SUMMER SCHOOL 2020

MACHINE LEARNING DAY 2

DEEP LEARNING

Session III: Multi-Layer Perceptron

Isaac Ye, HPTC @ York University

Isaac@sharcnet.ca

Session III

- Running DL in Graham
- *Lab 3A - working in Graham and running a simple code*
- Binary classification
- Logistic model / cross entropy function
- Issue with linear regression
- XOR problem with Multi-Layer Perceptron (MLP)
- *Lab 3B: linear regression (multi-variables) with MLP*
- GPU on Graham / PyTorch + GPU
- *Lab 3C: running DL code on Graham using GPU*

Running DL in Graham

SHARCNET™

A consortium of 19 Ontario institutions providing advanced computing resources and support...

Shared
Hierarchical
Academic
Research
Computing
NETwork

compute canada

- Member of Compute Canada and Compute Ontario
- 3,000+ Canadian and international users

- **~50,000 CPU cores**
- **370+ GPUs**
- **10 Gb/s network**
- **100 Gb/s between national centres**

Virtual environment

Allows users to create virtual environments so that one can install Python modules easily

Many versions of same module are possible

```
[isaac@gra-login3 ~]$ module load python
[isaac@gra-login3 ~]$ module list

Currently Loaded Modules:
  1) nixpkgs/16.09   (S)      3) gcccore/.5.4.0  (H)   5) ifort/.2016.4.258 (H)   7) openmpi/2.1.1 (m)   9) python/3.7.4 (t)
  2) imkl/11.3.4.258 (math)   4) icc/.2016.4.258 (H)   6) intel/2016.4      (t)   8) StdEnv/2016.4 (S)

  Where:
   S:     Module is Sticky, requires --force to unload or purge
   m:     MPI implementations / Implémentations MPI
   math:  Mathematical libraries / Bibliothèques mathématiques
   t:     Tools for development / Outils de développement
   H:                Hidden Module



[isaac@gra-login3 ~]$ virtualenv --no-download ~/tf5
Using base prefix '/cvmfs/soft.computecanada.ca/easybuild/software/2017/Core/python/3.7.4'
New python executable in /home/isaac/tf5/bin/python
Installing setuptools, pip, wheel...
done.
[isaac@gra-login3 ~]$ source tf5/bin/activate
(tf5) [isaac@gra-login3 ~]$ deactivate
[isaac@gra-login3 ~]$
```

Lab 3A – Working in Graham

1. Log into graham.computecanada.ca with guest account and p/w

   (Use MobaXterm or Putty for Windows / Open terminal in Linux or Mac)

2. Load modules and make a virtual environment
   https://docs.computecanada.ca/wiki/Python#Creating_and_using_a_virtual_environment

```
module load python
module load scipy-stack
virtualenv --no-download ~/ENV
```

3. Activate, Upgrade 'PIP' and install 'PyTorch'
   https://docs.computecanada.ca/wiki/PyTorch#Installation

```
source ~/ENV/bin/activate
pip install --no-index --upgrade pip
pip install --no-index torch
pip install --no-index torch torchvision torchtext torchaudio
```

4. Getting out of virtual enviornment

```
deactivate
```

Lab 3A – Running simple code

Break room

1. Download *Lab2A_Linear_Reg_Vanilla.ipynb* as .py file from Google Colab
2. File transfer *Lab2A_Linear_Reg_Vanilla.py* to Graham using WinScp or MobaXterm (Windows) / sftp (Linux, Mac)
3. Activate virtual environment (make sure you load python and scipy-stack module)
4. Run it by 'python *Lab2A_Linear_Reg_Vanilla.py*'
5. Note you need to collect all import commands into the beginning of code using text editor (Nano/emacs/VI)

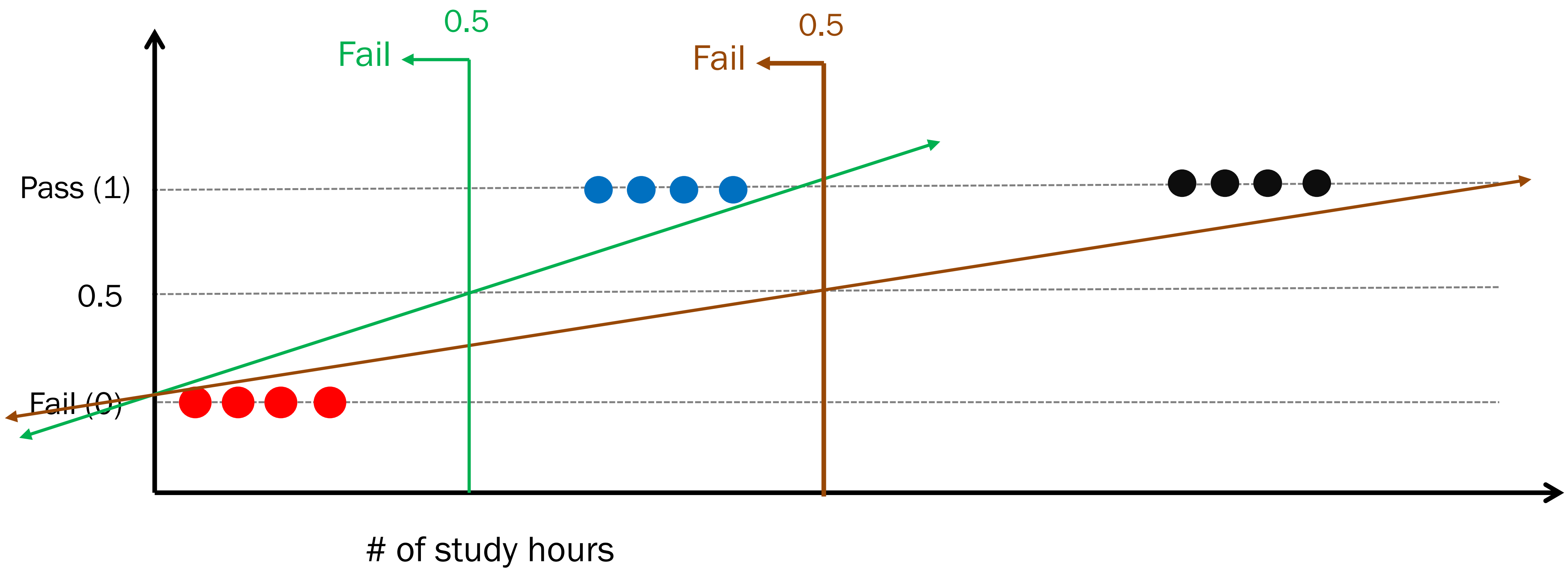
```
import matplotlib.pyplot as plt
import numpy as np
```

