# Homework 1 Bonus

Adam, AdamW and Dropout

11-785: Introduction to Deep Learning (Spring 2024)

Out: **January 19, 2024**
Due: **March 2, 2024 11:59 PM EST**

## Start Here

- **Collaboration policy:**
  - You are expected to comply with the University Policy on Academic Integrity and Plagiarism.
  - You are allowed to help your friends debug
  - You are allowed to look at your friends code
  - You are allowed to copy math equations from any source that are not in code form
  - You are not allowed to type code for your friend
  - You are not allowed to look at your friends code while typing your solution
  - You are not allowed to copy and paste solutions off the internet
  - You are not allowed to import pre-built or pre-trained models
  - 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.

  We encourage you to meet regularly with your study group to discuss and work on the homework. You will not only learn more, you will also be more efficient that way. However, as noted above, the actual code used to obtain the final submission must be entirely your own.

- **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.
  - We recommend that you look through all of the problems before attempting the first problem. However we do recommend you complete the problems in order, as the difficulty increases, and questions often rely on the completion of previous questions.
- **Overview:**
  - Adam
  - AdamW
  - Dropout

# Homework objective

After this homework, you would ideally have learned:

- How to write code to implement different optimizers from scratch
  - How to implement Adam
  - How to implement AdamW
- How to write code to implement dropout from scratch
  - How to implement Forward Propagation with Dropout
  - How to implement Back Propagation with Dropout

# Contents

# 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:

- nn
  - __init__.py
  - loss.py (Copy your file from HW1P1)
  - activation.py (Copy your file from HW1P1)
  - batchnorm.py (Copy your file from HW1P1)
  - linear.py (Copy your file from HW1P1)
  - dropout.py
- models
  - __init__.py
  - mlp.py (Copy your file from HW1P1)
- optim
  - __init__.py
  - sgd.py (Copy your file from HW1P1)
  - adamW.py
  - adam.py

- **For** using code from Homework 1, ensure that you received all 100 autograded points
- **Install** Python3, NumPy and PyTorch in order to run the local autograder on your machine:
  - `pip3 install numpy`
  - `pip3 install torch`
- **Hand-in**
  - Use the terminal to run the command `"tar cvf handin.tar mytorch"` in your handout directory
  - This will create handin.tar
  - Submit handin.tar to AutoLab
- **Autograde** your code by running the following command from the top level directory:
  - `python3 hw1p1_bonus_autograder.py`
- **Handout**:
  - You can download the handout from autolab. The handout might have an extension like **handout.tar.112**. In such a case, you will have to first rename downloaded file as **handout.tar** by removing the **.112** extension and then untar the file.
- **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.

1st/2nd moment = mean/variance
Adjusting by the first moment is like momentum
it will scale this by the variance, so if we have high variance don't trust the momentum we build in the mean

## 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/optim/adam.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.

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:

Get momentum going early,

$$\hat{m}_t = m_t/(1 - \beta_1^t)$$
$$\hat{v}_t = v_t/(1 - \beta_2^t)$$

Lastly, the parameters are updated via

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

where $\alpha$ 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.

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 AdamW [5 points]

AdamW is an optimizer which uses weight decay regularization with Adam. Implement the `adamW` class in `mytorch/optim/adamW.py`.

If you implemented Adam, then the only additional parameter in AdamW is the weight decay. In this, we reduce the network parameter a portion of the model parameter at each time step.

$$W_t = W_t - W_{t-1} * \alpha * \lambda$$
$$b_t = b_t - b_{t-1} * \alpha * \lambda$$

Where $W_t$ and $b_t$ are your parameters after the standard Adam update in iteration $t$, $\lambda$ is weight decay and $\alpha$ is the learning rate. Alternatively, you can *first* perform the following two weight decay updates:

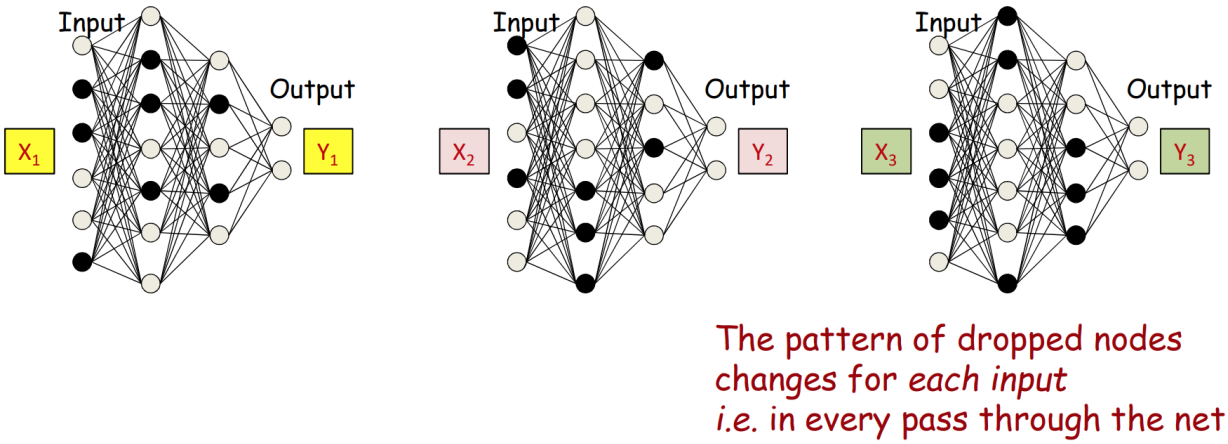
$$W_t = W_{t-1} - W_{t-1} * \alpha * \lambda$$
$$b_t = b_{t-1} - b_{t-1} * \alpha * \lambda$$

and then, add the standard Adam update to $W_t$ and $b_t$ obtained above using $W_{t-1}$ and $b_{t-1}$.
(**NOTE:** Observe the subscripts indicating the iteration number in the two presented ways).
For more detailed explanations, we recommend that you reference the original paper

# 4 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/nn/modules/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 neuron 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 by $1 - p$ 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.