Review Questions 4

Group 1
Chuan Su
Diego Alonso Guillen Rosaperez

December 1, 2019

1. RNNs learns to use past information to perform present prediction task. However, if the gap between the relevant information and the place where it's needed is large, i.e. when training an RNN on long sequences, you will need to run it over many time steps, making the unrolled RNN a very deep network. Just like any deep neural network, it may suffer from the vanishing/exploding gradients problem and RNNs become unable to learn to connect the information, since the gradients would then to be zero or a very large number, respectively.

2. The key idea of LSTM is that the network can learn what to store in the long-term state, what to throw away, and what to read from it.

   (a) **forget gate** The forget gate (controlled by $f_{(t)}$) controls which parts of the long-term state should be erased or kept. This is done with both the information of the previous step and the current step.

   (b) **input gate** The input gate (controlled by $i_{(t)}$) controls which parts of $g_{(t)}$ should be added, and with which relevance (0 not important, 1 very important) to the long-term state (this is why we said it was only partially stored).

   (c) **output gate** The output gate (controlled by $o_{(t)}$) controls which parts of the long-term state should be carried over to the next time step.

3.
$$\frac{\partial E}{\partial u} = \frac{\partial E^{(2)}}{\partial u} + \frac{\partial E^{(1)}}{\partial u}$$

$$\frac{\partial E^{(1)}}{\partial u} = \frac{\partial E^{(1)}}{\partial y^{(1)}} \frac{\partial y^{(1)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial s^{(1)}} \frac{\partial s^{(1)}}{\partial u}$$

$$\frac{\partial E^{(2)}}{\partial u} = \frac{\partial E^{(2)}}{\partial y^{(2)}} \frac{\partial y^{(2)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial s^{(2)}} \frac{\partial s^{(2)}}{\partial u} + \frac{\partial E^{(2)}}{\partial y^{(2)}} \frac{\partial y^{(2)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial s^{(2)}} \frac{\partial 2^{(2)}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial s^{(1)}} \frac{\partial s^{(1)}}{\partial u}$$

4. It is not a good autoencoder since the the decoder $f_4(h3)$ just learns the reverse mapping. Obvisouly such an autoencoder will reconstruct the training data perfectly, but

it will not have learned any useful data representation in the process (and it is unlikely to generalize well to new instances). So in this sense, it will over-fit. To improve this, we could drop out some layers while training. This would increase the performance of its generalization.

5. **Gibbs sampling** is an approach to building a Markov chain that samples from $pmodel(x)$, in which sampling from $T(x'|x)$ is accomplished by selecting one varable x_i and sampling it from p_{model} conditioned on its neighbors in the undirected graph G defining the structure of the energy-based model. When deep learning models contain a very large number of latent variables and we are able to group the hidden units into layers with a matrix describing the interaction between two layers, which allow the indicidual steps of the algorithm to be implemented with efficient matrix product operations or sparsely connected generalizations. In the example of RBM, it has a single layer of latent variables that may be used to learn a representation for the input. Its hiddent units are organized into large groups called layers, the connectivity between layers is described by a matrix, the connectivity is relatively dense, this model is designed to allow efficient Gibbs sampling.

6. When an autoencoder is neatly symmetrical, like the one we just built, a common technique is to tie the weights of the decoder layers to the weights of the encoder lay ers. This halves the number of weights in the model, speeding up training and limit ing the risk of overfitting. Specifically, In a network with N layers (not counting the input layer), the decoder layer weights can be defined as $W_{N-l+1} = W_1^T)$, with $l = 1, 2, ..., \frac{N}{2}$.