SUMMER SCHOOL 2020

MACHINE LEARNING DAY 2

DEEP LEARNING

Session III: Multi-Layer Perceptron

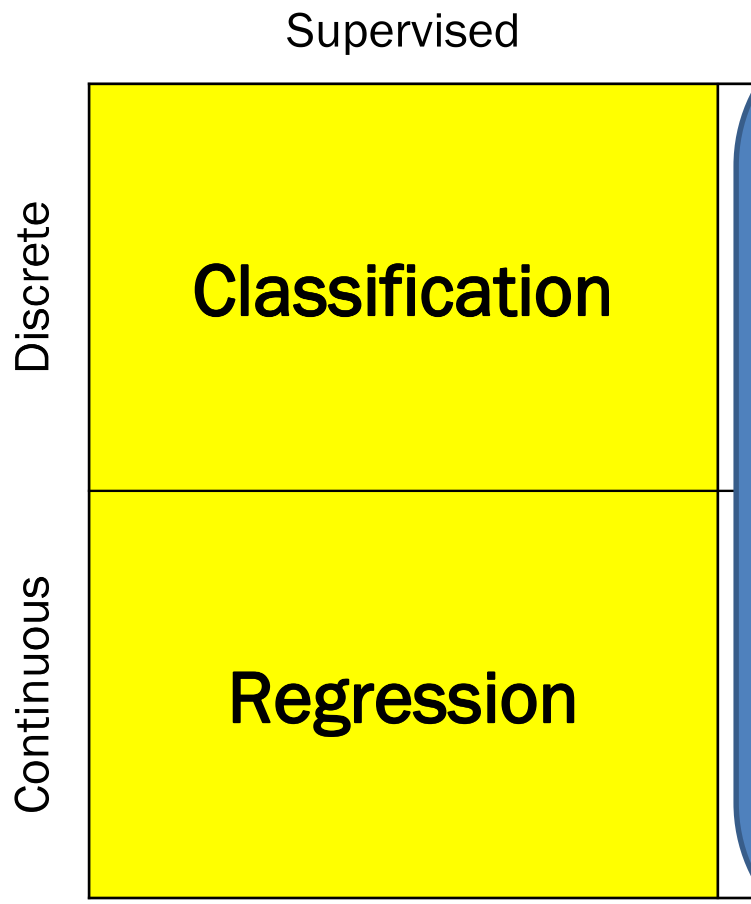
Isaac Ye, HPTC @ York University

Isaac@sharcnet.ca

Session III

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

Supervised learning



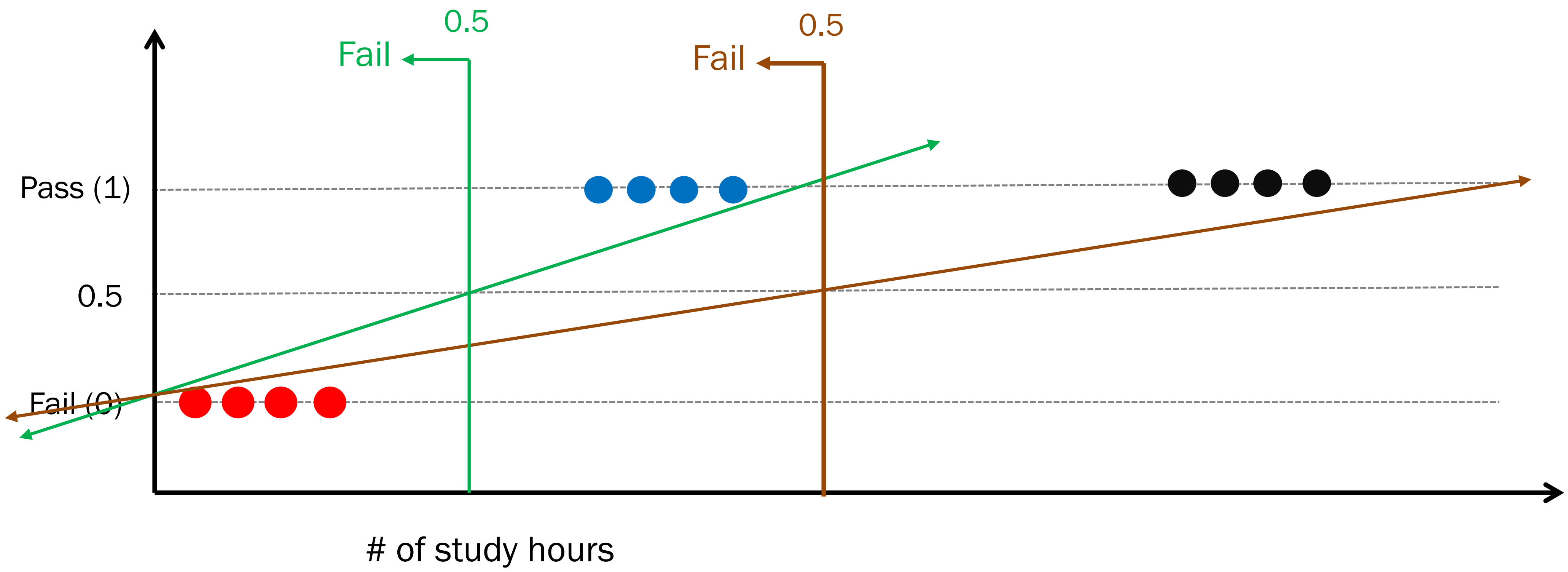
Supervised

Input	Data (x, y) x: data, y: label
Goal	learn a function to map x to y
Examples	classification, regression, object detection semantic segmentation, image captioning

