Parrot Deep Learning
Session 03. Optimizer

지난 시간 리뷰

주어진 데이터에 대해 원하는 출력을 생성하는 모델을 훈련시키려고 합니다.

Input

Model

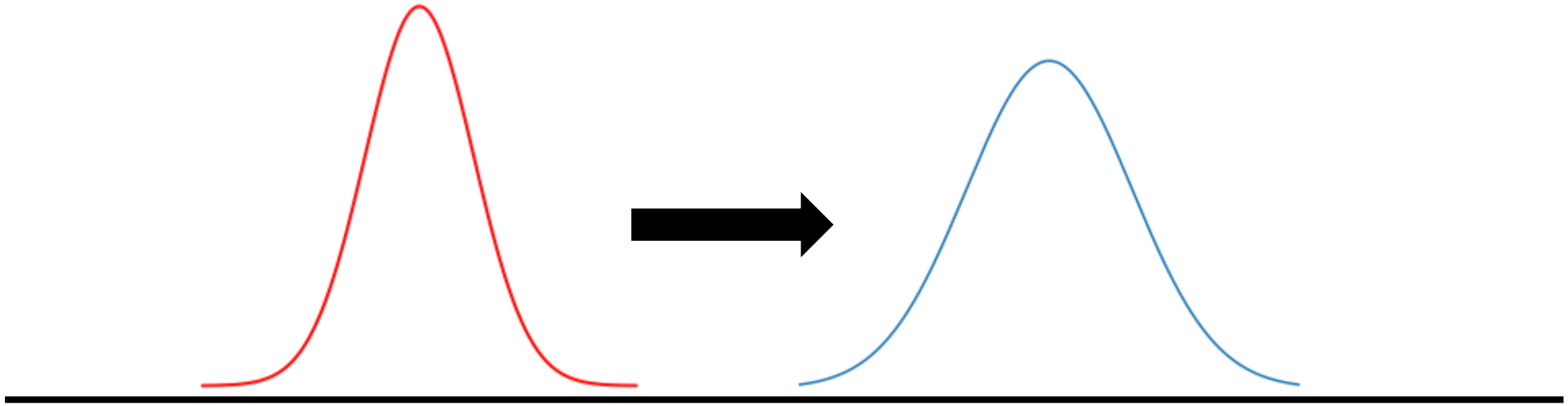
Dog

Output

지난 시간 리뷰

그런데 “*모든 데이터는 하나의 분포에서 나온다*” 라는 점을 생각하였을 때,

내가 만든 모델의 분포 P_{model}이 실제 데이터의 분포 P_{data} 와 같아지도록 하면 됩니다.



지난 시간 리뷰

How? 모든 샘플에 대한 Likelihood를 최대화

➡ ***Maximum Likelihood Estimation (MLE)***

기억이 안 나신다면 어떤 데이터가 현재 내가 만든 모델의 분포에서 나왔을 확률(정확히는 likelihood)를
최대화 시키는 방향으로 모델의 파라미터를 조정한다고 생각하면 됩니다.

$$\hat{\mathbf{w}} = \arg\max_{\mathrm{w}} p_{model}(\mathrm{X}|\mathrm{w})$$

뭔가 무서운 수식이 보이지만 현재 파라미터 W가 주어졌을 때 데이터 X가 나왔을 확률을 최대로 하는 W를 고른다는 의미입니다.

지난 시간 리뷰

그런데 데이터들은 independent 하고 identically 하게 뽑히므로...

$$\hat{\mathbf{w}} = \arg\max_{\mathrm{w}} p_{model}(\mathrm{X}|\mathrm{w})$$

$$\overset{\text{iid}}{=} \arg\max_{\mathrm{w}} \prod_{i=1}^{N} p_{model}(x_i|\mathrm{W})$$

(1) 각각의 likelihood가 곱셈으로 묶이고 (독립사건 생각하시면 편합니다)

$$= \arg\max_{\mathrm{w}} \sum_{i=1}^{N} \log p_{model}(x_i|\mathrm{W})$$

(2) 로그를 씌우면 합으로 나타낼 수 있습니다.

"로그를 씌웠는데 왜 똑같나요?" 라고 생각할 수 있지만

여기서 잘 보시면 우리가 원하는 것은 이 친구들(likelihood)을 최대로 하는 W이기 때문에 등식이 성립합니다.

지난 시간 리뷰

보통 로그를 씌운 likelihood를 사용합니다 (Log likelihood)

여기서 모델의 분포함수를 무엇을 쓰는지에 따라 loss 함수 형태가 결정

$$\hat{\mathbf{w}} = \arg\max_{\mathrm{w}} p_{model}(\mathrm{X}|\mathrm{w})$$

$$\overset{\text{iid}}{=} \arg\max_{\mathrm{w}} \prod_{i=1}^{N} p_{model}(x_i|\mathrm{W})$$

$$= \arg\max_{\mathrm{w}} \sum_{i=1}^{N} \log p_{model}(x_i|\mathrm{W})$$

Laplace - L1 loss

$$p(y) = \frac{1}{2b} e^{(-\frac{|y-\mu|}{b})}$$

Gaussian – L2 loss

$$p(y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{(-\frac{(y-\mu)^2}{2\sigma^2})}$$

이유...? (참고만 하세요!)

