

PART 1

단층 퍼셉트론(SLP)

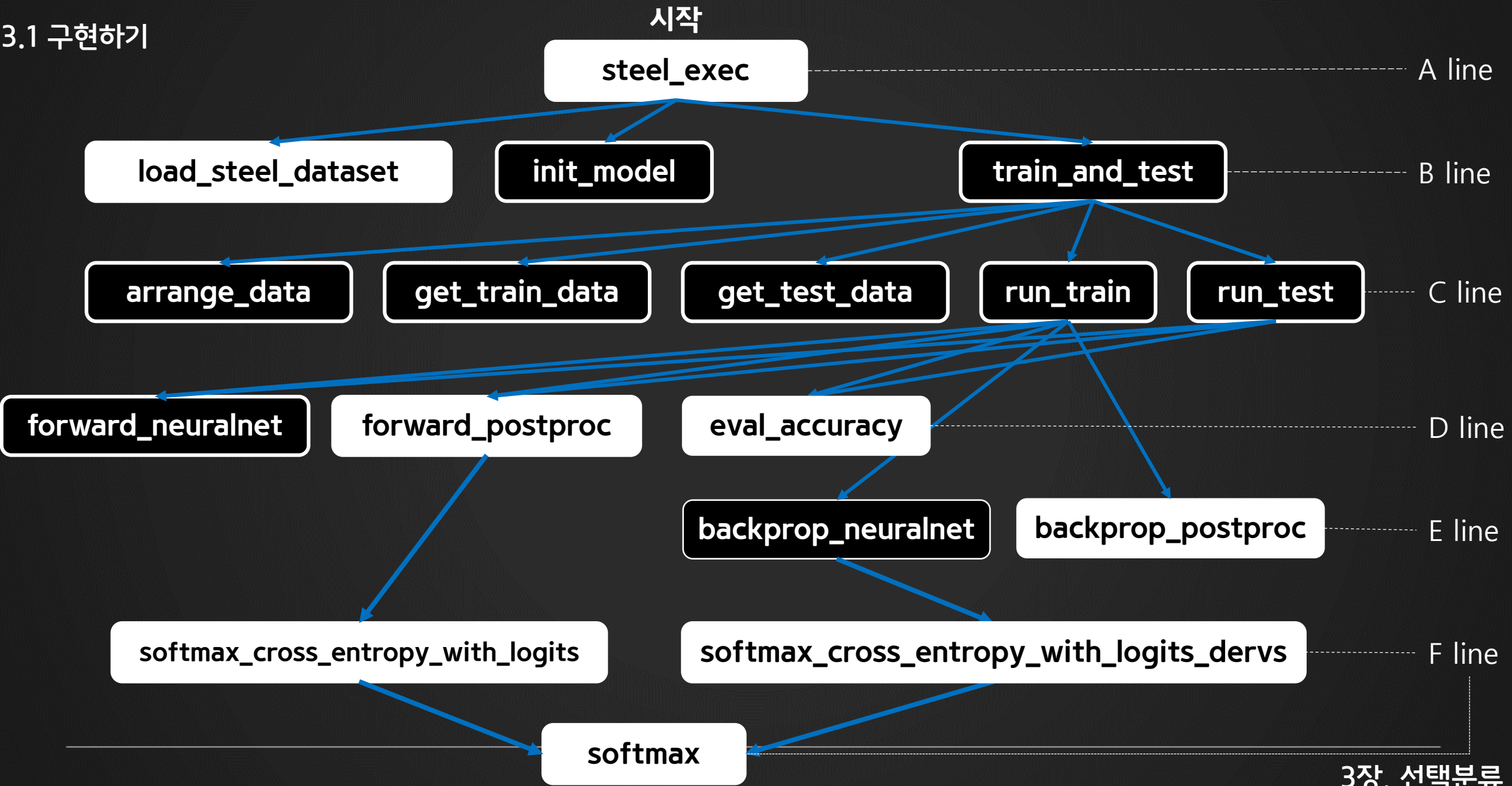
1장 회귀분석

2장 이진판단

3장 선택분류

딥러닝 & 강화학습 담당
이재화 강사

3.1 구현하기



3.0 코드 재활용을 위한 이전 파일 실행

```
%run ../../leeyua/AI_CODE/AI_abalone.ipynb
```

3.1 메인 함수 정의

```
def steel_exec(epoch_count=10, mb_size = 10, report =1):  
    load_steel_dataset()  
    init_model()  
    train_and_test(epoch_count,mb_size,report)
```

이전 수업과 동일한 과정

