Carnegie Mellon University

Homework 1 Recitation

SEPTEMBER 16, 2022

10-417 / 10-617 Intermediate Deep Learning

Reminder: AWS Credits Signup

- Please fill up the google form before September 19 at 5 PM
- You need a unique AWS account for this course
- If you don't do it before the deadline, you'll need to pay yourself for any AWS resource you'll need during the course
- Please refer to the Piazza post for more details

Agenda

1. Homework 1 overview
2. Probability review
3. Feed-forward neural network architecture
4. Backpropagation
5. Programming tips

Homework 1 Overview

- Released: Wednesday, September 14, 2022
- Due: Monday, October 3, 2022
- Written:
  - 4 questions for students in 10-417 (40 pts)
  - 5 questions for students in 10-617 (50 pts)
- Programming (60 pts)
  - Auto-grader (24 pts)
  - Experiments (36 pts)
- <u>Start early!</u>

Homework 1 Overview

Important instructions

- You may submit to Gradescope as often as you'd like before the deadline
- You have 5 late days for the semester, and you can use at most 3 per homework
    - If you are 1 minute late, that is considered 1 day
    - Your latest submission between written and programming is considered for the late days
- Write your answers in Latex within the solution boxes
- Do not change the location and dimensions of the solution boxes

Probability Review

Maximum Likelihood Estimate (MLE)

Given $\{y_i\}_{i=1}^{N}$ from a probability distribution P with parameter θ and probability density function $f_\theta(x)$, the MLE for θ is

$$\hat{\theta} = \arg\max_{\theta} \mathcal{L}\,(y_1, \dots, y_N)$$

Where the likelihood $\mathcal{L}$ is given by

$$\mathcal{L}(y_1, \dots, y_N) = \prod_{i=1}^{N} f_\theta\,(y_i)$$

It is often useful to consider the log-likelihood $\ell(y_1, \dots, y_N) = \log \mathcal{L}(y_1, \dots, y_N)$

Probability Review

Laplace distribution (or double exponential distribution)

- Denoted $Lap(\mu, b)$
- Probability density function

$$f(x; \mu, b) = \frac{1}{2b} e^{\frac{-|x-\mu|}{b}}$$

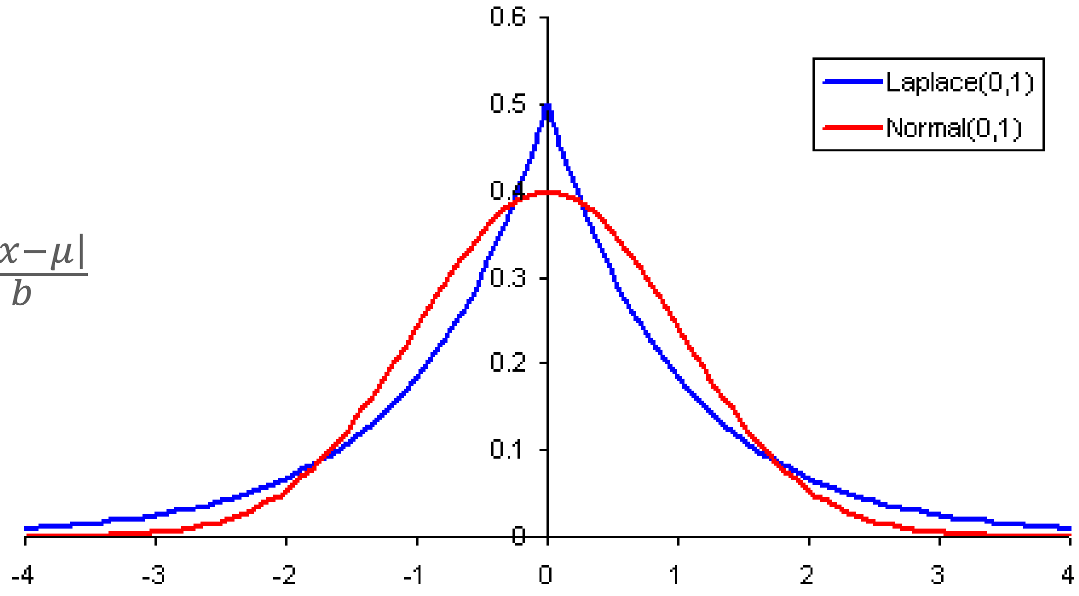


Image source: https://www.vosesoftware.com/riskwiki/Laplacedistribution.php

Probability Review

Expectation and variance

Let X be a random variable with support $\mathcal{X}$ and PMF/PDF f. Let $g: \mathcal{X} \to \mathbb{R}$ be a measurable function

