Technische Universität Braunschweig

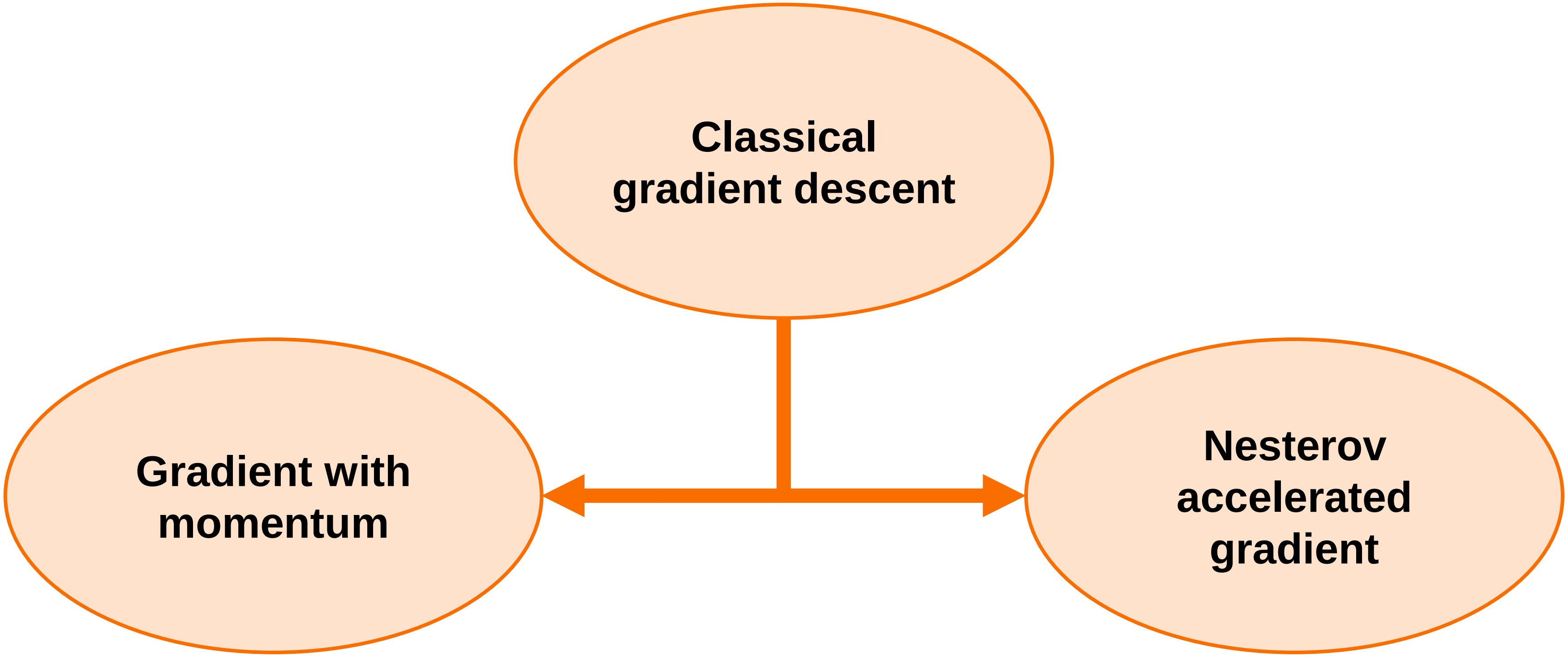
Machine learning

Mini-project: Gradient Descent with Momentum and Nesterov-Accelerated Gradient

Noah Gollnick

Wirtschaftsinformatik

Introduction



Main goal: Accelerate the machine learning process

Learning-rate-Dilemma in classic Gradient descent

High learning rate ⟺ **Low learning rate**

- Faster training process, smaller number of iterations for best result
- Higher probability of overshooting and oscillating around the minimum

- Slower training process, higher number of iterations for best result
- Smaller probability of overshooting, higher chance to find the perfect minimum

Approach of using momentum

Possible Approach: Dynamic learning rate

Assumption 1:
The risk of overshooting is higher at the end of the training process

Assumption 2:
The loss is getting generally better over time, so the steps are getting smaller