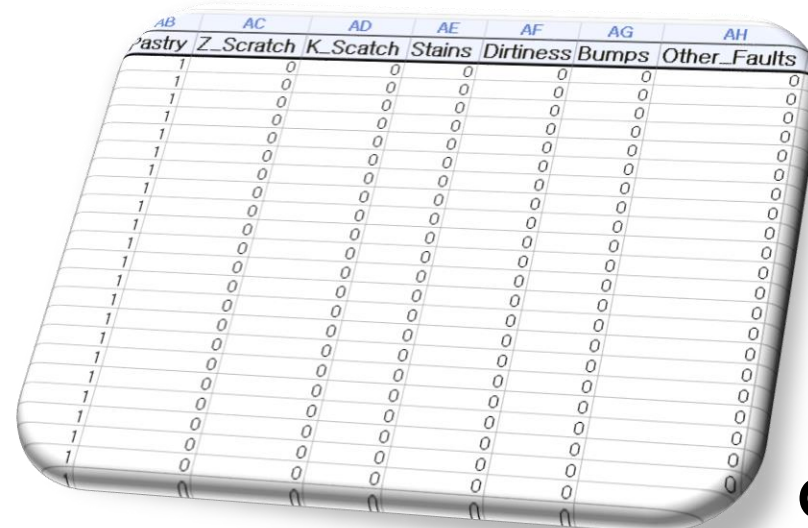
3.2 데이터 적재 함수 정의

```
def load_steel_dataset():  
    with open('faults.csv') as csvfile:  
        csvreader = csv.reader(csvfile)  
        next(csvreader, None)  
        rows = []  
        for row in csvreader:  
            rows.append(row)  
  
    global data, input_cnt, output_cnt  
  
    input_cnt, output_cnt = 27, 7  
    data = np.asarray(rows, dtype='float32')
```