$$\mathbb{E}[g(X)] = \begin{cases} \sum_{x \in \mathcal{X}} g(x) f(x), & \text{X is discrete} \\ \int_{\mathcal{X}} g(x) f(x)\, dx, & \text{X is continuous} \end{cases}$$

$$Var(X) = \mathbb{E}[(X - \mathbb{E}[X])^2]$$

Probability Review

Properties of Expectation and Variance

If X, Y are random variable and $a, b, c \in \mathbb{R}$

Linearity of Expectation

$$\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y]$$

For variance

$$Var(aX + bY + c) = a^2 Var(X) + b^2 Var(Y)$$

Decomposition of variance

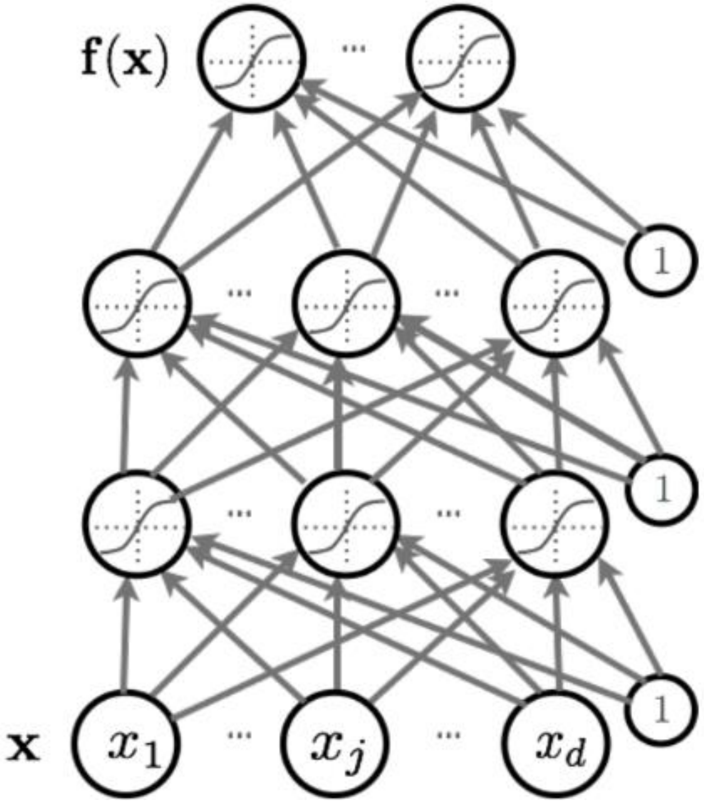
$$Var(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$$

Neural Network Architecture

What's the power of neural networks?

Play around with neural networks and see what they can do:

https://playground.tensorflow.org/



Neural Network Architecture

Activation functions

- Sigmoid

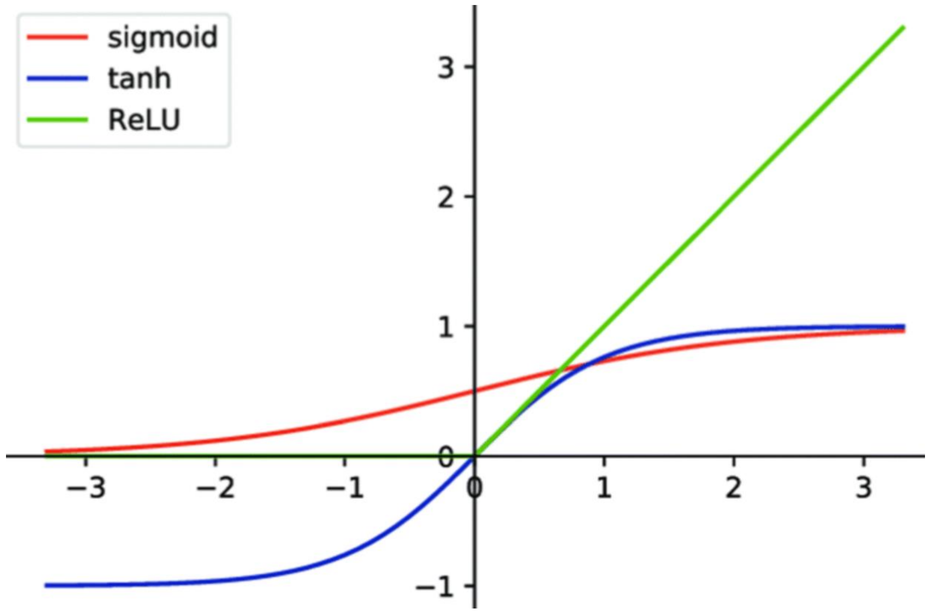
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- Hyperbolic tangent

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- Rectified Linear Unit

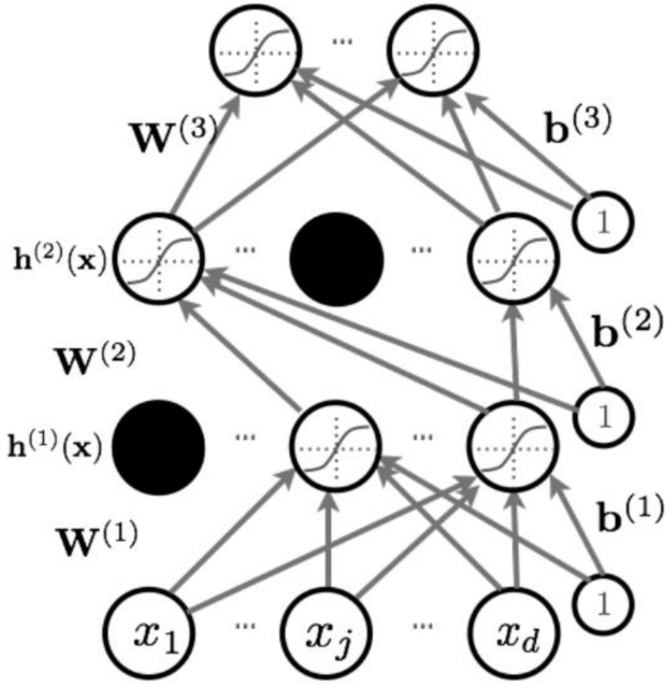
$$ReLU(x) = \max(0, x)$$



Neural Network Architecture

Dropout

- At training time, we drop a neuron with probability p .
- Helps prevent overfitting
  - Neurons might otherwise learn co-dependencies between them
- At test time, we don't dropout, instead we scale the output by $1 - p$



Neural Network Architecture

Momentum

Gradient descent without momentum

$$\theta_t \leftarrow \theta_{t-1} - \lambda \frac{\partial L}{\partial \theta_{t-1}}$$

Gradient descent with momentum

$$G_t \leftarrow \alpha G_{t-1} + \frac{\partial L}{\partial \theta_{t-1}}$$
$$\theta_t \leftarrow \theta_{t-1} - \lambda G_t$$

