Paper submitted to *JHM*
Vol. 00 No. 0 (Jan. 2014), 1–23

# Gradient Descent based Optimization Algorithms for Deep Learning Models Training

Nam EonWoo 남언우

본 논문에서는 심층 신경망 모델 학습을 위한 경사하강법 기반 최적화 알고리즘을 소개하는 것을 목표로 합니다. 여러 비선형 투영층을 포함하는 딥 러닝 모델은 훈련하기가 매우 어렵습니다. 오늘날 대부분의 딥러닝 모델 훈련은 실제로 여전히 역전파 알고리즘에 의존하고 있습니다. 역전파에서는 모델 변수가 경사하강법 기반 최적화 알고리즘과 수렴할 때까지 반복적으로 업데이트됩니다. 기존의 바닐라 경사하강법 알고리즘 외에도 학습 성능을 향상시키기 위해 최근에는 Momentum, Adagrad, Adam, Gadam 등 많은 경사하강의 변형이 제안되었으며, 이에 대해서는 모두 본 논문에서 소개합니다.

*Keywords:* 경사 하강법; 최적화 알고리즘; 딥러닝

*MSC:*

## 1 Introduction

딥러닝에 관한 실제 연구 및 응용 작업에서 딥러닝 모델을 효과적으로 훈련하는 것은 여전히 연구자와 실무자 모두에게 가장 어려운 작업 중 하나로 남아 있습니다. 이러한 맥락에서 지금까지 대부분의 심층 모델 훈련은 여전히 역전파 알고리즘을 기반으로 하며, 이는 출력 레이어의 오류를 뒤로 전파하고 경사 하강 기반 최적화 알고리즘을 사용하여 레이어별로 변수를 업데이트합니다. 경사하강법은 딥러닝 모델을 훈련하는 데 중요한 역할을 하며, 성능을 더욱 향상시키기 위해 최근 몇 년 동안 많은 새로운 변형 알고리즘이 제안되었습니다. 뉴턴 방법 등과 같은 고차 도함수 기반 알고리즘과 비교할 때 경사하강법은 1차 도함수를 기반으로 한 다양한 변형 알고리즘과 함께 심층 모델에 훨씬 더 효율적이고 실용적입니다. 본 논문에서는 딥러닝 모델 훈련 알고리즘에 대해 포괄적으로 소개합니다.

공식적으로, $n$쌍의 특징-레이블 벡터와 관련된 훈련 데이터 $\mathcal{T} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \cdots, (\mathbf{x}_n, \mathbf{y}_n)\}$이 주어졌다, 여기서 특징 벡터 $\mathbf{x}_i \in \mathbb{R}^{d_x}$ 및 레이블 벡터 $\mathbf{y}_i \in \mathbb{R}^{d_y}$이고 $d_x$와 $d_y$는 각각의 차원을 나타냅니다. 데이터를 매핑으로 분류하는 데 사용되는 딥러닝 모델을 $F(\cdot;\theta)\colon X \to Y$로 표현할 수 있습니다. 여기서 $X$와 $Y$는 각각 특징과 레이블 공간을 나

Nam EonWoo: Dept. of Math. HUFS. E-mail: new271@naver.com
*Submitted on May. 2, 2024.*

타내고 $\theta$는 함수에 포함된 변수 벡터를 나타냅니다. 벡터 $\mathbf{x}_i \in X$로 표현된 입력 데이터 인스턴스에 대해, 딥러닝 모델에 의해 출력된 결과를 $\hat{\mathbf{y}}_i = F(\mathbf{x}_i; \theta)$로 나타낼 수 있습니다. 입력 인스턴스의 실제 레이블 벡터 $\mathbf{y}_i$와 비교하여 손실 항 $\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i)$을 사용하여 모델에 의해 발생한 오류를 나타낼 수 있습니다. 이 문서의 나머지 부분에서는 차이 없이 오류와 손실을 같은 의미로 사용할 수 있습니다.

도입된 모델 오류를 측정하기 위해 다양한 손실 함수, 즉 $\ell(\cdot, \cdot)$가 제안되었습니다. 대표적인 예는 다음과 같습니다.

• Mean Square Error (MSE):

$$\ell_{MSE}(\hat{\mathbf{y}}_i, \mathbf{y}_i) = \frac{1}{d_y} \sum_{j=1}^{d_y} (\mathbf{y}_i - \hat{\mathbf{y}}_i(j))^2 \tag{1}$$

• Mean Absolute Error (MAE):

$$\ell_{MAE}(\hat{\mathbf{y}}_i, \mathbf{y}_i) = \frac{1}{d_y} \sum_{j=1}^{d_y} |\mathbf{y}_i - \hat{\mathbf{y}}_i(j)|^2 \tag{2}$$

• Hinge Loss:

$$\ell_{Hinge}(\hat{\mathbf{y}}_i, \mathbf{y}_i) = \sum_{j \neq l_{\mathbf{y}_i}} \max(0, \hat{\mathbf{y}}_i(j) - \hat{\mathbf{y}}_i(l_{\mathbf{y}_i}) + 1), \tag{3}$$

여기서 $\ell_{\mathbf{y}_i}$는 벡터 $\mathbf{y}_i$에 따른 인스턴스의 실제 클래스 레이블 인덱스를 나타냅니다.

• Cross Entropy Loss:

$$\ell_{CE}(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\sum_{j1}^{d_y} \mathbf{y}_i(j) \log \hat{\mathbf{y}}_i(j). \tag{4}$$

모델에 사용된 활성화 함수와 관련된 클래스의 수에 따라, 교차 엔트로피 함수는 시그모이드 교차 엔트로피와 소프트맥스 교차로 더욱 세분화되어 구분될 수 있습니다. 시그모이드 교차 엔트로피는 출력층의 활성화 함수로 시그모이드 함수를 사용하는 이진 분류 작업을 위한 함수이고, 소프트맥스 교차 엔트로피는 출력층의 활성화 함수로 소프트맥스 함수를 사용하는 다중 클래스 분류 작업을 위한 함수입니다.

평균 제곱 오차와 평균 절대 오차는 일반적으로 회귀 작업에 사용되며, 힌지 손실과 교차 엔트로피는 분류 작업에 주로 사용됩니다. 위에서 소개한 이러한 손실 함수 외에도, 휴버 손실, 코사인 거리 등 특정 학습 문제에 사용할 수 있는 많은 다른 손실 함수들이 있으며, 독자들은 자세한 정보를 얻기 위해 참고 문헌을 참조할 수 있습니다.

힌지 손실과 교차 엔트로피 손실 함수에 대한 추가적인 정보를 제공하고자, 실제 문제에서 어떻게 활용되는지 예시를 통해 설명할 것입니다.

**Example 1** 그림 1에서 보여주듯이, 세 개의 입력 이미지(즉, 데이터 인스턴스)를 기반으로, 아래 표에 표시된 대로 딥러닝 모델에 의해 그들의 예측 레이블을 얻을 수 있습니다. 개 이미지 입력(즉, 인스턴스 1)의 경우, 그 진짜 레이블은 $\mathbf{y}_1 = [1, 0, 0]^T$ 이어야 하며, 예측 레이블은 $\hat{\mathbf{y}}_1 = [0.49, 0.43, 0.08]^T$ 입니다. 이 벡터의 세 항목은 각각 개, 고양이, 새 클래스 레이블에 해당합니다. 유사하게, 고양이 이미지와 새 이미지(즉, 인스턴스 2와 인스턴스 3)

| Class | Instance 1 | Instance 2 | Instance 3 |
|---|---|---|---|
| **Dog** | 0.49 | 0.45 | 0.21 |
| **Cat** | 0.43 | 0.53 | 0.15 |
| **Bird** | 0.08 | 0.02 | 0.64 |

Figure 1. An Example to Illustrate the Hinge Loss and Cross Entropy Loss Functions.

의 정답 레이블과 예측 레이블을 각각 벡터 $\mathbf{y}_2 = [0, 1, 0]^T$, $\hat{\mathbf{y}}_2 = [0.45, 0.53, 0.02]^T$, 그리고 $\mathbf{y}_3 = [0, 0, 1]^T$, $\hat{\mathbf{y}}_3 = [0.21, 0.15, 0.64]^T$로 표현할 수 있습니다.

정답 레이블과 예측 레이블을 기반으로, 딥러닝 모델에 의해 도입된 손실 항은 다음과 같이 표현할 수 있습니다:

- 인스턴스 1: 인스턴스 1의 경우, 그 정답 레이블이 '개'임을 알고 있습니다 (즉, 레이블 벡터의 항목 1). 그리고 그 정답 레이블 인덱스는 $l_{\mathbf{y}_1} = 1$입니다. 따라서, 인스턴스 1에 대한 도입된 힌지 손실을 다음과 같이 표현할 수 있습니다.

$$\begin{aligned} \ell_{Hinge}(\hat{\mathbf{y}}_1, \mathbf{y}_1) &= \sum_{j \neq 1} \max(0, \hat{\mathbf{y}}_1(j) - \hat{\mathbf{y}}_1(1) + 1) \\ &= \max(0, \hat{\mathbf{y}}_1(2) - \hat{\mathbf{y}}_1(1) + 1) + \max(0, \hat{\mathbf{y}}_1(3) - \hat{\mathbf{y}}_1(1) + 1) \\ &= \max(0, 0.43 - 0.49 + 1) + \max(0, 0.08 - 0.49 + 1) \\ &= 1.53 \end{aligned} \tag{5}$$

- 인스턴스 2: 인스턴스 2의 경우, 정답 레이블이 고양이임을 알고 있습니다 (즉, 레이블 벡터의 항목 2), 그리고 그 정답 레이블 인덱스는 $l_{\mathbf{y}_2} = 2$입니다. 따라서 인스턴스 2에 대한 도입된 힌지 손실을 다음과 같이 표현할 수 있습니다.

$$\ell_{Hinge}(\hat{\mathbf{y}}_2, \mathbf{y}_2) = \sum_{j \neq 2} \max(0, \hat{\mathbf{y}}_2(j) - \hat{\mathbf{y}}_2(2) + 1)$$

$$
\begin{aligned}
&= \max(0, \hat{\mathbf{y}}_2(1) - \hat{\mathbf{y}}_2(2) + 1) + \max(0, \hat{\mathbf{y}}_2(3) - \hat{\mathbf{y}}_2(2) + 1) \\
&= \max(0, 0.45 - 0.53 + 1) + \max(0, 0.02 - 0.53 + 1) \\
&= 1.41
\end{aligned} \tag{6}
$$

- 인스턴스 3: 마찬가지로, 입력 인스턴스 3의 경우, 그 정답 레이블 인덱스는 $l_{\mathbf{y}_3} = 3$ 이며, 인스턴스에 대한 도입된 힌지 손실을 표시할 수 있습니다.

$$
\begin{aligned}
\ell_{Hinge}(\hat{\mathbf{y}}_3, \mathbf{y}_3) &= \sum_{j \neq 3} \max(0, \hat{\mathbf{y}}_3(j) - \hat{\mathbf{y}}_3(3) + 1) \\
&= \max(0, \hat{\mathbf{y}}_3(1) - \hat{\mathbf{y}}_3(3) + 1) + \max(0, \hat{\mathbf{y}}_3(3) - \hat{\mathbf{y}}_3(3) + 1) \\
&= \max(0, 0.21 - 0.64 + 1) + \max(0, 0.15 - 0.64 + 1) \\
&= 1.08
\end{aligned} \tag{7}
$$

