CS2109S Tutorial 9

Back Propagation (Last F2F Tutorial)

(AY 24/25 Semester 2)

April 11, 2025

(Prepared by Benson)

Contents

Recap. Back Propagation

Warm-Up Exercise: Let $\hat{\boldsymbol{y}} = \boldsymbol{W}^\top \boldsymbol{x}$. ($m\times 1$ $\quad m\times n$ $\quad n\times 1$)

Find $\dfrac{\partial \mathcal{E}}{\partial \boldsymbol{x}}$. ($n\times 1$)

$\boldsymbol{W}$: $n\times m$, $\frac{\partial \mathcal{E}}{\partial \hat{\boldsymbol{y}}}$: $m\times 1$

A. $\boldsymbol{W} \cdot \dfrac{\partial \mathcal{E}}{\partial \hat{\boldsymbol{y}}}$ $(n\times m)\cdot(m\times 1)$

B. $\boldsymbol{W}^\top \cdot \dfrac{\partial \mathcal{E}}{\partial \hat{\boldsymbol{y}}}$

C. $\dfrac{\partial \mathcal{E}}{\partial \hat{\boldsymbol{y}}} \cdot \boldsymbol{W}$

D. $\dfrac{\partial \mathcal{E}}{\partial \hat{\boldsymbol{y}}} \cdot \boldsymbol{W}^\top$

Find $\dfrac{\partial \mathcal{E}}{\partial \boldsymbol{W}}$. ($n\times m$)

$\boldsymbol{x}$: $n\times 1$, $\frac{\partial \mathcal{E}}{\partial \hat{\boldsymbol{y}}}$: $m\times 1$

A. $\boldsymbol{x} \cdot \left(\dfrac{\partial \mathcal{E}}{\partial \hat{\boldsymbol{y}}}\right)^\top$ $(n\times 1)\cdot(1\times m)$

B. $\boldsymbol{x}^\top \cdot \dfrac{\partial \mathcal{E}}{\partial \hat{\boldsymbol{y}}}$

C. $\left(\dfrac{\partial \mathcal{E}}{\partial \hat{\boldsymbol{y}}}\right)^\top \cdot \boldsymbol{x}$

D. $\dfrac{\partial \mathcal{E}}{\partial \hat{\boldsymbol{y}}} \cdot \boldsymbol{x}^\top$

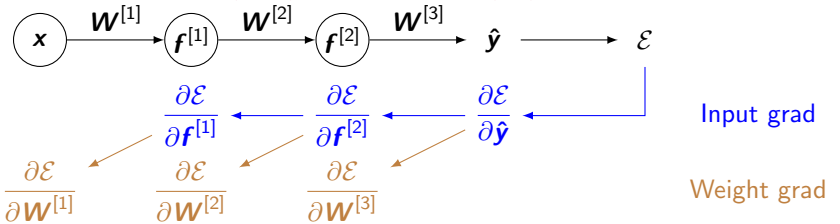
Recap. Back Propagation

Forward propagation helps us to evaluate $\hat{\boldsymbol{y}}$ in a neural network given $\boldsymbol{x}$. To perform gradient descent, we also need to evaluate $\dfrac{\partial \mathcal{E}}{\partial \boldsymbol{W}^{[i]}}$ for all layers (to update the weights).

$$\boldsymbol{f}^{[1]} = (\boldsymbol{W}^{[1]})^\top \boldsymbol{x} \qquad \boldsymbol{f}^{[2]} = (\boldsymbol{W}^{[2]})^\top \boldsymbol{f}^{[1]} \qquad \hat{\boldsymbol{y}} = (\boldsymbol{W}^{[3]})^\top \boldsymbol{f}^{[2]}$$

$$\frac{\partial \mathcal{E}}{\partial \boldsymbol{W}^{[1]}} = \boldsymbol{x} \left(\frac{\partial \mathcal{E}}{\partial \boldsymbol{f}^{[1]}} \right)^\top \qquad \frac{\partial \mathcal{E}}{\partial \boldsymbol{f}^{[1]}} = \boldsymbol{W}^{[2]} \frac{\partial \mathcal{E}}{\partial \boldsymbol{f}^{[2]}} \qquad \frac{\partial \mathcal{E}}{\partial \boldsymbol{f}^{[2]}} = \boldsymbol{W}^{[3]} \frac{\partial \mathcal{E}}{\partial \hat{\boldsymbol{y}}}$$

