15780: GRADUATE AI (SPRING 2019)

Homework 3: Deep Learning and Probabilistic Modelling

Release: March 21, 2019,
Due: April 4, 2019, 5:00pm

We will use scientific Python for the implementation portions in this course. If you have not used Python before, we recommend downloading the Anaconda distribution (`https://www.continuum.io/downloads`) and looking through introductory resources like Google's Python Class (`https://developers.google.com/edu/python/`) and the Python Beginner's Guide (`https://wiki.python.org/moin/BeginnersGuide`). If you have not used scientific Python before, we recommend following introductions to NumPy (`http://www.numpy.org/`) and matplotlib (`http://matplotlib.org/`).

Please make sure your code submission works with the grading environment, which uses Python 3.7 and numpy 1.15.0.

1 Probability (20 points) [Ivan]

1.1 Probabilistic warm-up

1. **[3 points]** Construct 3 random variables X_1, X_2 and Z such that X_1 and X_2 are independent given Z, but are NOT marginally independent (you need to formally show this).
2. **[3 points]** Construct 3 random variables X_1, X_2 and Z such that X_1 and X_2 are marginally independent, but are NOT independent given Z (you need to formally show this).
3. **[3 points]** Prove that if X_1 and X_2 are independent, then $\text{Cov}(X_1, X_2) = 0$. Let now Y_1 and Y_2 be random variables such that $\text{Cov}(Y_1, Y_2) = 0$, can we conclude that Y_1 and Y_2 are independent? If yes, provide a proof, if no, provide a counterexample.

   Clarification for 3:[1] for this question feel free to assume that all random variables are discrete and can take only finite set of values.

[1] clarification was added

1.2 Regressions

Consider a linear regression model:

$$y^{(i)} = \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)} + \varepsilon^{(i)},$$

where $\varepsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$. Assume we observe m triples $(y^{(1)}, x_1^{(1)}, x_2^{(1)}), \ldots, (y^{(m)}, x_1^{(m)}, x_2^{(m)})$.

1. **[3 points]** Show that MLE estimate of $\beta = (\beta_1, \beta_2)$ coincides with the solution of least squares problem:

$$\text{minimize}_\beta \frac{1}{2}\|X\beta - y\|_2^2$$

Hint: Recall that MLE chooses the parameter that maximizes the probability of observing the data.

2. **[4 points]** Now assume that we put a prior on coefficients:

$$\beta_1 \sim \mathcal{L}(0, \gamma) \qquad \beta_2 \sim \mathcal{L}(0, \gamma),$$

where $\mathcal{L}(0, \gamma)$ is the Laplace distribution with the density:

$$p(x) = \frac{1}{2\gamma} \exp\left\{-\frac{|x|}{\gamma}\right\}.$$

Show that MAP estimate of $\beta = (\beta_1, \beta_2)$ coincides with the solution of the following regularized least squares problem

$$\text{minimize}_\beta \left(\frac{1}{2}\|X\beta - y\|_2^2 + \frac{\lambda}{2}\|\beta\|_1\right)$$

for some value of λ. Specify that value of λ.

Hint: Recall that MAP estimates maximizes the posterior probability $p(\beta|\text{Data})$.

1.3 MLE vs MAP

1. **[4 points]** Let $X_1, \ldots, X_n$ be i.i.d. from Bern(θ) (Bern(θ) is a Bernoulli distribution with probability of success θ) and assume we want to estimate θ. Show that if we put uniform prior on θ:

$$\theta \sim \mathcal{U}([0, 1]),$$

then MLE will coincide with MAP.

Hint: Use the Bayes rule.

2 Playing Bayes-Ball with LDA (45 points) [Ivan]

In this section, we will understand conditional independencies between variables in a Bayesian network. Specifically, we will consider the Bayesian network corresponding to the Latent Dirichlet Allocation (LDA) generative model for documents.