Learning algorithm adjusts to the size of the last gradient descent step

Concept and Computation of Momentum

Momentum: Adding a velocity vector to the changes made in one machine learning step. The velocity equals the difference between the parameters of the last 2 steps:

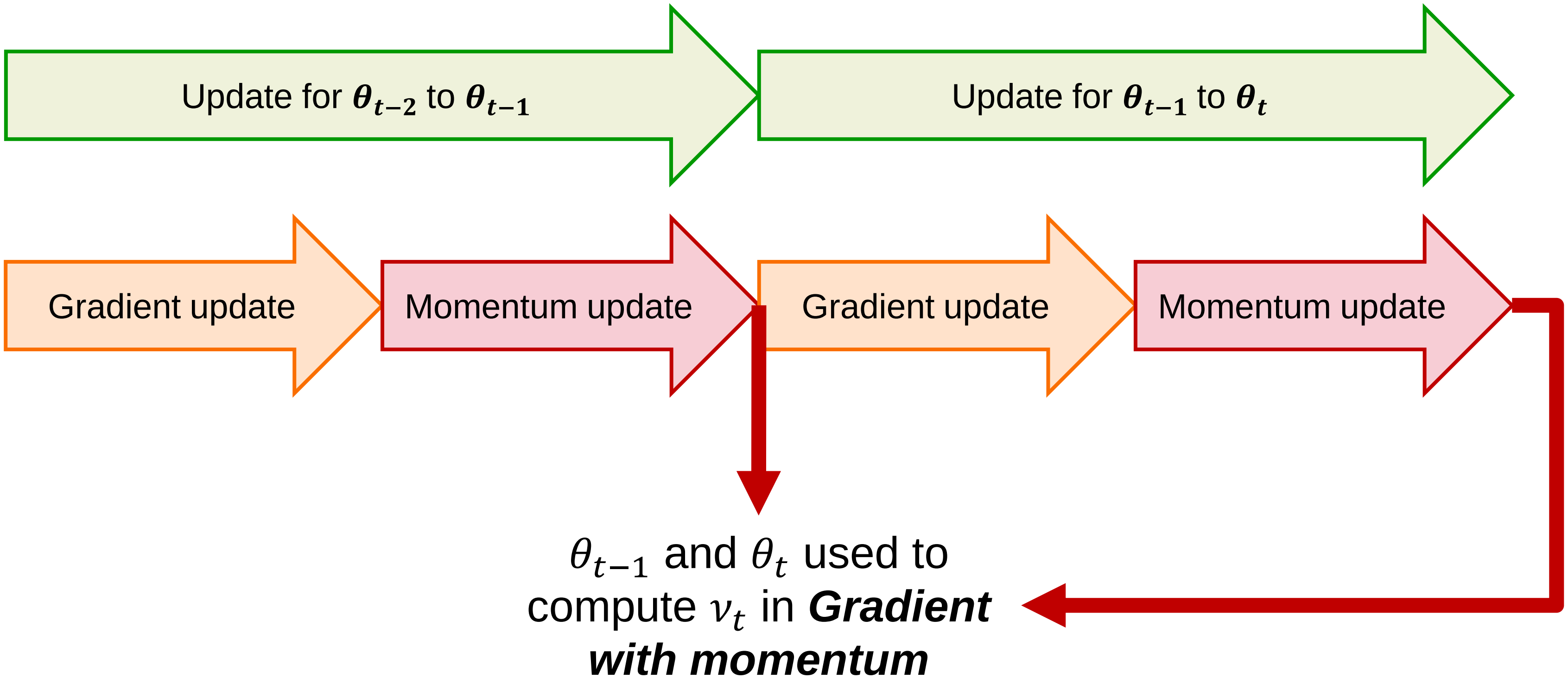
$$\boldsymbol{\nu_t = \theta_t - \theta_{t-1}}$$

The velocity vector is added to the next step (multiplicated with a scalar factor) while subtracting the gradient:

$$\boldsymbol{\theta_{t+1} = \theta_t - \varepsilon \Delta f(\theta_t) + \mu \nu_t}$$

