Neuronale Netze - Übung 9

Tobias Hahn
3073375

January 17, 2017

1 LSTM

1.1 Aufgabenstellung

Die Aufgabenstellung war es, ein LSTM zu konstruieren welches 10 Eingaben nach einer 1 diese wieder ausgibt, wenn dazwischen nur 0en waren. So würde z.B. folgende Eingabe die Ausgabe 1 hervorrufen: 10000000000. Immer, wenn 10 0en auf eine 1 folgen wird also eine 1 ausgegeben, ansonsten 0.

Um diesen Effekt mit einer LSTM zu erreichen verwenden wir eigene Aktivierungsfunktionen und folgende Features: c, i und f. Die Aktivierungsfunktionen für die Gates werden jeweils mit angegeben.

$$
\begin{aligned}
f^t &= 1 - x^t \\
i^t &= 1 - x^t \\
y^t &= \begin{cases} 1 \text{ für } c^t = 10 \\ 0 \text{ sonst} \end{cases} \\
c^t &= i^t + f^t * c^{t-1}
\end{aligned}
$$

Dieser Automat addiert ständig 1 zum Status hinzu, solange der Input 0 ist, wenn der Input einmal 1 ist wird der Status vergessen und der Counter somit neu gestartet. Um sicherzugehen dass keine 1 ausgegeben wird obwohl noch keine 1 kam muss der Counter initial auf -unendlich gesetzt werden.

1.2 Rekurränte McCulloch Pitts Zellen

Ja, diese Aufgabe kann auch mit diesen Zellen erledigt werden. Dabei können wir einen FSM erstellen der 10 Zustände und einen Startzustand hat. Vom Startzustand kommt man mit 1 auf den ersten Zustand, mit 0 wieder auf den Startzustand. Von da an kommt man von Zustand 1 bis 9 jeweils mit 0 zum weitern Zustand, mit 1 wieder zurück zu Zustand 1. Bei Zustand 10 kommt man mit 1 zurück zu Zustand 1 und mit 0 zurück zum Startzustand. Es wird in jedem Zustand 0 ausgegeben außer im Zustand 10, wo 1 ausgegeben wird.

Diesen FSM können wir mit der in der Vorlesung vorgestellten Methode in ein Netz aus McCulloch Pits Zellen verwandeln.

2 LSTM Train

Das LSTM wurde mit Keras trainiert, welches intern Tensorflow benutzt. Dafür wurde Random-Data erstellt und dann trainiert.

2.1 Code

2.1.1 Data Generation

"../data/gen_data.py"

```
1  import random
2
3  class Counter():
4      def __init__(self):
5          self.state = -1
6
7      def count(self, i):
8          if (i == 1):
9              self.state = 0
10         elif (i == 0 and self.state != -1):
11             self.state += 1
12         if (self.state == 10):
13             self.state = -1
14             return 1
```

```
15          return 0
16
17  if __name__ == "__main__":
18      buffer_in = []
19      buffer_out = []
20      input_file = open("./in.dat", "w")
21      output_file = open("./out.dat", "w")
22      input_file.truncate()
23      output_file.truncate()
24      count = Counter()
25
26      for i in range(0,1000000):
27          rand = int(bool(random.getrandbits(1)))
28          buffer_in.append(rand)
29          buffer_out.append(count.count(rand))
30          if (len(buffer_in) == 10000):
31              input_file.write("".join([str(val) for val in buffer_in]))
32              output_file.write("".join([str(val) for val in buffer_out]))
33              buffer_in = []
34              buffer_out = []
35
36      output_file.close()
37      input_file.close()
```