한편, 이 세 입력 데이터 인스턴스의 정답 레이블과 예측 레이블 벡터를 기반으로, 그들의 도입된 교차 엔트로피 손실을 다음과 같이 표현할 수 있습니다:

- 인스턴스 1:

$$
\begin{aligned}
\ell_{CE}(\hat{\mathbf{y}}_1, \mathbf{y}_1) &= -\sum_{j=1}^{d_y} \mathbf{y}_1(j) \log \hat{\mathbf{y}}_1(j) \\
&= -1 \times \log 0.49 - 0 \times \log 0.43 - 0 \times \log 0.08 \\
&= 0.713
\end{aligned} \tag{8}
$$

- 인스턴스 2:

$$
\begin{aligned}
\ell_{CE}(\hat{\mathbf{y}}_2, \mathbf{y}_2) &= -\sum_{j=1}^{d_y} \mathbf{y}_2(j) \log \hat{\mathbf{y}}_2(j) \\
&= -0 \times \log 0.45 - 1 \times \log 0.53 - 0 \times \log 0.02 \\
&= 0.635
\end{aligned} \tag{9}
$$

- 인스턴스 3:

$$
\begin{aligned}
\ell_{CE}(\hat{\mathbf{y}}_3, \mathbf{y}_3) &= -\sum_{j=1}^{d_y} \mathbf{y}_3(j) \log \hat{\mathbf{y}}_3(j) \\
&= -0 \times \log 0.21 - 0 \times \log 0.15 - 1 \times \log 0.64 \\
&= 0.446
\end{aligned} \tag{10}
$$

또한, 훈련 세트의 모든 데이터 인스턴스에 대해 도입된 손실을 기반으로, 딥러닝 모델에 의해 도입된 총손실 항을 다음과 같이 나타낼 수 있습니다.

$$
\mathcal{L}(\theta : \mathcal{T}) = \sum_{(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{T}} \ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = \sum_{(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{T}} \ell(F(\mathbf{x}_i; \theta), \mathbf{y}_i) \tag{11}
$$

훈련 데이터에 적합함으로써, 딥러닝 모델 훈련은 훈련 세트에 도입된 손실을 최소화하여

최적의 변수, 즉 최적의 θ를 달성하는 것을 목표로 합니다. 공식으로, 딥러닝 모델 훈련의 목적 함수는 다음과 같이 표현될 수 있습니다:

$$\min_{\theta \in \Theta} \mathcal{L}(\theta; \mathcal{T}) \tag{12}$$

여기서 Θ는 변수 도메인을 나타내며, 학습할 변수에 다른 제약이 없는 경우 $\Theta = \mathbb{R}^{d_\theta}$ ($d_\theta$는 변수 벡터 $\theta$의 차원을 나타냅니다)를 가질 수 있습니다. 목적 함수를 다루기 위해, 다음 섹션에서는 손실 함수가 볼록(convex)인 경우 또는 비볼록(non-convex)인 경우에 따라 딥러닝 모델의 전역적으로(globally) 최적화되거나 지역적으로(locally) 최적화된 변수를 학습할 수 있는 일련의 경사 하강법 기반 최적화 알고리즘을 소개할 것입니다. 이 알고리즘들의 성능에 대한 보다 포괄적인 실험 분석은 이 논문의 마지막 부분에서 몇 가지 실세계의 데이터셋에 대한 실습 실험과 함께 제공될 것입니다.

## 2 전통적인 경사 하강법 기반 학습 알고리즘

이번 절에서는 기존의 바닐라 경사하강법, 확률적 경사하강법, 미니배치 경사하강법 알고리즘을 소개합니다. 이들 간의 주요 차이점은 각 반복에서 모델 변수를 업데이트하는 데 사용되는 훈련 데이터의 양에 있습니다. 이는 또한 다음 섹션에서 소개될 변형 알고리즘의 기본 알고리즘 역할을 합니다.

### 2.1 바닐라 경사하강법 (Vanilla Gradient Descent)

바닐라 경사하강법은 배치 경사하강법으로도 잘 알려져 있습니다. (논문 번역에서는 그냥 바닐라 경사하강법이라고 번역하였습니다) 이전 절에서 소개된 훈련 세트 $\mathcal{T}$가 주어진 상태에서, 바닐라 경사하강 최적화 알고리즘은 다음 방정식을 사용하여 모델 변수를 반복적으로 최적화합니다:

$$\theta^{(\tau)} = \theta^{(\tau-1)} - \eta \cdot \nabla_\theta \mathcal{L}(\theta^{(\tau-1)}; \mathcal{T}) \tag{13}$$

$\tau \geq 1$은 업데이트 반복을 나타냅니다.

위의 업데이트 방정식에서, $\theta(\tau)$, $\nabla_\theta \mathcal{L}(\theta^{(\tau-1)}; \mathcal{T})$, $\eta$의 물리적 의미는 다음과 같습니다:

1. 항 $\theta(\tau)$은 반복 $\tau$을 통해 업데이트된 모델 변수 벡터를 나타냅니다. $\tau = 0$에서, 벡터 $\theta(0)$은 보통 일정한 분포(예: 균등 분포 $\mathcal{U}(a, b)$ 또는 정규 분포 $N(\mu, \sigma^2)$)를 따라 무작위로 초기화된 초기 모델 변수 벡터를 나타냅니다. 더 나은 성능을 달성하기 위해, 균등 분포의 하이퍼파라미터 $a, b$ 또는 정규 분포의 $\mu, \sigma$를 파인 튜닝할 수 있습니다.

2. 항 $\nabla_\theta \mathcal{L}(\theta^{(\tau-1)}; \mathcal{T})$는 전체 훈련 세트 $\mathcal{T}$와 $\tau - 1$번 반복하여 얻은 변수 벡터 값 $\theta^{(\tau-1)}$를 기반으로 손실 함수 $\mathcal{L}(\theta; \mathcal{T})$의 변수 $\theta$에 대한 도함수를 나타냅니다.

3. 항 $\eta$는 경사 하강 알고리즘에서 학습률(learning rate)을 나타내며, 보통 작은 값(예: $10^{-3}$)의 하이퍼파라미터입니다. 실제 바닐라 경사 하강 알고리즘의 응용에서 빠른 수렴을 달성하기 위해 파라미터의 파인 튜닝이 필요합니다.

이러한 반복적인 업데이트 과정은 수렴할 때까지 계속되며, 수렴에서 달성된 변수 벡터 θ는 딥러닝 모델에 대한 (전역적 또는 지역적으로) 최적의 변수로 출력됩니다. 바닐라 경사 하강 알고리즘의 의사코드는 알고리즘 1에 제공되어 있습니다. 알고리즘 1에서 모델 변수는 정규 분포로 초기화됩니다. 정규 분포의 표준 편차 $\sigma$는 입력 파라미터이며, 평균 $\mu$는 0으로 설정됩니다. 수렴 조건은 (1) 연속적인 두 반복에서의 손실 항의 변화가 사전에 지정된 임계값보다 작을 때, (2) 모델 변수 θ의 변화가 사전에 지정된 범위 내일 때, 또는 (3) 사전에 지정된 반복 횟수에 도달했을 때가 될 수 있습니다.

**Algorithm 1** Vanilla Gradient Descent

**Require**: Training Set: $\mathcal{T}$; Learning Rate $\eta$; Normal Distribution Std: $\sigma$.
**Ensure**: Model Parameter $\theta$
1: Initialize parameter with Normal distribution $\theta \sim N(0, \sigma^2)$
2: Initialize convergence *tag* = *False*
3: **while** *tag* == *False* **do**
4: Compute gradient $\nabla_\theta \mathcal{L}(\theta; \mathcal{T})$ on the training set $\mathcal{T}$
5: Update variable $\theta = \theta - \eta \cdot \nabla_\theta \mathcal{L}(\theta; \mathcal{T})$
6: **if** convergence condition holds **then**
7: *tag* = *True*
8: **end if**
9: **end while**
10: **Return** model variable $\theta$

설명에 따르면, 바닐라 경사 하강법 알고리즘에서는 모델 변수를 업데이트하기 위해서 각 반복에서 변수에 대한 손실 함수의 기울기, 즉 $\nabla_\theta \mathcal{L}(\theta^{(\tau-1)}; \mathcal{T})$를 계산해야 하는데, 이는 대규모 훈련 세트에서 일반적으로 매우 시간이 많이 소요됩니다. 학습 효율을 개선하기 위해, 다음 두 하위 섹션에서 확률적 경사 하강법과 미니 배치 경사 하강법 학습 알고리즘을 소개할 것입니다.

## 2.2 확률적 경사 하강법 (Stochastic Gradient Descent)

바닐라 경사 하강법은 전체 훈련 세트를 사용하여 손실 함수의 기울기를 계산하는데, 훈련 세트에 많은 데이터 인스턴스가 포함되어 있는 경우 매우 비효율적일 수 있습니다. 전체 훈련 세트를 사용하는 대신, 확률적 경사 하강법(SGD) 알고리즘은 인스턴스별로 손실 함수의 기울기를 계산하여 모델 변수를 업데이트합니다. 따라서 확률적 경사 하강법은 바닐라 경사 하강법보다 훨씬 빠를 수 있지만, 업데이트 과정에서 손실 함수의 변동이 심할 수도 있습니다.

공식으로, 훈련 세트 $\mathcal{T}$와 초기화된 모델 변수 벡터 $\theta^{(0)}$가 주어진 경우, $\mathcal{T}$에 속하는 각 인스턴스 $(\mathbf{x}_i, \mathbf{y}_i)$에 대해 확률적 경사 하강 알고리즘은 다음 방정식을 사용하여 모델 변수를 반복적으로 업데이트할 것입니다:

$$\theta^{(\tau)} = \theta^{(\tau-1)} - \eta \cdot \nabla_\theta \mathcal{L}(\theta^{(\tau-1)}; (\mathbf{x}_i, \mathbf{y}_i)) \tag{14}$$

여기서 손실 항 $\mathcal{L}(\theta; (\mathbf{x}_i, \mathbf{y}_i) = \ell(F(\mathbf{x}_i; \theta), \mathbf{y}_i)$는 데이터 인스턴스 $(\mathbf{x}_i, \mathbf{y}_i)$에서 모델에 의해 도입된 손실을 나타냅니다. 확률적 경사 하강 학습 알고리즘의 의사 코드는 알고리즘 2에 제공되어 있습니다.

알고리즘 2에 따르면, 업데이트 과정 전에 전체 훈련 세트는 섞일 것입니다. 비록 확률적

**Algorithm 2** Stochastic Gradient Descent

**Require**: Training Set: $\mathcal{T}$; Learning Rate $\eta$; Normal Distribution Std: $\sigma$.
**Ensure**: Model Parameter $\theta$
1: Initialize parameter with Normal distribution $\theta \sim N(0, \sigma^2)$
2: Initialize convergence *tag* = *False*
3: **while** *tag* == *False* **do**
4: Shuffle the training set $\mathcal{T}$
5: **for** each data instance $(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{T}$ **do**
6: Compute gradient $\nabla_\theta \mathcal{L}(\theta; (\mathbf{x}_i, \mathbf{y}_i))$ on the training set $(\mathbf{x}_i, \mathbf{y}_i)$
7: Update variable $\theta = \theta - \eta \cdot \nabla_\theta \mathcal{L}(\theta; (\mathbf{x}_i, \mathbf{y}_i))$
8: **end for** 9: **if** convergence condition holds **then**
10: *tag* = *True*
11: **end if**
12: **end while**
13: **Return** model variable $\theta$

경사 하강의 학습 과정은 많이 변동될 수 있지만, 이는 실제로 확률적 경사 하강에게 지역 최적점(local optimum)에서 벗어날 수 있는 능력을 제공합니다. 한편, 작은 학습률 $\eta$을 선택하고 학습 과정에서 그것을 감소시킴으로써, 확률적 경사 하강은 거의 확실하게 지역적 최적점 또는 전역적 최적점(global optimum)에 수렴할 수 있습니다.

