TTIC 31020: Introduction to Machine Learning
Autumn 2020

Problem Set #5

Out: Monday November 30, 2020

Due: Monday December 7, 11:59pm

Instructions

How and what to submit? Please submit your solutions electronically via Canvas. Please submit two files:

1. A PDF file with the written component of your solution including derivations, explanations, etc. You can create this PDF in any way you want, e.g., LaTeX (recommended), Word + export as PDF, scan handwritten solutions (note: must be legible!), etc. Please name this document ⟨`firstname-lastname`⟩`-sol5.pdf`.

2. The empirical component of the solution (Python code and the documentation of the experiments you are asked to run, including figures). Rename the notebook ⟨`firstname-lastname`⟩`-sol5.ipynb`.

3. The bonus question (if you choose to submit it) in a *link to a Colab notebook*. Rename the notebook ⟨`firstname-lastname`⟩`-sol5-bonus.ipynb`

Late submissions: there will be a penalty of 25 points for any solution submitted within 24 hours past the deadlne. No submissions will be accepted past then.

What is the required level of detail? When asked to derive something, please clearly state the assumptions, if any, and strive for balance: justify any non-obvious steps, but try to avoid superfluous explanations. When asked to plot something, please include in the `ipynb` file the figure as well as the code used to plot it. If multiple entities appear on a plot, make sure that they are clearly distinguishable (by color or style of lines and markers) and references in a legend or in a caption. When asked to provide a brief explanation or description, try to make your answers concise, but do not omit anything you believe is important. If there is a mathematical answer, provide is precisely (and accompany by only succinct words, if appropriate).

When submitting code (in Jupyter notebook), please make sure it's reasonably documented, runs and produces all the requested results. If discussion is required/warranted, you can include it either directly in the notebook (you may want to use the markdown style for that) or in the PDF writeup.

Collaboration policy : collaboration is allowed and encouraged, as long as you (1) write your own solution entirely on your own, (2) specify names of student(s) you collaborated with in your writeup.

Total Points This assignment will be scored out of 100 points. Observe that there are 125 total points available. Any points you earn in excess of the 100 will be scored as bonus.

1 Information Theory

Problem 1 [34 points]
[Maximum Entropy Distribution] Let X be a random variable in $[m] \coloneqq \{1, 2, \ldots, m\}$ according to some distribution $\mathcal{D}$. Show that the $\mathcal{D}$ maximizing $H(X)$ is the uniform distribution over $[m]$.

Hint: There are at least two ways to do this problem. One is to set up an optimization problem, use Lagrange multipliers, and solve. Another is to let u be the uniform distribution over $[m]$ and calculate $D(p||u)$ for any arbitrary distribution p. The second method is far cleaner.

End of problem 1

Problem 2 [33 points]
[Maximum Likelihood and KL Divergence] In this problem, all logarithms are base e.

Consider a parameter estimation problem where we wish to maximize log-likelihood. That is, we're given the population X from some underlying model parameterized by θ^*, a likelihood function $P_\theta(\cdot)$, and we wish to maximize $\log(P_\theta(X))$ over θ.

Show that maximizing population log-likelihood amounts to minimizing $D(P_{\theta^*}(X)||P_\theta(X))$ over θ. Specifically, show that:

$$\operatorname*{argmax}_\theta \mathbb{E}_{P_{\theta^*}(X)}\left[\log\left(P_\theta(X)\right)\right] = \operatorname*{argmin}_\theta D(P_{\theta^*}(X)||P_\theta(X))$$

End of problem 2

2 Building a Pipeline for training Neural Networks

In this section we will go through the some of the major components involved in building a pipeline for training a fully connected network[1]: construction and initialization of a fully connected network, training by back-propagation, regularization and applying it to the Fashion MNIST data set (FMNIST). The train/validation/test partitions for this data set is provided in the assignment page.

The code in the notebook has a few missing pieces. Otherwise, it implements all you need for a basic fully-connected network. The network can be constructed as a sequence of modules (layers), and a module has two essential functionalities:

[1] this is the basic neural network, in which each unit in layer t is connected to each unit in layer $t+1$, as discussed in class. Such a network is also known as multi-layer perceptron (MLP). This is in contrast to *locally connected* networks where each unit is only connected to a subset of units in the layer below; combining that pattern with a weight sharing scheme between units yields *convolutional* networks.

- forward computation (converting input into output);
- backward computation (converting error signal from output to error signal for the input, as well as computing gradient of the error with respect to the model parameters, if any.

Two additional files include implementation of general optimization tools (utils.py) and some tools specific to the image data, FMNIST utils.py). The main type of layer we will work with is the "fully connected layer", which is the basic layer we covered in class; given the output $\mathbf{z}_{t-1}$ of the previous layer, it computes its output as

$$\mathbf{z}_t = h(\mathbf{W}_t \mathbf{z}_{t-1} + \mathbf{b}_t) \tag{1}$$

where $\mathbf{W}_t$ is the matrix of weights, and $\mathbf{b}_t$ is the vector of biases.

Problem 3 **[13 points]**

Model Construction and Initialization: First we begin by constructing individual components of our Neural Network (model).

- Provide Forward and backward passes for both a fully connected layer and ReLU nonlinearity. Provide initialization for fully connected layer. Complete code in classes `FullyConnected` and `ReLU`.
- Provide Backward pass for softmax layer. Complete code in class `SoftMaxLoss`.
- Now that the individual layers are ready, combine them to construct a toy 2-layer network with a hidden dimension of 10 (for testing purposes). Remember to add the softmax layer.