Multivariate Calculus

Recall: Chain Rule

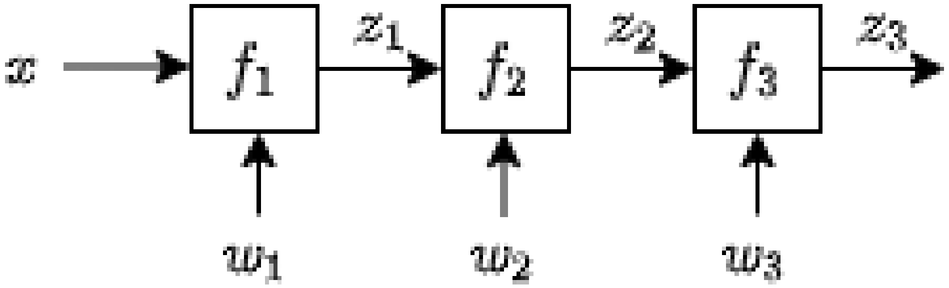
Let $y = f(z)$ and $z = g(x)$, then

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x}$$

Multivariate Calculus

Optimizing networks

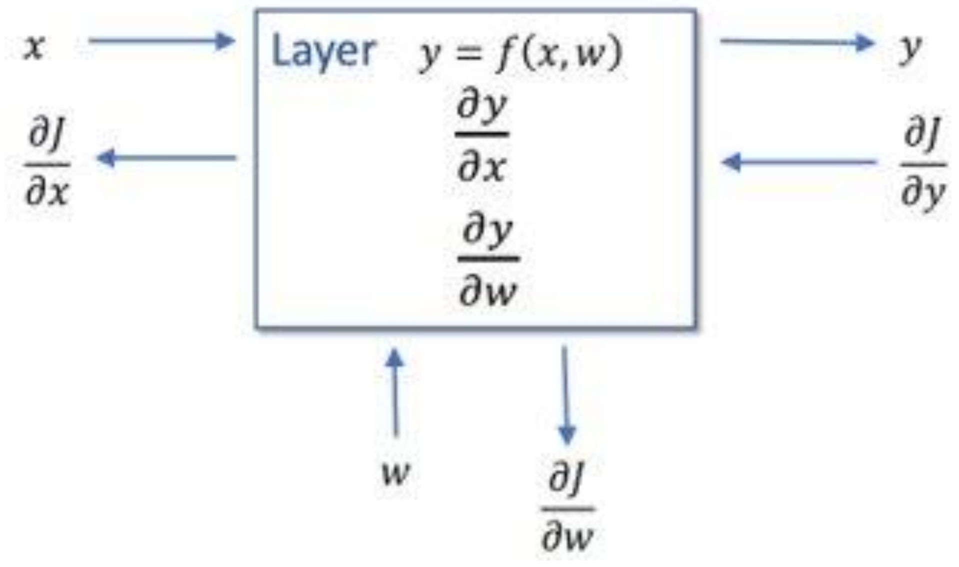
$$J(\boldsymbol{w}) = z_3$$
$$z_3 = f_3(w_3, z_2)$$
$$z_2 = f_2(w_2, z_1)$$
$$z_1 = f_1(w_1, x)$$



$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial z_3}\frac{\partial z_3}{\partial z_2}\frac{\partial z_2}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$

Backpropagation

- Forward pass: the layer takes in $\boldsymbol{x}$ and the weights $\boldsymbol{w}$ and outputs $\boldsymbol{y}$
- Backward pass: the layer takes in the derivative of the loss with respect to $\boldsymbol{y}$, $\frac{\partial J}{\partial \boldsymbol{y}}$, and outputs the derivative with respect to the weights, $\frac{\partial J}{\partial \boldsymbol{w}}$, and to its input, $\frac{\partial J}{\partial \boldsymbol{x}}$



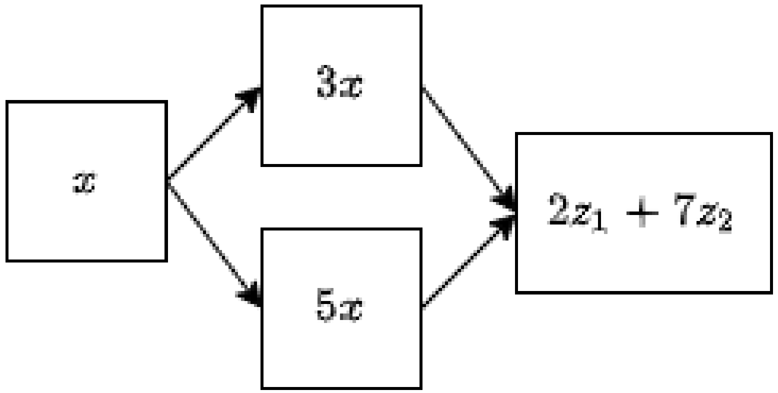
Carnegie Mellon University

Multivariate Calculus

Multivariate chain rule

$$g_1(x) = 3x$$
$$g_2(x) = 5x$$
$$f(z_1, z_2) = 2z_1 + 7z_2$$
$$y = f\big(g_1(x), g_2(x)\big)$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g_1}\frac{\partial g_1}{\partial x} + \frac{\partial f}{\partial g_2}\frac{\partial g_2}{\partial x}$$