## 2.3 미니 배치 경사 하강법 (Mini-batch Gradient Descent)

바닐라 경사 하강법과 확률적 경사 하강법 사이의 균형을 맞추기 위해, 미니 배치 경사 하강법은 훈련 인스턴스의 미니 배치로 모델 변수를 대신하여 업데이트하도록 제안합니다. $\mathcal{B} \subset \mathcal{T}$가 훈련 세트 $\mathcal{T}$에서 샘플링된 훈련 인스턴스의 미니 배치를 나타낸다고 했을 때 미니 배치를 사용하여 딥러닝 모델의 변수 업데이트 방정식을 다음과 같이 공식으로 나타낼 수 있습니다:

$$\theta^{(\tau)} = \theta^{(\tau-1)} - \eta \cdot \nabla_\theta \mathcal{L}(\theta^{(\tau-1)}; \mathcal{B}) \tag{15}$$

여기서 $\mathcal{L}(\theta; \mathcal{B})$는 미니 배치 $\mathcal{B}$에 대한 모델에 의해 도입된 손실 항을 나타냅니다. 미니 배치 경사 하강 알고리즘의 의사 코드는 알고리즘 3에서 나타납니다.

**Algorithm 3** Mini-batch Gradient Descent

**Require**: Training Set: $\mathcal{T}$; Learning Rate $\eta$; Normal Distribution Std: $\sigma$; Mini-batch Size $b$.
**Ensure**: Model Parameter $\theta$
1: Initialize parameter with Normal distribution $\theta \sim N(0, \sigma^2)$
2: Initialize convergence *tag* = *False*
3: **while** *tag* == *False* **do**
4: Shuffle the training set $\mathcal{T}$
5: **for** each mini-batch $\mathcal{B} \subset \mathcal{T}$ **do**
6: Compute gradient $\nabla_\theta \mathcal{L}(\theta; \mathcal{B})$ on the mini-batch $\mathcal{B}$
7: Update variable $\theta = \theta - \eta \cdot \nabla_\theta \mathcal{L}(\theta; \mathcal{B})$
8: **end for** 9: **if** convergence condition holds **then**
10: *tag* = *True*
11: **end if**
12: **end while**
13: **Return** model variable $\theta$

미니 배치 경사 하강 알고리즘에서 배치 크기 $b$는 파라미터로 제공되며, 64, 128, 256과 같은 값이 될 수 있습니다. 대부분의 경우, $b$는 너무 크지 않아야 하며 그 구체적인 값은 훈련 세트의 크기에 많이 의존합니다. 보통 미니 배치는 훈련 세트 $\mathcal{T}$에서 순차적으로 샘플링됩니다. 즉, 훈련 세트 $\mathcal{T}$는 크기 $b$의 여러 배치로 나눌 수 있으며, 이 배치들은 차례대로 선택되어 모델 변수를 업데이트하는데 순차적으로 사용될 수 있습니다. 일부의 미니 배치 경사 하강법 버전에서는 순차적 배치 선택 대신에, 훈련 세트에서 미니 배치를 무작위로 샘플링하 것을 제안하는데 이는 랜덤 미니 배치 생성 과정으로도 알려져 있습니다.

바닐라 경사 하강법과 비교할 때, 미니 배치 경사 하강 알고리즘은 특히 매우 큰 크기의 훈련 세트에 대해 훨씬 더 효율적입니다. 한편, 확률적 경사 하강법과 비교하여, 미니 배치 경사 하강 알고리즘은 모델 변수 업데이트 과정에서의 변동을 크게 줄이고 훨씬 더 안정적인 수렴을 달성할 수 있습니다.

## 2.4 바닐라 GD, SGD, 미니 배치 GD의 성능 분석

여기서 소개된 세 가지 경사 하강법 알고리즘은 많은 최적화 문제에 잘 작동하며, 모두 가능성이 있는 (지역적 또는 전역적) 최적점으로 수렴할 수 있습니다. 그러나 이러한 알고리즘들은 다음과 같은 몇 가지 문제점을 겪고 있습니다:

- 학습률 선택(Learning Rate Selection): 학습률 $\eta$는 경사하강법 알고리즘의 수렴에 큰 영향을 미칠 수 있습니다. 큰 학습률은 학습 과정을 발산시킬 수 있으며, 작은 학습률은 수렴을 너무 느리게 합니다. 따라서, 좋은 학습률을 선택하는 것은 경사하강법 알고리즘이 매우 중요합니다.

- 학습률 조정(Learning Rate Adjustment): 대부분의 경우, 전체 업데이트 과정에서 고정된 학습률은 경사 하강법 알고리즘에 잘 작동하지 않습니다. 초기 단계에서는 알고리즘이 빠르게 좋은 (지역적 또는 전역적) 최적점에 도달하기 위해 더 큰 학습률을 필요로 할 수 있습니다. 그러나 후반 단계에서는 성능을 파인 튜닝하기 위해서, 알고리즘이 학습률을 더 작은 값으로 조정할 필요가 있을 수 있습니다.

- 변수별 개별 학습률(Variable Individual Learning Rate): 서로 다른 변수들에 대해 업데이트 과정에서 실제로는 다른 학습률을 필요로 할 수 있습니다. 따라서 서로 다른 변수에 대해 개별 학습률을 사용하는 방법이 필요합니다.

- 안장점 피하기(Saddle Point Avoid): 공식에서 안장점은 모든 차원에서 기울기가 0인 점을 나타냅니다. 그러나 일부 차원에서는 안장점이 지역 최소점(local minimum)이고, 다른 일부 차원에서는 그 점이 지역 최대점(local maximum)입니다. 이러한 안장점에서 벗어나는 방법은 매우 어려운 문제입니다.

이 논문의 다음 부분에서는 위의 문제들 중 하나 또는 여러 개를 해결하기 위해 주로 제안된 경사 하강법 기반의 학습 알고리즘의 여러 변형을 소개할 것입니다.

## 3 Momentum 기반 학습 알고리즘 (Momentum based Learning Algorithms)

이 섹션에서는 현재 경사와 과거 경사를 동시에 사용하여 모델 변수를 업데이트하는 것을 제안하는 경사 하강 변형 알고리즘 그룹을 소개합니다. 이를 운동량 기반 경사하강 알고리즘이라고 합니다. 여기서는 미니 배치 GD를 기본 학습 알고리즘으로 사용하여 이러한 알고리즘에 대해 설명합니다.

### 3.1 Momentum

경사하강법 알고리즘(예: 미니 배치 경사하강법)의 학습 과정에서 발생하는 변동을 완화하기 위해, Momentum [6]은 변수의 업데이트 수렴을 가속화하기 위해 제안되었습니다. 공식에서, Momentum은 다음 방정식을 사용하여 변수를 업데이트합니다:

$$\theta^{(\tau)} = \theta^{(\tau-1)} - \eta \cdot \Delta\mathbf{v}^{(\tau)} \text{ where. } \Delta\mathbf{v}^{(\tau)} = \rho \cdot \Delta\mathbf{v}^{(\tau-1)} + (1-\rho) \cdot \nabla_\theta \mathcal{L}(\theta^{(\tau-1)}), \quad (16)$$

위의 방정식에서, $\mathbf{v}^{(\tau)}$는 $\tau$번째 반복까지의 이전의 기울기를 기록하기 위해 도입된 Momentum 항을 나타냅니다. 함수 $L(\theta) = L(\theta; \mathcal{B})$는 미니 배치 $\mathcal{B}$에 대한 손실 함수를 나타냅니다. 매개변수 $\rho \in [0, 1]$는 Momentum 항의 가중치를 나타냅니다. Momentum 기반 경사하강 학습 알고리즘의 의사 코드는 알고리즘 4에서 제공됩니다.

**Algorithm 4** Momentum based Mini-batch Gradient Descent

**Require**: Training Set: $\mathcal{T}$; Learning Rate $\eta$; Normal Distribution Std: $\sigma$; Mini-batch Size $b$; Momentum Term Weight: $\rho$.
**Ensure**: Model Parameter $\theta$
1: Initialize parameter with Normal distribution $\theta \sim N(0, \sigma^2)$
2: Initialize Momentum term $\Delta\mathbf{v} = 0$
3: Initialize convergence *tag* = *False*
4: **while** *tag* == *False* **do**
5: Shuffle the training set $\mathcal{T}$
6: **for** each mini-batch $\mathcal{B} \subset \mathcal{T}$ **do**
7: Compute gradient $\nabla_\theta \mathcal{L}(\theta; \mathcal{B})$ on the mini-batch $\mathcal{B}$
8: Update term $\Delta\mathbf{v} = \rho \cdot \Delta\mathbf{v} + (1-\rho) \cdot \nabla_\theta \mathcal{L}(\theta; \mathcal{B})$
9: Update variable $\theta = \theta - \eta \cdot \Delta\mathbf{v}$
10: **end for**
11: **if** convergence condition holds **then**
12: *tag* = *True*
13: **end if**
14: **end while**
15: **Return** model variable $\theta$

이러한 $\theta^{(\tau)}$ 벡터 방정식에 대한 재귀적인 업데이트를 기반으로, 다음 보조정리(Lemma)에 따르면 손실 함수의 기울기 항만을 사용하여 $\theta^{(\tau)}$를 실제로 동등하게 표현할 수 있습니다.

**Lemma 3.1:** 벡터 $\Delta\mathbf{v}^{(\tau)}$는 다음 방정식으로 표현될 수 있습니다:

$$\Delta\mathbf{v}^{(\tau)} = \sum_{t=0}^{\tau-1} \rho^t \cdot (1-\rho) \cdot \nabla_\theta \mathcal{L}(\theta^{(\tau-1-t)}) \quad (17)$$

*Proof.* Lemma는 $\tau$에 대한 귀납법으로 증명될 수 있습니다.