$$\frac{\partial \mathcal{E}}{\partial \boldsymbol{W}^{[2]}} = \boldsymbol{f}^{[1]} \left(\frac{\partial \mathcal{E}}{\partial \boldsymbol{f}^{[2]}} \right)^\top \qquad \frac{\partial \mathcal{E}}{\partial \boldsymbol{W}^{[3]}} = \boldsymbol{f}^{[2]} \left(\frac{\partial \mathcal{E}}{\partial \hat{\boldsymbol{y}}} \right)^\top$$



Recap. Back Propagation

Suppose a neural network takes time T for inference. What's the closest estimate of the training time using the same data?

A. T
B. $2T$
C. $3T$
D. $4T$
E. None of the other options

Solution: Forward propagation takes time T. Backward propagation requires time $2T$ (2 matrix multiplications are involved for input gradient & weight gradient). Hence the total time is $3T$.

Q1. Back Propagation

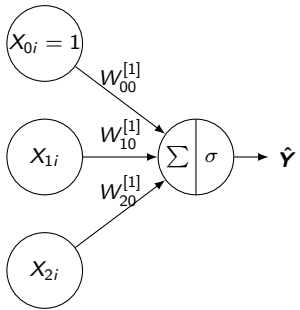
$$\mathcal{E} = -\frac{1}{n}\sum_{i=0}^{n-1}\Big\{[Y_{0i}\log(\hat{Y}_{0i})] + [(1-Y_{0i})\log(1-\hat{Y}_{0i})]\Big\}$$

$$\mathcal{E} = -\Big\{[Y_{00}\log(\hat{Y}_{00})] + [(1-Y_{00})\log(1-\hat{Y}_{00})]\Big\} \blacktriangleleft n=1$$

(i) When $n=1$, **show that** $\frac{\partial \mathcal{E}}{\partial \hat{\boldsymbol{Y}}} = \left[-\frac{Y_{00}}{\hat{Y}_{00}} + \frac{1-Y_{00}}{1-\hat{Y}_{00}}\right]$.

$$\frac{\partial \mathcal{E}}{\partial \hat{\boldsymbol{Y}}} = \left[\frac{\partial \mathcal{E}}{\partial \hat{Y}_{00}}\right] = \left[-\left(\frac{Y_{00}}{\hat{Y}_{00}} + \frac{1-Y_{00}}{1-\hat{Y}_{00}} \times (-1)\right)\right] = \left[-\frac{Y_{00}}{\hat{Y}_{00}} + \frac{1-Y_{00}}{1-\hat{Y}_{00}}\right]$$

($\log x \to \frac{1}{x}$; $\frac{\partial}{\partial \hat{Y}_{00}}(1-\hat{Y}_{00})$)



$$\boldsymbol{f}^{[1]} = (\boldsymbol{W}^{[1]})^\top \boldsymbol{X}$$
$$\hat{\boldsymbol{Y}} = g^{[1]}(\boldsymbol{f}^{[1]})$$

where $\boldsymbol{W}^{[1]} = \begin{bmatrix} W_{00}^{[1]} & W_{10}^{[1]} & W_{20}^{[1]} \end{bmatrix}^\top$,
$\boldsymbol{X} \in \mathbb{R}^{3\times n}$, and
$g^{[1]}(s) = \sigma(s) = \frac{1}{1+e^{-s}}$.

Q1. Back Propagation

(ii) When $n = 1$, **show that** $\frac{\partial \mathcal{E}}{\partial \boldsymbol{f}^{[1]}} = \hat{\boldsymbol{Y}} - \boldsymbol{Y}$.