The LDA Model. We want to model M documents where each document (indexed as $i = 1, \dots M$) is represented as a sequence of N words $w_{i,j}$ for $j = 1, \dots, N$. Assume that each word belongs to a dictionary of D words. Under the LDA model, we assume there also exists a set of K topics, where each topic defines a distribution over words. Furthermore, each document corresponds to a distribution over topics. We first 'generate' these topic distributions for each document and the word distributions for each topic, and then 'generate' the words in the document using these distributions. To generate these distributions, we fix some prior distributions $\mathcal{D}_{\text{topics}}$ and $\mathcal{D}_{\text{words}}$ over the spaces $\{(c_1, c_2, \dots, c_K) | \sum_{k=1}^{K} c_k = 1,\ c_k \in [0,1]\ \forall k\}$ and $\{(c_1, c_2, \dots, c_D) | \sum_{d=1}^{D} c_d = 1,\ c_d \in [0,1]\ \forall d\}$ respectively[2]. Then:

- For each document $i = 1, \dots M$, we sample $\theta_i \sim \mathcal{D}_{\text{topics}}$, that defines the parameters of the topic distribution for that document.
- For each topic $k = 1, \dots K$, we sample $\phi_k \sim \mathcal{D}_{\text{words}}$, that defines the parameters of the word distribution for that topic.

Having generated the above two sets of parameters, we generate word $w_{i,j}$ as follows:

- Sample the topic of word $w_{i,j}$ (denoted by $z_{i,j}$) from the categorical distribution over topics $\{1, \dots K\}$ (defined by the topic distribution of document i), i.e. Categorical(θ_i).
- Sample $w_{i,j}$ from the categorical distribution over the D words in the dictionary (defined by the word distribution of topic $z_{i,j}$), i.e. Categorical($\phi_{z_{i,j}}$).

Bayesian network for the LDA model. Observe that the joint probability distribution for the random variables in the LDA model can be written as:

$$p(\theta, z, w, \phi) = \left(\prod_k p(\phi_k)\right)\left(\prod_i p(\theta_i)\right)\left(\prod_{i,j} p(z_{i,j} \mid \theta_i)\right)\left(\prod_{i,j} p(w_{i,j} \mid z_{i,j}, \phi)\right), \tag{1}$$

where each of the conditional probabilities has been defined by the generative model. Observe that this factorization corresponds to a Bayesian network, say G_{LDA}, with a node corresponding to each random variable in the model. In this problem, we will study the conditional independencies represented by this Bayesian network graph G_{LDA}.

Bayes-Ball Algorithm. The Bayes-Ball algorithm is a technique to infer conditional independencies between the variables in a Bayesian network. Consider a Bayesian network G over a set of variables $X_1, X_2, \dots, X_T$. Assume we have observed a subset of variables, $\mathcal{K} = \{X_i : X_i \text{ is observed}\}$. Let $\mathcal{J}$

[2] In the actual LDA model, $\mathcal{D}_{\text{topics}}$ and $\mathcal{D}_{\text{words}}$ are both Dirichlet distributions, thus giving the model its name. However, for the purposes of this question, we will keep it a bit more general.

be a subset of the unobserved variables. The Bayes-Ball algorithm finds the set $\mathcal{I}$ of all variables that are conditionally independent of $\mathcal{J}$ according to the Bayesian network graph G[3], given that the variables in $\mathcal{K}$ are observed i.e., the set $\mathcal{I} = \{X_i : X_i \perp X_J | \mathcal{K}\}$. The algorithm places a ball at each of the nodes[4] in $\mathcal{J}$ and sends these around to other nodes in the graph according to some rules. The unobserved variables in the graph not in $\mathcal{J}$ that never received the ball during the Bayes-Ball game are added to $\mathcal{I}$.