1. 기본 경우: $\tau = 1$인 경우 $\theta$의 재귀적 표현에 따르면 다음과 같습니다.

$$\begin{aligned}\Delta\mathbf{v}^{(1)} &= \rho \cdot \Delta\mathbf{v}^{(0)} + (1-\rho) \cdot \nabla_\theta \mathcal{L}(\theta^{(0)}) \\ &= (1-\rho) \cdot \nabla_\theta \mathcal{L}(\theta^{(0)})\end{aligned} \tag{18}$$

   여기서 $\Delta\mathbf{v}(0)$은 0 벡터로 초기화됩니다

2. 가정: 방정식이 $\tau = k$에 대해 성립한다고 가정합니다.

$$\Delta\mathbf{v}^{(k)} = \sum_{t=0}^{k-1} \rho^t \cdot (1-\rho) \cdot \nabla_\theta \mathcal{L}(\theta^{(k-1-t)}). \tag{19}$$

3. 귀납법: $\tau = k+1$에 대해 방정식 (16)을 기반으로 벡터 $\Delta\mathbf{v}^{(k+1)}$을 다음과 같이 나타낼 수 있습니다.

$$\begin{aligned}\Delta\mathbf{v}^{(k+1)} &= \rho \cdot \Delta\mathbf{v}^{(k)} + (1-\rho) \cdot \nabla_\theta \mathcal{L}(\theta^{(k)}) \\ &= \rho \cdot \sum_{t=0}^{k-1} \rho^t \cdot (1-\rho) \cdot \nabla_\theta \mathcal{L}(\theta^{(k-1-t)}) + (1-\rho) \cdot \nabla_\theta \mathcal{L}(\theta^{(k)}) \\ &= \sum_{t=1}^{k} \rho^t \cdot (1-\rho) \cdot \nabla_\theta \mathcal{L}(\theta^{(k-t)}) + (1-\rho) \cdot \nabla_\theta \mathcal{L}(\theta^{(k)}) \\ &= \sum_{t=0}^{k} \rho^t \cdot (1-\rho) \cdot \nabla_\theta \mathcal{L}(\theta^{(k-t)})\end{aligned} \tag{20}$$

이것으로 이 증명을 결론지을 수 있습니다. □

매개변수 $\rho \in [0, 1]$을 고려하면 현재 반복에서 멀리 떨어진 반복의 기울기는 기하급수적으로 감소합니다. 기존 관습적인 경사하강법에 비해 Momentum은 더 빠른 수렴 속도를 얻을 수 있으며 이는 위 방정식에서 업데이트된 경사를 기반으로 설명할 수 있습니다. 경사하강법 알고리즘의 경우 반복 $\tau$에서 업데이트된 경사는 $\nabla_\theta \mathcal{L}(\theta^{(\tau-1)})$로 표시할 수 있는 반면, 모멘텀의 업데이트된 경사는 $\Delta\mathbf{v}(\tau) = (1-\rho) \cdot \nabla_\theta \mathcal{L}(\theta^{(\tau-1)}) + \rho \cdot \Delta\mathbf{v}^{(\tau-1)}$. 항 $\Delta\mathbf{v}^{(\tau-1)}$은 과거 경사를 기록합니다. 경사하강법 알고리즘이 가파른 구역에서 상대적으로 평평한 구역으로 변수를 업데이트함에 따라 $\Delta\mathbf{v}^{(\tau-1)}$에 저장된 큰 과거 경사 항은 실제로 알고리즘을 가속화하여 (로컬 또는 전역) 최적에 도달하게 합니다.

### 3.2 Nesterov Accelerated Gradient

지금까지 소개된 경사하강법 알고리즘의 경우 모두 학습 과정에서 도달하게 될 미래 지점에 대한 지식 없이 과거 또는 현재 지점의 경사를 기반으로 변수를 업데이트합니다. 이는 학습 과정을 맹목적으로 만들고 학습 성과를 예측할 수 없게 만듭니다.

Nesterov Accelerated Gradient (즉, NAG) [5] 방법은 대신에 근사된 미래 지점에서 경사로 변수를 업데이트하여 이 문제를 해결하도록 제안합니다. 단계 $\tau$에서 달성할 변수 $\theta$는 $\theta^{(\tau)}$로 표시할 수 있으며, $\hat{\theta}^{(\tau)} = \theta^{(\tau-1)} - \eta \cdot \rho \cdot \Delta\mathbf{v}^{(\tau-1)}$로 추정할 수 있습니다. 여기서 $\Delta\mathbf{v}^{(\tau-1)}$은 반복 $\tau - 1$에서의 운동량 항을 나타냅니다. 공식적으로 예측 경사 및 모멘텀의

도움을 받아 변수가 업데이트됩니다. NAG의 방정식은 공식적으로 다음과 같이 표현될 수 있습니다.

$$\theta^{(\tau)} = \theta^{(\tau-1)} - \eta \cdot \Delta \mathbf{v}^{(\tau)}, \quad \begin{cases} \hat{\theta}^{(\tau)} = \theta^{(\tau-1)} - \eta \cdot \rho \cdot \Delta \mathbf{v}^{(\tau-1)} \\ \Delta \mathbf{v}^{(\tau)} = \rho \cdot \Delta \mathbf{v}^{(\tau-1)} + (1-\rho) \cdot \nabla_\theta \mathcal{L}(\hat{\theta}^{(\tau)}) \end{cases} \tag{21}$$

NAG 알고리즘은 한 단계 앞으로 경사를 계산함으로써 모델 변수를 훨씬 더 효과적으로 업데이트할 수 있습니다. NAG 학습 알고리즘의 의사 코드는 알고리즘 5에서 사용할 수 있습니다.

**Algorithm 5** NAG based Mini-batch Gradient Descent

**Require**: Training Set: $\mathcal{T}$; Learning Rate $\eta$; Normal Distribution Std: $\sigma$; Mini-batch Size $b$; Momentum Term Weight: $\rho$.
**Ensure**: Model Parameter $\theta$
1: Initialize parameter with Normal distribution $\theta \sim N(0, \sigma^2)$
2: Initialize Momentum term $\Delta \mathbf{v} = 0$
3: Initialize convergence *tag* = *False*
4: **while** *tag* == *False* **do**
5: Shuffle the training set $\mathcal{T}$
6: **for** each mini-batch $\mathcal{B} \subset \mathcal{T}$ **do**
7: Compute $\hat{\theta} = \theta - \eta \cdot \rho \cdot \Delta \mathbf{v}$
8: Compute gradient $\nabla_\theta \mathcal{L}(\theta; \mathcal{B})$ on the mini-batch $\mathcal{B}$
9: Update term $\Delta \mathbf{v} = \rho \cdot \Delta \mathbf{v} + (1-\rho) \cdot \nabla_\theta \mathcal{L}(\theta; \mathcal{B})$
10: Update variable $\theta = \theta - \eta \cdot \Delta \mathbf{v}$
11: **end for**
12: **if** convergence condition holds **then**
13: *tag* = *True*
14: **end if**
15: **end while**
16: **Return** model variable $\theta$

## 4 Adaptive Gradient based Learning Algorithms

이 절에서는 적응형 학습률로 변수를 업데이트하는 학습 알고리즘 그룹을 소개합니다. 여기에 소개된 알고리즘에는 각각 Adagrad, RMSprop 및 Adadelta가 포함됩니다.

### 4.1 Adagrad

이전에 소개된 학습 알고리즘의 경우 이러한 방법의 학습률은 대부분 고정되어 있으며 벡터 $\theta$의 모든 변수에 대해 동일합니다. 그러나 변수 업데이트 프로세스에서는 다양한 변수에 대해 필요한 학습률이 다를 수 있습니다. 변수가 최적에 도달하려면 더 작은 학습률이 필요합니다. 최적에서 멀리 떨어진 변수의 경우 최적에 더 빨리 도달하기 위해 상대적으로 더 큰 학습률을 사용해야 할 수도 있습니다. 이 두 가지 문제를 해결하기 위해 이 부분에서는 Adagrad [3]라는 적응형 그래디언트 방법을 소개합니다.

공식적으로 Adagrad의 학습 방정식은 다음과 같이 표현될 수 있습니다.

$$\theta^{(\tau)} = \theta^{(\tau-1)} - \frac{\eta}{\sqrt{\text{diag}(\mathbf{G}^{(\tau)}) + \epsilon \cdot \mathbf{I}}} \mathbf{g}^{(\tau-1)}, \begin{cases} \mathbf{g}^{(\tau-1)} = \nabla_\theta \mathcal{L}(\theta^{(\tau-1)}, \\ \mathbf{G}^{(\tau)} = \sum_{t=0}^{\tau-1} \mathbf{g}^{(t)} (\mathbf{g}^{(t)})^T \end{cases} \tag{22}$$

위의 업데이트 방정식에서 연산자 $\text{diag}(\mathbf{G}^{(\tau)})$는 $\mathbf{G}$와 동일한 차원의 대각행렬을 정의하지만 $\mathbf{G}^{(\tau)}$의 대각선에 있는 요소만 사용하며 행렬의 대각선에서 0 값의 분할을 방지하기 위한 평활화항입니다. 행렬 $\mathbf{G}^{(\tau)}$는 처음부터 현재 반복까지 계산된 과거 기울기의 기록을 유지합니다. 행렬 $\mathbf{G}^{(\tau)}$의 대각선 값이 다를 것이라는 점을 고려하면 벡터 $\theta(i)$의 반복변수 $\tau$에 대한 $\eta_i^{(\tau)} = \dfrac{\eta}{\sqrt{G^{(\tau)}(i,i)+\epsilon}}$의 학습 속도가 달라집니다. 또한 행렬 $\mathbf{G}^{(\tau)}$는 각 반복마다 업데이트되므로 동일한 변수에 대한 학습률도 서로 다른 반복에서 달라지므로 이 알고리즘을 적응형 학습 알고리즘이라고 합니다. Adagrad의 의사 코드는 알고리즘 6에 제공됩니다.

**Algorithm 6** Adagrad based Mini-batch Gradient Descent

**Require**: Training Set: $\mathcal{T}$; Learning Rate $\eta$; Normal Distribution Std: $\sigma$; Mini-batch Size $b$.
**Ensure**: Model Parameter $\theta$
1: Initialize parameter with Normal distribution $\theta \sim N(0, \sigma^2)$
2: Initialize Matrix $\mathbf{G} = 0$
3: Initialize convergence *tag* = *False*
4: **while** *tag* == *False* **do**
5: Shuffle the training set $\mathcal{T}$
6: **for** each mini-batch $\mathcal{B} \subset \mathcal{T}$ **do**
7: Compute gradient $\mathbf{g} = \nabla_\theta \mathcal{L}(\theta; \mathcal{B})$ on the mini-batch $\mathcal{B}$
8: Update matrix $\mathbf{G} = \mathbf{G} + \mathbf{g}\mathbf{g}^T$
9: Update variable $\theta = \theta - \dfrac{\eta}{\sqrt{\text{diag}(\mathbf{G}) + \epsilon \cdot \mathbf{I}}}\mathbf{g}$
11: **end for**
12: **if** convergence condition holds **then**
13: *tag* = *True*
14: **end if**
15: **end while**
16: **Return** model variable $\theta$

Adagrad의 이러한 학습 메커니즘은 수동 조정이 필요 없는 학습 프로세스의 적응형 학습 속도와 다양한 변수에 대한 다양한 학습 속도를 모두 허용합니다. 또한 반복이 계속됨에 따라 행렬 $\mathbf{G}$의 대각선 값은 감소하지 않습니다. 즉, 학습 과정에서 변수의 학습률이 계속 감소합니다. 또한 변수가 더 이상 훈련 데이터의 정보로 효과적으로 업데이트될 수 없기 때문에 이후 반복에서 학습에 문제가 발생합니다. 또한, Adagrad는 학습 프로세스를 시작하기 위해 초기 학습률 매개변수 $\eta$가 여전히 필요하며, 이는 Adagrad의 단점 중 하나로 처리될 수도 있습니다.

