Aalto University
School of Science
and Technology

Classifying MNIST Handwritings with Vanilla Recurrent Neural Network

Group 3: Antti Keurulainen, Julius Wang, Mika Juuti, Darshan Mallenahalli Shankara Lingappa

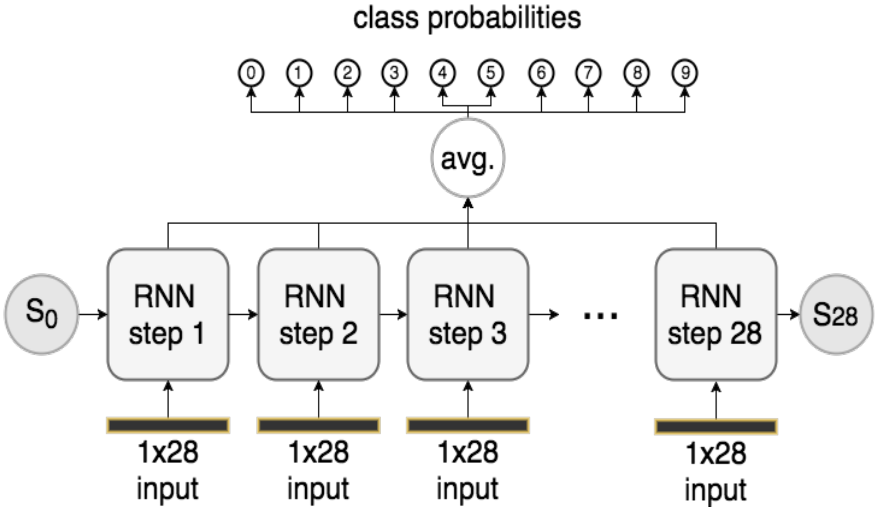
Department of Computer Science
Aalto University, School of Science and Technology

October 20, 2016

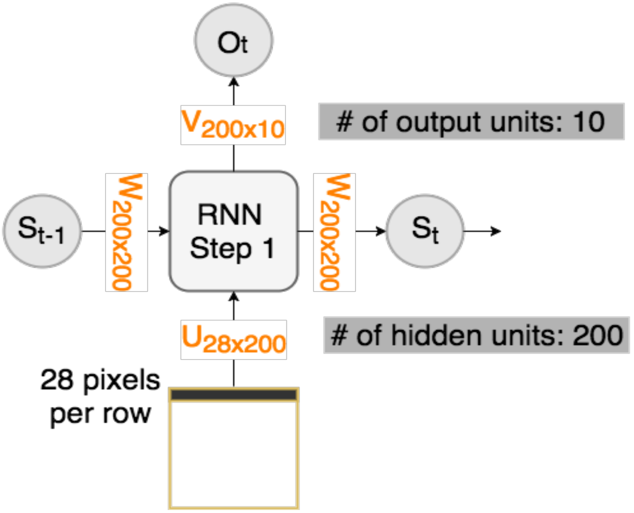
References

- Our literature presentation: `https://mycourses.aalto.fi/mod/folder/view.php?id=157611`
- Our implementation is available at: `https://github.com/juliuswang0728/rnn_theano`
- IMPLEMENTING A RNN WITH PYTHON, NUMPY AND THEANO: `http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-2-implementing-`
- Other RNN implementation with Theano: `https://github.com/gwtaylor/theano-rnn`

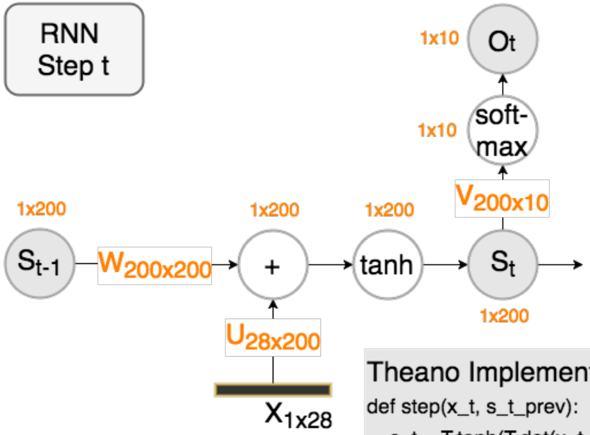
RNN: Feedforward, Single Image



RNN: Feedforward, Single Image (cont.)



