# Population Coding, LSTMs, Adversarial Inputs

Due at 4:00pm on 5 April 2019

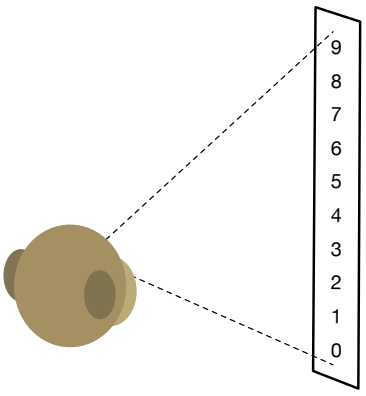
## What you need to get

- `YOU_a5.ipynb`: a Python notebook (hereafter called "the notebook")
- `monkey0.npz` - `monkey4.npz`: Data files (just need one)
- `lif.py`: Module containing LIF population coding stuff
- `Network.py`: Neural network class.
- `mnist.pkl`: MNIST dataset
- `mnist_loader.py`: Module for loading MNIST
- `MNIST_trained_784-100-100-10.npy`: Pre-trained neural network

## What you need to know

**In Q1 of this assignment, you will pretend you are a neurophysiologist performing an experiment on a monkey. However, you will *simulate* the monkey's neurons using spiking leaky integrate-and-fire (LIF) neurons.**

You are an electrophysiologist and have successfully installed 40 probes in a monkey's nuclei prepositus hipoglossi (NPH, the name is not important, but it sounds cool to say it to your friends at a party). You have noticed that these neurons seem to be involved in the coding of up-down gaze direction. To measure the association of the neural responses to gaze direction, you have created a lab setup like the one shown in the figure on the right. The monkey's vertical gaze direction is measured on a scale from 0 to 9.




Since you don't really have a monkey with electrodes in its brain (do you?), you will simulate the experiment. The notebook `YOU_a5.ipynb` contains a function called `MonkeyFixationSpikes`, which can be called like this:

```
st = MonkeyFixationSpikes(x, T, pop)
```

where `x` is the gaze direction (on the scale from 0 to 9), `T` is the duration of the fixation (in seconds), and `pop` is the matrix containing the parameters for the neurons (the notebook will read this matrix in from a file). The output `st` holds the resulting spike times; it's a list that contains 40 arrays, and each array holds the spike times for one neuron.

The script `YOU_a5.ipynb` sets up the experiment. It reads in a matrix containing the parameters for a population of 20 LIF neurons and stores them as an array in the variable `pop` (each column holds the 6 LIF parameters of a single neuron).

While recording the neural spikes, the monkey performed a sequence of 5 fixations, each 0.5 seconds long. The corresponding spike-times are stored in a file that this script reads in. The goal of Q1 is to decode the secret code directly from the monkey's neural spikes.

**What to do**

# 1. Electrophysiology Experiment

(a) **Behavioural Sampling** [4 marks]

**Edit the script to sample the tuning curves for the 40 neurons over a range of different inputs.**

Here is your experimental protocol:

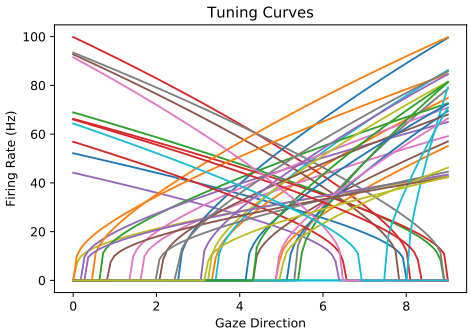
i. Choose 100 different gaze directions.
*In other words, choose 100 $x$-values in the range [0,9]. Store them in an array $X$.*

ii. Fix the monkey's gaze on each chosen direction for 0.4 seconds. Record the corresponding spikes.
*In other words, call `MonkeyFixationSpikes` for each of those $x$-values. For each value, estimate the firing rate of each neuron. Store all these firing rates in a $100 \times 40$ matrix called $A$.*

(b) **View Tuning Curves** [1 mark]

**Plot the tuning curves of the 40 neurons on a single axis.**

The plot should have gaze direction on the horizontal axis, and firing rate (Hz) on the vertical axis, plotting one curve for each of the 40 neurons (all on the same axis). That is, plot the tuning curves for all of the neurons. Be sure to label your axes appropriately. An example is shown on the right.