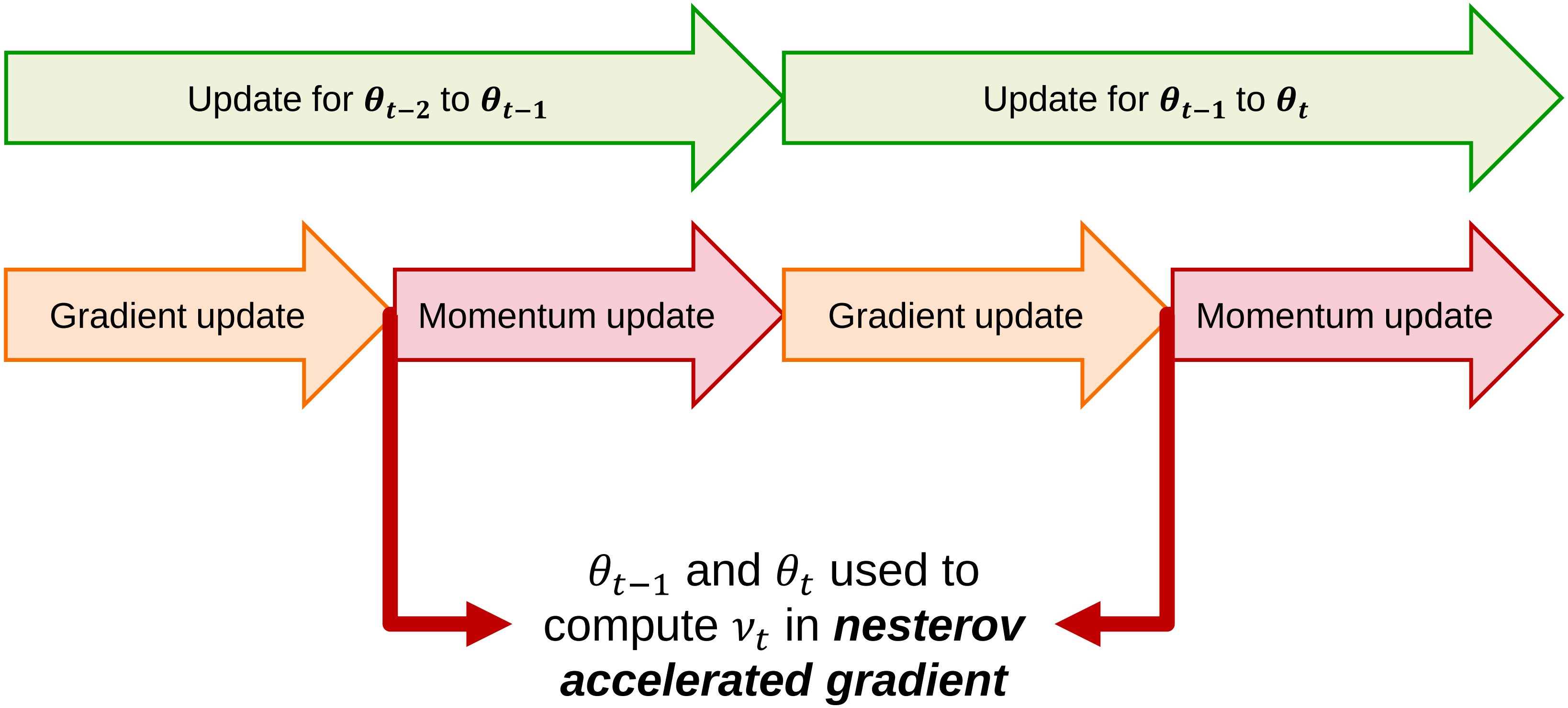
Difference between GDM and NAG

Difference between classical Gradient with momentum and Nesterov accelerated gradient:
NAG splits up gradient update step and momentum update step



