Logistic --> BCE --> CE

Pablo Ruiz – Harvard University – April 2019

Table of Contents

[Context 2](#_Toc6917008)

[Logistic Regression 2](#_Toc6917009)

[Introduction 4](#_Toc6917010)

[Binary Cross Entropy 5](#_Toc6917011)

[Bibliography 7](#_Toc6917012)

# Context

Note: this chapter reviews the concept of log-odds a logistic regression to give a motivation to explain softmax and cross entropy. If you already feel comfortable with those terms, feel free to skip this context section.

Before getting into the details on why we need the softmax layer and why we choose cross entropy to evaluate the fit of our predictions, let's start from the simple case of doing a logistic regression to perform a binary classification.

Simply speaking, we have just 1 output Y (which of course depends on some input X) that can take one of 2 possible values.

Let's say, as we have done in other posts, that we have the hours spent studying by the students of a class and the hours they spent sleeping the day before the exam. Those are the *X*. The output *Y* would be the label representing if they pass the exam. Therefore, we have 1 output that can be "Yes" or "No". However, neural networks don't like strings, so we will say that "Yes" is 1, and "No" is 0.

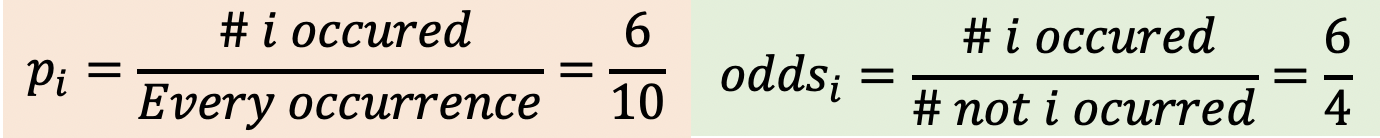
## Logistic Regression

Logistic Regression consists on estimating the parameters of a logistic model, which uses a [logit function](https://en.wikipedia.org/wiki/Logit) to model a binary dependent variable. This logit function has the ability to convert from probability to the log-odds. Ok... what are those?

For illustration purposes, imagine that we have 10 students, and 6 of them passed the exam.

Probability is the ratio between the number of times something happened and everything that could happen. However, the odds are the ratio of something happening against something not happening.

In our case:



The log-odds are simply the result of taking the logarithm of the odds. But, why do we care about that?

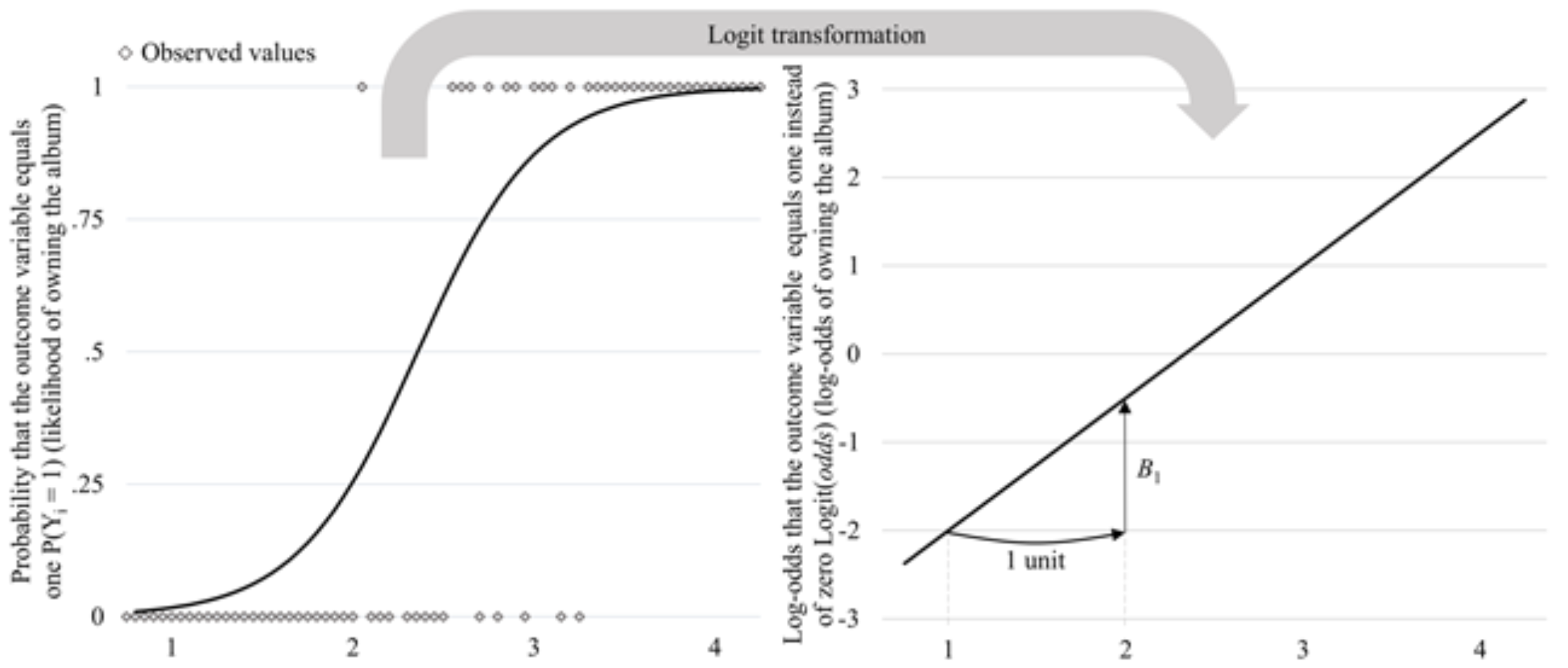
Well, have you ever wondered why we call the model logistic **regression** even if we are performing classification?