6. Note that you need to save/close your plots with proper filename for each plotting command like below

```
plt.scatter(X,Y)
plt.savefig('datascatter.png')
plt.close()
```

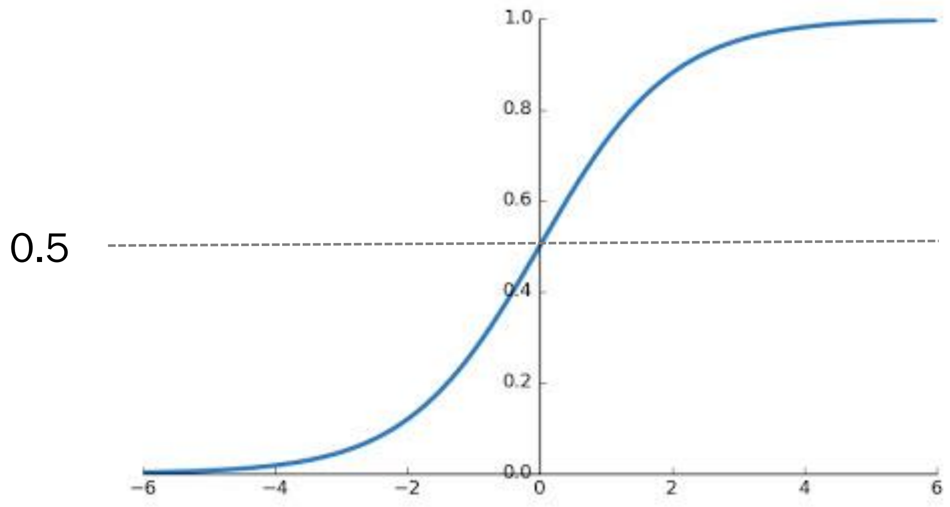
7. File transfer plotting files to your local computer using WinScp or MobaXterm (Windows) / sftp (Linux, Mac) and check it out

Binary classification



Linear regression is not good to solve binary problem!

Model: Logistic (Sigmoid) hypothesis



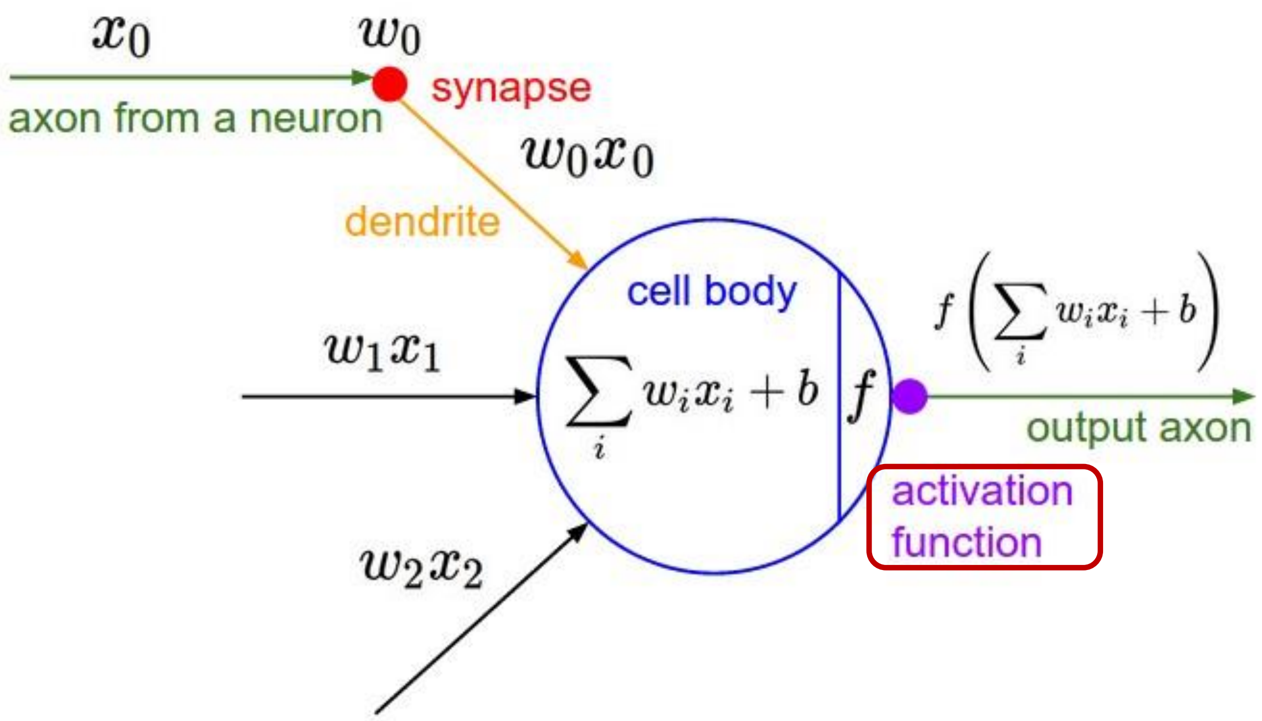
$$f(z) = \frac{1}{1 + e^{-z}}$$

$$H(x) = f(Wx + b)$$

$$z = Wx + b$$

$$H(z) = f(z)$$

Neural Network



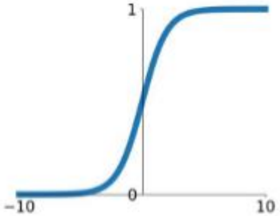
http://cs231n.github.io/neural-networks-1/