Difference between GDM and NAG

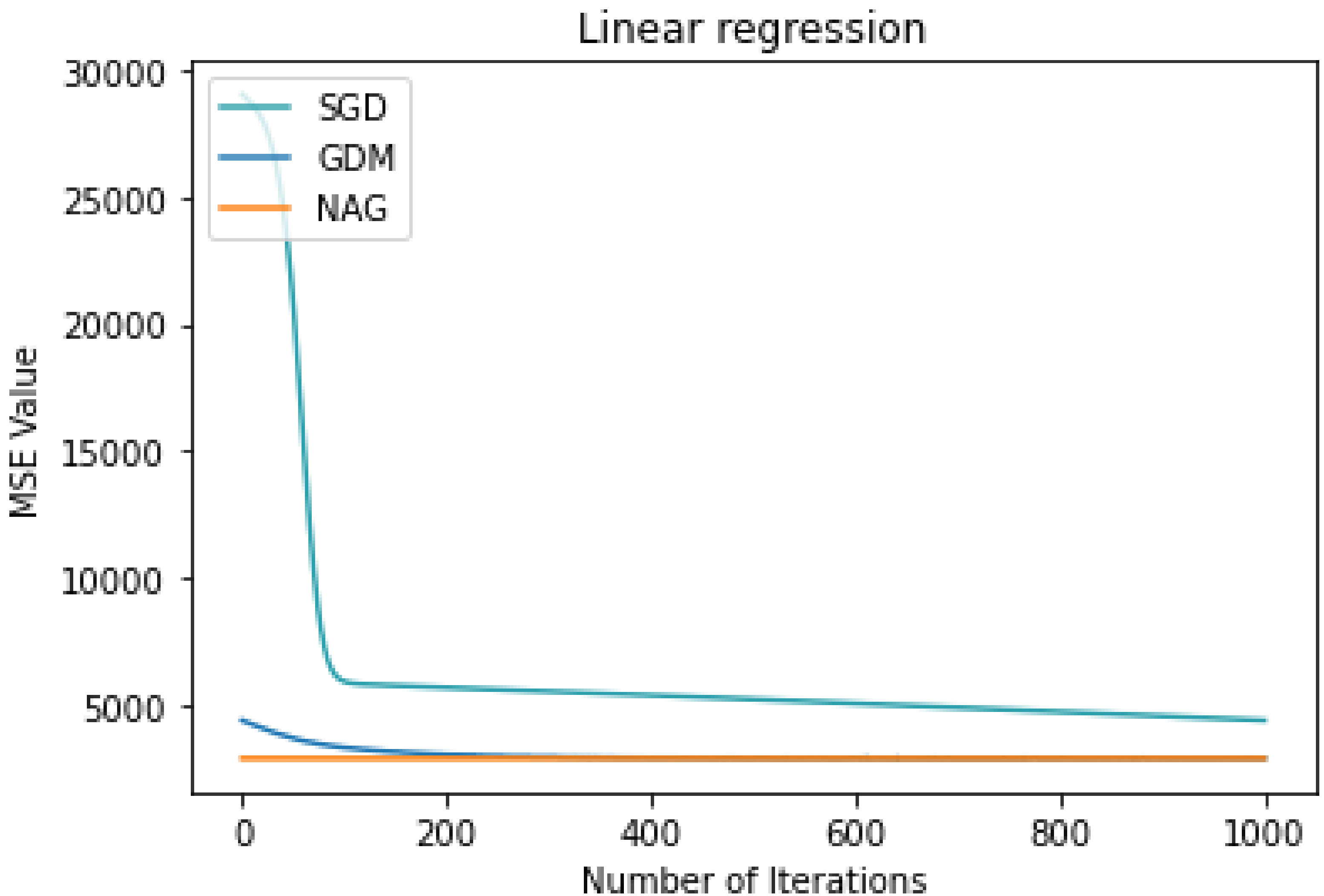
Difference between classical Gradient with momentum and Nesterov accelerated gradient:
NAG splits up gradient update step and momentum update step



