Handling Imbalanced Datasets in Deep Learning

Bring balance to the force!

George Seif Follow
Nov 19, 2018 · 6 min read ★

Bring balance to the force!

Not all data is perfect. In fact, you'll be extremely lucky if you ever get a perfectly balanced real-world dataset. Most of the time, your data will have some level of class imbalance, which is when each of your classes have a different number of examples.

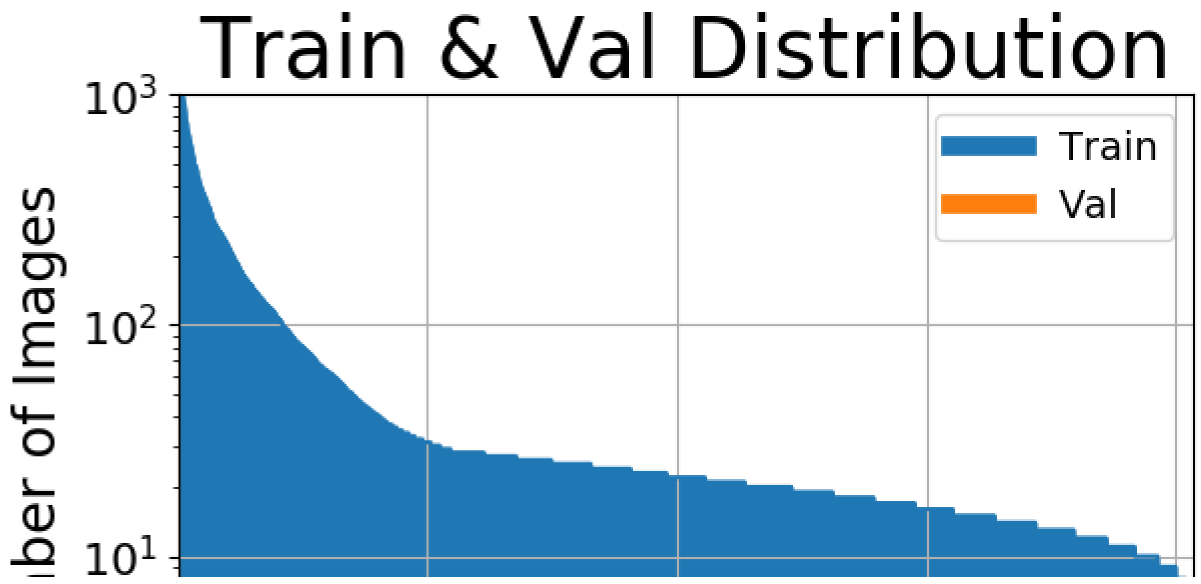
Why do we want our data to be balanced?

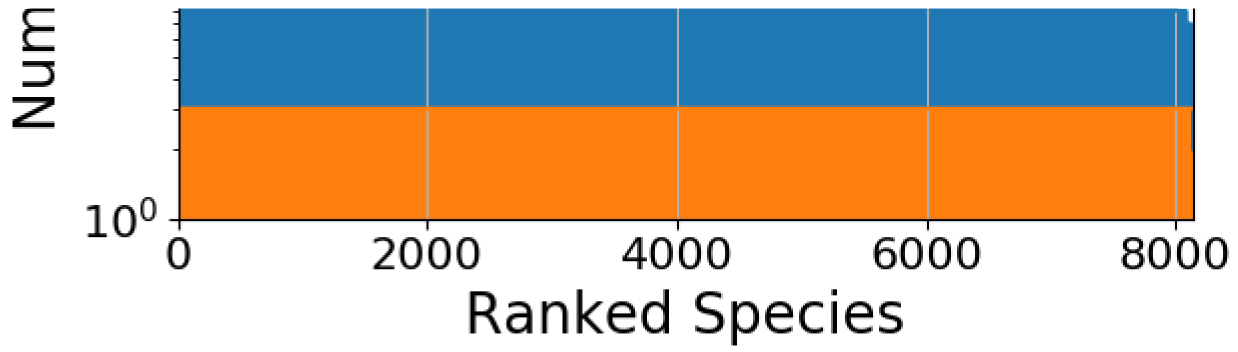
Before committing time to any potentially lengthy task in a Deep Learning project, it's important to understand *why* we should do it so that we can be sure it's a valuable investment. Class balancing techniques are only really necessary when we *actually care* about the minority classes.

For example, let's say we are trying to predict whether or not we should buy a house based on the current state of the market, the house's attributes, and our budget. In this case it's very important that if we buy then it is the right decision since it's such a huge investment. At the same time it's not a really big deal if our model says not to buy when we should have. There will always be other houses to buy if we miss out on one, but making the wrong investment on such a huge asset would be really bad.

In that example, we absolutely **need** our minority "buy" class to be extremely accurate, while for our "don't buy" class it's not such a big deal. Yet in a practical case, since buying will be far more rare than not buying in our data, our model will be biased to learning the "don't buy" class very well since it has the most data and might perform poorly on the "buy" class. That calls for balancing our data so that we can put more weight on the "buy" predictions being correct!