Mathematical model

Activation functions

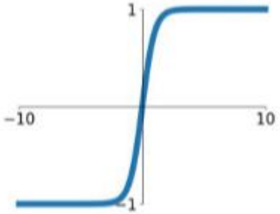
Sigmoid

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



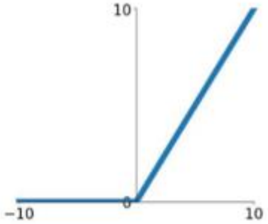
tanh

$\tanh(x)$



ReLU

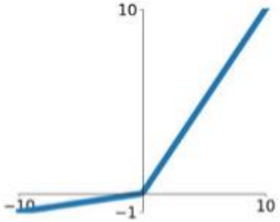
$\max(0, x)$



Most commonly used

Leaky ReLU

$\max(0.1x, x)$

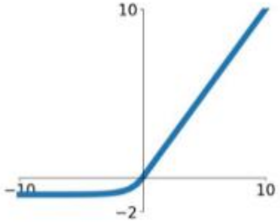


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



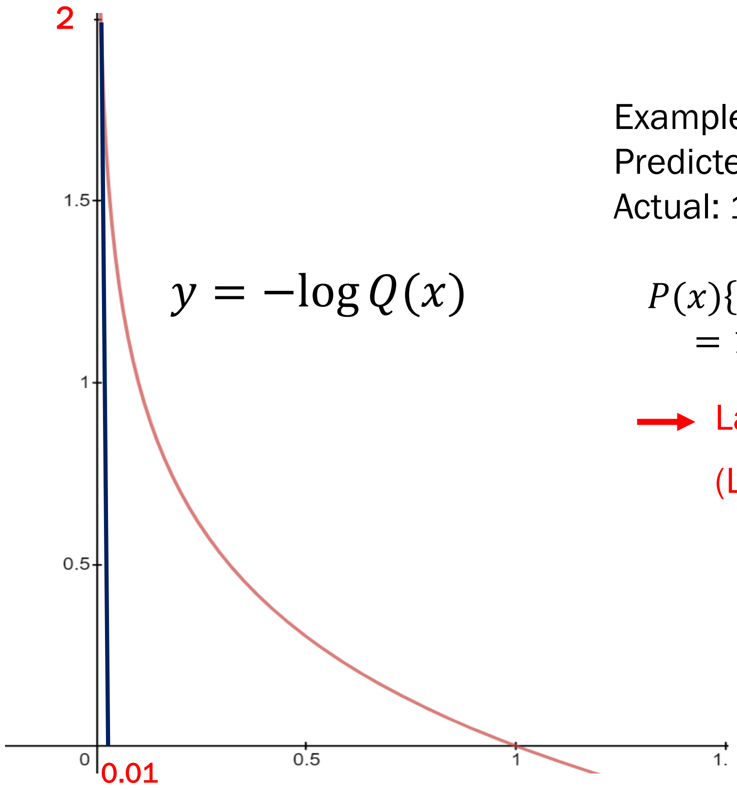
Cost function: Cross Entropy

Cross entropy: difference between two probability distribution

$$H(P,Q) = -\sum P(x)\log Q(x)$$

$P(x)$: actual probability

$Q(x)$: predicted probability



Example
Predicted: 0.01
Actual: 1

$$P(x)\{-\log Q(x)\} \\ = 1\{2\} = 2$$

→ Large entropy

(Large error)

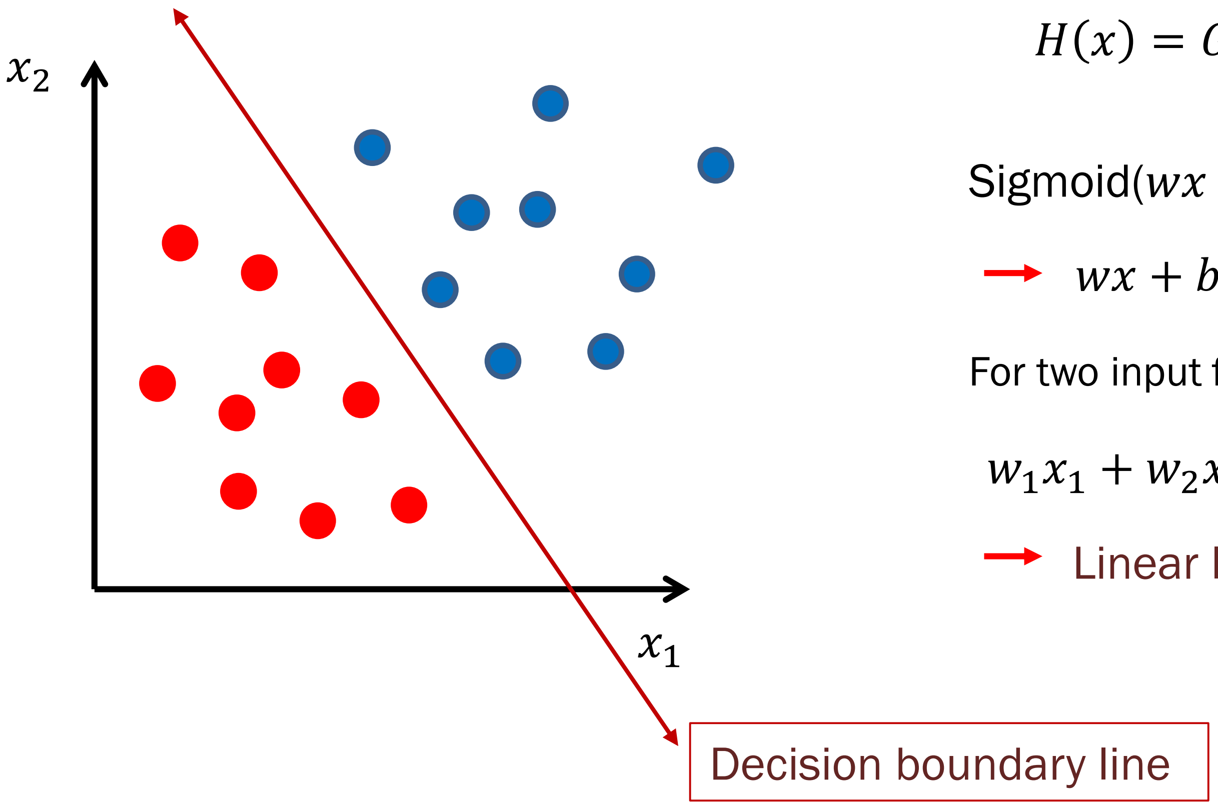
CROSSENTROPYLOSS

CLASS `torch.nn.CrossEntropyLoss(weight=None, size_average=None, ignore_index=-100, reduce=None, reduction='mean')` [SOURCE]

This criterion combines `nn.LogSoftmax()` and `nn.NLLLoss()` in one single class.

It is useful when training a classification problem with C classes. If provided, the optional argument `weight` should be a 1D *Tensor* assigning weight to each of the classes. This is particularly useful when you have an unbalanced training set.

