CSE 571 - Robotics
Assignment 1 - Bayes Filters and Fully Connected Networks

Due Tuesday, April 19th @ 11:59pm

This homework involves three math problems and a programming assignment in Python. The goal of the math problems is to help you understand conditional distributions and Bayesian inference. For the programming portion, you will be implementing a simple neural network in Python without using any existing deep learning frameworks. Our aim is to teach you what these frameworks do, so you'll have a better understanding when it's time to actually use them. The code for this homework is available at `https://courses.cs.washington.edu/courses/cse571/22sp/homeworks/assignment1.zip`

Useful reading material: Lecture notes, **Chapter 2 of Probabilistic Robotics, Thrun, Burgard and Fox (pdf shared with class)** and Chapter 2 of Gaussian Processes for Machine Learning, Rasmussen and Williams (Available online at: `http://www.gaussianprocess.org/gpml/chapters`)

Collaboration: Students can discuss questions and work together to solve the problems, but each student MUST write up their own answers and code their own software. We will be checking both the code and PDFs for plagiarism. If we find that you have shared your solutions or code online, you will get an automatic 50% on the assignment. If we find that you have plagiarized solutions from another student or the internet, you will get an automatic 0% on this assignment.

Late Policy: You are allowed 6 late days for the entire quarter. After using these up, you will incur a penalty of 20% of the maximum grade per day. Please plan ahead!

1 Writing Assignments [40 points]

1.1 Conditional Independence [5 points]

Let X, Y and Z be three random variables. Assuming that X and Y are conditionally independent given Z

$$p(X, Y \mid Z) = p(X \mid Z)p(Y \mid Z)$$

Show that

$$p(X \mid Z) = p(X \mid Z, Y)$$
$$p(Y \mid Z) = p(Y \mid Z, X)$$

1.2 Bayes Filter [15 points]

A special-purpose robot is equipped with a vacuum unit to clean the floor. The robot has a binary sensor to detect whether a floor tile is clean or dirty. However, neither the cleaning unit nor the sensor are perfect. From previous experience, you know that the robot succeeds in cleaning a dirty floor tile with a probability of

$$\mathbb{P}(x_{t+1} = \text{clean} \mid x_t = \text{dirty}) = 0.6,$$

where x_t is the state of the floor tile at time t and x_{t+1} is the resulting state after the action has been applied. Activating the cleaning unit when the tile is clean will never make it dirty. Assume the robot always cleans at every time t (i.e. the transition probabilities model the fact that the robot is cleaning the floor tile).

The probability that the sensor indicates that the floor tile is clean although it is dirty is $p(z_t = \text{clean} \mid x_t = \text{dirty}) = 0.2$ and the probability that the sensor correctly detects a clean tile is $p(z_t = \text{clean} \mid x_t = \text{clean}) = 0.6$.

Unfortunately, you have no knowledge about the current state of the floor tile. However, after cleaning the tile, the robot's sensor indicates that it is clean. Compute $p(x_{t+1} = \text{clean} \mid z_{t+1} = \text{clean})$, *i.e.*, the probability that at time $t + 1$, the floor tile is now clean given that the sensor indicates it is clean. Assume a prior distribution at time t as $p(x_t = \text{clean}) = c$, where $0 \leq c \leq 1$. Then, using a software package of your choice, plot $p(x_{t+1} = \text{clean} \mid z_{t+1} = \text{clean})$ for $0 \leq c \leq 1$. You do not need to submit your code for plotting.

1.3 Gaussian Conditioning [20 points]

Let X and Y denote two scalar random variables that are jointly Gaussian:

$$\begin{aligned} p(x, y) &= \mathcal{N}(\mu, \Sigma) \\ &= \frac{1}{2\pi\sqrt{|\Sigma|}} \exp\left\{-\frac{1}{2}\left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} \mu_X \\ \mu_Y \end{bmatrix}\right)^{\intercal} \Sigma^{-1} \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} \mu_X \\ \mu_Y \end{bmatrix}\right)\right\} \end{aligned}$$

where $\mu = \begin{bmatrix} \mu_X & \mu_Y \end{bmatrix}^{\intercal}$ and $\Sigma = \begin{bmatrix} \sigma_X^2 & \sigma_{XY} \\ \sigma_{XY} & \sigma_Y^2 \end{bmatrix}$ are the mean and covariance, respectively.

