**자주 사용되는 최적화 함수의 종류와 기능**

**목적: 어떤 목적 함수의 함수값을 최적화(최대화 또는 최소화)시키는 파라미터(변수) 조합을 찾기 위하여 사용됨.**

**More knowledge about optimize function**

-대부분의 최적화 알고리즘에는 매 반복마다 진행하는 정도를 나타내는 거리의 개념이 존재함

학습률로 이 거리를 조절하게 됨. 하지만 이 때 학습률이 지나치게 높으면 최소값을 그냥 지나쳐 지나갈 수 있으며 학습률이 지나치게 낮으면 수렴까지 너무 오래 걸리게 됨

**-기울기 소실 및 기울기 발산 문제(Vanishing and exploding gradient problem)**

1) Vanishing gradient problem

🡺 Gradient-based learning methods나 backpropagation을 이용한 Artificial neural networks에서 가끔 보이는 문제점. 이러한 method들에서, each of the NN’s weights receives on update proportional to the partial derivative of the error function with respect to the current weight in each iteration or training. The problem is that in some case, the gradient will be vanishingly small, effectively preventing the weight from changing its value. (최소값이 지나치는 경우에 해당)

Iteration을 하면서 weight를 보존하기 위해서 값을 조절하면서 생기는 loss가 Gradient에 미치는 영향으로 이해하면 쉬움

이 경우의 반대 (최대값이 지나치는 경우)가 exploding gradient problem

**Solution**

1) Multi-level hierarchy: 비지도 학습으로 미리 one level 학습이 된 network를 이용. 역전파를 통해 fine-turn이 됨

2) Related approach: Similar ideas have been used in feed-forward neural network for unsupervised pre-training to structure a neural network, making it first learn generally useful [feature detectors](https://en.wikipedia.org/wiki/Feature_detection_(nervous_system)). Then the network is trained further by supervised [back-propagation](https://en.wikipedia.org/wiki/Back-propagation) to classify labeled data. The [deep belief network](https://en.wikipedia.org/wiki/Deep_belief_network)model by Hinton et al. (2006) involves learning the distribution of a high level representation using successive layers of binary or real-valued [latent variables](https://en.wikipedia.org/wiki/Latent_variable). It uses a [restricted Boltzmann machine](https://en.wikipedia.org/wiki/Restricted_Boltzmann_machine) to model each new layer of higher level features. Each new layer guarantees an increase on the [lower-bound](https://en.wikipedia.org/wiki/Lower_bound) of the [log likelihood](https://en.wikipedia.org/wiki/Log_likelihood) of the data, thus improving the model, if trained properly. Once sufficiently many layers have been learned the deep architecture may be used as a [generative model](https://en.wikipedia.org/wiki/Generative_model) by reproducing the data when sampling down the model (an "ancestral pass") from the top level feature activations

3) Long short-term memory: Another tech particularly used for recurrent neural network (순환신경망) is the long short-term memory(LSTM)

\*LSTM: Long short-term memory (LSTM) is an artificial [recurrent neural network](https://en.wikipedia.org/wiki/Recurrent_neural_network) (RNN) architecture used in the field of [deep learning](https://en.wikipedia.org/wiki/Deep_learning). Unlike standard [feedforward neural networks](https://en.wikipedia.org/wiki/Feedforward_neural_network), LSTM has feedback connections that make it a "general purpose computer" (that is, it can compute anything that a [Turing machine](https://en.wikipedia.org/wiki/Turing_machine) can). It can not only process single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks such as unsegmented, connected [handwriting recognition](https://en.wikipedia.org/wiki/Handwriting_recognition) or [speech recognition](https://en.wikipedia.org/wiki/Speech_recognition). [Bloomberg Business Week](https://en.wikipedia.org/wiki/Bloomberg_Business_Week) wrote: "These powers make LSTM arguably the most commercial AI achievement, used for everything from predicting diseases to composing music."

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three *gates* regulate the flow of information into and out of the cell.