The trick is that what we actually do is a linear regression model on the log-odds!

This means that changing the value of one input, multiplicatively sale the odds of the given outcome at a constant rate (linear) by their coefficient.

So, when we train a linear regression model, we try to estimate the coefficients of a linear model that predicts the log-odds as a linear combination of the features (inputs).

And the probability is calculated as,



The idea is basically that we will iteratively change the values of the parameters to change the slope of the straight line to better fit the ground-truth log-odds. Then, we make use of the logistic function (our beloved **sigmoid**) to move back from log-odds space to probability, which is what we care about at the end of the day.

The sigmoid is therefore the inverse of the logit functions, and scales back any number on the log-odds space (which goes from - to + ) to be between 0 and 1.

The reason to take the logarithms is simply because the odds themselves are not bounded; they can go up to infinite. But we solve that issue by moving to logarithmic scale, obtaining symmetry with respect to the origin.

But the point of this post is not this, so I encourage you to visit this [amazing video series](https://www.youtube.com/watch?v=yIYKR4sgzI8&feature=youtu.be) to get a better understanding of log-odds and logistic regression.

# Introduction

Cool, cool, cool... but we should be talking about deep learning and neural networks here, right?

I just wanted to reinforce the concept of using a logistic function to squeeze any value to be between 0-1 and be transformed into a probability.

If we want to make a more powerful model (like the neural network of Figure 1) to solve our binary classification problem, we now know that we will need a sigmoid "gate" at the end of our network. This way, no matter what the values of the weights are at any moment, we are sure that the output will be a value between 0-1. This value, the prediction *y\_hat*, represents the probability of something occurring, in this case, to pass the exam.

