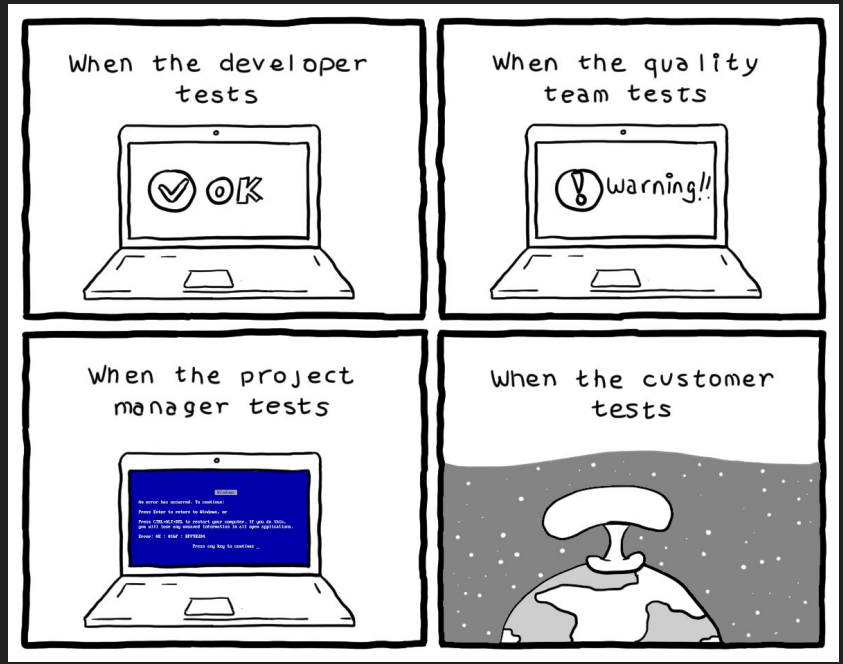
End-End Deep Learning in Practice: Development and Deployment

Mustafa Alghali June 7 2020

Cocktail of Tips and Tricks

Machine Learning is a unique deployment problem

- The most common way machine learning gets deployed today is on powerpoint slides.
- Less than 5 percent of commercial data science projects make it to production.
- Hard to think of all constraints together!



Source turnoff.us

Why will it fail ?

- **Neural net training is a leaky abstraction**

```
>>> your_data = # plug your awesome dataset here
>>> model = SuperCrossValidator(SuperDuper.fit, your_data, ResNet50, SGDOptimizer)
# conquer world here
```

  Looks like a courageous developer has taken the burden of understanding the underlying complexity for us!

- **Neural net training fails silently**

  Everything could be correct syntactically, but the whole thing isn't arranged properly, and it's really hard to tell or unit test because. Because The "possible error surface" is much larger and logical (as opposed to syntactic)→ attention to details

Debugging Neural Network

- Start from simple to complex and at every step of the way we make concrete hypotheses about what will happen

- Avoid introducing of a lot of “unverified” complexity at once which bound to introduce bugs/misconfigurations that will take forever to find (if ever).

Data Pipeline

Debugging Neural Network: Data first

- Inspect the data, get intuition about the complexity of the task you're trying to achieve.
- Give it time, understand distributions, observe patterns and notice their statistical properties.
- Clean the data! Dirty data can drive you crazy! (outliers, corrupted data, duplicate samples).
- Over/under-sample unbalanced data.
- Features scale ? downsampling ? variations ? features spatio/temporal properties?

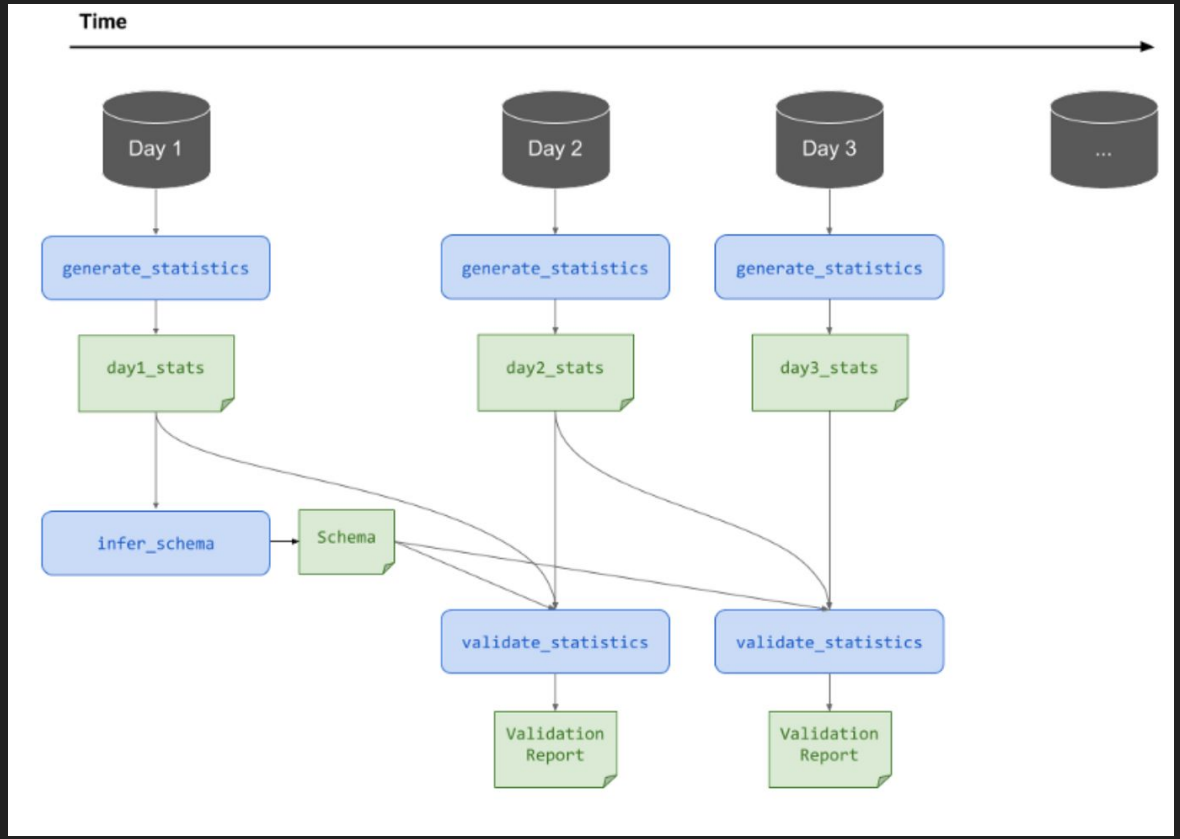
Debugging Neural Network: Data first

- Feature scaling vs normalization (the two optimization problems to be addressed)
  - Optimization landscape
  - Non zero centered activations

- To Automate the data validation process, you can use Data Validation schema that detects anomalies in training and serving time.
  - Define a data validation schema like (ranges, data types, statistical characters and columns ..etc)
  - Save the schema and use it for further validation during the test time.
  - E.g libs Tensorflow Data Validation (TFDV)

Debugging Neural Network: Data first

TFDV schema validation



Development Pipeline

Set up the end-to-end training/evaluation skeleton + get dumb baselines

- At this stage it is best to pick some simple model that you couldn't possibly have screwed up somehow, a linear classifier, or a very tiny ConvNet. We'll want to train it, visualize the losses, any other metrics (e.g. accuracy), model predictions.