LSTM networks are well-suited to [classifying](https://en.wikipedia.org/wiki/Classification_in_machine_learning), [processing](https://en.wikipedia.org/wiki/Computer_data_processing) and [making predictions](https://en.wikipedia.org/wiki/Predict) based on [time series](https://en.wikipedia.org/wiki/Time_series) data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the exploding and [vanishing](https://en.wikipedia.org/wiki/Vanishing_gradient_problem) gradient problems that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs, [hidden Markov models](https://en.wikipedia.org/wiki/Hidden_Markov_models) and other sequence learning methods in numerous application

* Want more : <https://en.wikipedia.org/wiki/Long_short-term_memory>

4) Faster hardware

5) Residual networks(ResNets): One of the newest and most effective ways to resolve the vanishing gradient problem is with residual neural networks, ResNets, not to be confused with recurrent neural networks. It was noted prior to ResNets that a deeper network would actually have higher *training* error than the shallow network. This intuitively can be understood as data disappearing through too many layers of the network, meaning output from a shallow layer was diminished through the greater number of layers in the deeper network, yielding a worse result. Going with this intuitive hypothesis, Microsoft research found that splitting a deep network into three layer chunks and passing the input into each chunk straight through to the next chunk, along with the residual-output of the chunk minus the input to the chunk that is reintroduced, helped eliminate much of this disappearing signal problem.] No extra parameters or changes to the learning algorithm were needed. ResNets yielded lower training error (and test error) than their shallower counterparts simply by reintroducing outputs from shallower layers in the network to compensate for the vanishing data.

Note that ResNets are an ensemble of relatively shallow Nets and do not resolve the vanishing gradient problem by preserving gradient flow throughout the entire depth of the network – rather, they avoid the problem simply by constructing ensembles of many short networks together.

\*HighwayNets : ResNets With an additional weight matrix to learn the skip weight it is referred to as Highway Nets

\*DenseNets : ResNets with several parallel skips it is referred to as DensNets

\*Non-resodual neural network: Described as a plain network in the context of residual neural networks

6) Using ReLU 🡪 They only saturate in one direction

**About** **optimization function**

**1) Gradient Descent Optimization Algorithms**

**-Neural network의 weight를 조절하는 과정에서 ‘Gradient Descent method’를 사용하는 방법. 네트워크의 parameter를 output과 target 사이의 차이를 정의하는 함수인 Loss function**  **의 값을 최소화하기 위해 기울기**  **를 이용하는 방법. GD에서는**  **에 대해 gradient의 반대 방향으로 일정 크기만큼(learning rate) 이동해내는 것을 반복하여 Loss function** **의 값을 최소화하는** **를 찾음**

**🡪 한 iteration에서의 변화식**

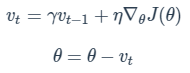
**이때** **는 Step size(learning rate)로 보통 0.01~0.001 정도로 잡게 됨**

**-1) Batch GD = All train set을 이용하여 loss function을 계산하는 것**

**-2) Stochastic GD = Mini-batch 만큼 (한번의 Step 당) loss function을 계산하는 것. Iteration이 많이 일어날수록 그 값은 Batch GD에 수렴하게 되며 local minima에 빠지는 것을 방지하기 위하여 사용됨**

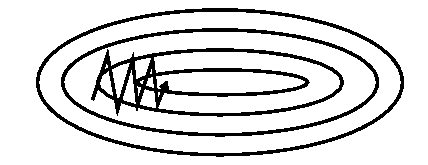
**단, SGD(Naïve Stochastic Gradient Descent 알고리즘)의 경우 다른 SGD의 변형인 Momentum, NAG, Adagrad, AdaDelta, RMSprop 등보다 이동속도도 느리며 방향을 제대로 못 잡기도 하고 이상한 방향으로 가는 경우가 많음 (by Sebastian Ruder)**

**2) Momentum**

**Gradient Descent Optimization을 통해 이동하는 과정에 일종의 ‘관성’을 넣어주는 것으로 현재 Gradient를 통해 이동하는 방향과는 별개로, 과거에 이동했던 방식을 기억하면서 그 방향으로 일정 정도를 추가적으로 이동하는 방식임.** **를 time step에 t에서의 벡터 이동이라고 하였을 때,** 

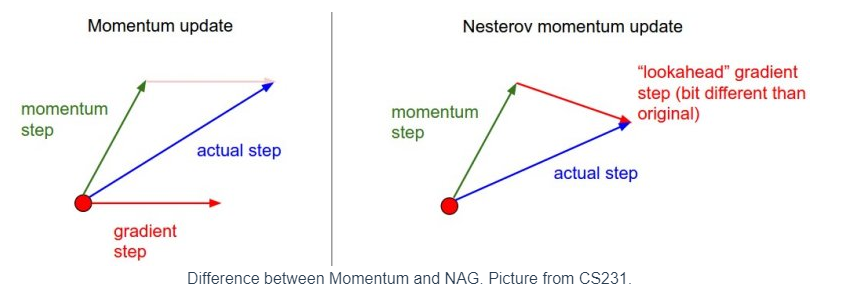
**을 만족한다. 이때** **는 Momentum term으로 Momentum의 정도를 나타내며, 보통 0.9 정도를 사용하게 된다. 위 식은 간단히 말하자면 과거에 얼마나 이동했는지에 대한 이동한 방향 w를 기억하고, 새로운 이동항을 구할 경우 과거에 이동했던 정도에 관성 값만큼 곱해준 뒤에 이를 Gradient을 이용한 이동 Step항을 더해주는 것이다. 그러면 이 식은** **와 같이 표현될 수 있으며 Gradient들이 지수평균을 이용하여 이동한다고 해석할 수 있다.**

