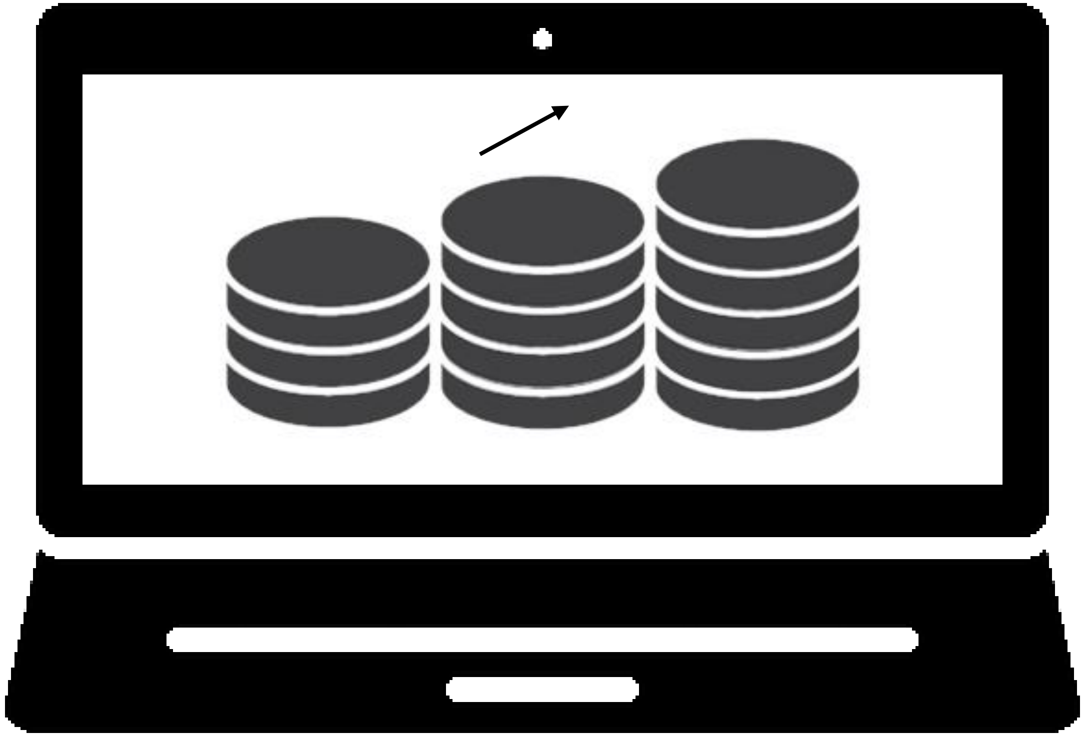
Scaling Tricks



Learning a large scale model, Step 0

Large scale: costly and difficult to train.

First try to **downsize** the dataset and train on a **single machine.**

➔ Gives you a better understanding of the task
➔ Model learned might be good enough !
➔ These tricks will still be interesting when going distributed

My data does not fit in RAM !

```python
df = pd.read_csv("my_big_file.csv")
```

Pandas optimization

- Pandas is quite memory inefficient
- One issue is that at column type inference time, it casts them to the largest possible type
- Pandas types
  - **int8 / uint8 :** consumes 1 byte of memory, range between -128/127 or 0/255
  - **bool** : consumes 1 byte, true or false
  - **float16 / int16 / uint16:** consumes 2 bytes of memory, range between -32768 and 32767 or 0/65535
  - **float32 / int32 / uint32 :** consumes 4 bytes of memory, range between -2147483648 and 2147483647
  - **float64 / int64 / uint64:** consumes 8 bytes of memory

Pandas optimization

- Categorical columns
  - If you have some categorical columns in your dataset (with strings inside for instance) they are stored as objects
  - If the number of possible categories is limited you can force pandas to use a virtual mapping table where all unique values are mapped via an integer instead of a pointer. This is done using the **category datatype**.