- Avoid random initialization variations (fix a random seed)

- verify loss the initial losses e.g. -(1/n-classes) for classification problems

- Initialize well - incorporate priors such as unbalanced dataset 1:10 set init weights to predict 0.1 to speed up the convergence

Set up the end-to-end training/evaluation skeleton + get dumb baselines

- overfit one batch and check your model variance through the training error.
- Visualize your tensors before after preprocessing in the feedforward pass!
- Be careful with broadcasts, views, transpose, that mix information across the batch dimension!
- Generalize your code later! E.g. vectorizing an implementation.
- Relu is safe choice! Converge 6x times faster than sigmoid/tanh

Set up the end-to-end training/evaluation skeleton + get dumb baselines

- Initialization go for kaiming (better with Relu) or Xavier, and Fixup initialization for Resnet blocks.

Overfit!

- Don’t be a hero ! avoid exotic architectures in the beginning.
- Adam is safe: more forgiving to a wider range of hyperparameters
- Complexify only one at once: e.g start with small images and then use big ones.
- Do not trust learning rate decay defaults: your code could secretly be driving your learning rate to zero too early.

Regularize!

- Get more data: but a preference to get to more data than further tuning on small dataset.
- Data Augmentation: half fake, or fake (like GANs or Domain Randomization)
- Transfer learning
- Stick with supervised learning: , no version unsupervised Learning has reported strong results in modern computer vision, except for NLP due to higher to signal ratio.
- Smaller input dimensionality: spurious dimensions are another opportunity to overfit

Regularize!

- Smaller model size: e.g. Global average pooling layer vs fully connected layer.
- smaller batch size: -> noisy gradients + batch norm layer statistics
- Dropout!
- weight decay
- Visualize learned features

Dropout vs Data Augmentation

- Blindly reducing a networks capacity.
- Introduce model sensitive hyper-parameters

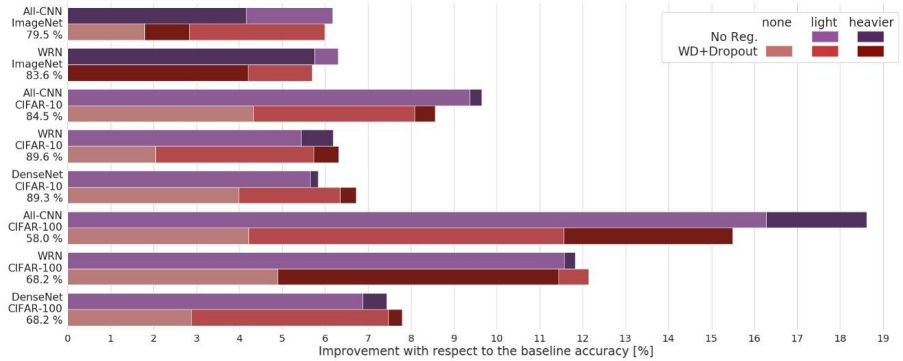


Figure 1: Relative improvement of adding data augmentation and explicit regularization to the baseline models, $(accuracy - baseline)/accuracy * 100$. The baseline accuracy is shown on the left. The results suggest that data augmentation alone (purple bars) can achieve even better performance than the models trained with both weight decay and dropout (red bars).

Dropout vs BatchNorm

- Dropout + BN → often lead to a worse performance
  - Dropout would shift the variance from train to test
  - BN would maintain it
  - This causes the unstable numerical behavior in inference "variance shift"

- More in [Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift]

Imbalanced datasets

- Resampling
    - Undersampling if sufficient quantity else Oversampling
    - Weighted sampler

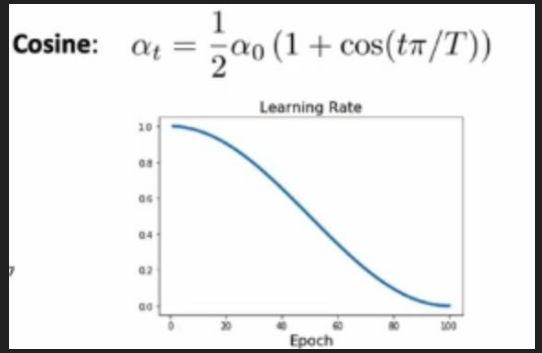
- K-fold cross validation
    - Should be done before Oversampling -> avoid overfit artificial distribution

- Penalized models
    - additional cost on mistakes on the minority class to bias the model to pay more attention to it e.g. “class weights” in Keras

- Watch for BatchNorm: boost the bias into the majority class

Hyperparameters

- Learning rate: start big and then decay,
  - Use schedulers → cosine has less additional hyperparameters compared to step

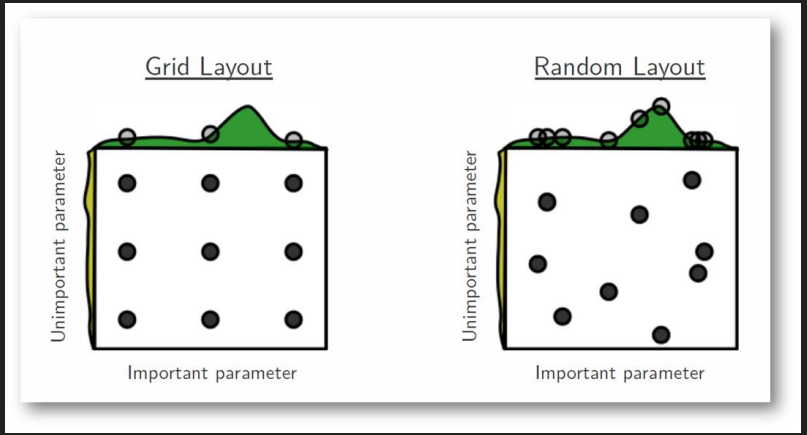


- For gigantic models you might need to warm up!

