CPSC 440/550 Advanced Machine Learning (Jan-Apr 2025)
Assignment 2 – due Tuesday March 25 at **11:59pm**

The assignment instructions are the same as for the previous assignment, but for this assignment you can work in groups of 1 or 2. Please only hand in one assignment for the group, and use the Gradescope group feature.

1 Categorical Inference [20 points]

Consider a categorical distribution that takes value 1 with probability 0.41, 2 with probability 0.17, and 3 with probability 0.42.

[1.1] [2 points] Suppose we generate four iid samples $\{x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}\}$ according to this model. What is the probability that all four samples are equal to 3?

Answer: TODO

[1.2] [2 points] Write a Python function that generates t iid examples from this distribution, based on numpy's `rng.random()` function (which returns uniformly pseudo-random numbers in $[0, 1]$) as the source of randomness. Do not use other random generation functions or functions designed specifically for this purpose, though "general-purpose" numpy routines are fine. Your implementation does not need to be particularly efficient. Hand in your code.

Answer: TODO

[1.3] [2 points] Consider a game where you get \$12 if a sample from this distribution is even, and lose \$2 if the sample is odd. What is the expected \$ value you get from playing this game once?

Answer: TODO

[1.4] [6 points] Supposing you weren't able to do that math, one way to approximate the answer of the previous question is to take

$$\frac{1}{t}\sum_{i=1}^{t} f(x^{(i)}),$$

where each $x^{(i)}$ is an iid sample, and f gives the \$ value for that sample (either 12 or -2).

Implement this approximation, and use the code in `expval_mc` (inside `main.py`) to plot the following: draw a line of the "running mean" of the approximation as you see more samples, going from 1 to 100,000 samples. Show three independent runs on the same plot, in different colours (just calling `ax.plot` more than once and using its default colour cycle is fine). Also use `ax.axhline` to show the analytical expected value from the previous part. Hand in this plot.

Answer: TODO

[1.5] [8 points] Now, suppose that you start playing the game with \$30. You'd really like to turn that into \$250 to have a night on the town, but if you ever can't pay your debt (that is, your balance goes below \$0) then you need to go on the lam. What's the probability that you'll be able to have a fun night? Answer to within 1%, i.e. if the answer were exactly 21.34% (it's not) then it's okay if your answer is anywhere between 20.34% and 22.34%. Hand in your code and a justification that your answer is accurate enough.

You could use Monte Carlo, or you could compute it analytically, or various other approaches. Remember not to look up specific homework problems online!

Answer: TODO

2 Maximizing a simple expectation [20 points]

[2.1] [4 points] When we derived the EM algorithm for a general $p(\mathbf{X}, \mathbf{Z} \mid \theta)$ with a variational distribution q (depending on $\mathbf{X}$ and θ) over the variable $\mathbf{Z}$, we used

$$\mathrm{ELBO}_q(\mathbf{X}) = \mathbb{E}_{\mathbf{Z}\sim q} \log p(\mathbf{X}, \mathbf{Z} \mid \theta) + \mathrm{Entropy}(q).$$

When deriving VAEs, we instead used a per-sample ELBO with a variational distribution q specifically for the Z corresponding to a given X:

$$\mathrm{ELBO}_{q_x}(x) = \mathbb{E}_{Z\sim q_x} \log p(x, Z \mid \theta) + \mathrm{Entropy}(q_x).$$

Show that if $(x^{(i)}, z^{(i)})$ are iid under p and $q(\mathbf{Z}) = \prod_{i=1}^n q_i(z^{(i)})$, then $\mathrm{ELBO}_q(\mathbf{X}) = \sum_{i=1}^n \mathrm{ELBO}_{q_i}(x^{(i)})$. You may assume that z is discrete, if you prefer, although it won't fundamentally change the proof.