Categories of ML problems

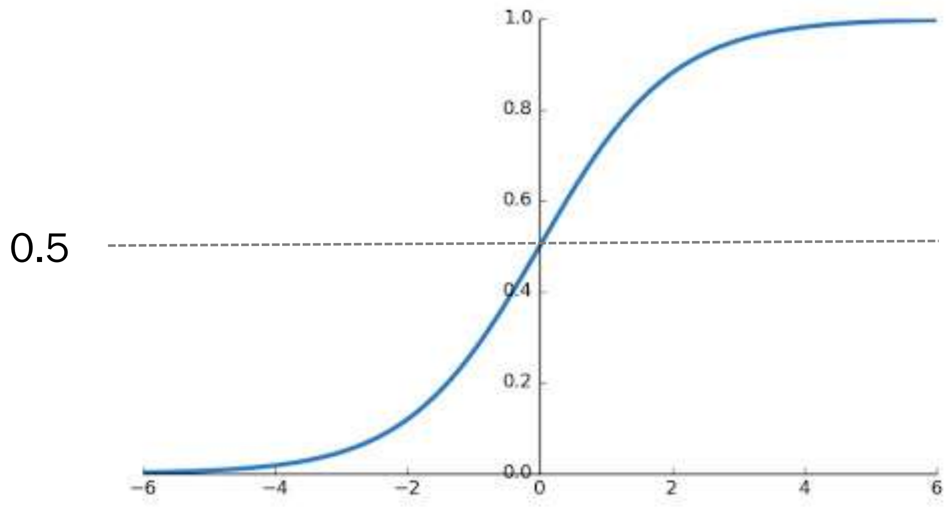
	Supervised	Unsupervised	Reinforcement
Discrete	Classification	Clustering	Action space agent
Continuous	Regression	Dimensionality reduction	Action space agent

Binary classification



Linear regression is not good to solve binary problem!