Decision boundary



$$H(x) = G(Wx + b)$$

Sigmoid$(wx + b)$ = 0.5

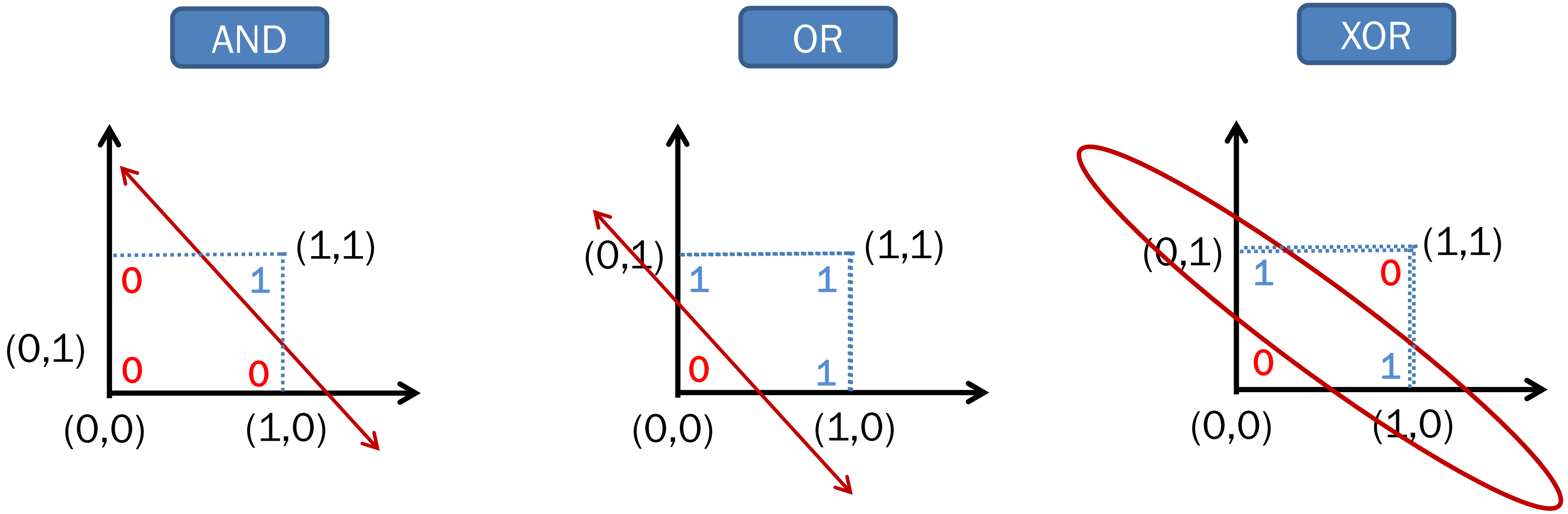
→ $wx + b = 0$

For two input feature problem, one can have

$$w_1x_1 + w_2x_2 + b = 0$$

→ Linear line!

