목차

SLP와 MLP 코드 비교

MLP에 무엇이 추가되었는가

1. Hidden Layer 추가

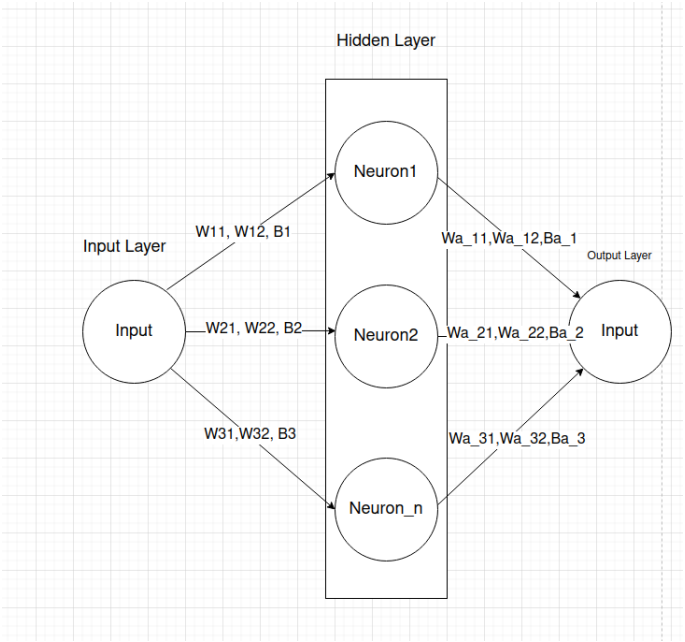
```
class MLP_XOR_with_Backpropagation:
    def __init__(self, input_size=2, hidden_size=8, output_size=1, learning_rate=0.05):
        self.input_size = input_size
```

- Hidden Layer가 추가되었다
- 해당 코드에서 `hidden_size`는 Hidden Layer에 존재하는 뉴런의 개수를 의미함
- 뉴런 1개당 SLP 하나와 상동하다

2. Hidden Layer를 위한 Weight

```
# 은닉층 가중치: (입력 크기, 은닉층 크기)
self.weights_input_hidden = np.random.uniform(-1, 1, (self.input_size, self.hidden_size))
# 은닉층 바이어스: (1, 은닉층 크기)
self.bias_hidden = np.random.uniform(-1, 1, (1, self.hidden_size))
```

- 기존 SLP는 입력층 하나에 출력층 하나로 연결되어 가중치와 바이어스 set가 1set만 존재했다
- MLP에서는 Hidden Layer의 뉴런 개수만큼 가중치-바이어스 set가 필요하다



- 각 뉴런으로 향하는 라인마다 가중치와 바이어스를 달리 줌
- 행렬 연산!! --> w11, w12, w21, w22, ...

AI를 위한 하드웨어가 필요한 이유🌟🌟🌟

- 행렬곱과 같이 병렬 데이터를 한번에 처리하기 위해서는 전용 HW가 필요하다
- CPU는 1번에 하나의 연산 밖에 수행 못함
- 10X10 행렬곱을 처리하면 CPU는 100번의 연산 진행해야함
- HW는 한번에 처리 가능

3. Back-Propagation

- 기존 SLP는 활성화함수를 거치고 나온 오차로 weight와 bias를 update했음
- MLP에서 Hidden Layer의 각 Neuron들에게 할당된 weight와 bias update 필요함
    - 뉴런하나마다 나오는 오차를 가지고 update를 하기에는 cost가 많이 듦
        - 각 뉴런별 에러를 가지고 각자 update를 진행해야함
        - 시간비용이 너무 많이 듦
- 출력층에서 나오는 최종 출력에 대한 오차만을 가지고 모든 뉴런들의 가중치와 바이어스를 업데이트 해줌
    - 시간 비용을 줄일 수 있다

> Back Propagation이 없으면 시간이 얼마나 오래 걸릴까?
>
> > 초당 8억번 계산하는 CPU가 28x28 픽셀을 역전파없이 계산하면
> >
> > > 약 123시간 정도 걸림