**이러한 Momentum 방식은 SGD가 Oscilation 현상을 겪고 있을 때 사용된다. 한번의 Step으로 이동할 수 있는 Step Size는 한계가 있기 때문에 이러한 현상이 일어나게 될 때 이동은 좌우로 계속 진동하게 된다. 이러한 경우 Momentum 방식은 자주 이동하는 방향으로 관성이 걸리게 되고 이러한 관성에 의해 진동이 발생하더라도 Optimize로 가는 방향에 힘을 얻기 때문에 SGD에 비해서 상대적으로 빠르게 이동이 가능하다는 장점을 가진다. 또한 local minima에서 빠져나오는 것 또한 가능하게 해준다. 다만 이 방법은 보통의 방법보다 2배의 메모리가 필요하다. 아래는 Oscilation이 발생한 경우이다.**

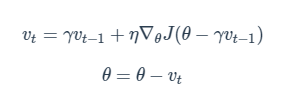


**3) NGA(Nesterov Accelerated Gradient)**

**NAG는 Momentum방식을 기초로 한 방식이지만, Gradient를 계산하는 방식이 좀 다르다.**



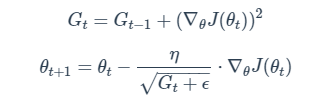
**Momentum에서는 이동벡터를 계산할 때 현재 위치에서의 gradient와 momentum step을 독립적으로 계산하고 합친다. 반면, NAG에서는 momentum step을 먼저 고려하여, Momentum step을 먼저 이동했다고 가정한 뒤 그 자리에서 gradient를 구해서 gradient step을 이동한다.**



**Momentum 방식의 경우 멈춰야 할 시점에서도 관성에 의해서 훨씬 멀리 갈 수도 있다는 단점이 존재하지만, NAG 방식의 경우 일단 모멘텀으로 이동을 반 정도 이동한 후 어떤 방식으로 이동해야 할 지를 결정하게 된다. 따라서 momentum 보다 빠르면서도 적절한 시점에서 제동을 거는 데에 훨씬 용이하다.**

**4) Adagrad(Adaptive Gradient)**

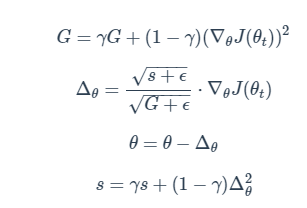
**Adagrad는 변수들을 update할 때 각각의 변수마다 step size를 다르게 설정하여 이동하는 방식이다. 이 알고리즘의 기본 아이디어는 ‘지금까지 많이 변화하지 않은 변수들은 Step size를 크게 하고, 지금까지 많이 변화했던 변수들은 Step size를 작게 하자’이다. 자주 등장하거나 변화를 많이 한 변수들의 경우 optimum에 가까이 있을 확률이 높기 때문에 작은 크기로 이동하면서 세밀하게 값을 조절하고, 그 반대의 경우 optimum에 멀리 있을 확률이 높기 때문에 빠르게 loss값을 줄이는 방향으로 이동하는 방식이다. 특히 word2vec이나 GloVe 같이 word representation을 학습시킬 경우 단어의 등장 확률에 따라 variable의 사용 비율이 확연하게 차이가 나기 때문에 이 방법을 사용한다. Adagrad의 한 스텝은 다음과 같다.**



**Neural Network의 Parameter가 K개라고 할 때 Gt는 K차원 벡터로서 ‘time step t까지 각 변수가 이동한 gradient의 sum of square’를 저장한다.** **를 업데이터 하면서 기존 step size에 Gt의 루트값에 반비례한 크기로 이동을 진행하여, 지금까지 많이 변화한 변수일수록 적게 이동하고 적게 변화한 변수일수록 많이 이동하도록 한다. 이때** **은 0으로 나뉘어지는 것을 방지하기 위한 작은 값이다. Adagrad를 사용하면 학습을 진행하면서 굳이 step size decay등을 신경 써주지 않아도 된다는 장점이 있다. 다만 문제는 학습이 오래 진행될수록 step size가 너무 작아져서 결국에는 움직이지 않게 된다.**