Chain Rule

$$\frac{\partial \mathcal{E}}{\partial \boldsymbol{f}^{[1]}} = \begin{bmatrix} \frac{\partial \mathcal{E}}{\partial f_{00}^{[1]}} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathcal{E}}{\partial \hat{Y}_{00}} \cdot \frac{\partial \hat{Y}_{00}}{\partial f_{00}^{[1]}} \end{bmatrix}$$

$\hat{\boldsymbol{Y}}_{00} = \sigma(\boldsymbol{f}_{00}^{[1]})$

$\sigma'(x) = \sigma(x)(1 - \sigma(x))$

from (i)

$$= \begin{bmatrix} \left(-\frac{Y_{00}}{\hat{Y}_{00}} + \frac{1 - Y_{00}}{1 - \hat{Y}_{00}} \right) \cdot \hat{Y}_{00}(1 - \hat{Y}_{00}) \end{bmatrix}$$

$$= \begin{bmatrix} -Y_{00}(1 - \hat{Y}_{00}) + (1 - Y_{00})\hat{Y}_{00} \end{bmatrix}$$

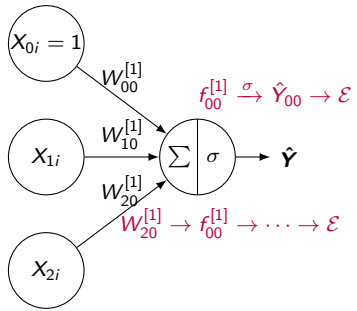
$$= \begin{bmatrix} -Y_{00} + \hat{Y}_{00} \end{bmatrix} = \hat{\boldsymbol{Y}} - \boldsymbol{Y}$$

(iii) When $n = 1$, **show that** $\frac{\partial \mathcal{E}}{\partial W_{20}^{[1]}} = \left(\frac{\partial \mathcal{E}}{\partial \boldsymbol{f}^{[1]}} \right)_{00} X_{20}$.

Chain Rule

$$\frac{\partial \mathcal{E}}{\partial W_{20}^{[1]}} = \frac{\partial \mathcal{E}}{\partial f_{00}^{[1]}} \cdot \frac{\partial f_{00}^{[1]}}{\partial W_{20}^{[1]}} = \left(\frac{\partial \mathcal{E}}{\partial \boldsymbol{f}^{[1]}} \right)_{00} X_{20}$$

$f_{00}^{[1]} = W_{20}^{[1]} \cdot X_{20} + \cdots$



$$\boldsymbol{f}^{[1]} = (\boldsymbol{W}^{[1]})^\top \boldsymbol{X}$$

$$\hat{\boldsymbol{Y}} = g^{[1]}(\boldsymbol{f}^{[1]})$$

where $\boldsymbol{W}^{[1]} = \begin{bmatrix} W_{00}^{[1]} & W_{10}^{[1]} & W_{20}^{[1]} \end{bmatrix}^\top$,

$\boldsymbol{X} \in \mathbb{R}^{3 \times n}$, and

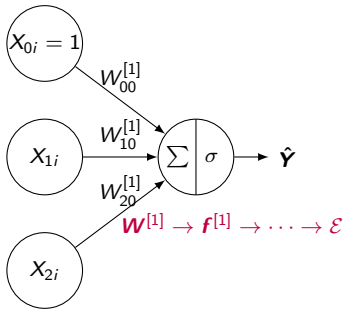
$g^{[1]}(s) = \sigma(s) = \frac{1}{1 + e^{-s}}$.

Q1. Back Propagation

(b) Derive an expression for $\dfrac{\partial \mathcal{E}}{\partial \boldsymbol{W}^{[1]}}$.

$$\boldsymbol{f}^{[1]} = (\boldsymbol{W}^{[1]})^\top \boldsymbol{X}$$

$$\frac{\partial \mathcal{E}}{\partial \boldsymbol{W}^{[1]}} = \underbrace{\frac{\partial \mathcal{E}}{\partial \boldsymbol{f}^{[1]}}}_{\text{from (ii)}} \cdot \frac{\partial \boldsymbol{f}^{[1]}}{\partial \boldsymbol{W}^{[1]}} = (\hat{\boldsymbol{Y}} - \boldsymbol{Y})\boldsymbol{X}$$



$$\boldsymbol{f}^{[1]} = (\boldsymbol{W}^{[1]})^\top \boldsymbol{X}$$

$$\hat{\boldsymbol{Y}} = g^{[1]}(\boldsymbol{f}^{[1]})$$

where $\boldsymbol{W}^{[1]} = \begin{bmatrix} W_{00}^{[1]} & W_{10}^{[1]} & W_{20}^{[1]} \end{bmatrix}^\top$,
$\boldsymbol{X} \in \mathbb{R}^{3 \times n}$, and
$g^{[1]}(s) = \sigma(s) = \frac{1}{1+e^{-s}}$.