선택분류에 알맞도록
출력 벡터 크기를 분류 항목 수 인 7로 지정

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	42	50	2538079	2538108	270944	108	17	42	24220	11397	0	0	0	0	0	0	0	0	0	0	0	0	0
2	646	651	1563913	1563931	360415	2409	20	432	20007	7972	0	0	0	0	0	0	0	0	0	0	0	0	0
3	829	860	369370	498335	630	230	20	432	20007	7972	0	0	0	0	0	0	0	0	0	0	0	0	0
4	853	1306	498078	100337	9952	11	26	432	20007	7972	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1299	441	138468	210956	138883	132	15	167	60393	31062	0	0	0	0	0	0	0	0	0	0	0	0	0
6	430	440	210956	429227	429253	1506	15	69	61966	31062	0	0	0	0	0	0	0	0	0	0	0	0	0
7	413	200	429227	779008	442	42	64	38917	61966	31062	0	0	0	0	0	0	0	0	0	0	0	0	0
8	190	343	779144	813600	284	15	60	69258	119540	61966	0	0	0	0	0	0	0	0	0	0	0	0	0
9	330	90	813452	106668	480	22	68	119540	61966	31062	0	0	0	0	0	0	0	0	0	0	0	0	0
10	74	118	106668	179312	435	30	133	670911	59760	14807	0	0	0	0	0	0	0	0	0	0	0	0	0
11	106	515	179268	230704	728	59	282	59760	14807	24807	0	0	0	0	0	0	0	0	0	0	0	0	0
12	505	58	230644	368208	1097	167	76	14807	24807	32604	0	0	0	0	0	0	0	0	0	0	0	0	0
13	46	590	368143	491684	5044	38	26	32604	17753	13829	0	0	0	0	0	0	0	0	0	0	0	0	0
14	581	466	491552	714055	552	8	27	17753	13829	32175	0	0	0	0	0	0	0	0	0	0	0	0	0
15	451	684	713788	751132	137	15	34	32175	27349	16645	0	0	0	0	0	0	0	0	0	0	0	0	0
16	689	192	751069	844729	209	18	29	16645	38722	8948	0	0	0	0	0	0	0	0	0	0	0	0	0
17	156	104	844704	21376	284	13	19	38722	8948	368381	0	0	0	0	0	0	0	0	0	0	0	0	0
18	90	99	21376	42717	153	10	39	8948	368381	36875	0	0	0	0	0	0	0	0	0	0	0	0	0
19	82	1613	42683	88313	106	15	35	36875	20751	28738	0	0	0	0	0	0	0	0	0	0	0	0	0
20	1601	28	86284	115504	264	17	23	28738	20751	28738	0	0	0	0	0	0	0	0	0	0	0	0	0
21	17	52	115485	149083	201	9	39	20751	28738	28738	0	0	0	0	0	0	0	0	0	0	0	0	0
22	43	72	149044	184389	171	19	18	28738	20751	28738	0	0	0	0	0	0	0	0	0	0	0	0	0
23	65	92	184350	2129907	395	9	253	20751	28738	28738	0	0	0	0	0	0	0	0	0	0	0	0	0
24	82	84	2128884	2356435	85	92	38	28738	20751	28738	0	0	0	0	0	0	0	0	0	0	0	0	0
25	17	1372	2356396	5202324	3214	15	33	28738	20751	28738	0	0	0	0	0	0	0	0	0	0	0	0	0
26	1383	1372	5202306	4332815	386	9	63	28738	20751	28738	0	0	0	0	0	0	0	0	0	0	0	0	0
27	1358	1410	4332815	4822354	192	19	2	28738	20751	28738	0	0	0	0	0	0	0	0	0	0	0	0	0
28	1404	1302	4822354	1874415	527	19	2	28738	20751	28738	0	0	0	0	0	0	0	0	0	0	0	0	0
29	1281	52	1874382	2378236	150	19	2	28738	20751	28738	0	0	0	0	0	0	0	0	0	0	0	0	0
30	49	837	2378173	2378173	2378173	2378173	2378173	2378173	2378173	2378173	0	0	0	0	0	0	0	0	0	0	0	0	0
31	830	104	2378173	2378173	2378173	2378173	2378173	2378173	2378173	2378173	0	0	0	0	0	0	0	0	0	0	0	0	0
32	91	2378173	2378173	2378173	2378173	2378173	2378173	2378173	2378173	2378173	0	0	0	0	0	0	0	0	0	0	0	0	0

※ 27가지 특징값
steel_dataset



	AB	AC	AD	AE	AF	AG	AH
1	1	0	0	0	0	0	0
2	0	1	0	0	0	0	0
3	0	0	1	0	0	0	0
4	0	0	0	1	0	0	0
5	0	0	0	0	1	0	0
6	0	0	0	0	0	1	0
7	0	0	0	0	0	0	1

※ 7가지 철판 불량 상태
one_hot vector

3.3 후처리 과정에 대한 순전파와 역전파 함수 재정의

#손실함수를 계산해 순전파 과정을 마무리 짓는 함수

```
def forward_postproc(output, y):  
    entropy = softmax_cross_entropy_with_logits(y, output)  
    loss = np.mean(entropy) #범주별 entropy값에 대한 평균  
    return loss, [y, output, entropy] #loss 반환 및 역전파에 사용될 보조정보 반환
```