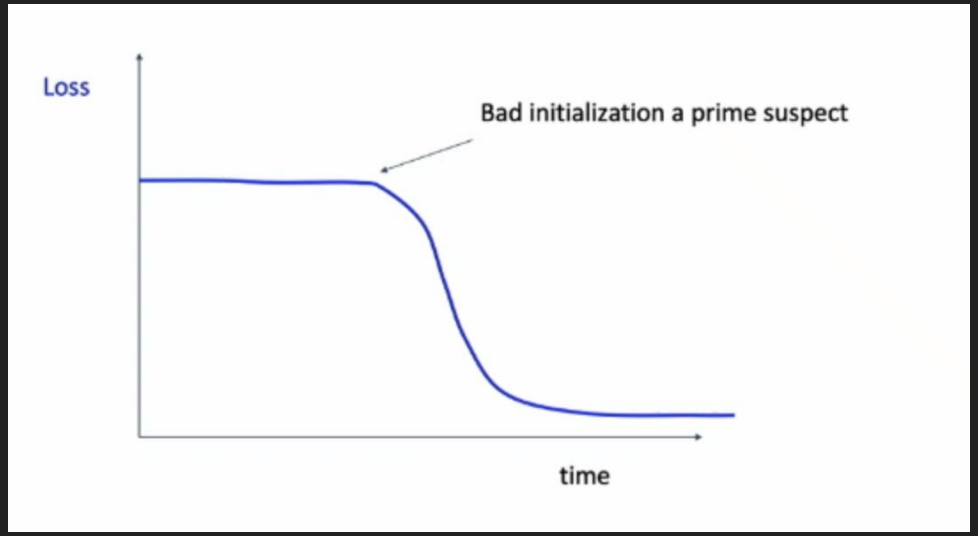
Hyperparameters

- Resist the idea to do manual search
- Invest time to design hyperparameter search pipeline

