Group Presentation:

- Each group will do 10 minutes (~15 slides)
- Individual grades (70%) group grades (30%)
- Each person pick two sub-topics (you can also do more but all 4 should be covered by the group)
- Combine your ppt in one file and labeled with your names

Subtopics

Background

Data

Methods in Literature

Possible Future Works

Background:

Why do this? (Clinical relevance)

How have this been done (Traditional method)

What's the drawback? (inaccurate? costly?)

How can ML/DL potentially help?

Data:

Description of data (CT, MRI, 2D, 3D)

What's publicly available?

- number of subjects
- types of data
- types of annotation

How's the data looks like in private studies?

How difficult it is to gather new data?

How can you potential gather / create new data?

Methods in literature:

What has been done in this problem?

Pick ~5 paper, or few good review paper

What's the model?

What's their performance?

What's their short-coming?

Potential Future Works

What's the new model you want to use?

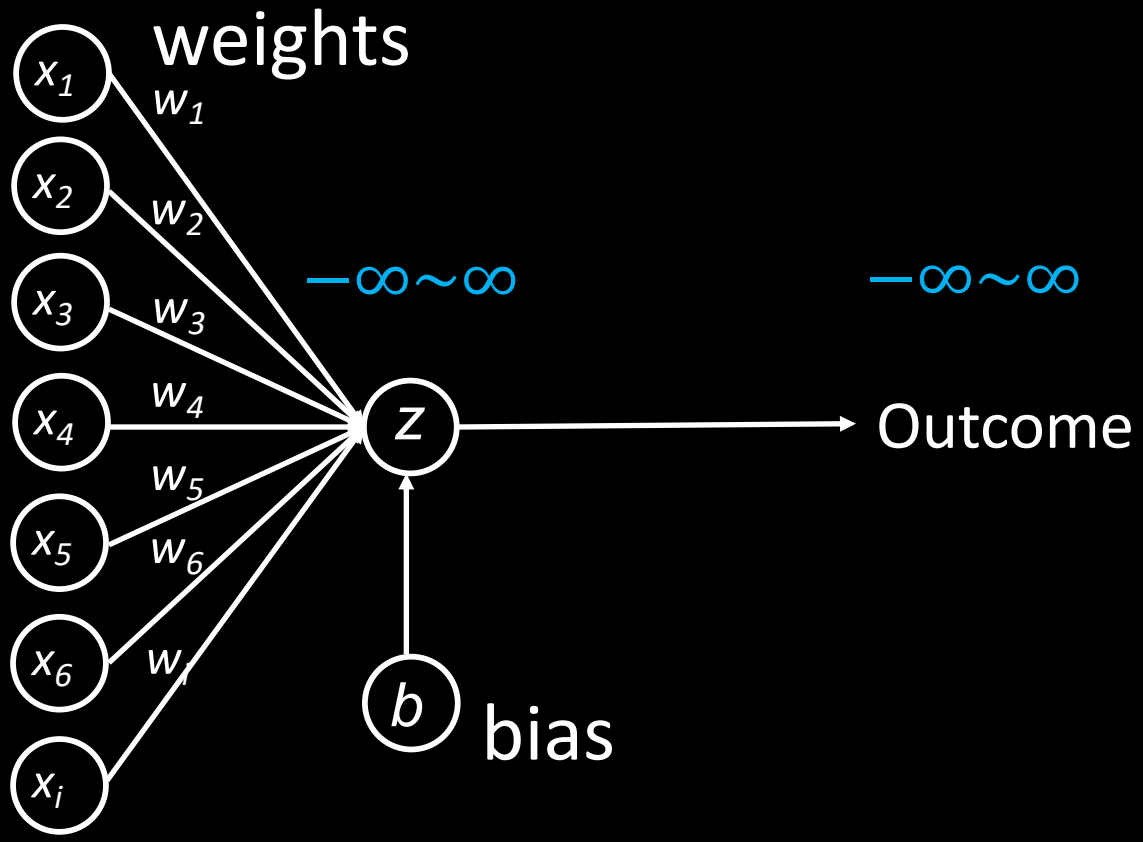
How is the new model relevant to your problem?

What's the potential benefit and how are you going to evaluate it?

Is data enough for the model?
What to do when data is not enough?

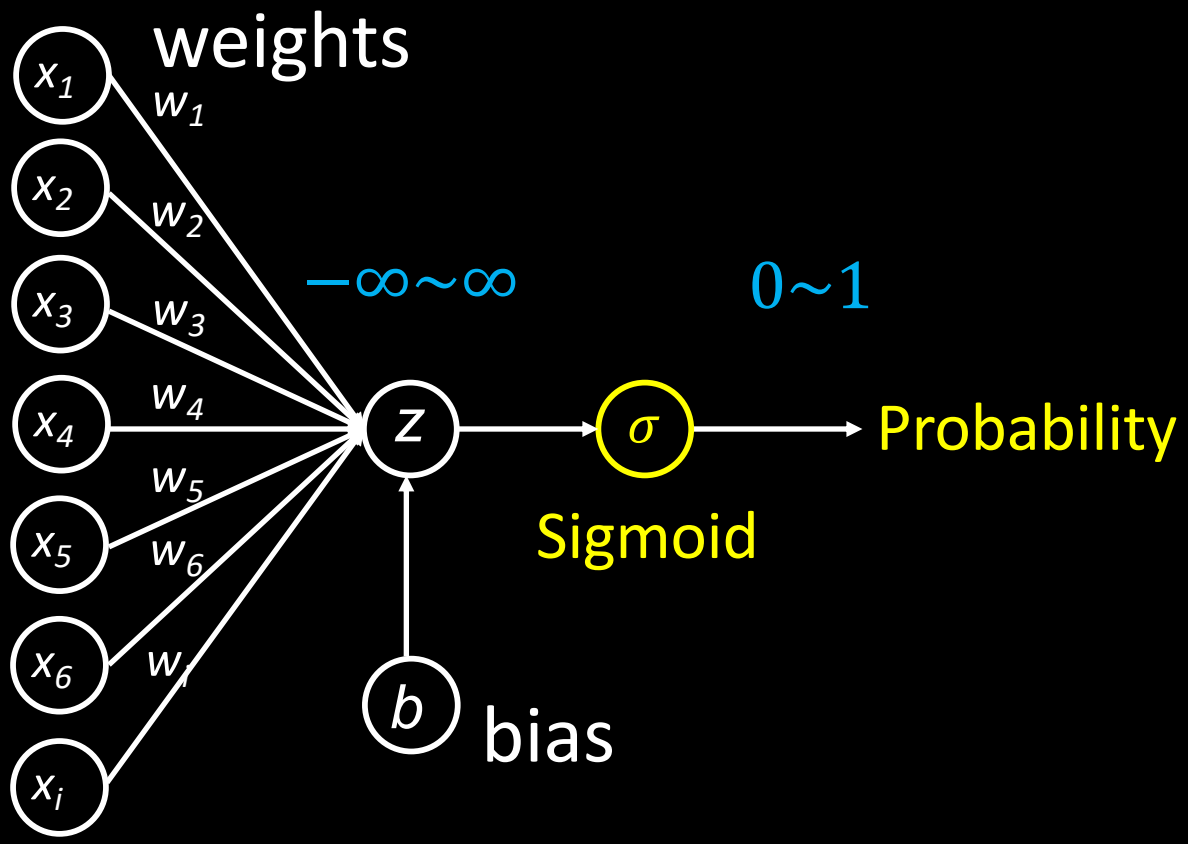
Recap: Regression vs Classification

Graph of Linear Regression

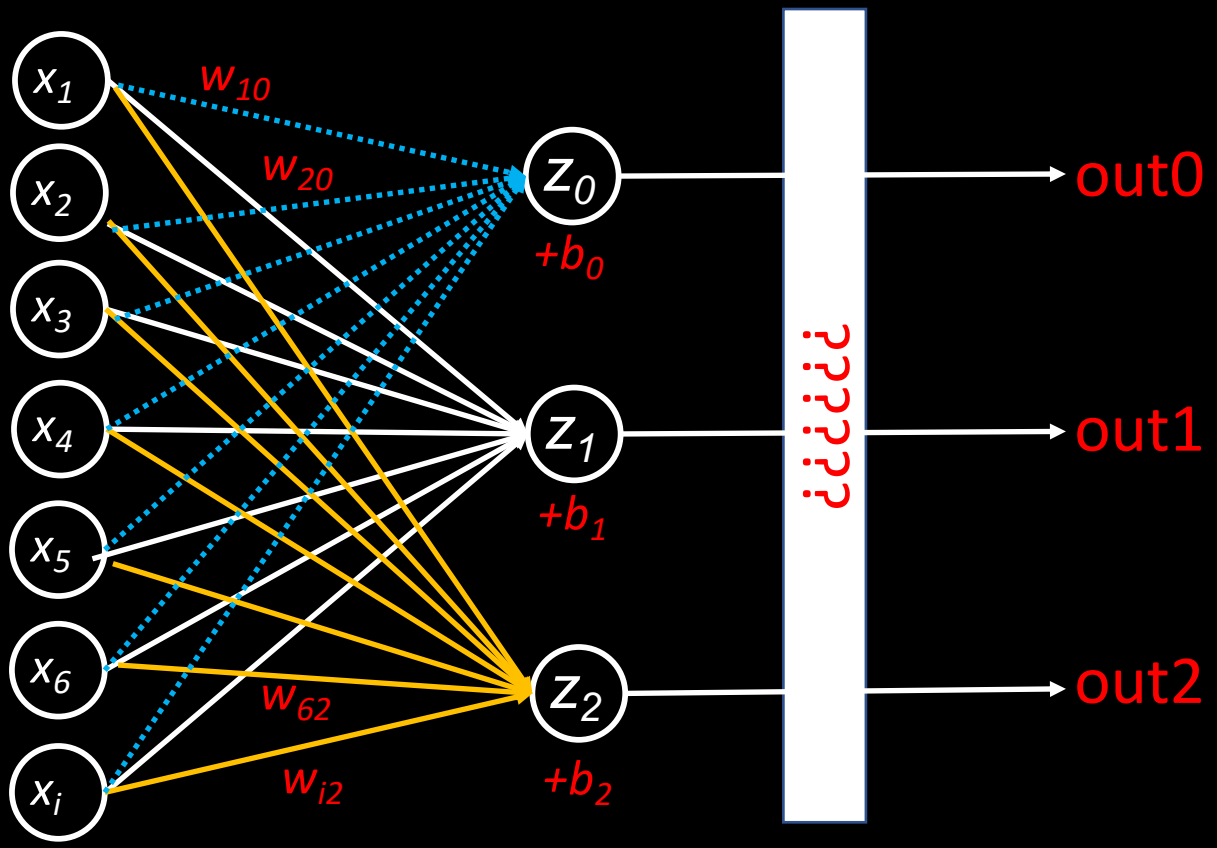


Graph of Logistic Regression

$$p = \frac{1}{1+exp^{-(\Sigma_i w_i x_i + b)}} = \text{Sigmoid}(\Sigma_i w_i x_i + b) = S(WX)$$

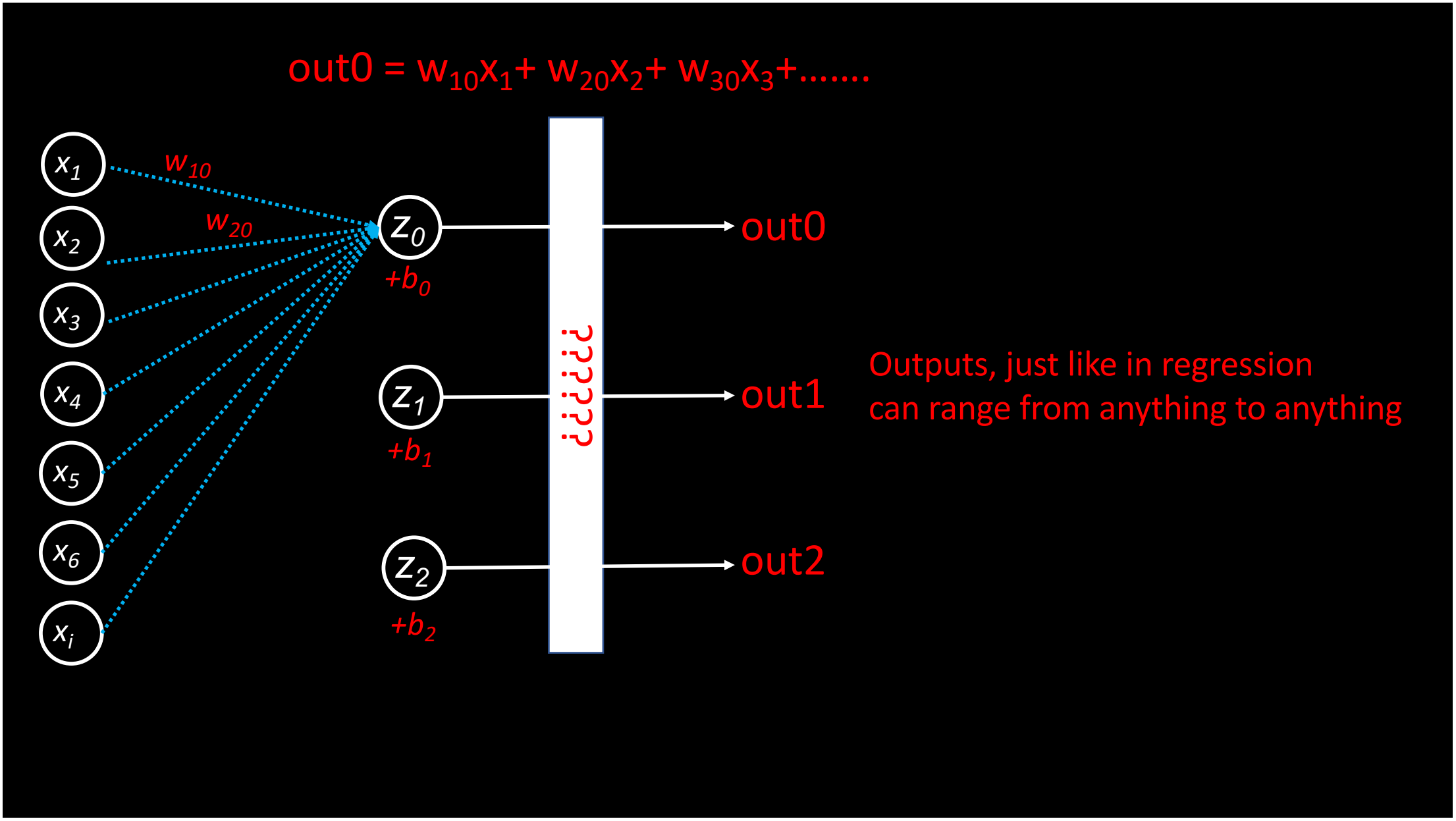


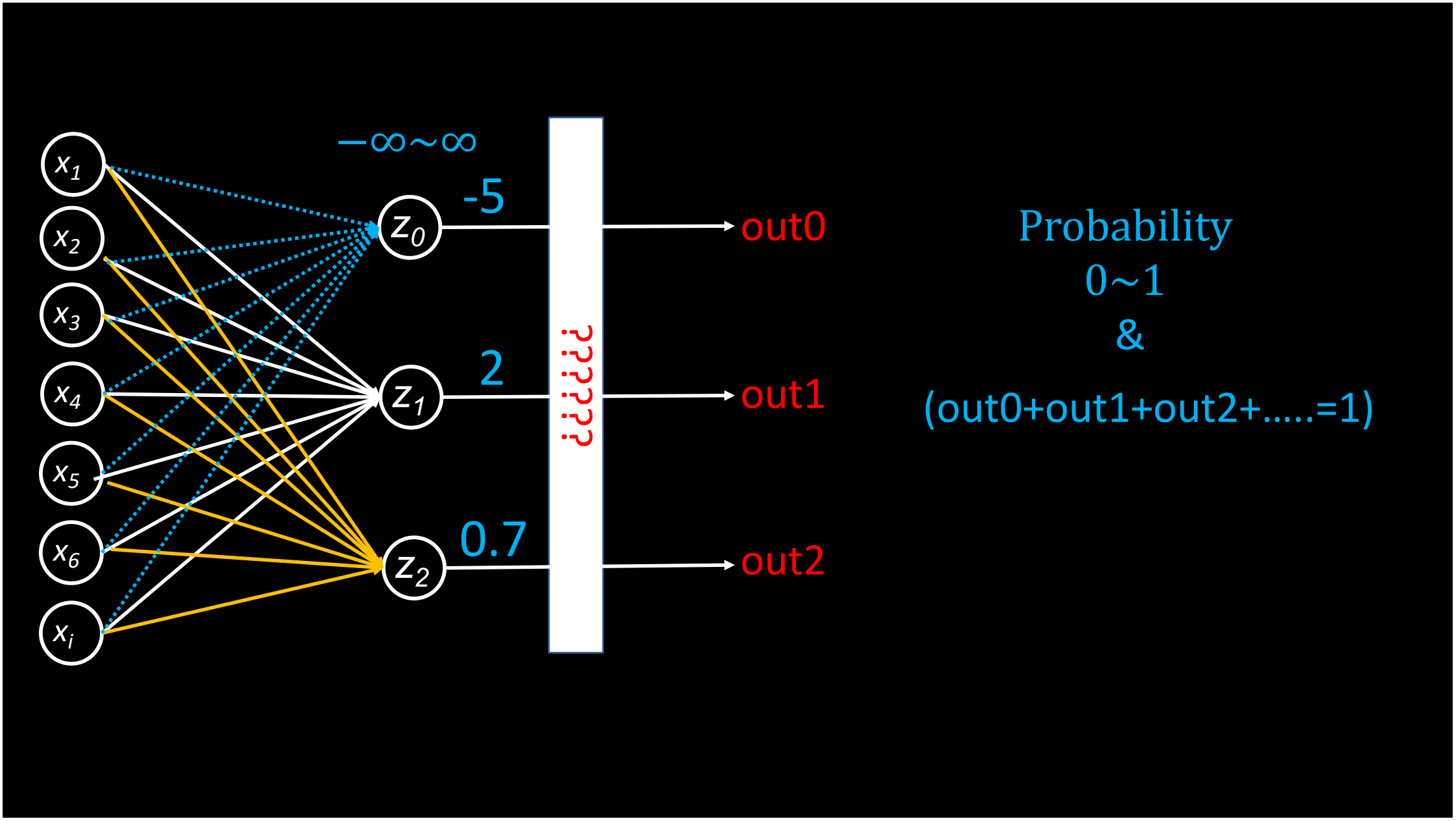
Multi-class Logistic Regression



$out_j =$

$w_{1j}x_1 + w_{2j}x_2 + w_{3j}x_3 + \ldots\ldots$

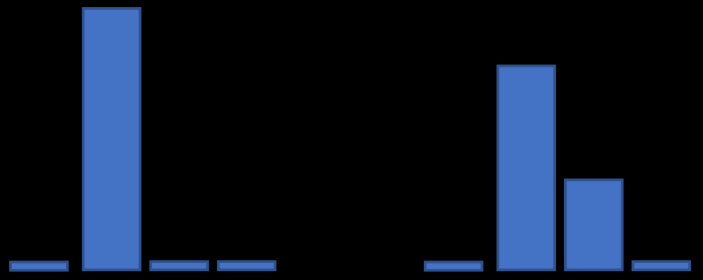

out0 = $w_{10}x_1 + w_{20}x_2 + w_{30}x_3 + \ldots\ldots$
x_1
x_2
x_3
x_4
x_5
x_6
x_i
w_{10}
w_{20}
z_0
$+b_0$
z_1
$+b_1$
z_2
$+b_2$
out0
out1
out2
Outputs, just like in regression
can range from anything to anything


x_1
x_2
x_3
x_4
x_5
x_6
x_i
$-\infty\sim\infty$
z_0
z_1
z_2
-5
2
0.7
??????
out0
out1
out2
Probability
0~1
&
(out0+out1+out2+.....=1)

Softmax ==
Sigmoid for multi-class

$$p(y = N) = Softmax(W_N X) = \frac{exp-(W_N X)}{\sum_n exp-(W_n X)}$$

Data	Output (Wx)	exp(Wx)	argmax	Softmax exp(Wx) / sum(exp(Wx)))
3	-0.84	0.43	0	0.02
4	2.69	14.73	1	0.68
5	1.76	5.81	0	0.27
6	-0.69	0.50	0	0.02



$$p = \frac{1}{1+exp-(\Sigma_i w_i x_i + b)} = \text{Sigmoid}(w_i x_i + b) = S(WX)$$