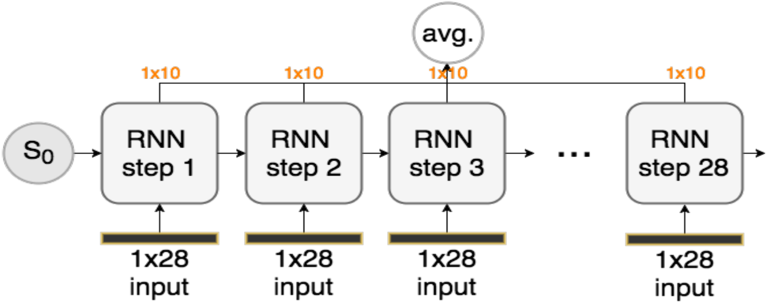
RNN: Feedforward, Single Image (cont.)



Theano Implementation

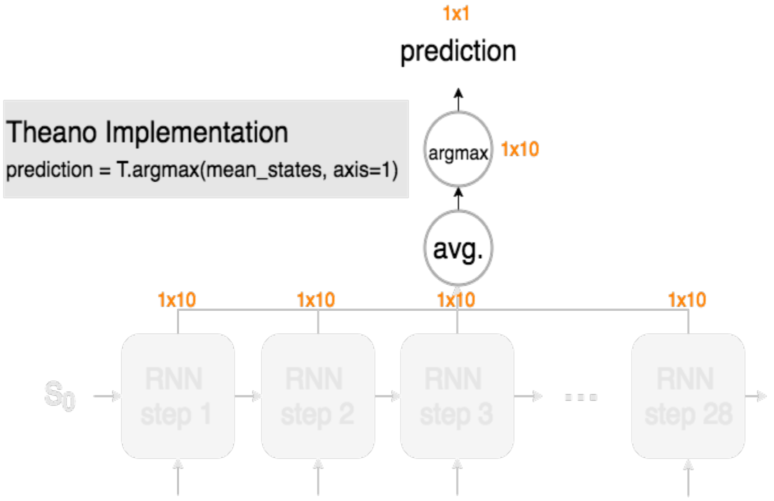
```
def step(x_t, s_t_prev):
    s_t = T.tanh(T.dot(x_t, U) + T.dot(s_t_prev, W))
    o_t = T.nnet.softmax(T.dot(s_t, V))
return s_t, o_t
```

RNN: Feedforward, Single Image (cont.)



```
x = T.ftensor3('input') # (# of image, # of rows, # of columns) = (1, 28, 28)
sum_states = T.zeros((x.shape[0], self.output_dim), dtype='f')
states = T.zeros((x.shape[0], self.hidden_dim), dtype='f')
for step_idx in range(0, 28):
    o_t, states = step(x[:, step_idx, :], states)
    sum_states += o_t
mean_states = sum_states / 28
```

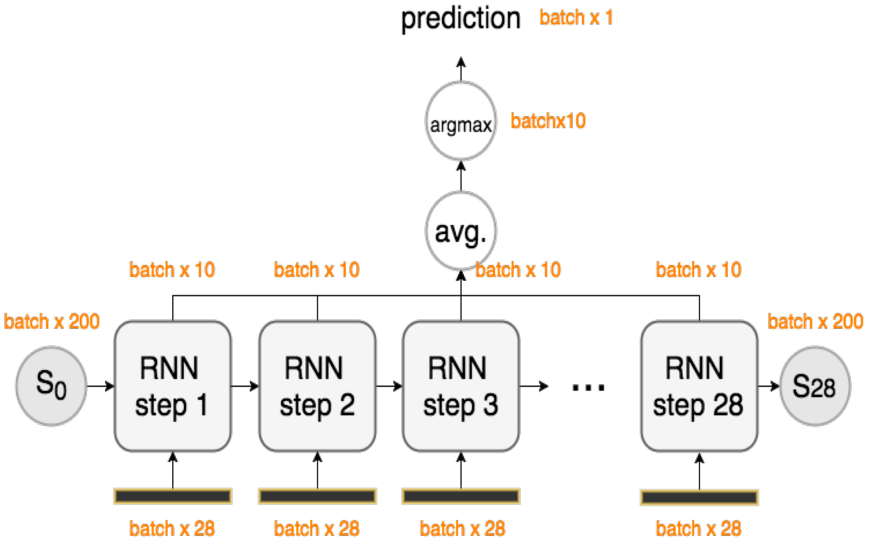
RNN: Feedforward, Single Image (cont.)