$$\arg\max_{\mathrm{w}} \sum_{i=1}^{N} \log p_{model}(x_i|\mathrm{W})$$

시작은 항상 log likelihood를 최대화 시키는 것으로 시작합니다

Laplace - L1 loss

$$p(y) = \frac{1}{2b} e^{(-\frac{|y-\mu|}{b})}$$

$$\begin{aligned}
\hat{\mathbf{w}}_{ML} &= \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=1}^{N} \log p_{model}(y_i|\mathbf{x}_i, \mathbf{w}) \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} -\sum_{i=1}^{N} \log(2b) - \sum_{i=1}^{N} \frac{1}{b}|f_{\mathbf{w}}(\mathbf{x}_i) - y_i| \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} -\sum_{i=1}^{N} |f_{\mathbf{w}}(\mathbf{x}_i) - y_i| \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{N} \underbrace{|f_{\mathbf{w}}(\mathbf{x}_i) - y_i|}_{L_1 \text{ Loss}}
\end{aligned}$$

Gaussian – L2 loss

$$p(y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{(-\frac{(y-\mu)^2}{2\sigma^2})}$$

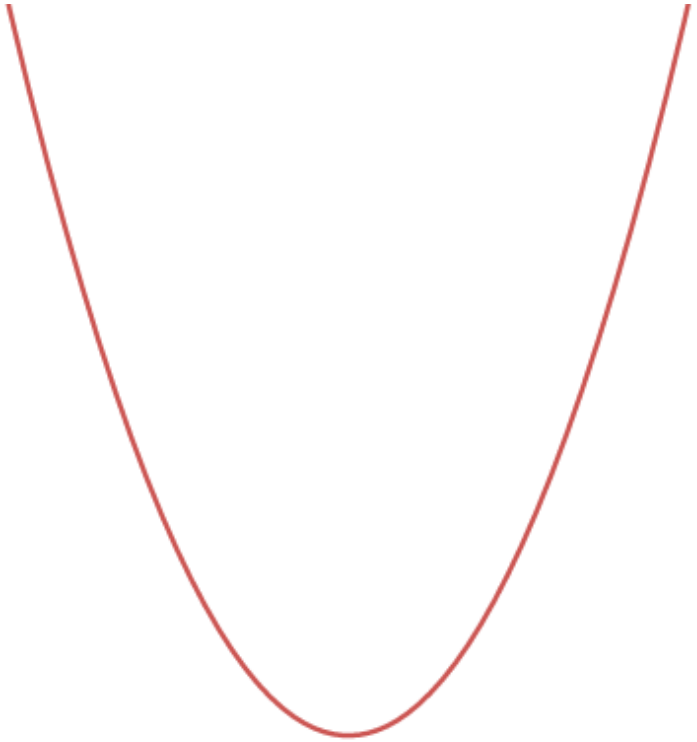
$$\begin{aligned}
\hat{\mathbf{w}}_{ML} &= \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=1}^{N} \log p_{model}(y_i|\mathbf{x}_i, \mathbf{w}) \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} -\sum_{i=1}^{N} \frac{1}{2}\log(2\pi\sigma^2) - \sum_{i=1}^{N} \frac{1}{2\sigma^2}(f_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2 \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} -\sum_{i=1}^{N} (f_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2 \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{N} \underbrace{(f_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2}_{L_2 \text{ Loss}}
\end{aligned}$$

이제 loss 함수가 어떻게 나왔고

미분하는 방법 (Back propagation)도 알고 있습니다.

이제 loss 함수만 줄이면 됩니다!

Optimization



1. 미분해서 0이 되는 점 찾기

~~2. 계속 찍기~~ (과연 제가 그냥 농담으로 넣은 걸까요)

Optimization

1. 미분해서 0이 되는 점 찾기

미분해서 0이 되는 점을 찾으면 한번에 답을 구할 수 있습니다.

그런데 다음 조건을 만족해야 합니다.

1. 함수가 quadratic하다. (2차 함수 생각하시면 됩니다)
2. convex 함이 보장 (아래로 볼록 생각하시면 됩니다)

불행히도 대다수의 loss 함수에서 위 조건들이 보장이 안됩니다.

Optimization

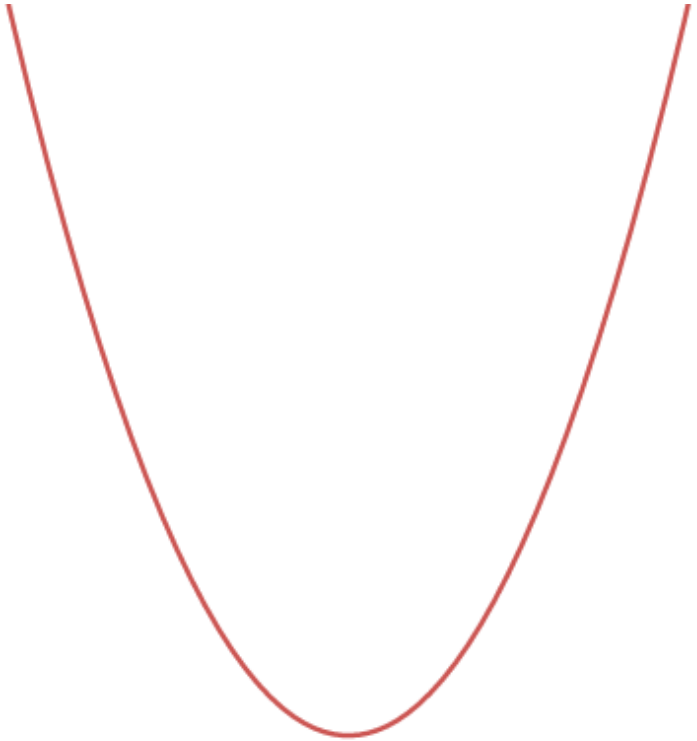
1. 미분해서 0이 되는 점 찾기 (이계도함수를 기억하시는 분들 한정)

