Saint Petersburg

3-4 November 2017

#NoTPU

When you’re simply not google enough

Anton Kiselev

Lead Data Scientist

Vladimir Shulyak

Data Science Engineer

We’ll cover

Application of Deep Learning to business tasks

Faster training with distributed Deep Learning

Why are we doing it?

Figure out ourselves as we have tasks like this @ work

Share the results with you


python

1. Modeling
    a. Motivation
    b. Classical approach
    c. Deep learning
    d. Practical example
2. Performance optimization

1. Modeling
    a. Motivation
    b. Classical approach
    c. Deep learning
    d. Practical example
2. Performance optimization

Why Deep Learning?

- Retail / e-commerce / banking / SaaS / telco / gaming

Why Deep Learning?

- Retail / e-commerce / banking / SaaS / telco / gaming
- No
  - Voice assistants
  - Robots
  - AI go-players

Why Deep Learning?

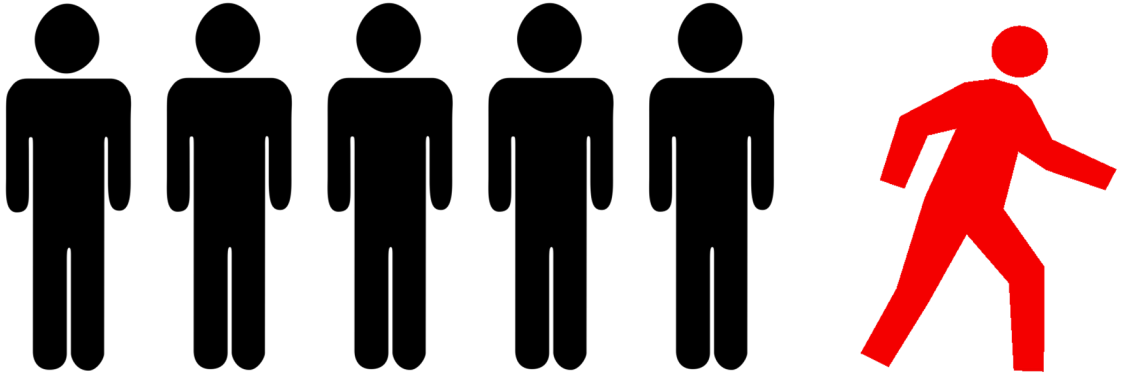
- Retail / e-commerce / banking / SaaS / telco / gaming
- No
  - Voice assistants
  - Robots
  - AI go-players
- Do I really need Deep Learning?

Why Deep Learning?

- Retail / e-commerce / banking / SaaS / telco / gaming
- No
  - Voice assistants
  - Robots
  - AI go-players
- Do I really need Deep Learning?

E.g. Churn Prediction

What is Churn Prediction?