V S

$$p(y=0) = SX(W_0 X) = \frac{exp-(W_0 X)}{\sum_n exp-(W_n X)}$$

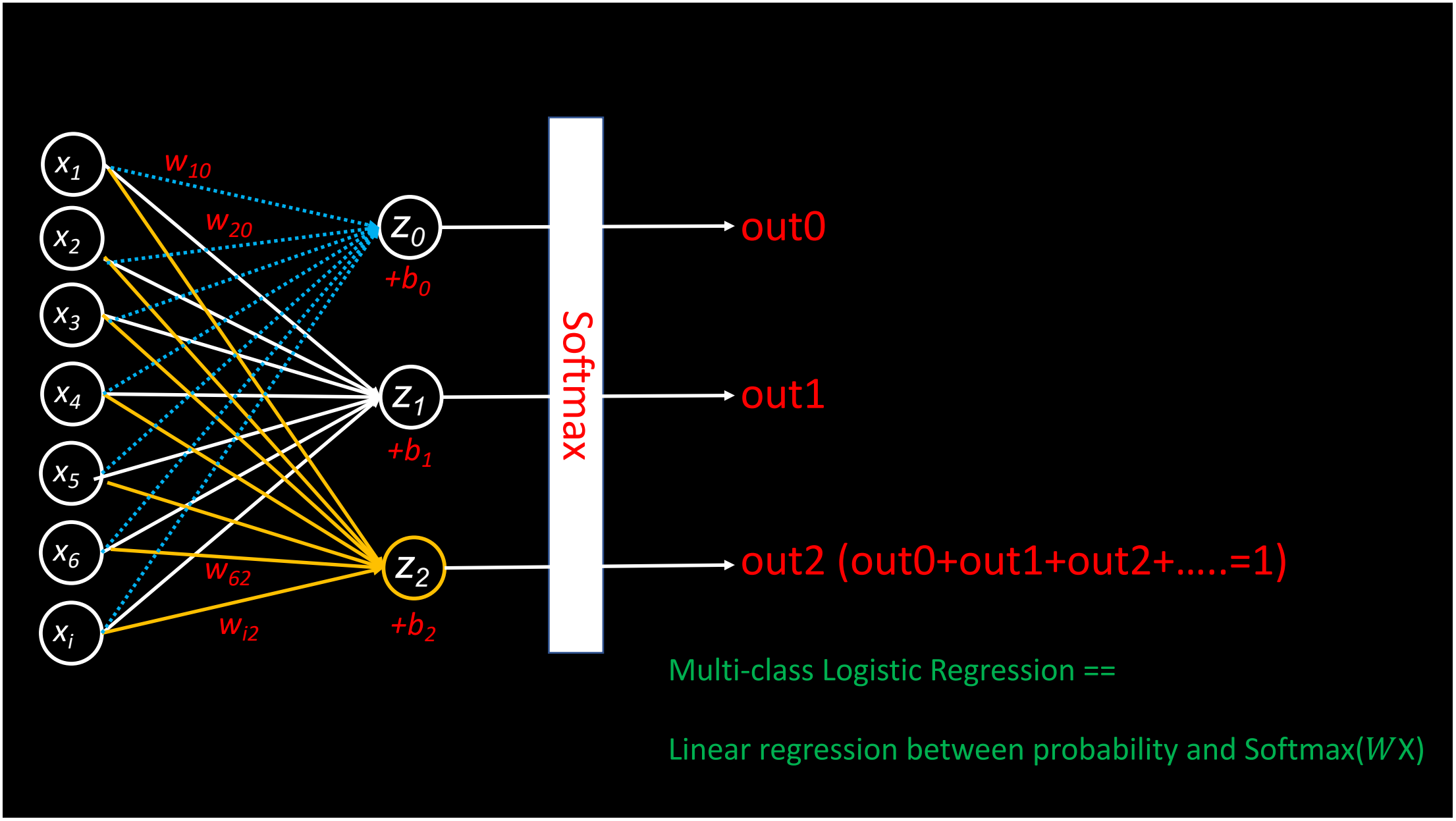
$$p(y=1) = SX(W_1 X) = \frac{exp-(W_1 X)}{\sum_n exp-(W_n X)}$$

•
•
•

$$p(y=N) = SX(W_N X) = \frac{exp-(W_N X)}{\sum_n exp-(W_n X)}$$

For multi-classes, you will have one *W* for each class

The summation for all the P Should be 1


x_1
x_2
x_3
x_4
x_5
x_6
x_i
w_{10}
w_{20}
w_{62}
w_{i2}
z_0
z_1
z_2
$+b_0$
$+b_1$
$+b_2$
Softmax
out0
out1
out2 (out0+out1+out2+.....=1)
Multi-class Logistic Regression ==
Linear regression between probability and Softmax(WX)

Binary Cross Entropy

$$-\sum_n [\mathrm{y} \cdot \log(p(\hat{y})) + (1-y) \cdot \log(p(1-\hat{y}))])$$

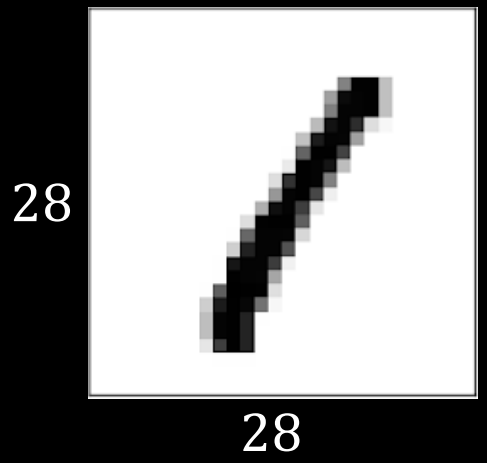
Multi-Class Cross Entropy

$$-\sum_n \mathrm{y} \cdot \log(p(\hat{y}))$$

y	$\hat{y}$		
0	0.6		
1	0.3	=	-1 * log(0.3)
0	0.1		

Image Classification by Logistic Regression

MNIST dataset



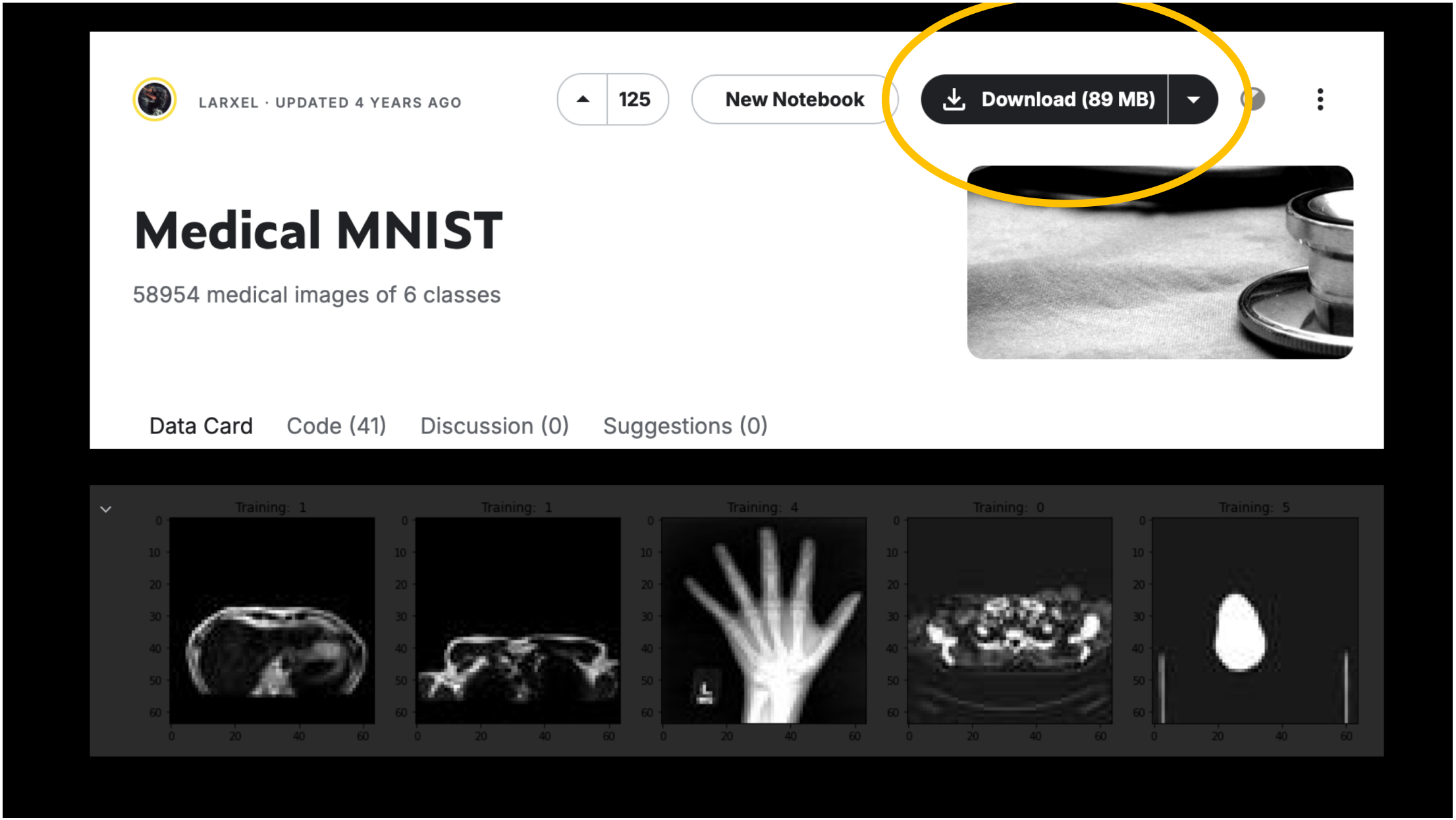
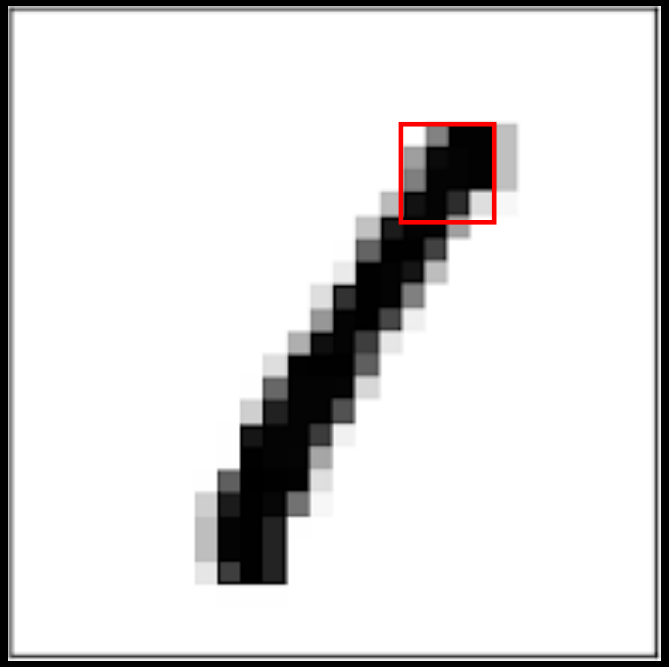
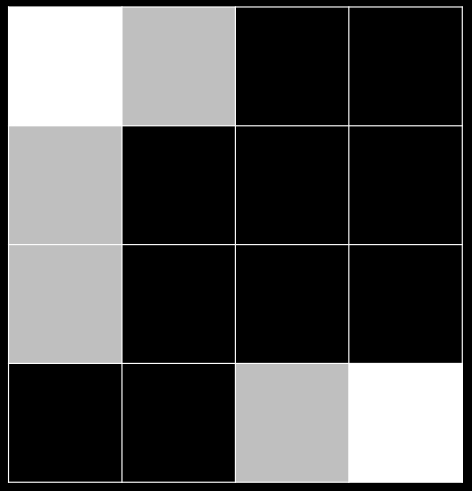

LARXEL · UPDATED 4 YEARS AGO
125
New Notebook
Download (89 MB)
Medical MNIST
58954 medical images of 6 classes
Data Card
Code (41)
Discussion (0)
Suggestions (0)
Training: 1
Training: 1
Training: 4
Training: 0
Training: 5

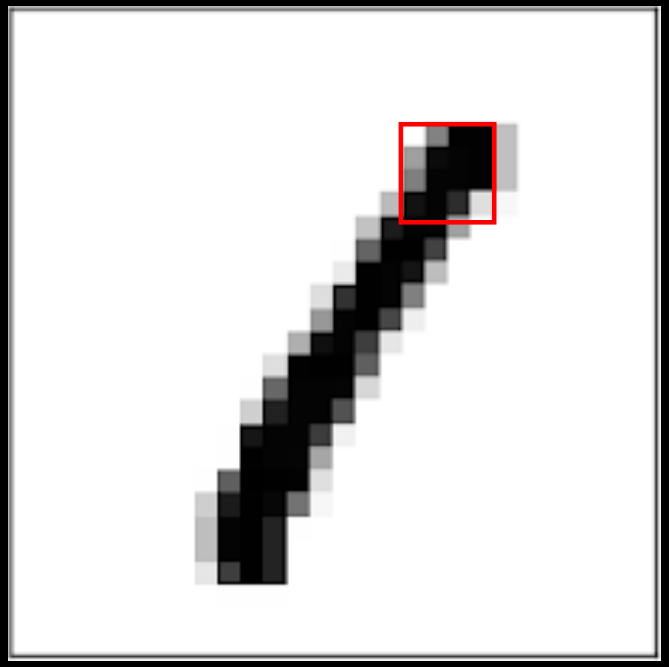
Image to grayscale pixel value





1	0.5	0	0
0.5	0	0	0
0.5	0	0	0
0	0	0.5	1

Image to grayscale pixel value



1	0.5	0	0
0.5	0	0	0
0.5	0	0	0
0	0	0.5	1

1
0.5
0
0
0.5
0
0
0
0.

numpy.ndarray.flatten

torch.array.view(-1)

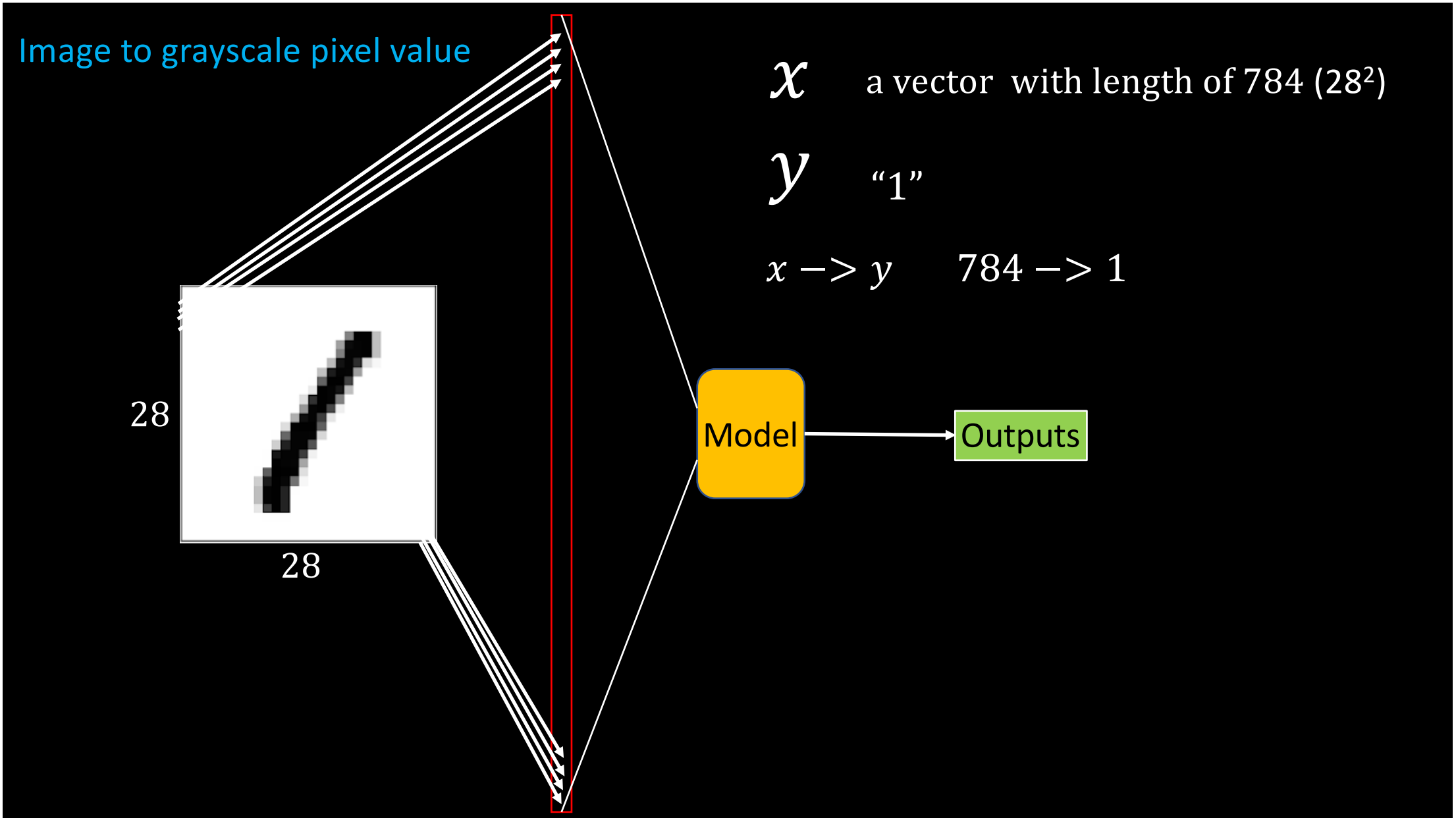


Image to grayscale pixel value
28
28
x a vector with length of 784 (28^2)
y "1"
$x -> y$ $784 -> 1$
Model
Outputs

```
First, download medical_mnist.zip from Kaggle
https://www.kaggle.com/datasets/andrewmvd/medical-mnist
or from dropbox
https://www.dropbox.com/scl/fi/puo3gd1wmnat5ggz2naih/mmnist
 .zip?rlkey=uhiyemgjg6xl67qt36fdkyuny&e=2&dl=0
unzip the data to Exercise2 / mmnist/
```