예전 고등학교에서는 n차함수에 대하여

1) 미분해서 0이 되는 지점들을 구하고
2) 한번 더 미분해서 이계도함수가 0보다 크면 극솟값 (local minima)이 되고
3) (구간이 정해져 있다면) 그 안에서 가장 작은 값이 최솟값 (global minima) 라고 구했습니다.

불행히도 이계도함수 (Hessian matrix) 구하는 cost가 상당히 큽니다...

(dataset size를 n, parameter size를 p라고 할 때 $O(np^2 + p^3)$, 최적화하면 $O(np + rt * p)$ (여기서 $rt \sim O(n)$) 입니다.)



~~**1. 미분해서 0이 되는 점 찾기**~~

2. 계속 찍기

물론 무작정 찍지 말고 '잘' 찍으면 됩니다.

Gradient Descent

(간략화 버전)

1. 현재 위치(W)에서 loss 함수가 줄어드는 방향을 계산합니다.

2. 해당 방향으로 사전에 정한 값 (학습율이라고 합니다) 만큼 움직입니다.

3. 위 과정을 계속해서 반복합니다.

여기서 기울기를 Gradient 라고 하고,

위 방식을 Gradient Descent(GD) 라고 합니다.

Gradient Descent

(디테일 버전)

1. 가중치를 초기화하고(W^0), 학습률 η을 정합니다.
2. 모든 데이터포인트 $i \in \{0, \ldots, N\}$에 대하여 다음을 시행합니다.

   a) forward propagate를 통해 현재 prediction을 구합니다

   b) 이를 사용해서 Backpropagation으로 현재 gradient를 구합니다.
3. 가중치를 다음과 같이 업데이트 합니다.

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{1}{N} \sum_i \nabla_{\mathbf{w}} \mathcal{L}_i(\mathbf{w}^t)$$

4. 위 과정을 계속해서 반복합니다.

하지만 모든 데이터포인트에 대해 진행하는 점을 생각해봅시다.

일반적으로 수십만 되는 데이터를 사용하고,

모델의 파라미터 또한 일반적으로 상당히 큽니다.

따라서 한번의 과정 (iteration)에서 cost가 상당히 큽니다.

Stochastic Gradient Descent

굳이 모든 데이터에 대해 계산하지 말고 좀 더 작은 subset인 Batch, *B* 를 랜덤하게 뽑아서 계산합시다.

1. 가중치를 초기화하고(W^0) 학습률 η을 정합니다.
2. 랜덤하게 뽑은 데이터포인트 $i \in \{0, \dots, B\}$에 대하여 다음을 시행합니다.
   a) forward propagate를 통해 현재 prediction을 구합니다
   b) 이를 사용해서 Backpropagation으로 현재 gradient를 구합니다.
3. 가중치를 다음과 같이 업데이트 합니다.

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{1}{B} \sum_b \nabla_{\mathbf{w}} \mathcal{L}_b(\mathbf{w}^t)$$

4. 위 과정을 계속해서 반복합니다.

이제 계산 cost뿐만 아니라 메모리도 덜 차지합니다.

Stochastic Gradient Descent

이거 왜 되나요...? (역시 참고용 입니다)

- 전체 훈련셋에서의 total loss는 다음과 같이 표현된다.

$$\frac{1}{N}\sum_i \mathcal{L}_i(\mathbf{w}^t) = \mathbb{E}_{i\sim\mathcal{U}\{1,N\}}\left[\mathcal{L}_i(\mathbf{w}^t)\right]$$

- 이런 Expectation 값은 훨씬 작은 subset 인 batch, B에 근사된다.

$$\mathbb{E}_{i\sim\mathcal{U}\{1,N\}}\left[\mathcal{L}_i(\mathbf{w}^t)\right] \approx \frac{1}{B}\sum_b \mathcal{L}_b(\mathbf{w}^t)$$

- 따라서 다음 식이 성립하게 되기에 전체 gradient를 batch의 gradient로 사용할 수 있다.

$$\frac{1}{N}\sum_i \nabla_{\mathbf{w}}\mathcal{L}_i(\mathbf{w}^t) = \mathbb{E}_{i\sim\mathcal{U}\{1,N\}}\left[\nabla_{\mathbf{w}}\mathcal{L}_i(\mathbf{w}^t)\right] \approx \frac{1}{B}\sum_b \nabla_{\mathbf{w}}\mathcal{L}_b(\mathbf{w}^t)$$

Learning rate

1. 가중치를 초기화하고(W^0) **학습률 η을 정합니다.**
2. 랜덤하게 뽑은 데이터포인트 $i \in \{0, \ldots, B\}$에 대하여 다음을 시행합니다.
   a) forward propagate를 통해 현재 prediction을 구합니다
   b) 이를 사용해서 Backpropagation으로 현재 gradient를 구합니다.
3. 가중치를 다음과 같이 업데이트 합니다.

