Homework 1 Bonus

ADAM and Dropout

11-785: Introduction to Deep Learning (Spring 2020)

Out: **February 20, 2021**
Due: **April 29, 2021**

Start Here

- **Collaboration policy:**
  - You are expected to comply with the University Policy on Academic Integrity and Plagiarism
  - You are allowed to talk with / work with other students on homework assignments
  - You can share ideas but not code, you must submit your own code. All submitted code will be compared against all code submitted this semester and in previous semesters using MOSS.
- **Overview:**
  - MyTorch
  - ADAM
  - Dropout
- **Directions:**
  - You are required to do this assignment in the Python (version 3) programming language. Do not use any auto-differentiation toolboxes (PyTorch, TensorFlow, Keras, etc) - you are only permitted and recommended to vectorize your computation using the Numpy library.
  - If you haven't done so, use pdb to debug your code effectively.

1 MyTorch

The culmination of all of the Homework Part 1's will be your own custom deep learning library, which we are calling MyTorch. It will act similar to other deep learning libraries like PyTorch or Tensorflow. The files in your homework are structured in such a way that you can easily import and reuse modules of code for your subsequent homeworks. For Homework 1 bonus, MyTorch will have the following structure:

- mytorch
  - loss.py (Copy your file from HW1P1)
  - activation.py (Copy your file from HW1P1)
  - batchnorm.py (Copy your file from HW1P1)
  - linear.py (Copy the forward and backward methods from your solution)
  - optimizer.py
  - dropout.py
- hw1
  - hw1.py (Copy your file from HW1P1)
- autograder
  - hw1_autograder
    - runner.py
- create_tarball.sh

- **For** using code from Homework 1, ensure that you received all 100 points
- **Install** Python3, NumPy and PyTorch in order to run the local autograder on your machine:
  - `pip3 install numpy`
  - `pip3 install torch`
- **Hand-in** your code by running the following command from the top level directory, then **SUBMIT** the created *handin.tar* file to autolab:
  - `sh create_tarball.sh`
- **Autograde** your code by running the following command from the top level directory:
  - `python3 autograder/hw1_autograder/runner.py`
- **DO:**
  - We strongly recommend that you review the ADAM and Dropout papers.
- **DO NOT:**
  - Import any other external libraries other than numpy, as extra packages that do not exist in autolab will cause submission failures. Also do not add, move, or remove any files or change any file names.

2 ADAM [5 points]

Adam is a per-parameter adaptive optimizer that considers both the first and second moment of the current gradient. Implement the `adam` class in `mytorch/optimizer.py`.

At any time step t, Adam keeps a running estimate of the mean derivative for each parameter m_t and the mean squared derivative for each parameter v_t. t is initialized as 0 and m_t, v_t are initialized as 0 tensors. They are updated via:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

where g_t are the current gradients for the parameters.
β_1. The exponential decay rate for the first moment estimates (e.g. 0.9).
β_2. The exponential decay rate for the second-moment estimates (e.g. 0.999).
Then, as m_t and v_t are initialized as 0 tensors, they are biased towards 0 in earlier steps. As such, Adam corrects this with:

$$\hat{m}_t = m_t / (1 - \beta_1^t)$$

$$\hat{v}_t = v_t / (1 - \beta_2^t)$$

Lastly, the parameters are updated via

$$\theta_t = \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t + \epsilon})$$

where α is the learning rate. Recall that we intuitively divide $\hat{m}_t$ by $\sqrt{\hat{v}_t}$ in the last step to normalize out the magnitude of the running average gradient and to incorporate the second moment estimate.

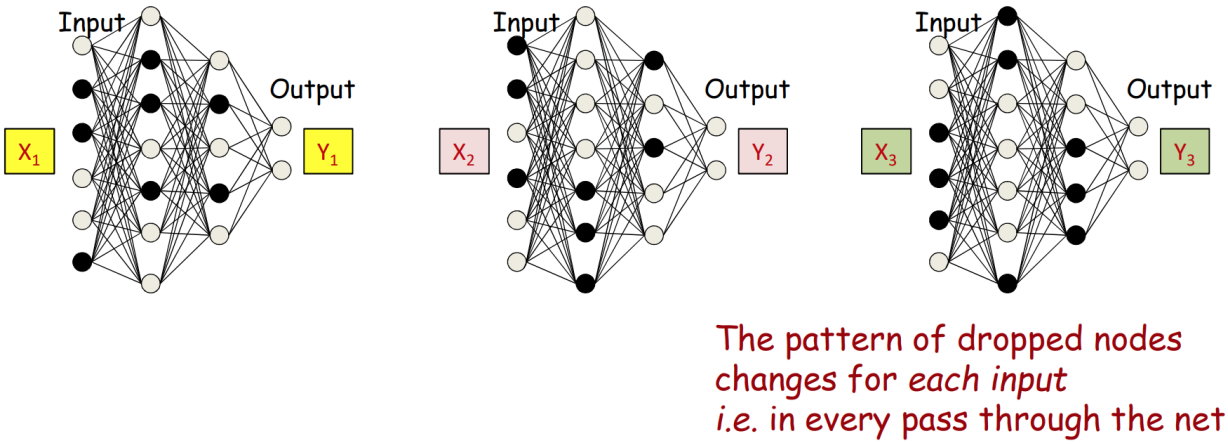
For more detailed explanations, we recommend that you reference the original paper as well as the formulae in lecture 7.

You need to keep a running estimate for some parameters related to W and b of the linear layer. We have defined instance variables inside our implementation of the Linear class that you can use.

3 Dropout [10 points]

Dropout is a regularization method that approximates ensemble learning of networks by randomly "turning off" neurons in a network during training. Implement the `Dropout` class in `mytorch/dropout.py`.

For every input, the neurons that are "turned off" are randomly chosen via a parameter p. The probability of zero-ing out a neruon output is then $1 - p$.



We can implement this by generating and applying a binary mask to the output tensor of a layer. As dropout zeros out a portion of the tensor, we need to re-scale the remaining numbers so the total "intensity" of the output is same as in testing, where we don't apply dropout.

For more detailed explanations, we recommend that you reference the paper.

Implementation Notes:

- You should use `np.random.binomial`
- You should scale during training and not during testing.