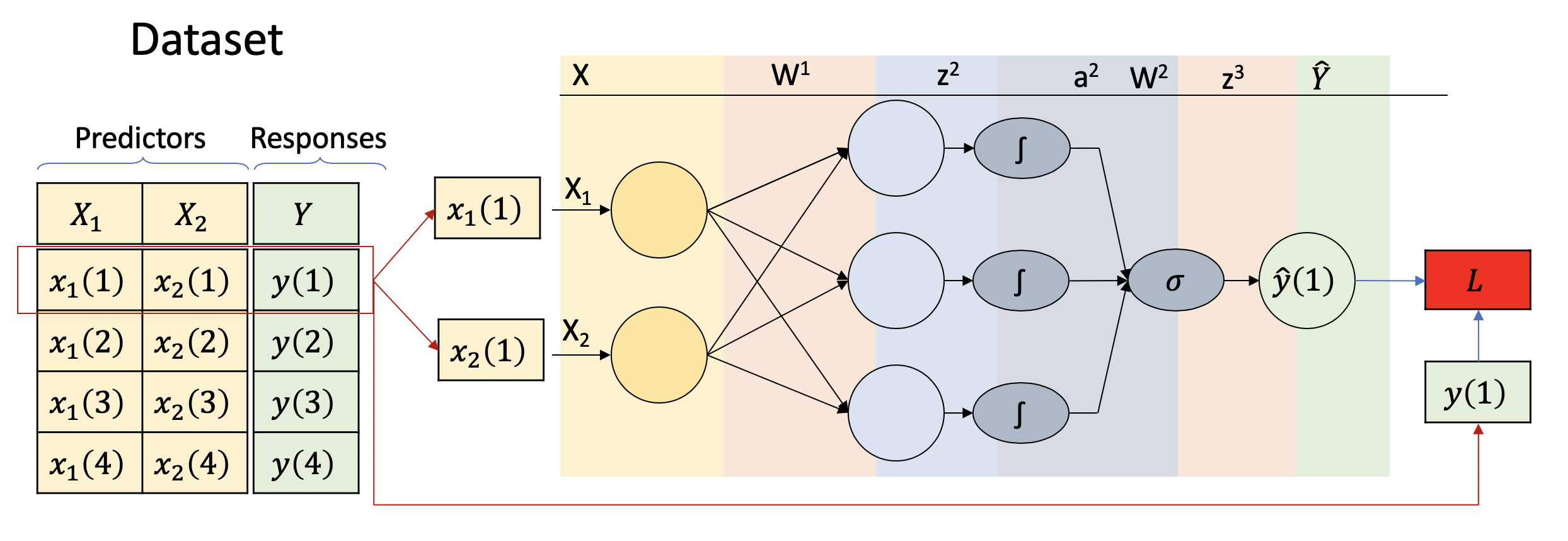


Figure . Neural network to perform binary classification.

Review the notation of the figure:

* X1 and X2 are the inputs. The hours studying and the hours sleeping respectively.
* Y is the ground truth, 1 in the case the student passed, 0 otherwise.
* (1) represent that is the first observations of the dataset
* We are using a mini-batch size of 1
* If you need to review the math, check the [posts on neural network](https://medium.com/@pabloruizruiz/neural-networks-notes-fa42ab388bb8) where this is done step by step.

What the neural network is doing is applying a non-linear transformation on the inputs to move them to a different space to layer apply a sigmoid gate and predict the probability of passing the exam.

This transformation is expressed as (the biases where not represented in the figure for simplification):

|  |  |
| --- | --- |
|  | (Eq. 1) |

Once we have y\_hat, we apply the same principles we did for Logistic Regression. If the value is below 0.5, we assign the prediction to one class, and to the other class if y\_hat is above 0.5. For instance, we will classify all the instances with y\_hat below 0.5 as *"not passing the exam"*, and *"passing the exam"* to all the rest where y\_hat is >= 0.5.

## Binary Cross Entropy

We have already reached the end of the forward pass and have our predictions. We can now evaluate the fitness of our model by comparing the predictions with the truth labels.

In deep learning we call Loss Functions to this particular set of functions that evaluate the fitness of our models, and we try to minimize by changing the value of the weights at every layer. The weights are updated based on the gradient of the error following an algorithm called backpropagation. Take look at [part 4](https://medium.com/@pabloruizruiz/neural-networks-iv-the-graph-approach-cb25590a7f24) and [part 5](https://medium.com/@pabloruizruiz/neural-networks-v-back-propagation-60121cd9b904) of the posts to have a detailed walkthrough on computation graphs and backpropagation of neural networks.

