Machine Learning & its Application

Gradient Descent & Backpropagation

02/10/2023

Done by: Issa Hammoud

Outline

- Introduction
- Gradient Descent
- Backpropagation
- Conclusion

Introduction

- We need an iterative algorithm to solve logistic regression.
- As we will see, the same algorithm will be used later with neural networks.
- This algorithm is based on gradient descent, with many variants.
- We will explore how it works and how can be used in practice.
- In addition, we need an efficient way to compute the gradients.
- We will present backpropagation algorithm and which problem it solves.

Outline

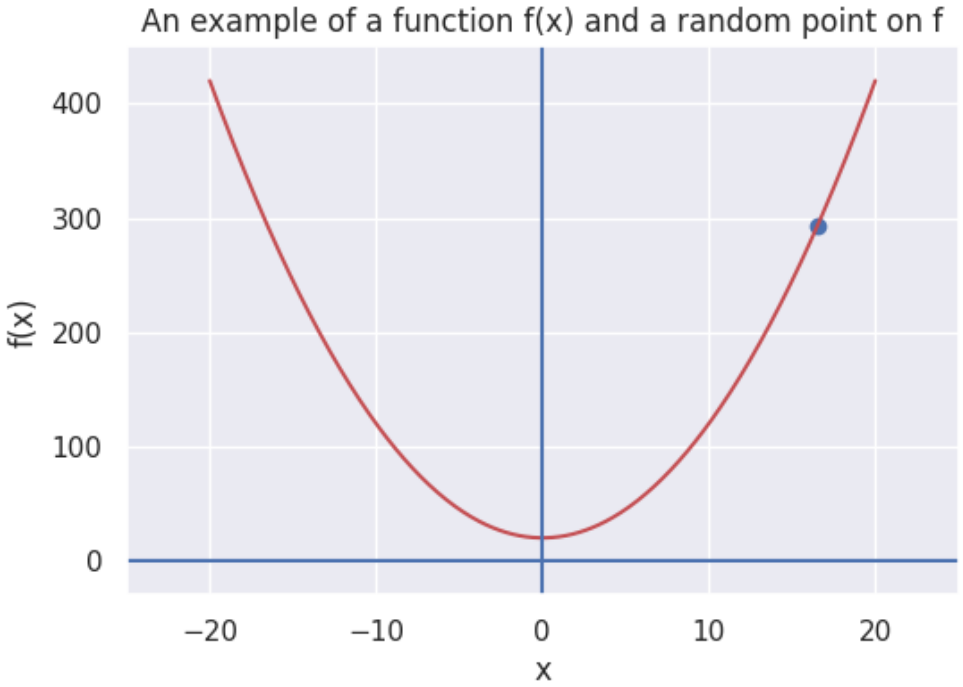
- Introduction
- Gradient Descent
- Backpropagation
- Conclusion

Gradient Descent

- How to find a minimum
  - An intuitive approach
  - Gradient Descent
  - Global and local minimum
- From Optimization to Learning
  - Stochastic Gradient Descent
- Improvements on SGD
  - Learning Rate Schedules
  - Momentum SGD
  - Adaptive SGD

An intuitive approach

- Let's begin with an easy case as follows.
- We have a function f and a random point on it.
- We want to go from this point to the minimum.
- An intuitive approach is as follow:
  - Define a small step ε.
  - Compute f(x+ε) and f(x-ε).
  - Take the direction with the smallest value.
  - Repeat until f(x+ε) and f(x-ε) are both greater than f(x).



An intuitive approach

- ε defines the precision we want to achieve.
- For instance, if the minimum is an irrational number and $\varepsilon=0.1$, we will achieve the minimum up to 1 decimal values.
- More precision means smaller steps, and thus the algorithm is slower.
- In addition, at each step, we need to evaluate the function twice.
- This is impractical, specifically if we have a higher dimensional function.

Gradient Descent

- How to find a minimum
  - An intuitive approach
  - Gradient Descent
  - Global and local minimum
- From Optimization to Learning
  - Stochastic Gradient Descent
- Improvements on SGD
  - Learning Rate Schedules
  - Momentum SGD
  - Adaptive SGD

An intuitive approach

```
def find_the_minimum(function, initial_guess, step=0.01):
    """
    This function accepts a function as input (to find its minimum), an initial guess of the minimum
    and a step to move.
    """
    number_of_iterations = 0
    x = initial_guess
    while function(x) > function(x - step) or function(x) > function(x + step):
      if function(x - step) > function(x + step):
        x = x + step
      else
        x = x - step
      number_of_iterations+=1
    print(f"We iterated {number_of_iterations} times to find the minimum.")
    return x
```

In the above example, the initial guess was 16.5, and we needed 1450 iterations to find the exact minimum, which is 2.

Gradient Descent

- We would like to avoid evaluating the function twice at x+ε and x-ε.
- Let's review some math properties from school:

$$f(x+\epsilon) \approx f(x) + \epsilon f'(x)$$

- Let's see what we can deduce about f(x+ε) and f(x-ε).

$$f(x+\epsilon) - f(x) \approx \epsilon f'(x)\,;\; f(x-\epsilon) - f(x) \approx -\epsilon f'(x)$$

$$if\ f'(x) > 0 \rightarrow f(x+\epsilon) - f(x) > 0\ and\ f(x-\epsilon) - f(x) < 0$$

$$f(x-\epsilon) < f(x) < f(x+\epsilon)$$

$$if\ f'(x) < 0 \rightarrow f(x+\epsilon) - f(x) < 0\ and\ f(x-\epsilon) - f(x) > 0$$

$$f(x+\epsilon) < f(x) < f(x-\epsilon)$$

Gradient Descent

- We can use the direction of the derivative (sign) instead of trying both directions!
- However, we still need the same number of iterations as before.