Model: Logistic (Sigmoid) hypothesis



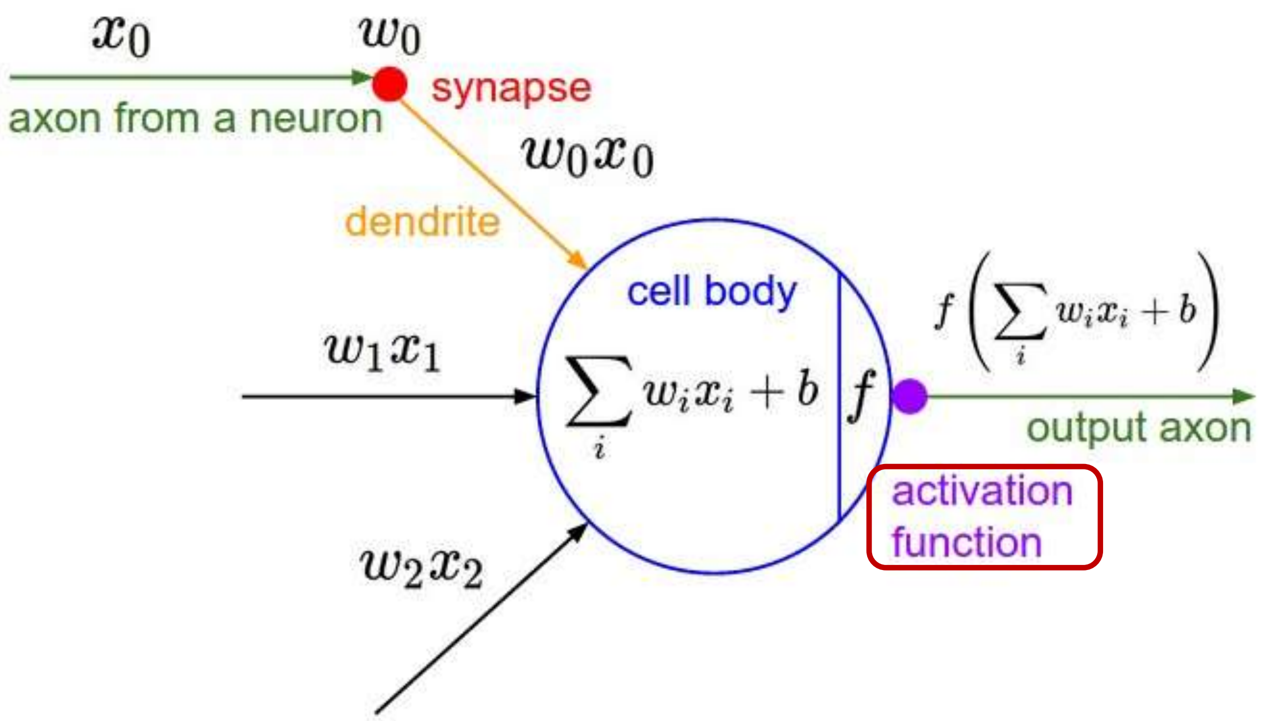
$$H(x) = f(Wx + b)$$

$$z = Wx + b$$

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

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

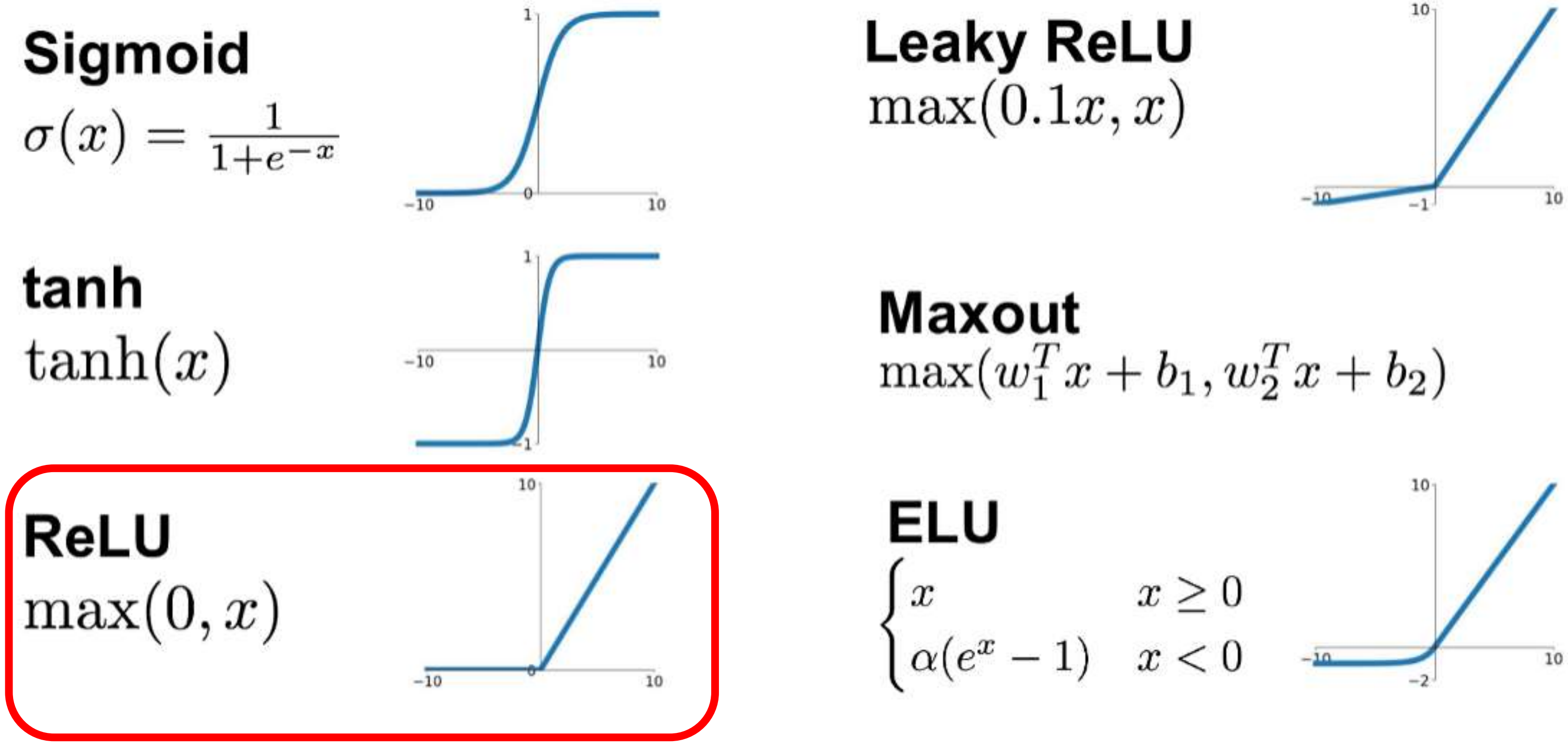
Neural Network



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

Mathematical model