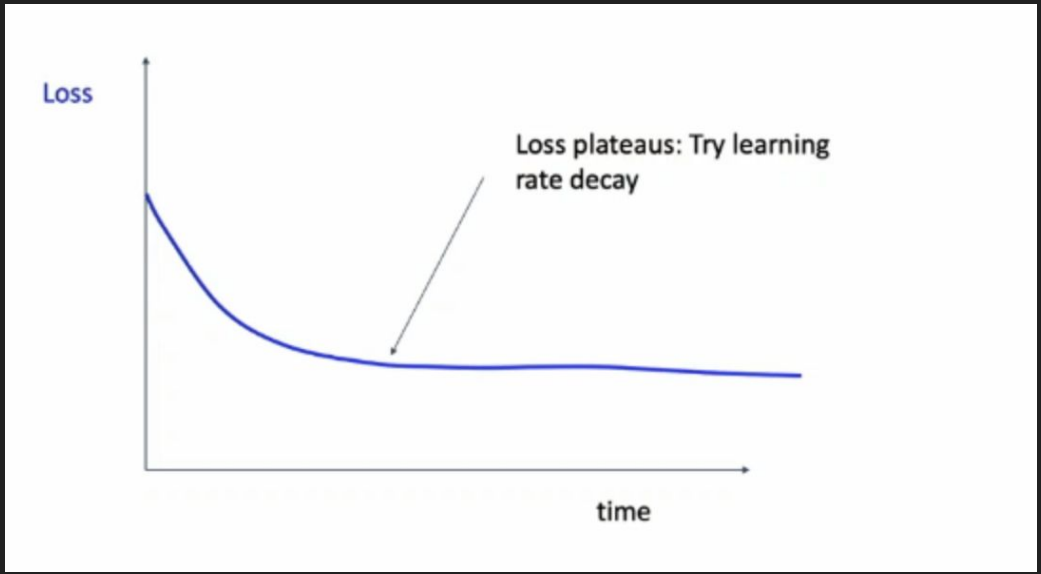
- Grid vs Random search



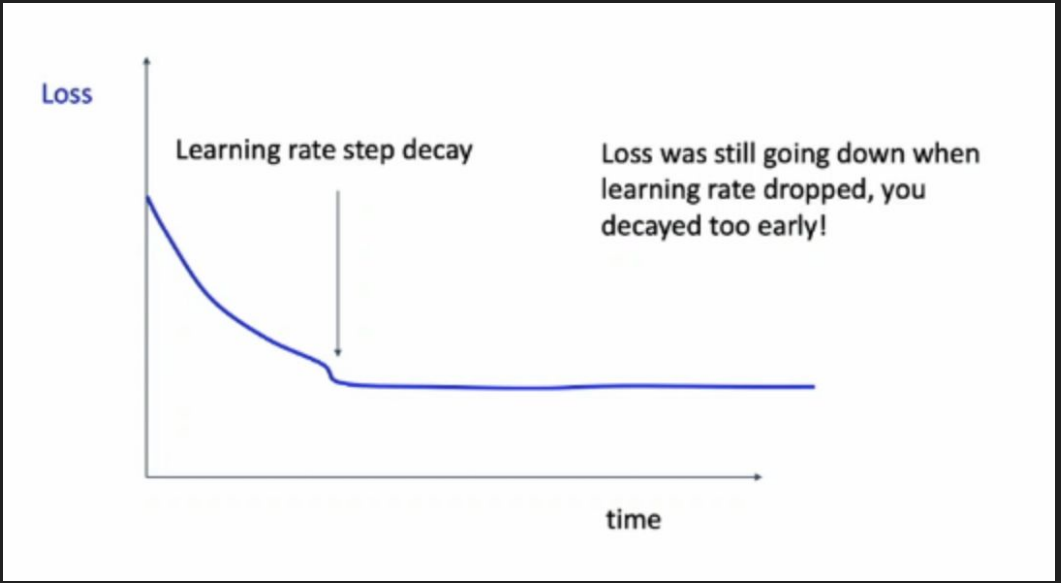
Look at learning curves



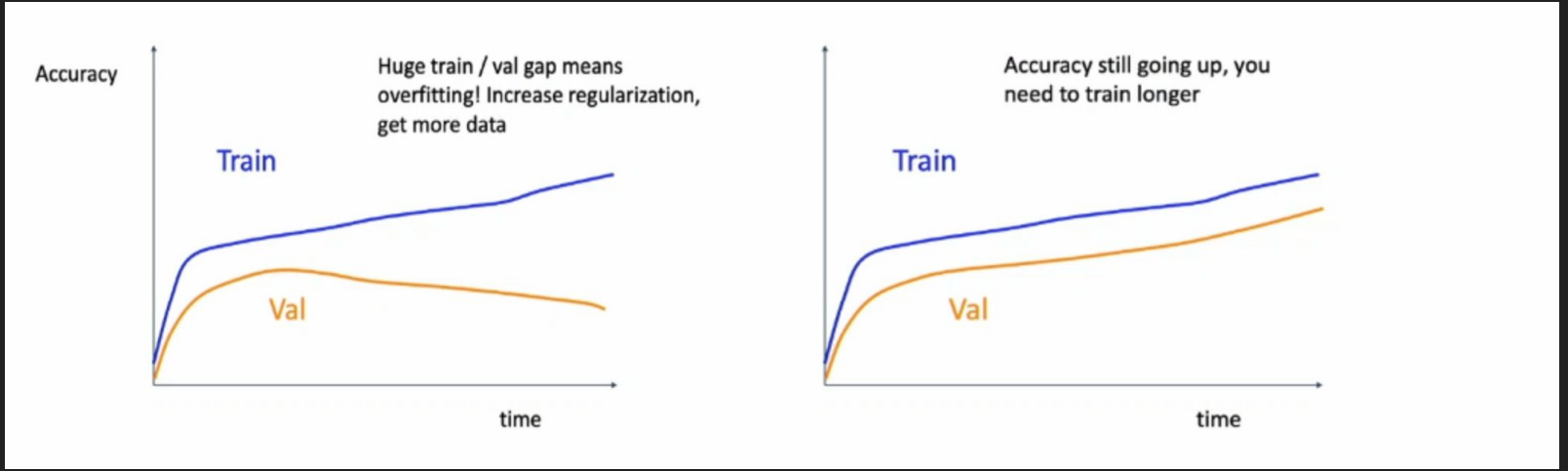
Look at learning curves



Look at learning curves

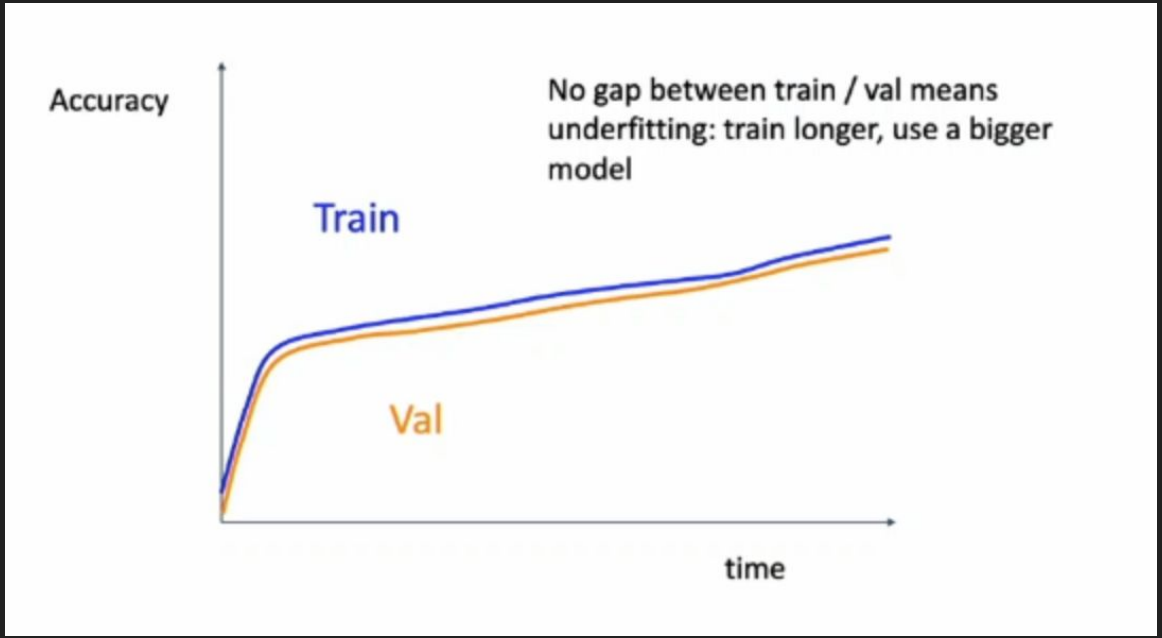


Look at learning curves



Look at learning curves

Tricky one!



Look at learning curves

- Experiments visualization center
  - Tensorboard
  - **Weight and biases** (wandb)
  - Comet.ml

What you can do more ?

- Model ensembles

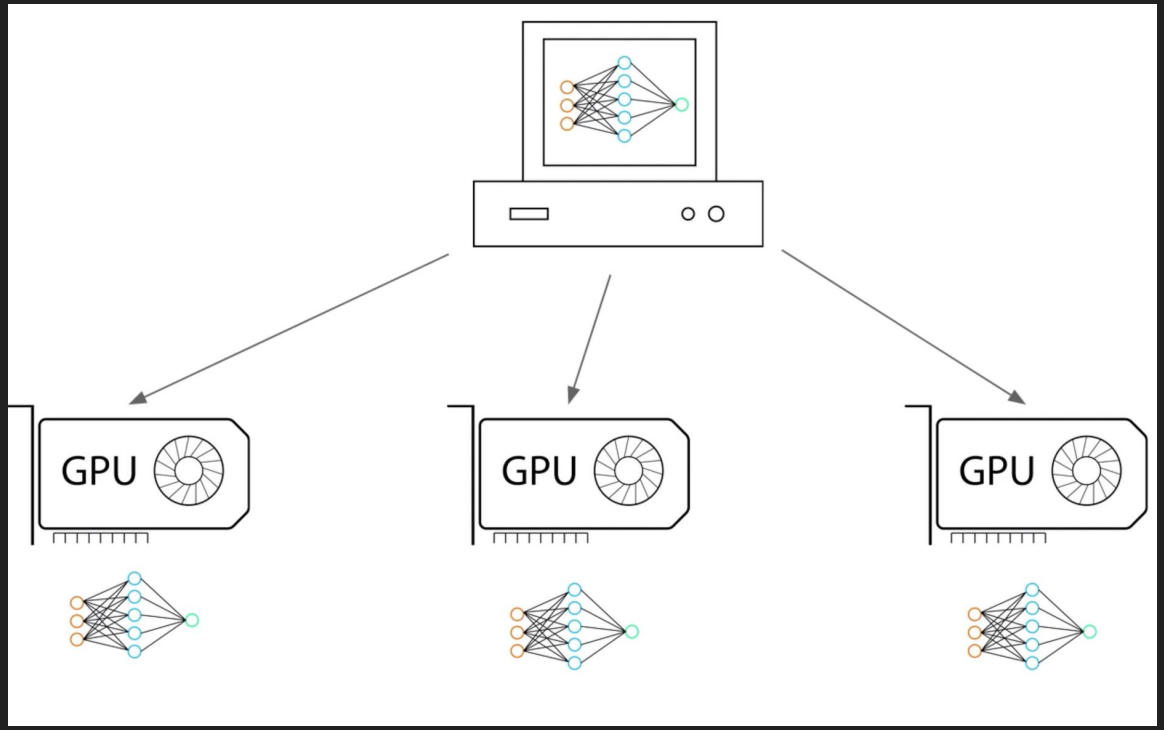
- Transfer learning and fine-tuning when ?
  - Large and different dataset → train model from scratch (but also recommended to fine tune)
  - Large and similar dataset → fine tune the model
  - **Small and different dataset** → fine tune fewer layers/ linear classifier from different stages
  - Small and similar dataset → transfer learning