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{1}{B} \sum_b \nabla_{\mathbf{w}} \mathcal{L}_b(\mathbf{w}^t)$$

4. 위 과정을 계속해서 반복합니다.

학습률을 어떻게 정하는 지를 생각해봅시다.

가중치 초기화 하는 방식도 다양하긴 한데 Batch Normalization이 등장함에 따라 그 중요도가 낮아져서 생략했습니다.

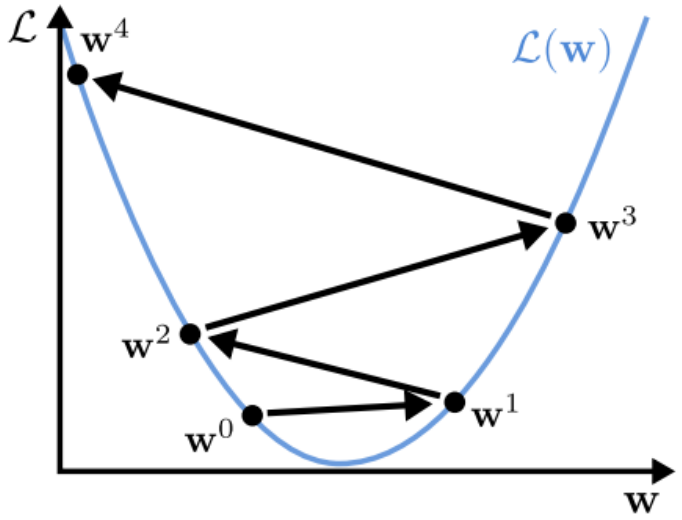
Scale of Learning rate

Large Learning rate

Gradient Descent:

$$\mathbf{w}^0 = \mathbf{w}^{\text{init}}$$

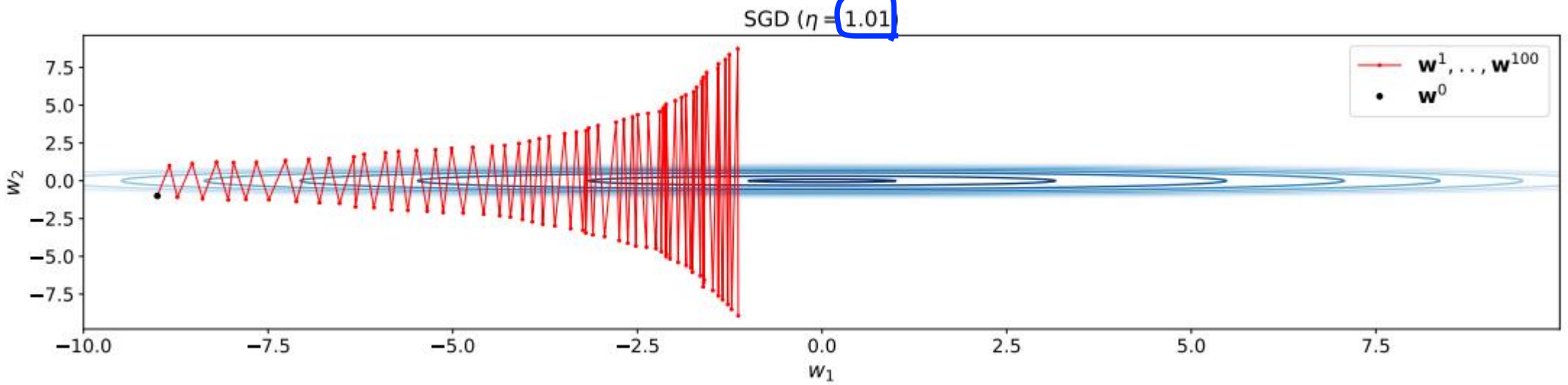
$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}^t)$$

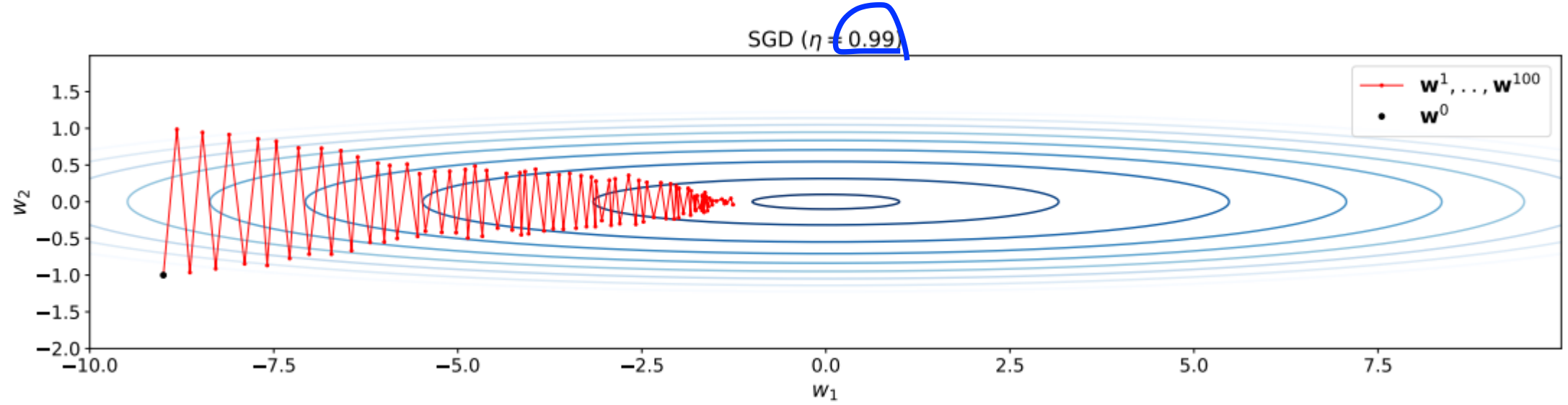


그렇다고 작게 설정한다면 내려오는데 한세월 걸릴 겁니다.