Q1. Back Propagation

$$\mathcal{E} = -\frac{1}{n}\sum_{i=0}^{n-1}\left\{[Y_{0i}\log(\hat{Y}_{0i})] + [(1-Y_{0i})\log(1-\hat{Y}_{0i})]\right\}$$

(c) Consider a general case where $n \in \mathbb{N}$. Find $\dfrac{\partial \mathcal{E}}{\partial \boldsymbol{f}^{[1]}}$.

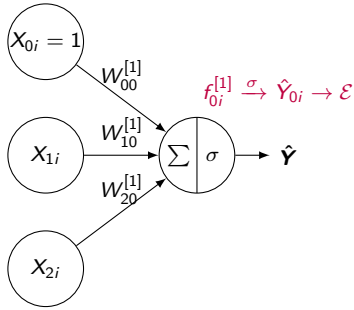
$$\frac{\partial \mathcal{E}}{\partial \hat{\boldsymbol{Y}}} = \left[\frac{\partial \mathcal{E}}{\partial \hat{Y}_{0i}}\right]_{i=0}^{n-1} = \left[\frac{1}{n}\left(-\frac{Y_{0i}}{\hat{Y}_{0i}} + \frac{1-Y_{0i}}{1-\hat{Y}_{0i}}\right)\right]_{i=0}^{n-1} \quad \text{from (i)}$$

Chain Rule

$$\frac{\partial \mathcal{E}}{\partial \boldsymbol{f}^{[1]}} = \left[\frac{\partial \mathcal{E}}{\partial f_{0i}^{[1]}}\right]_{i=0}^{n-1} = \left[\frac{\partial \mathcal{E}}{\partial \hat{Y}_{0i}} \cdot \frac{\partial \hat{Y}_{0i}}{\partial f_{0i}^{[1]}}\right]_{i=0}^{n-1} \quad \hat{Y}_{0i} = \sigma(f_{0i}^{[1]})$$

$$= \left[\frac{1}{n}\left(-\frac{Y_{0i}}{\hat{Y}_{0i}} + \frac{1-Y_{0i}}{1-\hat{Y}_{0i}}\right) \cdot \hat{Y}_{0i}(1-\hat{Y}_{0i})\right]_{i=0}^{n-1} \quad \sigma'(x) = \sigma(x)(1-\sigma(x))$$

$$= \left[\frac{1}{n}(-Y_{0i} + \hat{Y}_{0i})\right]_{i=0}^{n-1} = \frac{1}{n}(\hat{\boldsymbol{Y}} - \boldsymbol{Y})$$



$$\boldsymbol{f}^{[1]} = (\boldsymbol{W}^{[1]})^\top \boldsymbol{X}$$
$$\hat{\boldsymbol{Y}} = g^{[1]}(\boldsymbol{f}^{[1]})$$

where $\boldsymbol{W}^{[1]} = \begin{bmatrix} W_{00}^{[1]} & W_{10}^{[1]} & W_{20}^{[1]} \end{bmatrix}^\top$,
$\boldsymbol{X} \in \mathbb{R}^{3\times n}$, and
$g^{[1]}(s) = \sigma(s) = \frac{1}{1+e^{-s}}$.

Q1. Back Propagation

(d) Let's say that Grace has 100 samples from cultivar A and 1000 samples from cultivar B that make up the 1100 total samples. To deal with the imbalanced data set, she decided to introduce two hyper-parameters α and β in the loss function, as shown below:

$$\mathcal{E} = -\frac{1}{n}\sum_{i=0}^{n-1}\Big\{\alpha[Y_{0i}\cdot\log(\hat{Y}_{0i})] + \beta[(1-Y_{0i})\log(1-\hat{Y}_{0i})]\Big\}$$

Why do you think that she introduced the hyper-parameters α and β? How should she set their values?

- We would like the weight of cultivar A $\approx$ the weight of cultivar B.
- $\alpha = 10\beta$ (since the dataset has 10 times more samples from cultivar B than those from cultivar A).

