Homework 3

Deadline: Wednesday, Nov. 3, at 11:59pm.

Submission: You will need to submit three files:

- Your answers to all of the questions, as a PDF file titled `hw3_writeup.pdf`. You can produce the file however you like (e.g. LaTeX, Microsoft Word, scanner), as long as it is readable. If you need to split your writeup into multiple files, that's OK, as long as we can figure out what you did.
- The completed Python files `naive_bayes.py` and `q4.py`.

Neatness Point: One point will be given for neatness. You will receive this point as long as we don't have a hard time reading your solutions or understanding the structure of your code.

Late Submission: 10% of the marks will be deducted for each day late, up to a maximum of 3 days. After that, no submissions will be accepted.

Homeworks are individual work. See the Course Information handout[1] for detailed policies.

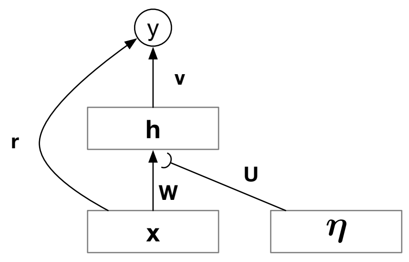
1. [**5pts**] **Backprop**

   In this question, you will derive the backprop updates for a particular neural net architecture. The network is similar to the multilayer perceptron architecture from lecture, and has one linear hidden layer. However, there are two architectural differences:

   - In addition to the usual vector-valued input $\mathbf{x}$, there is a vector-valued "context" input $\boldsymbol{\eta}$. (The particular meaning of $\boldsymbol{\eta}$ isn't important for your derivation, but think of it as containing additional task information, such as whether to focus on the left or the right half of the image.) The hidden layer activations are *modulated* based on $\boldsymbol{\eta}$; this means they are multiplied by a value which depends on $\boldsymbol{\eta}$.
   - The network has a *skip connection* which sends information directly from the input to the output of the network.

   The loss function is squared error. The forward pass equations and network architecture are as follows (the symbol $\odot$ represents elementwise multiplication, and σ denotes the logistic function):

$$
\begin{aligned}
\mathbf{z} &= \mathbf{W}\mathbf{x} \\
\mathbf{s} &= \mathbf{U}\boldsymbol{\eta} \\
\mathbf{h} &= \mathbf{z} \odot \sigma(\mathbf{s}) \\
y &= \mathbf{v}^\top \mathbf{h} + \mathbf{r}^\top \mathbf{x} \\
\mathcal{L} &= \frac{1}{2}(y-t)^2
\end{aligned}
$$



   The model parameters are matrices $\mathbf{W}$ and $\mathbf{U}$, and vectors $\mathbf{v}$ and $\mathbf{r}$. Note that there is only one output unit, i.e. y and t are scalars.

[1] `http://www.cs.toronto.edu/~rgrosse/courses/csc311_f21/syllabus.pdf`

(a) **[1pt]** Draw the computation graph relating $\mathbf{x}$, $\mathbf{z}$, $\boldsymbol{\eta}$, $\mathbf{s}$, $\mathbf{h}$, and the model parameters.

(b) **[4pts]** Derive the backprop formulas to compute the error signals for all of the model parameters, as well as $\overline{\mathbf{x}}$ and $\overline{\boldsymbol{\eta}}$. Also include the backprop formulas for all intermediate quantities needed as part of the computation. You may leave the derivative of the logistic function as σ' rather than expanding it out explicitly.

2. **[13pts] Fitting a Naïve Bayes Model**

In this question, we'll fit a Naïve Bayes model to the MNIST digits using maximum likelihood. In addition to the mathematical derivations, you will complete the implementation in `naive_bayes.py`. The starter code will download the dataset and parse it for you: Each training sample $(\mathbf{t}^{(i)}, \mathbf{x}^{(i)})$ is composed of a vectorized binary image $\mathbf{x}^{(i)} \in \{0,1\}^{784}$, and 1-of-10 encoded class label $\mathbf{t}^{(i)}$, i.e., $t_c^{(i)} = 1$ means image i belongs to class c.

Given parameters $\boldsymbol{\pi}$ and $\boldsymbol{\theta}$, Naïve Bayes defines the joint probability of the each data point $\mathbf{x}$ and its class label c as follows:

$$p(\mathbf{x}, c \mid \boldsymbol{\theta}, \boldsymbol{\pi}) = p(c \mid \boldsymbol{\theta}, \boldsymbol{\pi}) p(\mathbf{x} \mid c, \boldsymbol{\theta}, \boldsymbol{\pi}) = p(c \mid \boldsymbol{\pi}) \prod_{j=1}^{784} p(x_j \mid c, \theta_{jc}).$$

where $p(c \mid \boldsymbol{\pi}) = \pi_c$ and $p(x_j = 1 \mid c, \boldsymbol{\theta}, \boldsymbol{\pi}) = \theta_{jc}$. Here, $\boldsymbol{\theta}$ is a matrix of probabilities for each pixel and each class, so its dimensions are 784×10, and $\boldsymbol{\pi}$ is a vector with one entry for each class. (Note that in the lecture, we simplified notation and didn't write the probabilities conditioned on the parameters, i.e. $p(c|\boldsymbol{\pi})$ is written as $p(c)$ in lecture slides).