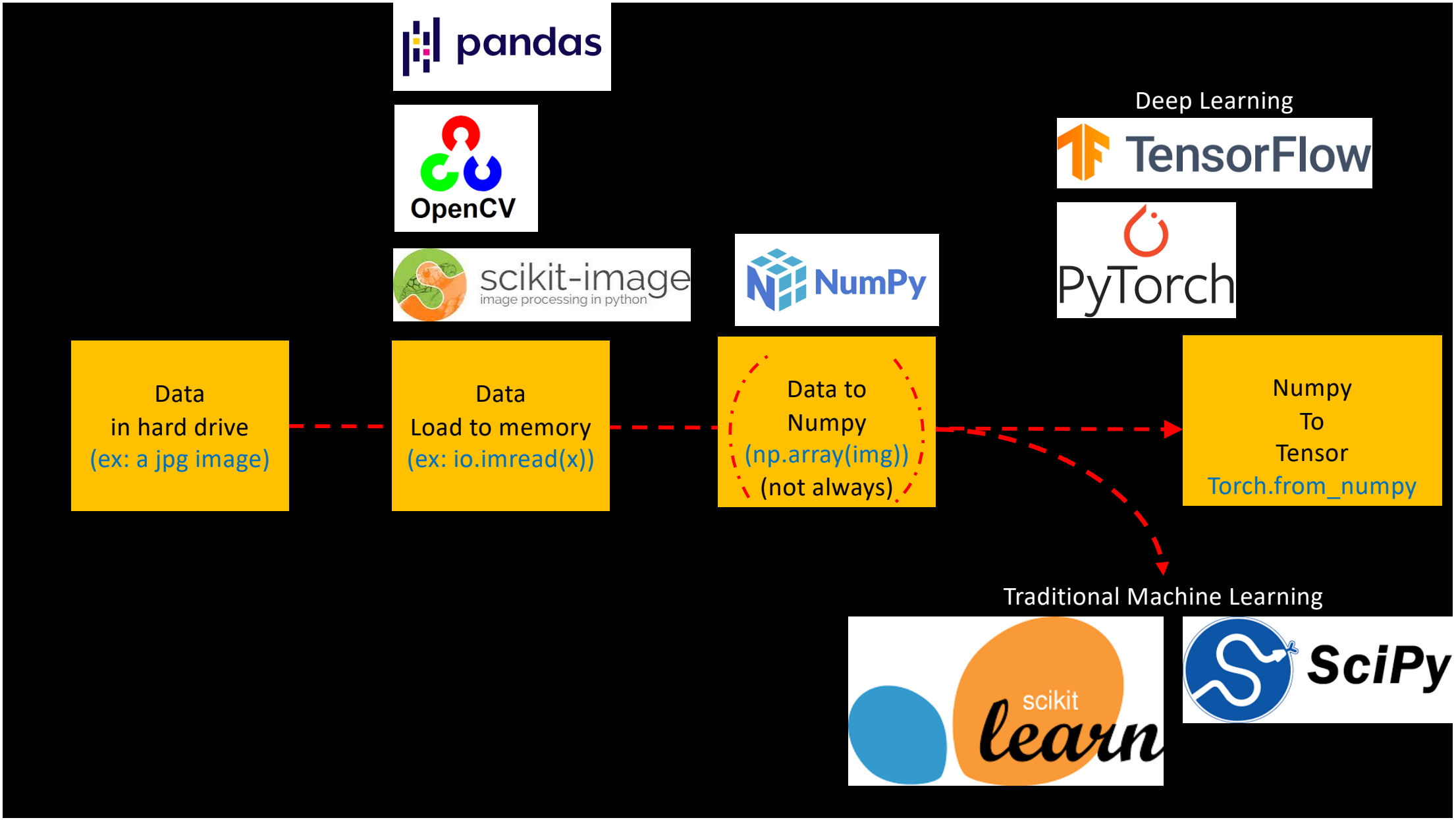
LARXEL · UPDATED 4 YEARS AGO

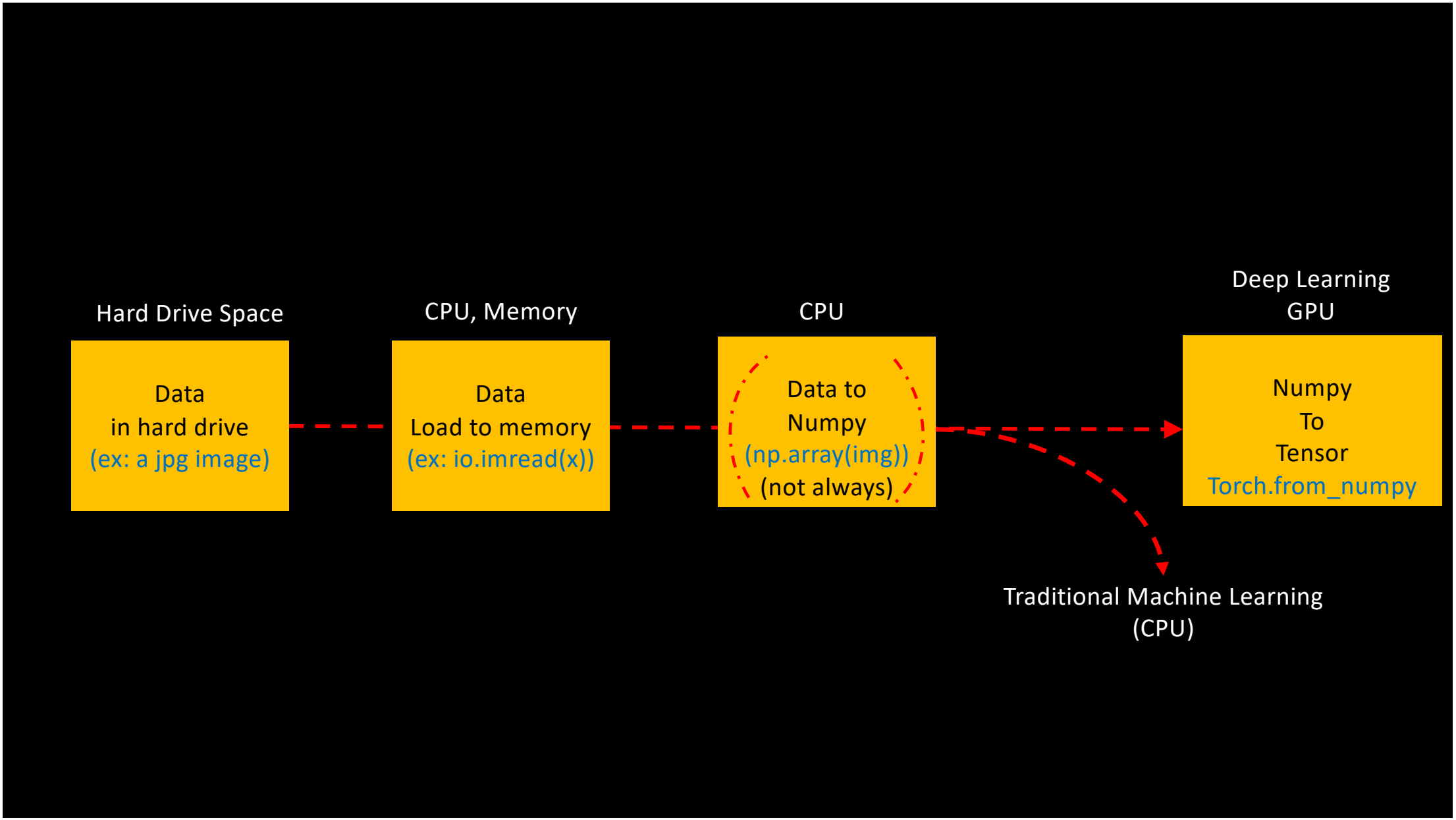
125 | New Notebook | Download (89 MB)

Medical MNIST

58954 medical images of 6 classes

Data Card | Code (41) | Discussion (0) | Suggestions (0)


pandas
OpenCV
scikit-image
image processing in python
NumPy
Deep Learning
TensorFlow
PyTorch
Data
in hard drive
(ex: a jpg image)
Data
Load to memory
(ex: io.imread(x))
Data to
Numpy
(np.array(img))
(not always)
Numpy
To
Tensor
Torch.from_numpy
Traditional Machine Learning
scikit
learn
SciPy

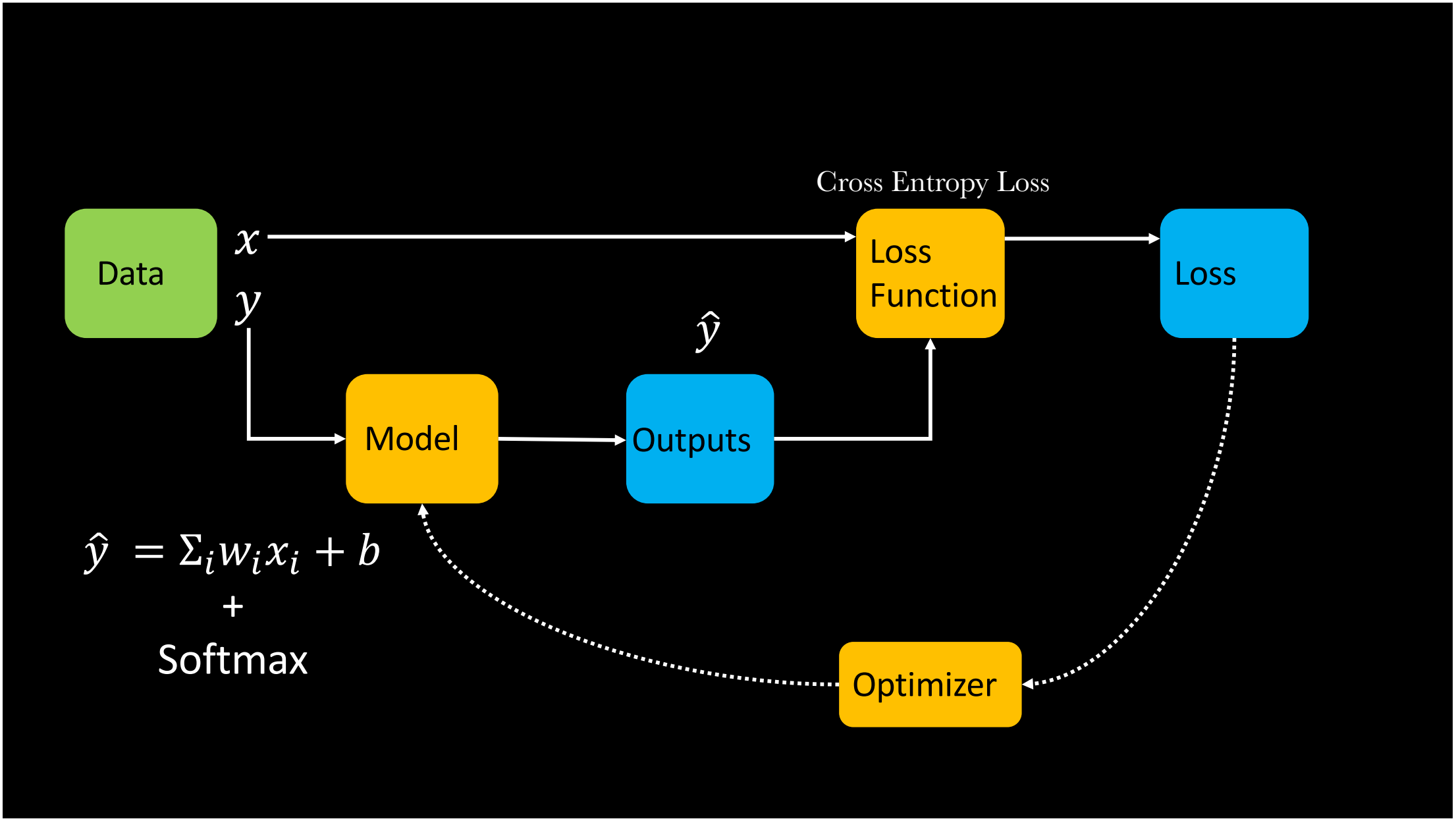

Hard Drive Space
Data
in hard drive
(ex: a jpg image)
CPU, Memory
Data
Load to memory
(ex: io.imread(x))
CPU
Data to
Numpy
(np.array(img))
(not always)
Deep Learning
GPU
Numpy
To
Tensor
Torch.from_numpy
Traditional Machine Learning
(CPU)

Pytorch Installation Pytorch.org

START LOCALLY

Select your preferences and run the install command. Stable represents the most currently tested and supported version of PyTorch. This should be suitable for many users. Preview is available if you want the latest, not fully tested and supported, 1.12 builds that are generated nightly. Please ensure that you have **met the prerequisites below (e.g., numpy)**, depending on your package manager. Anaconda is our recommended package manager since it installs all dependencies. You can also install previous versions of PyTorch. Note that LibTorch is only available for C++.

PyTorch Build	Stable (1.12.1)	Preview (Nightly)	LTS (1.8.2)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python	C++ / Java		
Compute Platform	~~CUDA 10.2~~	~~CUDA 11.3~~	~~CUDA 11.6~~	~~ROCm 5.1.1~~	Default
Run this Command:	`conda install pytorch torchvision torchaudio -c pytorch`			


Cross Entropy Loss
Data
x
y
Loss Function
Loss
$\hat{y}$
Model
Outputs
$\hat{y} = \Sigma_i w_i x_i + b$
+
Softmax
Optimizer

Data

```
# Medical MNIST dataset (images and labels)
train_loader, validation_loader = get_medical_mnist(args=args)
print('Done with data preparation')
```

```
data folder:  mmnist/
length of all images: 42000
length of all images: 16954
Done with data preparation
```

Model

```
# get model
if args['model'] == 'logistic_regression':
    print('Using logistic regression')
    model = nn.Linear(args['img_size'], args['num_classes'])
elif args['model'] == 'MLP':
    print('MLP')
    model = MLP(dropout=0, hidden_1=512, hidden_2=512)
```

Loss

Optimizer

```
# Loss and optimizer
loss_function = nn.CrossEntropyLoss() # this combined the Log
optimizer = torch.optim.SGD(model.parameters(), lr=args['lear
```

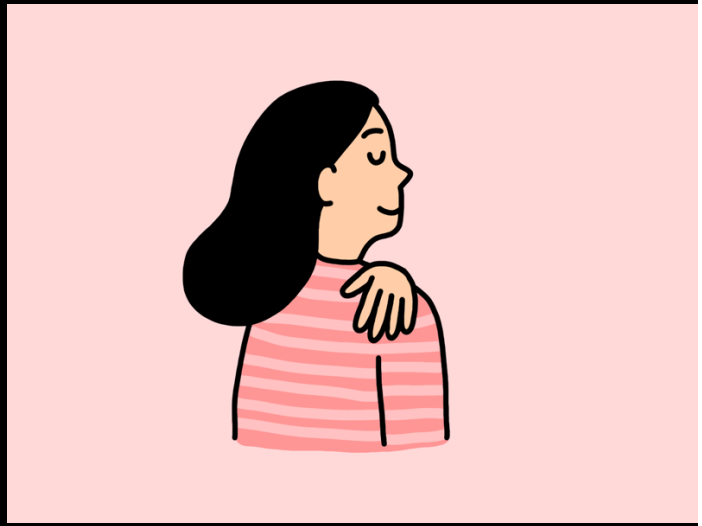
Hyper-parameters

```python
# arguments
def get_arguments():
    # Hyper-parameters
    args = {'img_size': 64 * 64,
            'num_classes': 10,
            'num_epochs': 50,
            'batch_size': 16,
            'learning_rate': 0.001,
            'model': 'logistic_regression'} # MLP or logistic_regression
    return args

args = get_arguments()
print(args)
```

Trainer

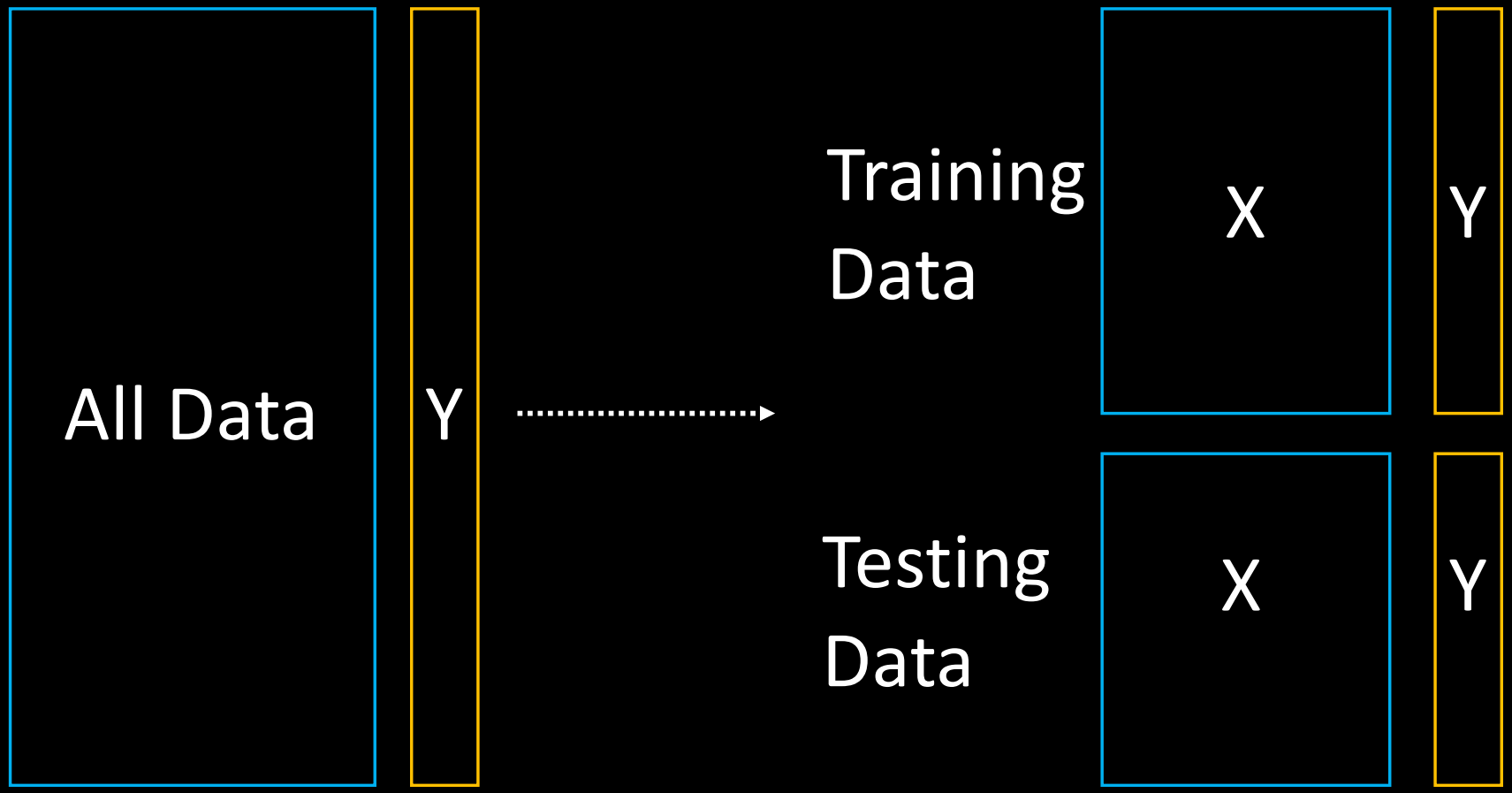
```
class Trainer():
    def __init__(self, args, train_loader, validation_loader, model, loss_function, optimizer):
        pass

    def overall_loop(self):
        for epoch in range(args['num_epochs']):
            train_loss = self.training_loop(train_loader)  # do the training look
            validation_loss = self.validation_loop(validation_loader) # do the validation loop
            # print out the training and validation loss per epoch
            print('Epoch [{}/{}], Train Loss: {:.4f}, Validation Loss: {:.4f}'
                  .format(epoch + 1, args['num_epochs'],
                          sum(train_loss) / len(train_loss), sum(validation_loss) / len(validation_loss)))
```

```
$ python mnist_logistic.py
Epoch [42/50], Step [100/600], Train Loss: 0.4635, Test Loss: 0.4393
Epoch [43/50], Step [100/600], Train Loss: 0.4607, Test Loss: 0.4365
Epoch [44/50], Step [100/600], Train Loss: 0.4579, Test Loss: 0.4339
Epoch [45/50], Step [100/600], Train Loss: 0.4552, Test Loss: 0.4313
Epoch [46/50], Step [100/600], Train Loss: 0.4527, Test Loss: 0.4289
Epoch [47/50], Step [100/600], Train Loss: 0.4502, Test Loss: 0.4265
Epoch [48/50], Step [100/600], Train Loss: 0.4478, Test Loss: 0.4243
Epoch [49/50], Step [100/600], Train Loss: 0.4455, Test Loss: 0.4221
Epoch [50/50], Step [100/600], Train Loss: 0.4433, Test Loss: 0.4200
Accuracy of the model on the 10000 test images: 89.110 %
```



Details of Model Training

Train-Testing Splitting



Scenario A

Training in :

Using (testing) in :

Scenario B

Training in :

Using (testing) in :

Why can’t the data be the same?