Q2. Back Propagation for a Deep(er) Network

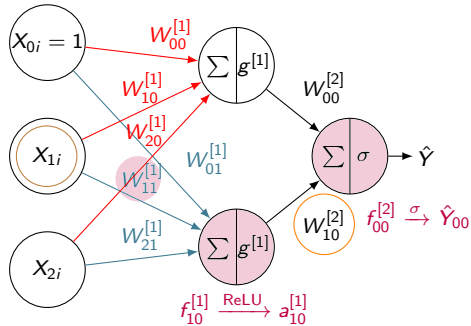
Compute $\frac{\partial \mathcal{E}}{\partial W_{11}^{[1]}}$.

$$\frac{\partial \mathcal{E}}{\partial W_{11}^{[1]}} = \underbrace{\frac{\partial \mathcal{E}}{\partial \hat{Y}_{00}}}_{\text{from (i)}} \cdot \underbrace{\frac{\partial \hat{Y}_{00}}{\partial f_{00}^{[2]}}}_{\sigma'(x)} \cdot \frac{\partial f_{00}^{[2]}}{\partial a_{10}^{[1]}} \cdot \underbrace{\frac{\partial a_{10}^{[1]}}{\partial f_{10}^{[1]}}}_{\text{ReLU}'(x)} \cdot \frac{\partial f_{10}^{[1]}}{\partial W_{11}^{[1]}}$$

$$= \left[-\frac{Y_{00}}{\hat{Y}_{00}} + \frac{1 - Y_{00}}{1 - \hat{Y}_{00}} \right] \cdot \hat{Y}_{00}(1 - \hat{Y}_{00}) \cdot W_{10}^{[2]}$$

$$\cdot \left(\begin{cases} 1 & \text{if } f_{10}^{[1]} > 0 \\ 0 & \text{otherwise} \end{cases} \right) \cdot X_{10}$$

$$= \begin{cases} (\hat{Y}_{00} - Y_{00}) W_{10}^{[2]} X_{10} & \text{if } f_{10}^{[1]} > 0 \\ 0 & \text{otherwise} \end{cases}$$

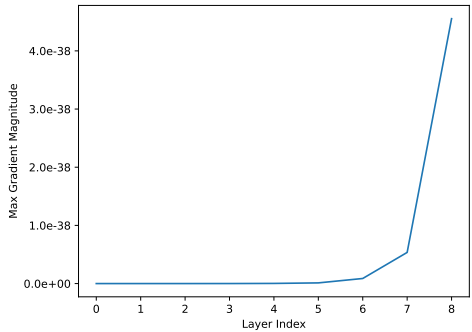


$$\boldsymbol{f}^{[1]} = (\boldsymbol{W}^{[1]})^\top \boldsymbol{X}$$
$$\boldsymbol{a}^{[1]} = g^{[1]}(\boldsymbol{f}^{[1]}) = \text{ReLU}(\boldsymbol{f}^{[1]})$$
$$\boldsymbol{f}^{[2]} = (\boldsymbol{W}^{[2]})^\top \boldsymbol{a}^{[1]}$$
$$\hat{\boldsymbol{Y}} = g^{[2]}(\boldsymbol{f}^{[2]}) = \sigma(\boldsymbol{f}^{[2]})$$

Q3. Training Deep Neural Networks

Suppose we use σ (the sigmoid function) as our activation function in a neural network with 50 hidden layers, as per the code in the accompanying Python notebook.

(a) Play around with the code. Notice that when performing back propagation, the gradient magnitudes of the first few layers are extremely small. What do you think causes this problem?

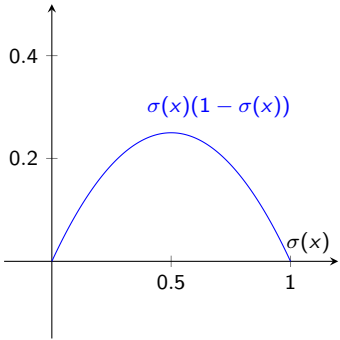


Q3. Training Deep Neural Networks

$$\frac{\partial \mathcal{E}}{\partial f^{[2]}} \sigma'(x) \frac{\partial \mathcal{E}}{\partial a^{[2]}} W^{[2]} \frac{\partial \mathcal{E}}{\partial f^{[3]}} \sigma'(x) \frac{\partial \mathcal{E}}{\partial a^{[3]}} W^{[3]} \frac{\partial \mathcal{E}}{\partial f^{[4]}} \sigma'(x) \frac{\partial \mathcal{E}}{\partial a^{[4]}} W^{[4]} \frac{\partial \mathcal{E}}{\partial f^{[5]}} \sigma'(x) \frac{\partial \mathcal{E}}{\partial a^{[5]}} \frac{\partial \mathcal{E}}{\partial \hat{Y}}$$

$$= \sigma(x)(1-\sigma(x)) \quad = \sigma(x)(1-\sigma(x)) \quad = \sigma(x)(1-\sigma(x)) \quad = \sigma(x)(1-\sigma(x))$$

$$\leq 0.25 \quad \leq 0.25 \quad \leq 0.25 \quad \leq 0.25$$



Vanishing gradient!