For binary data ($x_j \in \{0,1\}$), we can write the Bernoulli likelihood as

$$p(x_j \mid c, \theta_{jc}) = \theta_{jc}^{x_j} (1 - \theta_{jc})^{(1-x_j)}, \tag{1}$$

which is just a way of expressing $p(x_j = 1|c, \theta_{jc}) = \theta_{jc}$ and $p(x_j = 0|c, \theta_{jc}) = 1 - \theta_{jc}$ in a compact form. For the prior $p(\mathbf{t} \mid \boldsymbol{\pi})$, we use a categorical distribution (generalization of Bernoulli distribution to multi-class case),

$$p(t_c = 1 \mid \boldsymbol{\pi}) = p(c \mid \boldsymbol{\pi}) = \pi_c \quad \text{or equivalently} \quad p(\mathbf{t} \mid \boldsymbol{\pi}) = \Pi_{j=0}^{9} \pi_j^{t_j} \quad \text{where} \quad \sum_{i=0}^{9} \pi_i = 1,$$

where $p(c \mid \boldsymbol{\pi})$ and $p(\mathbf{t} \mid \boldsymbol{\pi})$ can be used interchangeably. You will fit the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$ using MLE and MAP techniques. In both cases, your fitting procedure can be written as a few simple matrix multiplication operations.

(a) **[3pts]** First, derive the *maximum likelihood estimator* (MLE) for the class-conditional pixel probabilities $\boldsymbol{\theta}$ and the prior $\boldsymbol{\pi}$. Derivations should be rigorous.

Hint 1: We saw in lecture that MLE can be thought of as 'ratio of counts' for the data, so what should $\hat{\theta}_{jc}$ be counting?

Hint 2: Similar to the binary case, when calculating the MLE for π_j for $j = 0, 1, ..., 8$, write $p(\boldsymbol{t}^{(i)} \mid \boldsymbol{\pi}) = \Pi_{j=0}^{9} \pi_j^{t_j^{(i)}}$ and in the log-likelihood replace $\pi_9 = 1 - \Sigma_{j=0}^{8} \pi_j$, and then take derivatives w.r.t. π_j. This will give you the ratio $\hat{\pi}_j / \hat{\pi}_9$ for $j = 0, 1, .., 8$. You know that $\hat{\pi}_j$'s sum up to 1.

(b) **[1pt]** Derive the log-likelihood $\log p(\mathbf{t}|\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\pi})$ for a single training image.

(c) **[3pt]** Fit the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$ using the training set with MLE, and try to report the average log-likelihood per data point $\frac{1}{N}\Sigma_{i=1}^{N}\log p(\mathbf{t}^{(i)}|\mathbf{x}^{(i)},\hat{\boldsymbol{\theta}},\hat{\boldsymbol{\pi}})$, using Equation (1). What goes wrong? (it's okay if you can't compute the average log-likelihood here).

(d) **[1pt]** Plot the MLE estimator $\hat{\boldsymbol{\theta}}$ as 10 separate greyscale images, one for each class.

(e) **[2pt]** Derive the *Maximum A posteriori Probability* (MAP) estimator for the class-conditional pixel probabilities $\boldsymbol{\theta}$, using a Beta(3, 3) prior on each θ_{jc}. Hint: it has a simple final form, and you can ignore the Beta normalizing constant.

(f) **[2pt]** Fit the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$ using the training set with MAP estimators from previous part, and report both the average log-likelihood per data point, $\frac{1}{N}\Sigma_{i=1}^{N}\log p(\mathbf{t}^{(i)}|\mathbf{x}^{(i)},\hat{\boldsymbol{\theta}},\hat{\boldsymbol{\pi}})$, and the accuracy on both the training and test set. The accuracy is defined as the fraction of examples where the true class is correctly predicted using $\hat{c} = \text{argmax}_c \log p(t_c = 1|\mathbf{x},\hat{\boldsymbol{\theta}},\hat{\boldsymbol{\pi}})$.

(g) **[1pt]** Plot the MAP estimator $\hat{\boldsymbol{\theta}}$ as 10 separate greyscale images, one for each class.

