An Overview:

Classical and Quantum Optimizer

by Tjark Ziehm (Dez. 2021)

Fachhochschule Kiel
University of Applied Sciences

Introduction

Thanks to Mustafa

Related Document: [Visualization4Costs]

Optimization is a technology in machine learning to get the “best” possible result for a parameter. Important to know is the dual usability for classical, hybrid or quantum learning models (computation). The basic idea behind is to find the minimal or optimal solution for a defined problem or way.

The concept of optimization like in a beginner calculus course. It’s solving for a maximum or minimum of a given function.

“Since variational algorithms have shown a lot of promise in terms of near-term, NISQ computations, it’s important to find better ways to optimize them.” - Lana Bozanic

The gradient descent is the Slope or rise

Table of contents

SGD

Comparison:
SDG,Momentum,NAG,AdaGrad,AdaDelta,RMSprop Compare

What are Optimizer?

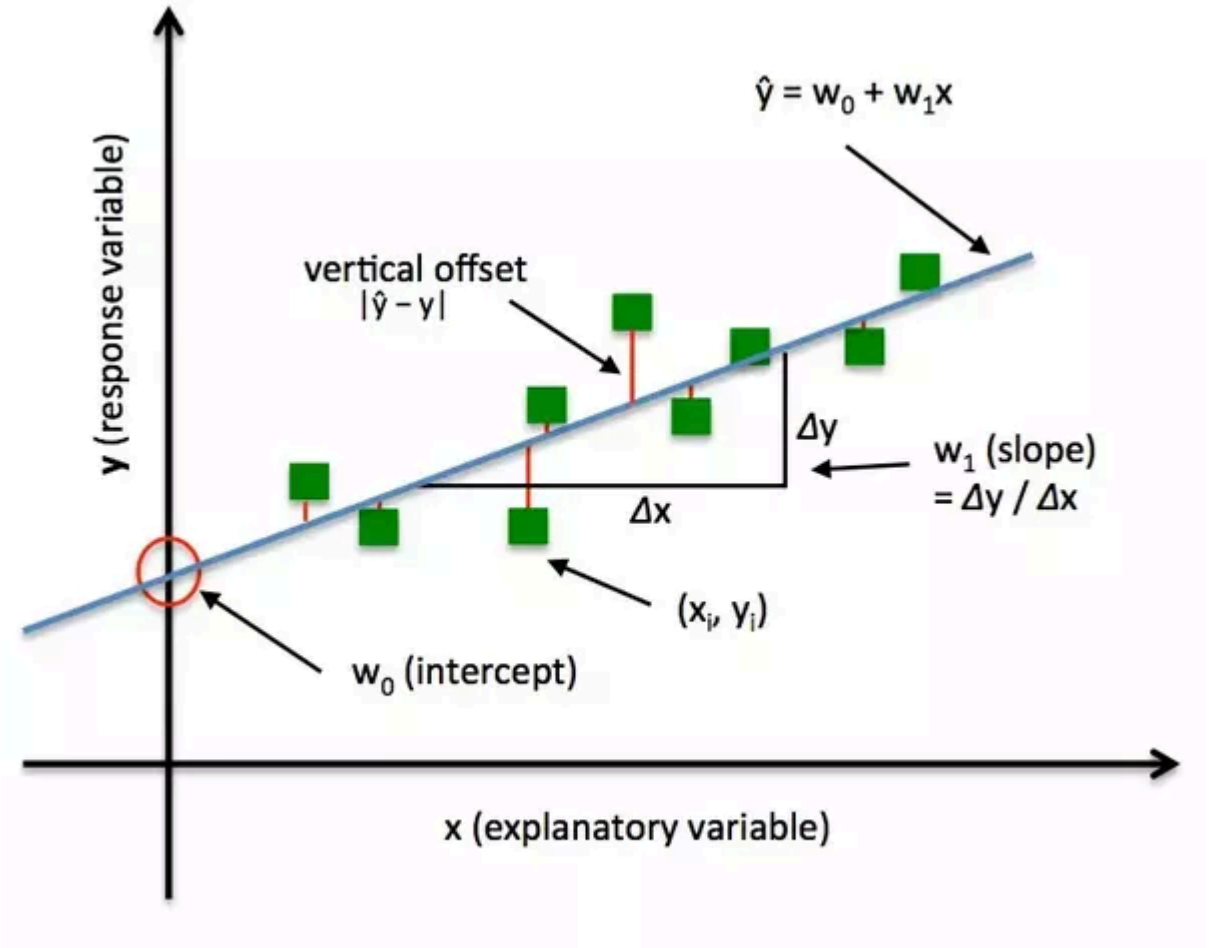
Optimizers are the part of the calculation which are able to change the parameters from a Quantum-, Classical- or Hybrid-System of the Machine Learning. The Optimizer Algorithm will change this parameter automatically and can act in different access points of the calculation.
The Core function of an optimizer is to find the optimal parameters with the minimal cost for the parameter calculation.

This Function is used to minimize an error function called “loss function” and for maximize the efficiency of production. If you don't know about the “loss function” yet, please read the chapter “Loss Function” now.

Depending on the model the optimizer is a mathematical function with learnable parameters like the “Weight” and “Bias”

Generally said the optimizer tries to predict the result and find the positive or negative change in comparison with the previous prediction. This will give a difference in the parameters which the optimizer will now compare and improve.
The Mathematical view to this is *abs(y_predicted – y). The first prediction has no important influence if it is too high or too low. The Result from this in comparison is the main information. (For some algorithms this could be a significant feature !)*

Here a example as a linear explainable correlation between explanatory variable and response variable:



Mean squared error

The MSE is the main part most algorithms with the loss function.
To calculate MSE, you take the difference between your predictions and the ground truth, square it, and average it out across the whole dataset.