## 4.2 RMSprop

Adagrad에서 학습률(즉, 행렬 $\dfrac{\eta}{\sqrt{\text{diag}(\mathbf{G}^{(\tau)}) + \epsilon \cdot \mathbf{I}}}$ 내 항목들)이 단조롭게 감소하는 문제를 해결하기 위해 이 부분에서는 또 다른 학습알고리즘, 즉 RMSprop cite9을 소개합니다. RMSprop은 Adagrad에 정의된 행렬 $\mathbf{G}$의 과거 누적 기울기의 가중치를 적절하게 감소시키고 업데이트 프로세스에서 학습 속도를 조정할 수도 있습니다. 공식적으로 RMSprop의 변수 업데이트 방정식은 다음 방정식으로 표현될 수 있습니다.

$$\theta^{(\tau)} = \theta^{(\tau-1)} - \frac{\eta}{\sqrt{\text{diag}(\mathbf{G}^{(\tau)}) + \epsilon \cdot \mathbf{I}}}\mathbf{g}^{(\tau-1)}, \tag{23}$$

$$\begin{cases} \mathbf{g}^{(\tau-1)} = \nabla_\theta \mathcal{L}(\theta^{(\tau-1)}), \\ \mathbf{G}^{(\tau)} = \rho \cdot \mathbf{G}^{(\tau-1)} + (1-\rho) \cdot \mathbf{g}^{(\tau-1)}(\mathbf{g}^{(\tau-1)}) \end{cases} \tag{24}$$

위 방정식에서 분모항은 일반적으로 $RMS(\mathbf{g}^{(\tau-1)}) = \sqrt{\text{diag}(\mathbf{G}^{(\tau)} + \epsilon \cdot \mathbf{I}}$는 벡터 $\mathbf{g}^{(\tau-1)}$에 대한 제곱 평균 제곱근을 나타냄)로 표시되기도 합니다. 그러므로 업데이트 방정식은 일반적으로 다음과 같이 작성됩니다.

$$\theta^{(\tau)} = \theta^{(\tau-1)} - \frac{\eta}{RMS(\mathbf{g}^{(\tau-1)})}\mathbf{g}^{(\tau-1)}. \tag{25}$$

행렬 $\mathbf{G}$의 표현에서 매개변수 $\rho$는 역사적으로 누적된 기울기의 가중치를 나타내며, Geoff Hinton의 Coursera 강의에 따르면 $\rho$는 일반적으로 $0.9$로 설정됩니다. $\mathbf{G}$의 표현을 기반으로 현재 반복의 $t$반복에서 계산된 그래디언트에 대해 매우 작은 가중치, 즉 $t$에 따라 기하급수적으로 감소하는 가중치 $\rho^{(t)} \cdot (1-\rho)$가 할당됩니다. RMSprop의 의사 코드는 알고리즘 7에 제공되며 여기서 학습률 $\eta$는 여전히 필요하며 알고리즘의 매개변수로 제공됩니다.

**Algorithm 7** RMSprop based Mini-batch Gradient Descent

**Require**: Training Set: $\mathcal{T}$; Learning Rate $\eta$; Normal Distribution Std: $\sigma$; Mini-batch Size $b$; Parameter $\rho$.
**Ensure**: Model Parameter $\theta$
1: Initialize parameter with Normal distribution $\theta \sim N(0, \sigma^2)$
2: Initialize Matrix $\mathbf{G} = 0$
3: Initialize convergence *tag* = *False*
4: **while** *tag* == *False* **do**
5: Shuffle the training set $\mathcal{T}$
6: **for** each mini-batch $\mathcal{B} \subset \mathcal{T}$ **do**
7: Compute gradient $\mathbf{g} = \nabla_\theta \mathcal{L}(\theta; \mathcal{B})$ on the mini-batch $\mathcal{B}$
8: Update matrix $\mathbf{G} = \mathbf{G} + \mathbf{g}\mathbf{g}^T$
9: Update variable $\theta = \theta - \frac{\eta}{\sqrt{\text{diag}(\mathbf{G}) + \epsilon \cdot \mathbf{I}}}\mathbf{g}$
11: **end for**
12: **if** convergence condition holds **then**
13: *tag* = *True*
14: **end if**
15: **end while**
16: **Return** model variable $\theta$

### 4.3 Adadelta

Adadelta [10]와 RMSprop은 거의 동시에 다른 사람들에 의해 별도로 제안되었으며, 이는 동일한 방법으로 Adagrad에서 단조 감소하는 학습률을 해결합니다. 한편, RMSprop과 비교하여 변수 방정식 업데이트에서 학습률을 제거하는 메커니즘을 도입하여 Adagrad를 더욱 향상 시킵니다. 이전에 RMSprop에서 소개한 다음 업데이트방정식을 기반으로 합니다.

$$\theta^{(\tau)} = \theta^{(\tau-1)} - \frac{\eta}{RMS(\mathbf{g}^{(\tau-1)})}\mathbf{g}^{(\tau-1)} \tag{26}$$

그러나 [10]에서 소개한 바와 같이 지금까지 소개된 학습 알고리즘은 업데이트 과정에서 학습 변수의 단위를 고려하지 못하고 있다. 여기서 단위는 "km", "s" 또는 "kg"일 수 있는 변수의 물리적 의미를 효과적으로 나타냅니다. 매개변수 $\theta$에 가상 단위가 있는 경우 매개변수의 업

데이트, 즉 $\Delta\theta = \nabla_\theta \mathcal{L}(\theta)$ 단위도 같아야 합니다. 그러나 이전에 소개한 SGD, Momentum, NAD 및 Adagrad와 같은 학습 알고리즘의 경우 이러한 가정이 유지될 수 없습니다. 예를 들어, SGD의 경우 업데이트항의 단위는 실제로 $\theta$ 단위의 역수에 비례합니다.

$$\Delta\theta \propto \mathbf{g} \propto \frac{\partial \mathcal{L}(\cdot)}{\partial \theta} \propto \frac{1}{\theta}. \tag{27}$$

이러한 문제를 해결하기 위해 Adadelta는 헤세 근사법을 사용하는 뉴턴 방법과 같은 2차 방법을 살펴볼 것을 제안합니다. 뉴턴 방법에서 업데이트된 변수의 단위는 다음과 같이 표시될 수 있습니다.

$$\Delta\theta \propto \mathbf{H}^{-1}\mathbf{g} \propto \frac{\frac{\partial \mathcal{L}(\cdot)}{\partial \theta}}{\frac{\partial^2 \mathcal{L}(\cdot)}{\partial \theta^2}} \propto \theta \ . \tag{28}$$

여기서 $\mathbf{H} = \frac{\partial^2 \mathcal{L}(\cdot)}{\partial \theta^2}$는 손실함수의 도함수로 계산된 헤세행렬을 나타냅니다. 단위를 일치시키기 위해 뉴턴 방법을 다음과 같이 재배열합니다.

$$\Delta\theta = -\frac{\frac{\partial \mathcal{L}(\cdot)}{\partial \theta}}{\frac{\partial^2 \mathcal{L}(\cdot)}{\partial \theta^2}}, \tag{29}$$

따라서 다음을 유도할 수 있습니다.

$$\mathbf{H}^{-1} = -\frac{1}{\frac{\partial^2 \mathcal{L}(\cdot)}{\partial \theta^2}} = -\frac{\Delta\theta}{\frac{\partial \mathcal{L}(\theta)}{\partial \theta}} \tag{30}$$

따라서 뉴턴 방법을 기반으로 변수에 대한 업데이트 방정식을 다음과 같이 작성할 수 있습니다.

$$\begin{aligned} \theta^{(\tau)} &= \theta^{(\tau-1)} - (\mathbf{H}^{(\tau)})^{-1}\mathbf{g}^{(\tau-1)} \\ &= \theta^{(\tau-1)} - \frac{\Delta\theta^{(\tau)}}{\frac{\partial \mathcal{L}(\theta^{(\tau)})}{\partial \theta}}\mathbf{g}^{(\tau-1)} \end{aligned} \tag{31}$$

RMSprop과 유사하게 Adadelta는 이전 기울기의 RMS를 사용하여 위 방정식의 분모를 근사화합니다. 한편, 현재 반복에서 $\Delta\theta^{(\tau)}$항은 아직 알려지지 않았습니다. Adadelta는 위 방정식에서 분자를 근사화할 것을 제안하고 $\Delta\theta$ 대신 $RMS(\Delta\theta)$를 사용하는 비슷한 방법을 채택하였다. 공식적으로 Adadelta의 변수 업데이트 방정식은 다음과 같이 작성할 수 있습니다.

$$\theta^{(\tau)} = \theta^{(\tau-1)} - \frac{RMS(\Delta\theta^{(\tau-1)})}{RMS(\mathbf{g}^{(\tau-1)})}\mathbf{g}^{(\tau-1)} \tag{32}$$

$RMS(\Delta\theta^{(\tau-1)})$는 이전 반복 $\tau-1$까지 이전 반복에서의 $\Delta\theta$ 기록을 유지합니다. Adadelta 알고리즘의 의사 코드는 알고리즘 8에 제공됩니다. 알고리즘 설명에 따르면 Adadelta는 업데이트 방정식에서 학습률을 사용하지 않으므로 이전에 소개된 Adagrad의 두 가지 약점을 효과적으로 해결합니다.

## 5 Momentum & Adaptive Gradient based Learning Algorithms

이번 절에서는 Adam과 Nadam을 포함하여 Momentum 알고리즘과 적응형 학습률 알고리즘의 장점을 결합한 학습 알고리즘을 각각 소개하겠습니다.

**Algorithm 8** Adadelta based Mini-batch Gradient Descent

**Require**: Training Set: $\mathcal{T}$; Learning Rate $\eta$; Normal Distribution Std: $\sigma$; Mini-batch Size $b$; Parameter $\rho$.
**Ensure**: Model Parameter $\theta$
1: Initialize parameter with Normal distribution $\theta \sim N(0, \sigma^2)$
2: Initialize Matrix $\mathbf{G} = 0$
3: Initialize Matrix $= 0$
4: Initialize convergence *tag* = *False*
5: **while** *tag* == *False* **do**
6: Shuffle the training set $\mathcal{T}$
7: **for** each mini-batch $\mathcal{B} \subset \mathcal{T}$ **do**
8: Compute gradient $\mathbf{g} = \nabla_\theta \mathcal{L}(\theta; \mathcal{B})$ on the mini-batch $\mathcal{B}$
9: Update matrix $\mathbf{G} = \mathbf{G} + \mathbf{g}\mathbf{g}^T$
10: Compute updating vector $\Delta\theta = -\frac{\sqrt{\text{diag}(\ ) + \epsilon \cdot \mathbf{I}}}{\sqrt{\text{diag}(\mathbf{G}) + \epsilon \cdot \mathbf{I}}}$
11: Update matrix $= \rho \cdot + (1-\rho) \cdot \Delta\theta(\Delta\theta)^T$
12: Update variable $\theta = \theta + \Delta\theta$
13: **end for**
14: **if** convergence condition holds **then**
15: *tag* = *True*
16: **end if**
17: **end while**
18: **Return** model variable $\theta$

## 5.1 Adam

최근에는 메모리 요구사항이 거의 없이 1차 기울기만 계산하는 *Adaptive Moment Estimation* (Adam) [4]이라는 새로운 학습알고리즘이 도입되었습니다. RMSprop 및 Adadelta와 유사하게 Adam은 과거 1차 기울기의 제곱에 대한 기록을 유지합니다. Adam은 또한 과거의 1차 기울기에 대한 기록도 유지하는데, 둘 다 학습과정에서 기하급수적으로 감소합니다. 공식적으로 벡터 $\mathbf{m}^{(\tau)}$ 및 $\mathbf{v}^{(\tau)}$를 사용하여 1차 기울기와 1차 기울기의 제곱을 각각 저장하는 항을 나타낼 수 있으며, 구체적인 표현은 다음과 같습니다.

$$\begin{aligned} \mathbf{m}^{(\tau)} &= \beta_1 \cdot \mathbf{m}^{(\tau-1)} + (1-\beta_1) \cdot \mathbf{g}^{(\tau-1)}, \\ \mathbf{v}^{(\tau)} &= \beta_2 \cdot \mathbf{v}^{(\tau-1)} + (1-\beta_2) \cdot \mathbf{g}^{(\tau-1)} \odot \mathbf{g}^{(\tau-1)} \end{aligned} \tag{33}$$

여기서 벡터 $\mathbf{g}^{(\tau-1)} = \nabla_\theta \mathcal{L}(\theta^{(\tau-1)})$, $\mathbf{g}^{(\tau-1)} \odot \mathbf{g}^{(\tau-1)}$ 는 벡터의 요소별 곱을 나타냅니다. 위의 등식에서 $\mathbf{v}^{(\tau)}$는 실제로 방정식 (24)에서 사용된 $\text{diag}(\mathbf{G}^{(\tau)})$와 동일한 정보를 저장합니다. 그러나 Adam 알고리즘은 전체 행렬을 저장하는 대신 이러한 벡터 레코드를 유지하여 공간을 덜 차지하는 경향이 있습니다.