Distributed Computing

1) Model parallelism is Inefficient
    - Split model into parts (sequentially) across GPUs → waiting
    - Split model into branches (parallel) → lost of syncs

Distributed Computing

2) Data parallelism (Mirrored strategy)



Distributed Computing

Data parallelism (Mirrored strategy) more efficient

- Minibatch
- Replicate model
- Could be as simple as one line of code e.g tf.distribute.MirroredStrategy()
- Communicate at the end of a forward pass (exchange gradients , sum -> updates)
- You're basically using larger batch! Then keep in mind **learning rate** will also be affected (they are correlated)

- Batch size K*n → learning rate K*r
- Big learning rate may lead divergence at the beginning → use warmup (start with small learning rate and then increase it)
- Check how gradients are combined in Pytorch and Tensorflow (added vs averaged)
- Compare models by number of epochs not iterations

State of the Art

- Make sure you aren't betting on irreproducible results
  - Eye-catching advances in some AI fields are not real

Machine Learning at Scale

Machine Learning at scale: DevOps

- Writing software and writing software for scale

- This has led to the dawn of an entirely new field called DevOps to create scalable, resilient, and distributed systems, resulted in the growth of new paradigms like microservices, and contaraization.

- According to Algorithmia, a majority of data scientists report spending over 25% of their time on model deployment alone

- Unless you're working with tech giants (e.g Uber Michelangelo) you probably will be a victim of these operations.

Machine Learning at scale: DevOps

- Trending paradigms:
  - Event-driven, Serverless computing
  - Containerization, share images among development groups, reproducibility, e.g dockers and singularity containers.
  - Microservices: individual hybrid components interact through API calls.

Machine Learning at scale: Big Data

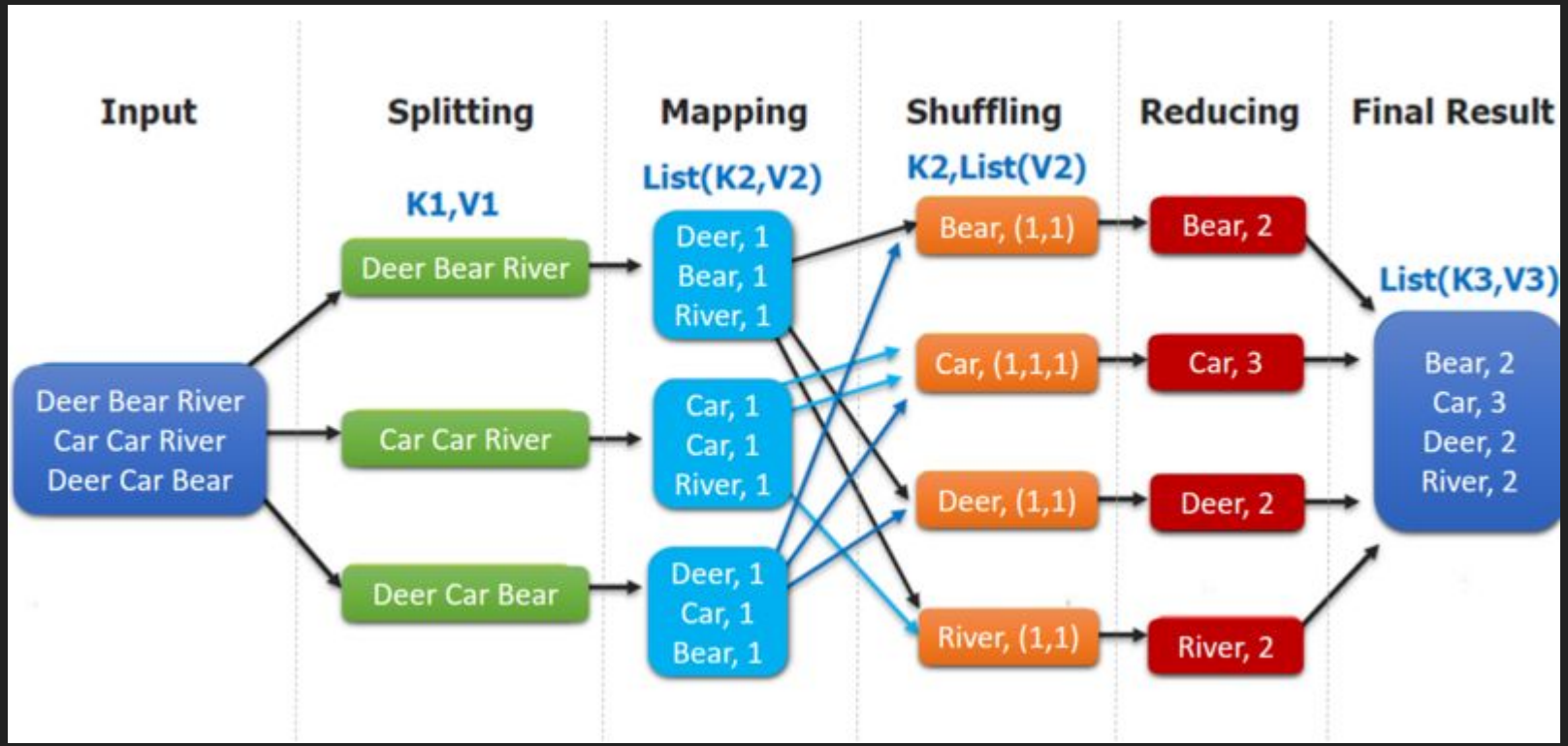
- What if the data can't fit into your ram (e.g > 100 GB):
  - Try to move the data into hard drive
  - Use distributed system with multiple machines