Show that conditioning on Y results in a Gaussian over X:

$$\begin{aligned} p(x \mid y) &= \mathcal{N}(\mu_{X|Y}, \sigma_{X|Y}^2) \\ &= \frac{1}{\sqrt{2\pi}\sigma_{X|Y}} \exp\left\{-\frac{1}{2}\frac{(x - \mu_{X|Y})^2}{\sigma_{X|Y}^2}\right\} \end{aligned}$$

with $\mu_{X|Y} = \mu_X + \frac{\sigma_{XY}}{\sigma_Y^2}(y - \mu_Y)$ and $\sigma_{X|Y}^2 = \sigma_X^2 - \frac{\sigma_{XY}^2}{\sigma_Y^2}$.

Hints

- Use the definition of the conditional distribution and "complete the square" to get the answer
- Given $p(x, y)$ is jointly Gaussian, the marginal distribution of Y is also Gaussian: $p(y) = \mathcal{N}(\mu_Y, \sigma_Y^2)$
- For a 2x2 matrix positive definite matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, $A^{-1} = \frac{1}{|A|}\begin{bmatrix} d & -c \\ -b & a \end{bmatrix}$, $|A| = ad - bc$

2 Programming Problems [60 points]

In this assignment, we're going to implement part of a standard multi-layer perceptron consisting of linear layers plus non-linear functions. You will train a classifier on MNIST, a common image dataset on handwritten numbers, while learning the mechanisms of a neural network, including the forward and backward pass and how gradients flow.

We have provided some tests for checking your implementation. The tests are intentionally missing some cases, but feel free to write more tests yourself. To run the tests from the outermost directory, simply run

```
pytest tests/hw1_tests
```

Or to run them for an individual file (for example `test_linear_layer`), run

```
pytest tests/hw1_tests/test_linear_layer.py
```

Rules You may not use PyTorch or any other deep learning package. Only Numpy is allowed. Functions like `numpy.matmul` are fine to use. You may only modify the files we mention (those in the `submit.sh` script). We will not grade files outside of these. You may not change the signatures or return types of `__init__`, `forward`, `backward` or `step` or you will fail our tests. You may add object fields or helper functions within the files. You may talk with others about the homework, but you must implement by yourself.

2.0 Set up the codebase

Download and unzip the codebase from `https://courses.cs.washington.edu/courses/cse571/22sp/homeworks/assignment1.zip`. Install Miniconda (`https://docs.conda.io/en/latest/miniconda.html`) and run the following commands.

```
cd assignment1
conda env create -f environment.yml
conda activate robotics-class
```

Layers Look at `../nn/layers/layer.py` where the base `Layer` class is defined. All of your neural network layers will inherit from this class, which helps create and track the computation graph. When you overload `Layer` you will need to implement a `forward` and a `backward` function. The `forward` function should take an input array and return a new array. The `backward` function will take partial gradients with respect to the layer's output, accumulate gradients with respect to the parameters, and return gradients with respect to the input. For this homework, each `backward` you implement should return a single array. You can also optionally implement `selfstr` which prints out information about the layer.

Parameters Look at `../nn/parameter.py` to see the data structure used to hold the weights and biases of the network. For the forward pass, you will need to access the `param.data` field, and for the backward pass, you will need `param.grad`. Note that assigning to `param.grad` actually does the `+=` operation in order to accumulate gradients.

2.1 Linear Layer [10 points]

Open `../nn/layers/linear_layer.py`. Implement the forward and backward pass of a linear layer, which defines the following operation

$$y = Wx + b$$

where y is the output, x is the input, W is the weight matrix and b is the bias vector.

