정확한 대규모 미니배치 SGD: 1시간 만에 ImageNet 훈련하기

프리야 고얄 Piotr Dolla´r 로스 거식 피터 노르트후이스
루카스 웨솔로프스키 아포 키롤라 앤드류 툴록 양칭 지아카이밍 허

페이스북

초록

딥 러닝은 대규모 신경망과 대규모 데이터 세트에서 성공할 수 있습니다. 그러나 네트워크와 데이터 세트가 커지면 학습 시간이 길어져 재검색과 개발 진행에 방해가 됩니다. 분산 동기식 SGD는 병렬 작업자 풀을 통해 SGD 미니 배치를 분할함으로써 이 문제를 해결할 수 있는 잠재적인 솔루션을 제공합니다. 하지만 이 방식이 효율적이려면 작업자당 워크로드가 커야 하며, 이는 SGD 미니 배치 크기가 크게 증가한다는 것을 의미합니다. 이 논문에서는 ImageNet 데이터 세트에서 대규모 미니배치는 최적화 문제를 야기하지만, 이러한 문제를 해결하면 학습된 네트워크가 우수한 일반화를 보인다는 것을 실증적으로 보여줍니다. 특히 최대 8192개 이미지의 대규모 미니배치 크기로 훈련할 때 정확도 손실이 없는 것으로 나타났습니다. 이러한 결과를 달성하기 위해 유니티는 미니배치 크기에 따라 학습 속도를 조정하기 위해 매개변수가 없는 선형 스케일링 규칙을 채택하고, 훈련 초기에 최적화 문제를 극복하는 새로운 워밍업 체계를 개발했습니다. 이러한 간단한 기법을 통해 유니티의 카페2 기반 시스템은 256개의 GPU에서 8192개의 미니배치 크기로 1시간 만에 작은 미니배치 정확도를 맞추면서 ResNet-50을 훈련할 수 있습니다. 상용 하드웨어를 사용하여 구현한 결과, 8개에서 256개 GPU로 이동할 때 약 90%의 확장 효율을 달성했습니다. 이 연구 결과를 통해 인터넷 규모의 데이터에 대한 시각 인식 모델을 높은 효율로 훈련할 수 있습니다.

1. 소개

규모가 중요합니다. 우리는 데이터와 모델 규모가 증가함에 따라 컴퓨터 비전[22, 41, 34, 35, 36, 16], 음성[17, 40], 자연어 처리[7, 38]의 정확도가 빠르게 향상되고 있는 AI 연구 역사상 전례 없는 시대를 맞이하고 있습니다. 딥 컨볼루션 신경망[23, 22]으로 학습된 시각적 표현은 Ima- geNet 분류[33]와 같이 이전에는 어려웠던 작업에서 탁월한 성능을 보여주며, 물체 감지 및 세그먼트령과 같은 어려운 인식 문제에도 적용될 수 있습니다.

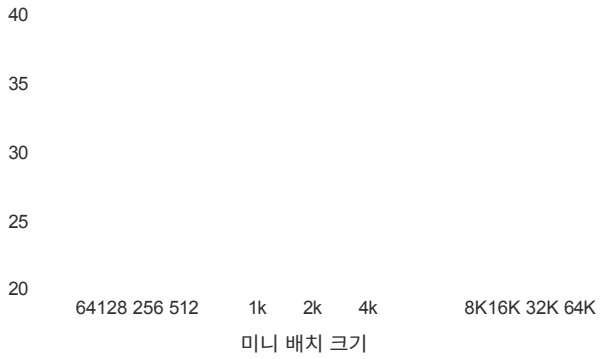
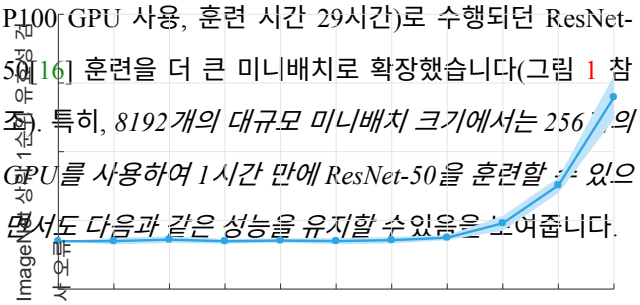


그림 1. **ImageNet 상위 1순위 유효성 검사 오류와 미니배치 크기 *비교*** 표준편차 플러스/마이너스 *2의* 오류 범위가 표시됩니다. *소규모 미니배치 학습의 상위 1% 오류를 유지하면서* 분산 동기식 SGD를 최대 8k 이미지의 미니배치로 확장하는 간단하고 일반적인 기법을 소개합니다. 모든 미니배치 크기에 대해 학습 속도를 미니배치 크기의 *선형* 함수로 설정하고 훈련의 처음 몇 회기에 대해 간단한 워밍업 단계를 적용합니다. 다른 모든 하이퍼파라미터는 고정된 상태로 유지됩니다. 이 간단한 접근 방식을 사용하면 모델의 정확도가 미니배치 크기(최대 8k 미니배치 크기)에 관계없이 변하지 않습니다. 이 기술을 사용하면 대규모 미니배치 크기로 확장할 때 90%의 효율로 훈련 시간을 선형적으로 단축할 수 있으므로 256개의 GPU에서 1시간 만에 정확한 8k 미니배치 ResNet-50 모델을 훈련할 수 있습니다.

tation [8, 10, 28]. 또한 이러한 패턴은 일반화되어 데이터 세트와 신경망 아키텍처가 클수록 사전 학습의 혜택을 받는 모든 작업에서 일관되게 정확도가 향상됩니다 [22, 41, 34, 35, 36, 16]. 그러나 모델과 데이터 규모가 커질수록 학습 시간도 늘어나기 때문에 대규모 딥러닝의 잠재력과 한계를 발견하려면 학습 시간을 관리할 수 있는 새로운 기술을 개발해야 합니다.

이 보고서의 목표는 분산 *동기식* 확률적 경사 하강(SGD)을 사용한 대규모 훈련의 가능성을 입증하고 이에 대한 실용적인 가이드를 전달하는 것입니다. 예를 들어, 원래 256개 이미지의 미니배치 크기(8개의 Tesla P100 GPU 사용, 훈련 시간 29시간)로 수행되던 ResNet-50[16] 훈련을 더 큰 미니배치로 확장했습니다(그림 1 참조). 특히, *8192개의 대규모 미니배치 크기에서는 256개의 GPU를 사용하여 1시간 만에 ResNet-50을 훈련할 수 있으면서도 다음과 같은 성능을 유지할 수 있음을 보여줍니다.*



256 미니배치 기준과 동일한 수준의 정확도를 제공합니다. 분산 동기식 SGD는 이제 보편화되었지만, 8192개에 달하는 미니배치로 일반화 정확도를 유지할 수 있거나 이렇게 짧은 시간에 높은 정확도의 모델을 훈련할 수 있다는 기존 결과는 없습니다.

이러한 비정상적으로 큰 미니배치 크기를 처리하기 위해, 저희는 학습률만을 광고하기 위해 간단하고 매개변수가 없는 *선형 스케일링 규칙을* 사용합니다. 이 규칙은 선행 연구[21, 4]에서 발견되었지만, 그 경험적 한계가 잘 알려져 있지 않으며 비공식적으로 연구 커뮤니티에 널리 알려지지 않았습니다. 이 규칙을 성공적으로 적용하기 위해 우리는 새로운 *워밍업* 전략, *즉* 훈련 시작 시 낮은 학습률을 사용하는 전략[16]을 제시하여 초기 최적화의 어려움을 극복합니다. 중요한 것은 이 접근 방식이 기준 *검증* 오류와 일치할 뿐만 아니라 *다음과 거의 일치하는 훈련 오류 곡선을 생성한다는 점입니다.*

소규모 미니배치 기준선. 자세한 내용은 §2에 나와 있습니다.

5의 종합적인 실험을 통해 대규모 미니배치에서는 옵티마이제이션 난이도가 주요 문제임을 알 수 있습니다, *일반화가 잘 되지 않는다는* 최근의 일부 연구[20]와는 대조적으로 (적어도 ImageNet에서는) 오히려 *일반화가 잘 됩니다.* 또한 선형 스케일링 규칙과 워밍업이 객체 감지 및 인스턴스 분할을 포함한 더 복잡한 작업[9, 31, 14, 28]에 일반화됨을 보여주며, 이는 최근에 개발된 Mask R-CNN[14]을 통해 입증되었습니다. 광범위한 미니배치 크기를 처리하기 위한 근본적이고 성공적인 가이드라인은 이전 연구에서 제시되지 않았습니다. 우리가 제공하는 전략은 간단하지만, 이를 성공적으로 적용하려면 딥 러닝 라이브러리 내에서 사소해 보이고 종종 잘 이해되지 않는 구현 세부 사항에 대한 올바른 구현이 필요합니다. SGD 구현의 미묘한 차이로 인해 잘못된 솔루션을 발견하기 어려울 수 있습니다. 보다 유용한 지침을 제공하기 위해 일반적인 함정과 관련 구현 방법을 설명합니다. 이러한 함정을 유발할 수 있는 세부 정보는 §3에서 확인할 수 있습니다.

2. 대형 미니배치 SGD

먼저 다음 섹션에서 논의의 기초가 될 확률적 그래디언트 하강(SGD)의 공식을 검토합니다. 이 공식의 손실 $L(w)$를 최소화하는 방식으로 슈퍼비전 학습을 고려합니다:

$$L(w) = \frac{1}{|X|} \sum_{x \in X} l(x, w). \quad (1)$$

여기서 w는 네트워크의 가중치, X는 레이블이 지정된 훈련 세트, $l(x, w)$는 샘플 $x \in X$와 해당 레이블 y에서 계산된 손실입니다. 일반적으로 l은 분류 손실(*예:* 교차 엔트로피)과 w에 대한 정규화 손실의 합입니다.

미니배치 확률론적 경사 하강[32]은 미니배치에서 작동하지만 최근 문헌에서는 일반적으로 SGD로 간단히 다시 표현되며, 다음과 같은 업데이트를 수행합니다:

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in B} \nabla l(x, w_t). \quad (2)$$

이러한 전략은 프레임워크에 관계없이 적용되지만, 효율적인 선형 확장을 달성하려면 특별한 통신 알고리즘이 필요합니다. 우리는 오픈 소스인 *Caffe2*[1] 딥 러닝 프레임워크와 표준 이더넷 네트워킹(특수 네트워크 인터페이스가 아닌)을 사용하여 효율적으로 작동하는 *Big Basin* GPU 서버[24]를 사용합니다. 이 접근 방식이 최대한의 잠재력을 발휘할 수 있도록 하는 시스템 알고리즘은 §4에서 자세히 설명합니다.

이 보고서에서 설명하는 실질적인 발전은 다양한 영역에 걸쳐 도움이 됩니다. 산업 분야에서 유니티 시스템은 인터넷 규모의 데이터에서 시각 모델을 훈련할 수 있는 잠재력을 발휘하

여 *하루에 수십억 개의 이미지로* 훈련할 수 있습니다. 또한 연구 분야에서는 하이퍼파라미터 검색 없이도 단일 GPU에서 멀티 GPU 구현으로 알고리즘을 마이그레이션하는 것을 간소화할 수 있다는 사실을 발견했습니다(*예:* Faster R-CNN [31] 및 ResNets [16]을 1에서 8개의 GPU로 마이그레이션한 경험).