Churn prediction = detect customers who are about to leave

Churn types

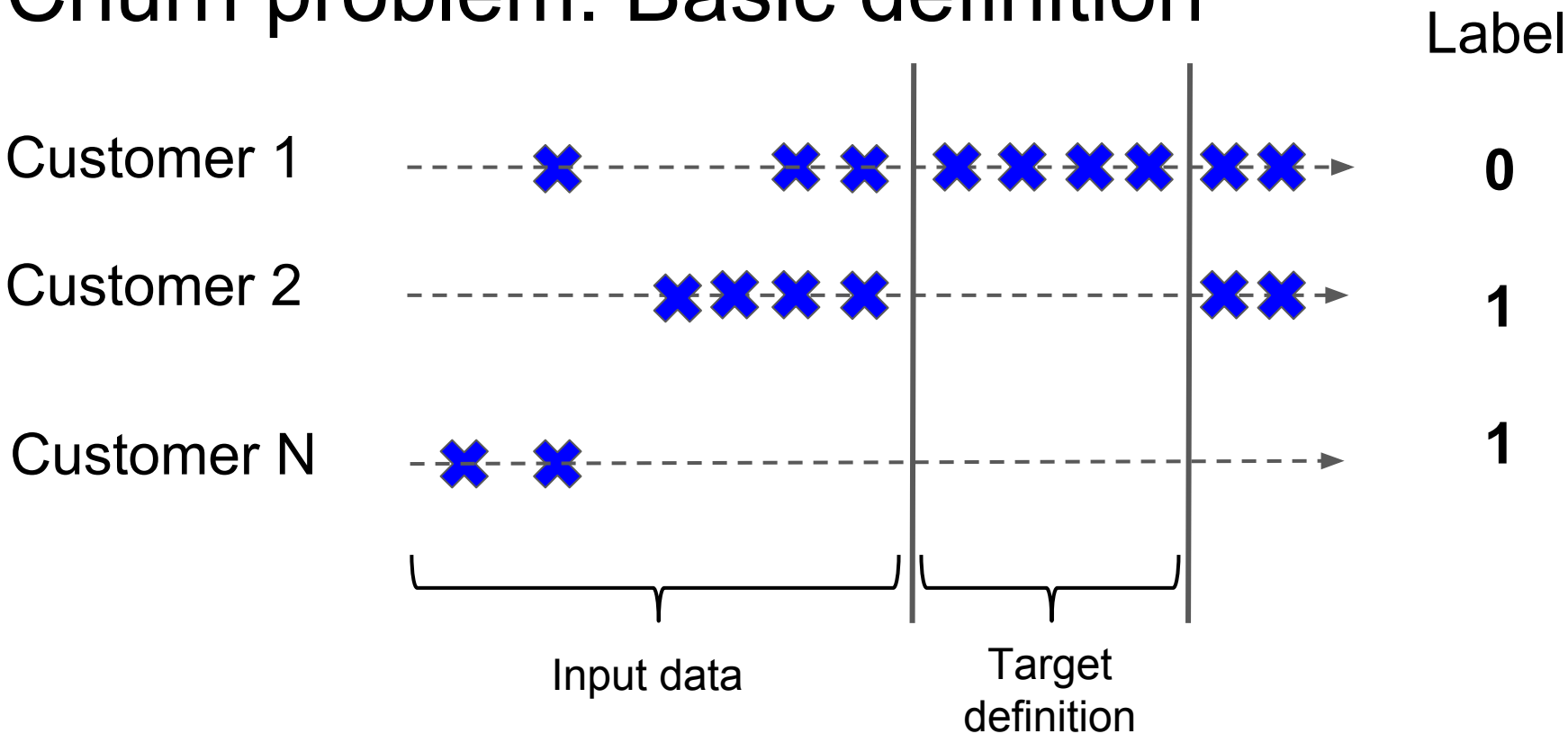
Contractual

Churn types

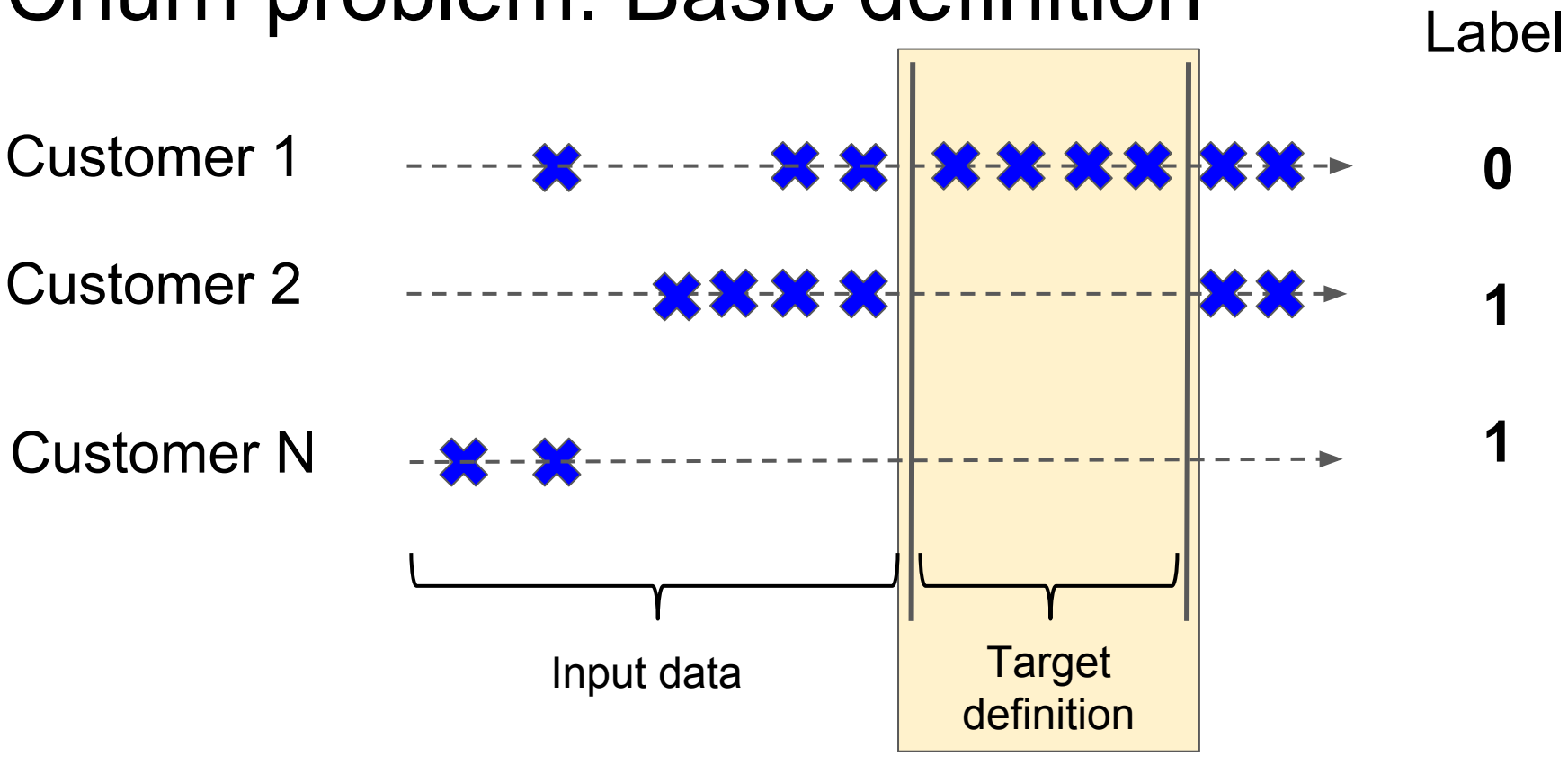
Contractual

Non-contractual

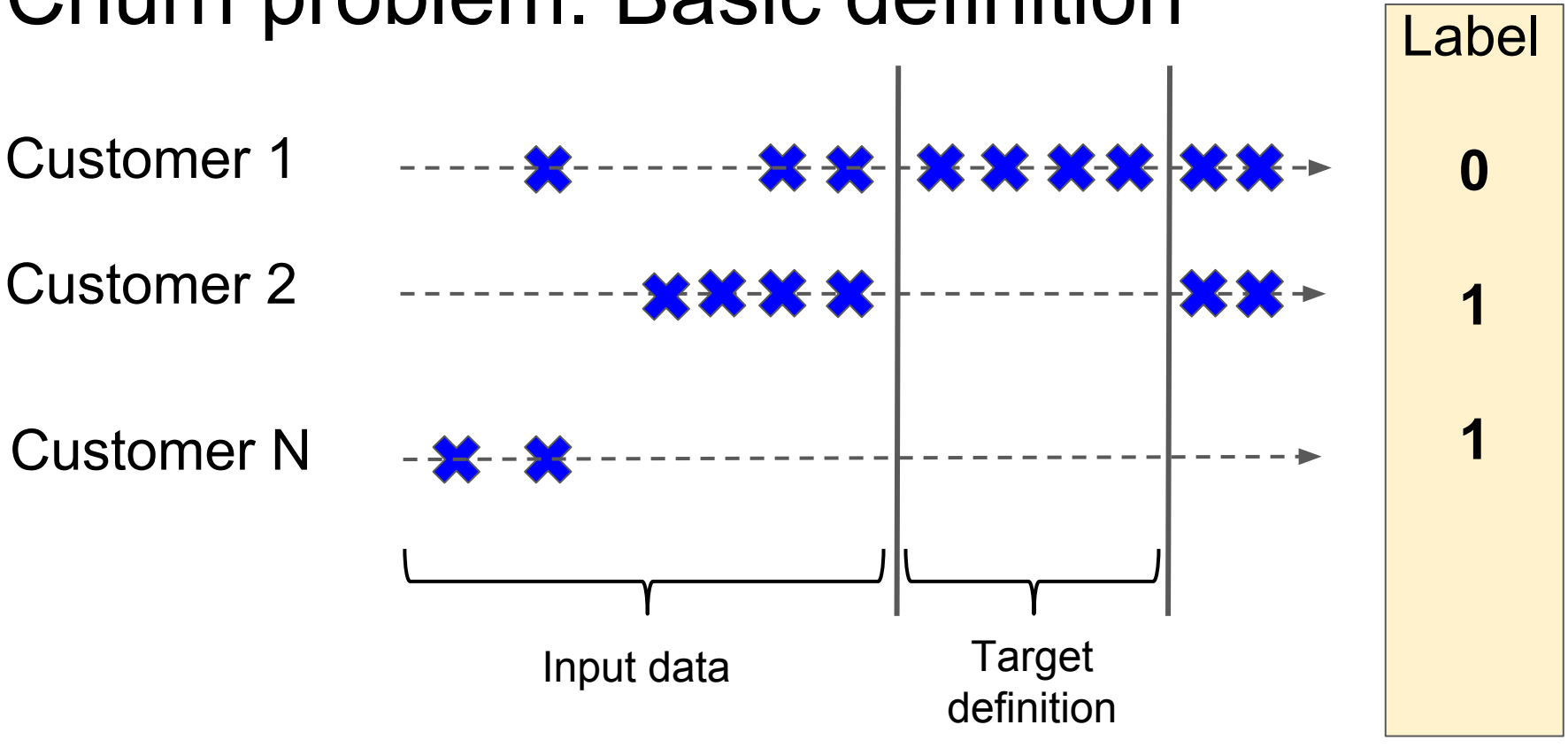
Churn problem. Basic definition



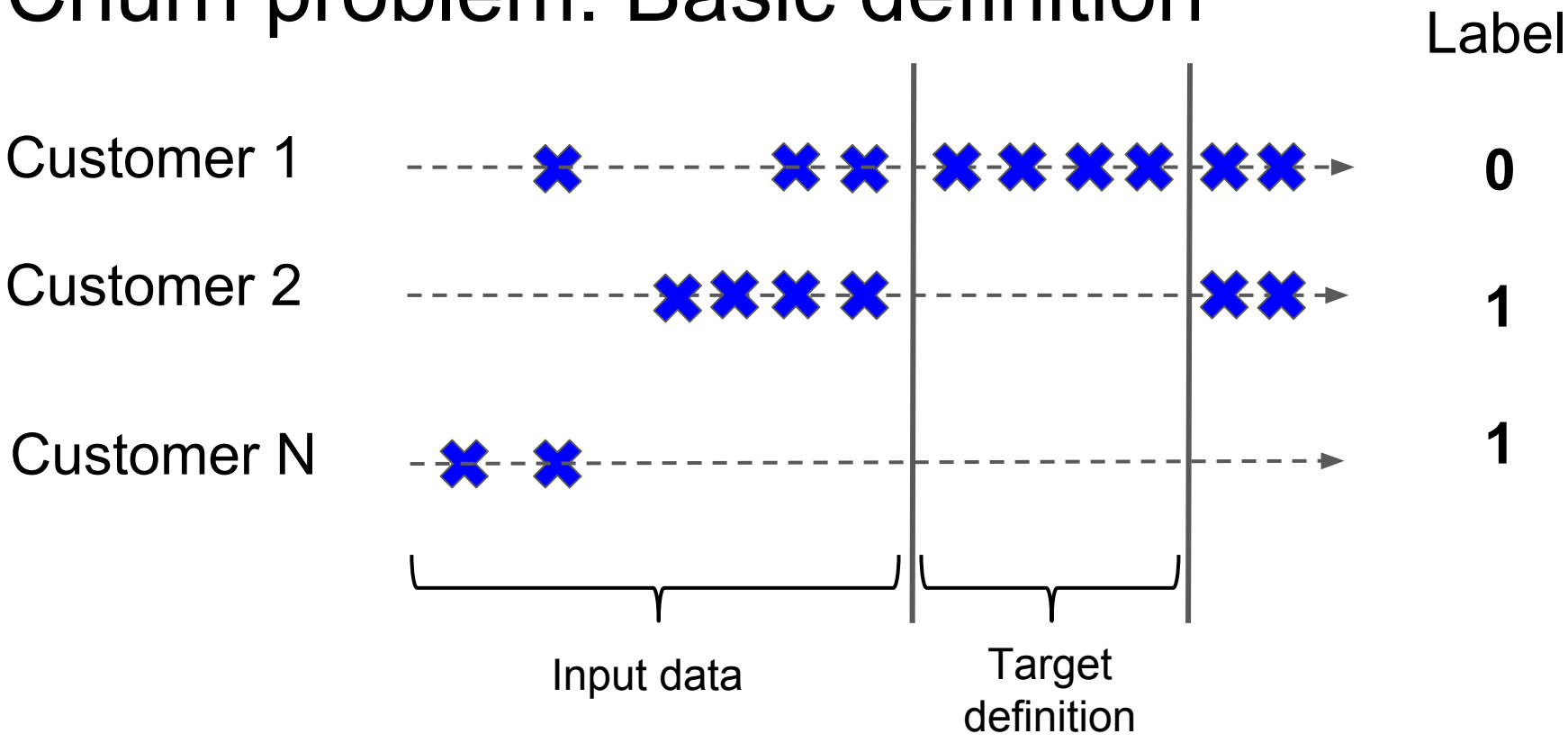
Churn problem. Basic definition



Churn problem. Basic definition

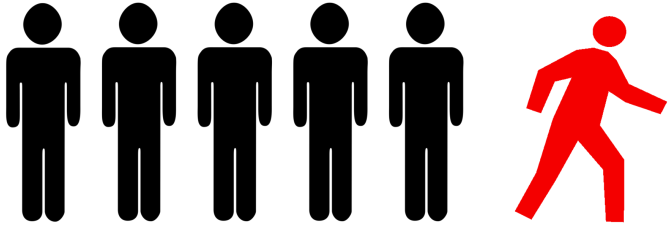


Churn problem. Basic definition



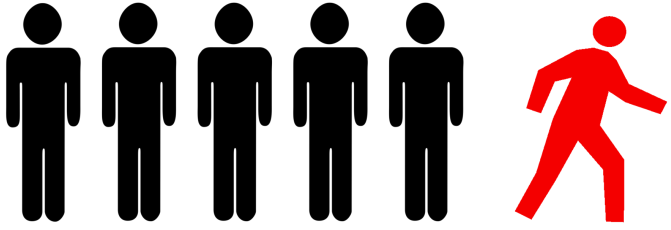
1. Modeling
    a. Motivation
    b. Classical approach
    c. Deep learning
    d. Practical example
2. Performance optimization

Classical approach



id	f_1	f_2	f_3	...	f_n	label
1	US	130	F	...	-1.2	0
2	RU	1	M	...	11.0	0
3	DE	24	F	...	NaN	1
...	...	...	...	...	...	...

Classical approach



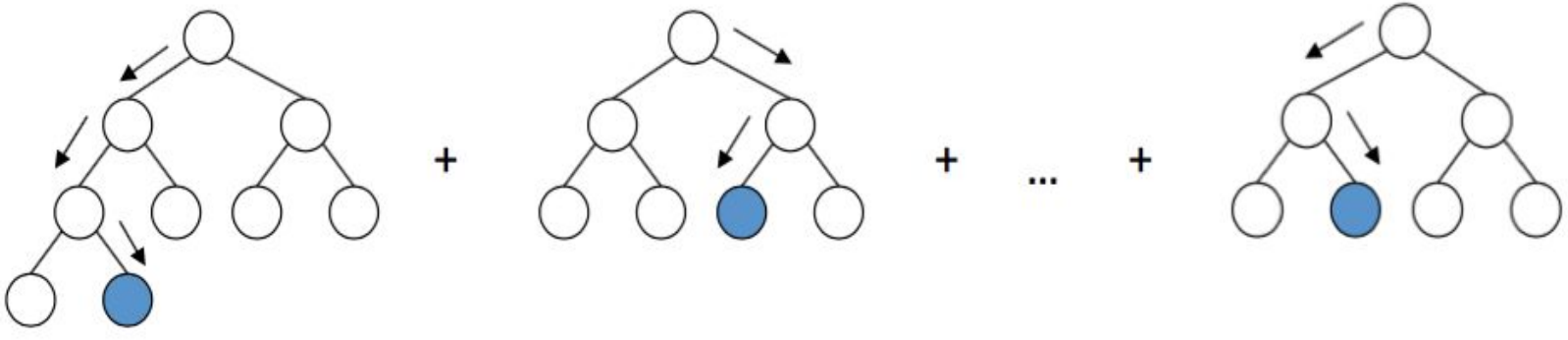
dmlc XGBoost

LightGBM

id	f_1	f_2	f_3	...	f_n	label
1	US	130	F	...	-1.2	0
2	RU	1	M	...	11.0	0
3	DE	24	F	...	NaN	1
...	...	...	...	...	...	...

Gradient Boosting

- Combination of weak trees
- Next tree is built to fix errors of the previous one.

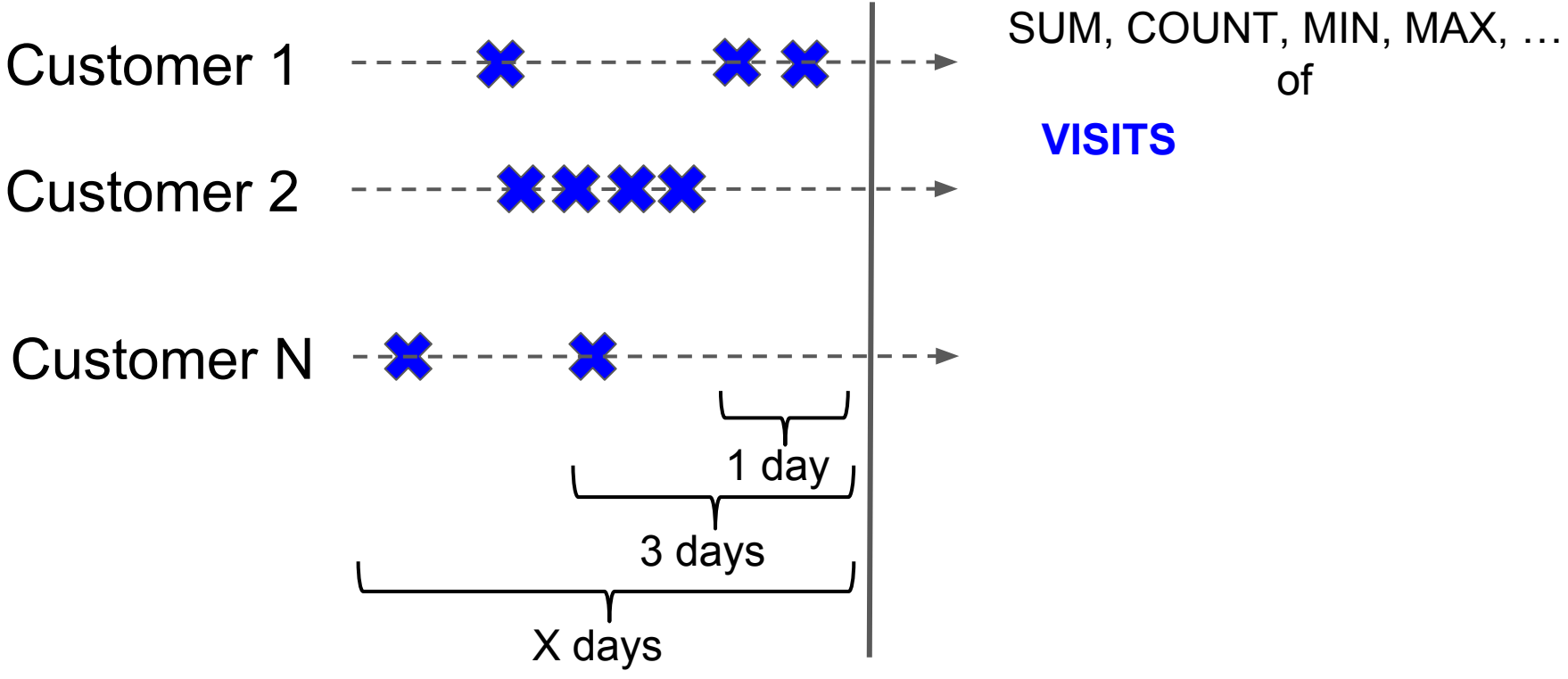


Feature calculation is a little problem

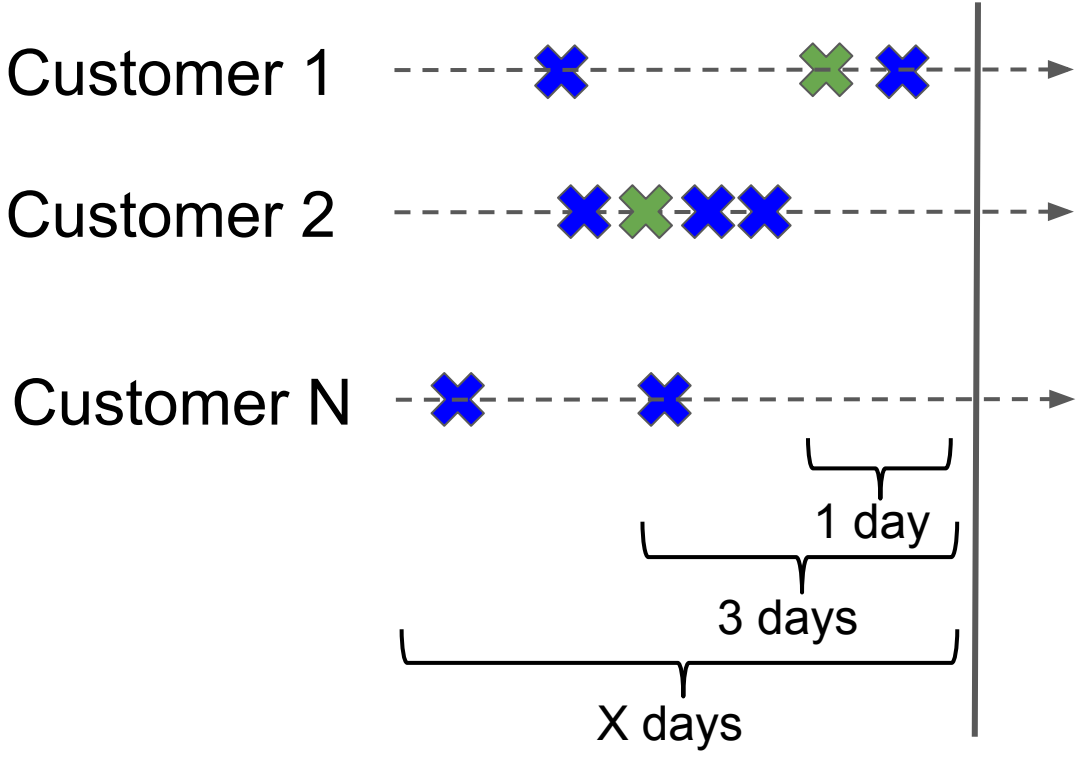
id	timestamp	event_type	event_value
1	1508167056	connect	NaN
1	1508168792	purchase	100
...	...	...	...

id	f_1	f_2	f_3	...	f_n	label
1	US	130	F	...	-1.2	0
2	RU	1	M	...	11.0	0
3	DE	24	F	...	NaN	1
...	...	...	...	...	...	...

Feature calculation is a little problem

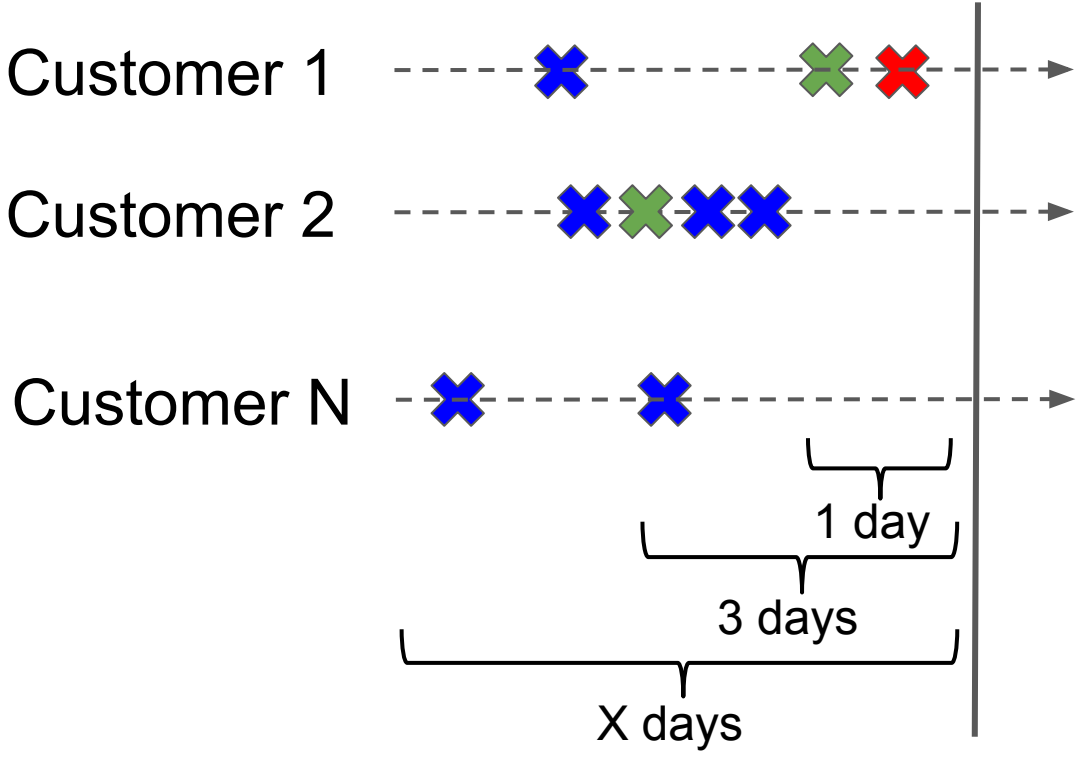


Feature calculation is a little problem



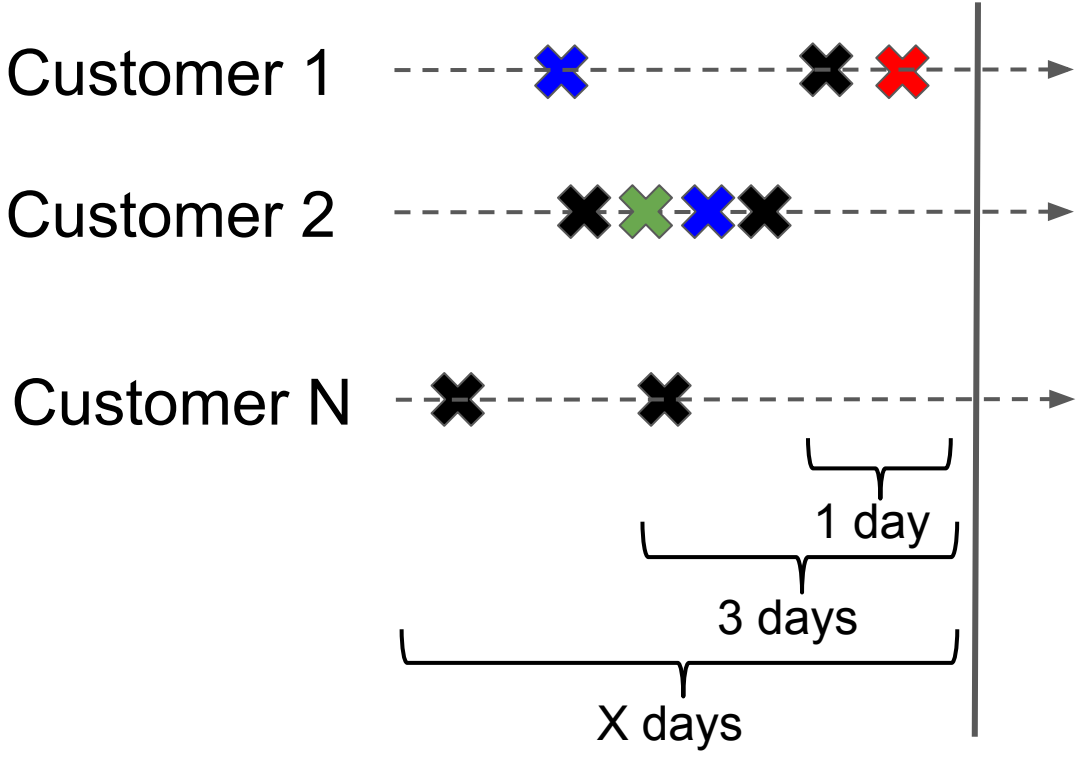
SUM, COUNT, MIN, MAX, …
of
VISITS **PURCHASES**

