Wrocław
University
of Science
and Technology

Large Scale Data Processing

Lecture 6 - Large-scale Machine Learning

Piotr Bielak, dr inż. Tomasz Kajdanowicz

December 21, 2020

HR EXCELLENCE IN RESEARCH

Wrocław University of Science and Technology

Overview

Introduction

Model simplification

Optimization approximation

Overview

Source material

Introduction

This presentation is based on **Wang et al., "A survey on Large-scale Machine Learning"** (https://arxiv.org/pdf/2008.03911.pdf).

Machine learning task setting

Introduction

Given n instances $\mathcal{X} = \{x_1, \ldots, x_n\}$ sampled from a d-dimensional space, and class labels $\{y_i\}_{i=1}^{n_L}$, the goal is to learn an instance-to-label mapping model: $f : \mathbf{x} \rightarrow \mathbf{y}$ from a family of functions $\mathcal{F}$.

Notation

Introduction

$Q(f)_n$ – empirical risk of a model trained over n instances
$Q(f)$ – expected risk ($n \to \infty$)

$f^* = \text{argmin}_f Q(f)$ – optimal solution (may not belong to $\mathcal{F}$)
$f^*_{\mathcal{F}} = \text{argmin}_{f \in \mathcal{F}} Q(f)$ – optimal solution in $\mathcal{F}$

$f_n = \text{argmin}_{f \in \mathbf{F}} Q(f)_n$ – opt. sol. that minimizes empirical risk
$\tilde{f}_n$ – approximation obtained by iterative optimization

Error decomposition

Introduction

Let $T(\mathcal{F}, n, \rho)$ be the computational time for an expected tolerance ρ with $Q(\tilde{f}_n)_n - Q(f_n)_n < \rho$.
We can decompose the execess error ϵ obtained within time cost T_{max}:

$$\text{argmin}_{\mathcal{F},n,\rho} \epsilon_{\text{app}} + \epsilon_{\text{est}} + \epsilon_{\text{opt}}, \text{s.t.} T(\mathcal{F}, n, \rho) \leq T_{\text{max}}$$

Errors vs efficiency

Introduction

Approximation error:

- $\epsilon_{\text{app}} = \mathbb{E}[Q(f^*_{\mathcal{F}}) - Q(f^*)]$
- how closely functions in $\mathcal{F}$ can approximate the optimal solution beyond $\mathcal{F}$,
- size of $\mathcal{F} \rightarrow$ trade-off between ϵ_{app} and computational complexity,

Estimation error:

- $\epsilon_{\text{est}} = \mathbb{E}[Q(f_n) - Q(f^*_{\mathcal{F}})]$
- evaluates the effect of minimizing the empirical risk instead of the expected risk,
- required n for large $\mathcal{F}$ can be huge,
- every instance is traversed at least once,

Optimization error:

- $\epsilon_{\text{opt}} = \mathbb{E}[Q(\tilde{f}_n) - Q(f_n)]$
- measures the impact of the approximate optimization on the generalization performance,
- affiliated with optimization algorithms and processing systems