Example

@source [LINK]

y: the actual prediction from the calculation
y_prediction: the result with the parameter which gives us the way to optimize
**2: Using the squared difference allow us to compare negative values like an amount of a value
sum_squared_error:

```
def MSE(y_predicted, y):
          squared_error = (y_predicted - y) ** 2
```

```
        sum_squared_error = np.sum(squared_error)
        mse = sum_squared_error / y.size
    return(mse)
```

Likelihood loss

A commonly used function for classification problems.This Function uses as base the probability for each input example and multiplies them. This method is helpfull to compare models
“The likelihood function (often simply likelihood), occasionally called the plausibility function or the conjecture function,[1] is a special real-valued function in mathematical statistics that is obtained from a probability density function or a count density by treating a parameter of the density as a variable. Central use of the likelihood function is the construction of estimator functions by the maximum likelihood method. In addition, other functions such as the log-likelihood function and the score function are derived from it, which are used, for example, as auxiliary functions in the maximum likelihood method or for the construction of optimality criteria in estimation theory.” - wikipedia [LINK]

Example:
Outputs probabilities of [0.4, 0.6, 0.9, 0.1] for the ground truth labels of [0, 1, 1, 0]

Log loss (cross entropy loss)

Like the Likelihood-Function mainly used for binary classification with the mathematical definition:

$$-(y\log(p)+(1-y)\log(1-p))$$

This is exactly the same formula as the regular likelihood function, but with logarithms added in it.

@article [LINK]

Summary

On one side the loss function is a static way for the static representation for the performance of you model. Additionally it shows you how the data of your model fit. Each learning machine learning algorithm is using a loss function in the process of optimization or finding the best parameters (weights) for the dataset.

In linear regression the MSE is used to find for each set of data the related “cost”. The optimizer algorithm model (as example gradient descent) then optimize the MSE functions to make it “lowest” costs possible

More Accuracy with Optimizer

Additional for the model accuracy the optimizer should have a highly important meaning for our works in machine learning.

Differences in QML and classical maschine learning

@paper [Quantum Natural Gradient Descent]

The difference of the QML (QNGD) is the Usage of the parame trized quantum space with more efficiency for optimization.

The basis Gradient Descent is writen like this:

$$\theta_{n+1} = \theta_n - \eta \nabla \mathcal{L}(\theta)$$

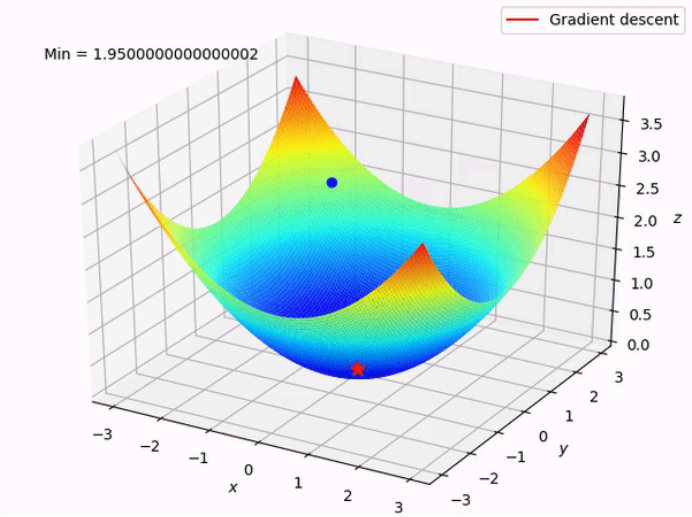
parameter:

θ_(n+1) new parameter
θ_n current parameter

η “step-size” [movement of the parameter]

∇L(θ) the gradient in respect to our cost function

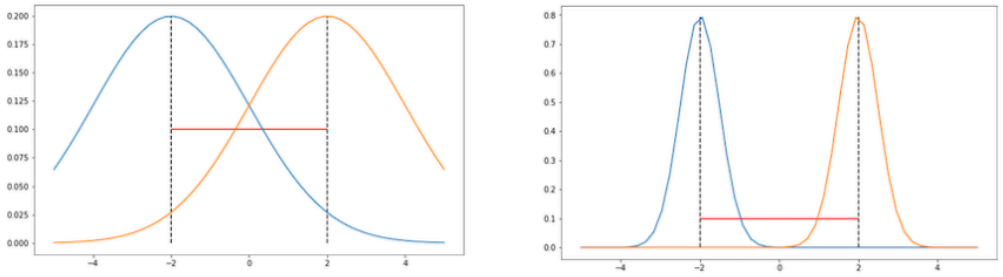
Note: We subtract from the actual parameter the gradient multiplied with steps sizes.
The Gradient gives the direction the steepest ascent. This is the reason why the negative version is needed to get to the minimum (steepest descent) instead of the given direction of the steepest ascent. The general usage of gradient descent is to optimize the parameters to move to the minima and reduce the costs of the function space.



Classical Natural Gradient Descent

The classical natural gradient descent is nearly a regular gradient descent with the difference in optimizing the parameter space by changing the the output distribution space. In most times this is leading to an efficient optimization.

For this the Scale has to be changed to a different metric. The regular gradient descent is using the euclidean metric to measure the distance between the parameters, which does not fit to the output distribution.



from https://wiseodd.github.io/techblog/2018/03/14/natural-gradient/
The red line shows the euclidean metric in the gaussian curve with the same distance. The overlap comparison of the first plot and the second one gives us the idea of the needed transformation. This makes the work with a euclidean parameter room dangerous for our result and learning accuracy if we wish to have a realistic accuracy in the output distribution.

To measure the difference in two output distributions, it is needed to use the Kullback Leibler (KL) divergence.

The Kullback Leibler divergence measures the overlap and closeness between the two output distributions.
The KLD is no metric by it self but has a approximately symmetric in specific cases and is useable for us in this cases.

The KLD let us use the distribution space and not the parameter space. The focus is the output distribution of our model which enable us the usage of the KLD.

The Fisher Information Matrix is an additional important tool to know about.
The KL-Divergence is the optimization of the distribution space.The Fisher information matrix is the way to represent this.

The resulting formula for the natural gradient descent is :

$$\theta_{n+1} = \theta_n - \eta F^{-1} \nabla \mathcal{L}(\theta)$$

It is similar to the classical gradient descent math with the difference of the F-Inverse (inverse function) behind the “step size”. Now we are able to calculate in the the distribution space rather than in the parameter space. With The F-Inverse the density of information about the geometry and the control over the movement of the model in the distribution output is given.This calculation is now detached from the movements in the cost space.

Quantum Natural Gradient Descent

The Quantum natural gradient descent (QNG) based on the upper versions of the gradient descent methods. The QNG takes the advantage geometric properties of parameterized quantum states. The Fisher metric has to be changed to quantum state metric. The Fubini-Study-metric (or quantum fisher metric) is used for this.