3. **[7pts] Categorial Distribution.** In this problem you will consider a Bayesian approach to modelling categorical outcomes. Let's consider fitting the categorical distribution, which is a discrete distribution over K outcomes, which we'll number 1 through K. The probability of each category is explicitly represented with parameter θ_k. For it to be a valid probability distribution, we clearly need $\theta_k \geq 0$ and $\sum_k \theta_k = 1$. We'll represent each observation $\mathbf{x}$ as a 1-of-K encoding, i.e, a vector where one of the entries is 1 and the rest are 0. Under this model, the probability of an observation can be written in the following form:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \prod_{k=1}^{K} \theta_k^{x_k}.$$

Suppose you observe a dataset,

$$\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}.$$

Denote the count for outcome k as $N_k = \sum_{i=1}^{n} x_k^{(i)}$. Recall that each data point is in the 1-of-K encoding, i.e., $x_k^{(i)} = 1$ if the ith datapoint represents an outcome k and $x_k^{(i)} = 0$ otherwise. In the previous assignment, you showed that the maximum likelihood estimate for the counts was:

$$\hat{\theta}_k = \frac{N_k}{N}.$$

(a) **[2pts]** For the prior, we'll use the Dirichlet distribution, which is defined over the set of probability vectors (i.e. vectors that are nonnegative and whose entries sum to 1). Its PDF is as follows:

$$p(\boldsymbol{\theta}) \propto \theta_1^{a_1-1}\cdots\theta_K^{a_K-1}.$$

Determine the posterior distribution $p(\boldsymbol{\theta}\,|\,\mathcal{D})$. Based on your answer, is the Dirichlet distribution a conjugate prior for the categorial distribution?

(b) **[3pts]** Still assuming the Dirichlet prior distribution, determine the MAP estimate of the parameter vector $\boldsymbol{\theta}$. For this question, you may assume each $a_k > 1$.

Hint: Remember that you need to enforce the constraint that $\sum_k \theta_k = 1$. You can do this using the same parameterization trick you used in Question 2. Alternatively, you could use Lagrange multipliers, if you're familiar with those.

(c) **[2pts]** Now, suppose that your friend said that they had a hidden $N + 1$st outcome, $\mathbf{x}^{(N+1)}$, drawn from the same distribution as the previous N outcomes. Your friend does not want to reveal the value of $\mathbf{x}^{(N+1)}$ to you. So, you want to use your Bayesian model to predict what *you* think $\mathbf{x}^{(N+1)}$ is likely to be. The "proper" Bayesian predictor is the so-called *posterior predictive distribution*:

$$p(\mathbf{x}^{(N+1)}|\mathcal{D}) = \int p(\mathbf{x}^{(N+1)}|\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D})\,d\boldsymbol{\theta}$$

What is the probability that the $N+1$ outcome was k, i.e., the probability that $x_k^{(N+1)} = 1$, under your posterior predictive distribution? *Hint: A useful fact is that if $\boldsymbol{\theta} \sim \text{Dirichlet}(a_1, \ldots, a_K)$, then*

$$\mathbb{E}[\theta_k] = \frac{a_k}{\sum_{k'} a_{k'}}.$$

Report your answers to the above questions.

4. **[5pts] Gaussian Discriminant Analysis.** For this question you will build classifiers to label images of handwritten digits. Each image is 8 by 8 pixels and is represented as a vector of dimension 64 by listing all the pixel values in raster scan order. The images are grayscale and the pixel values are between 0 and 1. The labels y are $0, 1, 2, \ldots, 9$ corresponding to which character was written in the image. There are 700 training cases and 400 test cases for each digit; they can be found in the `data` directory in the starter code.

A skeleton (`q4.py`) is is provided for each question that you should use to structure your code. Starter code to help you load the data is provided (`data.py`). Note: the `get_digits_by_label` function in `data.py` returns the subset of digits that belong to a given class.

Using maximum likelihood, fit a set of 10 class-conditional Gaussians with a separate, full covariance matrix for each class. Remember that the conditional multivariate Gaussian probability density is given by,

$$p(\mathbf{x}\,|\,y = k, \boldsymbol{\mu}, \Sigma_k) = (2\pi)^{-d/2}|\Sigma_k|^{-1/2} \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T\Sigma_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)\right\} \tag{2}$$

You should take $p(y = k) = \frac{1}{10}$. You will compute parameters μ_{kj} and Σ_k for $k \in (0...9), j \in (1...64)$. You should implement the covariance computation yourself (i.e. without the aid of `np.cov`). *Hint: To ensure numerical stability you may have to add a small multiple of the identity to each covariance matrix. For this assignment you should add* $0.01\mathbf{I}$ *to each matrix.*

(a) **[3pts]** Using the parameters you fit on the training set and Bayes rule, compute the average conditional log-likelihood, i.e. $\frac{1}{N}\sum_{i=1}^{N} \log(p(y^{(i)}\,|\,\mathbf{x}^{(i)}, \theta))$ on both the train and test set and report it. *Hint: for numerical stability, you will want to use* ***`np.logaddexp`****, as discussed in Lecture 4.*

(b) **[1pt]** Select the most likely posterior class for each training and test data point as your prediction, and report your accuracy on the train and test set.

(c) **[1pt]** Compute the leading eigenvectors (largest eigenvalue) for each class covariance matrix (can use `np.linalg.eig`) and plot them side by side as 8 by 8 images.

Report your answers to the above questions, and submit your completed Python code for `q4.py`.