Aspects of Large-scale Machine Learning

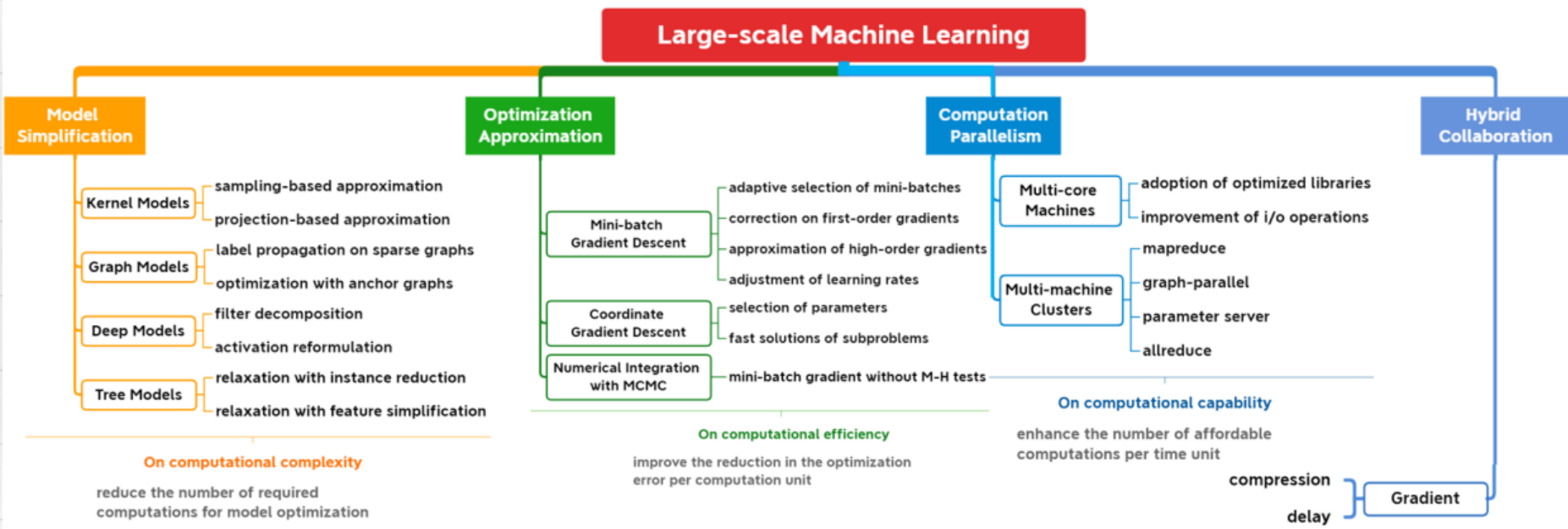
Introduction

We will consider 3 aspects of LML:

- Model simplification
- Optimization approximation
- Computation parallelism

Aspects of Large-scale Machine Learning

Introduction



Overview

Introduction

Model simplification

Optimization approximation

Introduction

Model simplification

This approach improves efficiency from the perspective of **computational complexities**. Domain knowledge can be used for reducing ϵ_{app}.

Categories	Strategies	Representative Methods
Kernel-based Models	sampling-based approximation	uniformly sampling [118], [214], incremental sampling [34], [73].
	projection-based approximation	Gaussian [139], orthonormal transforms [215], ramdom features [134], [161].
Graph-based Models	label propagation on sparse graph	approximate search [110], [227], [232], division and conquer [45], [197].
	optimization with anchor graph	single layer [132], [201], [228], hierarchical layers [77], [200].
Deep Models	filter decomposition	on channels [97], [117], [181], [189], on spatial fields [182], [190], [218].
	activation reformulation	at hidden layers [44], [128], [137], [149], at the output layer [143], [144].
Tree-based Models	relaxation with instance reduction	random sampling [76], sparse-based [52], importance-based [112].
	relaxation with feature simplification	random sampling [36], histogram-based [21], [52], exclusiveness-based [112].

Kernel methods

Model simplification

Recap:

- popular approach in ML,
- kernel-induced Hilbert space $\phi(\mathbf{x})$
- kernel evaluation as dot product between instances: $K_{ij} =< \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) >$

Problem:

Given n instances, the complexity of constructing a kernel matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ is $O(n^2 d)$. Moreover, including class labels $\mathbf{Y} \in \mathbb{R}^{n \times c}$, we model can be "solved" using matrix inversion:

$$(\mathbf{K} + \sigma \mathbf{I})^{-1} \mathbf{Y}$$

This requires a complexity of $O(n^3 + n^2 c)$.

Examples:

Gaussian process, kernel rigde regression, least squares SVM.