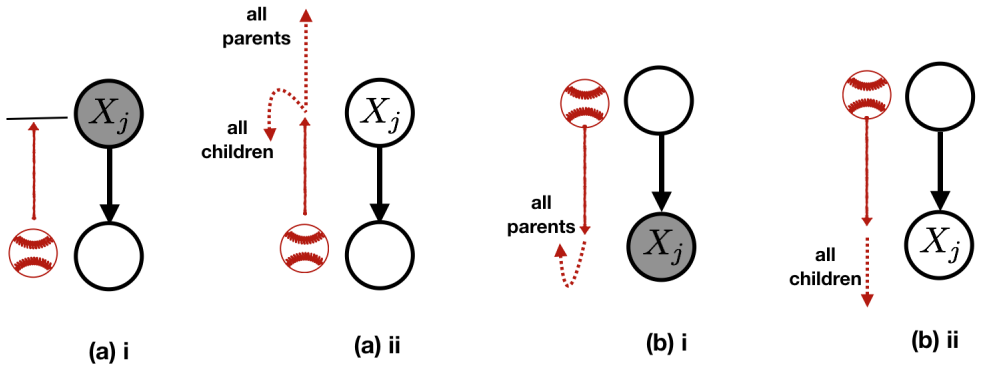

Figure 1: Illustration of the Bayes-Ball game: Observed nodes are shaded in grey.

The algorithm is as follows (illustrated in Figure 1):

1. Every node that belongs to $\mathcal{J}$ receives a ball (as if the ball was sent by one of its children[5]).
2. When a node X_j receives a ball:
   (a) If the ball was sent by a child of X_j:
      i. If $X_j \in \mathcal{K}$, do nothing (i.e., the ball never moves from there!).
      ii. If $X_j \notin \mathcal{K}$, send a copy of the ball to all the parents and all the children of X_j (including the node that sent this ball!).
   (b) If the ball was sent by a parent of X_j:
      i. If $X_j \in \mathcal{K}$, then send a copy of the ball to each of X_j's parents (including the parent who sent it!).
      ii. If $X_j \notin \mathcal{K}$, then send a copy of the ball to each of X_j's children.
3. Finally, $\mathcal{I}$ is the set of all unobserved variables in the graph that never received a ball. [6]

[3]Note that a Bayesian network is essentially a representation for a family of joint distributions over the T variables, that can be factorized in a specific way. The structure of the network allows us to infer certain conditional independencies that every distribution in the family must satisfy. However, each distribution might have certain independencies specific to it that the Bayesian network may not represent. The Bayes-Ball algorithm only recovers the independencies encoded by the network.

[4]For simplicity, we will use the term 'node' to refer to the variable corresponding to the node and vice versa.

[5]This means you must apply step 2a at all these nodes.

[6]It may not be clear when the given algorithm terminates. You may assume that when a node receives the ball from a specific direction for the second time, it does nothing.

Conditional Independences in LDA. We will study some of the conditional independencies in the Bayesian network G_{LDA} of the LDA model by: (1) applying the Bayes-Ball algorithm to G_{LDA}, and (2) deriving them from first principles using the joint distribution. For the rest of this homework, you can assume that we have just 2 documents, 2 words per document and 2 topics, i.e. $M = N = K = 2$ (although the results hold in general). To be able to run the Bayes-Ball algorithm easily, draw out the Bayesian network G_{LDA} for these parameters. For simplicity, you might also want to write out equation (1) for the special case of $M = N = K = 2$.

1. **[13 points]** Let i be a specific document and j be a specific word position. Suppose we observe $z_{i,j}$ (the topic of word $w_{i,j}$). Is θ_i conditionally independent of $w_{i,j}$ in G_{LDA}?

   (a) Run the Bayes-Ball algorithm to show that indeed $\theta_i \perp w_{i,j} | z_{i,j}$. You must place the ball in the node of either θ_i or $w_{i,j}$ and argue why it will never be received by the other node.

   (b) Next, prove this result starting from the factorized joint distribution given in equation (1) and using first principles from probability theory.

2. **[13 points]** Let i be a specific document, and j_1, j_2 be two distinct word positions. Is z_{i,j_1} independent of z_{i,j_2} in the Bayesian network when nothing is observed i.e., is $z_{i,j_1} \perp z_{i,j_2}$?

   (a) Run the Bayes-Ball algorithm to show that z_{i,j_1} and z_{i,j_2} are actually dependent in G_{LDA}. You must place the ball at one of these nodes and state the steps of the given algorithm by which the ball will be received by the other node.