[1] http://www.caffe2.ai

다음은 $\mathcal{B}$ X*에서* 샘플링된 미니배치이고 n =는 $|\mathcal{B}|$ *미니배치 크기*, η는 *학습 속도*, t는 반복 인덱스입니다. 실제로는 모멘텀 SGD를 사용하며, 모멘텀에 대한 논의는 §3에서 다시 다루겠습니다.

2.1. 대규모 미니배치를 위한 학습 속도

우리의 목표는 *훈련과 일반화의 정확도를 유지하면서* 소규모 미니배치 대신 대규모 미니배치를 사용하는 것입니다. 이는 여러 작업자로 확장할 수 있기 때문에 분산 학습에서 특히 흥미롭습니다.[2] 작업자당 워크로드를 줄이면서 모델 정확도를 떨어뜨리지 않고도 간단한 데이터 병렬 처리를 사용할 수 있기 때문입니다.

종합적인 실험에서 보여드리겠지만, 다음과 같은 학습률 스케일링 규칙이 광범위한 미니배치 규모에 놀라울 정도로 효과적이라는 사실을 발견했습니다:

> ***선형 스케일링 규칙:*** *미니배치 크기에* k*를 곱하면 학습률에* k*를 곱합니다.*

다른 모든 하이퍼파라미터(가중치 감쇠 *등*)는 변경되지 않은 상태로 유지됩니다. 5장에서 살펴보겠지만, *선형 스케일링 규칙은* 작은 미니배치와 큰 미니배치 간의 정확도를 일치시킬 뿐만 아니라, 마찬가지로 중요한 것은 훈련 곡선을 크게 일치시켜 수렴 전에 실험을 빠르게 디버깅하고 비교할 수 있게 해줍니다.

해석. 선형 스케일링 규칙에 대한 비공식적인 논의와 이 규칙이 효과적인 이유를 소개합니다. 가중치 w_t 와 각각 크기 n*의* $\mathcal{B}$ $0 \; j < k$*에* 대한 k ≤*개의* 미니배치$_j$ 의 시퀀스가 있는 반복 t*의* 네트워크를 고려해 보겠습니다. *작은 미니배치* $\mathcal{B}_j$ 와 학습률 η*를 가진* k *개의* SGD 반복을 실행하는 효과와 크기 kn*의 큰 미니배치* $\cup \mathcal{B}_{jj}$ 와 학습률 $\eta\hat{}$*를 가진* 단일 반복을 실행하는 효과를 비교합니다.

2이 작업에서는 '워커'와 'GPU'라는 용어를 혼용하여 사용하지만, '워커'를 다른 방식으로 구현할 수도 있습니다. '서버'는 네트워크를 통한 통신

이 필요 없는 8개의 GPU 세트를 의미합니다.

(2)에 따르면, 학습률 η와 미니배치 크기 n으로 SGD를 k번 반복하면 다음과 같은 결과를 얻을 수 있습니다:

$$w_{t+k} = w_t - \eta \frac{1}{n} \sum_{j<k} \sum_{x \in B_j} \nabla l(x, w_{t+j}). \quad (3)$$

반면에 크기 kn, 학습률 $\hat{\eta}$의 대형 미니 배치 $\cup B_j$로 한 단계를 수행하면 산출됩니다:

$$\hat{w}_{t+1} = w_t - \hat{\eta} \frac{1}{kn} \sum_{j<k} \sum_{x \in B_j} \nabla l(x, w_t). \quad (4)$$

예상대로 업데이트가 다르며, 다음과 같은 문제가 발생할 *가능성은 낮습니다*.

$\hat{w}_{t+1} = w_{t+k}$. 그러나 $j < k$에 대해 $l(x, w_t) \approx l(x, w_{t+j})$을 가정할 수 있다면 $\hat{\eta} = k\eta$로 설정하면 다음과 같은 결과가 나옵니다.

$\hat{w}_{t+1} \approx w_{t+k}$, 소규모 및 대규모 미니 배치 SGD의 업데이트는 비슷할 것입니다. 이것은 강력한 가정이지만, 만약 이것이 사실이라면 두 업데이트는 $\hat{\eta} = k\eta$로 설정한 경우에만 유사하다는 점을 강조합니다.

위의 해석은 선형 스케일링 규칙이 적용되기를 기대할 수 있는 한 가지 경우에 대한 직관을 제공합니다. $\hat{\eta} = k\eta$(및 워밍업)를 사용한 실험에서 소규모 및 대규모 미니배치 SGD는 동일한 실제 정확도를 가진 모델을 생성할 뿐만 아니라 훈련 곡선도 거의 일치합니다. 우리의 경험적 결과는 위의 근사치가 대규모의 실제 데이터에서도 유효할 수 있음을 시사합니다.

그러나 $\nabla l(x, w_t) \approx \nabla l(x, w_{t+j})$ 조건이 충족되지 않는 경우가 적어도 두 가지 있습니다. 첫째, 네트워크가 빠르게 변화하는 초기 훈련에서는 이 조건이 유지되지 않습니다. 우리는 2.2절에서 설명한 워밍업 단계를 사용하여 이 문제를 해결합니다. 둘째, 미니배치 크기는 확실히 확장할 수 없습니다. 넓은 범위의 크기에서는 결과가 안정적이지만, 특정 지점을 넘어서면 정확

2.2. 워밍업

앞서 설명했듯이 대형 미니배치(*예:* 8k)의 경우, 린

이어 스케일링 규칙은 네트워크가 빠르게 변화할 때 무너지는데, 이는 일반적으로 기차의 초기 단계에서 발생합니다.

ing. 이 문제는 적절하게 설계된 *워밍업*[16], 즉 훈련을 시작할 때 덜 공격적인 학습 속도를 사용하는 전략으로 완화할 수 있다는 것을 발견했습니다.

지속적인 워밍업. 16]에 제시된 워밍업 전략은 다음과 같습니다.

의 처음 몇 회기 동안 낮은 *일정한* 학습 속도를 사용합니다.

도가 급격히 저하됩니다. 흥미롭게도 이 지점은 ImageNet 실

험에서 ~8k만큼 큽니다.

토론. 위의 선형 스케일링 규칙은 Krizhevsky[21]가 먼저 채택한 것입니다. 그러나 Krizhevsky는 미니배치 크기를 128에서 1024로 늘렸을 때 1%의 오류 증가를 다시 포팅한 반면, 우리는 훨씬 더 광범위한 미니배치 크기 영역에서 정확도를 유지하는 방법을 보여줍니다. Chen *등*[5]은 다양한 SGD 변형에 대한 비교를 제시했으며, 그들의 작업에서도 선형 스케일링 규칙을 사용했지만 작은 미니배치 기준선을 설정하지는 않았습니다. Li [25](§4.6)는 수렴 후 정확도 손실 없이 최대 5120개의 미니배치를 사용한 분산 Ima-Net 훈련을 보여주었습니다. 그러나 그들의 연구는 미니배치 크기의 함수로 학습 속도를 조정하기 위한 하이퍼파라미터 검색이 필요 없는 규칙을 입증하지 못했는데, 이는 본 연구의 핵심적인 기여입니다.

최근 연구에서 Bottou *등*[4](§4.2)은 미니배칭의 이론적 트레이드 오프를 검토하고 선형 스케일링 규칙을 사용하면 솔버가 보이는 예제 수의 함수와 동일한 훈련 곡선을 따르고 학습 속도를 제안합니다.
는 최소 요금과 무관하게 최대 요금을 초과해서는 안됩니다.

훈련. 5장에서 설명하겠지만, 지속적인 워밍업은 새로 초기화된 레이어와 함께 사전 학습된 레이어를 미세 조정하는 객체 감지 및 분할 방법[9, 31, 26, 14]의 프로토타입 제작에 특히 유용하다는 것을 알게 되었습니다.

kn 크기의 대규모 미니배치를 사용한 이미지넷 실험에서는 처음 5회 동안 낮은 학습률 η로 훈련한 다음 목표 학습률인 $\hat{\eta} = k\eta$로 복귀하려고 시도했습니다. 그러나 *k*가 크면 이러한 일정한 워밍업으로는 최적화 문제를 해결하기에 충분하지 않으며, 낮은 학습률의 워밍업 단계에서 벗어나면 학습 오류가 급증할 수 있다는 사실을 발견했습니다. 따라서 다음과 같은 점진적 워밍업을 제안합니다.

점진적 워밍업. 학습 속도를 작은 값에서 큰 값으로 *점진적으로* 높이는 대체 워밍업을 제시합니다. 이 램프는 학습률이 갑자기 증가하는 것을 방지하여 훈련 시작 시 건강한 컨버전스를 가능하게 합니다. 실제로 *kn* 크기의 대규모 미니배치에서는 학습률 η에서 시작하여 각 반복마다 학습률을 일정량씩 증가시켜 5회 반복 후에 $\hat{\eta} = k\eta$에 도달합니다(재설정은 정확한 워밍업 기간에 대해 견고합니다). 워밍업이 끝나면 원래 학습 속도 스케줄로 돌아갑니다.

배치 크기(워밍업을 정당화). 저희는 전례 없는 미니배치 크기로 이러한 이론을 실증적으로 테스트합니다.

2.3. 대규모 미니배치를 사용한 배치 정규화

배치 정규화(BN)[19]는 미니배치 차원을 따라 통계를 계산하는데, 이는 각 샘플 손실의 독립성을 깨뜨리고 미니배치 크기를 변경하면 최적화되는 손실 함수의 기본 정의가 변경됩니다. 다음에서는 통신 오버헤드를 피하기 위해 실용적으로 고려하는 것처럼 보일 수 있는 일반적으로 사용되는 '지름길'이 실제로는 미니배치 크기를 변경할 때 손실 함수를 보존하는 데 필요하다는 것을 보여드리겠습니다. (1)과 (2)는 샘플당 손실 $l(x, w)$이 다른 모든 샘플과 독립적이라고 가정한다는 점에 유의하세요. BN이 수행되고 샘플 전체에 걸쳐 활성화가 계산되는 경우에는 *그렇지 않습니다*. 단일 샘플 *x*의 손실이 해당 미니배치 B에 있는 모든 샘플의 통계에 따라 달라진다는 것을 나타내기 위해 $l_{\mathcal{B}}(x, w)$를 작성합니다.

크기 *n*의 B은 $L(\mathcal{B}, w) = \frac{1}{|\mathcal{B}|}\sum_{x\in\mathcal{B}} l_{\mathcal{B}}(x, w)$로 표현됩니다. BN을 사용하면

트레이닝 세트는 모든 고유한 하위 n

원래 훈련 집합 *X*에서 추출한 크기 *n*의 집합을 X^n 로 표시합니다. 그러면 훈련 손실 $L(w)$는 다음과 같이 됩니다:

$$L(w) = \frac{1}{|X^n|}\sum_{\mathcal{B}\in X^n} L(\mathcal{B}, w). \qquad (5)$$

X에서 '단일 B 샘플'로 보는 경우^n , 각 단일 샘플의 손 실 은 *독립적으로* 계산됩니다.