Machine Learning at scale: Big Data: Hadoop

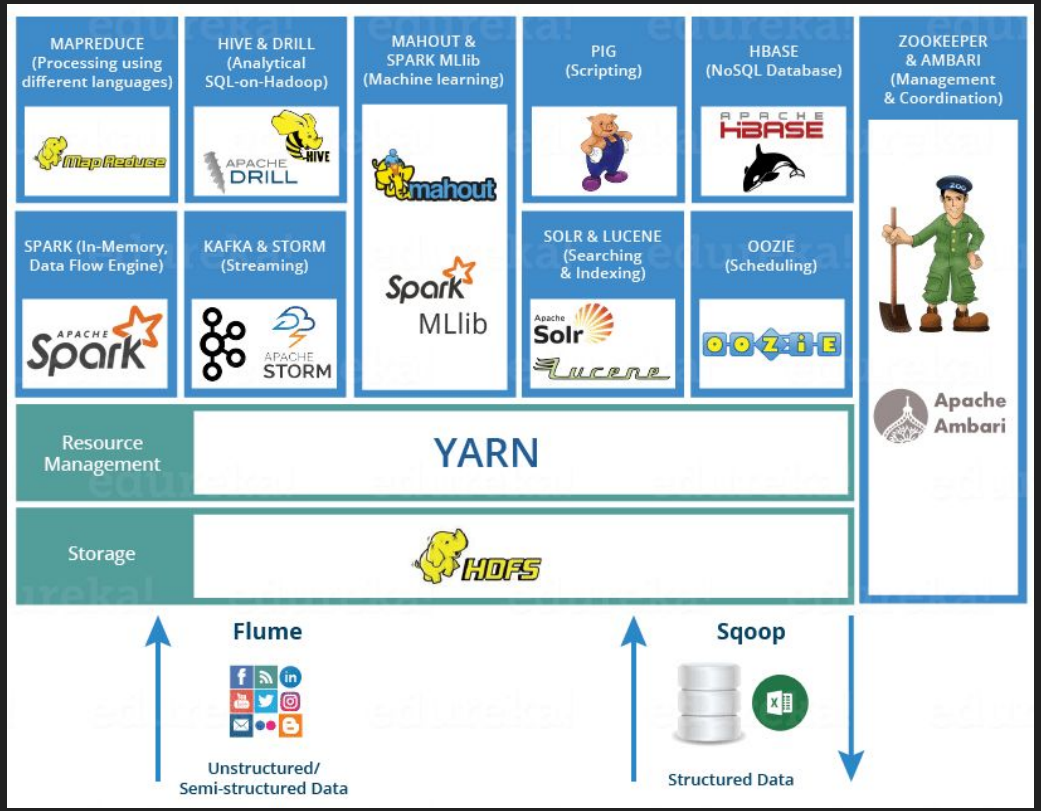
- Distribute very large files across multiple machines.
    - Uses Hadoop Distributed File Systems (HDFS)
    - Split and Duplicate the Data as block in nodes for fault tolerance
    - uses Map-Reduce for batch processing - distribute computational tasks!