This Metric is defining the classical metric to a distance within a quantum geometric space.

$$\gamma(\psi, \phi) = \gamma(Z, W) = \arccos \sqrt{\frac{Z_\alpha \overline{W}^\alpha \; W_\beta \overline{Z}^\beta}{Z_\alpha \overline{Z}^\alpha \; W_\beta \overline{W}^\beta}}.$$

$$F_{ij} = \mathrm{Re}(\langle \partial_i \phi | \partial_j \phi \rangle) - \langle \partial_i \phi | \phi \rangle \langle \phi | \partial_j \phi \rangle.$$

|φ(θ)⟩ initial ansatz
∂|φ(θ)⟩/∂θ_i is the partial derivative of |φ(θ)⟩ with respect to θ

https://medium.com/@ziyu.lili.maggie/rethinking-gradient-descent-with-quantum-natural-gradient-330da14f621

Comparison in a table : [LINK]

1. Optimizer with Pennylane:

   Link to pennylane Optimizer with Numpy Interface:

   1. AdagradOptimizer
   2. AdamOptimizer
   3. CVNeuralNetLayers (https://arxiv.org/abs/1806.06871)
   4. MomentumOptimizer
   5. NesterovMomentumOptimizer
   6. QNGOptimizer
   7. QuantumMonteCarlo
   8. RMSPropOptimizer
   9. RotoselectOptimizer
   10. RotosolveOptimizer()
   11. ShotAdaptiveOptimizer(
   12. Gradient Descent Optimizer

VQECost

2. Optimizer with PyTorch:
   (https://pytorch.org/docs/stable/optim.html)

   1. Adadelta
   2. Adagrad
   3. RMSprop
   4. Adam
   5. AdamW
   6. SparseAdam
   7. Adamax
   8. ASGD

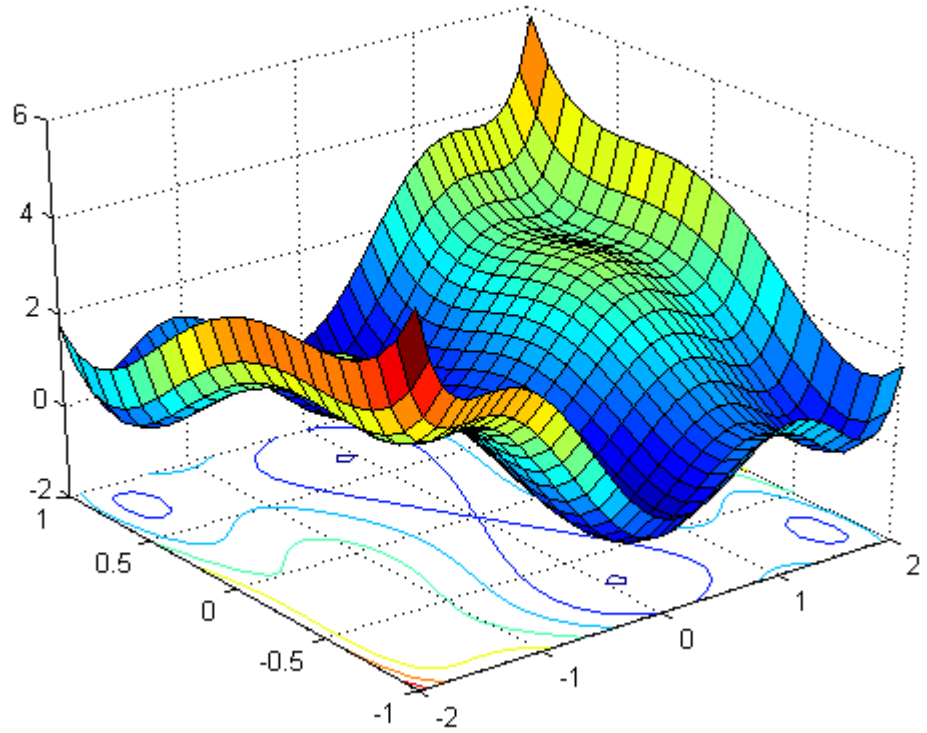
9. LBFGS
10. NAdam
11. RAdam
12. Rprop
13. SGD

Optimizer for Machine Learning:

Lets first ask “what is a Optimizer?”. Is is a method or algorithm which helps to find the minimum of an error function (loss function) or to optimize the efficiency of output.
From a technical view optimizer are mathematical functions which are dependent to parameters of the used model like weights and biases. Optimizer helps to understand the action of the weights and the learning rate and are able to optimize the losses.

The class of optimizers can be separated in different classes.

Loss Function:



Picture: [Link]

The Loss Function is the main essence of Machine-Leanring to get from mathematical form to a reproducible and repeatable process of computation with amazing results.

What is the “Loss Function”?

Generally said the “loss function” is a method of evaluating how well the algorithm calculates and models the given dataset. The result from the optimizer gives you a characteristic output for too high loss or small number for a good result.
The quality of the LF is related to the model accuracy.

Quantum gradient transforms

@pennylane [LINK]
@pennylane [Gradient Transformation]

The quantum gradient transformatation is a strategie for computing the gradient of a quantum circuit that works by transfomring the quantum circuit into one ore more gradient circuits. These gradient circuits, once executed and post-processed, return the gradient of the original circuit. Examples of quantum gradient transformation include finite-differences and parameter-shift-rules.
This model provides a selection of devices-independent, differentiable qauntum gradient transformations.As such, these quantum gradient transforms can be used to compute the gradients of quantum circuits on both simulators and hardware.