BN 통계가 계산되는 미니배치 크기 n은 손실의 핵심 구성 요소입니다. 작업자당 미니배치 샘플 크기 n이 변경되면 *최적화되는 기본 손실 함수 L이 변경됩니다*. 좀 더 구체적으로 설명하면, n이 다른 BN에 의해 계산된 평균/분산 통계는 서로 다른 수준의 무작위 변동을 나타냅니다.

분산(및 멀티 GPU) 훈련의 경우 작업자당 샘플 크기 n이 고정되어 있고 총 미니 배치 크기가 kn인 경우, 각 샘플 B_j 가 X^n 에서 독립적으로 선택된 k 샘플의 미니 배치로 볼 수 있으므로 다음과 같습니다.

기본 손실 함수는 변경되지 않고 여전히 제거됩니다.

X에서 벌금^n . 이 관점에서, BN 설정에서 k 미니배치 B_j 를 본 후 (3) 및 (4)가 됩니다:

$$w_{t+k} = w_t - \eta \sum_{j<k} \nabla L(B_j, w_{t+j}), \qquad (6)$$

$$w_{t+1} = w_t - \hat{\eta}\frac{1}{k}\sum_{j<k} \nabla L(B_j, w_t). \qquad (7)$$

2.1에서와 유사한 논리에 따라 $\hat{\eta} = k\eta$로 설정하고 작업자 *수 k를 변경할 때 작업자당 표본 크기 n을 일정하게* 유지합니다.

이 작업에서는 광범위한 데이터 세트와 네트워크에서 좋은 성능을 보인 $n = 32$를 사용합니다[19, 16]. n을 조정하면 BN의 하이퍼 파라미터로 간주해야 합니다, 분산 고유의 아닙니다. 또한 *BN 통계*에 따르면 *모든 작업자에* 대해 *계산해서는* 안됩니다.

통신을 줄이기 위한 목적뿐만 아니라 동일한 기본 손실 기능을 최적화하여 유지하기 위한 목적도 있습니다.

3. 분산형 SGD의 미묘한 차이와 함정

실제로 분산 구현에는 많은 세부 사항이 있습니다. 많은 일반적인 구현 오류로 인해 하이퍼파라미터의 정의가 변경되어 모델이 학습되지만 오류가 예상보다 클 수 있으며, 이러한 오류는 발견하기 어려울 수 있습니다. 아래의 설명은 간단하지만 기본 솔버를 충실히 구현하기 위해 설명적으로 고려하는 것이 중요합니다.

가중치 감쇠. 가중치 감쇠는 실제로 *손실 함수에서* L2 정규화 항의 기울기의 결과입니다. 보다 공식적으로, (1)의 샘플당 손실 $\varepsilon(x, w_t)$에 의해 기여합니다. 가중치 감쇠 항이 없는 경우, $\varepsilon(x, w_t)$를 스케일링하는 것을 포함하여 학습률을 스케일링하는 여러 가지 동등한 방법이 있습니다. 그러나 식 (8)에서 볼 수 있듯이 일반적으로는 그렇지 *않습니다*. 다음 설명에서 이러한 관찰 결과를 요약합니다:

비고 1: 교차 엔트로피 손실 스케일링은 다음과 같습니다.

학습 속도를 확장하는 것과 동일하지 않습니다.

모멘텀 보정. 모멘텀 SGD는 (2)의 바닐라 SGD에 일반적으로 채택된 수정입니다. 모멘텀 SGD의 참조 구현은 다음과 같은 형태입니다:

$$u_{t+1} = mu_t + \frac{1}{n}\sum_{x\in B} \nabla(x, w_t)$$
$$w_{t+1} = w_t - \eta u_{t+1}. \qquad (9)$$

여기서 m은 운동량 감쇠 계수이고 u는 업데이트 텐서입니다. 널리 사용되는 변형은 학습률 η를 업데이트 텐서에 흡수하는 것입니다. (9)에서 ηu_t 에 v_t 를 대입하면 다음과 같은 결과가 나옵니다:

$$v_{t+1} = mv_t + \eta\frac{1}{n}\sum_{x\in B} \nabla(x, w_t)$$
$$w_{t+1} = w_t - v_{t+1}. \qquad (10)$$

고정된 η의 경우 이 둘은 동일합니다. 그러나 u는 기울기에만 의존하고 η와는 독립적인 반면, v는 η와 얽혀 있다는 점에 유의합니다. η가 변경되면 (9)의 참조 변형과 동등성을 유지하기 위해 다음에 대한 업데이트가 수행됩니다.

v는 다음과 같아야 합니다: $v_{t+1} = m\frac{\eta_{t+1}}{\eta_t}v_t + \eta_{t+1}\frac{1}{n}\sum \nabla(x, w_t)$. We $\frac{\eta_{t+1}}{\eta_t}$ 인자를 참조하십시오. 를 *모멘텀 보정으로* 사용합니다. 우리

실은 $l(x, w) = \frac{1}{2}\lambda\|w\|^2 + \varepsilon(x, w)$로 작성할 수 있습니다. 여기서 $\frac{1}{2}\lambda\|w\|^2$ 은 가중치에 대한 샘플 독립적인 L2 정규화이고 $\varepsilon(x, w)$는 교차 엔트로피 손실과 같은 샘플 의존적인 용어입니다. 식 (2)의 SGD 업데이트는 다음과 같이 작성할 수 있습니다:

$$w = w - \eta\lambda w - \eta \frac{1}{n}\sum_{x\in B} \nabla\varepsilon(x, w). \quad (8)$$

$$w_{t+1} = w_t - \eta_t\lambda w_t - \eta_t \frac{1}{n}\sum_{x\in B} \nabla\varepsilon(x, w_t)$$

실제로는 일반적으로 샘플 종속 용어인 $\sum_{x\in B} \nabla\varepsilon(x, w_t)$는 배경에 의해 계산되며, 용어 λw_t 는 다음과 같습니다.

개별적으로 계산되어 집계된 그라데이션에 추가됩니다.

는 η_{t+1} η_t , 그렇지 않으면 히스토리 용어 v_t 가 너무 작아 불안정성을 초래하는 경우($\eta_{t+1} < \eta_t$ 모멘텀 보정은 덜 중요함)에 특히 트레이닝 안정화에 중요하다는 것을 발견했습니다. 이것이 두 번째 설명으로 이어집니다:

참고 2: (10)을 사용하는 경우 학습률을 변경한 후 운동량 보정을 적용합니다.

그라데이션 집계. 작업자당 미니배치 크기가 n*인 작업자가* 각각 k*명인* 경우, (4)에 따라 전체 예제 집합 kn*에* 대해 그라데이션 집계가 수행되어야 합니다.

에 따르면 $\frac{1}{kn}\sum_{x\in B} \nabla l(x, w_t)$. 손실 계층은 일반적으로 *로칼* 입력에 대한 평균 손실을 계산하기 위해 구현되었으며, 이는 작업자 당 손실을 계산하는 것과 같습니다. $\frac{1}{n}\sum_{x\in B} l(x, w_t)$. 이를 감안할 때 올바른 집계를 위해서는 *av-누락된 1/k* 인자를 복구하기 위해 k 그라데이션을 *에징합니다.* 그러나 allreduce[11]와 같은 표준 통신 프리미티브는 평균이 아닌 합계를 수행합니다. 따라서 *1/k* 스케일링을 손실에 흡수하는 것이 더 효율적이며, 이 경우 입력에 대한 손실의 기울기만 스케일링하면 되므로 전체 기울기 벡터의 스케일링이 필요하지 않습니다. 이를 요약하면 다음과 같습니다:

비교 3: 작업자당 손실을 다음과 같이 정규화합니다. 작업자당 크기 N이 아닌 총 미니배치 크기 KN입니다.

또한 $\hat{\eta} = \eta$(kη가 아닌)로 설정하고 손실을 *1/n*(*1/kn*이 아닌)으로 정규화하여 'k를 취소'하는 것은 잘못된 무게 감쇠로 이어질 수 있습니다(*비고 1* 참조).

데이터 셔플링. SGD는 일반적으로 데이터를 무작위로 *샘플링하여 교체하는* 과정으로 분석됩니다. 실제로, 일반적인 SGD 구현은 각 SGD 에포크 동안 훈련 세트의 *무작위 셔플링을* 적용하여 더 나은 결과를 얻을 수 있습니다 [3, 13]. 셔플링을 사용하는 기준선(*예:* [16])과 공정한 비교를 제공하기 위해, k*명의* 워커가 수행한 한 에포크의 샘플이 훈련 세트의 일관된 단일 런덤 셔플링에서 나온 것임을 보장합니다. 이를 위해 각 에포크에 대해 k*개의* 부분으로 분할된 무작위 셔플링을 사용하며, 각 부분은 k*명의* 워커 중 한 명이 처리합니다. 여러 워커에서 랜덤 셔플링을 올바르게 구현하지 못하면 눈에 띄게 다른 동작이 발생하여 결과와 결론이 오염될 수 있습니다. 요약하면 다음과 같습니다:

참고 4: 훈련 데이터의 단일 무작위 셔플(에포크당)을 사용하여 모든 k 작업자에게 나눕니다.

4. 커뮤니케이션

단일 빅베이슨 서버[24]의 GPU 8개 이상으로 확장하려면 네트워크의 여러 서버에 걸쳐 그라데이션 집계를 수행해야 합니다. 거의 완벽한 선형 확장이 가능하려면 백그라운드와 *병렬로* 집계가 수행되어야 합니다. 이는 레이어 간 그라데이션 간에 데이터 의존성이 없기 때문에 가능합니다. 따라서 한 레이어의 그라데이션이 계산되는 즉시 여러 레이어에 걸쳐 집계되며, 다음 레이어의 그라데이션 계산은 계속됩니다([5]에서 설명한 대로). 자세한 내용은 다음에 설명하겠습니다.

4.1. 그라데이션 집계

모든 그레이디언트에 대해 집계는 *올리듀스* 연산(MPI 집합_연산 *MPI 올리듀스*[11]와 유사)을 사용하여 수행됩니다. 올리듀스가 시작되기 전에 모든 GPU에는 로컬로 계산된 그라데이션이 있고, 올리듀스가 완료된 후 모든 GPU에는 모든 k 그라데이션의 합이 있습니다. 파라미터의 수가 증가하고 GPU의 계산 성능이 향상됨에 따라 백그라운드 단계에서 집계 비용을 숨기기가 더 어려워집니다. 이러한 효과를 극복하기 위한 훈련 기법은 본 연구의 범위를 벗어납니다(*예:* 양자화된 그라데이션 [18], 블록-모멘텀 SGD [6]). 그러나 이 작업의 규모에서는 최적화된 올리듀스 구현을 사용하여 거의 선형에 가까운 SGD 확장을 달성할 수 있었기 때문에 집단적 통신이 병목 현 상 이 되지 않았습니다.

서버 내 및 서버 간 통신을 위한 올리듀스 구현은 (1) 서버 내 8개 GPU의 버퍼를 각 서버의 싱글레 버퍼로 합산하고, (2) 결과 버퍼를 모든 서버에서 공유하고 합산하며, 마지막으로 (3) 결과를 각 GPU로 브로드캐스트하는 세 단계로 구성됩니다. (1)과 (3)의 로컬 축소 및 브로드캐스트에는 256KB 또는 그 이상의 버퍼에 대해 NCCL(NVIDIA Collective Communication Library)을 사용했습니다.[3] 을 사용했으며, 256KB 이상의 버퍼를 위해 다음과 같이 구성된 간단한 구현을 사용했습니다.