Feature calculation is a little problem



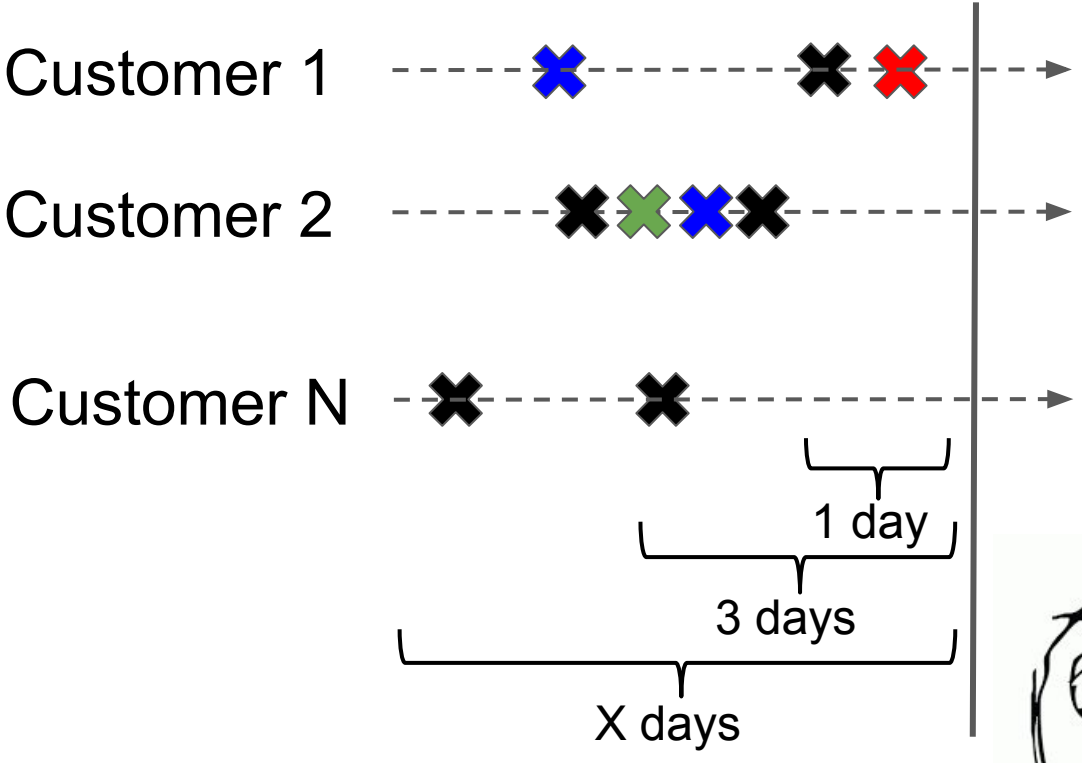
SUM, COUNT, MIN, MAX, …
of
VISITS **PURCHASES**
REFUNDS

Feature calculation is a little problem

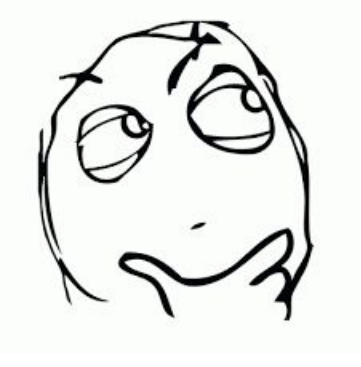


SUM, COUNT, MIN, MAX, …
of
VISITS **PURCHASES**
REFUNDS **BATTLES**

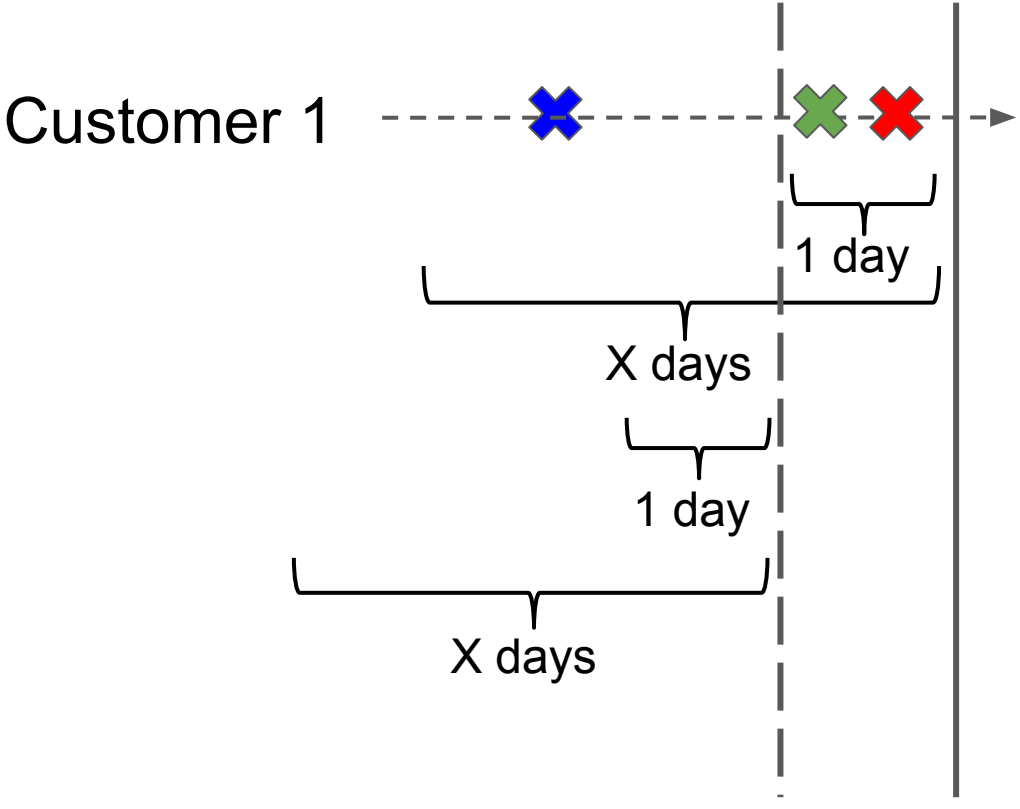
Feature calculation is a little problem



SUM, COUNT, MIN, MAX, …
of
VISITS **PURCHASES**
REFUNDS **BATTLES**

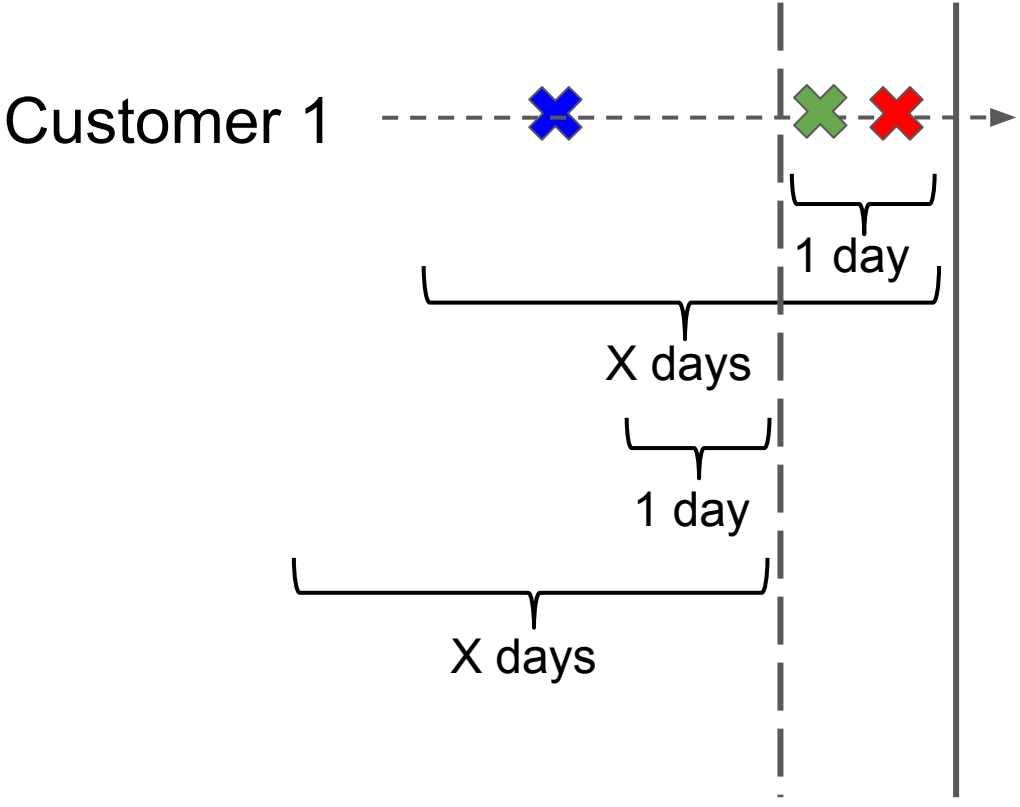


Feature calculation is a problem



SUM, COUNT, MIN, MAX, …
of
VISITS **PURCHASES**
REFUNDS **BATTLES**

Feature calculation is a problem

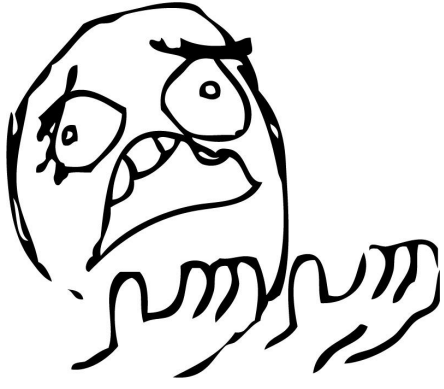


SUM, COUNT, MIN, MAX, …
of
VISITS **PURCHASES**
REFUNDS **BATTLES**

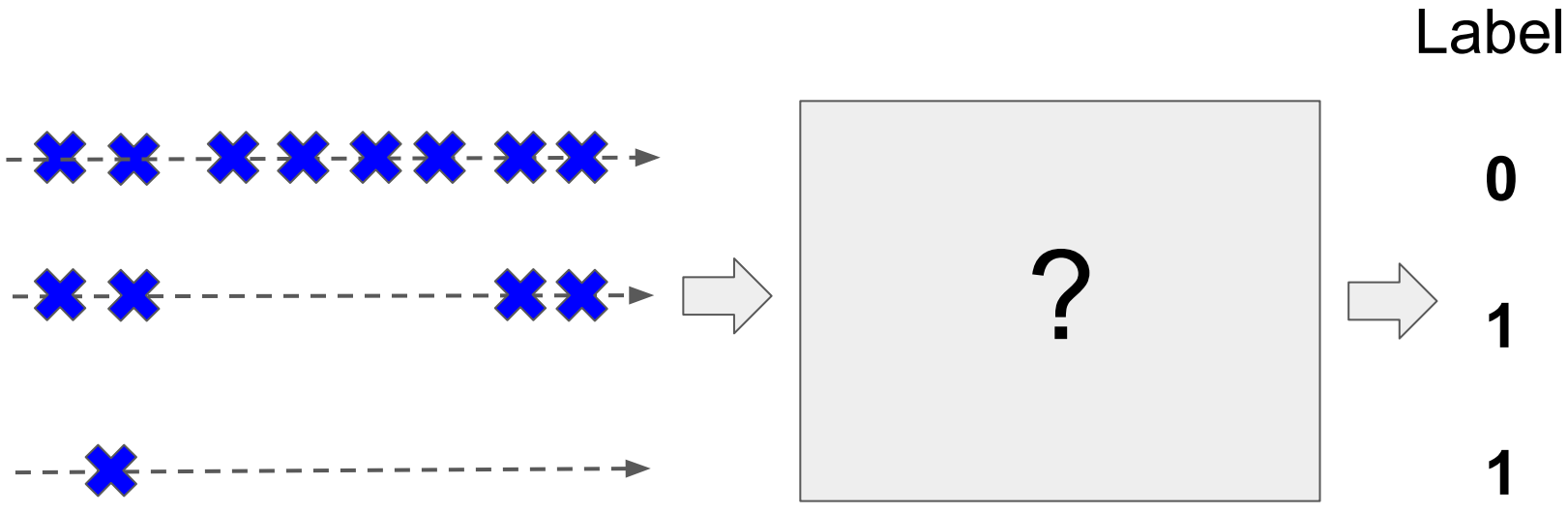
Feature calculation is a big problem

50	×	5	×	10	×	10	=	25 000
event types		aggreg. functions		aggreg. periods		lags		columns

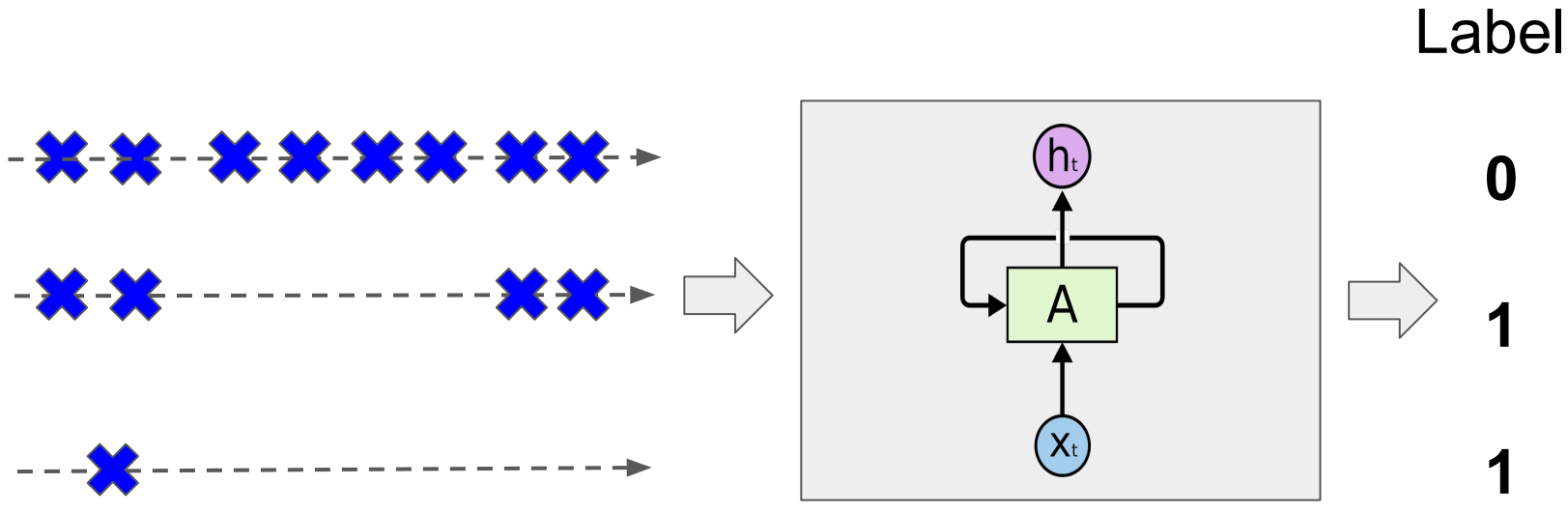
Are the aggregations and periods chosen correctly?