Gradient Transformation-Overview:

finit_diff (finit difference gradient of all gate parameters with respect to the inputs)
param_shift (parameter shift gradient)
param_shift_cv (compute the parameter-shift gradient of all gate parameters with respect to its inputs)

Quantum Gradients

@pennylane [LINK]

Quantum Differentiable Programming

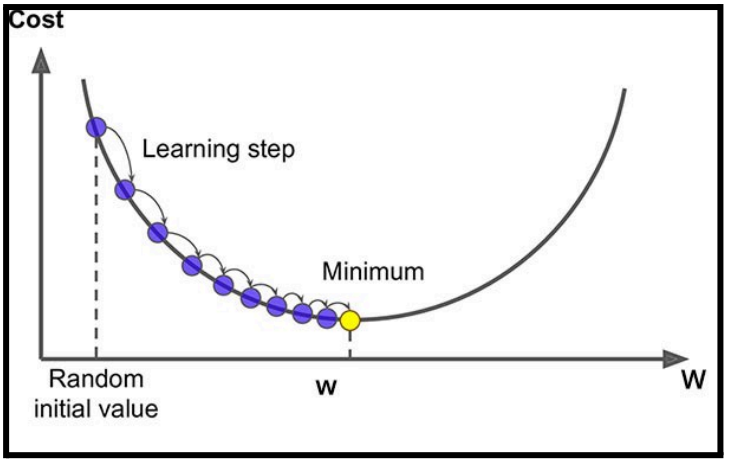
@pennylane [LINK]

In quantum computing, one can automatically compute the derivatives of variational circuits with respect to their input parameters. Quantum differentiable programming is a paradigm that leverages this to make quantum algorithms differentiable, and thereby trainable.

Gradient Descent

> The Gradient Descent is a convex Function which optimize the parameter iteratively. This optimize a given function to the minima for a local minima. . This means that the gradient descent minimize

iteratively the loss function. The idea to success this algorithm is the to use the direction opposite of the steepest ascent.
This depends on the derivatives of the loss function for finding minima and uses the complete training set to calculate the gradient of the cost function for the parameters.
This uses a lot of memory and slows down by size.



[LINK]

$$W_{new} = W_{old} - \alpha * \frac{\partial(Loss)}{\partial(W_{old})}$$

Pro:

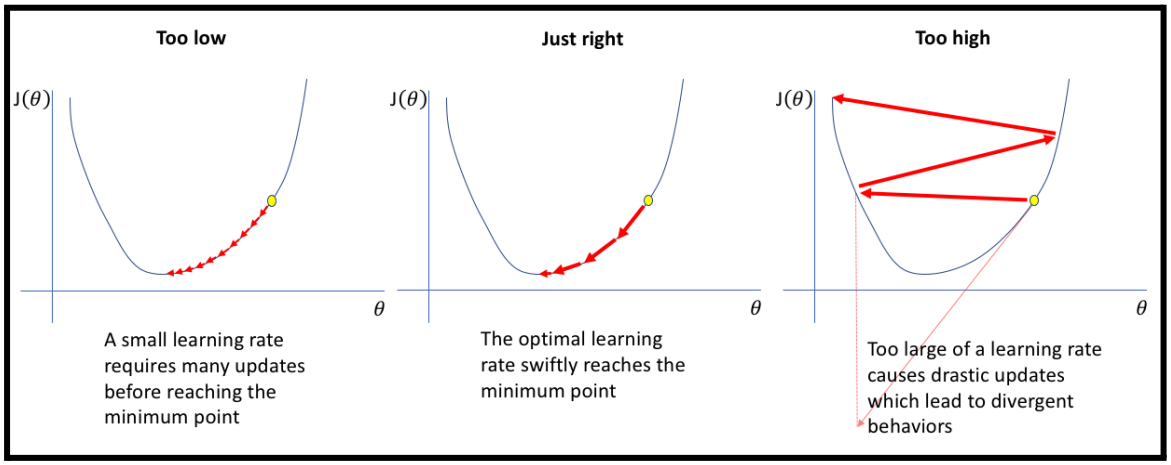
1. lightweight of machine learning (learning view)
1. easy to use
2. lot of documentation
3. nearly in each library available

Contra:

1. iterative calculation over the complete dataset (slow)
2. need much memory

Learning Rate:

How big and small steps the gradient descent takes towards the local minimum is determined by the learning rate, which indicates how fast or slow we are moving towards the optimum weights.



[LINK]

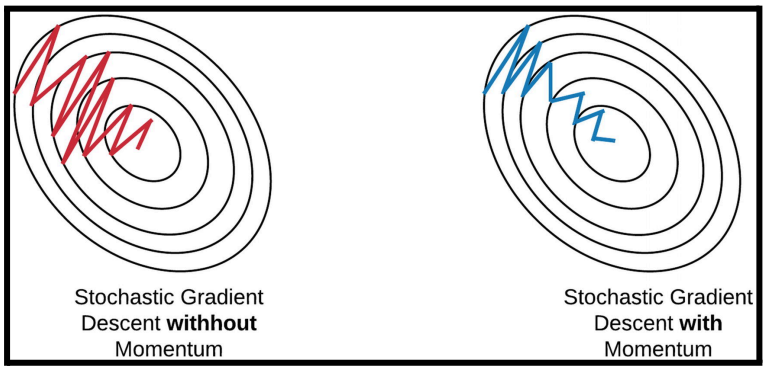
Gradient Descent - How it works

@page [LINK]

The Gradient Descent is an iterative process to find the minima of a function.
If an optimization problem is not solvable deterministically we use Approximate solution.
The goal is to maximize or minimize the function .
Obviosly a solution of this kind has different ways to solve. This group is called the Gradient Descent (GD). This Class is a subfield of the optimization Algorithms which calculates the local minimum of a (convex) function by changing (iterating) the parameters of this function.

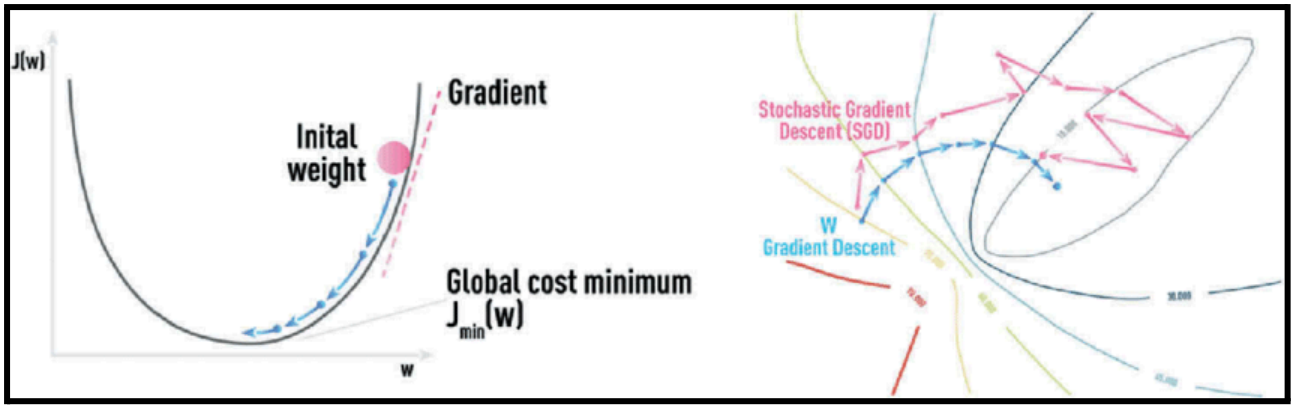
Stochastic Gradient Descent:

The SGD “Stochastic Gradient Descent” is a avariation of the normal Gradient Descent.The difference is the update of the model parameters one by one.
It is not using the iterative way. it uses the complete dataset in each data as one step.