Activation functions



Most commonly used

Cost function: Cross Entropy

Cross entropy: difference between two probability distribution

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

$P(x)$: actual probability

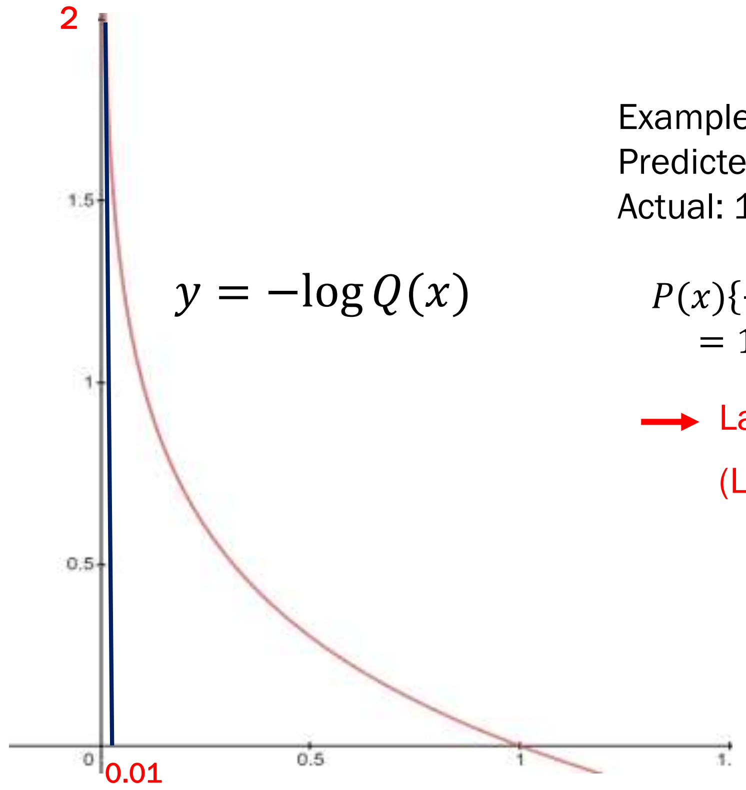
$Q(x)$: predicted probability

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.