3.3 후처리 과정에 대한 순전파와 역전파 함수 재정의

#신경망에 대한 출력에 대한 손실 기울기를 계산해 역전파 과정을 시작 하는 함수

```
def backprop_postproc(G_loss, aux):
    y, output, entropy = aux #리스트 타입의 보조정보를 각 변수로 독립

    g_loss_entropy = 1.0 / np.prod(entropy.shape) #  $\frac{\partial L}{\partial L}$  와  $G_{entropy}$  사이의 부분기울기
    g_entropy_output = softmax_cross_entropy_with_logits_derv(y, output)

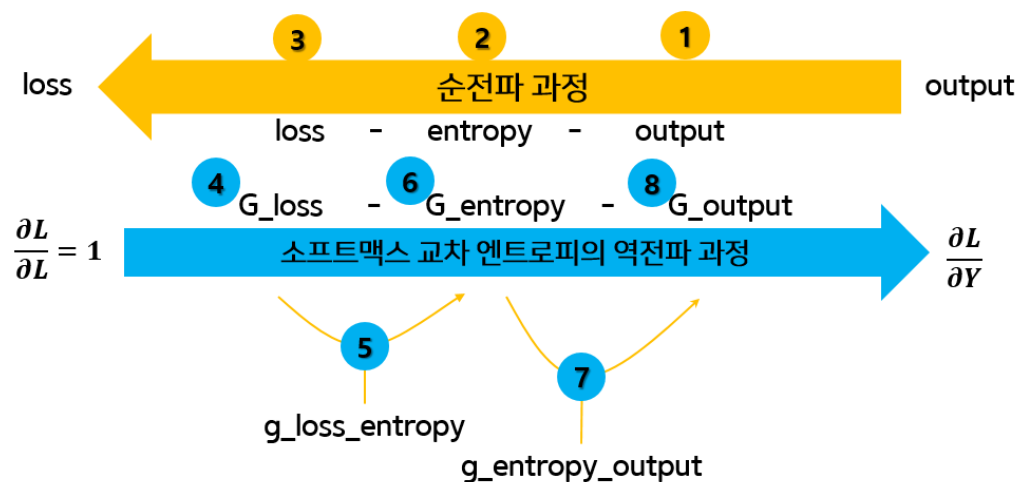
    G_entropy = g_loss_entropy * G_loss
    G_output = g_entropy_output * G_entropy

    return G_output # $G_{entropy}$  (손실함수에  $Y$ 의 편미분)
```

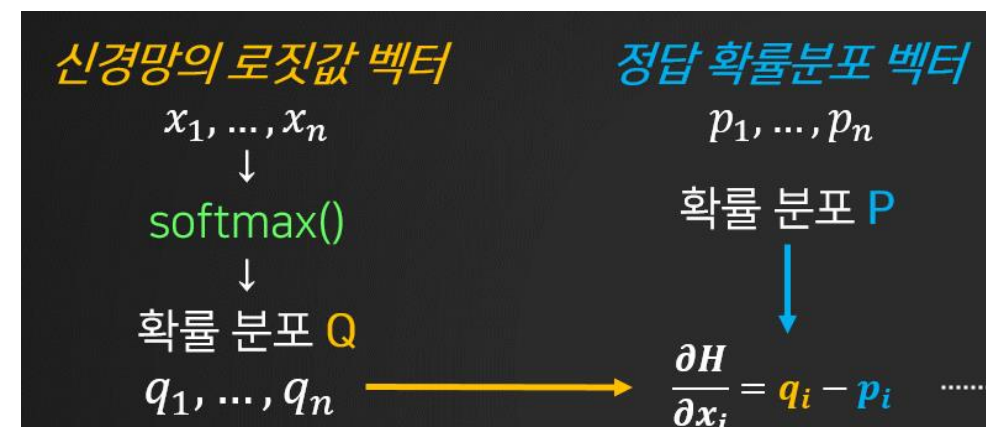
소프트맥스 교차 엔트로피의 역전파 처리

순전파의 역순으로 loss / entropy / output 에 대한 역전파 처리를 차례로 수행

```
def backprop postproc(G loss, aux):
```



#각 단계의 입출력간 손실 기울기



#소프트맥스 교차 엔트로피의 편미분 과정.

