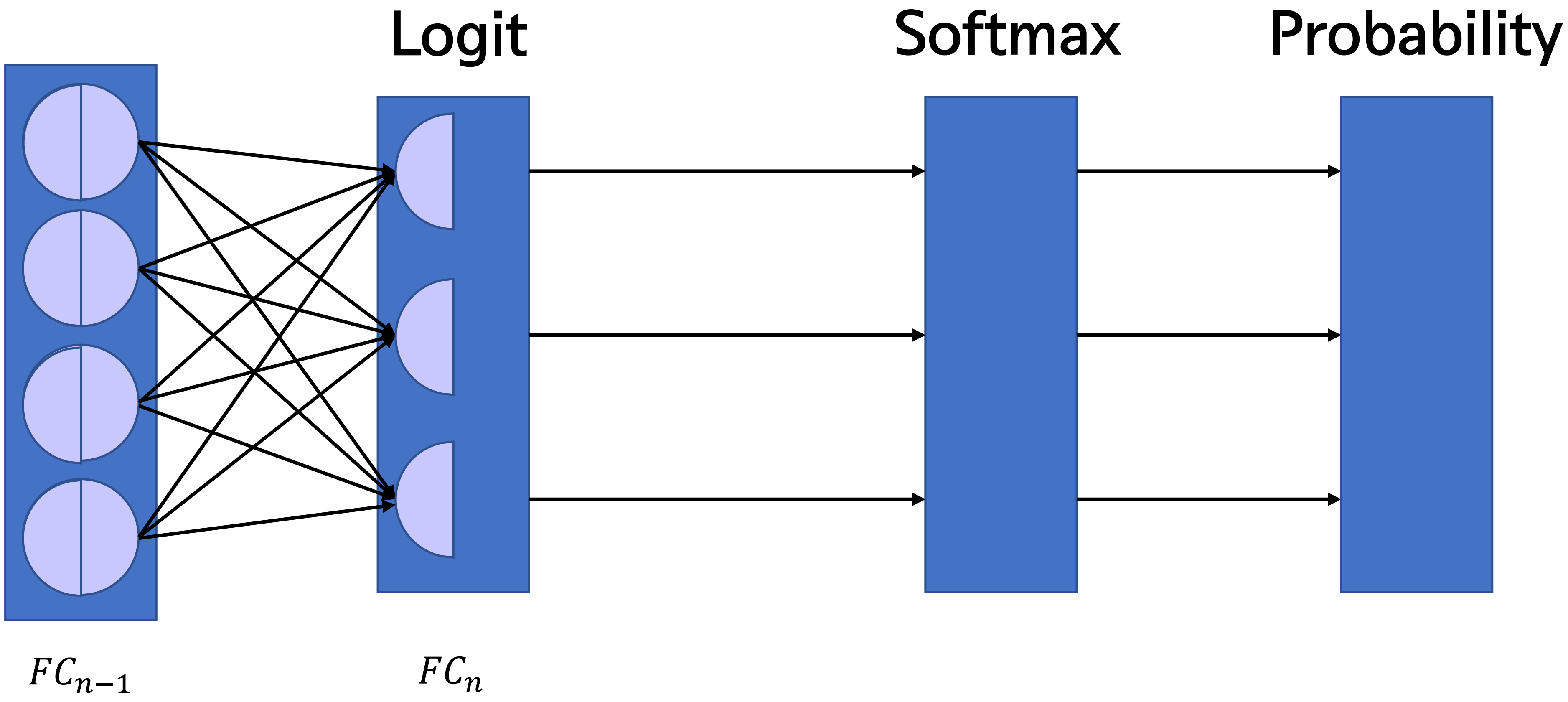
Deep Byun

양혁진

양근제

나준영

01 Logit과 Odds와 Probability의 차이점



01 Odds

정의

어떤 사건이 일어날 확률과 일어나지 않을 **확률의 비**를 수식으로 표현한 것

역할

두 확률의 비교 및 비율 표현

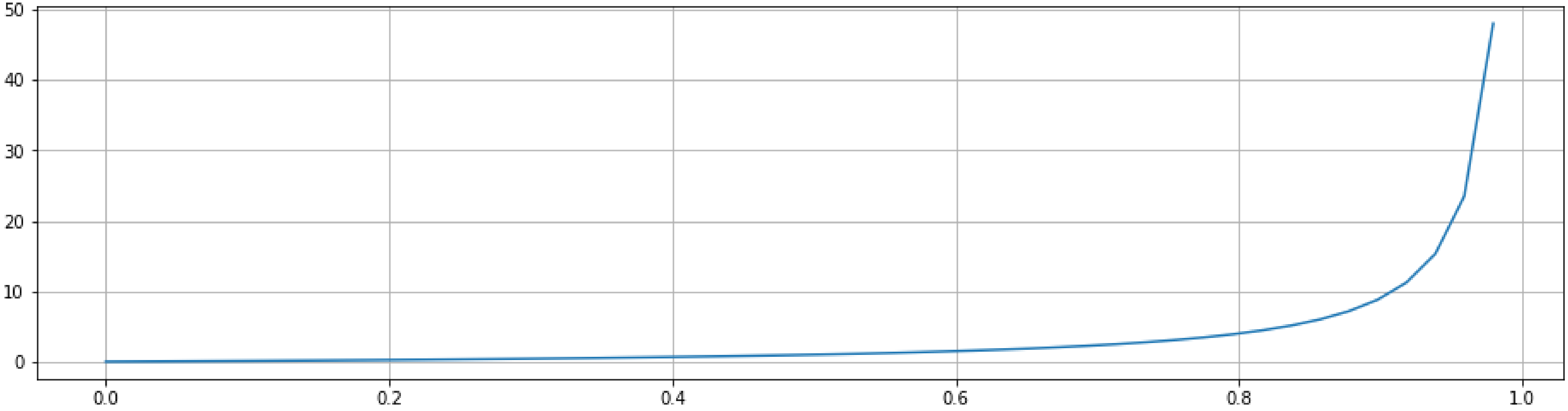
$$\frac{P(a)}{P(b)}$$

01 Odds

수식