XOR problem

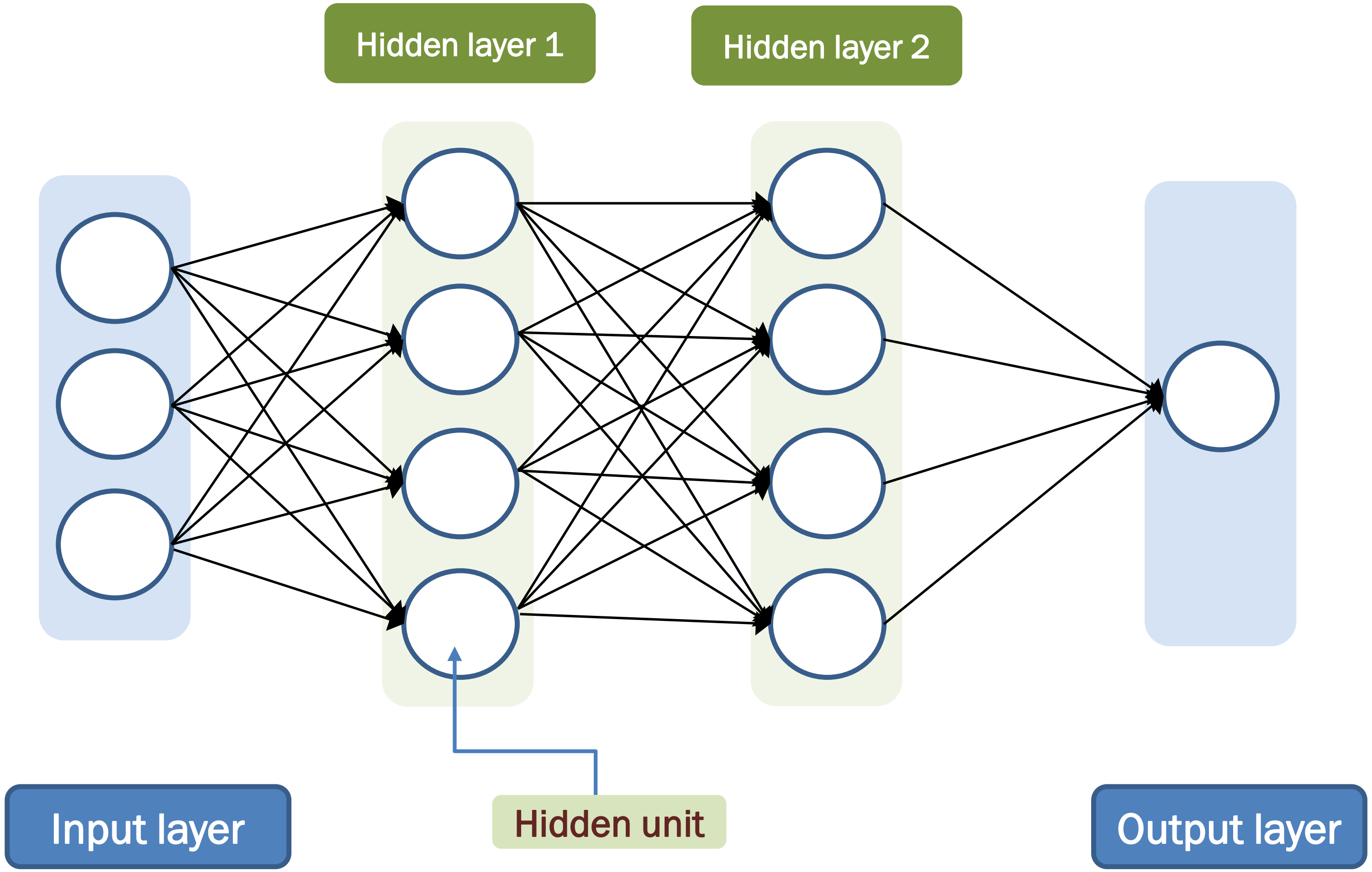


False: 0

True: 1

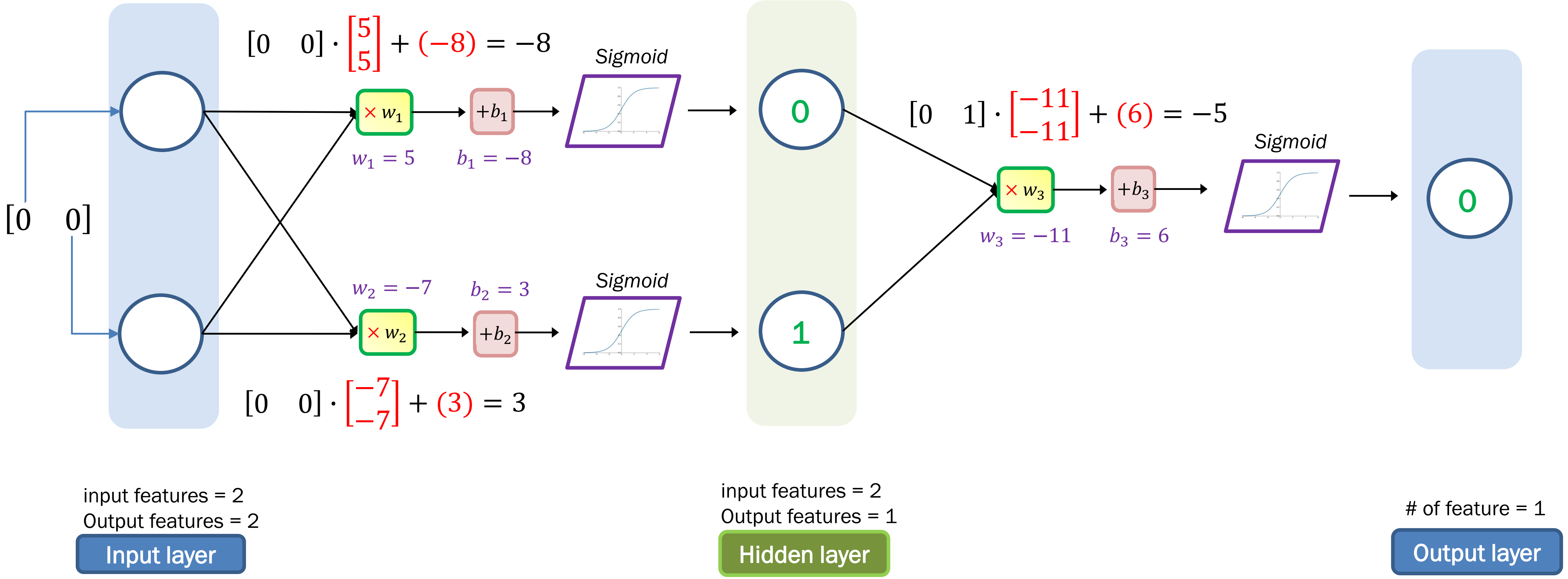
Fail to find a decision line !