I'll leave here the equation for BCE Loss and will get back to it in the next section. For now:

|  |  |
| --- | --- |
|  | (Eq. 1) |

**Note 1**

The reason why we use Cross Entropy as the Loss Function for classification arises naturally from the fact of applying the maximum likelihood principle to the data distribution and the model distribution. I will probably dedicate an entire post to maximum likelihood and the prove of this particular case soon. For the moment, I strongly recommend you to read [Deep Learning by Ian Goodfellow](https://www.deeplearningbook.org/contents/ml.html), in particular chapter 5 and section 5.5 Maximum Likelihood.

**Note 2**

The Binary Cross Entropy is no different from Cross Entropy. They yield the same fitness if the same predictions are input. However, we need each of them depending on how we express the prediction. We will further express Cross Entropy in detail but take a look for a moment at the Figure 2.

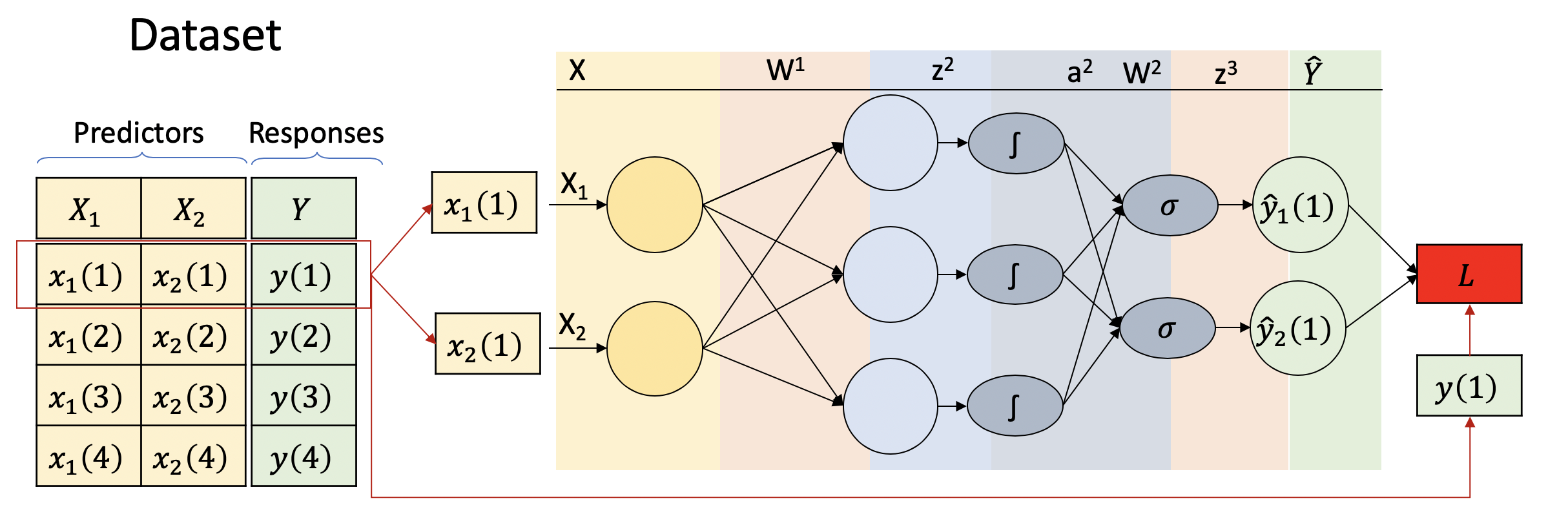


Figure . Moving from 1 output for 2 classes to 2 output

*What we have done here?*

We have split the prediction output into two. Before, we had a prediction and we were assigning a 0 if the output was below 0.5 and 1 otherwise. Meaning, we were predicting the most probable class according to our output.

But it will be possible to have a different output for each of the classes, right? We could train our neural network to tell us, individually, which is the probability for each of the classes. But we quickly come to an issue if we just implement the network as we see it, can you think of it?

*Try to think about it before continuing ;)*

**Exclusive Cases and Probability distributions**

Well, if we don't explicitly handle it, the outputs could be any number between 0 and 1 since they are the output of sigmoid gates, but they won't necessarily add up to 1. In fact, since the values of the weights are random, it is very unlucky that by chance they do add up to 1. For instance, we could have 0.8 for y\_hat1 and 0.7 for y\_hat2. Does it make sense to you that if we only have 2 choices (passing the exam or not), one option has 70% chance and the other 80% chance? Of course not, that would be only possible in case where the different possibilities are not exclusive. However, one student cannot pass and fail the exam at the same time: they are exclusive.