$$\frac{P(x)}{1 - P(x)} = \frac{\text{이길 확률}}{\text{질 확률}} \qquad \frac{P(x_i)}{P(x_k)} = \frac{\text{타겟 확률}}{\text{특정 확률}}$$

Odds Graph : $\frac{P(x)}{1-P(x)}$



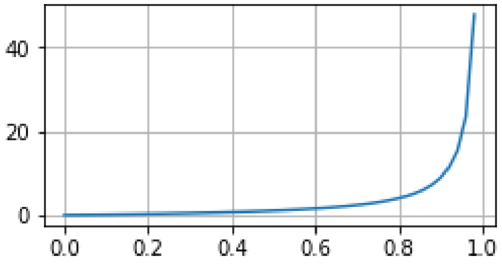
01 Logit

정의

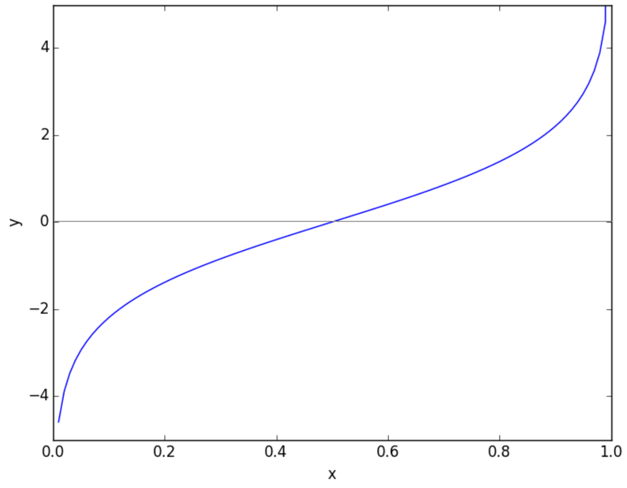
odds에 log를 붙여 확률의 비율을 표현하는 방법 **log + Odds**