Multi-Layer Perceptron



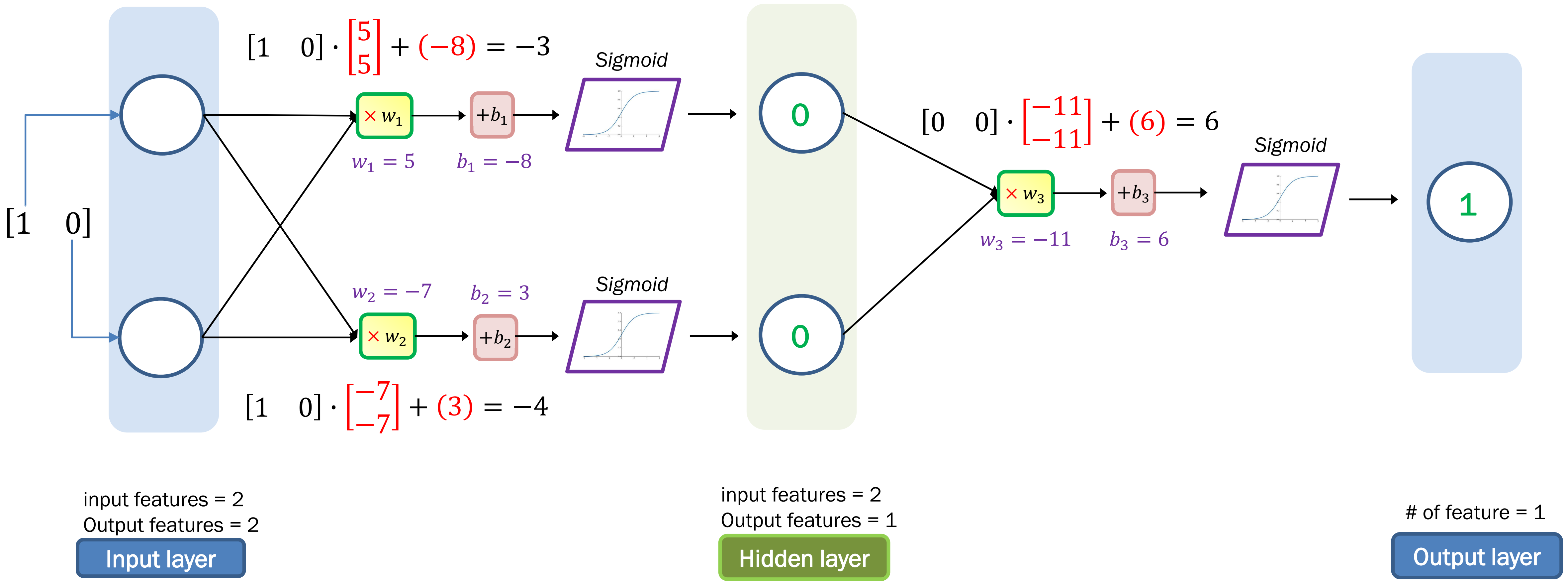
XOR problem

x_1	x_2	y_1	y_2	$\hat{y}$	y
0	0	0	1	0	0
1	0				1
0	1				1
1	1				0

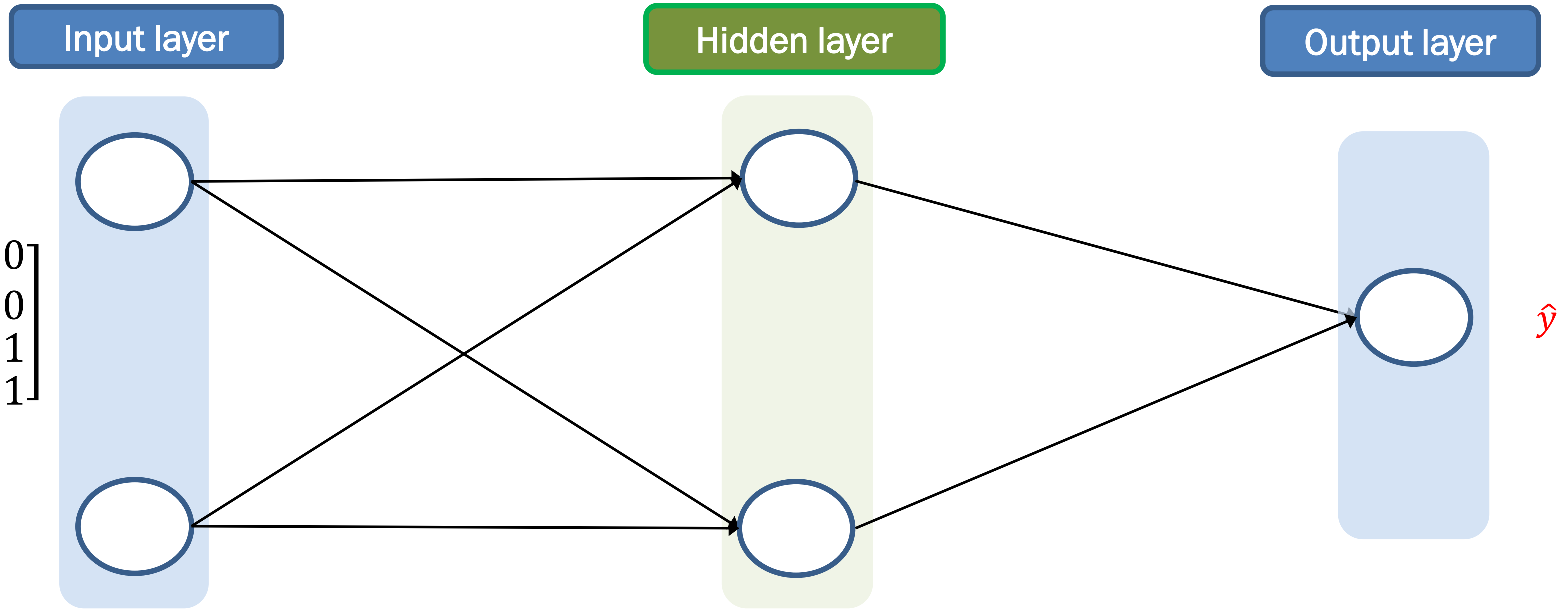


XOR problem

x_1	x_2	y_1	y_2	$\hat{y}$	y
0	0	0	1	0	0
1	0				1
0	1				1
1	1				0



XOR problem



$$X = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\hat{y} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

x_1	x_2	y_1	y_2	$\hat{y}$	y
0	0	0	1	0	0
1	0	0	0	1	1
0	1	0	0	1	1
1	1	1	0	0	0

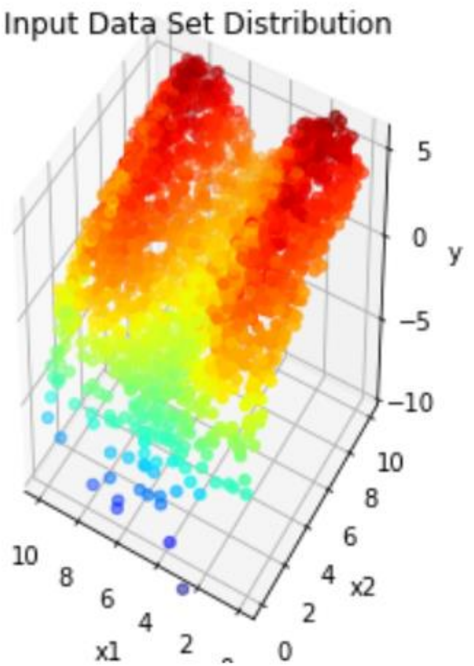
$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 5 & -7 \\ 5 & -7 \end{bmatrix} + \begin{bmatrix} -8 & 3 \\ -8 & 3 \\ -8 & 3 \\ -8 & 3 \end{bmatrix} = \begin{bmatrix} -8 & 3 \\ -3 & -4 \\ -3 & -4 \\ 2 & -11 \end{bmatrix} \xrightarrow{\text{Sigmoid}} \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}$$

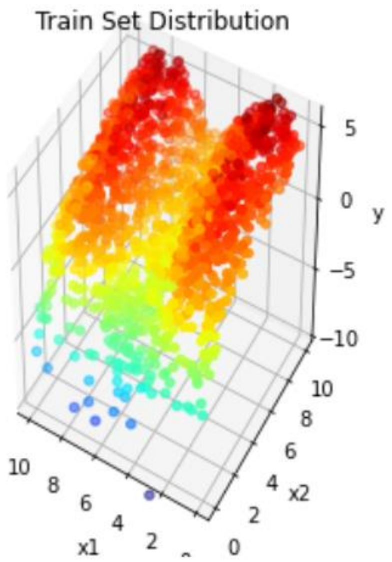
$$\begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} -11 \\ -11 \end{bmatrix} + \begin{bmatrix} 6 \\ 6 \\ 6 \\ 6 \end{bmatrix} = \begin{bmatrix} -5 \\ 6 \\ 6 \\ -5 \end{bmatrix} \xrightarrow{\text{Sigmoid}} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Data preparation

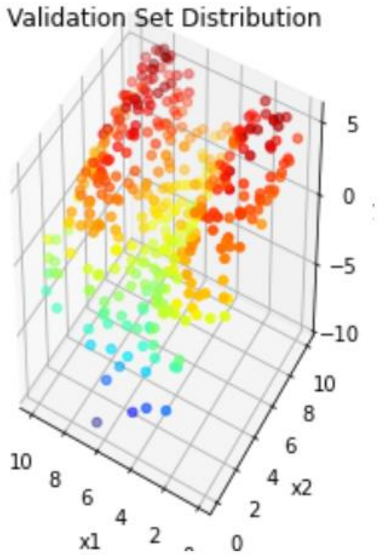
Data Preparation

x_1	x_2	y
3.91870851	2.32626914	0.73817558
2.59194437	6.00656071	4.3940048
6.46991632	3.57514815	0.61488728
:	:	:
4.56486433	2.14296641	3.95964088
1.29483514	1.67730041	3.48018992