Pandas optimization

- So what can we do ?
- You can:
  - Inspect a representative number of lines of your dataset
  - Downcast or convert into categorical the appropriate columns
  - Use this new schema to load your whole dataset

Pandas optimization

- Don’t forget to only load the subset of columns you are going to use

```
df_optimized = pd.read_csv("my_big_file.csv", dtype=column_types, usecols=["float_col", "int_col", "cat_col"])
```

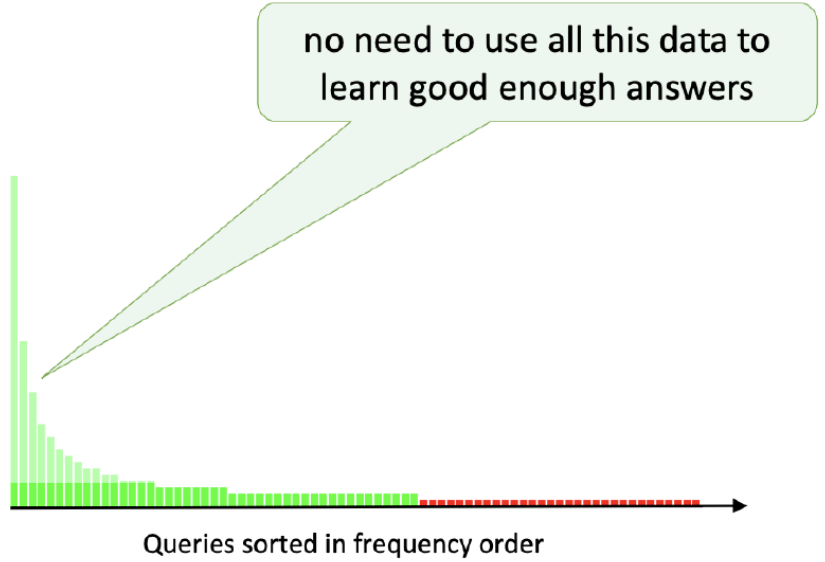
My data still does not fit in RAM !

Sampling

Maybe we can rely on a smaller version of the dataset

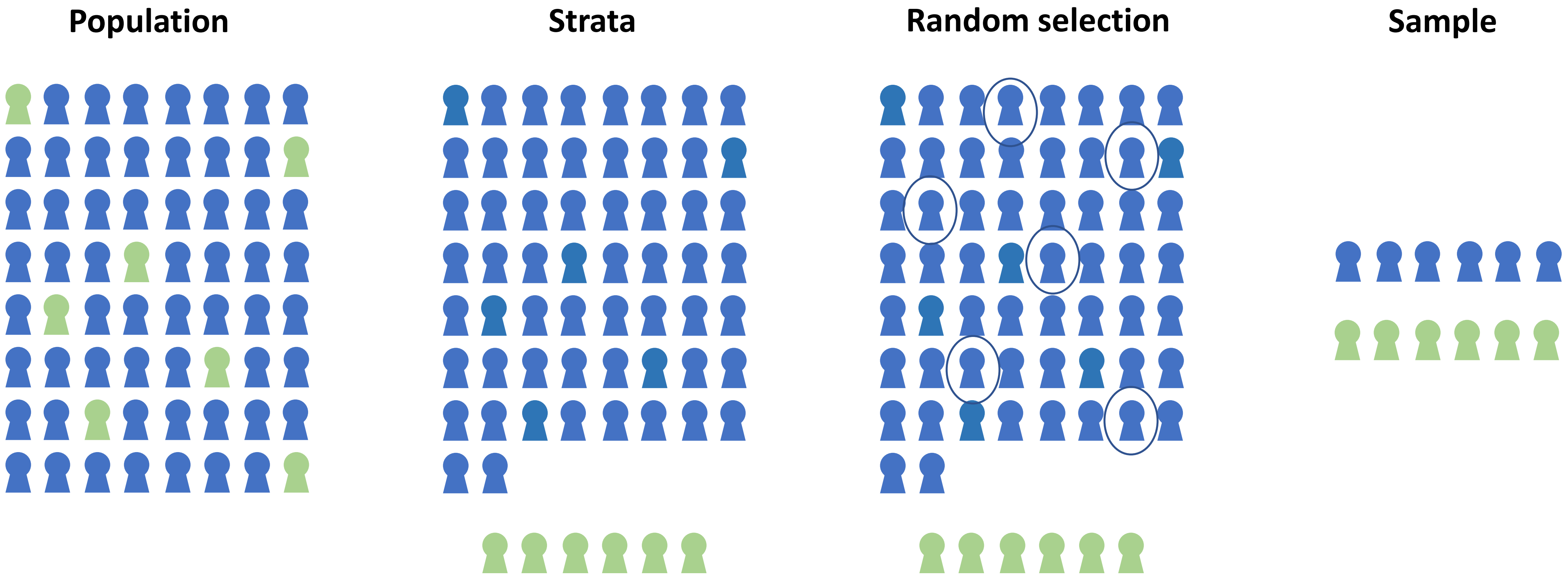
Uniform Sampling

Maybe we can do better



Sampling

Stratified Sampling



You can use a different sampling ratio for each group

Sampling

Grouping criterion ?