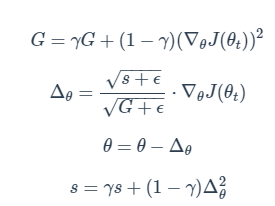
**5) RMSProp**

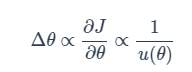
**이 방법은 제프리 힌톤이 제안한 방법으로, Adagrad의 단점을 해결하기 위한 방법이다. Adagrad의 식에서 gradient의 제곱값을 더해나가면서 Gt 부분을 합이 아니라 지수평균으로 바꾸어 대체한 방법이다. 이렇게 할 경우 Gt가 무한정 커지지 않으면서 최근 변화량의 변수간 상대적인 크기 차이를 유지할 수 있다.**

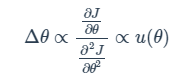


**6) AdaDelta**

**AdaDelta(Adaptive Delta)는 RMSProp와 유사하게 AdaGrad의 단점을 보완하기 위해 제안된 방법이다. AdaDelta는 PMSProp와 동일하게 G를 구할 때 합을 구하는 대신 지수평균을 구한다. 다만, 여기에서는 step size를 단순하게** **로 사용하는 대신 Step size의 변화값의 제곱을 가지고 지수평균 값을 사용한다.**

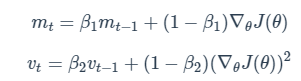


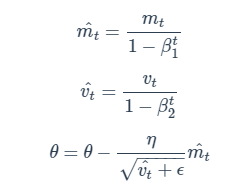
**사실 이 방법은 First-order optimization이 아니라 Second-order optimization을 approximate 하기 위한 방법이다. 실제로 AdaDelta 방법을 제시한 논문에서 저자는 SGD, Momentum, Adagrad와 같은 식들의 경우** **를 구해보면** **가 아니라** **의 역수를 따른다는 사실에 대해서 지적함. 따라서** **을** **라고 하고 J는 unit이 없다라고 생각 할 경우 first-order optimization에서는** **이 된다.**

**반면, Newton method와 같은 second-order optimization의 경우** **이 되어 바른 Unit을 가지게 된다. 그 방법은 이 사실을 기반으로 만들어진 최적화 함수이다.**

**7)Adam (AdamOptimizer)**

**Adam(Adaptive Moment Estimation)은 RMSProp와 Momentum의 방식을 합친 것 같은 알고리즘이다. 이 방식에서는 Momentum 방식과 유사하게 지금까지 계산해온 기울기의 지수평균을 저장하며, RMSProp와 유사하게 기울기의 제곱값의 지수평균을 저장한다.**



**다만, Adam에서는 M과 V가 처음에는 0으로 초기화 되어 있기 때문에 학습의 초반부에서는** **형태로 펼친 후 양변에 expectation을 씌어서 정리해보면, 다음과 같은 보정을 통해 unbiased된 expectation을 얻을 수 있다. 이 보정된 expectation을 가지고 gradient가 들어가는 자리에****,** **가 들어갈 자리에** **을 넣어 계산을 진행한다. 그렇게 하면,** 

**보통, β1 로는 0.9, β2 로는 0.999, ε으로는 10^-8 정도의 값을 사용한다.**

**\*학습률 크기에 따른 장점/단점**

|  |  |  |
| --- | --- | --- |
| **학습률 크기** | **장점/단점** | **사용 방법** |
| **좀 더 작은 학습률** | **수렴이 더 느리지만, 결과가 더 안정적임** | **안정적인 답이 나오지 않으면 학습률을 낮추는 것을 먼저 시도해 본다** |
| **좀 더 큰 학습률** | **결과가 덜 정확하지만, 더 빨리 수렴됨** | **경우에 따라 답이 정체에 빠지는 문제를 해결해 줄 수 있다.** |

**\*참고**

<http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html>

**: 간략한 한국어 설명 및 가장 많이 사용되는 최적화 함수에 대한 설명**

<http://ruder.io/optimizing-gradient-descent/>

: **영어로 되어 있지만 좀 더 자세한 설명을 제공해 주고 그 밖의 여러가지 최적화 함수에 대하여 잘 설명해 주고 있음. [최적화 함수에 대하여 어느정도 알고 있다면 이 사이트를 추천]**

<https://darkpgmr.tistory.com/142>

: **흔히 자주 쓰이기 때문에 여기서는 오로지 Gradient Descent optimizer만 정리했는데 그 밖에 사용되는 최적화 함수에 대한 설명들이 나와 있음**

<https://darkpgmr.tistory.com/149> : **최적화 기법에 대한 자세한 이해가 가능하게 해주는 설명**