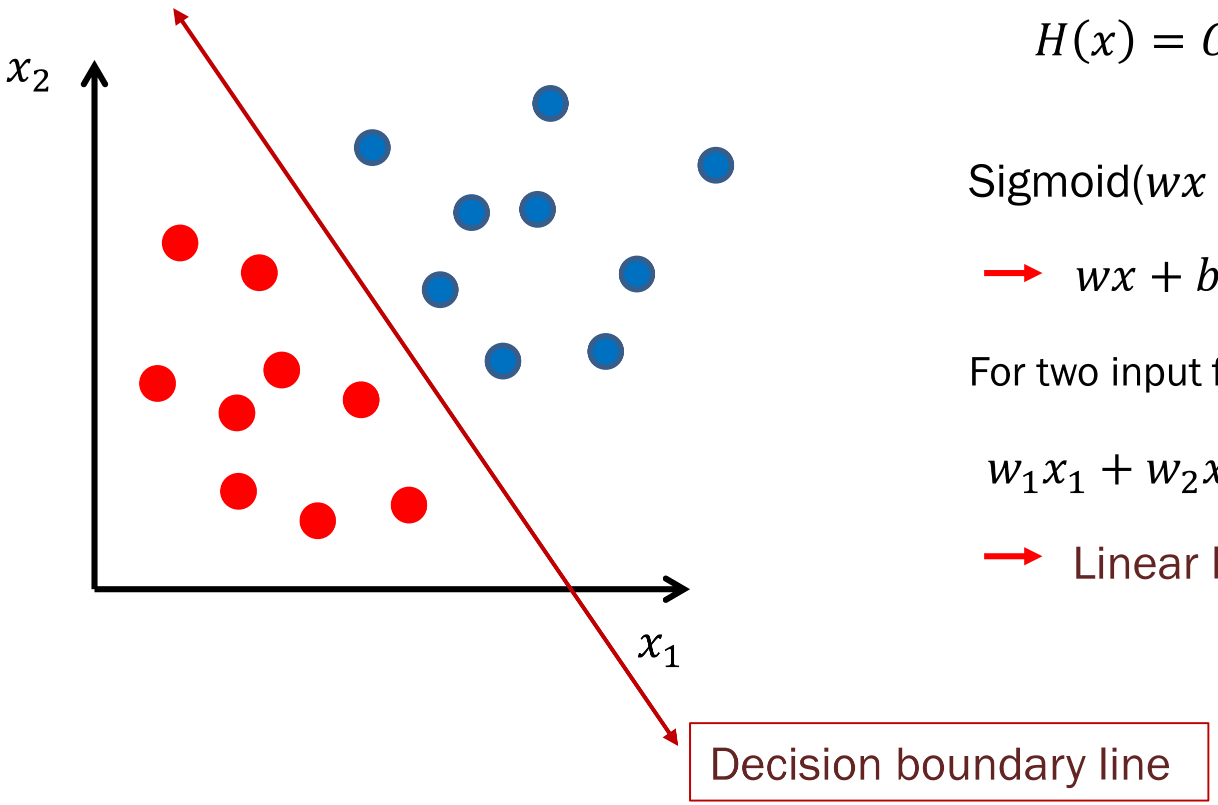


Example
Predicted: 0.01
Actual: 1

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

→ Large entropy
(Large error)

Decision boundary



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

Sigmoid$(wx + b)$ = 0.5

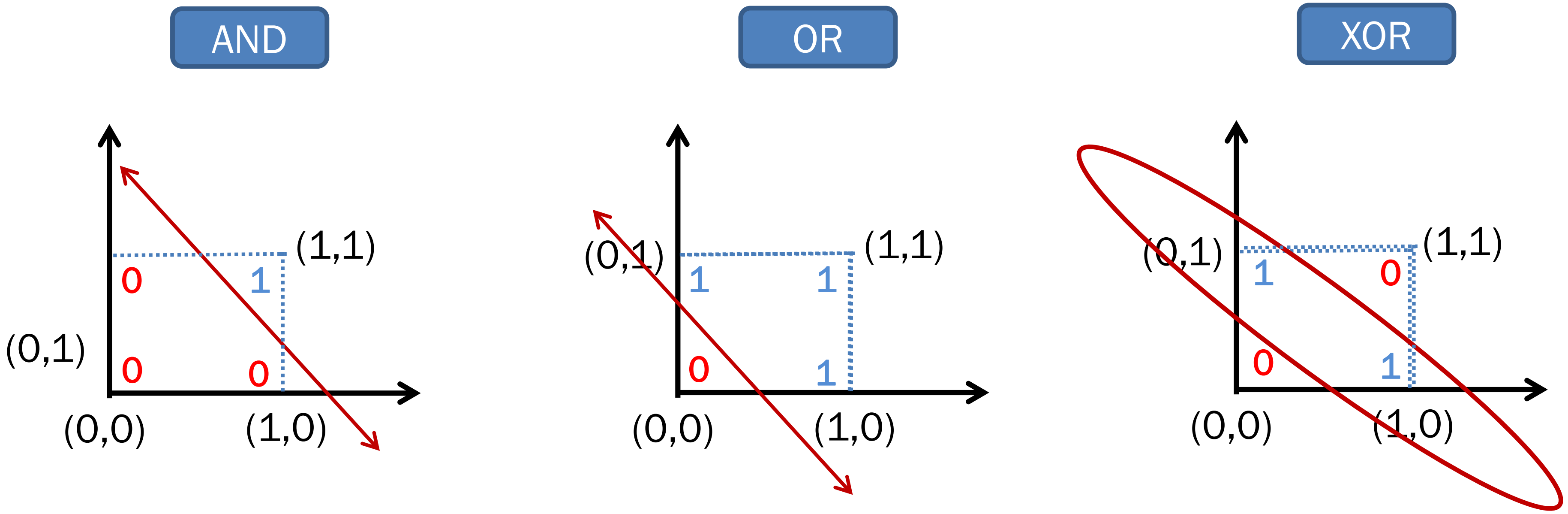
→ $wx + b = 0$

For two input feature problem, one can have

$$w_1x_1 + w_2x_2 + b = 0$$

→ Linear line!

