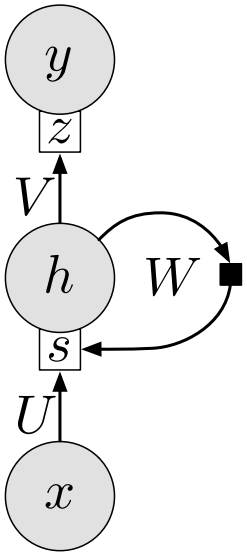
Recurrent Neural Networks

Question 1: BPTT

[5 marks] The figure on the right shows an RNN, in which

$$
\begin{aligned}
s &= xU + hW + b \\
h &= \sigma(s) \\
z &= hV + c \\
y &= \sigma(z)
\end{aligned}
$$



Notice that we are using the mathematical convention of assuming vectors are row-vectors.

For the following questions, assume you are given a dataset that has many samples of sequences of inputs and output targets. Each sequence consists of inputs x^i, for $i = 1, \ldots, \tau$, that produces a sequence of network outputs y^i, which you wish to match to a corresponding sequence of targets, t^i. The cost function for such a sequence is,

$$E(y^1, \ldots, y^\tau, t^1, \ldots, t^\tau) = \sum_{i=1}^{\tau} C(y^i, t^i)$$

Show that

$$\nabla_W E = \sum_{i=1}^{\tau-1} (h^i)^{\mathrm{T}} \left[\nabla_{h^{i+1}} E \odot \sigma' \left(s^{i+1} \right) \right]$$

Clearly justify each step.

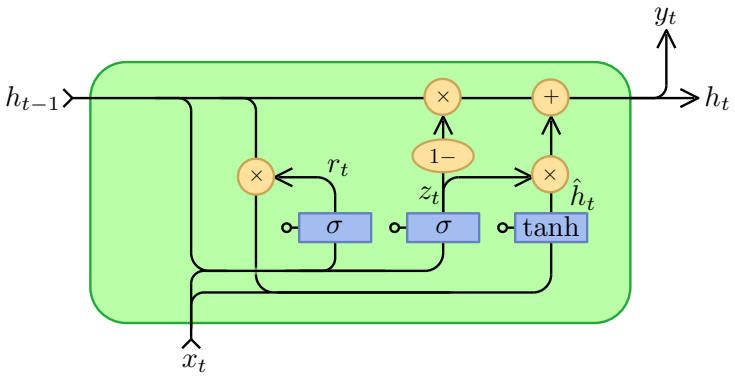
Question 2: Implementing a GRU

Download the jupyter notebook `as08.ipynb`. This notebook also requires PyTorch (`torch`), `tqdm`, and the file `origin_of_species.txt`.

In this question, you will implement a Gated Recurrent Unit (GRU) network. The figure below shows a schematic of a GRU; the corresponding mathematical formulation is:

$$
\begin{aligned}
z_t &= \sigma\left(x_t U_z + h_{t-1} W_z + b_z\right) \\
r_t &= \sigma\left(x_t U_r + h_{t-1} W_r + b_r\right) \\
\hat{h}_t &= \tanh\left(x_t U_h + \left(r_t \odot h_{t-1}\right) W_h + b_h\right) \\
h_t &= \left(1 - z_t\right) \odot h_{t-1} + z_t \odot \hat{h}_t \\
y_t &= \phi\left(h_t V + b_y\right)
\end{aligned}
$$

where σ is the logistic function, and ϕ is the logarithm of the softmax function, $U_\square$, $W_\square$, and V are connection-weight matrices, and $b_\square$ are biases.



Gated Recurrent Unit. Adapted from Jeblad under Creative Commons License

Much of the functionality for running and training a GRU (using backprop through time, or BPTT) is implemented in the class `GRU` in the notebook. However, parts of it still need to be filled in.

(a) [5 marks] Complete the implementation of the class by completing the following functions, as described by their docstrings:

   (a) `__init__`: sets up all the weight matrices and activation functions

   (b) `step`: takes one timestep

   (c) `output`: produces the output, y_t, from the hidden state

   Note that you should **NOT** use PyTorch's `nn.GRU` or `nn.GRUCell` functions. You must implement the guts of the GRU yourself, using other, more basic PyTorch modules like `nn.Linear` and `nn.Sigmoid`, etc.

(b) [4 marks] Add some lines to the notebook that will run an experiment that measures the number of predictions that match for at least n characters. That is, given many different priming strings of 10 characters (at least 200 strings, chosen randomly from the training text), how often does your model's predictions match the subsequent text...

   - for at least 0 characters?
   - for at least 1 character?
   - for at least 2 characters?

     ⋮

   - for at least 100 characters?

   You might find the Python function `find` useful for this task, and you can use the `Dataset` attribute `text` instead of the `DataLoader`. Keep in mind that a priming string of length 10 might occur multiple times in the training text. You should consider each occurrence when assessing the predicted string, and choose the longest matching string of subsequent characters. For example, the string '`ch species`' occurs 3 times between characters 1,000 and 11,000 in the full text. The prediction '`ch species has been independestly foll...`' matches one of the occurrences in the first 19 characters, while the other 2 occurrences only match 1, and 3 characters. So, the matching length for the prediction on that sequence is 19.

   Produce a plot, with "Matching String Length" on the horizontal axis and "Trials" on the vertical axis. Trials refers to the number of trials with the given matching string length. For each length of matching string (0 to 100), plot a point indicating how many of the predictions tested were correct to at least that many characters.

What to submit

Question 1: You can:

- typeset your solutions using a word-processing application, such as Microsoft Word, LaTeX, Google docs, etc., or
- write your solutions using a tablet computer, or
- write your solutions on paper and take photographs.

In any case, it is **your responsibility** to make sure that your solutions are sufficiently legible.

Submit your solution to Crowdmark only. Crowdwark will accept PDFs or image files (JPEG or PNG). You may submit multiple files for each question, if needed.

Question 2: You must submit your jupyter notebook in two places: **Crowdmark**, and **D2L**.

Crowdmark: Export your jupyter notebook as a PDF, and submit the PDF to Crowdmark.

D2L: Submit your `ipynb` file to Desire2Learn in the designated dropbox. You do not need to zip the file.

Make sure you submit your solutions, and not just a copy of the supplied skeleton code. We suggest you rename the `ipynb` file by replacing "`YOU`" with your WatIAM ID (eg. `a08_jorchard.ipynb`).

Important note about submitting code
We re-run your notebook in its entirety. So make sure that all your code cells run, from top to bottom. Any code cell that crashes will stop us from re-running your notebook (and you don't want that). Also, we extract the class and function definitions and autotest them. To facilitate this extraction process, please ensure that:

- all `import` commands are in a single code cell (at the top of the notebook), and
- code cells that contain function definitions and/or class definitions should not include any lines of code that are not part of the definition.