You can assume that `LinearLayer` takes a 2D array of shape (batch size, input feature dimension) as input and returns a 2D array of shape (batch size, output feature dimension) as output.

2.2 ReLU Layer [10 points]

Rectified Linear Unit (ReLU) is a simple yet most widely used non-linear activation function in neural networks. Open `../nn/layers/relu_layer.py` and implement the forward and backward pass of the ReLU function defined as follows

$$f(x) = \max(0, x)$$

ReLU layers (and all the non-linearities you implement) should accept arrays of arbitrary shape.

2.3 Softmax Cross Entropy Loss Forward [15 points]

Cross entropy is a commonly used loss function for classification problems. It measures the discreprency between the ground truth distribution p and predicted distribution q over n possible values.

$$H(p, q) = -\sum_{i=1}^{n} p(i) \log(q(i))$$

Usually, p is a one-hot vector where $p(j) = 1$ for the ground truth label j and $p(i) = 0$ for $i \neq j$, so $H(p, q)$ reduces to

$$H(p, q) = -\log(q(j))$$

It is often used together with the softmax activation function, which turns any real-valued vector x of dimension n into a normalized distribution over n values.

$$q(j) = \frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}}$$

Note that the softmax function is prone to overflow and underflow during computation if any x_i is large. By implementing the softmax and cross entropy operation together, better numerical stability can be achieved via the so-called log-sum-exp trick. Let $m = \max_i(x_i)$. We have

$$\begin{aligned}
\log q(j) &= \left(\frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}}\right) \\
&= \log(e^{x_j}) - \log\left(\sum_{i=1}^{n} e^{x_i}\right) \\
&= x_j - \log\left(e^m \sum_{i=1}^{n} e^{x_i - m}\right) \\
&= x_j - m - \log\left(\sum_{i=1}^{n} e^{x_i - m}\right)
\end{aligned}$$

Open `../nn/layers/losses/softmax_cross_entropy_loss_layer.py`. Implement the forward pass of the softmax cross entropy operation. The inputs include the logits (output of the network), the targets (integer indicating the ground truth classes) and the axis along which the softmax normalization should be taken. Make sure to implement both mean and sum reduction.

Hint You can assume the input is 2D at first, but to get full credit you need to make softmax cross entropy work for inputs with arbitrary dimensions (warning, harder than it sounds). Take a look at these useful functions: `numpy.reshape`, `numpy.moveaxis`, `numpy.expand_dims`, `numpy.take_along_axis`.

2.4 Softmax Cross Entropy Loss Backward [15 points]

Implement the backward pass of the softmax cross entropy operation.

Hint You should use class variables to store relevant values from the forward pass in order to use them in the backward pass. With some fancy math, we can show that the gradient of the cross entropy loss with respect to the inputs is actually quite simple

$$\frac{\partial H}{\partial x_i} = q(i) - p(i).$$

Remember to scale the loss appropriately if the reduction was mean.

2.5 SGD Update [10 points]

Open `../nn/optimizers/sgd_optimizer.py`. Recall that each `Parameter` has its own data and grad variables. Based on the other parts you wrote, the gradients should already be ready inside the `Parameter`. Now we just have to use them to update the weights.

Our normal SGD update with learning rate η is:

$$w \Leftarrow w - \eta \frac{\partial L}{\partial w}$$

With this done, we can now train our first neural network! Open `main.py`. We have already provided code to train and test a simple three layer neural network. Have a look at the training loop. First, you load the data. Then you call the forward function on the network to get its outputs. Finally, you zero the previous gradients, call backward on the network, and update the weights. You can run it by calling

```
python main.py
```

After 1 epoch, you should see about 70% test accuracy. After 10 epochs, you should see about 90% accuracy.

3 Submission

You will be using Gradescope `https://www.gradescope.com` to submit the homework. Please submit the written assignment answers as a PDF. For the code, use the submit script in the homework directory and submit the compressed directory that it produces.