We need something different

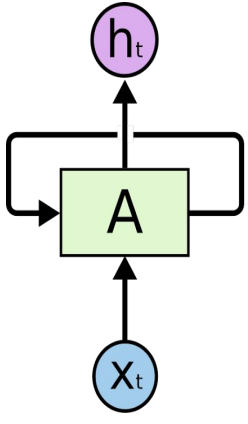


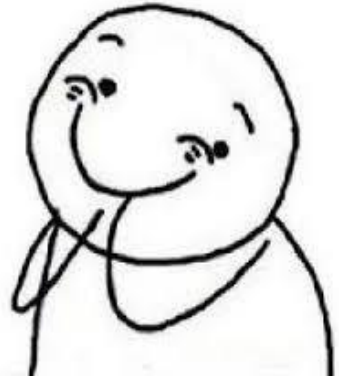
Deep Learning goes on stage



Feature calculation is <u>not a problem</u>

50	×	5	×	10	×	10	=	250
event types		aggreg. functions		aggreg. periods		lags		columns

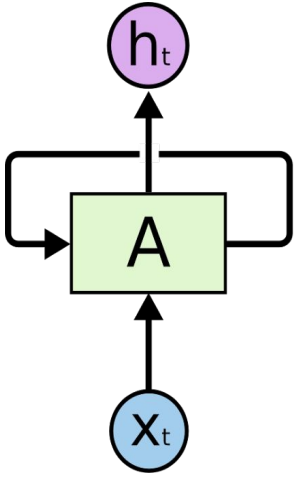




1. Modeling
    a. Motivation
    b. Classical approach
    c. Deep learning
    d. Practical example
2. Performance optimization

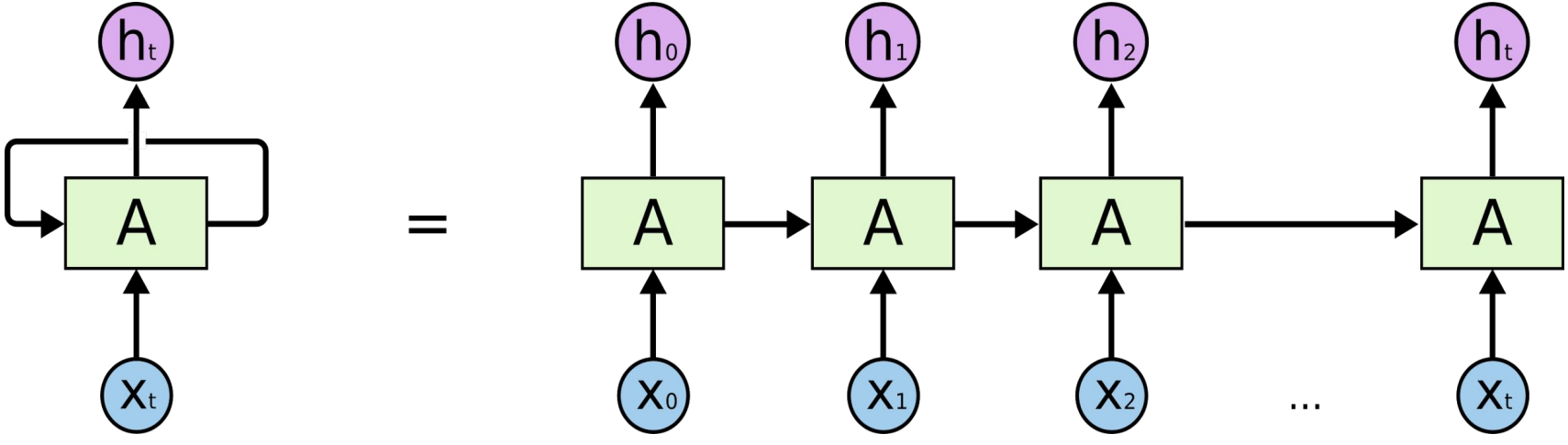
Recurrent Neural Network

Neural Network that accepts **sequences as an input**

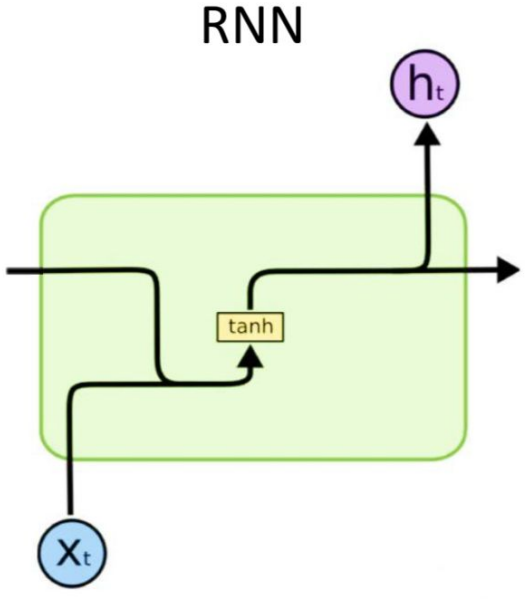


Recurrent Neural Network

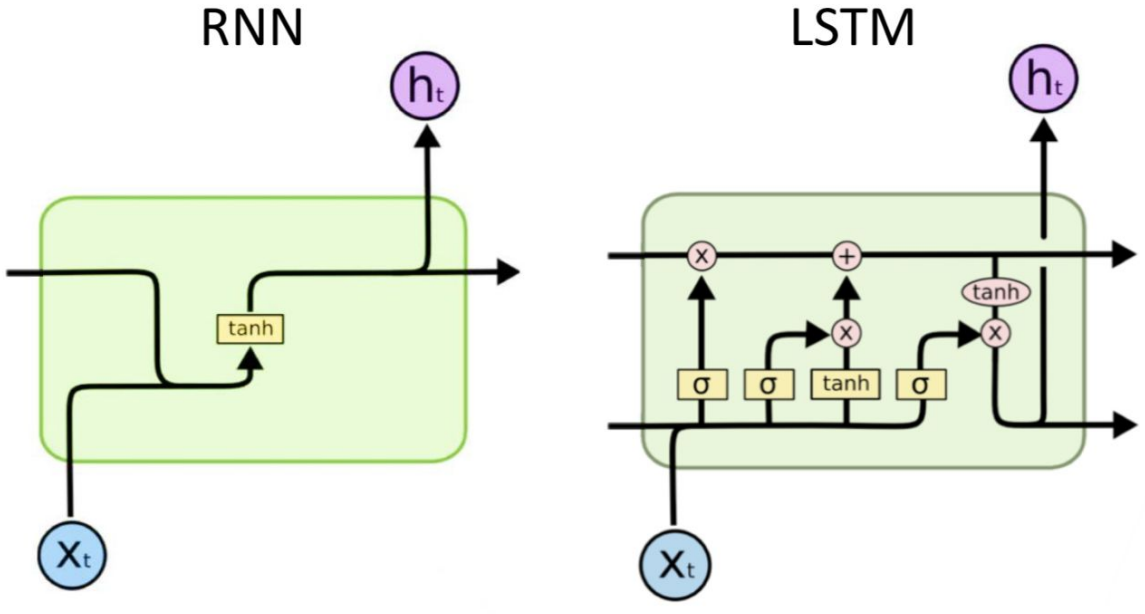
Neural Network that accepts **sequences as an input**



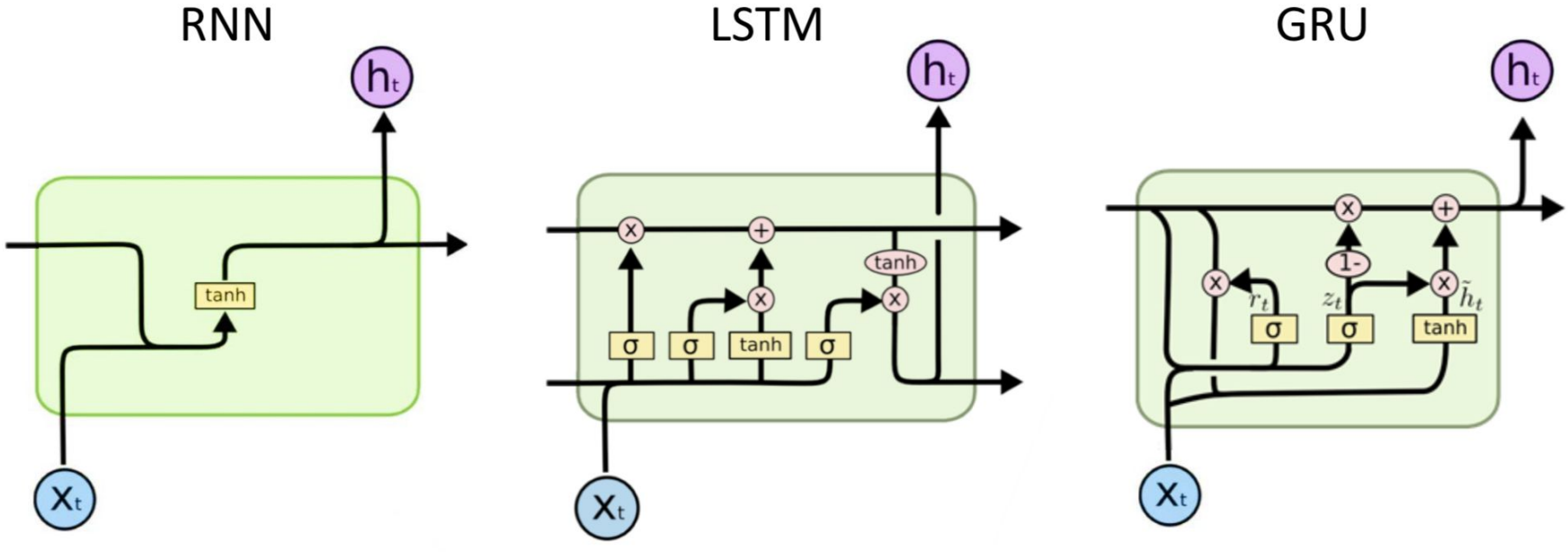
LSTM and GRU



LSTM and GRU



LSTM and GRU



TensorFlow and Keras

- TensorFlow is an ML library from Google
- Flexible, supports LSTM/GRU, has python API

TensorFlow and Keras

- Keras is a high-level NN API in Python
- Runs on top of TensorFlow, CNTK, or Theano

1. Modeling
    a. Motivation
    b. Classical approach
    c. Deep learning
    d. Practical example
2. Performance optimization

Vikings

- Mobile and browser MMO

Vikings

- Mobile and browser MMO
- Actions:
  - create and develop clans,
  - train troops
  - upgrade heroes and towns
  - obtain resources
  - attack others

Vikings

- Mobile and browser MMO
- Actions:
  - create and develop clans,
  - train troops
  - upgrade heroes and towns
  - obtain resources
  - attack others
- Social interactions, competitions
- Tons of data

Vikings dataset

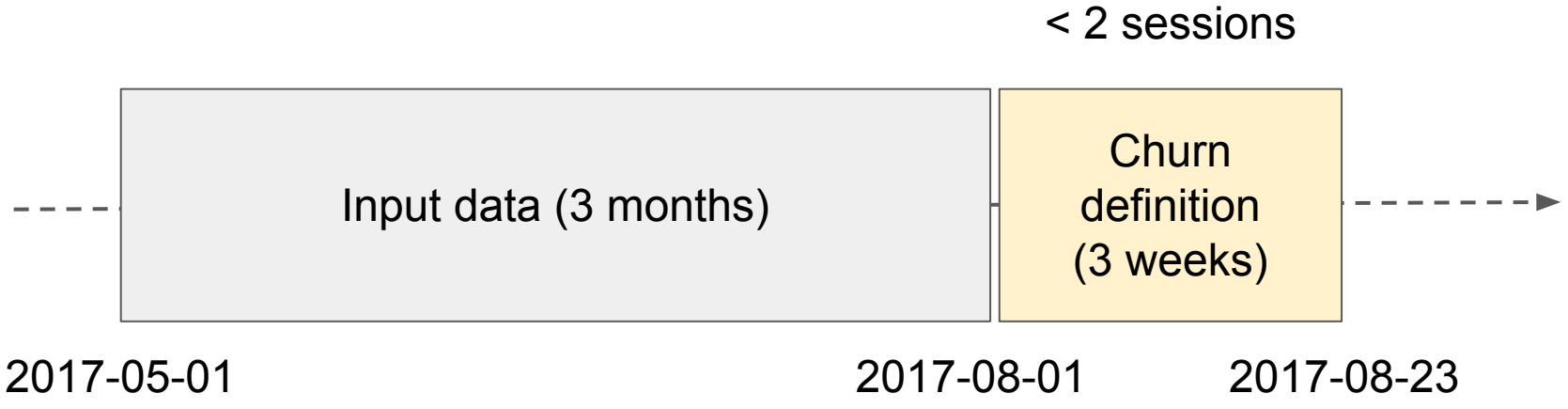
Vikings dataset

- 3 months

Vikings dataset

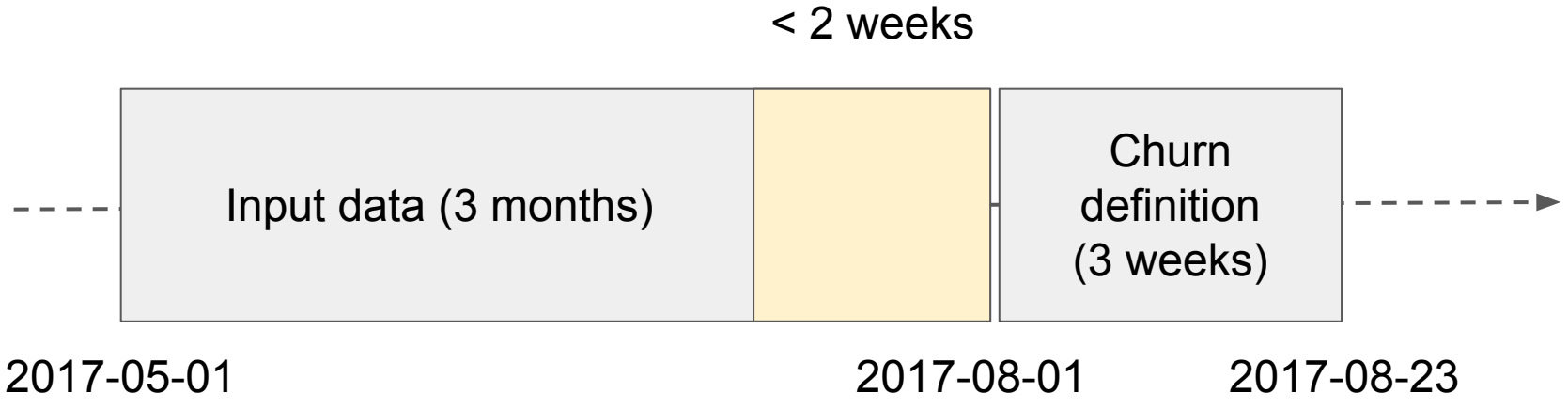
- 3 months

Vikings dataset



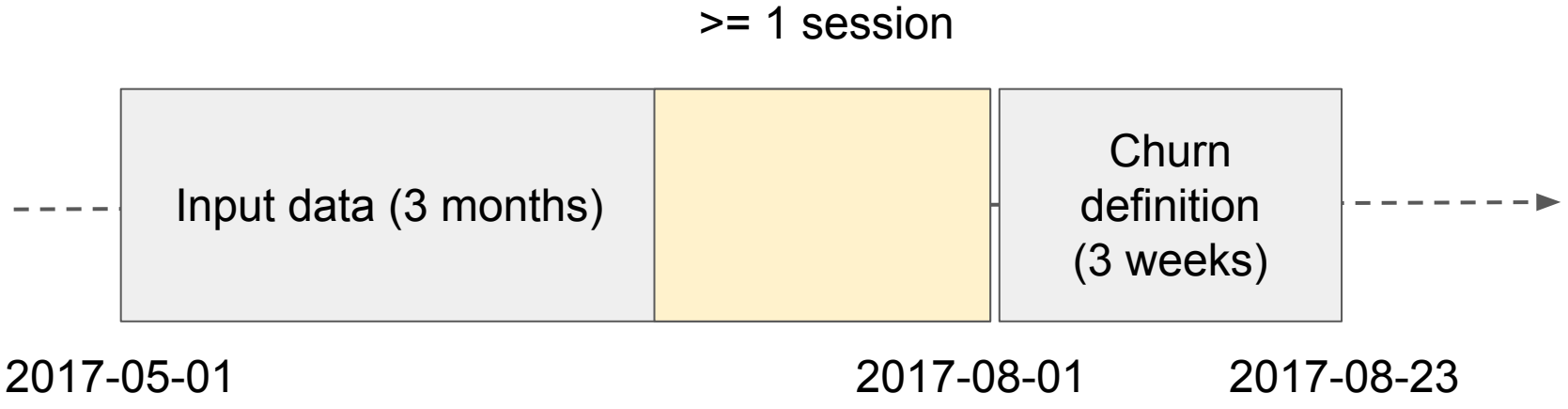
- 3 months
- Churn is < 2 sessions for the first 3 weeks of August