End of problem 3

Note that the softmax "layer" is technically not a layer but a different object called "criterion" – that's a common jargon in modern neural network implementations for the loss "layer" (the criterion to be optimized). It has forward/backward functionalities like the network layers, but it interacts with an external source of information – the true label values.

Also, note that the ReLU layer does not have trainable parameters (weights or biases); nonetheless, it does perform computation in both the forward and the backward pass.

Advice: Please use test softmax layer, sub-components and the final neural network using functions `test_sm` *and* `test_module`

Next we will set up the optimizer – the machinery used to train the network. We will use SGD, our trusty machine learning workhorse (and the most commonly used algorithm for learning neural networks).

Problem 4 **[10 points]**

- Provide gradient descent update. Complete code in function `sgd`.

Advice: Please use the code provided to test your SGD update

End of problem 4

As with any ML method, we need to consider **regularization**. Here, we will use the dropout, which is a staple of modern day neural networks. Code for the forward and backward pass of dropout layer is already provided, you would only have add it to the model (in the `build_model` function).

We are now ready to run an experiment with a dataset – the "Fashion MNIST" dataset, which we provide. Like with any ML tool, neural networks have various hyper-parameters with each having its own effect on the network's ability to generalize; you will need to tune these to find settings that provide good validation (and ultimately, we hope, test) accuracy.

Problem 5 [10 points]
Train a multi-layer network on the Fashion MNIST data, and use the best model you can find to submit predictions on the test set to Kaggle.

- Complete code in `build_model` function to build a network capable of classifying FMNIST images.
- Use the model to classify FMNIST images (script provided).

Tuning the Network:

- Please describe the effects of increasing and decreasing the hidden size of the network (number of hidden units in a layer), and likewise changing the depth of the network from 2 to 3.

Further, you are free to experiment with activation functions (ReLU, sigmoid, etc.) as as well as with optimization techniques (different update rules, dropout, weight decay, learning rate schedules). However please do not introduce new layer/connection types (no convolutional layers, for instance). Many of the relevant tools (e.g., dropout) are included in the code; others (e.g., tanh activations) you may need to code up yourselves.

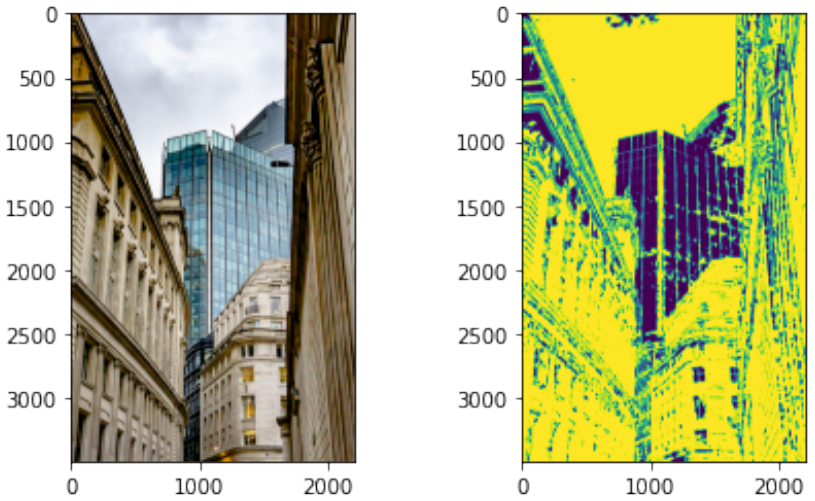
End of problem 5

3 Mixtures of Gaussians

Problem 6 [25 points]
In this problem, we (i.e., you) will implement an algorithm performing the E-M steps to learn the parameters for a Mixture of Gaussians. To evaluate our implementation, we'll use EM to perform image segmentation. That is, we assume there are k distinct "components"

to an image that generate its elements, and we'll use EM to attribute each pixel to its source component. For instance, if we set $k = 2$, we may expect the image to be separated into a low-variance background and a possibly higher-variance subject that's in the foreground.

Concretely, consider the following example:



(a) Original (b) EM-Segmented

Figure 1: Side-by-side comparison of original image vs EM-segmented image

There are more examples in the notebook.

While implementing, it may help you to use the following procedure. (no need to write anything here, this is just a guide).

1. Figure out how to initialize the centers of the Gaussians.
2. Write out the expectation step and the maximization step.
3. Implement each step.
4. Run one iteration at a time to debug. Each iteration on one of the sample images took me roughly a minute.
5. Once you have a working implementation, run your code on the sample images for a few iterations. One iteration for all three images took me roughly 4 minutes to run.

We've included a reference implementation (courtesy of scikit-learn) to help you test your implementation of EM.

Once your implementation works, try several different values for k and several different input images. In particular, try running your segementation algorithm on images that have a more cluttered foreground; on the other hand, also try running the algorithm on an image with a single, simple subject in the foreground against a clean or monotone background.

Submission: Paste your sample images (original vs generated) in your submission file, and paste a link to your Colab notebook in this solution file. Be sure to set the sharing permissions to "anybody on the internet can view."

We will actually run your code, so please don't use the reference implementation from scikit-learn to generate your samples (we'll find out!).

End of problem 6