VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

Deep Learning

Gradient Descent

Jan Platoš, Radek Svoboda

March 24, 2024

Department of Computer Science
Faculty of Electrical Engineering and Computer Science
VŠB - Technical University of Ostrava

Gradient Descent

Gradient Descent - Introduction

- Gradient Descent is an optimization algorithm used to minimize the cost function of a machine learning model.
- It's important in machine learning because the cost function measures the difference between the predicted output of the model and the actual output.
- Gradient Descent is widely used in machine learning to optimize models such as linear regression, logistic regression, and neural networks.
- It is an important tool for achieving high accuracy in machine learning applications.

Gradient Descent - How it Works

- Initialize the model's parameters with some random values.
- Compute the **cost function** for the current parameters.
- Compute the **gradient** of the cost function with respect to each parameter.
- Update each parameter by subtracting the product of the gradient and the **learning rate**.
- Repeat until the cost function reaches a **minimum**.

Gradient Descent - Simple example

- Let the cost/loss function as $y = x^2$.
- The gradient of the function is then $y' = 2x$.
- Let start with $x = 10$, the gradient is then -20.
- The new x depends on the learning rate λ.
- Repeat until the cost function reaches a minimum.

Gradient Descent - Simple example

- Let the cost/loss function as $y = x^4 - 5x^2 - 3x$.
- The gradient of the function is then $y' = 4x^3 - 10x - 3$.

Gradient Descent - Back-propagation alg.

- *Forward phase:*
    - The input is fed into input neurons.
    - The computed values are propagated using the current weights to the next layers.
    - The final predicted output is compared with the class label and the error is determined.

Gradient Descent - Back-propagation alg.

- *Backward phase:*
  - The main goal is to learn weights in the backward direction by providing the error estimation from later layers to the earlier layers.
  - The estimation in the hidden layer is computed as a function of the error estimate and weight is the layers ahead.
  - The error is estimated again using the gradient method.
  - The process is complicated by the using of non-linear functions n the inner nodes.

Gradient Descent - Back-propagation alg.

- Lets have an example multi-layer neural network with single output neuron.
- In each iteration do take the i-th input vector.
- Pass it through the networks using the forward pass.
- Compare the i-th output o_i to the expected value y_i.
- Compute the error and update the weight using the learning rate η.
- The goal is to optimize the weights w_i to minimize the error function of the differences between y_i and o_i.

Gradient Descent - Back-propagation alg.

- The error function E over whole dataset of size n may be defined as follows:

$$E = \frac{1}{2}\sum_{i=0}^{n}(y_i - o_i)^2$$

- The weights of the neurons must be adapted according to the error produced by the neuron weight.

$$w_{i+1} = -\eta\frac{\partial E}{\partial w_i} + \mu w_i$$

Gradient Descent - Back-propagation alg.

- The partial derivation may be computed using so called chain rule.

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w_i}$$

- where

$$y = \frac{1}{1 + e^{-\lambda z}} \qquad z = \sum_{i=0}^{m} w_i x_i$$

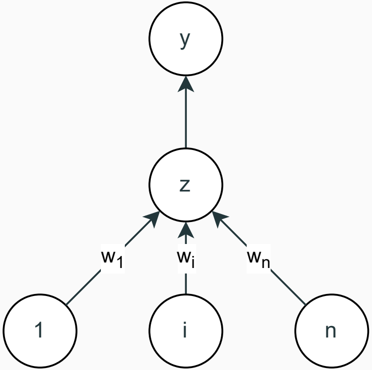
- therefore

$$\frac{\partial z}{\partial w_i} = x_i \qquad \frac{\partial y}{\partial z} = y \cdot (1 - y)\lambda$$

Gradient Descent - Back-propagation alg.

- The first partial derivation computation differs for neuron from output and hidden layer.
- The solution for the output layer and *i*-th output is as follows:

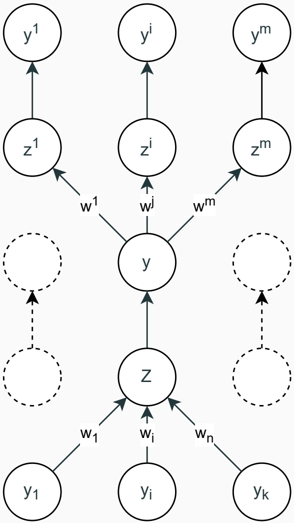
$$\frac{\partial E}{\partial y} = (y_i - o_i)$$



Gradient Descent - Back-propagation alg.