Vikings dataset



- 3 months
- Churn is < 2 sessions for the first 3 weeks of August
- No newcomers, > 7 game days

Vikings dataset



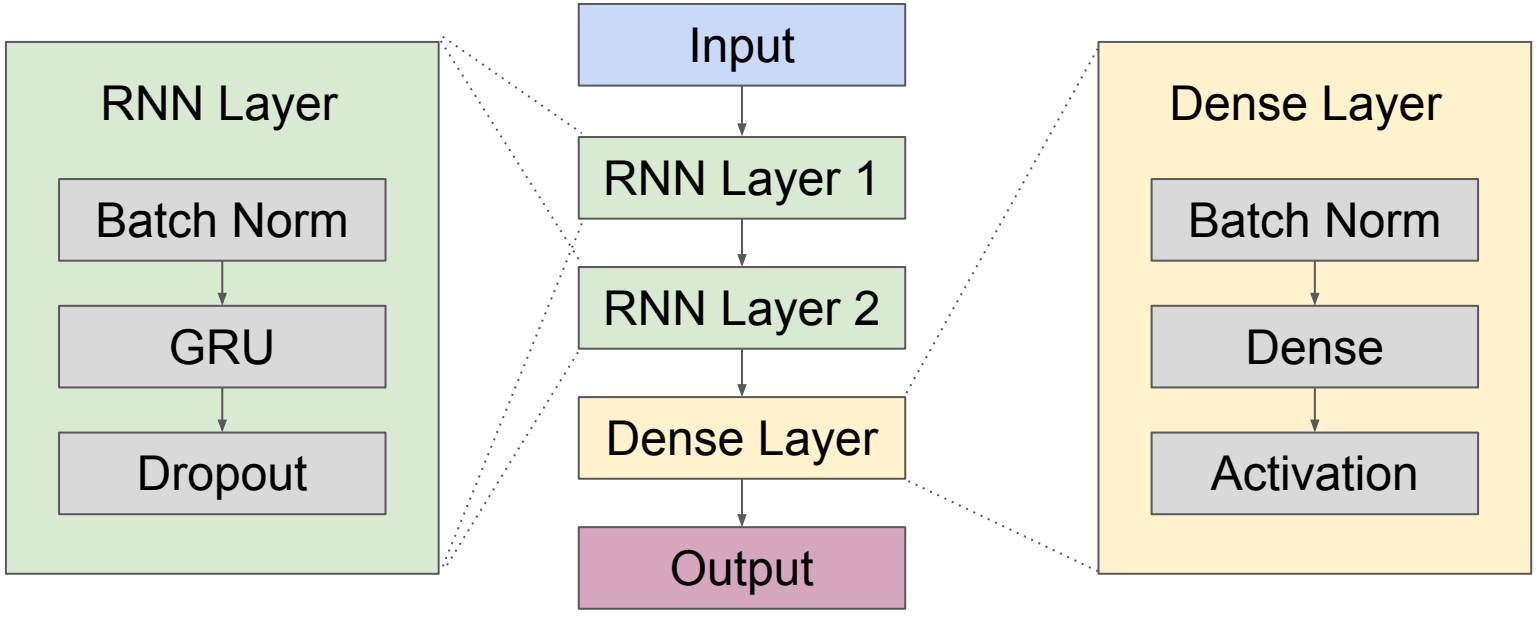
- 3 months
- Churn is < 2 sessions for the first 3 weeks of August
- No newcomers, > 7 game days
- Relatively active

Vikings dataset

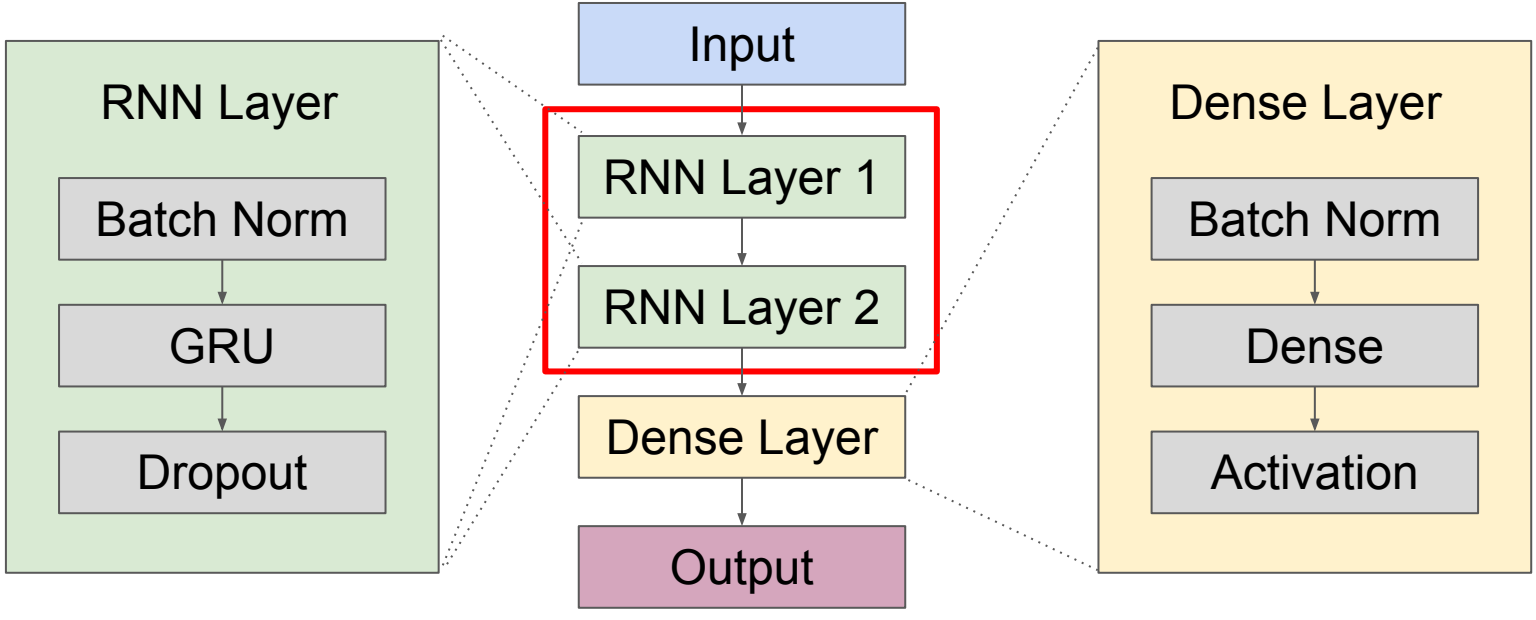
100k random players, **121** features, **92** days

~9 Gb unarchived

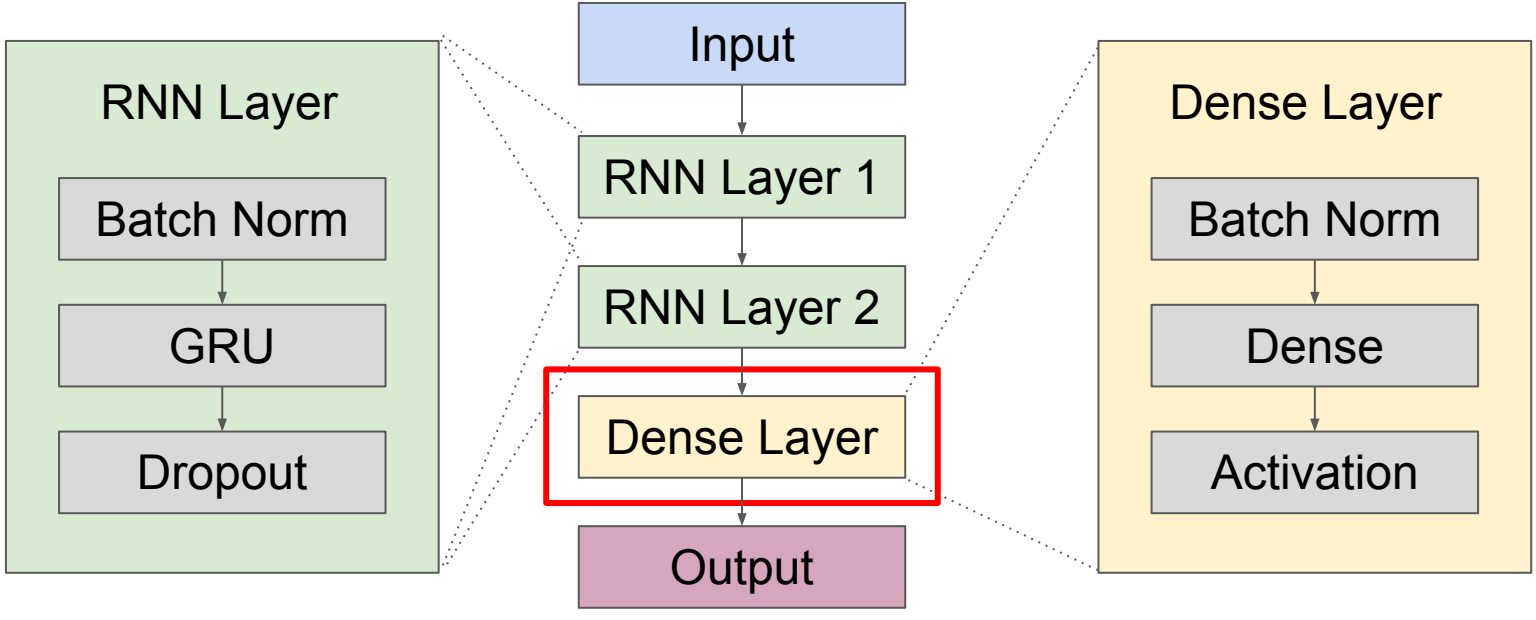
Neural Network Architecture



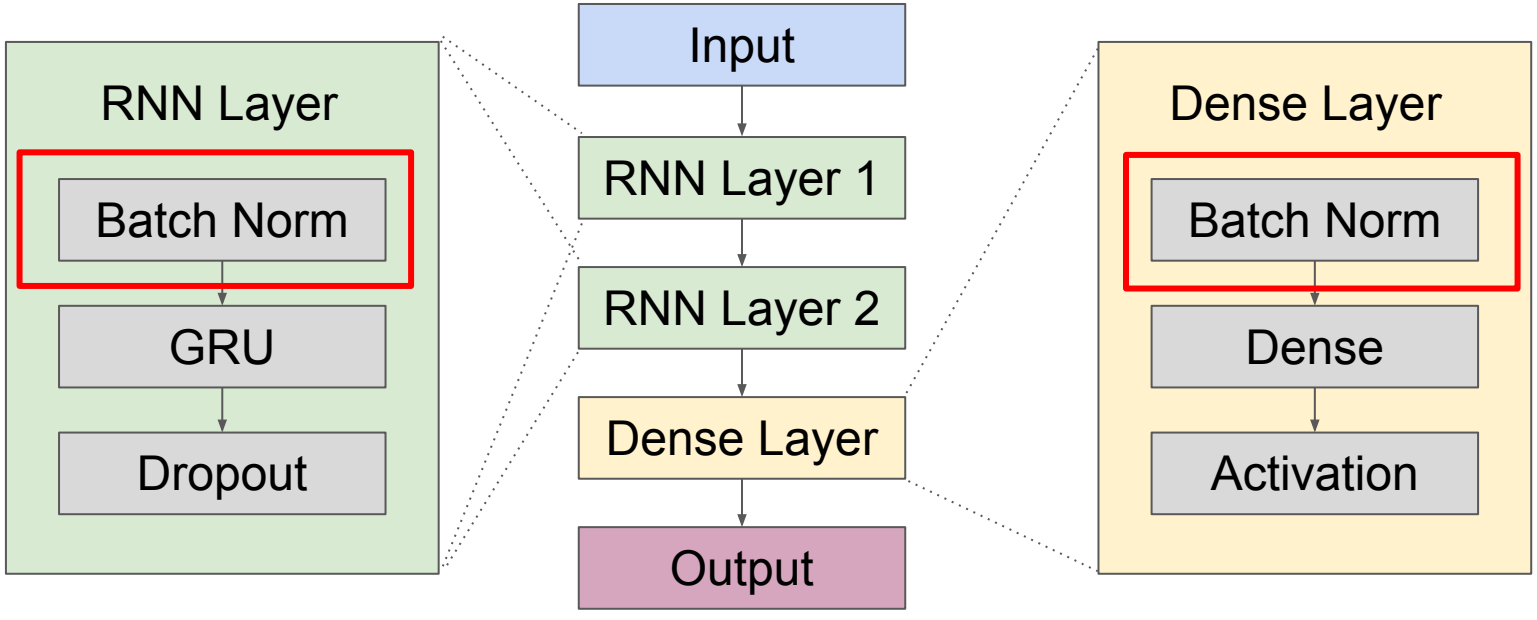
Neural Network Architecture



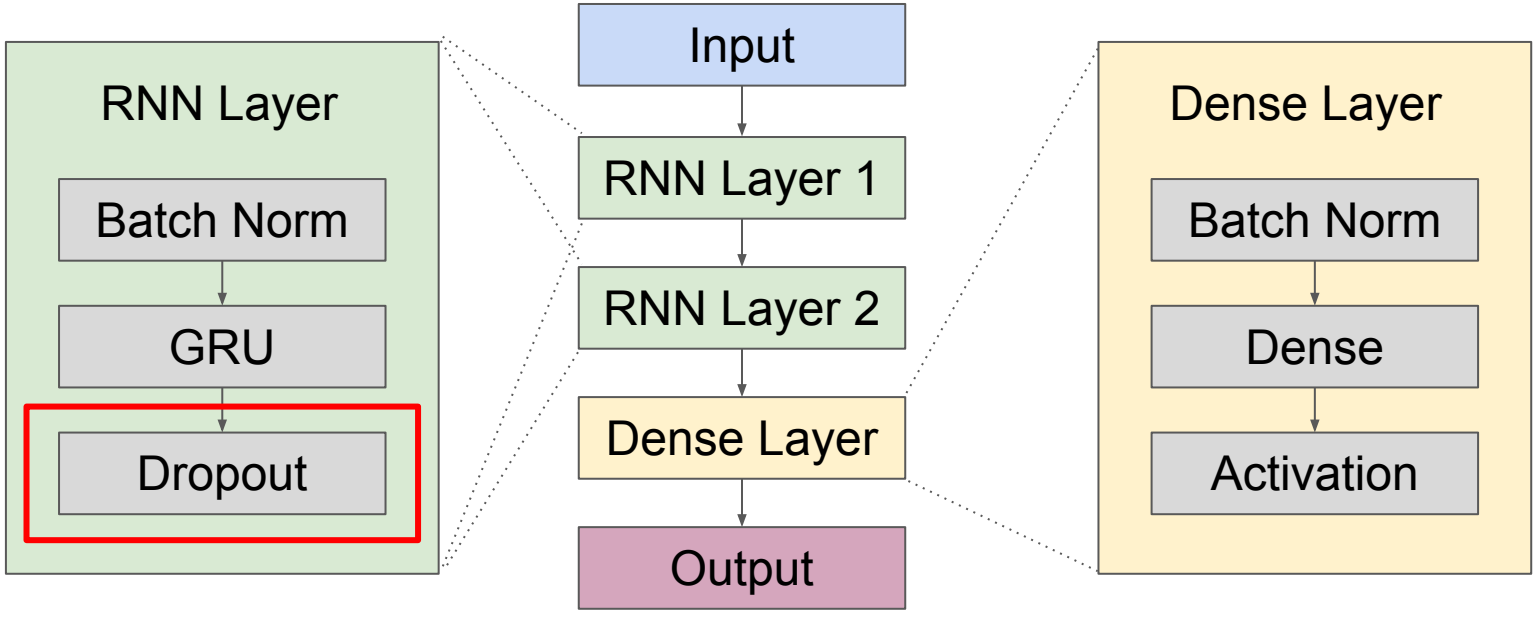
Neural Network Architecture



Neural Network Architecture



Neural Network Architecture



Let’s code (finally)

```
X = np.load('X.npy')
y = np.load('y.npy')
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.2, random_state=42)
n_hidden, n_feat, n_days = (512, X.shape[2], X.shape[1])
```