Issue: Convergence will be slow.

Q3. Training Deep Neural Networks

Suppose we use σ (the sigmoid function) as our activation function in a neural network with 50 hidden layers, as per the code in the accompanying Python notebook.

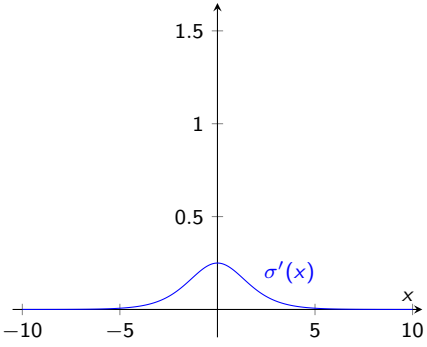
(b) Based on what we have learnt thus far, how can we **mitigate** this problem? Test out your solution by modifying the code and checking the gradient magnitudes.

Q3. Training Deep Neural Networks

Sigmoid Function:

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

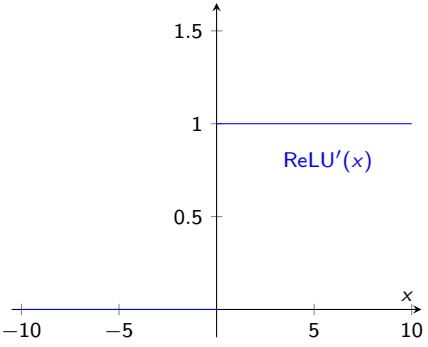
$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$



ReLU Function:

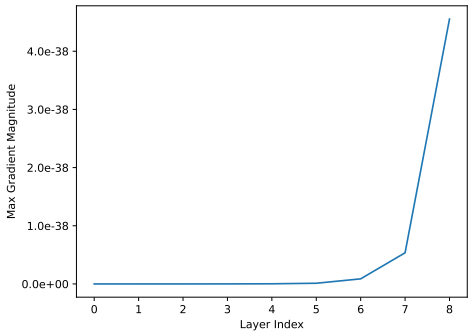
$$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{ReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

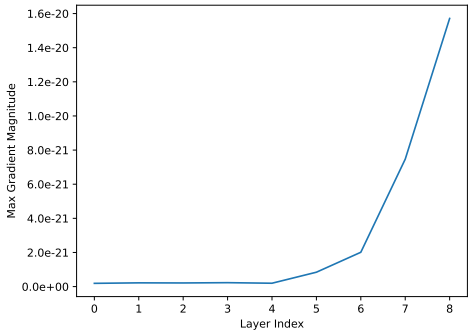


Q3. Training Deep Neural Networks

Sigmoid Function:

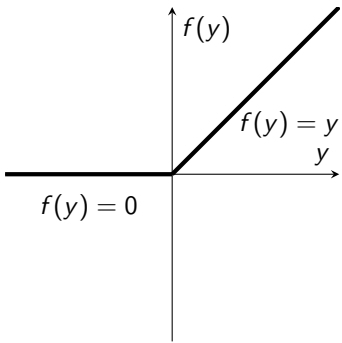


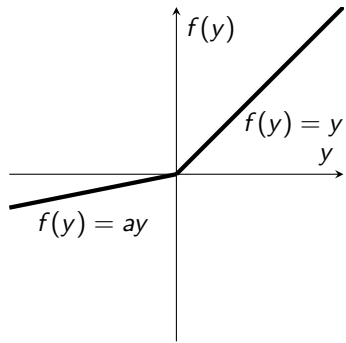
ReLU Function:



Q4. Dying ReLU Problem

This problem occurs when majority of the activations are 0 (meaning the underlying pre-activations are mostly non-positive), resulting in the network dying midway. The gradients passed back are also 0 which leads to poor gradient descent performance and hence poor learning. Refer to the figure below on the ReLU and Leaky ReLU activation functions.