Multivariate Calculus

Scalar to vector to scalar

$$y = f(\boldsymbol{z})$$
$$\boldsymbol{z} = g(x)$$

$$\frac{\partial y}{\partial x} = \sum_j \frac{\partial y}{\partial z_j} \frac{\partial z_j}{\partial x}$$

Vector to vector to vector

$$\boldsymbol{y} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = g(\boldsymbol{x})$$

$$\frac{\partial y_i}{\partial x_k} = \sum_j \frac{\partial y_i}{\partial z_j} \frac{\partial z_j}{\partial x_k}$$

Multivariate Calculus

Numerator-layout vs denominator layout

	Numerator layout	Denominator layout
Shape	# Outputs × # Inputs	# Inputs × # Outputs
Vector in, scalar out $y = f(\boldsymbol{x})$ $y \in \mathbb{R}, \boldsymbol{x} \in \mathbb{R}^M$	$\frac{\partial y}{\partial \boldsymbol{x}} \in \mathbb{R}^{1\times M}$	$\frac{\partial y}{\partial \boldsymbol{x}} \in \mathbb{R}^{M\times 1}$
Vector in, vector out $\boldsymbol{y} = f(\boldsymbol{x})$ $\boldsymbol{y} \in \mathbb{R}^N, \boldsymbol{x} \in \mathbb{R}^M$	$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \in \mathbb{R}^{N\times M}$	$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \in \mathbb{R}^{M\times N}$

Multivariate Calculus

Matrix in, scalar out

Dimensions of derivative are the same as the matrix

$$y = f(\boldsymbol{X})$$

$$y \in \mathbb{R}, \boldsymbol{X} \in \mathbb{R}^{N \times M}$$

$$\frac{\partial y}{\partial \boldsymbol{X}} \in \mathbb{R}^{N \times M}$$

Multivariate Calculus

Derivatives of Functions with Respect to Vectors

	Numerator layout	Denominator layout	Notes
$\frac{\partial}{\partial \mathbf{v}}\ \mathbf{v}$	I_N	I_N	$\mathbf{v} \in \mathbb{R}^N$
$\frac{\partial}{\partial \mathbf{v}}\ \mathbf{v}^T$	I_N	I_N	$\mathbf{v} \in \mathbb{R}^N$
$\frac{\partial}{\partial \mathbf{v}}\ t\mathbf{v}$	tI_N	tI_N	$\mathbf{v} \in \mathbb{R}^N$
$\frac{\partial}{\partial \mathbf{u}}\ \mathbf{u}^T\mathbf{v}$	$\mathbf{v}^T$	$\mathbf{v}$	
$\frac{\partial}{\partial \mathbf{v}}\ \mathbf{u}^T\mathbf{v}$	$\mathbf{u}^T$	$\mathbf{u}$	
$\frac{\partial}{\partial \mathbf{v}}\ \mathbf{v}^T\mathbf{v}$	$2\mathbf{v}^T$	$2\mathbf{v}$	
$\frac{\partial}{\partial \mathbf{v}}\ \mathbf{v}^TA\mathbf{v}$	$\mathbf{v}^T(A + A^T)$	$(A + A^T)\mathbf{v}$	
$\frac{\partial}{\partial \mathbf{v}}\ \mathbf{v}^TA\mathbf{v}$	$2\mathbf{v}^TA$	$2A\mathbf{v}$	If $A = A^T$
$\frac{\partial}{\partial \mathbf{v}}\ A\mathbf{v}$	A	A^T	
$\frac{\partial}{\partial \mathbf{v}}\ \mathbf{v}^TA$	A^T	A	