Each individual of the group should look alike !

Should I go for clustering technique ?

Maybe, but it may be costful. In practice (advertising, fraud detection, cheminformatics) we can do simpler:

- Use the Label
- Use a given column of the dataset
- Sometimes the criterion may be dictated by your infra

But ideally make sure that you:

- don’t remove individuals from the ‘long tail’
- don’t degrade your loss

Sampling

Don't forget to correct your loss

Sampling Ratio

Group	Sampling Ratio
A	1 (unchanged)
B	1/10
C	1/30

Loss

Loss = Loss(X_a, y_a) + Loss(X_b, y_b) + Loss(X_c, y_c)

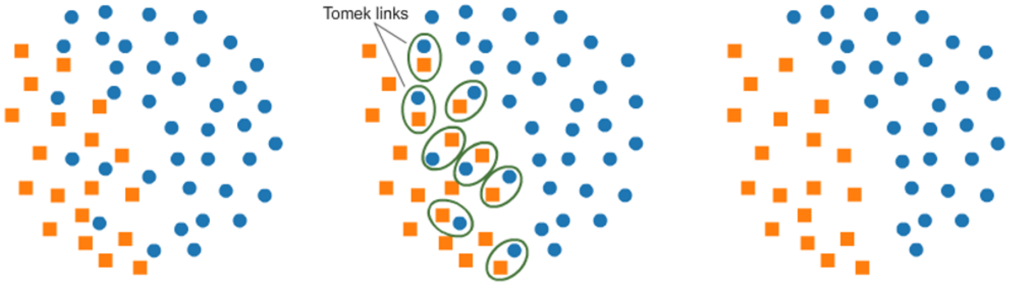
*Loss = 1 * Loss(X_sampled_a, y_sampled_a)*
*+ 10 * Loss(X_sampled_b, y_sampled_b)*
*+ 30 * Loss(X_sampled_c, y_sampled_c)*

Sampling

Emptying the drawers

Condensend nearest neighbours : stream through the examples, keep example if it cannot be correctly classified by the content of the added examples so far (using nearest neighbour methods)

Tomek links : remove element from pair of closest elements, with different label (remove element from majority class)



Scaling ???

My data still does not fit in RAM !

Out of Core algorithms

No need to fit whole dataset into RAM to build your model

Dataset $\{(x_i, y_i)_{i=1..N}\}$ grouped into batches $\{(X_b, Y_b)_{b=1..B}\}$

Differentiable predictor $f_w(X)$

Cost function $J(f(X), Y)$

Repeat until some stopping criterion is met:

$w \coloneqq w - \alpha \nabla_w J(X_b, Y_b)$

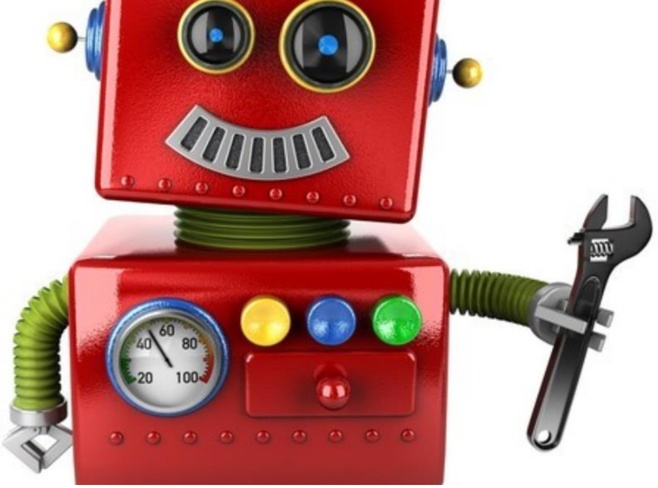
update α

Out of Core algorithms

Batch Gradient Descent

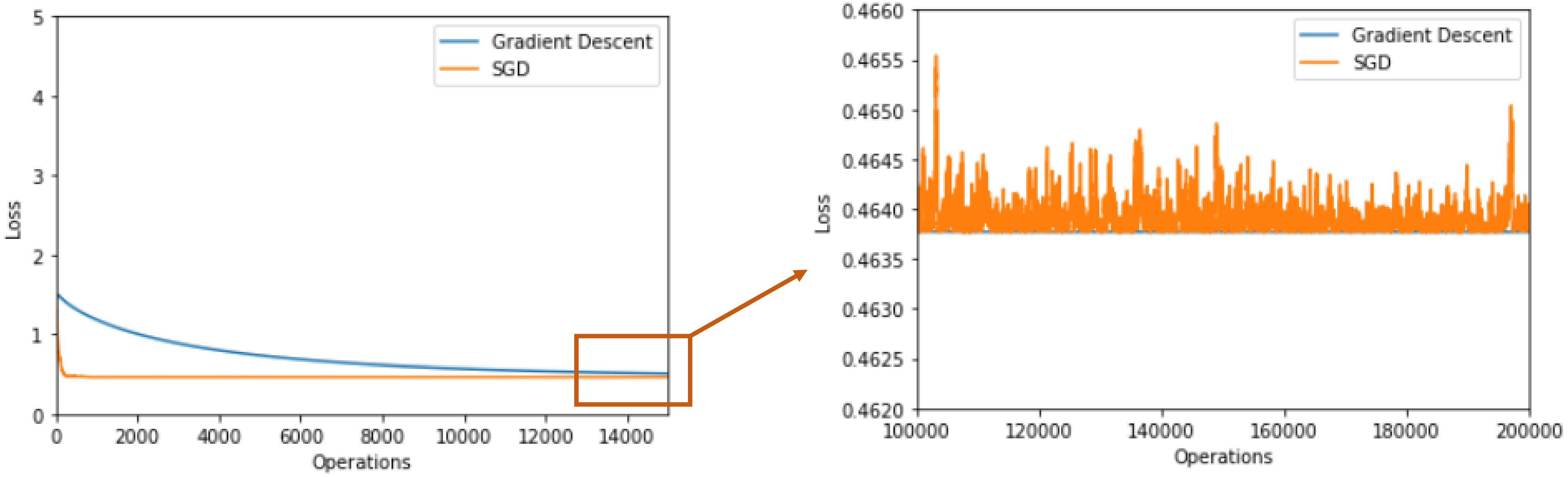
Out of Core algorithms

Toy example



Simple linear regression example:

```
x = np.array(range(1000)) / 100
y = 6 * x + 3 + np.random.normal(0, 16, x.shape)
```