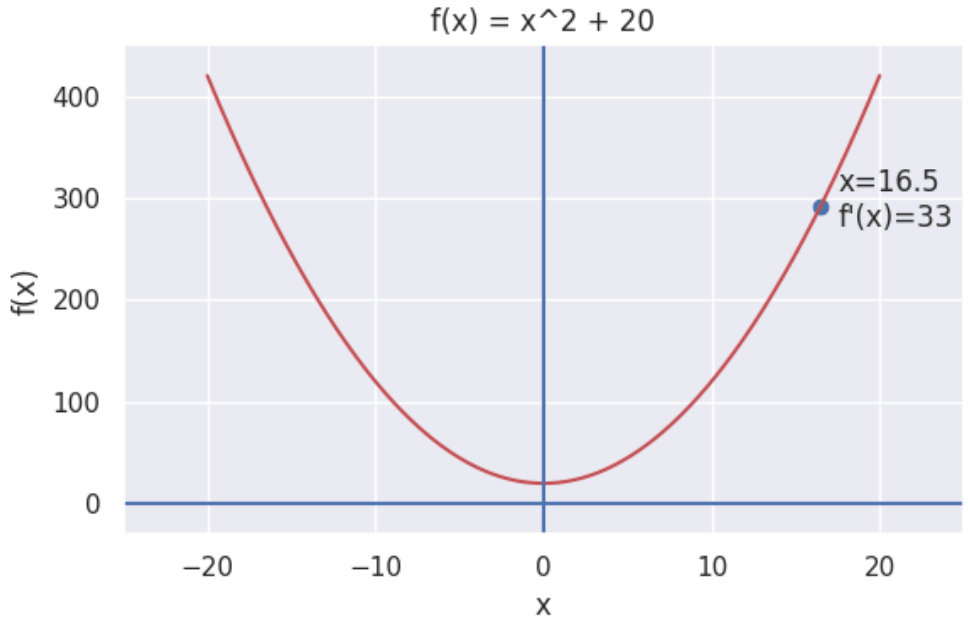
```
def find_the_minimum(function, derivative, initial_guess, step=0.01):
    """
    This function accepts a function as input (to find its minimum), the derivative of the function (which is a function as well)
    an initial guess of the minimum and a step to move.
    """
    number_of_iterations = 0
    x = initial_guess
    while function(x) > function(x - step) or function(x) > function(x + step):
      derivative_sign = 1 if derivative(x) > 0 else -1
      x = x - step * derivative_sign
      number_of_iterations+=1
    print(f"We iterated {number_of_iterations} times to find the minimum.")
    return x
```

Gradient Descent

- Let's take a closer look about the approximation above:
  - If $f(x) = x$, then $f(x+\varepsilon) = f(x)+\varepsilon$.
  - If $f(x) = x^2$, then $f(x+\varepsilon) \approx f(x)+2x\varepsilon$.
- In the first case, a small change in x yields the same change in y.
- However, in the second case, it yields a change proportional to $2x\varepsilon$.
- The derivative represents the steepness of the function at a point x.
- This is useful because if we are on a steep point, we would like to descent quickly.

Gradient Descent

- Thus, we want to use the derivative to get the ***direction*** and ***scale*** of the descent.
- In addition, when achieving the minimum, the derivative will become too small.
- So can we eliminate the step variable now? Not really because we may diverge.



Gradient Descent

- We will a step not to define precision, but to control the amount of the update.
- This parameter is known as the ***learning rate***. We can write now the algorithm:

```
def find_the_minimum(function, derivative, initial_guess, step):
    """
    This function accepts a function as input (to find its minimum), the derivative of the function (which is a function as well)
    an initial guess of the minimum and a step to move.
    """
    number_of_iterations = 0
    x = initial_guess
    while np.abs(derivative(x)) > 0.0001: # iterate until the absolute value of the gradient is too small
      x = x - step * derivative(x)
      number_of_iterations+=1
    print(f"We iterated {number_of_iterations} times to find the minimum.")
    return x
```

Gradient Descent

- We presented the case of 2D functions but the same thing works in higher spaces.
- The only difference is instead of talking about derivative, we use gradients.

$y = f(X)\,;\; y \in R\,;\;\; X \in R^p,\; X = \{x_1,\; x_2,\; \ldots,\; x_p\}$

$\frac{\partial y}{\partial x_i}$: partial derivative of y w. r. t to element x_i

$$\nabla_X f(X) = \begin{pmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial y}{\partial x_p} \end{pmatrix}$$ gradient of y w. r. t X

Example : $y = f(X) = x_1^2 + x_2^2 + \ldots + x_p^2$

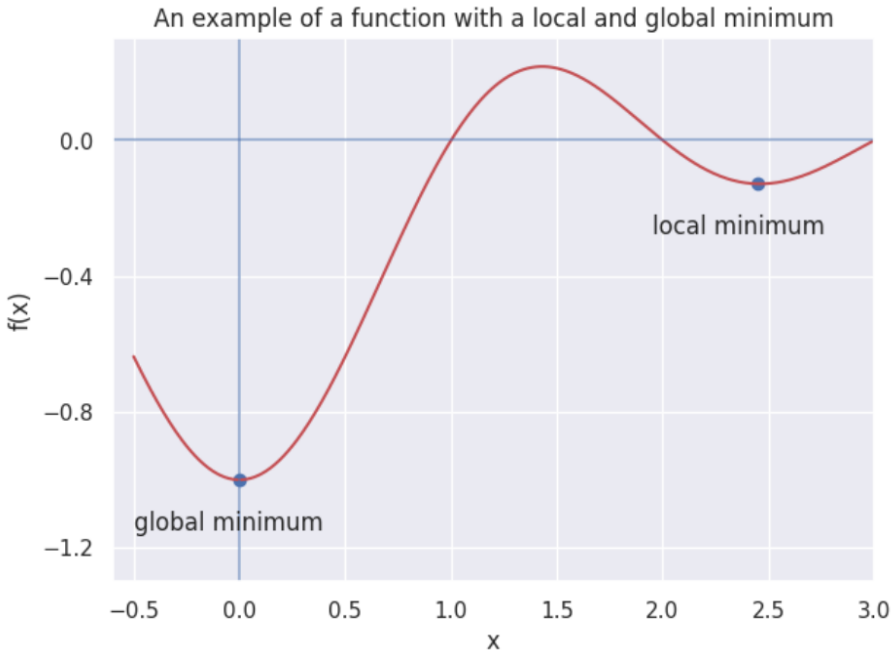
$$\nabla_X f(X) = \begin{pmatrix} 2x_1 \\ 2x_2 \\ \cdot \\ \cdot \\ \cdot \\ 2x_p \end{pmatrix} = 2 \begin{pmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_p \end{pmatrix} = 2X$$