Theano Implementation

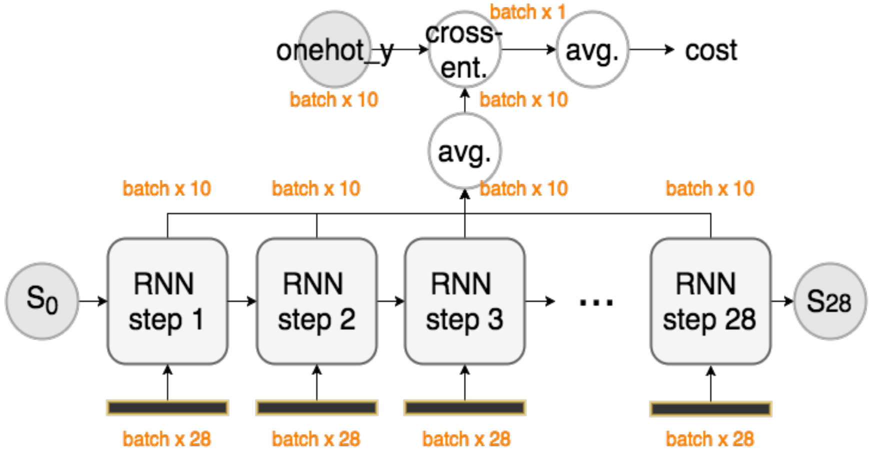
```
prediction = T.argmax(mean_states, axis=1)
```

RNN: Feedforward, Mini-Batch



RNN: Computing Cost, Mini-Batch

```
cost = T.mean(T.nnet.categorical_crossentropy(mean_states, onehot_y))
```



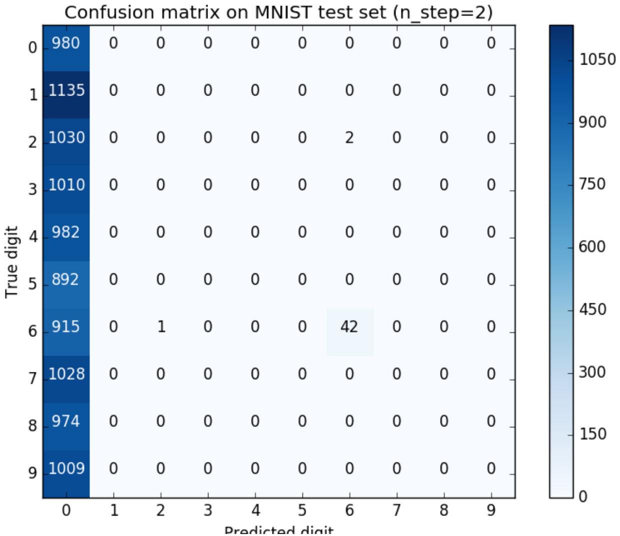
RNN: Computing Gradient and Updates, Mini-Batch

```
dU = T.grad(cost, U)
dV = T.grad(cost, V)
dW = T.grad(cost, W)
learning_rate = T.scalar('learning_rate', dtype='float32')

sgd_step = theano.function([x, onehot_y, learning_rate], [],
                           updates = [(U, U - learning_rate * dU),
                                      (V, V - learning_rate * dV),
                                      (W, W - learning_rate * dW)])


for each minibatch of x and y:
    sgd_step(x, y, 0.01)
```

