9.49 / 9.490 Neural Circuits for Cognition
Homework 4
Due: **Tuesday** November 19, by midnight.

In this assignment, we will explore gradient learning by stochastic estimation: we will derive some basic results and try our hand at coding such learning to study learning speed. We will also take the first steps toward thinking about unsupervised learning.

1) **Stochastic gradient learning with perturbations.**

a. Recall that the characteristic eligibility in REINFORCE is defined to be $e \equiv \frac{\partial \log P_W(\Omega)}{\partial W}$, where $P_W(\Omega)$ is the probability density of trajectory Ω with parameters W. First, using the fact that probabilities sum to one, (i.e. $\int P_W(\Omega) d\Omega = 1$ for continuous-valued Ω, or alternatively $\sum_\Omega P_W(\Omega) = 1$ when Ω takes discrete values), prove that the expected value of the eligibility taken over $P_W(\Omega)$ satisfies $\langle e \rangle_\Omega = 0$ always. Thus, the REINFORCE updates $\Delta W \propto Re$ are only non-zero on average when the reinforcement signal covaries with the eligibility. Second, suppose that the reinforcement signal is made up of a piece $\bar{R}$ that does not co-vary with the eligibility, and a piece δR that does. Write down the signal-to-noise ratio of the weight updates as the ratio of $\langle \Delta W \rangle$ to the square-root of the variance, $\sqrt{\langle (\Delta W)^2 \rangle}$, where $\Delta W = Re$. By expanding the variance in terms of $\bar{R}, \delta R$, explain why subtraction of a baseline $\bar{R}$ that is uncorrelated with e is important. In reinforcement learning, REINFORCE is a method for direct policy gradient learning (changing parameters of an actor based on reinforcement-based gradients to directly optimize its outputs); however, appropriate computation and subtraction of a baseline $\bar{R}$, often given by the expected value of reinforcement, is critical to making direct policy gradient work. This is the job of the critic in actor-critic models.

b. Gradient learning on an error function (negative reward). Consider a linear feed-forward neural network with N independent identically distributed (iid) Gaussian inputs $\mathbf{x}$ (zero mean, unit variance) and M output units $\mathbf{y} = \mathbf{w}\mathbf{x}$, where $\mathbf{w}$ are the weights connecting the inputs to the outputs. Let the desired outputs be $\mathbf{y} = \mathbf{d}$, where we have assumed there is some set of weights $\mathbf{w}^*$ such that $\mathbf{d} = \mathbf{w}^*\mathbf{x}$ (thus ensuring that the desired outputs are achievable). The error in performance is then given by

$$E = \frac{1}{2}|\mathbf{y} - \mathbf{d}|^2 = \frac{1}{2}|(\mathbf{w} - \mathbf{w}^*)\mathbf{x}|^2 \equiv \frac{1}{2}||\mathbf{W}\mathbf{x}||^2 = \frac{1}{2}\mathbf{x}^T\mathbf{W}^T\mathbf{W}\mathbf{x}.$$

First, show that the expected error $\langle E \rangle_x = \frac{1}{2}||\mathbf{W}||^2 = \frac{1}{2}\sum_{ij} W_{ij}^2$. Second, compute the one-step gradient update $-\eta \nabla_{\mathbf{W}} E$, use this value for $\Delta\mathbf{W}$ in the expression $\mathbf{W}^{t+1} = \mathbf{W}^t + \Delta\mathbf{W}^t$, and derive the expression for the expectation value of $\langle ||\mathbf{W}^{t+1}||^2 \rangle_x$ in terms of its value at iteration t. Iterate this recursion to obtain the following expression for error as a function of time:

$$\langle E^t \rangle_x = \left(1 - 2\eta + (N+2)\eta^2\right)^t \langle E^0 \rangle_x \tag{1}$$

Third, find the optimal value of η (defined by maximizing the drop in error at each time-step); insert this value into Eq. 1 and take the limit of large N to obtain that:

$$\langle E^t \rangle_x = \left(1 - \frac{1}{N}\right)^t \langle E^0 \rangle_x \approx e^{-t/\tau} \langle E^0 \rangle_x. \tag{2}$$

where here, for direct online gradient learning, $\tau \sim N$. Below we will show that the learning time τ is bigger for weight and node perturbation. [Hints: Recall that for zero-mean Gaussian r.v.'s x_i with variance σ^2, $\langle x_i \rangle_x = 0$; $\langle x_i x_j \rangle_x = \sigma^2 \delta_{ij}$, and also note that $\langle x_i^2 x_j^2 \rangle_x = \sigma^4(1 - \delta_{ij}) + 3\sigma^4 \delta_{ij}$.]

c. In class, we discussed the parallelized versions of learning by stochastic gradient estimation though weight and node perturbation (see Werfel, Xie, Seung 2004), in addition to the online gradient rule derived above:

