Homework 5 – Deep Learning (CS/DS541, Whitehill, Spring 2020)

You may complete this homework assignment either individually or in teams up to 2 people.

In this assignment you will train a recurrent neural network to solve a simple simulation task (n-back problem). The RNN you create and train should adhere to the following specification:

$$\begin{aligned} J_t(\mathbf{U},\mathbf{V},\mathbf{w}) &= \frac{1}{2}(\hat{y}_t - y_t)^2 \\ \hat{y}_t &= \mathbf{h}_t^\top \mathbf{w} \\ \mathbf{h}_0 &= \mathbf{0} \\ \mathbf{h}_t &= \tanh(\mathbf{z}_t) \\ \mathbf{z}_t &= \begin{bmatrix} \mathbf{U} & \mathbf{V} \end{bmatrix} \begin{bmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{U}\mathbf{h}_{t-1} + \mathbf{V}\mathbf{x}_t \end{bmatrix} \end{aligned}$$

For simplicity, each $y_t \in \mathbb{R}$ (i.e., a scalar). We will use the sum-of-squared-errors cost function.

Your job is to (1) derive a recursive algorithm to compute the gradient updates for the network; and (2) implement SGD for this procedure to solve the n-back problem (for just $n = 2$). In particular, for a sequence containing T terms (with inputs $\mathbf{x}_1, \ldots, \mathbf{x}_T$ and corresponding outputs $y_1, \ldots, y_T$), you should compute the gradient, w.r.t. each of $\mathbf{U}, \mathbf{V}, \mathbf{w}$, of the *total* cost $J(\mathbf{U},\mathbf{V},\mathbf{w}) = \sum_{t=1}^{T} J_t(\mathbf{U},\mathbf{V},\mathbf{w})$. See `rnn.pdf` for guidance.

1. **RNN for the n-back problem (40 points)**: Implement SGD for an RNN in Python code (using numpy, no other libraries). Your network should contain exactly 6 hidden units, 1 input unit, and 1 output unit. **Include a screenshot** showing how the loss decreases durig training. There is no validation or test set in this assignment – just focus on driving the loss toward 0. Depending on the initial seed of the data generator, you should be able to obtain a loss value < 0.05, e.g.:

   ```
   11.4657453617
   11.2834015915
   11.079596572
   10.849944084
   10.5912803802
   10.3018414418
   9.98134933669
   9.63088887735
   9.25253518532
   8.84889117335
   8.42302162225
   7.97962392769
   7.52826431536
   ...
   0.0530270146118
   0.0499058526839
   ```

   Note that, even with correctly implemented gradient expressions, the cost function may not decrease monotonically (depending on the learning rate). For this reason, it is strongly recommended that you use `check_grad` or `approx_fprime` to help you debug your gradient descent procedure.

In addition to your Python code (`homework5_WPIUSERNAME1.py` or `homework5_WPIUSERNAME1_WPIUSERNAME2.py` for teams), create a PDF file (`homework5_WPIUSERNAME1.pdf` or `homework5_WPIUSERNAME1_WPIUSERNAME2.pdf` for teams) containing the screenshots described above.