XOR problem

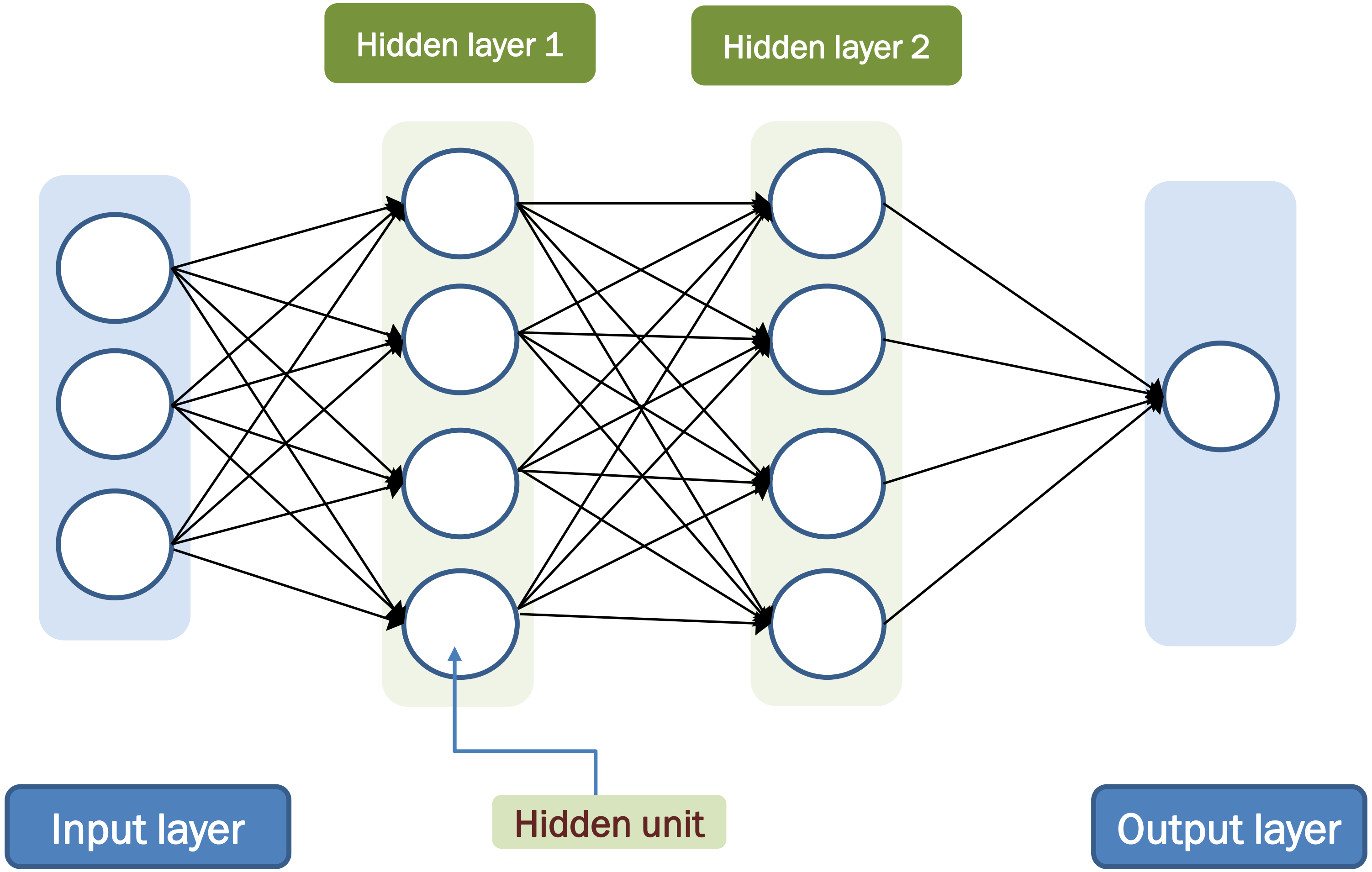


False: 0

True: 1

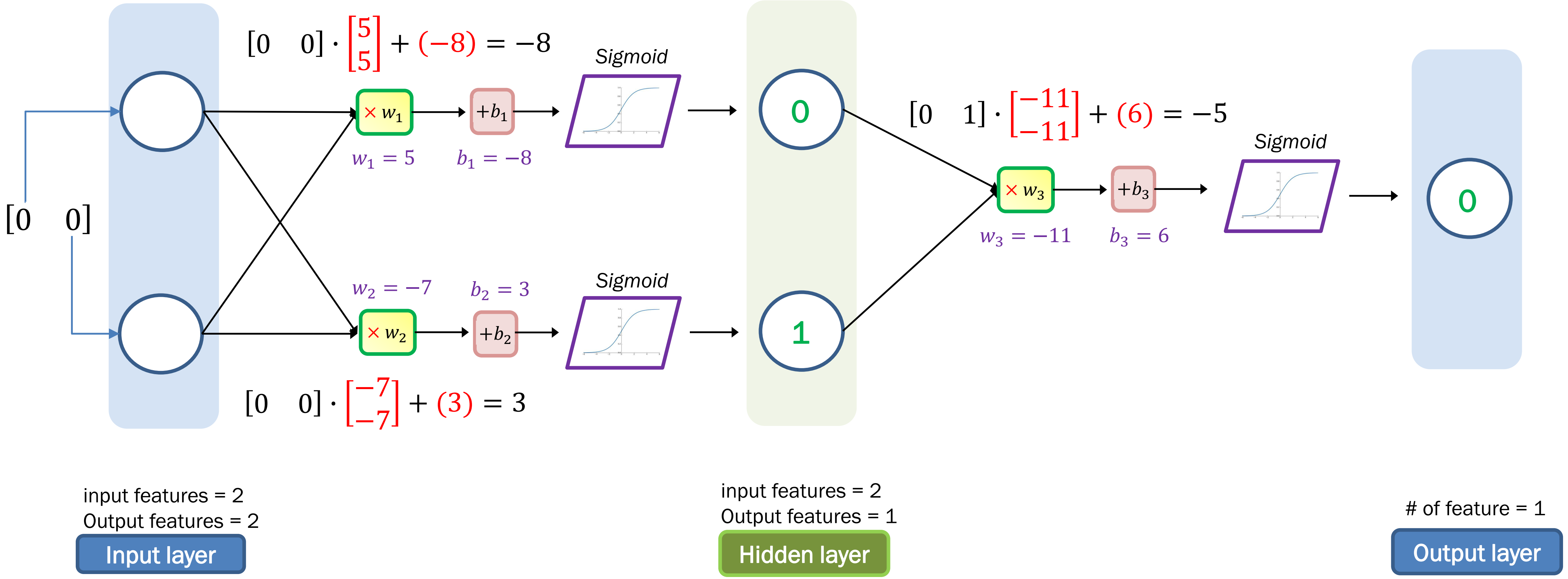
Fail to find a decision line !