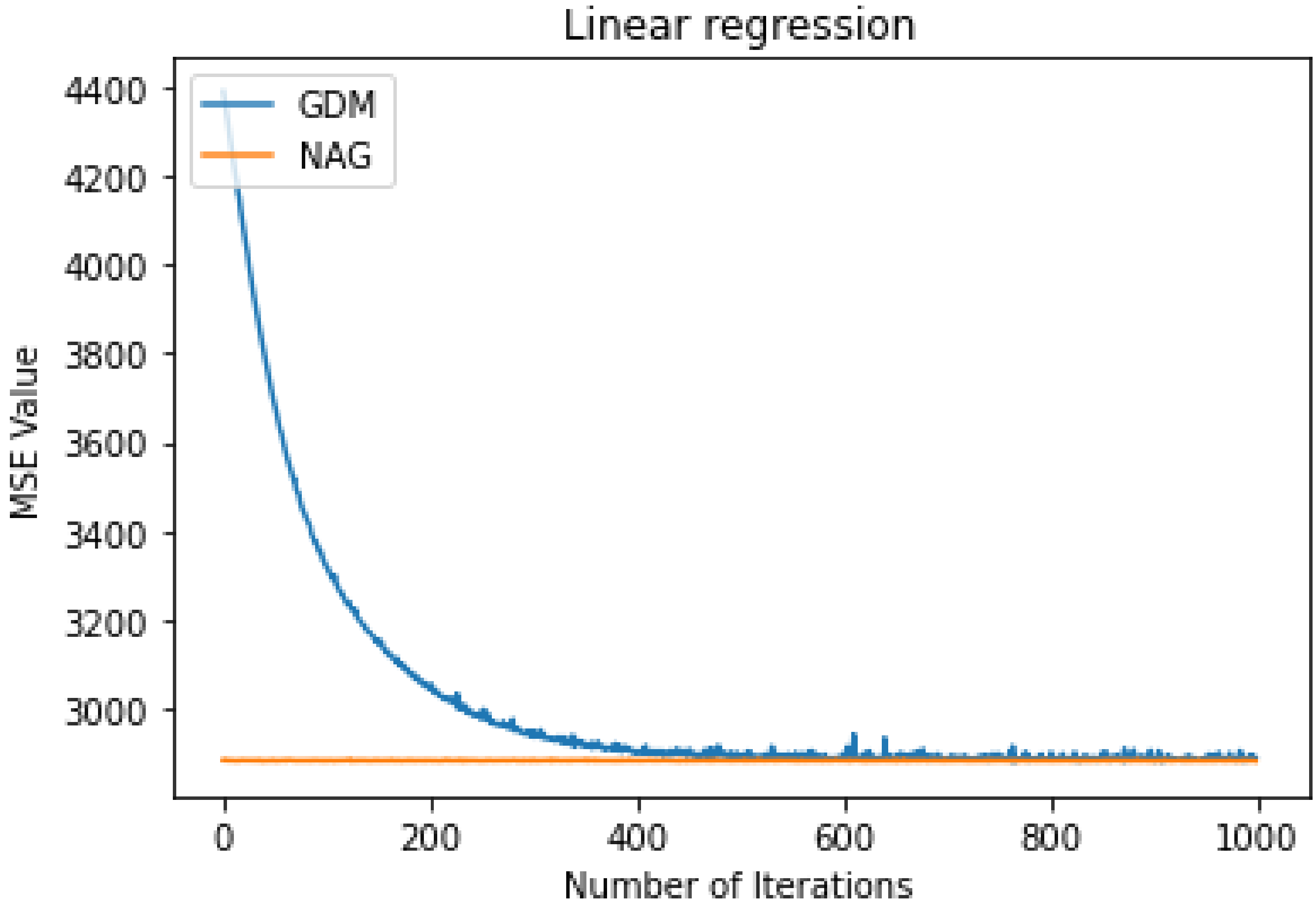
Comparing Methods for Linear Regression

Tested Model:	Linear Regression
Dataset:	Diabetes Dataset (Scikit-Learn)
Number of Samples:	422
Number of Features	10
Loss function:	Mean Squared Error
Learning Rate:	0.00001
Momentum rate (if Momentum):	0.9
Number of iterations:	1000
Random Seed:	57

