Qualifying exam, September 2020

Please do all six problems below, and turn in your completed exam as a jupyter notebook to Mary Brown by email by noon on Wednesday, September 16th. The problems are equally weighted.

You may use course notes, books, or existing internet references, but should not discuss the problems with anyone else (which, obviously, means that you shouldn't *post* the problems on the internet, nor solicit help from others). You may adapt computer code you have written elsewhere in these courses.

Please explain your answers, by citing results or theorems (e.g., "
$\mathrm{var}[A+B] = \mathrm{var}[A] + 2\mathrm{cov}[A,B] + \mathrm{var}[B]$ *by bilinearity of covariance*") and justifying algorithms (e.g., *"all nodes are eventually reached because the graph is connected"*). If you do use facts from elsewhere, please cite your sources.

As far as we are aware, these problems are original and haven't appeared in this form before. However, if you do find one of the problems posted and/or solved on the internet or some other public resource, please let us know as soon as possible.

1. Planar rooted binary trees by depth

A "planar rooted binary tree" is a directed acyclic graph on zero or more nodes which is either the empty graph, or is a graph with a distinguished node (the 'root') for which each node has a parent (unless it is the root), a left child, and a right child (which can both be empty). Recall that the number of planar rooted binary trees with n nodes is the Catalan number C_n, and that the generating function for the Catalan numbers $G(q) = \sum_n C_n q^n = 1 + q + 2q^2 + 5q^3 + \dots$ satisfies the functional equation

$$G(q) = 1 + qG(q)^2.$$

In this problem we'll refine this counting problem. Let

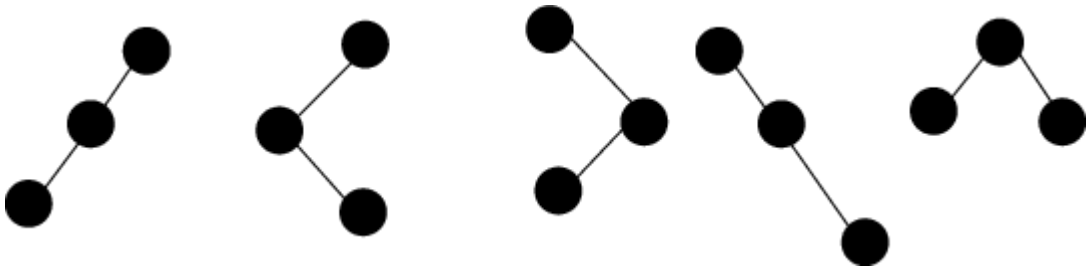
$$\begin{aligned} G(q,t) &= \sum_T q^{w(T)} t^{d(T)} \\ &= 1 + q + 2tq^2 + (t^2 + 4t^3)q^3 + \dots, \end{aligned}$$

where $w(T)$ is the number of nodes in the tree, and $d(T)$ is the *total depth* of the tree T. That is, if u is a node, we define $d(u)$, the depth of a node u, to be the length of the path from the root to u. With that definition,

$$d(T) = \sum_{u \in T} d(u),$$

where the sum runs over all $w(T)$ nodes of the tree.

For example: let's understand the $(t^2 + 4t^3)q^3$ term, which is the contribution from the five planar rooted binary trees on 3 nodes, depicted here (the root is the node at the top in each tree):



The first four trees each have the root at depth 0, with a depth 1 node below it, and a depth 2 node below that, so the total depth of each of these trees is 0+1+2=3 and thus each of these 4 trees contributes t^3q^3 to the generating function. The fifth tree has total depth 2, so its contribution to the generating function is t^2q^3.

(a) Find a functional equation satisfied by $G(q,t)$ which reduces to the ordinary one for $G(q)$ when we put $t = 1$.

(b) Explain how to use your answer for (a) to compute the series $G(q,t)$ out to order n in the variable q.

(c) Another way to compute $G(q,t)$ out to order n in the variable q is to brute-force it: generate all the planar rooted binary trees and make note of the total depth of each. Argue that the algorithm you gave in part (b) is in principle faster than the brute-force method, for large n.

Note: you don't need to submit any python code for this question - but if you do choose to implement these problems in code, please be aware that this is a surprisingly subtle problem to implement efficiently, and that the the time it takes your code to run may be misleading. The brute force algorithm of (c) might well be faster than doing the algebraic computation suggested in (b) for reasonable tree sizes, depending on how fast the computer algebra system you use is.