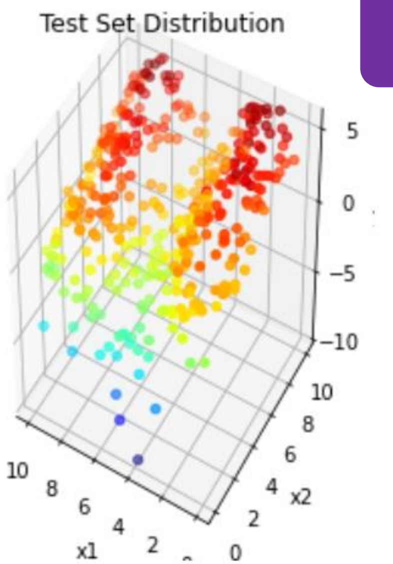




Train set



Validation set



Testing set

In the code

```
train_X, train_y = X[:1600, :], y[:1600]
val_X, val_y = X[1600:2000, :], y[1600:2000]
test_X, test_y = X[2000:, :], y[2000:]
```

Model define

Model define

In the code

```
import torch
import torch.nn as nn

class MLPModel(nn.Module):
    def __init__(self):
        super(MLPModel, self).__init__()
        self.linear1 = nn.Linear(in_features=2, out_features=200)
        self.linear2 = nn.Linear(in_features=200, out_features=1)
        self.relu = nn.ReLU()

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

Cost (loss) function + Optimizer

Cost function + optimizer

Loss function

In the code

```
cls_loss = nn.CrossEntropyLoss()
```

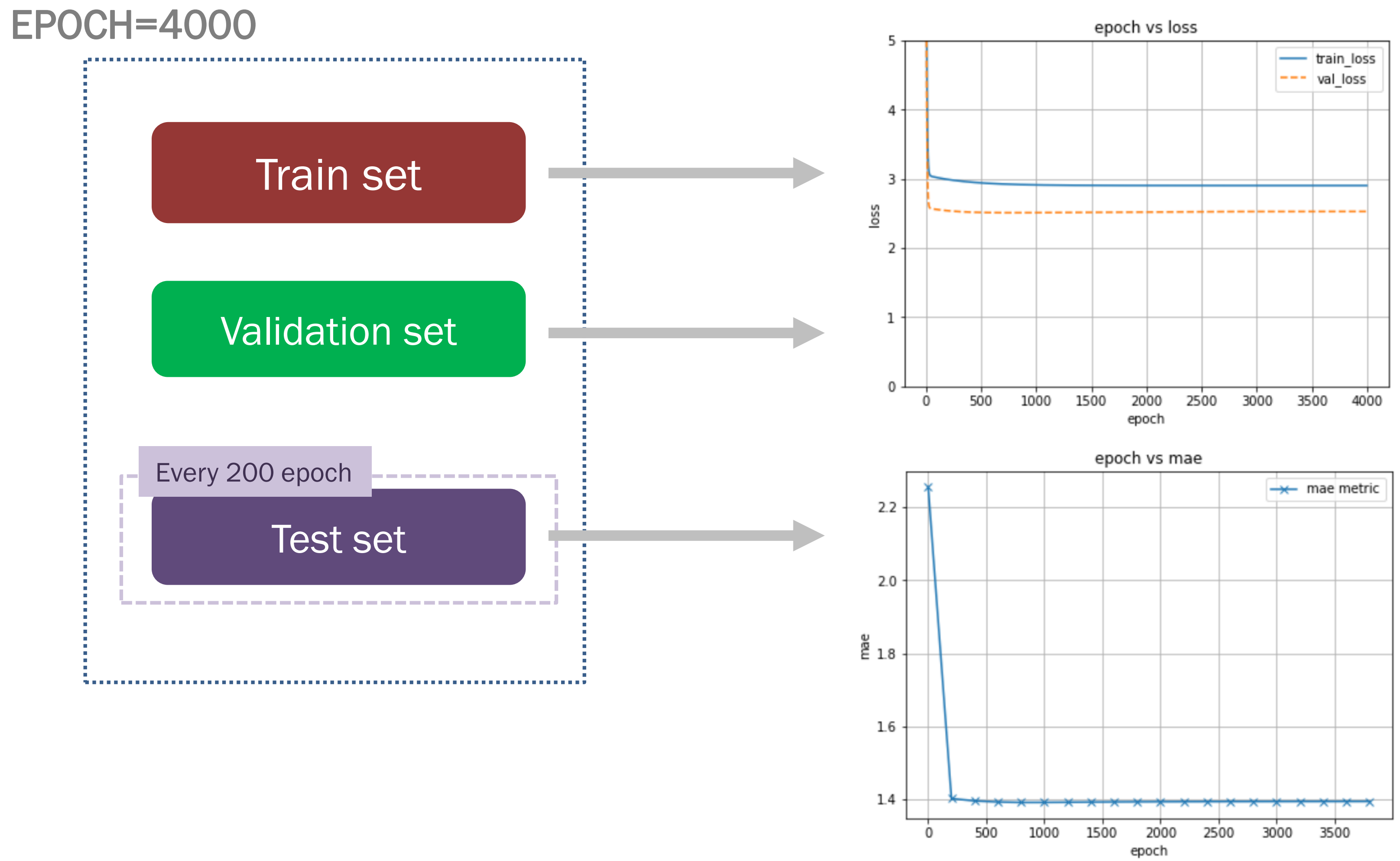
Optimizer

In the code