[3] https://developer.nvidia.com/nccl

GPU-호스트 메모리 복사본 수를 늘리고 그렇지 않으면 CPU를 줄입니다. NCCL은 GPU 커널을 사용하여 트레이서버 내 집합을 가속화하므로 이 접근 방식은 GPU에서 백그라운드에 더 많은 시간을 할애하는 동시에 유휴 상태였던 CPU 리소스를 사용하여 처리량을 개선합니다.

서버 간 올리듀스의 경우 대역폭이 제한된 시나리오에 가장 적합한 두 가지 알고리즘, 즉 필기체 *반감 및 배가 알고리즘*[30, 37]과 *버킷 알고리즘*(링 알고리즘이라고도 함)[2]을 구현했습니다. 두 알고리즘 모두 각 서버는 2^{p-1} 바이트의 데이터를 주고받으며, 여기서 b는 버퍼 크기(바이트)이고 p는 서버 수를 나타냅니다. 반감기/배수 알고리즘은 2로그$_2$ (p) 통신 단계로 구성되는 반면, 링 알고리즘은 2$(p$ 1) 단계로 구성됩니다. 따라서 일반적으로 지연 시간이 제한된 시나리오(*즉,* 버퍼 크기가 작거나 서버 수가 많은 경우)에서는 반감기/배수 알고리즘이 더 빠릅니다. 실제로 반감기/배수 알고리즘은 버퍼 크기가 최대 100만 개(서버 수가 많을 경우 그 이상)인 경우 링 알고리즘보다 훨씬 더 나은 것으로 나타났습니다. 32개 서버(256개 GPU)에서 반감기/배수 증가를 사용하면 링 알고리즘에 비해 3배의 속도 향상을 가져왔습니다.

반감기/배수 알고리즘은 분산 감소 집합과 올게더로 구성됩니다. 분산 감소의 첫 번째 단계에서 서버는 쌍으로(0번과 1번, 2번과 3번 *등)* 통신하여 입력 버퍼의 서로 다른 절반에 대해 송수신합니다. 예를 들어, 랭크 0은 버퍼의 후반부를 1로 보내고 버퍼의 전반부를 1로부터 받습니다. 수신된 데이터에 대한 감소가 다음 단계로 진행되기 전에 수행되며, 이때 대상 랭크까지의 거리는 두 배가 되고 송수신되는 데이터는 반으로 줄어듭니다. 감소-산란 단계가 완료되면 각 서버는 최종 감소된 벡터의 일부를 갖게 됩니다.

그 다음에는 리듀스-스캐터의 통신 패턴을 역추적하는 올개더 단계가 이어지며, 이번에는 최종적으로 축소된 벡터의 일부를 단순히 연결합니다. 각 서버에서는 리듀스-스캐터에서 전송되던 버퍼의 일부가 올게더에서 수신되고, 이제 수신되던 버퍼의 일부가 전송됩니다. 2의 거듭제곱이 아닌 서버 수를 지원하기 위해 *바이너리 블록 알고리즘을* 사용했습니다[30]. 이는 서버를 2의 거듭제곱 블록으로 분할하고 블록 내 감소-산란 직후와 블록 내 올개더링 전에 두 개의 추가 통신 단계를 사용하는 반감기/배수 알고리즘의 일반화된 버전입니다. 2승이 아닌 경우에는 2승에 비해 p 어느 정도 부하 불균형이 발생하지만, 실행 결과 다음과 같이 나타났습니다. 성능 저하가 크게 나타나지 않았습니다.

4.2. 소프트웨어

설명된 모든 축소 알고리즘은 다음에서 구현됩니다. *Gloo*[4]는 집단 커뮤니케이션을 위한 라이브러리입니다. 다음을 지원합니다.

[4] https://github.com/facebookincubator/gloo

여러 통신 컨텍스트를 사용할 수 있으므로 여러 감축 인스턴스를 병렬로 실행하는 데 추가 동기화가 필요하지 않습니다. 로컬 축소 및 브로드캐스트(단계 (1) 및 (3)으로 설명됨)는 가능한 경우 서버 간 축소와 함께 파이프라인으로 연결됩니다.

Caffe2는 훈련 반복을 나타내는 컴퓨팅 그래프의 멀티 스레드 실행을 지원합니다. 하위 그래프 간에 데이터 종속성이 없는 경우 여러 스레드가 해당 하위 그래프를 병렬로 실행할 수 있습니다. 이를 백그라운드에 적용하면 올리듀스 또는 가중치 업데이트를 처리하지 않고도 로컬 그래디언트를 순차적으로 계산할 수 있습니다. 즉, 백그라운드에서 *실행 가능한 서브그래프* 세트가 실행할 수 있는 속도보다 빠르게 증가할 수 있습니다. 모든 감소 실행이 포함된 서브그래프의 경우, 모든 서버는 실행 가능한 서브그래프 집합에서 동일한 서브그래프를 실행하도록 선택해야 합니다. 그렇지 않으면 서버가 교차하지 않는 서브그래프 집합을 실행하려고 시도하는 분산 교착 상태가 발생할 위험이 있습니다. 올리듀스는 *집단* 연산이므로 서버가 대기 시간 초과를 일으킬 수 있습니다. 올바른 실행을 보장하기 위해 이러한 서브그래프에 부분적인 순서를 부과합니다. 이는 주기적 제어 입력을 사용하여 구현되는데, n *번째* 올리듀스가 완료되면 $(n + c)$번째 올리듀스의 실행이 차단 해제되며, 여기서 c*는* 동시 올리듀스 실행의 최대 수입니다. 이 숫자는 전체 계산 그래프를 실행하는 데 사용되는 스레드 수보다 낮게 선택해야 합니다.

4.3. 하드웨어

실험에는 Facebook의 빅베이슨[24] GPU 서버를 사용했습니다. 각 서버에는 NVIDIA NVLink로 상호 연결된 8개의 NVIDIA Tesla P100 GPU가 포함되어 있습니다. 로컬 스토리지의 경우, 각 서버에는 3.2TB의 NVMe SSD가 있습니다. 네트워크 연결을 위해 서버에는 Mellanox ConnectX-4 50Gbit 이더넷 네트워크 카드가 있으며 Wedge100 [1] 이더넷 스위치에 연결됩니다.

다음 분석에 따르면 ResNet-50의 분산 동기식 SGD에 충분한 50Gbit의 네트워크 대역폭을 발견했습니다. ResNet-50에는 약 25백만 개의 파라미터가 있습니다. 즉, 파라미터의 총 크기는
$25\ 10^6$ sizeof(float) = 100MB. 단일 NVIDIA Tesla P100 GPU에서 ResNet-50의 백그라운드는 120ms가 소요됩니다. 올리듀스가 작동하는 값에 비해 네트워크에서 ~2바이트가 필요하다는 점을 감안하면 통신 오버헤드를 고려하지 않고 200MB/0.125초 = 1600MB/s 또는 12.8Gbit/s의 피크 대역폭 재사용으로 이어집니다. 네트워크 오버헤드에 대한 스머지 계수를 추가하면 ResNet-50의 피크 대역폭 요구 사항은 ~15Gbit/s에 도달합니다.

이 최대 대역폭 요구 사항은 백그라운드에서만 적용되므로, 네트워크는 포워드 패스 중에 데이터 읽기 또는 네트워크 스냅샷 저장과 *같이* 지연에 덜 민감한 다른 작업에 자유롭게 사용할 수 있습니다.

5. 주요 결과 및 분석

그 결과 1시간에 256명의 워커를 사용해 ImageNet[33]에서 ResNet-50[16]을 훈련하면서 소규모 미니배치 훈련의 정확도를 맞출 수 있게 되었습니다. 워밍업 전략과 함께 선형 스케일링 규칙을 적용하면 추가적인 하이퍼 파라미터를 조정하거나 정확도에 영향을 주지 않고도 소규모와 대규모 미니배치(최대 8k 이미지) 간에 원활하게 스케일링할 수 있습니다. 다음 하위 섹션에서는

(1) 실험 설정을 설명하고, (2) 대규모 미니배치 훈련의 효과를 입증하고, (3) 심층적인 실험 분석을 수행하고, (4) 연구 결과를 객체 감지/세분화에 일반화할 수 있음을 보여주고, (5) 타이밍을 제시합니다.

5.1. 실험적 설정

1000-way ImageNet 분류 작업[33]이 주요 실험 벤치마크 역할을 합니다. 약 128만 개의 훈련 이미지로 모델을 학습시키고 5만 개의 검증 이미지에서 상위 1%의 오차를 기준으로 평가합니다.

여기서는 [12]의 ResNet-50 [16] 변형을 사용하며, 보폭-2 컨볼루션이 3개 레이어가 아닌 3개 레이어에 있다는 점에 유의합니다.

1 [16]에서와 같이 1 레이어. 우리는 [12]에 따라 m*이* 0.9인 네스테로프 운동량[29]을 사용하지만 [16]에서 사용된 표준 운동량도 똑같이 효과적이라는 점에 유의합니다. 0.0001의 가중치 감쇠 λ*를* 사용하며 [16]에 따라 학습 가능한 BN 계수(즉, [19]의 γ 및 β*)*에 가중치 감쇠를 적용하지 않습니다. 2.3절에서 설명한 대로 BN 배치 크기 n*에* 따라 달라지는 훈련 목표를 고정하기 위해 전체 미니배치 크기에 관계없이 전체적으로 $n = 32$를 사용합니다. 12]에서와 마찬가지로, 운동량 0.9를 사용하여 실행 평균을 사용하여 BN 통계를 계산합니다.

모든 모델은 미니 배치 크기에 관계없이 90개의 에포크 동안 훈련됩니다. 2.1절의 *선형 스케일링* 규칙을 적용하고 미니 배치 크기 kn*에서* 선형인 $\eta = 0.1^{kn}$ 의 학습 속도를 사용합니다. $k = 8$개의 워커(GPU), 워커당 샘플 수 $n = 32$개인 경우 [16]에서와 같이 $\eta = 0.1$이 됩니다. 이 수치(0.1^{kn})를 *기준 학습률이라고* 부르고, [16]과 유사하게 30회, 60회, 80회에서 1/10로 줄입니다.

모든 컨볼루션 레이어에 대해 [15]의 초기화를 채택합니다. 1000방향 완전 연결 레이어는 표준 편차가 0.01인 제로-평균 가우시안에서 가중치를 끌어와 초기화합니다. 미니배치가 작은 SGD는 BN으로 인해 초기화에 민감하지 않지만, 미니배치가 상당히 큰 경우에는 그렇지 않다는 것을 발견했습니다. 또한 초기 훈련에서 최적화 문제를 피하기 위해 적절한 워밍업 전략이 필요합니다.

BN 레이어의 경우 학습 가능한 스케일링 계수 γ는 1로 *초기화되지만, 각 잔여 블록의 마지막* BN은 γ*가 0으로 초기화됩니다.* 각 잔여 블록의 마지막 BN에서 $\gamma = 0$으로 설정하면 순방향/역방향 신호가 ResNet의 ID 지름길을 통해 전파되므로 학습 시작 시 최적화를 쉽게 할 수 있습니다. 이 초기화는 모든 모델을 개선하지만 특히 대규모 미니배치 훈련에 유용합니다.