2.1.2 Training

../code/lstm.py

```
1  import numpy as np
2  from keras.models import Sequential
3  from keras.layers import LSTM
4  from keras.layers import Dense
5
6  if __name__ == "__main__":
7      train_in_file = open("../data/in.trn", "r")
8      train_in_string = np.fromstring(train_in_file.read(), "int8") - 48
9      train_in_file.close()
10
11     train_out_file = open("../data/out.trn", "r")
12     train_out_string = np.fromstring(train_out_file.read(), "int8") - 48
13     train_out_file.close()
14
15     dim = len(train_in_string) - 11
16     train_in = np.zeros((dim, 11, 1))
17     train_out = np.zeros((dim, 1))
18
19     for i in range(0, len(train_in_string)-11):
20         train_in[i] = np.reshape(train_in_string[i:i+11], (11,1))
21         train_out[i] = train_out_string[i+11]
22
23     model = Sequential()
24     model.add(LSTM(1, input_shape=(11, 1)))
25     model.add(Dense(1))
26     model.compile(loss="mean_squared_error", optimizer="adam")
27     model.fit(train_in, train_out, nb_epoch=1, batch_size=1, verbose=2)
28
29     # serialize model to JSON
30     model_json = model.to_json()
31     with open("model.json", "w") as json_file:
32             json_file.write(model_json)
33             # serialize weights to HDF5
34             model.save_weights("model.h5")
35             print("Saved model to disk")
```

2.2 Results

2.2.1 Console Output

```
1  Using TensorFlow backend.
```

```
Epoch 1/1
7940s - loss: 5.0080e-04
Saved model to disk
```

2.2.2 Saved Model

../code/model.json

```
{"class_name": "Sequential", "config": [{"class_name": "LSTM", "config": {"name": "lstm_1",
    "trainable": true, "batch_input_shape": [null, 11, 1], "input_dtype": "float32", "
    return_sequences": false, "go_backwards": false, "stateful": false, "unroll": false, "
    consume_less": "cpu", "input_dim": 1, "input_length": null, "output_dim": 1, "init": "
    glorot_uniform", "inner_init": "orthogonal", "forget_bias_init": "one", "activation": "
    tanh", "inner_activation": "hard_sigmoid", "W_regularizer": null, "U_regularizer": null,
     "b_regularizer": null, "dropout_W": 0.0, "dropout_U": 0.0}}, {"class_name": "Dense", "
    config": {"name": "dense_1", "trainable": true, "output_dim": 1, "init": "glorot_uniform
    ", "activation": "linear", "W_regularizer": null, "b_regularizer": null, "
    activity_regularizer": null, "W_constraint": null, "b_constraint": null, "bias": true, "
    input_dim": 1}}], "keras_version": "1.2.0"}
```

2.2.3 Prediciton and error evalutation

../code/test.py

```
import numpy as np
from keras.models import load_model
from keras.models import model_from_json

if __name__ == "__main__":
    test_in_file = open("../data/in.tst", "r")
    test_in_string = np.fromstring(test_in_file.read(), "int8") - 48
    test_in_file.close()

    test_out_file = open("../data/out.tst", "r")
    test_out_string = np.fromstring(test_out_file.read(), "int8") - 48
    test_out_file.close()

    dim = len(test_in_string) - 11
    test_in = np.zeros((dim, 11, 1))
    test_out = np.zeros((dim, 1))

    for i in range(0, len(test_in_string)-11):
        test_in[i] = np.reshape(test_in_string[i:i+11], (11,1))
        test_out[i] = test_out_string[i+11]

    model = model_from_json(open("model.json", "r").read())
    model.load_weights("model.h5")
    predictions = model.predict(test_in)

    predictions_file = open("predictions.out", "w");
    predictions_file.write("".join([str(predictions[x][0]) + "," for x in range(0, len(
        predictions))]))
    predictions_file.close()
```

../code/error.py

```
if __name__ == "__main__":
    true_out_file = open("../data/out.tst", "r")
    true_out_string = true_out_file.read()
    true_out_file.close()
    true_out = [float(x) for x in list(true_out_string)[11:]]

    pred_out_file = open("predictions.out", "r")
    pred_out_string = pred_out_file.read()[:-1]
    pred_out_file.close()
    pred_out = [float(x) for x in pred_out_string.split(',')]

    num = len(true_out)
    error = 0

    for x,y in zip(pred_out, true_out):
```

```
16            error += (x - y)**2
17        print("Mean squared error: {0}".format(error/num))
```

Console output Mean squared error: 0.00045656099664777614