Comparing Methods for Linear Regression



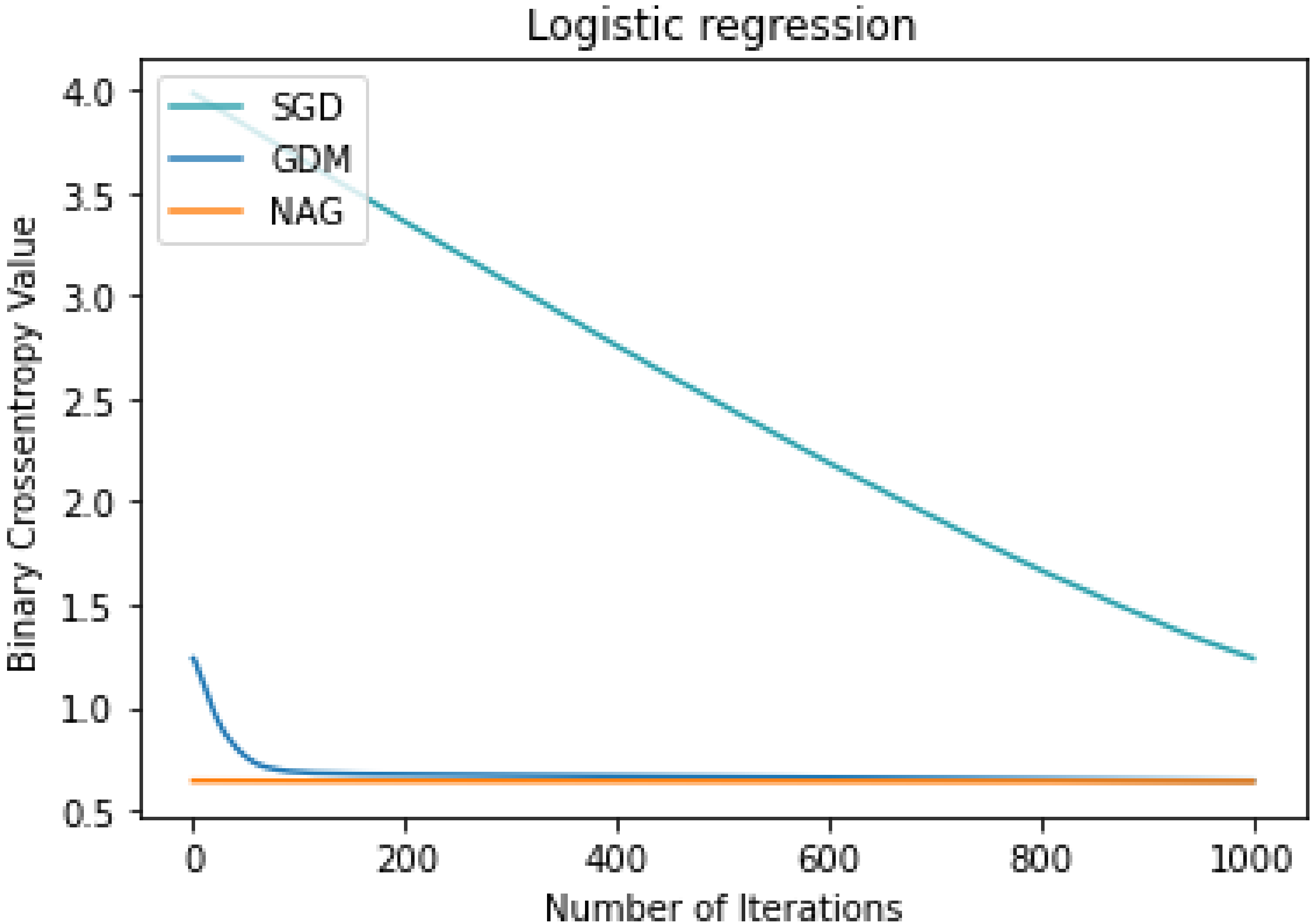
Comparing Methods for Linear Regression



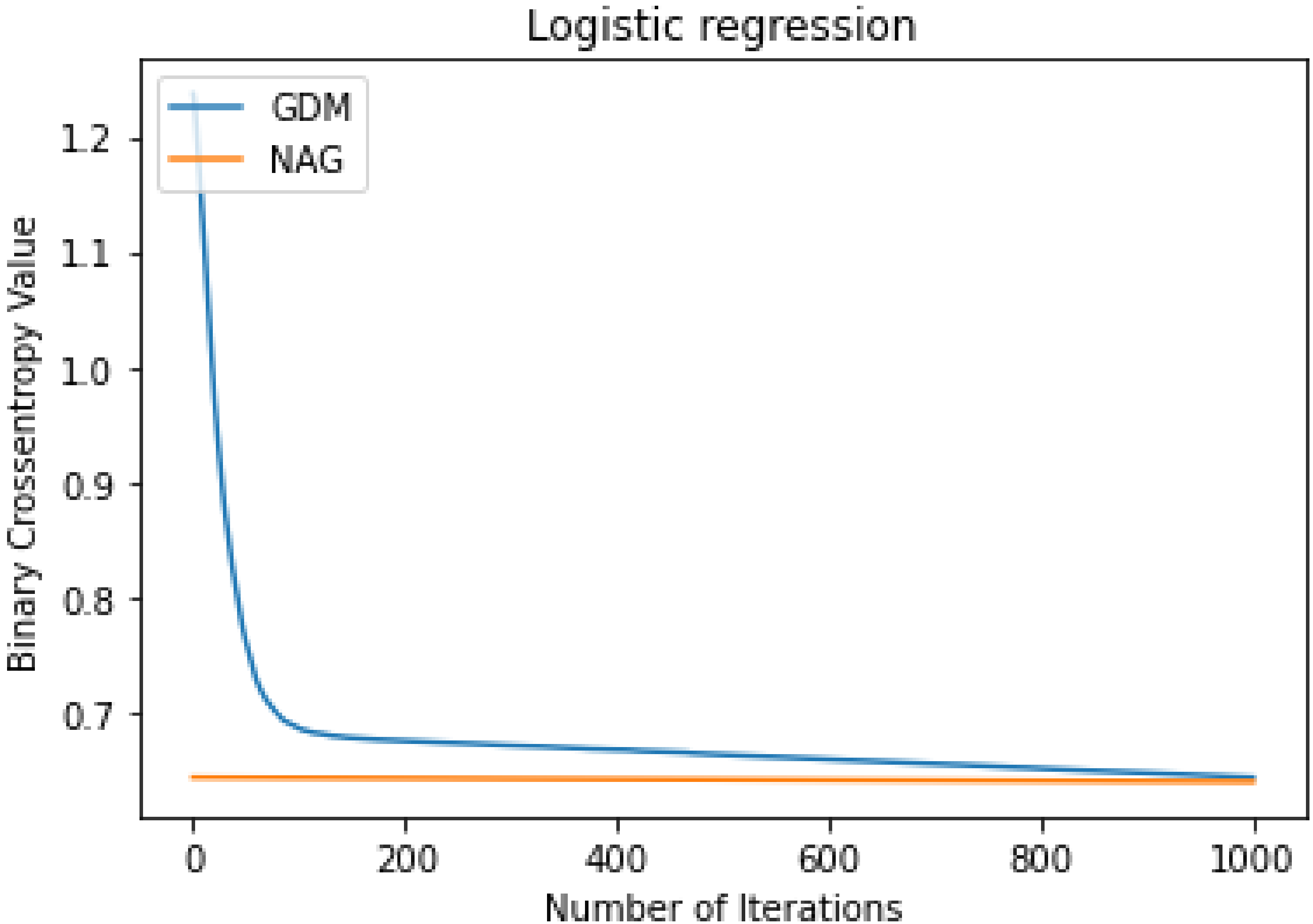
Comparing Methods for Linear Regression

Tested Model:	Logistic Regression
Dataset:	Iris Dataset (Scikit-Learn)
Number of Samples:	100
Number of Features	4
Loss function:	Binary Crossentropy
Learning Rate:	0.001
Momentum rate (if Momentum):	0.9
Number of iterations:	1000
Random Seed:	57

Comparing Methods for Logistic Regression



Comparing Methods for Logistic Regression



Library Implementations for Sklearn/Pytorch/Keras

Scikit-learn:

No Explicit Bulit-in Implementation

Theano/Lasagne:

- Function sgd for gradient descent
- Function apply_momentum for momentum
- Function apply_nesterov_momentum for NAG

Pytorch:

- Class SGD for gradient descent
- momentum as float argument
- nesterov as boolean argument

Tensorflow/Keras:

- Class SGD for gradient descent
- momentum as float argument
- nesterov as boolean argument

Conclusion

Momentum and NAG does not give you better results, but it give you a good result much faster!

You might use it when:

- …the prediction just have to be good, but not perfect
- …fast Training process is required

For Example: model is trained live while usage multiple times

You might not use it when:

- …there is a high requirement for the best accuracy possible
- …time is no important matter

For example: model is only trained once without gathering new data

References

http://www.cs.utoronto.ca/~ilya/pubs/ilya_sutskever_phd_thesis.pdf

https://jlmelville.github.io/mize/nesterov.html

https://scikit-learn.org/stable/index.html

https://github.com/Lasagne/Lasagne

https://pytorch.org/

https://keras.io/