[4]에 소개된 것처럼 벡터 $\mathbf{m}^{(\tau)}$와 $\mathbf{v}^{(\tau)}$는 특히 $\beta_1$과 $\beta_2$가 1에 가까울 때 0 쪽으로 편향됩니다. 왜냐하면 $\mathbf{m}^{(0)}$과 $\mathbf{v}^{(0)}$이 각각 0벡터로 초기화되기 때문입니다. 이러한 문제를 해결하기 위해 Adam은 다음과 같이 용어 재조정을 도입합니다.

$$\begin{aligned} \hat{\mathbf{m}}^{(\tau)} &= \frac{\mathbf{m}^{(\tau)}}{1-\beta_1^\tau}, \\ \hat{\mathbf{v}}^{(\tau)} &= \frac{\mathbf{v}^{(\tau)}}{1-\beta_2^\tau}, \end{aligned} \tag{34}$$

여기서 $\beta_1^\tau$ 및 $\beta_2^\tau$ 항의 위첨자 $\tau$는 이전에 쓰인 반복 횟수 지수 $(\tau)$ 대신 거듭제곱을 나타냅니다.

재조정된 벡터 $\hat{\mathbf{m}}^{(\tau)}$ 및 행렬 $\hat{\mathbf{v}}^{(\tau)}$를 기반으로, Adam은 다음 방정식을 사용하여 모델 변수를 업데이트합니다.

$$\theta^{(\tau)} = \theta^{(\tau-1)} - \frac{\eta}{\sqrt{\hat{\mathbf{v}}^{(\tau)}} + \epsilon} \odot \hat{\mathbf{m}}^{(\tau)} \tag{35}$$

위의 설명에 따르면 Adam은 RMSprop 알고리즘과 Momentum 알고리즘을 통합한 것으로 볼 수 있으며, 이는 더 빠른 수렴과 적응형 학습 속도를 동시에 가능하게 합니다. 공식적으로 Adam 학습 알고리즘의 의사 코드는 알고리즘 9에 제공되며, 여기서 $\beta_1$과 $\beta_2$는 알고리즘의 매개변수로 입력됩니다. [4]에 따르면 Adam의 매개변수는 다음과 같이 초기화 될 수 있습니다: $\epsilon = 10^{-8}, \beta_1 = 0.9$ 및 $\beta_2 = 0.999$.

**Algorithm 9** Adam

**Require**: Training Set: $\mathcal{T}$; Learning Rate $\eta$; Normal Distribution Std: $\sigma$; Mini-batch Size $b$; Decay Parameters $\beta_1, \beta_2$.
**Ensure**: Model Parameter $\theta$
Initialize parameter with Normal distribution $\theta \sim N(0, \sigma^2)$
Initialize vector $\mathbf{m} = 0$
Initialize vector $\mathbf{v} = 0$
Initialize step $\tau = 0$
Initialize convergence *tag* $=$ *False*
**while** *tag* $==$ *False* **do**
  Shuffle the training set $\mathcal{T}$
  **for** each mini-batch $\mathcal{B} \subset \mathcal{T}$ **do**
    Update step $\tau = \tau + 1$
    Compute gradient vector $\mathbf{g} = \nabla_\theta \mathcal{L}(\theta; \mathcal{B})$ on the mini-batch $\mathcal{B}$
    Update vector $\mathbf{m} = \beta_1 \cdot \mathbf{m} + (1 - \beta_1) \cdot \mathbf{g}$
    Update vector $\mathbf{v} = \beta_2 \cdot \mathbf{v} + (1 - \beta_2) \cdot \mathbf{g} \odot \mathbf{g}$
    Rescale vector $\hat{\mathbf{m}} = \dfrac{\mathbf{m}}{(1 - \beta_1^\tau)}$
    Rescale vector $\hat{\mathbf{v}} = \dfrac{\mathbf{v}}{(1 - \beta_2^\tau)}$
    Update variable $\theta = \theta - \dfrac{\eta}{\sqrt{\hat{\mathbf{v}}} + \epsilon} \odot \hat{\mathbf{m}}$
  **end for**
  **if** convergence condition holds **then**
    *tag* $=$ *True*
  **end if**
**end while**
**Return** model variable $\theta$

## 5.2 Nadam

앞절에서 소개한 Adam은 Momentum 기반 학습 알고리즘과 바닐라 모멘텀을 적용한 Adaptive Gradient 기반 학습 알고리즘을 통합한 것으로 볼 수 있습니다. [2]에서는 바닐라 운동량을 대신 Nesterov의 가속 경사(NAG)로 대체하는 학습 알고리즘이 도입되었으며, 새로운 학습 알고리즘을 Nesterov accelerated Adam(Nadam)이라고 합니다. Nadam의 업데이트 방정식을 제공하기 전에 NAG의 업데이트 방정식을 다시 다음과 같이 설명하고

싶습니다 (21):

$$\theta^{(\tau)} = \theta^{(\tau-1)} - \eta \cdot \Delta\mathbf{m}^{(\tau)}, \quad \begin{cases} \Delta\mathbf{v}^{(\tau)} = \rho \cdot \Delta\mathbf{v}^{(\tau-1)} + (1-\rho) \cdot \nabla_\theta \mathcal{L}(\hat{\theta}^{(\tau)})), \\ \hat{\theta}^{(\tau)} = \theta^{(\tau-1)} - \eta \cdot \rho \cdot \Delta\mathbf{v}^{(\tau-1)}. \end{cases} \tag{36}$$

위 업데이트 방정식에 따르면 운동량항 $\Delta\mathbf{v}^{(\tau-1)}$는 이 과정에서 두 번 사용됩니다.

(1) $\hat{\theta}^{(\tau)})$를 계산하는 데 사용되는 $\hat{\theta}^{(\tau)}$; 그리고

(2) $\hat{\theta}^{(\tau)})$를 계산하는 데 사용되는 $\Delta\mathbf{v}^{(\tau-1)}$

Nadam은 대신 다음 업데이트 방정식을 사용하여 위의 방법을 변경할 것을 제안합니다.

$$\theta^{(\tau)} = \theta^{(\tau-1)} - \eta \cdot (\rho \cdot \Delta\mathbf{v}^{(\tau)} + (1-\rho) \cdot \nabla_\theta \mathcal{L}(\theta^{(\tau-1)})), \tag{37}$$

여기서

$$\Delta\mathbf{v}^{(\tau)} = \rho \cdot \Delta\mathbf{v}^{(\tau-1)} + (1-\rho) \cdot \nabla_\theta \mathcal{L}(\theta) \tag{38}$$

NAG와 비교하여 위의 방정식은 기울기항을 계산할 때 미리 예측하지 않습니다. 바닐라 운동량과 비교할 때 위 방정식은 현재 반복에서 운동량 항과 기울기항을 모두 사용합니다. 업데이트 방정식에 사용된 항 $\rho \cdot \Delta\mathbf{v}^{(\tau)} + (1-\rho) \cdot \nabla_\theta \mathcal{L}(\theta^{(\tau-1)})$은 실제로 다음 반복에서 $\Delta\mathbf{v}^{(\tau+1)}$에 대한 근사치이며, 이는 변수 업데이트를 미리 예측하는 목적을 달성합니다.

한편, 방정식 (35)에 따르면 Adam의 업데이트 방정식을 다음과 같이 다시 작성할 수 있습니다.

$$\theta^{(\tau)} = \theta^{(\tau-1)} - \frac{\eta}{\sqrt{\hat{\mathbf{v}}^{(\tau)}} + \epsilon} \odot \hat{\mathbf{m}}^{(\tau)}, \tag{39}$$

$$\hat{\mathbf{m}}^{(\tau)} = \frac{\mathbf{m}^{(\tau)}}{1-\beta_1^\tau}, \ \mathbf{m}^{(\tau)} = \beta_1 \cdot \mathbf{m}^{(\tau-1)} + (1-\beta_1) \cdot \mathbf{g}^{(\tau)}. \tag{40}$$

$\hat{\mathbf{m}}^{(\tau)}$ 항을 방정식 (39)로 대체하면 다음과 같이 다시 작성할 수 있습니다.

$$\begin{aligned} \theta^{(\tau)} &= \theta^{(\tau-1)} - \frac{\eta}{\sqrt{\hat{\mathbf{v}}^{(\tau)}} + \epsilon} \odot \left( \frac{\beta_1 \cdot \mathbf{m}^{(\tau-1)}}{1-\beta_1^\tau} + \frac{(1-\beta_1) \cdot \mathbf{g}^{(\tau)}}{1-\beta_1^\tau} \right) \\ &= \theta^{(\tau-1)} - \frac{\eta}{\sqrt{\hat{\mathbf{v}}^{(\tau)}} + \epsilon} \odot \left( \beta_1 \cdot \hat{\mathbf{m}}^{(\tau-1)} + \frac{(1-\beta_1) \cdot \mathbf{g}^{(\tau)}}{1-\beta_1^\tau} \right) \end{aligned} \tag{41}$$

식 (37)에 제공된 분석과 유사하게, Nadam은 괄호 안에 사용된 $\hat{\mathbf{m}}^{(\tau-1)}$ 항을 $\hat{\mathbf{m}}^{(\tau)}$로 대체하여 예측을 제안하고, 이를 이용하여 다음 업데이트 방정식을 얻을 수 있다.

$$\theta^{(\tau)} = \theta^{(\tau-1)} - \frac{\eta}{\sqrt{\hat{\mathbf{v}}^{(\tau)}} + \epsilon} \odot \left( \beta_1 \cdot \hat{\mathbf{m}}^{(\tau)} + \frac{(1-\beta_1) \cdot \mathbf{g}^{(\tau)}}{1-\beta_1^\tau} \right) \tag{42}$$

Nadam 알고리즘의 의사 코드는 알고리즘 10에 제공되며, 여기서 대부분의 코드는 변수 $\theta$는 업데이트 되는 마지막 줄을 제외하고는 알고리즘 9의 코드와 동일합니다.