(산 내려오는데 5분에 한 계단씩 내려오는 거랑 같습니다)

Scale of Learning rate



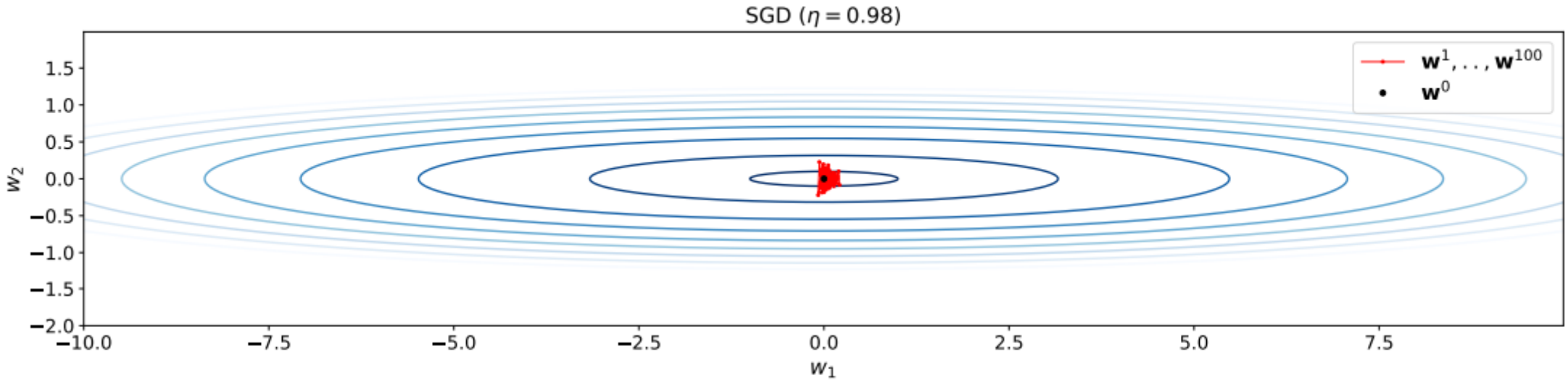


일반적으로 큰 학습률로 학습 시켜보고

불안정하다면 학습률을 낮추는 식으로 진행합니다.

하지만 이래도 여전히 최적점에 수렴하진 않습니다.

Fixed Learning rate



$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{1}{B} \sum_b \nabla_{\mathbf{w}} \mathcal{L}_b(\mathbf{w}^t)$$

어찌어찌 최적점 근처에 오더라도 그 주변에서 진동하고 수렴하지 않습니다.

위 그림은 아예 처음부터 w를 최적 값으로 설정했지만 진동합니다.

(batch를 랜덤하게 뽑기에 생기는 문제입니다)

Learning rate Scheduling

계속해서 우리가 가중치 W를 최적화 시킨다면 언젠가는 최적점에 도착해서 멈춰야 합니다. (수렴)

즉, 언젠가는 W의 변화가 없어야 합니다.

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)$$

$$\mathbf{w}^1 = \mathbf{w}^0 - \eta \nabla_{\mathbf{w}} \mathcal{L}_0$$

$$\mathbf{w}^2 = \mathbf{w}^1 - \eta \nabla_{\mathbf{w}} \mathcal{L}_1 = \mathbf{w}^0 - \eta \nabla_{\mathbf{w}} \mathcal{L}_0 - \eta \nabla_{\mathbf{w}} \mathcal{L}_1$$

$$\mathbf{w}^3 = \mathbf{w}^2 - \eta \nabla_{\mathbf{w}} \mathcal{L}_2 = \underbrace{\mathbf{w}^0 - \eta \nabla_{\mathbf{w}} \mathcal{L}_0}_{=\mathbf{a}_1} \underbrace{-\eta \nabla_{\mathbf{w}} \mathcal{L}_1}_{=\mathbf{a}_2} \underbrace{-\eta \nabla_{\mathbf{w}} \mathcal{L}_2}_{=\mathbf{a}_3}$$

Learning rate Scheduling

가중치 W를 하나의 series (급수) 로 생각해볼 때, 이 series가 수렴할 조건은 다음과 같습니다.

$$\sum_{t=1}^{\infty} \eta_t = \infty \qquad \text{and} \qquad \sum_{t=1}^{\infty} \eta_t^2 < \infty$$

이를 만족하는 가장 대표적인 예시는 다음과 같습니다.

$$\eta_t = \frac{\eta}{t}$$

즉, 시간이 지남에 따라 학습률이 감소하면 됩니다.

증명은 여기를 참고해주세요

Robbins and Monro. A Stochastic Approximation Method. The Annals of Mathematical Statistics, 1951.