Additional information:
https://golden.com/wiki/Stochastic_gradient_descent_(SGD)

Compare the gradient descent and the stochastic gradient descent:



[Link]

Pro:

- Frequent updating of model parameters
- Less memory required.
- Large datasets can be used as only one sample needs to be updated at a time.

Contra:

- Frequent updates can also lead to noisy gradients which increase rather than decrease the error.
- High variation.
- Frequent updates require high computational cost.

Mini-Batch Gradient Descent:

The MBGD is using the concept of the batch gradient descent and the SGD.

By splitting the traning dataset into small pieces (batches) and calculate th updates for ech of those batches. This makes the MBGD more robust then the stochastic gradient descent and more effective then the basic batch gradient descent.
The MBGD reduce the variance of the own parameters and the convergence gets more stable. The Bachtes has a size from 50 till 256 examples by random choise.

Pro:

- It leads to more stable convergence.
- More efficient gradient calculations.
- Less memory required.

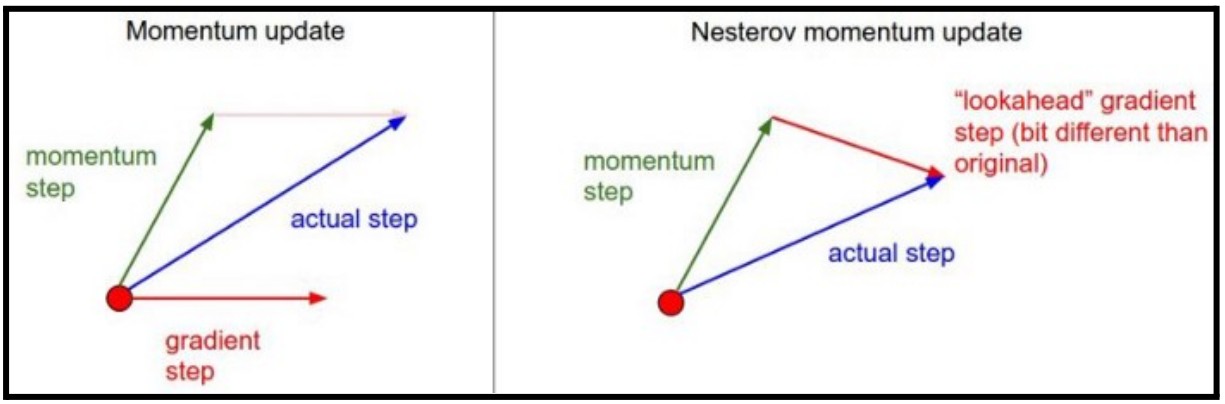
Contra:

- Small batch gradient descent does not guarantee good convergence,
- If the learning rate is too low, the convergence speed will be slow.
- If it is too high, the loss function will fluctuate or even deviate from the minimum value.
- Learning rate constant

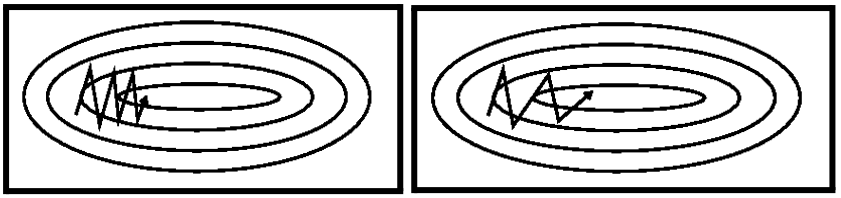
SGD with Momentum (as seen in the picture above)

By adding a term of momentum to the stochastic optimizer method the function gets more inertia for the object of movement (test point). The movement of the in front located calculation is influencing the habit .

The first actualisation is used and tuned with a resulting “smoothed” reaction. This stabilisation can have a positive impact of the modell. This method is faster trainable, because it can solve local optimization problems.



$$\nu_{new} = \eta * \nu_{old} - \alpha * \frac{\partial(Loss)}{\partial(W_{old})}$$



[LINK]

Pro:

- Momentum helps reduce noise.
- An exponential weighted average is used to smooth the curve.

Contra:

- Additional hyperparameters are added.
- Learning rate constant

Quantum-Optimizer with pennylane:

Gradients and Traning: [LINK]
QML Gradients: [LINK]
Optimizer: [LINK]

AdagGradOptimizer

AdaGrad(Adaptive Gradient Descent) is in comparison to the basic descent gradient algorithms a very flexible system. The flexibility comes from the different learning rates for each and every neuron and every hidden layer based on the different iterations.

$$W_{new} = W_{old} + \frac{\alpha}{\sqrt{cache_{new}} + \epsilon} * \frac{\partial(Loss)}{\partial(W_{old})}$$

Pro:

- The learning rate changes adaptively through iterations
- useable with small datasets

Contra:

- When the neural network is very deep, the learning rate becomes a very small number, leading to the dead neuron problem.