## 6 Hybrid Evolutionary Gradient Descent Learning Algorithms

Adam은 다양한 변형과 함께 대규모 문제 그룹을 최적화하는 데 효과적인 것으로 나타났습니다. 그러나 딥 러닝 모델의 볼록하지 않은 목적 함수의 경우 Adam은 반복 업데이트

**Algorithm 10** Nadam

**Require**: Training Set: $\mathcal{T}$; Learning Rate $\eta$; Normal Distribution Std: $\sigma$; Mini-batch Size $b$; Decay Parameters $\beta_1, \beta_2$.
**Ensure**: Model Parameter $\theta$
Initialize parameter with Normal distribution $\theta \sim N(0, \sigma^2)$
Initialize vector $\mathbf{m} = 0$
Initialize vector $\mathbf{v} = 0$
Initialize step $\tau = 0$
Initialize convergence *tag* $=$ *False*
**while** *tag* $==$ *False* **do**
  Shuffle the training set $\mathcal{T}$
  **for** each mini-batch $\mathcal{B} \subset \mathcal{T}$ **do**
    Update step $\tau = \tau + 1$
    Compute gradient vector $\mathbf{g} = \nabla_\theta \mathcal{L}(\theta; \mathcal{B})$ on the mini-batch $\mathcal{B}$
    Update vector $\mathbf{m} = \beta_1 \cdot \mathbf{m} + (1 - \beta_1) \cdot \mathbf{g}$
    Update vector $\mathbf{v} = \beta_2 \cdot \mathbf{v} + (1 - \beta_2) \cdot \mathbf{g} \odot \mathbf{g}$
    Rescale vector $\hat{\mathbf{m}} = \dfrac{\mathbf{m}}{(1 - \beta_1^\tau)}$
    Rescale vector $\hat{\mathbf{v}} = \dfrac{\mathbf{v}}{(1 - \beta_2^\tau)}$
    Update variable $\theta = \theta - \dfrac{\eta}{\sqrt{\hat{\mathbf{v}}} + \epsilon} \odot \left( \beta_1 \odot \hat{\mathbf{m}} + \dfrac{(1 - \beta_1) \cdot \mathbf{g}}{1 - \beta_1^\tau} \right)$
  **end for**
  **if** convergence condition holds **then**
    *tag* $=$ *True*
  **end if**
**end while**
**Return** model variable $\theta$

프로세스가 필연적으로 로컬 최적 상태에 걸릴 수 있는 전역적으로 최적의 솔루션을 식별한다고 보장할 수 없습니다. Adam의 성능은 그다지 강력하지 않습니다. 이는 매끄럽지 않은 모양이나 잡음이 많은 데이터(noisy data)로 오염된 학습 시나리오의 목적 함수에 대해 크게 저하됩니다. 또한 Adam의 분산 계산 프로세스에는 많은 동기화가 필요하므로 대규모 클러스터 기반 분산 계산 플랫폼에서의 채택을 방해할 수 있습니다.

한편, 진화 알고리즘의 자연 선택 과정에서 영감을 얻은 메타휴리스틱 알고리즘인 유전 알고리즘(GA)도 많은 최적화 문제의 해결책을 학습하는 데 널리 사용되어 왔습니다. GA에서는 후보 솔루션 집단이 초기화되고 더 나은 솔루션으로 발전됩니다. 경사 하강 기반 방법 대신 심층 신경망 모델 [12, 7, 1]을 훈련하기 위해 GA를 사용하려는 여러 시도도 있었습니다. GA는 여러 로컬 최적값을 포함하는 볼록하지 않은 목적 함수, 매끄럽지 않은 모양의 목적 함수, 많은 수의 매개변수 및 잡음이 있는 환경과 같은 많은 학습 시나리오에서 탁월한 성능을 보여주었습니다. GA는 또한 병렬/분산 컴퓨팅 설정에 매우 적합하며, 학습 프로세스를 병렬/분산 컴퓨팅 플랫폼에 쉽게 배포할 수 있습니다. 한편, Adam과 비교하여 GA는 최적화 목적 함수를 해결하는 데 수렴하는 데 더 많은 라운드가 필요할 수 있습니다.

이번 절에서는 Adam과 GA를 통합 학습 방식으로 통합한 Gadam(Genetic Adaptive Momentum Estimation)[11]이라는 새로운 최적화 알고리즘을 소개합니다. 하나의 단일 모델 솔루션을 배우는 대신 Gadam은 잠재적인 단위 모델 솔루션 그룹과 함께 작업합니다. 학습 과정에서 가담은 아담과 함께 단위 모델을 학습하고 유전 알고리즘을 통해 이를 새로운

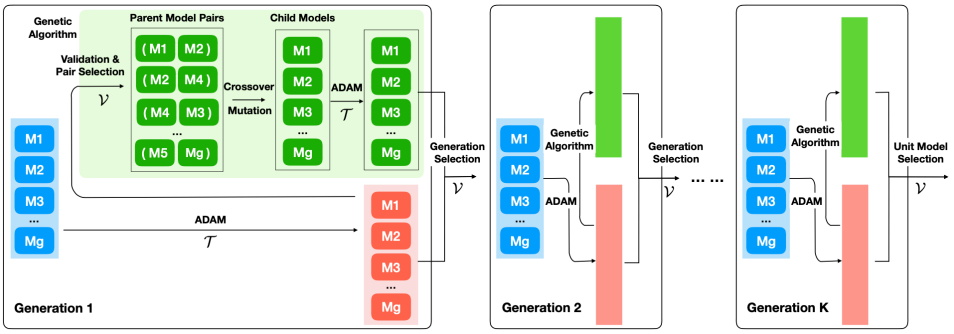


Figure 2. Overall Architecture of Gadam Model.

세대로 진화시킨다. 또한 알고리즘 Gadam은 독립형 및 병렬/분산 모드 모두에서 작동할 수 있으며 이 절에서도 이에 대해 연구합니다.

## 6.1 Gadam

Gadam의 모델 학습 프로세스의 전체 프레임워크는 그림 2에 설명되어 있으며, 여기에는 여러 학습 세대가 포함됩니다. 각 세대마다 데이터로부터 Adam을 통해 단위 모델 변수 그룹이 학습되며, 이는 유전 알고리즘을 통해 효과적으로 진화됩니다. 좋은 후보 변수가 선정되어 다음 세대를 형성하게 됩니다. 이러한 반복적인 모델 학습 과정은 수렴될 때까지 계속되며, 최종 생성 시 최적의 단위 모델 변수가 출력 모델 솔루션으로 선택됩니다. 단순화를 위해 단위 모델과 해당 변수를 차이점을 구별하지 않고 서로 바꿔서 참조하겠습니다.

### 6.1.1 Model Population Initialization

Gadam은 단위 모델 집합을 기반으로 최적의 모델 변수 (즉, 기본적으로 이러한 단위 모델), 즉, 단위 모델 모집단, 여기서 초기 단위 모델 생성은 집합 $\mathcal{G}^{(k)} = \{M_1^{(0)}, M_2^{(0)}, \cdots, M_g^{(0)}\}$($g$는 인구의 크기, 위첨자는 생성 인덱스를 나타낸다)으로 표시할 수 있습니다. Gadam은 초기 세대를 기반으로 단위 모델을 새로운 세대로 진화시킬 예정이며, 이는 각각 $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \cdots, \mathcal{G}^{(K)}$로 표현할 수 있습니다. 여기서 매개변수 $K$는 전체 생성 수를 나타냅니다.

각 단위 모델, 예를 들어, 초기 생성 $\mathcal{G}^{(0)}$, 예를 들어 $M_i^{(0)}$에서 변수 $\theta_i^{(0)}$는 특정 분포(예: 표준 정규 분포)에서 샘플링된 무작위 값으로 Gadam에서 초기화됩니다. 이러한 초기 생성은 검색 시작 지점의 역할을 하며, 이를 통해 Gadam은 최적의 솔루션을 식별하기 위해 다른 지역으로 확장됩니다. 한편, 다음 세대의 단위 모델의 경우 변수 값은 각각 상위 모델에서 GA를 통해 생성됩니다.

### 6.1.2 Model Learning with Adam

학습 과정에서 임의의 모델 생성 $\mathcal{G}^{(k)}(k \in \{1, 2, \cdots, K\})$가 주어지면 Gadam은 Adam과 함께 각 단위 모델에 대한 (국소적으로) 최적 변수를 학습한다. 공식적으로 Gadam은 여러 에포크로 단위 모델을 학습한다. 각 에포크에서 각 단위 모델 $M_i^{(k)} \in \mathcal{G}^{(k)}$에 대해 훈련 데이

터 세트에서 분리된 훈련 배치가 무작위로 샘플링되며, 이는 $\mathcal{B} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \cdots, (\mathbf{x}_b, \mathbf{y}_b)\} \subset \mathcal{T}$로 나타낸다. (여기서 $b$는 배치 크기를 나타내고 $\mathcal{T}$는 전체 훈련 세트를 나타낸다.) 미니 배치 $\mathcal{B}$를 훈련하기 위해 단위 모델 $M_i^{(k)}$에 의해 도입된 손실 함수를 $\ell(\theta_i^{(k)})$라고 하고, 훈련 인스턴스에 의해 학습된 모델 변수 벡터를 다음과 같이 나타낼 수 있다.

$$\bar{\theta}_i^{(k)} = Adam(\ell(\theta_i^{(k)})) \tag{43}$$

손실 함수는 특정 단위 모델 및 애플리케이션 설정에 따라 평균 제곱 손실, 힌지 손실 또는 교차 엔트로피 손실과 같은 서로 다른 표현을 갖게 됩니다. 이러한 학습 과정은 수렴할 때까지 계속되며, (국소적으로) 최적 변수를 가진 업데이트 된 모델 생성 $\bar{\mathcal{G}}^{(k)} = \{\bar{M}_1^{(k)}, \bar{M}_2^{(k)}, \cdots, \bar{M}_g^{(k)}\}$($g$로 표현될 수 있으며, 이들의 해당 변수 벡터는 각각 $\{\bar{\theta}_1^{(k)}, \bar{\theta}_2^{(k)}, \cdots, \bar{\theta}_g^{(k)}\}$로 표현될 수 있습니다.

### 6.1.3 Model Evolution with Genetic Algorithm

이 부분에서 Gadam은 학습된 단위 모델, 즉 $\bar{\mathcal{G}}^{(k)}$를 기반으로 유전자 알고리즘을 통해 효과적으로 단위 모델에 대한 더 나은 솔루션을 추가로 검색할 것입니다.

