Exercise - DL Tutorial 5

Please complete the following notebook and submit it by the 25th Nov 23:59 to manuel.milling@informatik.uni-augsburg.de

student name:

In [22]:
```
# Equation numbers refer to handout 5

import numpy as np

np.random.seed(42)
```

In [23]:
```
def sigmoid(X):
    return 1/(1 +np.exp(-X))

def del_sigmoid(h):
    return h * (1 - h)
```

Implement a method that creates binary addition data.

In [24]:
```
def generate_data(num_examples, max_len):
    # generate num_examples * 2 ints
    rand_numbers = np.random.randint(0, 2**(max_len-1)-1, size=(num_examples * 2),
dtype=np.uint8)
    rand_numbers_bits = np.unpackbits(rand_numbers)
    rand_numbers = rand_numbers.reshape(num_examples, 2)
    rand_numbers_bits = rand_numbers_bits.reshape(num_examples, 2, max_len)
    rand_results = np.sum(rand_numbers, axis=1, dtype=np.uint8)
    # add 3rd dimension to tensor
    rand_results_bits = np.unpackbits(rand_results).reshape(num_examples, max_len,
1)
    # data should be of form (num_examples, sequence_length, num_features)
    rand_numbers_bits = np.transpose(rand_numbers_bits, axes=[0,2,1])

    return rand_numbers_bits, rand_results_bits
```

Implement the mean squared error as a loss function

In [25]:
```
def mean_square_error(pred, y):
    return np.mean((pred-y)**2)
```

Iimplement the accuracy of the predictions

In [26]:
```
def accuracy(pred, y):
    rounded = np.rint(pred)
    return np.mean(rounded==y)
```

Implement the RNN class, implement the forward propagation, implement the BPTT and implement the gradient step

In [27]:
```
class one_layer_rnn:
    def __init__(self, n_input, n_hidden, n_out):
        #initialisation of weights, no bias
        self.W_1 = np.random.randn(n_input, n_hidden)
        self.U = np.random.randn(n_hidden, n_hidden)
        self.W_2 = np.random.randn(n_hidden, n_out)
        self.X = None
        self.H = None
        self.out = None
        self.dW1 = None
        self.db_h1 = None
        self.dU = None
        self.dW2 = None
        self.db_out = None

    def forward_propagation(self, X):
        self.X = X
        # "dot" multiplication of X and W_1 is performed over the last dimension of
X (the features for one sequence
        # and one training example) and W_1. Result: H without any horizontal infor
mation flow.
        # (2)
        self.H = np.dot(self.X, self.W_1)
        prev = np.zeros(self.H[:, 0, :].shape)
        # loop over sequence
        # numbers have to be added from right to left
        for i in range(self.X.shape[1]-1, -1, -1):
            # matrix multiplication of ith element of sequence
            # (2)
            self.H[:, i, :] = sigmoid(self.H[:, i, :] + np.dot(prev, self.U))
            prev = self.H[:, i, :]
        # (3)
        self.out = sigmoid(np.dot(self.H, self.W_2))
        return self.out

    def backprop_through_time(self, Y):
        # derivative of mean-square error
        # dimension of delta for matrix multiplication: num_sequence, num_feature
(1), num_examples
        # (6)
        self.d_out = np.transpose(2*(self.out - Y) * del_sigmoid(self.out), [1,2,
0])
        #print(self.d_out.shape)
        self.dW2 = np.zeros(self.W_2.shape)
        # backprop: left to right.
        for i in range(self.X.shape[1]):
            # sum up contribution of all sequence results to dW2.
            # basically sum over (5), like in (14)
            self.dW2 += np.transpose(np.matmul(self.d_out[i,:,:], self.H[:,i,:]))
        # (13): W^n \delta^{n,\tau} part, vertical backprop
        # sum of dot multiplication is second-to last index of d_out
        # --> transpose to get num_features back to second index
        self.d_hidden = np.transpose(np.dot(self.W_2, self.d_out), [1,0,2])
        #print(self.d_hidden.shape)
        prev = np.zeros(self.d_hidden[0, :, :].shape)
        #backprop: left to right
        for i in range(self.X.shape[1]):
            # (13): U^{n-1} \delta^{n-1,\tau + 1} part, horizontal backprop
            self.d_hidden[i,:,:] += np.matmul(self.U, prev)
            self.d_hidden[i, :, :] *= np.transpose(del_sigmoid(self.H[:,i,:]))
            prev = self.d_hidden[i, :, :]
        # average gradient for every sequence element
        self.dW2 /= self.X.shape[0]
        self.dW1 = np.zeros(self.W_1.shape)
```

Implement the learning routine

In [28]:
```
learning_rate = 0.1
train_iters = 3000
print_iters = 100

trainX, trainY = generate_data(100, 8)
testX, testY = generate_data(10, 8)
net = one_layer_rnn(2, 16, 1)
for i in range(train_iters):
    if i% print_iters == 0:
        result = net.forward_propagation(testX)
        print("Test loss: \t{}".format(mean_square_error(result, testY)))
        print("Test acc: \t{}".format(accuracy(result, testY)))
        result = net.forward_propagation(trainX)
        print("Train loss: \t{}".format(mean_square_error(result, trainY)))
        print("Train acc: \t{}".format(accuracy(result, trainY)))
    result = net.forward_propagation(trainX)
    net.backprop_through_time(trainY)
    net.gradient_step(learning_rate)
```

```
Test loss:      0.5365197757564597
Test acc:       0.425
Train loss:     0.4856609082054297
Train acc:      0.48625
Test loss:      0.25153028086883544
Test acc:       0.5
Train loss:     0.2481795358491336
Train acc:      0.555
Test loss:      0.2455683905001258
Test acc:       0.475
Train loss:     0.2443045985320066
Train acc:      0.52375
Test loss:      0.24100159227377316
Test acc:       0.55
Train loss:     0.24190537041769664
Train acc:      0.5575
Test loss:      0.23692596905867064
Test acc:       0.5625
Train loss:     0.23915349581434533
Train acc:      0.57375
Test loss:      0.2323586025516562
Test acc:       0.6
Train loss:     0.23524240649455563
Train acc:      0.57375
Test loss:      0.22666308229285786
Test acc:       0.6
Train loss:     0.2293833258167404
Train acc:      0.59125
Test loss:      0.21952703145298597
Test acc:       0.6
Train loss:     0.2211532472825828
Train acc:      0.60625
Test loss:      0.21082664631622708
Test acc:       0.6625
Train loss:     0.2105662609641598
Train acc:      0.64125
Test loss:      0.19978939666508397
Test acc:       0.725
Train loss:     0.19728047678793792
Train acc:      0.71375
Test loss:      0.18421501848956834
Test acc:       0.75
Train loss:     0.18050648500512267
Train acc:      0.775
Test loss:      0.162216799167218
Test acc:       0.775
Train loss:     0.15982103139759585
Train acc:      0.83625
Test loss:      0.13656765530622889
Test acc:       0.8375
Train loss:     0.13667576629601857
Train acc:      0.86875
Test loss:      0.11175745154310199
Test acc:       0.95
Train loss:     0.11352084775104755
Train acc:      0.92625
Test loss:      0.08939666503406743
Test acc:       0.9625
Train loss:     0.09176345909991727
Train acc:      0.95875
Test loss:      0.06927975117187826
Test acc:       0.975
Train loss:     0.07191436417612207
Train acc:      0.985
```