Training Data

Testing Data

Source 1

Source 2

Training Data

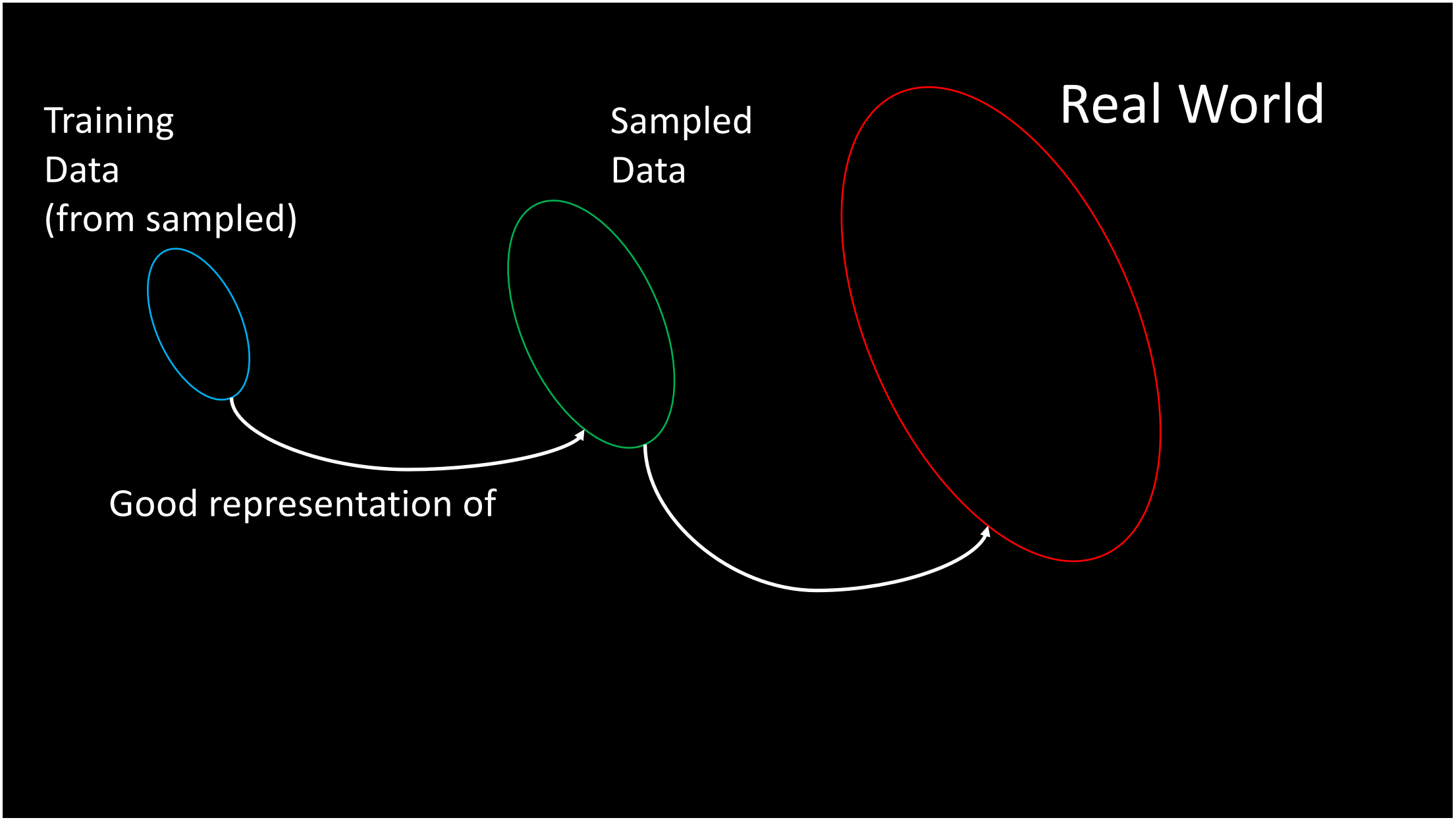
Validation

Testing

Source 1

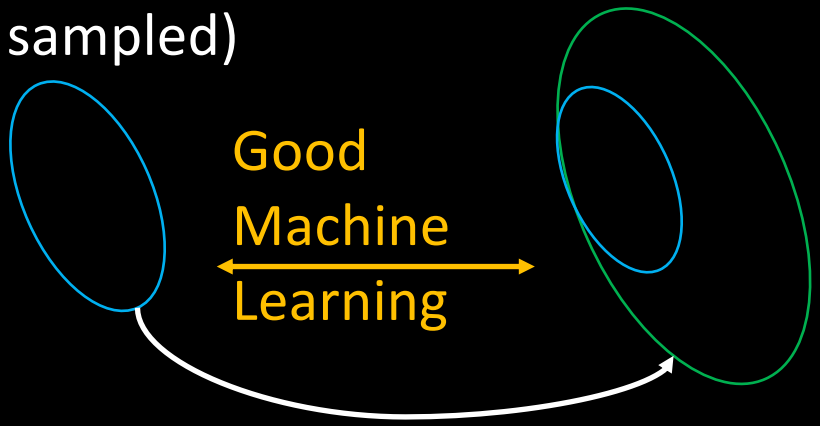
Source 2

“Internal Testing”


Training
Data
(from sampled)
Sampled
Data
Real World
Good representation of

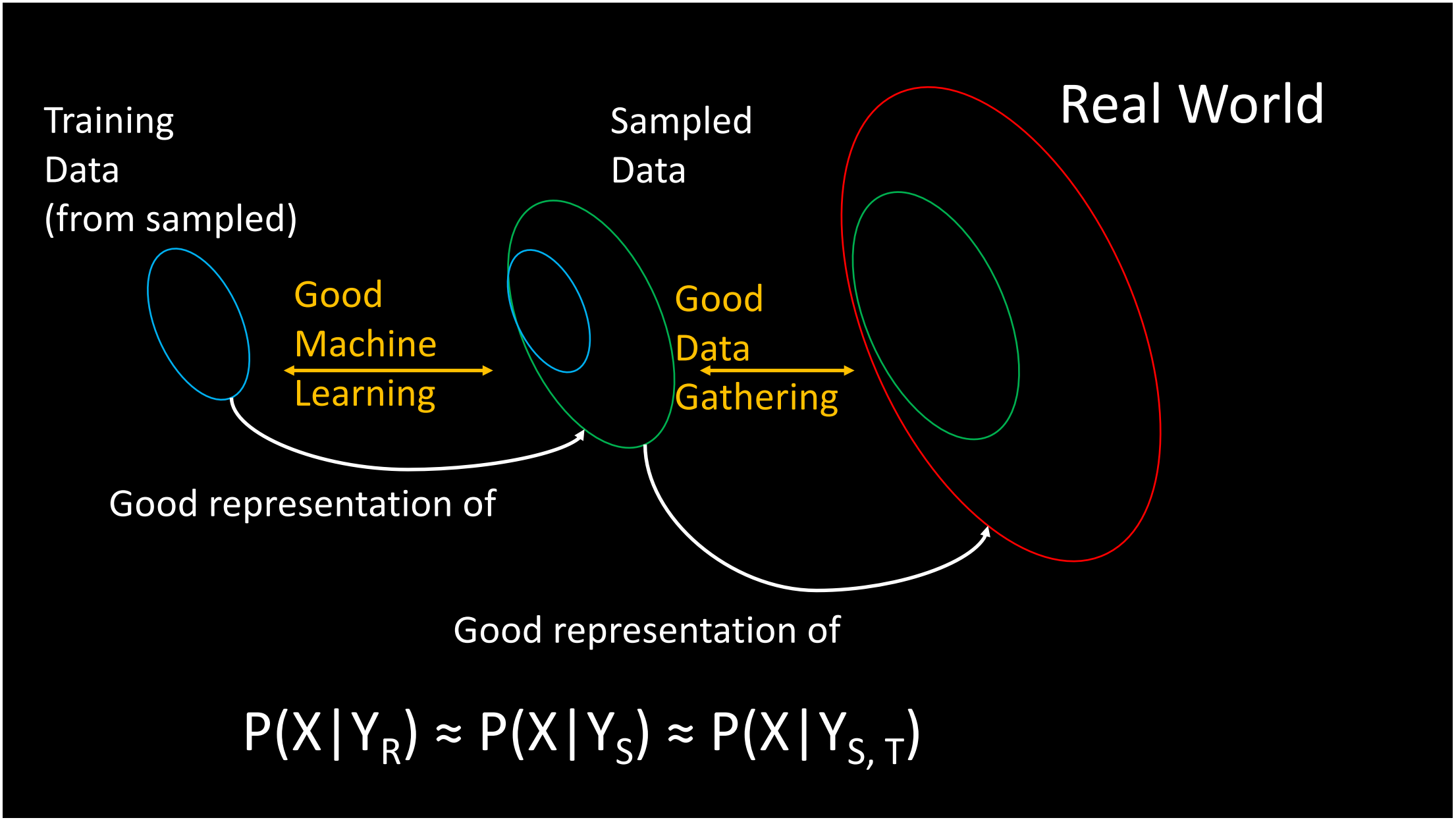
Training
Data
(from sampled)

Sampled
Data

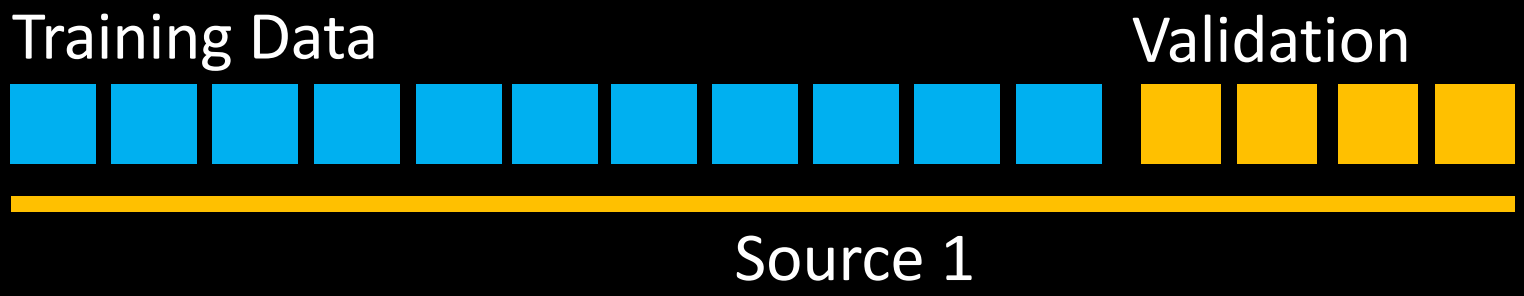


Good representation of

$$P(X|Y_R) \approx P(X|Y_S)$$


Training
Data
(from sampled)
Good
Machine
Learning
Sampled
Data
Good
Data
Gathering
Real World
Good representation of
Good representation of
$P(X|Y_R) \approx P(X|Y_S) \approx P(X|Y_{S,T})$

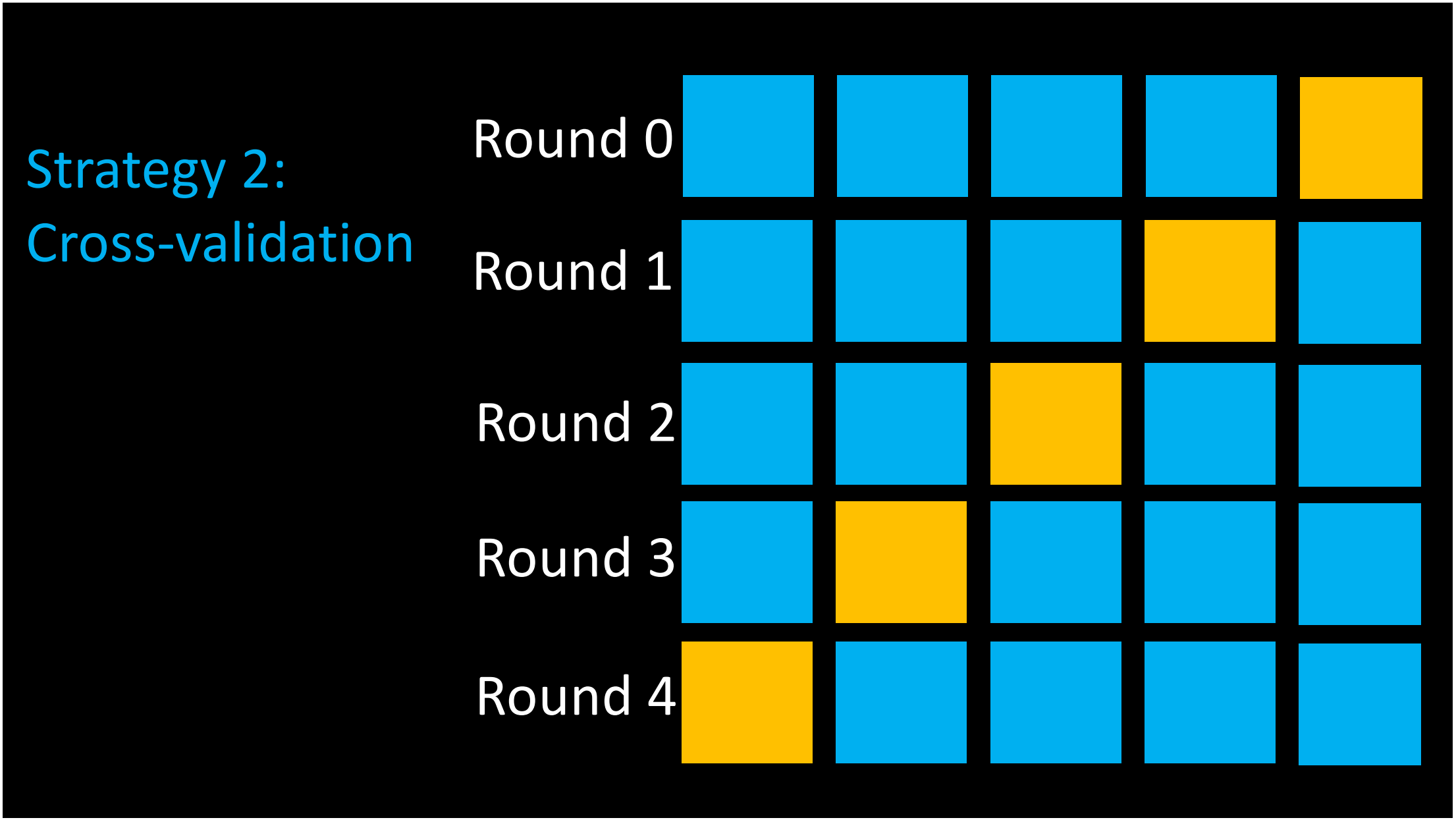
How to split
Train / validation?



Strategy 1:

Random Splitting:

Strategy 2: Cross-validation



Train-Validation:

Easy / wasting data

10 (k) fold:

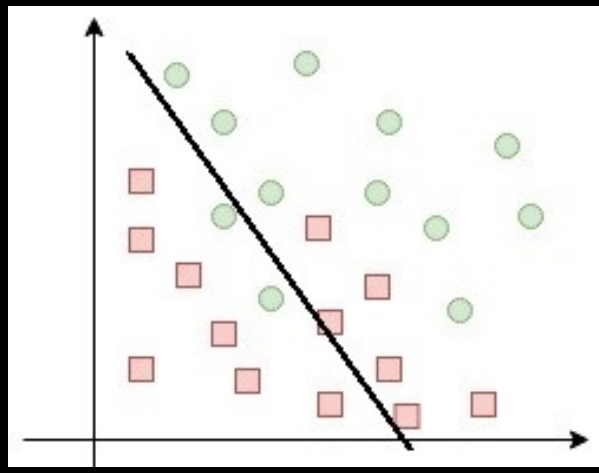
Waste 1/k data, stable

Leave one out:

Waste 1 data, unstable

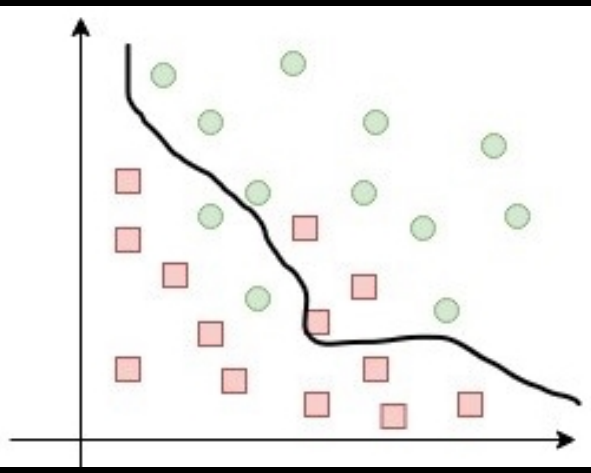
Training / Testing Error

High Training error



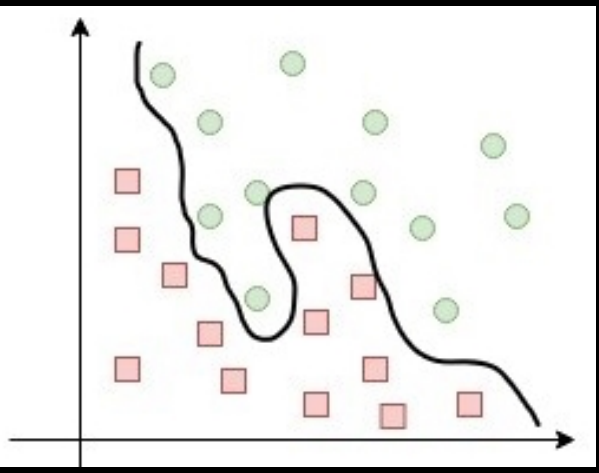
2 parameters
(linear model)

Some Training error



$>$ 2 parameters
(nonlinear model)

0 Training error

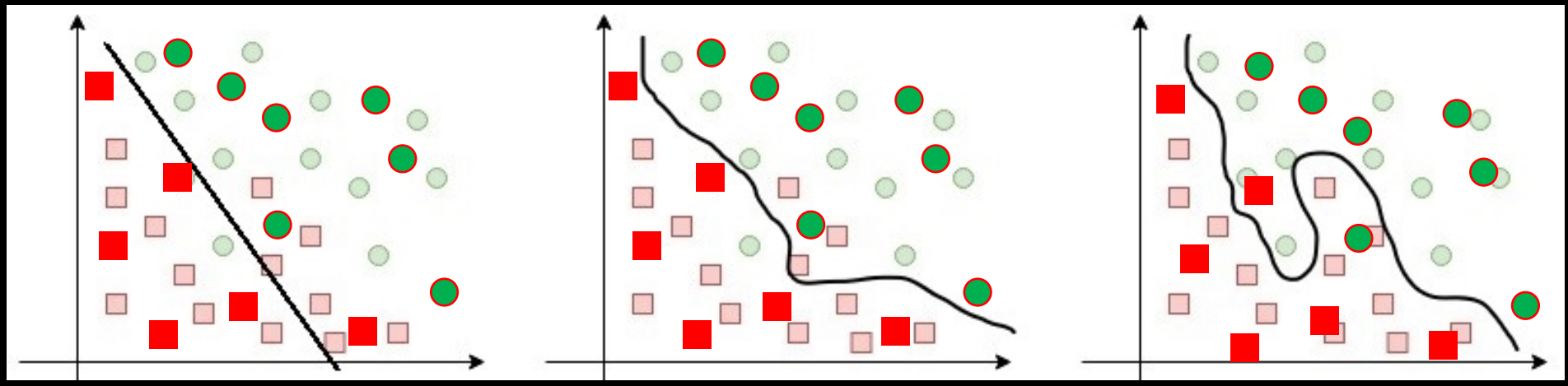


Even more parameters
(nonlinear model)

High Training error

Some Training error

0 Training error



Under-fitting

2 parameters

(linear model)

Normal Fitting

> 2 parameters

(nonlinear model)

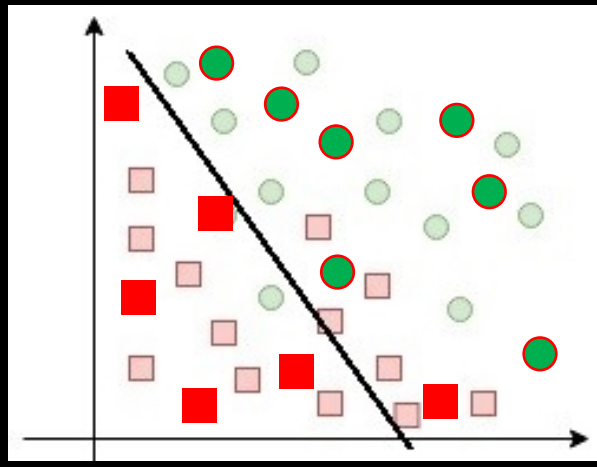
Over-fitting

Even more parameters

(nonlinear model)

Low Testing error

High Training error



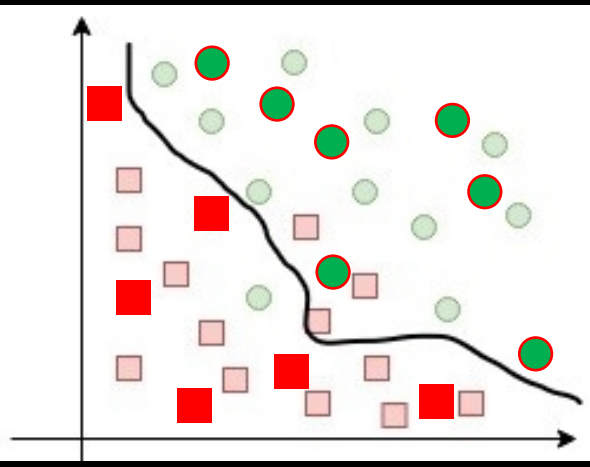
Under-fitting

2 parameters

(linear model)