역할

[0, ∞)의 범위를 가진 odd에 Log를 취하여 (- ∞, ∞)의 범위를 만들어 내는 역할

log + 

=



01 Logit

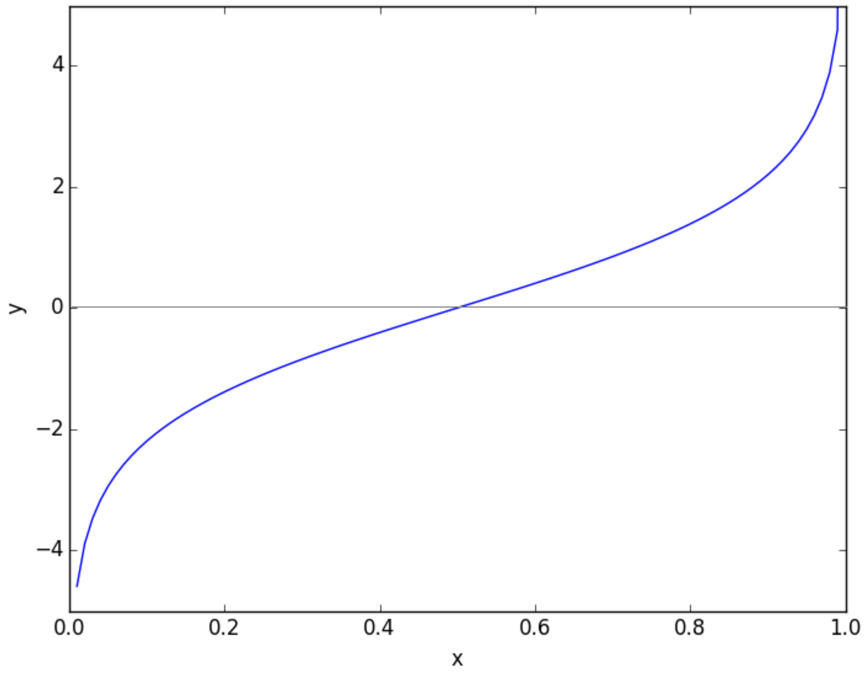
수식

$$\log \frac{P(x)}{1 - P(x)} = \log \frac{\text{이길 확률}}{\text{질 확률}}$$

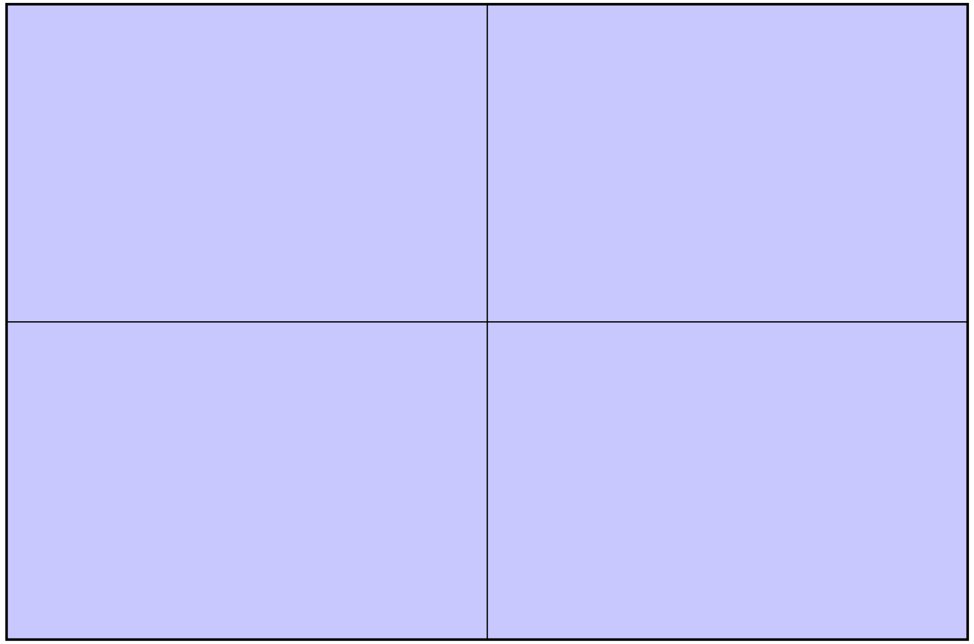
$$\log \frac{P(x_i)}{P(x_k)} = \log \frac{\text{타겟 확률}}{\text{특정 확률}}$$

01 Logit

$$\log \frac{P(x)}{1-P(x)}$$



$$W_n X_n + W_{n-1} X_{n-1} + \cdots + W_2 X_2 + W_1 X_1 + W_0$$



01 Probability

정의
모든 경우의 수에 대한 원하는 경우의 수의 비

역할
MLP에서 확률을 예측하고 예측한 확률을 label과 비교하여 parameter를 업데이트 시키는데 사용

logit을 softmax식으로 연산하면 확률 나옴

02 Softmax

$$P(C_1), \quad P(C_2), \quad P(C_3), \quad \cdots \quad P(C_m), \quad \cdots \quad P(C_k)$$