Now what about if we don't really care about the minority classes? For example, let's say we are doing image classification and your class distribution looks something like this:





At first glance it may seem like balancing our data would help. But maybe we're not very interested in those minority classes. Perhaps our main goal is to get the **highest possible percentage accuracy**. In that case it doesn't really make sense to do any balancing since most of our percentage accuracy will come from the classes with more training examples. Secondly, categorical crossentropy losses tend to perform quite well when aiming for the highest percentage accuracy even when the dataset is imbalanced. All in all our minority classes don't contribute much to achieving our main goal, so balancing isn't necessary.

With all of that being said, when we do encounter a case where we want to balance our data there are two techniques that we can use to help us out.

(1) Weight balancing

Weight balancing balances our data by altering the *weight* that each training example carries when computing the loss. Normally, each example and class in our loss function will carry equal weight i.e 1.0. But sometimes we might want certain classes or certain training examples to hold more weight if they are more important. Referring again to our example of buying a house, since the accuracy of the "buy" class is most important to us, training examples within that class should have significant effect on the loss function.

We can give weight to the classes simply by multiplying the loss of each example by a certain factor depending on their class. In Keras we can do something like this:

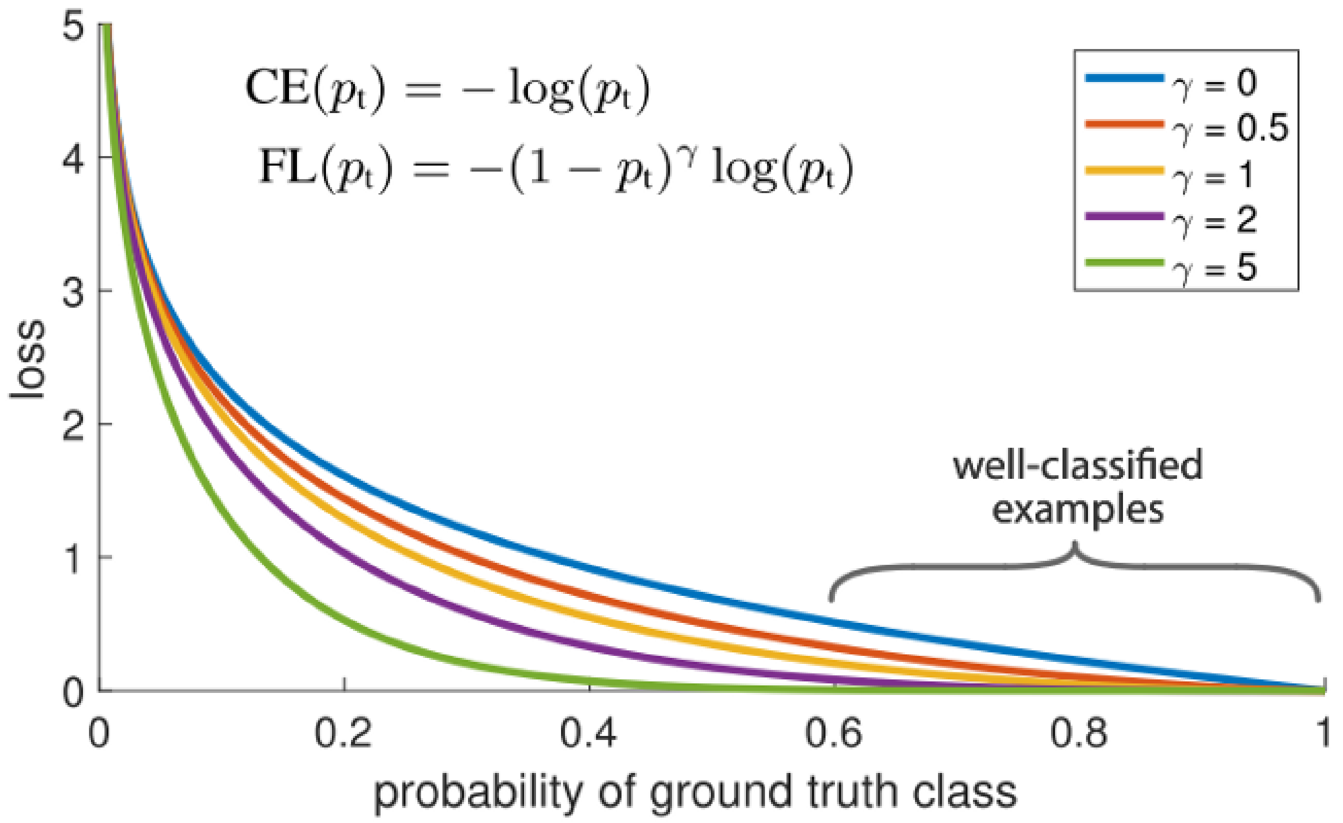
```
import keras

class_weight = {"buy": 0.75,
                "don't buy": 0.25}

```

```python
model.fit(X_train, Y_train, epochs=10, batch_size=32, class_weight=class_weight)
```

class_balance.py hosted with ❤ by **GitHub** **view raw**

We created a dictionary that basically says our "buy" class should hold 75% of the weight for the loss function since it is more important that the "don't buy" class which we accordingly set to 25%. Of course these values can easily be tweaked to find the most optimal settings for your application. We can also use this method of balancing if one of our classes has significantly more examples than the other. Instead of spending time and resources trying to collect more for the minority class, we can try to use weight balancing to make all classes contribute equally to our loss.