Some Training error

Some Training error



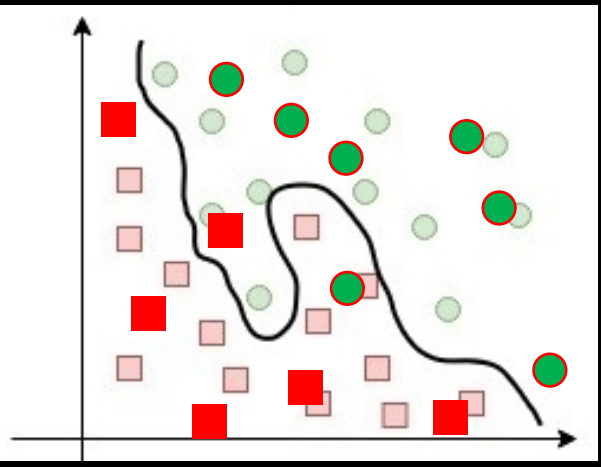
Normal Fitting

> 2 parameters

(nonlinear model)

High Testing error

0 Training error

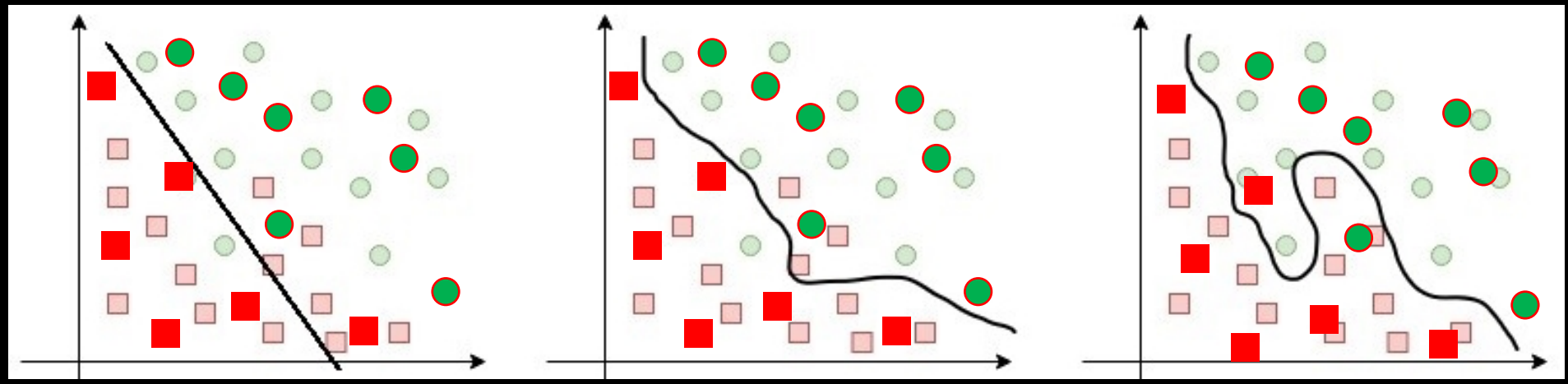


Over-fitting

Even more parameters

(nonlinear model)

More probable according to data



Less probable according to some assumption about model (prior)

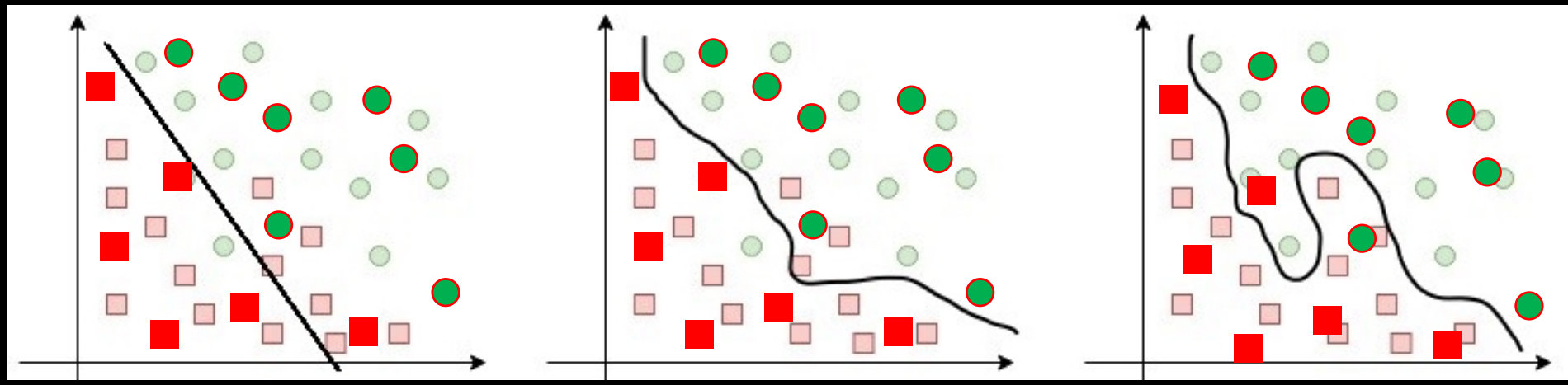
Maximum Likealihood (MLE)

More probable according to data

High Training error

Some Training error

0 Training error



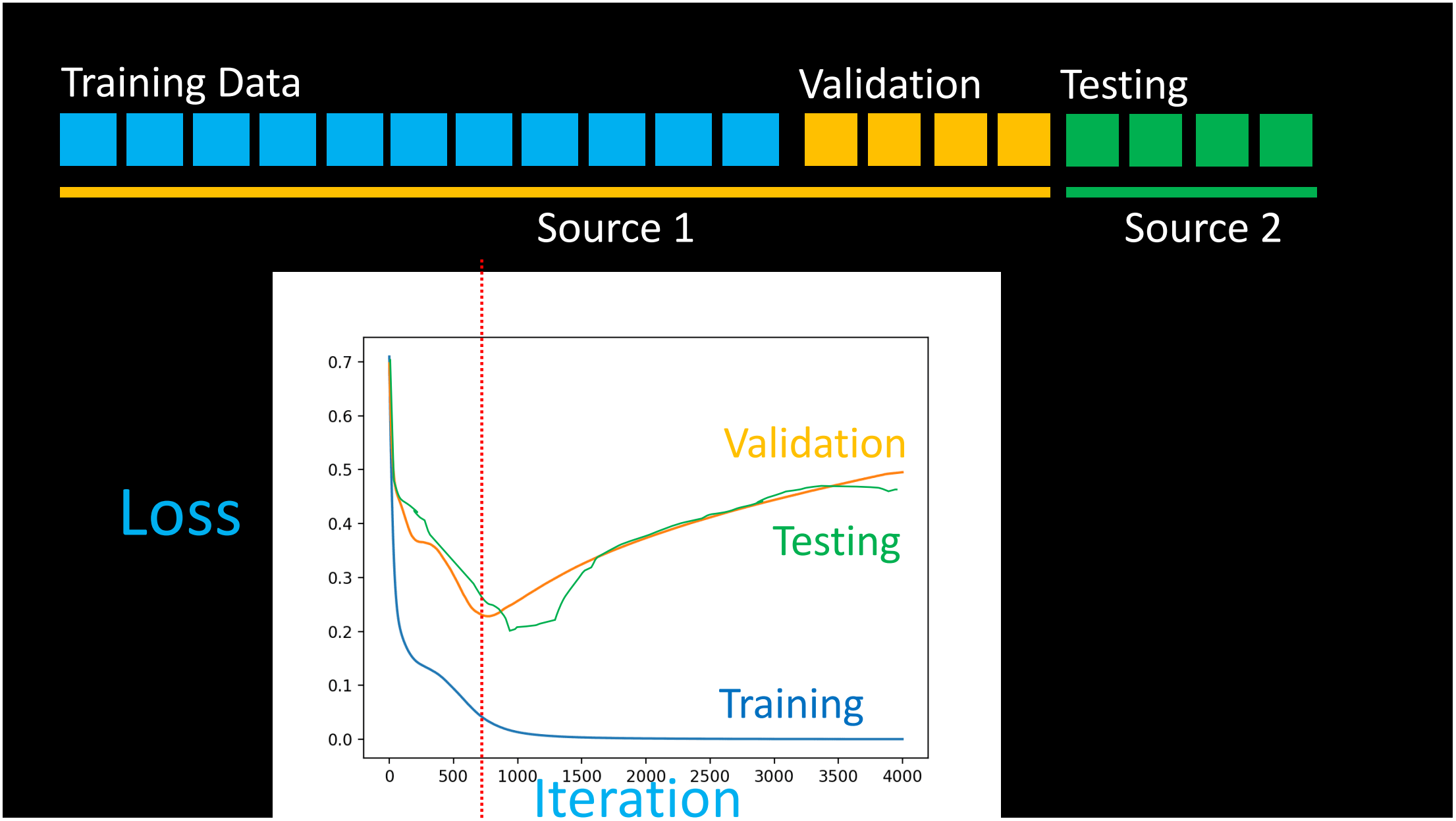
Low Testing error

Some Training error

High Testing error

Maximize a Priori (MAP)

Less probable according to some assumption about model (prior)

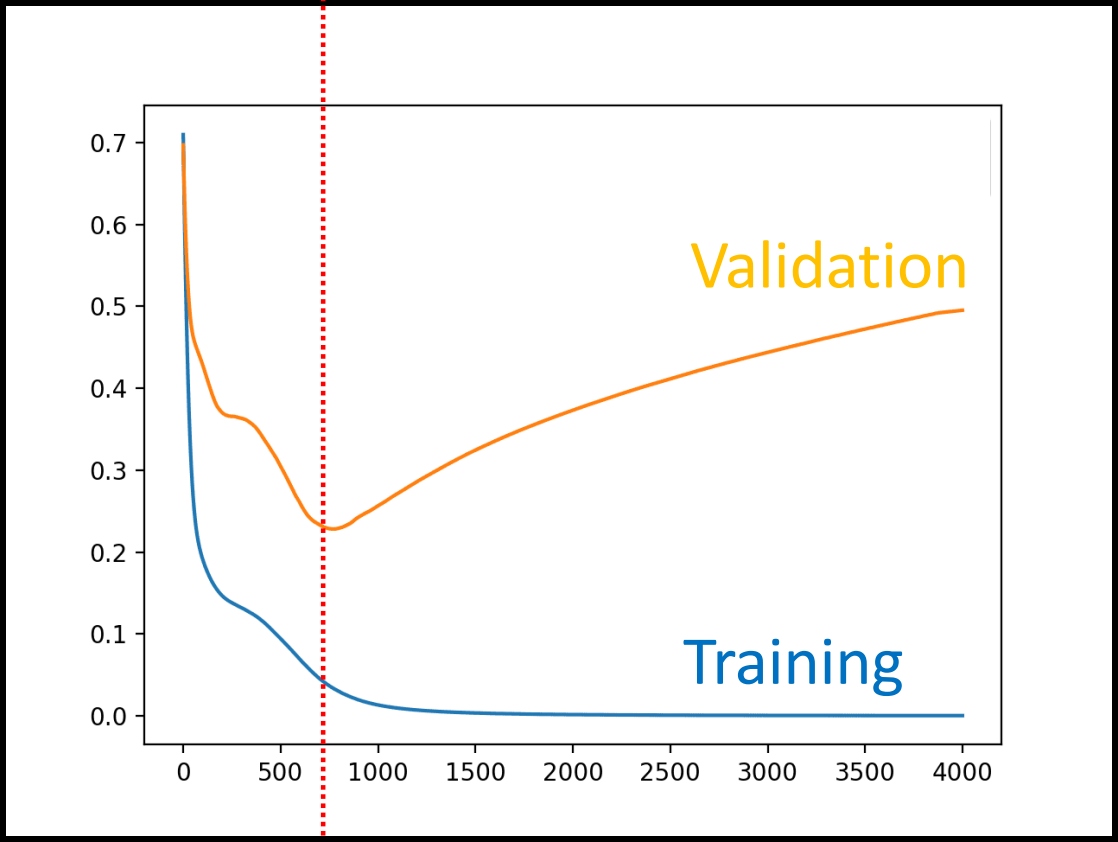

Training Data
Validation
Testing
Source 1
Source 2
Loss
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
500
1000
1500
2000
2500
3000
3500
4000
Validation
Testing
Training
Iteration

Training Data

Validation Data

Source 1

Loss



Underfitting

Overfitting

What cause overfitting?

-- too little training data

-- too much features (compared to the number of training data)

-- model too complicated (again,
compared to the number of training data)

Combat overfitting

--less complicated model

-- tune hyperparameter (learning rate scheduling….)

-- using ensemble models (will talk later)

-- make your data stronger (will talk later)

What's considered cheating (data leakage)

Training mixed up with validation

Training and validation are correlated by some factor
(same subject, related instance)

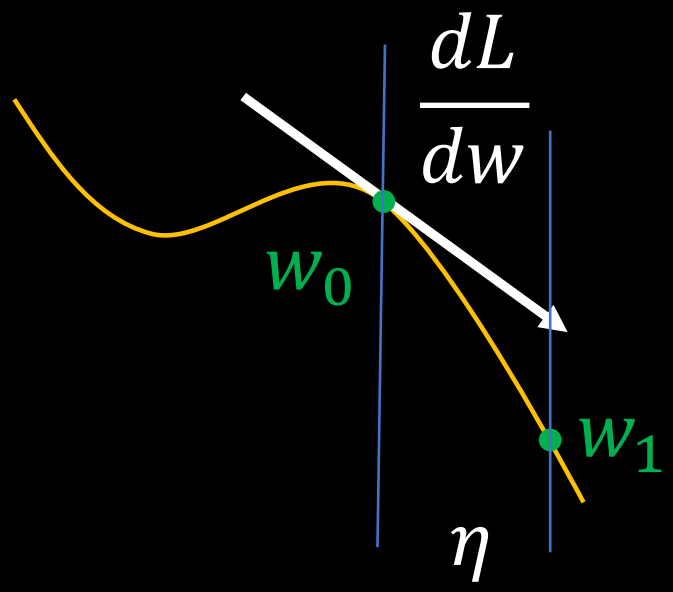
Training and validation had different and identifiable noise / pattern

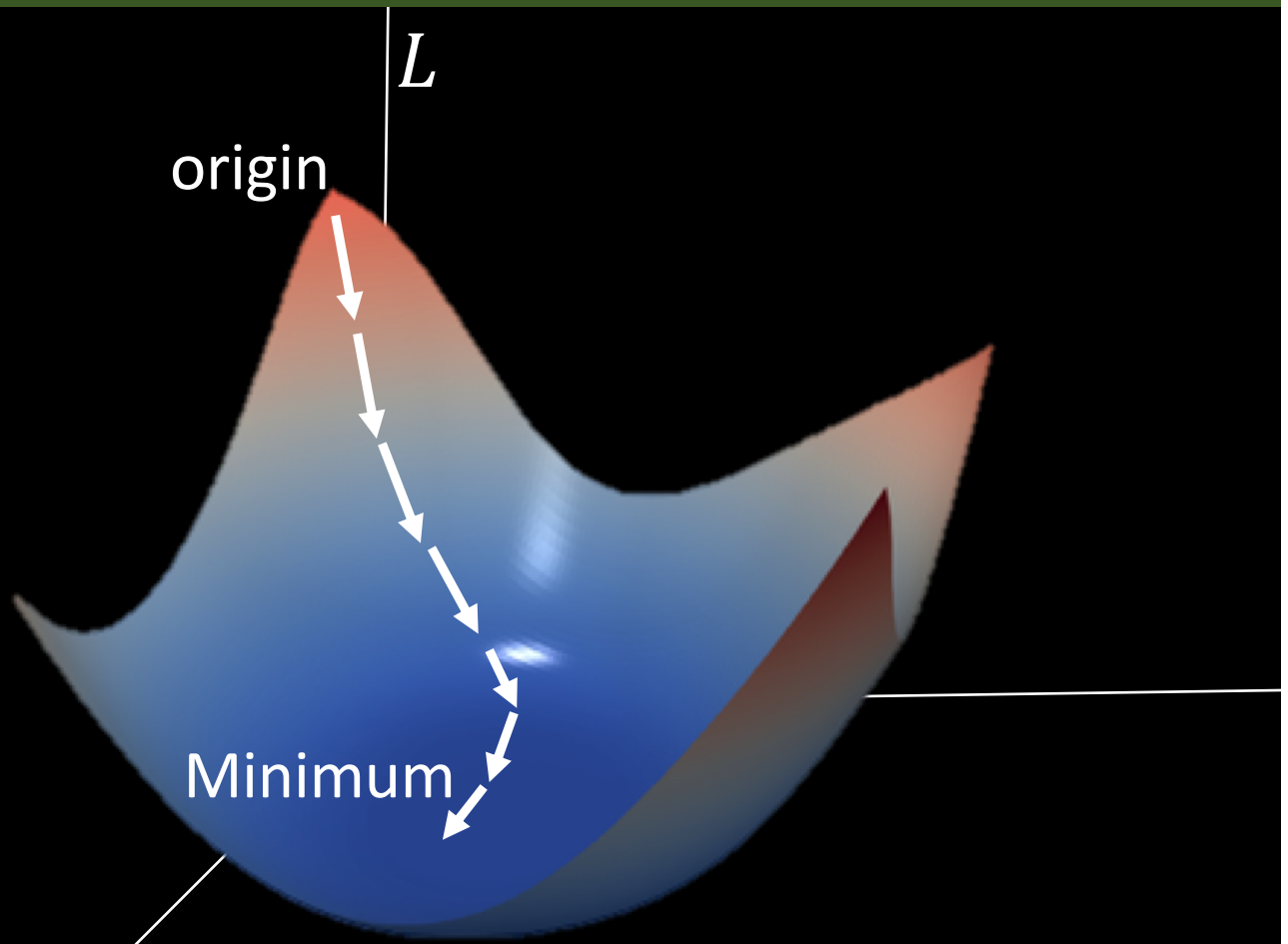
Optimization & Stochastic Gradient Descend (SGD)

