1. 예시 데이터 관찰

입력은 3차원 벡터 두 개로 구성된 2×3 행렬이다. 이 행렬을 FFN에 넣어 각 토큰(행)이 어떻게 변하는지 시각화해봤고, 어렵지만 희미하게나마 단계별 차이를 비교할 수 있었다. 가중치를 항등 행렬처럼 설정했을 때는 bias가 색에 미세한 변화를 주는 모습을 관찰할 수 있었고, 주어진 코드대로 가중치를 줄 경우 각 값마다 출력 색이 확연히 달라지는 점이 특히 인상 깊었다.

2. 단계별 변화 추적

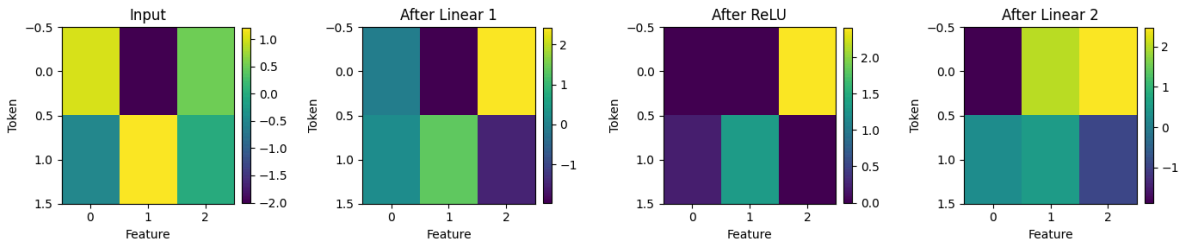
코드를 해석해보면 연산 순서는 선형변환 → ReLU → 또 다른 선형변환의 구조였다. 하지만 시각화만으로는 부족해서 print문을 활용해 내부 값을 직접 확인하며 추적했다. 작은 비선형 함수가 예상보다 큰 변화를 일으킨다는 사실이 흥미로웠다. ReLU는 max(0, x)처럼 항상 0 이상을 출력하는 활성화 함수로 알려져 있지만, 특히 입력이 양수일 때는 가중치가 ReLU를 leverage point처럼 크게 흔든다는 느낌을 받았다. 또한 함수열처럼 각 토큰이 완전히 독립적으로 처리된다는 점도, 알고 나서 다시 보니 꽤 인상 깊고 신기했다.

3. 패턴 설명

W1, W2를 full rank로 구성한 경우 출력의 색 분포가 훨씬 다양해졌고, ReLU를 거친 후에도 값 차이가 뚜렷하게 유지되었다. 반면 W1, W2가 항등 행렬에 가까운 경우 출력이 거의 입력과 비슷했고, bias만이 유일한 변화 요인처럼 보였다. 즉, FFN이 입력을 얼마나 바꾸는지는 가중치 구조와 비선형 함수의 반응에 따라 달라진다고 해석해도 무방하다고 느꼈다.

4. 핵심 개념 요약

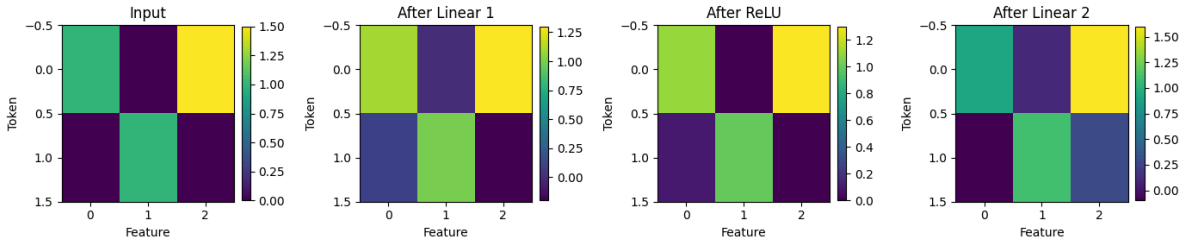
Feed Forward Network(FFN)는 attention 이후 각 토큰을 독립적으로 처리하여 표현을 보완하는 구조라고 생각된다. ReLU는 음수를 제거하는 비선형 함수로 흔히 알려져 있지만, 실험을 통해 입력 분포나 가중치 구조에 따라 예상보다 민감하게 반응한다는 사실이 새로웠다. 특히 W1, W2가 항등행렬이 아닐수록 출력의 색(값) 차이가 뚜렷하게 나타나 FFN이 더 능동적으로 표현을 바꾸는 역할을 했다고 느꼈다. 반면 항등행렬에 가까운 구조에서는 FFN의 효과가 제한적이었고, 그때는 bias가 출력을 주도한다는 점도 흥미로웠다. 이런 결과를 바탕으로, FFN이 어떤 조건에서 더 의미 있는 변화를 유도하는지 더 실험해보고 싶다는 생각이 들었다.



[그림 1] Discord에 주어진 코드의 시각화

```
Input:
 [[ 1.  -2.   0.5]
 [-0.5  1.2  0. ]]
After Linear 1:
 [[-0.1  -1.95  2.4 ]
 [ 0.21  1.34 -1.52]]
After ReLU:
 [[0.   0.   2.4 ]
 [0.21 1.34 0.  ]]
After Linear 2 (Output):
 [[-1.88   2.02   2.46 ]
 [ 0.299  0.573 -0.935]]
```

[그림 2] Discord에 주어진 코드의 결과(ReLU가 음수를 제거하는 역할을 한다.)



[그림 3] 가중치 행렬이 항등행렬일때 시각화(가중치가 영향이 없다보니 편향벡터가 미미하게 영향을 주는 것이 보임.)

```
Input:
 [[1.  0.  1.5]
 [0.  1.  0. ]]
After Linear 1:
 [[ 1.1  0.   1.3]
 [ 0.1  1.  -0.2]]
After ReLU:
 [[1.1 0.  1.3]
 [0.1 1.  0. ]]
After Linear 2 (Output):
 [[ 0.9  0.1  1.6]
 [-0.1  1.1  0.3]]
```

[그림 4] 가중치 행렬이 항등행렬일때의 결과(변화가 크지 않음.)