2. Infinite Scrabble

Here are the points which the English version of a popular word game assigns to each letter:

```
{'A': 1, 'B': 3, 'C': 3, 'D': 2, 'E': 1, 'F': 4, 'G': 2, 'H': 4, 'I': 1, 'J': 8,
 'K': 5, 'L': 1, 'M': 3,
 'N': 1, 'O': 1, 'P': 3, 'Q': 10, 'R': 1, 'S': 1, 'T': 1, 'U': 1, 'V': 4, 'W': 4,
 'X': 8, 'Y': 4, 'Z': 10}
```

Let $\mathcal{L}$ denote the 26 letters, and $W = \bigcup_{n\geq 0} \mathcal{L}^n$ the set of "words" of length 0 or greater. For $w \in W$, let $h(w)$ denote the "score" of that word, i.e., the sum of the points of each constituent letter. (*Note:* these need not be actual words.)

Denote the sum of $e^{-\beta h(w)}$ over all n-letter words by

$$\begin{aligned} Z_n(\beta) &= \sum_{w\in\mathcal{L}^n} e^{-\beta h(w)} \\ &= \left(e^{-\beta h(a)} + e^{-\beta h(b)} + \cdots + e^{-\beta h(z)}\right)^n. \end{aligned}$$

and the sum over *all* words by

$$\begin{aligned} Z(\beta) &= \sum_{n\geq 0} Z_n(\beta) \\ &= \left(1 - (e^{-\beta h(a)} + e^{-\beta h(b)} + \cdots + e^{-\beta h(z)})\right)^{-1}, \end{aligned}$$

which converges for $\beta > \beta_0 \approx 2.33$.

a) Define a discrete-time Markov chain on W whose unique stationary distribution is given by

$$\pi(w) = \frac{1}{Z(\beta)} \exp(-\beta h(w)),$$

for an arbitrary $\beta > \beta_0$. Show that the chain converges to the correct stationary distribution (you can appeal to general results about Markov chains).

b) Simulate at least 10^6 steps from this Markov chain at $\beta = 3$, and check that it gives approximately the right distribution of word lengths (e.g., length frequencies within 0.01 for lengths up to 10).

3. The immortal jellyfish

Here is a simplified model for the life of the "immortal jellyfish" (*Turritopsis dohrnii*). It can be in one of five states at any point in time: *planula*, *polyp*, *medusa*, *ball of cells*, or *dead*. Let us model the life of one such jelly as a continuous-time Markov chain L_t, with transition rates

$$\begin{aligned} \text{planula} &\longrightarrow \text{polyp} && \text{at rate} && 1 \\ \text{polyp} &\longrightarrow \text{medusa} && \text{at rate} && 1 \\ \text{medusa} &\longrightarrow \text{ball of cells} && \text{at rate} && 0.2 \\ \text{ball of cells} &\longrightarrow \text{polyp} && \text{at rate} && 1 \\ \text{medusa} &\longrightarrow \text{dead} && \text{at rate} && \mu. \end{aligned}$$

Every jelly begins as a planula, and the lifetime extends until death.

a) Let L_a denote the mean remaining lifetime of a jellyfish currently in life stage a. Explain how to find L_a, and prove correctness of your method (you may use basic properties of Markov chains). Implement the method in python.

b) Implement a simulation of the process in python.

c) Use (a) and (b) to make a plot of values of L_{polyp} against $0.1 \leq \mu \leq 2$, comparing values as computed from theory using your method in (a) to empirical values from at least 1,000 simulations per value of μ.