Answer: TODO

Now, let's consider a particular problem: the one-dimensional data distribution

$$Z \sim \mathrm{Bern}(1/2) \qquad X \mid (Z = z) \sim \mathcal{N}(\mu z, 1)$$

so that the marginal distribution of X is simply $\frac{1}{2}\mathcal{N}(0, 1) + \frac{1}{2}\mathcal{N}(\mu, 1)$. Given n iid samples $\mathbf{X} = (x^{(1)}, \dots, x^{(n)})$ from X, we'd like to estimate the single parameter μ.

Here $z^{(i)}$ is a binary random variable; we can therefore say that $q_i = \mathrm{Bern}(r^{(i)})$, for a parameter $r^{(i)} \in [0, 1]$. Let $\psi(t) = -t \log t - (1 - t) \log(1 - t)$ be a function defined on $(0, 1)$.

[2.2] [4 points] Show that, given parameters μ and $r^{(i)}$, we have in this problem that

$$\mathrm{ELBO}_{q_i}(x^{(i)}) = -\frac{1}{2}\log(8\pi) - \frac{1}{2}\left(x^{(i)}\right)^2 + \mu r^{(i)} x^{(i)} - \frac{1}{2}\mu^2 r^{(i)} + \psi\left(r^{(i)}\right). \tag{2.1}$$

Answer: TODO

[2.3] [3 points] In the argument deriving the E step of EM, we appealed to the KL divergence to show that maximizing the ELBO matches q_i to the conditional distribution of $Z^{(i)} \mid x^{(i)}$, which in this case will be the Bernoulli with parameter equal to the responsibility

$$\begin{aligned}
r^{(i)} &= p(Z^{(i)} = 1 \mid \theta, x^{(i)}) \\
&= \frac{p(Z^{(i)} = 1)p(x^{(i)} \mid Z^{(i)} = 1, \theta)}{p(Z^{(i)} = 0)p(x^{(i)} \mid Z^{(i)} = 0, \theta) + p(Z^{(i)} = 1)p(x^{(i)} \mid Z^{(i)} = 1, \theta)} \\
&= \frac{\frac{1}{2}p(x^{(i)} \mid Z^{(i)} = 1, \theta)}{\frac{1}{2}p(x^{(i)} \mid Z^{(i)} = 0, \theta) + \frac{1}{2}p(x^{(i)} \mid Z^{(i)} = 1, \theta)} \\
&= \frac{e^{-\frac{1}{2}(x^{(i)}-\mu)^2}}{e^{-\frac{1}{2}(x^{(i)})^2} + e^{-\frac{1}{2}(x^{(i)}-\mu)^2}} \\
&= \frac{1}{1 + e^{-\mu x^{(i)} + \frac{1}{2}\mu^2}} = \sigma\left(\mu x^{(i)} - \frac{1}{2}\mu^2\right),
\end{aligned}$$

where $\sigma(t) = 1/(1 + \exp(-t))$ is the logistic sigmoid function. *Checking that this result makes sense: as $\mu x^{(i)} \to \infty$, $r^{(i)} \to 1$; as $\mu x^{(i)} \to -\infty$, $r^{(i)} \to 0$; if $x^{(i)} = \mu/2$, then $r^{(i)} = \frac{1}{2}$.*

Show directly, without appealing to the responsibility or the KL divergence argument, that this is indeed the same value which maximizes $\mathrm{ELBO}_{q_i}(x^{(i)})$ for a given μ.

Answer: TODO

[2.4] [3 points] Show that, for fixed $r^{(1)}, \dots, r^{(n)}$, the $\mathrm{ELBO}_q(\mathbf{X})$ is maximized when $\mu = \frac{\sum_i r^{(i)} x^{(i)}}{\sum_i r^{(i)}}$.

Answer: TODO

[2.5] [6 points] The function `plot_em` in `main.py` loads some data sampled from this distribution, with some particular value for μ. Fill in the code to compute the `log_liks` array and to run EM; hand in your code and the resulting figure, which should show EM quickly converging towards the maximum of this 1d likelihood function.