   (b) Verify the result obtained above by constructing a probability distribution that can be represented by the LDA model and show that z_{i,j_1} and z_{i,j_2} are not independent for this distribution.

   Hints for 2b:

   (a) You can design a $\mathcal{D}_{\text{topics}}$ such that for some $k_1 \neq k_2$ and for some k_3, $p(z_{i,j_2} = k_3 | z_{i,j_1} = k_1) \neq p(z_{i,j_2} = k_3 | z_{i,j_1} = k_2)$ under the given generative process.

   (b) Consider $\mathcal{D}_{\text{topics}}$ to be equally distributed over exactly two points in its support: one point where the k_1th co-ordinate is 1 and all other co-ordinates are zero, and another point where the k_2th co-ordinate is 1 and all other co-ordinates are zero.

   (c) Intuitively, by observing $z_{i,j_1} = k_1$, what can you say about z_{i,j_2}? Now make this argument formal.

3. **[13 points]** Let i be a specific document, j be a specific word position, and k be a specific topic. Is $z_{i,j}$ independent of ϕ_k in G_{LDA} when $w_{i,j}$ is observed, i.e. is $z_{i,j} \perp \phi_k | w_{i,j}$?

   (a) Run the Bayes-Ball algorithm to show that $z_{i,j}$ and ϕ_k are actually dependent in G_{LDA} when $w_{i,j}$ is observed.

   (b) Verify the result obtained above by constructing a probability distribution that can be represented by the LDA model and show that $z_{i,j}$ and ϕ_k are not independent for this distribution, when $w_{i,j}$ is observed.

   Hint for 3b: Consider a distribution $\mathcal{D}_{\text{words}}$ such that every point in its support has exactly one of the D co-ordinates to be 1 and all other co-ordinates to be 0.

4. **[6 points]** Let i_1, i_2 be two distinct documents. Suppose all the words of document i_1 are observed, and only the first word of document i_2 is observed. In other words, the complete set of observed variables is $\{w_{i_1,1}, w_{i_1,2}, \ldots, w_{i_1,N}, w_{i_2,1}\}$. Now, is the topic of the second word of document i_2 (i.e. $z_{i_2,2}$) conditionally independent of the topic distribution of document i_1 (i.e. θ_{i_1}) in G_{LDA}? Use the Bayes-Ball algorithm to answer this query. [You are not required to prove the result from first principles.]

Hint for Bayes-Ball: $X \perp Y|Z$ if and only if $Y \perp X|Z$. So, you can choose the easier of X or Y to release balls from to run the Bayes-Ball algorithm.

Hints for proving using first principles: Let $\mathbb{X}$ denote the set of all variables. And, suppose we want to prove (or disprove) that $X \perp Y|Z$.

1. For independence, it must be true that $p(X|Y,Z) = p(X|Z)$.
2. By definition of conditional probability $p(X|Y,Z) = \frac{p(X,Y,Z)}{p(Y,Z)}$.
3. To obtain expressions for both the numerator and denominator, we can marginalize the joint distribution. Mathematically, $p(X,Y,Z) = \sum_{\mathbb{X}\setminus\{X,Y,Z\}} p(\mathbb{X})$, and $p(Y,Z) = \sum_{\mathbb{X}\setminus\{Y,Z\}} p(\mathbb{X})$. [Recall that we have access to the expression for the joint distribution $p(\mathbb{X})$ from equation (1).]
4. Each summation can be substantially simplified by moving some of the summations inside the factored $p(\mathbb{X})$, and observing that $\sum_U p(U|V) = 1$ for any V, as long as U does not appear in any other parts of the whole expression.
5. Some terms can be taken completely out of the summation, and canceled if they are common to both the numerator and denominator.
6. This can help you completely eliminate Y from the expression for $p(X|Y,Z)$, to prove independence.

3 Metropolis-Hastings Implementation [10 points] (Junjue)

This portion is not graded on Autolab. Instead, we will run your code and check your outputs below.