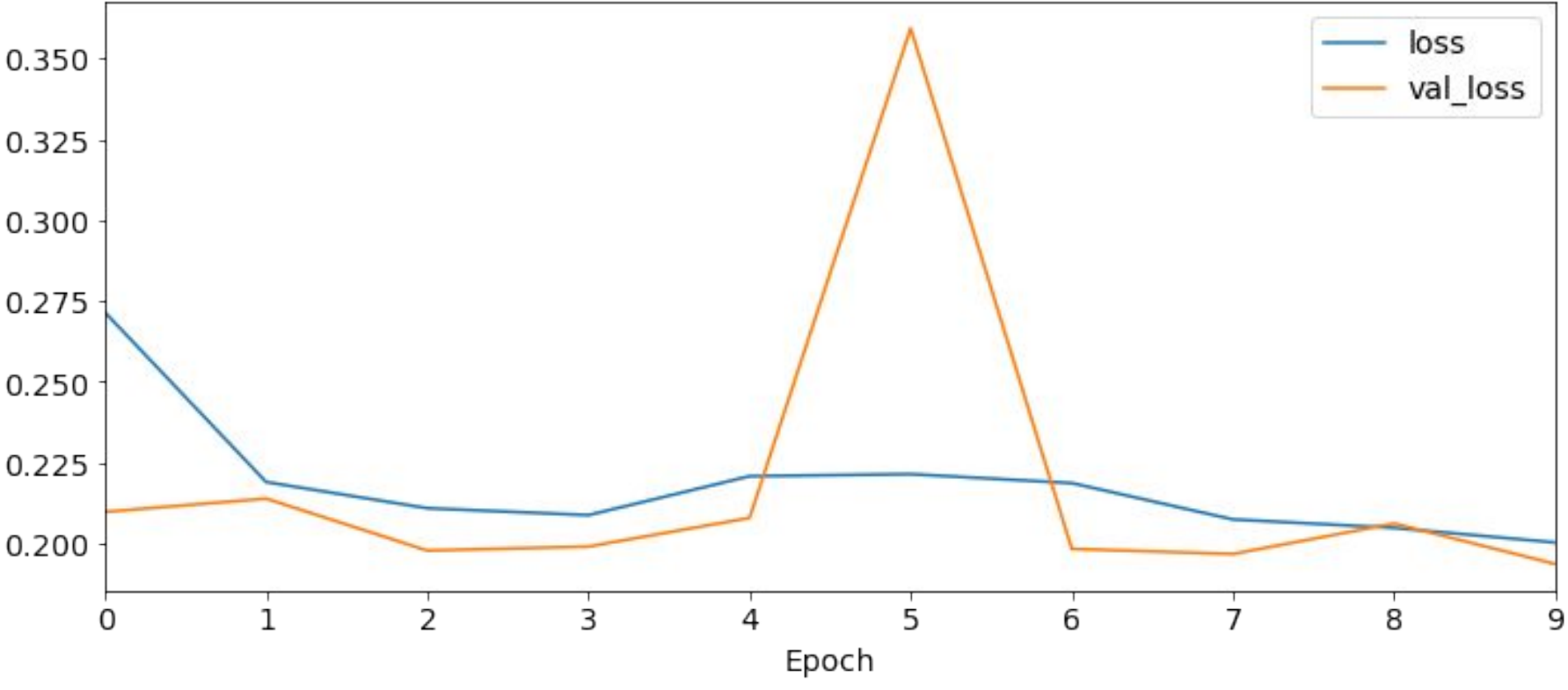
Let’s code (finally)

```
6   model = Sequential([
7           BatchNormalization(input_shape=(n_days, n_feat),
8                              dtype='float32'),
9           GRU(n_hidden, return_sequences=True),
10          Dropout(0.5),
11          BatchNormalization(),
12          GRU(n_hidden),
13          Dropout(0.5),
14          BatchNormalization(),
15          Dense(2, activation='softmax')
16      ])
```

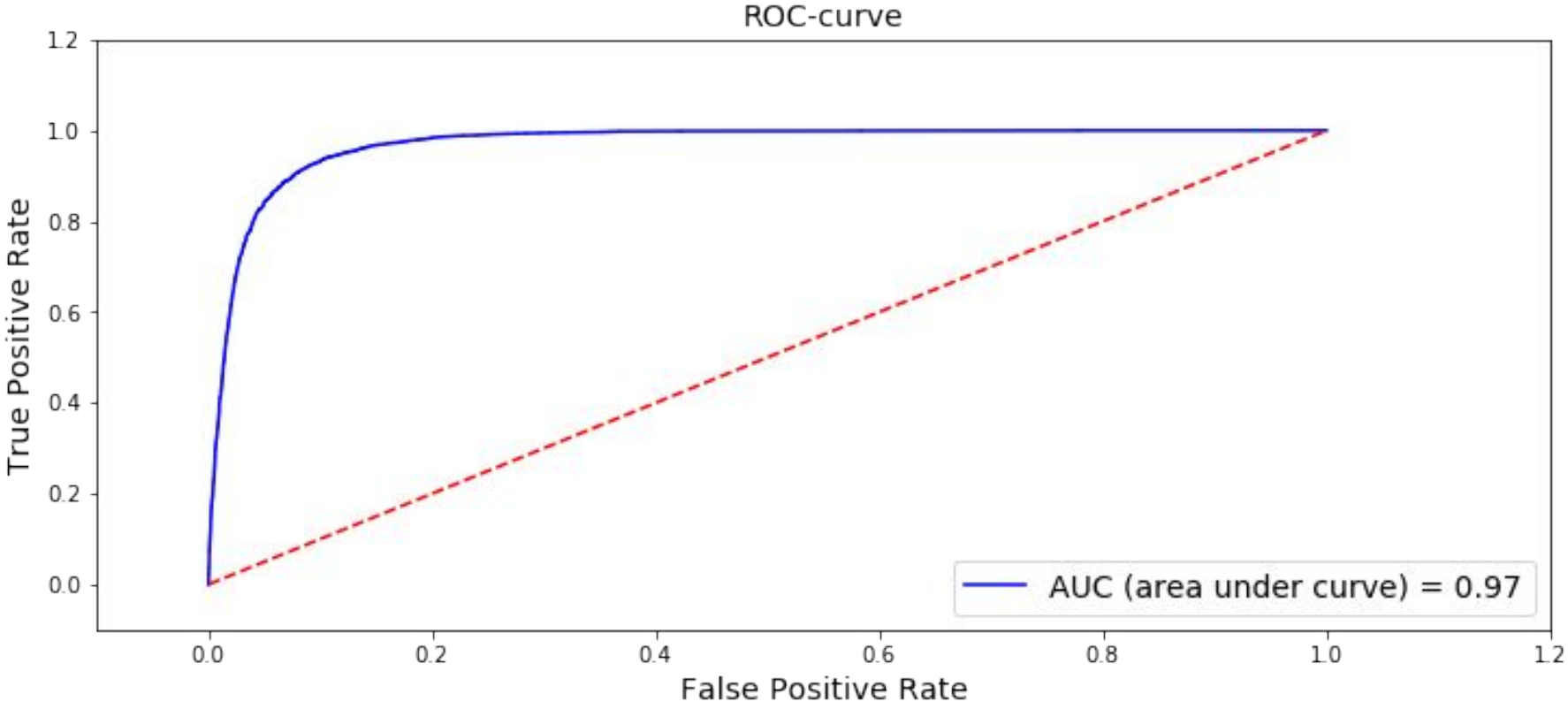
Let’s code (finally)

```
17  model.compile(loss='categorical_crossentropy',
18                optimizer=Adam())
19  model.fit(X_train, y_train, validation_split=0.2,
20            batch_size=64, epochs=10)
21  pred = model.predict(X_test)
22  metrics.roc_auc_score(y_test[:, 1], pred[:, 1])
```

Let’s train


loss
val_loss
0.350
0.325
0.300
0.275
0.250
0.225
0.200
0
1
2
3
4
5
6
7
8
9
Epoch

Model performance



Speed performance

Train on 64000 samples, validate on 16000 samples
Epoch 1/10
*64000/64000 [==============================] - **952s** - loss: 0.2714 - val_loss: 0.2098*
Epoch 2/10
*64000/64000 [==============================] - **945s** - loss: 0.2190 - val_loss: 0.2139*
...
Epoch 9/10
*64000/64000 [==============================] - **940s** - loss: 0.2048 - val_loss: 0.2062*
Epoch 10/10
*64000/64000 [==============================] - **944s** - loss: 0.2003 - val_loss: 0.1937*

- Training - 16 minutes per epoch, ~3 hours in total

Speed performance

Train on 64000 samples, validate on 16000 samples
Epoch 1/10
64000/64000 [==============================] - 952s - loss: 0.2714 - val_loss: 0.2098
Epoch 2/10
64000/64000 [==============================] - 945s - loss: 0.2190 - val_loss: 0.2139
...
Epoch 9/10
64000/64000 [==============================] - 940s - loss: 0.2048 - val_loss: 0.2062
Epoch 10/10
64000/64000 [==============================] - 944s - loss: 0.2003 - val_loss: 0.1937

- Training - 16 minutes per epoch, ~3 hours in total
- Execution - 1 minute for each 10k customers

Speed performance

Train on 64000 samples, validate on 16000 samples
Epoch 1/10
64000/64000 [==============================] - 952s - loss: 0.2714 - val_loss: 0.2098
Epoch 2/10
64000/64000 [==============================] - 945s - loss: 0.2190 - val_loss: 0.2139
...
Epoch 9/10
64000/64000 [==============================] - 940s - loss: 0.2048 - val_loss: 0.2062
Epoch 10/10
64000/64000 [==============================] - 944s - loss: 0.2003 - val_loss: 0.1937

- Training - 16 minutes per epoch, ~3 hours in total
- Execution - 1 minute for each 10k customers
- Hardware: 2 x Intel Xeon CPU E5-2620 v4 @ 2.10GHz (16 threads each)

What we’ve learned so far

Deep Neural Networks are

- very flexible

Deep Neural Networks are

- very flexible
- well performing

Deep Neural Networks are

- very flexible
- well performing
- simplify time series feature calculation

Deep Neural Networks are

- very flexible
- well performing
- simplify time series feature calculation
- not scary, especially with Keras

Deep Neural Networks are

- very flexible
- well performing
- simplify time series feature calculation
- not scary, especially with Keras
- demanding for resources

Deep Neural Networks are

- very flexible
- well performing
- simplify time series feature calculation
- not scary, especially with Keras
- demanding for resources
- slow....

How to make it faster?

Deep Neural Networks are

- very flexible
- well performing
- simplify time series feature calculation
- not scary, especially with Keras
- demanding for resources
- slow....

How to make it faster?

1. Churn Prediction
2. Performance optimization
    a. Background
    b. History
    c. Parallelism Types
    d. Existing Frameworks
    e. Benchmarks

1. Churn Prediction
2. Performance optimization
    a. Background
    b. History
    c. Parallelism Types
    d. Existing Frameworks
    e. Benchmarks

Which Types of Tasks Need Speedup?

In Big Data era, the more data = the better.

Models for regular business tasks = lots of data (users x events)!

Speech/image recognition, physics modelling = lots of data

So how do we make training process faster?

1. Churn Prediction
2. Performance optimization
    a. Background
    b. History
    c. Parallelism Types
    d. Existing Frameworks
    e. Benchmarks

Jeff Dean

Large Scale Distributed Deep Networks

Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen,
Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato,
Andrew Senior, Paul Tucker, Ke Yang, Andrew Y. Ng
{jeff, gcorrado}@google.com
Google Inc., Mountain View, CA

Abstract

Recent work in unsupervised feature learning and deep learning has shown that being able to train large models can dramatically improve performance. In this paper, we consider the problem of training a deep network with billions of parameters using tens of thousands of CPU cores. We have developed a software framework called *DistBelief* that can utilize computing clusters with thousands of machines to train large models. Within this framework, we have developed two algorithms for large-scale distributed training: (i) Downpour SGD, an asynchronous stochastic gradient descent procedure supporting a large number of model replicas, and (ii) Sandblaster, a framework that supports a variety of distributed batch optimization procedures, including a distributed implementation of L-BFGS. Downpour SGD and Sandblaster L-BFGS both increase the scale and speed of deep network training. We have successfully used our system to train a deep network 30x larger than previously reported in the literature, and achieves state-of-the-art performance on ImageNet, a visual object recognition task with 16 million images and 21k categories. We show that these same techniques dramatically accelerate the training of a more modestly- sized deep network for a commercial speech recognition ser-

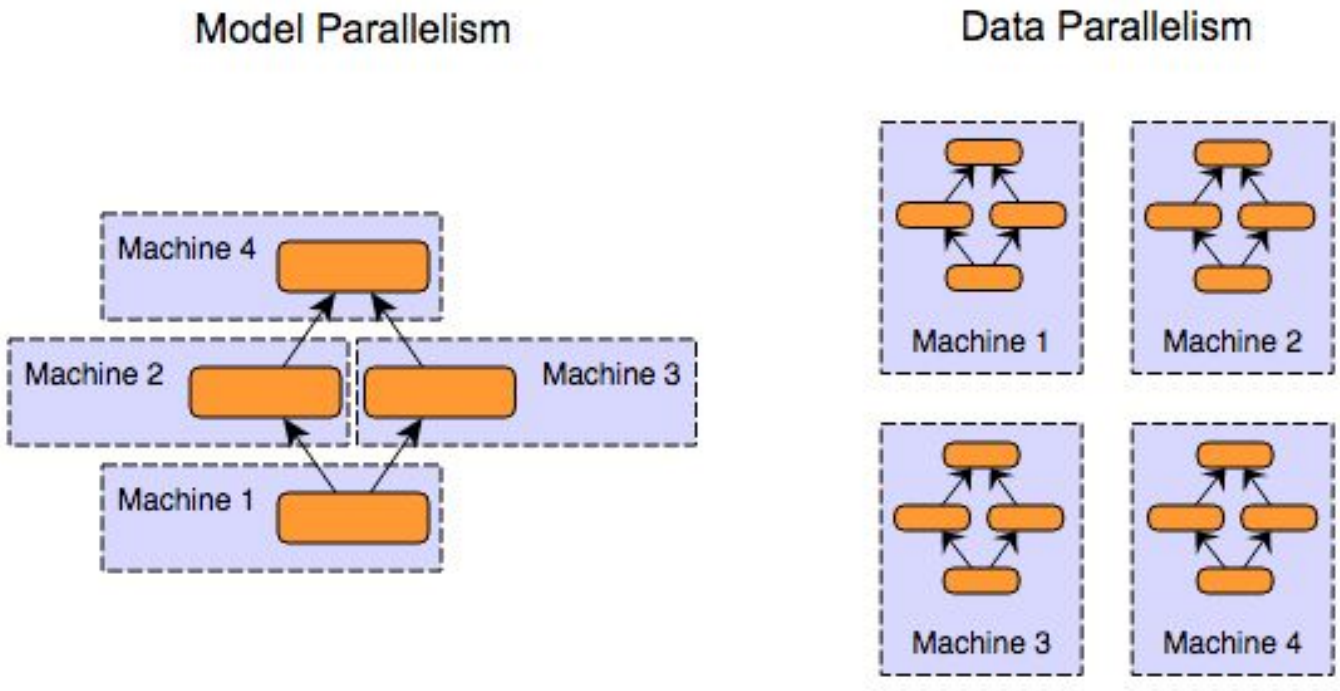
History

This paper introduced a new approaches

1. Downpour SGD
2. Sandblaster L-BFGS

...These are **Asynchronous Data Parallelism** algorithms!