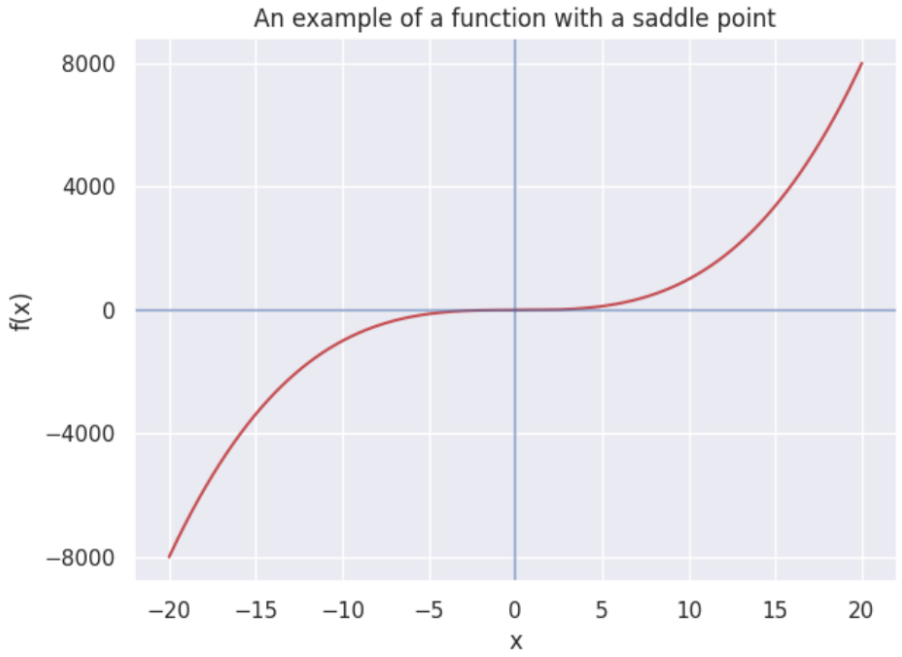
Gradient Descent

- How to find a minimum
  - An intuitive approach
  - Gradient Descent
  - Global and local minimum
- From Optimization to Learning
  - Stochastic Gradient Descent
- Improvements on SGD
  - Learning Rate Schedules
  - Momentum SGD
  - Adaptive SGD

Global and Local Minimum

- We presented the easy case where there is only one minimum.
- However, in practice we have more complex settings: local minima and saddle points.





Global and Local Minimum

- Gradient descent cannot skip local minima and saddle points.
- There is no guaranty that it will converge to a global minimum.
- This created skepticism among researchers in early days of neural networks.
- However, we found that gradient descent converges to good local minima.
- We will show later some techniques that made this possible.

Gradient Descent

- How to find a minimum
  - An intuitive approach
  - Gradient Descent
  - Global and local minimum
- From Optimization to Learning
  - Stochastic Gradient Descent
- Improvements on SGD
  - Learning Rate Schedules
  - Momentum SGD
  - Adaptive SGD

From Optimization to Learning

- We have solutions to solve linear regression and classification.
- In addition, we have an algorithm to estimate our model parameter.
- Can we say that machine learning is just solving an optimization problem?
- Well no. Let's discover what is the difference.
- We can define any model as $p(y/x; A)$ and thus write its negative log likelihood.

From Optimization to Learning

- Let's call the data-generating process p(x, y), p_{data}.
- This distribution generates the training data, and any new data in the future.
- However, we only have access to the training (observable) data.
- Let's define the empirical distribution over the training data as $\hat{p}_{data}$. Thus:

$$\text{Quantity to minimize } J(A) = -\sum_{i}^{n} \log p(Y_i / X_i;\ A) \equiv -\frac{1}{n}\sum_{i}^{n} \log p(Y_i / X_i;\ A)$$

$$J(A) = -E_{x,y \sim \hat{p}_{data}} \log p(y / x;\ A)\ ;\ E_{x,y \sim \hat{p}_{data}} : \text{expectation over random variables } x, y \text{ following } \hat{p}_{data}$$

$$\nabla_A J(A) = -E_{x,y \sim \hat{p}_{data}} \nabla_A \log p(y / x;\ A)$$

From Optimization to Learning

- This means that the optimization problem is done over $\hat{p}_{data}$.
- However, what we are interested in is to minimize the error over p_{data}.
- In other words, we want the model to be good not only on training but also on unseen data.
- This is not always the case because of a problem called overfitting (see next lesson).
- Because of this, we use a validation set in addition to the training set.

From Optimization to Learning

- In learning, we are interested in a quantity different than the one we optimize.
- We optimize the cross entropy of a classifier on a training set, but we are interested in its accuracy on q validation (unseen) set.
- Thus, the stop condition is not defined based on the gradient norm.
- We define a number of iterations to achieve a good metric (e.g. accuracy) on a validation set.