Answer: TODO

3 Naive Bayes [25 points]

Let's fit the following naive Bayes model on binarized MNIST:

$$Y \sim \mathrm{Cat}(\theta) \qquad \theta_j \sim \mathrm{Beta}(\alpha, \beta) \qquad X_j \mid (Y = y, \theta_j) \sim \mathrm{Bern}(\theta_{j|y}).$$

(We don't really need to put a prior on the Y parameter θ, since we have plenty of samples there, but if we don't use a prior on θ_j we'll get some `NaN`s.)

[3.1] [10 points] Implement MAP in this model in `naive_bayes.py`. Hand in your code, and the output of `python main.py mnist-nb`, which tests with several different choices for $\alpha = \beta$.

Answer: TODO

This model is okay, but not great. How much better can we expect to get? `python main.py mnist-logreg` will fit a multiclass logistic regression (aka softmax regression) model from scikit-learn, which gets about 7.6% error, less than half the error rate of naive Bayes. (Various nonlinear models can get almost zero error on this problem.) So, let's try a way to make naive Bayes less naive, which will substantially improve its performance.

It seems reasonable that there would be clusters in the classes. For example, there might are several different "general ways" to draw the digit 7. Instead of assuming that the features are independent given the class label, we might instead assume they're independent given the *cluster* that they are in.

We could treat this all probabilistically, but that'd entail a somewhat complicated EM training loop. Instead, consider a *vector-quantized naive Bayes* (VQNB) implementing a version of this idea:

- It clusters the examples associated with *each digit* into k clusters, using k-means clustering (k clusters for each class, $10k$ clusters total). We'll use $z^{(i)}$ as the cluster number of example i. The value $z^{(i)}$ will be from 1 to k, with $y^{(i)}$ determining whether we are considering the k "0" clusters, the k "1" clusters, or so on.
- Since we don't know $z^{(i)}$ (it is a "latent variable"), we can marginalize over its possible values. Using the marginalization and then the product rule, we can write the joint probability as

$$\begin{aligned}
p(x_1^{(i)}, x_2^{(i)}, \ldots, x_d^{(i)}, y^{(i)}) &= \sum_{z=1}^{k} p(x_1^{(i)}, x_2^{(i)}, \ldots, x_d^{(i)}, y^{(i)}, z^{(i)} = z) \\
&= \sum_{z=1}^{k} p(x_1^{(i)}, x_2^{(i)}, \ldots, x_d^{(i)} \mid y^{(i)}, z^{(i)} = z)\, p(y^{(i)}, z^{(i)} = z) \\
&= \sum_{z=1}^{k} p(x_1^{(i)}, x_2^{(i)}, \ldots, x_d^{(i)} \mid y^{(i)}, z^{(i)} = z)\, p(z^{(i)} = z \mid y^{(i)})\, p(y^{(i)}) \\
&= p(y^{(i)}) \sum_{z=1}^{k} p(z \mid y^{(i)}) \left[\prod_{j=1}^{d} p(x_j^{(i)} \mid y^{(i)}, z) \right];
\end{aligned}$$

  the last line uses independence of the features given the cluster.

[3.2] [12 points] Implement the VQNB method (there's a stub in `naive_bayes.py`). Hand in your code, and the test error you obtain with this model for $k = 2$ through $k = 5$.

Hint: The same KMeans class as last time is in the handout code for you to use, or feel free to use `scikit-learn`'s.

Hint: $p(y)$ you can handle just as before. $p(z \mid y)$ is a categorical variable. $p(x_j \mid y, z)$ is Bernoulli; you'll have one Bernoulli parameter for each (y, z) pair, which it might be convenient to organize in a $10 \times k$ array.