Machine Learning at scale: Big Data: Hadoop

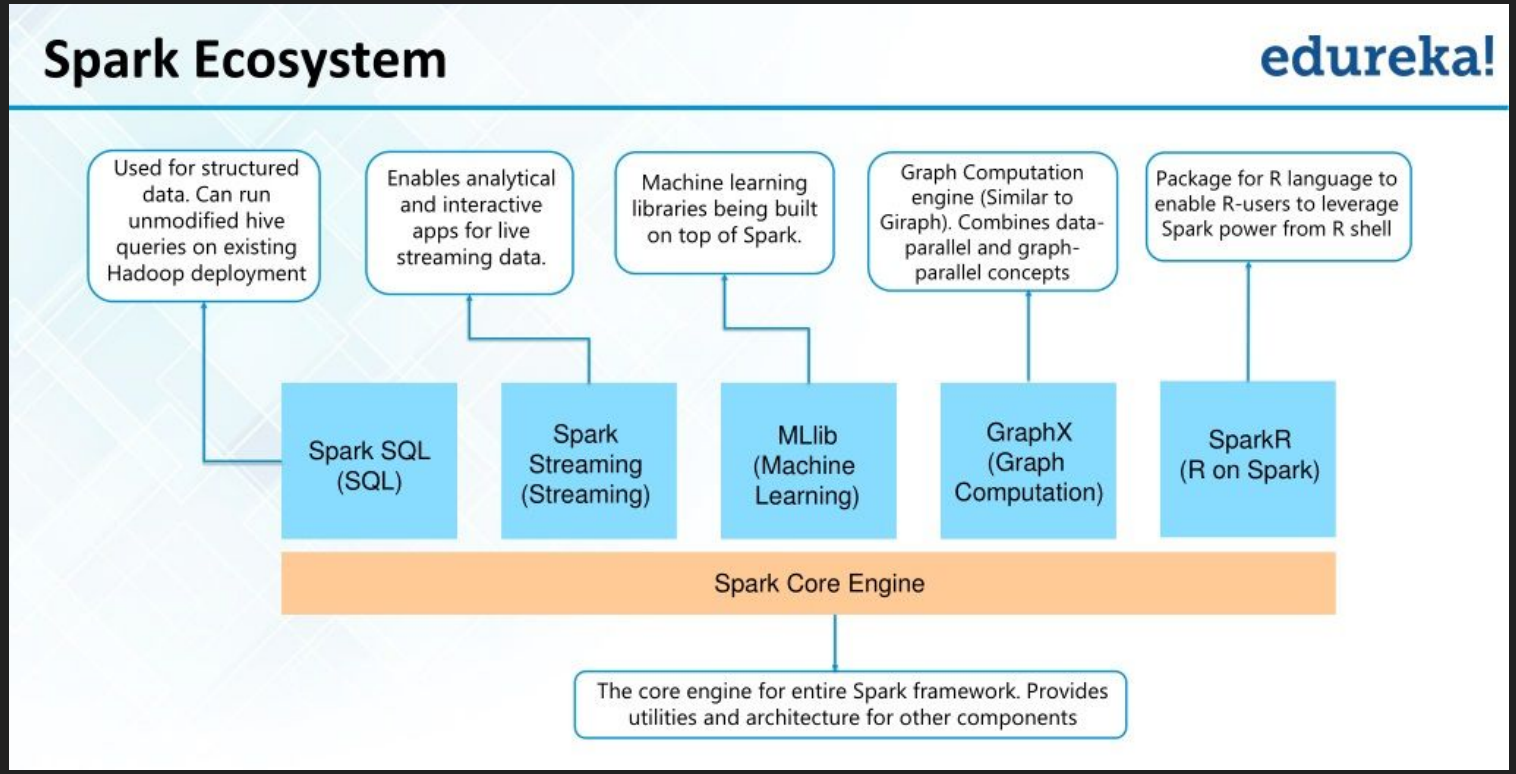
MapReduce



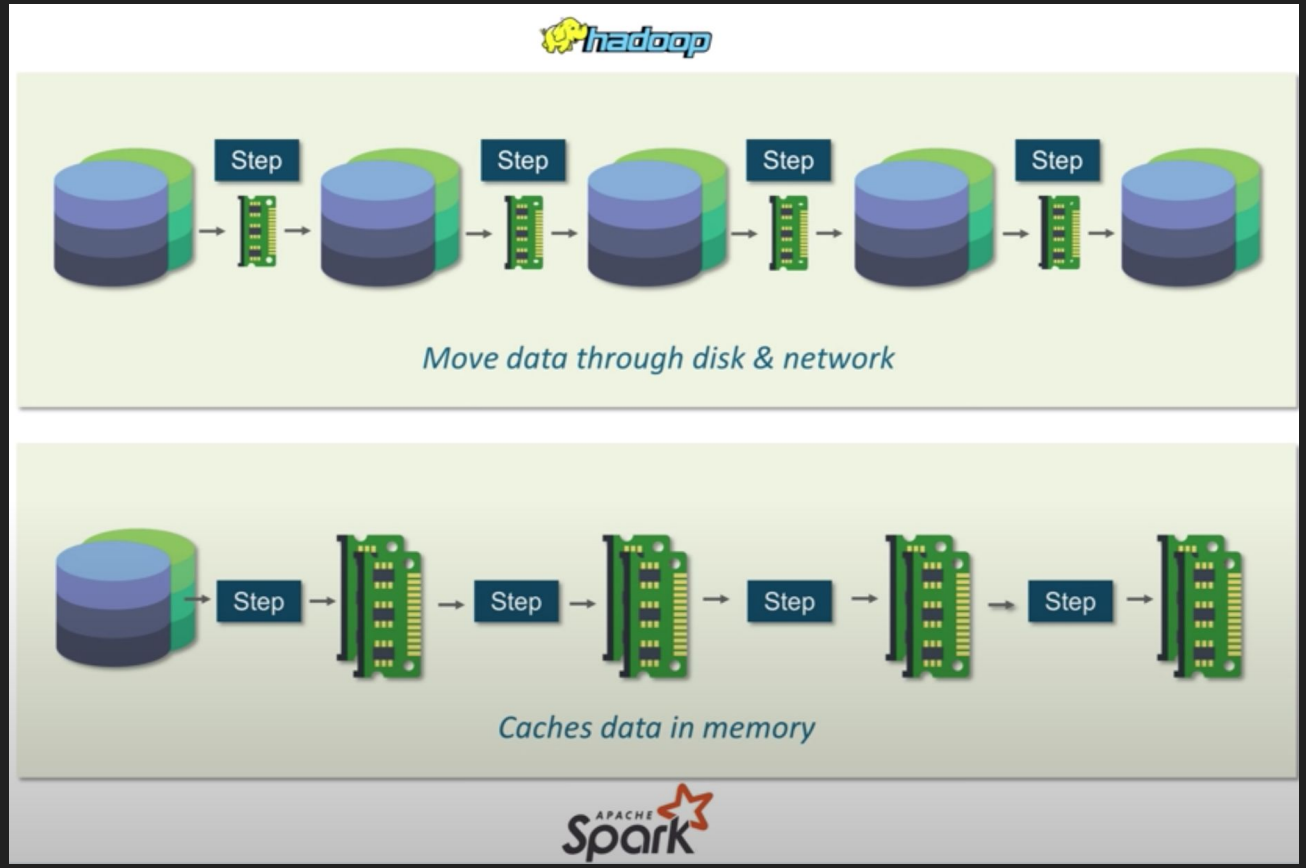
Machine Learning at scale: Big Data: Hadoop



Machine Learning at scale: Big Data: Spark



Machine Learning at scale: Big Data: Spark vs Hadoop



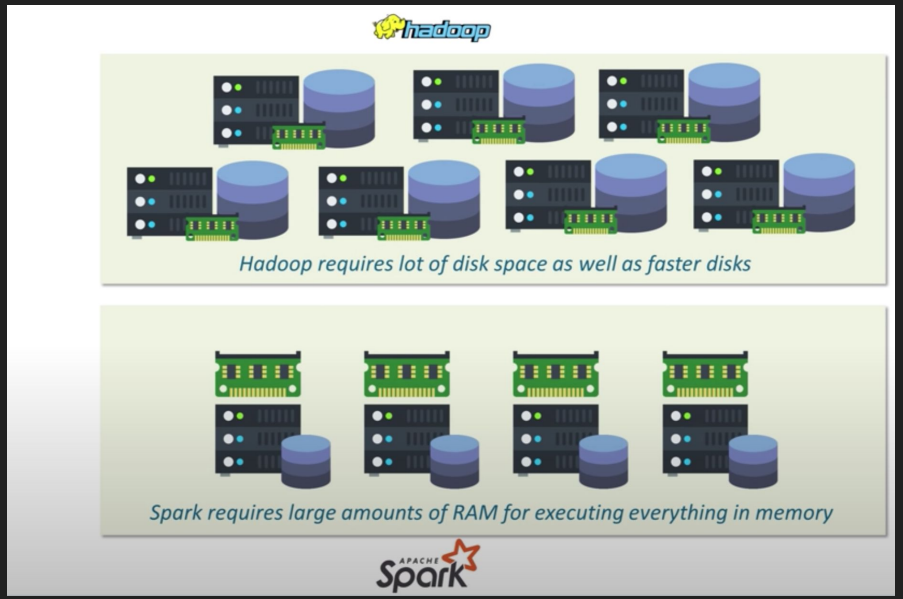
Machine Learning at scale: Big Data: Spark vs Hadoop

Performance

- Can process data stored in HDFS and other formats
- MapReduce requires files to be stored in HDFS, Spark doesn't
- MapReduce write most data to disk after map and reduce operation
- Spark keeps most of the data in memory after each transformation
- Spark can spill over to disk if the memory is filled.
- Spark therefore process the data up to 100 times faster than MapReduce.
- This makes Spark for applications like machine learning, security analysis...etc.
- MapReduce is built for parallel processing not real time processing, with Hadoop to aggregate the data, process it and store it in distributed environment.