Learning rate Scheduling

다양한 LR 스케줄러가 나와있기에 상황에 따라 맞는 것을 사용하시면 됩니다.

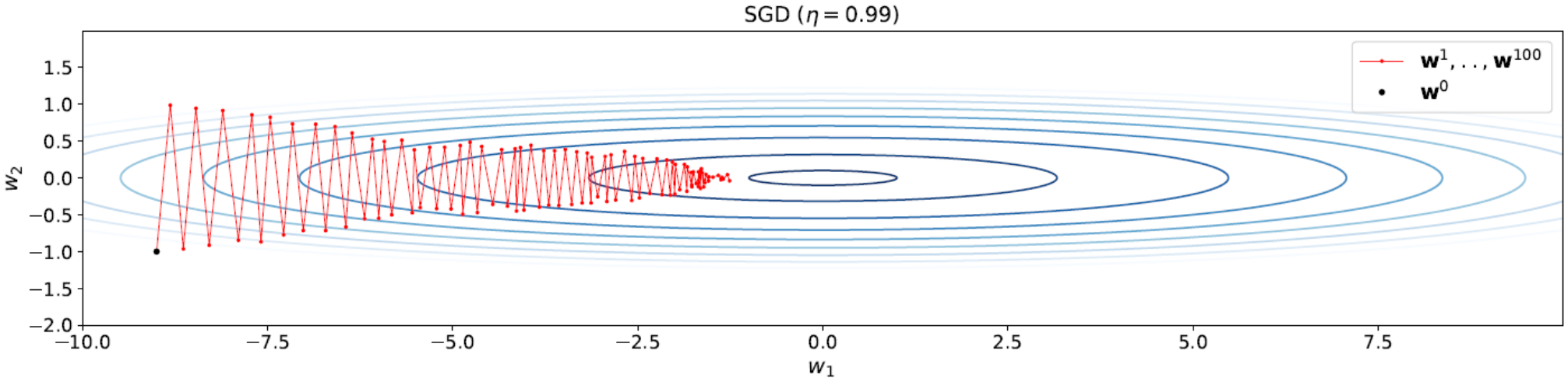
(COSINEANNEALINGLR 을 최근 대회에서는 많이 사용하는 추세입니다.)

`lr_scheduler.LambdaLR`	Sets the learning rate of each parameter group to the initial lr times a given function.
`lr_scheduler.MultiplicativeLR`	Multiply the learning rate of each parameter group by the factor given in the specified function.
`lr_scheduler.StepLR`	Decays the learning rate of each parameter group by gamma every step_size epochs.
`lr_scheduler.MultiStepLR`	Decays the learning rate of each parameter group by gamma once the number of epoch reaches one of the milestones.
`lr_scheduler.ConstantLR`	Decays the learning rate of each parameter group by a small constant factor until the number of epoch reaches a pre-defined milestone: total_iters.
`lr_scheduler.LinearLR`	Decays the learning rate of each parameter group by linearly changing small multiplicative factor until the number of epoch reaches a pre-defined milestone: total_iters.
`lr_scheduler.ExponentialLR`	Decays the learning rate of each parameter group by gamma every epoch.
`lr_scheduler.PolynomialLR`	Decays the learning rate of each parameter group using a polynomial function in the given total_iters.

Momentum

앞서 봤던 것처럼 학습률도 잘 정하고 스케줄링도 잘 해주면 끝일까요...?

이제는 학습이 잘 되긴 합니다만 여전히 느립니다.



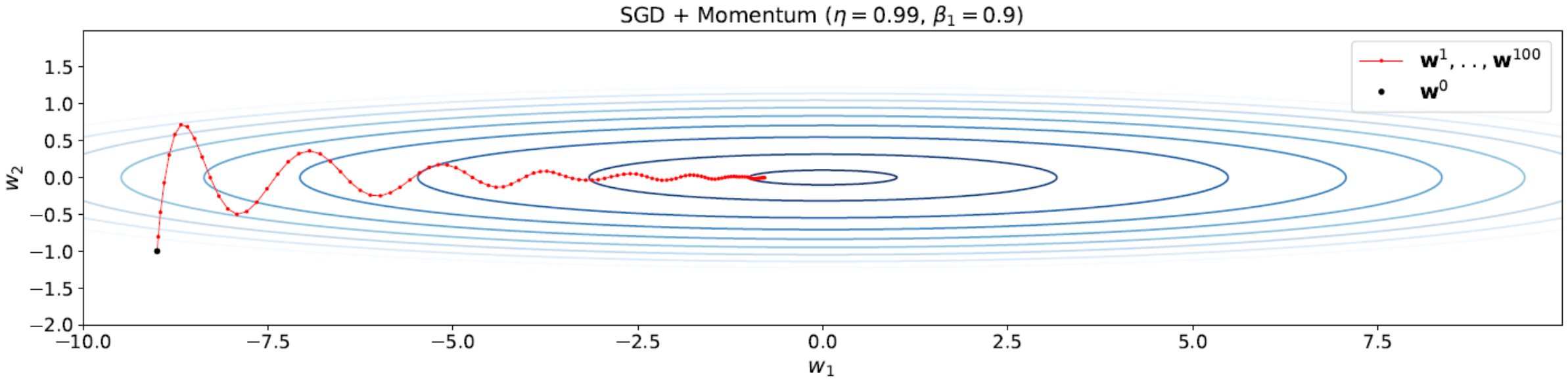
Momentum

만약 굴러가던 방향을 기억해서 다음번에 반영해준다면 어떻게 될까요?