(c) **Compute the Decoding Weights** [2 marks]

**Edit the notebook to compute the optimal linear decoding weights from your behavioural sampling data.**

Use your dataset of gaze directions and associated firing rates ($X$ and $A$) to compute the optimal linear decoders. To solve the resulting least-squares problem, you may only use NumPy's built-in `inv` and `dot` functions; you may NOT use `lstsq`, `solve`, or `pinv`.

(d) **View Spike Raster of Unknown Sequence** [1 mark]

**Edit the notebook to display the spikes for the 20 probed neurons as the monkey selected its secret code.**

The monkey chose an unknown 5-digit code by fixating on a sequence of 5 digits on the scale. For each digit, the monkey held its gaze for 0.5 seconds. The notebook automatically reads in these spike times from a file. Create a spike raster plot by calling the function `PlotSpikeRaster` supplied in `lif.py`. Label it.

(e) **Decode the Unknown Code** [3 marks]

**Edit the notebook to decode the supplied spike trains so you can tell where the monkey's gaze was fixed.**

Add lines to the notebook to estimate the firing rate of each of the 20 neurons for each fixation. You can use the function `CountSpikes` from the `lif.py` module if you like (type '`? lif.CountSpikes`' to see its help file). Use those firing rates, and the decoding weights from part 1c, to infer the monkey's gaze direction.

(f) **Display the Unknown Code** [2 marks]

**Plot the sequence of gaze directions, and display the secret code.**

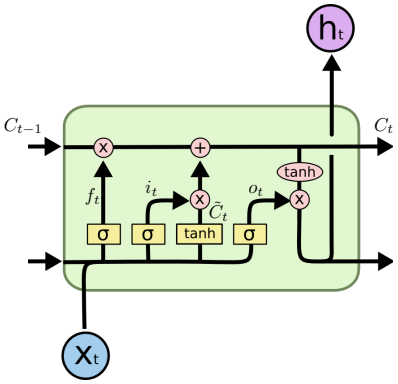
Add lines to the notebook to plot the estimated gaze direction (vertical axis) over the 5 fixations (horizontal axis). Plotting a single dot per fixation is sufficient. Turn on the grid using the command `grid('on')`. Label, as usual, and **add the decoded secret code to the plot title.**

## 2. LSTM

The figure on the right shows a Long Short-Term Memory (LSTM) unit. At each step, it receives an input, $x$, a also recurrently recycles its hidden state, $h$, and the "cell" state, $C$. The formulas on the left detail the operation of the various gates.

Let $v_t = \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}$ be the aggregated input.

$$
\begin{aligned}
f_t &= \sigma\left(W_f v_t + b_f\right) \\
i_t &= \sigma\left(W_i v_t + b_i\right) \\
o_t &= \sigma\left(W_o v_t + b_o\right) \\
\tilde{C}_t &= \tanh\left(W_C v_t + b_C\right) \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \\
h_t &= o_t \odot \tanh\left(C_t\right)
\end{aligned}
$$




This figure was adapted from Colah's blog.

Let the connection weights be,

$$
\begin{aligned}
W_f &= [0\ 8\ 0\ 0] \quad b_f = -4 & \qquad W_i &= [0\ 0\ 9\ 0] \quad b_i = -4.5 \\
W_o &= [0\ 0\ 0\ 10] \quad b_o = -5 & \qquad W_C &= [1\ 0\ 0\ 0] \quad b_C = 0
\end{aligned}
$$

And the initial states be,

$$h_0 = [0.05] \qquad C_0 = [-0.02]$$

Note that $\frac{d}{d\phi}\tanh\phi \approx 1$ for small values of $\phi$.

(a) [6 marks] Determine the outputs, $C_t$ and $h_t$, for the inputs listed below. For each, show the values of the gates ($f_t$, $i_t$, $o_t$) and describe how the updated values, $C_t$ and $h_t$, relate to the input values $C_{t-1}$ and $h_{t-1}$.

i. $x_t = [1\ 0\ 0]$ ii. $x_t = [0\ 1\ 0]$ iii. $x_t = [1\ 0\ 1]$

(b) [2 marks] Suppose you want your new cell state, $C_t$, to approximate the sum of your old cell state and your hidden state, $h_{t-1}$. What should your input $x_t$ be? Justify your answer, and demonstrate your solution on the LSTM setup given above.