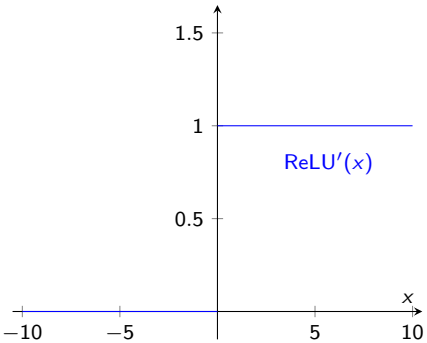
How does `Leaky ReLU` fix this? What happens if we set $a = 1$ in `Leaky ReLU`?

Q4. Dying ReLU Problem

ReLU Function:

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{ReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

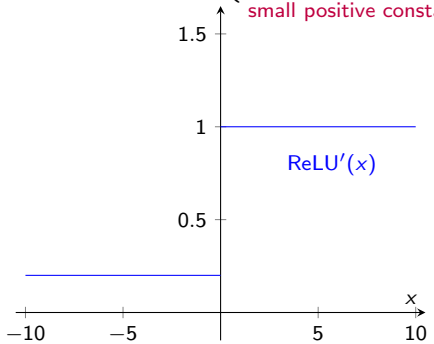


Leaky ReLU Function:

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{otherwise} \end{cases}$$

$$\text{ReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ a & \text{otherwise} \end{cases}$$

a: small positive constant



Beyond CS2109S

Introductory

CS2109S Intro to AI and ML

Cool 5k mods

CS5339 Theory and Algo for ML

Algos, e.g. Perceptron, SVM, Kernels
ML Theory, e.g. Conc. Measures, VC-dim
Prerequisite: CS3264

CS5275 Algo Designer's Toolkit

Math Tools for Algo/ML: Randomized, Optimization, Info Theory, and more
Prerequisite: CS3230

Theoretical Foundations

CS3263 Foundations of AI

↪ **CS4246** AI Decision Making

CS3264 Foundations of ML

Sem 1 (Harold): Math-intensive version of CS2109S (linear algebra?)
Sem 2 (Bryan): Focuses on proving / more "old-school" techniques

Applications

CS4243 Computer Vision

CS4248 Natural Language P.

Student Feedback Exercise:

Your Voice Matters!

Provide your feedback now >>

https://blue.nus.edu.sg/blue/

Bonus. Behind Pytorch Autograd (Practical)

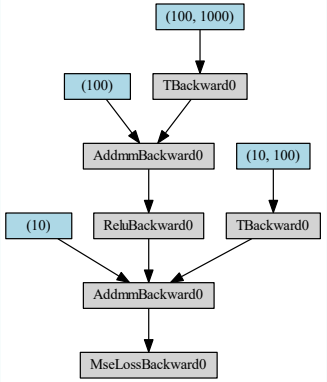
Extra Slide

Copy the template code from the website.
Complete the function `visualize_autograd_graph(loss)` to print out the Pytorch autograd graph. Get started by reading the provided starter code and the comments.

```python
# Sample neural network with 2 linear layers
class TwoLayerNet(nn.Module):
    def __init__(self, D_in, H, D_out):
        super(TwoLayerNet, self).__init__()
        self.linear1 = nn.Linear(D_in, H)
        self.relu = nn.ReLU()
        self.linear2 = nn.Linear(H, D_out)

    def forward(self, x):
        x = self.linear1(x)
        x = self.relu(x)
        x = self.linear2(x)
        return x

model = TwoLayerNet(1000, 100, 10)
loss = nn.MSELoss()
```



Bonus. Behind Pytorch Autograd (Practical)

Extra Slide

Solution.[1]

```
def add_nodes(var):
    if var in visited:
        return
    visited.add(var)

    if hasattr(var, "variable"):
        create_parameter_node(var, var.variable.size())
    else:
        create_func_node(var)

    if hasattr(var, "next_functions"):
        for u in var.next_functions:
            if u[0] is not None:
                create_edge(u[0], var)
                add_nodes(u[0])
```

[1] Adapted from https://www.kaggle.com/code/ehsanmjz/torchviz-example.