Multi-Layer Perceptron



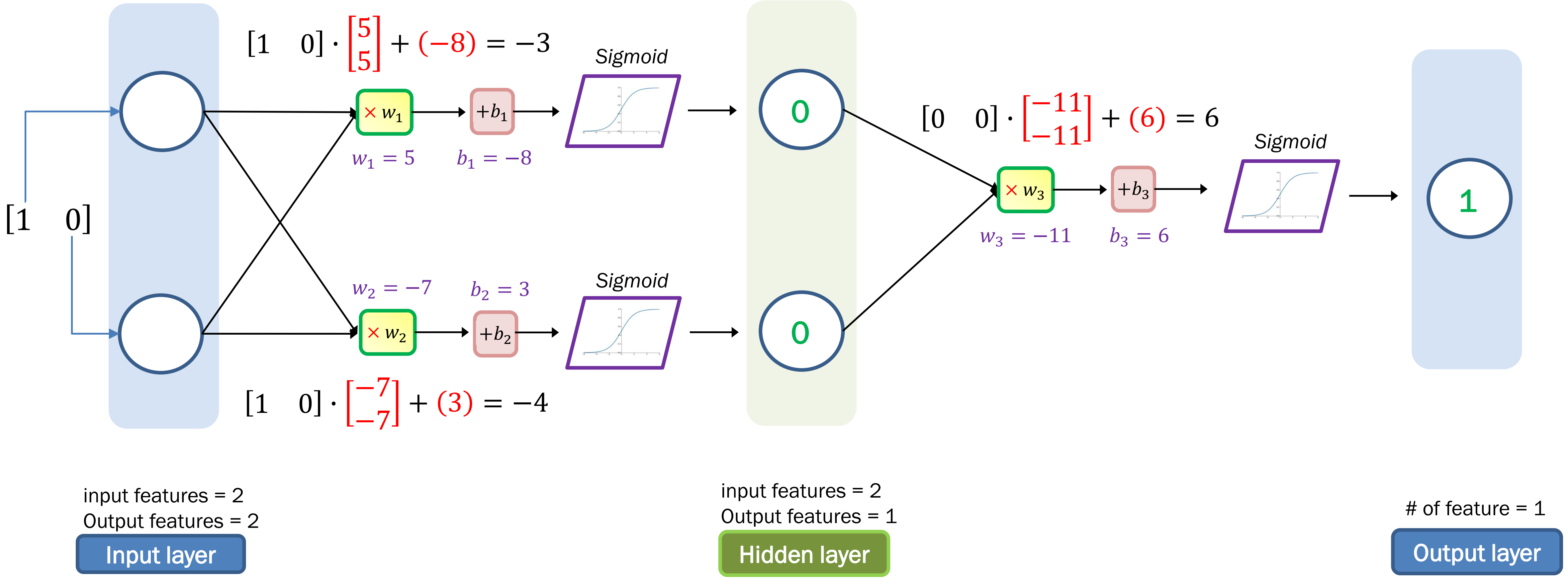
XOR problem

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



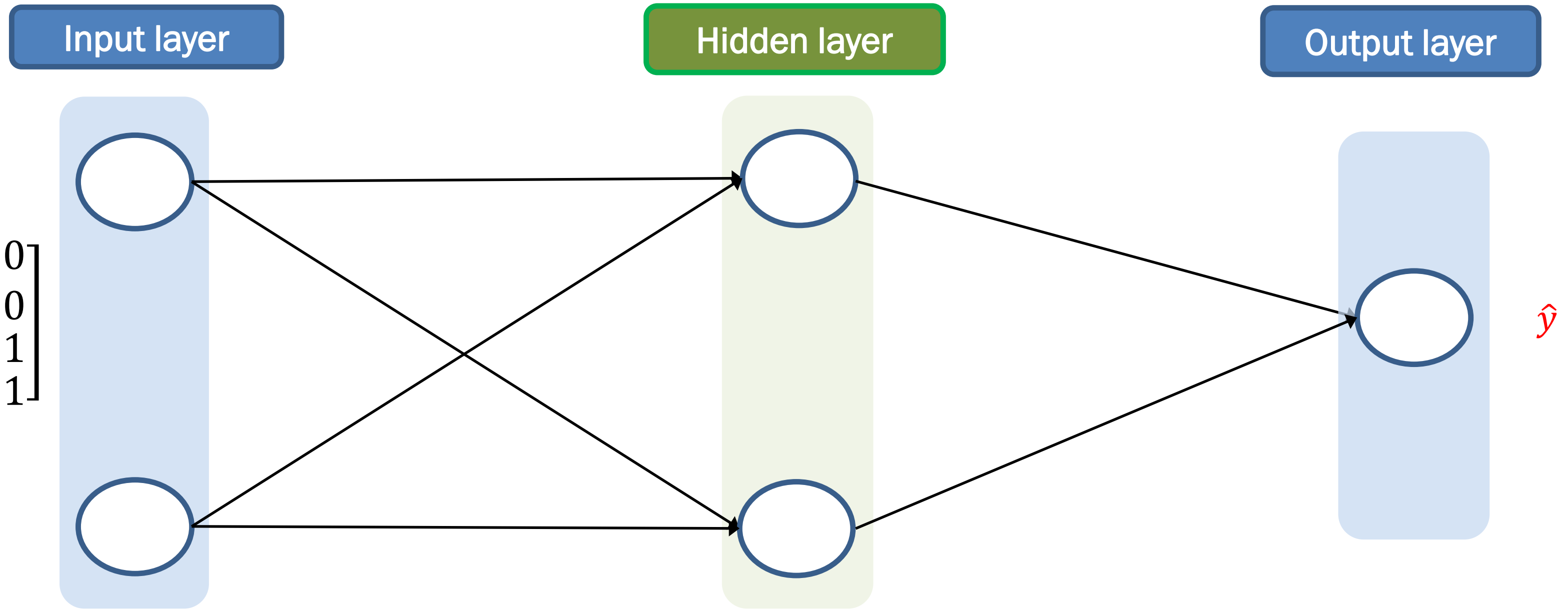
XOR problem

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

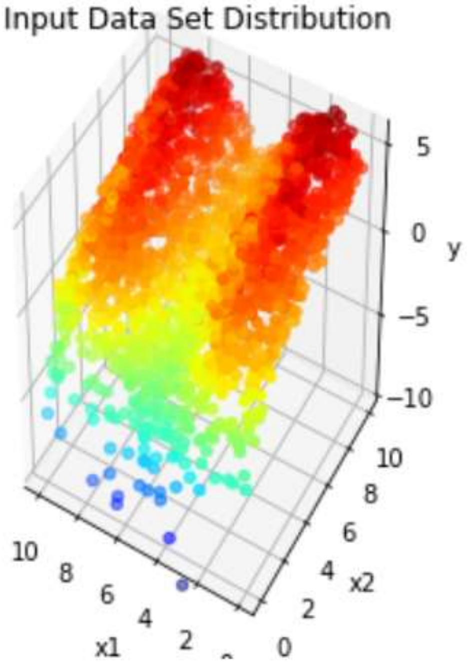
$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 5 & -7 \\ 5 & -7 \end{bmatrix} + \begin{bmatrix} -8 & 3 \\ -8 & 3 \\ -8 & 3 \\ -8 & 3 \end{bmatrix} = \begin{bmatrix} -8 & 3 \\ -3 & -4 \\ -3 & -4 \\ 2 & -11 \end{bmatrix} \xrightarrow{\text{Sigmoid}} \begin{bmatrix}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

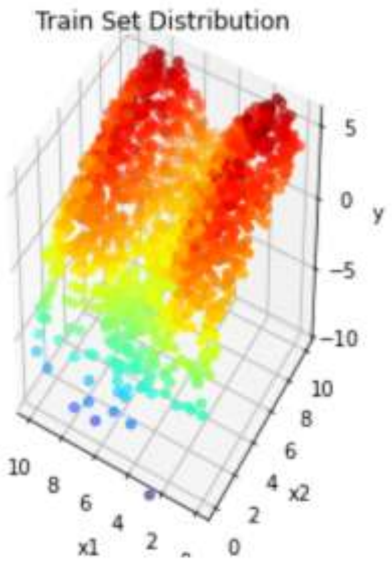
$$\begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} -11 \\ -11 \end{bmatrix} + \begin{bmatrix} 6 \\ 6 \\ 6 \\ 6 \end{bmatrix} = \begin{bmatrix} -5 \\ 6 \\ 6 \\ -5 \end{bmatrix} \xrightarrow{\text{Sigmoid}} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Data preparation