Gradient Descent

- How to find a minimum
  - An intuitive approach
  - Gradient Descent
  - Global and local minimum
- From Optimization to Learning
  - Stochastic Gradient Descent
- Improvements on SGD
  - Learning Rate Schedules
  - Momentum SGD
  - Adaptive SGD

Stochastic Gradient Descent

- The gradient to be used for descent is the expectation over all the observable data.
- However, it is very costly to compute the average gradient over all the training set.
- As we are estimating the expectation, we can use a random sample instead.
- This random sample size is known as the **batch size**.
- At each iteration, we will use a new random sample and apply gradient descent.
- A full pass over the data is called an epoch. We train for many epochs in general.

Stochastic Gradient Descent

- The batch size should be statistically significant to estimate well the gradient.
- However, does this mean that bigger batches are better? Let's delve deeper:
  - A bigger batch means, less iterations per epoch. So we should iterate for more epochs to compare.
  - A smaller batch estimate a noisy gradient, which can be useful as a regularization technique.
  - A very small batch may be problematic with others techniques like batch normalization (later lesson).
  - A twice bigger batch yields a more precise estimation by $\sqrt{2}$.
- In practice, batch size ranges between 32 and 256 based on the available memory.
- In some specific applications (like self-supervised learning) we may use huge batches.

Gradient Descent

- How to find a minimum
  - An intuitive approach
  - Gradient Descent
  - Global and local minimum
- From Optimization to Learning
  - Stochastic Gradient Descent
- Improvements on SGD
  - Learning Rate Schedules
  - Momentum SGD
  - Adaptive SGD

Improvements on SGD

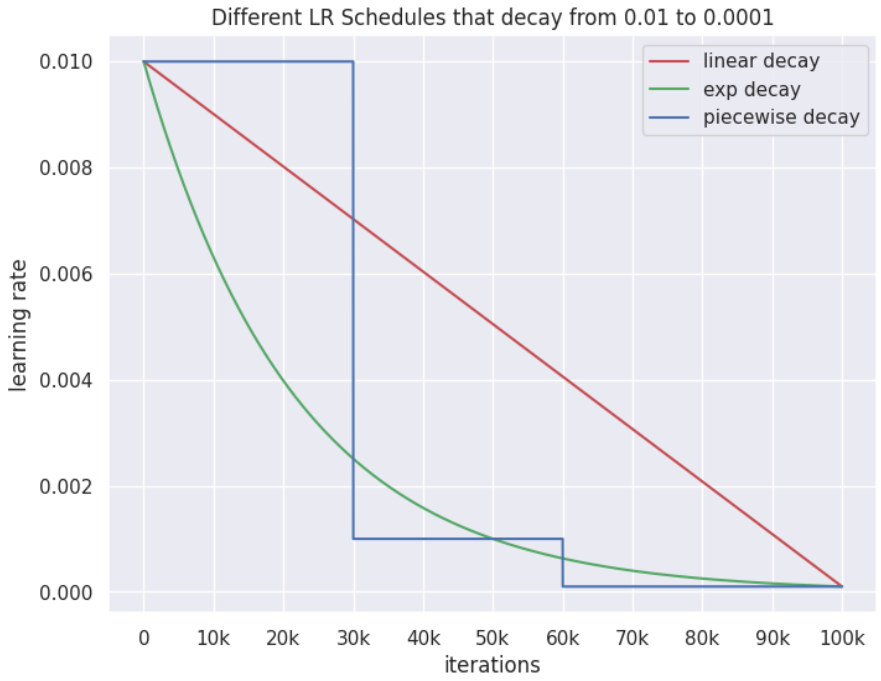
- Improvements on SGD either make it faster or try to overcome some limitations.
- The current update rule of SGD for a model with parameter A is:

$$A = A - \epsilon \hat{g};\ \epsilon:\ \text{learning rate};\ \hat{g}:\ \text{gradient estimate}$$

- A bigger learning rate means faster convergence.
- It also help to avoid local minima and saddle points.
- However, too big values can make it diverge. A big value is in the range of [0.1, 0.01].

Learning Rate Schedules

- Big learning rate are good to take faster steps and avoid bad minima.
- However, they can make the training diverge.
- One solution is to begin with a big value, then decrease with time, a.k.a scheduling.