Compromise memory/speed/optimization error

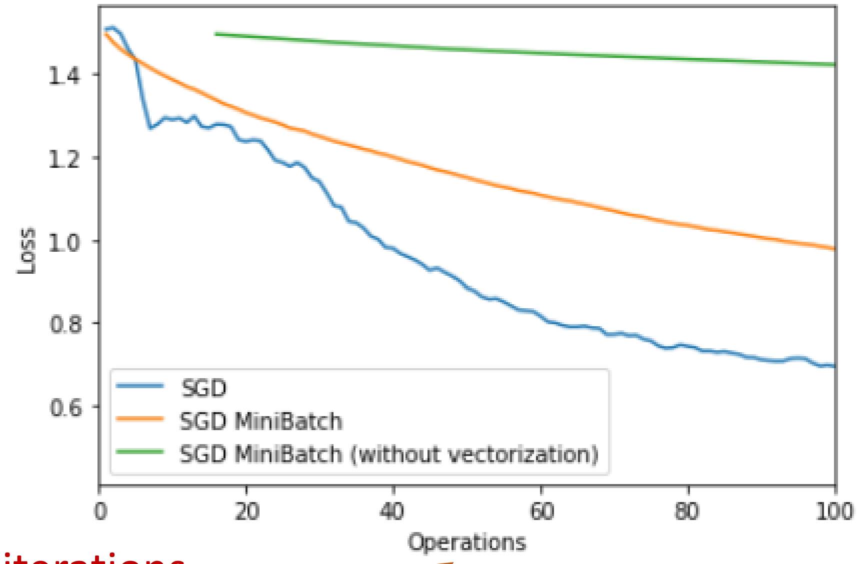
Out of Core algorithms

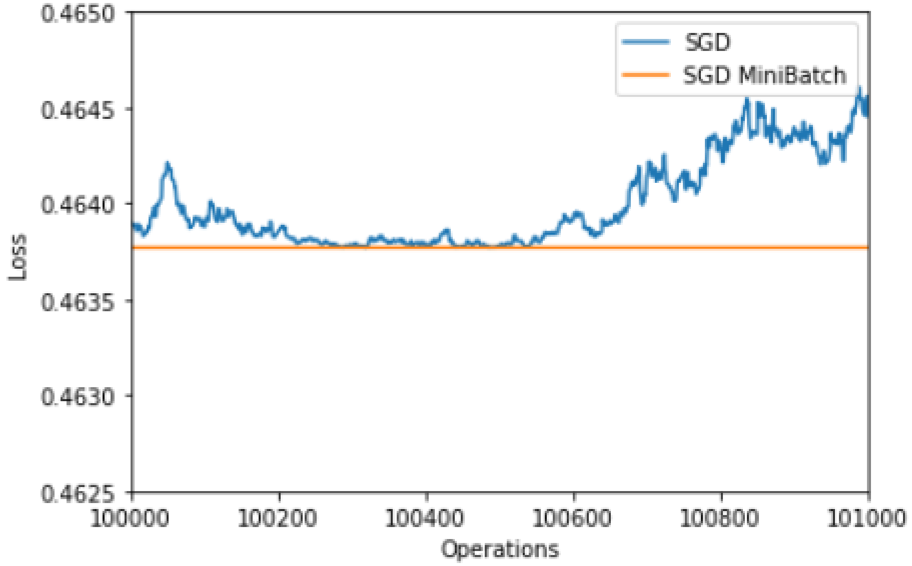
Mini batch, what's the deal ?

Smoother convergence ?

Vectorization (SIMD, GPU) speed-up ?

Faster convergence (smoother gradient) ?





Operations, not iterations
(lines processed + vectorization)

Out of Core algorithms

Scikitlearn

All algorithms with a partial fit method

sklearn.linear_model.SGDClassifier

sklearn.linear_model.SGDRegressor

…

SGD without mini batch

```
model = linear_model.SGDRegressor()
for (X, y) in chunks:
    model.partial_fit(X, y)
```

Learning rate updated at each row

By the way, don't forget to normalize your data

Are we good to go ?

Shuffling

Why should I care about shuffling ?