**Model Fitness Evaluation**

성능이 좋은 단위 모델은 학습 과제에 더 잘 맞을 수 있다. 무작위로 모델을 진화시키는 대신, Gadam은 진화할 현재 세대에서 좋은 단위 모델을 선택할 것을 제안한다. 샘플링된 검증 배치 $\mathcal{V} \subset \mathcal{T}$에 기초하여, 단위 모델의 적합도 점수, 예를 들어 $\bar{M}_i^{(k)}$는 다음과 같이 $\mathcal{V}$에 도입된 손실 항에 기초하여 효과적으로 계산될 수 있다.

$$\mathcal{L}_i^{(k)} = \mathcal{L}(\bar{M}_i^{(k)}; \mathcal{V}) = \sum_{(\mathbf{x}_j, \mathbf{y}_j) \in \mathcal{V}} \ell(\mathbf{x}_j, \mathbf{y}_j; \bar{\theta}_i^{(k)}). \tag{44}$$

계산된 손실 값을 바탕으로 단위 모델 $\bar{M}_i^{(k)}$의 선택 확률은 다음 softmax방정식으로 정의될 수 있다.

$$P(\bar{M}_i^{(k)}) = \frac{\exp^{(-\hat{\mathcal{L}}_i^{(k)})}}{\sum_{j=1}^{g} \exp^{(-\hat{\mathcal{L}}_j^{(k)})}} \tag{45}$$

손실 값의 필요한 정규화는 일반적으로 실제 애플리케이션에서 필요한데 $\exp^{(-\mathcal{L}_i^{(k)})}$가 아주 큰 양의 손실 값 또는 아주 작은 음의 손실 값 $\mathcal{L}_i^{(k)}$에 대해 0 또는 $\infty$에 접근할 수 있기 때문이다. 확률 방정식에서 알 수 있듯이 모든 단위 모델의 정규화된 손실항은 형식적으로 $[\hat{\mathcal{L}}_1^{(k)}, \hat{\mathcal{L}}_2^{(k)}, \cdots, \hat{\mathcal{L}}_g^{(k)}]$로 표현할 수 있으며, 여기서 $\hat{\mathcal{L}}_1^{(k)} \in [0, 1], \forall i \in \{1, 2, \cdots, g\}$. 계산된 확률에 따라, 단위 모델 집합 $\bar{\mathcal{G}}^{(k)}$으로부터 단위 모델들의 $g$개의 쌍이 진화의 상위 모델로 대체되어 선택되며, 이는 $\mathcal{P} = \{(\bar{M}_{i_1}^{(k)}, \bar{M}_{j_1}^{(k)}), (\bar{M}_{i_2}^{(k)}, \bar{M}_{j_2}^{(k)}), \cdots, (\bar{M}_{i_g}^{(k)}, \bar{M}_{j_g}^{(k)})\}$로 표시될 수 있습니다.

**Unit Model Crossover**

단위 모형 쌍 예를 들어 $(\bar{M}_{i_p}^{(k)}, \bar{M}_{j_p}^{(k)}) \in \mathcal{P}$가 주어지면, Gadam은 교차 연산을 통해 변수를 자식 모형에 상속한다. 교차 연산에서 부모 모형 즉, $\bar{M}_{i_p}^{(k)}$와 $\bar{M}_{j_p}^{(k)}$는 성능이 더 좋은 부모 모형이 더 유리한 경향이 있는 채로 서로 경쟁할 것이다. 우리는 $(\bar{M}_{i_p}^{(k)}, \bar{M}_{j_p}^{(k)})$로부터 생성된 자식 모형을 $\tilde{M}_p^{(k)}$로 나타낼 수 있다. 자식 모형 $\tilde{M}_{i_p}^{(k)}$의 가중치 변수 $\tilde{\theta}_p^{(k)}$의 각 항목에 대해,

Gadam은 다음과 같이 값을 초기화한다:

$$\tilde{\theta}_p^{(k)}(m) = \mathbf{1}(\text{rand} \leq p_{i_p,j_p}^{(k)}) \cdot \bar{\theta}_{i_p}^{(k)}(m) + \mathbf{1}(\text{rand} > p_{i_p,j_p}^{(k)}) \cdot \bar{\theta}_{j_p}^{(k)}(m). \tag{46}$$

위의 식에서 이진 함수 $\mathbf{1}(\cdot)$의 함수값이 1이기 위한 필요충분조건은 조건이 만족하는 것이다. "rand"항이 $[0,1]$ 구간에 있는 임의의 수를 나타낸다. 확률 경계값 $p_{i_p,j_p}^{(k)}$는 부모 모델의 성능을 바탕으로 하여 정의된다.

$$p_{i_p,j_p}^{(k)} = \frac{\exp^{(-\hat{\mathcal{L}}_{i_p}^{(k)})}}{\exp^{(-\hat{\mathcal{L}}_{i_p}^{(k)})} + \exp^{(-\hat{\mathcal{L}}_{j_p}^{(k)})}}. \tag{47}$$

만약, $\hat{\mathcal{L}}_{i_p}^{(k)}) > \hat{\mathcal{L}}_{j_p}^{(k)})$, 즉 모델 $\bar{M}_{i_p}^{(k)}$가 $\bar{M}_{j_p}^{(k)}$보다 큰 손실을 초래하면, $0 < p_{i_p,j_p}^{(k)} < \frac{1}{2}$가 된다.

이러한 과정을 통해 집합 $\mathcal{P}$의 전체 부모 모델 쌍을 기반으로 Gadam은 자식 모델 집합 $\tilde{\mathcal{G}}^{(k)} = \{\tilde{M}_1^{(k)}, \tilde{M}_2^{(k)}, \cdots, \tilde{M}_g^{(k)}\}$을 생성할 수 있습니다.

**Unit Model Mutation**

단위 모델들이 로컬 최적점에 갇히는 것을 피하기 위해, Gadam은 생성된 자식 모델의 변수 값을 집합 $\{\tilde{M}_1^{(k)}, \tilde{M}_2^{(k)}, \cdots, \tilde{M}_g^{(k)}\}$에서 조정하는 돌연변이라는 연산을 채택한다. 형식적으로, 벡터 $\tilde{\theta}_q^{(k)}$로 매개변수화된 각 자식 모델 $\tilde{M}_q^{(k)}$에 대해, Gadam은 다음 방정식에 따라 변수 벡터를 돌연변이시킬 것이고, 여기서 $m_{th}$ 항목은 다음과 같이 업데이트 될 수 있다.

$$\tilde{\theta}_q^{(k)}(m) = \mathbf{1}(\text{rand} \leq p_q) \cdot \text{rand}(0,1) + \mathbf{1}(\text{rand} > p) \cdot \tilde{\theta}_q^{(k)}(m), \tag{48}$$

식에서 용어 $p_q$ 항은 부모 모델의 성능과 밀접한 상관관계가 있는 돌연변이 비율을 나타냅니다:

$$p_q = p \cdot \left(1 - P(\bar{M}_{i_q}^{(k)}) - P(\bar{M}_{j_q}^{(k)})\right) \tag{49}$$

부모 모델이 좋은 자식 모델의 경우 돌연변이율이 더 낮습니다. 항 $p$는 일반적으로 작은 값 (예: 0.01)인 기본 돌연변이율을 나타내며, 확률 $P(\bar{M}_{i_q}^{(k)})$ 및 $P(\bar{M}_{j_q}^{(k)})$는 식 (45)에 정의됩니다. 이러한 단위 모델은 수렴할 때 까지 Adam과 함께 추가로 학습되어 $k$번째 자식 모델 $\tilde{\mathcal{G}}^{(k)} = \{\tilde{M}_1^{(k)}, \tilde{M}_2^{(k)}, \cdots, \tilde{M}_g^{(k)}\}$ 생성으로 이어집니다.

### 6.1.4 New Generation Selection and Evolution Stop Criteria

학습된 부모 모델 집합 $\bar{\mathcal{G}}^{(k)} = \{\bar{M}_1^{(k)}, \bar{M}_2^{(k)}, \cdots, \bar{M}_g^{(k)}\}$ 및 자식 모델 집합 $\tilde{\mathcal{G}}^{(k)} = \{\tilde{M}_1^{(k)}, \tilde{M}_2^{(k)}, \cdots, \tilde{M}_g^{(k)}\}$에서 학습된 단위 모델 중 Gadam은 공유된 새로운 검증 배치를 기반으로 적합도 점수를 재평가할 것입니다.$\bar{\mathcal{G}}^{(k)} \cup \tilde{\mathcal{G}}^{(k)}$의 모든 단위 모델 중 상위 $g$ 단위 모델을 선택하여 $(k+1)$번째 세대를 형성하며, 이는 공식적으로 집합 $\mathcal{G}^{(k+1)} = \{M_1^{(k+1)}, M_1^{(k+1)}, \cdots, M_g^{(k+1)}\}$로 표현할 수 있습니다. 이러한 진화적 학습 프로세스는 최대 생성 수에 도달했거나 결과적으로 세대 간에 큰 개선이 없는 경우 (예를 들어, $\mathcal{G}^{(k)}$ 및 $\mathcal{G}^{(k+1)}$) 중지됩니다:

$$\left| \sum_{M_i^{(k)} \in \mathcal{G}^{(k)}} \mathcal{L}_i^{(k)} - \sum_{M_i^{(k+1)} \in \mathcal{G}^{(k+1)}} \mathcal{L}_i^{(k+1)} \right| \leq \lambda, \tag{50}$$

위의 방정식은 Gadam의 중지 기준을 정의하며, 여기서 $\lambda$는 진화 중지 임계값입니다.

Gadam 최적화 알고리즘은 실제로 Adam과 유전 알고리즘의 장점을 모두 통합합니다. Adam을 사용하면 단위 모델은 몇 번의 훈련 기간을 통해 매우 빠르게 (로컬/글로벌) 최적의 솔루션을 효과적으로 달성할 수 있습니다. 한편, 다수의 단위 모델을 기반으로 한 유전자 알고리즘을 통해 여러 출발점에서 솔루션을 검색하고 로컬 최적점에서 뛰어오르는 기회도 제공할 것입니다. [4]에 따르면 매끄러운 함수의 경우 함수 기울기가 사라지면 Adam은 수렴하게 됩니다. 반면에 유전자 알고리즘은 [8]에 따라 수렴할 수도 있습니다. 이러한 사전 지식을 바탕으로 [11]에서 소개한 바와 같이 가담의 수렴을 증명할 수 있습니다. Gadam을 사용한 단위 모델 훈련은 병렬/분산 컴퓨팅 플랫폼에 효과적으로 배포될 수 있으며, 여기서 Gadam과 관련된 각 단위 모델은 별도의 프로세스/서버를 통해 학습될 수 있습니다. 프로세스/서버 중 통신 비용은 미미하며 이는 단지 크로스오버 단계에만 존재합니다. 말 그대로 각세대의 모든 $g$ 단위 모델 중에서 통신비용은 $O(k \cdot g \cdot d_\theta)$ 입니다. 여기서 $d_\theta$는 벡터 $\theta$의 차원을 나타내고 $k$는 Adam의 수렴을 달성하는데 필요한 훈련 epoch를 나타냅니다.

## 7 A Summary

본 논문에서는 딥러닝 모델학습을 위한 다양한 최신 변형알고리즘과 함께 경사 하강법 알고리즘을 소개했습니다. 이 방향의 최근 발전상황을 바탕으로 이 문서는 가까운 시일 내에 적절하게 업데이트될 예정입니다.

## References

1. Eli David and Iddo Greental. Genetic algorithms for evolving deep neural networks. *CoRR*, abs/1711.07655, 2017.
2. Timothy Dozat. Incorporating Nesterov Momentum into Adam.
3. John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.
4. Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.
5. Yurii Nesterov. A method of solving a convex programming problem with convergence rate O(1/sqr(k)). *Soviet Mathematics Doklady*, 27:372–376, 1983.
6. Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Netw.*, 12(1):145–151, January 1999.
7. Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competi- tive alternative for training deep neural networks for reinforcement learning. *CoRR*, abs/1712.06567, 2017.
8. Dirk Thierens and David E. Goldberg. Convergence models of genetic algorithm selection schemes. In *Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature*, PPSN III, pages 119–129, London, UK, UK, 1994. Springer-Verlag.

9. T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. *COURSERA*: Neural Networks for Machine Learning, 2012.
10. Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.
11. Jiawei Zhang and Fisher B. Gouza. GADAM: genetic-evolutionary ADAM for deep neural network optimization. *CoRR*, abs/1805.07500, 2018.
12. Jiawei Zhang and Fisher B. Gouza. SEGEN: sample-ensemble genetic evolutional network model. *CoRR*, abs/1803.08631, 2018.