CIS 419/519 Applied Machine Learning
Assignment 5

Due: Wed Apr 1, 2020 11:59pm

Instructions

Read all instructions in this section thoroughly.

Collaboration: Make certain that you understand the collaboration policy described on the course website.

You should feel free to talk to other members of the class in doing the homework. I am more concerned that you learn how to solve the problem than that you demonstrate that you solved it entirely on your own. You may *discuss* the homework to understand the problems and the mathematics behind the various learning algorithms, but you must **write down your solutions yourself**. In particular, **you are not allowed to share problem solutions or your code with any other students.**

We take plagiarism and cheating very seriously, and will be using automatic checking software to detect academic dishonesty, so please don't do it.

You are also prohibited from posting any part of your solution to the internet, even after the course is complete. Similarly, please don't post this PDF file or the homework skeleton code to the internet.

Formatting: This assignment consists of two parts: a problem set and program exercises.

For the problem set, you must write up your solutions electronically and submit it as a single PDF document. We will not accept handwritten or paper copies of the homework. Your problem set solutions must use proper mathematical formatting.

All solutions must be typeset using LaTeX; handwritten solutions of any part of the assignment (including figures or drawings) are not allowed.

Your solutions to the programming exercises must be implemented in python, following the precise instructions included in Part 2. Portions of the programming exercise will be graded automatically, so it is imperative that your code and output follows the specified API. A few parts of the programming exercise asks you to create plots or describe results; these should be included in the same PDF document that you create for the problem set.

Homework Template and Files to Get You Started: Skeleton code is available at:
https://www.seas.upenn.edu/~cis519/spring2020/homework/hw5_skeleton.ipynb.

Citing Your Sources: Any sources of help that you consult while completing this assignment (other students, textbooks, websites, etc.) ***MUST*** be noted in the your PDF file. This includes anyone you briefly discussed the homework with. If you received help from the following sources, you do not need to cite it: course instructor, course teaching assistants, course lecture notes, course textbooks or other readings.

Submitting Your Solution: You will submit this assignment through Gradescope.

Files to submit:

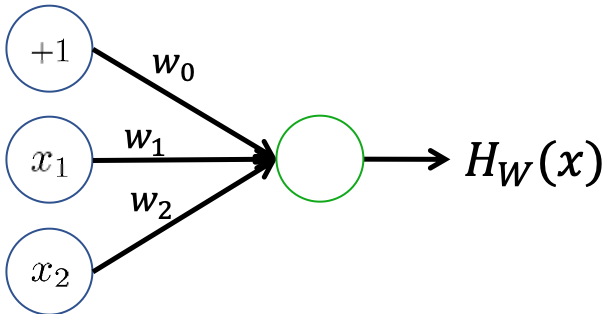
- **HW5_report.pdf**
- **HW5.ipynb**
- **cnn.th**

Assignment Version 20200303

PART I: PROBLEM SET

Your solutions to the problems will be submitted as a single PDF document. Be certain that your problems are well-numbered and that it is clear what your answers are.

1 Logical Functions with Neural Nets [10 pts]



A neural network with apropriate weights can compute logical functions. For example, consider the simple 2-layer network in the figure above, with 2-dimensional binary input $\mathbf{x}$, where each of x_1 and x_2 can take values 0 or 1 to denote false or true, so there are a total of four possible values that the vector $\mathbf{x}$ can take: $\{[0,0],[0,1],[1,0],[1,1]\}$. The activation function at the output node is a sigmoid $\sigma(z) = \frac{1}{1+\exp(-z)}$. So the output of this neural net is:

$$H(\mathbf{x}) = \sigma(w_0 + w_1 x1 + w_2 x_2).$$

Suppose we set $w_0 = -30$, $w_1 = 20$, and $w_2 = 20$. What is the output of the network for each of the four values of the input $\mathbf{x}$?

x_0	x_1	$H(\mathbf{x})$
0	0	σ(-30)=0
0	1	σ(-10)=0
1	0	σ(-10)=0
1	1	$\sigma(10) = 1$

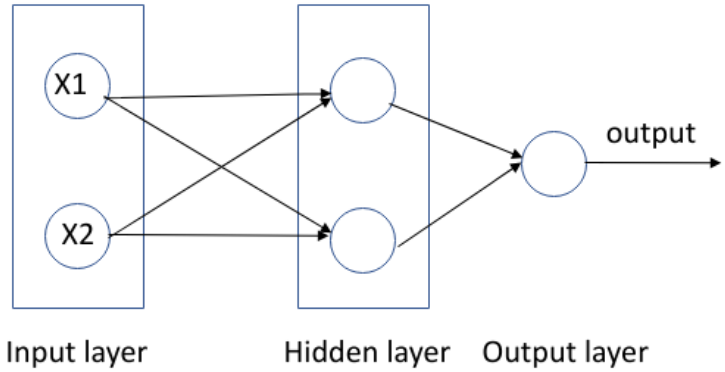
Observe that this table is exactly the "truth table" for the logical AND function, so this setting of w_0, w_1, w_2 exactly computes the AND function.