1. Churn Prediction
2. Performance optimization
    a. Background
    b. History
    c. Parallelism Types
    d. Existing Frameworks
    e. Benchmarks



Parallelism Types

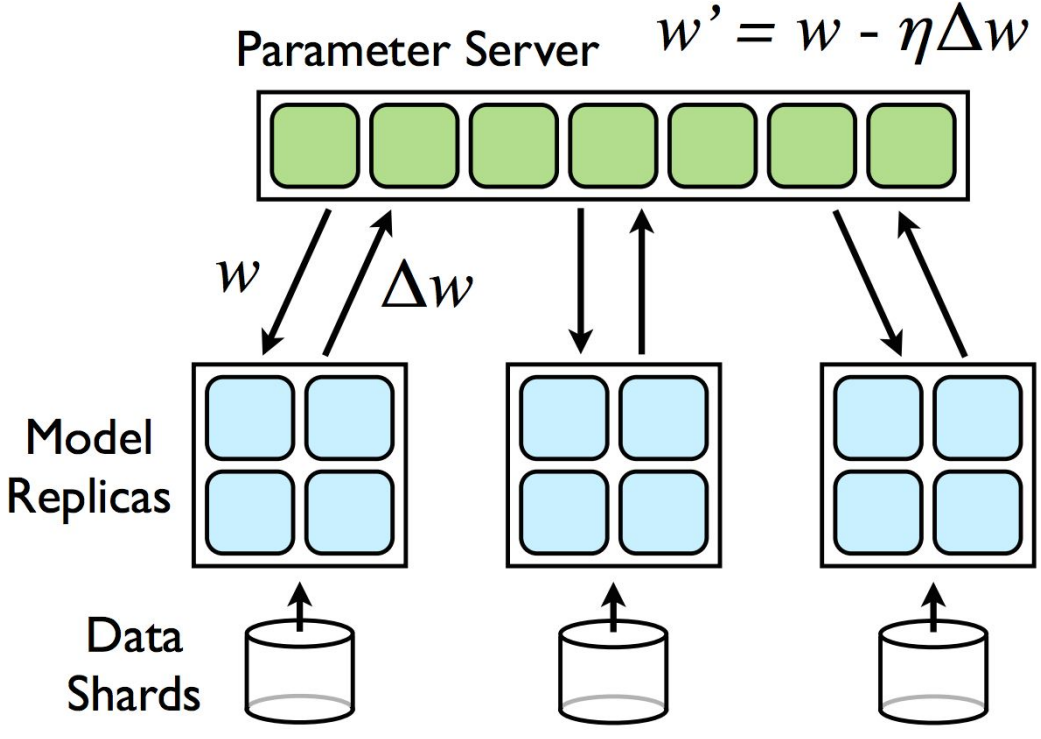
Model Parallelism

Tricky to implement for fully connected networks:

- Layers will have to communicate across different devices
- You have to understand you model very well

Let’s settle on Data Parallelism

How do we synchronize all models?



Architecture for Data Parallelism

Data Parallelism: Synchronous

Upsides

- Potentially bigger learning rates

Downsides:

- Limited parallelism
- Usually slower

Data Parallelism: Asynchronous

Upsides:

- Better Scalability
- Better Cluster Load

Downsides:

- “the straggler problem” – due to staleness of coefficients
- not converging sometimes

Revisiting Distributed Synchronous SGD

Jianmin Chen, Rajat Monga, Samy Bengio & Rafal Jozefowicz
Google Brain
Mountain View, CA, USA
{jmchen,rajatmonga,bengio,rafalj}@google.com

1 The Need for a Large Scale Deep Learning Infrastructure

The recent success of deep learning approaches for domains like speech recognition (Hinton et al., 2012) and computer vision (Ioffe & Szegedy, 2015) stems from many algorithmic improvements but also from the fact that the size of available training data has grown significantly over the years, together with the computing power, in terms of both CPUs and GPUs.

While a single GPU often provides algorithmic simplicity and speed up to a given scale of data and model, there exist an operating point where a distributed implementation of training algorithms for deep architectures becomes necessary.

2 Asynchronous Stochastic Gradient Descent

In 2012, Dean et al. (2012) presented their approach for a distributed stochastic gradient descent algorithm. It consists of two main ingredients. First, the parameters of the model can be distributed on multiple servers, depending on the architecture. This set of servers are called the *parameter servers*. Second, there can be multiple workers processing data in parallel and communicating with the parameter servers. Each worker processes a mini-batch of data independently of the other ones, as follows:

- it fetches from the parameter servers the most up-to-date parameters of the model needed to process the current mini-batch;
- it then computes gradients of the loss with respect to these parameters;
- finally, these gradients are sent back to the parameter servers, which then updates the model

VS

Master Thesis

On Scalable Deep Learning and Parallelizing Gradient Descent

Joeri R. Hermans

Master Thesis DKE 17-11

Thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science of Artificial Intelligence

Thesis Committee:
Dr. Gerasimos Spanakis
Dr. Rico Möckel

Maastricht University
Faculty of Humanities and Sciences
Department of Data Science & Knowledge Engineering
Maastricht, The Netherlands

1. Churn Prediction
2. Performance optimization
    a. Background
    b. History
    c. Parallelism Types
    d. Existing Frameworks
    e. Benchmarks

Alright, let’s use TF in distributed mode…


TensorFlow
DEV SUMMIT 2017

O'REILLY®

Hands-On Machine Learning with Scikit-Learn & TensorFlow

CONCEPTS, TOOLS, AND TECHNIQUES TO BUILD INTELLIGENT SYSTEMS

Aurélien Géron

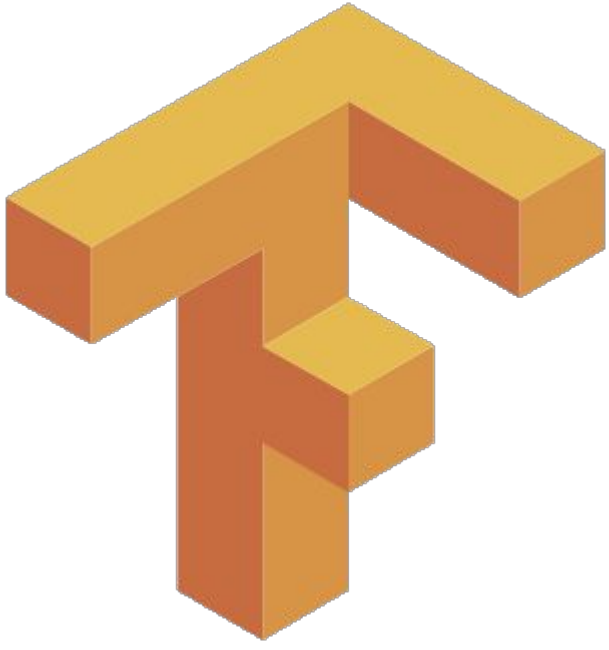
Or we had better find an alternative?

TF Distributed is quite tedious to configure

You have to run it manually


APACHE
Spark
TM

+



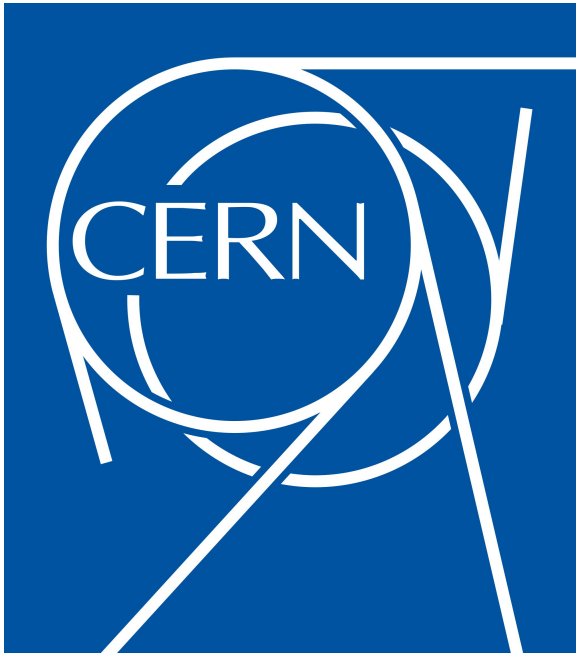

hadoop
YARN

https://github.com/yahoo/TensorFlowOnSpark

DEEPLEARNING4J

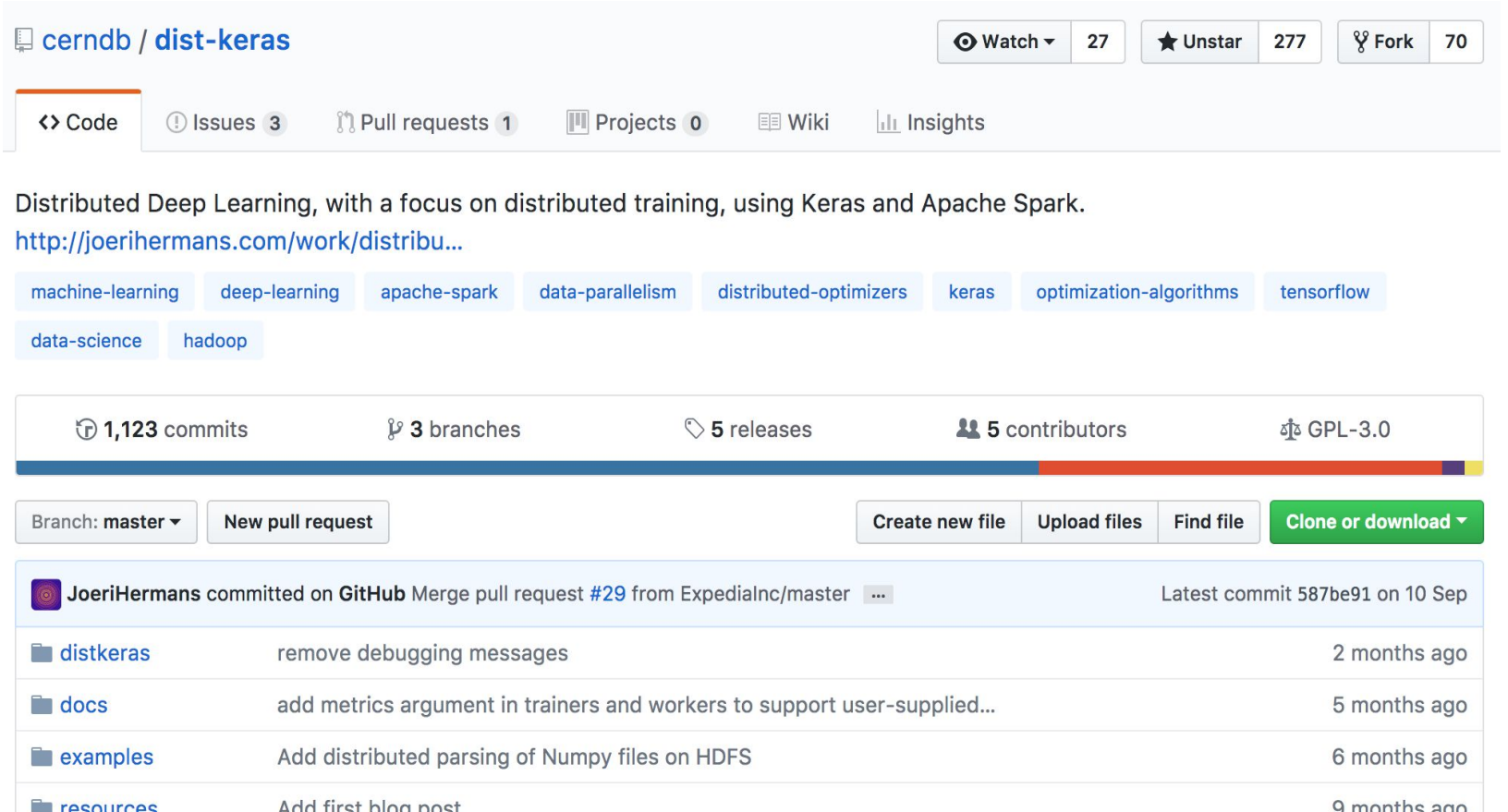
https://deeplearning4j.org/

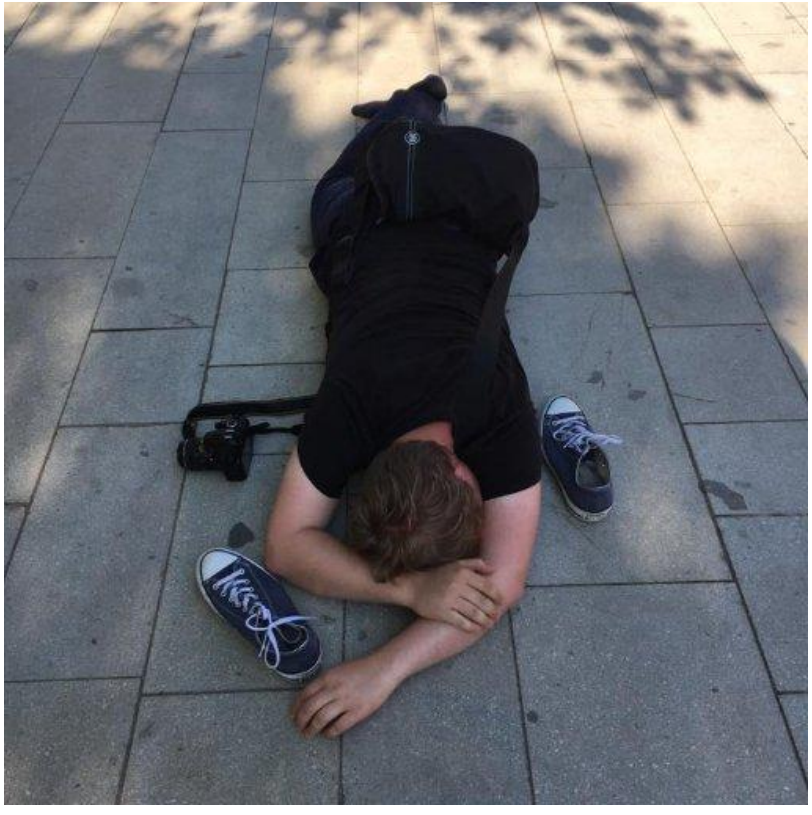
A better alternative would be...



https://github.com/cerndb/dist-keras

DistKeras


cerndb / dist-keras
Watch 27
Unstar 277
Fork 70
Code
Issues 3
Pull requests 1
Projects 0
Wiki
Insights
Distributed Deep Learning, with a focus on distributed training, using Keras and Apache Spark.
http://joerihermans.com/work/distribu...
machine-learning
deep-learning
apache-spark
data-parallelism
distributed-optimizers
keras
optimization-algorithms
tensorflow
data-science
hadoop
1,123 commits
3 branches
5 releases
5 contributors
GPL-3.0
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
JoeriHermans committed on GitHub Merge pull request #29 from Expedialnc/master
Latest commit 587be91 on 10 Sep
distkeras
remove debugging messages
2 months ago
docs
add metrics argument in trainers and workers to support user-supplied...
5 months ago
examples
Add distributed parsing of Numpy files on HDFS
6 months ago



Joeri Hermans + CERN = AWESOMENESS

DistKeras

Well-written

Jupyter Notebooks to get you started

Anyone can understand the code, take a look at it!

...and it works on Spark!