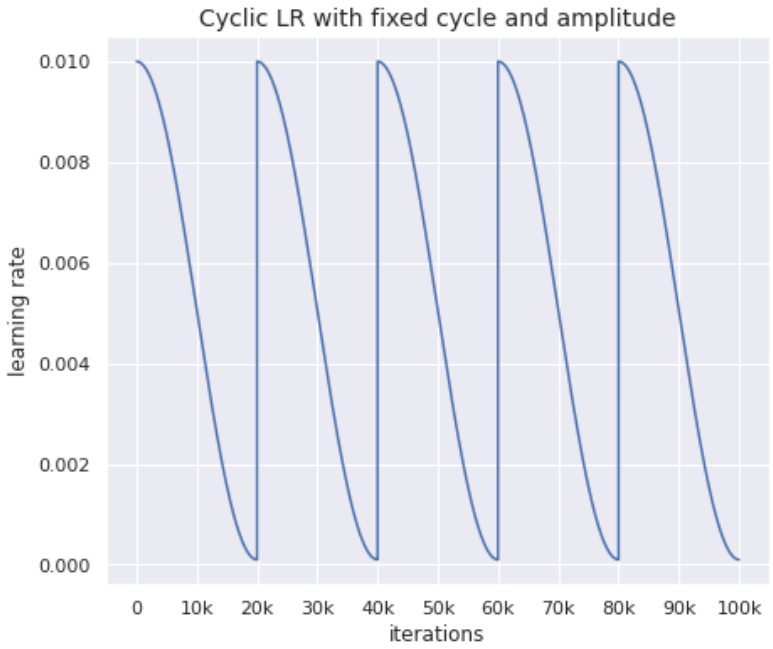


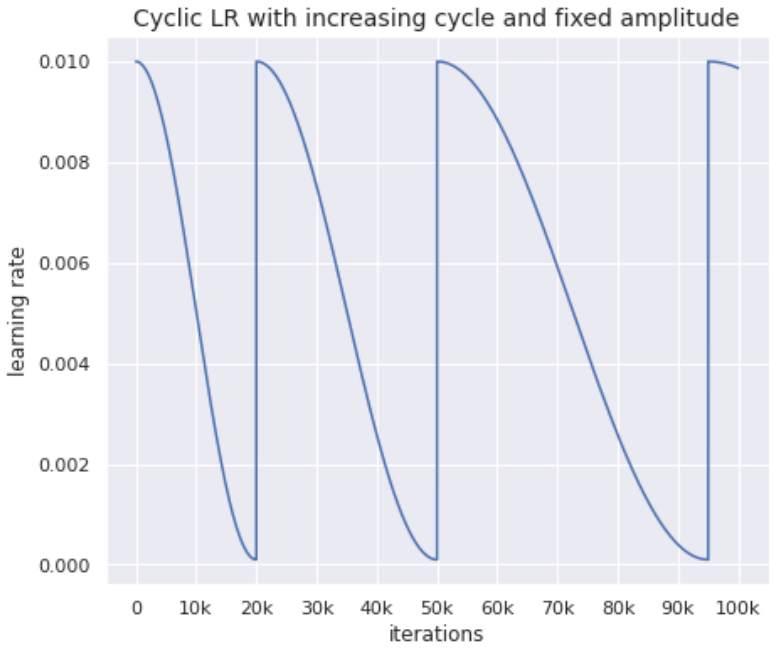
Learning Rate Schedules

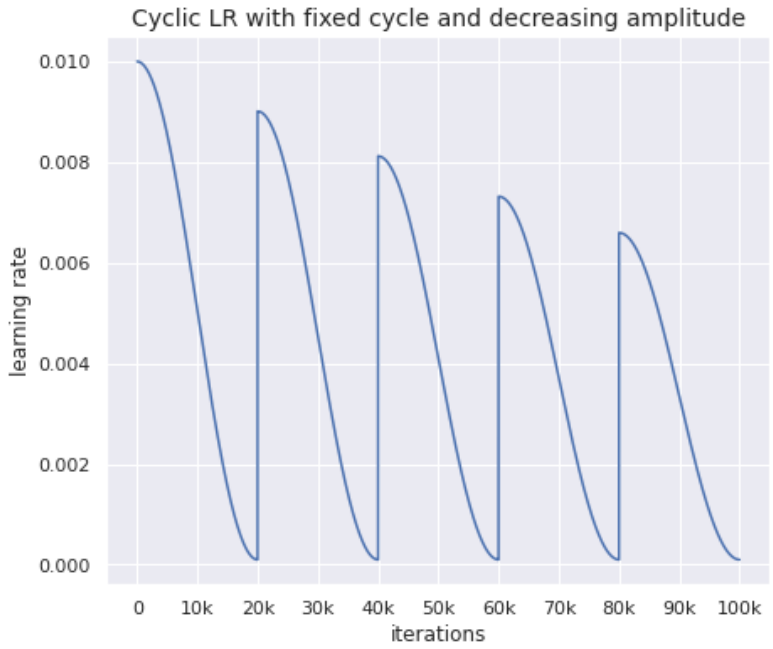
- Linear schedules decrease linearly the learning rate from a starting to an end point.
- Piecewise decay reduces suddenly the value after some iterations.
- Exponential decay decreases quickly the value at first, then continue slowly to the end.
- All schedules begin from a big value and then decrease it into a smaller one.
- A more interesting approach applies a schedule for a cycle.
- Cyclic learning rate gives the opportunity to explore a new minimum at each cycle.

Learning Rate Schedules

- They are preferred in practice, and can have a fix/variable cycle length/magnitude.
- The update rule becomes: $A = A - \epsilon_t \hat{g}$; ϵ_t : learning rate at iteration t