Kernel methods

Model simplification

Solution:

Perform **low-rank approximation** based in SPSD sketching models and matrix inversion using Woodbury matrix identity. Let $\mathbf{S} \in \mathbb{R}^{n \times m}$ with $m \leq n$ be the sketching matrix. If we assume that $\mathbf{C} = \mathbf{KS}$ and $\mathbf{W} = \mathbf{S}^\mathbf{T}\mathbf{KS}$, then $\mathbf{C}\mathbf{W}^\dagger\mathbf{C}^\mathbf{T}$ is the low-rank approximation of $\mathbf{K}$ (rank at most m). The model can be now "solved" as:

$$\frac{1}{\sigma}[\mathbf{Y} - \mathbf{C}(\sigma\mathbf{W} + \mathbf{C}^\mathbf{T}\mathbf{C})^{-1}(\mathbf{C}^\mathbf{T}\mathbf{Y})]$$

Such approach reduces the complexity to $O(m^3 + nmc)$.

The crucial point is the choice of S or efficient construction of C.

KM – Sampling based approximation

Model simplification

- **S** – sparse matrix with one nonzero in each column
- sampling columns:
  - at random with replacement (Nyström approximation),
  - K-Means – $O(nmdt)$, where t is the number of iterations,
- incremental sampling – randomly sample columns then pick the column with:
  - the smallest variance of the similarity matrix between sampled columns and the remaining ones,
  - the lowest sum of squared similarity between selected columns
- computation complexity (in general): $O(nmp)$, where p - size of subset

KM – Projection based approximation

- **S** – dense matrix of random linear combinations of all columns of kernel matrices
- e.g., Gaussian distribution to build random projection,
- improvement using orthonormal columns (span uniformly random subspaces)
- e.g., random sample and rescale rows of a orthonormal matrix (Fourier transform, Hadamard matrix)
- complexity: up to $O(n^2 log m)$

Graph-based models

Model simplification

Recap:

- nodes represent instances in dataset,
- edge weights represent similarity,
- classification $\rightarrow$ label smoothness over the graph

Problem:

Given n instances, we construct a graph using $W_{ij} = \text{RBF}(x_i, x_j)$. Graph-based models update labels of nearby instances to be similar and the predicted labels to the ground truth. The soft label matrix $\mathbf{F}$ can be found using:

$$\mathbf{F} = (\mathbf{I} + \alpha\mathbf{L})^{-1}\mathbf{Y}$$

$$\mathbf{L} = \text{diag}(\mathbf{1}^T\mathbf{W}) - \mathbf{W}$$

where $\mathbf{L}$ is the graph Laplacian matrix. Computational complexities are considered by two aspects: **graph construction** $O(n^2d)$ and **matrix inversion** $O(n^3)$.

Solution:

- label propagation on sparse graphs,
- optimization with anchor graphs

GM – label propagation on sparse graphs

Model simplification

- accelerate spread of labels: $\mathbf{F}^{t+1} = \alpha(\mathbf{W})\mathbf{F}^{t} + (1-\alpha)\mathbf{Y}$, $\mathbf{F}^{0} = \mathbf{Y}$
- complexity: $O(nkC)$ vs original $O(n^2C)$ (k - avg. nonzero element in each row of $\mathbf{W}$)
- graph construction $\rightarrow$ hierarchical division of datasets
- complexity: $O(nlog(n)d)$
- e.g., approximate nearest neighbor search (ANNS) – build index using hierarchical trees or hash tables, then search
- this can be repeated and then combined into a single graph

GM – optimization with anchor graphs

Model simplification

- sample m instances from dataset as **anchors**
- compute similarity between all instances and anchors $\mathbf{Z} \in \mathbb{R}^{n \times m}$
- predictions are inferred from these anchors (few parameters),
- soft labels: $\mathbf{F} = \mathbf{Z}(\mathbf{Z}^T\mathbf{Z} + \alpha\tilde{\mathbf{L}})^{-1}\mathbf{Z}^T\mathbf{Y}$, where $\tilde{\mathbf{L}} \in \mathbb{R}^{m \times m}$ – reduced Laplacian over anchors
- complexity: graph construction reduced to $O(nmd)$ or even $O(ndlog(m))$ (ANNS)
- total complexity: $O(nm^2 + m^3)$