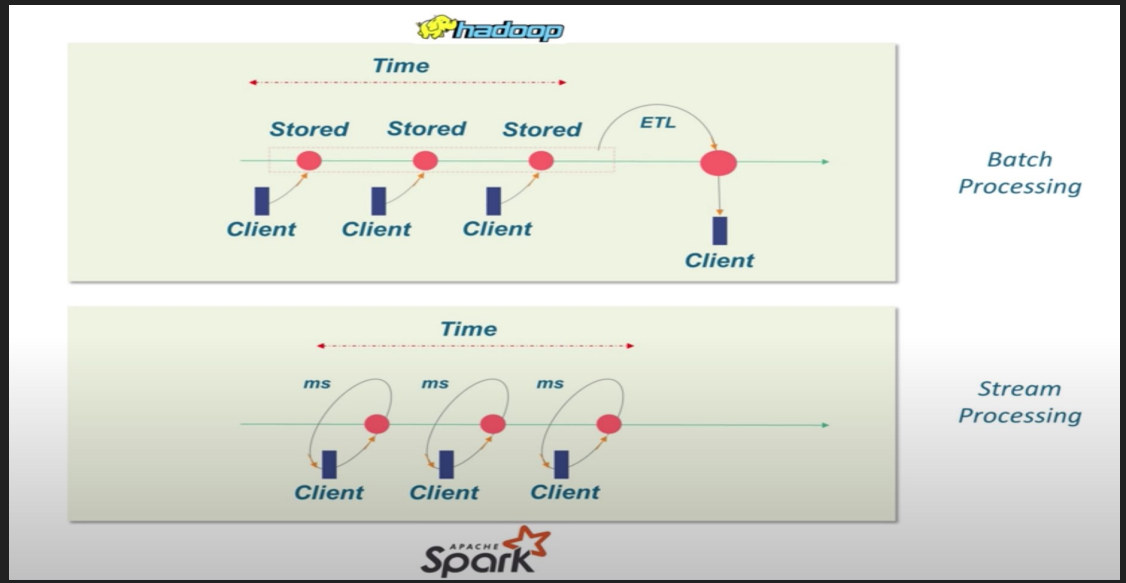
Machine Learning at scale: Big Data: Spark vs Hadoop

- Cost
  - Hadoop requires faster disks and a lot disk space
  - Spark requires large amounts of RAM!
  - But Spark requires significantly less number of instances over all.



Machine Learning at scale: Big Data: Spark vs Hadoop

- Data processing
  - Batch processing works with high volume static data collected over a period of time example use extracting insights of archived data
  - Stream processing react in real time example use, iterative algorithms (such as Machine Learning training), interactive queries, graph processing..

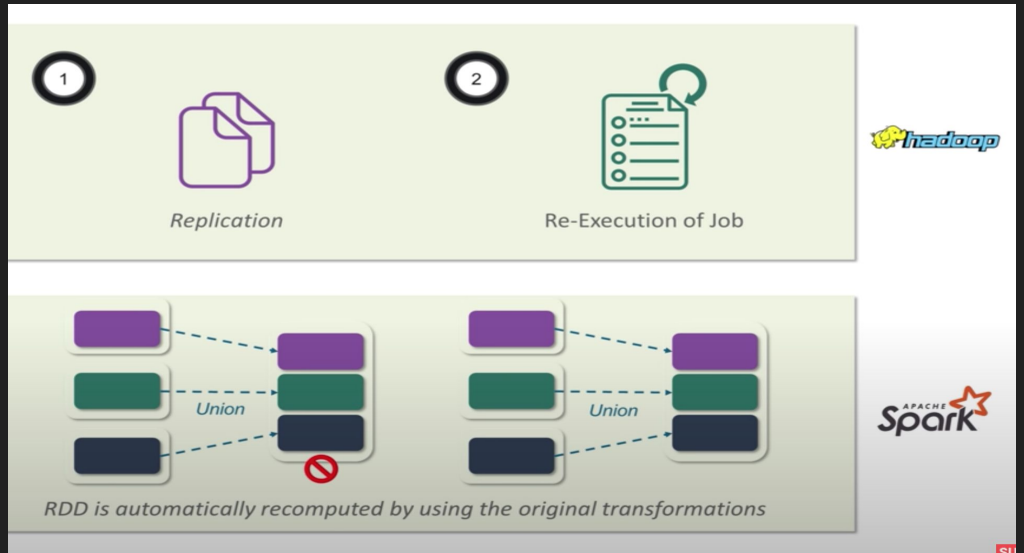


Machine Learning at scale: Big Data: Spark vs Hadoop

Fault tolerance:

Hadoop: If slave daemon fails the master will reschedule the operations to another slave

Spark: Uses Resilient Distributed Dataset (RDDs), immutable, lazily evaluated, cacheable, RDD can be re-computed by using the original transformation.



Machine Learning at scale: Apache Beam

- Apache Beam framework provides an abstraction between your application logic and the big data ecosystem.
- Unifying batch and streaming: Data source can be batches, micro-batches or streaming data
- Runner — Once the application logic is written then you may choose one of the available runners (Apache Spark, Apache Flink, Google Cloud Dataflow, Apache Apex, Apache Gear pump (incubating) or Apache Samza)

- Read more about the comparison between Spark vs Beam

References

- https://algorithmia.com/blog/deploying-machine-learning-at-scale
- http://karpathy.github.io/2019/04/25/recipe/
- https://arxiv.org/abs/1801.05134
- https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/
- https://arxiv.org/abs/1801.05134
- https://www.sciencemag.org/news/2020/05/eye-catching-advances-some-ai-fields-are-not-real
- https://towardsdatascience.com/hands-on-tensorflow-data-validation-61e552f123d7
- https://medium.com/analytics-vidhya/apache-beam-a-beginners-approach-4783dfc6fea
- http://leccap.engin.umich.edu/leccap/site/jhygcph151x25gjj1f0?fbclid=IwAR3wq10aKyvn0xJ9XZYWkdq6MUUO4p15VWX3gddwGJ3JzC4U7uwf63dV8i4