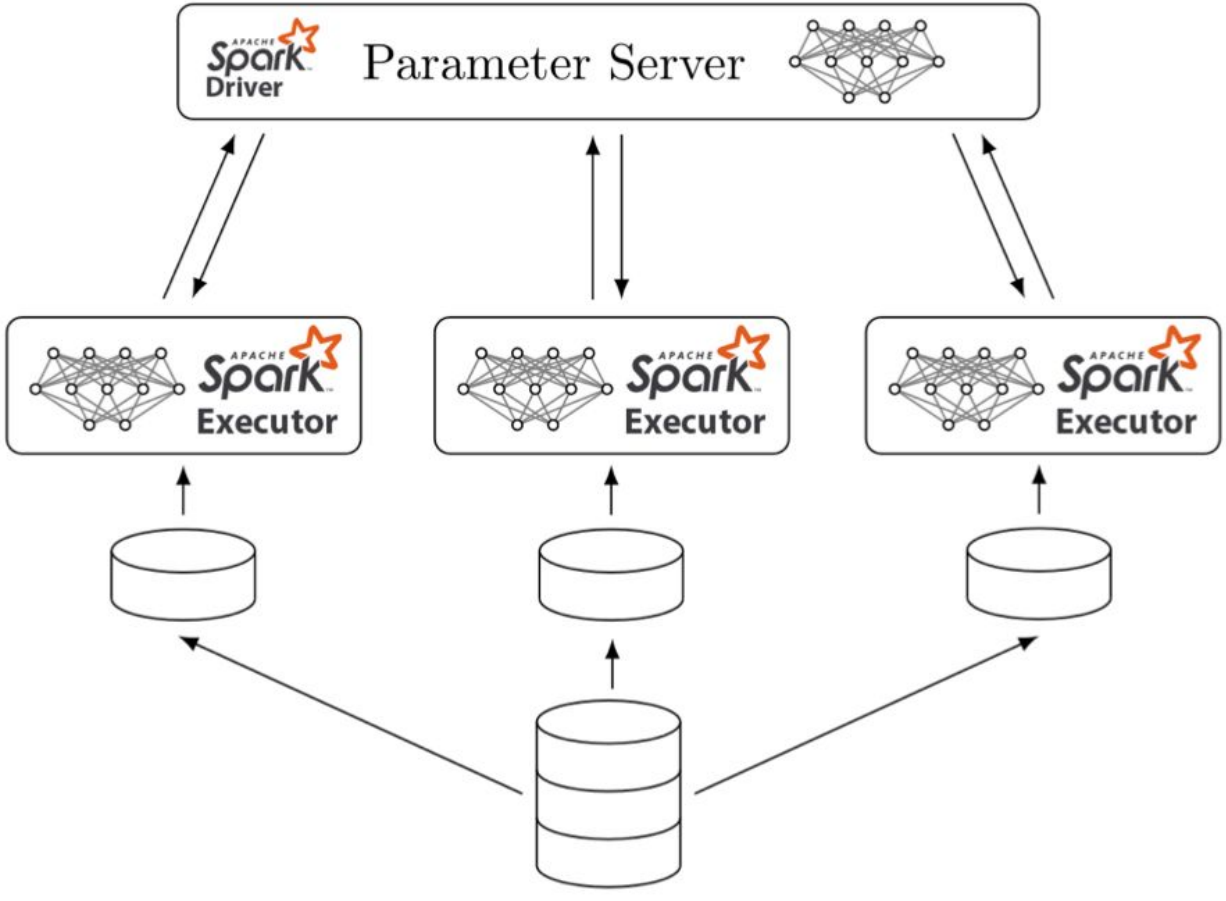
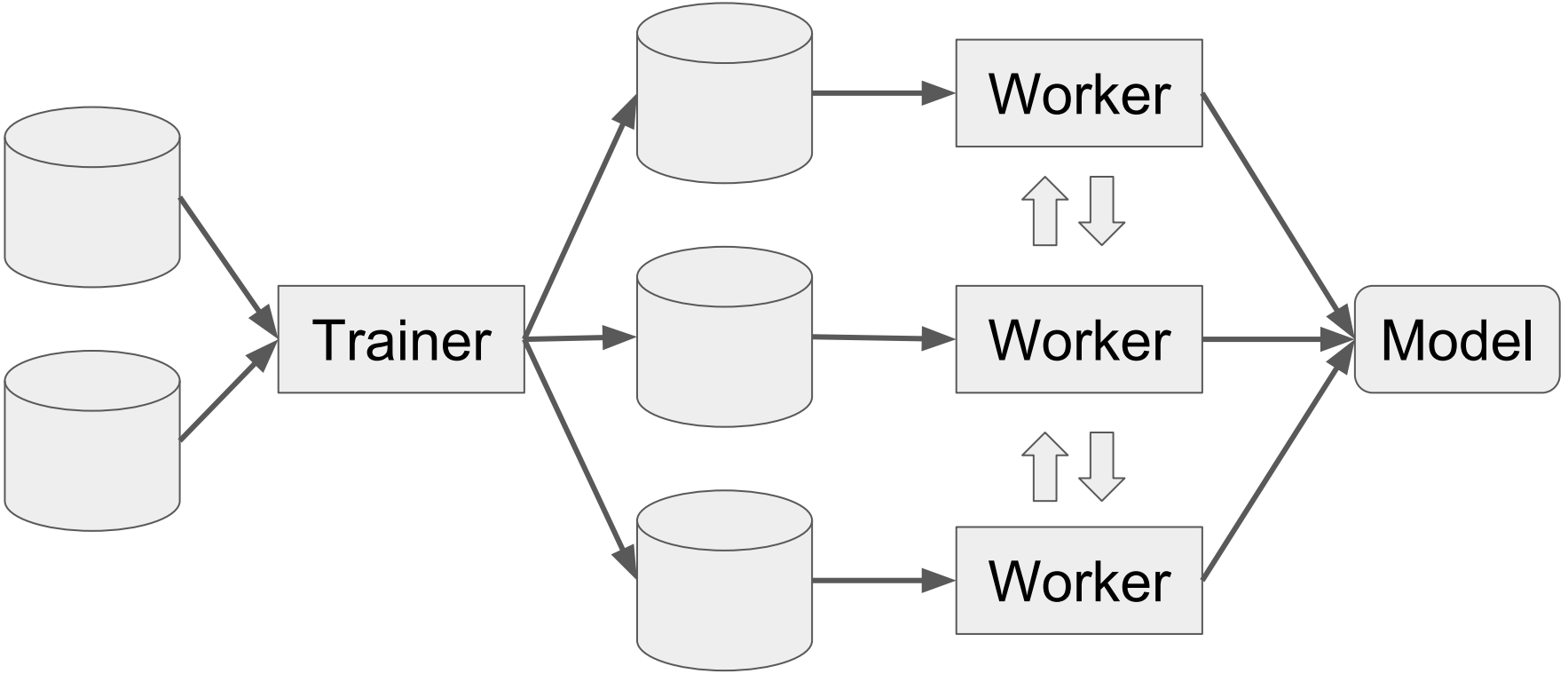


Image: Joeri Hermans, On Scalable Deep Learning and Parallelizing Gradient Descent

DistKeras Architecture



API (AEASGD)

```
trainer = AEASGD(keras_model=model, worker_optimizer=optimizer,
                 loss=loss, num_workers=num_workers,
                 batch_size=BATCH_SIZE,
                 features_col="features", label_col="label",
                 num_epoch=num_epoch, communication_window=32,
                 rho=5.0, learning_rate=0.01)

trainer.set_parallelism_factor(1)
trained_model = trainer.train(training_set)
```

Trainer (AEASGD)

```
parallelism = self.parallelism_factor * self.num_workers
# Check if we need to repartition the dataframe.
if num_partitions >= parallelism:
    dataframe = dataframe.coalesce(parallelism)
else:
    dataframe = dataframe.repartition(parallelism)
# Start the training procedure.
self.record_training_start()
# Iterate through the epochs.
self.history = dataframe.rdd.mapPartitionsWithIndex(worker.train).collect()
# End the training procedure.
self.record_training_end()
# Stop the communication service.
self.stop_service()

return self.parameter_server.get_model()
```

Worker (AEASGD)

```
def optimize(self):
    """Specific training procedure for AEASGD."""
    while True:
        X, Y = self.get_next_minibatch()
        if self.iteration % self.communication_window == 0:
            self.pull()
            W = np.asarray(self.model.get_weights())
            E = self.alpha * (W - self.center_variable)
            W = W - E
            self.model.set_weights(W)
            self.commit(E)
        h = self.model.train_on_batch(X, Y)
        self.add_history(h)
        self.iteration += 1
```

1. Churn Prediction
2. Performance optimization
    a. Background
    b. History
    c. Parallelism Types
    d. Existing Frameworks
    e. Benchmarks

Hardware

Couldn’t use cluster @ Work

Instead, we used AWS

Methods

We used: Averaging, ADAG, AEASGD

Ran on 1, 4, 8 nodes

Experimented with params, settled for defaults

Dataset

Dataset we used is very similar to the one from Vikings

It comes from GDMC 2017
https://cilab.sejong.ac.kr/gdmc2017/

1 Machine (CPU)

c4.4xlarge: 16 cores, 30 GB RAM

Training time: 8251 seconds (~2,3 hours) for 10 epochs

Score: 0.958

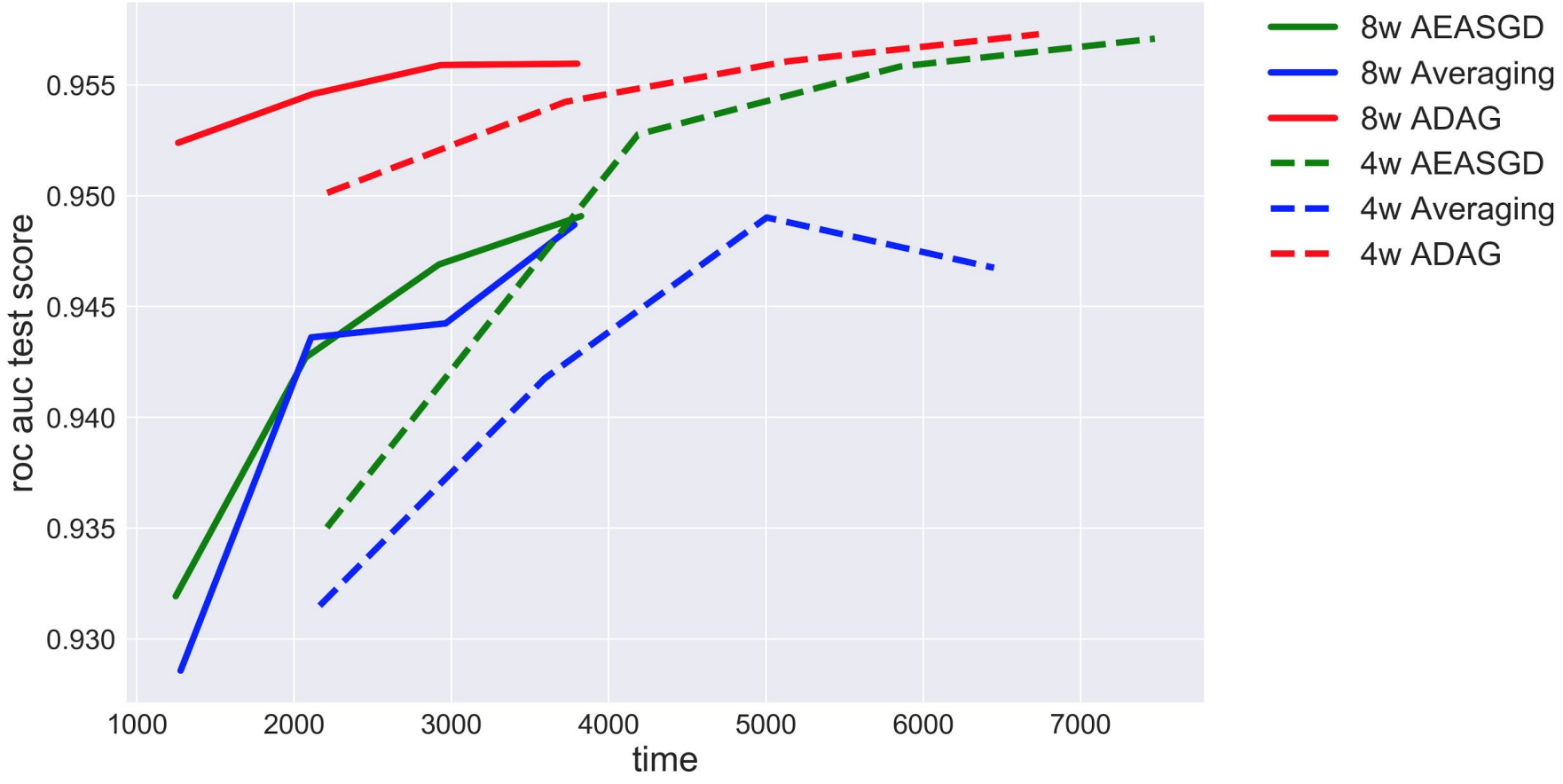
EMR Cluster

m4.xlarge: 4 cores, 16 GB RAM (c4 is twice more expensive)

Played with many configurations

But we show results only for 4 and 8 worker machines

DistKeras: Test Score / Time



1 Machine (GPU)

P2 instances

P3 instances (new!)

Used CUDA 9

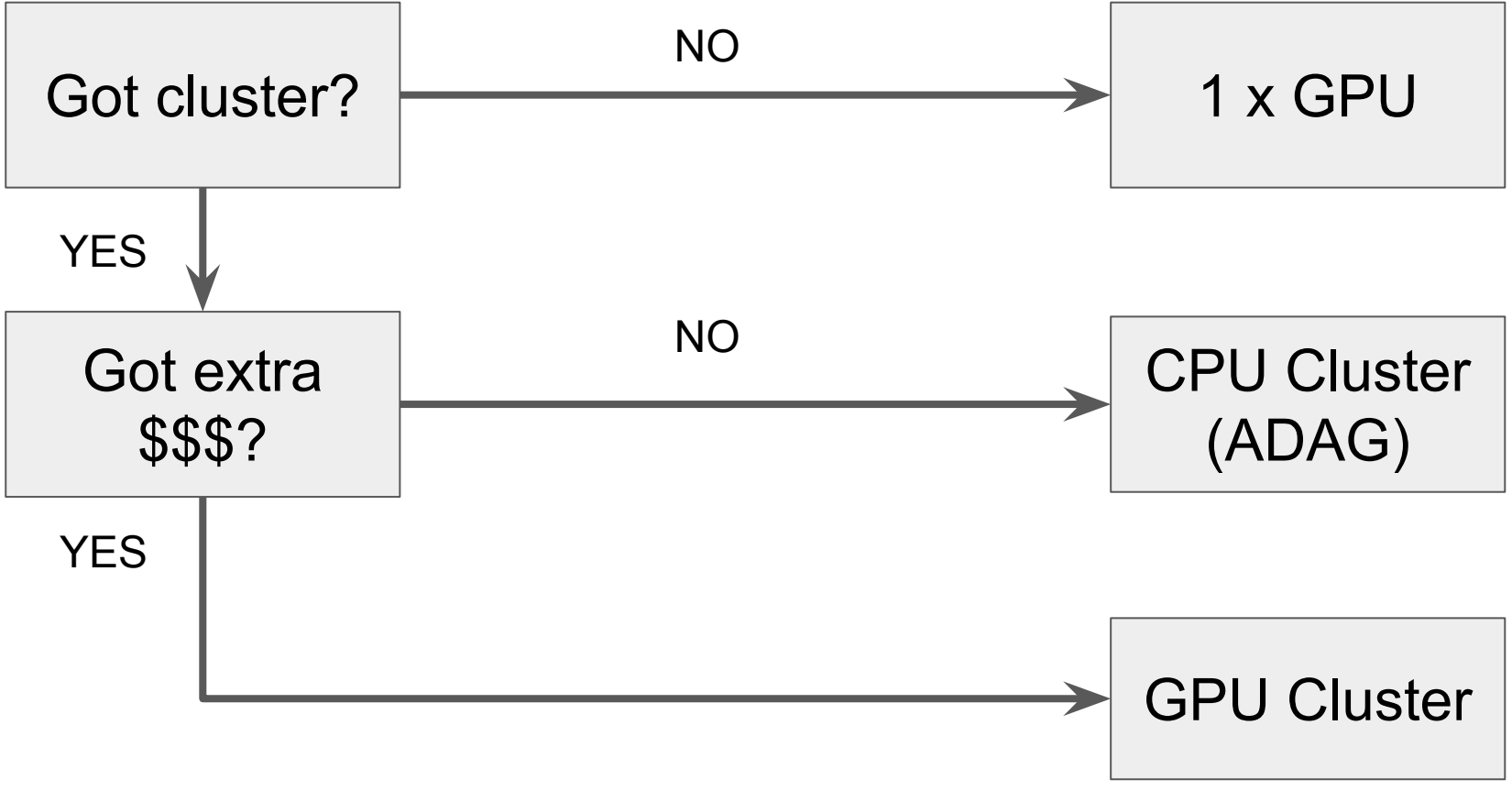
Machine	**Hardware**	**Score**	**Time**
p2.xlarge	Tesla K80	0.957	2780.87
p3.2xlarge	Volta V100	0.957	**2037.63**

Cost Efficiency

Type	Hardware	Price/hour	Score	Time	Cost
CPU	c4.4xlarge	$0.796	0.958	8251	$1.82
EMR	1 + 4 m4.xlarge	5 x $0.26*	0.957	6734**	$2.43
EMR	1 + 8 m4.xlarge	9 x $0.26*	0.956	2933***	**$1.91**
GPU	1 x p2.xlarge	$0.9	0.957	2781	**$0.69**
GPU	1 x p3.2xlarge	$3.06	0.957	2038	$1.73

* price with EMR service cost = 0.06
** 9 epochs to reach this score
*** 7 epochs to reach this score

Take-home Lesson



Recommendations

DistKeras + Spark cluster = Quick Setup

Use asynchronous updates (ADAG!)

Watch for loss explosions

Recommendations

Try Deep Learning for your tasks

Start with good problem definition

Keras is a good starting point

Use Dropout, BatchNorm, modern optimizers (like Adam)

Thank you

Plarium Krasnodar is hiring

- Data Engineer
- Data Scientist

hr.team@plarium.com