Hint: If you're working with log-probabilities (which is a good idea), `scipy.special.logsumexp` *might be helpful for the sum operation.*

Hint: To help with debugging, note that you should get the naive Bayes model in the special case of $k = 1$. Further, with this type of model you usually see the biggest performance gain when going from $k = 1$ to $k = 2$.

Hint: You probably won't be able to exactly match the performance of logistic regression, but it'll get closer.

Answer: TODO

[3.3] [3 points] For a run of the method with $k = 5$, show the images obtained by plotting the estimates for $p(x_j = 1 \mid z, y)$ for all j as a 28 by 28 image, for each value of z and y (so there should be 50 images). There's a code stub in there for plotting it, just fill that out slightly.

Answer: TODO

4 Neural Networks [35 points]

[4.1] [8 points] `main.py nn-regression` runs a stochastic gradient method to train a neural network on the `basis_data` dataset from a previous assignment. However, in its current form it doesn't fit the data very well. Modify the training procedure and model to improve the performance of the neural network. Hand in your plot after changing the code to have better performance, and list the changes you made.

Hint: There are many possible strategies you could take to improve performance. Below are some suggestions, but note that the some will be more effective than others:

- *Changing the network structure (`hidden_layer_sizes` gives the number of hidden units per layer).*
- *Changing the training procedure (you can change the stochastic gradient step-size, use decreasing step sizes, use mini-batches, run it for more iterations, add momentum, switch to `findMin`, use Adam, and so on).*
- *Transform the data by standardizing the features, standardizing the targets, and so on.*
- *Add regularization (L2-regularization, L1-regularization, dropout, and so on).*
- *Change the initialization.*
- *Add bias variables.*
- *Change the loss function or the non-linearities (right now it uses squared error and* tanh *to introduce non-linearity).*
- *Use mini-batches of data, possibly with batch normalization.*

Answer: TODO

Okay, that's enough of our hand-coded neural net. Time for some autodiff!

We're going to use PyTorch; see `https://pytorch.org` for installation instructions (the CPU version is fine; if you're having trouble, you could also use Google Colab). This tutorial page is a decent starting place for reference.

[4.2] [3 points] For the provided code in `TorchNeuralNetClassifier(layer_sizes=[3])`, how many parameters are there, and what do they represent?

Hint: `list(thing.parameters())` will get you the parameters of either a layer or a `torch.nn.Module`.

Answer: TODO

[4.3] [4 points] Modify the network to use a more typical classification loss: maximizing the likelihood of a categorical likelihood, aka softmax loss, aka cross-entropy. Hand in the modifications to your code.

You don't need to implement it yourself from scratch – you can use anything from PyTorch here.

Answer: TODO

[4.4] [3 points] PyTorch doesn't have an easy built-in way to compute the loss after going through a `torch.nn.Softmax` layer. Why not? Why does it insist on log inputs? Describe an example where using log-probabilities would give meaningfully different results from "plain" probabilities.

Answer: TODO

[4.5] [7 points] Change `TorchNeuralNetClassifier` to work well on MNIST dataset – that is, at least match logistic regression (error rate 7.5%), but you can probably do better. You'll probably do similar stuff to Question [4.1]. Describe the changes you made and your best test performance.

Answer: TODO

[4.6] [7 points] Change the model in `Convnet.build` to use convolutional layers, while getting test error at least (approximately) as good as the MLP got. (Feel free to make it less generic, e.g. not supporting arbitrary `layer_sizes` anymore.) Submit your `build()` method, and any other relevant changes, as well as your final test error. (We're probably overfitting a bit at this point, though. . . .)

Hint: To make it a little faster by using your GPU, you can pass `device="cuda"` if you have an appropriate version of PyTorch installed, or you can try `device="mps"` if you're on a recent Mac.

Answer: TODO

[4.7] [3 points] If we use a layer `torch.nn.Conv2d(in_channels=1, out_channels=128, kernel_size=(4, 4))`, what are the resulting parameter shapes, and why? Why don't we need to pass in the shape of the input image?

Answer: TODO