Training Neural Networks: optimization

Optimizer

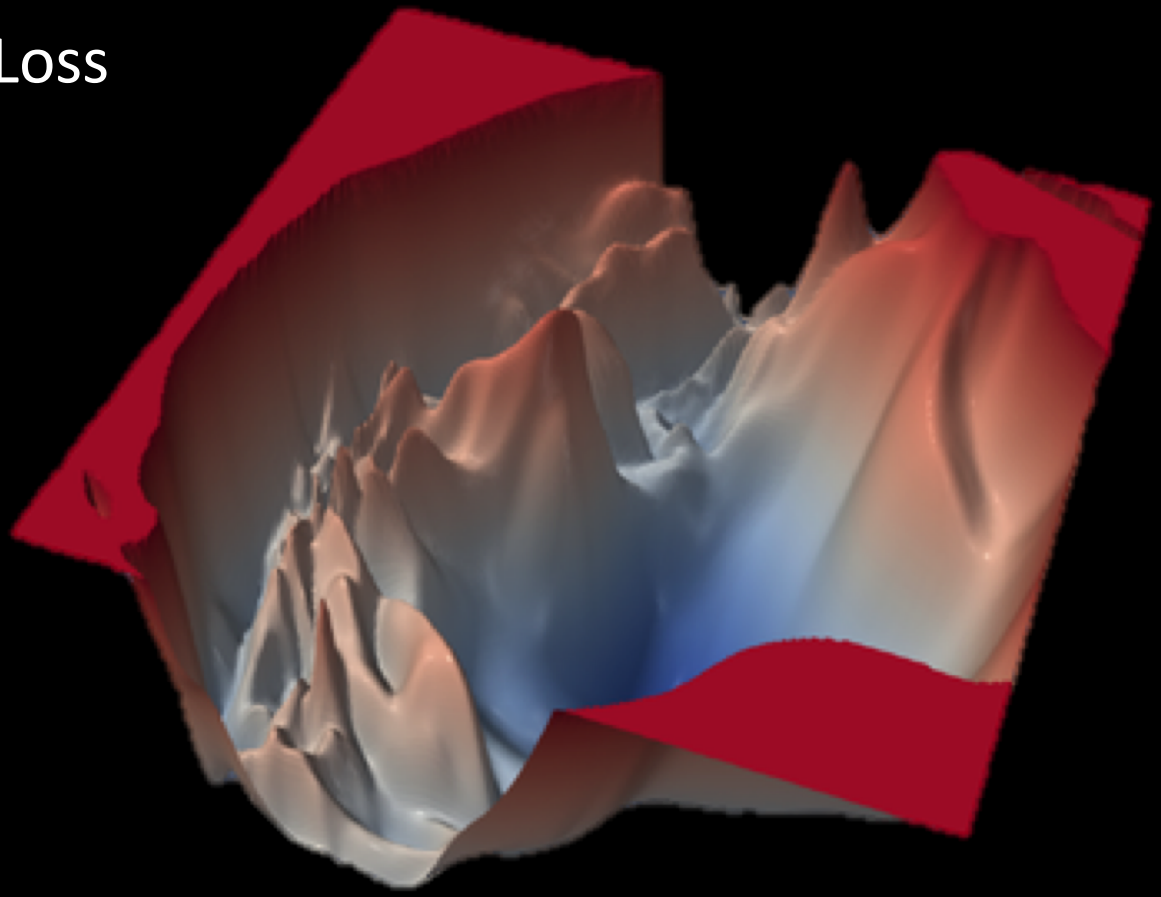
$argmin L(w_i, b)$

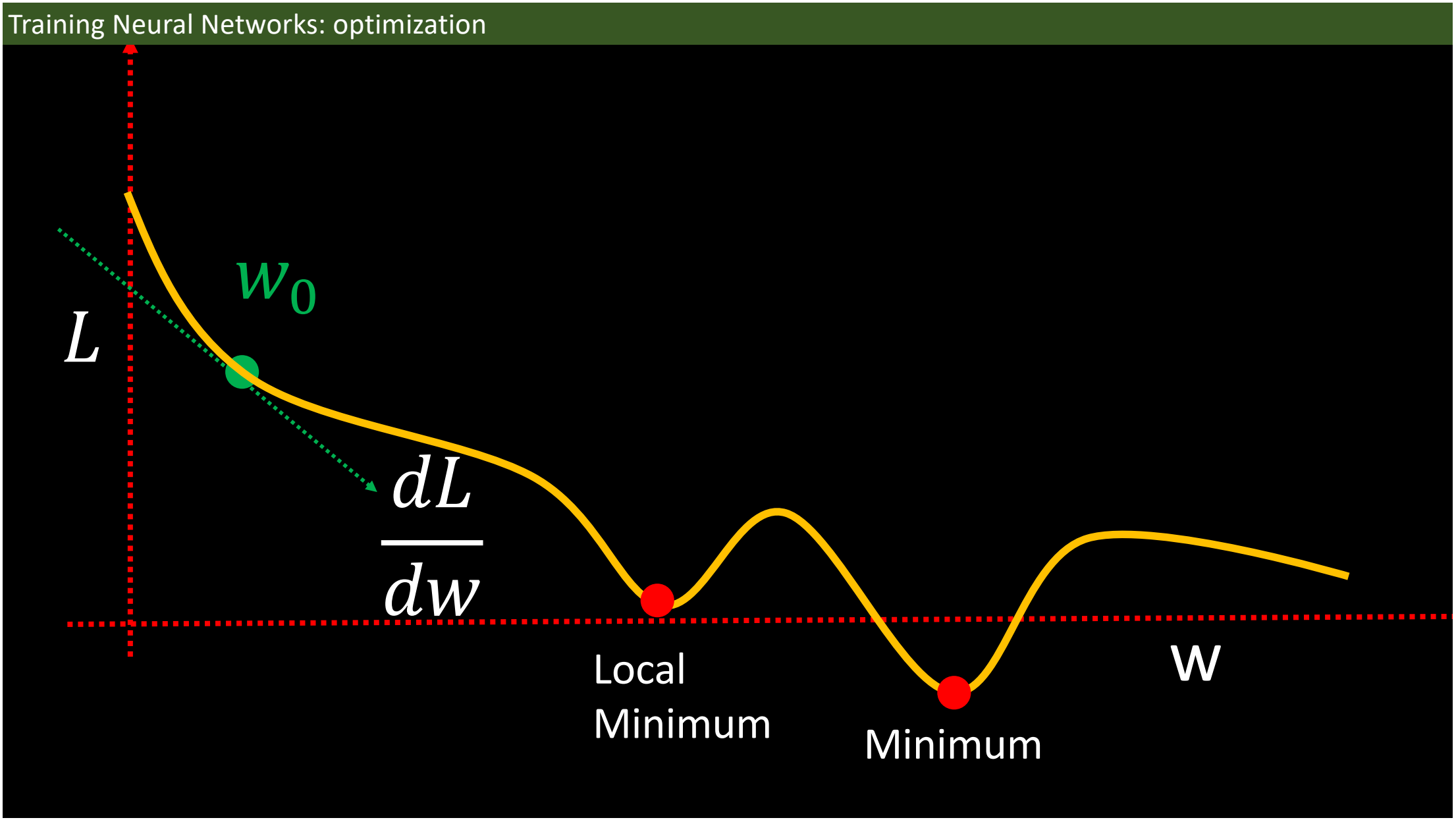




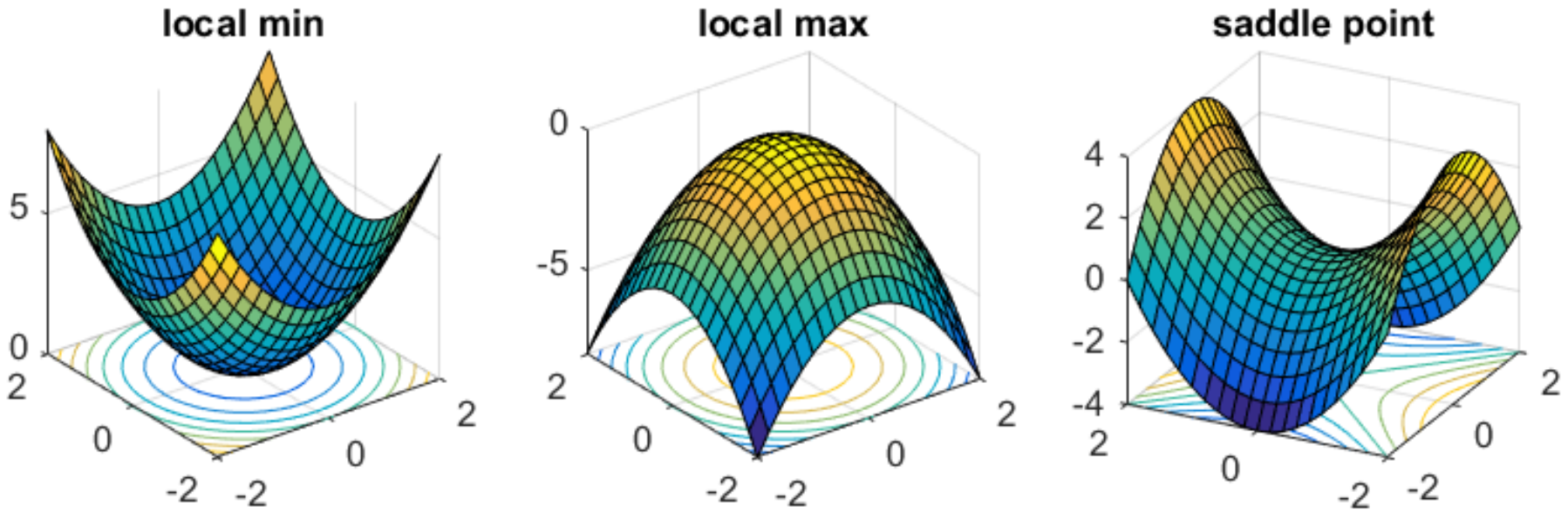
$$w_1 = w_0 - \frac{dL}{dw}\eta$$

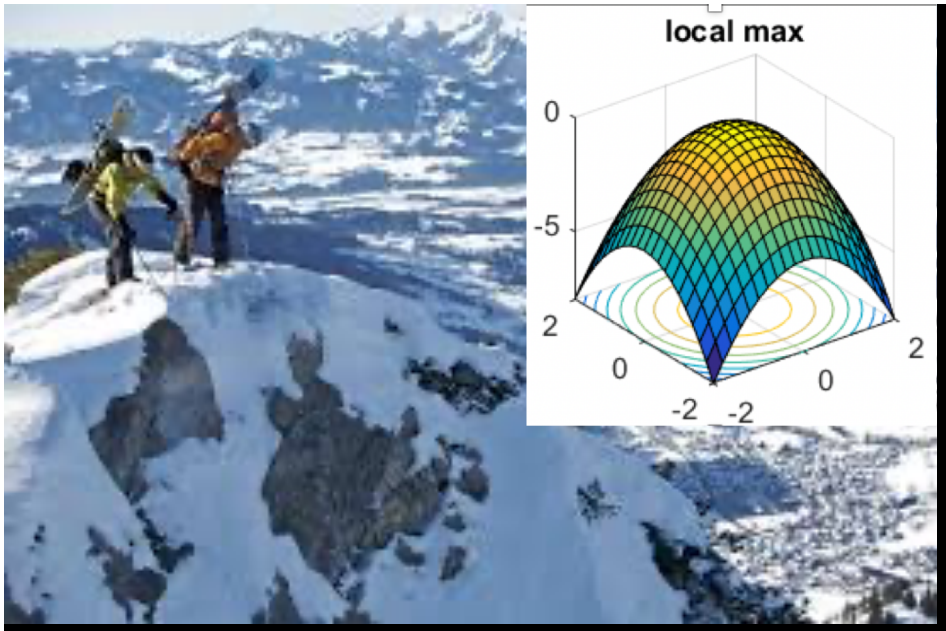
Real Surface of Loss



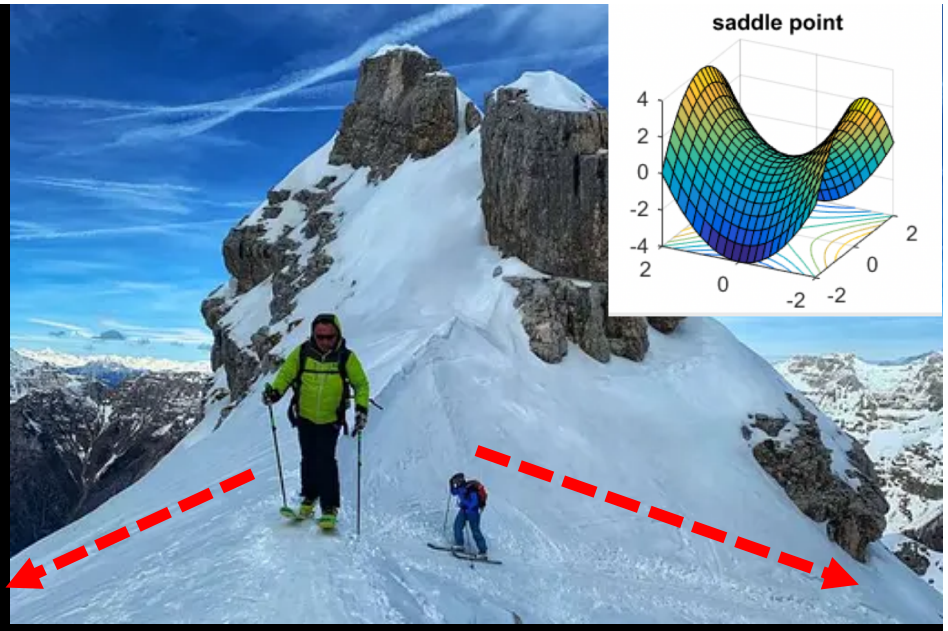

Training Neural Networks: optimization
L
w_0
$\frac{dL}{dw}$
Local
Minimum
Minimum
W

Training Neural Networks: optimization

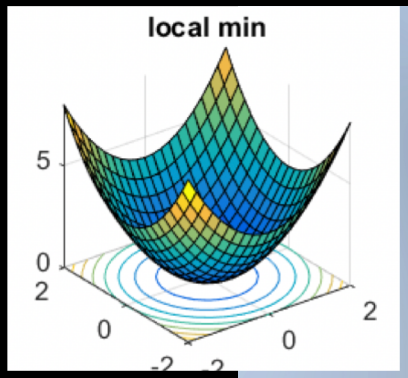




We don't care about this

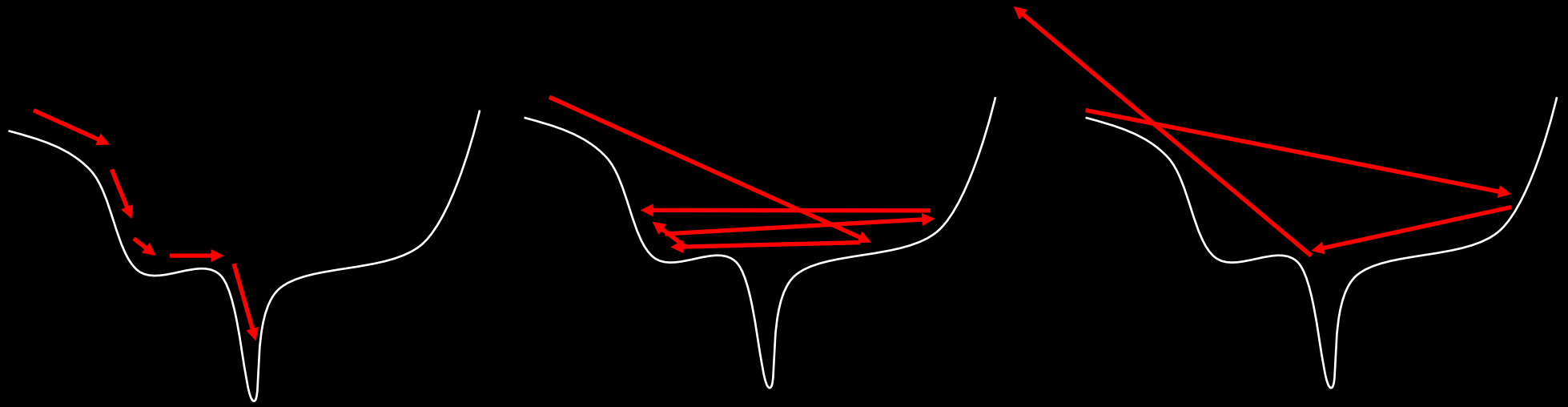


Most common

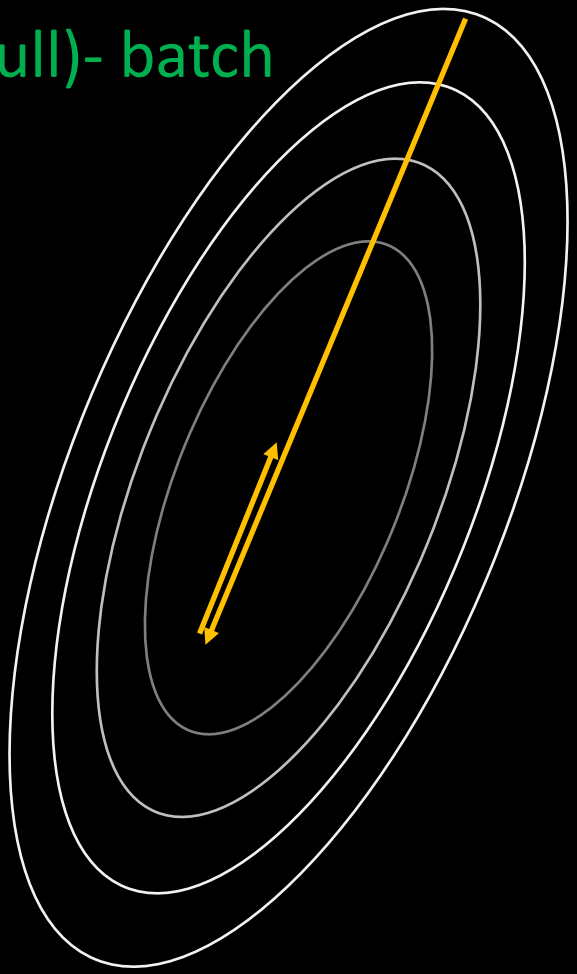


Your model is stuck

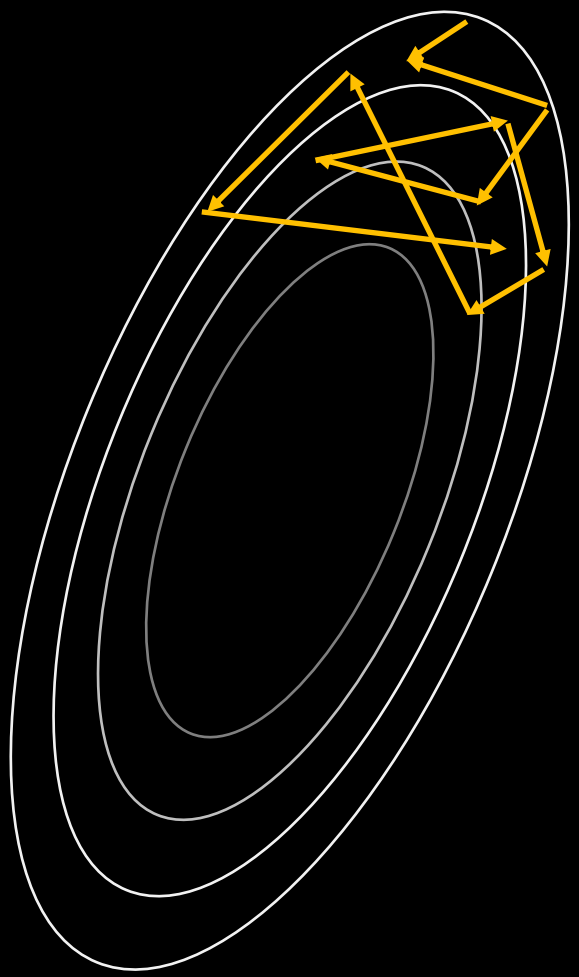
Avoiding stuck in local minimum
1: learning rate



(Full)- batch



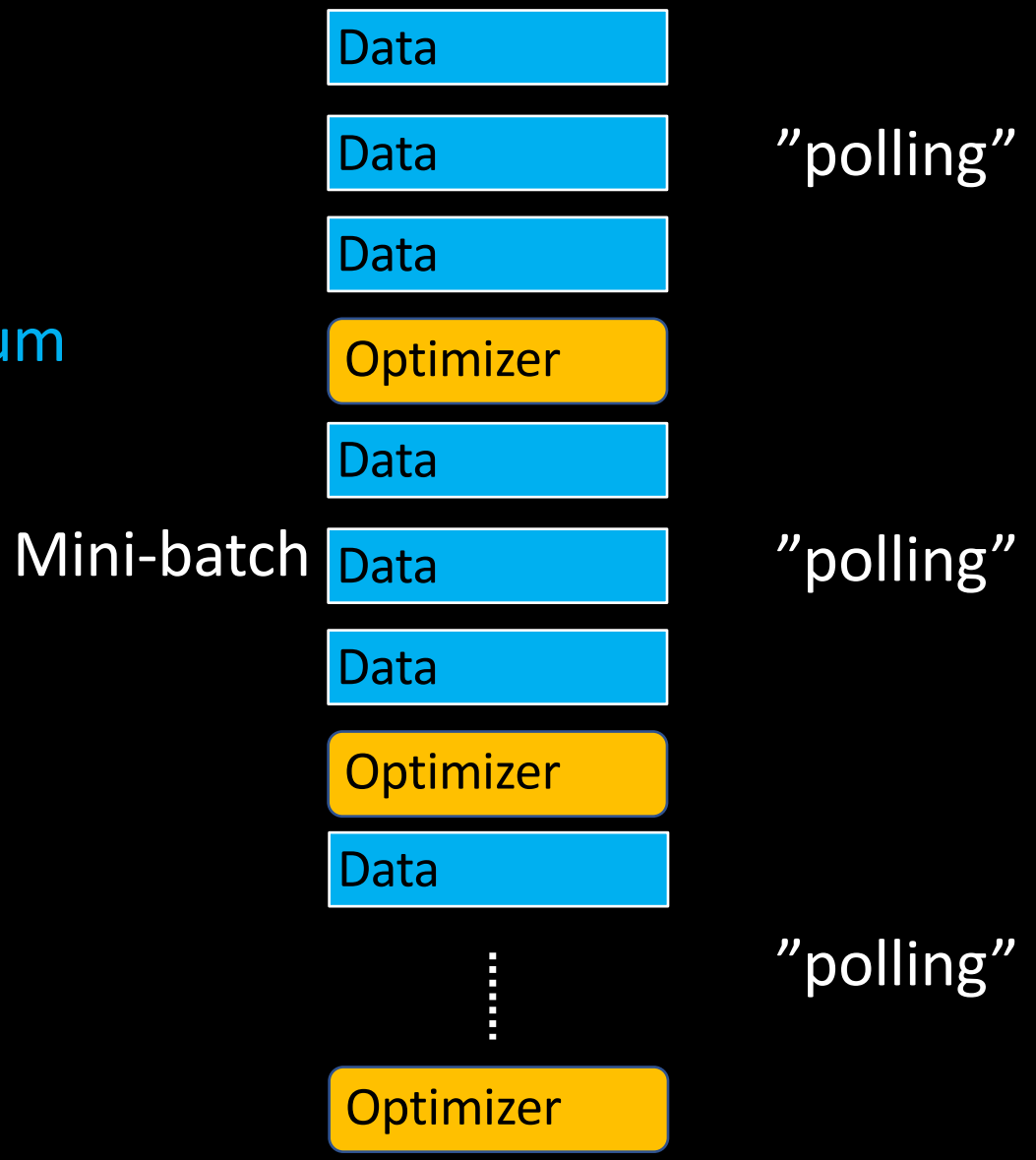
“deterministic”