Based on this example, for each of the logical functions below, draw the neural network that computes the function, show the values of the weights, and give a truth table showing the inputs, the value of the logical function, and the output of the neural network verifying that the neural network is correct. Show your work for the computations of the neural network's output.

(a) The NAND of two binary inputs

(b) The parity of three binary inputs (the parity function is 1 if and only if the number of ones in the input vector is odd)

2 Calculating Backprop by Hand [20 pts]



Consider the two-layer neural network shown above. You will use the sign function as the activation function in the hidden layer, but the output layer has a sigmoid activation function. The input vector is $\mathbf{x} = [5 \quad 4]^\mathsf{T}$. Compute the output gradient with respect to each of the weights. Be sure to show your work.

Use the following values for the input and weights:
Weight matrix for the first layer (between input and the hidden layer):

$$\mathbf{W}^1 = \begin{bmatrix} 0.1 & 0.2 \\ -0.4 & 0.3 \end{bmatrix}$$

Weight matrix between the hidden layer and output:

$$\mathbf{W}^2 = \begin{bmatrix} 0.1 & 0.2 \end{bmatrix}$$

$$Sign(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

$$\nabla Sign(z) = \begin{cases} 1 & \text{if } -1 \leq z \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

Notation: W_{ij} denotes the weight between j^{th} component of the input vector (the left layer) and the i^{th} neuron of the next layer to the right. For example, to compute the input to the first neuron (start counting from the top) of the hidden layer:

$$[0.1 \quad 0.2] \times [5 \quad 4]^T = 1.3$$

.

PART II: PROGRAMMING EXERCISES

3 Neural Nets in SuperTuxKart [50 pts]

In this homework, we will train a CNN to classify images of objects from a car racing video game, called SuperTuxKart. The dataset is at: https://drive.google.com/open?id=1jMxlTktHT137wqkBgcIvUNwSnGc8ot2R. There are 6 classes of objects. kart is 1, pickup is 2, nitro is 3, bomb is 4 and projectile 5. The background class (all other images) is assigned the label 0.

The starter code contains a data directory where you'll copy (or symlink) the SuperTuxKart dataset. Unzip the data directly into the homework folder, replacing the existing data directory completely. Make sure you see the following directories and files inside your main directory: `homework`, `grader`, `bundle.py`, `data`, `data/train`, and `data/valid`. You will run all scripts from inside this main directory.

3.1 Loading Data [10 pts]

First, you need to load data in a way that Pytorch can deal with easily. We will lean on Pytorch's `Dataset` class to do this. Complete the `SuperTuxDataset` class that inherits from `Dataset`.

The `__len__` function should return the size of the dataset, i.e., the number of samples. The `__getitem__` function should return a python `tuple` of `(image, label)`. The image should be a `torch.Tensor` of size $(3, 64, 64)$ with range `[0,1]`, and the label should be `int`. `__init__` is a constructor, and would be the natural place to perform operations common to the full dataset, such as parsing the labels and image paths. The labels and image paths are stored in `labels.csv`, their headers are the file and label.

Once you have written this function, you will be able to visualize objects in each category, by calling `visualize_data()`.

We recommend using the `csv` package to read `csv` files and the `PIL` library (`Pillow` fork) to read images in Python. Use `torchvision.transforms.ToTensor()` to convert the `PIL` image to a pytorch tensor. You have (at least) two options on how to load the dataset. You can load all images in the `__init__` function, or you can lazily load them in `__getitem__`. If you load all images in `__init__`, make sure you convert the image to a tensor within the constructor, otherwise, you might get an `OSError: [Errno 24] Too many open files`.

Some Relevant Operations: `torchvision.transforms.ToTensor`, `torch.utils.data.Dataset`, `csv.reader`, `PIL.Image.open`

3.2 Implementing the classification loss [10 pts]

Next, we will implement `ClassificationLoss` in `models.py`. We will later use this loss to train our classifiers. You should implement the log-likelihood of a softmax classifier. Assume that the scores from your

model are unnormalized at this point, that is, they have not been converted to probabilities using the logistic function.

Remember, we will use Pytorch's `autograd` during training, when we'll have to find the gradient of this loss. For that to be possible, you will need to use existing pytorch functions to implement this.

Some Relevant Operations: `torch.nn.functional`

3.3 Implement the CNN model [10 pts]

[NOTE: Mar 11, 2020. We have not yet covered CNN's in class. If you are anxious to get started, you could choose to hold off on implementing this, or implement a non-convolutional architecture for the moment that deals with $64x64$ images as 4096-dimensional vectors. Doing this now will enable you to get an early start on implementing and debugging neural net training (which will remain similar for convolutional networks) and also give you a sense for the impact of convolutional architectures!]