Some Reminders on Implementation

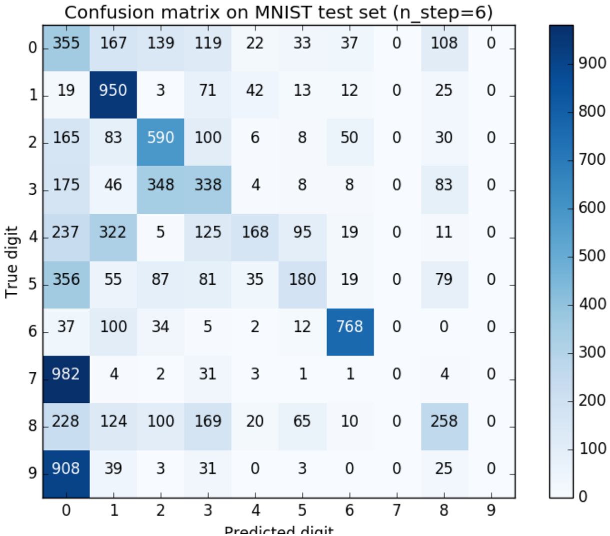
- Make sure that your initial state variables should be matrix filled with zeros whenever a mini-batch arrives
- We did not include bias term in our implementation

Experiments



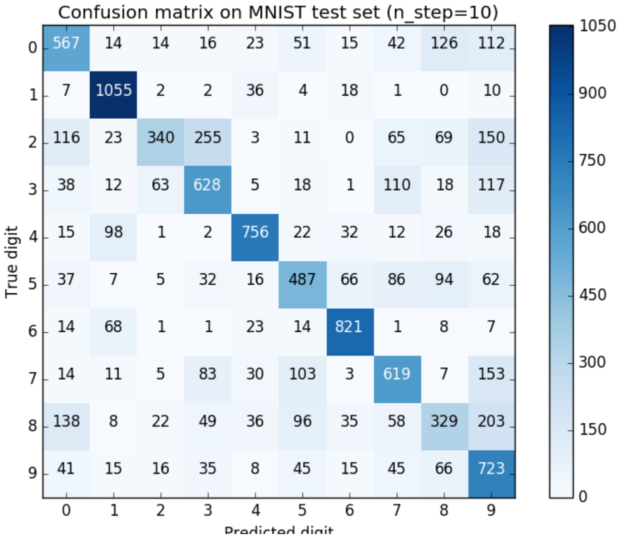
True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	980	0	0	0	0	0	0	0	0	0
1	1135	0	0	0	0	0	0	0	0	0
2	1030	0	0	0	0	0	2	0	0	0
3	1010	0	0	0	0	0	0	0	0	0
4	982	0	0	0	0	0	0	0	0	0
5	892	0	0	0	0	0	0	0	0	0
6	915	0	1	0	0	0	42	0	0	0
7	1028	0	0	0	0	0	0	0	0	0
8	974	0	0	0	0	0	0	0	0	0
9	1009	0	0	0	0	0	0	0	0	0

Experiments



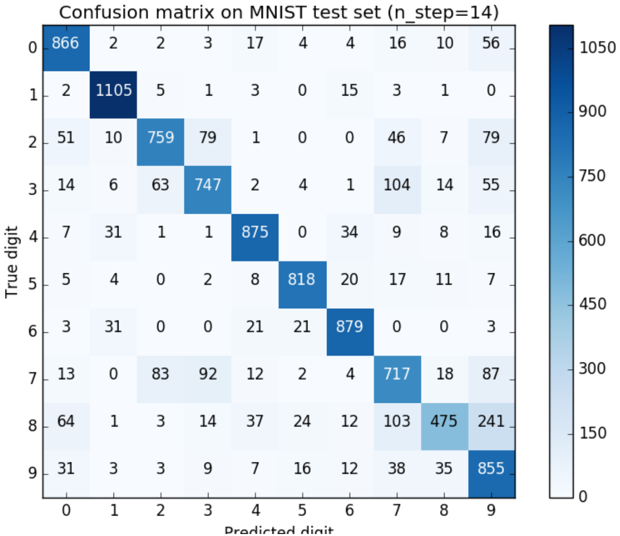
True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	355	167	139	119	22	33	37	0	108	0
1	19	950	3	71	42	13	12	0	25	0
2	165	83	590	100	6	8	50	0	30	0
3	175	46	348	338	4	8	8	0	83	0
4	237	322	5	125	168	95	19	0	11	0
5	356	55	87	81	35	180	19	0	79	0
6	37	100	34	5	2	12	768	0	0	0
7	982	4	2	31	3	1	1	0	4	0
8	228	124	100	169	20	65	10	0	258	0
9	908	39	3	31	0	3	0	0	25	0