In this section, we will implement the Metropolis-Hastings algorithms to generate samples from the unnormalized distribution

$$\tilde{p}(x) = e^{-x^2} + 1.3e^{-(x-2)^2}$$

with a Gaussian sampling function centered at the previous point with some standard deviation σ.

1. **[5 points]** Run your implementation for every combination of $x_0 \in \{-2, 2\}$ and $\sigma \in \{0.1, 1.0, 10\}$. Submit plots (1. samples over time and 2. a histogram) for all of these in your writeup. We will grade these plots together with your implementation.
2. **[3 points]** How and why does σ impact the samples? For all three values, you should state what happens and why it happens.
3. **[2 points]** What is the impact of x_0 on the samples?

3.1 Getting started

Download the handout source code and edit the `mh.py` source file. This file contains a `main` method that will run your implementation and generate sampling plots and histograms. In this file, finish the Metropolis-Hastings implementation in the `mh` function from the description in Slides 17 (Probabilistic Modeling III: MCMC).

4 Programming a Deep Neural Network [25 points] (Chun Kai)

In Homework 2, you developed a linear classification model to classify MNIST digits. In this homework, you will implement a deep neural network for the same classification task. As a reminder, MNIST digit classification is a k-class classification task (where in this case $k = 10$). Our training set is of the form $(x^{(i)}, y^{(i)})$, $i = 1, \ldots, m$, with $y^{(i)} \in \{0, 1\}^k$, where $y_j^{(i)} = 1$ when j is the target class, and 0 otherwise (i.e., the output is one-hot encoded). Under the model, our hypothesis function $\hat{y} = h_\theta(x)$ outputs *vectors* in $\mathbb{R}^k$, where the relative size of the $\hat{y}_j$ corresponds roughly to how likely we believe that the output is really in class j. If we want the model to predict a single class label for the output, we simply predict class class j for which $\hat{y}_j$ takes on the largest value.

Our loss function $\ell : \mathbb{R}^k \times \{0, 1\}^k \to \mathbb{R}_+$ is the softmax loss, given by

$$\ell(\hat{y}, y) = \log\left(\sum_{j=1}^{k} e^{\hat{y}_j}\right) - \hat{y}^T y. \tag{2}$$

This loss function has the gradient

$$\nabla_{\hat{y}} \ell(\hat{y}, y) = \frac{e^{\hat{y}}}{\sum_{j=1}^{k} e^{\hat{y}_j}} - y \tag{3}$$

where the exponent $e^{\hat{y}}$ is taken elementwise.

To repeat a note from last time: In practice, you would probably want to implement regularized loss minimization, but for the sake of this problem set, we'll just consider minimizing loss without any regularization (at the expense of overfitting a little bit).

4.1 Implementation task [20 points]

You will implement a deep neural network to classify MNIST digits, trained by stochastic gradient descent. Specifically, you must implement the following three functions in `cls_nn.py`, with the inputs and outputs as described in the comments[7].

```
def nn(x, W, b, f):
"""
Compute output of a neural network.

Input:
```

[7] Please make sure to stick to the specified conventions when you submit to autolab.

```
    x: numpy array of input
    W: list of numpy arrays for W parameters
    b: list of numpy arrays for b parameters
    f: list of activation functions for each layer

    Output:
    z: list of activationsn, where each element in the list is a tuple:
    (z_{i}, z'_{i}),
    where z_{i}=f_{i-1}(W_{i-1}z_{i-1}+b_{i-1}) (for i>=2),
    z'_{i}=f'_{i-1}(W_{i-1}z_{i-1}+b_{i-1}) (for i>=2),
    z_{1}=x and z'_{1}=None.

    """

    def nn_loss(x, y, W, b, f):
    """
    Compute loss of a neural net prediction, plus gradients of parameters

    Input:
    x: numpy array of input
    y: numpy array of output
    W: list of numpy arrays for W parameters
    b: list of numpy arrays for b parameters
    f: list of activation functions for each layer

    Output tuple: (L, dW, db)
    L: softmax loss on this example
    dW: list of numpy arrays for gradients of W parameters
    db: list of numpy arrays for gradients of b parameters
    """

    def nn_sgd(X,y, Xt, yt, W, b, f, epochs=10, alpha = 0.005):
    """
    Run stochastic gradient descent to solve linear softmax regression.

    Inputs:
    X: numpy array of training inputs
    y: numpy array of training outputs
    Xt: numpy array of testing inputs
    yt: numpy array of testing outputs
    W: list of W parameters (with initial values)
    b: list of b parameters (with initial values)
    f: list of activation functions
    epochs: number of passes to make over the whole training set
    alpha: step size

    Output: None (you can directly update the W and b inputs in place)
    """
```