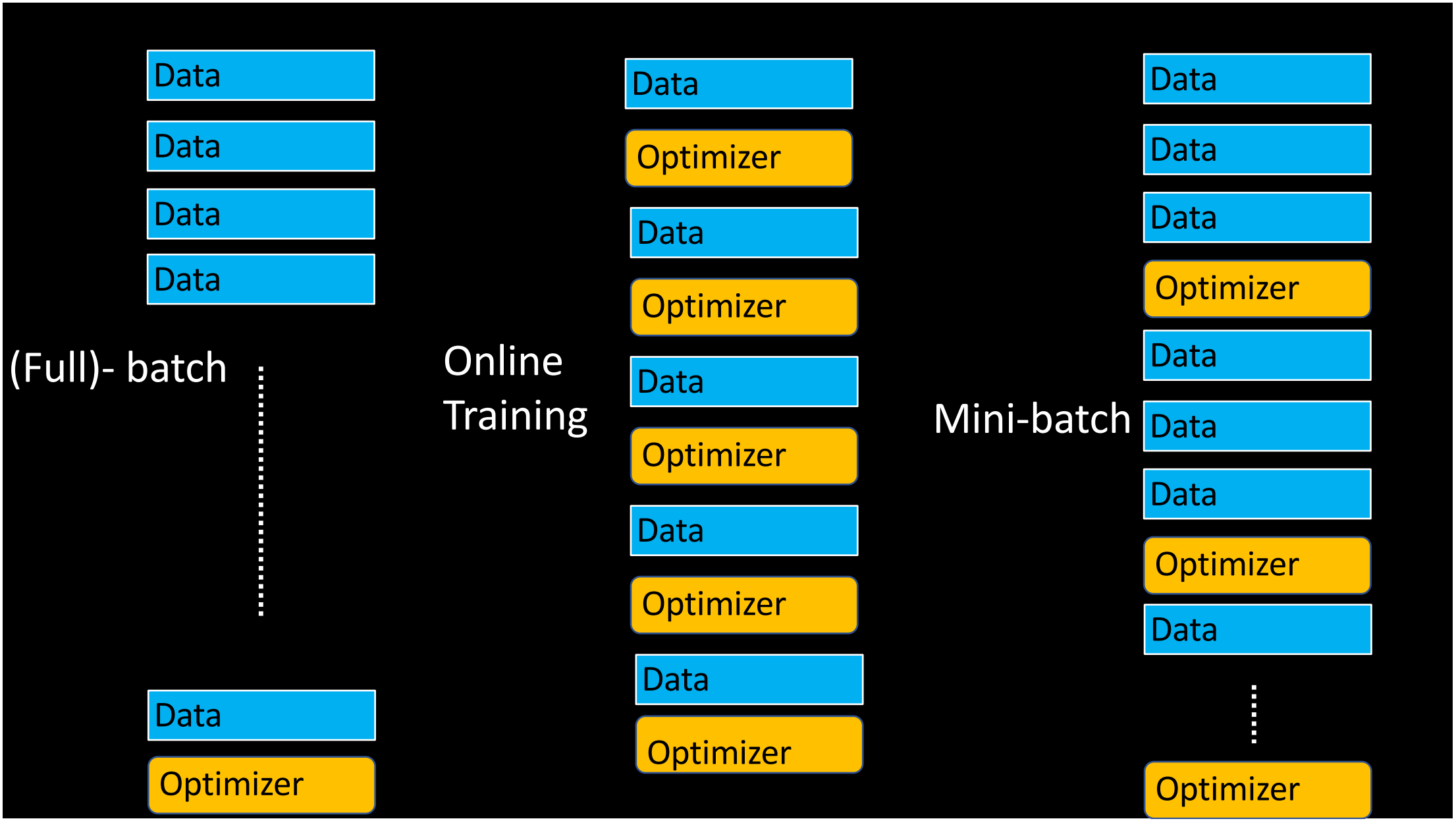


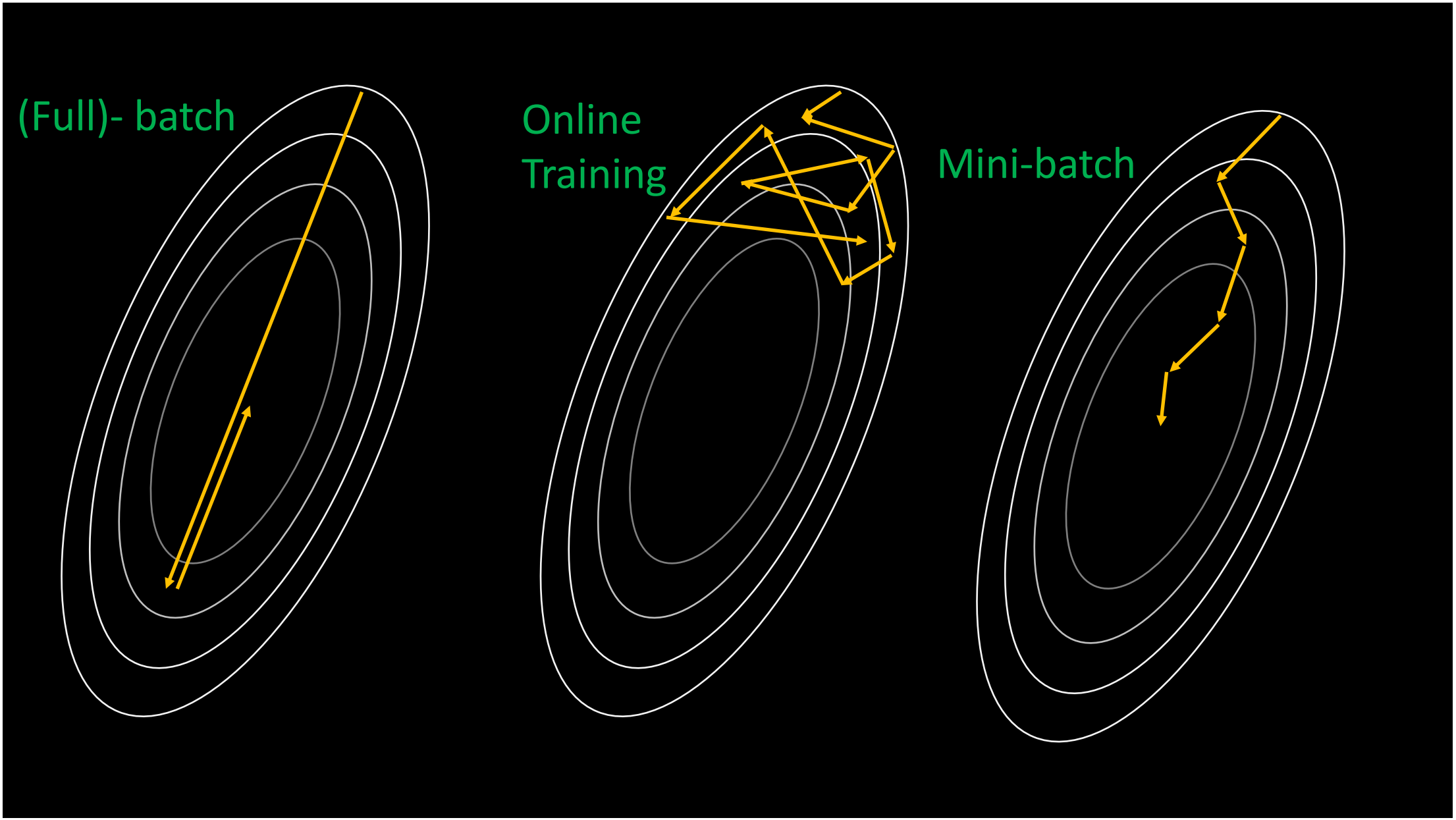
Online (one by one)
Training

“tend to be crazy”

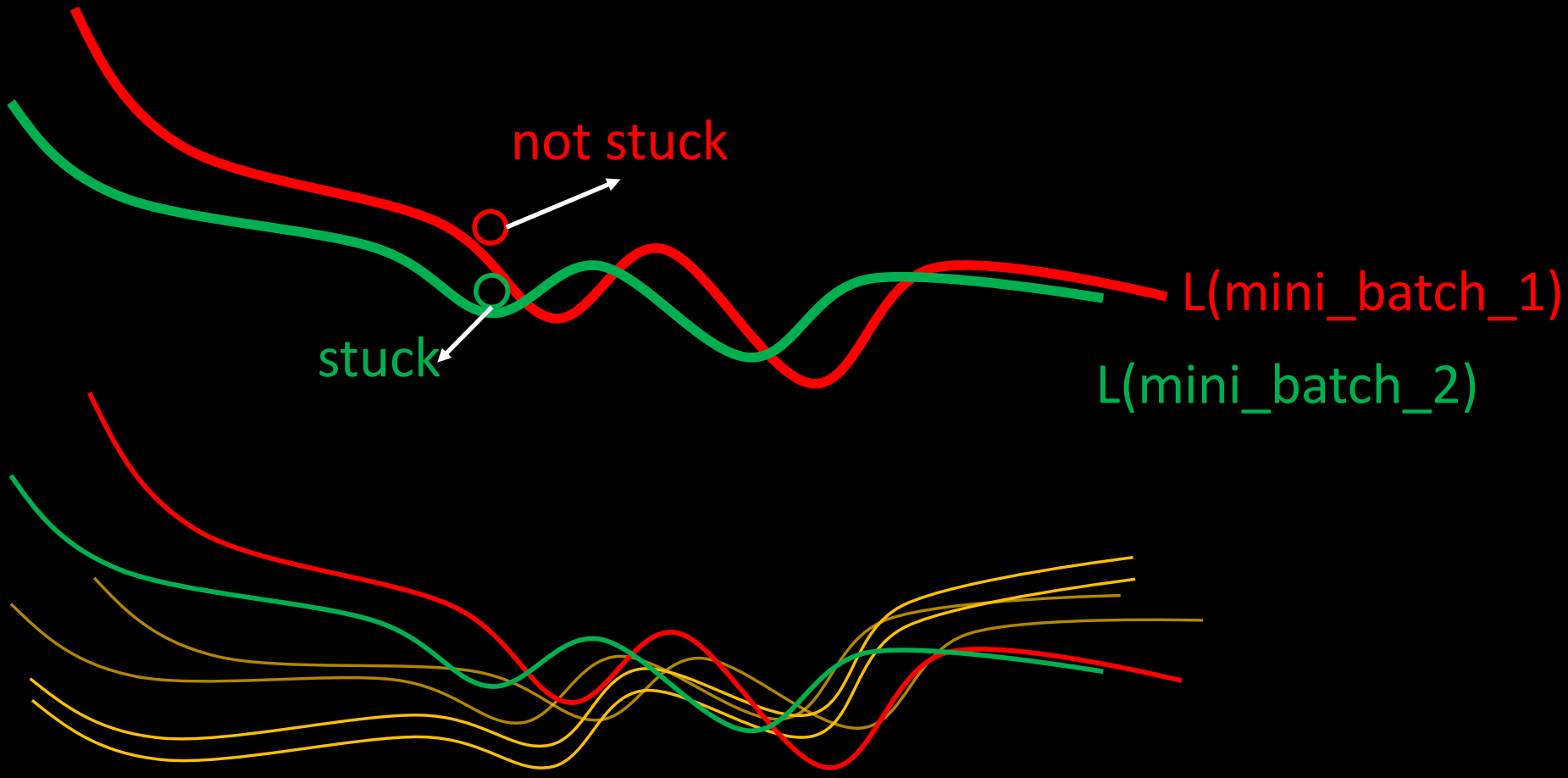
Avoiding stuck in local minimum
2: minibatch sampling



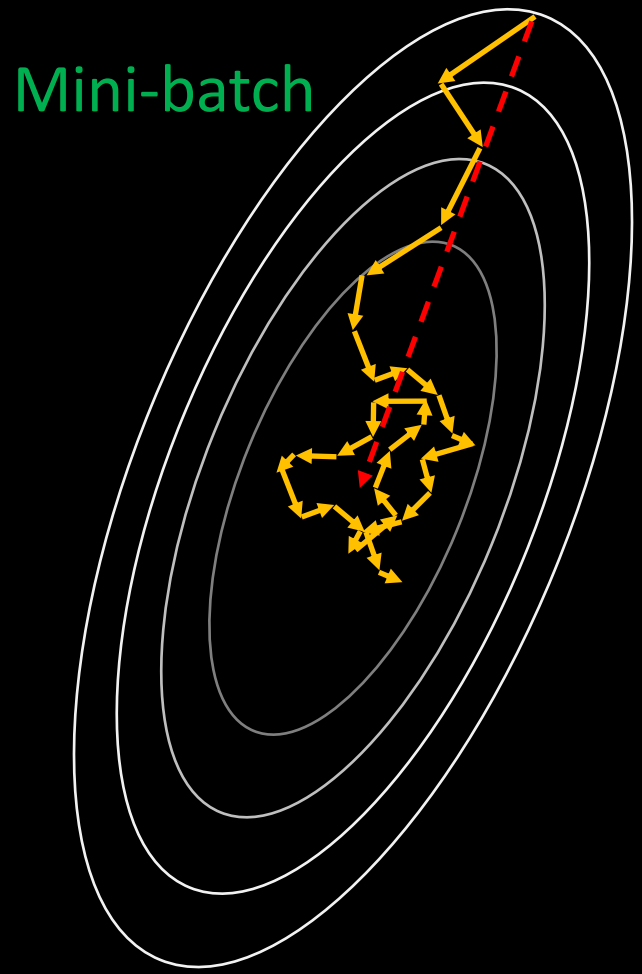

Data
Data
Data
Data
(Full)- batch
Data
Optimizer
Online
Training
Data
Optimizer
Data
Optimizer
Data
Optimizer
Data
Optimizer
Data
Optimizer
Mini-batch
Data
Data
Data
Optimizer
Data
Data
Data
Optimizer
Data
Optimizer