(c) [3 marks] Suppose, instead, that you want your LSTM's output, $h_t$, to approximate the average of your previous cell state and the input value of $h_{t-1}$. What should your input $x_t$ be? Justify your answer, and demonstrate your solution on the LSTM setup given above.

## 3. Adversarial Inputs

(a) **Gradient with respect to Input:** [3 marks] Implement the function `GradientInput` to compute $\frac{\partial E}{\partial x}$ for a network given input $x$ and target $t$.

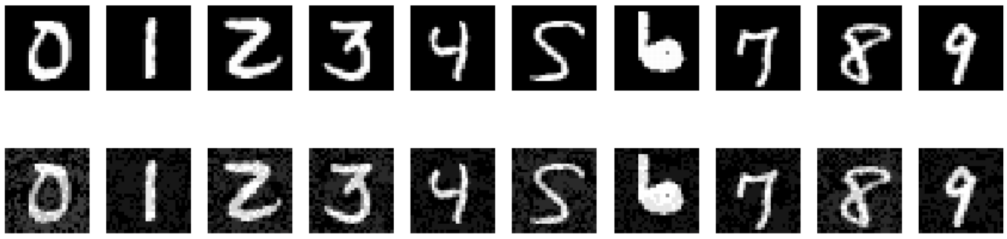
(b) **FGSM:** [6 marks] Complete the function `FGSM`, which implements the Fast Gradient Sign Method. See the code for details on how it is supposed to work. You function must call your `GradientInput` function, and it must work for both targeted and untargeted adversarial attacks.

(c) **Untargeted Adversarial Perturbation:** [4 marks] Create and train a neural network on the MNIST dataset. Your network should use logistic hidden layers, softmax activation for the output layer, and categorical cross entropy as a cost function. You can use the supplied SGD to train your network, which should achieve at least 90% accuracy on the MNIST test set. Alternatively, you can load the pre-trained network (see the notebook for how).

Extract one of each digit (from 0 to 9) from either the MNIST test or training set and ensure that each digit sample is **correctly classified** by your network.

For each digit, make a copy of it, then run your `FGSM` function until it is **incorrectly classified** by your network with classification value of **at least 50%**. Print the original class of the digit, the perturbed class, the perturbed classification value, and show the original images in one row, and the adversarial images in the row below. Try to make the adversarial images look as similar to the original images as possible. An example of what you should print is shown below.

```
digit 0 classified as 2 with classification value 0.8891168549669541
digit 1 classified as 8 with classification value 0.7364861442830957
digit 2 classified as 3 with classification value 0.9628042081084451
digit 3 classified as 5 with classification value 0.9685646687861471
digit 4 classified as 9 with classification value 0.9949879591566249
digit 5 classified as 8 with classification value 0.9327411334641802
digit 6 classified as 4 with classification value 0.974070234926126
digit 7 classified as 9 with classification value 0.7033308850905025
digit 8 classified as 5 with classification value 0.9961964004600434
digit 9 classified as 4 with classification value 0.9389675101093534
```



(d) **Targeted Adversarial Perturbation:** [4 marks] Find and make a copy of a correctly-classified and unperturbed 5, 1, and 7 from either the MNIST train or test set. Use your `FGSM` function again to do each of the following:

i. Perturb the 5 until your network incorrectly classifies it as either a 3 or a 9 with classification value of at least 80%.

ii. Perturb the 1 until your network incorrectly classifies it as either a 0, a 2, or an 8 with classification value of at least 80%.

iii. Perturb the 7 until your network incorrectly classifies it as either a 6 or a 4 with classification value of at least 80%.

For each digit, print the original class, the perturbed class, the perturbed classification value, and show just the perturbed images. As before, try to make the perturbed digit look as similar to the original digit as possible. An example of what you should print is shown below.

```
perturbed 5 classified as 3 with classification value 0.9924956406055785
perturbed 1 classified as 8 with classification value 0.9985827844791121
perturbed 7 classified as 6 with classification value 0.8976119780906765
```

## What to submit

Your assignment submission should be a single jupyter notebook file, named (`<WatIAM>_a5.ipynb`), where `<WatIAM>` is your UW WatIAM login ID (not your student number). The notebook must include solutions to **all** the questions. Submit this file to Desire2Learn. You do not need to submit any of the additional files supplied for the assignment.