RMSprop (Root Mean Square Propagation)

@pennylane [LINK]
RMS-Prop essentially combines Momentum and AdaGrad. RMS-Prop is a special version of Adagrad where the learning rate is an exponential average of the gradients rather than a sum.

$$cache_{new} = \gamma * cache_{old} + (1 - \gamma) * (\frac{\partial(Loss)}{\partial(W_{old})})^2$$

or

$$a_i^{(t+1)} = \gamma a_i^{(t)} + (1 - \gamma)(\partial_{x_i} f(x^{(t)}))^2$$

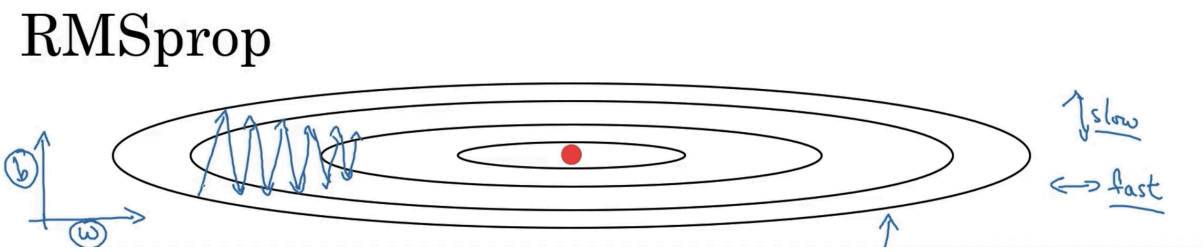
η: stepsize
γ: learning rate decay
eps: offset ϵ added for numeric stability (like Adagrad)

Pro:

- **low configuration level**

Contra:

- Slow Learning



AdaDelta

The AdaDelte is an extension to the AdaGrad algorithm with an optimized learning rate.
It optimizes the monotony of the AdaGrad and the decreasing learning rate.

Pro:

- is using a default learning rate

Contra:

- needs lot of computer resources

Adam

The Adam (Adaptive Momentum Estiamation) uses adaptive learning rates for ech paramter. It stores the decaying average ot the last gradients like the momentum. Similar to the RMS-Prop the decaying average of the lst squared gradients are used.
On this way it is the combination of the positive part of the RMS-Prop and AdaDelta

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{S_{dw_t} - \varepsilon}} * V_{dw_t}$$

$$b_t = b_{t-1} - \frac{\eta}{\sqrt{S_{db_t} - \varepsilon}} * V_{db_t}$$

Pro:

- easy usage
- computationally efficient
- little memory usage

GradientDescentOptimizer

@pennylane [LINK]

$$x^{(t+1)} = x^{(t)} - \eta \nabla f(x^{(t)})$$

η as an user-defined hyperparameter corresponding to step size

---

LieAlgebraOptimize

@pennylane [LINK]

Riemannian gradient descent algorithms can be used to optimize a function directly on a Lie group as opposed to on a Euclidean parameter space.

@paper [LINK]

MomentumOptimizer

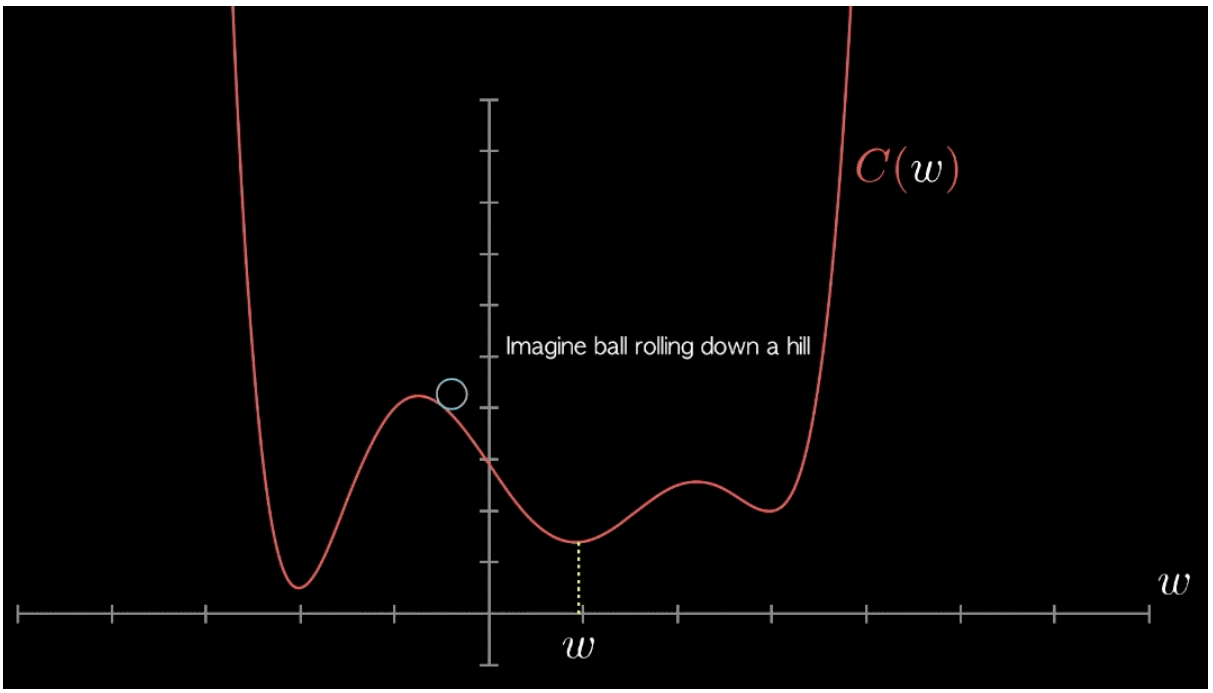
@pennlane [LINK]
The Momentum Optimizer uses an additional therm at the gradient descent.
The Momentum Optimizer helps us to get to a local minimum faster and can find the local (or global minima) more efficient then the SGD.

$$x^{(t+1)} = x^{(t)} - a^{(t+1)}$$

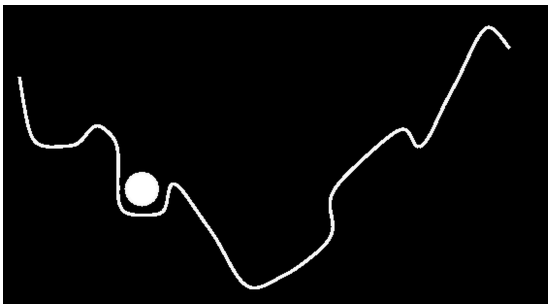
were a (accumulator) is updated:

$$a^{(t+1)} = ma^{(t)} + \eta \nabla f(x^{(t)})$$

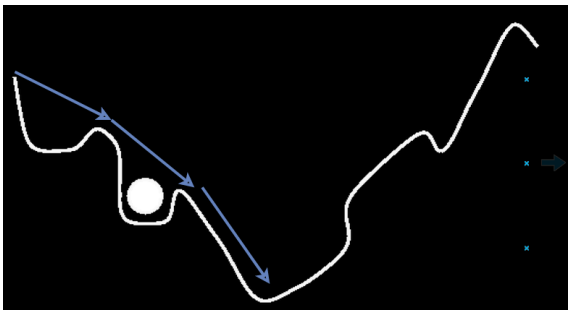
m : momentum as a user-defined hyperparameter
η: step size as a user-defined hyperparameter



[source]



This picture above shows the 2 dimensional simplified problem, where the ball (algorithm) is stuck nearly at the first position (local minima) instead of finding the right global minima. Adding the therm with a dynamic part in it helps to roll above the local minima. This method uses the temporal element as a dynamical driver above local minimas.