Experiments



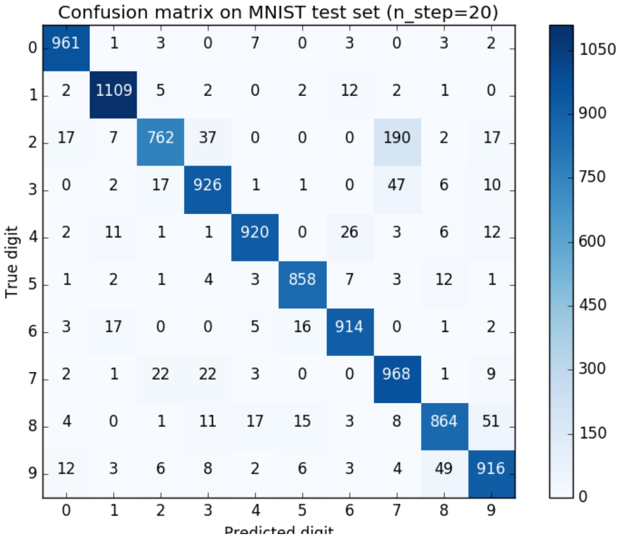
True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	567	14	14	16	23	51	15	42	126	112
1	7	1055	2	2	36	4	18	1	0	10
2	116	23	340	255	3	11	0	65	69	150
3	38	12	63	628	5	18	1	110	18	117
4	15	98	1	2	756	22	32	12	26	18
5	37	7	5	32	16	487	66	86	94	62
6	14	68	1	1	23	14	821	1	8	7
7	14	11	5	83	30	103	3	619	7	153
8	138	8	22	49	36	96	35	58	329	203
9	41	15	16	35	8	45	15	45	66	723

Experiments



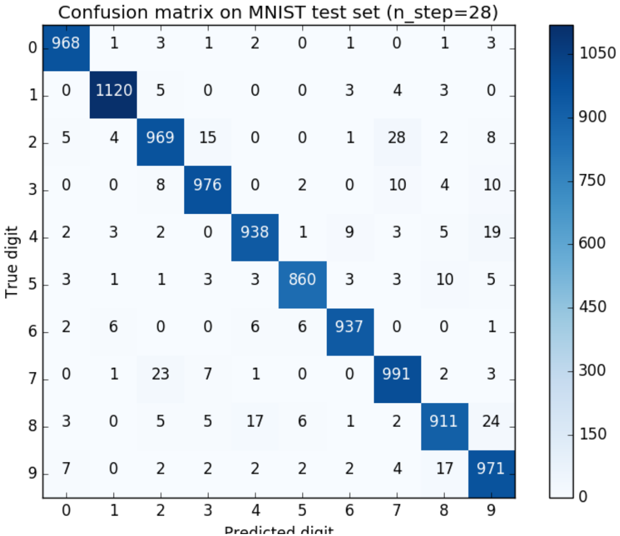
True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	866	2	2	3	17	4	4	16	10	56
1	2	1105	5	1	3	0	15	3	1	0
2	51	10	759	79	1	0	0	46	7	79
3	14	6	63	747	2	4	1	104	14	55
4	7	31	1	1	875	0	34	9	8	16
5	5	4	0	2	8	818	20	17	11	7
6	3	31	0	0	21	21	879	0	0	3
7	13	0	83	92	12	2	4	717	18	87
8	64	1	3	14	37	24	12	103	475	241
9	31	3	3	9	7	16	12	38	35	855