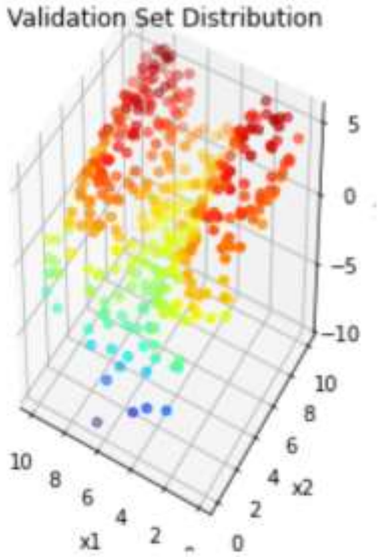
Data Preparation

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

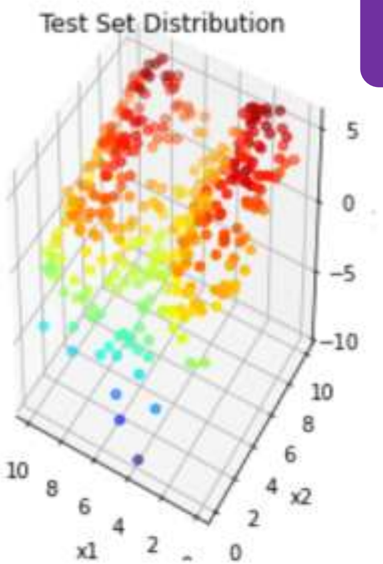




Train set



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

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

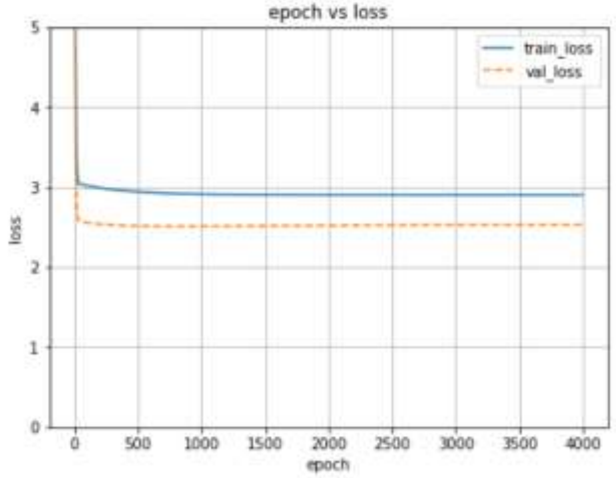
EPOCH=4000

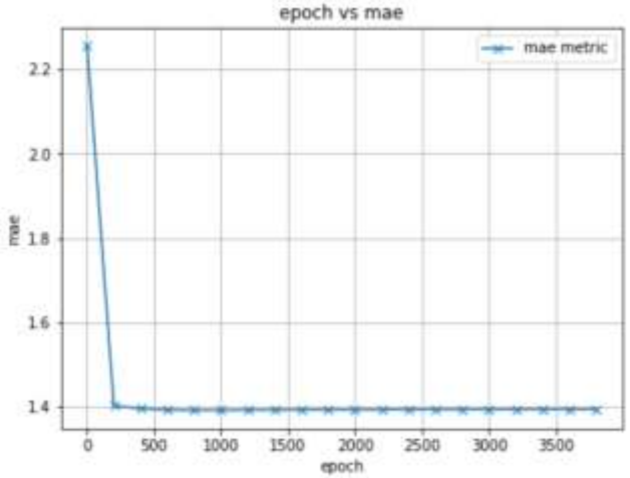
Train set

Validation set

Every 200 epoch

Test set





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 3A: Linear regression – MLP

Exercise 1: CPU/GPU	cores
Run 1	CPU
Run 2	GPU

Exercise 2: more layers w/ 100 units	# of layers
Run 1	2
Run 2	4
Run 3	8

Exercise 3: more units w/ 2 layers	# of units
Run 1	100
Run 2	200

Exercise 4: different activation	Activation function
Run 1	relu
Run 2	sigmoid

https://pytorch.org/docs/stable/nn.html
#non-linear-activations-weighted-sum-nonlinearity

You may want to try

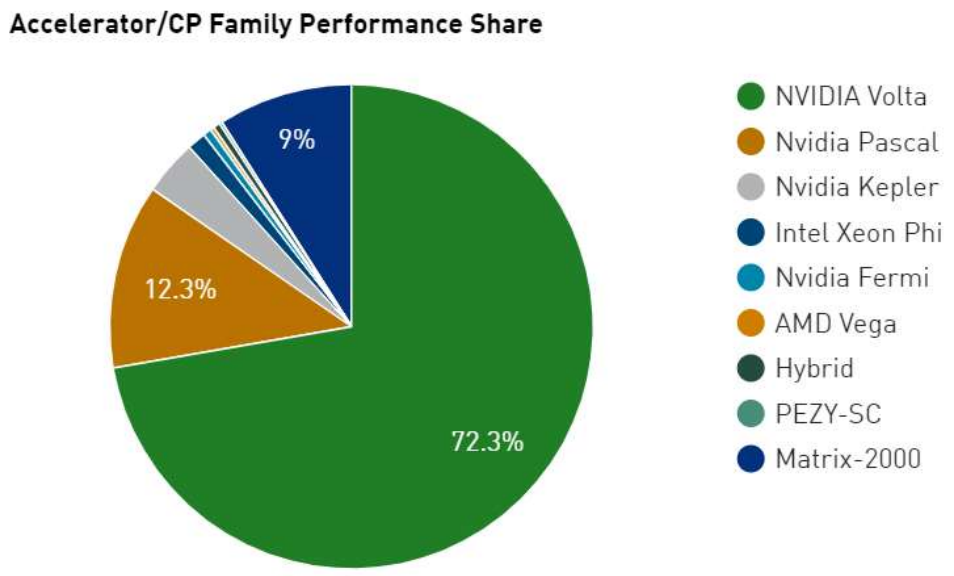
1. Increase size of data and re-run it
2. Use different optimizers
   https://pytorch.org/docs/stable/optim.html?highlight=optimizer
   #torch.optim.Optimizer

Break room

Running a DL code using GPU in Graham

DL and HPC architectures

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

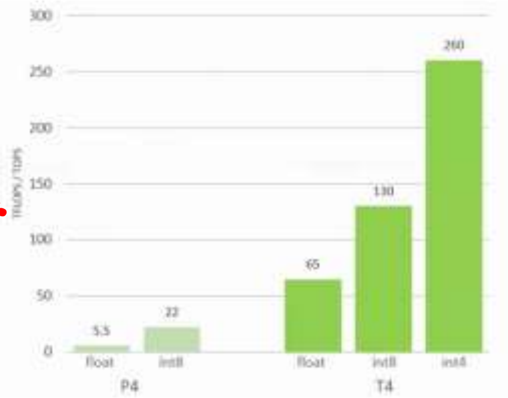


https://top500.org

NVIDIA T4 Turing GPU on Graham

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		

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

Running a DL code on `CPU` *interactively* in Graham

1.
```
cd /home/$USER/scratch/$USER/SS2020V2_ML_Day2/Session_3
```

2. Start interactive running mode

```
salloc --time=0:30:0 --ntasks=1 --cpus-per-task=3 --nodes=1 --mem=1000M --account=def-training-wa --reservation=snss20_wr_cpu
```

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

```
module load python
module load sci-py-stack
source ~/ENV/bin/activate
```

4. Run it

```
python SS20_lab3_LR_MLPg.py
```

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

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

Running a DL code *via scheduler* in Graham

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

```
#!/bin/bash
#
#SBATCH --nodes=1
#SBATCH --gres=gpu:t4:1
#SBATCH --mem=20000M
#SBATCH --time=0-30:00
#SBATCH --account=def-training-wa
#SBATCH --reservation=snss20_wr_gpu
#SBATCH --output=slurm.%x.%j.out

module load python scipy-stack
source ~/ENV/bin/activate
cd /home/$USER/scratch/$USER/SS2020V2_ML_Day2/Session_3
python /home/$USER/scratch/$USER/SS2020V2_ML_Day2/Session_3/SS20_lab3_LR_MLPg.py
```

2. Submit it

```
sbatch job_s.sh
```

3. Check the submitted job

```
squeue -u $USER
```

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

Session break:

Please come back by 3:30 PM