The time element is adding a swing to a ball. This effect we call momentum.

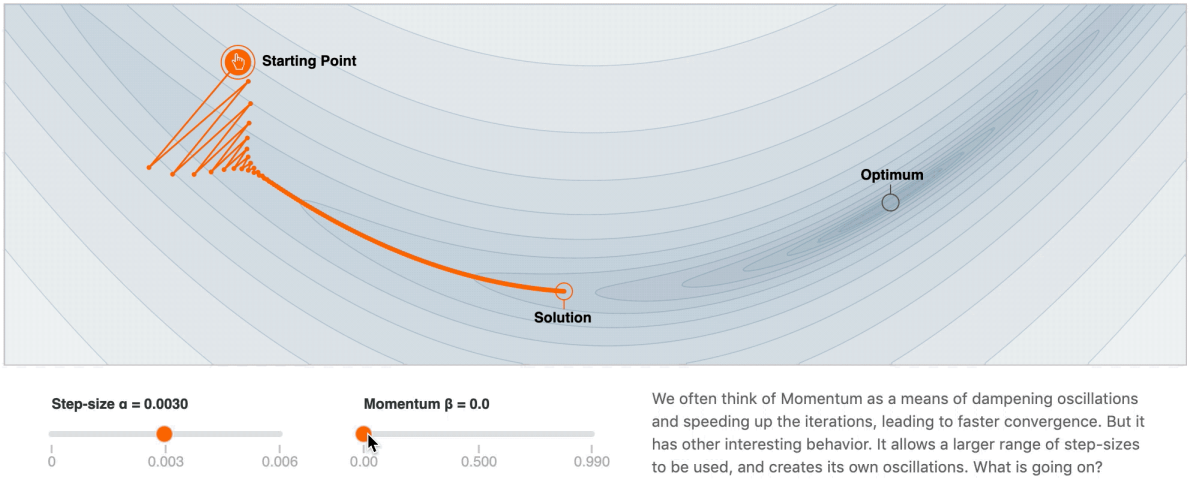
$$x^{(t+1)} = x^{(t)} - \left(ma^{(t)} + \eta \nabla f(x^{(t)}) \right)$$

more mathematically:

$$\theta = \theta - das \nabla J(\theta) + Cv_t$$

- Theta θ as a parameter (weight, bias or activation)
- “das” is the learning rate (or steps) sometime calles Alpha, ein or ε
- J as the target function which we optimize (cost- or loss-function)
- Gamma γ,a constant term. Also called the momentum, and rho ρ is also used instead of γ sometimes
- Last change (last update) to θ is called vt

This benefit to get a faster result has the price. The momentum can “kick” the messurment above the wished result as seen in the following picture:



[source]

@paper [LINK]
@page [LINK]

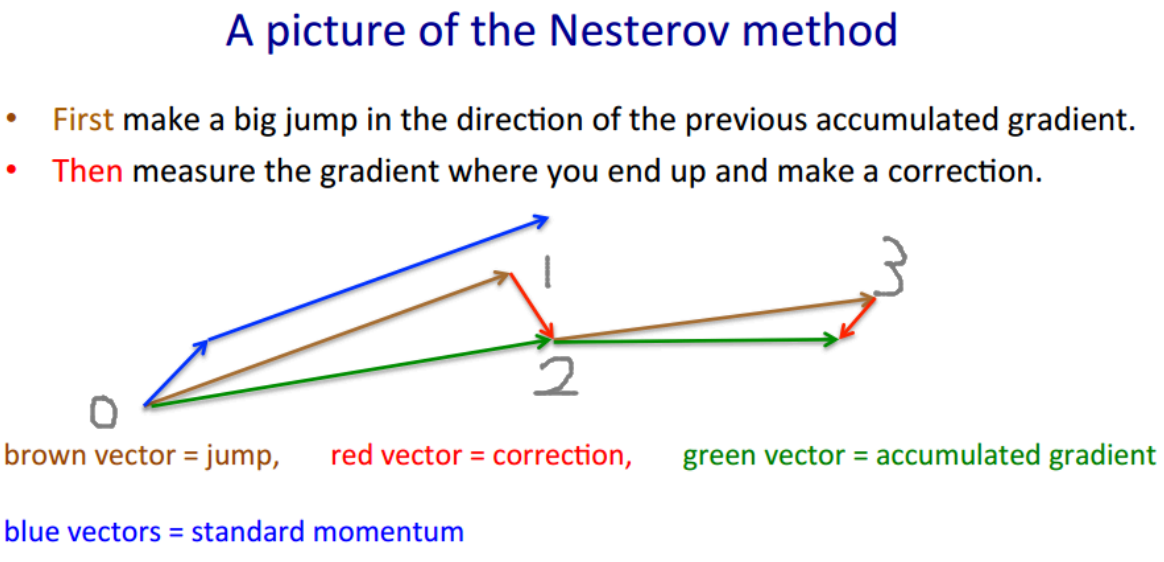
NesterovMomentumOptimizer

The NesterovMomentum Optimizer works like a basic Momentum Optimizer. Additionally it moves the shift of the current input by the Nesterov Momentum term when the computing of the gradient of the objective function is done.