평지를 만나더라도 기존 방향을 기억해서 굴러갈 수 있습니다.

현재 위치에서는 위 아래로 가라고 하더라도 오른쪽으로 이동하던 모멘텀이 있기에

최적점으로 빠르게 이동할 수 있습니다.



Momentum

수식으로 나타내면 다음과 같습니다.

$$\mathbf{m}^{t+1} = \beta_1 \mathbf{m}^t - \eta \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)$$
$$\mathbf{w}^{t+1} = \mathbf{w}^t + \mathbf{m}^{t+1}$$

β_1 는 이전의 값을 얼마나 기억할지 나타내는 하나의 파라미터입니다.

현재 반영할 모멘텀 m^{t+1} 에 과거의 모멘텀 m^t 와 현재 gradient 를 더합니다.

그리고 나서 현재 가중치 W를 업데이트 할 때 더해줍니다.

(이전에 이동하던 방향을 기억했다가 현재에 더해준다고 생각해주세요)

Momentum

그런데 생각해보면 계속해서 특정 t 시점의 m에 β_1 이 계속해서 곱해집니다.

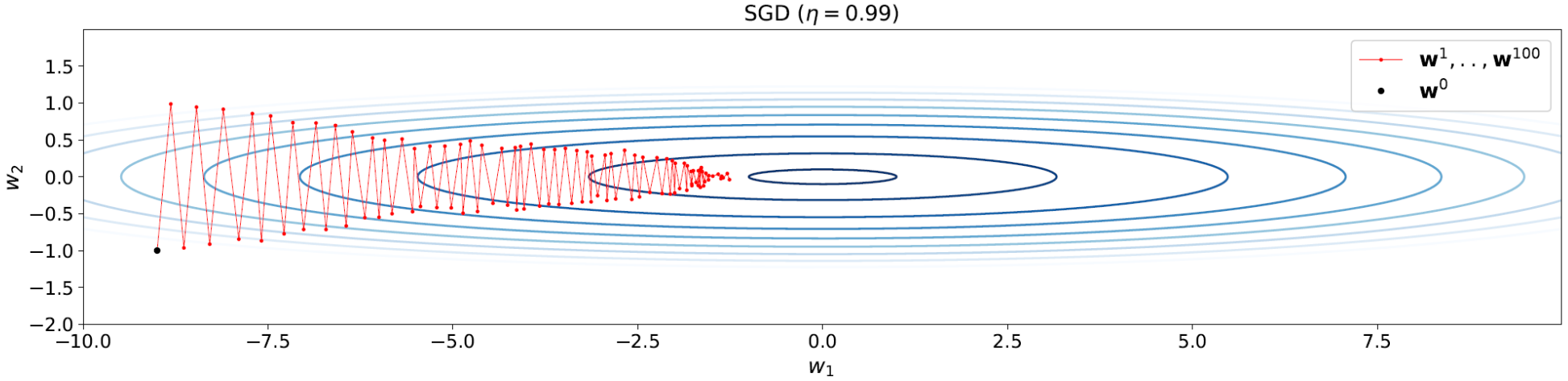
$$\mathbf{m}^{t+1} = \beta_1 \mathbf{m}^t - \eta \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)$$
$$\mathbf{w}^{t+1} = \mathbf{w}^t + \mathbf{m}^{t+1}$$

따라서 β_1 은 과거의 gradient를 얼마나 기억할지 나타내는 파라미터다 라고 생각할 수 있습니다.

(당연하게도 0보다 크고 1보다 작은 값으로 설정되는데 1에 가까울 수록 오래 기억됩니다.)

RMSprop

Gradient를 실제로 계산해보면 특정 방향으로는 되게 큰데 다른 쪽은 작은 경우가 많습니다.

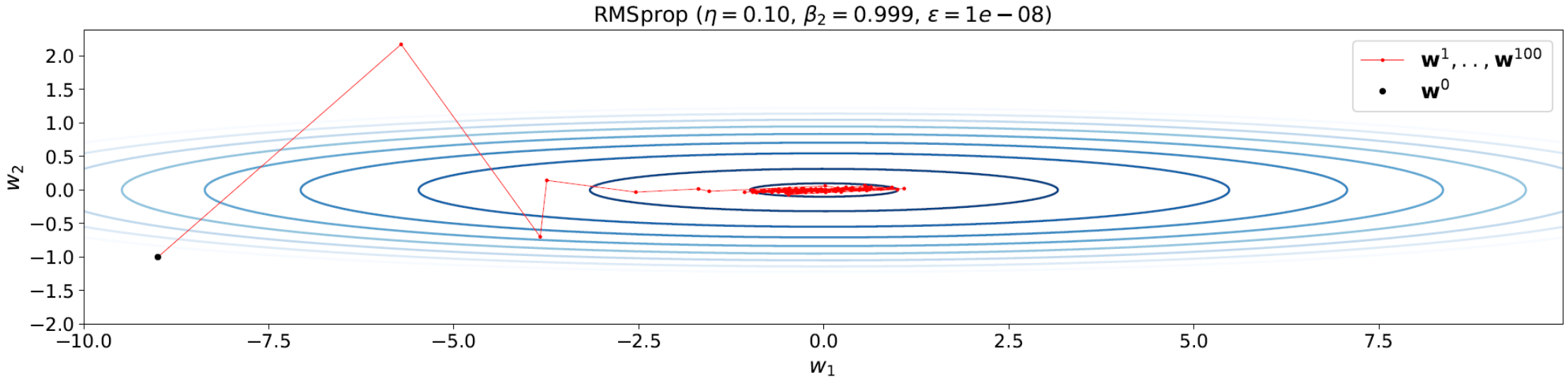


W_2 방향을 보면 크게 움직입니다 (Gradient가 크므로)