Deep models

Model simplification

Recap:

- FCN (fully connected) are costly,
- usage of CNN and RNN to obtain local receptive fields,
- this section covers only a subset of approaches for deep learning models

Problem:

Let's consider a CNN model, where a convolution can be treated as a 4D tensor. Suppose input feature have a size of $d_I \times d_I$ and there are m_I channels. A convolutional layer can be parameterized by kernels of size $d_K \times d_K \times m_I \times m_O$, where d_K is the kernel size and m_O is the number of output channels. Assume that stride is 1. Finally, the computational cost becomes:

$$d_I \times d_I \times d_K \times d_K \times m_I \times m_O$$

Although the filter (convolution) sizes are small (and hence there are not many parameters), deep models often have many layers. This can lead to CNN models with millions of parameters.

Solution:

- filter decomposition
- activation reformulation

DM – filter decomposition

Model simplification

- intuition: there is a significant amount of redundancy in 4D tensors (convolutions)
- gathering information from different channels hierarchically
- **AlexNet** – group convolutions
- **MobileNet** – separable depthwise convolutions (70% reduction)
- **Inception v1** – dim. reduction on channels of input features + filters with original receptive fields
- **ShuffleNet** – generalization of group convolutions and depthwise convolutions; based on channel shuffling
- **VGG** – replace one layer of large-size filters with two layers of smaller-sized filters (28% reduction)
- **Inception v3** – asymmetric convolutions $\rightarrow$ decompose 3x3 filters into two cascaded 1x3 and 3x1 (33% less parameters)
- **Dilated convolution** – exponential expansion of receptive fields (without loss of resolution), e.g., 7x7 replaced by two dilated 3x3 (80% less computations)

DM – activation reformulation

Model simplification

- sigmoid and tanh cause expensive exponential operations at each neuron,
- these suffer also from the vanishing gradient problem,
- a solution might be: **ReLU** = $\max(0, x_{\text{input}})$
- when dealing with many inactive ReLU neurons, one can use **leakyReLU**
- to ensure training stability one can use **bounded ReLU and leakyReLU**,
- for output neurons using classical softmax can be expensive (normalization factor)
- use **hierarchical softmax**

Tree models

Model simplification

Recap:

- such models build trees by recursively splitting instances at each node using some decision rules, e.g., Gini Index or entropy
- random forests (RF) – each tree trained independently
- gradient boosting decision trees (GBDT) – covered last lecture

Problem:

Given n instances with d features, finding the best split for each tree node leads to a complexity of $O(nd)$ (check every instance and every feature).

Solution:

Relaxation of decision rules performed from the perspective of **instances** and **features**

TM – relaxation with instance reduction

- simplest approach: instance sampling while growing trees
- applicable for both RF and GBDT
- for sparse features: ignore invalid instances when evaluating splits (linear complexity with respect to non-missing instances)
- for GBDT only: prefer instances with larger gradients

TM – relaxation with feature simplification

- feature sampling (consider subset of features at each split node)
- histogram-based boosting (group features into a few bind using quantile sketches in either global or local manner; then split based on bins)

Wrocław University of Science and Technology

Overview

Introduction

Model simplification

Optimization approximation

Introduction

Optimization requires:

- multiple iterations
- gradients computation
- convergence

Optimization approximation is one of Large Machine Learning approaches to scale up machine learning from the perspective of computational efficiency.
While approximating the optimization:

- same as in general case – gradients are calculated in each iteration
- compute them over a few instances
- or compute them over few parameters