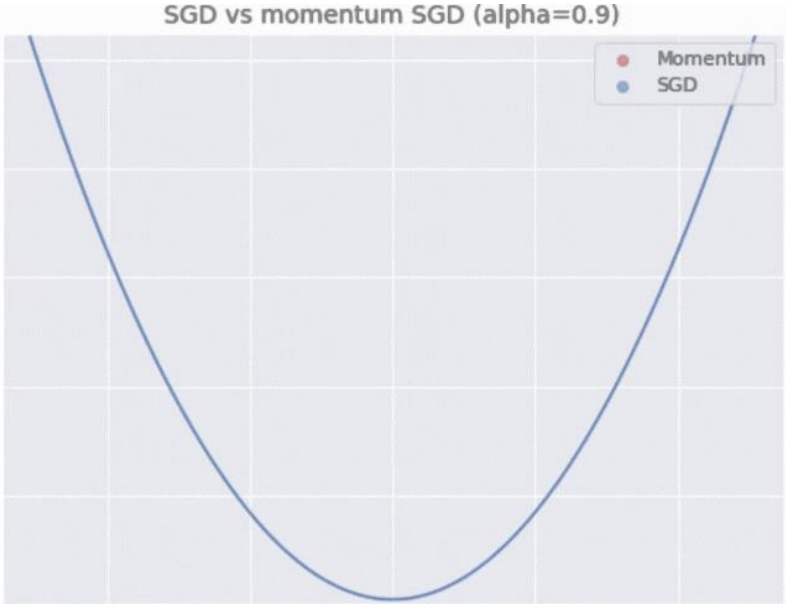


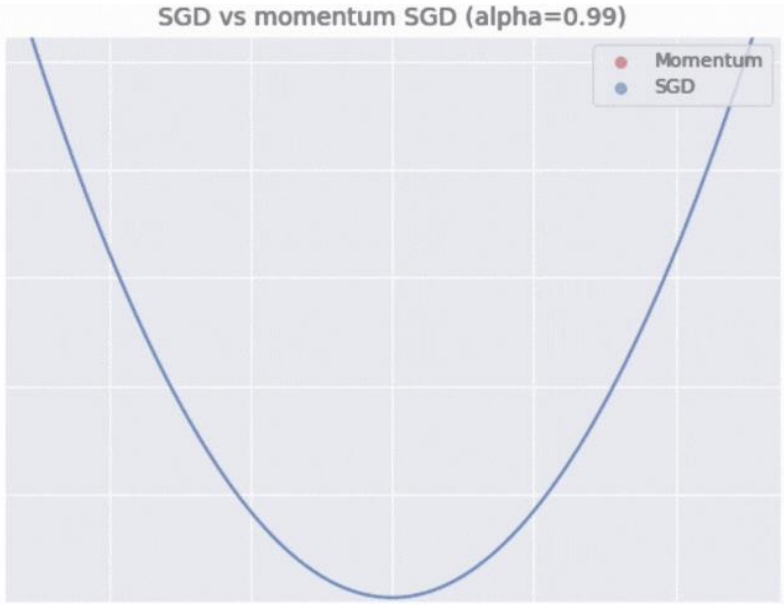




Momentum SGD

- Momentum SGD aims to increase the convergence speed of SGD. The idea is simple:
  - If with iterations, we are heading in the same direction, let's accelerate.
  - If we are changing directions, let's decelerate.





Momentum SGD

- We can define it mathematically using an exponential moving average as follows:

SGD :

$$A = A - \epsilon \hat{g}$$

$\hat{g}$: *gradient*

Momentum SGD :

$$\begin{cases} v = \alpha v - \epsilon \hat{g} \; ; \; 0 < \alpha < 1 \\ A = A + v \end{cases}$$

v : speed variable

$$\text{at } t = 0 \rightarrow v_0 = -\epsilon \hat{g}_0$$

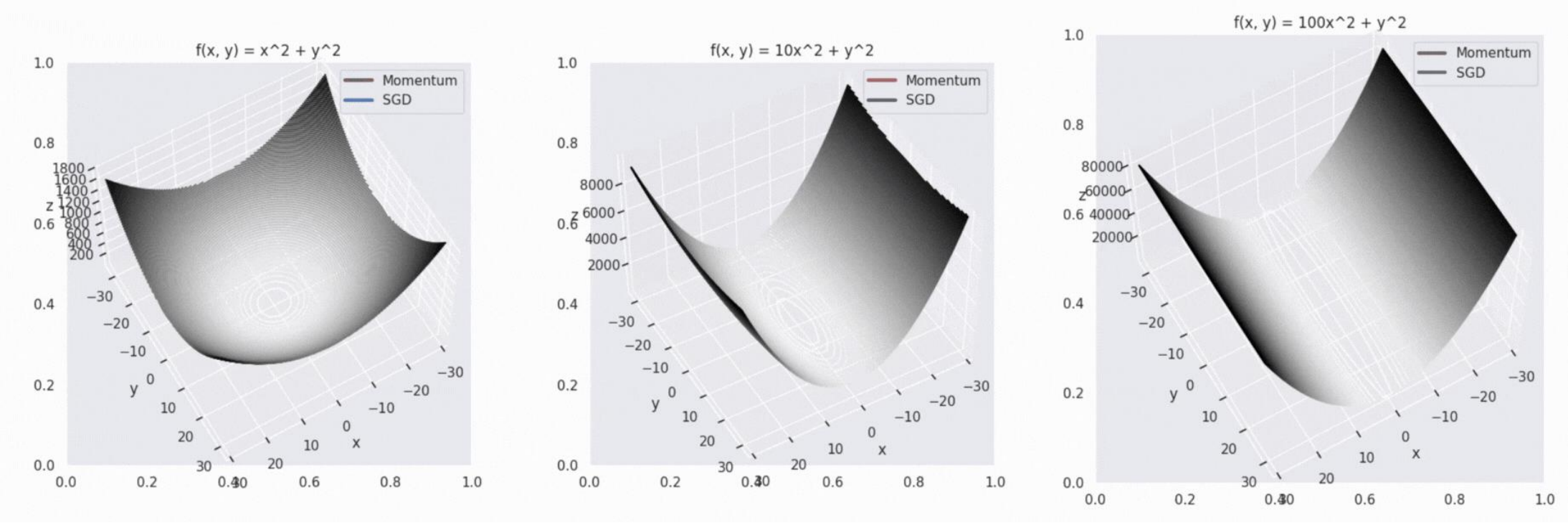
$$\text{at } t = 1 \rightarrow v_1 = \alpha v_0 - \epsilon \hat{g}_1 = -\epsilon(\alpha \hat{g}_0 + \hat{g}_1)$$

$$\text{at } t, \; v_t = -\epsilon\left(\alpha^t \hat{g}_0 + \alpha^{t-1} \epsilon \hat{g}_1 \; \ldots \; + \hat{g}_t\right)$$

- If g_t has the same sign, the update will increase. Otherwise, it will decrease.
- All past gradients contributes to the current update, with an exponentially decreasing weight a^t.

Adaptive SGD

- To understand the need for adaptive SGD, we need to take an example in 3D.
- $f_1(x, y)=x^2+y^2$; $f_2(x, y)=10x^2+y^2$; $f_3(x, y)=100x^2+y^2$.



Adaptive SGD

- In the first 2 functions, gradient descent converged. However, in the third it diverged.
- Here we are optimizing w.r.t x and y, so we will compute the gradient w.r.t them.
- grad_f_1=(2x, 2y); grad_f_2 = (20x, 2y) ; grad_f_3 = (200x, 2y).
- If we began the iterations from a point where x = y, a single update in y is equivalent to:
  - A single step in x direction for f_1.
  - 10 steps in x direction for f_2.
  - 100 steps in x direction for f_3.
- This problem is known as ill-conditioning.