출처: 신박사AI-역전파 backpropagation

역전파에서 Update하는 방법

```
    # 역전파 (Backward Propagation) 및 가중치 업데이트
    def backward(self, x, y, output):
        # 출력층 에러 계산
```

```
        output_error = y - output
        output_delta = output_error * self.sigmoid_derivative(output)

        # 은닉층 에러 계산
        hidden_error = np.dot(output_delta, self.weights_hidden_output.T)
        hidden_delta = hidden_error *
self.sigmoid_derivative(self.hidden_output)

        # 가중치 및 바이어스 업데이트
        self.weights_hidden_output += np.dot(self.hidden_output.T,
output_delta) * self.learning_rate
        self.bias_output += np.sum(output_delta, axis=0, keepdims=True) *
self.learning_rate

        self.weights_input_hidden += np.dot(x.T, hidden_delta) *
self.learning_rate
        self.bias_hidden += np.sum(hidden_delta, axis=0, keepdims=True) *
self.learning_rate
```

경사하강법

- 경사하강법
  - 함수의 기울기를 구함
  - 기울기의 경사에 반대방향으로 값을 계속 이동시킴
  - 극값에 이를때까지(최저점) 반복
- 손실함수
  - 예측치와 실제값의 오차를 정량적으로 측정하는 함수
  - 이 손실함수의 극소값이 도달할때까지 경사하강법 반복

> 경사하강법을 사용하면 빠르게 손실함수의 최저점에 도달할 수 있다

- 손실함수의 최저점 == 오차가 가장 적은 부분

Delta value

- 출력층에서 나온 오차에 각 뉴런들이 얼마나 영향을 줬는지 파악 필요
  - 이를 반영해 각 뉴런들의 가중치와 바이어스를 업데이트 해줘야한다
- 활성화 함수를 거쳐 나온 출력에 대한 오차에
  - 활성화함수의 기울기(== 도함수)를 곱하면
  - 해당 출력이 오차에 얼마나 민감하게 영향을 주는지 확인할 수 있다
- Delta값이 클수록, 해당 뉴런이 오차에 영향을 많이 준 것이다
- 본 코드에서 `output_delta`는
  - 출력층의 오차만을 의미
- `hidden_delta`는
  - 은닉층의 각 뉴런이 출력층의 오차에 얼마나 영향이 있는지를 보여준다

> 정확한 이해를 위해서는 `Chain Rule`을 이해햐아한다(추후 업데이트 예정)

4. 연산 단계의 차이

- SLP
    1. 순전파
    2. 오차계산
    3. Weight, Bias 업데이트
- MLP
    1. 순전파
    2. 오차계산
    3. **역전파**🌟
    4. Weight, Bias 업데이트

실험 - XOR: 여러개의 SLP 이용 Vs MLP 사용(역전파 유무 차이)

정의

1. 여러개의 SLP를 이용

- XOR: AxorB = AND(NAND(A, B), OR(A, B))
- NAND, OR, AND에 대해 각각 단층 perceptron을 만듦
- 각 뉴런에 대해 따로 가중치와 bias를 update
    - SLP 방식으로 update
    - 역전파 사용X
- NAND, OR의 출력을 AND Perceptron의 입력값으로 넣음

2. MLP 사용
    - 기능이 정의된(NAND, OR, AND) SLP를 사용하는 대신
    - Hidden Layer에 임의의 개수 뉴런과 임의의 가중치-바이어스 set를 할당
    - 가중치-바이어스 업데이트는 역전파 방식 이용

비교

역전파 없는 경우

```
Average prediction time per sample: 0.000036182 seconds
```

- 샘플 당 평균 예측시간
    - 약 36us

역전파를 사용하는 경우

```
Average prediction time per sample: 0.000017115 seconds
```

- 샘플 당 평균 예측시간
    - 약 17us

결과

- 역전파를 사용한 경우가 아닌 경우에 대해 약 2배넘게 빠른 속도를 보임
- 뉴런 각각을 업데이트한 후 병합하는 것보다 역전파를 이용해 학습과정에서 출력층 오차만으로 전체 뉴런을 업데이트 하는 것이 더 효율적

주의

- MLP의 경우 Error가 발생하지 않는 순간까지 시간을 측정한 것이 아님
    - 주어진 epochs만큼 돌리는데 걸리는 시간을 체크한 것
- Error가 발생하지않는 순간을 기준으로 하면 더 빠른 연산속도를 가짐을 유추가능