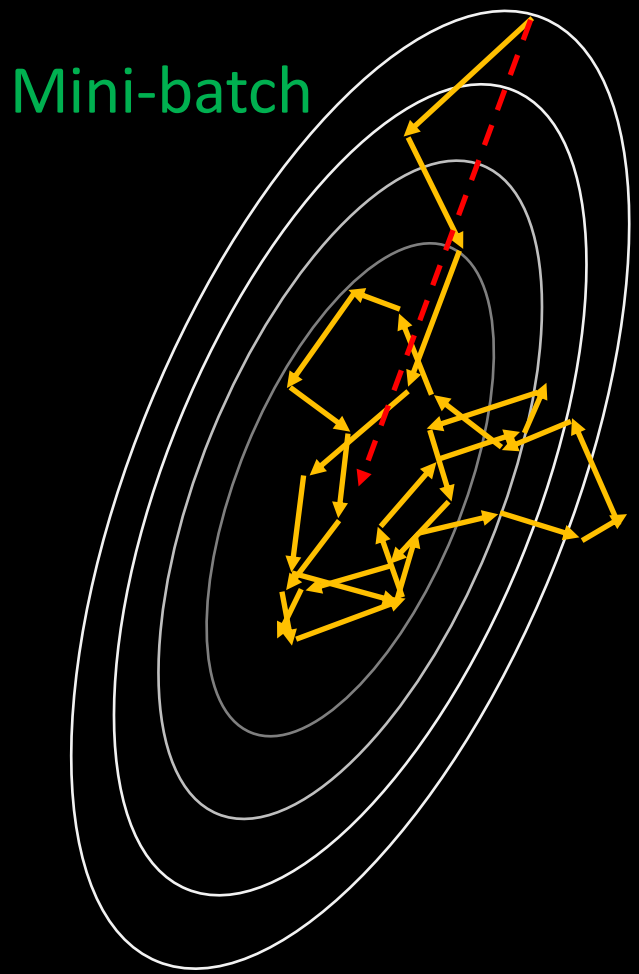

(Full)- batch
Online
Training
Mini-batch

Minibatch sampling “shake up” the loss function

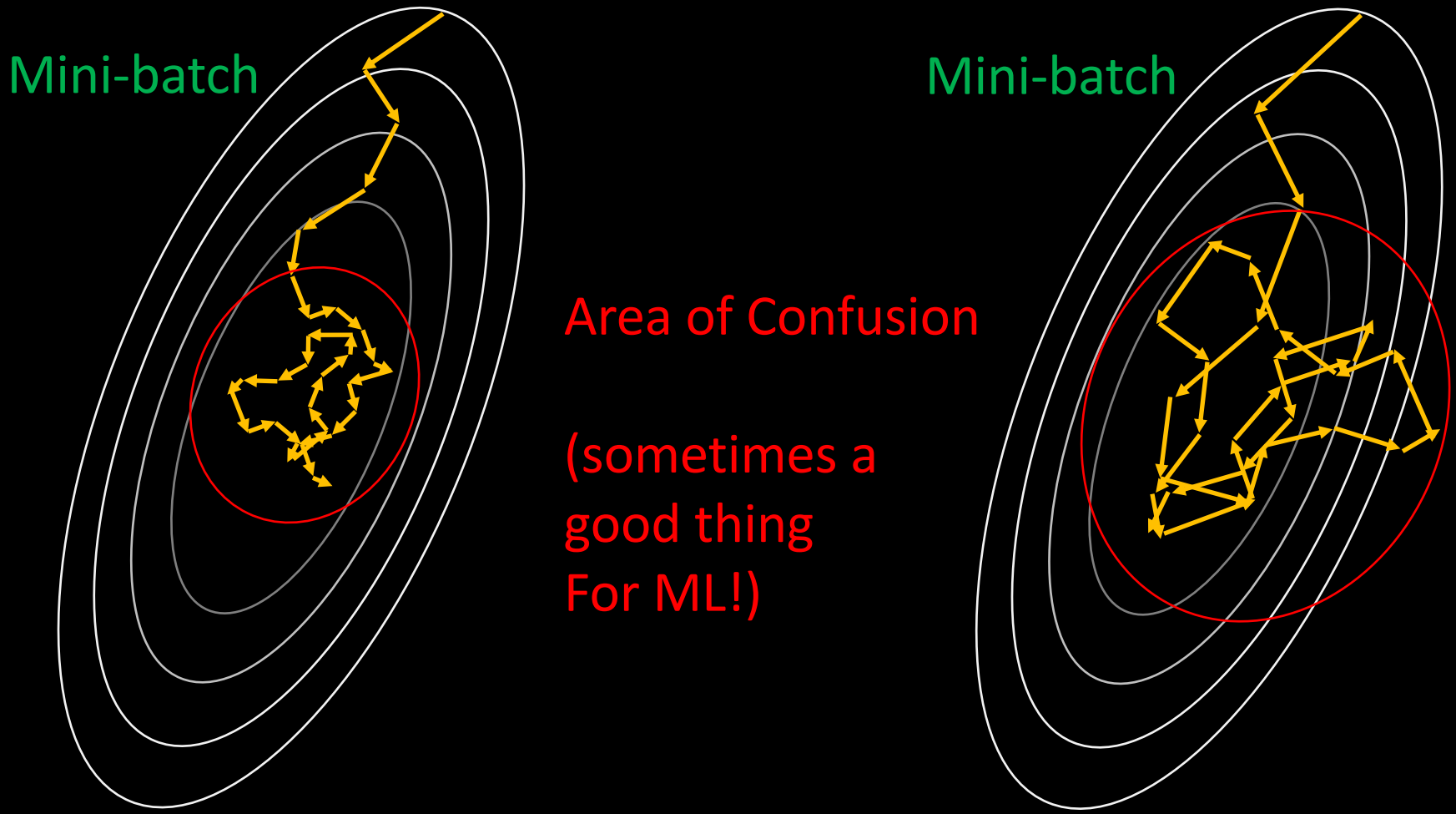


Training Neural Networks: minibatch





Training Neural Networks: minibatch

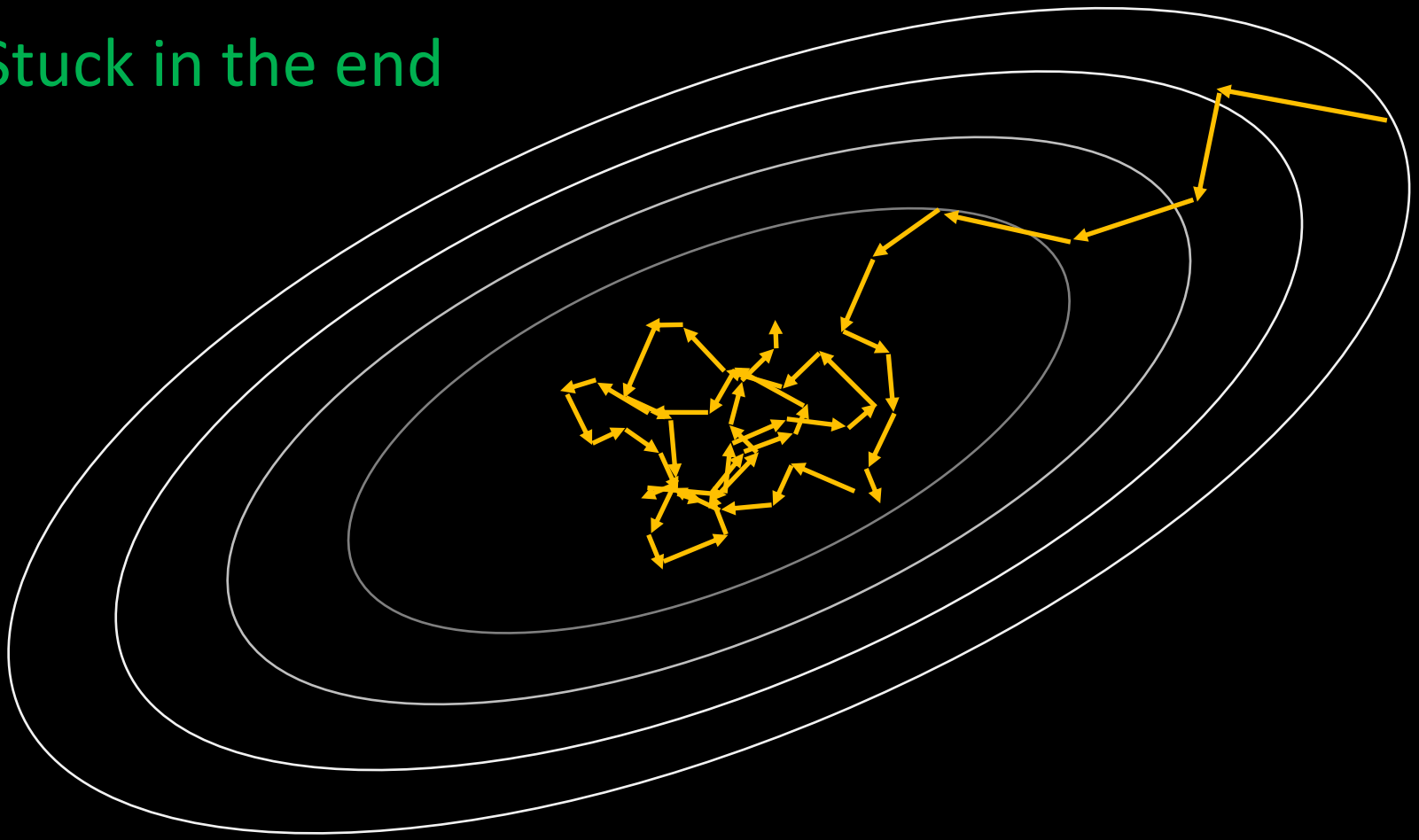


One example of modern deep learning training

	Hardware	Chips	Batch	Optimizer	BN	Accuracy	Time
Goyal et al. [6]	P100	256	8192	Momentum	Local	76.3%	1 hour
Smith et al. [16]	TPU v2	128	8192 → 16384	Momentum	Local	76.1%	30 mins.
Akiba et al. [2]	P100	1024	32768	RMS + Mom.	Local	74.9%	15 mins.
Jia et al. [10]	P40	1024	65536	LARS	Local	76.2%	8.7 mins.
Baseline	TPU v2	4	1024	Momentum	Local	76.3%	8.0 hours
Ours	TPU v2	256	16384	Momentum	Local	75.1%	10 mins.
Ours	TPU v2	256	32768	LARS	Local	76.3%	8.5 mins.
Ours	TPU v3	512	32768	LARS	Local	76.4%	3.3 mins.
Ours	TPU v3	1024	32768	LARS	Distributed	76.3%	**2.2 mins.**

Fast Progress in the Beginning

Fussy / Stuck in the end



Good for big data!

All Data

0	1	2	3	4	5	6	7	8	9

Random Shuffle

8	3	2	7	0	1	6	5	4	9

Sample without replacement…

8	3	2	7	0	1	6	5	4	9

Sample with replacement…

3	2	7	2	1	9	8	1	4	3

There are tones of research go into
how to fix crazy/stuck in the end

(not always matters in practice)

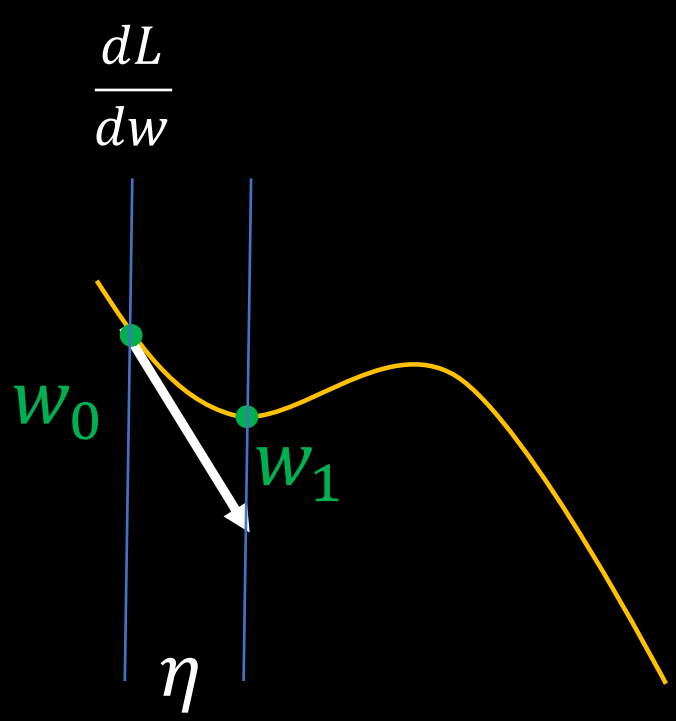
TIPS

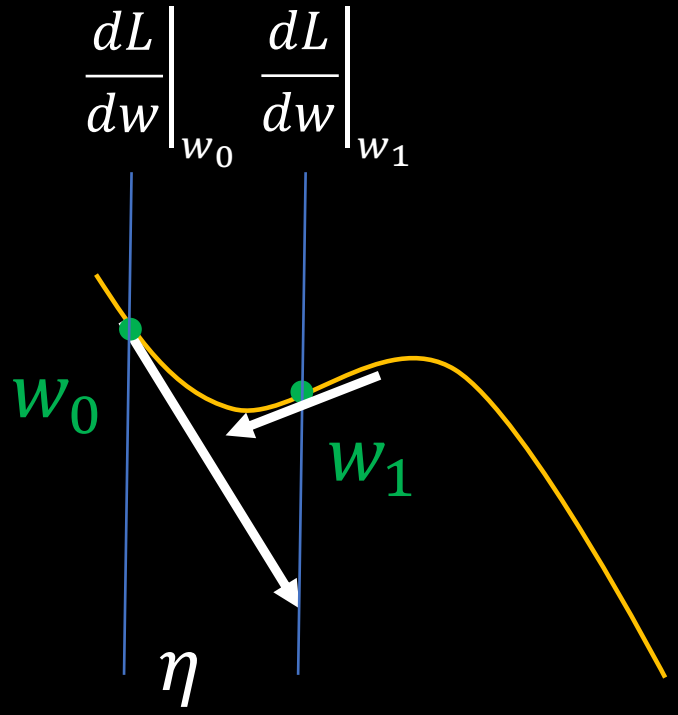
- schemes of gradient calculation

- find the right learning rate to start

- find the good way to change learning rate during training

Momentum





$$w_1 = w_0 - \frac{dL}{dw}\eta$$

$$w_1 = w_0 - \left(\left.\frac{dL}{dw}\right|_0 + \beta \left.\frac{dL}{dw}\right|_{-1}\right)\eta$$

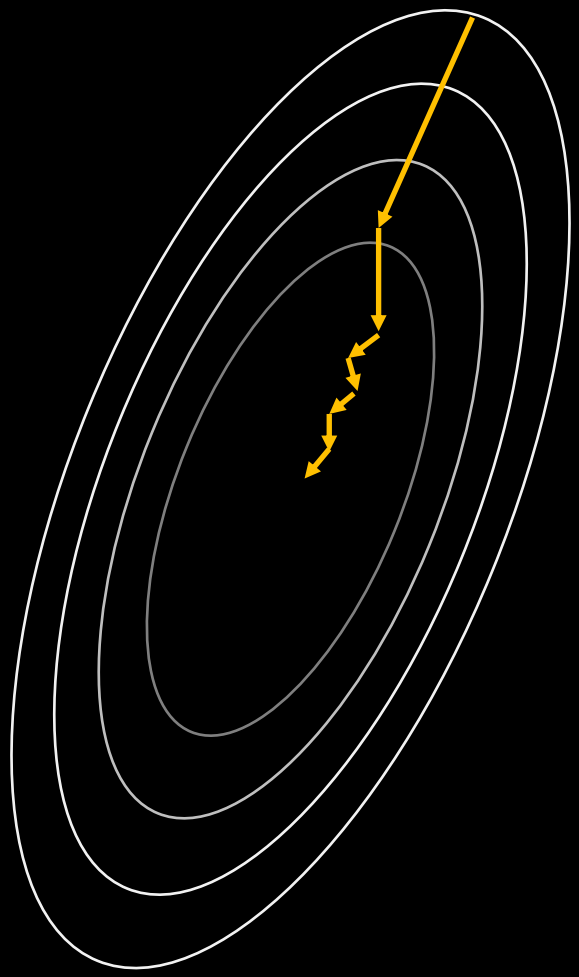
Momentum


gettyimages
Credit: Ashley Jouhar
82214105

Adagrad (adaptive)

$$w_1 = w_0 - \eta \left.\frac{dL}{dw}\right|_0 \epsilon$$

$$\epsilon = \frac{1}{\sqrt{\sum_0^t (\frac{dL}{dw})^2 + \varepsilon}}$$

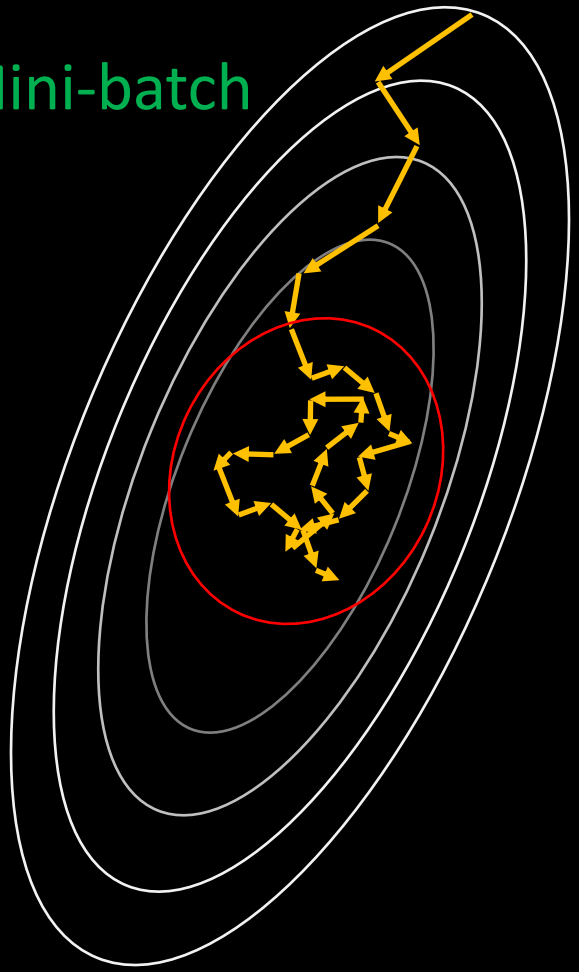


Adams

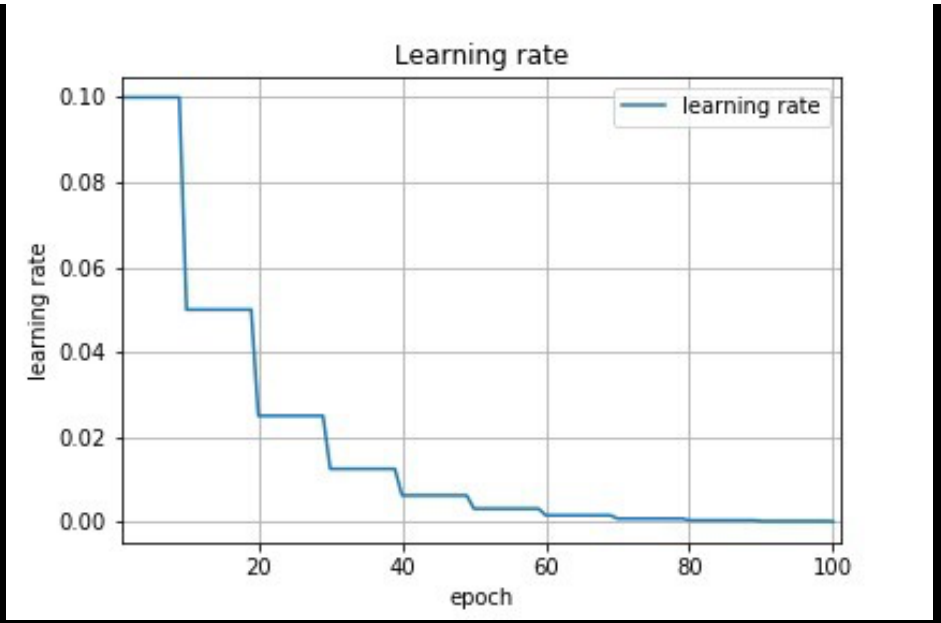
$$w_1 = w_0 - \eta \left(\left.\frac{dL}{dw}\right|_0 + \beta \left.\frac{dL}{dw}\right|_{-1}\right)\epsilon$$

Momentum Ada

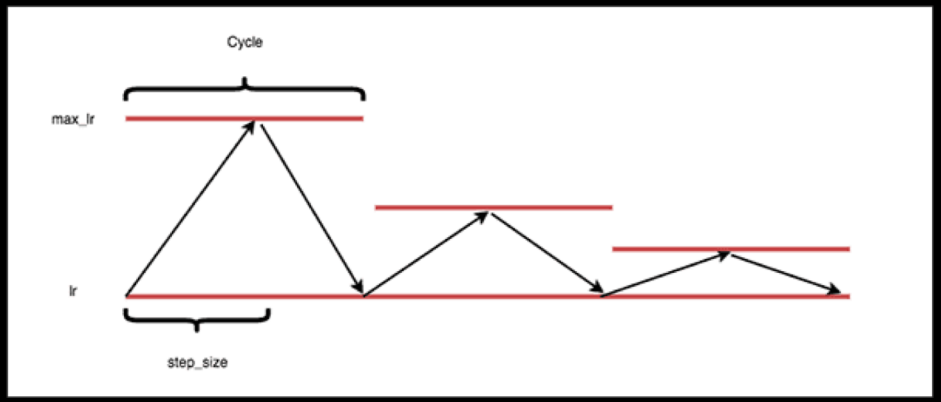
Mini-batch

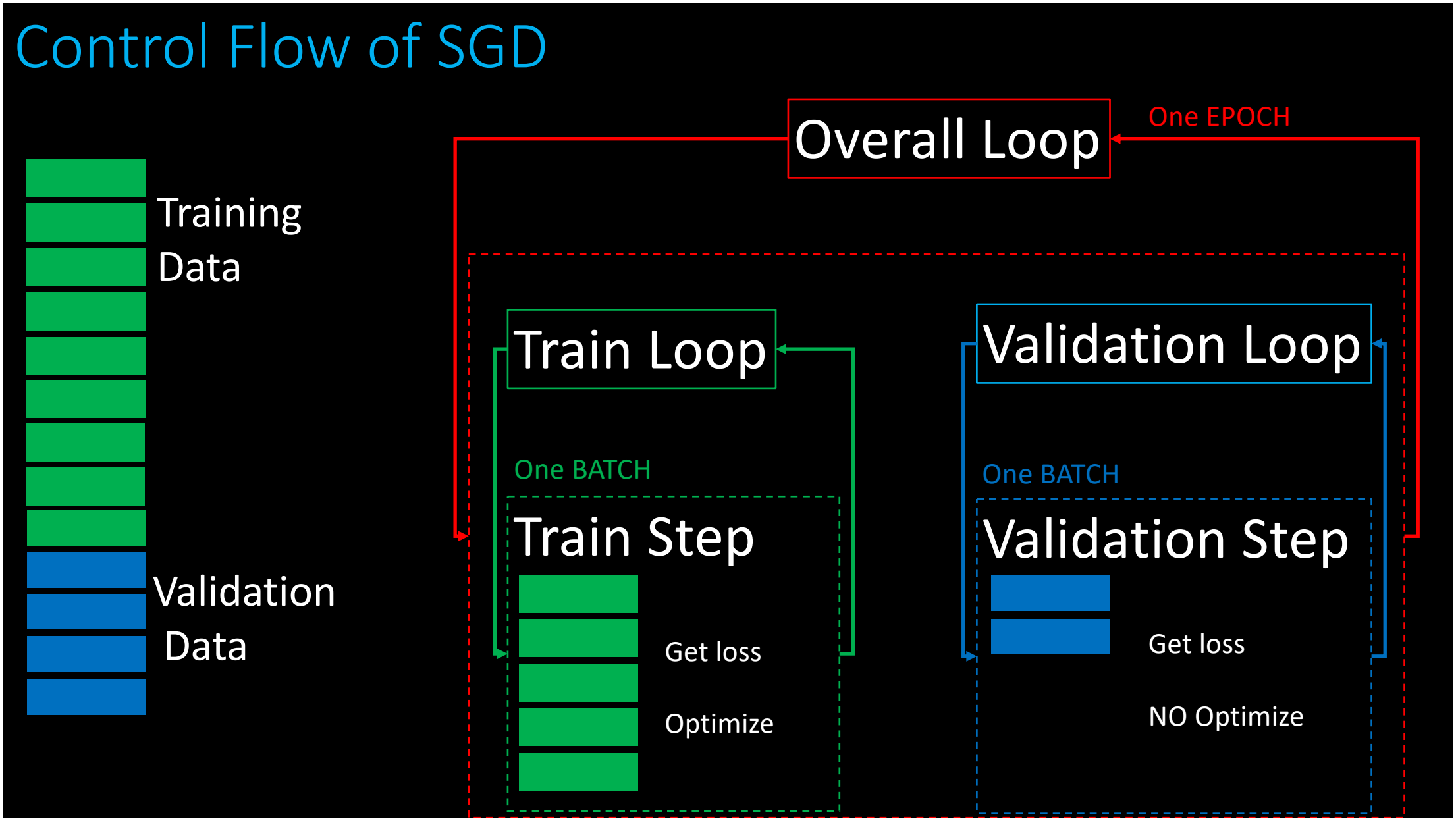


Learning rate decay



Cyclic learning rate




Control Flow of SGD
Training Data
Validation Data
Overall Loop
One EPOCH
Train Loop
One BATCH
Train Step
Get loss
Optimize
Validation Loop
One BATCH
Validation Step
Get loss
NO Optimize

Control Flow of SGD

Overall Loop

For epoch = 1…..n_epoch:

Train Loop

for batch = 1 ….. n_batch_training:

do training_step()
do optimize()

Validation Loop

for batch = 1 ….. n_batch_validation:

do validation_step()