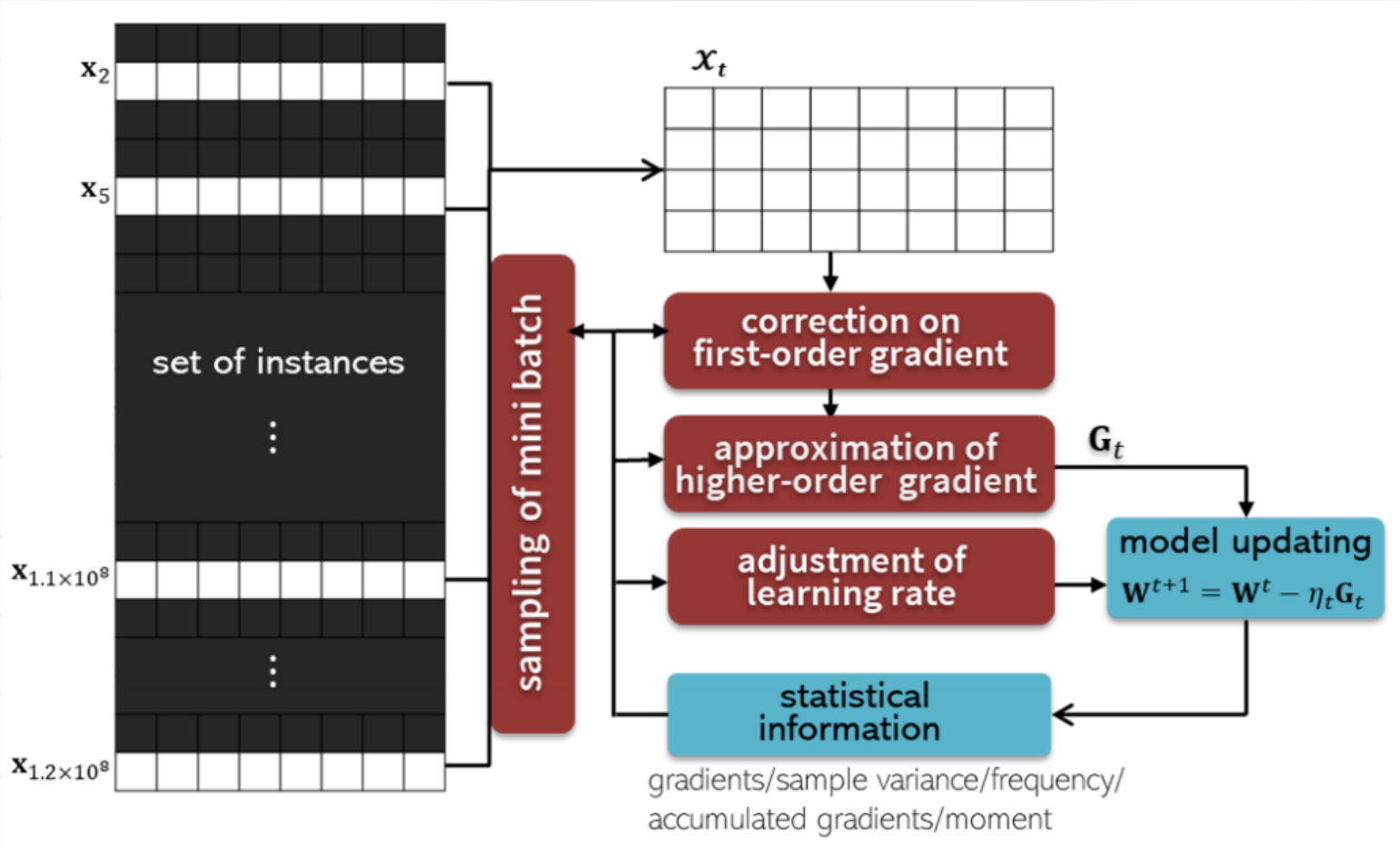
Features, problems, solutions

Approximate optimization features:

1. increase the reduction in the optimization error per computation unit
2. obtain an approximate solution with fewer computations

Main problem: guarantee the convergence.
Solutions:

- mini-batch gradient descent
- coordinate gradient descent
- numerical integration based on Markov chain Monte Carlo