$$\Delta W_G = -\eta \mathbf{W}\mathbf{x}\mathbf{x}^T \tag{3}$$

$$\Delta W_{WP} = \frac{-\eta}{\sigma^2}(E' - E)\xi_{WP} \tag{4}$$

$$\Delta W_{NP} = \frac{-\eta}{\sigma^2}(E' - E)\xi_{NP} x^T \tag{5}$$

where ξ_{WP}, ξ_{NP} are $M \times N$ and $M \times 1$-dimensional iid perturbations of zero mean and variance σ^2; E is the error in a given iteration of learning without any perturbation, while E' is the error in the same iteration when the corresponding perturbation is applied, so that $E' = \frac{1}{2}||(\mathbf{W} + \xi_{WP})\mathbf{x}||^2$ for weight perturbation, while $E' = \frac{1}{2}||\mathbf{W}\mathbf{x} + \xi_{NP}||^2$ for node perturbation. Code up these three learning rules using the following optimal learning rates for each rule: $\eta_G = \frac{1}{2+N}$; $\eta_{WP} = \frac{1}{(MN+2)(N+2)}$; $\eta_{NP} = \frac{1}{(M+2)(N+2)}$. It is important for the perturbations to be small compared to the weights or to the postsynaptic inputs for weight and node perturbation, respectively; I used the following parameters $\{N, M\} = \{100, 20\}$, $\mathbf{w}, \mathbf{w}^*$ uniformly randomly on the interval $[0, 1]$ (using the same initial $\mathbf{W}$ across rules), and $\sigma = 0.01$, but feel free to explore.

d. In each case, plot the learning curves (error as a function of iteration number) from [c.] above on a semilog plot (log on the y-axis). Explain why we should plotting the learning curve on a semilog plot, and explain what the slopes should mean. Determine the ratios of the learning curve slopes across the different learning rules, and relate them to M, N. What do you conclude? Show that changing η from the optimal values does not speed up convergence on average, thus the differences in learning rate are not due to the scaling of our choices of learning rate, but are intrinsic to the learning rule.

e. In each iteration of [c.], we ran the network once with perturbation, and then once without to obtain a baseline E. This is inefficient, because the network has to run twice. Run the learning without subtracting E, and find how much smaller of a learning rate you need for convergence. Show the learning curves in both cases; this illustrates again the importance of baseline subtraction. Next, we will try something more efficient than E, which is to only run the network with perturbations, and replace E by Eb, a running average of past errors E'. We do this by iterating $Eb = (1-\alpha)Eb + \alpha E'$ (using $\alpha = 0.1$ gives a $\approx$ 10-back running average). Be careful when defining Eb to not include the current iteration's E' into the estimate (explain why?). Try running with the same learning rates as you had when subtracting E, and compare performance; is this a reasonable replacement?

f. Modify your code so that $y = \tanh(wx)$, i.e., the units are now nonlinear, and modify your error function to also make that some nonlinear function of the output and desired output (but it should be minimized when the actual output equals the desired). Try out your code, making sure that perturbations are still small, to explore learning and show that it works.

g. Extra credit: implement handwritten digit recognition using node perturbation in a feedforward network with one hidden layer and the abridged MNIST database; use a $784 \times 25 \times 10$ network of sigmoid units. Learn not just the weights, but also the biases ($\Delta b_{NP} = \frac{-\eta}{\sigma^2}(E' - E)\xi_{NP}$). If you can, compare this performance with backprop.

2) **Finding the largest eigenvalue and eigenvector of a matrix by iteration.** Given any $N \times N$ matrix $\mathbf{A}$, there is a very simple method for computing its largest eigenvalue (λ_1) and the associated normalized eigenvector ($\mathbf{v}_1$), which is as follows. Take a random $N \times 1$ vector $\mathbf{x}$, then iterate

$$\mathbf{x}(n+1) = \frac{\mathbf{A}\mathbf{x}(n)}{||\mathbf{A}\mathbf{x}(n)||}$$

until steady state, which is, until $\mathbf{x}(n+1) = \mathbf{x}(n) \equiv \bar{\mathbf{x}}$.

a. Show that $\bar{\mathbf{x}} = \mathbf{v}_1$, the eigenvector corresponding to the largest eigenvalue. Do this by writing $\mathbf{x}(0)$ in terms of the eigenvectors of $\mathbf{A}$ with coefficients $c_\alpha(0)$; use the iteration update above to compute the coefficients $c_\alpha(n)$ of $\mathbf{x}(n)$ and show that the ratio

$$\frac{c_\alpha(n)}{c_1(n)} \to 0$$

as n increases, for any $\alpha \neq 1$. Thus, show that $\bar{\mathbf{x}} = c_1(n) \sum_\alpha \frac{c_\alpha(n)}{c_1(n)} \mathbf{v}_\alpha \approx c_1(n)\mathbf{v}_1$. It follows that $\frac{\bar{\mathbf{x}}^T \mathbf{A} \bar{\mathbf{x}}}{\bar{\mathbf{x}}^T \bar{\mathbf{x}}} = \lambda_1$.

b. Consider the matrix

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Derive or compute (e.g. using `eig` in Matlab) the eigenvectors and eigenvalues of this matrix. Next, plot the trajectories of a few random vectors as they evolve when iterated as above, to reach a steady state equal to $\mathbf{v}_1$.

c. Now consider the familiar linear neural network equation

$$\tau \frac{d\mathbf{x}}{dt} = -\mathbf{x} + \mathbf{A}\mathbf{x}.$$

where $\mathbf{A}$ is a diagonalizable square matrix. Show by writing down the solution for the network state $\mathbf{x}(t)$ in terms of the coefficients $c_\alpha(t)$ of eigenvectors of $\mathbf{A}$, and taking the appropriate ratios, that the network state tends toward a vector proportional to $\mathbf{v}_1$, if λ_1 is sufficiently large compared to the remaining eigenvalues. What about the magnitude of $\mathbf{x}$? We will see in class that, if we equate $\mathbf{x}$ with weights to a postsynaptic neurons, and $\mathbf{A}$ with the input correlation matrix, then this equation describes how the weights come to encode the top eigenvector of the inputs.