```
lr = 0.005
optimizer = optim.SGD(model.parameters(), lr =lr)
```

Model test

Model Test



Lab 3B: Linear regression – MLP

1. Check the model define (MLP)
2. Check the result by varying learning rate
3. Check the result with different number of Epoch
4. Check the result with more fully connected layers

   /different number of hidden units
5. Check the result with different activation functions(Sigmoid, ReLU, Leaky ReLU)
6. Check the result with different loss function

Break room

Lab 3B – Running it on Graham (Interactive mode)

1. Download *Lab3B_Linear_Reg_MLP.ipynb* as .py file from Colab
2. File transfer *Lab3B_Linear_Reg_MLP.py* to Graham using WinScp or MobaXterm (Windows) / sftp (Linux, Mac)
3. Start interactive running mode

   salloc --time=0:30:0 --ntasks=1 --cpus-per-task=3 --nodes=1 --mem=1000M --account=def-training-wa

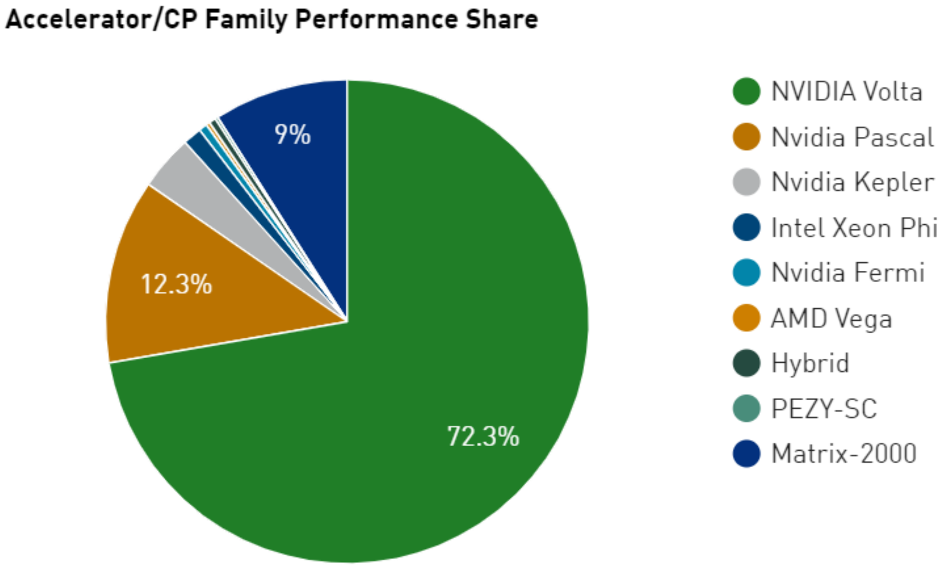
4. Activate virtual environment (make sure you load python and scipy-stack module)
5. Run it by 'python *Lab3B_Linear_Reg_MLP.py*'
6. Note you need to collect all import commands into the beginning of code using text editor (Nano/emacs/VI)

6. Note that you need to save/close your plots with proper filename for each plotting command like below

7. File transfer plotting files to your local computer using WinScp or MobaXterm (Windows) / sftp (Linux, Mac) and check it out

DL and HPC architectures

NVIDIA GPUs are the main driving force for faster training DL models



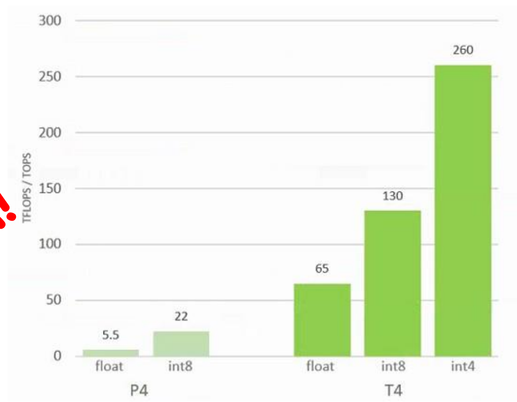
https://top500.org

NVIDIA T4 Turing GPU on Graham

Latest addition!

Streaming Multiprocessor
8 Tensor cores

40 SM in T4
8.1 Tflops FP32
65 Tflops FP16



GPU resources in Compute Canada

As of Feb, 2020

	# of nodes	GPU type	Note
Graham	160	P100 Pascal	--gres=gpu:1
	7	V100 Volta	CPU/GPU ≤ 3.5 --gres=gpu:v100:1
	36	**T4 Turing (for DL)**	CPU/GPU ≤ 3.5 --gres=gpu:t4:2
Cedar	146	P100 Pascal	--gres=gpu:1
Beluga	172	V100 Volta	CPU/GPU ≤ 3.5 --gres=gpu:v100:1
Niagara	None		

PyToch + GPU

Using GPU, one can reduce runtime!

Check if PyToch recognize ‘GPU’

```
# ====== GPU selection ======= #
device = 'cuda:0' if torch.cuda.is_available() else 'cpu'
model.to(device)
```

Put data to the detected device

```
if device != 'cpu':
    input_x = input_x.to(device)
    true_y = true_y.to(device)
```

Move data back to ‘CPU’

```
if device != 'cpu':
    list_train_loss.append(loss.cpu().detach().numpy())
else:
    list_train_loss.append(loss.detach().numpy())
```

Lab 3C: Linear regression – MLP w/GPU

1. Compare the running time of the training code block for CPU and GPU
2. You may increase size of data, # of Epoch, # of linear layers and # of hidden units to compare the result

Break room

Lab 3C – Running it on Graham (Interactive mode)

1. Copy *Lab3C_Linear_Reg_MLP_GPU.py* from /home/isaac/SS20_ML_Day2

   `cp /home/isaac/SS20_ML_Day2/Lab3C_linear_Reg_MLP_GPU.py /home/$USER`

2. Start interactive running mode with T4 GPU in Graham

   `salloc --time=0:30:0 --ntasks=1 --cpus-per-task=3 --gres=gpu:t4:1 --nodes=1 --mem=1000M --account=def-training-wa_gpu`

3. Activate virtual environment (make sure you load python and scipy-stack module)

4. Run it by 'python *Lab3C_Linear_Reg_MLP_GPU.py*'

6. File transfer plotting files to your local computer using WinScp or MobaXterm (Windows) / sftp (Linux, Mac) and check it out

Lab 3C – Running it on Graham (batch mode)

1. **Write** a submission script *'job_s.sh'* like below text editor

```
#!/bin/bash
#
#SBATCH --nodes=1
#SBATCH --gres=gpu:t4:1
#SBATCH --cpus-per-task=3
#SBATCH --mem=20000M
#SBATCH --time=0-30:00
#SBATCH --account=def-training-wa_gpu
#SBATCH --output=slurm.%x.%j.out

module load python scipy-stack
source ~/ENV/bin/activate
python Lab3C_Linear_Reg_MLP_GPU.py
```

2. **Submit** it by typing *'sbatch job_s.sh'*
3. **Check** it by typing *'squeue -u $USER'*

Session break:

Please come back by 3:45 PM