×

×

- 256

- 256

12]에서와 같이 스케일 및 종횡비 데이터 증강[36]을 사용합니다. 네트워크 입력 이미지는 증강 이미지의 224픽셀 랜덤 크롭 또는 수평 플립입니다. 입력 이미지는 [12]에서와 같이 색상별 평균과 표준 편차로 정규화됩니다.

무작위 변동 처리하기. 모델 훈련 시 무작위 변이가 발생할 수 있으므로 모델의 오차율은 최종 5회차의 오차 *중앙값으로* 계산합니다. 또한 *5개의 독립적인 실행에서 얻은* 오차의 평균과 표준편차(std)를 보고합니다. 이를 통해 결과에 대한 신뢰도를 높이고 모델 안정성을 측정할 수 있습니다.

이미지넷 모델의 무작위 변이는 일반적으로 이전 연구에서 보고되지 않았습니다(주로 리소스 제한으로 인해). 무작위 변이를 무시하면 특히 단일 실험의 결과 또는 여러 실험 중 가장 좋은 결과일 경우 신뢰할 수 없는 결론을 내릴 수 있다는 점을 강조합니다.

기준선. 이러한 설정에서 [16]에서와 같이 $k = 8$(한 서버에 8개의 GPU), 작업자당 $n = 32$개의 이미지(미니배치 크기 $kn = 256$)를 사용하여 ResNet-50 기준선을 설정합니다. 우리의 기준선은 23.60% 0.12의 상위 1% 검증 오류를 가지고 있습니다. 참고로 `fb.resnet.torch` [12]의 ResNet-50은 24.01%, 원본 ResNet 논문 [16]의 오류는 더 약한 데이터 증강에서 24.7%입니다.

5.2. 최적화 또는 일반화 문제?

최적화 및 일반화 동작을 탐색하여 대규모 미니배치 훈련에 대한 주요 결과를 확립합니다. 적절한 워밍업 전략을 사용하면 대규모 미니배치 SGD가 소규모 미니배치 SGD의 *훈련 곡선과* 일치할 뿐만 아니라 *검증* 오류도 일치할 수 있음을 입증할 것입니다. 즉, 실험에서 대규모 미니배치 훈련의 최적화와 *일반화는* 모두 소규모 미니배치 훈련과 일치합니다. 또한, 5.4절에서는 이러한 모델이 물체 감지/세분화 전송 작업에 대한 일반화 동작이 우수하여 소규모 미니배치 모델의 전송 품질과 일치한다는 것을 보여줄 것입니다.

다음 결과에서는 $k = 256$, $n = 32$를 사용하므로 미니 배치 크기는 $kn = 8$k가 됩니다(1024를 나타내기 위해 '1k'를 사용함). 앞서 설명한 대로 기준선의 미니 배치 크기는 $kn = 256$이고 기준 학습률은 $\eta = 0.1$입니다. 선형 스케일링 규칙을 적용하면 대규모 미니 배치 실행에 대한 기준 학습률은 $\eta = 3.2$가 됩니다. 2.2절에서 설명한 대로 워밍업 *없음*, 5 회에 걸쳐 $\eta = 0.1$로 *일정한 워밍업*, $\eta = 0.1$로 시작하여 5회에 걸쳐 $\eta = 3.2$로 *선형적으로* 증가하는 *점진적 워밍업* 등 세 가지 워밍업 전략을 테스트합니다. 모든 모델은 처음부터 훈련되며 다른 모든 하이퍼파라미터는 고정된 상태로 유지됩니다. 특정 미니배치 크기에 대해 하이퍼파라미터를 최적화하면 더 나은 결과를 얻을 수 있지만, *우리의 목표는 각 미니배치 크기에 대한 하이퍼파라미터 튜닝을 피하는 일반적인 전략을 사용하여 미니배치 크기 전반에 걸쳐 오류를 일치시키는 것입니다.*

k	n	kn	η	상위 1% 오류(%)
8	32	256	0.1	23.60 ±0.12
256	32	8k	3.	224.84 ±0.37
256	32	8k	3.	225.88 ±0.56
256	32	8k	3.	223.74 ±0.09

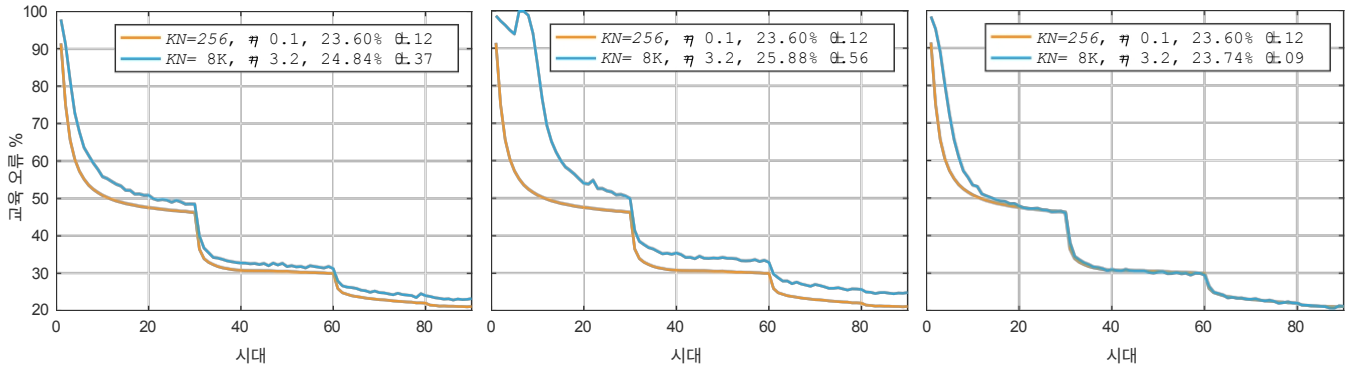
표 1. **ResNet-50을 사용한 ImageNet의 검증 오류**(5회에 걸쳐 계산된 평균 및 표준편차). 다양한 워밍업 전략을 사용하여 소규모 미니배치 모델(*kn=256*)과 대규모 미니배치 모델(*kn=8k)을* 비교합니다. 소규모 및 대규모 미니배치 훈련(점진적 워밍업 포함)의 상위 1% 검증 오차는 각각 23.60% ±0.12% *대* 23.74% ±0.09%로 매우 비슷합니다.

훈련 오류. 훈련 곡선은 그림 2에 나와 있습니다. 워밍업이 없는 경우(2a), kn = 8k의 대규모 미니배치에 대한 훈련 곡선은 모든 에포크에 걸쳐 kn = 256의 소규모 미니배치를 사용한 훈련보다 열등합니다. 일정한 워밍업 전략(2b)은 실제로 결과를 저하시킵니다. 작은 일정한 학습 속도가 워밍업 동안 오류를 줄일 수 있지만, 워밍업 직후 오류가 급증하고 훈련이 완전히 회복되지 않습니다.

주요 결과는 점진적인 워밍업을 통해 대규모 미니배치 훈련 오차가 소규모 미니배치를 통해 얻은 기준 훈련 곡선과 일치한다는 것입니다(그림 2c 참조). 워밍업 단계에서는 낮은 η*로* 인해 대규모 미니배치 곡선이 더 높게 시작하지만, 곧 따라잡힙니다. 약 20개의 에포크가 지나면 작은 미니배치 훈련 곡선과 큰 미니배치 훈련 곡선이 거의 일치합니다. *워밍업이*없는 경우와 점진적 워밍업의 비교는 *대규모 미니배치 크기가 초기 훈련에서 최적화 어려움으로 인해 어려움을 겪고* 있으며, 이러한 어려움이 해결되면 훈련 오류와 그 곡선이 소규모 미니배치 기준선과 거의 일치할 수 있음을 시사합니다.

유효성 검사 오류. 표 1은 세 가지 워밍업 전략에 대한 *유효성 검사 오류를* 보여줍니다. 워밍업 없음 변형은 다음과 같습니다. 기준선보다 ~2% 높은 검증 오류는 과적합이나 다른 원인으로 인한 생성 오류보다는 훈련 오류의 ~2.1% 증가로 인한 것일 가능성이 높습니다(그림 2a). 이러한 주장은 점진적 워밍업 실험을 통해 더욱 뒷받침됩니다. 점진적 워밍업 변형의 *검증* 오차는 기준선의 0.14% 이내입니다(이 추정치의 표준편차는 ~0.1%입니다). 이 경우 최종 훈련 오차(그림 2c)가 잘 일치한다는 점을 감안할 때, *최적화 문제가 해결되면* 미니배치 크기가 256에서 8k로 증가하더라도 *대규모 미니배치 훈련에서 일반화 성능 저하가 관찰되지 않는다는* 것을 *알 수* 있습니다.

마지막으로 그림 4는 점진적인 워밍업이 포함된 대규모 미니배치 훈련의 훈련 및 검증 곡선을 모두 보여줍니다. 그림에서 볼 수 있듯이, 두 번째 학습률 하락 이후 검증 오차가 기준선과 거의 일치하기 시작하는데, 실제로 실행 평균을 사용하는 대신 오류를 평가하기 전에 BN 통계를 다시 계산하면 검증 곡선이 더 일찍 일치할 수 있습니다(그림 4의 캡션도 참조).



(a) 워밍업 없음 (b) 지속적인 워밍업 (c) 점진적인 워밍업

그림 2. **워밍업.** 미니배치 크기 256과 비교하여 다양한 워밍업 전략을 사용한 미니배치 크기 8192의 학습 오류 곡선. *유효성 검사* 오류(5회 실행의 평균±표준편차)는 범례에 미니배치 크기 kn 및 참조 학습률 η와 함께 표시됩니다.

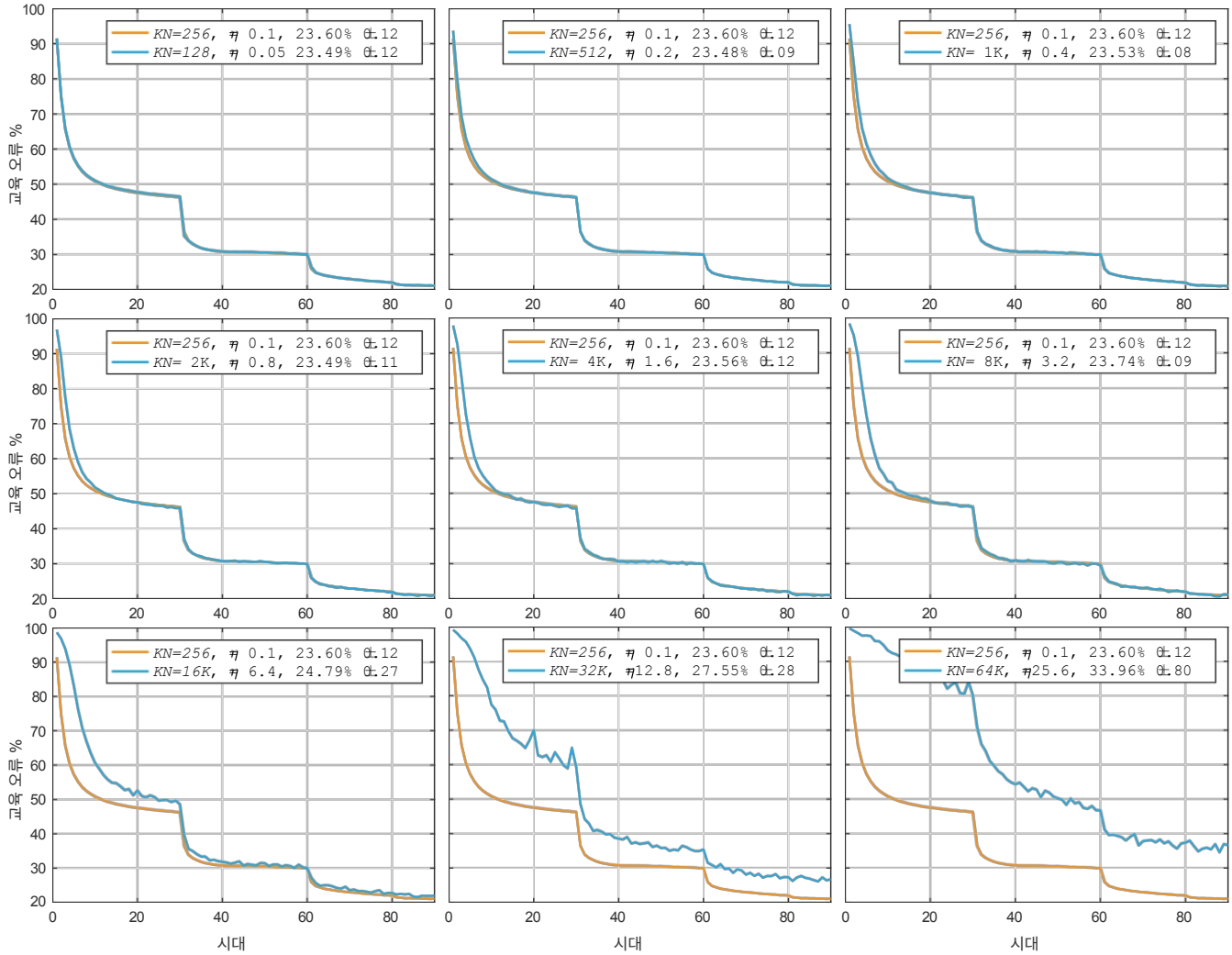


그림 3. **훈련 오류와 미니배치 크기 *비교*** 점진적 워밍업과 선형 스케일링 규칙을 사용한 256개 미니배치 기준선 및 더 큰 미니배치에 대한 훈련 오류 곡선. 훈련 곡선이 8k 미니배치까지 기준선(워밍업 기간 제외)과 거의 일치하는 것을 확인할 수 있습니다. *유효성 검사* 오차

(5회 실행의 *평균±표준편차*)는 범례에 미니배치 크기 kn 및 기준 학습률 η와 함께 표시되어 있습니다.

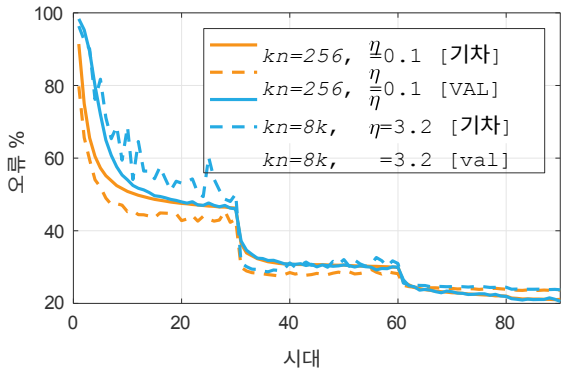
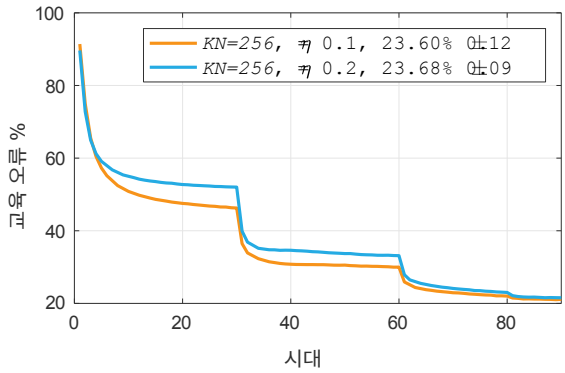


그림 4. 점진적 워밍업이 포함된 대규모 미니배치 SGD와 소규모 미니배치 SGD의 **훈련 및 검증 곡선** *비교*. 두 곡선 세트 모두 충분한 에포크 기간 동안 훈련한 후 거의 일치합니다. BN 통계(추론 전용)는 대규모 미니배치에서는 업데이트 빈도가 적고 따라서 초기 훈련에서 더 노이즈가 많은 *실행* 평균을 사용하여 계산된다는 점에 유의하십시오(초기 에포크에서 검증 오류의 변동폭이 더 큰 것을 설명해 줍니다).

5.3. 분석 실험

미니배치 크기와 오류 *비교*. 그림 1(1페이지)은 64~6536(64k) 범위의 미니배치 크기로 학습된 모델에 대한 상위 1%의 검증 오류를 보여줍니다. 모든 모델에 대해 선형 스케일링 규칙을 사용했으며 기준 학습률을 $\eta = 0.1\frac{kn}{256}$로 설정했습니다. *kn이 256을 초과하는* 모델의 경우, 항상 $\eta = 0.1$로 시작하여 5회 에포크 후에 기준 학습률로 선형적으로 증가하는 점진적 워밍업 전략을 사용했습니다. 그림 1은 64~8k의 광범위한 미니 배치 크기에서 유효성 검사 오류가 안정적으로 유지되다가 그 이후부터 증가하기 시작하는 것을 보여줍니다. 선형 학습률 스케일링 규칙을 사용할 경우 64k를 초과하면 훈련이 달라집니다.[5]

다양한 미니배치 크기에 대한 훈련 곡선. 그림 3의 9개 플롯 각각은 256개 미니배치 기준선(주황색)에 대한 상위 1%의 훈련 오류 곡선과 다양한 크기의 미니배치에 해당하는 두 번째 곡선(파란색)을 보여줍니다. 유효성 검사 오류는 플롯 범례에 표시되어 있습니다. 미니배치 크기가 증가함에 따라 모든 훈련 곡선은 훈련 시작 시 기준선과 약간의 차이를 보입니다. 그러나 최종 검증 오차가 기준선(*kn* 8k)과 거의 일치하는 경우, 초기 에포크 이후 훈련 곡선도 거의 일치합니다. 검증 오차가 일치하지 않는 경우(*kn* 16k), 모든 에포크의 훈련 곡선에 눈에 띄는 차이가 있습니다. 이는 새로운 설정을 비교할 때 훈련이 완료되기 훨씬 전에 훈련 곡선을 신뢰할 수 있는 성공 프록시로 사용할 수 있음을 시사합니다.



대체 학습률 규칙. 표 2a는 여러 학습률에 대한 결과를 보여줍니다. 작은 미니배치($kn = 256$)의 경우,

[5] 하드웨어의 가용성 때문에 단일 서버에서 매우 큰 미니배치(12k)의 분산 훈련을 시뮬레이션하여 SGD 업데이트 사이에 여러 단계의 그라데이션 누적 단계를 거쳤습니다. 단일 서버에서 그라데이션 축적이 분산 훈련과 동등한 결과를 산출한다는 것을 철저히 검증했습니다.

그림 5. **학습률 η가 다른 소규모 미니배치에 대한 훈련 곡선.** 예상대로 η를 변경하면 커브가 *일치하지 않습니다*. 이는 배치 크기(및 선형 스케일링 η)를 변경하면 *일치하는* 곡선이 생성되는 것과는 대조적입니다(*예:* 그림 3 참조).

$\eta = 0.1$이 가장 좋은 오차를 제공하지만 이보다 약간 작거나 큰 η도 잘 작동합니다. 8k 이미지의 미니 배치에 선형 스케일링 규칙을 적용하면 $\eta = 0.1$ 32에서도 최적의 오차가 발생하여 선형 스케일링 규칙이 성공적으로 적용되었음을 알 수 있습니다. 그러나 이 경우 결과는 η 변경에 더 민감하게 반응합니다. 실제로는 중단점에 가깝지 않은 미니배치 크기를 사용하는 것이 좋습니다.

그림 5는 $\eta = 0.1$ 또는 0.2를 사용하는 256개 미니배치의 훈련 곡선을 보여줍니다. 일반적으로 학습률 η를 변경하면 최종 오차가 비슷하더라도 학습 곡선의 전체적인 모양이 달라지는 것을 보여줍니다. 이 결과를 선형 스케일링 규칙(미니배치 크기가 변경될 때 최종 오류와 학습 곡선을 모두 일치시킬 수 있음)의 성공과 대조하면 작은 미니배치와 큰 미니배치 간에 유지되는 근본적인 불변성을 확인할 수 있습니다.

또한 η를 0.1로 고정하거나 [21]에서 이론적으로 정당화된 제곱근 스케일링 규칙에 따라 0.1 32를 사용하는 두 가지 대안 전략을 보여줍니다. 이 규칙은 점진적 추정기의 표준편차 감소의 반비례만큼 η를 스케일링한다는 근거로 사용됩니다. 공정한 비교를 위해 0.1 32에 대한 점진적 워밍업도 사용합니다. 결과에서 알 수 있듯이 두 정책 모두 실제로는 제대로 작동하지 않습니다.

일괄 정규화 γ 초기화. 표 2b는 새로운 BN γ 초기화의 영향에 대한 제어입니다.

§5.1. 표준 BN 초기화(모든 BN 레이어에 대해 $\gamma = 1$)와 유니티의 초기화(각 잔여 블록의 최종 BN 레이어에 대해 $\gamma = 0$)를 사용한 미니배치 크기 256 및 8k의 결과를 보여줍니다. 결과는 두 미니배치 크기 모두에서 $\gamma = 0$일 때 향상된 성능을 보여주며, 8k 미니배치 크기에서 그 개선 폭이 약간 더 큽니다. 이 동작은 또한 대규모 미니배치가 최적화 어려움에 더 쉽게 영향을 받는다는 것을 시사합니다. 개선된 최적화 및 초기화 방법이 대규모 미니배치 훈련의 한계를 뛰어넘는 데 도움이 될 것으로 기대합니다.

ResNet-101. ResNet-101[16]에 대한 결과는 표 2c에 나와 있습니다. $kn = 8$k의 배치 크기로 ResNet-101 훈련하기

kn	η	상위 1% 오류(%)
256	0.0523	.92 ±*0*.10
256	0.1023	.60 ±*0*.12
256	0.2023	.68 ±*0*.09
8k0.05 - 3224		.27 ±*0*.08
	8k0.10 - 3223	.74 ±*0*.09
	8k0.20 - 3224.	05 ±*0*.18
8k	0 . 1 √0	41.67 ±*0*.10
8k0.10 -	3226	.22 ±*0*.03

(a) **학습률 스케일링 규칙 비교.** 기준 학습률 $\eta = 0.1$은 $kn = 256$(오차 23.68%)에서 가장 잘 작동합니다. 선형 스케일링 규칙은 $kn = 8$k일 때 $\eta = 0.1$ 32를 제안하며, 이 역시 최상의 성능을 제공합니다(23.74% 오차). η를 스케일링하는 다른 방법은 더 나쁜 결과를 제공합니다.

kn		ηγ-inittop-1 오류(%)	
256	0.1	1.	023.84 ±*0*.18
256	0.1	0.	023.60 ±*0*.12
8k	3.2	1.024	.11 ±*0*.07
8k	3.2	0.	023.74 ±*0*.09

×

(b) **배치 정규화 γ 초기화.** 각 잔여 블록의 *마지막* BN 레이어에서 $\gamma = 0$으로 초기화하면 소규모 및 대규모 미니배치 모두에서 결과가 향상됩니다. 이 초기화는 더 나은 최적화 동작으로 이어져 대규모 미니배치로 훈련할 때 더 큰 긍정적인 영향을 미 칩 니 다 .

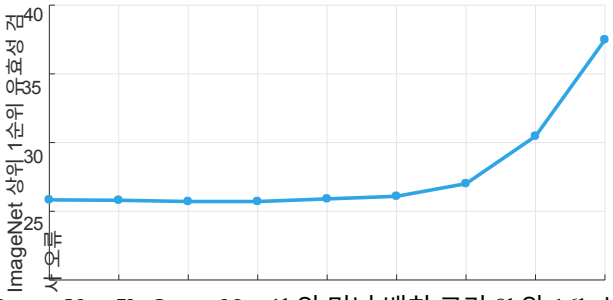
모델 유형	*kn*	η	상위 1% 오류 (%)
ResNet-101	256	0.1	22.08 ±*0*.06
ResNet-101	8k	3.2	22.36 ±*0*.09

(c) **ResNet-101에 적용된 선형 스케일링 규칙.** 소규모와 대규모 미니배치 훈련 간의 오차 차이는 약 0.3%입니다.

표 2. **ImageNet 분류 실험.** 특별한 언급이 없는 한 모든 실험은 ResNet-50을 사용하며 5번의 실험을 통해 평균을 냈습니다.

선형 스케일링된 $\eta = 3.2$는 22.36%의 *오류를 발생시키며*, $kn = 256$ 기준선은 $\eta = 0.1$로 22.08%를 달성합니다. 즉, 미니 배치 8k로 훈련된 ResNet-101은 기준선 대비 오차가 0.28% 소폭 증가했습니다. 미니배치 크기 8k는 ResNet-101에 유용한 미니배치 훈련 체제의 가장자리에 위치하며, 거의 ResNet-50과 유사합니다(그림 1 참조).

256개의 Tesla P100 GPU와 8k의 미니 배치 크기를 사용한 구현에서 ResNet-101의 훈련 시간은 92.5분이었습니다. 속도와 정확도의 절충점을 고려할 때 ResNet-101의 이 같은 결과는 매우 매력적인 결과라고 생각합니다.



ImageNet-5k. ImageNet-1k의 미니 배치 크기 8k와 16k 사이에서 검증 오류가 급격히 증가하는 것을 관찰하면(그림 1), 오류 곡선에서 이 '팔꿈치'의 위치가 데이터 세트 정보의 함수인지 자연스럽게 질문할 수 있습니다. 내용 이 질문을 조사하기 위해Xie *등*[39]이 제안한 ImageNet-5k 데이터 세트를 채택하여 ImageNet-1k에서 4k 카테고리를 추가하여 ImageNet-22k[33]의 680만 이미지로 확장(약 5개 더 큰 규모) 합 니 다 . 우리는 [39]에서와 같이 원래 ImageNet-1k 검증 세트에 대해 1k 방향 분류 오류를 평가합니다.

ImageNet-5k의 미니배치 크기 대 유효성 검사 오류 곡선은 그림 6에 나와 있습니다. 질적으로, 곡선

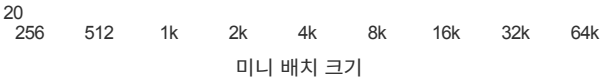


그림 6. 90개의 고정된 에포크 트레이닝 스케줄을 사용한 **ImageNet-5k** 상위 1% 검증 오류와 미니배치 크기 *비교*. 이 곡선은 훈련 데이터가 *5배* 증가해도 최대 유효 미니배치 크기에 큰 변화가 없음을 보여주는 ImageNet-1k의 결과(그림 1)와 질적으로 유사합니다.

35.1 ±*0.333* .2 ±*0.3*

(a) **대규모 미니배치 사전 훈련의 학습을 마스크 R-CNN으로 전송합니다.** 256~8k 예제의 미니배치로 사전 훈련된 50개의 모델에 대해 박스 및 마스크 AP(COCO 미니벌의 경우)가 거의 동일합니다. 미니배치 사전 훈련 크기가 16k인 경우 ImageNet 검증 *오류와* COCO AP가 모두 악화됩니다. 이는 이미지넷 오류가 일치하는 한, 대규모 미니배치는 전송 학습 성능을 저하시키지 않는다는 것을 나타냅니다.

(b) **마스크 R-CNN에 적용된 선형 학습 속도 스케일링.** 16]의 sin-gle ResNet-50 모델을 사용하여(따라서 표준값이 보고되지 않음) 선형 학습 속도 스케일링 규칙에 따라 1~8개의 GPU를 사용하여 Mask R-CNN을 훈련합니다. 박스 및 마스크 AP는 모든 구성에서 거의 동일하게 나타나 분류를 넘어 규칙이 성공적으로 일반화되었음을 보여줍니다.

표 3. **마스크 R-CNN을 사용한 COCO에서의 물체 감지 [14].**

# GPU	*kn*	η - 1000	반복	상자 AP (%)	마스크 AP (%)
1	2	2.5			
2	4	5.0	1,280,000	35.7	33.6
4	8	10.0	640,000	35.7	33.7
8	16	20.0	320,000	35.7	33.5
			160,000	35.6	33.6

×

는 ImageNet-1k 곡선과 매우 유사하며, 실무자의 경우 데이터 세트 크기가 5배 증가하더라도 사용 가능한 미니배치 크기가 의미 있게 증가하지 않을 가능성이 높다는 것을 보여줍니다. 정량적으로 8k 미니배치를 사용하면 유효성 검사 오류가 256 미니배치의 25.83%에서 26.09%로 0.26% 증가합니다. 일반화 오류, 미니배치 크기, 데이터 세트 정보 콘텐츠 간의 정확한 관계에 대한 이해는 향후 연구를 위해 열려 있습니다.

5.4. 탐지 및 세분화에 대한 일반화

ImageNet의 낮은 오류율은 일반적으로 최종 목표가 아닙니다. 대신 ImageNet 훈련의 유용성은 학습에 있습니다.

이미지넷 사전 교육			COCO	
kn	η	상위 1% 오류(%)	상자 AP (%)	마스크 AP (%)
256	0.1	23.60 ±*0.*12	35.9 ±*0.*1	33.9 ±*0.*1
512	0.2	23.48 ±*0.*09	35.8 ±*0.*1	33.8 ±*0.*2
1k	0.4	23.53 ±*0.*08	35.9 ±*0.*2	33.9 ±*0.*2
2k	0.8	23.49 ±*0.*11	35.9 ±*0.*1	33.9 ±*0.*1
4k	1.6	23.56 ±*0.*12	35.8 ±*0.*1	33.8 ±*0.*1
8k	3.2	23.74 ±*0.*09	35.8 ±*0.*1	33.9 ±*0.*2
16k	6.4	24.79 ±*0.*27		

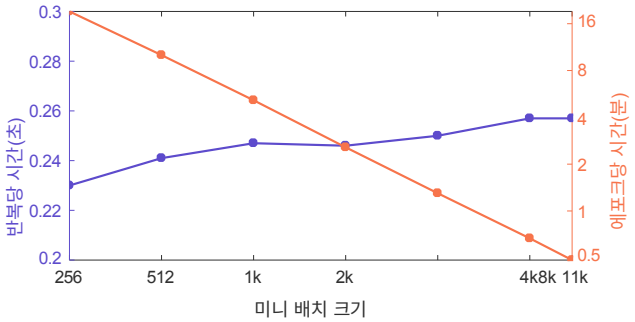
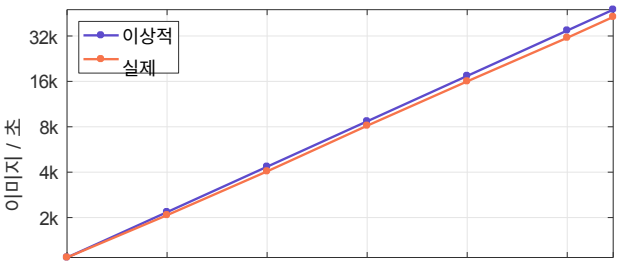


그림 7. **분산 동기식 SGD 타이밍.** 다양한 미니배치 크기로 트레이닝할 때의 이터레이션당 시간(초) 및 ImageNet 에포크당 시간(분). 기준선($kn = 256$)은 단일 서버에서 8개의 GPU를 사용하며, 다른 모든 훈련 실행은 ($kn/256$) 서버에 걸쳐 훈련을 분산합니다. 352개의 GPU(44개 서버)로 구현하면 약 30초 만에 약 128만 개의 이미지넷 트레이닝 이미지를 모두 한 번에 완료할 수 있습니다.

좋은 기능을 다시 수행되는 작업에 잘 전달하거나 일반화할 수 있습니다. 중요한 질문은 대규모 미니배치로 학습한 기능이 소규모 미니배치로 학습한 기능만큼 일반화되는지 여부입니다.

이를 테스트하기 위해 COCO[27]에서 물체 감지 및 자세 세분화 작업을 채택했는데, 이러한 고급 인식 작업은 ImageNet 사전 학습[10]의 이점을 크게 활용하기 때문입니다. 최근에 개발된 Mask R-CNN을 사용합니다.
객체 인스턴스를 감지하고 분할하는 방법을 학습할 수 있는 [14] 시스템입니다. 여기서는 [14]에서 사용된 모든 하이퍼파라미터 설정을 따르고 마스크 R-CNN 학습을 초기화하는 데 사용되는 ResNet-50 모델만 변경합니다. COCO `trainval35k` 분할에 대해 Mask R-CNN을 훈련하고 [14]에서 사용된 5k 이미지 미니벌 분할에 대한 결과를 보고합니다.

마스크 R-CNN에서 미니배치 크기의 개념이 분류 설정과 다르다는 점은 흥미롭습니다. *이미지 중심의* 고속/고속 R-CNN[9, 31]의 확장으로서, Mask R-CNN은 *레이어마다 다른 미니배치 크기를* 나타냅니다. 네트워크 백본은 GPU당 2개의 이미지를 사용하지만, 각 이미지는 분류(다항식 ×



교차 엔트로피), 경계 상자 회귀(부드러운-L1/Huber), 픽셀 단위 마스크(2 8 개 의 이항식 교차 엔트로피) 손실을 계산하는 데 512개의 관심 영역(Regions-of-Interest)에 기여합니다. 이러한 다양한 미니배치 크기와 손실 함수는 유니티 접근 방식의 견고성을 입증하는 좋은 테스트 사례입니다.

대규모 미니배치 사전 훈련에서 전이 학습. 대규모 미니배치 *사전 훈련이* Mask R-CNN에 어떤 영향을 미치는지 테스트하기 위해 ImageNet-1k에서 훈련된 ResNet-50 모델을 256~16k 미니배치로 가져와 Mask R-CNN 훈련을 초기화하는 데 사용했습니다. 각 미니배치 크기에 대해 5개의 모델을 사전 훈련한 다음 COCO에서 5개의 모델(총 35개 모델)을 모두 사용하여 Mask R-CNN을 훈련합니다. 표 3a에는 5번의 실험에 대한 평균 박스 및 마스크 AP가 나와 있습니다. 결과에 따르면 이미지넷 검증 오류가 낮게 유지되는 한(최대 8k 배치 크기까지 해당) 객체 제거에 대한 일반화가 가능합니다.

그림 8. **분산 동기식 SGD 처리량.** 8개의 GPU가 있는 단일 서버에서 멀티 서버 분산 트레이닝으로 전환할 때의 작은 오버헤드(그림 7, 파란색 곡선)로 인해 이상적인 확장(~90% 효율)에 약간 못 미치는 선형 처리량 확장으로 이어집니다. 대부분의 올리듀스 통신 시간은 그라디언트 계산으로 올리듀스 연산을 파이프라인화하여 숨겨져 있습니다. 또한 이는 상용 이더넷 하드웨어로 달성할 수 있습니다.

검출은 소규모 미니배치 기준선의 AP와 일치합니다. 대규모 미니배치로 학습된 모델을 사용하여 데이터 세트 간(ImageNet에서 COCO로), 작업 간(분류에서 탐지/세분화까지) 전송할 때 *일반화 문제가* 관찰되지 *않았다는 점을* 강조합니다.

마스크 R-CNN에 적용된 선형 스케일링 규칙. 또한 마스크 R-CNN에 사용된 선형 스케일링 규칙의 일반성에 대한 증거를 보여줍니다. 사실 이 규칙은 이미 [16]에서 명시적인 논의 없이 사용되었으며, 8개의 GPU를 사용할 때 기본 Mask R-CNN 훈련 체계로 효과적으로 적용되었습니다. 표 3b는 1, 2, 4 또는 8개의 GPU로 훈련할 때 선형 학습률 규칙이 일정한 박스 및 마스크 AP를 생성한다는 것을 보여주는 실험 결과를 제공합니다. 이 실험에서는 [14]에서 수행한 것과 같이 출시된 MSRA ResNet-50 모델에서 마스크 R-CNN을 초기화합니다.

5.5. 실행 시간

그림 7은 시스템의 런타임 특성에 대한 두 가지 시각화를 보여줍니다. 파란색 곡선은 미니배치 크기가 256에서 11264(11k)로 변화할 때의 반복당 시간입니다. 이 곡선은 비교적 평탄하며, 미니배치 크기를 다음과 같이 확장하는 동안 반복당 시간이 12%만 증가한다는 점을 주목할 필요가 있습니다.

44 . 다른 방식으로 시각화하면 주황색 곡선은 에포크당 시간이 16분 이상에서 단 30초로 거의 선형적으로 감소했음을 보여줍니다. 런타임 성능은 그림 8과 같이 처리량(이미지/초) 측면에서도 볼 수 있습니다. 8 GPU 기준선의 완벽한 효율성과 비교했을 때, 유니티의 구현은 다음을 달성합니다.
~90%의 확장 효율성.

감사의 말. 이론적 배경에 대한 유용한 토론을 해주신 Leon Bottou, 효율적인 데이터 로딩에 대한 토론을 해주신 Jerry Pan과 Christian Puhrsch, 분산 훈련 디버깅에 도움을 주신 An- drew Dye, Big Basin 및 하드웨어 지원을 해주신 Kevin Lee, Brian Dodds, Jia Ning, Koh Yew Thoon, Micah Harris, John Volk에게 감사의 말씀을 전합니다.

×

참조

[1] J. Bagga, H. Morsy, Z. Yao. 6팩과 웨지 100의 오프닝 디자인. https://code.facebook.com/posts/203733993317833/opening-designs-for-6-pack-and-wedge-100, 2016.

[2] M. Barnett, L. 슐러, R. 반 데 가인, S. 굽타, D. G. 페인, 및 J. 와츠. 프로세서 간 집단 통신 라이브러리 (인터콤). *확장 가능한 고성능 통신 퍼팅 컨퍼런스*, 1994.

[3] L. Bottou. 일부 확률적 경사 하강 알고리즘의 신기할 정도로 빠른 수렴. 미공개 공개 문제- 2009 년 SLDS 2009 컨퍼런스 참석자에게 전달되었습니다.

[4] L. Bottou, F. E. 커티스, 및 J. 노세달. Opt. methods for 대규모 머신 러닝. *arXiv:1606.04838*, 2016.

[5] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Joze- fowicz. 분산 동기식 SGD 재검토. *arXiv:1604.00981*, 2016.

[6] K. 첸과 Q. 후오. 블록 내 대립 유전자 최적화 및 블록별 모델 업데이트 필터링을 통한 점진적 블록 학습을 통한 딥 러닝 모델의 확장 가능한 학습. In *ICASSP*, 2016.

[7] R. 콜로버트, J. Weston, L. Bottou, M. Karlen, K. 카부쿠오글루, P.. 자연어 프로- 처음부터 (거의) 중단. *JMLR*, 2011.

[8] J. 도나휴, Y. 지아, O. 빈얄스, J. 호프만, N. 장, E. 쳉, 및 T. 대럴. Decaf: 일반 시각 인식을 위한 심층 컨볼루션 연산 vation 기능. In *ICML*, 2014.

[9] R. Girshick. 빠른 R-CNN. In *ICCV*, 2015.

[10] R. Girshick, J. Donahue, T. Darrell, 및 J. Malik. 정확한 객체 감지 및 시맨틱 세분화를 위한 풍부한 기능 계층 . In *CVPR*, 2014.

[11] W. Gropp, E. Lusk, 및 A. Skjellum. *MPI 사용하기: 메시지 전달 인터페이스를 사용한 휴대용 병렬 프로그래밍*. MIT Press, Cambridge, MA, 1999.

[12] S. Gross and M. Wilber. Resid- ual Nets 교육 및 조사. https://github.com/facebook/fb.resnet.torch, 2016.

[13] M. 구르부즈발라반, A. 오즈다글라, 및 P. 파릴로. 무작위 재구성이 확률적 경사 하강을 능가하는 이유. *arXiv:1510.08560*, 2015.

[14] K. He, G. Gkioxari, P. Dolla´r, and R. Girshick. 마스크 R-CNN. *arXiv:1703.06870*, 2017.

[15] K. He, X. Zhang, S. Ren, J. Sun. 정류기에 대한 심층 탐구: imagenet 분류에서 인간 수준의 성능을 뛰어넘기. In *ICCV*, 2015.

[16] K. He, X. Zhang, S. Ren, and J. Sun. 이미지 인식을 위한 심층 잔여 학습 . In *CVPR*, 2016.

[17] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath 등. 음성 인식에서 음향 모델링을 위한 심층 신경망: 네 연구 그룹의 공통된 견해. *IEEE 신호 처리 매거진*, 2012.

[18] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, 및 Y. 벤지오. 양자화된 신경망: 저정밀 가중치 및 활성화로 신경망 훈련. *arXiv:1510.08560*, 2016.

[19] S. 이오페와 C. 세게디. 일괄 정규화: 내부 공변량 이동을 줄여 심층 네트워크 학습을 가속화합니다. In *ICML*, 2015.

[20] N. S. 케스카, D. 무디게르, J. 노세달, M. 스멜리얀스키, 및 P. T. P. Tang. 딥 러닝을위한 대규모 배치 교육에: Gen- 실현 격차와 날카로운 최소. *ICLR*, 2017.

[21] A. Krizhevsky. 컨볼 루션 신경망을 병렬화하기 위한 한 가지 이상한 트릭 , 2014.

[22] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classi- 심층 컨볼루션 신경망을 사용한 fication. In *NIPS*, 2012.

[23] Y. 르쿤, B. 보서, J. S. 덴커, D. 헨더슨, R. E. 하워드, W. 허바드, L. D. 재켈. 손으로 쓴 우편 번호 인식에 적용된 역전파. *Neural compu- tation*, 1989.

[24] K. Lee. 빅베이슨을 소개합니다: https://code.facebook.com/posts/1835166200089399/introducing-big-basin, 2017.

[25] M. Li. *시스템 및 알고리즘 공동 설계를 통한 분산 머신 러닝 확장*. 박사 학위 논문, 카네기 멜론 대학교 versity, 2017.

[26] T.-Y. Lin, P. Dolla´r, R. Girshick, K. He, B. Hariharan, 및 S. Belongie. 객체 감지를 위한 특징 피라미드 네트워크. In *CVPR*, 2017.

[27] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ra- manan, P. Dolla´r, 및 C. L. Zitnick. Microsoft COCO: Com- mon 객체를 컨텍스트에서. In *ECCV*. 2014.

[28] J. Long, E. Shelhamer, 및 T. Darrell. 의미론적 세분화를 위한 완전 컨볼루션 네트워크. In *CVPR*, 2015.

[29] Y. 네스테로프. *볼록 최적화에 대한 입문 강의: 기본 과정*. Springer, 2004.

[30] R. 라벤세이프너. 집단 감축 운영의 최적화- ations. In *ICCS*. Springer, 2004.

[31] S. 렌, K. 허, R. 기르식, J. 선. 더 빠른 R-CNN: 지역 제안 네트워크를 통한 병동 실시간 물체 감지- 작동. In *NIPS*, 2015.

[32] H. 로빈스와 S. 몬로. 확률적 근사치 방법. *수학 통계 연대기*, 1951.

[33] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, 및 L. Fei-Fei. ImageNet 대규모 비주얼 인식 챌린지. *IJCV*, 2015.

[34] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, 및 Y. LeCun. 과체중: 통합 인식, 현지화 및 컨볼루션 네트워크를 사용한 탐지. In *ICLR*, 2014.

[35] K. Simonyan과 A. Zisserman. 대규모 이미지 인식을 위한 매우 심층적인 컨볼루션 네트워크. In *ICLR*, 2015.

[36] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. 앙겔로프, D. 에르한, V. 반후케, 및 A. 라비노비치. 컨볼루션으로 더 깊이 들어가기. In *CVPR*, 2015.

[37] R. Thakur, R. Rabenseifner, 및 W. Gropp. MPICH에서 집단 통신 운영 최적화. *IJHPCA*, 2005.

[38] Y. Wu, M. 슈스터, Z. 첸, Q. V. 르, M. 노루지, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey 외. 구글의 신경망 기계 번역 시스템: 인간과 기계 번역 사이의 격차 해소. *arXiv:1609.08144*, 2016.

[39] S. Xie, R. Girshick, P. Dolla´r, Z. Tu, and K. He. 심층 신경망을 위한 집계된 잔여 변환. In *CVPR*, 2017.

[40] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stol- cke, D. Yu, and G. Zweig. Microsoft 2016 컨버사- tional 음성 인식 시스템. *arXiv:1609.03528*, 2016.

[41] M. D. Zeiler와 R. Fergus. 컨볼루션 신경망 시각화 및 이해. In *ECCV*, 2014.