Adaptive SGD

- One solution is to use a very small step. However, this will slow down much the algorithm.
- A more intelligent solution is to use a different step for each direction, an adaptive step.
- Algorithms like RMSProp and ADAM are examples of such algorithms.

RMSProp :

$$g_x = \frac{\partial f}{\partial x} ; g_y = \frac{\partial f}{\partial y} ; r_x = r_y = 0 \text{ initially} ; \rho \in]0, 1[$$

$$r_x = \rho r_x + (1-\rho) g_x^2 ; r_y = \rho r_y + (1-\rho) g_y^2 ;$$

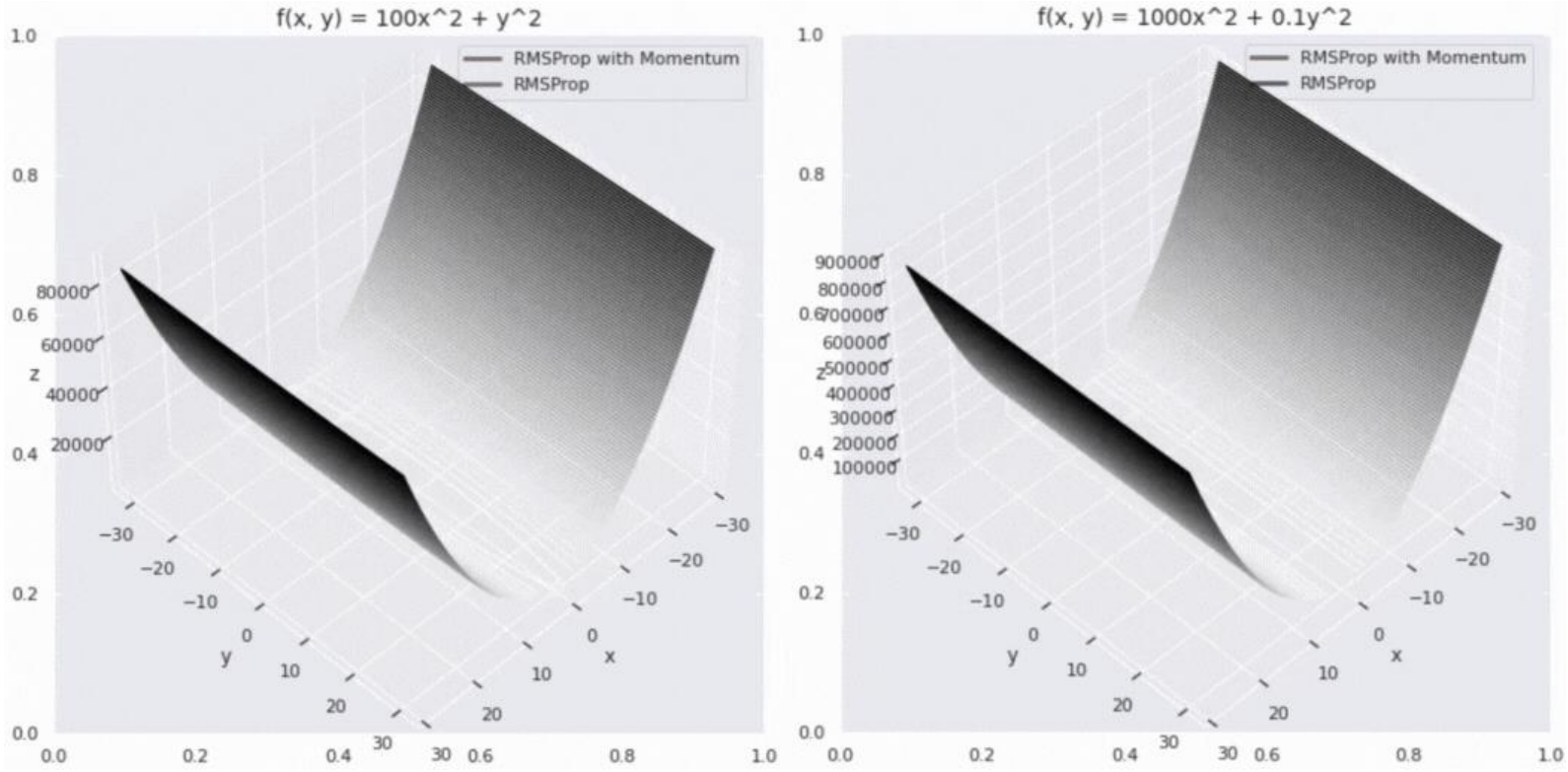
compute a moving weighted average of the gradient square.
Note that $r_x \neq r_y$

$$x = x - \frac{\epsilon}{\sqrt{r_x}} g_x ; y = y - \frac{\epsilon}{\sqrt{r_y}} g_y$$

steps in x an y directions are different and normalized with the root mean square (RMS).

Adaptive SGD

- We can use the momentum trick with them as well.
- Such algorithms can overcome severe ill-conditioning without diverging.



Exercise

- In this exercise we will solve a binary classification in Google Colab.
- We will implement logistic regression model and the gradient descent algorithm.
- We will plot the line that separates the 2 classes.
- We will compare the result with sklearn.LogisticRegression class.

Outline

- Introduction
- Gradient Descent
- Backpropagation
- Conclusion

Backpropagation

- Backpropagation is an automatic differentiation algorithm to compute gradients.
- For now, we use simple models that have single projection (matrix multiplication).
- Later with neural networks, we will have n stages of projections.
- From this arises 2 problems:
  - Manual derivation of the derivatives.
  - Redundant subexpressions in the computation.
- Backpropagation is a way to compute the gradient automatically in an efficient way.