$\mathbf{x}_2$
$\mathbf{x}_5$
set of instances
$\mathbf{x}_{1.1\times10^8}$
$\mathbf{x}_{1.2\times10^8}$
sampling of mini batch
x_t
correction on first-order gradient
approximation of higher-order gradient
$\mathbf{G}_t$
adjustment of learning rate
model updating
$\mathbf{W}^{t+1} = \mathbf{W}^t - \eta_t \mathbf{G}_t$
statistical information
gradients/sample variance/frequency/
accumulated gradients/moment

Mini-batch Gradient Descent (MGD)

Aim:

- solve the problems with a modest number of parameters
- but a large number of instances

Compared with stochastic gradient descent:

- MGD utilizes better gradients estimated over more instances per iteration
- generally obtains fast local convergence with lower variances

Mini-batch Gradient Descent (MGD) - formulation

$\mathcal{X}_t$ - mini-batch og instances with the size of m_f
Q - objective function built upon the parameter matrix W
Let $\partial Q(W; x_i)$ be stochastic gradient on x_i and
$G_t = \frac{1}{m_t} \sum_{i \in \mathcal{X}_t} \partial Q(W^t; x_i)$ indicates the aggregated stochastic gradient on $\mathcal{X}_t$

$$W^{t+1} = W^t - \eta G_t$$

with a learning rate of η.
For large-scale datasets:

- updating parameters based a few instances leads to large variances of gradients
- makes optimization unstable

We can estimate gradients with large and fixed batch sizes but it remarkably increases per-iteration costs.

Adaptive sampling of mini-batches

- take into account both the sizes of mini-batches and the sampling of instances
- start with small batches and increase the sizes gradually via a *prescribed sequence*, e.g. linear scaling
- consider both data distributions and gradient contributions, e.g. sampling weights of instances to be proportional to the L2 norm of their gradients
- maintain a distribution over bins and learning the distribution per t iterations

Correction of first-order gradients

Aim:

- improve the quality of search directions with lower variances
- enable a larger learning rate for accelerations

Solution:

- Gradients descent with Nesterov momentum - performs a simple step towards the direction of the previous gradient and then estimates the gradient
- SAG utilizes the average of its gradients over time to reduce the variances of current gradients
- SVRG develops a memory-efficient version which only needs to reserve the scalars to constrict the gradients at subsequent iterations

Approximation of higher-order gradients

Aim:

- when the condition numbers of objective functions become larger, the optimization can be extremely hard

Solution:

- approximation of second-order information with successive re-scaling
- use conjugate gradient algorithms to estimate Hessian matrices (can be noisy - mini-batch size is small)
- mini-batch-based Quasi-Newton algorithm: L-BFGS to approximate the inverse of Hessian based on the latest parameters and a few gradients at previous mini-batches results

Adjustment of learning rates

Aim:

- keep convergence while playing with learning rate
- make not to small rate (slows down the convergence) and not to large rate (can hinder convergence and cause the objective function to fluctuate around its minimum)

Solution:

- the rate decayed by the number of iterations
- *Adagrad* prefers smaller rates for the parameters associated with frequently occurring dimensions and larger rates for the ones associated with infrequent dimensions
- *Adadelta* takes account of the decaying average over past squared gradients
- learning rates can also be updated based on the second moment: *Adam* - prefers flat minima in error surfaces

Experimental comparison of Mini-batch Gradient Descent techniques

- adaptive sampling of mini-batch can clearly improve computational efficiency
- doubling batch sizes during training, reduced the time cost from 45 mins to 30 mins on ImageNet
- introduced adaptive sampling and only used 30% epochs obtained the same accuracy of uniform sampling
- using the second-order information achieved 6x to 35x faster convergence
- taking the frequency of parameters and the decaying of gradients into account for learning rates, Adagrad and Adadelta could only use 20% epochs to achieve the same result of normal optimizers
- in Adam, adaptive moment estimation further reduced more than 50% computations

Wrocław University of Science and Technology

Large Scale Data Processing

Lecture 6 - Large-scale Machine Learning

Piotr Bielak, dr inż. Tomasz Kajdanowicz

December 21, 2020