Implement the `CNNClassifier` in `models.py`. Define the architecture and all layers in the `__init__` function, then implement `forward`. `forward` receives a $(B, 3, 64, 64)$ tensor as an input and should return a $(B, 6)$ `torch.Tensor` (one value per class). These outputs will be the "logits" that we will feed into `ClassificationLoss` later.

Some Relevant Operations: `torch.nn.Conv2d`, `torch.nn.Linear`, `torch.tensor.View`, `torch.nn.Sequential`, `torch.nn.ReLU`

3.4 Implementing the optimization procedure [20 pts]

Train your CNN in `train.py`. You should implement the full training procedure:

- Create a model, loss, optimizer
- Load the data: training and validation.
- Run SGD for several epochs
- Save your final model, using `save_model`

You can train your network using `train()`.

Hint: You might find it useful to allow training of an existing model to continue. Use the `torch.load` function for that. See `load_model`.

Some Relevant Operations: `torch.optim.Optimizer`, `torch.optim.SGD`, `torch.Tensor.backward`

3.5 Logging [10 pts]

In this part, we learn how to use `tensorboard`. We created a dummy training procedure in `test_logging()`, and provided you with two `tb.SummaryWriter` as logging utilities. Use those summary writers to log the training loss at every iteration, the training accuracy at each epoch and the validation accuracy at each epoch. Here is a simple example of how to use the `SummaryWriter`.

```
import torch.utils.tensorboard as tb
logger = tb.SummaryWriter('cnn')
logger.add_scalar('train/loss', t_loss, 0)
```

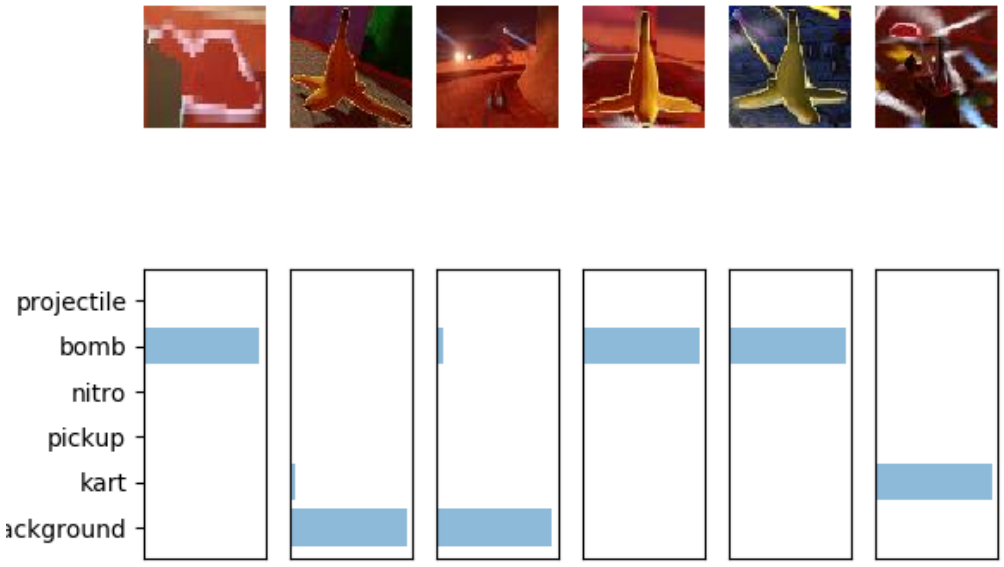
In `test_logging()`, you should not create your own `SummaryWriter`, but rather use the one provided. The skeleton code calls `test_logging()` in the code snippet right below it.

Some Relevant Operations `torch.utils.tensorboard.SummaryWriter`, `torch.utils.tensorboard.SummaryWriter.add_scalar`

3.6 Training the CNN [40 pts]

[NOTE: Mar 11, 2020. We have not yet covered training tips in class, so while you might be able to implement a vanilla version of this portion right away, you may find that you are able to boost performance significantly based on ideas we will cover in the next instruction week.]

Train your model and save it as `cnn.th`. We highly recommend you incorporate the logging functionality from the previous section into your training routine, so that you can diagnose how to improve your model. Once you have trained your model, you can optionally visualize your model's prediction using `visualize_predictions()`.



You should get 85% accuracy on the test set to get full points for this portion. Note that we will use a different testing dataset to grade the accuracy of your model, so don't overfit to the validation set.

Attribution: This programming assignment and starter code are based on materials originally developed by Philipp Krahenbuhl and Chao-Yuan Wu: http://www.philkr.net/dl_class/