Backpropagation

- It is based on the chain rule of calculus to compute derivatives as chain as follows:

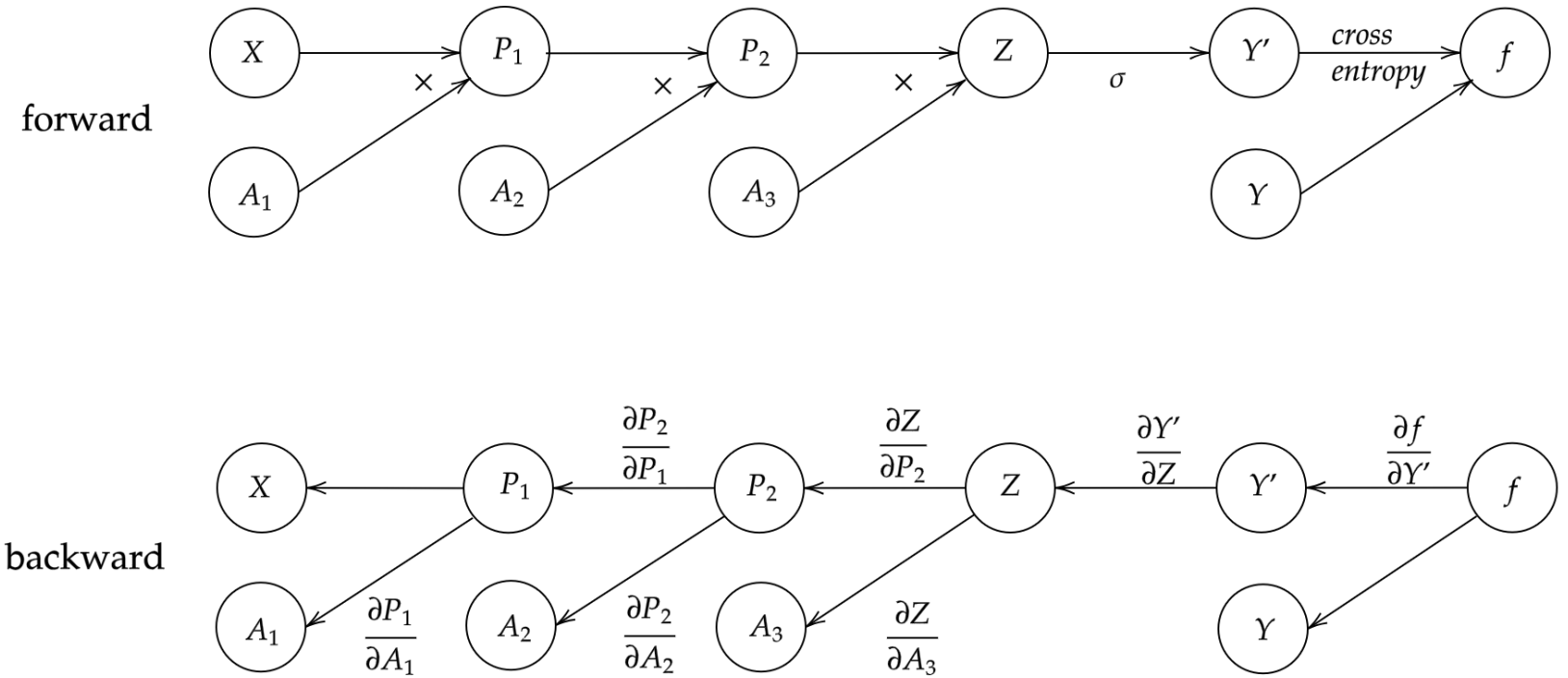
$$y = f(x)\,;\; z = g(y)\,;\; x \in R^n,\; y \in R^m,\; z \in R^p$$

$$\frac{\partial z}{\partial x} = \frac{\partial y}{\partial x} . \frac{\partial z}{\partial y}\,;\;\; \frac{\partial y}{\partial x} \in R^{(n,\,m)},\; \frac{\partial z}{\partial y} \in R^{(m,\,p)}$$

- Note that the derivative order goes from left to right.
- We can easily compute complex derivatives if we know the derivative of each term.
- It helps to create a computational graph even before seeing any data!

Backpropagation

- We can define the forward propagation (prediction) and backward (gradient computation) with a graph where nodes are variables and vertices are operations.



Backpropagation

- In Deep Learning frameworks, there are 2 types of computational graphs:
  - Static: the graph is built once, and then executed.
  - Dynamic: the graph is rebuilt at each iteration.
- Static graphs offer more possibilities to optimize the computation and make it faster.
- However, it makes debugging and implementing dynamic models harder.
- On the other hand, dynamic graphs are more flexible but lack optimization opportunity.
- TensorFlow 1 uses static graphs, however TensorFlow 2 and PyTorch uses dynamic ones.

Exercises Solution

- Solution for linear and logistic regression exercises is here.

Outline

- Introduction
- Gradient Descent
- Backpropagation
- Conclusion

Conclusion

- Gradient descent uses the **direction** and **scale** of gradient to find the steepest direction.
- Gradient descent can stuck in local minima and saddle points, but in practice it works.
- There is 2 main differences between pure optimization and learning:
  - In machine learning, we optimize a quantity different than the quantity of interest.
  - The stop condition depends on the validation performance, instead of the gradient norm.
- Stochastic gradient descent estimates the expected gradient with a random sample.
- This sample size is called batch size. It range between 32 and 256 for many applications.

Conclusion

- Many improvements were proposed to SGD:
  - Learning Rate Schedules: increase the convergence speed and help explore new local minima.
  - Momentum SGD: increase the convergence speed.
  - Adaptive SGD: overcome the problem of ill-conditioning.
- Backpropagation is used to automatically compute the gradients.
- It is also efficient because it avoid the burden of repetitive subexpressions.
- There are 2 types of computational graphs: static and dynamic.