4. Brownian ants

Suppose there are N ants sitting on a meter stick at positions $0 < x_1 < x_2 < \cdots < x_N < 1$. They are startled, and each follows an independent Brownian motion along the stick until they reach an end, at which point they fall off. After this experiment, we label their initial locations according to *which end* they fell off; let $0 < a_1 < a_2 < \cdots < a_{N_a} < 1$ be the initial positions of those ants that fell off the left end of the stick, and $0 < b_1 < b_2 < \cdots < b_{N_b} < 1$ the remainder. So, $\mathcal{A} = \{a_k\}_{k=1}^{N_a}$ and $\mathcal{B} = \{b_k\}_{k=1}^{N_b}$ are disjoint, and their union is $\mathcal{X} = \{x_k\}_{k=1}^{N}$. Make sure to explain and justify any general results that you use in answering the questions below.

a) Find the probability that an ant that began at x falls off the left end of the stick (i.e., hits 0 before 1).

b) Suppose that $\mathcal{X}$ is a Poisson process with uniform intensity λdx. Show that $\mathcal{A}$ and $\mathcal{B}$ are independent Poisson processes with intensities $(1-x)\lambda dx$ and $x\lambda dx$, respectively.

c) Simulate this process, and make a plot where one axis is time and the other axis is position along the stick, with lines showing the trajectories of the ants, for $\lambda = 10$.

5. A perceptron learning rule

Consider a perceptron with activity rule $g(h) = \frac{1}{1+e^{-h}}$, where $h = \sum_{i=1}^{N} w_i \xi_i$ is the activation for an N-dimensional input pattern ξ_i. The training set includes p patterns $\{\xi_i^\mu, \zeta^\mu\}_{\mu=1,\dots,p}$, where ζ^μ are the targets.

(a): Derive the perceptron learning rule starting from the following cost function:

$$E = \frac{1}{2} \sum_{\mu=1}^{p} \left[\zeta^\mu - g(h^\mu)\right]^2 , \tag{1}$$

by applying gradient descent. Note that $\partial_h g = g(1-g)$.

(b): Add a weight decay term to the cost function and derive the new perceptron learning rule. Explain in words what the Bayesian interpretation of the weight decay term is, and what issue with training it may alleviate.

6. Supervised learning of a classification task with multilayer networks

Consider a multilayer network architecture to learn S different classes of N-dimensional input patterns. Let's fix the dimension of the input space to $N = 100$. In this problem, you will:

1. Generate and analyze a dataset for a classification task.
2. Train a multilayer network to perform the classification task.

Generate the dataset

You will generate and analyze the dataset following these steps. The dataset consists of S classes, and each class includes n input patterns.

- For each class μ, initialize the *parent* ξ_i^μ, for $i = 1, \ldots, N$ and $\mu = 1, \ldots, S$ as a random pattern with values $\{\pm 1\}$ and probability $p(\xi_i^\mu = +1) = 0.1$.
- Starting from the *parent* ξ_i^μ, now generate $n = 200$ *children* patterns in class μ such that each children is obtained from its parent ξ_i^μ by independently flipping each spin with probability $p_{flip} = 0.2$.
- At the end of the procedure, you will have a total of $S * n$ input patterns equally distributed in S classes, and target labels for each pattern equal to their class μ.
- Calculate the correlation between children in the same class, and between the parents in different classes (report the average correlation coefficient and its standard deviation).

Perform the classification task

- You will specialize to a dataset with $S = 8$ classes. Build a deep neural network with 2 layers (one hidden layer and output layer) which uses backpropagation to learn the classification task of the S classes from the input patterns. Fix the output layer to have S neurons with 'softmax' activations, and the hidden layer to have L neurons with 'relu' activation; choose categorical crossentropy as your cost function. You may use the 'adam' optimizer and an early stopping callback. You may use any feature for building the network and optimizing the weights, provided you justify your choice. The total number of inputs is $S * n$ and the target label for each input pattern is its class S. Remember that the format for the labels must be compatible with the cost function. Remember to include enough training epochs to achieve a reasonable convergence in the optimization.
- You will split your $S * n$ patterns into a training set, and a test set, the latter containing 33\% of the input patterns, equally balanced between different classes.
- Your goal is to train several different networks with different size L of the hidden layer. L will vary between 2 and $2 * S$. Because of the stochastic nature of stochastic gradient descent, for each value of L you may train $n_{repeat} = 5$ different networks.
- For an example network, plot the training history: the classification accuracy and the loss in the training and validation sets as a function of the training epoch, and on the test set at the end of training.
- Plot the classification accuracy on the test set as a function of the number of hidden layer neurons L, across all n_{repeat}. Explain in words the qualitative features of this result: How does the accuracy depend on L? What is the smallest value of L such that the performance reaches 90% accuracy.

Recommendation: You may design and train the network via tensorflow 2.0 with Keras, using the conda environment tf_2 we used in class.