- The solution for the hidden layer and i-th output is as follows:

$$\frac{\partial E}{\partial y} = \sum_{j=0}^{m} \frac{\partial E}{\partial z^j} \cdot \frac{\partial z^j}{\partial y} = \sum_{j=0}^{m} \frac{\partial E}{\partial z^j} \cdot w^j$$



Gradient Descent - General Definition

- Let use a regression with the loss function defined as RMSE.

$$L(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- The gradient is then defined as:

$$\nabla L = \frac{\partial L}{\partial \mathbf{w}} = \left(\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots, \frac{\partial L}{\partial w_m}\right)$$

- Weights can be calculated using the following:

$$\mathbf{w} = \mathbf{w} - \eta \nabla L$$

Gradient Descent - General Definition

- Computation of true gradient is usually difficult to compute.
- It may be easily approximate using a the following formula:

$$\frac{\partial f}{\partial a_i} = \frac{f(a_1, a_2, \ldots, a_i + \epsilon, \ldots, a_m) - f(a_1, a_2, \ldots, a_i, \ldots, a_m)}{\epsilon}$$

- This approximation is simple to implement but expensive for computation.

Gradient Descent - Stochastic Gradient Descent

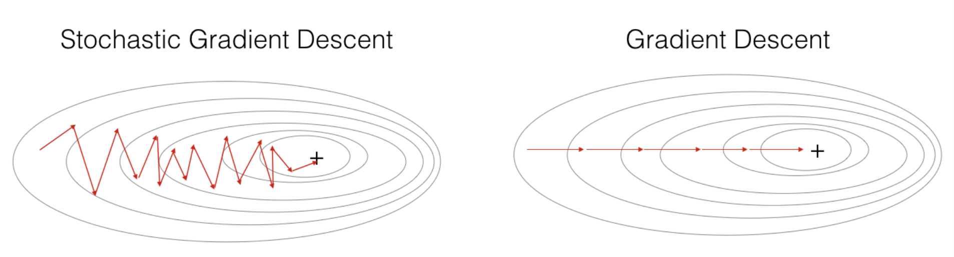
- Gradient Descent is computationally very expensive (consider 1M of samples and 100k weights).
- It leads to almost perfect approximation of the loss function.
- Stochastic gradient descent decreases the complexity by replacing the whole computation using only a single data point.

$$\mathbf{w} = \mathbf{w} - \eta \nabla L_i$$

$$L_i(y_i, \hat{y}_i) = \frac{1}{n}(y_i - \hat{y}_i)^2$$

- How it works? May it works?

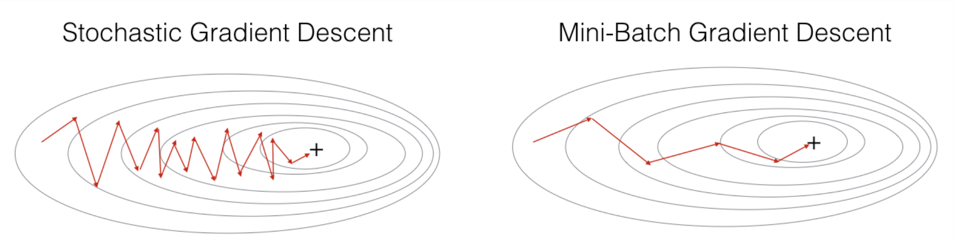
Gradient Descent - Stochastic Gradient Descent



https://github.com/Kulbear/deep-learning-coursera/blob/master/Improving%20Deep%20Neural%20Networks%20Hyperparameter%20tuning%2C%20Regularization%20and%20Optimization/Optimization%20methods.ipynb

- SGD may lead to a really chaotic behaviour.
- Increasing the number of samples used for gradient computation may improve the stability of the optimization.
- The amount is called **batch** that are usually very small in comparison to the whole dataset.
- Stochastic gradient descent randomly divides the set of observations into minibatches.
- For each minibatch, the gradient is computed and the vector is moved.
- Once all minibatches are used, you say that the iteration, or epoch, is finished and start the next one.

