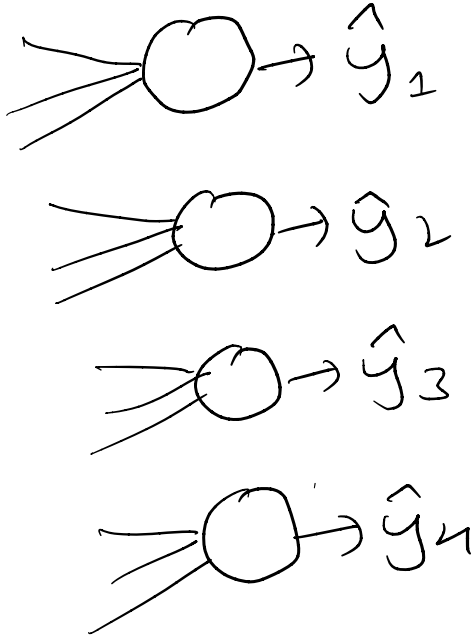
Softmax motivation.

→ Once we get the output from the final layer, we apply softmax function on it to change the output to a probability distribution. The softmax function is

$$S(x_i) = \frac{e^{x_i}}{\sum_k e^{x_k}}$$

The question we want to answer is why do we just use general normalizing function i.e $\frac{x_i}{\sum_k x_k}$ instead of softmax.

To understand this, let's take an example neural network which has four neurons at the output layer. This means that the neural network classifies inputs (x) into four classes. We don't care about the dimension of the input as it can be anything.



let the output value of some input(x) be $a = [2, 4, 2, 1]$ where the real output is $y = [0, 1, 0, 0]$ denoting the input (x) belongs to 2nd class. as $y_2 = 1$ and rest $y_{i \neq 2} = 0$.

Then, to predict the output class ($\hat{y}$) from the given output value a, we apply the argmax on it.

argmax (X, i) = returns the set of indices (i) for which the target function g(i) attains the maximum value.

For our case, $\hat{y} = \text{argmax}(a) = [0, 1, 0, 0]$ which is the target output y which our trained

neural network should be targetting. But argmax is not differentiable, hence we can't use it as we need differentiable function for gradient descent. Let's see if we can come up with another function that achieves what argmax does.

If we look closely at the output of argmax [0, 1, 0, 0], we could view each value as a probability i.e at each index (i), the value $\hat{y}[i]$ gives the probability of y belonging to class 'i' given x.

For our example, $P[y=2|x] = 1$ and for rest

$P[y=\{1, 3, 4\}|x] = 0$.

This implies we have to come up with a way that maps the output values (a_i) to a probability distribution. Normalizing the output value is one way i.e applying $N(n_i) = \frac{n_i}{\sum_k n_k}$ to the output values a_i's. In our example

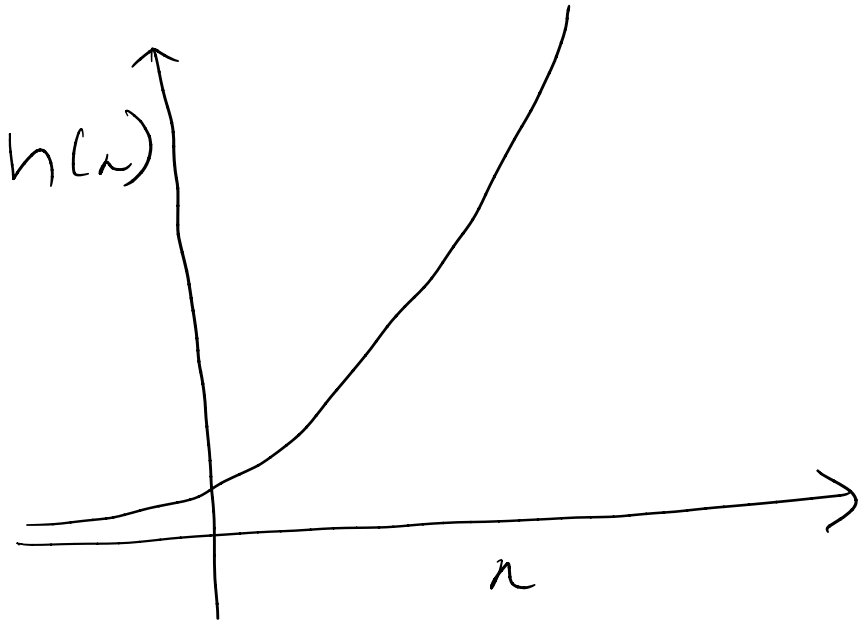
$N(a)$ gives $\hat{y} = [0.22, 0.44, 0.22, 0.11]$ which is very far from the target $y = [0,1,0,0]$.

Also, if we have negative values, then the normalizing function N would assign negative probabilities which shouldn't be the case.

Hence, instead of directly normalizing the raw output values (a_i's), we need to first apply a

mapping which increases the probability of higher output values (a_i's) and decreases the probability of lower output values and also output non-negative values only.

One of such mapping could be $h(n) = e^n$



It does both the things we want.

i) hugely increases the value of higher output values and decreases the value of lower output values

ii) convert negative output values to non-negative.

Then, our final function that will mimic the argmax will be

$$S(x_i) = \frac{e^{x_i}}{\sum_{k} e^{x_k}}$$

Applying it on a, we get,

$$\hat{y} = S(a) = [0.10, 0.75, 0.1, 0.05]$$

which is much closer to our target $y = [0, 1, 0, 0]$.

Note: The name softmax comes from the fact that argmax is known as hardmax and as softmax is able to do what argmax does that is give at least a minimal amount of probability to each value in the output vector 'a' and is also differentiable, hence "soft".