A [probability distribution](https://en.wikipedia.org/wiki/Probability_distribution) must add up to 1. All the different possibilities all together should represent the 100% chances of being! So, we need another trick to solve this problem... have we mentioned **softmax**?

You may be wondering why we don't just use one output as we did before? Well, we could do that in fact, but this second way allow us to scale it to any number of different labels to classify, we simply need one output neuron for each of them! That's also the reason why the output dimension of a classification neural network corresponds to the number of labels to classify. Now we know why softmax is so popular :)

The neural network will then look like:

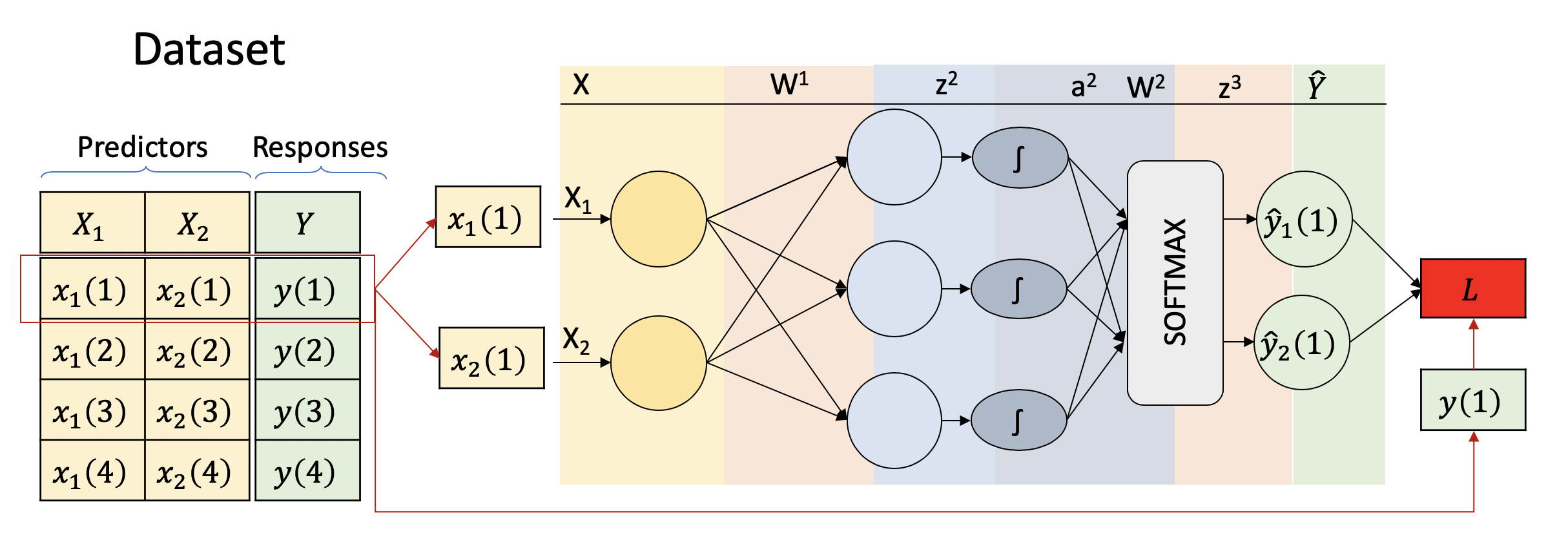


Figure . Neural Network for Multi-Label Classification

## Cross Entropy

We are now generalizing the equation above to evaluate how far is the prediction to the ground truth now that we have more than one class.

|  |  |
| --- | --- |
|  | (Eq. 1) |

Note how in Cross Entropy we are minimizing the negative log likelihood, which is equivalent to maximize the likelihood.

# Bibliography

|  |  |
| --- | --- |
| [1] | K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in *CVPR*, 2016. |