Gradient Descent - Mini-batch Stochastic Gradient Descent

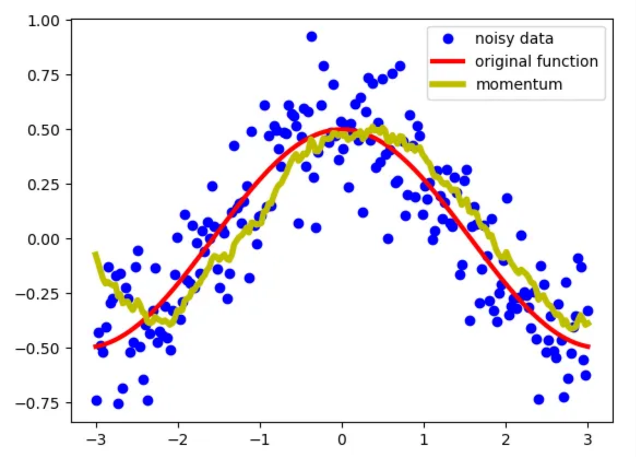


https://github.com/Kulbear/deep-learning-coursera/blob/master/Improving%20Deep%20Neural%20Networks%20Hyperparameter%20tuning%2C%20Regularization%20and%20Optimization/Optimization%20methods.ipynb

Gradient Descent - Momentum

- The nature of SGD with or without mini-batches is rather chaotic.
- The information gained from previous steps and previous batch is forget.
- The momentum introduce some kind of memory into the computation by preserving the past behaviour.
- It may be understand as a kind of moving average on our intermediate computation.
- The more precise will be exponential weighted moving average.

Gradient Descent - Momentum



https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d

Gradient Descent - Momentum

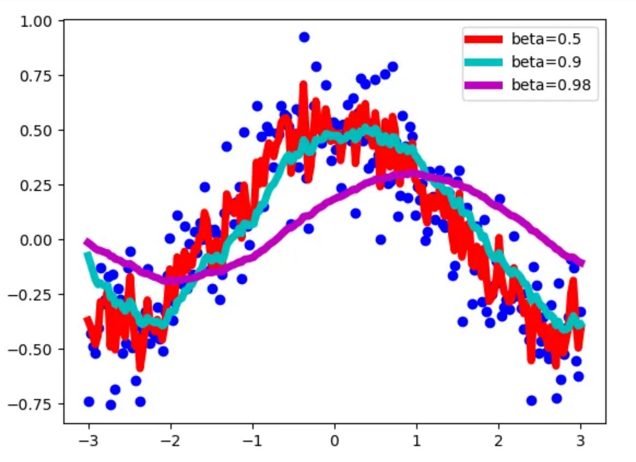
- The computation of the true value corresponds to the following formula:

$$v_t = \beta v_{t-i} + (1 - \beta)x_i,\ \beta \in [0, 1]$$

- SGD with momentum if then defined as:

$$\mathbf{v_t} = \beta \mathbf{v_{t-1}} + (1 - \beta)\nabla L$$
$$\mathbf{w} = \mathbf{w} - \mathbf{v_t}$$

Gradient Descent - Momentum



https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d

Gradient Descent - Summary

- Gradient Descent is a powerful optimization algorithm that is widely used in machine learning for minimizing cost functions.
- It's important to understand the types of Gradient Descent, the learning rate, and the advantages and disadvantages of the algorithm.
- Advantages: simplicity, effectiveness, scalability.
- Disadvantages: risk of getting stuck in local minima, sensitivity to the initial parameters, and the need for a large amount of data.

Gradient Descent - References

1. https://realpython.com/gradient-descent-algorithm-python/
2. https://towardsdatascience.com/how-do-we-train-neural-networks-edd985562b73
3. https://www.tomasbeuzen.com/deep-learning-with-pytorch/chapters/chapter1_gradient-descent.html
4. https://github.com/Kulbear/deep-learning-coursera/blob/master/Improving%20Deep%20Neural%20Networks%20Hyperparameter%20tuning%2C%20Regularization%20and%20Optimization/Optimization%20methods.ipynb

Questions?