$$\downarrow \quad \downarrow \quad \downarrow \quad \quad \downarrow \quad \quad \downarrow$$

$$\frac{P(C_1)}{P(C_m)}, \quad \frac{P(C_2)}{P(C_m)}, \quad \frac{P(C_3)}{P(C_m)}, \quad \cdots \quad \frac{P(C_m)}{P(C_m)}, \quad \cdots \quad \frac{P(C_\mathrm{k})}{P(C_m)}$$

$$\downarrow \quad \downarrow \quad \downarrow \quad \quad \downarrow \quad \quad \downarrow$$

$$\log\left(\frac{P(C_1)}{P(C_m)}\right) \quad \log\left(\frac{P(C_2)}{P(C_m)}\right) \quad \log\left(\frac{P(C_3)}{P(C_m)}\right) \quad \cdots \quad \log\left(\frac{P(C_\mathrm{m})}{P(C_m)}\right) \quad \cdots \quad \log\left(\frac{P(C_\mathrm{k})}{P(C_m)}\right)$$

02 Softmax

$$\log\left(\frac{P(C_1)}{P(C_m)}\right) \quad \log\left(\frac{P(C_2)}{P(C_m)}\right) \quad \log\left(\frac{P(C_3)}{P(C_m)}\right) \quad \cdots \quad \log\left(\frac{P(C_\mathrm{m})}{P(C_m)}\right) \quad \cdots \quad \log\left(\frac{P(C_\mathrm{k})}{P(C_m)}\right)$$

$$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow$$

$$\alpha_1, \quad \alpha_2, \quad \alpha_3, \quad \cdots, \quad \alpha_m, \quad \cdots \quad \alpha_k$$

$$\therefore \log\left(\frac{P(C_i)}{P(C_m)}\right) = \alpha_i, \qquad e^{\alpha_i} = \frac{P(C_i)}{P(C_m)}$$

02 Softmax

$$\log\left(\frac{P(C_i)}{P(C_m)}\right) = \alpha_i, \qquad e^{\alpha_i} = \frac{P(C_i)}{P(C_m)}$$

$$\sum_{j=1}^{k}\left(\frac{P(C_j)}{P(C_m)} = e^{\alpha_j}\right) = \frac{P(C_1) + P(C_2) + \cdots + P(C_k)}{P(C_m)} = \frac{1}{P(C_m)}$$

$$\frac{1}{P(C_m)} = e^{\alpha_1} + e^{\alpha_2} + \cdots + e^{\alpha_k} \quad \rightarrow \quad P(C_m) = \frac{1}{e^{\alpha_1} + e^{\alpha_2} + \cdots + e^{\alpha_k}}$$

$$P(C_i) = \frac{e^{\alpha_i}}{e^{\alpha_1} + e^{\alpha_2} + \cdots + e^{\alpha_k}}$$

02 Softmax

Softmax

CLASS `torch.nn.Softmax(dim=None)` [SOURCE]

Applies the Softmax function to an n-dimensional input Tensor rescaling them so that the elements of the n-dimensional output Tensor lie in the range [0,1] and sum to 1.

Softmax is defined as:

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

02 Softmax

OUTPUT LAYER

Softmax

$$logit(1) = \alpha_1 \quad \frac{e^{\alpha_1}}{\sum_{j=1}^{k} e^{\alpha_j}} = P(C_1)$$

$$logit(1) = \alpha_2 \quad \frac{e^{\alpha_2}}{\sum_{j=1}^{k} e^{\alpha_j}} = P(C_2)$$

$$\vdots \qquad \vdots$$

$$logit(\mathrm{k}) = \alpha_{\mathrm{k}} \quad \frac{e^{\alpha_k}}{\sum_{j=1}^{k} e^{\alpha_j}} = P(C_k)$$

CrossEntropyLoss

CrossEntropyLoss

CLASS `torch.nn.CrossEntropyLoss(weight=None, size_average=None, ignore_index=-100, reduce=None, reduction='mean')` [SOURCE]

This criterion combines `nn.LogSoftmax()` and `nn.NLLLoss()` in one single class.

It is useful when training a classification problem with C classes. If provided, the optional argument `weight` should be a 1D *Tensor* assigning weight to each of the classes. This is particularly useful when you have an unbalanced training set.

The *input* is expected to contain raw, unnormalized scores for each class.

input has to be a Tensor of size either $(minibatch, C)$ or $(minibatch, C, d_1, d_2, ..., d_K)$ with $K \geq 1$ for the *K*-dimensional case (described later).

