d) The output layer should have as many units as the number of unique characters in our text, in this case, 64. Since this is a classification problem, the cross-entropy loss in combination with softmax activation in the output layer should be used.

```
model = Sequential()
model.add(LSTM(256, input_shape=(len_seq, len(chars))))
model.add(Dense(len(chars), activation='softmax'))

optimizer = Adam(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=optimizer)
```

(e) Call `model.fit` to train the network. The weights can be saved using the keras function `model.save_weights`.

(f) After only 20 epochs, the LSTM output with a random seed looks like this :

```
ließ' es dem großen herren gut,
das arme rung,
und wie der geistern auf.
```

```
mephistopheles:
die pracht geschmückt' er soll erstaaben.
und mit der schlagen, wo sich der schlassen
die gewalt sich welt und gar zu regen;
und, die meiner weise schweigen schließen
und trete doch diese schaffen schallen,
in der feind, geleben auf.
das glauben frei, was ihr gedrängen,
wenn ich mich heran.

helena:
wie weiß die schärfe nacht die schlagen,
ich habe mich mit der menser gehn
die fester glücknen stürke,
```

Although the model is still far from producing meaningful text, we can see that the LSTM has learned many words as well as the structure of the text very well. It imitates the style of the text we provided it.

(g) The output of a simple RNN after 20 epochs still looks like this :

```
die früh sich einst dem trüben blick gest sterst stest
stest stis st stis sie sstest slust!
serst srürs sterstst ut ust irst sest sten isteitst set
 sterst ssestirst st stestist siestst slistest.
dat st st st werstst sis stest stes sisst sist stst srerst
sterstest srerst ist tist st esst.
sest st st sustst diest sist st,
sist sers st steststst st sistüst stist sist sterst.
stist dist sis ststest,
sst.
stersest stes st stest stist sut stes sterst sstst
ststestersterst stestest ssts sisest.
ser striesterst sus stest sttest
```

While the LSTM generates *Faust*-like text, the RNN gets stuck in predictive loops and does not produce any meaningful words. This is because of the inability of the RNN to learn long-range dependencies.