ECE1513H - Winter 2020 **Assignment 6**
University of Toronto - LEC0101 (Prof. Papernot) **Due April 6**

This assignment contains 9 questions worth a total of 11 points. Because assignments will help you learn things that are complementary to what we covered in class, they should be completed on your own. Otherwise, you will not learn from taking this course and you are harming yourself.

This assignment comes with a starter notebook here:

`https://colab.research.google.com/drive/1KH67ZWuQgbxT8SYkyA3gT1Nf91i7tCdF`

It is **mandatory** to hand in a Python script (i.e., file with extension `.py`) with this assignment. You can generate a Python script automatically through the `File > Download .py` menu in Colab. Submitting a link to a Colab notebook does **not** constitute a valid submission.

Recall that you can attach a free GPU to your Colab notebook and accelerate your computations. Go to `Runtime > Change runtime type`, and choose GPU under "hardware accelerator".

Problem 1 The purpose of this problem is to illustrate how to train recurrent neural networks (RNNs). We will illustrate concepts from the class, and learn a RNN that models an Ornstein-Uhlenbeck process. You do not need to know anything about Ornstein-Uhlenbeck processes for the purpose of this assignment, but if you are curious, you can learn more about them online (e.g., `https://en.wikipedia.org/wiki/Ornstein-Uhlenbeck_process`).

Code in the starter notebook includes a function `generate_ou_process` to sequentially generate Ornstein-Uhlenbeck time series, to which Gaussian noise is added. The first graph in the notebook includes an example with the ground truth Ornstein-Uhlenbeck time series and its noisy counterpart. Our goal is to train a RNN to denoise the time series.

The RNN we use in this assignment uses Gated Recurrent Units (GRUs) rather than LSTMs we studied in class. GRUs are similar to LSTMs but do not include an output gate. Here are the 3 gates implemented by the GRU:

Dear students,

In case it was ambiguous, I have updated the equation of the output gate in assignment 6 to make it clear that it takes the Hadamard product between the reset gate and previous hidden state.

-NP

- Update gate: $z_t \leftarrow \sigma(W_z \cdot x_t + U_z \cdot h_{t-1} + b_z)$
- Reset gate: $r_t \leftarrow \sigma(W_r \cdot x_t + U_r \cdot h_{t-1} + b_r)$
- Output gate: $o_t \leftarrow z_t \circ h_{t-1} + (1 - z_t) \circ \tanh(W_o \cdot x_t + U_o \circ (r_t \circ h_{t-1}) + b_o)$

where σ is the sigmoid function, x_t the input at time step t, h_t the hidden state at time step t, and all other parameters either weights or biases. For instance, W_z, U_z, and b_z are the parameters for the update gate. They respectively represent the weight of the connection to the input, the weight of the connection to the previous hidden state, and the bias.

1. (1 point) Fill the line implementing the forward pass for the update gate in `apply_fun_scan`.
2. (1 point) Fill the line implementing the forward pass for the reset gate in `apply_fun_scan`.
3. (1 point) Fill the line implementing the forward pass for the output gate in `apply_fun_scan`.
4. (1 point) Fill the missing line in the function `mse_loss`. The function returns the mean squared error loss between the model's predictions (i.e., `preds`) and the target sequence (i.e., `targets`).

np.multiply: Multiply arguments element-wise.

output_gate = np.multiply(update_gate, hidden) + np.multiply(1-update_gate, np.tanh(np.dot(inp, out_W) + np.dot(np.multiply(reset_gate, hidden), out_U) + out_b))

it means it takes all but EXCLUDE the last step

5. (2 points) Fill the missing lines at the top of the cell titled "Training the RNN". These lines should use the optimizers pre-built into JAX to instantiate an Adam optimizer. As seen in previous homeworks, you should obtain three things from the pre-built JAX optimizer: a method `opt_init` that takes in a set of initial parameter values returned by `init_fun` and returns the initial optimizer state `opt_state`, a method `opt_update` which takes in gradients and parameters and updates the optimizer states by applying one step of optimization, and a method `get_params` which takes in an optimizer state and returns current parameter values.

6. (1 point) Fill the lines that define `x_in` (the input) and `y` (the output) of our recurrent neural network. The RNN should take in all but the last step of the noisy time series, and predict all but the first step of the ground truth time series (i.e., the time series before it was noised).

7. (1 point) Fill the line that calls `update` to take one step of gradient descent on the batch of training data sampled.

8. (2 points) As done in prior assignments, perform a hyperparameter search to find a good value for your learning rate. Describe briefly how you conducted the search, the value you chose, and why you chose that value. You may find it useful to call `plot_ou_loss(train_loss_log)`

9. (1 point) Using the last cell of the notebook, comment qualitatively on the difference between the predicted time series, the ground truth, and the noisy time series. You will have to reuse the definition of `x_in` (the input) and `y` from the question above.

*
* *

but EXCLUDE the first step