This criterion expects a class index in the range $[0, C-1]$ as the *target* for each value of a 1D tensor of size *minibatch*; if *ignore_index* is specified, this criterion also accepts this class index (this index may not necessarily be in the class range).

CrossEntropyLoss

$$H(p,q) = -\sum_{x\in\mathcal{X}} p(x)\,\log q(x)$$

$p(x) = target$

q(x)= $prediction$

LogSoftMax

$$\text{Softmax} = \frac{e^{\alpha^m}}{\sum_{i=1}^{k} e^{\alpha^i}}$$

$$\alpha_i = log\frac{P(C_i)}{P(C_k)}$$

$$\text{LogSoftmax} = log\frac{e^{\alpha^m}}{\sum_{i=1}^{k} e^{\alpha^i}} = log(P)$$

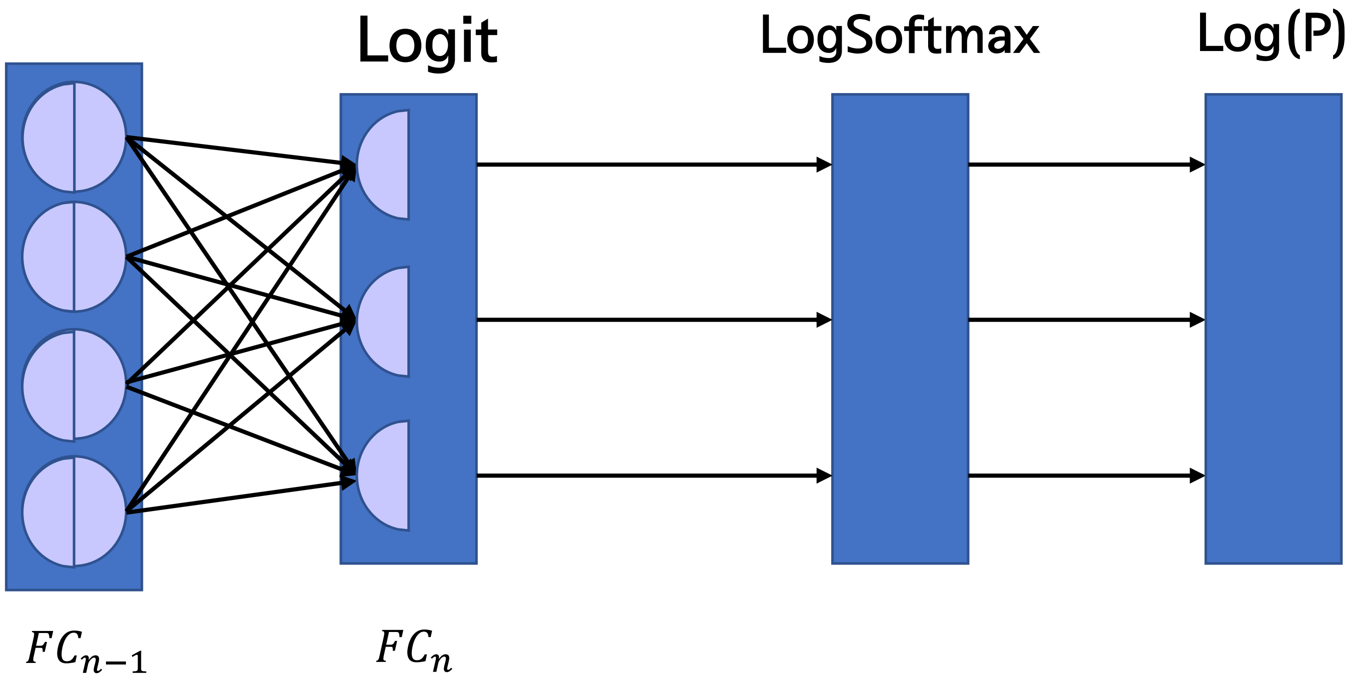
NLL loss

$$L(y) = -\log(y)$$

LogSoftmax = $log(P)$

$$L(y) = -\log(\log(P))$$

$$L(y) = -\cancel{\log}(\log(P))$$


Logit
LogSoftmax
Log(P)
FC_{n-1}
FC_n

$$H(p,q) = -\sum_{x\in\mathcal{X}} p(x)\,\log q(x)$$

$$L(y) = -\log(P)$$