하지만 W_1 방향을 보면 되게 조금씩 움직이고 있습니다. (Gradient가 작으므로)

RMSprop

따라서 최근에 이동한 방향을 기억해줬다가 그 방향으로는 '덜' 움직이면 빠르게 수렴할 수 있습니다.



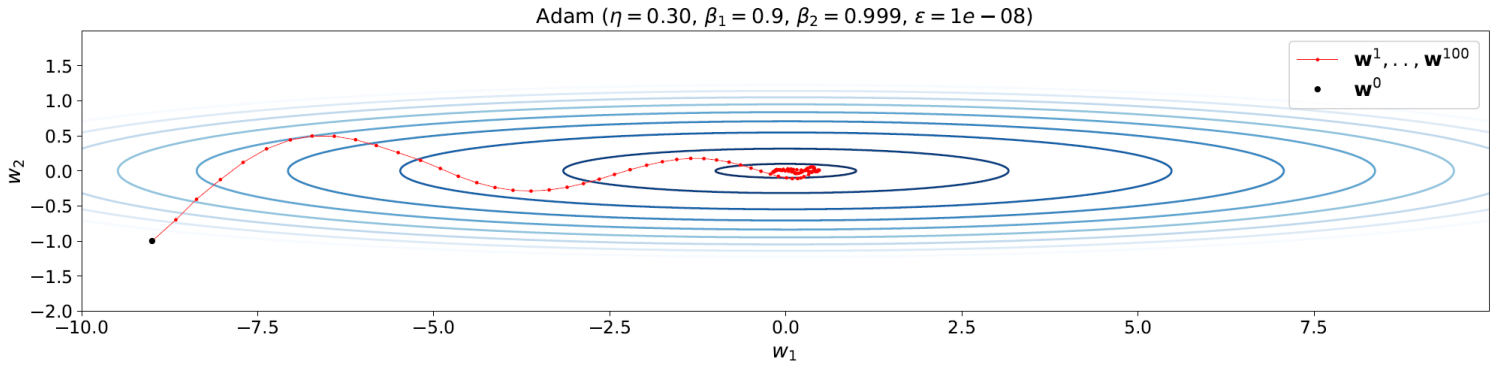
$$\mathbf{v}^{t+1} = \beta_2 \mathbf{v}^t + (1 - \beta_2) \left(\nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}^t) \odot \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}^t) \right)$$

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{\nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)}{\sqrt{\mathbf{v}^{t+1}} + \epsilon}$$

여기서 EMA 대신 그냥 더하면 AdaGrad라고 부릅니다.

Adam = RMSprop + Momentum

둘이 섞은 버전입니다. (일반적으로)[a] 가장 좋은 성능을 낸다고 알려져 있습니다.



$$\mathbf{m}^{t+1} = \beta_1 \mathbf{m}^t + (1-\beta_1)\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)$$

$$\mathbf{v}^{t+1} = \beta_2 \mathbf{v}^t + (1-\beta_2)\left(\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}}(\mathbf{w}^t) \odot \nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)\right)$$

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{\mathbf{m}^{t+1}}{\sqrt{\mathbf{v}^{t+1}} + \epsilon}$$

Momentum

$$\mathbf{m}^{t+1} = \beta_1 \mathbf{m}^t - \eta \nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)$$

$$\mathbf{w}^{t+1} = \mathbf{w}^t + \mathbf{m}^{t+1}$$

RMSprop

$$\mathbf{v}^{t+1} = \beta_2 \mathbf{v}^t + (1-\beta_2)\left(\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}}(\mathbf{w}^t) \odot \nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)\right)$$

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)}{\sqrt{\mathbf{v}^{t+1}} + \epsilon}$$

a) 물론 task 마다 다릅니다. SGD가 가장 좋은 성능을 내는 경우도 존재합니다.

Summary

1. 모델의 분포와 데이터의 분포를 비슷하게 만들자
2. 이 과정에서 두 분포의 차이를 loss 함수로 두고 optimize
3. Back Propagation으로 기울기, 즉 Gradient를 구하자
4. 기울기가 감소하는 방향, 즉 loss가 줄어드는 쪽으로 Update

참고 자료

1. Tubingen University, Deep Learning : lecture 06 Optimization

   강의자료 : https://drive.google.com/file/d/1QpJWFXLVibJJhYTz8i46j2k0HufnL74G/view

   강의영상 : https://www.youtube.com/playlist?list=PL05umP7R6ij3NTWIdtMbfvX7Z-4WEXRqD

2. Adam 논문

   https://arxiv.org/pdf/1412.6980.pdf

3. PyTorch LR scheduler Guide

   https://www.kaggle.com/code/isbhargav/guide-to-pytorch-learning-rate-scheduling

4. PyTorch optimizer docs

   https://pytorch.org/docs/stable/optim.html