# OpenEquivariance가 SevenNet을 빠르게 만드는 원리

쉬운 설명과 수식을 함께 읽는 학습자료 · 2026년 9월 9일

## 먼저 답부터

SevenNet은 원자마다 이웃 원자의 정보와 그 이웃이 놓인 방향을 결합합니다. 이 계산을 원자쌍마다, 여러 신경망 층에서 반복합니다. OpenEquivariance는 이 결합 연산을 GPU에 맞게 실행하여 시간을 줄입니다.

핵심은 세 가지입니다. 수학적으로 0인 항은 계산하지 않고, 계산 중인 값을 GPU 안의 빠른 기억장소에서 재사용하며, 지원되는 융합 경로에서는 이웃별 결과를 저장했다가 다시 읽는 과정도 줄입니다. 같은 모델을 더 효율적으로 계산하려는 것이지, 작은 모델로 바꾸거나 학습을 생략하는 방법이 아닙니다. 공식 소개

읽는 순서는 원자 한 개 → 방향 → 특징 결합 → GPU → 힘 계산입니다. 숫자 예제는 이해를 위한 장난감 계산이며, SevenNet의 실제 예측값이나 성능 측정값이 아닙니다.

## 1 원자 하나를 따라가 보자

원자 A 주변에 B와 C가 있다고 생각해 봅시다. SevenNet은 B가 어떤 특징을 가지고 있는지, A에서 얼마나 떨어져 있는지, 어느 방향에 있는지를 함께 사용합니다. C에 대해서도 같은 계산을 합니다.

여기서 특징(feature)은 신경망이 사용하는 숫자 묶음입니다. 전하·에너지처럼 사람이 이름 붙일 수 있는 물리량과 항상 일대일로 대응하지는 않습니다.

원자 A가 받는 정보를 다음처럼 씁니다.

$$\mathsf{a}_A = \mathsf{m}_{A\leftarrow B} + \mathsf{m}_{A\leftarrow C}.$$

- $\mathsf{m}_{A\leftarrow B}$: B에서 A로 보내는 숫자 묶음, 즉 메시지.
- $\mathsf{a}_A$: A에 도착한 메시지들의 합.

숫자 예를 들어 B의 메시지가 $(2, 1)$이고 C의 메시지가 $(-1, 3)$이면 합은 $(1, 4)$입니다. 이것이 aggregation, 즉 이웃 정보 합산입니다. 다만 실제 모델은 이 합만으로 끝나지 않고 선형 변환, 비선형 함수, 자기 원자 정보와의 결합 등을 거칩니다.

원자쌍을 그래프의 edge, 원자를 node라고 부릅니다. 여기서는 $j \to i$를 정보를 보내는 방향으로 정의합니다. 소스코드를 볼 때는 해당 구현의 sender/receiver 인덱스 규칙을 따로 확인해야 합니다.

## 2 왜 방향을 특별하게 취급할까

책상 위의 분자를 통째로 90도 돌렸다고 결합 에너지가 달라지지는 않습니다. 그러나 힘을 나타내는 화살표는 분자와 함께 돌아야 합니다.

원자 좌표 전체를 R, 회전행렬을 $Q$라고 쓰면 필요한 관계는 다음과 같습니다.

$$E(Q\mathsf{R}) = E(\mathsf{R}), \qquad \mathsf{F}_i(Q\mathsf{R}) = Q\mathsf{F}_i(\mathsf{R}).$$

첫 번째는 불변성입니다. 두 번째는 등변성입니다. 각 원자 좌표에 $Q$를 적용한다는 뜻으로 $Q\mathsf{R}$를 썼습니다.

예를 들어 힘 $(1, 0, 0)$을 z축 둘레로 90도 회전하면 $(0, 1, 0)$이 됩니다. 힘의 값이 그대로여야 한다는 뜻이 아니라, 물리적으로 같은 방향 관계를 유지해야 한다는 뜻입니다.

특징에도 회전 규칙이 있다

스칼라 특징은 회전해도 그대로이고, 벡터 특징은 3개 성분이 함께 변합니다. 더 복잡한 방향 패턴은 더 많은 성분이 필요합니다.

| 차수 $l$ | 성분 수 $2l+1$ | 쉬운 의미 |
|---|---|---|
| 0 | 1 | 방향 없는 스칼라 |
| 1 | 3 | 벡터처럼 회전하는 특징 |
| 2 | 5 | 대칭이고 trace가 0인 2차 텐서에 대응하는 방향 패턴 |

이 기본 묶음을 기약표현(irrep)이라고 합니다. 특징의 회전은 다음처럼 표현합니다.

$$\mathsf{h}^{(l)} \longmapsto D^{(l)}(Q)\mathsf{h}^{(l)}.$$

$D^{(l)}(Q)$는 해당 묶음의 회전 규칙을 나타내는 $(2l+1)\times(2l+1)$ 행렬입니다. 계수값을 외울 필요는 없습니다. “서로 다른 종류의 특징은 서로 다른 규칙으로 돌아간다”가 핵심입니다.

`32x1o`는 3성분짜리 $l=1$ 묶음이 32개라는 뜻이므로 총 96개 숫자입니다. `o`는 공간 반전에서 부호가 바뀌는 odd parity, `e`는 유지되는 even parity입니다. 보통의 위치·힘 벡터는 `1o`, 외적으로 얻는 축벡터는 `1e`처럼 구별합니다.

### 3 거리와 방향을 나누어서 계산한다

수신 원자를 $i$, 이웃을 $j$라 하고 다음을 정의합시다.

$$\mathsf{r}_{ij} = \mathsf{r}_j - \mathsf{r}_i, \qquad d_{ij} = \|\mathsf{r}_{ij}\|, \qquad \hat{\mathsf{r}}_{ij} = \mathsf{r}_{ij}/d_{ij}.$$

거리 $d_{ij}$는 회전해도 같습니다. 단위 방향 $\hat{\mathsf{r}}_{ij}$는 회전합니다. 주기 경계에서는 적절한 periodic image의 변위를 사용합니다.

거리는 거리용 신경망에 넣어 가중치를 만듭니다.

$$w_{ij,p} = f_{\theta,p}(B(d_{ij})).$$

$B$는 거리의 basis 표현, $f_\theta$는 학습된 radial network, $p$는 결합 경로 번호입니다. 실제 구현에는 cutoff 처리와 정규화 등이 포함될 수 있습니다.

방향은 구면조화함수로 표현합니다.

$$y^{(l)}_{ij,m} = Y^{(l)}_m(\hat{\mathsf{r}}_{ij}).$$

구면조화함수는 “방향을 여러 각도 패턴으로 바꾸는 함수”입니다. $l=0$은 방향에 무관하고, $l=1$은 기저와 정규화에 따라 x·y·z 방향 성분에 대응합니다. 이것은 임의로 학습되는 함수가 아니라 정해진 수학적 함수입니다.

## 4 텐서곱은 두 특징을 어떻게 섞을까

먼저 단순한 외적을 봅시다. x $= (2, 3)$, y $= (5, 7)$이면 가능한 곱을 모두 모아 다음을 얻습니다.

$$\mathrm{x} \otimes \mathrm{y} = \begin{pmatrix} 10 & 14 \\ 15 & 21 \end{pmatrix}.$$

결합 정보는 생겼지만, 출력의 회전 규칙에 맞춰 정리하는 단계가 필요합니다. 이때 사용하는 고정된 변환 계수가 Clebsch–Gordan 계수, 줄여서 CG 계수입니다.

### 벡터 두 개를 섞으면 익숙한 연산이 나온다

3성분 벡터 두 개의 외적에는 9개 성분이 있습니다. 회전 특성에 따라 이를 다음과 같이 나눌 수 있습니다.

$$1 \otimes 1 = 0 \oplus 1 \oplus 2, \qquad 3 \times 3 = 1 + 3 + 5.$$

왼쪽의 숫자는 성분 수가 아니라 $l$ 값입니다. $\oplus$는 서로 다른 종류의 묶음으로 분해한다는 뜻입니다.

| 출력 묶음 | 익숙한 대응물 | 성분 수 |
|---|---|---|
| $l = 0$ | 내적 x · y | 1 |
| $l = 1$ | 외적 x × y | 3 |
| $l = 2$ | 대칭 무대각합 텐서 | 5 |

마지막 묶음은 다음 텐서로 나타낼 수 있습니다.

$$T_{ab} = \frac{x_a y_b + x_b y_a}{2} - \frac{\mathrm{x} \cdot \mathrm{y}}{3} \delta_{ab}.$$

대칭행렬의 독립 성분은 6개지만 대각합이 0이라는 조건 때문에 5개만 독립입니다. 위 대응은 기저·정규화에 따른 상수와 부호 차이를 제외한 설명입니다.

예를 들어 x $= (1, 0, 0)$, y $= (0, 1, 0)$이면 내적은 0, 외적은 $(0, 0, 1)$이고 $T_{xy} = T_{yx} = 1/2$입니다. “두 벡터를 섞는다”는 말에는 서로 다른 회전 성질의 결과를 만드는 여러 방법이 들어 있습니다.

### 정확한 CG 결합식

채널을 잠시 생략하면 다음과 같습니다.

$$t^{(l_o)}_{m_o} = \sum_{m_1=-l_1}^{l_1} \sum_{m_2=-l_2}^{l_2} C^{l_o m_o}_{l_1 m_1, l_2 m_2} x^{(l_1)}_{m_1} y^{(l_2)}_{m_2}.$$

$C$가 CG 계수입니다. $l_1, l_2$는 입력 종류, $l_o$는 출력 종류, $m$은 각 묶음 안의 성분 번호입니다. 모든 곱이 모든 출력에 영향을 주는 것은 아닙니다.

허용되는 출력은 다음 범위에 있습니다.

$$|l_1 - l_2| \leq l_o \leq l_1 + l_2.$$

반전 대칭까지 고려하면 출력 parity는 입력 parity의 곱입니다. 복소 구면 기저에서는 $m_o = m_1 + m_2$라는 선택 규칙이 나타납니다. e3nn 계열의 실수 기저에서는 비영 항의 구체적인 위치가 달라지므로 그 규칙을 실수 배열 인덱스에 그대로 적용하면 안 됩니다.

## 5 CG 계수와 학습 가중치는 다른 것이다

CG 계수는 "회전 규칙을 지키는 섞는 방법", 학습 가중치는 "각 섞는 방법을 얼마나 사용할지"를 담당합니다.

채널과 경로까지 넣은 메시지의 개념식은 다음과 같습니다.

$$m_{ij,c_o,m_o}^{(l_o)} = \sum_{p \in \mathcal{P}(l_o,c_o)} w_{ij,p} \sum_{m_1,m_2} C_{l_1(p)m_1,l_2(p)m_2}^{l_o m_o} h_{j,c_1(p),m_1}^{(l_1(p))} Y_{m_2}^{(l_2(p))}(\hat{\mathrm{r}}_{ij}).$$

$\mathcal{P}$는 모델이 선택한 허용 결합 경로 집합입니다. 경로 $p$가 어떤 입력 채널과 각도 차수를 어떤 출력으로 연결할지 지정합니다. 정규화 계수는 여기서 $w$ 또는 경로 정의에 흡수했습니다. SevenNet 버전의 모든 층을 그대로 옮긴 코드식이 아니라, 메시지 연산의 역할을 드러내는 표현입니다.

이웃 합산은 바깥에 합 하나를 추가하는 것입니다.

$$a_{i,c_o,m_o}^{(l_o)} = \sum_{j \in \mathcal{N}(i)} m_{ij,c_o,m_o}^{(l_o)}.$$

즉 순서는 이웃 특징 × 방향 특징 → CG 규칙으로 결합 → 거리별 가중치 적용 → 이웃별 합산입니다. 수학적으로 같은 결과를 내면서 실제 연산 순서는 최적화될 수 있습니다.

## 6 같은 수식인데 어떻게 더 빨라질까

### 계산할 필요가 없는 곱을 없앤다

설명용으로 출력이 다음과 같다고 합시다.

$$z = 2x_1y_1 + 0x_1y_2 + 0x_2y_1 - 3x_2y_2.$$

전용 계산은 $z = 2x_1y_1 - 3x_2y_2$만 실행하면 됩니다. 여기서 0은 작은 값을 임의로 버린 결과가 아니라 수학적으로 정확히 0인 계수입니다. 그래서 가지치기나 저정밀 근사와 다릅니다.

OpenEquivariance는 CG의 희소 구조를 분석하여 0이 아닌 항을 계산하는 GPU 커널을 생성합니다. 기존 구현도 최적화되어 있을 수 있으므로 "기존 e3nn은 모든 0을 항상 계산한다"고 일반화하면 안 됩니다. 핵심은 더 세밀한 희소성 활용과 여러 연산의 공동 스케줄링입니다. 원 논문 3절

### 모델에 맞춘 계산 순서를 준비한다

원자들이 움직여도 "이 층은 어떤 종류의 특징을 몇 채널 결합한다"는 모델 구조는 대개 고정되어 있습니다. OpenEquivariance는 이를 이용해 전용 프로그램을 만듭니다. 이것이 JIT 컴파일입니다.

요리로 비유하면 주문마다 조리법을 다시 해석하기보다, 정해진 메뉴에 맞춘 작업 순서를 준비하는 셈입니다. 달라지는 좌표와 이웃 목록은 식재료에 해당합니다. 이 비유는 컴파일 구조와 입력 데이터의 차이를 설명하는 것이며, 모든 초기화 비용이 사라진다는 뜻은 아닙니다.

SevenNet 문서는 irreps 구성별 커널 컴파일을 설명합니다. 다만 실제 실행에서는 dtype, 장치, 라이브러리 버전, 통합 경로의 다른 준비 비용도 고려해야 합니다. SevenNet 가속기 문서

### 값을 멀리 가지러 가는 횟수를 줄인다

GPU의 큰 메모리는 창고, register는 작업자 손, shared memory는 가까운 공동 작업대라고 생각하면 됩니다. 창고에는 많이 저장할 수 있지만 매 곱셈마다 왕복하면 느립니다. 전용 커널은 곧 다시 쓸 값을 가까운 곳에 두고 계산 순서를 조정합니다.

이때 register와 shared memory는 작습니다. 모든 값을 무조건 올려놓을 수 없으므로, 필요한 부분을 나눠 불러오는 스케줄이 중요합니다. GPU가 동시에 처리할 수 있는 작업 수와 메모리 사용량 사이에도 절충이 있습니다.

## 7 융합은 결과를 만들자마자 합산하는 것이다

B의 메시지 $(2, 1)$과 C의 메시지 $(-1, 3)$을 다시 봅시다. 두 메시지 배열을 저장한 뒤 다시 읽어 합하면 $(1, 4)$입니다. 다른 방법은 B의 결과를 누적값에 더하고, C의 결과도 바로 더하는 것입니다.

| 비교 항목 | 분리 실행의 개념 | 융합 실행의 개념 |
|---|---|---|
| 계산 | 이웃마다 메시지 계산 | 같은 메시지 계산 |
| 중간 결과 | edge별 배열에 기록 | 커널 안에서 집계하여 기록을 줄임 |
| 합산 | 별도 단계에서 배열을 다시 읽음 | 메시지 계산과 함께 처리 |
| 최종 값 | 원자별 합 | 같은 원자별 합을 목표로 함 |

이를 커널 융합(kernel fusion)이라고 합니다. OpenEquivariance는 단독 텐서곱용 `TensorProduct`와 집계까지 묶는 `TensorProductConv`를 구분해 제공합니다. 설치되어 있다는 사실만으로 현재 SevenNet 실행이 융합 경로를 쓴다고 결론 내리면 안 됩니다. 통합 코드나 profiler로 실제 호출을 확인해야 합니다. API 문서

### 메모리 규모를 숫자로 느껴 보기

원자 10,000개, 원자당 방향 있는 이웃 연결 50개라면 edge 수는 약 500,000개입니다. 메시지당 128개 float32 숫자를 저장한다고 가정하면 중간 배열 하나는 다음 크기입니다.

$$500{,}000 \times 128 \times 4 \text{ bytes} = 256 \text{ MB}.$$

이 배열을 한 번 쓰고 한 번 읽으면 단순 계산으로 512 MB의 전송량입니다. 이는 예시 배열의 십진 단위 크기이며, 실제 모델 메모리나 반드시 절약되는 양이 아닙니다. 가중치·방향 특징·역전파용 정보와 집계 버퍼는 별도로 필요합니다.

또한 여러 edge가 같은 원자에 동시에 값을 더하므로 충돌 처리가 필요합니다. atomic 연산은 동시 덧셈을 처리하지만 합산 순서가 달라질 수 있습니다. 결정적 집계 방식은 정렬된 그래프와 추가 준비가 필요할 수 있습니다. 부동소수점에서는 덧셈 순서가 바뀌면 작은 차이가 날 수 있습니다.

## 8 MD에서도 역전파가 필요한 이유

"학습은 끝났는데 왜 backward를 계산하지?"라는 질문이 중요합니다. SevenNet이 예측한 총 에너지로부터 힘을 얻으려면, 좌표를 조금 바꿀 때 에너지가 어떻게 바뀌는지 알아야 합니다.

$$\mathsf{F}_i = -\frac{\partial E_\theta(\mathsf{R})}{\partial \mathsf{r}_i}.$$

쉬운 1차원 예를 보겠습니다. $E(x) = kx^2/2$라면 $F(x) = -kx$입니다. 오른쪽으로 갈수록 에너지가 증가하면 힘은 왼쪽을 향합니다. 이것이 마이너스 부호의 의미입니다.

MD에서는 학습 파라미터 $\theta$를 고정하고 좌표에 대한 미분을 구합니다. 파라미터 업데이트가 없어도 역전파가 필요한 이유입니다. 그러므로 에너지만 계산하는 forward 시간으로 MD 속도를 판단하면 안 됩니다.

텐서곱의 미분도 비슷한 계산이다

간단히 $z_k = \sum_{ab} C_{abk} x_a y_b$라고 합시다. 뒤쪽 층에서 전달된 미분을 $g_k = \partial E/\partial z_k$라 하면 연쇄법칙으로 다음을 얻습니다.

$$\frac{\partial E}{\partial x_a} = \sum_{bk} g_k C_{abk} y_b, \qquad \frac{\partial E}{\partial y_b} = \sum_{ak} g_k C_{abk} x_a.$$

forward와 마찬가지로 CG 계수의 비영 항을 사용한 곱과 합입니다. backward 커널도 최적화할 수 있는 이유입니다. 실제 좌표 미분은 방향 특징뿐 아니라 거리에서 생성한 가중치를 통한 경로도 포함합니다.

힘을 정답에 맞추어 학습할 때는 한 단계 더 필요합니다.

$$\mathcal{L}_F = \sum_i \|\mathsf{F}_i - \mathsf{F}_i^*\|^2, \qquad \frac{\partial \mathcal{L}_F}{\partial \theta} = -2 \sum_i (\mathsf{F}_i - \mathsf{F}_i^*)^\mathsf{T} \frac{\partial^2 E}{\partial \mathsf{r}_i \partial \theta}.$$

따라서 일반 MD의 힘 계산은 에너지의 1차 미분, 힘 손실을 통한 학습은 혼합 2차 미분이 필요합니다. 이 두 가지를 구별해야 합니다. 원 논문 3.3절과 3.4절

## 9 LAMMPS 전체가 같은 배수로 빨라지지는 않는다

LAMMPS는 원자 이동, 이웃 탐색, 통신, 출력 등을 담당하고, SevenNet 모델은 에너지와 힘을 계산합니다. OpenEquivariance의 주된 대상은 그 모델 안의 특정 연산입니다.

예를 들어 한 step이 원래 100 ms라고 합시다. 가속 가능한 부분이 70 ms이고 나머지가 30 ms라면, 해당 부분이 5배 빨라져도 전체는 44 ms입니다.

$$T_{\text{new}} = 30 + 70/5 = 44\,\text{ms}, \qquad S = 100/44 \approx 2.27.$$

일반적으로 가속 대상의 기존 시간 비율을 $p$, 그 부분의 가속 배수를 $s$라고 하면 다음과 같습니다.

$$S_{\text{total}} = \frac{1}{(1-p) + p/s}.$$

이 식은 추가 오버헤드가 없다고 가정합니다. 다른 실행 경로로 바꾸면서 데이터 변환 등의 비용이 늘면 실제 결과는 더 나쁠 수도 있습니다.

또한 원자 수만으로 작업량을 판단하면 부족합니다. 방향 있는 edge 수는 대략 $N\bar{n}$이며, $\bar{n}$은 평균 이웃 수입니다. 같은 원자 수라도 밀도와 cutoff가 다르면 작업량이 달라집니다. cutoff나 채널 수를 바꾸면 모델 자체가 달라질 수 있으므로 공정한 가속 비교에서는 고정합니다.

## 10 실제로 확인할 것은 세 가지

1. 같은 계산인가: 같은 체크포인트·구조·dtype에서 에너지, 힘, 필요 시 응력을 비교합니다. 허용오차는 목적과 정밀도에 맞춰 미리 정합니다.
2. 가속기가 실제 실행되는가: 설치 확인과 실제 커널 호출은 다릅니다. 배포 모델의 가속 설정 및 profiler의 실행 경로를 확인합니다. 융합 여부도 별도 확인 대상입니다.
3. 전체 MD가 빨라졌는가: 초기 컴파일을 제외한 구간을 반복 측정하고, neighbor rebuild 시점의 지연과 steady-state 비용을 구분합니다.

좌표가 움직이는 것, edge 수가 변하는 것, irreps 구조가 바뀌는 것은 서로 다릅니다. edge 변화 자체가 매번 CG 커널을 다시 컴파일해야 한다는 뜻은 아닙니다. 반대로 커널 재컴파일이 없다고 그래프 정렬·버퍼 준비·이웃 탐색 비용까지 없는 것은 아닙니다.

설치·빌드 명령은 이 개념 학습자료와 분리해 현재 사용하는 버전의 문서를 따르는 편이 안전합니다. SevenNet 0.13.0 가속기 문서는 OpenEquivariance를 experimental로 표시하며, PyTorch LAMMPS 경로의 `--enable_oeq` 지원을 안내합니다. LAMMPS 빌드 설정과 가속 모델 배포 설정을 모두 확인해야 합니다. 가속기 안내, LAMMPS PyTorch 안내

## 11 이해했는지 스스로 설명해 보기

왜 CG 계수를 학습하지 않을까? 회전 규칙에 맞는 결합을 정하는 수학적 계수이기 때문입니다. 모델은 허용 경로의 비중을 조절하는 가중치 등을 학습합니다.

왜 GPU에서 곱셈을 줄이는 것만큼 메모리 이동이 중요할까? 계산할 수 있는 속도에 비해 데이터를 가져오는 속도가 부족하면 연산 장치가 기다리기 때문입니다.

왜 원자가 움직여도 같은 커널을 사용할 수 있을까? 좌표·이웃 목록은 입력 데이터이고, 결합할 특징의 종류와 채널 구조는 모델에서 고정되어 있기 때문입니다. 단, 통합층의 다른 준비 작업은 남습니다.

왜 forward가 빨라져도 MD가 느릴 수 있을까? 힘을 위한 backward, 이웃 탐색, 통신 등도 한 step에 포함되기 때문입니다.

한 문장으로 설명하면? OpenEquivariance는 원자 정보와 방향 정보를 결합하는 동일한 수학을, 불필요한 곱셈과 메모리 왕복을 줄이는 GPU 프로그램으로 실행합니다.

## 더 읽을 자료

- OpenEquivariance 원 논문: 2.2절에서 결합식, 3절에서 커널 최적화, 3.5절에서 융합을 읽으면 됩니다.
- OpenEquivariance API: `TensorProduct`와 `TensorProductConv`의 차이를 확인합니다.
- SevenNet 공식 저장소: 사용하는 버전의 통합 구현을 확인합니다.

이 자료의 숫자는 모두 설명용 예제입니다. 특정 SevenNet 모델이나 GPU에서의 가속률을 측정하거나 보장한 자료는 아닙니다.