The size of the inputs `W,b,f` determine the effective size of the neural network. We have included starter code that constructs the network in such a format, so you just need to implement the above function. In particular, a deep network for the MNIST problem, initialized with random weights, with rectified linear units in all layers except just a linear unit in the last layer, is constructed by the code:

```
np.random.seed(0)
layer_sizes = [784, 200, 100, 10]
W = [0.1*np.random.randn(n,m) for m,n in zip(layer_sizes[:-1], layer_sizes[1:])]
b = [0.1*np.random.randn(n) for n in layer_sizes[1:]]
f = [f_relu]*(len(layer_sizes)-2) + [f_lin]
```

In particular, this creates a deep network with 4 total layers (2 hidden layers), of sizes 784 (the size of the input), 200, 100, and 10 (the size of the output) respectively. This means, for instance, that there will be 3 W terms: $W_1 \in \mathbb{R}^{200\times 784}$, $W_2 \in \mathbb{R}^{100\times 200}$, and $W_3 \in \mathbb{R}^{10\times 100}$.

You'll use virtually the same SGD procedure as in Homework 2, so the main challenge will be just to compute the network output and the gradients in the `nn` and `nn_loss` functions, using the backpropagation algorithm as specified in the class slides. We have included the various activation functions, which return both the non-linear function f and its elementsize subgradient f'.

Note that training a neural network with pure stochastic gradient descent (i.e., no minibatches) can be fairly slow, so we recommend you test your system on a small subset of (say) the first 1000 training and testing examples, and only run it on the final data set after you have debugged its performance on a smaller data set. As a reference point, our (quite unoptimized) implementation takes about a minute per epoch on the full training set, with the following performance over the first few epochs for the default $\alpha = 0.01$:

```
Test Err |Train Err |Test Loss |Train Loss
   0.9033|    0.9044|    2.4827|    2.4800
   0.0514|    0.0430|    0.1578|    0.1351
   0.0349|    0.0265|    0.1218|    0.0829
   0.0291|    0.0173|    0.1024|    0.0527
   0.0339|    0.0180|    0.1187|    0.0550
```

4.2 Written portion [5 points]

In addition to the code that you'll submit (which will be evaluated on simpler toy examples to check for correctness), also include with your submission

1. A figure showing the training error and testing error versus epoch for the two linear softmax regression algorithms from Homework 2 (using either your own code or the solution code in `cls_nn.py`) plus the neural network, as measured on the full MNIST digit classification data set.
2. A figure showing the average training and testing loss versus epoch for all three algorithms, again as measured on the MNIST problem.

For the above, you can use 10 epochs and the default learning rates given in the starter code (that is, $\alpha = 0.5$ for softmax GD, and $\alpha = 0.01$ for softmax SGD and the neural network).

5 Submitting to Diderot

Create a tar file containing your writeup for the first two problems and the completed `cls_nn.py` and `mh.py` modules for the programming problems. Make sure that your tar has these files at the root and not in a subdirectory. Use the following commands from a directory with your files to create a `handin.tgz` file for submission.

```
$ ls
cls_nn.py  mh.py writeup.pdf
$ tar cvzf handin.tgz writeup.pdf cls_nn.py mh.py
a writeup.pdf
a cls_nn.py
a mh.py
$ ls
cls_nn.py  handin.tgz  writeup.pdf mh.py
```