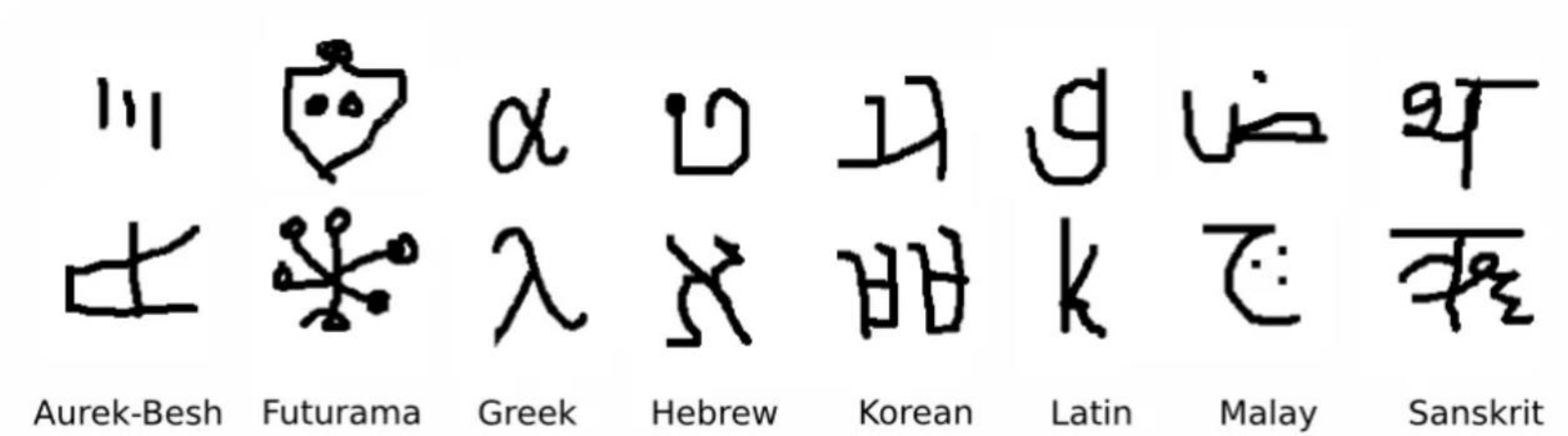
Multivariate Calculus

Make sure your dimensions make sense!

When in doubt, refer to the matrix cookbook: http://www.cs.toronto.edu/~bonner/courses/2012s/csc338/matrix_cookbook.pdf

Programming

Omniglot Dataset



Programming

Classes to implement

- ReLU (with dropout)
- LinearMap
- SoftmaxCrossEntropyLoss
- SingleLayerMLP
- TwoLayerMLP

Methods to implement

- forward()
- backward()
- step()
- zerograd()
- getW(), getb()
- loadparams()

The layers of the neural network will inherit from the Transform class.

You are being asked to implement the mlp.py file

```python
class Transform:
    """
    This is the base class. You do not need to change anything.

    Read the comments in this class carefully.
    """
    def __init__(self):
        """
        Initialize any parameters
        """
        pass

    def forward(self, x):
        """
        x should be passed as column vectors
        """
        pass

    def backward(self, grad_wrt_out):
        """
        In this function, we accumulate the gradient values instead of assigning
        the gradient values. This allows us to call forward and backward multiple
        times while only update parameters once.
        Compute and save the gradients wrt the parameters for step()
        Return grad_wrt_x which will be the grad_wrt_out for previous Transform
        """
        pass

    def step(self):
        """
        Apply gradients to update the parameters
        """
        pass

    def zerograd(self):
        """
        This is used to Reset the gradients.
        Usually called before backward()
        """
        pass
```

Programming

Training Loop

```
For epoch
    Shuffle training set
    For batch
        Reset gradients
        Forward pass
        Calculate loss & gradient
        Backward pass
        Apply gradients
```

Programming Tips

Use NumPy to vectorize operations!

- In Problem 4 (written part), you'll derive the matrix form of backpropagation
- Use NumPy to do matrix operations instead of using "for" loops (batch operations)
- You could get by without using NumPy for this homework, but that won't be possible in homework 2 so better get used to it now!