실제 정답에 의해 표현되는 확률분포 p 즉 y 값과,
신경망에 의한 추정확률분포 q , 즉 $output$ 을 활용하여
손실기울기를 구하는 함수입니다.

3.4 정확도 계산 함수의 재정의

```
def eval_accuracy(output, y):  
    estimate = np.argmax(output, axis = 1)  
    answer = np.argmax(y, axis = 1)  
    correct = np.equal(estimate, answer)  
  
    return np.mean(correct)
```

#신경망 추정에서의 선택결과는 np.argmax()를 이용해
후보 항목에 관한 각 로짓값을 담고 있는 output 벡터에서
가장 큰 값이 어디에 있는지 조사

np.argmax()에 중심선 기준을 1, 즉 y축으로 맞춰
신경망이 출력한 값 중 가장 큰 값의 인덱스를 반환.
그리고 그 값을 'estimate'이라는 객체에 할당 및 저장.

※이때 확률 분포로 변환해도 로짓값의 대소관계는 그대로 유지.
굳이 소프트맥스 함수를 불러 로짓값 벡터를 확률분포로
변환할 필요가 없음

정답 데이터에 대해 가장 큰 값을 'answer' 이라는 객체에 넣어
신경망의 추정치와 비교할 수 있도록 준비.

#서로 비교하며 불리언 타입으로 결과를 저장해 주도록 합니다.

#그리고 결과값과 추정치의 선택 일치 여부를 따져
일치한 비율을 정확도로 보고해 주도록 합니다.

3.5 소프트맥스 관련 함수 정의(1)

#소프트맥스와 관련된 함수들을 정의하는 순서
(소프트맥스 변형식)

#데이터에 담긴 여러 데이터에 대한 벡터들을 담은 행렬을 처리 대상

```
def softmax(x):  
    max_elem = np.max(x, axis=1)  
    diff = (x.transpose() - max_elem).transpose()  
  
    exp = np.exp(diff)  
    sum_exp = np.sum(exp, axis=1)  
    probs = (exp.transpose() / sum_exp).transpose()  
  
    return probs
```

#소프트 맥스 변형식에 따라 입력으로 들어올
각 연산결과에 대해 가장 높은 값을 선택하여 저장

#i번째 x와 가장 큰 x값을 빼주는 과정

#transpose()를 적용한 이유 :
넘파이의 행렬연산에 있어 위에서 도출한 'max_elem'이
현재 '열 기준'으로 정렬.
열에 대한 반복 감산을 진행하기 위해서는
기존 x값의 행렬전환 시켜준 후 '감산' 을 진행하고
다시 행렬전환으로 원상복귀


이렇게 구한 차이에 지수함수 취하기

#이 부분은 분모가 되는 부분.
np.sum()을 통해 모든 값을 합산

#소프트맥스 변형식 처럼 식을 구축하기 위해
분모와 분자를 위치. 이때도 아까와 같은 이유때문에
행렬전환을 두번 진행

$$y_i = \frac{e^{x_i}}{e^{x_1} + e^{x_2} + \dots + e^{x_n}}$$

(일반식 : 계산과정에서 오류 발생)


$$y_i = \frac{e^{x_i - x_k}}{e^{x_1 - x_k} + e^{x_2 - x_k} + \dots + e^{x_n - x_k}}$$

(변형식)

3.5 소프트맥스 관련 함수 정의(2)

#소프트맥스 교차 엔트로피 변형식 구현부분.

```
def softmax_cross_entropy_with_logits(labels, logits):  
    probs = softmax(logits)
```