Sorted by height →

People Height	Shoe Size
130cm	35
130cm	36
131cm	35
…	

Training set

…

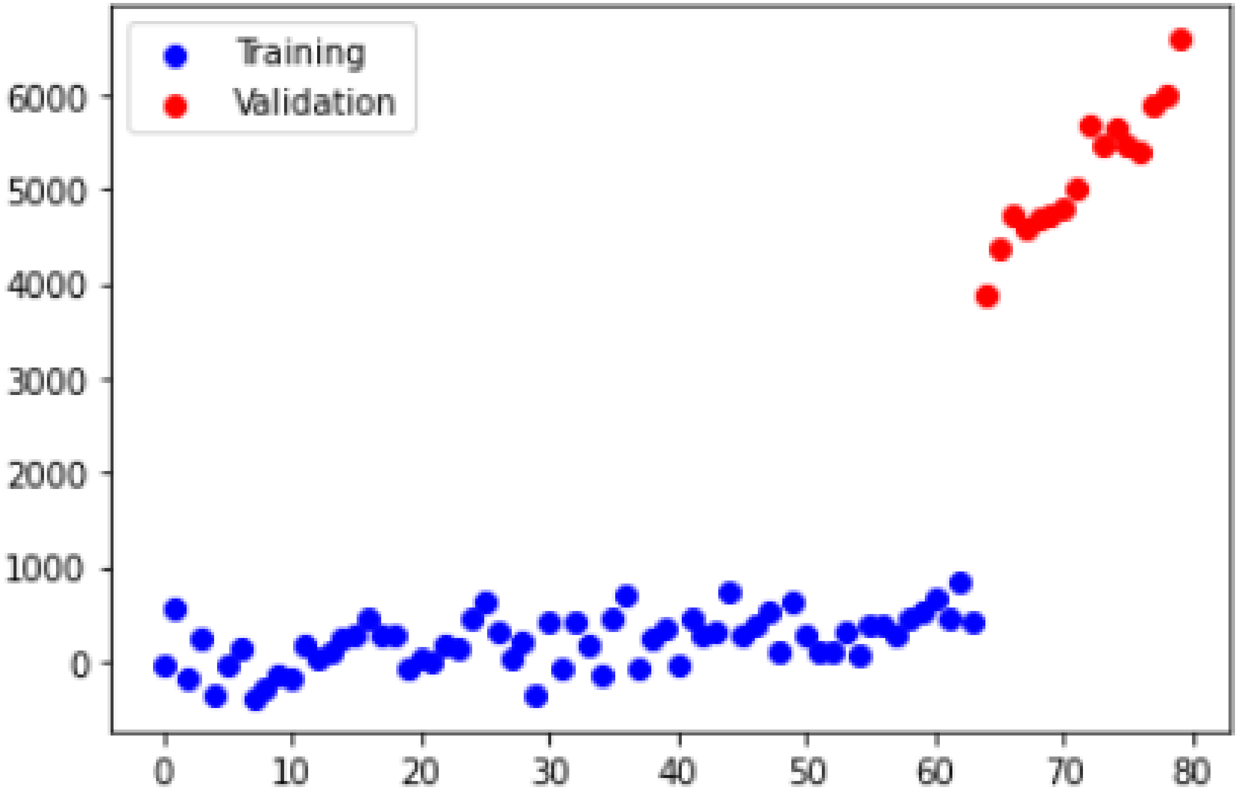
…	
210cm	48

Validation set

What happens ?

Shuffling

Why should I care about shuffling ?



Shuffling

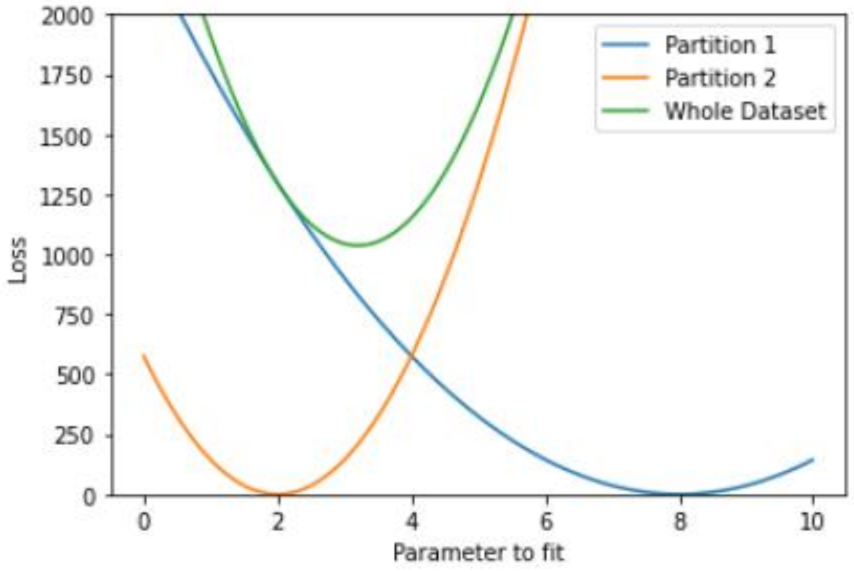
Batches should be representative of dataset

Otherwise, risk of overfitting:

- on last batches of the training

In theory, you need to shuffle the dataset after each epoch.

In non-convex optimization, helps to hop from one local optimum to another.



Shuffling

How do you shuffle a file that does not fit in RAM ?

One solution

One or multiple files

..
Row 1
Row 42
…
Row 123
…
Row n

Add random column

O(n)

Multiple files

random	…
261136629	Row 1
2	Row 42
..	
988215456	Row 123
535215832	Row n

External sort

O(n log(n))

One or multiple files

random	…
2	Row 42
…	…
261136629	Row 1
..	…
535215832	Row n
…	…
988215456	Row 123

Pick numbers from a large set so that probability that any two lines have same random number is small.