Experiments



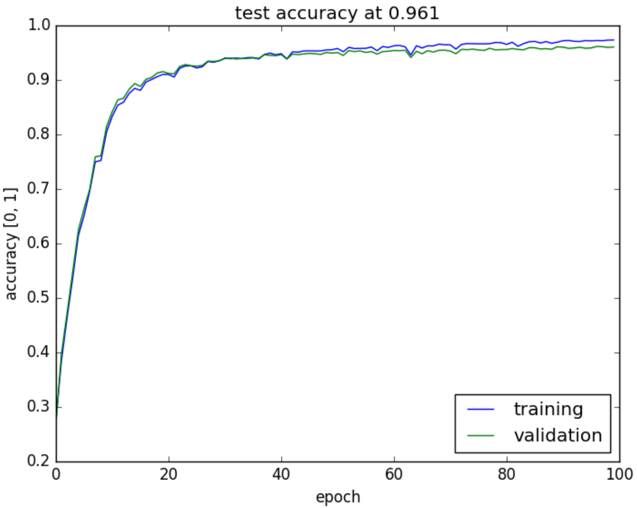
True digit \ Predicted digit	0	1	2	3	4	5	6	7	8	9
0	961	1	3	0	7	0	3	0	3	2
1	2	1109	5	2	0	2	12	2	1	0
2	17	7	762	37	0	0	0	190	2	17
3	0	2	17	926	1	1	0	47	6	10
4	2	11	1	1	920	0	26	3	6	12
5	1	2	1	4	3	858	7	3	12	1
6	3	17	0	0	5	16	914	0	1	2
7	2	1	22	22	3	0	0	968	1	9
8	4	0	1	11	17	15	3	8	864	51
9	12	3	6	8	2	6	3	4	49	916

Experiments

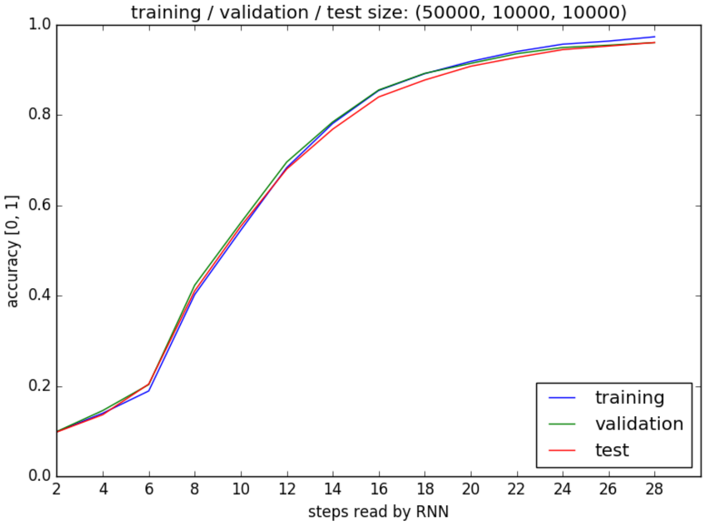


True digit \ Predicted digit	0	1	2	3	4	5	6	7	8	9
0	968	1	3	1	2	0	1	0	1	3
1	0	1120	5	0	0	0	3	4	3	0
2	5	4	969	15	0	0	1	28	2	8
3	0	0	8	976	0	2	0	10	4	10
4	2	3	2	0	938	1	9	3	5	19
5	3	1	1	3	3	860	3	3	10	5
6	2	6	0	0	6	6	937	0	0	1
7	0	1	23	7	1	0	0	991	2	3
8	3	0	5	5	17	6	1	2	911	24
9	7	0	2	2	2	2	2	4	17	971

Accuracy vs Epoch for n_steps = 28



Accuracy vs RNN rows


training / validation / test size: (50000, 10000, 10000)
accuracy [0, 1]
steps read by RNN
training
validation
test