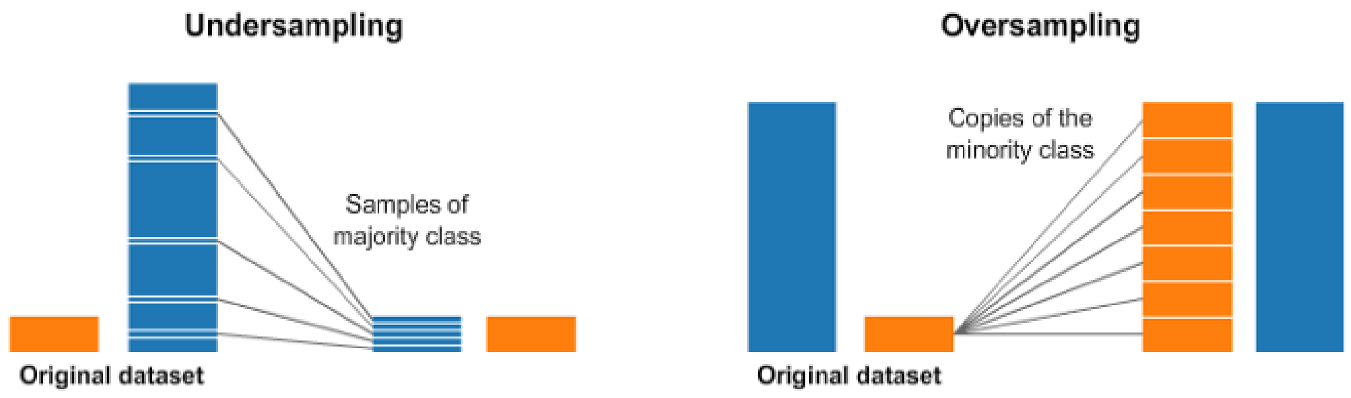
Another method which we can use to balance the weighting of our training examples is the *Focal Loss* shown below. Here's the main idea: in our dataset, we will naturally have some training examples that are easier to classify than others. During training, these examples will be classified with 99% accuracy, while other more challenging ones may still exhibit poor performance. The problem is that those easily classified training examples are *still contributing to the loss.* Why are we still giving them equal weight when there are other more challenging data points that if correctly classified can contribute much more to our overall accuracy?!



And that's exactly the problem that the focal loss can address! Instead of giving equal weighting to all training examples, focal loss *down-weights the well-classified examples*. This has the net effect of putting more training emphasis on that data that is hard to classify! In a practical setting where we have a data imbalance, our majority class will quickly become well-classified since we have much more data for it. Thus, in order to insure that we also achieve high accuracy on our minority class, we can use the focal loss to give those minority class examples more relative weight during training. The focal loss can easily be implemented in Keras as a custom loss function:

(2) Over and under sampling

Selecting the proper class weights can sometimes be complicated. Doing a simple inverse-frequency might not always work very well. Focal loss can help, but even that will down-weight all well-classified examples of *each class equally*. Thus, another way to balance our data is by doing so *directly*, via sampling. Check out the image below for an illustration.



Under and and Over Sampling

In both the left and right side of the image above, our blue class has far more samples than the orange class. In this case, we have 2 pre-processing options which can help in the training of our Machine Learning models.

Undersampling means we will select only *some* of the data from the majority class, only using as many examples as the minority class has. This selection should be done to maintain the probability distribution of the class. That was easy! We just evened out our dataset by just taking less samples!

Oversampling means that we will *create copies* of our minority class in order to have the same number of examples as the majority class has. The copies will be made such that the distribution of the minority class is maintained. We just evened out our dataset without getting any more data! Sampling can be a good alternative to class balancing if you find that the class weights are difficult to set effectively.

. . .

Like to learn?

Follow me on twitter where I post all about the latest and greatest AI, Technology, and Science! Connect with me on LinkedIn too!

Recommended Reading

Want to learn more about Deep Learning? The ***Deep Learning with Python*** book will teach you how to do *real* Deep Learning with the easiest Python library ever: Keras!

And just a heads up, I support this blog with Amazon affiliate links to great books, because sharing great books helps everyone! As an Amazon Associate I earn from qualifying purchases.

Machine Learning Artificial Intelligence Technology Innovation Science