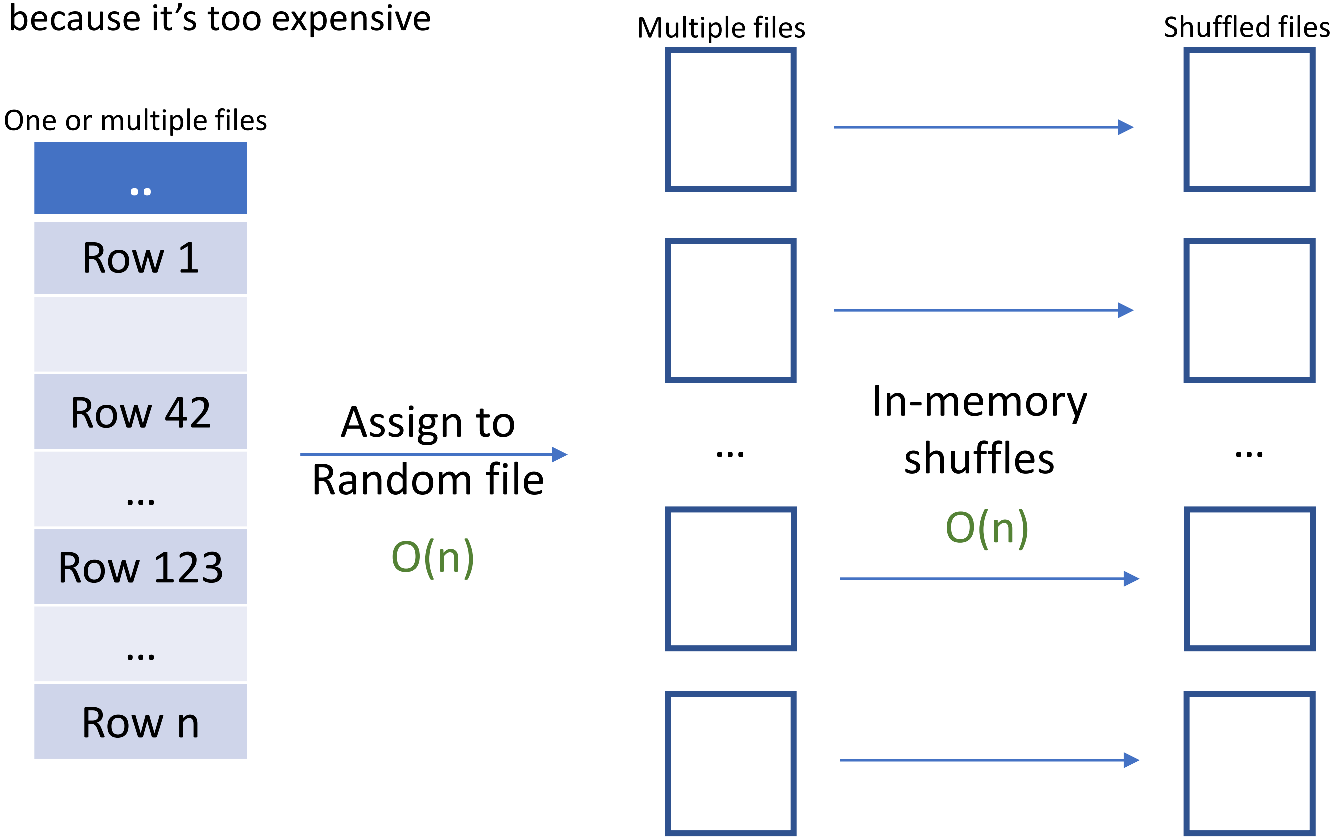
Some versions of the UNIX utility sort already have this:

cat -n myfile.csv | sort --random-sort | cut -f 2

Shuffling

How do you shuffle a file that does not fit in RAM ?

Better solution : don’t sort because it’s too expensive



Pick amount of files so that each file can be processed in-memory

Need to know big file size

Unix : no simple one-liner for the first operation !

Conclusion

Model Performance limited by **training cost/time**, not number of samples.

More data will increase **model breadth.** What is your *task* ?

Some task require manual data annotation that will not scale. Sometimes some **cheap ways to generate labels** exist.

We need to **decrease training costs/time**:

- Use **sampling**
- Replace polynomial algorithms by **linear algorithms**
- **Out of core algorithms** (some kind of divide and conquer strategy)
- Optimize **RAM usage** (avoids I/O)
- Parallelize computations, use dedicated hardware (not covered)