#퍼셉트론 연산을 통과한 신경망의 추정치 로짓값을
소프트맥스 함수로 통과시키켜 각 확률분포 값으로 저장

```
    return -np.sum(labels * np.log(probs + 1.0e-10), axis=1)
```

#신경망의 추정치에 대한 확률분포값에 계산 폭주를 막기위한 방법으로
아주 작은 엡실론값을 더해주도록 합니다.

$$H(P, Q) = - \sum p_i \log q_i \approx H(P, Q) = - \sum p_i \log(q_i + \epsilon)$$

아주 작은 값을 더해줌으로써 문제 해결
※ 미미한 존재로 별다른 영향을 주지 않는다.
※ 0에 매우 가까운 하한선 역할을 수행

#소프트맥스 교차 엔트로피 편미분 부분입니다.

```
def softmax_cross_entropy_with_logits_derv(labels, logits):  
    return softmax(logits) - labels
```

#신경망이 추정한 확률분포 값들에 단순히 정답에 의한 확률분포값들을
감산해주는 것만으로 간단하게 편미분을 구해낼 수 있게 됩니다.

신경망의 로짓값 벡터

x_1, \dots, x_n



softmax()



확률 분포 Q

q_1, \dots, q_n

정답 확률분포 벡터

p_1, \dots, p_n

확률 분포 P



$\frac{\partial H}{\partial x_i} = q_i - p_i \dots\dots\dots$



3.6 실행하기

`steel_exec()`

```
Epoch 1: loss=15.984, accuracy=0.306/0.320
Epoch 2: loss=15.509, accuracy=0.326/0.197
Epoch 3: loss=15.984, accuracy=0.306/0.348
Epoch 4: loss=15.004, accuracy=0.348/0.197
Epoch 5: loss=15.286, accuracy=0.336/0.202
Epoch 6: loss=15.390, accuracy=0.332/0.440
Epoch 7: loss=15.509, accuracy=0.326/0.442
Epoch 8: loss=15.628, accuracy=0.321/0.455
Epoch 9: loss=15.360, accuracy=0.333/0.322
Epoch 10: loss=15.316, accuracy=0.335/0.455
```

Final Test: final accuracy = 0.455

`LEARNING_RATE = 0.0001`

`steel_exec()`

```
Epoch 1: loss=16.471, accuracy=0.284/0.110
Epoch 2: loss=15.479, accuracy=0.328/0.414
Epoch 3: loss=15.509, accuracy=0.326/0.402
Epoch 4: loss=15.316, accuracy=0.335/0.432
Epoch 5: loss=15.479, accuracy=0.328/0.338
Epoch 6: loss=14.796, accuracy=0.357/0.332
Epoch 7: loss=15.346, accuracy=0.334/0.215
Epoch 8: loss=15.524, accuracy=0.326/0.164
Epoch 9: loss=15.375, accuracy=0.332/0.281
Epoch 10: loss=15.331, accuracy=0.334/0.189
```

Final Test: final accuracy = 0.189

낮은 품질의 신경망 모델에 대한 원인으로 자주 손꼽히는 원인은 '데이터의 양'.

딥러닝에서는 학습데이터로부터 입력 벡터의 분포 특성을 파악해, 이 특성을 분류에 활용.

즉 27차원 벡터 공간을 7개 구역으로 적절히 분할 할 수 있어야 한다는 의미.

인간은 감히 상상할수 없는 공간, 이러한 초 고차원의 공간을 7개 구역으로 나누기 위해서는 적어도 약 2천여개의 데이터는 부족하다는 의미.

또 다른 이유로는 생각보다 신경망 구조가 너무 단순해서 일 수도 있습니다.

현재 퍼셉트론의 수는 7개로 늘어났지만 층 자체는 단층이기에 이후 층을 깊게 쌓아보며 다시 이 문제를 풀어도록 하겠습니다.