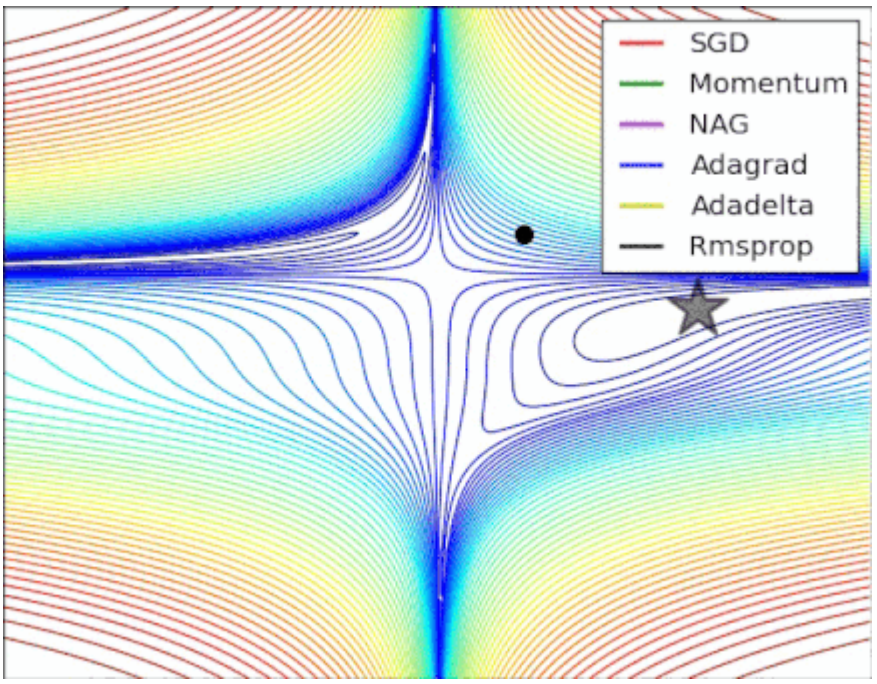
parameters:

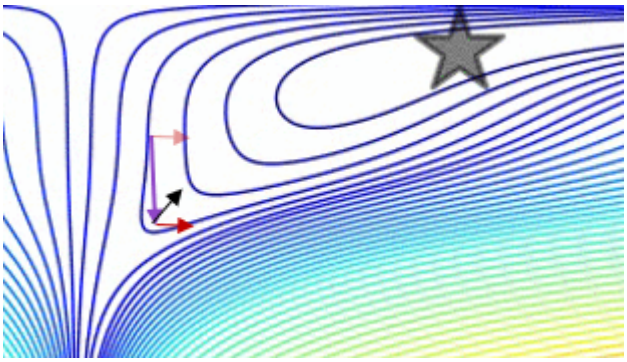
- η as the step size
- m as the momentum

$$a^{(t+1)} = ma^{(t)} + \eta \nabla f(x^{(t)} - ma^{(t)})$$



Lets use this knowledge and add it to this habit in two pictures:


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop



QNGOptimizer

@pennylane [LINK]
@GitHub [LINK]
@medium [LINK]
@paper [LINK]
The Quantum Natural Gradient
parameters:

Optimizer with adaptive learning rate, via calculation of the diagonal or block-diagonal approximation to the Fubini-Study metric tensor. A quantum generalization of natural gradient descent.

$$x^{(t+1)} = x^{(t)} - \eta g(f(x^{(t)}))^{-1} \nabla f(x^{(t)})$$

here the expection value of some observable messurment on the variational quantum circuit $U(x^{(t)})$

$$f(x^{(t)}) = \langle 0|U(x^{(t)})^{\dagger} \hat{B} U(x^{(t)})|0\rangle$$

ShotAdaptiveOptimizer

@pennylane [LINK]

Optimizer where the shot rate is adaptively calculated using the variances of the parameter-shift gradient.

Gradient-Free Optimizers:

RotoSolveOptimizer

@pennylane [LINK]
@paper [LINK]
@tutorial [LINK]
@paper compare VQE:Reinforcement Learning with RotoSolver [LINK]

The RotoSolve Optimizer is a **gradient-free** Optimizer which optimizer minimizes an objective function with respect to the parameters of a quantum circuit without the need for calculating the gradient of the function.

RotoselectOptimizer

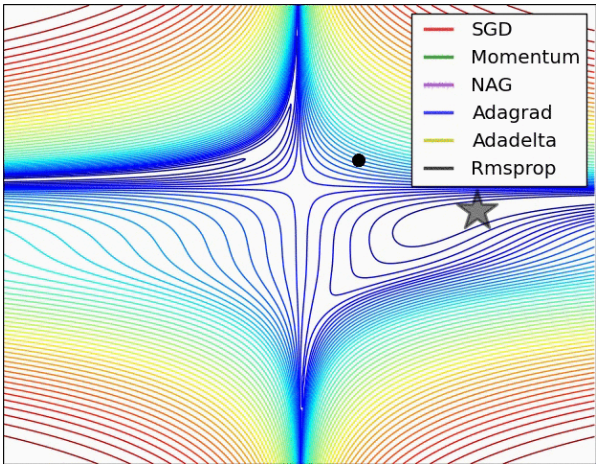
@pennylane [LINK]
Rotoselect is a gradient-free optimizer

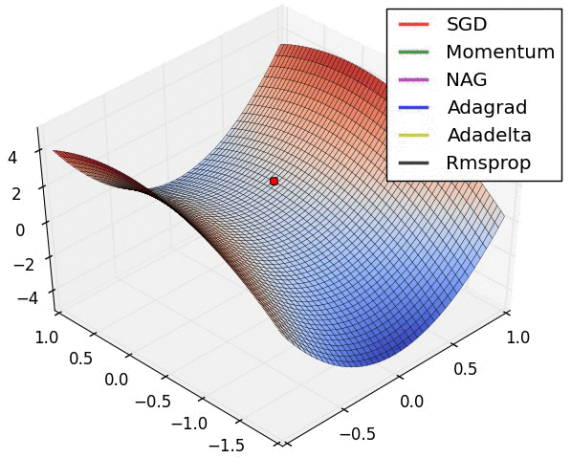
The Rotoselect optimizer minimizes an objective function with respect to the rotation gates and parameters of a quantum circuit without the need for calculating the gradient of the function.

Comparison:

SDG,Momentum,NAG,AdaGrad,AdaDelta,RMSprop Compare

NAG: https://github.com/numericalalgorithmsgroup/NAGPythonExamples





Paper: