추가 스터디

왜 0벡터인가?

식(1) 을 이용하여 R^2에서 두 벡터가 선형 독립임을 확인하였고, 선형 독립인 두 벡터가 R^2의 모든 벡터를 생성할 수 있는지 확인하여 선형 벡터 공간의 차원이 몇인지 증명하였습니다. 식(1)에서 왜 0벡터가 되어야 선형 독립인지 궁금했습니다. 벡터가 선형 독립이라는 것은

$a_i = 0$ 일 때 다음 조건을 만족합니다.

$$a_1|1> + a_2|2> + ... + a_n|n> = |0>$$

여기서 $|0>$는 0벡터를 나타내며, 모든 스칼라 a_i가 0일 때만 이 식이 성립합니다. 벡터의 선형 조합이 0벡터가 될 수 있는 방법은 모든 계수가 0인 경우를 의미합니다. 선형 독립 벡터들은 벡터 공간의 기저를 형성하는데 공간의 모든 벡터를 표현할 수 있는 최소 집합입니다. 예를 들어 R^2에서 두 벡터 (1, 0), (0, 1)가 존재할 때 식(1)에 대입 한다면,

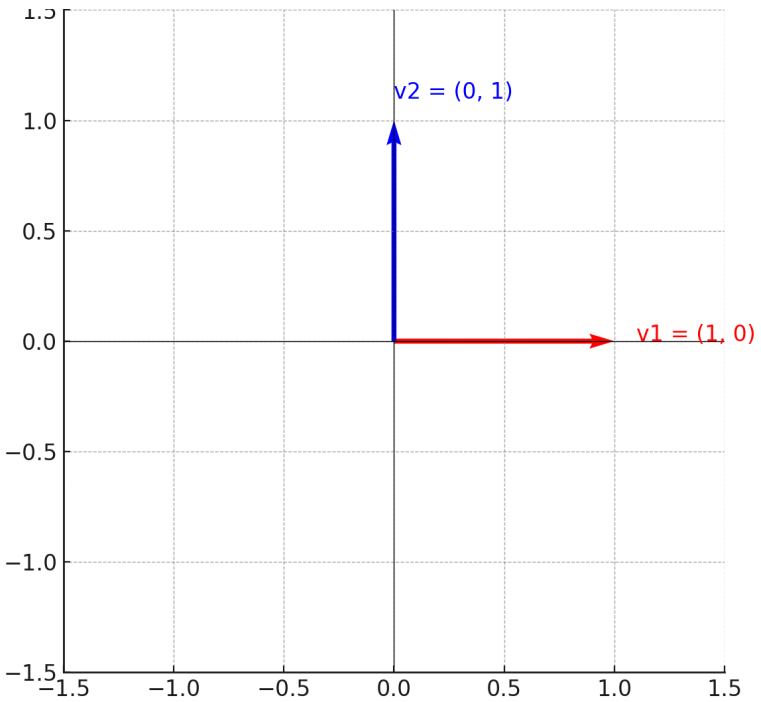
$$a_1(1,0) + a_2(0,1) = (0,0)$$

분배 법칙에 의해 표현이 가능합니다.

$$a_1 \cdot 1 + a_2 \cdot 0 = 0$$
$$a_1 \cdot 0 + a_2 \cdot 1 = 0$$

$$a_1 = 0$$
$$a_2 = 0$$

따라서, a_1과 a_2가 모두 0일 때 이 식이 성립하며 두 벡터는 선형 독립임을 의미합니다. 이는 아래 그림과 같이 각 벡터가 서로 다른 방향을 가리키며 이 벡터들이 중복되지 않는다는 것을 깨달았습니다.



선형벡터공간 활용

데이터의 벡터화, 특징 추출, 모델 학습, 신경망의 선형 변환, 거리와 유사도 측정 등 여러 과정에서 선형 벡터 공간의 개념이 사용되는데 지식 추적 알고리즘에도 사용될 수 있는 것을 배웠습니다. 예를 들어 DKT(Deep Knowledge Tracing)이 있는데 학습자의 퍼포먼스를 바탕으로 학습자 전체 지식 수준을 평가하는 모델입니다. 학습자의 지식을 추적하기 위해 RNN / LSTM / GRU 알고리즘을 활용합니다.

DKT 모델에서는 기본적으로 문제 정답여부에 대한 데이터를 통해 다음 문제에 대한 정답 여부를 추론하는 데이터를 가장 많이 사용하기 때문에, 문제의 종류와 문제 정답 여부 등의 속성을 원 핫 인코딩으로 다루는 경우가 많습니다.

🔥 원 핫 인코딩

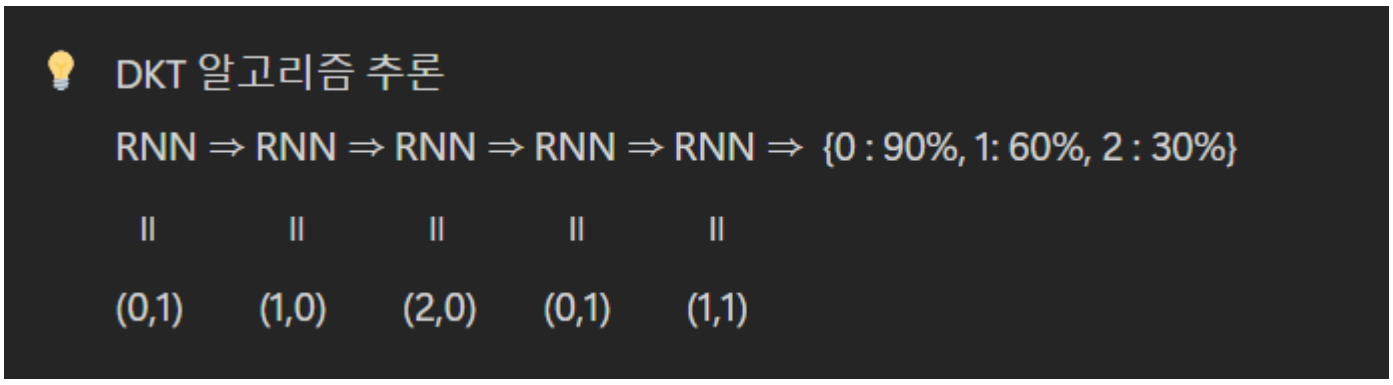
덧셈 (O) 0 0 0 0 1 0 0 0 / 덧셈 (X) 1 0 0 0 0 0 0 0

뺄셈 (O) 0 0 0 0 0 1 0 0 / 뺄셈 (X) 0 1 0 0 0 0 0 0

곱셈 (O) 0 0 0 0 0 0 1 0 / 곱셈(X) 0 0 1 0 0 0 0 0

나눗셈 (O) 0 0 0 0 0 0 0 1 / 나눗셈 (X) 0 0 0 1 0 0 0 0

위 예시처럼 연산 문제를 맞고 틀린 상황을 표현할 수 있습니다. 하지만 문제 수가 늘어날 때마다 벡터값이 기하급수적으로 늘어나 효율적인 표기는 아닌 것 같습니다.



학습자가 각 문제에 대해 (문제ID, 정오답) 형태의 벡터로 표현하고 RNN 입력으로 사용됩니다. 마지막 은닉층을 바탕으로 문제의 정답 확률을 예측합니다. 각 단계별 순차적으로 학습을 진행하는데, 전 단계에서 추론한 다음 문항에 대한 정답률과 실제 정답과의 오차가 최소가 되도록 오차 역전파를 합니다.

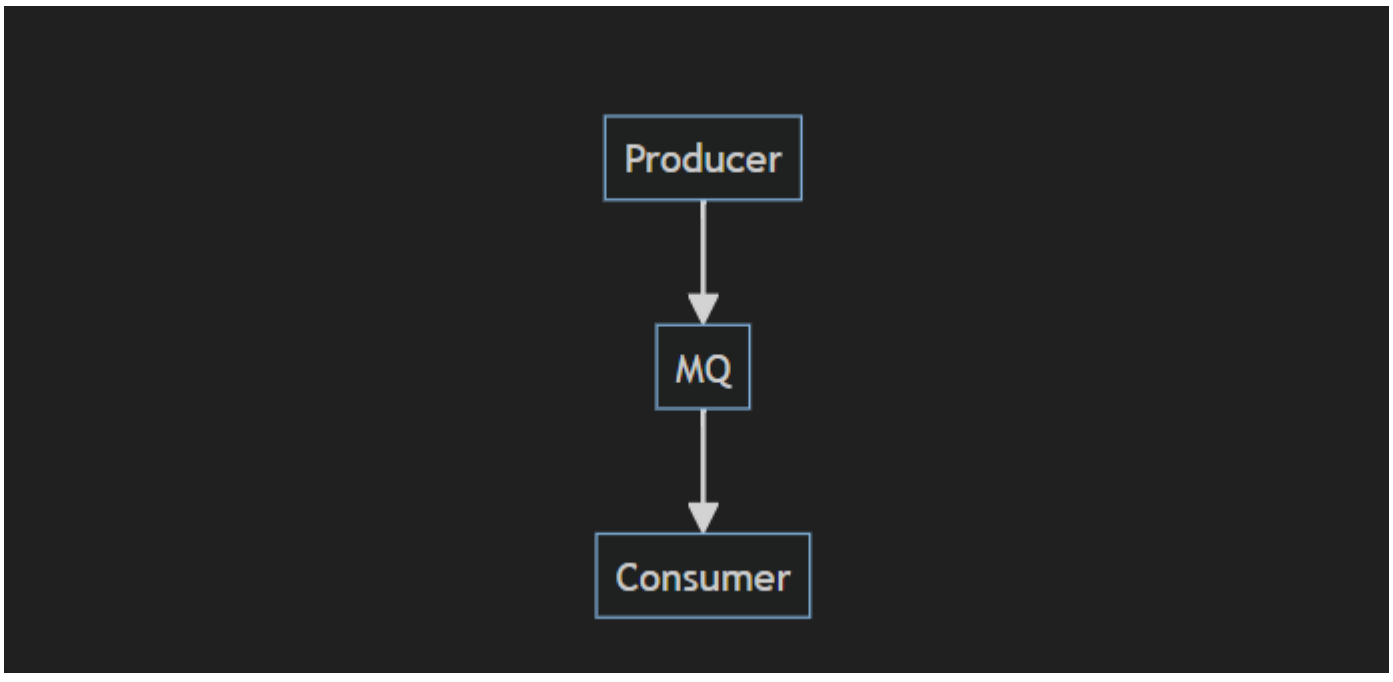
- 오차 역전파(Backpropagation) : 시작 단계에서 난수의 가중치, 임계값으로 한 학습이 끝나면 정답과 비교해 오차를 최소화하는 방향, 역으로 가중치, 임계값을 조정
- 인공신경망의 학습 방향은 출력(가중지, 임계값 조절) → 은닉층처리(가중치, 임계값 조절)의 역방향으로 진행

MQ

데이터를 크롤링하거나 정제된 데이터를 활용하여 처리한 경험이 있지만 서버에서 데이터가 어떻게 처리되는지 직접적으로 알지 못하였습니다. 세부적으로 내용을 살펴보기 전에 큰 틀에서 학습을 진행하여 과제를 제출했습니다. 하지만 상세한 사례를 알아보며 놓쳤던 부분을 확인하고 싶었습니다.

MQ 이전에 MOM(Message Oriented Middleware), 독립된 서비스 간에 데이터를 주고받을 수 있는 형태의 미들웨어가 있는데, 이는 여러 분산되어 있는 시스템 간의 결합성을 낮추고, 실시간으로 비동기식 데이터를 교환할 수 있도록 하는 소프트웨어입니다.

MQ를 사용하는 이유



P에서 C로 데이터를 전송할 때, MQ로 데이터를 전달합니다. C는 MQ로부터 요청 데이터를 수신하여 처리하게 되고 C가 장애 상황이라면 C가 받기 전까지 MQ에 남아있게 됩니다. 장애 복구 이후엔 다시 MQ를 통해 데이터를 수신해 처리가 가능합니다. MQ 덕분에 데이터 파이프라인이 간소화 되고 관리하기 편해지는 효과가 있습니다. 또한 여러 C에 대해 분산 처리도 가능합니다. 서버 처리량에 맞게 태스크를 가져와 처리함으로써 분산 처리도 가능하게 됩니다. 이를 통해 부하가 많은 작업도 여러 서버가 나누어서 작업하며 동시에 처리가 가능합니다. MQ는 생산된 메시지의 저장, 전송에 대해 큐에 메시지를 생산할 뿐이므로 부가적인 작업을 처리할 때까지 기다릴 필요가 없어 핵심 로직에 집중할 수 있습니다.

단점으로는 메시지 큐를 운영하고 관리하기 위한 시스템이 필요하여 운영 및 관리 비용이 늘어나고 전체적인 시스템 구조가 복잡해질 수 있어 오버헤드가 발생할 수 있습니다. MQ에 장애가 발생하여 다운 된다면 MQ에 메시지를 발행하고 컨슘하는 모든 시스템 또한 다운됩니다. 큐에서 메시지를 컨슘하여 DB를 업데이트하는 과정에서 다른 Consumer가 업데이트에 실패하면 데이터 일관성 문제가 발생할 수 있습니다. 따라서 MQ에 대한 의존성이 높아질 수 있습니다

MQ에 대해 상세히 알아보면서 Kafka 외에도 RabbitMQ, ActiveMQ 등 다양한 오픈소스 메시지 큐가 있다는 것을 알게 되었습니다. 추후 다양한 MQ를 학습해보고, 각각의 장단점을 비교해 실무에 적용할 수 있는 방안을 모색하고 싶습니다.